National Technical University of Athens
School of Electrical & Computer Engineering
Department of Computer Science

Design Methodologies for Resource Management of Many-core Embedded Systems

Vasileios Tsoutsouras

Supervisor:
Associate Prof. Dimitrios Soudris

Ph.D. Dissertation

July 2018

Design Methodologies for Resource Management of Many-core Embedded Systems

Vasileios TSOUTSOURAS

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Computer Science

July 2018

This work is partially supported by European Commission projects Swan-iCare under FP7 Grand Agreement No: FP7-ICT-20011-8, AEGLE under H2020 Grant Agreement No: 644906 and VINEYARD under H2020 grant agreement No: 687628.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Design Methodologies for Resource Management of Many-core Embedded Systems

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

ΒΑΣΙΛΕΙΟΥ Σ. ΤΣΟΥΤΣΟΥΡΑ

Διπλωματούχου Ηλεκτρολόγου Μηχανικού &
Μηχανικού Υπολογιστών Ε.Μ.Π. (2013)

Συμβουλευτική Επιτροπή: **Δημήτριος Σούντρης**
Κιαμιάλ Πεχμεστζή
Cristina Silvano

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την $6^{η}$ Ιουλίου 2018.

....	....	....
Δ. Σούντρης	Κ. Πεχμεστζή	C. Silvano
Αν. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Assoc. Professor POLI.MI.
....	....	....
Ν. Κοζύρης	Δ. Γκιζόπουλος	Δ. Πνευματικάτος
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Κ.Π.Α.	Καθηγητής Π.Κ.
	....	
	Jörg Henkel	
	Professor K.I.T.	

Αθήνα, Ιούλιος 2018

. . . .

ΒΑΣΙΛΕΙΟΣ Σ. ΤΣΟΥΤΣΟΥΡΑΣ
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Extended Abstract

The current status of embedded systems contains a variety of complex computing devices featuring high-end, architecturally rich processors, heterogeneous devices and many-core systems. Furthermore, new computing architectures have been proposed at the system level, extending the concept of Internet of Things (IoT) to a multi-layer distributed infrastructure, known as Edge (or Fog) computing. This infrastructure stems from the intention to mitigate a number of inefficiencies of the original Cloud-centric deployment of IoT systems, suffering from dependency on Cloud resources, connectivity issues and unacceptably high bandwidth requirements. In such deployments, the numerous, involved computing nodes must cooperate in order to execute the variety of input tasks resulting from the highly dynamic setup of the system, which includes mobile users and unpredictable application execution requests. The developed IoT applications, must also be designed under the consideration of the updated distributed architecture to be able to fully take advantage of it.

The course of this dissertation, begins by focusing on the requirements and design of embedded applications, operating on systems of multiple nodes. The target applications belong to the medical domain and thus their design requirements include but are not limited to performance, since dependability and accuracy of operations is critical in this field. The design of the IoT-oriented applications is also performed in a modular, pipelined manner in order to provide different run-time configuration knobs, for the effective operation of the device in a Gateway based offloading environment. Automated HW/SW co-design approaches using High Level Synthesis are employed in order to provide a version of the developed applications that is capable of using HW accelerators on combined CPU-FPGA Systems-on-Chip, that are able to significantly decrease the execution latency of the computational intensive parts of the application.

With respect to the design choice of the IoT Gateway, a many-core embedded system with Network-on-Chip topology is considered as a promising design alternative to meet the computational and communicational requirements, resulting from the interaction of the Gateway with numerous IoT nodes. An efficient run-time decision making mechanism is necessary for the many-core system to yield high performance operation. Due to the complexity of dynamically mapping many applications on a many-core system, a Distributed Run-Time Resource Management (DRTRM) framework is designed, implemented and evaluated on top of Intel SCC, an actual many-core NoC based computing platform.

Motivated by the highly dynamic IoT environment, an additional analysis is performed to investigate the correlation of the arrival rate of incoming application requests and the effectiveness of DRTRM on allocating the available system resources. The analysis shows, that a fast and resource hungry scenario of incoming applications can be the breaking point for the effectiveness of DRTRM. Moreover, the enforcement of a relevant run-time mitigation scheme is complicated due to the distributed decision making, which requires the consensus of many agents, thus adding up to the required decision-making latency. This issue is mitigated by use of a Voltage and Frequency Scaling regulation policy, which indirectly slows down application admission, while requiring the cooperation of only a small subset of the agents of the system. The policy is implemented and evaluated on top of DRTRM, showing that it can relieve the congestion of applications under stressful conditions.

The deep scaling of modern many-core systems, combined with the long-operation cycles increase the probability of errors in their processing elements. Taking this into account, due to the importance of DRTRM for the operation of multiple IoT nodes and applications, SoftRM is introduced, a DRTRM augmented with fault tolerant features. The design of SoftRM relies on dynamic, workload-aware error mitigation and refrains from the provisioning of spare cores, via the self-organization of healthy agents in order to replace the failed ones. In addition, an error detection mechanism is implemented, which takes advantage of the communication patterns of DRTRM in order to reduce the overhead of error detection on the operation of healthy agents.

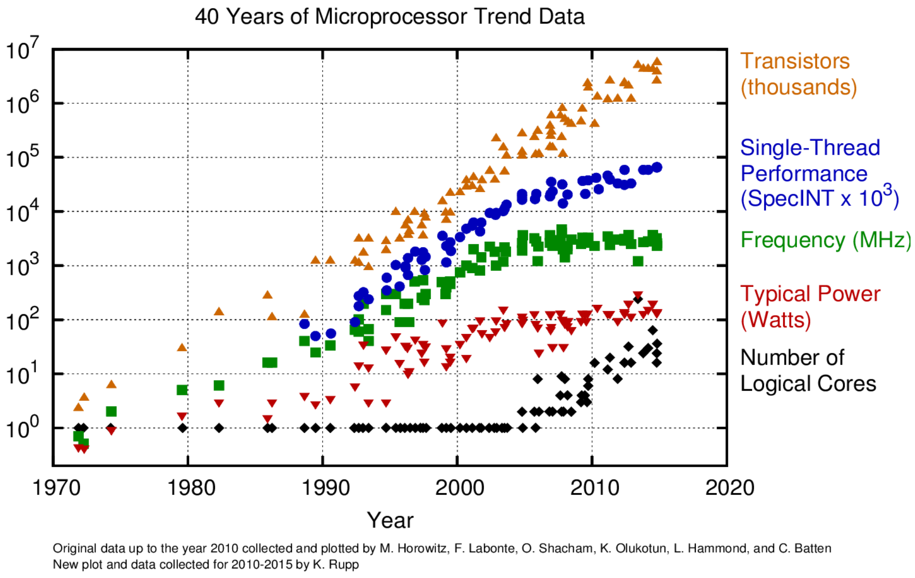
Last, the concepts of distributed management utilized in DRTRM are extended to aid the negotiation of resources at Edge computing systems with multiple intermediate IoT Gateways. These distributed nodes, make use of trade based mechanisms, in order to dynamically optimize the offered Service Quality to their subscribed IoT devices, while meeting their run-time constraints. These mechanisms allow to dynamically achieve more efficient binding of IoT devices to Gateways and thus fully exploit the resources of the latter in order to aid the operation of the first.

Keywords: Distributed Run-Time Resource Management, Edge computing, IoT Gateways, IoT applications

Εκτεταμένη Περίληψη

Τα σύγχρονα ενσωματωμένα συστήματα, περιέχουν πληθώρα σύνθετων υπολογιστικών συσκευών συμπεριλαμβάνοντας αρχιτεκτονικά πλούσιους επεξεργαστές υψηλού κόστους, ετερογενείς συσκευές καθώς και πολυπύρηνους επεξεργαστές. Συνάμα, στο επίπεδο του συστήματος, έχουν προταθεί νέες αρχιτεκτονικές που επεκτείνουν την ιδέα του Διαδικτύου των Πραγμάτων (IoT), κάνοντας χρήση μιας πολύ-επίπεδης κατανεμημένης υποδομής, γνωστή ως υπολογισμός στα άκρα του δικτύου (Edge computing). Η σύλληψη αυτής της υποδομής πηγάζει από την ανάγκη να αντιμετωπιστούν μια σειρά από ανεπάρκειες της αρχικής υποδομής του IoT, που ήταν βασισμένη στο Νέφος (Cloud) και ήταν ως εκ τούτου εξαρτημένη από αυτό, έπασχε από προβλήματα συνδεσιμότητας και οδηγούσε σε μη αποδεκτά μεγάλες ανάγκες για επικοινωνία από τις IoT συσκευές προς το Νέφος. Στις προτεινόμενες υπολογιστικές υποδομές, προκύπτει η ανάγκη συνεργασίας των υπολογιστικών κόμβων με σκοπό να εκτελεστεί η ποικιλία των εργασιών που προέρχονται από τα εξαιρετικά δυναμικά χαρακτηριστικά του συστήματος, το οποίο περιλαμβάνει κινούμενους χρήστες και αδυναμία πρόβλεψης των αιτημάτων εκτέλεσης εφαρμογών. Επιπρόσθετα, οι νέες υπό ανάπτυξη εφαρμογές, πρέπει να έχουν σχεδιαστεί έχοντας υπόψιν την κατανομή των συσκευών, ώστε να είναι σε θέση να επωφεληθούν πλήρως από την ανανεωμένη υποδομή.

Η τρέχουσα διατριβή ξεκινά εστιάζοντας στις απαιτήσεις και τον σχεδιασμό εφαρμογών που απευθύνονται σε ενσωματωμένα συστήματα πολλαπλών κόμβων. Οι υπό σχεδίαση εφαρμογές προέρχονται από τον ιατρικό κλάδο και ως εκ τούτου οι σχεδιαστικές απαιτήσεις τους δεν περιορίζονται στην υψηλή απόδοση, καθώς η ορθή και ακριβής λειτουργία των συσκευών είναι κρίσιμη για τον εν λόγω τομέα. Οι υπό ανάπτυξη εφαρμογές που στοχεύουν στην IoT αρχιτεκτονική, σχεδιάζονται ώστε να περιλαμβάνουν διακριτά, διαδοχικά στάδια εκτέλεσης που αποτελούν διαφορετικές δυναμικές διαμορφώσεις της συσκευής ώστε να μπορεί μια εφαρμογή να εκτελεστεί αποτελεσματικά σε ένα περιβάλλον που κομμάτια της ανατίθενται να εκτελεστούν σε άλλες συσκευές πύλες (Gateways). Κεφάλαια της διατριβής αφορούν επίσης τον αυτοματοποιημένο συ-σχεδιασμό υλικού και λογισμικού με χρήση σύνθεσης υψηλού επιπέδου (High Level



Σχήμα 1: Τάσεις στον τομέα της σχεδίασης μικροεπεξεργαστών.

Synthesis), ώστε να χτιστούν επιταχυμένες εκδόσεις υπολογιστικών πυρήνων για συστήματα που συνδυάζουν κεντρικές μονάδες επεξεργασίας (CPU) καθώς και επαναδιαμορφούμενο υλικό (FPGA). Οι μεθοδολογίες αυτές καταφέρουν αξιοσημείωτη μείωση στον χρόνο εκτέλεσης των υπολογιστικά απαιτητικών τμημάτων των IoT εφαρμογών.

Στα σύγχρονα υπολογιστικά συστήματα, τόσο τα ενσωματωμένα όσο και τα γενικού σκοπού, μια ευρέως χρησιμοποιούμενη τεχνική για την αύξηση των υπολογιστικών επιδόσεων σε συνδυασμό με αποδεκτή κατανάλωση ισχύος είναι η ενσωμάτων όλο και περισσότερων υπολογιστικών στοιχείων στην ίδια ψηφίδα (chip). Η τάση αυτή αποτυπώνεται στο Σχήμα 1, που δείχνει προβλέψεις ότι σε σύντομο χρόνο από την ολοκλήρωση της διατριβής τα υπολογιστικά συστήματα θα περιέχουν εκατοντάδες επεξεργαστές. Η βιομηχανία ήδη συμπλέει με αυτή την ιδέα με αποτέλεσμα να έχουν παρασκευαστεί πρωτότυπα ή και εμπορικές συσκευές με μεγάλο αριθμό επεξεργαστικών στοιχείων. Συγκεκριμένα, η Intel έχει παρουσιάσει πλατφόρμες με 80, 48 και 50 επεξεργαστικά στοιχεία, ενώ η Tilera με 100 ανα ψηφίδα. Ο απώτερος στόχος είναι ψηφίδες με χιλιάδες επεξεργαστικά στοιχεία κάτι το οποίο ήδη προσεγγίζεται από την ακαδημία [42] και την βιομηχανία [170, 23].

Ο αυξημένος αριθμός επεξεργαστών επί της ίδιας ψηφίδας, επηρεάζει και την διασύνδεση τους καθώς σχεδιασμοί με συνεκτικότητα κρυφής μνήμης, κλιμακώνονται αποδοτικά μόνο μέχρι ένα μικρό αριθμό υπολογιστικών στοιχείων. Όσο ο αριθμός των ενσωματωμένων υπολογιστικών στοιχείων αυξάνει, τέτοιου είδους μνήμη μπορεί να αποτελέσει σημείο καμπής για την αποδοτικότητα του συστήματος και ως εκ τούτου απαιτούνται άλλοι τρόποι διασύνδεσης. Η διασύνδεση Δικτύου-σε-Ψηφίδα (Network-on-Chip) έχει επικρατήσει ως η προτεινόμενη σχεδιαστική επιλογή καθώς επιτρέπει μεγάλη κλιμάκωση και περιορισμένη κατανάλωση ενέργειας [41]. Συνολικά, η αφθονία υπολογιστικών στοιχειών σε ένα πολυπύρηνο ενσωματωμένο σύστημα το καθιστά μια πολλά υποσχόμενη λύση για τον σχεδιασμό μιας IoT πύλης (Gateway) ικανή να πληροί τις υπολογιστικές και επικοινωνιακές ανάγκες που προκύπτουν από την συνεργασία την πύλης με πληθώρα IoT συσκευών.

Η δυναμική διαχείριση πόρων σε πολυπύρηνα συστήματα μπορεί να εφαρμοστεί κεντρικά ή κατανεμημένα. Στις παραδοσιακές κεντρικές λύσεις [186], ένας εκ των υπολογιστικών πυρήνων είναι υπεύθυνος για να διερευνήσει τις ιδιότητες των εκτελουμένων εφαρμογών και να πραγματοποιήσει την δυναμική χαρτογράφηση (mapping) τους. Παρότι η προσέγγιση αυτή είναι πιο προφανής και εύκολη στην υλοποίηση, χαρακτηρίζεται από έλλειμα κλιμάκωσης, γεγονός που μπορεί να οδηγήσει σε μη αποδεκτές καθυστερήσεις στην λήψη αποφάσεων, ιδιαίτερα σε συστήματα με πολλές εφαρμογές και συχνές δυναμικές μεταβολές. Επιπρόσθετα, η κεντρική λήψη αποφάσεων είναι επιρρεπής σε αποτυχία, καθώς σε περίπτωση σφάλματος του κεντρικού διαχειριστή, η λειτουργία του συστήματος καταρρέει.

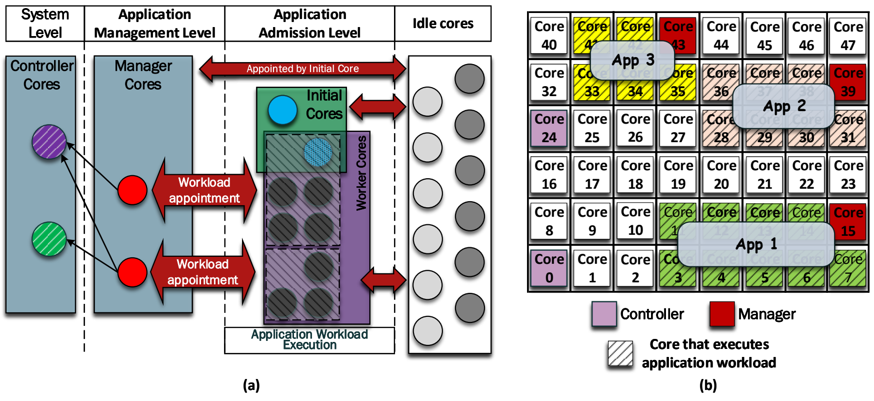
Στον αντίποδα, η κατανεμημένη διαχείριση των πόρων του συστήματος [56, 18], έχει κερδίσει έντονο ενδιαφέρον. Έρευνες [21, 130] έχουν δείξει ότι οι κατανεμημένες στρατηγικές μπορούν να οδηγήσουν σε εξαιρετικά αποδοτική διαχείριση των εφαρμογών και των πόρων του συστήματος. Έχοντας αυτή την γνώση ως κίνητρο, στην τρέχουσα διατριβή σχεδιάστηκε, υλοποιήθηκε και αναλύθηκε πειραματικά ένας Κατανεμημένος Δυναμικός Διαχειριστής Πόρων (Distributed Run-Time Resource Management (*DRTRM*)) απευθυνόμενος σε πολυπύρηνα υπολογιστικά συστήματα με διασύνδεση Δικτύου-σε-Ψηφίδα.

Ο στόχος του προτεινόμενου διαχειριστή πόρων είναι να παρέχει κατανεμημένη διαχείριση τόσο των πόρων όσο και των παράλληλων εφαρμογών σε ένα πολυπύρηνο επεξεργαστικό σύστημα. Η κατανομή του υπολογιστικού φόρτου σε πληθώρα κατανεμημένων έξυπνων πρακτόρων (agents), αποτελεί ελκυστική επιλογή καθώς η χαρτογράφηση των εφαρμογών είναι ένα γνωστό NP-hard αλγοριθμικό πρόβλημα [207]. Το εγγενές πρόβλημα που προκύπτει από την επιλογή αυτή είναι η απαίτηση για περιορισμένη και αποδοτική επικοινωνία μεταξύ των κατανεμημένων πρακτόρων. Ο εν λόγω διαχειριστής πόρων είναι σχεδιασμένος για παράλληλες εφαρμογές με έμφαση στις εύπλαστες, οι οποίες μπορούν να αναπροσαρμοστούν δυναμικά σε τυχόν μεταβολή των πόρων τους.

Η κεντρική ιδέα του *DRTRM* έγκειται στη εναλλαγή ρόλων των διαφόρων πυρήνων καθ' όλη την διάρκεια της εκτέλεσης του συστήματος, με στόχο να υποστηριχθούν οι διάφορες διαχειριστικές αποφάσεις καθώς και η εκτέλεση του φόρτου εργασίας των παράλληλων εφαρμογών. Η τεχνική αυτή έχει ήδη οδηγήσει σε πολύ καλά αποτελέσματα, όπως παρουσιάζονται στις εργασίες των [130, 20, 21, 132]. Στον υπό παρουσίαση σχεδιασμό, κυριαρχούν τρείς κατηγορίες κατανεμημένων οντοτήτων-πρακτόρων:

- Αρχικός πυρήνας (*Initial core*): Η λειτουργία του ενεργοποιείται όταν μια καινούρια εφαρμογή ζητά να ξεκινήσει την εκτέλεση της στο σύστημα. Στόχος της λειτουργίας του είναι να ψάξει σε μια περιοχή του συστήματος ώστε να εντοπίσει έναν ή περισσότερους επεξεργαστές στους οποίους θα ανατεθεί η εκτέλεση της νέας εφαρμογής. Για κάθε καινούρια εφαρμογή ανατίθεται ένας καινούριος Αρχικός πυρήνας οι υποχρεώσεις του οποίου ολοκληρώνονται με το πέρας της αρχικοποίησης της εφαρμογής.
- Διαχειριστής πυρήνας (*Manager core*): Η λειτουργία αυτής της οντότητας είναι συνυφασμένη με όλον τον κύκλο ζωής μιας εφαρμογής. Κάθε εφαρμογή έχει τον δικό της Διαχειριστή, ο οποίος είναι υπεύθυνος για την διαχείριση των πόρων της και την κατανομή του φόρτου εργασίας της στους πυρήνες εργάτες. Οι Διαχειριστές πυρήνες όλων των εφαρμογών του συστήματος, δρουν συνεργατικά για την κατανομή και ανταλλαγή των πόρων μεταξύ των εφαρμογών ώστε να μεγιστοποιείται η αποδοτικότητα του συστήματος. Τα καθήκοντα του Διαχειριστή πυρήνα ολοκληρώνονται με το πέρας της εκτέλεσης της εφαρμογής.
- Ελεγκτής πυρήνας (*Controller core*): Η οντότητα αυτή αποτελεί την ραχοκοκαλιά του *DRTRM* και είναι υπεύθυνη για την παρακολούθηση και διαχείριση του συστήματος. Οι Ελεγκτές πυρήνες ανατίθενται στην αρχικοποίηση του συστήματος και ο ρόλος τους δεν μεταβάλλεται καθ' όλη της διάρκεια λειτουργίας της πολυπύρηνης πλατφόρμας. Κάθε ένας από αυτούς είναι υπεύθυνος για την παρακολούθηση ενός συμπλέγματος (Cluster) πυρήνων, που αντιστοιχεί σε μια μη επικαλυπτόμενη περιοχή του συστήματος. Ο Ελεγκτής καταγράφει τις εφαρμογές που δραστηριοποιούνται μέσα στην περιοχή αυτή και παρέχει αυτή την πληροφορία σε όποια άλλη κατανεμημένη οντότητα το ζητήσει. Ο ρόλος του είναι απαραίτητος καθώς καμία οντότητα δεν έχει πλήρη εικόνα του συστήματος και ως εκ τούτου οι χωρικές πληροφορίες αποθηκεύονται και παρέχονται κατανεμημένα από τους Ελεγκτές.

Ο προτεινόμενος σχεδιασμός επιβάλλει έμμεσα μια ιεραρχία στους ρόλους των διαφόρων κατανεμημένων οντοτήτων, όπως παρουσιάζεται στο Σχήμα 2(α). Στο επίπεδο διαχείρισης συστήματος οι Ελεγκτές πυρήνες είναι τα δομικά



Σχήμα 2: (α) Ιεραρχία πυρήνων στον DRTRM (β) Παράδειγμα πολυπύρηνης πλατφόρμας που εκτελείται ο DRTRM.

στοιχεία του *DRTRM* και επιβλέπουν της λειτουργία της πλατφόρμας. Στο επίπεδο διαχείρισης των εφαρμογών, οι Διαχειριστές πυρήνες γνωστοποιούν την ύπαρξη τους στις τοπικές πληροφορίες των Ελεγκτών πυρήνων και ταυτόχρονα ασχολούνται με την διαχείριση των εφαρμογών. Οι αρχικοί πυρήνες είναι σε χαμηλότερο ιεραρχικό επίπεδο και είναι υπεύθυνοι για την αρχικοποίηση των εφαρμογών. Στο ίδιο ιεραρχικό επίπεδο βρίσκονται και οι πυρήνες εργάτες που εκτελούν τον φόρτο εργασίας των εφαρμογών. Όλοι οι παραπάνω δυναμικοί ρόλοι ανατίθενται στο κατώτατο όριο ιεραρχίας που είναι οι αδρανείς πυρήνες, οι οποίοι και αναβαθμίζονται. Ο ιεραρχικός σχεδιασμός δεν παρουσιάζει μόνο μια υψηλού επιπέδου απεικόνιση των εργασιών των κατανεμημένων οντοτήτων, αλλά ταυτόχρονα υποδεικνύει τα όρια και περιορισμούς της λειτουργίας τους καθώς και την ροή πληροφορίας από τη μια οντότητα στην άλλη. Ταυτόχρονα, η επιβαλλόμενη ιεραρχία είναι μια κρίσιμη σχεδιαστική παράμετρος του *DRTRM* ώστε να μπορεί να ενσωματώσει δυναμικές πολιτικές σε συστήματα μεγάλης κλίμακας όπου η ομόφωνη λειτουργία όλων των οντοτήτων είναι ανέφικτη και ως εκ τούτου οι κρίσιμες αποφάσεις είναι το αποτέλεσμα της συνεργασίας λίγων, προνομιούχων οντοτήτων.

Μια σφαιρική άποψη του παρουσιαζόμενου διαχειριστή πόρων και των καθηκόντων του κάθε πυρήνα φαίνεται στο Σχήμα 2(β), σε ένα παράδειγμα πολυπύρηνης πλατφόρμας με 48 ενσωματωμένους επεξεργαστές. Οι πυρήνες 0 και 24 είναι οι δυο Ελεγκτές πυρήνες που επιβλέπουν το σύστημα. Εις εξ αυτών παρακολουθεί μια ορθογώνια περιοχή, ενώ οι δυο περιοχές δεν έχουν κοινά σημεία. Στο συγκεκριμένο παράδειγμα υπάρχουν τρείς εκτελούμενες εφαρμογές οι οποίες διαχειρίζονται από τρείς Διαχειριστές πυρήνες. Ο Διαχειριστής της πρώτης εφαρμογής (App 1) είναι ο πυρήνας 15, της δεύτερης εφαρμογής (App 2) ο πυρήνας 39 και της τρίτης (App 3) ο 43.

Η τρέχουσα διατριβή στοχεύει σε παράλληλες, δυναμικές εφαρμογές με κύρια έμφαση στις λεγόμενος εύπλαστες (malleable), καθώς οδηγούν στην βέλτιστη χρήση των πόρων του συστήματος. Υποθέτοντας ότι μόνο μια εξ αυτών εκτελείται σε ένα πολυπύρηνα σύστημα, η επιτάχυνση της (speedup) $S(n)$, ορίζεται ως ο λόγος του χρόνου που χρειάζεται η εφαρμογή για να εκτελεστεί σε έναν πυρήνα προς τον χρόνο που χρειάζεται για να εκτελεστεί σε n πυρήνες. Επίσης, ο εναπομείνας χρόνος εκτέλεσης της εφαρμογής σχετίζεται με τον εναπομείναντα φόρτο εργασίας W όπως περιγράφει η Εξίσωση 1 [130].

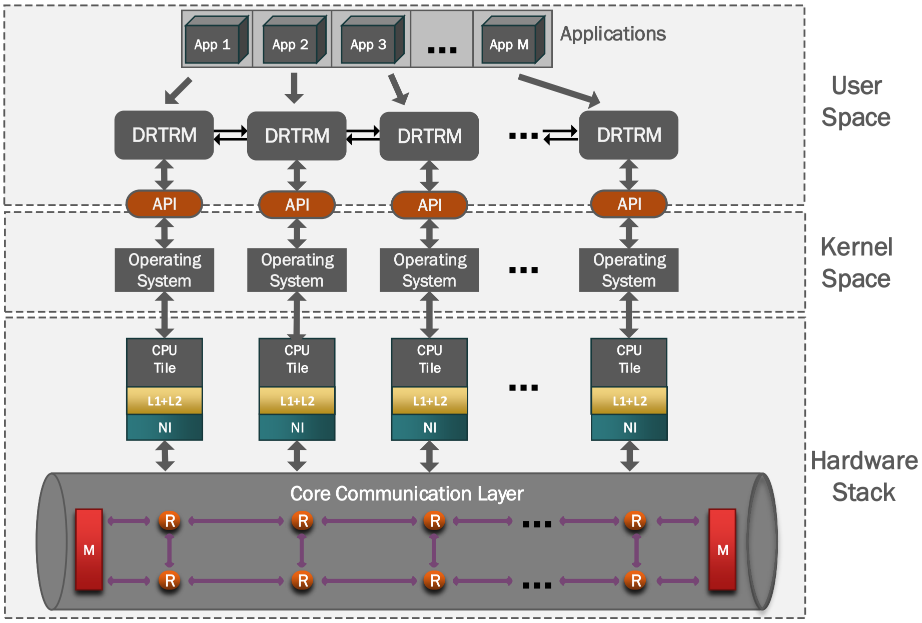
$$S(n) = \frac{T(1)}{T(n)}; \quad T_{finish} = \frac{W}{S(N)} \tag{1}$$

Ωστόσο, η χρήση της Εξίσωσης 1 είναι περιορισμένη καθώς απαιτεί την εκ των προτέρων γνώση του χρόνου εκτέλεσης των εφαρμογών. Κατ' επέκταση, παρόμοιες εργασίες στην κατανεμημένη διαχείριση πόρων [130, 21] έχουν κάνει χρήση του μοντέλου εύπλαστων εφαρμογών που παρουσιάζεται στην εργασία [84] και παρέχει μια εκτίμηση για την επιτάχυνση των εύπλαστων εφαρμογών. Το μοντέλο αυτό περιγράφει κάθε εύπλαστη εφαρμογή μέσω τριών παραμέτρων, οι οποίες είναι ο φόρτος εργασίας W, ο μέσος παραλληλισμός A και η απόκλιση παραλληλισμού σ. Με βάση αυτές τις παραμέτρους, η ιδανική επιτάχυνση της εφαρμογής όταν εκτελείται μόνη της σε ένα πολυπύρηνο σύστημα ορίζεται σύμφωνα με την Εξίσωση 2. Το μοντέλο που ορίζεται από τις Εξισώσεις 1 και 2 παρέχει μια πρόβλεψη της αποδοτικότητας και του εναπομείναντα χρόνου εκτέλεσης μια παράλληλης εύπλαστης εφαρμογής βάσει των προαναφερθέντων χαρακτηριστικών.

$$\underbrace{S(n) = \begin{cases} \frac{nA}{A+\frac{\sigma}{2(n-1)}} & : 1 \le n < A \\ \frac{nA}{\sigma(A-\frac{1}{2})+n(1-\frac{\sigma}{2})} & : A \le n < 2A-1 \\ A & : n \ge 2A-1 \end{cases}}_{\sigma < 1} \tag{2α'}$$

$$\underbrace{S(n) = \begin{cases} \frac{nA(\sigma+1)}{\sigma(n+A-1)+A} & : 1 \le n \le A + A\sigma - \sigma \\ A & : n > A + A\sigma - \sigma \end{cases}}_{\sigma \ge 1} \tag{2β'}$$

Στο Σχήμα 3 παρουσιάζεται μια πιο λεπτομερής περιγραφή του υλικού που αποτελεί τον στόχο του δυναμικού διαχειριστή πόρων και τις αρχές πάνω στις οποίες σχεδιάστηκε ο *DRTRM*. Μια δυάδα επεξεργαστικών στοιχείων,



Σχήμα 3: Πλατφόρμα και μοντέλο συστήματος για την υλοποίηση του DRTRM.

διασυνδέεται σε ένα Δίκτυο-σε-Ψηφίδα διαμόρφωσης πλέγματος, όπου ένας δρομολογητής πακέτων ανά δυάδα επεξεργαστών προωθεί τα πακέτα στον σωστό παραλήπτη. Κάθε δυάδα επεξεργαστών μοιράζεται μια κοινή κρυφή μνήμη δευτέρου επιπέδου (L2 cache), ενώ ο καθένας έχει την ιδιωτική του κρυφή μνήμη πρώτου επιπέδου (L1 cache) και διαχειρίζεται από έναν πυρήνα του λειτουργικού συστήματος Linux. Ο *DRTRM* έχει σχεδιαστεί ως μια υπηρεσία επί του λειτουργικού συστήματος του κάθε επεξεργαστή. Αυτή η σχεδιαστική επιλογή επιτρέπει στην αρχιτεκτονική του *DRTRM* να χρησιμοποιεί αποδοτικά την υποδομή του λειτουργικού συστήματος χωρίς να χρειάζεται η τροποποίηση του πυρήνα του. Κατ' επέκταση αυτό ενισχύει την δομοστοιχειότητα (modularity) και την δυνατότητα μεταφοράς του σχεδιασμού σε άλλες πλατφόρμες παρόμοιας αρχιτεκτονικής με περιορισμένη προσπάθεια.

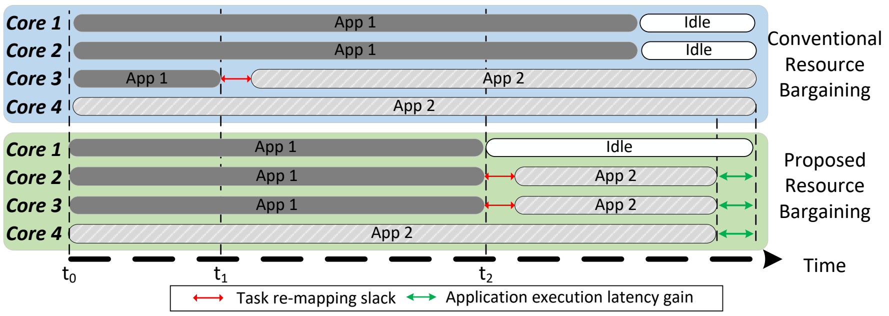
Αναφορικά με την διαχείριση των παράλληλων εφαρμογών, όπως έχει ήδη ειπωθεί, ένας Διαχειριστής πυρήνας ανατίθεται ανά εφαρμογή με σκοπό να παρακολουθεί την κατάσταση της, να ενορχηστρώνει την κατανομή του φόρτου εργασίας της και να αναζητά καινούριους υπολογιστικούς πόρους ώστε να μεγιστοποιήσει τον αριθμό των εργατών και επομένως την επιτάχυνση της εφαρμογής. Παράλληλα, απαντά και σε αντίστοιχα αιτήματα από άλλους Διαχειριστές εφαρμογών. Όλες αυτές οι δραστηριότητες συνδέονται με την διαπραγμάτευση των πόρων του συστήματος με απώτερο σκοπό την βέλτιστη χαρτογράφηση των εφαρμογών. Η σχεδιαστική επιλογή της ανάθεσης ενός Διαχειριστή πόρων ανά εφαρμογή πραγματοποιήθηκε ώστε ο δημιουργός της εφαρμογής να μπορεί να καθορίσει την βέλτιστη πολιτική διαχείρισης των πόρων της εφαρμογής [68], η οποία πολιτική θα επιβάλλεται απρόσκοπτα από εξωτερικές παρεμβάσεις και εκτέλεση φόρτου εργασίας της εφαρμογής.

Το αντικείμενο βελτιστοποίησης στους τρέχοντες σχεδιασμούς που στοχεύουν στην μεγιστοποίηση της απόδοσης του συστήματος, είναι η μεγιστοποίηση της επιτάχυνσης των εκτελούμενων εφαρμογών [130, 21]. Ο *DRTRM* επεκτείνει αυτή την ιδέα στοχεύοντας στην ελαχιστοποίηση του χρόνου εκτέλεσης των εφαρμογών λαμβάνοντας ταυτόχρονα υπόψη των εναπομείναντα χρόνο εκτέλεσης τους όταν αποτιμάται μια πιθανή μεταφορά πόρων. Στον *DRTRM* όταν ο Διαχειριστής πόρων της εφαρμογής A, ζητά πόρους από τον Διαχειριστή πόρων της εφαρμογής B, το πρώτο που αποτιμάται είναι αν η μεταφορά των πόρων θα οδηγήσει σε καλύτερη κατάσταση το συνολικό σύστημα, αυξάνοντας την αθροιστική επιτάχυνση των εφαρμογών. Αυτό περιγράφεται και στην Εξίσωση 3 όπου η μεταφορά πόρων πραγματοποιείται μόνο αν το κόστος είναι θετικό, δηλαδή η αθροιστική επιτάχυνση των εφαρμογών A και B είναι μεγαλύτερη μετά την μεταφορά των πόρων. Η αποτίμηση αυτή είναι άπληστη υπό την έννοια ότι περαιτέρω πόροι μεταφέρονται από την εφαρμογή B στην A εφόσον τα κέρδη του παραλήπτη είναι μεγαλύτερα από τις απώλειες του παρόχου των πόρων.

$$Cost = gain[S'_A(n_{old}) \rightarrow S'_A(n_{new})] - loss[S'_B(n_{old}) \rightarrow S'_B(n_{new})] \qquad (3)$$

Ονομάζουμε αυτή την άπληστη προσέγγιση στην διαπραγμάτευση των πόρων ως συμβατική (conventional resource bargaining), η οποία χαρακτηρίζεται από το μειονέκτημα του να μην λαμβάνει υπόψη τόσο τον εναπομείναντα χρόνο εκτέλεσης της εφαρμογής που παρέχει πόρους όσο και τον απαιτούμενο χρόνο ώστε αυτοί οι πόροι να χαρτογραφηθούν εκ νέου στην νέα εφαρμογή. Η προτεινόμενη στρατηγική διαπραγμάτευσης πόρων βασίζεται στην συν-εκτίμηση όλων αυτών των παραμέτρων. Πιο συγκεκριμένα, μια ανταλλαγή πόρων πραγματοποιείται όταν έχει νόημα για την αύξηση της επιτάχυνσης αλλά ταυτόχρονα είναι αποδοτική αναφορικά με το εκτιμώμενο πέρας της εφαρμογής που παραδίδει τους πόρους.

Η λογική πίσω από αυτή την επιλογή συνοψίζεται στο Σχήμα 4, όπου παρουσιάζεται το χρονοδιάγραμμα της εκτέλεσης μιας εφαρμογής με τέσσερις πυρήνες εργάτες σε ένα πολυπύρηνο σύστημα. Την χρονική στιγμή t_0, η εφαρμογή 2 ζητά πόρους από την εφαρμογή 1. Η εφαρμογή 1 αποτιμά την πιθανότητα παροχής ενός πυρήνα και αποφαίνεται ότι αξίζει να πραγματοποιηθεί η μεταφορά του πυρήνα καθώς μετά από αυτήν η αθροιστική επιτάχυνση των δύο εφαρμογών θα είναι μεγαλύτερη από την τρέχουσα. Έτσι την χρονική στιγμή t_1 ο πυρήνας 3 μεταφέρεται από την εφαρμογή 1 στην εφαρμογή 2. Ωστόσο, η

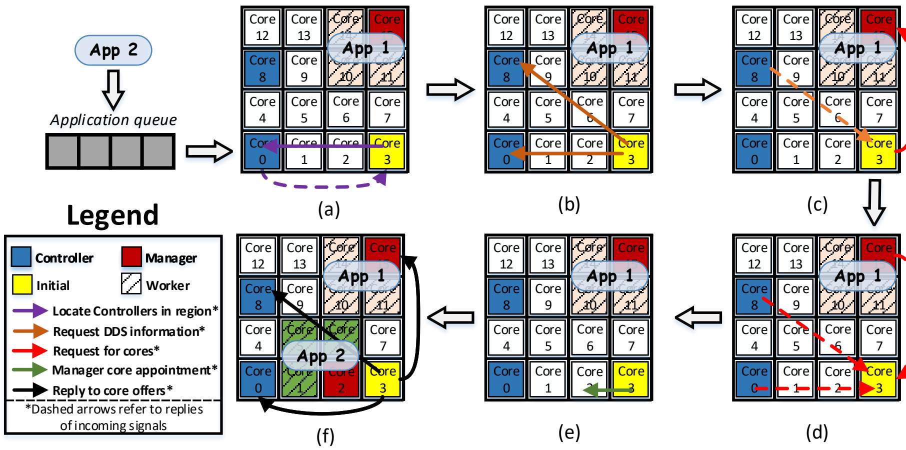

Core 1
Core 2
Core 3
Core 4
App 1
App 2
Idle
Conventional Resource Bargaining
Proposed Resource Bargaining
t0
t1
t2
Time
Task re-mapping slack
Application execution latency gain

Σχήμα 4: Παράδειγμα διαπραγμάτευσης πόρων (Σύγκριση συμβατικής και προτεινόμενης στρατηγικής).

μεταφορά αυτή δεν λαμβάνει υπόψιν της τον εναπομείναντα χρόνο της εφαρμογής 1. Στην αντίστοιχη περίπτωση, ο προτεινόμενος αλγόριθμος διαπραγμάτευσης πόρων αντιλαμβάνεται ότι η εφαρμογή 1 πλησιάζει στο τέλος της και επομένως η παροχή του πυρήνα της θα είναι μια μη αποδοτική κίνηση. Η απόφαση της μη παροχής του πυρήνα την χρονική στιγμή t_1 οδηγεί στην έγκαιρη ολοκλήρωση της εφαρμογής 1, αφήνοντας πληθώρα ελεύθερων πόρων για την εφαρμογή 2. Με αυτή την στρατηγική, την χρονική στιγμή t_2 η εφαρμογή 2 αποκτά 2 ελεύθερους πυρήνες και ολοκληρώνει την εκτέλεση της συντομότερα από την περίπτωση της συμβατικής διαπραγμάτευσης πόρων.

Η διαδικασία που ακολουθεί ένας πυρήνας Διαχειριστής για να μεγιστοποιήσει δυναμικά τους πυρήνες της εφαρμογής του ονομάζεται διαδικασία αυτό-βελτιστοποίησης (self-optimization). Η μεταφορά πυρήνων λαμβάνει χώρα μεταξύ δραστών που έχουν στην κατοχή τους πυρήνες, δηλαδή τους Διαχειριστές και τους Ελεγκτές (κατέχουν τους ανενεργούς πυρήνες). Η διαδικασία της διαρκούς αναζήτησης πόρων βασίζεται στο γεγονός ότι η διαθεσιμότητα τους μεταβάλλεται χρονικά, λόγω έναρξής ή ολοκλήρωσης εφαρμογών. Έτσι, ο Διαχειριστής πυρήνας ξεκινά περιοδικούς γύρους αναζήτησης πόρων. Η διαδικασία αυτή αποφεύγεται μόνο αν η εφαρμογή έχει πιάσει την μέγιστη επιτάχυνση της ή έχει μικρό προσδοκώμενο χρόνο περάτωσης. Η αναζήτηση για πόρους γίνεται σε μια περιοχή της πλατφόρμας με κέντρο τον Διαχειριστή πυρήνα και ακτίνα R. Η αναζήτηση περιορίζεται σε αυτή την περιοχή ώστε να μειώσει την διασπορά την εργατών μιας εφαρμογής και κατ' επέκταση τον περιορισμό της απόστασης επικοινωνίας τους.

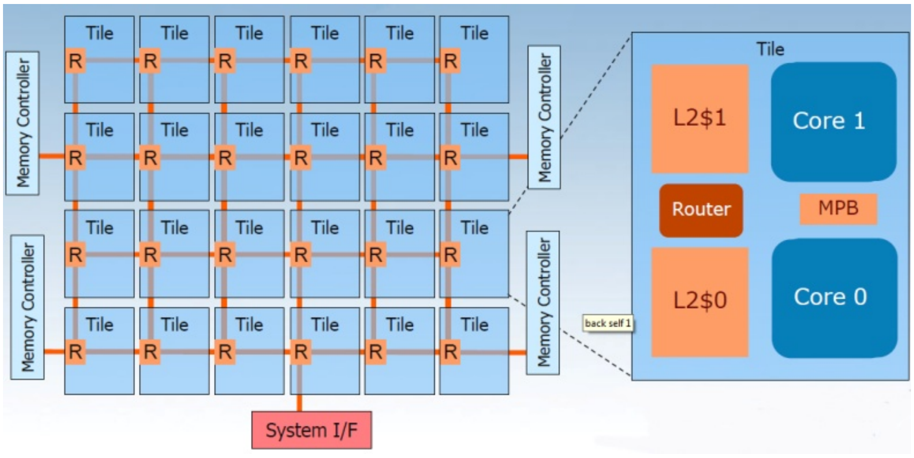
Στο Σχήμα 5 παρουσιάζεται ένα πλήρες παράδειγμα της λειτουργίας του προτεινόμενου κατανεμημένου διαχειριστή πόρων, με έμφαση στην αρχικοποίηση μια καινούριας εισερχομένης εφαρμογής στο σύστημα. Σε πρώτη φάση η καινούρια εφαρμογή αποστέλλεται μέσω του Ελεγκτή πυρήνα 0, στον Αρχικό πυρήνα



Σχήμα 5: Παράδειγμα κατανομής πόρων και επικοινωνίας πυρήνων.

3, ο οποίος και την αποδέχεται (Σχήμα 5a). Κατόπιν, ο Αρχικός πυρήνας στέλνει ένα σήμα στους Ελεγκτές πυρήνες ώστε να ενημερωθεί για το ποιοι Διαχειριστές πυρήνες είναι ενεργοί στο σύστημα (Σχήμα 5b). Μόλις λάβει αυτή την πληροφορία, αποστέλλει αιτήματα σε όλους τους ενεργούς Διαχειριστές ώστε να δεχτεί προσφορές για πόρους (Σχήμα 5c). Αυτοί με την σειρά τους αποτιμούν την πιθανότητα να του παρέχουν πόρους με βάση το κόστος της Εξίσωσης 3 και αποστέλλουν τις αντίστοιχες απαντήσεις (Σχήμα 5d). Έπειτα, ο Αρχικός πυρήνας αποτιμά ποια από αυτές τις προτάσεις είναι η καλύτερη και την αποδέχεται, απορρίπτοντας ταυτόχρονα όλες τις άλλες. Οι τελευταίες του ενέργειες είναι να γνωστοποιήσει το αποτέλεσμα της αναζήτησης στον καινούριο Διαχειριστή πυρήνα της εφαρμογής 2 (Σχήμα 5e) καθώς και να ενημερώσει τους υπολοίπους για την αποδοχή ή απόρριψη των προσφορών τους (Σχήμα 5f).

Για την υλοποίηση και την πειραματική επικύρωση της αποδοτικότητας του DRTRM, έγινε χρήση της πολυπύρηνης πλατφόρμας Intel Single-chip Cloud Computer (Intel SCC) [114]. Η Intel SCC είναι μια πλατφόρμα με 48 πυρήνες συνδεμένους σε ένα πλέγμα Δικτύου-σε-Ψηφίδα για την μεταξύ τους επικοινωνία. Οι επεξεργαστές είναι οργανωμένοι σε δυάδες, ενώ κάθε ένας από αυτού είναι ένας δεύτερης γενιάς Intel Pentium επεξεργαστής επί του οποίου τρέχει ένα πολύ βασικό λειτουργικό Linux. Κάθε επεξεργαστής έχει μια προσωπική κρυφή μνήμη πρώτου επιπέδου L1 cache για δεδομένα και εντολές μεγέθους 16 KB, ενώ και οι δύο επεξεργαστές της δυάδας μοιράζονται μια κρυφή μνήμη δευτέρου επιπέδου L2 cache μεγέθους 256 KB καθώς και 16 KB μια γρήγορης κοινής μνήμης πάνω στην ίδια ψηφίδα, η οποία ονομάζεται Message Passing Buffer (MPB). Η μνήμη αυτή παρέχει ένα γρήγορο και αξιόπιστο τρόπο για ανταλλαγή μηνυμάτων και



Σχήμα 6: Σχηματική απεικόνιση της Intel SCC platform [158].

δεδομένων μεταξύ των επεξεργαστών της πλατφόρμας, η οποία συνολικά διαθέτει 384 KB τέτοιας μνήμης.

Επιπρόσθετα, η πλατφόρμα διαθέτει 4 ελεγκτές μνήμης οι οποίοι παρέχουν πρόσβασης σε μια εξωτερική δυναμική μνήμη DDR3 DRAM μεγέθους έως και 64 GB, η οποία είναι ορατή από όλους τους επεξεργαστές της πλατφόρμας. Όταν ένα μήνυμα στέλνεται από τον έναν πυρήνα στον άλλον, τα δεδομένα ταξιδεύουν μέσω του MPB επί της ψηφίδας. Παρότι η πλατφόρμα δεν διαθέτει κάποιον μηχανισμό στο υλικό ώστε να παρέχει συνάφεια στην εικόνα της μνήμης από όλους τους επεξεργαστές, παρέχει στις εντολές του επεξεργαστή ένα ειδικό τύπο μνήμης που φροντίζει η ανάγνωση από τον MPB, να περιέχει πάντα έγκυρα δεδομένα. Η σχηματική άποψη της εν λόγω πλατφόρμας παρουσιάζεται στο Σχήμα 6.

Η αξιολόγηση των προτεινόμενων τεχνικών έγινε με χρήση του μοντέλου εύπλαστων εφαρμογών όπως παρουσιάζεται στην Εξίσωση 1, αλλά και μέσω ενός συνόλου πραγματικών εφαρμογών που αναπτύχθηκαν και παρουσιάζουν παρόμοια συμπεριφορά με αυτή που επιτάσσει το μοντέλο. Οι εφαρμογές που επιλέχθηκαν έχουν το κατάλληλο προφίλ στους υπολογισμούς που πραγματοποιούν ώστε αφενός να μπορούν να εκτελεστούν παράλληλα και αφετέρου να προσεγγίζουν το προαναφερθέν μοντέλο. Πιο συγκεκριμένα οι εφαρμογές αυτές είναι:

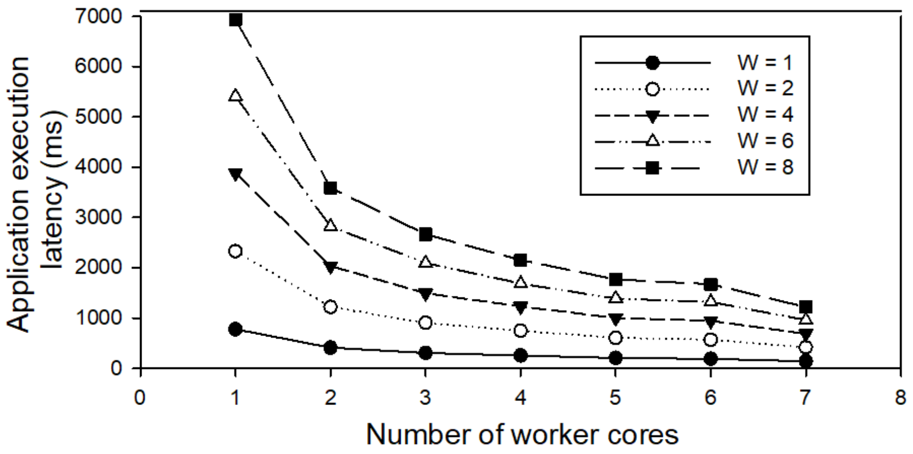
- **Πολλαπλασιασμός πίνακα με διάνυσμα (Matrix-Vector Multiplication (MVM))** που λαμβάνει ως είσοδο έναν δισδιάστο πίνακα ακεραίων αριθμών μεγέθους $A_{N \times N}$ και ένα διάνυσμα ακέραιων αριθμών $V = [v_1, v_2, \cdots v_N]^T$ και παράγει το αποτέλεσμα του πολλαπλασιασμού τους. Η είσοδος της εφαρμογής είναι ένας πυκνός πίνακας ακεραίων μεγέθους 4096 $\times$ 4096 και ένα διάνυσμα μεγέθους 4096 $\times$ 1 ακεραίων.

- **Ταξινομητής μηχανής υποστηρικτικών διανυσμάτων (Support Vector Machines Classifier (SVM))** [109] που είναι ένας πολύ δημοφιλής αλγόριθμος αναγνώρισης μοτίβων (pattern recognition) που βασίζεται στην μηχανική μάθηση (machine learning) και μπορεί να αντιμετωπίσει αποδοτικά σύνθετα μη-γραμμικά προβλήματα. Ο πηγαίος κώδικας της εφαρμογής αντλήθηκε από την ευρέως χρησιμοποιούμενη βιβλιοθήκη ανοιχτού κώδικα με όνομα libSVM [53]. Η είσοδος της εφαρμογής περιλαμβάνει έναν ταξινομητή αποτελούμενο από 4096 υποστηρικτικά διανύσματα (support vectors) κάθε ένα από τα οποία περιέχει 4096 χαρακτηριστικά εκπεφρασμένα ως αριθμούς κινητής υποδιαστολής.
- **Ταχύς μετασχηματισμός Φουριέ (Fast Fourier Transform (FFT))** που αποτελεί μια συνήθη επιλογή για το στάδιο εξαγωγής χαρακτηριστικών feature extraction σε ηλεκτρονικές εφαρμογές. Ο πηγαίος κώδικας αντλήθηκε από την σουίτα Parsec benchmark suite [39]. Η είσοδος της εφαρμογής είναι ένα σήμα 65536 σημείων κινητής υποδιαστολής.

Με σκοπό να δημιουργηθούν εφαρμογές που μπορούν να παραμετροποιηθούν αναφορικά με την ένταση των απαιτούμενων υπολογισμών, οι βασικοί υπολογισμοί της κάθε εφαρμογής (πχ πολλαπλασιασμός) επαναλαμβάνονται W φορές και αυτή η μεταβλητή ορίζεται ως ο φόρτος εργασίας (workload) της εφαρμογής. Το προφίλ της εφαρμογής σε σχέση με την επιτάχυνση της ως προς τον φόρτο εργασίας και τον αριθμό των εργατών, προσδιορίζεται με εκτεταμένες μετρήσεις της εφαρμογής (profiling) πάνω στην τελική πλατφόρμα.

Το Σχήμα 7 παρουσιάζει την αντίστοιχη ανάλυση για την εφαρμογή πολλαπλασιασμού πίνακα με διάνυσμα (MVM) στην πλατφόρμα Intel SCC. Παρατηρούμε ότι η κλιμάκωση (scaling) της εφαρμογής είναι ανεξάρτητη του φόρτου εργασίας W και ακολουθεί την απαίτηση για γραμμικότητα μεταξύ του φόρτου εργασίας και του εναπομείναντα χρόνου εκτέλεσης όπως απαιτείται από τις Εξισώσεις 1 και 2. Παρόμοια συμπεριφορά παρατηρείται και για τις άλλες δύο εφαρμογές που αναπτύχθηκαν. Οι πληροφορίες αυτές χρησιμοποιούνται κατά την διάρκεια του χρόνου εκτέλεσης ώστε να παίρνονται οι κατάλληλες αποφάσεις αναφορικά με την διαπραγμάτευση των υπολογιστικών πόρων.

Η πειραματική αξιολόγηση του DRTRM πραγματοποιήθηκε σε σχέση με άλλους παρόμοιους κατανεμημένους διαχειριστές πόρων. Πιο συγκεκριμένα, πραγματοποιήθηκε σύγκριση 1) με τον *DistRM* [130], ο οποίος αναθέτει δυναμικά σε κατανεμημένες οντότητες την ευθύνη να χαρτογραφήσουν συνεργατικά τις εισερχόμενες εφαρμογές και 2) με τον κατανεμημένο διαχειριστή *DRM*, που παρουσιάζεται στο [21] που αποτελεί έναν επίσης κατανεμημένο διαχειριστή πόρων



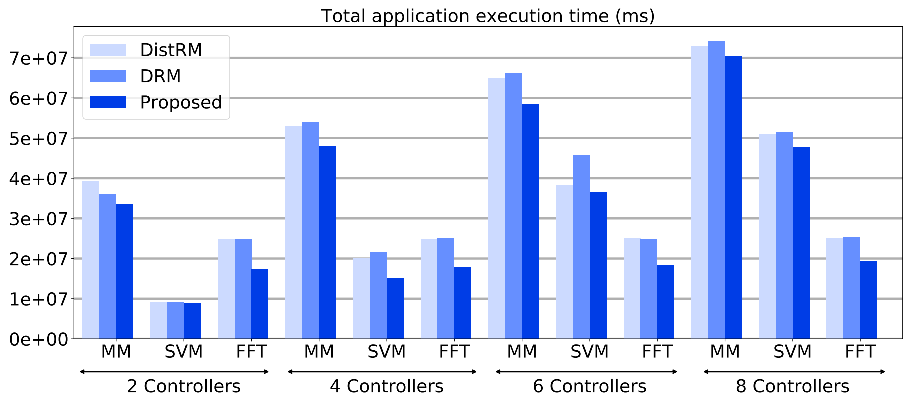
Σχήμα 7: Χρόνος εκτέλεσης της εφαρμογής πολλαπλασιασμού πίνακα σε σχέση με τον φόρτο εργασίας W και τον αριθμό των εργατών πυρήνων.

που περιέχει διαδικασίες αυτό-οργάνωσης και αυτό-βελτίωσης με σκοπό την διαδοχική βέλτιστη χαρτογράφηση των εφαρμογών. Η σύγκριση των διαφορετικών διαχειριστών πραγματοποιείται αναφορικά με διάφορα σενάρια εισερχόμενων εφαρμογών καθώς και εσωτερικών διαμορφώσεων των διαχειριστών. Ένα σενάριο θεωρείται επιτυχώς ολοκληρωμένο όταν όλες οι εφαρμογές έχουν εισέλθει και εκτελεστεί πλήρως στο σύστημα. Η μετρική σύγκρισης των διαφορετικών διαχειριστών είναι ουσιαστικά το πόσο καλή χαρτογράφηση των εφαρμογών πραγματοποιήθηκε ώστε να ελαχιστοποιηθεί ο χρόνος εκτέλεσης των εφαρμογών ή να μεγιστοποιηθεί η επιτάχυνσή τους. Όλα τα πειράματα πραγματοποιήθηκαν στην πλατφόρμα Intel SCC.

Τα πρώτα αποτελέσματα που παρουσιάζονται αφορούν την εκτέλεση 128 εφαρμογών λαμβάνοντας ταυτόχρονα υπόψιν μεταβλητό αριθμό Ελεγκτών πυρήνων. Τα αποτελέσματα παρουσιάζονται ομαδοποιημένα ανάλογα με τον αριθμό των Ελεγκτών πυρήνων και τον τύπο της παράλληλης εφαρμογής που χρησιμοποιήθηκε. Ο αριθμός των Ελεγκτών πυρήνων εκ φύσεως δημιουργεί μια κατάσταση συμβιβασμού (tradeoff), με δεδομένο ότι δεν συμμετέχουν στην εκτέλεση του υπολογιστικού φόρτου των εφαρμογών. Η αύξηση του αριθμού των Ελεγκτών συνεπάγεται ότι είναι υπεύθυνοι για την παρακολούθηση μικρότερων περιοχών και αυξάνεται ο αριθμός των αιτημάτων πληροφορίας από εφαρμογές που μπορούν να εξυπηρετηθούν παράλληλα. Στον αντίποδα, η αύξηση του αριθμού αυτού συνεπάγεται μείωση των διαθέσιμων εργατών πυρήνων στο σύστημα.

Στα παρακάτω πειράματα έγινε αξιολόγηση διαμορφώσεων με 2, 4, 6 και 8 Ελεγκτές πυρήνες. Η περίπτωση ενός μόνο Ελεγκτή δεν αξιολογήθηκε καθώς θεωρείται μια υποπερίπτωση κεντρικής διαχείρισης της πληροφορίας. Η χωρική κατανομή των Ελεγκτών έχει επιλεγεί ώστε να παρακολουθούν όσο το δυνατόν μεγάλες συνεχείς περιοχές και όχι πολλές μικρές και κατατμημένες.

Η γενική εποπτεία των αποτελεσμάτων που παρουσιάζονται στο Σχήμα 8 φανερώνει την ικανότητα του προτεινομένου τρόπου διαχείρισης στο να παίρνει καλύτερες αποφάσεις για την κατανομή των επεξεργαστών στις εφαρμογές κάτι που αποτυπώνεται από τον μειωμένο συνολικό χρόνο εκτέλεσης των εφαρμογών. Επιπρόσθετα, ο σχετικά μικρός αριθμός διαθέσιμων επεξεργαστών της πλατφόρμας, ενισχύει την αποτελεσματικότητα και την σημασία του προτεινόμενου αλγορίθμου διαπραγμάτευσης πόρων, ο οποίος αποφεύγει άσκοπες μεταφορές πόρων από την μια εφαρμογή στην άλλη και αφήνει ανεπηρέαστες τις εφαρμογές να ολοκληρώσουν νωρίτερα τους υπολογισμούς και τον κύκλο ζωής τους.

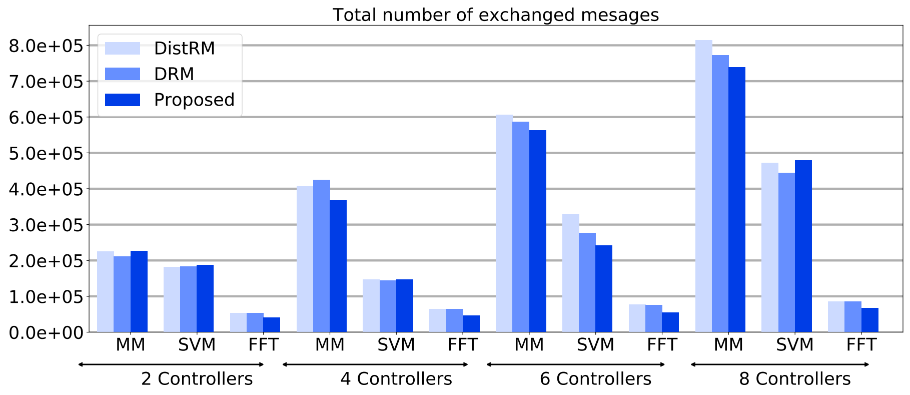


Σχήμα 8: Συνολικός χρόνος εκτέλεσης των εφαρμογών κατά την διάρκεια εκτέλεσης ενός σεναρίου.

Αναφορικά με την συμπεριφορά των εφαρμογών, παρατηρείται ότι ο ταξινομητής SVM είναι η πιο υπολογιστικά ελαφριά εφαρμογή, ενώ ο πολλαπλασιασμός πίνακα με διάνυσμα είναι η πιο απαιτητική με τον μεγαλύτερο χρόνο εκτέλεσης. Στις δύο αυτές εφαρμογές, υπάρχει ξεκάθαρη συσχέτιση μεταξύ του αριθμού των Ελεγκτών πυρήνων και του αυξημένου χρόνου εκτέλεσης. Πιο συγκεκριμένα, η μείωση του φόρτου εργασίας ανά Ελεγκτή όταν αυξάνεται ο αριθμός τους, δεν είναι δυνατό να εξισορροπήσει την μείωση των διαθέσιμων πυρήνων εργατών στο σύστημα.

Κατ' αντιστοιχία στην εφαρμογή μετασχηματισμού Φουριέ η οποία παρουσιάζει μικρή κλιμάκωση σε σχέση με τον αριθμό των πυρήνων, το φαινόμενο αυτό δεν είναι το ίδιο εμφανές. Η εν λόγω εφαρμογή επηρεάζεται άμεσα από την χωρική κατανομή των Αρχικών πυρήνων, κάτι στο οποίο η προτεινόμενη στρατηγική υπερτερεί και ως εκ τούτου οδηγεί σε μειωμένο χρόνο εκτέλεσης της εφαρμογής. Σε κάθε περίπτωση, η εξερεύνηση της συμπεριφοράς του κατανεμημένου διαχειριστή πόρων σε σχέση με τον αριθμό των Ελεγκτών πυρήνων είναι σημαντική καθώς επιβεβαιώνει την διατήρηση της αποδοτικότητας του διαχειριστή υπό διαρκώς μειούμενους διαθέσιμους πόρους. Συνοψίζοντας, η προτεινόμενη τεχνική οδηγεί σε κατά μέσο όρο 17.5% ταχύτερη εκτέλεση των εφαρμογών σε σχέση με τις υπόλοιπες συγκρινόμενες τεχνικές, ενώ αυτή η τιμή αγγίζει ένα μέγιστο της τάξης του 30%.

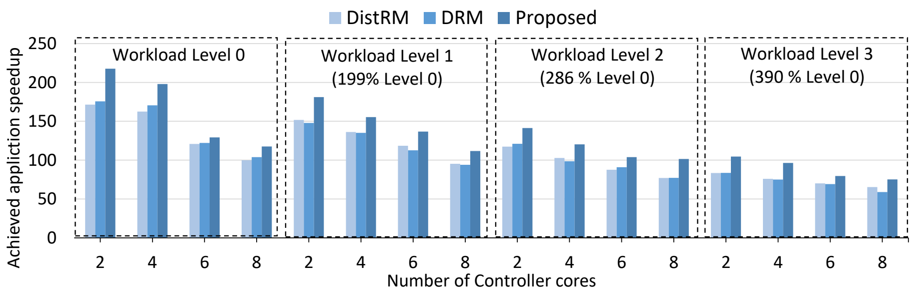
Το Σχήμα 9 συνοψίζει την επικοινωνιακή πλευρά της υπό παρουσίαση αξιολόγησης δείχνοντας ότι στην συντριπτική πλειονότητα των περιπτώσεων, η προτεινόμενη τεχνική καταφέρνει να παράγει λιγότερη επικοινωνιακή κυκλοφορία εν συγκρίσει με τις υπόλοιπες τεχνικές, φτάνοντας μέχρι μια μέση μείωση της τάξεως του 12.5% στα συνολικά μηνύματα που ανταλλάσσονται μεταξύ των κατανεμημένων πρακτόρων κατά την διάρκεια της εκτέλεσης ενός σεναρίου.



Σχήμα 9: Συνολικός αριθμός μηνυμάτων που ανταλλάχθηκαν κατά την διάρκεια εκτέλεσης ενός σεναρίου.

Η συμπεριφορά αυτή παρατηρείται επειδή περισσότεροι πυρήνες κατανέμονται για την εκτέλεση του φόρτου εργασίας των εφαρμογών, κάτι το οποίο χαρακτηρίζεται από πληθώρα υπολογισμών και όχι μηνυμάτων. Επιπρόσθετα, ο αριθμός των ανταλλασσόμενων μηνυμάτων σχετίζεται άμεσα με τον απαιτούμενο χρόνο εκτέλεσης των εφαρμογών καθώς η ταυτόχρονη ύπαρξη πολλών εφαρμογών στο σύστημα οδηγεί σε αυξημένη επικοινωνία μεταξύ των κατανεμημένων πρακτόρων με σκοπό την διαπραγμάτευση των πόρων του συστήματος.

Με σκοπό την περαιτέρω αξιολόγηση του προτεινόμενου κατανεμημένου διαχειριστή, πραγματοποιήσαμε μια σειρά πειραμάτων κάνοντας χρήση ενός μείγματος εφαρμογών οι οποίες χαρακτηρίζονται από ποικίλα χαρακτηριστικά αναφορικά με τον απαιτούμενο φόρτο εργασίας τους. Το μείγμα αυτό δημιουργήθηκε κάνοντας



Σχήμα 10: Αξιολόγηση της ποιότητας κατανομής πόρων για εφαρμογές που παράγονται με βάση το μοντέλο παράλληλων εύπλαστων εφαρμογών.

χρήση του μοντέλου εύπλαστων παράλληλων εφαρμογών [84], μέσω του οποίου παρήχθησαν εφαρμογές με τυχαία χαρακτηριστικά μέσω δειγματοληψίας του συνόλου τιμών 0.01 ως 100 για την παράμετρο σ και 2 εως 16 για την παράμετρο A. Σύμφωνα με αυτά τα χαρακτηριστικά και τον δυναμικά μεταβαλλόμενο αριθμό των πυρήνων εργατών η επιτάχυνση των εφαρμογών υπολογίζεται κάνοντας χρήση της Εξίσωσης 2.

Η αξιολόγηση των διαφορετικών διαχειριστών πόρων οφείλει επίσης να πραγματοποιηθεί για διαφορετικά επίπεδα έντασης του μέσου φόρτου εργασίας των εφαρμογών. Δημιουργήθηκαν 4 διαφορετικά τέτοια επίπεδα, όπου οι τιμές του φόρτου εργασίας προσδιορίζονται με χρήση μιας γεννήτριας τυχαίων αριθμών βασισμένη στην κανονική κατανομή. Παρέχοντας αυξανόμενες τιμές εισόδου στην γεννήτρια αυτή, καθίσταται δυνατή η αύξηση της μέσης τιμής του φόρτου εργασίας των εφαρμογών.

Τα αποτελέσματα της αξιολόγησης των τριών διαφορετικών διαχειριστών για ένα μείγμα 64 τεχνητών εύπλαστων εφαρμογών παρουσιάζονται στο Σχήμα 10. Η μετρική αξιολόγησης είναι η συνολική επιτάχυνση που επετεύχθη στις εφαρμογές, ως αποτέλεσμα της εκάστοτε διαχείρισης πόρων. Τα αποτελέσματα έχουν ομαδοποιηθεί σύμφωνα με τα διαφορετικά επίπεδα του φόρτου εργασίας. Επίσης, για κάθε επίπεδο γίνεται αξιολόγηση με βάση όλες τις διαφορετικές διαμορφώσεις Ελεγκτών πυρήνων όπως παρουσιάστηκαν στα προηγούμενα πειράματα.

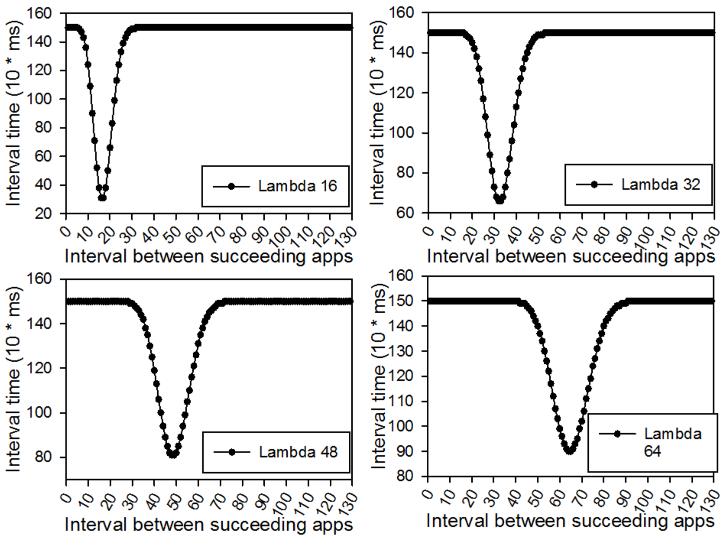
Παρατηρούμε ότι η προτεινόμενη μεθοδολογία οδηγεί σε αυξημένη επιτάχυνση των εφαρμογών σε όλες τις περιπτώσεις, με μέσα κέρδη πέραν του 20% που αγγίζουν μέχρι το 30%. Η ποικιλία των χαρακτηριστικών κλιμάκωσης των εύπλαστων εφαρμογών που αποτελούν την είσοδο των υπό εξέταση σεναρίων, μεγιστοποιεί τα οφέλη του προτεινόμενου αλγορίθμου διαπραγμάτευσης πόρων, επιτρέποντας στις εφαρμογές με μικρή κλιμάκωση να ολοκληρώσουν τον φόρτο εργασίας τους γρήγορα χωρίς εξωτερικές παρεμβολές. Έπειτα, οι πόροι που ελευθερώνονται μπορούν να χρησιμοποιηθούν από τις εφαρμογές με δυνατότητα υψηλής κλιμάκωσης μεγιστοποιώντας κατά αυτό τον τρόπο την επιτάχυνσή τους. Πέραν τούτου, η εν λόγω πειραματική αξιολόγηση επιβεβαιώνει την σταθερή ικανότητα του προτεινόμενου διαχειριστή να παρέχει αποδοτική διαχείριση πόρων ανεξάρτητα από την ένταση του φόρτου εργασίας των εκτελούμενων εφαρμογών. Τέλος, μια ακόμα σημαντική παρατήρηση είναι ότι η μείωση του αριθμό των διαθέσιμων εργατών πυρήνων (μέσω της αύξησης του αριθμού των Ελεγκτών πυρήνων), επιφέρει μεγάλο πλήγμα στην συνολική επιτάχυνση καθώς οι υψηλά κλιμακώσιμες εφαρμογές δεν μπορούν να φτάσουν την μέγιστη επιτάχυνση τους.

Η μέχρι τώρα ανάλυση αξιολόγησε την ποιότητα της κατανομής πόρων για διαφορετικές διαμόρφωσης του κατανεμημένου διαχειριστή (αριθμός Ελεγκτών πυρήνων), διαφορετικό αριθμό και είδος εφαρμογών καθώς και διαφορετικά επίπεδα φόρτου εργασίας. Ωστόσο, σε ρεαλιστικά σενάρια μεγάλης κλίμακας σπουδαίο ρόλο παίζει και ο ρυθμός έλευσης των εφαρμογών στο σύστημα. Ο ρυθμός αυτός είναι κρίσιμος καθώς μπορεί να οδηγήσει την πλατφόρμα σε οριακές καταστάσεις αν πληθώρα εφαρμογών παρουσιαστούν αναπάντεχα και δεν υπάρχει αντίστοιχο σύστημα διαχείρισης.

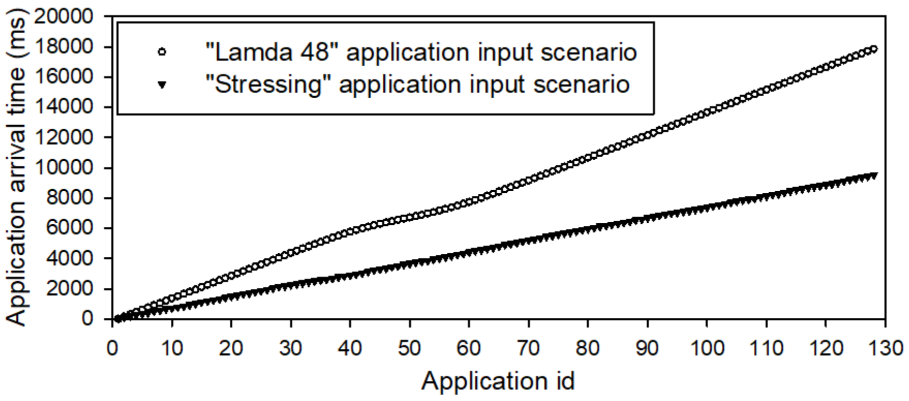
Στα πλαίσια αυτής της ανάλυσης, δημιουργήθηκε ένας αριθμός σεναρίων έλευσης εφαρμογών στο σύστημα, όπου διαφορετικά σενάρια παρουσιάζουν διαφορετικές απαιτήσεις σε πόρους. Ως βασική ομάδα σεναρίων έλευσης εφαρμογών θεωρούμε αυτά που τα χρονικά διαστήματα μεταξύ διαδοχικών εφαρμογών προκύπτουν με χρήση κατανομής Poisson. Παρήχθησαν 4 διαφορετικά τέτοια σενάρια μέσω τροποποίησης της παραμέτρου λ της κατανομής, στην οποία δόθηκαν οι τιμές 16, 32, 48 όπως και στην εργασία των [179].

Οι αντίστοιχες καμπύλές έλευσης εφαρμογών που παρήχθησαν παρουσιάζονται στο Σχήμα 11, όπου ο X άξονας αναφέρεται στο μοναδικό αναγνωριστικό της εφαρμογής id, ενώ ο Υ άξονας αναπαριστά το χρονικό διάστημα που περνά μεταξύ της έλευσης δυο οποιονδήποτε διαδοχικών εφαρμογών. Στην πράξη, τα εν λόγω σενάρια αντιπροσωπεύουν την κατάσταση ενός συστήματος όπου οι εφαρμογές φτάνουν με σταθερό ρυθμό και σε κάποια χρονική στιγμή ο ρυθμός αυτός αυξάνεται κατακόρυφα. Η διαφορά μεταξύ των τεσσάρων σεναρίων είναι η ακριβής χρονική στιγμή που παρατηρείται η εντατικοποίηση του ρυθμού εισόδου.

Πέρα από τα σενάρια που βασίζονται στην κατανομή Poisson, δημιουργήθηκε ένα σενάριο το οποίο εξ ορισμού είναι πολύ απαιτητικό σε πόρους σε σχέση με τους διαθέσιμους πόρους του συστήματος. Το σενάριο αυτό ονομάζεται "Stressing" και παρήχθη μέσω μιας γεννήτριας τυχαίων αριθμών. Στο Σχήμα 12 παρουσιάζεται ο ακριβής χρόνος έλευσης των εφαρμογών στο απαιτητικό σενάριο εν συγκρίσει με το σενάριο που παράγεται από την κατανομή Poisson με λ παράμετρο ίση με 48. Παρατηρούμε ότι στο απαιτητικό σενάριο, όλες οι εφαρμογές απαιτούν είσοδο στο σύστημα, σχεδόν στον μισό χρόνο συγκριτικά με το άλλο σενάριο.

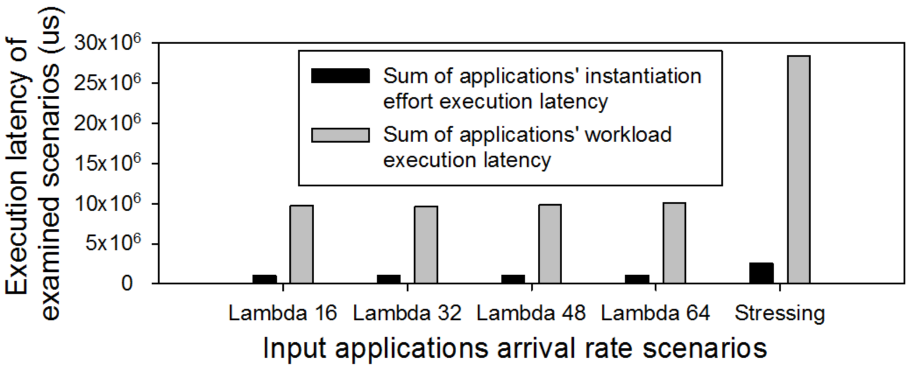


Σχήμα 11: Χρονικά διαστήματα έλευσης διαδοχικών εφαρμογών στο σύστημα.



Σχήμα 12: Χρόνοι άφιξης των εφαρμογών στο σύστημα.

Τα 5 διαφορετικά σενάρια έλευσης εφαρμογών που δημιουργήθηκαν, αποτιμώνται πειραματικά στην πλατφόρμα Intel SCC υπό την διαχείριση του προτεινόμενου κατανεμημένου διαχειριστή πόρων. Ως τύπος εφαρμογής εισόδου χρησιμοποιήθηκε η υλοποιημένη παράλληλη εφαρμογή πολλαπλασιασμού πίνακα με διάνυσμα. Συγκρίνοντας τον χρόνο εκτέλεσης των εφαρμογών του απαιτητικού σεναρίου συγκριτικά με τα υπόλοιπα, όπως φαίνεται στο Σχήμα 13, παρατηρούμε ότι όπως αναμενόταν υπάρχει μια πολύ μεγάλη αύξηση του χρόνου εκτέλεσης των



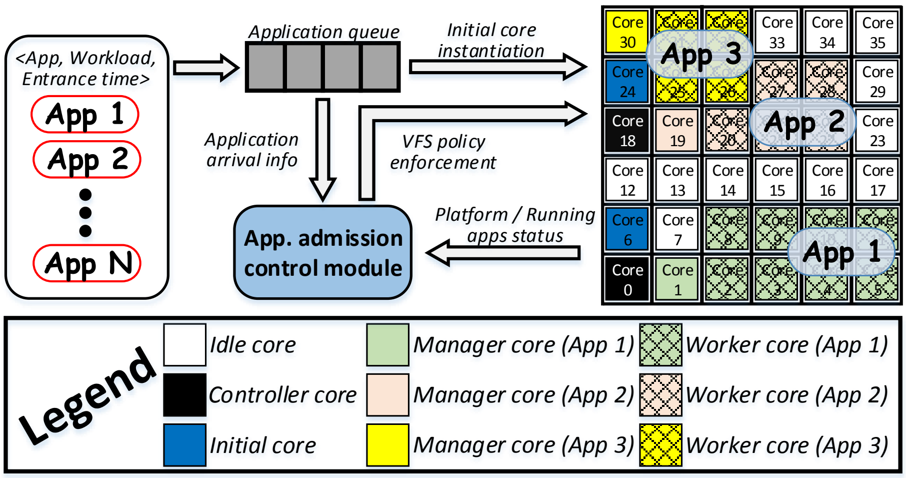
Σχήμα 13: Συμπεριφορά του κατανεμημένου διαχειριστή πόρων για διαφορετικά σενάρια έλευσης εφαρμογών στο σύστημα.

εφαρμογών που φτάνει το 190% κατά μέσο όρο.

Επιπρόσθετα, το Σχήμα περιλαμβάνει μια δεύτερη μετρημένη ποσότητα με το όνομα "Sum of applications' instantiation effort execution latency", η οποία δίνει μια ποσοτική εικόνα του χρόνου που χρειάστηκε ώστε οι εφαρμογές να αρχικοποιηθούν στο σύστημα. Βλέπουμε ότι και στην περίπτωση αυτής της μέτρησης, υπάρχει μια πολύ μεγάλη αύξηση 142% κατά μέσο όρο στην περίπτωση του απαιτητικού σεναρίου συγκριτικά με τα υπόλοιπα. Αυτό δείχνει ότι παρά το μεγάλο έλλειμα ελεύθερων πόρων στο σύστημα, οι Αρχικοί πυρήνες συνέχιζαν να ψάχνουν παρά την πολύ μικρή πιθανότητα να βρεθούν ελεύθεροι πόροι.

Αυτή η συμπεριφορά του κατανεμημένου διαχειριστή πόρων είναι μη αποδοτική καθώς επαναλαμβάνεται άσκοπα η αναζήτηση πυρήνων, παρεμποδίζοντας παράλληλα με αιτήματα τις εφαρμογές που τρέχουν. Τα αιτήματα αυτά καθυστερούν την ολοκλήρωση των εφαρμογών και έτσι εντείνουν το πρόβλημα της έλλειψης πόρων στο σύστημα, δημιουργώντας έναν φαύλο κύκλο. Επομένως, ο παρουσιαζόμενος διαχειριστής πόρων επεκτάθηκε με στόχο να επιλύσει αυτό το έλλειμα αποδοτικότητας παίρνοντας την μορφή που παρουσιάζεται στο Σχήμα 14.

Πιο συγκεκριμένα, στο Σχήμα 14 φαίνεται ότι το ανανεωμένο σύστημα περιέχει τα εξής στοιχεία. Ένα σενάριο εφαρμογών εισόδου, που περιέχει πληροφορίες σχετικά με την ακριβή χρονική στιγμή εισόδου καθώς και τα χαρακτηριστικά της εφαρμογής. Μια ουρά εφαρμογών στην οποία τοποθετούνται οι εφαρμογές έως ότου ξεκινήσει η διαδικασία αρχικοποίησης τους στο σύστημα. Ένα τμήμα του DRTRM που είναι υπεύθυνο για την αρχικοποίηση και εκτέλεση των εφαρμογών στο τελικό πολυπύρηνο σύστημα. Τέλος, ένα διαχειριστικό τμήμα (admission control module) είναι απαραίτητο ώστε να μεταβάλλεται ο ρυθμός με τον οποίον αποστέλλονται οι εφαρμογές από την ουρά στο σύστημα για αρχικοποίηση.



Σχήμα 14: Σχηματική αναπαράσταση του προτεινόμενου κατανεμημένου διαχειριστή πόρων εμπλουτισμένο με ανάδραση σχετική με τον ρυθμό έλευσης εφαρμογών στο σύστημα.

Η προτεινόμενη προσέγγιση υλοποιεί έναν βρόχο ανάδρασης στο σύστημα, κάτι το οποίο λείπει από τον μέχρι τώρα σχεδιασμό του DRTRM. Ο εν λόγω μηχανισμός ανάδρασης έχει στόχο την συλλογή πληροφορίων της τρέχουσας κατάστασης του συστήματος και του ρυθμού εισόδου εφαρμογών με απώτερο σκοπό την δυναμική αναδιοργάνωση τόσο του DRTRM όσο και του ρυθμού αποστολής εφαρμογών από την ουρά αναμονής στο σύστημα, με στόχο την ελάφρυνση της πλατφόρμας σε περίπτωση που όλοι οι πόροι έχουν κατανεμηθεί στις εκτελούμενες εφαρμογές.

Ιδανικά, η προαναφερθείσα προσαρμοστική συμπεριφορά του DRTRM θα μπορούσε να μεταφραστεί σε μια πολιτική στο επίπεδο του Αρχικού πυρήνα. Για παράδειγμα η επανάληψη της αναζήτησης του για πόρους θα μπορούσε να προσαρμόζεται σύμφωνα με το μέγεθος της χρήσης των πόρων του συστήματος. Με αυτό τον τρόπο οι Αρχικοί πυρήνες θα ήλεγχαν τον ρυθμό με τον οποίον γίνεται προσπάθεια αρχικοποίησης νέων εφαρμογών στο σύστημα. Στην πράξη όμως, αυτή η σχεδιαστική επιλογή χαρακτηρίζεται από δύο σημαντικά μειονεκτήματα, συνυπολογίζοντας την πηγαία κατανεμημένη φύση του DRTRM:

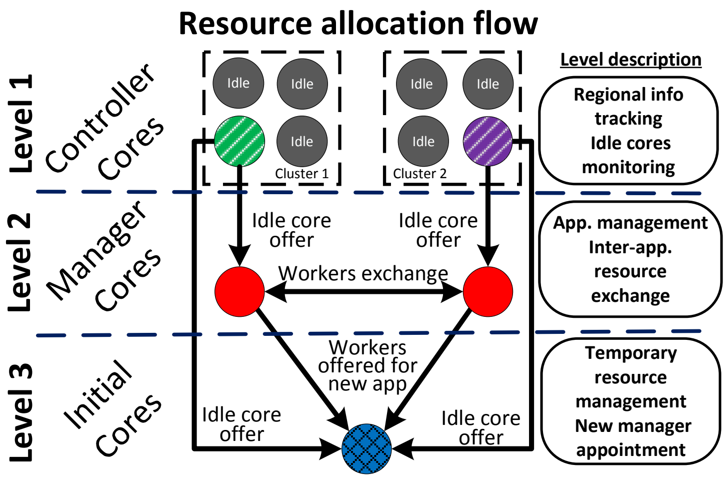
- Απαιτεί μια κεντρική απόφαση ώστε να εξασφαλιστεί η αποδοτική προσαρμογή του DRTRM σε σχέση με το ρυθμό εισόδου των εφαρμογών. Η κεντρική συγκέντρωση της πληροφορίας και της απόφασης έρχεται σε ευθεία αντίθεση με την κατανεμημένη φύση του διαχειριστή πόρων και ως εκ τούτου είναι μη αποδεκτή.

- Η λήψη μιας τέτοιας απόφασης στο επίπεδο των Αρχικών πυρήνων, χαρακτηρίζεται από μεγάλη χρονική διάρκεια συγχρονισμού. Ακόμη και αν ληφθεί κεντρικά η απόφαση, πρέπει να μεταδοθεί ένα σήμα σε όλη την πλατφόρμα ώστε να απαντήσουν όλοι οι Αρχικοί πυρήνες για την κατάσταση τους και κατόπιν της απόφασης πρέπει εκ νέου όλοι να ενημερωθούν. Οι χρονικές απαιτήσεις αυτού του εγχειρήματος είναι υπερβολικά μεγάλες, μη παρέχοντας την δυνατότητα στο σύστημα να είναι αποδοτικά προσαρμοστικό, καθώς την στιγμή που θα εφαρμόζεται η νέα πολιτική η κατάσταση του συστήματος μπορεί να είναι πολύ διαφορετική από την στιγμή που ξεκίνησε η διαδικασία λήψης απόφασης.

Με στόχο να προταθεί μια αμιγώς κατανεμημένη πολιτική, θα γίνει χρήση της ιεραρχίας κατανομής πόρων του DRTRM, όπως παρουσιάζεται στο Σχήμα 15, ώστε να περιοριστεί έμμεσα η λειτουργία των Αρχικών πυρήνων και κατ' επέκταση η έντασης της αρχικοποίησης των εφαρμογών στο σύστημα. Βλέπουμε στο Σχήμα 15, που παρουσιάζει την ιεραρχία, ότι η λειτουργία των Αρχικών πυρήνων είναι άρρηκτα συνδεδεμένη με πληροφορίες που του παρέχουν οι Ελεγκτές πυρήνες. Επομένως, η προτεινόμενη τεχνική μείωσης της έντασης της αναζήτησης πόρων από τους Αρχικούς πυρήνες, θα βασιστεί στην μείωση του ρυθμού παροχής πληροφορίας από τους Ελεγκτές πυρήνες. Αυτό επιτυγχάνεται με χρήση τεχνικών κλιμάκωσης της τάσης και συχνότητας λειτουργίας (Voltage and Frequency Scaling (VFS)) των Ελεγκτών πυρήνων. Σε αντίθεση με μια πολιτική κεντρικής απόφασης, η προτεινόμενη τεχνική απαιτεί την συνεργασία μόνο των Ελεγκτών πυρήνων, οι οποίοι εκ σχεδιασμού είναι λίγοι σε αριθμό. Συνεπώς, η απαιτούμενες αποφάσεις μπορούν αφενός να ληφθούν κατανεμημένα και αφετέρου με αποδεκτή χρονική καθυστέρηση.

Η ουσία την προτεινόμενης τεχνικής βασίζεται στο γεγονός ότι η συνεργασία των κατανεμημένων οντοτήτων του DRTRM ακολουθεί μοντέλο εξυπηρετητή-πελάτη (client-server model), όπου οι Αρχικοί και Διαχειριστές πυρήνες είναι οι πελάτες ενώ ό Ελεγκτής πυρήνας είναι ο εξυπηρετητής. Με αυτό ως δεδομένο, έπεται ότι η επιβράδυνση του εξυπηρετητή θα οδηγήσει σε αυξημένους χρόνους αναμονής στους πελάτες και κατ' επέκταση αυξημένο απαιτούμενο χρόνο εκτέλεσης των εργασιών τους. Η συμπεριφορά αυτή μεταφράζεται στο επίπεδο των Αρχικών πυρήνων σε αυξημένο χρόνο αναζήτησης πόρων, αυξημένο χρόνο επανάληψης της αναζήτησης και συνολικά σε επιβράδυνση της διαδικασίας αρχικοποίησης των εφαρμογών στο σύστημα.

Στο Σχήμα 16 παρουσιάζεται αναλυτικότερα η επικοινωνία μεταξύ των Ελεγκτών πυρήνων και των υπολοίπων κατανεμημένων δραστών στο σύστημα, δείχνοντας με μεγαλύτερη λεπτομέρεια πώς η επιβράδυνση των πρώτων οδηγεί στην επιβράδυνση της λειτουργίας των δεύτερων. Ο Αρχικός πυρήνας εκτελεί την αναζήτηση για πόρους η οποία είναι άμεσα εξαρτώμενη από πληροφορίες που του παρέχει ο

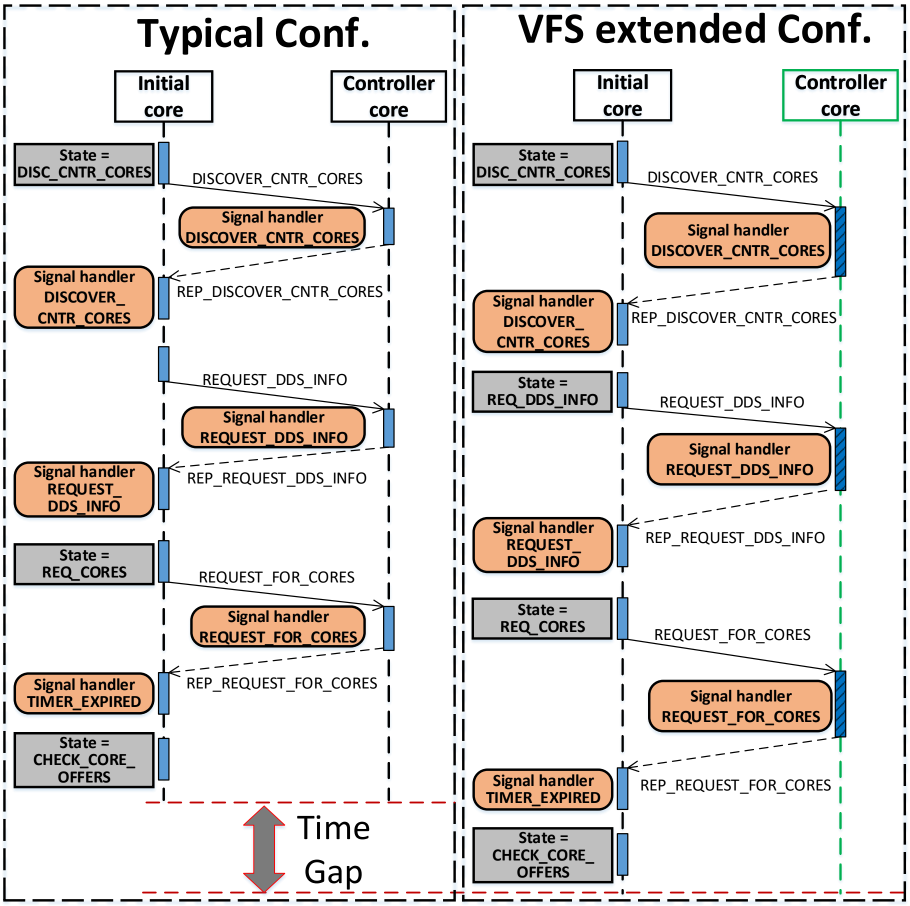

Resource allocation flow
Level 1
Controller Cores
Level 2
Manager Cores
Level 3
Initial Cores
Idle
Cluster 1
Cluster 2
Idle core offer
Workers exchange
Workers offered for new app
Level description
Regional info tracking Idle cores monitoring
App. management Inter-app. resource exchange
Temporary resource management New manager appointment

Σχήμα 15: Ιεραρχία πυρήνων και εξαρτήσεων στην κατανομή πόρων στον προτεινόμενο κατανεμημένο διαχειριστή πόρων.

Ελεγκτής πυρήνας.

Η αριστερή πλευρά του Σχήματος 16 παρουσιάζει την χρονική εξέλιξη της επικοινωνίας των δύο δραστών όταν ο Ελεγκτής πυρήνας είναι ρυθμισμένος στην τυπική συχνότητα λειτουργίας του. Η δεξιά πλευρά του Σχήματος παρουσιάζει τις ίδιες ακριβώς λειτουργίες και μηνύματα όταν η συχνότητα λειτουργίας του Ελεγκτή πυρήνα είναι μειωμένη. Η μειωμένη συχνότητα λειτουργίας οδηγεί σε αυξημένο χρόνο εκτέλεσης των εισερχόμενων αιτημάτων του Ελεγκτή πυρήνα. Ταυτόχρονα, η λειτουργία του Αρχικού πυρήνα μπλοκάρει έως ότου εξυπηρετηθούν τα αιτήματα που έχει αποστείλει στον Ελεγκτή, κάτι το οποίο εν τέλει οδηγεί στην αύξηση του χρόνου εκτέλεσης της συνολικής εργασίας που επιτελεί, που εν προκειμένω είναι η αναζήτηση πόρων για την αρχικοποίηση μιας νέας εφαρμογής. Ανάγοντας την συμπεριφορά αυτή στο σύνολο των Αρχικών πυρήνων στο σύστημα, παρατηρείται μειωμένος ρυθμός αρχικοποίησης όλων των εφαρμογών στο σύστημα.

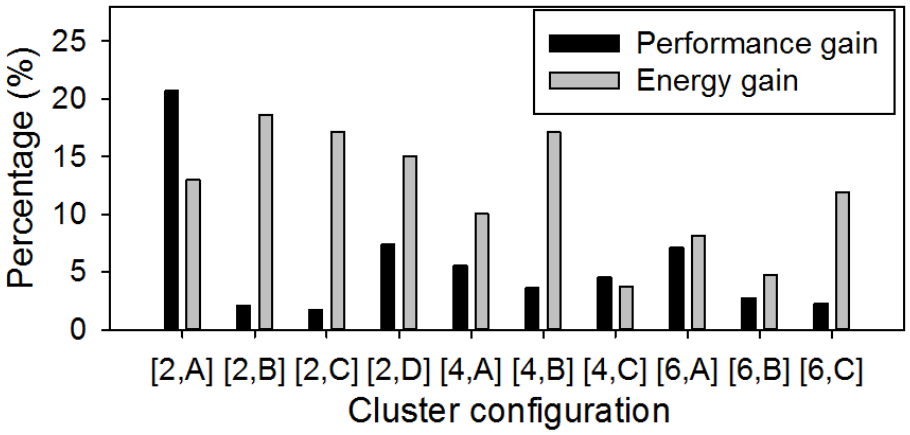
Η πειραματική αξιολόγηση της προτεινόμενης τεχνικής για προσαρμογής του διαχειριστή πόρων σε σχέση με τον ρυθμό εισόδου εφαρμογών στο σύστημα, πραγματοποιήθηκε στην πλατφόρμα Intel SCC. Η πλατφόρμα αυτή είναι χωρισμένη σε 6 νησιά τάσης (Voltage Islands), κάθε ένα από τα οποία μπορεί ξεχωριστά να τεθεί σε χαμηλότερη τάση από την ονομαστική. Στα πλαίσια αυτής της εργασίας, η μείωση της τάσης του ενός νησιού και η τοποθέτηση επί αυτού όλων των Ελεγκτών πυρήνων, ισοδυναμεί με την εφαρμογή της προτεινόμενης τεχνικής για την προσαρμοστική αρχικοποίηση εφαρμογών στο σύστημα. Η



Σχήμα 16: Αυξημένος χρόνος εκτέλεσης των Αρχικών πυρήνων στην περίπτωση που οι Ελεγκτές βρίσκονται σε καθεστώς μειωμένης συχνότητας λειτουργίας.

αξιολόγηση έγινε με βάση το απαιτητικό (Stressing) σενάριο εισόδου εφαρμογών στο σύστημα, κάνοντας χρήση της υλοποιημένης εφαρμογής πολλαπλασιασμού πίνακα με διάνυσμα καθώς και των εφαρμογών που προέρχονται από τον μοντέλο των εύπλαστων εφαρμογών. Η πλατφόρμα παρέχει επιπρόσθετα την κατάλληλη υποδομή ώστε να μπορεί να γίνεται μέτρηση της κατανάλωσης ενέργειας κατά την διάρκεια εκτέλεσης ενός σεναρίου. Η ανάλυση έγινε για σύνολο διαμορφώσεων 2, 4 και 6 Ελεγκτών πυρήνων. Σε όλες τις διαμορφώσεις οι Ελεγκτές τοποθετούνται στο πρώτο νησί τάσης, το οποίο λειτουργεί σε χαμηλότερη τάση από την τυπική.

Στο πρώτο σετ πειραμάτων αξιολογείται η αποδοτικότητα της προτεινόμενης



Σχήμα 17: Κέρδη απόδοσης και ενέργειας που προκύπτουν από την προτεινόμενη στρατηγική προσαρμογής του διαχειριστή πόρων με βάση τον ρυθμό εισόδου εφαρμογών στο σύστημα.

τεχνικής στην μείωση του συνωστισμού στην πλατφόρμα στην περίπτωση του απαιτητικού σεναρίου έλευσης εφαρμογών τύπου πολλαπλασιασμού πίνακα με διάνυσμά. Η συχνότητα λειτουργίας των Ελεγκτών πυρήνων έχει μειωθεί από 800 MHz στα 533 MHz, μέσω μείωσης της τάσης λειτουργίας τους από τα 1.1 V στα 0.8 V. Στο Σχήμα 17 παρουσιάζονται τα κέρδη της προτεινόμενης τεχνικής σε απόδοση (αριθμό εργατών ανά εφαρμογή) και ενέργεια συγκριτικά με την απλή μορφή του κατανεμημένου διαχειριστή πόρων.

Τα αποτελέσματα παρουσιάζονται σε σχέση με την αντίστοιχη διαμόρφωση Ελεγκτών πυρήνων που χρησιμοποιείται. Όπως φαίνεται στο Σχήμα 17, σε όλες τις περιπτώσεις παρουσιάζονται κέρδη της προτεινόμενης τεχνικής, τόσο στην αποδοτικότητα της χρήσης των πόρων όσο και στην καταναλισκόμενη ενέργεια. Ενδιαφέρουσα είναι η έλλειψη συμμετρίας μεταξύ των κερδών απόδοσης και ενέργειας. Για παράδειγμα, στην διαμόρφωση [2,A] παρατηρείται 20% βελτίωση στην απόδοση συνοδευόμενη από 12% μείωση στην ενέργεια, ενώ στην διαμόρφωση [4,B] οι αντίστοιχοι αριθμοί είναι 3% και 18%.

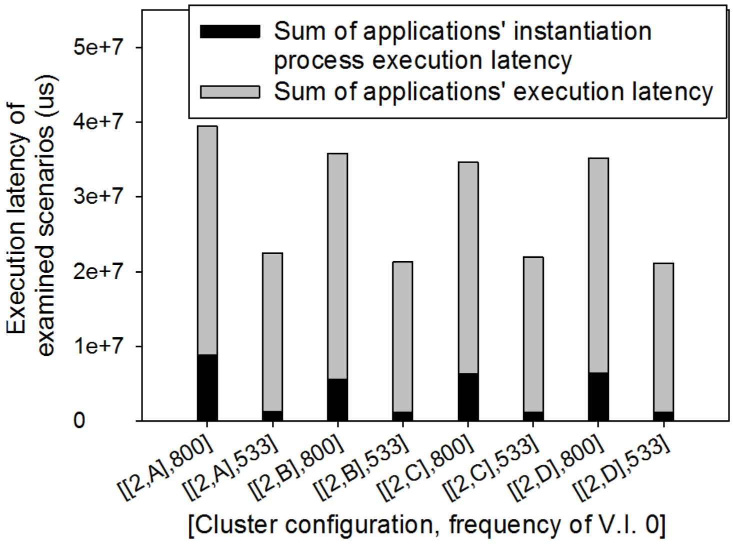
Η συμπεριφορά αυτή αποδίδεται στο γεγονός ότι η συγκεκριμένη μετρική αποδοτικότητας δεν παρέχει πληροφορία για τον βαθμό της παράλληλης εκτέλεσης των εφαρμογών του συστήματος. Η παραλληλία που επιτυγχάνεται επηρεάζει σε μεγάλο βαθμό τον συνολικό χρόνο εκτέλεσης ενός σεναρίου, κάτι που επιδρά καταλυτικά στην ενέργεια που καταναλώθηκε. Έτσι, στην περίπτωση της διαμόρφωσης [2,A] οι εφαρμογές απέκτησαν μεγάλο αριθμό πυρήνων εργατών, ο αθροιστικός τους χρόνος εκτέλεσης ήταν μικρός αλλά ταυτόχρονα εκτελέστηκαν με «σειριακό» τρόπο, ο οποίος δεν οδήγησε σε μεγάλα κέρδη σχετικά με την κατανάλωση ενέργειας. Στον αντίποδα, στην διαμόρφωση [4,B] οι εφαρμογές εκτελέστηκαν εν πολλοίς παράλληλα, γεγονός που σημαίνει ότι κατά μέσο όρο είχαν λιγότερους διαθέσιμους εργάτες. Συνολικά όμως, στην περίπτωση αυτή το πείραμα ολοκληρώθηκε συντομότερα και ως εκ τούτου τα κέρδη στην ενέργεια ήταν σαφώς μεγαλύτερα.

Το πείραμα επαναλαμβάνεται για τις διαμορφώσεις με 2 Ελεγκτές πυρήνες, κάνοντας χρήση των εφαρμογών που έχουν παραχθεί από το μοντέλο παράλληλων εύπλαστων εφαρμογών. Η επιλογή αυτή πραγματοποιήθηκε ώστε να αξιολογηθεί η προτεινόμενη τεχνική επί ενός πιο ποικίλου σετ εφαρμογών εν συγκρίσει με αυτό των εφαρμογών πολλαπλασιασμού πίνακα με διάνυσμα. Παρόλα αυτά, οι τιμές του φόρτου εργασίας W διατηρήθηκαν ίδιες και στα δύο πειράματα.

Τα αποτελέσματα του πειράματος παρουσιάζονται στο Σχήμα 18, ομαδοποιημένα ανάλογα με την διαμόρφωση των Ελεγκτών πυρήνων. Η τελευταία αριθμητική τιμή σε κάθε σύνολο τιμών στον άξονα X, υποδηλώνει την συχνότητα λειτουργίας των Ελεγκτών πυρήνων. Συχνότητα λειτουργίας ίση με 800 MHz αντιστοιχεί στον αρχικό κατανεμημένο διαχειριστή πόρων ενώ αντίστοιχα συχνότητα λειτουργίας 533 MHz στην επέκταση του η οποία λαμβάνει υπόψιν τον ρυθμό έλευσης εφαρμογών στο σύστημα. Το πείραμα επιβεβαιώνει την αποδοτικότητα της προτεινόμενης τεχνικής η οποία επιτυγχάνει να μειώσει τον συνολικό χρόνο εκτέλεσης των εφαρμογών κατά 43.8% και τον συνολικό χρόνο αρχικοποίησης τους κατά 472%, εν συγκρίσει με την αρχικό δυναμικό διαχειριστή πόρων.

Τα μεγάλα αυτά κέρδη αποδίδονται στα μεικτά χαρακτηριστικά κλιμάκωσης του σετ εφαρμογών εισόδου, τα οποία δίνουν την δυνατότητα στον DRTRM να δώσει περισσότερους εργάτες στις εφαρμογές με μεγάλη κλιμάκωση, οι οποίες έτσι ολοκληρώνουν ταχύτερα την εκτέλεση τους. Λαμβάνοντας υπόψιν ότι η προτεινόμενη τεχνική μειώνει την παρεμβολή της αρχικοποίησης νέων εφαρμογών στην λειτουργία των ήδη εκτελούμενων, οδηγούμαστε σε μια κατάσταση που οι δεσμευμένοι πόροι απελευθερώνονται πιο γρήγορα. Αυτό με την σειρά του σημαίνει ότι οι εισερχόμενες εφαρμογές μπορούν να βρουν συντομότερα διαθέσιμους πόρους για την αρχικοποίηση τους και ως εκ τούτου ο συνολικός απαιτούμενος χρόνος αρχικοποίησης μειώνεται δραματικά.

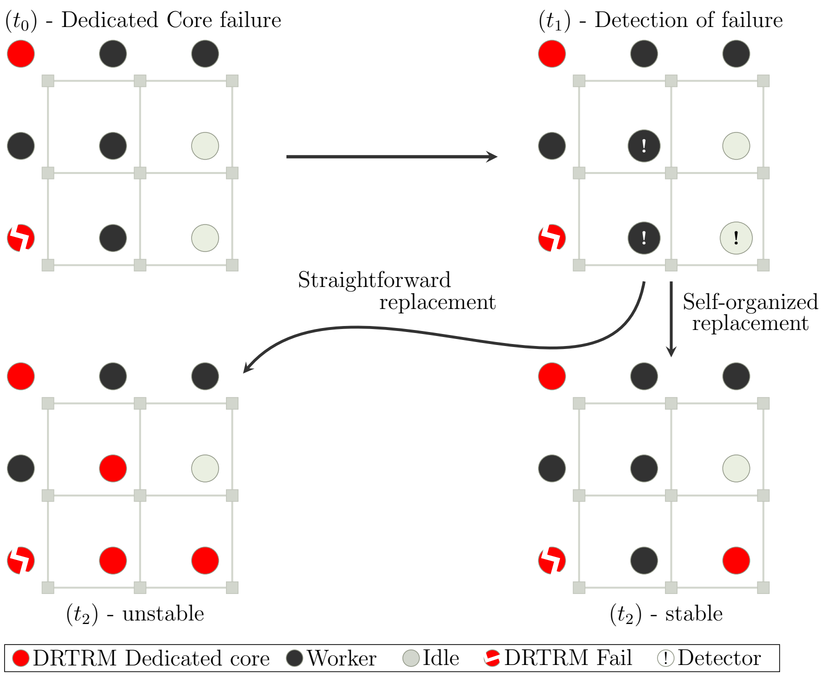
Η υψηλή κλιμάκωση των ψηφιακών κυκλωμάτων των σύγχρονων πολυπύρηνων συστημάτων, καθώς και η διαρκής και παρατεταμένη λειτουργία τους αυξάνει την πιθανότητα παρουσίασης σφαλμάτων στα επεξεργαστικά τους στοιχεία. Αυτό, σε συνδυασμό με την κρισιμότητα των εφαρμογών που εκτελούνται, οδήγησε στην επέκταση του κατανεμημένου διαχειριστή πόρων ώστε να χαρακτηρίζεται από ανοχή στα προαναφερθέντα σφάλματα. Η επέκταση αυτή βασίζεται στην δυναμική αντιμετώπιση των σφαλμάτων, λαμβάνοντας επίσης υπόψιν τον φόρτο εργασίας του κάθε πυρήνα κατά την διάρκεια της ανάνηψης από το σφάλμα.



Σχήμα 18: Κέρδη απόδοσης που προκύπτουν από την προτεινόμενη στρατηγική προσαρμογής του διαχειριστή πόρων με βάση τον ρυθμό εισόδου εφαρμογών στο σύστημα, στην περίπτωση των τεχνητών εύπλαστων εφαρμογών.

Πιο συγκεκριμένα εξετάζουμε ένα πολυπύρηνο σύστημα όπως φαίνεται στο Σχήμα 19 το οποίο διαχειρίζεται από ένα κατανεμημένο διαχειριστή πόρων όπως ο DRTRM. Οι επεξεργαστές με κόκκινο χρώμα υποδεικνύουν ένα σύνολο πυρήνων που η λειτουργία τους είναι αφιερωμένη στην σωστή λειτουργία του κατανεμημένου διαχειριστή πόρων. Οι υπόλοιποι επεξεργαστές επιτελούν άλλες εργασίες όπως εκτέλεση φόρτου εργασίας των εφαρμογών (μαύρο χρώμα) ή είναι σε ανενεργή κατάσταση (γκρι χρώμα). Το σύστημα βρίσκεται σε ευσταθή λειτουργία και όλοι οι επεξεργαστές και το λογισμικό είναι σε καλή κατάσταση.

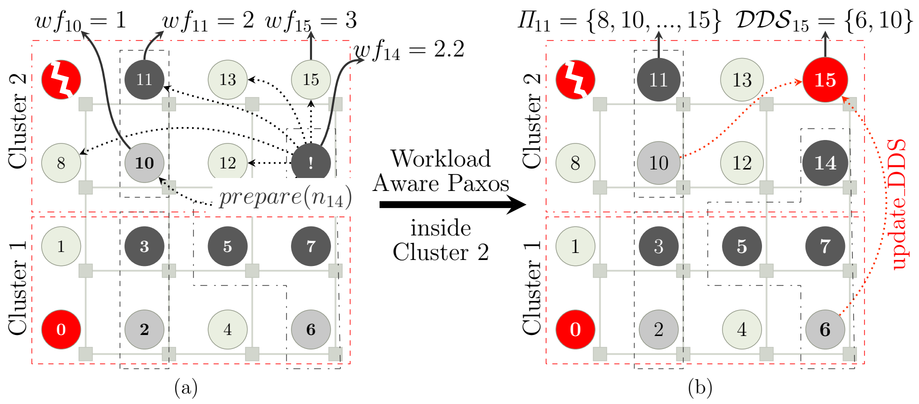
Σε ένα χρονικό σημείο t_0 ο πυρήνας του DRTRM που βρίσκεται στο κάτω αριστερά σημείο της πλατφόρμας παρουσιάζει ένα σφάλμα. Το σφάλμα είναι μη αναστρέψιμο και βασίζεται είτε σε σφάλμα του υλικού είτε σε σφάλμα του λογισμικού στο οποίο δεν μπορεί να γίνει ανάνηψη. Ο DRTRM δεν είναι σε θέση να λειτουργήσει ορθά χωρίς να αναπληρωθεί η λειτουργία του πυρήνα που παρουσίασε σφάλμα και ως εκ τούτου χρειάζεται να προσδιοριστεί ένας πυρήνας αντικαταστάτης. Με δεδομένο ότι το σύστημα είναι πλήρως κατανεμημένο, δεν υπάρχει ένας κεντρικό σημείο όπου θα προσδιοριστεί ο αντικαταστάτης και η απόφαση αυτή θα κοινοποιηθεί σε όλους τους άλλους υγιείς πυρήνες. Επομένως, μια διαδικασία αυτό-οργάνωσης μεταξύ των πυρήνων είναι απαραίτητη ώστε να προσδιοριστεί ο αντικαταστάτης.


(t_0) - Dedicated Core failure
(t_1) - Detection of failure
Straightforward replacement
Self-organized replacement
(t_2) - unstable
(t_2) - stable
DRTRM Dedicated core
Worker
Idle
DRTRM Fail
Detector

Σχήμα 19: Παράδειγμα σφάλματος και ανάνηψης σε ένα πολυπύρηνο υπολογιστικό σύστημα με κατανεμημένη διαχείριση πόρων.

Η διαδικασία αυτή πρέπει να οδηγεί σε ένα μοναδικό αποτέλεσμα και να εγγυάται ότι το αποτέλεσμα αυτό θα κοινοποιηθεί και θα είναι σεβαστό από όλους τους υγιείς πυρήνες όπως φαίνεται στην ευσταθή χρονική στιγμή t_2 στο κάτω δεξιά μέρος της εικόνας. Στον αντίποδα, μια πολύ απλή τακτική αντικατάστασης όπως πχ «Ο πρώτος πυρήνας που θα αντιληφθεί το σφάλμα, θα είναι αυτός ο αντικαταστάτης», είναι δεδομένο ότι θα αποτύχει λόγω της κατανεμημένης φύσης του συστήματος. Η απλή στρατηγική δεν μπορεί να εγγυηθεί ότι μόνο ένας πυρήνας θα αντιληφθεί το σφάλμα και θα προθυμοποιηθεί να γίνει ο αντικαταστάτης. Αυτό φαίνεται και από τους πυρήνες που αντιλήφθηκαν το σφάλμα σημειωμένους με '!' στην χρονική στιγμή t_1 του Σχήματος 19. Έτσι αν δύο ή περισσότεροι πυρήνες πάρουν την θέση αυτού που έπαψε να λειτουργεί, το σύστημα την χρονική στιγμή t_2 θα βρίσκεται πάλι σε μη ευσταθή κατάσταση, καθώς οι υγιείς πυρήνες δεν θα ξέρουν σε ποιον αντικαταστάτη να απευθυνθούν.

Η προτεινόμενη προσέγγιση για την επίλυση του συγκεκριμένου προβλήματος είναι ο σχεδιασμός ενός κατανεμημένου διαχειριστή πόρων ο οποίος είναι εμπλουτισμένος με χαρακτηριστικά αυτό-οργάνωσης ώστε να μπορεί να ανανήψει



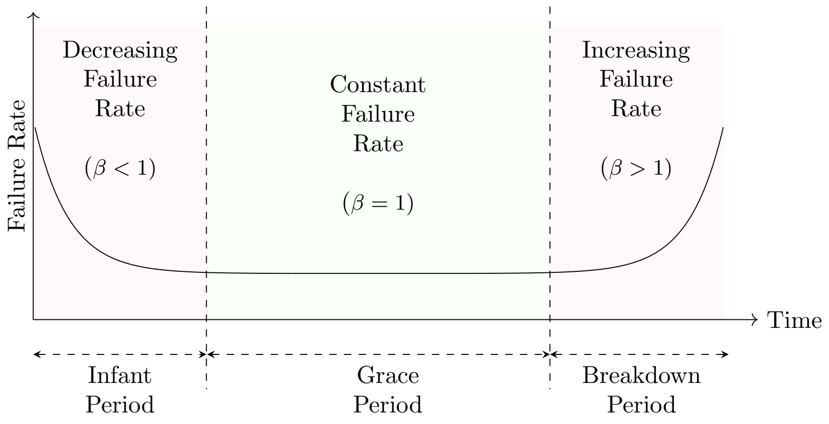
Σχήμα 20: Παράδειγμα λειτουργίας του SoftRM.

δυναμικά από σφάλματα στα επεξεργαστικά του στοιχεία. Ο διαχειριστής αυτός ονομάζεται SoftRM (Self-Organized Fault-Tolerant Resource Manager). Το βασικό δομικό στοιχείο του SoftRM είναι ο αλγόριθμος Paxos [138], ο οποίος επιτρέπει σε ένα σύνολο κατανεμημένων οντοτήτων να εκλέγει ομόφωνα μια μοναδική αριθμητική τιμή που γίνεται εν τέλει γνωστή σε όλους.

Ο αλγόριθμος Paxos χρησιμοποιείται ώστε οι υγιείς πυρήνες του συστήματος να εκλέξουν ποιος θα είναι ο πυρήνας αντικαταστάτης. Η βασική λειτουργία του αλγορίθμου επεκτάθηκε ώστε η επιλογή του αντικαταστάτη να γίνεται λαμβάνοντας υπόψιν και τον φόρτο εργασίας των πυρήνων. Για παράδειγμα, είναι πολύ πιο αποδοτική η επιλογή ενός ανενεργού πυρήνα ως αντικαταστάτη συγκριτικά με την επιλογή ενός πυρήνα εργάτη, καθώς η επιλογή του πρώτου έχει μηδενική επίδραση στην εκτέλεση των εφαρμογών ενώ η επιλογή του δεύτερου στερεί έναν εργάτη από μια εφαρμογή. Η επέκταση του αλγορίθμου Paxos, έγινε με την εισαγωγή του λεγόμενου παράγοντα προθυμίας (willingness factor), ο οποίος ποσοτικοποιεί την καταλληλότητα ενός πυρήνα να εκλεγεί ως αντικαταστάτης. Ο προσδιορισμός του παράγοντα προθυμίας γίνεται δυναμικά, καθώς ανά πυρήνα μπορεί να διαφέρει μεταξύ διαφορετικών χρονικών στιγμών.

Μια ακόμα πολύ σημαντική επιλογή αναφορικά με τον σχεδιασμό του SoftRM, είναι η απόχη από την χρήση εφεδρικών (spare) πυρήνων για την παροχή ανοχής σε σφάλματα. Η επιλογή αυτή έγινε με το σκεπτικό ότι η τεχνική αυτή αφενός μπορεί να ανταπεξέλθει σε περιορισμένο αριθμό σφαλμάτων (ίσα με τον αριθμό των εφεδρικών πυρήνων) και αφετέρου το σύστημα στερείται αποδοτικότητας εφόσον ένα ποσοστό των πυρήνων του παραμένουν ανενεργοί υπό την απουσία σφαλμάτων.

Στο Σχήμα 20 παρουσιάζεται ένα παράδειγμα της λειτουργίας του SoftRM.

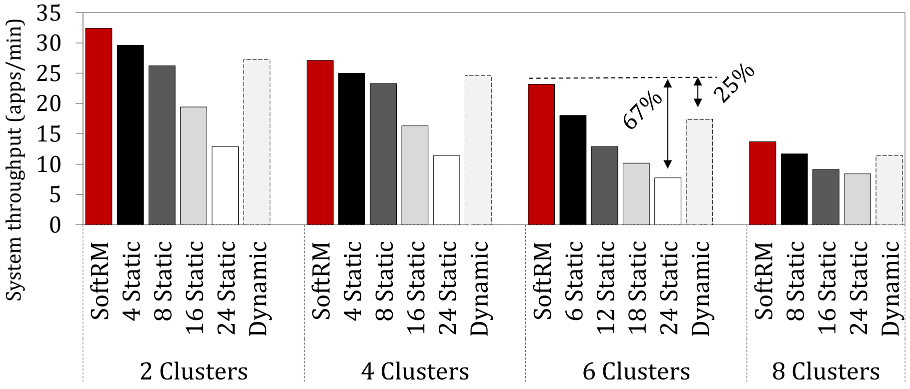


Σχήμα 21: Ρυθμός σφαλμάτων κατά τις διαφορετικές περιόδους ζωής της ψηφίδας.

Συγκεκριμένα, το Σχήμα 20a παρουσιάζει ένα χρονικό σημείο που ο Ελεγκτής πυρήνας 9 έχει σταματήσει να λειτουργεί. Ο πυρήνας 14 αντελήφθη την βλάβη του Ελεγκτή και ενεργοποίησε τον μηχανισμό ανάνηψης βασισμένο στον αλγόριθμο Paxos που λαμβάνει υπόψιν του τον φόρτο εργασίας των υγιών πυρήνων. Οι παράγοντες προθυμίας των διαφόρων επεξεργαστών έχουν σημειωθεί στο Σχήμα 20a (όλοι οι ανενεργοί πυρήνες έχουν ίσους παράγοντες προθυμίας).

Για να προσδιοριστεί η ταυτότητα του αντικαταστάτη Ελεγκτή πυρήνα, εκτελείται ένα στιγμιότυπου του ενισχυμένου αλγορίθμου Paxos μέσα στον χώρο που ονομάζεται Cluster 2. Μετά την εκτέλεση του αλγορίθμου, ένας ανενεργός πυρήνας έχει προσδιοριστεί ως αντικαταστάτης δεδομένου του ότι έχει τον υψηλότερο παράγοντα προθυμίας. Ο πυρήνας αυτός ενημερώνεται ότι είναι ο αντικαταστάτης και κατόπιν τούτου ενημερώνονται όλοι οι υγιείς πυρήνες της περιοχής. Κατόπιν πραγματοποιούνται όλες οι απαραίτητες ενέργειες ώστε οι διαχειριστικοί πυρήνες του SoftRM να έχουν ανανεωμένες πληροφορίες και το σύστημα συνεχίζει κανονικά την λειτουργία του, αποκλείοντας τον κατεστραμμένο πυρήνα (Σχήμα 20b).

Χωρίς βλάβη της γενικότητας, στην υπό παρουσίαση εργασία εξετάζουμε μόνιμα σφάλματα. Ως μόνιμα σφάλματα ορίζονται αυτά στα οποία η βλάβη που παρουσιάζεται δεν μπορεί να επισκευαστεί κατά την διάρκεια του χρόνο εκτέλεσης του συστήματος. Πιο συγκεκριμένα, εξετάζουμε σφάλματα στα επεξεργαστικά στοιχεία της πλατφόρμας και δεν εξετάζουμε σφάλματα στις συνδέσεις του δικτύου που χρησιμοποιείται για την επικοινωνία τους. Με άλλα λόγια, θεωρούμε ότι μεταξύ δύο υγειών επεξεργαστικών στοιχείων υπάρχει πάντα ένα λειτουργικό κανάλι επικοινωνίας. Η πιθανότητα εμφάνισης σφάλματος σε ένα επεξεργαστικό στοιχείο μοντελοποιείται με χρήση της κατανομής πιθανοτήτων Weibull [131] και



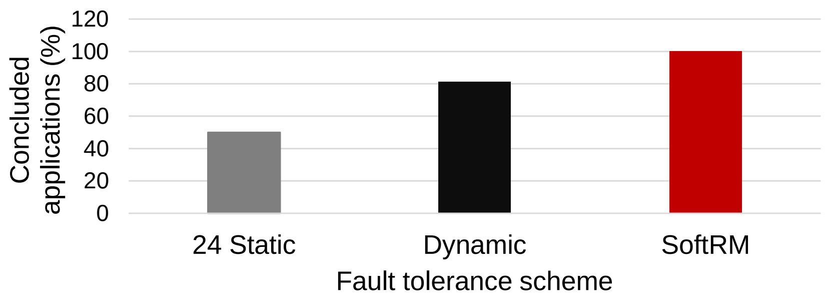
Σχήμα 22: Συγκριτική αξιολόγηση του SoftRM και των διαχειριστών που κάνουν χρήση εφεδρικών πυρήνων (Ελλείψει σφαλμάτων κατά τον χρόνο εκτέλεσης).

σχετίζεται με μια μέση τιμή αναμενόμενου χρόνου παρουσίασης σφάλματος (Mean Time To Failure (MTTF)).

Με στόχο την περαιτέρω βελτίωση του χρησιμοποιούμενου μοντέλου σφαλμάτων, απέχουμε από την χρήση σταθερού ρυθμού σφαλμάτων αλλά προσαρμόζουμε τις παραμέτρους της συνάρτησης πυκνότητας πιθανότητας σύμφωνα με την προσομοιούμενο κύκλο ζωής της πλατφόρμας. Πιο συγκεκριμένα, μια ψηφίδα παρουσιάζει διαφορετικούς ρυθμούς σφαλμάτων σε διαφορετικές φάσεις του χρόνου ζωής της. Στα πλαίσια αυτής της δουλειάς, οι εν λόγω ρυθμοί σφάλματος διαφοροποιούνται στον χρόνο όπως φαίνεται στο Σχήμα 21 [236]. Κατά την εμβρυική της φάση (infant period), η πλατφόρμα έχει υψηλό αλλά φθίνοντα ρυθμό σφαλμάτων, ο οποίος με το πέρας του χρόνου φτάνει σε έναν ελαχιστοποιημένο, σταθερό ρυθμό σφαλμάτων κατά της διάρκεια της λεγόμενης περιόδου χάριτος (grace period) της πλατφόρμας. Μετά από μακριά διάρκεια λειτουργίας, τα αθροιστικά φαινόμενα γήρανσης και εκφυλισμού του πυριτίου επιδεινώνονται οδηγώντας εν τέλει στην περίοδο κατάρρευσης της πλατφόρμας (breakdown period), όπου παρατηρείται διαρκώς αυξανόμενος ρυθμός σφαλμάτων.

Ο προτεινόμενος διαχειριστής πόρων SoftRM, σχεδιάστηκε και αναπτύχθηκε με στόχο την πλατφόρμα Intel SCC. Ένα σενάριο αξιολόγησης του SoftRM, περιλαμβάνει ένα σύνολο εισερχόμενων εφαρμογών στο σύστημα και ένα σύνολο δυναμικά εκδηλωμένων σφαλμάτων, όπως προκύπτουν από το προαναφερθέν μοντέλο. Στα πειράματα που πραγματοποιήθηκαν έγινε χρήση της παράλληλης εφαρμογής πολλαπλασιασμού πίνακα με διάνυσμα.

Η σύγκριση του SoftRM γίνεται ενάντια σε άλλους σύγχρονους διαχειριστές πόρων που παρουσιάζουν ανοχή σε σφάλματα. Οι εν λόγω διαχειριστές πόρων



Σχήμα 23: Συγκριτική αξιολόγηση του SoftRM και των διαχειριστών που κάνουν χρήση εφεδρικών πυρήνων (Με σφάλματα κατά τον χρόνο εκτέλεσης).

κάνουν χρήση εφεδρικών πυρήνων για να παρέχουν δυναμική ανάνηψη από σφάλματα. Οι εφεδρικοί πυρήνες μπορεί να έχουν προσδιοριστεί είτε στατικά κατά την διάρκεια του σχεδιασμού του συστήματος (static spare core provision) [59], είτε δυναμικά αφιερώνοντας ένας από τους πυρήνες της κάθε εφαρμογής ως εφεδρικό για την περίπτωση που προκύψει κάποιο σφάλμα (dynamic spare core provision) [127].

Στο πείραμα που παρουσιάζεται στο Σχήμα 22, αξιολογείται η μειωμένη επιβάρυνση του SoftRM στην εκτέλεση των εφαρμογών, υπό της απουσία σφαλμάτων. Το πείραμα αφορά ένα σενάριο 64 εισερχόμενων εφαρμογών και ποσοτικοποιεί τον ρυθμό ολοκλήρωσης εφαρμογών ανά λεπτό σε σύγκριση με τους διαχειριστές που κάνουν χρήση εφεδρικών πυρήνων. Τα αποτελέσματα είναι οργανωμένα ανάλογα με τον αριθμό των Ελεγκτών πυρήνων στο σύστημα. Παρατηρούμε στο Σχήμα 22, ότι η προσέγγιση μας οδηγεί σε όλες τις περιπτώσεις σε μεγαλύτερο ρυθμό ολοκλήρωσης εφαρμογών που οφείλεται στο γεγονός ότι όλοι οι πυρήνες του συστήματος χρησιμοποιούνται πλήρως και κανένας δεν είναι σε εφεδρεία. Το ποσοστό κέρδους φτάνει έως 25% ακόμα και συγκριτικά με την περίπτωση που οι εφεδρικοί πυρήνες προσδιορίζονται δυναμικά.

Το πείραμα με τις 64 εισερχόμενες εφαρμογές επαναλαμβάνεται υπό την παρουσία σφαλμάτων. Συγκεκριμένα, 18 σφάλματα εκδηλώνονται δυναμικά κατά τον χρόνο εκτέλεσης σε χρονικές στιγμές που προκύπτουν από το μοντέλο σφαλμάτων του συστήματος. Με δεδομένη την ύπαρξη σφαλμάτων, το πείραμα που παρουσιάζεται στο Σχήμα 23 ερευνά ποια από τις συγκρινόμενες τεχνικές είναι πιο αποδοτική στην ολοκλήρωση των εφαρμογών, στο ίδιο χρονικό πλαίσιο. Το πείραμα περιλαμβάνει μόνο την τεχνική με 24 στατικά εφεδρικούς πυρήνες καθώς οι υπόλοιπες του ίδιου είδους αδυνατούν να ξεπεράσουν και τα 18 σφάλματα που προκύπτουν. Όλες οι τεχνικές που συγκρίνονται είναι ικανές να ξεπεράσουν όλα τα σφάλματα με την διαφορά ότι ο SoftRM, που κάνει βέλτιστη χρήση των πόρων του συστήματος, το πραγματοποιεί πολύ συντομότερα. Το ποσοστό ταχύτερης ολοκλήρωσης αγγίζει το 50% για την χρήση στατικών και 20% για την χρήση δυναμικών εφεδρικών πυρήνων.

Συμπερασματικά, το κεντρικό κομμάτι της εν λόγω διδακτορικής διατριβής ασχολήθηκε με την σχεδίαση και υλοποίηση ενός διαχειριστή πόρων, όπου οι αποφάσεις για την κατανομή των πόρων λαμβάνονται δυναμικά και με κατανεμημένο τρόπο κατά την διάρκεια του χρόνο εκτέλεσης του συστήματος. Η δουλειά που πραγματοποιήθηκε στα πλαίσια της διατριβής απέδειξε ότι είναι εφικτή η σχεδίαση ενός τέτοιου αλγορίθμου, ο οποίος δύναται να διαχειριστεί αποδοτικά μεγάλό αριθμό εφαρμογών με ποικίλες απαιτήσεις σε φόρτο εργασίας. Η περαιτέρω ανάλυση του διαχειριστή πόρων, ανέδειξε ενδιαφέρουσες πτυχές της κατανεμημένης φύσης του, οι οποίες καθιστούν πιο περίπλοκη την δυναμική εφαρμογή πολιτικών καθώς και την ανάνηψη από σφάλματα. Μολαταύτα, αποδείχθηκε ότι με σωστό σχεδιασμό είναι εφικτό να επεκταθεί ο αρχικός σχεδιασμός του διαχειριστή ώστε με αποδοτικό τρόπο να είναι δυναμικά προσαρμοστικός σε εξωτερικούς παράγοντες όπως ο ρυθμός έλευσης εφαρμογών στο σύστημα και ανεκτικός σε σφάλματα.

Τέλος, οι βασικές ιδέες της κατανεμημένης διαχείρισης πόρων επεκτάθηκαν ώστε να υποστηρίξουν την δυναμική διαπραγμάτευση πόρων σε συστήματα υπολογισμού στην άκρη του δικτύου (Edge computing) με πολλούς ενδιάμεσους κόμβους πύλες. Αυτοί οι κατανεμημένοι κόμβοι, κάνουν χρήση μηχανισμών βασισμένων στις ιδέες του εμπορίου ώστε να βελτιστοποιήσουν την παρεχόμενη ποιότητα υπηρεσίας στους αντίστοιχους IoT κόμβους συνδρομητές στις υπηρεσίες της πύλης, τηρώντας ταυτόχρονα τους λειτουργικούς περιορισμούς αυτών των IoT συσκευών. Οι μηχανισμοί αυτοί οδηγούν στην πιο αποτελεσματική σύνδεση των IoT συσκευών στις πύλες, επιτρέποντας έτσι την πλήρη χρήση των πόρων των δεύτερων για την εξυπηρέτηση της λειτουργίας των πρώτων.

Λέξεις κλειδιά: Κατανεμημένη Δυναμική Διαχείριση Πόρων, Υπολογισμός στην άκρη του Δικτύου, πύλες IoT, εφαρμογές IoT

Acknowledgements

The following dissertation as well as my entire research work would have not been realized without the constant and uninterrupted help and support by a handful of people. Words are not enough to express the depth of my acknowledgements, a fact that makes this section one of the most challenging of the dissertation. Therefore, by adhering to the concept of simplicity, I will summarize my acknowledgements in few words, since increasing their number will not solve the riddle of how to fully express my gratitude.

To begin with, I have to thank all the people of the Microprocessors and Digital Systems Lab (MicroLab) of N.T.U.A., where we spent more than five years of daily cooperation, team effort and good company. Begging from the younger members, I have to thank all students which collaborated with me in their theses, during which I learned with them and produced results, which without them would have never been realized. With respect to core Microlab I have to thank Alexis Bartzas and Iraklis Anagnostopoulos, which were my first immediate collaborators and fuelled my interest for pursuing a PhD. This PhD would not have come into fruition without the help and guidance of Sotiris Xydis, which repeatedly guided my through the my research quests and provided me with insights and lessons, which were indispensable for the completion of my thesis. Last, my highest acknowledgements belong to my advisor Dimitrios Soudris, who trusted me with the opportunity to pursue my PhD degree and facilitated all the aspects of my research. His guidance, both moral and technical, was the key turning point for overcoming the pitfalls and difficulties in my research effort and will accompany me in my steps for the rest of my life.

Outside of Microlab, I was fortunate to collaborate with many excellent researchers and people, which helped my to extend my technical, scientific and team working aspects. Specifically, I would like to mention Gabriela Dudnik and Marc Correvon from CSEM in Switzerland and Giuseppe Gallo from Italy, which formed a team within the Swan-iCare project that eventually led to the highlight of my career where a working prototype, fully built within the project was evaluated on patients. In addition, I have to thank Farzad Samie of Karlsruhe Institute of Technology for many years of excellent collaboration, which resulted in numerous publications and of course Lars Bauer and Professor Jörg Henkel, who aided and facilitated this collaboration and proved to me that universities still can effectively work together and produce magnificent results, even with no previous common activities.

Last but not least, I would not be able to go through these years without the support of my friends and family. I am very fortunate in having a lot of good friends both from my home town and from my university years, which I cannot thank enough for all these years that we have spent together and the rest that will come. As far as my family is concerned, we are small in number but very big in supporting each other so my special thanks to my mother Sofia, my father Spiros, my sister Nefeli, my aunt Panagiota, my cousin Konstantinos and my uncles Giannis and Charalampos.

Contents

List of Figures

List of Tables

Chapter 1

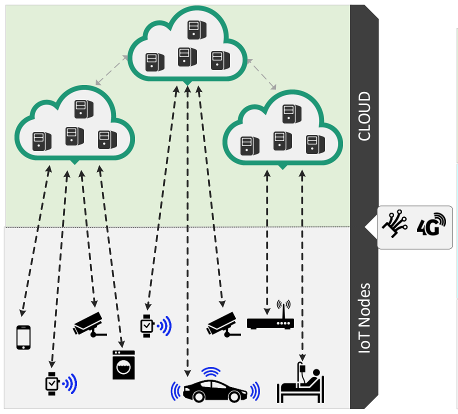
Introduction

1.1 Internet of Things

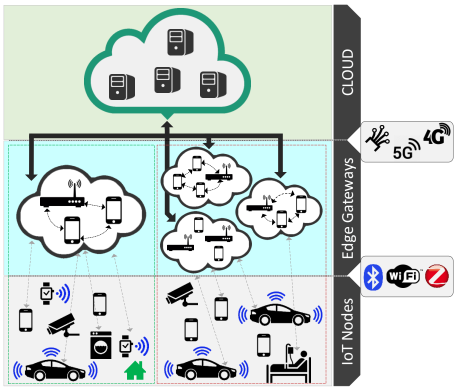
The Internet of Things (IoT) infrastructure has recently gained tremendous attention by the industrial and academic community as an architectural milestone, which will revolutionize the way that our world is sensed, perceived and acted upon [101]. Tiny, embedded, ubiquitous devices are envisioned to flood all aspects of human life, seamlessly integrated into their surroundings, acting as the sensing and computational front-end of an architecture that is backed-up by Cloud infrastructure to provide heavy processing and abundant storage. This inter-connection is a major leap of computing systems as it will create a digital veil over the real world which will enhance our understanding of all aspects of our life and allow us to optimize and simplify daily routines.

While revolutionary, the concept of IoT is simple and based on a principle of observe-analyze-act in order to conduct its intended goals. More precisely, the IoT nodes are responsible for data collection in the vicinity of the user, including some or no local processing and then data uploading to the Cloud for correlation and fusion of data from numerous IoT nodes. Eventually, the results of this Cloud based processing are transmitted back to the IoT nodes in order to reach the end user (Fig. 1.1a).

The deployment and testing of the original Cloud-centric IoT architecture showed that in certain applications it fails to provide an efficient system, either in terms of meeting the run-time requirements or deploying a cost-effective IoT solution. The unprecedented high number of IoT nodes, expected to surpass 20 billion nodes by 2020 [1], creates a set of communication bottlenecks, which stress out



(a) Original IoT architecture.



(b) Updated IoT architecture.

Figure 1.1: State-of-the-art IoT architecture.

the abilities of the Cloud-centric system [253, 145, 201, 57]. Applications in this setup are dependent on their connection to the Cloud, a feature that is a point of failure in case of connection loss or increased communication latency. In addition, the immense bandwidth requirements as a result of the high number of connected devices stress out the abilities of the wireless communication channel. For example, a city populated with 1 million people is expected to produce 180 PetaBytes per day by 2019 [166], produced by activities in transportation, utility, safety and healthcare.

A portion of the aforementioned issues, can be addressed by constant over-provisioning of resources in the Cloud but this design decision leads to a high increase in the cost of the applications, since Cloud resources are not abundant and most importantly they are priced according to utilization. Even if increased utilization is affordable, the overall architecture is not energy efficient, since it makes uses of a high-end, power hungry communication infrastructure in order to perform local, specific tasks which can be effectively executed by customized embedded systems in a highly energy efficient way.

Consequently, a new architecture for the deployment of IoT applications has been proposed, which includes intermediate processing and storage nodes between the IoT devices and the Cloud. This updated computing paradigm has been established by the name of Edge computing [201], as it directs the computation of data in the edge of the Cloud-centric network and the introduced nodes are referred to as Smart Edge Gateways (Fig. 1.1b). Tasks are conjointly executed by IoT nodes and Gateways, by *offloading* a portion of the required processing from the first to latter, which are richer in computational resources [45, 48].

Data are also stored locally and only a portion of them is uploaded to the Cloud. Local data transmission is performed using short range communication protocols, thus increasing the reliability of the system. In this way, this architecture allows the distribution of data storage and processing, leading to the utilization of the Cloud infrastructure for processing and long-term storage of only selected data. According to a report by International Data Corporation (IDC) Futurescape, around 40% of IoT-generated data will be processed, stored, and acted upon close to the edge of network [184].

The consumption of data close to their production is the essence of Edge computing. This principle is respected by other relevant IoT architectures, which have been proposed in literature, such as Fog Computing [44, 57, 117], Cloudlets [192] and Micro Data Centers (MDC) [40]. The difference of these concepts, is the properties of the infrastructural nodes that will carry out the processing offloaded by IoT nodes, while in many cases the concepts are used interchangeably by researchers [201, 191, 40].

1.2 Contemporary Embedded applications

The evolution of embedded systems, combined with the respective breakthroughs in sensing devices has opened up new possibilities with respect to current and future applications. The key power of IoT applications is the novel capability of monitoring processes and phenomena that only a few years ago it was impossible to follow closely. These data can be processed and cross-correlated in order to produce knowledge of the mechanisms within the monitored process and eventually lead to decisions on how to optimize them. This data centric approach impacts our view on the world and is combined with a clear potential for profits both for enterprises and individual users. Fig. 1.2, derived from a McKinsey institute report on IoT [2], summarizes the potential application domains as well as the expected profits, reaching up to trillion of dollars. Closer inspection of the figure also validates the diversity of potential applications, starting from individuals and scaling up to homes, enterprises and cities.

As a general rule, the concept of IoT refers mainly to Things as monitoring devices and source of information for further processing. The embedded systems with the ability of acting upon the physical world are referred to using the term Cyber-Physical systems [237]. While the distinction line between these two categories is not always clear, it is a fact that the infiltration of the physical world by the virtual one, leads to an immense set of non-functional requirements for embedded devices, that cannot be violated at run-time.

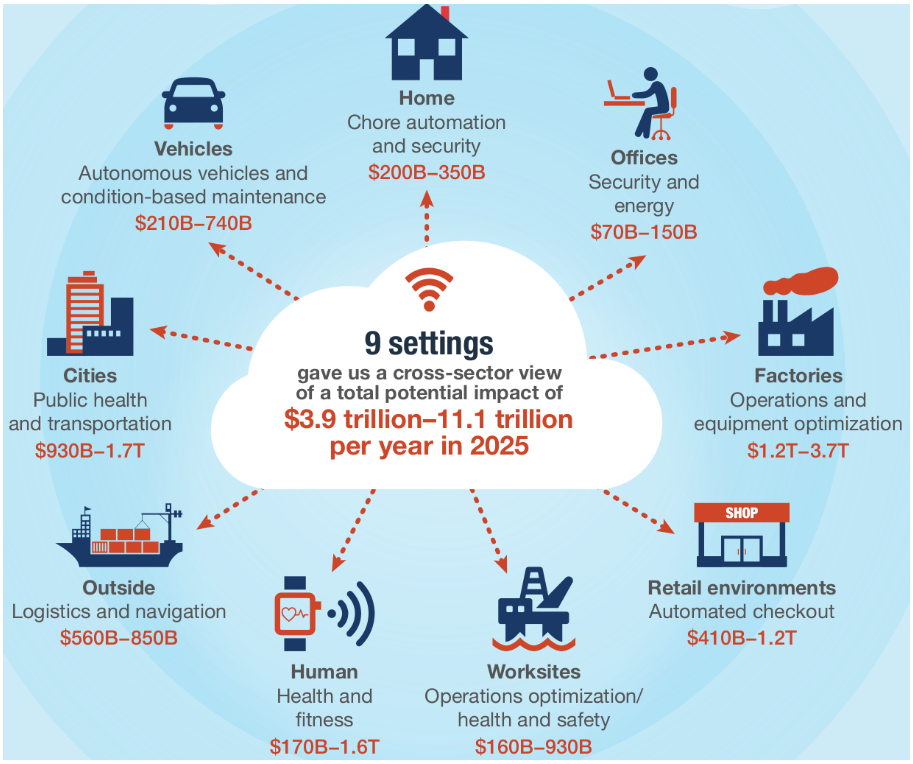

Home
Chore automation and security
$200B–350B
Vehicles
Autonomous vehicles and condition-based maintenance
$210B–740B
Offices
Security and energy
$70B–150B
Cities
Public health and transportation
$930B–1.7T
9 settings
gave us a cross-sector view of a total potential impact of
$3.9 trillion–11.1 trillion per year in 2025
Factories
Operations and equipment optimization
$1.2T–3.7T
SHOP
Outside
Logistics and navigation
$560B–850B
Retail environments
Automated checkout
$410B–1.2T
Human
Health and fitness
$170B–1.6T
Worksites
Operations optimization/ health and safety
$160B–930B

Figure 1.2: Application domains for Internet of Things [2].

Since their conception, embedded systems have been characterised by both hard and soft real-time operation requirements. The notion of real-time requirements has nowadays shifted from precise control of machinery to multimedia applications such as ultra high resolution video reproduction and virtual reality, executed on portable devices with battery supply. For example, a Virtual Reality application which makes us of head-tracked systems, requires execution latency 16 ms or less, in order to achieve stability of perception [191].

Despite the overwhelming computational complexity of such multimedia tasks, there is a reasonable margin for errors and deadline misses. On the contrary, this is not acceptable in the context of a self-navigating vehicle, where a possible error could result in injury or death of the users. This feature quantifies the car as a hard real-time system, which is expected to require the processing of about 1 Gigabyte of data per second in order to conduct its functionality [201]. In general, the danger for humans, imposed by Cyber-physical applications highlights the need for safe, secure and dependable design of embedded systems.

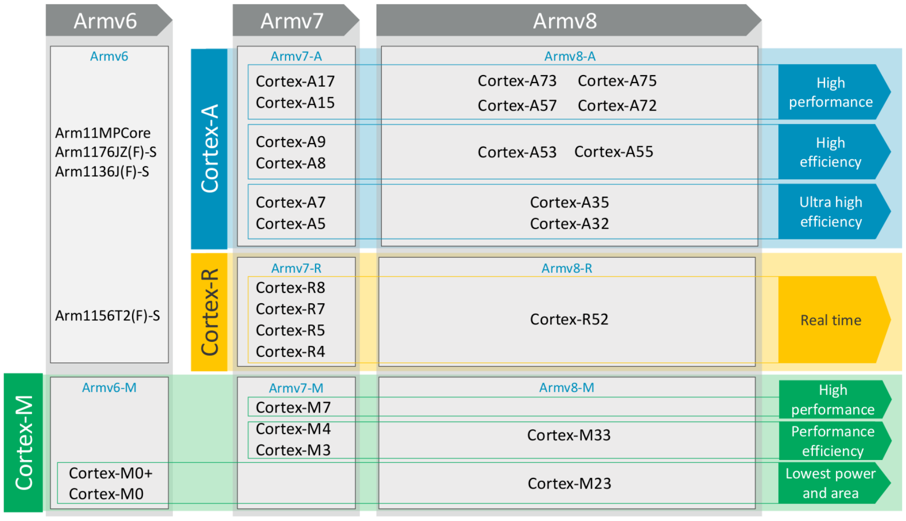
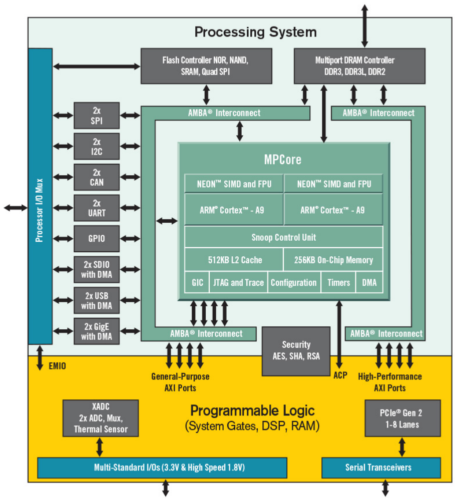


Figure 1.3: Spectrum of ARM Cortex processors [4].

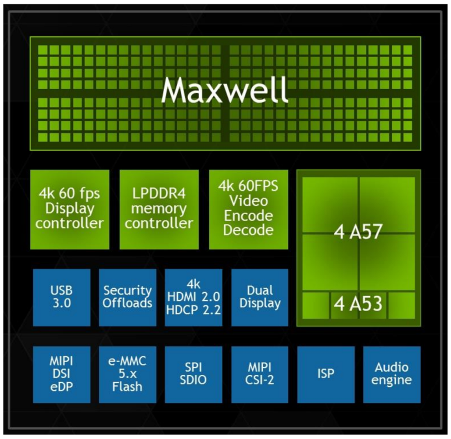
The explosion of data volume and velocity is also tightly coupled to the big steps in the Artificial Intelligence domain. This domain is expected to provide the algorithmic tools to interpret and correlate the vast amount of sensed data. The inherent tradeoff is that state-of-the-art machine learning models, such as Deep Neural Networks [194] are characterised by increased computational and storage requirements. Despite this fact, their adoption is constantly increasing, thus shifting the focus of industrial and academic research on holistic approaches aiming at their seamless integration in small-factor embedded systems [173].

1.3 Embedded hardware architectures

The essence of contemporary embedded systems has shifted from single- to multi-purpose systems, which communicate and act upon their surroundings. While the complexity of the design has been increased, the requirements for real-time responses, low-energy consumption and dependability are as prevalent as ever. In parallel to these increased requirements, advanced fabrication technology has enabled the hardware of embedded systems to evolve and provide abundant choices for designers. While only a few years ago embedded systems were built on top of simple micro-controllers, nowadays the heart of an embedded system often integrates architecturally rich Central Processing Units with high operating frequency. A distinctive example of this evolution, is the current



(a) Xilinx Zynq-7000 SoC Architecture [15].



(b) Nvidia TX1 Architecture [14].

Figure 1.4: Different available Hardware architectures.

offered spectrum of CPUs by ARM Ltd., a leading player in the design of CPUs for embedded systems, as illustrated in Fig. 1.3. We observe, a multi-layer spectrum of choices where the designer can identify low-power, low-end units such as the Cortex-M series, Real-time oriented CPUs such as the Cortex-R series as well as high-end, multi-core processors of the Cortex A series. This variety of choices enables the fine-tuning of the hardware for the intended application, but complicates the process of reaching this decision due to the higher number of design alternatives.

Despite their broad range, in many cases CPUs fail to meet the design requirements with respect to execution latency, energy consumption and cost. This led to the convergence of embedded systems with High Performance Computing, thus producing Systems on Chip (SoC) designs, which incorporate both General Purpose Computing Elements and other modules for acceleration, e.g. Graphic Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs). This integration allows for efficient data exchange between the accelerating and main unit of the SoC, which is translated to highly reduced execution latency and diminished energy consumption. Both design choices (GPUs and FPGAs) are characterized by advantages and disadvantages, thus complicating the decision as to which is the most efficient one per application. In general, GPUs have been established as a key feature of modern embedded systems such as mobile phones, tables and gaming machines but FPGAs are

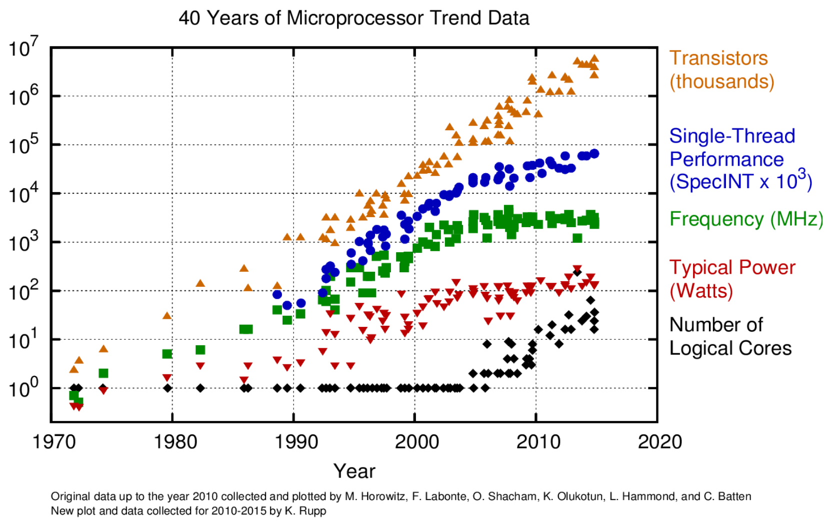


Figure 1.5: Trends of Microprocessors design.

gaining ground due to their re-configuration abilities, higher performance per Watt and thus have been included in mobile devices, e.g. Apple iPhone 7 [3].

Specifically for building an IoT node, the design space is also inflated due to the high number of available wireless communication interfaces. These correspond both to short and wide area network connections, providing different choices regarding the available bandwidth, data exchange rate, range of connection and energy consumption. Distinctive examples of such technologies for short range communication are Near Field Communication (NFC), ZigBee, Bluetooth Low Energy, Classic Bluetooth, Conventional and Low-power WiFi, etc., while for wide area network communication are Cellular networks (3G, 4G) and Low Power Wide Area Network (e.g. LoRaWAN, SigFox) [187].

In many cases, an embedded system may be equipped with more than one communication interfaces, in order to be able to support various combinations of short and wide are network communication. This mandates for more complex resource management techniques in order to effectively utilize the interfaces and reduce the energy consumption of the system. Besides the numerous choices regarding communication, the emerging adoption of technologies such as Software Defined Networking (SDN) [134] and Network Function Virtualization (NFV) [160] are expected to offer efficient and flexible solutions for the ever increasing number of inter-connected IoT nodes. Such technologies are predicted to be incorporated into the run-time functionality of Edge Gateways [191] and

Fog computing nodes [117].

Moreover, modern computing and embedded architectures are additionally characterized by the constantly accelerating increase in the number of integrated processing elements (PEs) integrated in a single chip [205], as also shown in the trend presented in Fig. 1.5. Large industrial companies endorse this vision and have started to experiment and commercialize many-core computing systems. Intel has already created platforms with 80, 48 and 50 processing cores [114, 195, 227, 122], while Tilera currently features up to 100 cores per chip [94]. The envisioned paradigm is thousand core chips [89], which is already approached by academia [42] and industry [170, 23].

The increased number of PEs on a single chip, creates new requirements regarding their inter-connection, since cache coherency scales well for a small amount of cores but becomes the bottleneck in high core-count processors [126, 208]. Networks on Chip have been established as the proposed communication architecture for overcoming this bottleneck [41]. The abundance of integrated computing cores, can be the key for unlocking the potential of Edge computing, by providing the necessary infrastructure for building powerful and flexible Gateways, able to support numerous IoT nodes concurrently.

1.4 Open Challenges of Gateway based IoT

Edge computing systems, correspond to an architecture of multiple levels of computing nodes aiming at efficiently utilizing the abundance of different hardware options presented in Section 1.3, in order to meet the strict application requirements summarized in Section 1.2. This multi-tier architecture also increases the dependability of the system by executing the critical functionality of IoT applications locally, thus minimizing their dependency on Cloud resources.

These features have led the adoption of the Edge computing concept by industry, leading to collaborative actions for the clarification of all aspects of the architecture. Such distinctive actions are the Open Edge Computing Initiative [1] including Intel and Deutche Telekom Group and the Open Fog Consortium [2] with funding members ARM, Cisco, DELL, Intel and Microsoft. The European Union is also supporting the development and deployment of Edge-Fog technologies via EU funded projects, e.g. FAR-EDGE [6], mF2C [9], FORA [7], PrEstoCloud [10] and DITAS [5].

[1]openEDGEcomputing [Online] Available: http://openedgecomputing.org/index.html
[2]OpenFogConsortium [Online] Available: https://www.openfogconsortium.org/

While distributed computing close to the edge of the network is a promising solution for providing the building blocks for the next generation of IoT applications, there are still a number of open challenges to be addressed in order for it to reach the maturity to be consolidated as the main design option [201, 117]. These challenges are related both to the design of individual Edge computing nodes and to the inter-play and run-time decision making of multiple nodes.

1.4.1 Single IoT node Application Design

The Cloud-centric view of IoT applications, provides benefits for application design due to the fact that the developer can operate under the abstraction of a complete view of all sensed data. Under this assumption, there are very efficient programming models e.g. Hadoop MapReduce [79] or Apache Spark [252], which provide a solid basis to design and implement an efficient application. Application design in the context of a multi-layer Edge computing infrastructure results in a number of challenges, since the aforementioned assumption about data aggregation in not always valid.

Edge computing is distributed by nature, which allows for the distribution of storage and processing but can impose overheads when a complete view of data is necessary. In addition, it requires that applications are adapted to this new multi-layer infrastructure, without a well defined standard for their development. The mapping of the application in the different computation layers is characterised by trade-offs given the communication latencies and synchronization issues. Most importantly, Gateway based applications must have the required structure to benefit from the offloading capabilities, as well as to adapt to the dynamics of this setup. More specifically, while the Cloud offloading paradigm ensured that resources are always available, in the case of local offloading, resources are shared between many IoT nodes, which dynamically relocate their position.

With respect to the internals of an Edge oriented IoT application, *the open challenge is to design the application in a modular manner, able to support partial or full offloading of its computations on the Edge Gateway, in an effort to minimize its execution latency or energy consumption [189].* From another point of view, this modular structure can be perceived as different operating modes of the application, which should be defined in such a way that the dynamic re-configuration of the application for switching between modes is both feasible and efficient.

Regardless of the adoption of the Edge computing architecture, application domains such as medical or Cyber-physical systems, require that the application is able to conduct critical functionality, without connection to the Cloud. This feature should be incorporated in the design cycle of the node and be extensively validated. The standalone function of an IoT node, i.e. operating with no offloading capabilities, can be aided by the integration of accelerating units which will boost its computational strength in case of real-time requirements. However, choosing to accelerate parts of the application, mandates for a comprehensive approach in order to designate and implement the parts to be executed on the accelerating unit, while respecting the limitations in the duration of design

1.4.2 Single Edge Gateway Design

An Edge (or Fog) Gateway, is responsible to collaborate with a number of IoT nodes, which dynamically subscribe (are bound) and share its service. To achieve and effective collaboration, the open challenges correspond to *(i) the run-time decision making of how to organise the sharing of resource between the IoT nodes and the Gateway and (ii) how to manage the resources of the Gateway in order to execute the required tasks offloaded by the IoT nodes.* The first challenge has been investigated [189], so the focus of the presented thesis is shifted to the second challenge, assuming that the Gateway is a many-core embedded system. We consider many-core SoCs as an emerging design alternative able to provide the necessary computing power to support both the computational and communicational requirements of an Edge Gateway.

In general, the increase in the number of computing resources on a single chip, has not yet yielded the expected performance gains. One main reason lies in the scalability of traditional run-time resource management schemes [47, 126]. Keeping caches coherent while managing a large number of cores requires customized approaches, especially in shared memory systems where the core-to-core communication is structured upon the underlying cache-coherency protocol [183, 136]. Moreover, scaling problems result in performance issues as traditional resource managers fail to exploit the underlying hardware and consequently do not provide the desired high performance expected from large pools of resources [80].

The run-time resource management paradigm is a key challenge for modern multi-core systems [18, 20], which is prominent due to the run-time dynamicity of modern parallel applications and platforms [130, 21]. The ability to allocate the right number of cores at run-time, to allow applications to handle their resources as well as to offer expand/shrink characteristics are the critical features for efficient system utilization. These principles have also affected the design of many-core oriented operating systems such as Barrelfish [34], Popcorn [30], Akaros [185] and FOS [235].

The combination of highly dynamic, parallel applications and emerging technologies in many-core systems dictates the need sophisticated logic in resource distribution, in an effort to meet the requirements of high performance, low power consumption, safety and reliability. Inevitably, this sophistication comes at the price of high computational requirements in order to provide results within acceptable time limits, while operating in systems where workload execution requests are issued dynamically by numerous users, e.g. offloaded tasks by IoT nodes. Last but not least, the synergistic execution of tasks in Gateway based systems mandates for the dependable operation of the resource allocation infrastructure, given that a possible failure will affect multiple nodes.

1.4.3 Multi-Gateway Edge System Design

The complete system of multiple IoT nodes and Gateways, is inherently a highly dynamic setup. Each IoT node is connected to an Edge Gateway, and its relocation might required its re-connection to a new Gateway, located in the vicinity of its new position. This disrupts an achieved equilibrium of the system and can potentially lead to a less efficient run-time operation point, with respect to resource usage and offered Service Quality. In addition, in many cases an IoT node will have more than one available Gateways for connection, thus requiring a consistent way of dynamically choosing the one to connect to.

This choice, affects the performance of the overall Edge system, since the resources of the Gateways are shared between many IoT nodes. The Gateways aid the operation of the IoT nodes and as a consequence the over-burdening of one Gateway can be a point of inefficiency for system. Since the setup is distributed by nature, without any central supervising entity, *there is the need for a distributed run-time resource management mechanism, which will take into account the dynamics of the system and promote its re-configuration with respect to maximization of the offered Service Quality from the point of view of the final IoT user.*

1.5 Contributions and Text Structure

The previous Sections have outlined the current status of embedded systems, including application requirements, the immense space of design alternatives as well as the challenges and open issues towards bridging this gap. The analysis has revolved around Edge computing, an evolving IoT architecture, however the challenges are not limited to this architecture but are the continuation of previous architectures and will pave the road for future ones. This thesis aims at laying the foundations for providing answers to some of the aforementioned open issues. The presented research extends two fundamental concepts, i.e. the design of an embedded application and the definition of the run-time management policies of an embedded system, regarding its resources and its collaboration with other nodes.

The overview of the contributions, follows a bottom up approach with respect to the complexity and computational power of the examined IoT nodes. At first the design issue of the single embedded system is tackled, followed by an in-depth investigation of the run-time management of a single many-core Edge Gateway and eventually reaching the problem of the co-ordination of many Edge Gateways. More specifically, the main contributions with respect to the design of an embedded node are as follows:

- The complete design cycle of a complex embedded medical device for Smart Wound Management is presented, characterized by strict requirements for validated dependable operation. An FPGA based HW/SW emulation unit is utilized, in order to avoid the streamlining of the design of the SW and HW parts of the final system. This emulation platform provided a solid starting point for the first prototype of the embedded SW, which eventually was deployed, verified and clinically validated on several patients.
- A methodology for the development and customization of an IoT oriented ECG analysis application is presented and evaluated. The ECG application, implemented and evaluated on a commercial IoT node, incorporates machine learning algorithms in order to provide the necessary algorithmic infrastructure to assess the status of an examined heart beat.
- The design of the ECG application is augmented to offer a matrix of different operating points, which provide tradeoffs between the consumption of resources on the device and the provided Service Quality to the user. This characteristic is a key feature to allow for the efficient, dynamic operation of the ECG analysis application in an Edge computing setup, including Gateways and multiple IoT nodes.
- We present a novel methodology for creating FPGA based HW accelerators of machine learning algorithms using High Level Synthesis (HLS) [97]. The methodology takes as input the high level (C-based) representation of a computational kernel and via source code restructuring and automated Design Space Exploration (DSE) of the HLS directives, provides a set of size vs efficiency Pareto optimal HW accelerator designs. The DSE execution latency is significantly reduced, by applying a number of proposed design space pruning guidelines.

The concept of the Edge Gateway, which support the execution of local processing of numerous IoT nodes, is considered to be taken up by a many-core system with Network-on-Chip interconnection of its processing elements. This kind of processing unit, provides the necessary infrastructure for the concurrent execution of numerous tasks, run-time encapsulation of the different applications, as well as enough resources to support multiple communication interfaces for connection with devices in lower and higher levels of the Edge architecture. The tradeoff of this choice is that mapping the incoming applications to a many-core system is such a computationally intensive task that can be the breaking point of the run-time efficiency of the system [116, 130]. Consequently, inspired by other researchers we adopt the distributed resource management paradigm, where decision making is broken up to more than one agents. The presented contributions are not limited to Edge computing systems and are as follows:

- We provide the detailed design of a Distributed Run-time Resource Management framework targeting many-core, NoC interconnected processing elements. We provide the high level algorithmic description of the involved distributed agents, as well as their low level implementation in the form of Finite State Machines. Furthermore, we detail the description of an inter-node signal and data exchange mechanism, necessary for the distributed decision making.

- We introduce novel run-time strategies with respect to the instantiation of incoming applications on the target system, as well as the bargaining of resources of the already admitted ones, in an effort to minimize their execution latency.

- We identify and analyze the correlation of the efficiency of DRTRM with respect to the arrival rate of incoming applications on the system. To mitigate the effect of stressing scenarios, we propose and design an Application-Arrival aware DRTRM framework for many-core systems, which utilizes VFS techniques to enforce an application admission regulation policy in a purely distributed manner.

- We provide a second level of exploitation of the VFS based techniques by meticulously mapping the parts of DRTRM on the system according to their computational requirements in order to maximize the energy consumption gains of our framework.

- We introduce SoftRM, a fault tolerant DRTRM framework which is able to dynamically react to failures of PEs in a self-organized way. This is performed using a consensus agreement algorithm enhanced with workload awareness to achieve an effective replacement policy.

- We incorporate in SoftRM a failure detection algorithm, which takes advantage of the communication between PEs for resource allocation purposes in order to significantly reduce the inflicted communication overhead of detection.
- We implement the aforementioned versions of DRTRM on Intel SCC, an actual many-core NoC based system. We fine-tune its design parameters and evaluate it against a mix of parallel applications, derived from the single IoT node.

Having identified the internals of the Single Edge Gateway, the analysis of the system proceeds to the identification of the properties and dynamics of its scaled up version of multiple Gateways. Prior work has proposed solutions to the problem of allocation of resources between IoT nodes and a single Gateway [189], but the co-ordination of multiple IoT nodes and multiples Gateways is still an open issue. In comparison to the single Gateway allocation problem, the multiple Gateways one is characterised by more dynamic re-allocation of resources, since IoT nodes are mobile and can connect to many Gateways. Consequently, a consistent way of collaborative resource negotiations amongst Gateways is necessary to provide guarantees that the IoT nodes are efficiently bound to Gateways and no Gateway is overburdened with tasks. The contributions of this thesis with respect to this issue are the following:

- We identify and formulate the binding problem of multiple IoT nodes to multiple Gateways in an Edge computing system, where the IoT nodes co-operatively execute their tasks with the aid of Gateways.
- We propose, design and evaluate a distributed run-time resource allocation scheme, for the negotiation of the bindings of IoT nodes to the Gateways. The proposed scheme is based on trade concepts in order to dynamically evaluate the negotiation possibilities.

The thesis is organized in Chapters, which group and highlight the aforementioned contributions, as presented in Fig. 1.6. In more detail, the included Chapters are organized as follows:

Chapter 2: This Chapter, provides an in-depth analysis of prior work regarding all the examined sub-topics of the presented thesis.

Chapter 3: This Chapter provides all the design and implementation details with respect to the contributions on the level of a single embedded node. Section 3.2, presents the architecture of a Smart Wearable Wound Management system, including details about its FPGA based simulation, initial SW prototype and its final HW and SW architecture. Section 3.3, presents the overview and

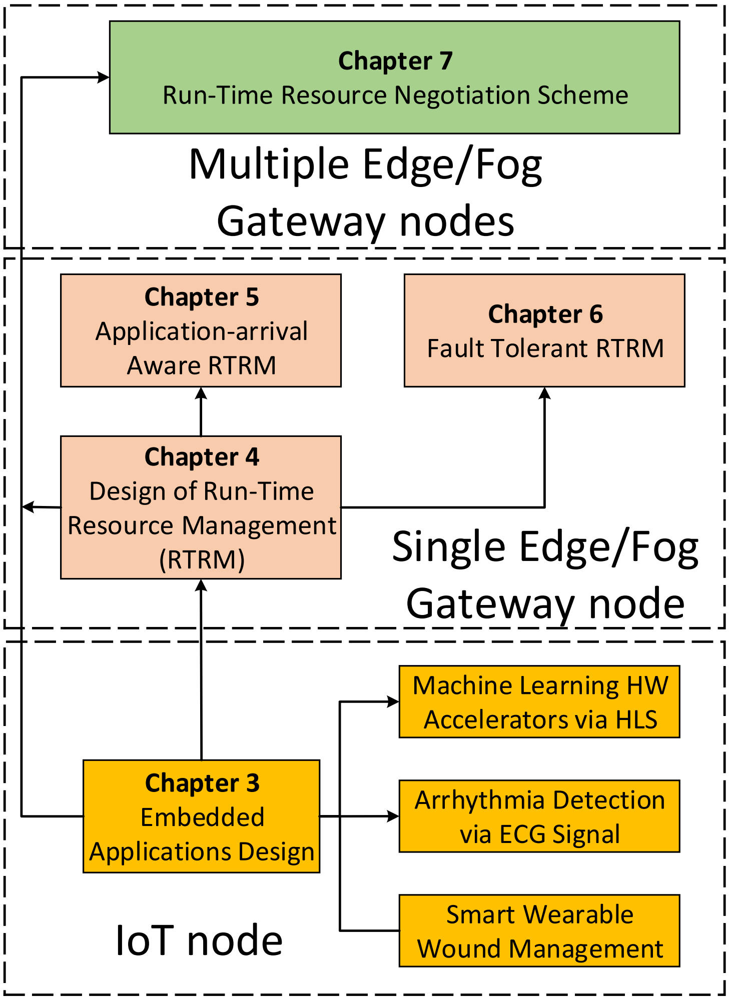

Chapter 7
Run-Time Resource Negotiation Scheme
Multiple Edge/Fog Gateway nodes
Chapter 5
Application-arrival Aware RTRM
Chapter 6
Fault Tolerant RTRM
Chapter 4
Design of Run-Time Resource Management (RTRM)
Single Edge/Fog Gateway node
Machine Learning HW Accelerators via HLS
Chapter 3
Embedded Applications Design
Arrhythmia Detection via ECG Signal
Smart Wearable Wound Management
IoT node

Figure 1.6: Schematic overview of the contributions of the current thesis.

building steps of an ECG analysis application for IoT nodes. The Section concludes with the customization of the application in order to provide dynamic configuration points for an Gateway based Edge computing system. Section 3.4, focuses on a methodology for developing efficient HW accelerators of the computationally intensive parts of the ECG application. The methodology aims at the design of HW accelerators via High Level Synthesis and targets embedded platforms, which incorporate CPU and FPGA logic on the same SoC, as illustrated in Fig. 1.4a.

Chapter 4: This Chapter introduces the details of a Distributed Run-Time Resource Management framework for many-core systems. The overview of the target system model is presented, with respect to HW infrastructure and target applications, followed by an in-depth analysis of the different distributed agents required for the decision making functionality to be carried out. The high level decision making logic is followed by details of the implementation specifics of the presented framework, focusing on its run-time states and their succession, as well as the necessary communication infrastructure for the co-ordination of the agents in the distributed system.

Chapter 5: Following the description of DRTRM, this Chapter provides an analysis of its effectiveness under different scenarios of input rate of incoming applications. The observed run-time inefficiencies are analysed, providing also insights on how the distributed management complicates the design of dynamic adaptation policies. This analysis lays the ground work for the presentation of an effective VFS based run-time mitigation strategy, which is incorporated into DRTRM and evaluated with respect to the VFS tuning knobs of the HW system architecture.

Chapter 6: The distributed nature of DRTRM, deprives the system of a centralized point of failure, but still errors in the distributed managing agents can lead to unacceptable system operation. In this Chapter the probability of such failures is modelled and an extension to DRTRM is presented, which is able to dynamically mitigate a permanent failure of a distributed agent and restore the system to a stable state. The recovery is performed in a self-organizing manner by means of a consensus algorithm called PAXOS, extended with workload awareness characteristics. SoftRM is eventually evaluated against other run-time techniques for fault tolerance in distributed systems.

Chapter 7: Chapter 7 applies the concepts of distributed resource management for run-time negotiation between multiple Edge Gateways. The Chapter provides a description and formal model of the setup of multiple Gateways and IoT nodes, including the problem of their run-time binding. This problem, is optimized dynamically by establishing trade based negotiations between the Gateways in order to re-organize the binding of nodes to Gateways and increase the offered Service Quality to the user. The proposed run-time scheme is evaluated using a custom built system simulator, which takes into account the topology and exchanged messages of the deployed computing devices.

Chapter 8: This final Chapter of the thesis summarizes the conclusions of the conducted research under description. In addition, for each of the aforementioned thematic units (Chapters) we provide a number of remaining open issues and direction for future research activities.

Chapter 2

Prior Art

2.1 Design of embedded applications

The following Subsections aim at providing a summary of well known works related to the design of applications with similar requirements as the applications presented in Chapter 3. The general topic of application design in contemporary embedded systems is out of the scope of this documentation of prior art.

2.1.1 Wearable Devices for Chronic Wounds Management

The first application of interest in this thesis, is the management of Chronic Wounds with the help of wearable devices. A particularly interesting topic of this field is the development of sensors for the real-time monitoring of in-wound parameters. In [234], authors have developed a wound monitoring device based on skin impedance. By measuring impedance over a large range of frequencies, the device is able to produce a mapping of the wound. The device is battery operated, based on Digital Signal Processor (DSP), including a front-end with an array of electrodes embedded in a sterile carrier dressing. The implemented system was clinically validated in a phase 2 clinical trial, including 34 patients. The trial showed that the impedance measurement can be useful in characterizing the status of tissues and can be crenellated with other wound parameters such as transepidermal water loss (TEWL). However, the implemented device had no tele-monitoring abilities, a feature which was set as a future goal of the design.

Authors of [159], make use of of-the-shelf temperature, passive moisture and pressure sensors in order to deploy a system for remote wound monitoring. The sensors are placed on the wound and are covered by a bandage. This deployment requires a wireless data transmission, like the one proposed in our work, and the authors chose Zigbee, due to its simplicity in implementation. All sensors were mounted on the same board and data collection was performed via a developed Android application on a tablet. Experiments on a healthy volunteer validated the ability of the system to provide reliable, real-time measurements.

Negative Pressure Wound Therapy (NPWT) is a well established treating method for curing chronic wounds. The negative pressure applied on the wound results in (i) shrinkage of the wound, (ii) removal of the wound exudate and (iii) stabilization of the wound environment [118]. Due to the effectiveness of the therapy, stationary and wearable devices, which deliver NPWT on patients can produce high revenues and consequently there are many relevant industrial products. One popular such device is the KCI Vacuum Assisted Closure, or V.A.C. Therapy System [118], which introduces a sponge into the wound, a dressing on top of it and a device for pumping the exudate from the wound.

Another representative of such systems is Smith and Nephew company, which offers a wide range of products in the domain. Its most compact device is PICO [11], a fully disposable NPWT system operating on battery supply for approximately one week. The device is oriented towards minimum size, maximum portability, it has an integrated canister, and no sophisticated electronic elements except from a very basic User Interface. RENASYS [12] is a line of disposable canister products which comes into three different variations. The basic one, is a stationary device with pure mechanical control, high durability and dependability, able to support large canisters. The device is mainly intended for wound care facilities and hospitals.

Its more lightweight version is RENASYS GO [12], which first introduced the use of more sophisticated electronic equipment with respect to User Interface and pump control, e.g. the user can see warnings, alarms and dynamically designate the type of therapy (continuous vs intermittent). TOUCH [13] is the newest product of the RENASYS line, which has significant electronic functionality, including a wide touch screen. This screen is used for the configuration of the device, as well as for on-board education, assistance and therapy logging tools. In addition, the applied therapy can be tailored to the each patient specifying parameters such as the rate of reaching the maximum vacuum on the wound.

In total, there is a gradual increase in the sophistication of wearable devices delivering NPWT, but still the features of remote connectivity and tele-monitoring of the therapy by medical experts have not yet been incorporated into mainstream industrial products. The real-time monitoring of wound parameters is still an open issue and its combination with remote connectivity completes the picture of Smart Wearable Wound Management Device.

2.1.2 Arrhythmia detection via the Electrocardiogram signal

The second contribution of this thesis is the development of an IoT based medical application for the analysis of the Electrocardiogram (ECG) signal in order to detect arrhythmias in the heart of the user. A recent literature survey of techniques for arrhythmia detection [149], validates the popularity and effectiveness of machine learning algorithms for assessment of the physiology of the ECG signal. One of the most frequently utilized classification algorithms is Support Vector Machines (SVM) classifier, achieving high classification accuracy of more than 80% in all cases, reaching up to 100% for certain datasets [149].

A feature extractions step proceeds the classification stage (both for SVM and other algorithms), based on combinations of time and/or frequency domain information of the input signal. The feature extractions step based on frequency domain characteristics are dominated by Wavelet Transformation (WT), which we also employ in the presented thesis. The choice of WT and SVM algorithms, is also advocated by their computational structure which can be optimized for fast execution on low-end embedded systems.

In [29] the authors make use of Wavelet Transformation in order to denoise incoming segmented heart beats and cross-correlate them according to template ones. This cross-crenellation is performed via the cross wavelet transform (XWT). For the segmentation of the heart beats, R-peak detection is utilized as in our implemented ECG analysis application. An input heart beat is eventually analysed according to its cross-correlation value and a number of thresholds.

With respect to the IoT infrastructure, authors of [28] provide a survey about key technologies and challenges regarding IoT in the domain of Smart Healthcare. Almost half of their work is devoted to the examination of different communication alternatives for the IoT nodes. As fas as short range communication is concerned, they evaluate the BLE and ZigBee, short range, wireless communication interfaces, concluding that BLE is more suitable for medical applications due to its secure features, acceptable range, low latency and power consumption as well as robustness to interference. Regarding long range wireless communication, the authors focus on Low-Power Wide-Area-Networks (LPWANs) comparing SigFox, LoRaWAN and NB-IoT and concluding that the latter is the most suitable due to its long range, high energy efficiency and ability to support many devices.

Some first efforts of IoT based ECG monitoring were performed in [161], where an Android application has been developed in order to sample the values of the ECG signal and upload them to a Cloud based infrastructure, while also performing visualisation and logging on the mobile device. The data are uploaded to the Cloud via a web service composed of a secure File Transfer protocol server.

The authors of [16], adopt the concept of Wireless Body Area Networks (WBANs) in order to conduct data aggregation from on-body sensors. WBANs are dependent on a Gateway or mobile device in order to upload their sensed data on a Back-end server, but the authors argue that all sensors should have the necessary communication infrastructure to upload data even in the case of a Gateway failure. Their approach, aims at classifying the outgoing packets of a WBAN according to their urgency and avoid the upload of non-urgent data. In this way, the system exhibits energy saving, which allow for both sensors and Gateways to have longer operation cycles.

In [212], a lightweight ECG acquisition sensor has been developed, which communicates the ECG signal data over ZigBee, short range wireless communication protocol. Local data are transmitted to a Gateway connected to the Internet, in order for the data to be uploaded to an IoT server. This server has the software infrastructure to convert the input from many different sensors to an universally accepted storage format. This work has focused only on ECG acquisition and does not provide solutions regarding the assessment of the captured data.

In [113] authors introduce an ECG based health monitoring infrastructure, combining IoT sensing and Cloud processing. Data are sampled from on-human sensors and are uploaded via a Gateway to the Cloud. Local processing includes the filtering of the captured ECG and the embedding of watermark data on the signal to protect it from forgery. The produced signal is uploaded to the Cloud, where several time and frequency domain features are extracted in order for the heart beat to be classified. The classification is performed using a binary Support Vectors Machines classifier, similar to the one utilized in our developed application. Nevertheless, in our case the classifier has been fine-tuned to be suitable for local processing and not require high-end Cloud infrastructure.

Authors of [27], propose a hierarchical approach based on Fog computing for the distribution of processing and decision making of Healthcare IoT. A Gateway is responsible for the execution of local analytics and ECG based heart beat assessment, using SVM classifiers as in our work. BLE interface is utilized as the wireless communication protocol for local acquisition of ECG data from sensors. Within predefined intervals, data are uploaded from the Fog Gateway to the Cloud, where further processing takes place. The management of all layers of computation is performed by an enhanced version of MAPE-K model proposed by IBM [27].

In summary, there are many works on the deployment of ECG analysis applications, but limited effort on the deployment of such applications on the context of IoT and especially Edge computing. Most importantly, the existing deployments of applications in this computing infrastructure focus on the mapping of the tasks in the different layers of the architecture. However, this mapping is static, i.e. it cannot dynamically adapt to the run-time condition of the system and fully take advantage of its offloading capabilities.

2.2 Automated High Level Synthesis HW accelerator generation

High Level Synthesis provides a transparent synthesis flow for designing hardware from high-level language implementations of algorithms. Several research works have proposed general design exploration strategies for HLS-based architectural optimization, to alleviate problems such as memory bottlenecks of the implemented accelerators [146, 193, 245, 246]. The proposed exploration framework forms an automated tool-flow solution already fully integrated with Vivado-HLS. Implementing the exploration framework in modular manner, i.e. not integrating the exploration engine inside the HLS engine (in any case not possible with Vivado-HLS, which is not an open source tool) is strategic design decision that enables the proposed framework to be reused with other HLS tools with little effort.

This type of modularity, i.e. treating the HLS engine as an external tool, has also been adopted by other state-of-art design space exploration frameworks targeting the HLS domain, but most of these works have proposed exploration strategies without taking into account the code structure of the target algorithm implementation [245, 146, 193, 246]. Recently, a lot of attention has been shifted towards more targeted HLS optimization techniques tailored to the specific structural characteristics of the algorithmic descriptions, e.g. stencil computations [36, 63, 70, 69, 233].

Concerning the efficiency of the way that an algorithm accesses memory, in [63] a methodology for automated memory partitioning is presented enabling parallel computation units to efficiently access multiple independent memory banks. The Z-polyhedral model for program analysis is used to address bank mapping and minimization of the total amount of memory required for the partitioned banks. In [36], authors perform memory profiling of various applications and split a single data structure into different memory banks for data parallelism to address the memory bottleneck problem. In [70], authors develop a micro-architecture with multiple memory systems, each one customized to allow parallel access to elements of one array. These systems manipulate an incoming stream of array elements so that those corresponding to parallel array references are led to different memory banks of the computation kernel. Similar to this principle, we focus on one computation kernel and partition or reshape arrays according to the access pattern of the algorithm so that the number of memory banks or their width is sufficient for the required parallelism.

In [69, 233] an exploration algorithm tries to determine the partitioning of an array so that elements accessed in parallel are assigned to different array partitions. Memory partitioning is combined with memory access scheduling in a cycle-accurate way. Authors of [214] target computational kernels at which parallel access to a coefficient matrix takes place. They propose that frequently used data are cached by on-chip registers organized as chains to enable data caching with reduced hardware overhead. Memory partitioning is then performed on infrequently re-used data, using a padding technique that minimizes storage overheads when the partitioned data include zero values. Our proposed techniques are closer to these approaches, since we also partition each array according to the access pattern in each loop iteration to allow concurrent access. We further examine grouping these elements in elements of greater word-width and also combine the two techniques. As opposed to utilizing one-dimensional arrays and perfect loop nests, we apply our methodology to both one and two- dimensional arrays and on an imperfect loop nest.

In [164], authors create FPGA based accelerators for Iterative Stencil Loop algorithms using data dependency analysis on their High Level representation and then design space exploration to identify Pareto-optimal solutions in terms of area and accelerator throughput. They propose a template computational unit architecture, which operates on a specific data window in order to evaluate the outcome of a number of iterations. Calculations are performed in a cone shaped manner, making use of data values from a variable depth of preceding iterations. An accelerator instance comprises of many such units, each one processing a different segment of the input data matrix at the same time. The specific characteristics of these units, i.e. the number and depth of the cones, is the outcome of the design space exploration and characterises the efficiency of the derived accelerator.

In [142], in order to allow concurrent access to memory references across different loop iterations, authors extend the algorithm proposed in [69] to address the issue of mapping array accesses to partitioned memory banks and scheduling conflicting memory accesses to different execution cycles of the HW accelerator. Similar to this idea, we utilize partitioning of the initial data set to enable parallel execution of our computational kernel on the segments of the initial set and thus allow concurrent execution of loop iterations. Nested loops are also targeted in [257], where the unroll factor of each loop and the presence or lack of dataflow directive is determined to optimize the throughput vs area tradeoff of the produced accelerator. The dataflow directive of Vivado HLS tool enables the concurrent execution of loops and the authors focus on minimizing the initiation interval of the kernel by iteratively optimizing the longest loop through exploration of various unroll factors. We also apply this directive to allow instances of the SVM kernel, which are basically nested loops, to execute in parallel. The instances are optimized using our proposed design space exploration, that includes extensive investigation of loop unrolling.

In [71] the authors propose a technique to optimize on-chip memory allocation using loop transformations. Buffers for reusable data are utilized to save data accessed by consecutive memory references. Loop transformation is used to reduce the buffer size by improving data locality of array accesses. Authors in [64] examine the adverse effects of typical memory partitioning techniques and especially bank switching and propose an exploration of loop unrolling to identify an unroll factor that alleviates the problem. Loop unrolling technique is extensively utilized in our work and is combined with partitioning techniques in order to match the array layout to the algorithm access pattern and address memory burst issues and bottlenecks.

The presented work in Section 3.4 includes a novel methodology, implemented on top of a commercial HLS tool, that combines data-, instruction-level parallelism and memory architecture customization to deliver efficient delay-area SVM design solutions. Manual HLS code source restructuring is combined with an automatic design space exploration framework, which fine tunes the knobs of the employed HLS tool. Efficient exploration is achieved by introducing a set of design space pruning heuristics and incorporating an optimization algorithm as a component of the framework.

2.3 Distributed Run-Time Resource Management

Several research works have presented run-time resource management schemes aiming at either efficient resource utilization or application optimization. Scheduling of tasks on many-core architectures remains a very active research topic and authors in [176, 177], provide works which make use of multi-agent systems in order to derive scheduling patterns in a computationally effective manner. The authors in [176] present a heuristic based decision making algorithm, which by taking advantage of the inherent properties of the execution profile of benchmarks on a varying number of processing elements, is proven to converge to an optimally fair scheduling of tasks. Similar principles are introduced in [177], where the concept of game theory is employed in order to model the application characteristics and deduce an equilibrium, which is translated to an optimal scheduling solution converging to the one provided by a centralized scheduling scheme. While built with inherent scalability, both works assume a platform which is based on a shared memory architecture for inter-core data exchange. This, combined with the fact that the communication requirements of the applications are not explicitly taken into account, may lead to sub-optimal solutions.

Run-time scheduling on many-core systems is also addressed in a centralized manner in [178]. The core of the proposed decision making is a greedy algorithm with takes advantage of the concave characteristics of the Instructions Per Cycle (IPC) metrics of the executing tasks. This property allows the scheduler to produce optimal results in simulation and near-optimal solutions in real systems. Following the centralized approach, authors in [167] present a run-time resource management policy able to efficiently manage a many-core system interconnected via a Network-on-Chip. Their targeted system contains fine grain reconfigurable HW tiles and two task migration algorithms are utilized to efficiently transfer the tasks between different processing elements. Nevertheless, as the number of tiles increases, system degradation appears.

On the field of run-time mapping on many-core systems, authors in [18] present a distributed, cluster oriented framework for homogeneous platforms, which is based on agents for the task-to-cluster mapping. A scheme that tries to reduce this on-chip node intercommunication is proposed in [73], where a decentralized cluster-based scheme for task mapping designed for reduction of the communication traffic between agents, is presented. A mapping approach that tries to take into account the NoC topology but it is based on static application models is presented in [248]. A stochastic hill climbing algorithm, which adapts in order to find the start node in the application mapping on a NoC is presented in [89]. The proposed scheme tries to map an application onto an optimum contiguous area of the available nodes. The communication aspects of the mapped applications are taken into account in [249], formulating the mapping as an Integer Linear Programming problem and proposing a centralized solution using a polynomial-time priority based heuristic. The topological properties of the allocated cores are well recognized as an important factor and they are also exploited in the work of [129]. On top of the topological characteristics, the authors also propose an on-the-fly application performance estimation scheme.

Apart from the topology, heterogeneity is another aspect of modern many-core systems. Authors in [20] present a divide and conquer based distributed run-time mapping framework for both homogeneous and heterogeneous platforms by introducing a matching factor for different PEs, while a system-level run-time resource management scheme that focuses on the management of errors is presented in [90]. However, even though the proposed approaches produce very good results, both in homogeneous and heterogeneous systems, they do not exploit any malleability aspect as they are designed for fixed-size applications only. In [83], authors present SPARTA a throughput-aware run-time task allocation scheme which incorporates machine learning techniques to predict the behaviour of running applications and produce an efficient scheduling. Similarly to our approach, this work makes use of existing Linux infrastructure in order to apply its resource allocation policy. However, it is built as a kernel module, while we refrain from this approach in order to introduce increased flexibility and safety of operations by running on the user-space level.

On the field of self-organized and dynamic systems, from the aspect of malleable or parallel applications, authors in [186] present a greedy centralized scheduling strategy and demonstrate that the importance of efficiency varies with respect to the characteristics of the workload. Regarding parallel applications, authors in [80] show that application malleability provides up to a 15% speedup over component migration alone on a dynamic cluster environment. In order to keep the system distributed and offer the ability to decide for itself and be self-organizable, some approaches implement machine-learning techniques such as artificial networks [120] and reinforcement learning [56] for estimating application performance and applying power optimizations, respectively. Authors in [130] and [21] present distributed application mapping for malleable applications supporting also self-organization. The properties of malleable applications have also been utilized in [199], where a distributed dynamic power management scheme is presented. The main concept is that each application is governed by its own power manager and power managers trade resources in order to optimize the energy efficiency of the system.

In [181] authors present PICASO, a run-time management system which targets many-core systems, where communication is performed in a message-passing manner. PICASO makes use of an hierarchy of micro and pico-kernels, where the first are responsible for job execution and a number of them is managed by a micro-kernel. However, resource exchange is performed only amongst different micro-kernels. This hierarchy is similar to the proposed methodology, but *DRTRM* provides a more fine-grained resource exchange infrastructure where resource bargaining is performed in application granularity. Consequently, the proposed approach is more flexible and permits the run-time adoption of various inter-application resource exchange policies with no need of re-initialization of the entire framework.

In the field of many-core operating systems, Barrelfish [34] is built in a highly distributed “multikernel” fashion. In each core of the system, a kernel of limited size is running and no memory is shared between these kernels. In Akaros [185], the kernel maintains a global overview of the system but the available processing elements are an allocatable resource to applications, in a manner similar to memory. Subsequently, each application implements its own, user-level scheduling policy in order to utilize its acquired resources. In Factored OS (FOS) [235], kernel services are distributed in a set of communicating servers on the chip. Tesselation OS [68] groups hardware resources in Cells and allows the user of these resources to apply its customized policies regarding the mapping of an application inside a Cell. A resource allocation broker service, facilitates the exchange or resources between Cells aiming at optimizing system wide objectives. In an effort to extend Linux in many-core heterogeneous systems, Popcorn Linux [30] proposes a design of multiple traditional kernel instances, where each instance governs over resources of the same Instruction Set Architecture. The kernel instances communicate directly in order to provide the required functionality to the user space as if there was a single kernel monitoring the entire system.

In summary, while the work on many-core run-time resource management system is great both in quantity and quality, the focus of the publications so far has been the resource management algorithms and policies. Chapter 4, presents the design and implementation of a self-contained distributed run-time resource management framework, targeting NoC based many core systems, which is evaluated on an actual target system in order to capture the dynamics of real workload execution and thus proceed one step further from simulation. The framework makes use of existing Linux infrastructure, refraining from introducing a new operating system design in order to reduce the design complexity and make use of a concrete foundation for the final implementation. Additionally, comprehensive resource management technique is presented that, taking account the application profiles, distributes the cores among multiple running applications, while maximizing the overall system performance and keeping communication overhead low.

2.4 Application-arrival aware DRTRM

As detailed in Section 2.3 the Distributed Run-Time Resource management paradigm has been investigated in various works, however in all of them the arrival rate of applications is not taken into account. This parameter is investigated in [256], where a heuristic for application admission control is proposed to aid the service provider of cloud or grid based systems to meet the Quality of Service requirements of the user. In [179], a job arrival aware scheduler is proposed targeting asymmetric multi-processors. The centralized scheduler has a complete overview of the system and uses queuing theory concepts and arrival rate predictions to make run-time decisions for the migration of jobs between different cluster of PEs.

Authors of [87] target embedded, hard, real-time systems and propose a feedback loop approach using control theory to regulate and fine-tune application admission. Their approach is similar to our proposed one, but targets systems of only a few processors, whereas ours is designed for many-core systems. In [140] the concept of a job queue in a many-core system is extended by introducing Isonet, a hardware based dynamic load distribution and balancing manager. This manager makes the selection of the jobs to be executed based on micro-network of load balancing modules, which take into account the current load conditions of the system, in an ultimate effort to maximize system speedup.

The described approaches, mainly focus on performance optimization, neglecting energy optimization goals, which is becoming an important requirement for contemporary management frameworks. In [155] authors propose ARTE, an Application-specific Run-Time Management framework, which makes use of queuing theory concepts to maximize the QoS of a many-core system, while respecting its power budget. The employed queuing model utilizes specific information for each application, resulting from design time analysis. In contrast to our approach, ARTE operates in a centralized manner embedded inside the OS of a multi-core system. Another centralized approach is VARSHA++ [125], a run-time framework extended with Dynamic Voltage Scaling capabilities, for application mapping in multi-processor chips operating under dark-silicon constraints. An input queue is used for application arrival, while task mapping and scheduling is performed taking into account variation characteristics and reliability predictions for the target chip.

Thermal management has also been addreesed, as on [88] where an agent-based framework is proposed, in order to reduce peak temperature on the target system, while offering enhanced performance and reduced energy consumption. Agents' negotiations are based on a supply/demand economical model that distributes power in a proactive manner. Agent-based management is also utilized in [98], performing distributed task migration for thermal management. Neural networks are used to predict peak temperatures in cases of workload variation, showing higher performance and reduced migration overhead compared to other centralized predictive dynamic thermal managers. An Agent-based power management presented in [198] provides the opportunity of power state control of resources for each individual application operating on the system. The determination of each power state is performed with respect to improving the overall energy consumption of the system under management. To achieve that, a distributed power management approach is proposed based on game-theory, which achieves significant results both in scalability and energy efficiency.

2.5 Fault-tolerant DRTRM

A considerable amount of work is focused on extending the lifetime reliability (i.e. minimizing core failures) of many-core systems by appropriately mapping tasks using metrics such as Mean Time to Failure (MTTF). A run-time task mapping design, which extends lifetime of the system by using a wear-based heuristic is presented in [105]. However, this work does not examine component failures and assumes that such failures are detected by the operating system or another hardware resource. Authors in [75] present a task-mapping technique to maximize MTTF of an MPSoC considering the ageing of NoC-links. However, in case of a permanent core failure the system collapses and has to be restarted in order to work properly.

The works in [147, 238], utilize Branch and Bound based algorithms to perform application mapping on reconfigurable many-core NoCs, where the configuration of PEs functions and interconnections is adjusted dynamically. Mapping decisions are centralized and the optimization objectives take into account reliability, energy and performance. The authors assume that spare cores can substitute any faulty PEs and thus focus on tolerating transient errors of the communication links between them. Application mapping is also targeted in [103], where a centralized feedback loop approach is employed to achieve maximum system performance under dark silicon thermal constraints and meet the reliability requirements of the applications under ageing and thermal effects.

Task mapping is combined with Dynamic Voltage and Frequency scaling in [76] to increase the reliability of the system, while transient faults in hardware are tolerated via selective task replication. A genetic algorithm is used to perform design space exploration and determine the task to PE binding, as well as its VFS values.Triple or Double Modular Redundancy (TMR or DMR) is extended in [165] to provide prolonged fault free system lifetime. In case of failures, instead of task remapping to sustain the number of redundant tasks, the authors propose to preserve the mapping and adjust the functionality of the included voter task. N Modular Redundancy (NMR) is the main fault tolerance mechanism in [162], targeting commercial many-core SoCs such as [23]. Transient faults in PEs are encountered by comparing the output of the N-times replicated task and repairing any instance that exhibits output divergence.The required voter task is also doubled and the two voters constantly repair one another in a round robin way. Authors in [43] propose a self-adaptive engine for fault management in multi-core systems. Failure detection is achieved by task replication and comparison of the different outputs. Application activity is coordinated by a centralized core called *fabric controller*, which executes the fault management layer and is considered hardened by-design.

Authors in [112] and [197] opt for increased system dependability by examining the combination of reliability techniques in different layers of many-core systems. Regarding resource allocation, in [112] the run-time system layer performs reliability-driven core allocation and task mapping. Similarly in [197], the resource manager utilizes redundant multi-threading for task execution resilience accompanied by a variation-aware task mapping. Both approaches depend on centralized resource allocation and no recovery mechanism is presented in case of resource manager failure.

In [59, 127] run-time fault-aware resource management is addressed using three types of cores, i.e. *regular*, *spare* and *manager*. Managers overview the system and monitor the status of other cores, while regular and spare cores execute incoming applications. In case of a permanent or transient failure of a regular core, spare cores are used to replace it and workload is migrated technique from the faulty to the spare core. The difference between the two works is that [59] employs static spare core provisioning while [127] dynamic, i.e. a subset of the cores provided to the application are provisioned for fault tolerance. However, a manager failure is not examined at all and in such a case the fault-aware nature of the proposed schemes disappears. Our work bridges this gap by focusing on the failure of similar managerial agents and provides a design alternative to both static and dynamic provisioning of spare cores. In [107] fault mitigation in networks consisting of nodes of re-configurable PEs (ReCoNodes) is proposed by leveraging both HW/SW techniques. An individual OS overviews each ReCoNode and lightweight "shadow" tasks replicate applications to provide error mitigation. The proposed system, provides an efficient intra-application fault tolerance infrastructure under the assumption that the administrative part (OS) is fault free, where our focus lies.

A System-level and Hierarchical Fault-Aware (SHiFA) approach is presented in [91] to provide run-time tolerance of manifested faults both in PEs and links of the NoC. To achieve efficient fault tolerant mappings and resilient application execution, SHiFA requires two kinds of distributed agents, i.e. system mobile master (MM) and application managers (AM) in its hierarchy. Despite the fact that this hierarchy is critical for the correct system operation, the authors provide no mechanism for recovery actions in case of one or more agent failures. Likewise, in [229] resource management is based on a hierarchical scheme, where a Global Manager (GMP) supervises the system and Local Managers (LMP) supervise application execution. Authors employ both hardware and software mechanisms to tolerate faults in the PEs and links of the NoC. Specifically, in the software stack a fault is mitigated in a cooperative manner between an LMP and GMP. Nevertheless, there is no provision for a dynamic scheme which will ensure system stability if one of these dedicated PEs fail. A holistic infrastructure for fault tolerant resource management is also presented in [25]. A System Health

Monitoring Unit has a global overview of the status of the links and PEs in a many-core NoC based system and collects and classifies information about manifested errors. This information is propagated to the Mapper-Scheduler unit, to produce appropriate run-time mappings of the applications.

In conclusion, reliability aware mapping techniques concentrate their effort on prolonging the lifetime of a system often neglecting to provide fault tolerance. Even if faults are addressed, the existing solutions mainly rely on utilizing redundancy techniques. On the other hand, our approach presented in Chapter 6 belongs to the category of fault tolerant frameworks, which aim at mitigating all manifested errors, while providing the infrastructure for application deployment and execution. In this case, reliable application mapping can still be used as a key component of resource management infrastructure. We highly differentiate from state-of-art [59, 127, 91, 229], by providing fault tolerance without spare core provisioning, thus constantly exploiting the full potential of our target many-core system as well as providing fault tolerant guarantees for all involved agents of the system, in all levels of the resource management hierarchy.

2.6 Multi-Gateway IoT systems

Although SQ management and computation offloading in Edge computing are among the foreseen challenges in IoT [60], there are not many research work addressing the joint problem. However, the problems of computation offloading, SQ management and quality of experience (QoE) have been addressed separately in other research fields including Wireless Sensor Networks (WSN) [239, 54, 31] and mobile computing [232, 135, 144].

Many research efforts have been conducted on computation offloading in mobile systems [135]. In these systems mobile devices are battery-powered and resource-constrained, but the destination of offloading (i.e. servers) are powerful computing devices. MiLAN [163, 110] is a middleware responsible to manage and allocate network resources for the applications, that *fuses* data from multiple sensors and needs to select optimal set of sensors. However, the SQ of each sensor is fixed. MiLAN only considers the bandwidth limitation of the network, while the processing capability is not modelled. Moreover, it does not model the on-board processing and therefore cannot support combinations of offloading schemes.

In [200], authors propose to minimize the energy consumption of sensor nodes by computation offloading. This approach considers each IoT device indvidually in order to find the optimal partitions of its application for computation offloading. The output of this approach can be used as the input to our problem. A decentralized game theoretic approach for computation offloading in mobile computing systems is presented in [55]. A single Gateway is considered, whose wireless channel is the scarce and shared resource, while the processing capability of the final destination server is unlimited. In addition, this approach does not support multiple levels of offloading for a device, limiting the run-time the decision to fully offloading the computation or fully processing on-board.

A joint optimization of bandwidth and computational resources for computation offloading in a dense deployment scenario is presented in [190], which considers the presence of radio interference. Nevertheless, the proposed solution is for single Gateway networks, and does not support multiple SQ levels of devices. In [241, 133], adaptive approaches are proposed to offload the computation from portable devices in order to extend their battery lifetime. Other research works have proposed to dynamically offload the computation [153, 119, 137], at a system with single Gateway (server) architecture. These solutions consider each device (or user) individually, thus not addressing the effect of different devices sharing the same limited resources.

Authors of [85], work on the computational offloading of tasks of mobile devices to Fog nodes and the scheduling of tasks inside the Fog node. They examine the problem of a single Fog node and multiple mobile device and formulate it as a non-convex quadratically constrained quadratic programming problem. Their optimization objective is to minimize the maximum $Cost$, which takes into account the Energy Consumption and execution latency of the applications. Their solution is heuristic and does not take into account the existence of multiple Fog nodes.

In [250] authors tackle the problem of computational offloading from mobile users to an Edge Cloud infrastructure, with the objective to minimize the weighted sum of the consumed energy by the mobile devices. Their problem is constrained by the time-sharing of the communication channel, the computation capacity of the Cloud infrastructure as well as the execution latency of the applications. By examining the problem of a single Edge Cloud infrastructure instance, they devise centralized policies for the decision of offloading per device, customized to the way that the communication channel is shared between mobile devices, i.e. time-division multiple access (TDMA) vs orthogonal frequency-division multiple access (OFDMA) sharing.

In [168] an approach based on game theory is proposed to dynamically allocate the bandwidth in a shared network channel in order to manage the quality of experience. Although the algorithm can be adjusted for the bandwidth allocation of a single IoT Gateway, it does not support any computation offloading aspect. The authors of [240] propose a feedback control scheme to manage the quality of service (QoS) of sensor nodes in a WSN. The QoS manager adapts the sampling rate, (i.e. SQ) of sensors at run-time based on observed parameters such as transmission delay or packet loss. However, the target system does not incorporate on-board processing (i.e. it always offloads all the gathered data). Packet loss is allowed to happen and then the system reacts to it by adjusting the SQ level of the sensor nodes.

Note that even though QoS is commonly used in networking literature to refer to the aspects of network service (e.g. throughput, delay, etc.), it may cover a wider range of parameters specially in the IoT domain [141]. For instance, the quality of the final results that user receives, i.e. Service Quality is a QoS parameter. In [150], a utility-based approach is studied for bandwidth allocation in wireless networks. The proposed approach is market-oriented but in a centralized fashion that is designed for single Gateway systems, which do not not support computation offloading.

The joint problem of bandwidth allocation and SQ management under resource constraints for a single Gateway IoT system has been studied in [189]. The optimal allocation bandwidth and computation offloading level is determined using a dynamic programming algorithm. However, in a multi-Gateway system, the problem has two interdependent sub-problems: binding and allocation. None of presented approaches is applicable to this problem as they assume a resourceful Cloud server as the offloading target, while in an Edge computing setup the computation is offloaded from IoT nodes to other devices with limited processing power. In addition, the assumption is made that the offloading policy of one device does not affect the other devices in their setup, which is not applicable to the limited, shared resources of Edge Gateways. Last, the examined systems do not involve multiple destinations for offloading (i.e. the binding problem) and application that operate on different SQ levels.

Chapter 3

Embedded applications design

3.1 Introduction

This Chapter provides an in-depth analysis of the design process of contemporary embedded systems. The presented case studies belong to the healthcare domain, which is a most promising target for the deployment of inter-connected sensor based systems. The added value of introducing computer science concepts in the health management are numerous, since patients will be offered better treatment and quality of life, as the provided healthcare services will be tailored to their individual needs. At the same time, the already stressed healthcare system will be alleviated by remote monitoring, which will reduce the need of patient hospitalization.

In this Chapter, the medical conditions under examination are (i) the management of Chronic, hard-to-heal wounds presented in Section 3.2 and (ii) arrhythmia detection using the Electrocardiogram signal, presented in Section 3.3. The focus of the wound management applications is to present the full design cycle of a clinically validated device prototype. The design of the ECG analysis application leans towards the IoT system architecture, including all the steps from high level algorithmic design to low level implementation, targeting Gateway based Edge computing systems. Using the algorithmic infrastructure produced in Section 3.3, a design methodology for efficient FPGA hardware accelerators for ECG analysis is presented in Section 3.4. These accelerators highly reduce the execution latency of the ECG analysis application and allow it to meet its critical run-time requirements.

3.2 Chronic Wound Management

In recent years there has been an increased focus on getting patients out of the hospital and back into their own homes as soon as possible. To reach this goal, new technologies that enable and assist this transfer have to be developed. A very important case is the one of chronic wound management of Venous Leg Ulcers (VLU) and Diabetic Foot Ulcers (DFU). Chronic venous insufficiency and leg ulcers affect approximately 1-2 people per 1000 of the general population, with approximately 10-20 people per 1,000 ever affected. In addition, amongst people with diabetes approximately 1-4% will develop a foot ulcer annually, and approximately 15% will develop at least one foot ulcer during their lifetime [33, 24].

The Negative Pressure Wound Therapy (NPWT) is increasingly applied in hospitals to treat chronic wounds by removing exudate and potentially infectious material and promoting also the formation of granulation tissue, thus accelerating the wound healing. Nevertheless, the healthcare costs of NPWT are relatively high since it necessitates hospitalization and medical acts. For many cases of NPWT patients, hospitalization is necessary but others are hospitalised simply because they require constant monitoring and immediate access to wound care specialists. Recently, portable NPWT systems have been developed and commercialised, providing several advantages for the patients such as discreteness and ease of operation. *However, the majority of these devices still offer only the single service of negative pressure on the wound and basic vacuum control.*

To enable monitoring and treatment of patients in their own home environment or long term care, an online wound monitoring ecosystem needs to be developed [216]. In the heart of this ecosystem, a Smart Negative Pressure Wearable Device (SNPWD) will make use of non-invasive sensors that allow objective, continuous, real-time monitoring of critical parameters of the patient's wound condition. In total, the wound management system will (i) collect data and monitor wound parameters via non-invasive integrated micro-sensors, (ii) offer the opportunity to provide personalised negative pressure wound therapy, and (iii) allow healthcare experts to be remotely aware of the patient's condition and receive alerts about situations that require direct actions.

3.2.1 System Architecture

Fig. 3.1 shows the overall architecture of the online wound management system as well as the localization of the respective sensors for wound healing/monitoring. The infrastructure is composed by a set of subsystems namely: (i) the Clinical

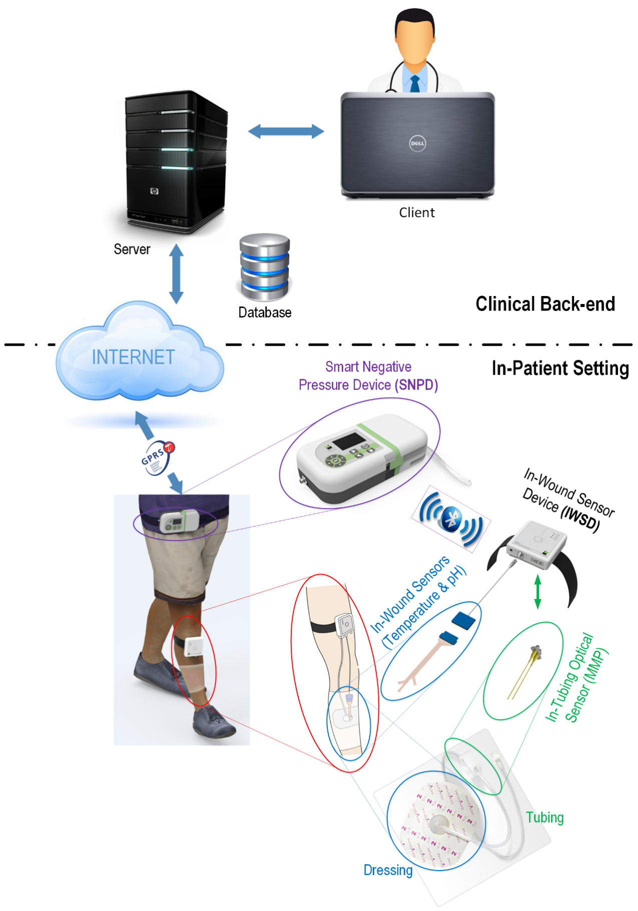

Server
Client
Database
Clinical Back-end
INTERNET
In-Patient Setting
Smart Negative Pressure Device (SNPD)
GPRS
In-Wound Sensor Device (IWSD)
In-Wound Sensors (Temperature & pH)
In-Tubing Optical Sensor (MMP)
Tubing
Dressing

Figure 3.1: Architecture of SWAN-iCare eco-system.

Back-End server integrated to the hospital infrastructure where sensor data are processed and stored, including a front-end for users to interact with the system, (ii) the In-Wound Sensor Device (IWSD) responsible for gathering data from on-wound sensors [216] (pH, Temperature, Matrix metalloproteinases (MMPs)) and communicating them to (iii) the Smart Negative Pressure Wearable Device that applies the negative pressure wound therapy, provides interfacing to the local user and uploads sensor data to the Back-end server.

IWSD transmits sensor data to the SNPWD via wireless Bluetooth Low Energy communication. The existence of the IWSD is a key factor towards deploying a usable and efficient wound management system, since it reduces the need for increased and lengthy wiring of the patient. Such wiring would be inconvenient and a most frequent point of failure, when in daily use of the system the wiring would be influenced or disconnected by patient activities. In addition, the modular architecture of the system enhances its flexibility and prospect to be incorporated into other wound monitoring systems, given that sensor data sampling can be performed independently of the treatment.

The main goal of SNPWD is to enable the application of negative pressure therapy on wounds that are hard to heel, with the additional feature of local and remote connectivity. In general, SNPWD is composed of the Smart Negative Pressure Device (SNPD), which is non-disposable and with a disposable part (Canister, Tubing, Dressing and Integrated Multi-Sensors). The SNPWD controls the pressure of a pump used to extract the excess of exudate generated by the wound and collects it in a disposable canister. Two pressure sensors are integrated in the device to enable the correct control of the applied pressure on the wound. The SNPD will be wirelessly connected to the Clinical Back-end through Wireless Wide Area Network (WWAN) over GPRS connection. The device is also equipped with a Bluetooth Low Energy communication module which is necessary for data exchange with the In-Wound sensor device.

3.2.2 Requirements of a HW emulation platform for SNPWD

Complex wearable devices such as SNPWD, require large design cycles, analysis and validation of the developed hardware (HW) and software (SW) components. The traditional design approaches that serialize HW/SW development are time consuming, especially within the context of research projects which allocate a significant amount of time to define the system requirements. Software simulation of the final system suffers from long simulation times, while at the same time it requires severe deprecation of the final code, since many components, e.g. Bluetooth devices, UIF, etc., are not realistically modelled.

Regarding the embedded software, a bare-metal design approach was adopted, to eliminate the validation and verification risks introduced by the software stack of an operating system,. Designing software for a wearable device, especially in the absence of OS, is directly coupled to the hardware, since there is no intermediate layer between the high level tasks and the low level software. The OS is substituted by a series of low level drivers which provide the means for the high level software to control the underlying hardware. Assuming that an SNPWD hardware prototype must be built, implying unexpected delays related to hardware assembly cycles, an emulation device with characteristics similar to the specifications of the SNPWD must be compiled in order to enable early SW development. In this way, issues regarding timing and communication

Table 3.1: Selection criteria and scoring for SNPWD emulation.

Type of emulation	Ease of programming without OS	uArch and performance close to original SNPWD	PCB port availability for high connectivity
Software	+	-	-
Fast processors	-	-	-
Slow processors	+	+	-
FPGA	+	+	+

can be addressed and software architectural design choices be validated and re-evaluated in case of inability to meet specific requirements.

In an effort to determine the best available platform to realize the SNPWD emulator, several options were examined. The basic qualitative selection criteria are i) the ease of the platform to be programmed in the absence of an OS, ii) the similarity of the CPU of the platform compared to the specifications of the CPU of SNPWD and iii) the available I/O interfaces to facilitate the building blocks of the SNPWD emulator. Specifically, we compare the cases of i) pure software emulation on personal desktop computers, ii) emulation using ARM-based embedded platforms with slow and fast cores and enriched peripheral hardware components (e.g. BeagleBoard [67]) and iii) reconfigurable FPGA devices. Table 3.1 reports the pros (+) and cons (-) of each device solution according to the desired qualitative criteria. As shown, the fine-grained emulation efficiency of FPGA devices fulfils the necessary requirements for building up the SNPWD system emulator, making possible to incorporate or simulate almost all required peripheral devices and develop the high level software in the absence of an OS.

3.2.3 FPGA-based emulation platform for SNPWD

Fig. 3.2 shows a photograph of the HW emulation platform, annotated with the specific components used to resemble the original SNPWD hardware. The basic element of the HW emulation platform is a Xilinx Spartan-III FPGA device [61]. The Spartan-III FPGA instantiates the control microprocessor and every interface of the peripheral devices is connected to it. We synthesize a MicroBlaze, a soft-core IP processor, provided by Xilinx. It is a RISC processor with 3-stage pipeline and clock frequency up to 50 MHz. Microblaze supports architectural parameters customization thus enabling exploration of differing

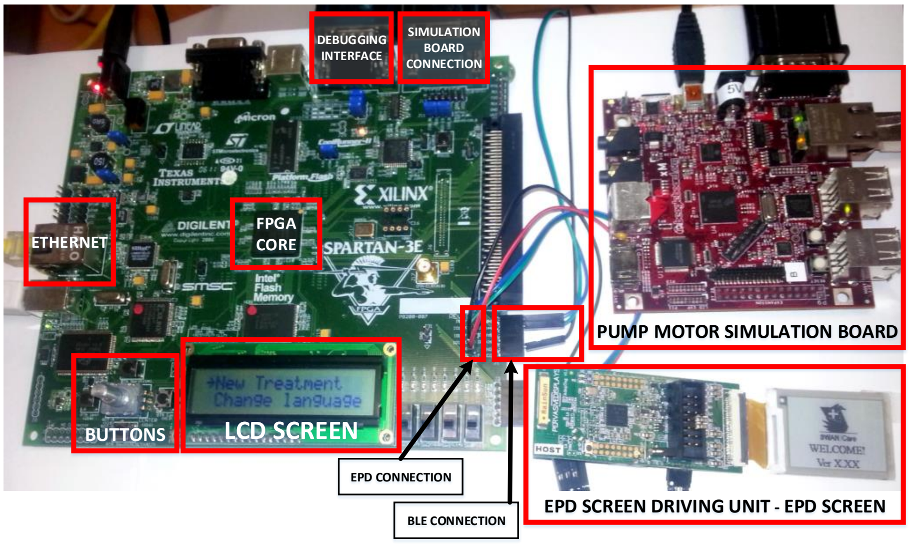

DEBUGGING INTERFACE
SIMULATION BOARD CONNECTION
PUMP MOTOR SIMULATION BOARD
ETHERNET
FPGA CORE
BUTTONS
LCD SCREEN
EPD CONNECTION
BLE CONNECTION
EPD SCREEN DRIVING UNIT - EPD SCREEN

Figure 3.2: The components of the HW emulation platform.

design configurations to be performed, in order to tailor the design to the characteristics of the application's SW components. It forms a quite good match considering the requirements of a wearable medical device since it is more powerful than typical microcontrollers and has similar architectural features to ARM processors which were chosen as the main processing unit of the SNPWD.

The FPGA comes as a part of a rich development kit, which provides many on-board modules and extension interfaces. The on-board Ethernet module of Spartan-III has was used as the communication interface with the clinical Back-end server, using the the LWIp TCP/IP protocol stack [86]. The Back-end server services for communicating the sensor data were implemented on a desktop computer where the medical device uploads its information. The communication between the on-wound sensors and the SNPWD is performed using the BLE protocol, via an external BLE transceiver from Dialog-semiconductors [196]. Since development time is an essential parameter, the transceiver was not fully incorporated into the FPGA, to avoid replication the communication interface between the two components. A desktop computer was used, which executed the Bluetooth stack necessary for the transceiver. This software was customized to forward all the incoming sensor readings to a serial port which was connected to the main processor of the SPARTAN-III FPGA.

The buttons of the SPARTAN-III kit were used as the UIF input, while the output was the SPARTAN-III LCD display and an external ePaper display

(EPD), the same as in the final SNPWD platform. The integrated ePaper Display was the 1.44” EPD panel of Pervasive Displays. To reduce complexity, the processor of the FPGA communicates via UART interface with an external board where the EPD is connected and runs an EPD management firmware.

Finally, the HW emulation framework integrates a software module for the pump speed controller, which forms one of the most critical components of the SNPWD functionality. To maximize flexibility, the decision was made to create a model of a pump, driven by a DC motor and to program another embedded board, external to the FPGA, to simulate the behaviour of the motor in real time [224]. The additional advantage of this choice is that it requires a simple control loop via a PID to be implemented, thus simulating the dynamics of the final control mechanism of SNPWD.

3.2.4 The embedded SW application prototype

The embedded SW application of SNPWD is responsible for the control of all the systems on the device. To further describe the application, the term task will be employed, to group a number of functions related to a specific sub-system of the device. Task grouping enables the implementation of a Finite State Machine of the interactions and priorities of tasks. The supported main tasks are: i) start-up system check & calibration tasks, ii) User Interface tasks, iii) communication tasks, iv) reading sensor data tasks, v) sensor data fusion, vi) pump control tasks.

The logic behind the combination and succession of all the events inside the high level software of SNPWD is implemented by the task management engine, which eventually is the core of the embedded application. The overview of the task management logic is depicted in Fig. 3.3 as a Finite State Machine. The rectangles with round edges correspond to tasks and their interior is described by other FSMs. The pump control interrupt service routing is also depicted in the same format as a separate task. The rest of the interrupt service routines are considered a distinct state, due to their small size and simple internal structure. The events handled by this engine can be categorized in three basic categories:

i Periodic events which take place in predefined intervals in an interrupt driven way, e.g. the main task and the pump control engine. The interrupt driven manner of their execution is highlighted because it ensures that the periodic constraints are met and not violated by asynchronous events.

ii Asynchronous interrupt driven events whose occurrence is unpredictable, e.g. new transmitted sensor data. Care is taken so that interrupt handling

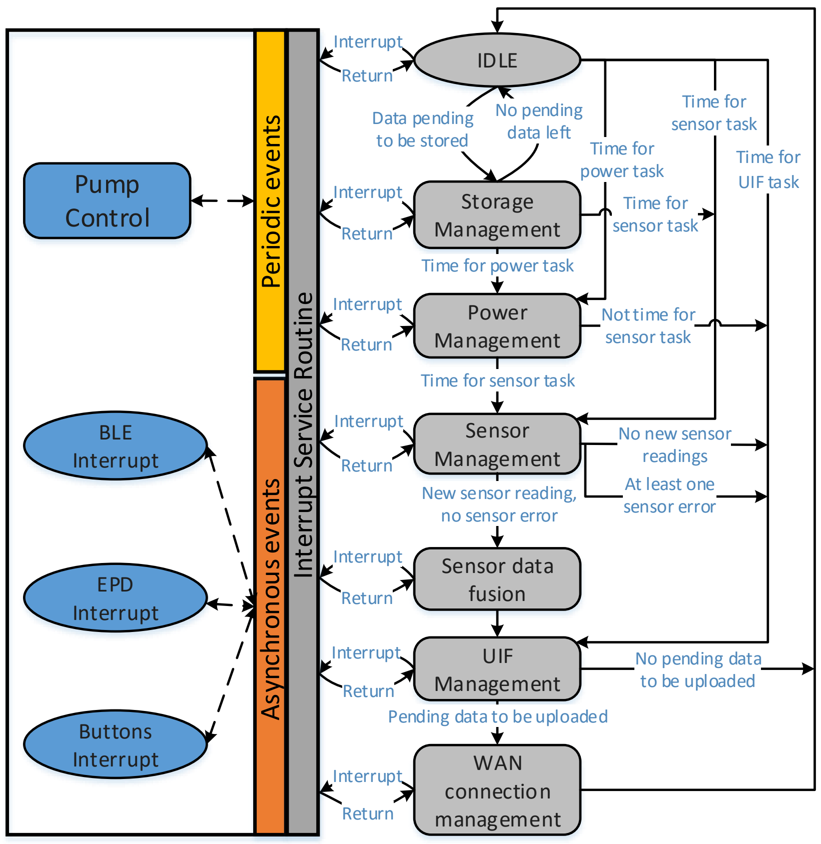


Figure 3.3: The interrupt handling and task management engine.

functions are as least time consuming as possible in order to minimize their effect on the main program execution.

iii Main program tasks, which group functions related to a specific device operation e.g. the UIF management. The succession of these tasks takes places both in periodic and event driven manner. For example, the power management task is set to be executed periodically every second, while the data fusion task only when a new on-wound sensor reading is available.

A sensor data fusion engine is integrated to the SW application for evaluating and combining the data sampled by the various sensors. The engine can generate either alarms related to mechanical malfunctions, or warnings related to the detection of medical related critical situations. The encoded medical conditions that suggest an alarm or warning generation can be found in [223], including the use of machine learning algorithms, i.e. Neural Networks (NN), Support Vector Machines (SVM) [72] and Decision Trees (D. Trees) [17].

3.2.5 Qualitative Evaluation: Functional requirements coverage

To qualitatively evaluate the effectiveness of the simulation approach, we examined the coverage of the functional requirements using the proposed emulation framework. Four attributes are used to define the status of the requirements:

- **Open:** The requirement is to be handled (at least partially) in software. Implementation is missing for first software prototype. Implementation will have to follow for final product.
- **Covered:** The requirement is to be handled (at least partially) in software. Implementation (in software) is finished.
- **Obsolete:** The requirement is not to be handled in software, due to modifications and updates on the system architecture.
- **Irrelevant:** The requirement is not a software requirement and does not require any software implementation. From the operating software point of view, irrelevant requirements are treated as non-existent.

Fig. 3.4 illustrates the coverage of the functional requirements, originally defined in the specifications of the system and achieved by the first prototype of the embedded SW application. As shown, 48% of the SW requirements are completely covered by the first prototype, while another 15% is partially covered. Thus, the overall coverage achieved is about 63%, where the remaining 37% is split among obsolete (4%), irrelevant (4%) and open requirements.

3.2.6 Final SNPWD HW architecture

Fig. 3.5 shows the block diagram representing the SNPWD hardware architecture as designed by Swiss Center for Electronics and Microtechnology (CSEM). The SNPD is based on three microcontrollers:

- An ARM Cortex M3 at 72 MHz, with 100 KB of RAM and 1 MB of Flash Memory as the “main microcontroller” or MAIN, that performs all the tasks requested during “normal mode” of operation.

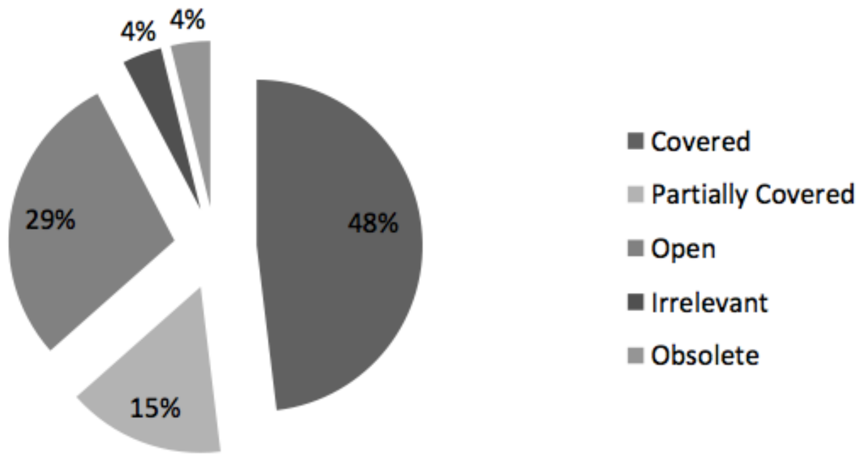


Figure 3.4: Analysis of functional requirements coverage.

- An ARM Cortex M3 at 72 MHz, with 20 KB of RAM and 128 KB of Flash Memory as the "UIF microcontroller" of UIFMC that is responsible for controlling the User Interface of the device.
- An ARM Cortex M3 at 72 MHz, with 20 KB of RAM and 128 KB of Flash Memory as the "supervisory microcontroller" or SUPERVISORY, that is activated only when MAIN stops working for an unforeseen reason.

There are a physical and a logical link between microcontrollers in pairs: the communication link and the supervision link (mutual watchdog). The first link is composed of a full-duplex serial port with hardware flow control and some additional control lines. This link is used to exchange information such as the sensor values or for sending commands, e.g. the main microcontroller requesting a change in the UIF by the UIFMC.

The second link is composed of one output line from the supervisory to each microcontroller and an input from them to the supervisory, used to send a signal at a certain frequency, notifying that each one is alive and working normally (mutual watchdog). The first microcontroller that detects that the other is not working will rise an alarm visible to the user, and if possible upload the alarm status. If MAIN stops working, the SUPERVISORY will raise a local alarm and shut down the device or to put it in a safe mode, where it can stay waiting for external intervention without any danger for the user.

UIFMC drives a full user interface with the following elements: (i) an EPD (Electronic Paper Display or ePaper) with a LED BAR and a light diffuser

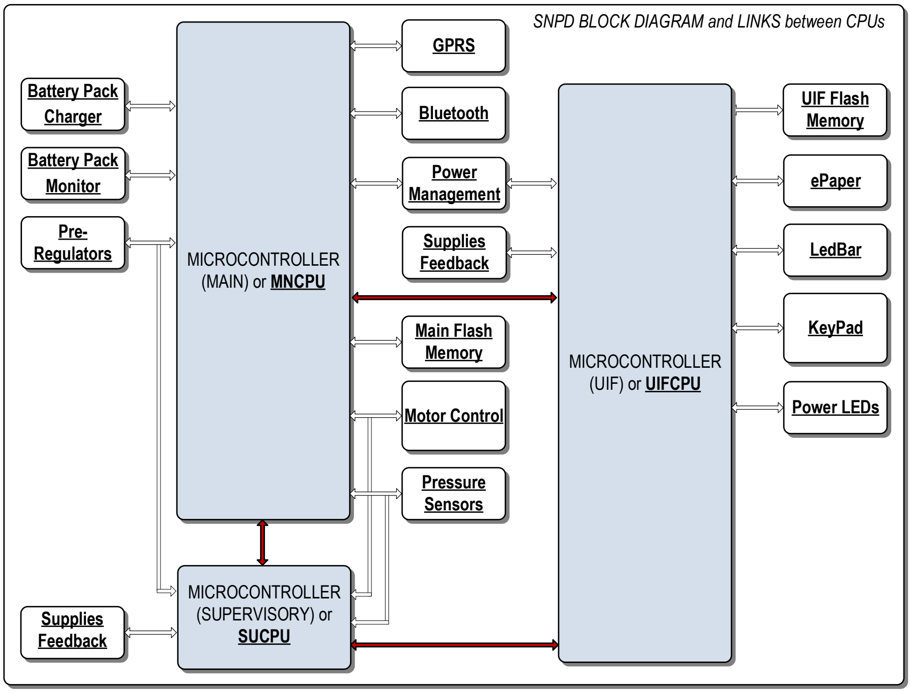


Figure 3.5: Final SNPWD HW architecture.

and (ii) a keypad, a buzzer and a Flash LED, all embedded in a molded front panel together with the EPD window and the LED BAR. MAIN can raise an alarm and provide detailed explanation about the origin of the problem by writing text or drawings symbols in the EPD, being also able to show different levels of messages such as normal information and warnings. To achieve it, an appropriate command is formulated and sent to the UIFMC.

Regarding connectivity, the SNPD implements two wireless communication links: (i) a Dual-Mode Bluetooth module with Bluetooth Classic 2.1 for data communication and Bluetooth 4.0 (BLE) to link the integrated sensors and (ii) a GPRS module that links directly to the Clinical Back-End. The SNPD also implements data storage capability with a NOR serial flash memory of 1Gbit (128Kbytes), which is large enough for the intended application. The UIFMC has also its dedicated flash memory of the same type and capacity in order to store the EPD screens and display them according to the directives of the high level application. The power management componet of the device integrates a rechargeable Battery Pack (BPACK) with a protection circuit with a SOC (State Of Charge) and status monitor, which expose battery capacity information to the software.

MAIN only has to take care of basic motor control tasks, such as enable the speed set point calculation. The control feedback and safety are implemented by means of the following elements: (i) Two pressure (or vacuum) sensors (one analog, one digital) as feedback for the control loop and (ii) a pressure (or vacuum) switch as safety element. The pressure switch controls the enabling of the motor power supply line, its disabling if overpressure (vacuum) is detected, thus stopping the pump without any microcontroller intervention.

3.2.7 The final embedded SW application

The software architecture of the SNPWD follows a layered approach, which defines two coarse layers i) the low levels firmware drivers (LLDs) that provides the API to expose the hardware functionality to the software layer and (ii) the high level embedded software application that is built on top of the LLDs and implements the actual device functionality exposed to the prospective users (patient and healthcare professionals). The software development follow the ISO IEC 62304 “Medical device software – Software life cycle processes” to achieve the goal of making the software safe for medical applications.

The Smart Negative Pressure Wearable Device (SNPWD) is the on-Patient medical device implementing the DFU/VLU negative pressure treatment. The embedded application has been developed in a bare-metal manner on the SNPWD platform and the overall embedded software architecture is depicted in Fig. 3.6. The execution of the application is performed on the Main micro-processor and is responsible for:

- i managing the medical pump to extract exudates from the wound
- ii communicating information, warnings and alarms to the Patient via the available User Interface components
- iii uploading via 3G wireless communication the extracted medical data from the SNPD to the Back-End system
- iv managing the Bluetooth Low Energy connection with in-Wound Sensor device and periodically gather its sampled sensor data
- v managing the storage and recovery of sensor and events data (e.g. alarms) in the on-board non-volatile flash memory
- vi ensuring the safe and reliable operation of the device while checking its available battery capacity

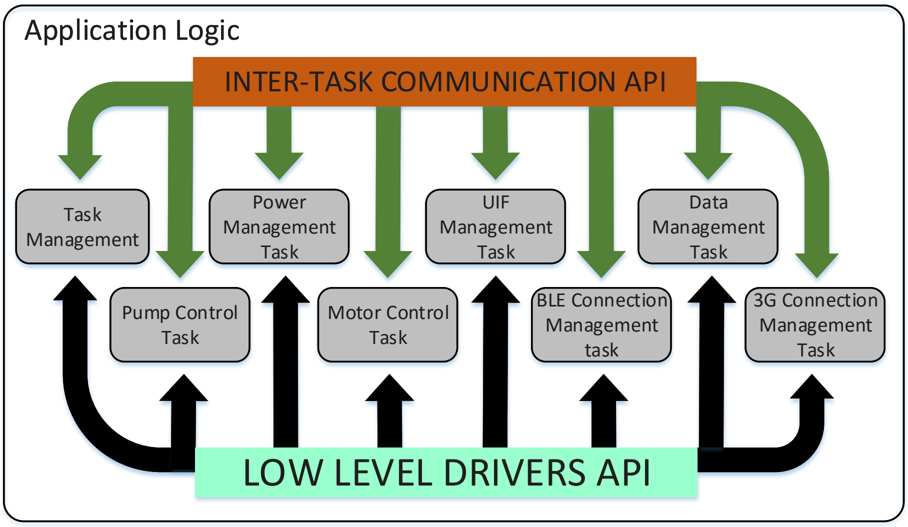


Figure 3.6: Embedded SW architecture and task set organization of the application layer.

The aforementioned functionality is grouped into tasks, scheduled accordingly by a hypervisor task, referred to as tasks management task. Each task is related to a different module of SNPD (e.g. motor control, 3G connection control, etc.) and includes a set of functions and APIs in order (i) to efficient controller the respective module and (ii) to allow the rest of the tasks to interact with it. In overall, to implement an effective mutual exclusion mechanism, the goal is each HW resource/interface to be managed by a limited number of tasks to avoid inter-task collisions. The final set of implemented tasks is the following:

Task Management: Responsible for managing and invoking the rest of the tasks in a round robin fashion, under normal operation. In addition, it triggers the initialization of the system, by executing the initialization functions exposed by the rest of the tasks. Its core functionality is illustrated in Fig. 3.7.

User Interface Management Task: Responsible for handling the User Interface of the device by issuing the appropriate commands to the UIF management CPU. In addition, it exposes the necessary APIs to the rest of the high level tasks in order for them to be able to alter the UIF according to asynchronous events.

3G Connection Management Task: Includes all the functionality related to managing the 2G/3G WWAN interface module of SNPWD. It is responsible for initializing the configuration of the GSM module and performing all the

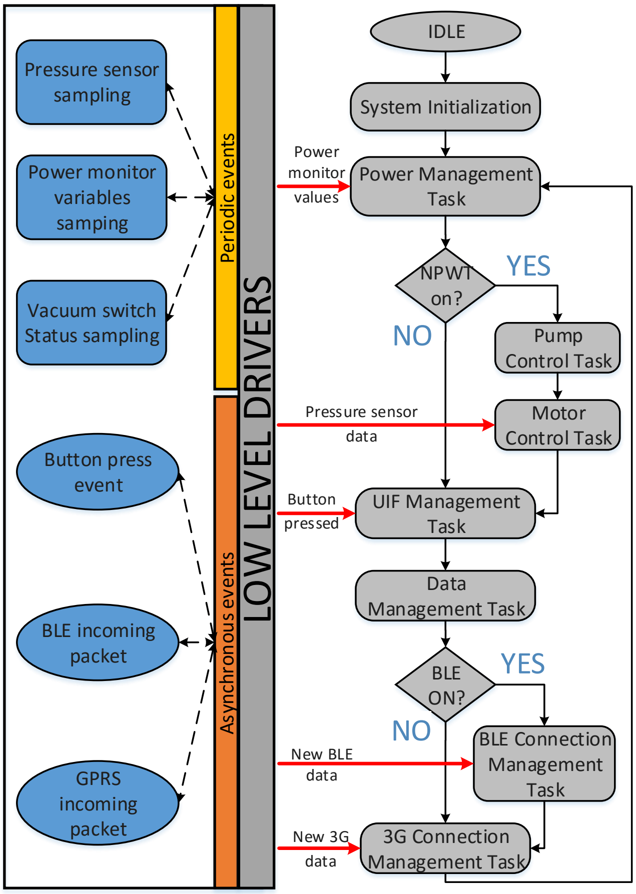


Figure 3.7: The interrupt handling and task management engine.

necessary actions to establish a connection to the back-end server and upload any pending data. The task exports the necessary APIs to the rest of the tasks in order for them to be able to queue up the data to be uploaded. These data include sensor data sampled from the In Wound Sensor Device, average NPWT pressure applied on the wound of the patient as well as any alarms associated with the evolution of the therapy or the operation of the device.

BLE Connection Management Task: Achieves the correct operation of the Bluetooth Low Energy communication module of SNPWD. This module is crucial for achieving communication with the In Wound Sensor Device, where the sensors of the wound are connected on. The BLE connection task is responsible for pairing with one IWSD and sampling it in predefined intervals according to the sensors requirements. The acquired sensor data are transferred to data management task for further handling and then IWSD is commanded to operate in a low power mode in order to save energy.

Motor Control Task: Interacts with the motor of SNPWD, which is the key element of NPWT. The task focuses on assessing the current state of the applied pressure on the wound by sampling the two pressure sensors of the device and then to trigger the motor control algorithm to determine the new operation point of the motor, according to the reference pressure of the therapy.

Pump Control Task: Responsible for the high level supervision of the operation of the pump, which is delivering the NPWT. It interacts with other tasks to determine the type of the applied treatment (continuous versus intermittent). In addition, it identifies pressure related alarms, by correlating the average applied pressure on the wound (exposed by the motor control task) and the duration that the NPWT is enforced.

Power Management Task: Invoked in a periodic way to check and react to the battery levels of the SNPWD. Whenever low battery levels are detected, it invokes the appropriate changes in the UIF, using the API provided by the UIF management task in order to notify the patient that the device must be recharged. In case of critically low battery, this task performs all the necessary actions to ensure a safe device shutdown.

Data Management Task: Performs all the necessary actions for the correct data management inside SNPWD. This includes data forwarding between different tasks (e.g. from BLE connection task to 3G connection management task) as well as data storage in the non-volatile flash memory of SNPWD. Since there is no file system on the device, the task is responsible for organizing both reading and writing to the non-volatile memory. It also maintains a certain sector of non-volatile memory, where system and user related data are stored in order to be restored during next system start-up.

3.2.8 Wearable device SW verification - Clinical validation

Due to the importance of the SNPWD SW for the correct operation of the SNPWD medical device, numerous verification tests were performed both on the device and SW. The tests aimed at validating the correct operation of the SW using a bottom up approach. First individual functions were tested, then individual tasks and eventually combinations of tasks. The tests also involve all the devices of the Wound Management architecture i.e. SNPWD, IWSD and

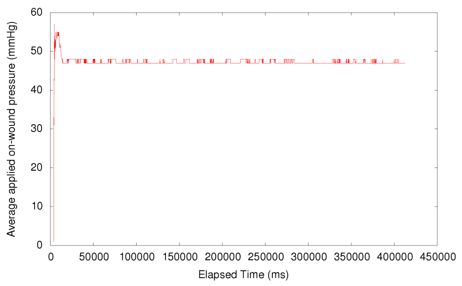
Back-End server. With respect to the embedded SW, the tests were grouped according to the following device functionalities:

- Negative Wound Pressure Treatment application (Pump Operation)
- Alarms generation for Negative Wound Pressure Treatment
- Bluetooth Low Energy Connection and Data transmission
- Local Data Management and non-volative storage
- WAN (GSM) Communication Management and Data uploading to Back-end server
- Power Management
- User Interface Management

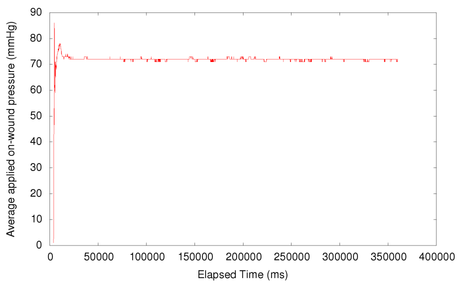
The tests validated the response of the device in timely manner (e.g. pump control and alarm generation), as well as its ability to react to external and internal events (e.g. over-pressure on the wound, battery depletion). In addition, meticulous effort was put on ensuring that data management is correctly performed, taking into account all the respective data production and aggregation points of the Wound management architecture. More precisely, the tests evaluated data generation and correct transmission from IWSD to SNPWD, local non-volatile storage on SNPWD to ensure data availability even if WAN connection is not active, and the integrity of data uploaded to Back-end server in cases of unpredictable GSM connection quality.

Special focus was put on the validation of the correct delivery of Negative Pressure Wound Therapy, which is the critical element of the medical device. The correct delivery of the therapy is achieved if the applied pressure on the wound is maintained constant and within an error margin of the reference point. A monitoring infrastructure was developed in order to acquire and raw data of applied wound pressure by SNPWD. Moreover, different wound models were created in order to simulate in the lab conditions that accurately capture the dynamics of the final setup, i.e. the wound of the patient.

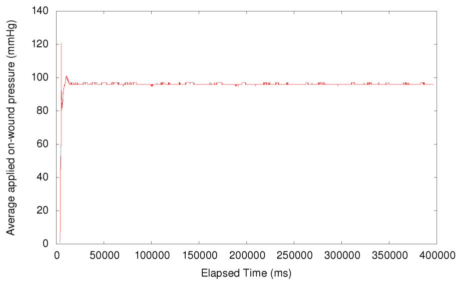
Fig. 3.8, illustrates an indicative example of such tests on a vertical wound model. The X axis corresponds to the duration of the applied NPWT, while the Y axis corresponds to the applied pressure on the wound model. The target reference points are negative 50 mmHg (Fig. 3.8a), 75 mmHg (Fig. 3.8b), 100 mmHg (Fig. 3.8c) and 125 mmHg (Fig. 3.8d). In all cases it is shown that the control task manages to regulate the pressure within an acceptable error margin of up to 5 mmHg, in comparison to the reference pressure. This behavior is an inherit



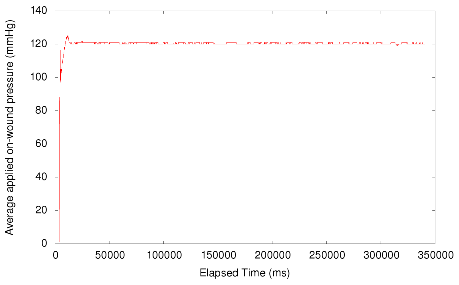
(a) Reference point of negative 50 mmHg.



(b) Reference point of negative 75 mmHg.



(c) Reference point of negative 100 mmHg.



(d) Reference point of negative 125 mmHg.

Figure 3.8: Average applied on-wound pressure for various reference points.

feature of the controller algorithm which is a plain Proportional one. However, this was chosen over a PI controller because it is less computationally intensive and thus less energy consuming. It was also chosen over a PID controller because the latter does not guarantee the stability of the control. The overshoot is also kept within limits and some single spikes are attributed to the proximity of the pump to the pressure sensors.

The aforementioned SW design methodology, combined with in-depth validation in all aspects of HW and SW allowed the deployment and verification a working prototype of a Smart Wearable Wound Management, which through the Swan-Care consortium [216], managed to succeeded at acquiring the necessary safety compliance certifications and the approval of the Italian Ministry of Health in order to conduct a clinical validation study in the Dermatology department of the University of Pisa. The study included 15 patients of VLU and DFU and took place over 5 weeks. The study validated the functionality of Swan-iCare system and provided valuable clinical data regarding the application of medical sensors in the wound caring eco-system.

3.3 Arrhythmia detection via the Electrocardiogram Signal

This Section presents the full design cycle of an Electrocardiogram application for arrhythmia detection on an IoT node. The core functionality of the application is performed via machine learning based classification, which in turn creates a large design space with respect to building and fine-tuning the classification algorithms. To achieve this goal, a complete design space exploration methodology was built, which was utilized in order to produce different classifier models offline and afterwards evaluate them in an actual IoT node.

3.3.1 The Electrocardiogram Signal

The electrocardiogram signal, most commonly referred to as ECG signal, is widely accepted as one of the most fundamental bio-signals for monitoring and assessing the health status of a patient. It is produced by recording the electrical activity of the heart over a period of time using electrodes placed on a patient's body. The electrodes record the sum of the electrical activity resulting from the cardiac muscle contraction in response to the electrical depolarization of the muscle cells.

Fig. 3.9, illustrates a typical example of an ECG waveform. Different parts of this waveform can be distinguished and these are utilized by medical experts to assess the status of the ECG signal. More specifically, the P wave indicates that the atria are contracting, pumping blood into the ventricles, and is usually 0.08 to 0.1 seconds in duration. The QRS complex represents ventricular depolarization and contraction, with duration normally 0.06 to 0.1 seconds. The T wave represents ventricular repolarization and is longer in duration than depolarization. Sometimes a small positive U wave may be following the T wave, which represents the last remnants of ventricular repolarization.

By convention, electrodes are placed on each arm and leg, and six electrodes are placed at defined locations on the chest. These electrode leads are connected to a device that measures potential differences between selected electrodes to produce the characteristic ECG tracings. The limb leads are called leads I, II, III, AVR, AVL and AVF. The chest leads are called V1, V2, V3, V4, V5 and V6. In this study, the ECG signals we examine are modified limb lead II, a bipolar lead parallel to the standard limb lead II, but acquired using electrodes placed on the torso, a requirement for long-term ECG monitoring.

One of the most common heart malfunctions is arrhythmia [154], where the heart beats differently compared to its normal rhythm. While this phenomenon

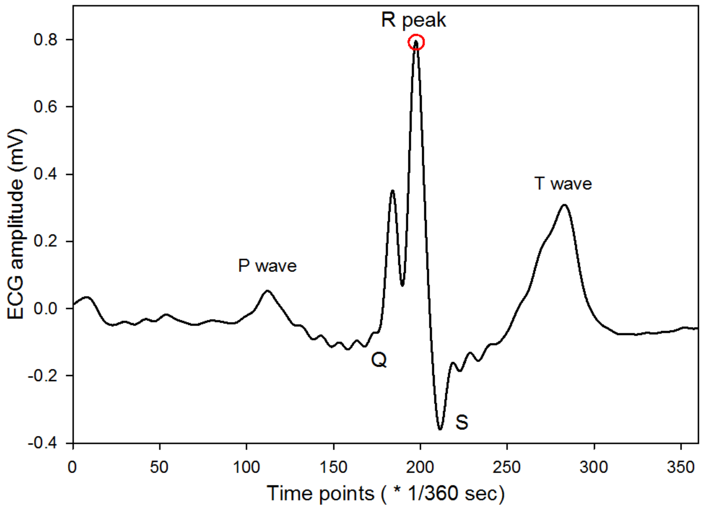


Figure 3.9: Typical waveform of the ECG signal of a heart beat.

is not always critical, it can lead to strokes or heart failure. Given the severity of these cases, arrhythmia and its detection using the ECG signal is a well studied domain. Recently, techniques from the machine learning domain have been adopted [225] for arrhythmia detection and in this work this concept is extended and customized, by designing a machine learning based ECG analysis flow targeting an IoT node.

To efficiently design and employ machine learning techniques, a rich and descriptive input data set of the phenomenon under analysis is required. For the purposes of this study, data from the MIT-BIH arrhythmia database [156] were utilized. The database is composed of 48 half-hour excerpts of two-channel (two leads) ambulatory ECG recordings, obtained from 47 subjects. Of these, twenty three recordings were chosen at random from a collection of over 4000 24-hour ambulatory ECG recordings, serving as a representative sample of routine clinical recordings. The remaining twenty five recordings include a variety of rare, but clinically important phenomena such as complex ventricular, junctional and supraventricular arrhythmias [154]. The subjects include 25 men aged from 32 to 89 years and 22 women aged from 23 to 89 years. Approximately 60% of the subjects were inpatients and 40% outpatients.

A crucial advantage of the MIT-BIH database is that it also provides annotations for each record, where cardiologists have designated a label for each individual

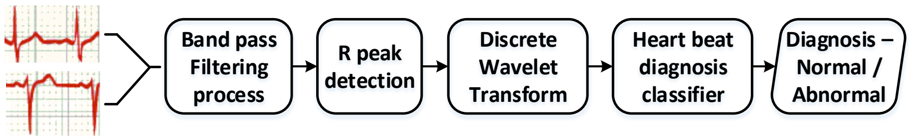

Band pass Filtering process
R peak detection
Discrete Wavelet Transform
Heart beat diagnosis classifier
Diagnosis – Normal / Abnormal

Figure 3.10: Utilized ECG analysis flow.

heart beat included in the record. There are approximately 110.000 annotations. Two arrhythmia groups are examined in this analysis, i.e. 'Normal' (N) and 'Abnormal', which includes all the different types of arrhythmia as they have been indicated by doctors.

3.3.2 Design and Exploration of ECG Analysis Flow

The first essential part of creating an ECG analysis IoT node is defining the exact components of the flow by exploring different design alternatives. This exploration is performed on the algorithm level and is executed offline on devices with high computational capabilities, in order to explore the biggest possible subset of the design space within a reasonable time frame. In this Section the overview of the ECG analysis flow is presented and then the focus is moved to each sub-component and the strategies to best define its structure.

Building blocks of the proposed ECG Analysis flow

Fig. 3.10 illustrates the structure of the proposed analysis steps for heartbeat classification. A digitized ECG signal is applied as the input to the system and the main processing stages are:

- **Noise removal** that filters the signal using a band-pass filter to remove artefacts resulting from patient breathing and movement or noise imported by the power line.
- **R peak detection and heart beat segmentation** that detects a heart beat inside the acquired ECG signal data. If an R peak is detected then a new heart beat has been located and its data are segmented for further processing.
- **Feature extraction process** is imposed on the heart beat in order to extract its characteristics and achieve greater classification performance. For each detected heartbeat, a feature vector is extracted, containing a smaller number of elements than the ECG samples forming the heartbeat.

- **Diagnosis classification**: that concludes whether the heart beat exhibits arrhythmia signs or not. Given the complexity of deriving accurate analytical models in order to assess and predict the status of heart activity, machine learning tools have been established as an appealing and fruitful alternative for ECG signal analysis [225, 254, 157].

In more detail, the **Noise removal** stage filters the power line interference and the baseline wandering, which are significant noise sources that can strongly affect the ECG signal analysis. According to [169], the signals were band-pass filtered at 1-50Hz, using a digital FIR filter implemented in software.

The **R peak detection and heart beat segmentation stage** involves the critical functionality of locating and segmenting a heartbeat, which is the key information under analysis by the implemented IoT application. In order to detect a heart beat, the application detects an R peak inside a predefined input ECG data window. The detection of an R peak is a complex procedure and still remains an active research topic [108], given its significance in ECG analysis. Still, QRS detection, especially detection of R wave in heart signal, is easier than other portions of ECG signal due to its structural form and high amplitude. In the implemented application, we relied on existing algorithms for the detection of an R peak, provided and validated in the software pack from PhysioNet [156]. More precisely, the *wqrs* function[259] implemented in C is applied to the signal, which provides the locations of all QRS complexes found, and each one of them corresponds to the detection of a possible heartbeat.

As suggested in [225], a segmented heartbeat corresponds to a data window of 257 samples, equally distributed around a possible heart beat. This data window is designated according to the QRS onset information provided by the *wqrs* function. The window is adapted to cover the PR and QT intervals (Fig. 3.9), by acquiring 86 samples before the QRS onset, and 170 samples after the QRS onset (257 samples in total). These values are the result on extensive statistical analysis over the ECG waveform [65] and have also been experimentally validated [220].

As a **Feature extraction** mechanism the Discrete Wavelet Transform (DWT) [77] was used, since it has been proven to produce very accurate results. The wavelet base for the DWT is Daubechies of order 2 (db2) [78] and 4 levels of decomposition are performed as proposed in [225]. The DWT is used to compute compressed parameters of the heartbeat data which are called features and characterize the behavior of the heartbeat. The method of using a smaller number of parameters to represent the heartbeat is particularly important for recognition and diagnostic purposes, since the information about the heartbeat is more focused and easily classified by a machine learning based algorithm.

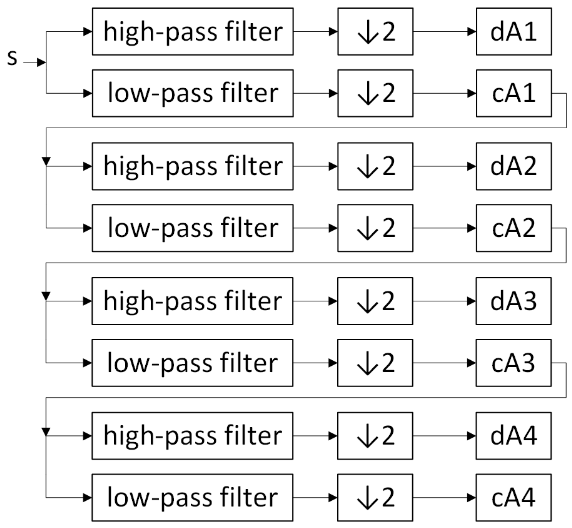


Figure 3.11: Discrete Wavelet Transformation of 4 levels.

The decomposition was performed in a depth of 4 levels, with each one producing a set of approximate and detailed coefficients. Since the heartbeat on which the DWT is applied consists of 257 samples, the number of wavelet coefficients for the first, second, third and fourth level, are respectively 130, 66, 34 and 18. In total, 494 wavelet coefficients are obtained for each heartbeat. However, the final feature vector that serves as input to the classification stage, resulted from a design space exploration on all combinations of the 8 sets of coefficients, thus highly minimizing the number of them that are used for classification.

The actual transformation was implemented in C programming language, according to the functionality of $wavedec()$ function of Matlab. Its calculation is based on consecutive decompositions of the signal using low-pass and high-pass filters, according to the utilized mother wavelet. The coefficient values of these filters were derived from the $wfilters()$ function of Matlab, for Daubechies of order 2 ('db2') mother wavelet. In addition, symmetric-padding (boundary value symmetric replication) was applied to the signal, which is the default discrete wavelet transform extension mode of Matlab. Finally, the convolution of the signal with each filter is implemented to produce the approximation and detail coefficients for each of the 4 levels of decomposition, according to the following process illustrated in Fig. 3.11: Given a signal s of length n, the DWT consists of log_2 n stages at most. The first step produces, starting from s, two sets of coefficients: approximation coefficients CA1, and detail coefficients CD1.

These vectors are obtained by convolving s with the low-pass filter Lo_D for approximation, and with the high-pass filter Hi_D for detail, followed by dyadic decimation (downsampling).

The last stage of the execution pipeline, i.e. **Diagnosis classification**, consists of a binary classifier, which labels each heartbeat as either 'Normal' or 'Abnormal'. In this study, the Support Vector Machine (SVM) classifier [72] was chosen, which has beem shown to exhibit high accuracy in detecting problematic beat patterns [225, 203]. Support vector machines (SVMs) are machine learning algorithms, which analyze data and recognize patterns based on statistical learning theory, supporting non-linear classification with high accuracy and reduced computation cost. The classifier follows the supervised learning principle, i.e. given a set of training samples and their respective labels an SVM training algorithm builds a model that assigns new samples into one of the two classes, resulting in a non-probabilistic binary linear classifier. Extending this notion, in this work an SVM classifier is employed as the means of distinguishing whether a heart beat is exhibiting arrhythmia or not.

3.3.3 Support Vectors Machines based Classifier

SVM classifiers [72] have grown very popular in many machine learning applications and have been extensively used for classification tasks in fields such as text recognition [218, 230], bio-medical applications [96, 172], image processing [217, 228, 255] and more recently activity recognition in mobile devices [22] and deep learning [215]. The popularity of these classifiers is twofold. On the one hand, they exhibit high classification accuracy, even in problems with complex, non-linear distribution of their extracted features space. On the other hand, their structure, based on stencil computation operations, forms a promising candidate for applying acceleration techniques [174, 50].

In a binary classification problem, where input data belong to two classes, a hyperplane can be defined as the geometrical division or separation of the two classes. An SVM is trained to classify an input feature vector of a new observation into one of two classes, which without loss of generality, for the context of this work will be identified as $\{1,-1\}$. The SVM training algorithm will try to deduce the optimal hyperplane, called maximum margin hyperplane that best divides the classes. This is achieved by discovering the hyperplane that has the largest distance from the nearest training-data point of any class, since in general the larger the margin is, the lower the generalization error of the classifier is. The optimal hyperplane is specified by a (normally small) subset of the training data, which defines its position. These points are referred to as the support vectors and hence the name of the classifier.

Without loss of generality, the same principles apply to any finite dimensional space, but it often happens that the training classes are not linearly separable in that space. For this reason, it has been proposed that the original finite-dimensional space is mapped via a kernel function to another space, where the separation by a linear hyperplane is feasible. Commonly chosen kernel functions are non-linear ones, e.g. polynomial function with order greater than one, hyperbolic tangent (tanh) and Gaussian radial basis function (RBF). In this work, Radial Basis Function (RBF) is the chosen kernel function K, as biomedical applications have proved to perform poorly when linear kernels are used, due to the non linear relations of their data. The advantage of the RBF kernel over other non linear kernels is that it has fewer parameters and fewer numerical difficulties. The RBF kernel for two vectors a and b is defined as:

$$K(\mathbf{a},\mathbf{b}) = exp(-\gamma||\mathbf{a}-\mathbf{b}||^2) \tag{3.1}$$

During inference, after having trained an SVM model of N_sv support vectors, the function for classifying an input feature vector $\mathbf{x}$ is of the following form:

$$Class = sgn(\sum_{i=1}^{N\_sv}(y_i * a_i * K(\mathbf{x}, \mathbf{sup\_vector}_i)) - b) \tag{3.2}$$

where K is the kernel function, $\mathbf{sup\_vector}_i$ is the i-th support vector and y_i, a_i are values derived from the training process. Coefficient b is a bias value, also a result of the training process and is constant over all support vectors.

```
const float sv_coef[N_sv];
const float sup_vectors[D_sv][N_sv];

void SVM_predict (float test_vector[D_sv],
 int * y) {

 loop_i:for (i=0; i<N_sv; i++){
  loop_j:for (j=0; j<D_sv; j++){
   diff=test_vector[j]-sup_vectors[j][i];
   norma = norma + diff*diff;
  }

  sum = sum + exp(-gamma*norma)*sv_coef[i];
  norma=0;
 }

 sum = sum - b;
 if (sum<0) *y = -1;
 else *y = 1;
}
```

Listing 3.1: SVM original prediction code

Table 3.2: Declaration of variables in Listing 1.

Variable	Description
N_sv	number of support vectors
D_sv	dimension/features of support vector
sv_coef	array of coefficients for each support vector
$sup\_vectors$	array of support vectors
$test\_vector$	feature vector of heartbeat
$diff$	difference between elements of vectors
$norma$	squared euclidean distance of vectors
$gamma$, b	parameters γ and b of RBF kernel product of y_i, α_i of Eq. 3.2
sum	accumulator for contribution of each support vector
y	pointer to classification result

In addition to its algorithmic properties, the SVM classifier comes with a well-established open source implementation library known as LIBSVM [53], which supports both Matlab and C instances of the classifier. This is important to facilitate an offline Design Space Exploration infrastructure in Matlab, the outcomes of which are directly implementable in the target IoT system. The outcome of the offline DSE is a file of an SVM model, that is used as input in the initialization part of the IoT ECG analysis application. In this way, the implemented application is parametric to the structure of the SVM model.

Listing 3.1, introduces the C-language based implementation of Eq. 3.2, according to LIBSVM. Its input is a new feature vector to be classified and its outcome is one of {-1,1}, i.e. the class label of the input vector. Table 3.2 includes all variables declared in Listing 3.1 accompanied by a short description. The number of support vectors N_sv and the length of the feature vector D_sv in addition to the selected kernel, have great impact on the performance and complexity respectively, of the classifier.

3.3.4 Design space exploration on SVM classifier

Having defined a training data set and a testing data set for evaluation, a Design Space Exploration is performed targeting the different choices of input feature vectors of the ECG signal, in order to designate the best SVM model for arrhythmia detection (different feature sets generate different SVM models).

A high level view of the process to designate the feature vectors combination which gives the model with the best performance is depicted in Fig. 3.12.

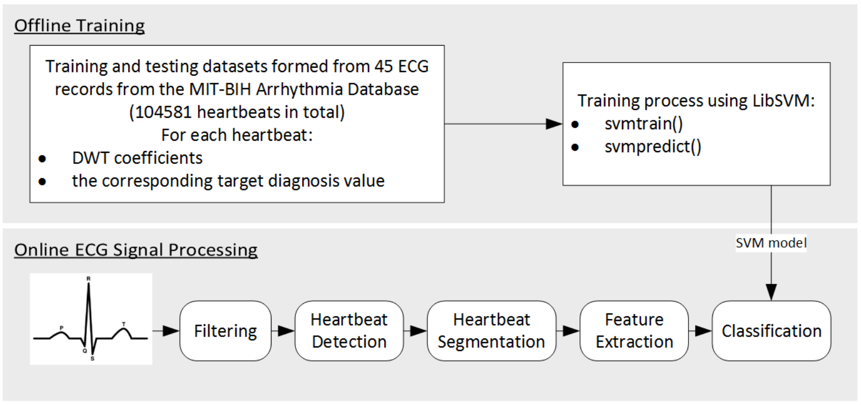

Offline Training
Training and testing datasets formed from 45 ECG records from the MIT-BIH Arrhythmia Database (104581 heartbeats in total)
For each heartbeat:
• DWT coefficients
• the corresponding target diagnosis value
Training process using LibSVM:
• svmtrain()
• svmpredict()
SVM model
Online ECG Signal Processing
R
P
T
Q
S
Filtering
Heartbeat Detection
Heartbeat Segmentation
Feature Extraction
Classification

Figure 3.12: Offline training and online classification.

The training phase is executed offline in a machine with high computational capabilities. Its result, is different models exhibiting trade-offs regarding their classification accuracy and computational requirements. This Section, elaborates on the required steps to set-up and perform the exploration of the design space.

Creating the Input data set for SVM Design Space Exploration

In order for the different SVM models to be produced, an input data set has to be appropriately formulated using the 45 selected ECG records from the MIT-BIH database. This input data wet will be afterwards divided to produce its training and testing sub-parts. As the previous flow indicates, all records were firstly filtered, the R-peaks were located, and then the records were segmented into single heartbeats, forming a set of 104581 heartbeats. As stated before, for each heartbeat a vector of attributes and a target class value is required. The vector of attributes used is the feature vector which contains the DWT coefficients of the heartbeat and is formed at the feature extraction stage. As target value, we use the label given for each heartbeat in the annotation files.

The problem at this point of the analysis, is the mismatch in the detected heartbeats by the heartbeat detection functions provided in the toolkit, and the actual heartbeats annotated by the doctors. A correctly detected R peak, a 'True' beat, correspond to a detection close to an R peak annotation. On the contrary, a faulty detected R peak, a 'False' beat, is one which is far from the corresponding R peak annotation. 'Missed' beat is considered an R peak which the detector failed to detect [221].

In an effort to create a training dataset that abides to the same principles of the implemented application, we overcome this problem by forming a procedure that allows us to match the correctly detected heartbeats with their corresponding labels in the annotation files. The type of detection ('True', 'False', 'Missed') is produced by calculating the distance from an annotated R peak and its comparison to a pre-defined threshold T [221].

This procedure allows us to match the true detected heartbeats with their corresponding labels in the annotation files. As for the falsely detected heartbeats, their are labelled as 'Abnormal'. To validate this choice, a classifier model was trained against only 'True' detected heartbeats and was used to classify 'False' heartbeats. The result was that about 86% of the 'False' heartbeats were indeed classified as Abnormal. The 'Missed' heartbeats were not considered, since they are not detected during the heartbeat detection stage. In total, the above procedure leads to a data set of 104581 heartbeats (100231 'True' heartbeats and 4350 'False' heartbeats), accompanied by a vector of their corresponding class values, and a matrix with the DWT coefficients for each heartbeat. Half of the heartbeats are used as the training data set for the SVM model, and the rest are used as the testing data set.

Design Space Exploration for SVM tuning

A design space exploration is performed over all combinations of the 8 sets of DWT coefficients produced in the feature extraction stage. To reduce exploration time, DWT is calculated only once, producing a vector containing all sets of coefficients for each heartbeat detected in the database. An iteration takes place, choosing a different combination of these sets to serve as input feature vectors, and producing a different classifier model. The goal is to deduce a classifier characterized by:

- maximum classification accuracy $(\frac{Number\ of\ correctly\ classified\ points}{Total\ number\ of\ points})$
- minimum computational cost

At this exploration phase, the computational cost is defined as the product of the number of support vectors and the size of feature vector, since this is the amount of multiplications required for a heartbeat to be classified [72].

Fig. 3.13 presents the results of the design space exploration, regarding accuracy and computational cost. In almost all cases the accuracy is above 97%, with only a few exceptions. The highest accuracy results from the feature vector that only contains the approximate coefficients of the 4th level of DWT decomposition.

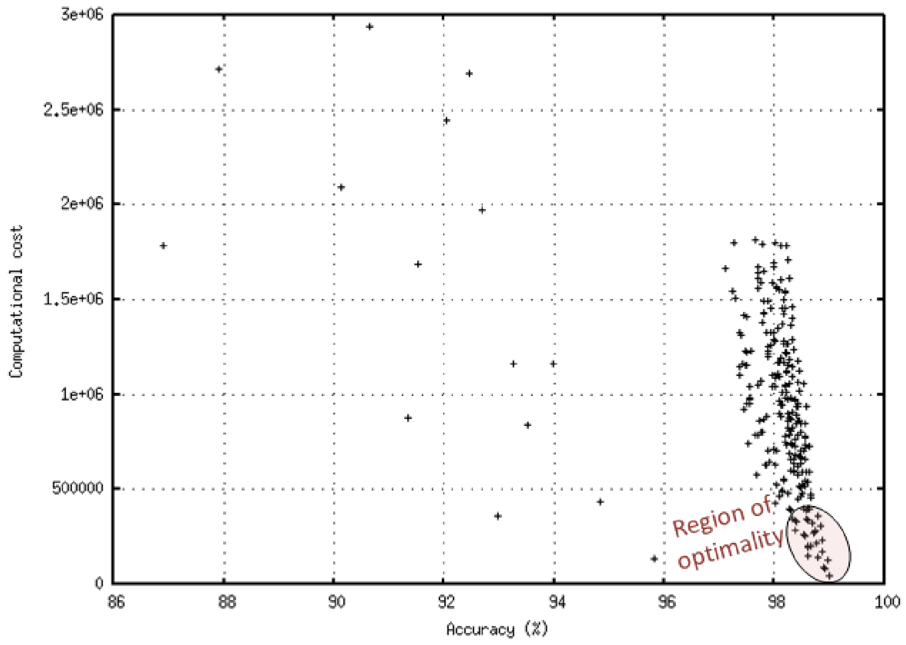


Figure 3.13: Design space exploration of SVM classifier.

Specifically this model exhibits accuracy of 98.9%, having 2493 support vectors of 18 features, each.

3.3.5 ECG Analysis Flow on Embedded IoT Platform

The design and optimization of the different parts of the ECG analysis and arrhythmia detection flow has been performed using high level tools such as Matlab, which inherently provide implementations of the various functional blocks required to perform both building and testing of the various components of the ECG processing flow. The proceeding goal is to port the designed application to a target embedded IoT device, which is much more constrained in terms of available off the self source code implementations and computational resources. This Section summarizes all the necessary decisions and implementation specifics of ECG analysis flow for IoT devices.

The envisaged IoT-based ECG monitoring design must include a sensing part where the ECG signal is captured, using an analog front-end. The signal is then processed in the wearable IoT device. If the sensing part is not mounted on the device, then ECG data can be performed via a short range wireless communication protocol, such as Bluetooth Low Energy, Zigbee, etc. The output of the computing devise is the heartbeat diagnosis, which can be visualized locally via a User Interface and/or be uploaded to a Back-end storage and visualization infrastructure.

In this work, the target IoT device was Intel Galileo board [182]. The Galileo is the first product to feature the Intel Quark SoC X1000, a chip designed for small-core products and low power consumption, and targeted at markets including the Internet of Things and wearable computing. The Quark SoC X1000 is a 32-bit, single core, single-thread, Pentium (P54C/i586) instruction set architecture (ISA)-compatible CPU, operating at speeds up to 400 MHz. The use of the Pentium architecture gives the Galileo the ability to run a fully-fledged Linux kernel. Furthermore, an on-board Ethernet port provides network connectivity, while also the underside of the board includes a mini-PCI Express slot, designed for use with Intel's wireless network cards to add Wi-Fi connectivity to designs.

The complete data flow of the IoT-based ECG monitoring design was ported on Intel Galileo, augmented by data communication support to a Back-end server, which will perform data aggregation and monitoring of the entire IoT based system. In the presented case study, the Galileo Board, which serves as the wearable IoT device, is connected via Ethernet to the network in order to transmit the output data of the application over Internet to the database of the back-end server. The uploaded information consists of the output of the classification stage, which indicates the detection of either a 'Normal' or an 'Abnormal' heartbeat in the ECG signal. The transmission is performed in real-time, after the detection and classification of a new heart beat.

In the software stack of the IoT node, libcurl library was used to perform HTTPS Post requests of the ECG monitoring application to a remote web server. Libcurl is a C-based client-side URL transfer library that supports numerous Internet protocols and SSL certificates, ensuring that data exchanged remains private and integral. Secure data transmission is especially critical in medical IoT monitoring, as the information transmitted is of high clinical significance, while also medical confidentiality must be preserved. The library is also IPv6 compatible which is important for IoT as it simplifies the assignment of a public IP address to an IoT device, thus simplifying peer-to-peer communication.

Evaluation of ECG analysis flow on the IoT node

In this Section, the behaviour and performance of the various components of the ECG analysis flow on the IoT platform are evaluated. The goal is to determine

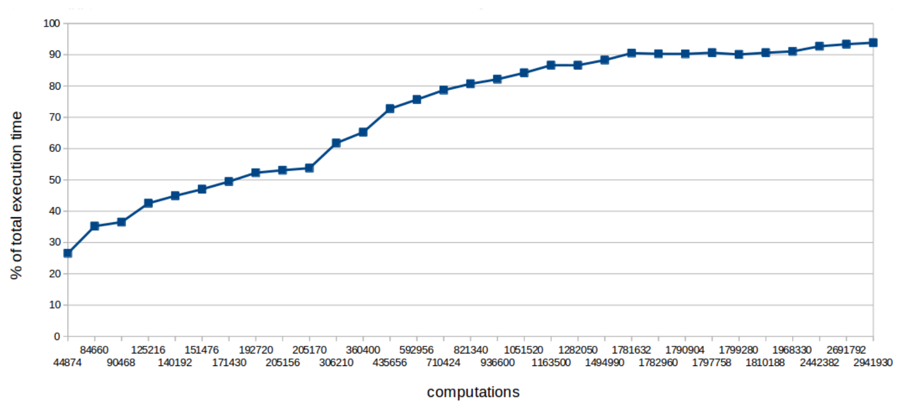


Figure 3.14: Scaling of execution time in accordance to the computations required for each SVM model.

the most time consuming parts of the flow and provide an insight of whether the trend of the computational requirements of the various parts is consistent with the theoretical expectation according to their complexity.

Fig. 3.14 summarizes the execution latency of various SVM models derived from the performed DSE on their accuracy. Using as basic discriminant the predicted computational intensiveness of the model, the 10 lighter, 10 heavier, and 11 configurations in-between were evaluated [220]. The measured values, validate the correlation between the employed computational cost metric and the actual exeuction latency of each SVM model. In all examined cases, the classification accuracy was above 90%, for the optimal cases being above 98.6%.

The same SVM configurations were utilized, to quantify the average execution latency percentage for the processing of a single heartbeat in the complete ECG analysis flow, as illustrated in Fig. 3.15. The examined SVM configuration are the same as In all cases, the inputs of the application were signals derived from MIT-BIH arrhythmia database. For application instances with small SVM models (Fig. 3.15a), the filtering part dominates the execution latency of the flow. Nevertheless, for SVM models of moderate complexity (Fig. 3.15b), the execution of the classifier dominates the required CPU time. This is emphasized in the case of SVM models with high complexity (Fig. 3.15c), where SVM execution requires in average more than 90% of the CPU time needed for processing a single beat.

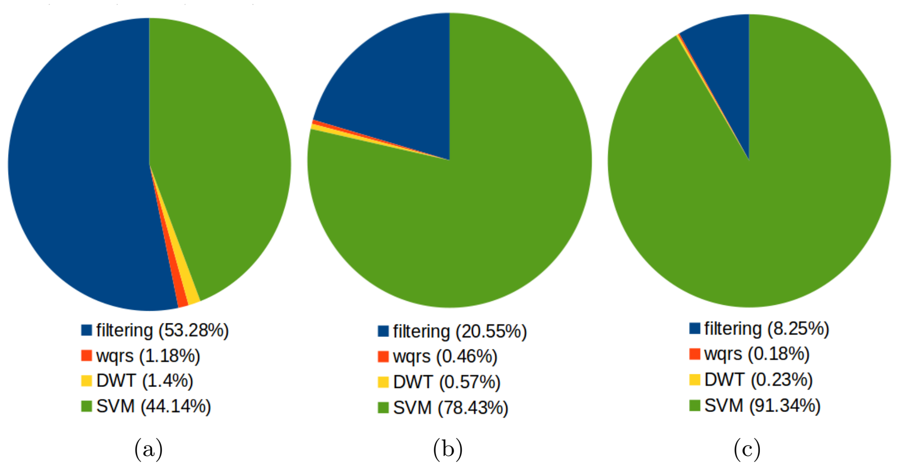


Figure 3.15: Average CPU utilization per heart beat processing (a) SVM models of low computational requirements (b) SVM models of moderate computational requirements, (c) SVM models of high computational requirements.

3.3.6 ECG Analysis Flow for Edge Computing

As presented in Section 3.3.2, the ECG analysis application is structured as a pipeline of processing stages. The outcome of each stage is meaningful as a standalone information, e.g. the a detected or segmented beat, but all stages have to executed in order to achieve the intended application functionality. This property of the flow enables the system to support offloading of processing to an Edge Gateway. All the pipeline stages of the application can be executed both on the IoT node and the Gateway. In this way, processing can be performed up to an arbitrary pipeline stage on the IoT node, then transmit its output to the Gateway, and resume the remaining stages of the pipeline there.

The run-time designation of the stages to be offloaded will be the main tuning knob of the Edge based ECG analysis system. In order to evaluate the different run-time alternatives, the concept of the provided Service Quality to the user is introduced. For the case of ECG analysis application, different Service Quality levels correspond to different input ECG signal sampling frequencies. Higher sampling frequency, results in input signals of higher quality and thus more detailed analysis of the ECG signal, in the cost of the higher computational requirements. The increased signal resolution enhances further analysis and diagnosis by medical experts thus leading to increased SQ for the patient. Five SQ levels were considered for input data rates of 180, 360, 720, 1440 and 2000 Hz [189]. Combinations of SQ levels and offloading levels (i.e. after which pipeline stage computation is offloaded to the Gateway) result in different data rates for input (x_d in Eq. (7.2)) and output (r_d in Eq. (7.3)) of the device. Table 3.3 summarizes these values for combinations of QoS Levels and offloading levels stages for our ECG monitoring prototype.

Table 3.3: Input data rates and transmission data rates for different SQ levels and offloading levels.

			Transmission rate r_d [B/s] for offloading after a certain pipeline stage			
SQ level	Sampling freq. [Hz]	Input data rate x_d [B/s]	Stage 1	Stage 2	Stage 3	Stage 4
1	180	720	720	360	104	1
2	360	1440	1440	720	192	1
3	720	2880	2880	1440	372	1
4	1440	5760	5760	2880	564	1
5	2000	8000	8000	4000	1024	1

Due to the increased sampling frequency of higher QoS levels we observe an increase in input and output data rate of each stage. For example, if our window of data analysis W is 256 points wide at sampling frequency of 360 Hz, then the corresponding window rises to 512 data points at double the sampling frequency. Inevitably, this affects all other pipeline stages given that they operate on greater amount of data. The only exception is the result of the analysis flow (Stage 4), which is always one value that corresponds to the diagnosis label of the processed heart beat.

Stages 1 to 3 of the flow have been designed to operate on a variable-sized input data window while a classifier model (stage 4) was trained for each QoS level. Therefore, there is an instance of the pipeline for each SQ Level, which operates on different amount of data. To comprehend how this fact affects the resources needed for the execution of each combination of SQ level and pipeline stage, we profiled the execution of the flow on the target IoT device (Section 3.3.5). Fig. 3.16 summarizes the percentage of execution of each processing stage over one minute for increasing SQ levels.

A expected, we observe that a higher QoS level comes at the price of increased computational requirements. Fig. 3.16 also shows the computational effort that is offloaded to the Gateway for the different offloading levels, as the breakdown for the computational complexity of all pipeline stages is shown. In all cases, the most computationally intensive stage is the diagnosis part due to its complex structure in an effort to provide accurate predictions. On the contrary, beat segmentation and DWT do not occupy the CPU for prolonged period. The

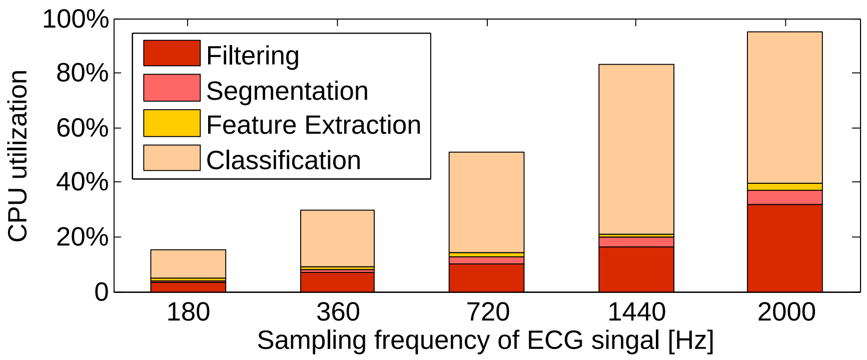


Figure 3.16: CPU utilization of ECG analysis stages.

rest of the time the CPU remains idle, which is the major reason of power consumption variations over different QoS levels.

A key component of the system model is determining the utility functions of each device. The SQ value of each combination of SQ level and ECG processing stage was set proportional to the ratio of the sampling frequency of ECG signal divided by the maximum available ECG sampling frequency (2 kHz), and multiplied by a diminishing factor of $(0.9)^{i-1}$, where i is the SQ level. We also enable the creation of more complex profiles of IoT devices, i.e. by allowing the user to specify a factor of how important high SQ levels are for this device.

3.4 Exploration Framework for Efficient High-Level Synthesis of Support Vector Machines

Future IoT applications are expected to exhibit data-intensive characteristics and inherent requirements for fast and efficient ways to process the acquired data both on site and remotely in order to produce knowledge that will aid the actions of the user. FPGA based acceleration of these tasks is an efficient design alternative for meeting the latency requirements, while minimizing the energy consumption of the system. To aid the accelerator development, the concept of High Level Synthesis (HLS) [97] has been introduced in order to compensate for the complexity of develeopment in Hardware Description Language (HDL).

Despite its high productivity, utilizing HLS in a straightforward manner usually delivers highly sub-optimal design solutions mainly due to the extremely large parameter space that has to be explored. Several researchers have identified these HLS inefficiencies [245, 146, 193, 246], proposing the usage of design space exploration techniques to better guide accelerator synthesis. Following a similar approach, we propose a systematic two-level methodology and prototype framework for producing high-performance HLS designs of hardware SVM accelerators targeting FPGAs. The methodology is based on Vivado-HLS [92], an HLS solution of industrial strength, that enables fast exploration and prototyping of different architectural design decisions.

The eventual goal, is to automate the HW/SW co-design paradigm, and instantiate the ECG analysis flow presented in Section 3.3.2 in a combined CPU-FPGA system. The CPU is occupied with system management and pre-diagnosis ECG processing, while the actual diagnosis is executed on a HW accelerator instantiated on the FPGA fabric. The SVM model used in the exploration process was selected according to the methodology presented in Section 3.3.4 so that it provides the highest classification accuracy, while being adequately small in size in order to have enough FPGA resources to instantiate its accelerator on the target FPGA.

The parameters of the chosen SVM model are stated in Table 3.4. It exhibits over 98% accuracy, sensitivity and specificity. The impact of differing arithmetic precision on how well the final SVM accelerator fares at these metrics, is not examined. Although the proposed methodology is applicable in a straightforward manner in SVM implementations of differing arithmetic accuracy, we consider this decision to be a priori taken by the application designer.

Table 3.4: Utilized SVM model parameters.

parameter	meaning	value
N_sv	number of support vectors	1274
D_sv	dimension/features of each support vector	18

3.4.1 High Level Synthesis

High level synthesis (HLS) [97] tools take an input in a high level description, such as C language and automatically generate effective RTL specification. HLS begins with the compilation of the functional specification, which transforms the input description into a formal model that incorporates the data and control dependencies through a Control and Data Flow Graph (CDFG).

The following steps i.e. allocation, scheduling and binding are the key steps of High Level Synthesis. Allocation defines the type and the number of hardware resources needed. Components can be added during the scheduling or the binding phase. During the scheduling process it is determined which operations will occur at which operation cycles. These operations can take place within one or several clock cycles, they can be chained or executed in parallel. The scheduling phase takes into account design, timing and user defined constraints. Binding is the process that determines which hardware resource implements each scheduled operation. The decisions taken in the binding and allocation process influence the scheduling of the operations, and thus these steps are intertwined rather than running in a serial fashion.

The decisions made in the preceding tasks are applied and generate an RTL model of the synthesized design. The initial C function is synthesized into an IP block which can be integrated into a hardware system. The main advantage of the HLS tools is that they can compile the C code into an implementation of high performance, while maintaining an efficient resource usage. This is accomplished by adding HLS-defined pragmas (directives) that are taken into account during the scheduling and binding process and result in an optimized IP block. HLS provides and optimum implementation based on its own default behavior, the constraints, and the directives that the users specify. Table 3.5 summarizes the HLS directives that have been incorporated in the design exploration described in the following Sections.

These optimization directives are selected so that the architecture created satisfies the desired performance and area goals. The main metrics used to quantify the quality of produced HW design are area, latency and initiation interval. Area is a measure of how many hardware resources are required to implement the design. In particular, HLS computes the utilization of available

Table 3.5: HLS directives [242].

Directive	Description
PIPELINE	Reduces the initiation interval by concurrent execution of operations within a loop or function.
DATAFLOW	Task level pipelining. Functions and loops are executed concurrently. Used to minimize interval.
INLINE	Inlines a function, removing all function hierarchy. Used to enable logic optimization across function boundaries and improve latency/interval by reducing function call overhead.
UNROLL	Unrolls for-loops to create multiple independent operations rather than serial executed ones.
ARRAY_PARTITION	Partitions large arrays into multiple smaller arrays or into individual registers, to improve access to data and remove block-RAM bottlenecks.
ARRAY_MAP	Combines multiple smaller arrays into a single large array to help reduce block-RAM resources.
ARRAY_RESHAPE	Reshape an array from one with many elements to one with greater word-width. Useful for improving block-RAM accesses without using more block-RAM.

resources such as Look Up Tables (LUTs), Registers, block-RAMs and DSPs. The latency of a function is the number of clock cycles required for the function to compute all output values, while the function Initiation Interval (II) is the number of clock cycles before the function can accept new input data.

3.4.2 Design exploration of accelerated SVM classifier

Fig. 3.17 depicts the proposed methodology for optimizing the structure of the High Level Synthesis description of an SVM accelerator in order to enable the HLS tool to synthesize efficient HW IPs. The input of the flow is the C source code of the algorithmic description of SVM classifier. The methodology targets FPGA based programmable System-on-Chip platforms, i.e. Zynq [260], which provides an ARM Cortex-A9 and FPGA fabric.

All levels of accelerator optimization are performed using Vivado-HLS tool [92], which enables user control over the synthesis process through the usage of directives. It performs some optimizations by default and also allows the user to impose directives and constraints of his own choice. The directives available from HLS aim at performance and area optimization and can be applied on functions, loops, arrays and regions containing one or more of the above.

Regarding the proposed methodology, initially the input source code is partitioned to the SVM kernel function and its wrapper logic, which utilizes the kernel function as a part of a complete application. The SVM kernel is then optimized through the proposed two-level design optimization strategy. In parallel, the wrapper logic is tailored to the communication interface of the HW/SW co-designed system.

At the first level of optimization, SVM source code is restructured to expose

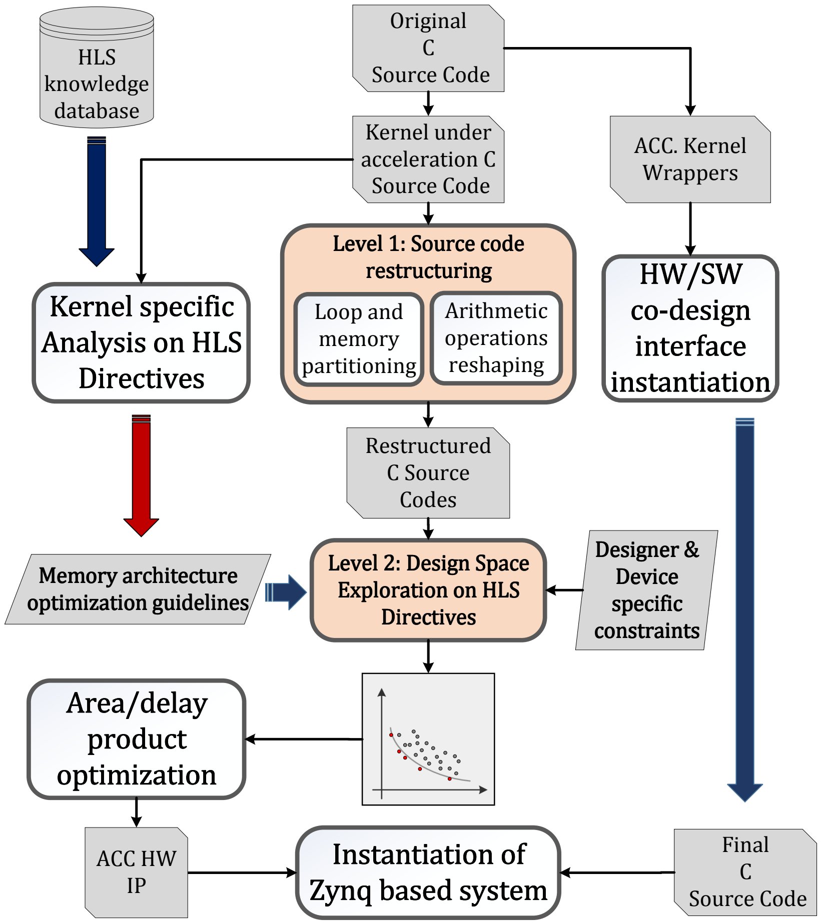

HLS knowledge database
Original C Source Code
Kernel under acceleration C Source Code
ACC. Kernel Wrappers
Kernel specific Analysis on HLS Directives
Level 1: Source code restructuring
Loop and memory partitioning
Arithmetic operations reshaping
HW/SW co-design interface instantiation
Restructured C Source Codes
Memory architecture optimization guidelines
Level 2: Design Space Exploration on HLS Directives
Designer & Device specific constraints
Area/delay product optimization
ACC HW IP
Instantiation of Zynq based system
Final C Source Code

Figure 3.17: Proposed HLS based HW design methodology.

higher data- and instruction-level parallelism than the one exploited by Vivado-HLS. At the second level, the restructured code is enhanced with HLS directives, whose form and position are automatically designated by the proposed framework via a design space exploration under designer and device specific constraints, e.g. maximum latency, FPGA resource utilization etc. The exploration is performed on a very compact design space, which enables fast extraction of high quality design solutions. This space is defined through a set of pruning guidelines derived and validated based on extensive analysis on the impact of different HLS directives on metrics and design objectives of the system. This analysis takes place offline and can be reused as a pre-existing knowledge database that guides the fitting of the pruning guidelines to a specific kernel instance.

3.4.3 Optimization Level 1: Code restructuring for HLS

We emphasize on two code restructuring strategies, which if employed assist the tool to produce much more efficient HW accelerators in terms of enhanced data- and instruction- parallel accelerator implementations.

Data-Level Parallelism in HLS Through Loop and Memory Partitioning

In Listing 3.1, the squared euclidean distance of the *test_vector* and each *sup_vector* is computed and used as input to the SVM kernel function. These values are weighted with a coefficient of the corresponding support vector and then accumulated in one variable to produce the classification result.

The contribution of each support vector to the total sum is irrelevant to the contribution of the others since there are no data dependencies in the computations performed between the test vector and each column of the support vectors array. As a result, these computations can be performed concurrently as illustrated in Fig. 3.18 where the use of different colours indicates that the computations performed between each coloured column and the test vector can be executed in parallel. To facilitate parallel execution, array *sup_vectors* is partitioned into smaller arrays of fewer support vectors. Since the number of attributes is constant, these arrays have the same number of rows but fewer columns and contribute a partial sum to the total, in parallel to the evaluation of the other partial sums.

We implement data-parallel SVM kernels by first modifying the structure of the code and then utilizing HLS directives to enable parallel execution [222]. The part of the code responsible for computing the total sum is implemented as a separate function invoked by the main function as many times as many array

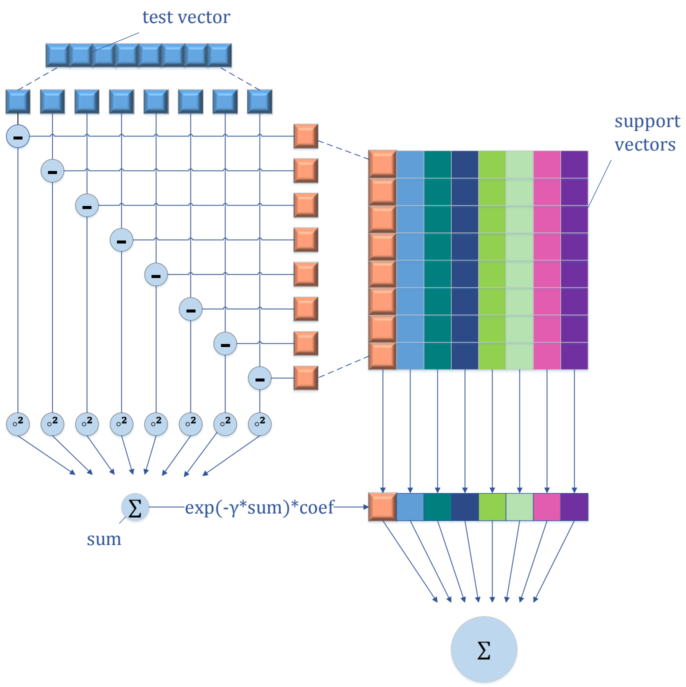


Figure 3.18: Data-level parallelism in SVM.

partitions exist. Each instance of this function processes a different partition of the *sup_vectors* and *sv_coef* arrays to compute its partial sum. Eventually, partial sums of all partitions are aggregated by the main function in order to produce the classification decision.

Concurrent execution of all partitions is hindered, since all functions require access to the same array simultaneously. An array in HLS is implemented using block RAMs, which can have at most two read ports. In order to address the problem of simultaneous memory accesses, the HLS tool would create a replicate of the required arrays for each function instance, thus leading to BRAM utilization burst. To resolve this inefficiency, we apply the *array_partition* Vivado-HLS directive for automatic array partitioning on the modified source code. Using this directive, different read ports for each partition are created and parallelization is possible without any adverse effects. *Array_partition* directive is applied on *sup_vectors* and *sv_coef* arrays. Array *sv_coef* is one-dimensional and thus it is partitioned in partitions of consecutive elements. Array *sup_vectors* is two-dimensional and is partitioned across its column dimension into a varying number of partitions of consecutive columns.

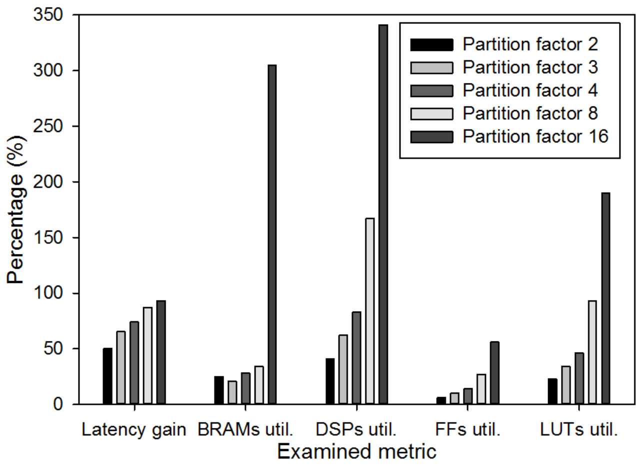


Figure 3.19: Performance and utilization for increasing number of partitions (automatic).

In the main function we invoke the computing function once for each array partition. Its arguments include the pointers to the arrays, a value indicating the size of each partition in columns, an offset to identify the exact location of the partition in the initial array and a pointer to the address of the partial sum to be computed. Then the *dataflow* Vivado-HLS directive is applied on the main function to allow functions within its scope to operate in parallel. HLS automatically detects that function instances can be executed simultaneously and synthesis is performed accordingly.

The described technique is evaluated for array partitioning of factors 2,3,4,8 and 16 and results are depicted in Fig. 3.19. Latency is assessed in terms of gain, in comparison to the latency of the accelerator derived by the original HLS code. Memory and area are presented in utilization percentages over the available resources of the target FPGA.

We observe that as the number of partitions increases, the latency of the design is reduced accordingly. In fact, the speedup of execution time is very close to the ideal speedup value, which is equal to the number of array partitions being used e.g. a speedup of 2× for a partition factor of 2 and close to 16× for a partition factor of 16. There is a clear trade-off between reduced latency and increased FPGA resources utilization since for each instance of the parallel executed function its logic is replicated. This explains the increase in DSP, Flip Flop and LUT utilization. For example, the loop body requires 45 DSP units for all the operations to be scheduled in the original code. For increasing

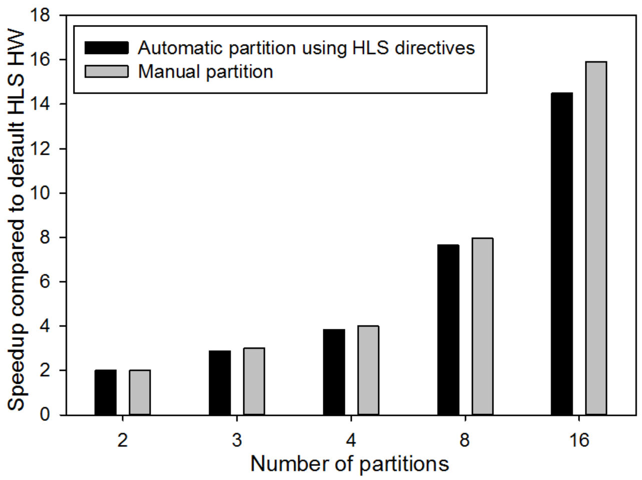


Figure 3.20: Speedup gain comparison (automatic vs manual).

partition factors, DSP utilization remains the same for each function instance but the aggregated utilization is proportional to the number of partitions used. This explains why for large partition factors, the amount of resources of the target FPGA is not sufficient i.e. values of 100% utilization are reported. The same principle applies for the utilization of Flip Flops and LUTs.

An interesting parameter is the utilization of BRAM required for array implementation. Using the partitioning directive, each function gains access to a read port of its partition of the array thus avoiding BRAM replication. This leads to an almost constant total number of used BRAMs in all cases except for the one of partition factor equal to 16. In that case the HLS tool exhibits an inconsistency and replicates of the arrays are created, resulting in an explosion in BRAM usage.

To avoid this behaviour, manual array partitioning is applied by allocating several, smaller arrays in the source code instead of declaring one large array and then partitioning it into smaller ones using the corresponding directive. Now in each function call, a different array pointer is passed pointing to one of the array partitions. As expected, the increase in area resources (DSPs, Flip Flops and LUTs) remains the same, in both cases. This means that parameters like utilization of DSPs still exceed the available FPGA resources. However, this increase follows a consistent trend and can be mitigated by the usage of an FPGA with more resources. On the contrary, the high increase in number of BRAMs when partitioning for a factor of 16 is now eliminated and BRAM utilization remains practically the same regardless of the number of partitions.

The gain in latency of automatic vs manual array partitioning is presented in Fig. 3.20. *We observe that the proposed manual partitioning is more effective than the automatic one, giving a speedup practically equal to the ideal one.*

Instruction Level Parallelism through arithmetic operation reshaping

While data-parallel optimization enables different support vectors to be calculated simultaneously, Instruction-Level Parallelism (ILP) aims at optimizing the performance of computations required for calculating the contribution of one support vector to the classification result. This contribution requires the computation of the squared euclidean distance between the current support vector and the test vector, performed in the inner loop, i.e. *loop_j*. The loop iterates over the elements of the two vectors, computes their difference, multiplies it with itself and accumulates this squared difference to a variable that holds the squared euclidean distance in the end of iterations. Applying the loop unroll directive (See Table 3.5) to this loop should increase ILP since the computations regarding each element of the vectors can be performed independently of each other and thus in parallel.

Automatic unrolling of the inner loop using Vivado-HLS loop unroll directive leads to a reduction of the accelerator latency, as it can be seen in Table 3.6, but the result is not the one anticipated. *Careful inspection of the scheduling reports generated by the tool showed that it does not fully exploit the available parallelism.* The subtractions and multiplications of different elements are not scheduled simultaneously and additions are scheduled in a serial manner even though there is no true data dependency between the added values.

Inspired by several works in computer arithmetic [244, 247, 175], we propose a more efficient implementation of loop unrolling by transforming the structure of the source code into a tree-based computation of the final result. In this way, HLS can schedule the independent calculations in parallel by allocating more hardware resources, if needed. The structure of the tree based data calculations is depicted in Fig. 3.21 for an unroll factor of value 6. The number of loop iterations is reduced to 3. In the internals of the third iteration, it is presented how *input_vector* and *support_vector* are subtracted, squared and finally added in different levels of a tree-like structure. Operations in each level can be executed in parallel and to produce the final result of a full *loop_j* execution, the outcome values of each iteration are accumulated in one variable. Accumulation is also performed in a tree-like way and thus it does not introduce any more latency to the hardware. In the proposed framework, tree-based code restructuring is automatically performed through a custom source-to-source

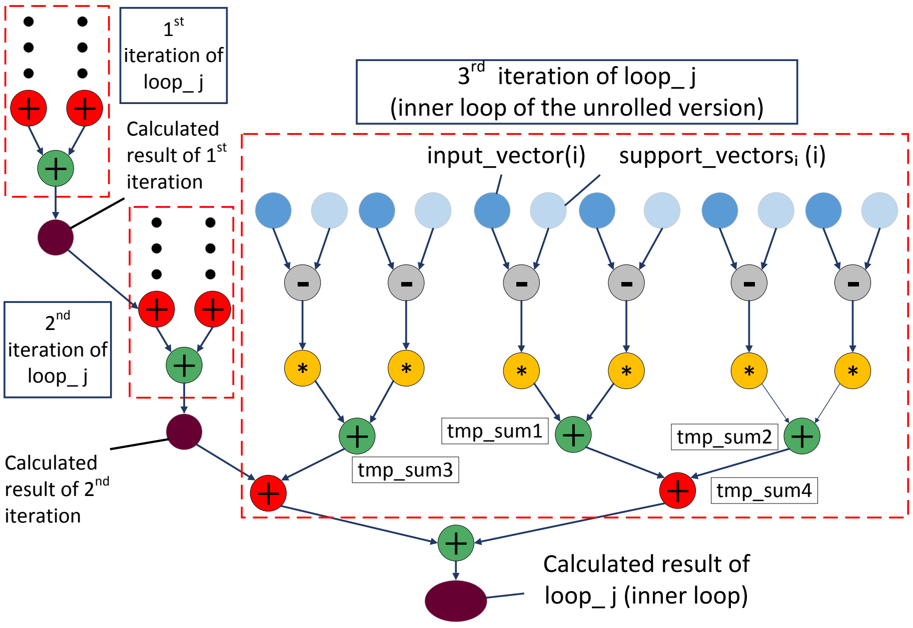

1st iteration of loop_ j
Calculated result of 1st iteration
2nd iteration of loop_ j
Calculated result of 2nd iteration
3rd iteration of loop_ j (inner loop of the unrolled version)
input_vector(i)
support_vectors_i (i)
tmp_sum1
tmp_sum2
tmp_sum3
tmp_sum4
Calculated result of loop_ j (inner loop)

Figure 3.21: Tree based computations for manual unrolling and HLS scheduling.

transformation script that generates the tree reconstructed SVM source code according to the level of the loop unrolling factor.

The proposed technique was examined for unrolling the inner loop of the SVM code 3, 6 and 18 times, respectively. Table 3.6 reports the difference in performance and resources utilization when applying the unroll directive on the inner loop versus manually unrolling it while using a tree-based expression balancing structure for the computations. *Significant improvement in latency gain is observed when applying manual instead of automatic unrolling. The greater the unroll factor is, the higher the performance gap of the two methods is.* The reason is that the tree based structure of performing computations allows data independent operations to be executed in parallel, thus significantly reducing inner loop latency and subsequently total design latency.

The expected trade-off of the proposed technique is increase in resources utilization and specifically in DSPs, FFs and LUTs. The implicit declaration of parallel subtractions and multiplications as well as of the parallel additions of the tree structure results to an increase in the number of computational instances required to achieve ILP. In other words, the HLS tool identifies the available parallelism and allocates more logic to exploit it.

Table 3.6: Evaluated metrics for automatic vs manual unrolling.

Version	Unroll factor	Automatic Unroll using directives					Manual Unroll using tree structure				
		latency (cycles)	BRAM (%)	DSP (%)	FF (%)	LUT (%)	latency (cycles)	BRAM (%)	DSP (%)	FF (%)	LUT (%)
initial	-	412783	24	20	3	11	412783	24	20	3	11
unrolled	3	252259	24	21	3	11	214039	47	26	4	14
unrolled	6	206395	24	23	3	11	149065	70	34	5	18
unrolled	18	173271	27	20	3	11	90461	27	50	8	29

When the loop is unrolled three times, three simultaneous accesses to support vectors array are required but a BRAM can offer at most two read ports. To achieve the reading of three elements, HLS creates a copy of the array and implements both copies using dual port BRAMs to provide four reading ports. As a result, there is a gain in parallelism and latency but BRAM utilization doubles, i.e. 128 allocated BRAMs instead of 64. In the case of unroll factor of 6, three copies of the same array implemented with dual port BRAMs are required, thus BRAM utilization triples. Similarly, when fully unrolling the loop we would expect nine copies of the array to be created in order for 18 elements of the same array to be read concurrently. However, HLS automatically partitions the array in 18 rows, each one being implemented as a dual port BRAM, thus allowing simultaneous access without memory increase. Consequently, HLS behaves in an inefficient and inconsistent manner. To tackle this issue and generate efficient IPs both in terms of performance and resources utilization, array partition directives can be applied as examined in Section 3.4.4.

3.4.4 Optimization Level 2: Design Space Exploration of HLS Directives

This Section focuses on the exploration of HLS knobs exposed as Vivado-HLS directives to enable further SVM accelerator tuning. Analysis of the impact of directives is performed using Vivado, which is an industrial strength HLS tool and also automates the instantiation of a Zynq based HW/SW co-designed system. However, any other HLS tool, which uses annotation of the input source code like LegUp [51] could be incorporated in our framework and derive an analysis similar to the one presented in the following Sections.

In previous Sections, techniques based on structural modifications of the HLS source code were developed to create effective RTL architectures. These formed a first level of optimization for producing efficient RTL. Nevertheless, performance can be further improved by meticulously tuning the available HLS directives.

HLS directives design space pruning guidelines

Although an exhaustive search of the exploration space of directives guarantees the discovery of optimal configurations, it requires extremely long run-times, thus forming an impractical solution. A more targeted exploration disposed of suboptimal design configurations, is highly desired in designing complex accelerator architectures in order to locate optimal configurations in less time.

In this Section, three pruning guidelines are defined and analysed, which by effectively parallelizing the inner loop of the algorithm (loop_j) succeed to exclude suboptimal design points of high execution latency from the search space. The proposed guidelines are based on the observations of extensive impact analysis of HLS directives [222]. They follow the principle of customizing the memory architecture of the accelerator according to its computation and memory access patterns. The only prerequisite for applying them is that the elements, which need to be accessed concurrently each time, maintain their offset in the initial array. The guidelines are summarized as follows:

PG1. *When the inner loop is unrolled by a factor, if the arrays referenced by the loop iterator are partitioned, the partition factor should be equal to the unroll factor.*

PG2. *When the inner loop is unrolled by a factor, if the arrays referenced by the loop iterator are reshaped, the reshape factor should be equal to the unroll factor.*

PG3. *When the inner loop is unrolled by a factor, if the arrays referenced by the loop iterator are both partitioned and reshaped, the product of partition and reshape factor should be equal to the unroll factor.*

The above rules address two main issues that arise when unrolling the inner loop. First, unrolling a loop implies that multiple elements of arrays referenced inside it, need to be manipulated in parallel. Each array is implemented as a BRAM with two read ports at most, thus a memory bottleneck is observed that limits potential performance gain despite the large parallelism potential. Array partitioning leads to memory partitioning into several banks thus allowing simultaneous access to multiple elements of the same array. Array reshaping recombines the elements of array partitions into a single BRAM that has wider data ports, allowing access to multiple elements of the initial array with a single

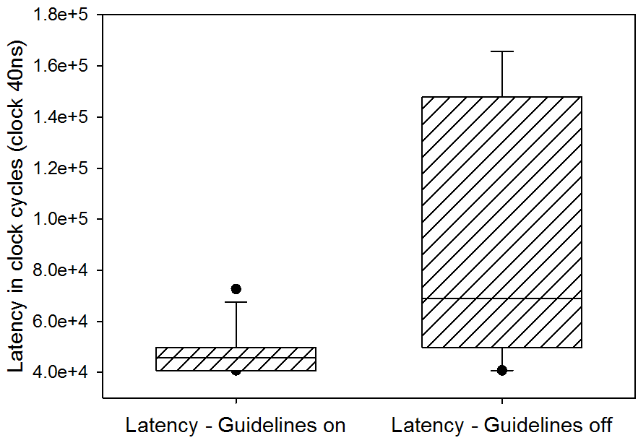


Figure 3.22: Impact of loop unroll directive in loop_j.

read. The combination of the two directives also allows simultaneous access to multiple elements. In total, to fully exploit the inherent parallelism, array restructuring must allow simultaneous access to at least as many elements as are referenced by the loop iterator, i.e. equal to the unroll factor.

On the other hand, partitioning or reshaping an array by a factor greater than required can have adverse effects. Since automatic partitioning and reshaping does not change the source code of the accelerator, array references are still tuned for the initial array. As a consequence the HLS tool compiler adds modulo operations to determine to which array partition the elements referenced in each iteration belong and a significant overhead to the total accelerator latency is introduced. The greater the partition factor is, the greater the overhead gets. Similarly, when reshaping an array, packing more elements than required in a single element of wider word width, introduces overhead to segment the part of the word that is actually required. Therefore, packing all the elements required with no inclusion of redundant elements is more efficient.

Validation of the proposed pruning guidelines

The presented intuitive guidelines need to be validated in a more robust way. Therefore, it is necessary to study the set of all possible configurations created by combining inner loop unrolling with partitioning and/or reshaping the arrays referenced inside the loop. No other directives are included in the exploration, to ensure that results are not distorted.

This set as well as a pruned one, derived by applying the guidelines, are evaluated in terms of latency of their produced accelerators. The left box-plot of Fig. 3.22 (Latency - Guidelines on) contains the configurations included in the pruned set while the right box-plot (Latency - Guidelines off) the excluded ones. The effectiveness of the pruning guidelines is proven by observing that the vast majority of low latency design points are included in the pruned set.

The pruning guidelines can now be applied on the full search space. The resulting pruned space should be evaluated in terms of its efficiency to locate the optimal points. For that purpose, a Pareto analysis is performed on both spaces. This Pareto analysis considers the trade-off between delay and area utilization. We employ the resource utilization of the FPGA, as our area complexity proxy metric, which combines BRAM, DSP, Flip Flop and LUT utilization percentage as shown in 3.3[1].

$$Area_{util} = \frac{BRAM_{util} + DSP_{util} + FF_{util} + LUT_{util}}{400} \tag{3.3}$$

Fig. 3.23a depicts the mapping of the pruned space (black 'x' symbols) on the full space (blue '+' symbols) and Fig. 3.23b includes the Pareto points of the two design spaces. Although full search space exploration is very time consuming[2], it was performed to enable validation of the efficiency of our approach. We do not consider that this full search is practical under realistic design time requirements especially when numerous SVM models need to be optimized.

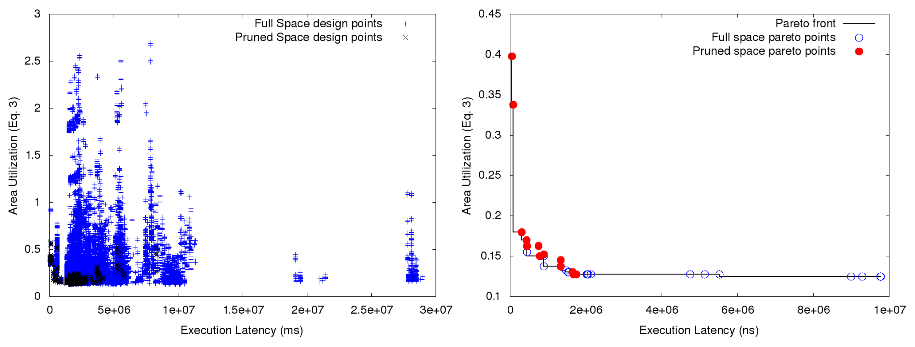
Statistical data are necessary for a fair comparison between the full and pruned spaces. The full design space for the SVM classifier includes 70962 configurations from which 30 distinct Pareto points are identified (Fig. 3.23a). *The pruned space is composed of merely 2212 configurations, i.e. around 3.1% of the initial solution space including 13 Pareto points. 10 of them are included in the pareto points of the full space, resulting in Pareto coverage of 33%. Therefore the proposed pruning guidelines deliver an extremely reduced design space spreading across the delay-area optimality region of SVM accelerator design solutions.*

Delay-Area Product Optimization

As shown in the previous Section, the proposed pruning guidelines result in a significant reduction, around 97.44%, of the original solution space that

[1]The values are provided as percentages in HLS reports. To get their average value, their sum is divided by 400 instead of 4

[2]Approximately 15 days on an Intel Xeon CPU E5-2650 v2@2.6 GHz, 64 GB RAM. All reported exploration latencies refer to the synthesis of RTL implementations of input HLS source codes.



(a) Combined Full and Pruned space. (b) Pareto points of Full and Pruned space.

Figure 3.23: Full and Pruned Design Space.

concentrates on the Pareto optimal region considering the delay-area metrics (Fig.3.23). Although feasible, an exhaustive evaluation based on exhaustive search optimization of the pruned solution space remains quite impractical, requiring around 5 hours of execution time. In addition, the derivation of a Pareto-front as the main output of the exploration procedure enables a more deep analysis on the delay-area trade-offs, however it leaves to the designer the final decision on which SVM design configuration should be selected and implemented, i.e. the more the Pareto points the more difficult the decision.

In order to enable a faster and thus more practical optimization procedure than full search as well as to provide to the designer a single optimized design solution, we implemented a single-objective optimization framework, as the final stage of our methodology, that receives as input the pruned design space, D, and solves the following optimization problem:

$$\min_{x \in \boldsymbol{D}} \left[\ Delay(\mathbf{x}) \times Area_{util}(\mathbf{x}) \ \right] \in R^2 \tag{3.4}$$

subject to:

$$\begin{bmatrix} BRAM_{util}(\mathbf{x}) \\ DSP_{util}(\mathbf{x}) \\ LUT_{util}(\mathbf{x}) \\ FF_{util}(\mathbf{x}) \end{bmatrix} \leq \begin{bmatrix} 100\% \\ 100\% \\ 100\% \\ 100\% \end{bmatrix} \tag{3.5}$$

The optimization goal is to find the configuration vector, $\mathbf{x}$, that minimizes the delay-area product in a solution space. A new possible design point is evaluated in terms of the optimization objective and then filtered according to all inequalities of Eq. 3.5. Given the compact solution space, we employ discrete steepest decent greedy optimizer to solve the aforementioned optimization problem. Given an initial set of randomly selected points from the pruned design space, it starts to greedily move within the design space towards a local minimum following the negative of the gradient.

In total, Fig. 3.24 illustrates and summarizes the three different options provided to a designer utilizing the presented framework given an input design space, which consists of a source code description of the function to be accelerated and a set of HLS directives, which can be applied on this code. In *Option A*, the framework is instructed to perform an exhaustive exploration of all the design space. It is the most time consuming option (15 days for the case study) and its outcome is considered optimal since the input design space is fully explored. In *Option B*, the previously described design space pruning guidelines are enforced on the input design space to produce a reduced design space, which is then exhaustively explored. It is a medium case regarding time consumption (5 hours for the case study) and is able to produce a sufficient number of optimal

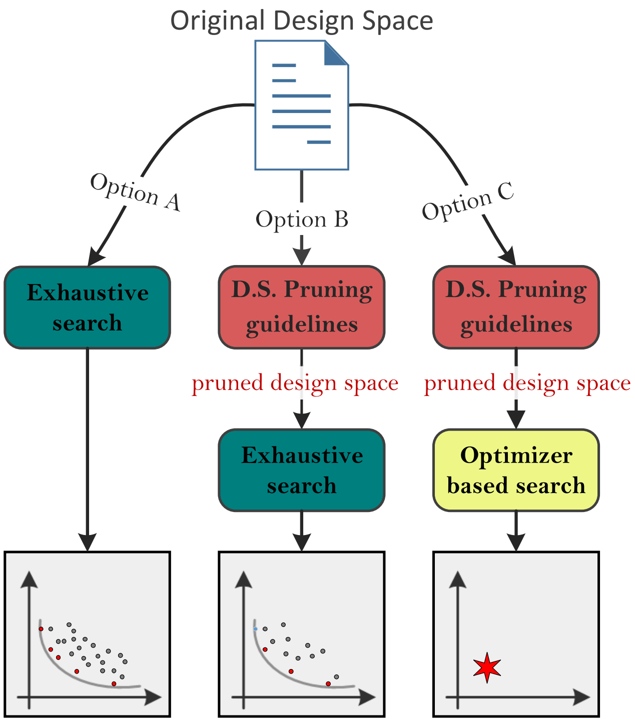


Figure 3.24: DSE options provided by the proposed framework.

design points. Last but not least, in *Option C* the input design space undergoes pruning but instead of an exhaustive traversal of the pruned space, an optimizer is employed providing the designer a single design point within a limited time frame (a few minutes for the case study), which optimizes a single objective.

Last but not least, we note that the proposed framework can support in a transparent manner any other form of optimization objective that best fits the requirements of the designer, e.g. to minimize the execution latency of the derived accelerators while maintaining area utilization less than a specific goal.

3.4.5 Experimental Evaluation

Experimental Setup

The target FPGA board in this work is a Zedboard Zynq Development Kit xc7z020clg484-1[81]. It provides a complete ARM based Processing System (PS) featuring a Dual ARM Cortex-A9 MPCore with integrated memory controllers, floating point operations support and full Linux OS compatibility. The PS side of the board is tightly integrated with the Programmable Logic (PL). The FPGA resources provided by the target board are adequate to support all proposed HW optimizations on the utilized SVM model. This model itself is constrained in its parameters size due to the fact that it has been produced via a structure/accuracy optimization search process [26]. In general, available FPGA resources are a major constraint of the HW optimization process and this has also been reflected in Section 3.4.4. However, providing that resources are available, the proposed framework can maintain efficiency of the derived HW accelerators when SVM model parameters are scaled (Section 3.4.5).

Xilinx Vivado-HLS (v2015.2)[92] has been used to derive all SVM accelerators mentioned in the experimental evaluation. The same tool was also utilized to instantiate the HW/SW co-designed ECG analysis system on the target board. To achieve communication of the PS system with the HW accelerators implemented on PL side, ARM AXI interfaces are used. AXI is part of Advanced Microcontroller Bus Architecture (AMBA). There are different types of AXI interfaces available for the target Zynq board. In this work AXI4 Lite interface is utilized, which is a subset of AXI4 Memory Mapped Interfaces. The Processing System implements the Master Interface of the AXI4 bus while the IP the Slave Interface, which is controlled by the Master through block level signals. The AXI4 Slave Lite Interface is added to the IP that will configure the PL.

The overview of the HW and SW parts of the co-designed system is illustrated in Fig. 3.25. The software is developed in Xilinx SDK 2014.4 and is executed on

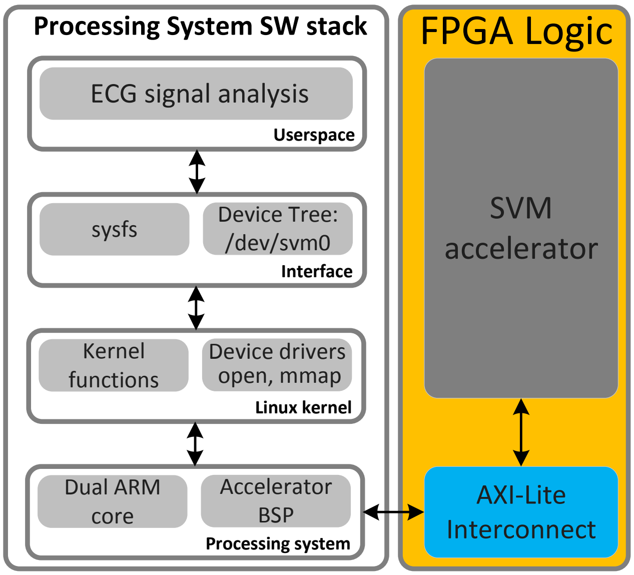


Figure 3.25: Target Zynq based system HW/SW overview.

the ARM Cores of the PS side. The ARM cores run Linux OS developed using Petalinux 2014.4 tool. Petalinux uses hardware configuration files generated by Vivado to build a Linux distribution that includes all drivers necessary for the communication between the PS and PL, including the custom IP. The IP is included in the Linux file system as a userspace I/O device and is mapped to memory via mmap system call. The software application uses this device as a common file and accesses it through a pointer to the corresponding mapped memory range. The accelerator Board Support Package provided by Vivado, encloses the information of the memory mapping of the registers of the IP to the ARM CPU. Using this mapping, the CPU feeds the accelerator IP with data, which in turn returns the classification result to the CPU.

Evaluation of the proposed DSE methodology

In this section, we evaluate the proposed pruning based design exploration strategy for SVM accelerator optimization in comparison to exhaustively traversing the original design space. To quantify the efficiency of the proposed exploration strategy, we make use of two optimization meta-heuristics to search for design points inside the full and the pruned design space. These optimization meta-heuristics are i) steepest descent (Section 3.4.4) and ii) a

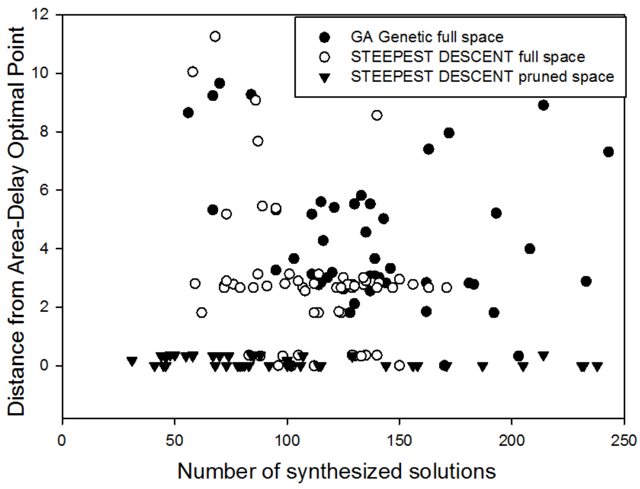


Figure 3.26: Average Distance from Optimal Design for Different Optimizers.

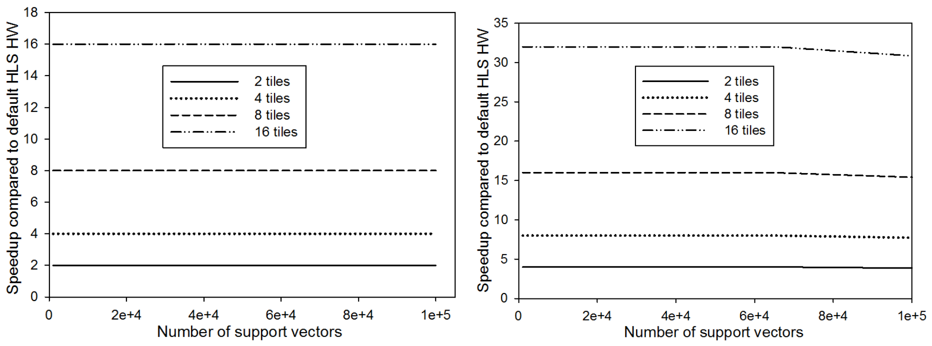
genetic optimizer[3]. All exploration strategies has been executed 50 times to account for the unpredictability imposed by the optimization procedure.

We compare the three exploration alternatives in terms of i) optimality of results through the distance metric with respect to the optimal solutions (the lower the better) derived by the exhaustive design space exploration and (ii) number of synthesized solutions that indicates exploration's run-time efficiency. Fig. 3.26 shows the distance from the optimal delay-area design point, by varying the number of synthesized designs for each exploration strategy.

As shown, the efficiency of each exploration strategy is layered in different ranging zones. It is clear that the zone of the proposed methodology dominates almost completely both optimization meta-heuristics applied on the full design space variants. For the same or smaller number of synthesized configurations, the proposed exploration delivers design solutions that are closer to the optimal SVM designs, delivering an average distance error 0.001 with a standard deviation of 0.14. The respective average distance values for the steepest descent on the full design space has been calculated to 2.83 (standard deviation: 2.37), while for the genetic optimizer to 4 (standard deviation: 2.47).

It is important to stress out that the two-phase exploration strategy presented in Section 3.4.4 and validated here is successful because the optimizer is able to search for design points within a greatly compact space, which includes very

[3]The genetic optimizer has been configured with the following parameters: population size: 20, generations: 4



(a) Speedup of tiling technique.

(b) Speedup of tiling combined with tree reduction of unroll factor 3.

Figure 3.27: Speedup of proposed techniques w.r.t scaling of N_sv.

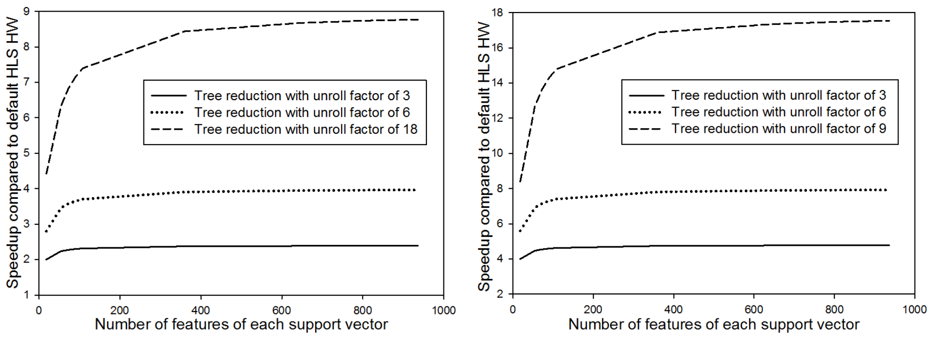
effective solutions. *In other words, it is the combined effect of the pruning guidelines and the optimizer that derive an efficient SVM accelerator.* The discrete steepest descent greedy optimizer was utilized as an example meta-heuristic to convey that even a simple optimizer is able to locate near optimal design points within the compact design space.

Evaluating derived SVM classifier accelerators at scale

Analysis of our proposed SVM classifier HW acceleration techniques has been presented using a specific SVM model. However, it is important to validate that the proposed work-flow retains its efficiency for SVM classifiers with different characteristics in terms of support vectors number and input feature vector size.

The effectiveness of data level parallelization technique (Section 3.4.3) that partitions the support vector array is tested using support vectors that scale from 1000 to 100000 with fixed feature dimension. Fig. 3.27a shows that speedup remains constant and equal to the number of parallelly executing functional blocks. Scaling is also evaluated in the combination of data-level and instruction-level parallelization technique using an unroll factor of 3. The speedup in that case is doubled and remains constant over different number of support vectors (Fig. 3.27b).

Similarly, the effectiveness of the instruction-parallel optimization technique (Section 3.4.3) is tested against support vectors with dimensions scaling from 10 to 1000. It can be seen in Fig. 3.28a that speedup increases until a maximum value is reached and then remains almost constant. The tree reduction technique



(a) Speedup of tree reduction technique. (b) Speedup of tree reduction technique combined with tiling of 2.

Figure 3.28: Gain of proposed techniques w.r.t scaling of D_sv.

is also investigated in combination with data level parallelism of a factor of 2. The speedup trend is maintained but its values are almost doubled (Fig. 3.28b). This leads to the conclusion that there is no interference in the combination of the two techniques in scaled up versions of the SVM classifier.

In total, results indicate that our proposed methodology retains high latency gains when applied to scaled up SVM classifier models taking into account both increased number of support vector machines number and feature vector size. To achieve that, resource utilization is significantly increased to support the computational requirement of the larger model. Inevitably, for high values of the SVM parameters, the utilization exceeds the available resources of the selected target development board. However, this is not a limitation of the proposed methodology but one induced by the target platform, and can be mitigated using a larger FPGA in terms of available resources.

3.4.6 SVM based ECG arrhythmia detection

In this Section, the efficiency of the optimized SVM classifier HW accelerator is evaluated using a HW/SW co-designed version of the arrhythmia detection application presented in Section 3.3.1. A comparison of the diagnosis part of the application is performed, implemented on various target HW platforms with focus on the execution latency of each implementation. The communication latency of providing new input data to the classifier is negligible (ranging from 10x to 900x less than computation time) since its input feature vector consists of only 18 points (Section 3.4). For fairness purposes all systems operate on

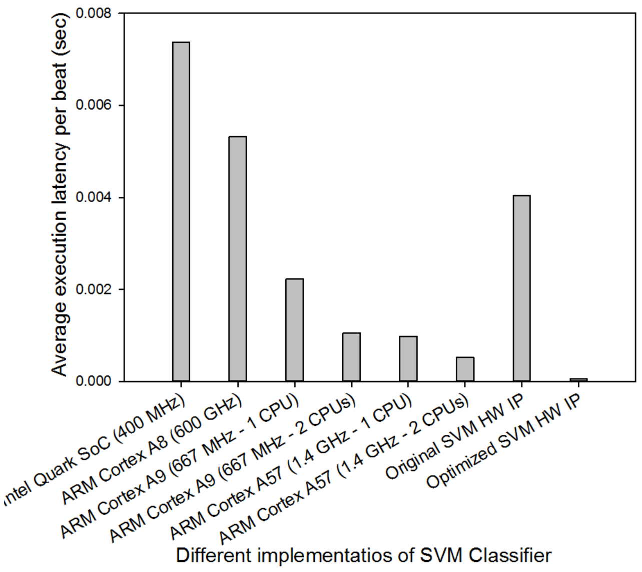


Figure 3.29: Average execution time per beat.

top of a Linux based Operating System and have been compiled using O3 flags of gcc [4]. More specifically the utilized target platforms are:

i Intel Quark SoC[182] operating at 400 MHz.

ii ARM Cortex A8 [67] operating at 600 MHz.

iii ARM Cortex A9 with 2 processing cores (Zynq Processing System) operating at 667 MHz.

iv ARM Cortex A57[243], which is a 64-bit CPU with 2 processing cores operating at 1.4 GHz.

v A Zedboard based HW/SW co-designed system. Its HW IP is derived from Vivado HLS with input SVM source code with no applied structural modifications or optimization directives (Max Clock Frequency at 100 MHz).

vi A Zedboard based HW/SW co-designed system. Its HW IP is the optimal in terms of execution latency derived from the Pareto optimal points provided by the proposed DSE (Max Clock Frequency at 25 MHz).

[4]According to the compiler reports, it was unable to automatically produce vectorized output due to the structure of the input source code.

The testing set is composed of 52291 test vectors, which correspond to feature vectors extracted from heart beats of the MIT-BIH ECG database. The vectors are provided as input to the different SVM classifier implementations and their average execution latency is presented in Fig. 3.29. We observe a correlation between CPU competency and reduced execution latency. In addition, in cases where two processing cores are available, the parallel version of SVM classifier is effective in reducing execution latency. We did not examine the case of more than two workers since the processing cores were fully utilized and thus introducing more working threads could be an overhead. Regarding the HW/SW co-designed systems, the naive implementation of HLS based SVM HW IP is in most cases not efficient even in comparison to CPUs. On the contrary, *the optimized version of the HW IP, is in every case much more efficient compared to all other design alternatives.* The achieved speedup compared to the default Vivado-HLS derived version reaches up to 78×. Interestingly, the expected speedup values derived from the HLS tool reports is 79.81×, which validates that the tool is a trustworthy guide for the designer. The optimized HW IP is also almost 10× faster in comparison to the dual core 64-bit ARM system.

3.5 Conclusions

This Chapter provided an analysis on the methodology and building blocks of contemporary IoT applications, which will be used in the following Chapters as the basic experimental setup for many-core IoT Gateways. It was shown, that through a meticulous design cycle and usage of HW/SW co-design techniques on FPGA based systems, it is feasible to produce realiable working prototypes of complex embedded devices. In addition, the incorporation of machine learning techniques for inference in IoT nodes, can highly increase the sophistication and capabilities of the device, while also providing run-time tuning knobs for the co-existence of multiple IoT nodes in the same deployment.

Last, Systems-on-Chip with combined CPU and FPGA fabric, are able to provide the necessary computational infrastructure in order to comply with strict run-time requirements, even when High Level Synthesis tools are used for the design of HW accelerators. To achieve this, a combined two-level optimization methodology was devised targeting both the re-construction of the original HLS source code as well as the introduction of pruning guidelines for the efficient automated Design Space Exploration of the HLS directives.

Chapter 4

Distributed Run-time Resource Management scheme for NoC based Many-Cores

4.1 Introduction

Run-time resource management in many-core systems can be applied either in a centralized or distributed manner. In traditional centralized approaches [186], a single core analyzes the malleability potential of applications, the available system resources and decides how each application will be served. Even though the centralized approaches are clearer to implement, they often exhibit limited scalability due to bottlenecks in processing and communication functions, especially in environments that require frequent configuration changes. In addition, a centralized system is more susceptible to failures, as errors affecting the central processor can lead to overall system breakdown [35].

On the other hand, distributed management [56, 18] has gained a lot of attention. Research works [21, 130] have shown that distributed systems can perform extremely well, unlock the platform's resources and exploit any underlying topology for optimizing the performance of applications. The basic idea is to assign different roles to cores and build a communication scheme for handling the arrival of new applications. Another important feature provided by distributed systems, is the ability to offer to the applications self-optimization and self-organization functions as in [21, 132].

In this Chapter, a Distributed Run-Time Resource Management *DRTRM* framework is presented, targeting many-core systems with Network-on-Chip infrastructure. Without loss of generality, in the context of Edge computing such platforms can be the heart of an IoT Gateway, since their ample computing power can facilitate the concurrent and more predictable execution of the critical tasks of the Gateway (e.g. communication with IoT nodes or the Cloud) as well as the offloaded tasks by the IoT nodes. The presented framework is intended to oversee the efficient, run-time mapping of the aforementioned tasks to the processing elements of the many-core Gateway. In more detail, the highlights of this Chapter are as follows.

- A Distributed Run-Time Resource Management *DRTRM* framework is presented, based on the idea of local controllers and managers [21], while an on-chip intercommunication scheme ensures decision distribution.
- State-of-the-art work [21] is extended with new algorithms regarding the resource negotiation process and the way that new applications are initialised on the system.
- The specifics of the design of DRTRM are presented by formulating the functionality of the different distributed agents via Finite State Machines, combined with the necessary signals in order to make easier the porting of the framework to multiple platforms.
- The design details of an efficient inter- and intra-core synchronization and communication mechanism are presented.
- The efficiency of DRTRM is evaluated on Intel Single Cloud Chip, using a variety of internal configuration options and scenarios of input applications.

More precisely, Section 4.3 presents the design objectives our proposed resource allocation scheme. Section 4.4 summarizes the employed application model, Section 4.5 elaborates on the various employed distributed agents, Section 4.5.4 summarizes their functionality as Finite State Machines and Section 4.5.5 describes the mechanisms for their communication in a NoC based system. Section 4.6 details the setup for the conducted experiments, while Sections 4.6.5 to 4.6.7 present and discuss the experimental evaluation of DRTRM. Last, Section 4.7 concludes the Chapter.

4.2 Centralized v.s. Distributed RTRM: Motivational Observations

Prior to the description of the philosophy and the internal parts of the proposed DRTRM, we present some observational insights on the main reasons a resource management framework should adopt distributed characteristics. As stated, the trend in many-core platforms development is to incorporate an ever increasing number of computational resources on the same system. Centralized solutions will inherently increase the complexity of managing all the resources in a scalable manner, thus a distributed approach seems to be prominent to tackle the aforementioned complexity.

To quantify this behaviour, we evaluated through simulation the computational effort a centralized RTRM in terms of resource management decisions, required to orchestrate a system of increasing resources. The analysis is based on a developed in-house simulator that uses the same design principles of the developed DRTRM (See Section 4.5.4) but is able to scale to a greater number of available computational resources. We simulate the behaviour of an extended Intel SCC platform (See Section 4.6.2) i.e. the available core are multiples of the original platform, their topology follows the principles of the original platform and communication of different processing elements is achieved via a shared memory interface. More precisely, we model each virtual processing element via a Linux process in order to create a large simulated many-core system. On top of each virtual processing element, there is a supervising instance of RTRM. In fact, apart from the centralized management the only difference in comparison to the deployed DRTRM on Intel SCC is the low-level implementation of the communication and data exchange functions between different PEs. The experiments on the simulator were conducted on a high-end processing system (Intel Xeon CPU E5-2650 v2@2.6 GHz, 64 GB RAM) in order to minimize the context switching overhead in the processes that simulate the processing elements of the target many-core.

One agent is responsible for the overview of the system and running applications negotiate resources at run-time via this agent. The experiments performed on our in-house simulator examine scaled versions of Intel SCC platform. The incoming applications on the system are based on the implemented malleable applications (See Section 4.6.3). In the experiments we maintained a constant ratio of incoming applications and available processing elements, while their arrival rate was identical. In Fig. 4.1,, the X axis represents the number of cores of the simulated platform laid on horizontally, while the respective Y axis declares the number cores laid on the vertically. Consequently, the total number of available processing cores is the product of values in X and Y axis.

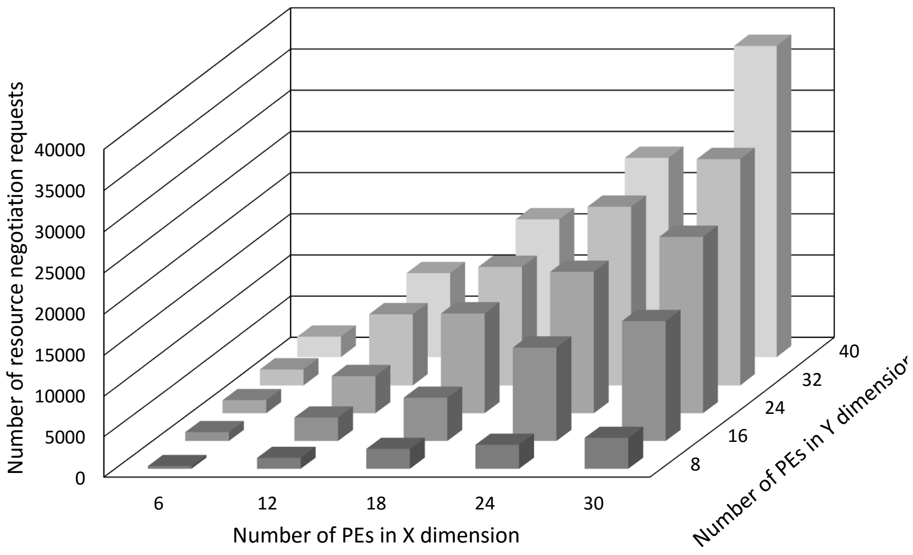


Figure 4.1: Scaling of decisions of RTRM against increasing platform dimensions

The Z axis, corresponds to the total number of evaluations of the resource negotiation algorithm that were required in order for all incoming applications to be instantiated, mapped and executed on the simulated many-core system.

We observe that there is a clear increase in the number of resource management decisions needed as the system size scales up. This means that for appropriate resource management, a single-point centralized management scheme is overburdened as the system size increases and therefore in more stressful scenarios it is likely to fail. Furthermore, the scaling of the requests is not proportional to the scaling of system size, i.e. when originally the system is 6x8 wide it requires 331 instances of resource management decisions, while when system size is 30x40 or 25× larger, it requires about 113× more resource management decisions. Execution latency results are not taken into account, since we consider this metric unreliable in a simulated environment, where distributed agents are affected by the context switching of the system. Nevertheless, given the high number of resource management related decisions that had to be made, we argue that a centralized entity would not be capable of providing adequate results in reasonable amount of time. It is also important to consider than in an actual many-core, the deployed processing elements would be of much less complex architecture and computational competency compared to the utilized CPU for the simulation.

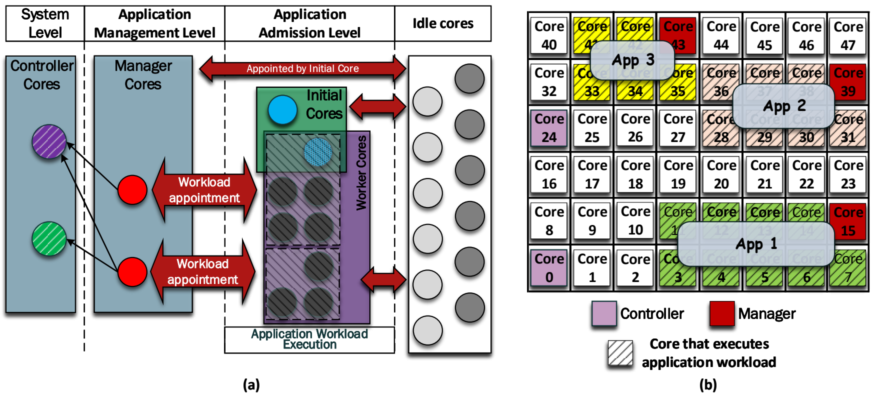


Figure 4.2: (a) Core hierarchy of the proposed framework; (b) Instance of a many-core platform running DRTRM.

4.3 DRTRM Overview

The goal of the proposed framework is to perform run-time resource management in NoC based many-core platforms, while handling parallel applications in a distributed way. The concept of distributed multi-agent systems [130, 20] has gained much popularity compared to centralized management, given the complexity of the resource allocation problem, which is a known NP-hard one [207]. The inherent trade-off of this design choice is the requirement for reduced and efficient communication between the different agents. The framework is designed for parallel applications and especially malleable ones [226], which can dynamically adapt to variations of resources.

The main idea behind *DRTRM* is that the majority of cores alternate between a number of roles throughout their lifetime in order to support one or more managerial or workload executing responsibilities, while maintaining system distribution. This technique has produced significant results as presented in [130, 20, 21, 132]. The proposed approach identifies three core categories: (i) *Initial core*: It is triggered when a new application arrives in the system and it is responsible for determining the cores on which the application will start running on; (ii) *Manager core*: It is responsible for the management of the resources of an application and instructs its resizing whenever it has more or less cores to run on; and (iii) *Controller core*: It is responsible for monitoring and providing activity information about certain regions of the platform.

Our design, implicitly imposes a hierarchy to the roles that each core can be assigned to, as illustrated in Fig. 4.2(a). At the System Level, Controller cores are the building blocks of DRTRM and supervise the functionality of the system. At the Application Management Level, Manager cores register their presence at the Controller cores' regional directory and monitor application execution. Initial cores form the Application Admission Level to facilitate new application instantiation on the system. At the same level of hierarchy, worker cores are responsible for Application Workload Execution. All the above run-time roles are dynamically appointed to cores which belong to the idle cores level. This design, does not represent only a high-level functionality of the distributed agents but also dictates and limits their run-time operations as well as the communication flow. This enforced hierarchy is the key design aspect of DRTRM, in order to incorporate system-wide, run-time policies in large scale many-core systems where the consensus of all PEs is inefficient and thus critical decisions must be the result of the co-operation of a small number of privileged agents.

An overview of the proposed framework and the responsibilities of each core are illustrated in Fig. 4.2(b), in an instance of a 48-core platform. There are two Controller cores supervising the system, which are core 0 and core 24. Each one of them monitors a rectangular area of cores. At this particular instance, three applications are running and there is one Manager core active per application. For the first application (App 1), the Manager is core 15, while for the second (App 2) and third (App 3) the Managers are cores 39 and 43 respectively.

4.4 Application and MPSoC Platform Model

For a given workload, a run-time manager must determine the time quantum on which the applications should run (time-sharing) and which cores they should occupy (space-sharing) by employing an efficient run-time mapping algorithm. Without loss of generality, we assume that an application consists of K tasks that can be executed in parallel. All the possible unique mappings, $map(K, C)$, of K tasks to C cores are $map(K, C) = \frac{\prod_{i=0}^{i<K/C} \binom{K-C\cdot i}{C}}{(\frac{K}{C})!}$. *This vast number of combinations for systems with high number of C, makes exhaustive run-time decision making infeasible and also requires heavy communication between cores in order to find an efficient solution.*

Parallel applications with different capabilities are categorized as [226, 93]: (i) *Moldable*, which can be stopped any time and can be restarted only on the same number of PEs; (ii) *Malleable*, which can be stopped at any time and can be restarted on a different number of PEs and (iii) *Migratable*, that can be stopped at any point of execution and be restarted on PEs in a different site, cluster or domain. In this work, we focus on *malleable applications*, which lead to best system utilization, even though *DRTRM* can support both moldable and migratable ones. Assuming that only one application is running on a many-core system, its speedup $S(n)$ is defined as the time it requires to execute its workload on one core, divided by the corresponding time on n cores. In addition, the remaining execution time is related to the remaining workload W as described in Eq. 4.1 [130].

$$S(n) = \frac{T(1)}{T(n)}; \quad T_{finish} = \frac{W}{S(N)} \tag{4.1}$$

However, the use of Eq. 4.1 is limited as it requires the knowledge of the execution latency of the applications. For that reason, works in distributed run-time mapping [130, 21] have employed the application and speedup model of the common malleable applications described in [84]. Each application is characterized by four parameters W, σ, A and Q, where W is the workload, σ is the parallelism variance, A is the average parallelism and Q is most preferred processing element (PE) type that the application is supposed to be executed on. According to the model presented in [84, 130], the ideal speedup of a parallel application, that is executed alone on a many-core system, is described by Eq. 4.2:

$$\underbrace{S(n) = \begin{cases} \frac{nA}{A+\frac{\sigma}{2(n-1)}} & : 1 \le n < A \\ \frac{nA}{\sigma(A-\frac{1}{2})+n(1-\frac{\sigma}{2})} & : A \le n < 2A-1 \\ A & : n \ge 2A-1 \end{cases}}_{\sigma < 1} \tag{4.2a}$$

$$\underbrace{S(n) = \begin{cases} \frac{nA(\sigma+1)}{\sigma(n+A-1)+A} & : 1 \le n \le A + A\sigma - \sigma \\ A & : n > A + A\sigma - \sigma \end{cases}}_{\sigma \ge 1} \tag{4.2b}$$

The model described by Eqs. 4.1 and 4.2 provides an estimation of the performance and the remaining execution time of a parallel malleable application, given its characteristics.

A many-core platform topology and its communication infrastructure can be uniquely described by a strongly connected directed graph $P(I, C)$. The set of vertices C is composed of two mutually exclusive subsets R_{tot} and $N_{Com}El$ containing the available *PE*s of the platform and its on-chip interconnection elements (*ComEl*), such as routers in an NoC technology. The set of edges I contains the interconnection information (e.g. physical connectivity) for the C set. Each PE of the platform can be of a specific type and differ from the other platform types (heterogeneous platforms) or all PEs can be of the same type and thus have the same functionality (homogeneous platform). In our framework

$T_{pe_i} \forall pe_i \in R_{tot}$ specifies the type of the PE pe_i. In an heterogeneous platform, the T_{pe_i} varies for each PE, while in an homogeneous platform T_{pe_i} is the same for all PEs. In order to comply with the requirements of an application in terms of required PE classes and have the best $Q \to T_{pe_i}$ utilization, we define $Util[pe_i] \in [0,1]$ that implies how appropriately $app(W, \sigma, A, Q)$ is served by the pe_i with type T_{pe_i} [20]. Last, we define the sets F and $offers[]$, which describe all the nodes that can appropriately serve an application based on their type and all the cores offered to the application respectively.

By defining as N_{apps} the total number of instantiated applications at a given moment, the relationship between N_{apps} and R_{tot} is:

$$1 \leq N_{apps} \leq R_{tot} \tag{4.3}$$

N_{apps} is always equal or greater to 1 under the convention that there is an administrative application which is responsible for idle computational resources handling (see Section 4.5.1). If we define as R_i the resources occupied by the i-th application then the following two equations are valid:

$$R_i \neq R_j, \forall i, j, i \neq j, i, j \leq N_{apps} \tag{4.4}$$

$$\sum_{i=1}^{N_{apps}} \aleph_{R_i} = R_{tot} \tag{4.5}$$

implying that that two applications cannot share the same computational resources (Eq. 4.4) and the total computational resources occupied by all applications are equal to the total resources of the system (Eq. 4.5). The maximum PEs that an application can occupy are dictated by the least number of cores that maximize its speedup.

Fig. 4.3 depicts the hardware stack and the system principles under which *DRTRM* was developed. Tiles are connected through a mesh NoC and a router is responsible to forward packets to the correct target. Each CPU tile consists of two cores, that share an L2 cache, while each one has its own private L1 and runs its own operating system. *DRTRM* is deployed as a service on top of the operating system for each core. This design choice allows the architecture of DRTRM to effectively utilize the infrastructure of the operating system without the need of modifying its internals. Consequently, it enhances the modularity and portability of our design, which can be migrated to similar NoC based systems with limited effort.

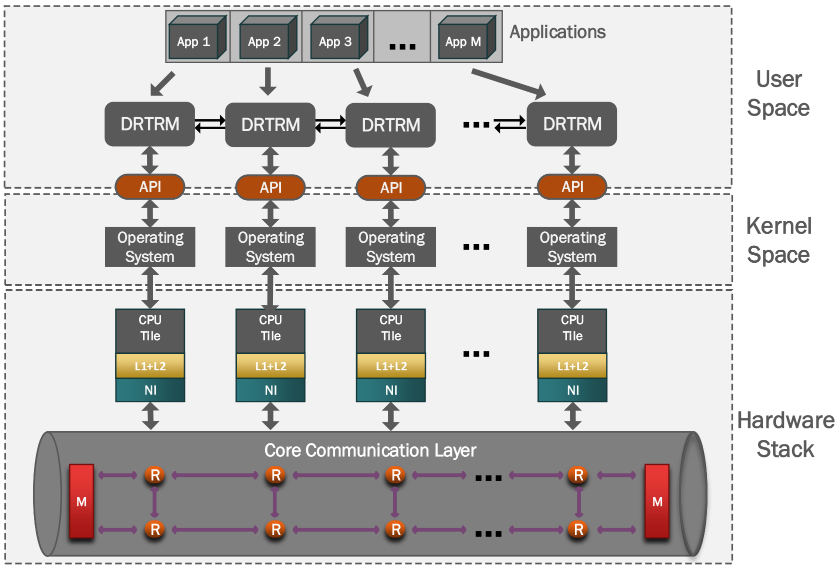


Figure 4.3: Platform and system model for the implementation of DRTRM.

4.5 Core Classification

As aforementioned, the proposed approach identifies three core categories: (i) Controller core; (ii) Initial core and (iii) Manager core, all of which are intended to be executed on general purpose computing units, with no limitations on the target Instruction Set Architecture, i.e. RISC or CISC.

4.5.1 Controller core

The actions of the Controller core are presented in Algorithm 1. Controller cores are the integral component of the proposed approach and are responsible for keeping a record of all PEs inside a specific region of the system, which we will refer to as *Cluster*. Specifically, this record is named *Distributed Directory Service (DDS)* and contains information regarding which PEs are free, which are utilized by an application or which are managing applications. *Clusters* span to non-overlapping areas which are fixed at design-time. The number of Controllers, their position and the topology of their *Clusters* is an important design parameter, which affects workload and traffic distribution on the system, it is defined at the initialization of the platform and cannot be changed at run-

Algorithm 1 Controller core algorithm

```
 1: procedure SIGNAL_HANDLER(sig_no, sender_id)
 2:    if sig_no = SIG_DISC_CNTR_CORES then
 3:       region R ← Read_signal_data () /* Get target region info from signal data */
 4:       cntr_cores_list ← Discover_cntr_func (R)
 5:       Send_reply (sender_id, cntr_cores_list)
 6:    else if sig_no = SIG_REQ_DDS_INFO then
 7:       region R ← Read_signal_data () /* Get target region info from signal data */
 8:       cores_list ← Request_DDS_info_func (R)
 9:       Send_reply (sender_id, cores_list)
10:    else if sig_no = SIG_ADD_CORES_DDS then
11:       cores_list ← Read_signal_data () /* Get the IDs of PEs from signal data */
12:       Add_cores_DDS_func (sender_id, cores_list)
13:    else if sig_no = SIG_REMOVE_CORES_DDS then
14:       cores_list ← Read_signal_data () /* Get the IDs of PEs from signal data */
15:       Rem_cores_DDS_func (sender_id, cores_list)
16:    else if sig_no = SIG_REQ_CORES then
17:       req_app ← Read_signal_data () /* Get the info of req. app from signal data */
18:       my_offer ← Core_offer_func (req_app)
19:       Send_reply (sender_id, my_offer)
20:    else if sig_no = SIG_REP_OFFERS then
21:       offer_reply ← Read_signal_data () /* Get accept/reject value from signal data */
22:       my_cores_list ← Update_my_cores (offer_reply)
23:    end if
24: end procedure
25:
26: procedure CONTROLLER_CORE
27:    while TRUE do
28:       pause () /* Sleep until a new signal is received */
29:    end while
30: end procedure
```

time. Furthermore, DDS provides information about other available Controllers in order to enable communication between different regions.

Algorithm 1 summarizes signal handling in each Controller core. Application activity on the system is provided with respect to a region *R* defined by a center C and a radius r. All cores with Manhattan distance from C less or equal to r, are included in *R*. In order for a core to acquire information about application activity inside the region *R*, it must issue such a request to the responsible Controller cores. A *SIG_DISC_CNTR_CORES* signal provides information about which Controller cores monitor region *R* (lines 2-5). A *SIG_REQ_DDS_INFO* signal informs about the active Manager cores inside region *R* (lines 6-9). *ADD_CORES_DDS* (lines 10-12) and *SIG_REMOVE_CORES_DDS* (lines 13-15) issue a request by a Manager for appropriate DDS list modification. An incoming *SIG_REQ_CORES* signal is an inquiry by a core for more resources which can be provided by the Controller if there are idle cores in its possession (lines 16-19). If an offer is made by the

Controller, it is eventually replied by a *SIG_REP_OFFERS* signal informing whether the offer was accepted or declined (lines 20-22).

The key role of Controller cores, in conjunction to the fact that they serve the requests of all the other run-time agents on the system led us to dedicate a PE per Controller, in order to ensure that they are always active and responsive to input requests. We consider this an acceptable overhead given that the presented design is intended for large scale many-core systems and that the number of Controllers is intended to be small. For instance, a *Cluster* size of 32 PEs would require only 3.125% of dedicated cores in many-core systems with 256, 512 or 1024 PEs.

4.5.2 Initial core

In Algorithm 2, the overview of the functionality of the Initial core is presented. Its operation consists of two basics parts, one dictated by the state of the execution and another one responsible for handling asynchronously received signals from other cores. Its task is completed when a least one available PE has been discovered to facilitate the execution of the new incoming application.

Firstly, a region R is defined where the Initial core will look for available resources. Then, a *SIG_DISC_CNTR_CORES* signal is sent to its Controller to discover which Controllers are monitoring region R. Having sent the signal, it pauses until the reply is received. Upon the reply reception, it proceeds to the next state where a *SIG_REQ_DDS_INFO* signal is sent to all Controller cores responsible for region R. This step will provide the information of which Manager cores possess resources inside R. Another pause phase is initiated which ends when replies from all Controllers have been gathered. Next, a *SIG_REQ_CORES* signal is sent to every Manager and Controller inside region R requesting resource offers for the new application. Then a timer is set and the Initial core pauses again waiting for offers. When the timer expires, if there is at least one not null offer, the new Manager core is instantiated accordingly (lines 37-38). In the opposite case, Algorithm 2 is re-executed after a pre-defined interval (lines 40-41).

The choice of the Initial core has an impact on the efficiency of *DRTRM* as an increased number of Initial cores inside the same region reduces the probability of a timely instantiation of the applications since free resources will be scarce. On the contrary, when they are dispersed on the system, the probability is much higher. In addition, it affects the average distance of messages exchanged on the system and the respective induced communication latency. To designate Initial cores, the scheme presented in [130] follows a simple and straightforward solution. The first set of possible Initial cores is composed of all nodes of

Algorithm 2 Initial core algorithm

```
 1: procedure SIGNAL_HANDLER($sig\_no, sender\_id$)
 2:   if $sig\_no = SIG\_DISC\_CNTR\_CORES\_REPLY$ then
 3:     Cntr_cores_list ← Read_signal_data ()
 4:     State ← REQ_DDS_INFO
 5:   else if $sig\_no = SIG\_REQ\_DDS\_INFO\_REPLY$ then
 6:     Mngr_cores_list ← Read_signal_data ()
 7:     State ← REQ_CORES
 8:   else if $sig\_no = SIG\_REQ\_CORES\_REPLY$ then
 9:     One_core_offer ← Read_signal_data ()
10:     $Offers\_list \leftarrow Offers\_list \cup$ One_core_offer
11:     State ← CHECK_CORE_OFFERS
12:   else if $sig\_no = SIG\_INIT\_TIMER$ then
13:     State ← REQ_DDS_INFO
14:   end if
15: end procedure
16:
17: procedure INITIAL_CORE($init\_app$)
18:   R ← define_search_region ()
19:   State ← DISC_CNTR_CORES
20:
21:   while $State \neq NEW\_MANAGER\_OK$ do
22:     if $State = INIT\_CORE$ then
23:       Send_signal(MY_CNTR_ID, SIG_DISC_CNTR_CORES, R)
24:       pause () /* Will proceed when reply is received */
25:     else if $State = REQ\_DDS\_INFO$ then
26:       for $Cntr\_Core$ in $Cntr\_cores\_list$ do
27:         Send_signal(Cntr_Core, SIG_REQ_DDS_INFO, R)
28:       end for
29:       pause () /* Will proceed when all replies are received */
30:     else if $State = INIT\_CORE\_SEND\_OFFERS$ then
31:       for $Mngr\_Core$ in $Mngr\_cores\_list$ do
32:         Send_signal(Mngr_Core, SIG_REQ_CORES, Init_App)
33:       end for
34:       pause () /* Will proceed when all replies are received */
35:     else if $State = INIT\_CORE\_CHK\_OFFERS$ then
36:       if $Offers\_list \neq \emptyset$ then
37:         Send_signal(New_Mngr_Id, SIG_INIT_MAN, Init_App)
38:         State ← NEW_MANAGER_OK
39:       else
40:         Set_process_repeat_timer ()
41:         pause () /* Will proceed when timer expires */
42:       end if
43:     end if
44:   end while
45: end procedure
```

the system excluding Controllers. A new Initial core is randomly chosen and removed from the set. When the set is empty, then it is renewed as it was originally created. In overall, this policy discovers an Initial core promptly but

Algorithm 3 Max Average Distance designation algorithm

```
function CreateCandidatesSet /* Create the set of candidate Initial cores */
    /* Include all PEs except Controller cores */
    CanditatesSet ← PlatformPEsSet - ControllerCoresSet
    return CanditatesSet
end function

function CreateInitialCoresSet(ApplicationsList IncomingApps)
    /* Initialize the set of candidate Initial cores */
    CanditatesSet ← CreateCandidatesSet()
    InitCoresSet ← ∅ /* Initialize the set of chosen Initial cores */

    /* Designate an Initial core for each new incoming application */
    for each $App \in IncomingApps$ do
        CurAvgMaxDistance ← 0

        /* If the set of Initial cores is NULL, randomly select the first Initial core */
        if $InitCoresSet = \emptyset$ then
            NewInitialCore ← RandomSelection(CanditatesSet)
        else
            for each $CandidateInitCore \in CanditatesSet$ do
                /* Calculate the average distance of a candidate core from all previous designated Initial cores */
                DistancesSet ← CalculateManhattanDist(CandidateInitCore, InitCoresSet)
                CurAvgDistance ← GetAverageValue(DistancesSet)

                /* Is distance from previous Initial cores maximized? */
                if $CurAvgDistance > CurAvgMaxDistance$ then
                    /* If yes, save candidate Initial core */
                    CurAvgMaxDistance ← CurAvgDistance
                    NewInitialCore ← CandidateInitCore
                end if
            end for
        end if

        /* All candidate cores have been examined. The saved one is the new Initial core */
        /* Add chosen core to the set of designated Initial cores */
        InitCoresSet ← InitCoresSet + NewInitialCore
        /* Remove chosen core from the set of candidate cores */
        CanditatesSet ← CanditatesSet - NewInitialCore
        if $CanditatesSet = \emptyset$ then /* Re-initialize candidate set if empty */
            CanditatesSet ← CreateCandidatesSet()
        end if
    end for

    return InitCoresSet
end function
```

it increases the probability of consecutive ones being active in the same region. A second strategy presented in [21] focuses on distributing Initial cores in different parts of the system. In this case, there are more than one candidate sets, one per monitored *Cluster*. The next set to be randomly sampled is chosen in a round robin way and the newly chosen Initial core is removed from its set. When one of the candidate sets is empty, it is re-initialized. In overall, this strategy aims at reducing the probability of increased density of consecutive Initial cores in the same region by distributing their locations in different areas, which are usually disjoint. Nevertheless, the strategy relies on an appropriate *Cluster* topology to achieve that. If the size and topology of *Clusters* is not carefully chosen, the effectiveness of this technique is reduced, due to the higher probability of Initial cores being mapped in close proximity.

DRTRM implements a policy named *Max Average Distance*, which ensures that a newly appointed Initial core will maximize its distance from the previous ones. The calculation of a new Initial core is performed at the node that handles the queue of incoming applications, which is typically the first Controller core. *Max Average Distance* is presented in Algorithm 3. At the beginning, the *CanditatesSet* contains all the nodes that can act as Initials (lines 6-9). For each incoming application, *DRTRM* has to designate an Initial core. For the very first application, a random core is chosen (lines 12-12). In the following cases, having defined the Initial core I_n of the n-th application, the candidate Initial cores for the $(n+1)$-th application must (i) belong to the available, not already sampled candidate cores set and (ii) maximize the average Manhattan distance from all previous Initial cores (lines 15-22). After the selection, the *CanditatesSet* is updated (lines 25-26) and if empty, it is re-initialized (lines 27-29). In total, *Max Average Distance* combines the distribution of Initial cores among different areas while maximizing the distance between them, thus increasing the probability of discovering free resources.

4.5.3 Manager core

The actions of a Manager core are presented in Algorithm 4. One dedicated Manager core is assigned to every application during its initialization, responsible for orchestrating and monitoring its execution. Particularly, the Manager core is responsible for two tasks: (i) Serve resource requests from other distributed agents (lines 2-6); and (ii) check the status, orchestrate workload distribution and launch the self-optimization process of the application (lines 7-15). Both actions are related to resource bargaining on the system aiming at optimal application mapping. Regarding the first task, when an Initial core starts the procedure of locating PEs for a new application or when another Manager core starts its self-optimization process, Managers send their offers to the requesting agent, based on the current speedup value of their application.

Algorithm 4 Manager core algorithm

```
1: procedure MANAGER_CORE
2:    if sig_rquest_cores then    /* A distributed agent requests core offers */
3:       if $Cost > 0$ then    /* Eq. 4.6 */
4:          $offers[] = offers[] + new\_offer$
5:       end if
6:    end if
7:    timer(SELF_OPT)
8:    while app.status <> TERMINATED do
9:       if (timer(SELF_OPT) == 0) && ($app.remaining\_time > SELF\_OPT$) then
   /* Start self-optimization */
10:         sig_request_cores{$C_{r_i}$}    /* Algorithm 1 is triggered in respective agents */
11:         $offers[] \leftarrow$ receive_offers($C_{r_i}$) /* Store incoming offers for cores */
12:         calculate($Cost$)
13:         update_application()
14:         distribute_workload()
15:      end if
16:   end while
17: end procedure
```

The design choice of dedicating one Manager per application aims at providing the application designer the flexibility to designate the preferable resource management policy, customized for the characteristics of each application [68], while ensuring that this policy will be enforced deprived of any latency introduced due to the management of more than one applications, or the co-scheduling of the Manager core with other heavyweight tasks, e.g. application workload execution. In the context of highly scaling applications, we consider this an acceptable overhead. For example, for an application with 16 worker PEs only 6.25% of them are dedicated for management purposes. Furthermore, the Manager core duties are allowed to co-exist with lightweight tasks such as Initial core duties, as will be thoroughly presented in Section 4.5.4.

The optimization objective in existing system performance oriented designs is the maximization of the total application speedup on the system [130, 21]. However, in the presented work, *DRTRM* focuses on the minimization of total application execution latency by taking into account the estimated remaining time of the involved applications, when evaluating a possible resource trade. In *DRTRM*, the Manager first checks if it can offer a core to a requesting application A, without losing more in terms of application speedup compared to the gain that the requesting application will exhibit by the addition of the offered resources (Algorithm 4, lines 3-6), as summarized in Eq. 4.6. Formally, this trade can be perceived as a Kaldor-Hicks improvement [66] in an effort to reach a global optimum system state. Since resource negotiation is greedy, the source offers cores to the destination as long as the gain of the destination is

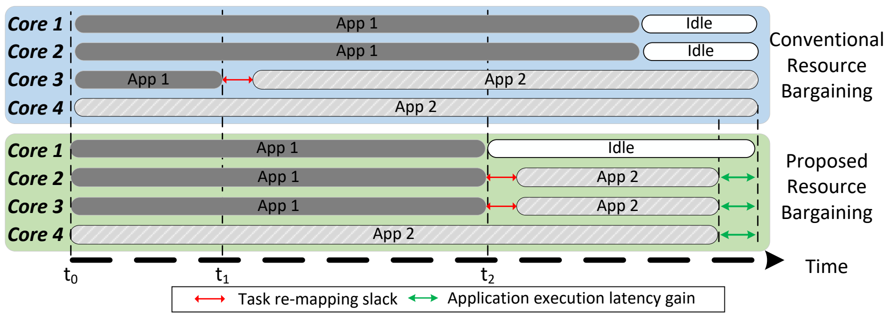

Core 1
Core 2
Core 3
Core 4
App 1
Idle
App 2
Conventional Resource Bargaining
Proposed Resource Bargaining
t0
t1
t2
Time
Task re-mapping slack
Application execution latency gain

Figure 4.4: Resource negotiation example (Comparison of conventional vs proposed resource bargaining).

greater than the loss of the source.

$$Cost = gain[S'_A(n_{old}) \rightarrow S'_A(n_{new})] - loss[S'_B(n_{old}) \rightarrow S'_B(n_{new})] \qquad (4.6)$$

We refer to this greedy resource negotiation approach as conventional resource bargaining, which suffers from the drawback that it does not take into account the remaining execution time of the application offering resources, as well as the required latency for the re-mapping of resources. The proposed framework incorporates a resource bargaining mechanism where all the aforementioned parameters are co-evaluated. More precisely, a resource exchange is performed if it leads to speedup gain but is also meaningful taking into consideration the estimated time for the conclusion of the offering application.

The rationale behind this objective is summarized in Fig. 4.4, where application execution is presented through the time-line of the utilization of 4 cores on a many-core system. At t_0, App. 2 (Core 4) requests more resources from App. 1 (Cores 1-3). The possible exchange is evaluated and validated since the combined speedup of both applications will be higher if Core 3 is traded to App. 2 at t_1. Nevertheless, this move does not take into account the remaining execution time of App. 1 when the trade is validated. At t_0, when the request of App. 2 is evaluated, the proposed algorithm indicates that given the little remaining workload of App. 1, its execution will be over soon and the trade is an overhead. This evaluation takes also into account the required latency for the Core migration to App. 2 and its re-mapping (red arrow). This choice, leads to a subsequent quick finish of App. 1 and eventually more resources are freed for App. 2. In this way at t_2, App. 2 gets 2 free Cores and finishes earlier (green arrow).

The second main task of the Manager core, mentioned as self-optimization,

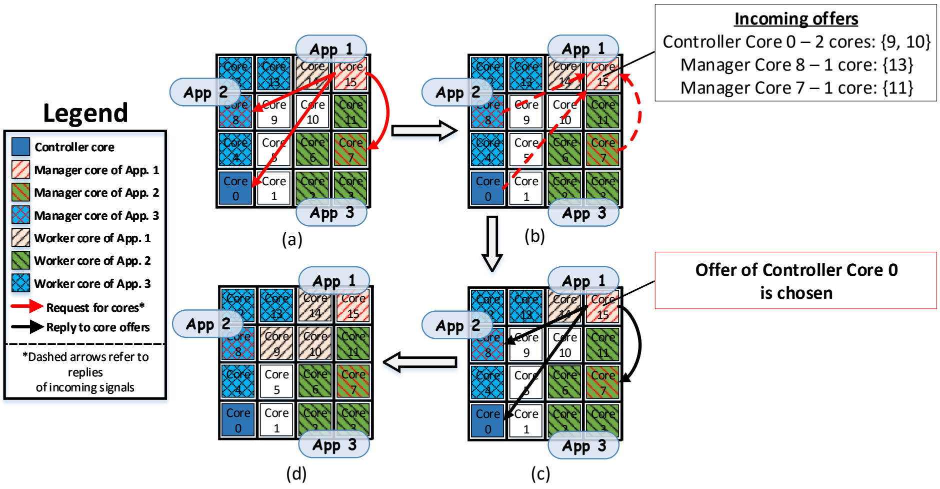


Figure 4.5: Steps of the self-optimization process.

refers to the effort of dynamically increasing the resources and speedup of the managed application. This resource bargaining takes place between agents that possess resources, i.e. Manager and Controller cores. The underlying concept is that the availability of free resources can vary over time due to application terminations, which are asynchronous events. Furthermore, self-optimization can remedy an initial allocation of resources which is sub-optimal with respect to the total application speedup. Therefore, the Manager core initiates rounds of resource searching in order either to locate free resources or acquire some from another application while satisfying the cost function of Eq. 4.6.

Fig. 4.5 presents an example of the concept of self-optimization. All PEs of an application are distinguished by the same color and pattern. In Fig. 4.5a, the Manager core of App. 1 (Core 15) asks for more resource from (i) the Manager Core of App. 2 (Core 8), (ii) the Manager Core of application 3 (Core 7) and (iii) Controller Core 0, which owns the idle cores of the system. The request of resources is performed by sending an appropriate signal to each one of them, which in turn evaluate the resource exchange possibility according to Eq. 4.6 and make their respective offers, by replying with an appropriate signal. The specifics of the offers are illustrated in Fig. 4.5b. Due to the fact that there are idle cores, the Controller core was the one to offer the most resources. Nevertheless, since App. 1 owns only two cores, there were also offers by the other running applications to give up some of their PEs in favour of the increase of the total speedup of all three applications. In Fig. 4.5c, the Manager core of Application 1 replies to all other agents whether their offers are accepted or not. The offer of Controller core is the one chosen because it includes the most offered resources in addition to not affecting the allocated resources of the rest of the running applications. The offers of the Manager cores of Apps. 2 and 3 are rejected. Fig. 4.5d illustrates the resulting state of the system, where Application 1 occupies 4 cores instead of 2 at the beginning of the example.

To achieve the periodic execution of the self-optimization process, a $SELF\_OPT$ timer is set and when it expires the Manager checks whether the application has maximized its speedup or is near to its completion (Algorithm 4, lines 7-9). If the application speedup is maximum (Eq. 4.1), then there is no need to search for more cores. The same applies if the estimated remaining time of the application is less than the required time for the self-optimization to be completed ($T_{finish} < SELF\_OPT$). If a self-optimization process of an application is decided, a negotiation round is performed between active applications inside a region of the platform with center the PE of the Manager and radius R. The search is limited inside this region in order to constraint the necessary communication for the self-optimization process as well as to decrease the dispersion of the workers of an application.

Resource allocation interplay example: Fig. 4.6 illustrates an example of the inter-play amongst cores with different roles on a 4×4 multi-core system managed by DRTRM. We assume that the platform has two Controller cores (Core 0 and Core 8), each of which monitors *Clusters* 0-7 and 9-15 respectively. In this example, we assume that there is one active application (App 1) with Core 15 as its Manager and Cores 10, 11 and 14 as workers. When another application (App 2) arrives, Controller core 0 instructs Core 3 to act as the Initial core for the new application. The newly appointed Initial core intends to acquire cores in area R which, without loss of generality, can be assumed to enclose the entire platform. Its first task is to send a signal in order to discover which Controller cores are active in region R (Fig. 4.6a).

Having acquired the ids, the Initial core sends a signal to all of them inquiring which Manager cores are active in region R (Fig. 4.6b). After this information is acquired, it sends a request for cores to all the Managers in region R (Fig. 4.6c). They evaluate the respective *Cost* in Eq. 4.6 and they respond appropriately (Fig. 4.6d). Then, the Initial core evaluates the best offer and accepts it. The last tasks are (i) to appoint the new Manager core for the application (Core 2- Fig. 4.6e) and (ii) to inform the Manager cores accordingly about the acceptance or rejection of their offers (Fig. 4.6f).

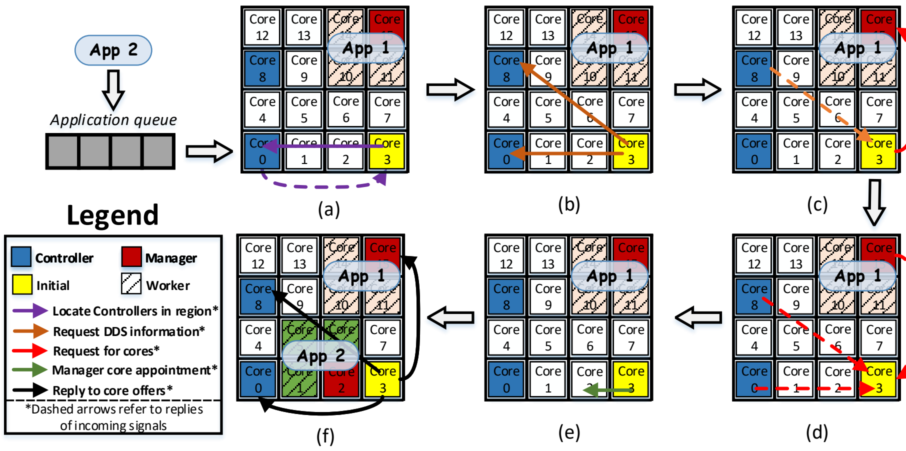


Figure 4.6: Resource allocation interplay example.

4.5.4 Node Internal States and Transitions

As stated, PEs in *DRTRM* have different roles. From these roles, the *Controllers* are defined at the initialization of the platform and they cannot change. All other computing resources can alternate between the roles of *Initial*, *Manager*, *Worker* and *Idle* at run-time. Fig. 4.7 illustrates the different possible states for an Initial and a Manager core. Transition between states occurs when specific signals are exchanged among cores or triggered internally (e.g. a timer expiration). Each edge of the graph is labelled with the corresponding received signal (ergo the SIG_ prefix), while there are signals that do not trigger a transition between states. Tables 4.2 and 4.3 provide more information about each state. The implemented signals used in *DRTRM* are described as follows:

Additionally, an application is characterised by a number of internal states corresponding to its different operational phases. The transition between different states is dictated by the Manager core of the application. In summary, the available application states are:

i. **RUNNING**: The application has been initialized and the worker cores are executing its workload.

ii. **TERMINATED**: The application proceeds asynchronously to this state when all its workload has been executed and the final operations for proper application termination need to be executed.

Table 4.1: Detailed information about the signals of DRTRM.

Signal	Description
SIG__INIT__APP	Signal that a node must act as an Initial core.
SIG__INIT__MAN	A signal sent by an Initial core for the initialization of a Manager core.
SIG__APPOINT__WORK	A Manager sends this signal to one of its workers for workload execution.
SIG__INIT__TIMER	A signal raised by the expiration of a timer in a Initial core.
SIG__SELFOPT__TIMER	A signal raised by the expiration of a timer in a Manager core.
SIG__SELF__OPT	A signal raised by the Manager core to notify its application that a self-optimization process has started.
SIG__FINISH	A working node sends this signal to its Manager core when it has finished the execution of its appointed workload. Additionally, when a Manager core goes into MANAGER__ENDING state i.e. its application has finished its life cycle, it informs its Controller core through this signal.
SIG__STATE__FINISHED	This is used for FSM completeness purposes in order to signify that the sequential part of a given state is over and the impending action is a transition to a new state.
SIG__ACK	Acknowledgement that a node is waiting for incoming data.
SIG__REQ__CORES	A signal sent to a core to request for resources in a region.
SIG__REQ__CORES__REPLY	Signal sent to offer resources after a relevant request.
SIG__REP__OFFERS	Signal sent to notify the acceptance or rejection of an offer for resources.
SIG__DISC__CNTR__CORES	Signal received by a Controller to inform the sender about the monitoring Controller cores of a specific region.
SIG__DISC__CNTR__CORES__REPLY	Signal sent by a Controller to inform the sender about the monitoring Controller cores of a specific region.
SIG__REQ__DDS__INFO	Signal received by a Controller to inform the sender about the running applications in a specific region.
SIG__REQ__DDS__INFO__REPLY	Signal sent by a Controller to inform the sender about the running applications in a specific region.
SIG__ADD__CORES__DDS	Signal sent from a Manager to a Controller asking to register new PEs belonging to the running application.
SIG__REMOVE__CORES__DDS	Signal sent from a Manager to a Controller asking to remove some registered PEs of its running application.

iii. **RESIZING**: This state dictates that the application is not executing any workload because the necessary actions for successful resizing are performed.

Table 4.2: Detailed information about the states of an Initial core.

State	Description
IDLE_CORE	A state where the core does nothing and waits for an incoming signal.
INIT_CORE	The first state of a new Initial core. Its normal next state is **INIT_CORE_REQ_DDS_INFO**, unless a SIG_INIT_MAN signal is received and the core must also act as the Manager of another application (different from the one under admission). In this case, it switches to its managerial role and when its duties are over, it resumes as an Initial core.
INIT_CORE_REQ_DDS_INFO	The Initial core sends requests to all Controller cores in a region in order to locate which applications (Manager cores) are active in it. Upon reception of all replies, it proceeds to **INIT_CORE_SEND_OFFERS**.
INIT_CORE_SEND_OFFERS	The Initial core sends requests for cores in various areas of the system. Then, a timer is set to wait for offers and a transition to **IDLE_INIT_CORE** is performed.
IDLE_INIT_CORE	The state is maintained, waiting for core offers. If a SIG_INIT_MAN is received, it follows that the Initial core is instructed to be the Manager of another application (different from the one under admission). Thus, the managerial flow is initialized and execution flow will return to Initial core duties after the application is over. If a SIG_APPOINT_WORK signal is received, it means that the node belongs to an application and must execute some workload proceeding to **WORKING_NODE** state.
INIT_CORE_CHK_OFFERS	After timer expiration, this state is used for evaluation of any offers. If there are none, a return to **INIT_CORE_SEND_OFFERS** state is mandatory to reset core search, since it is imperative that at least one PE is found for the new application. If offers have been received, the new Manager is instantiated. After that, any pending roles (Worker or Manager) are resumed or the core enters **IDLE_CORE** state.
WORKING_NODE	The core executes application workload. When it is over, transition to **IDLE_CORE** state is performed. If workload execution started while having Initial core duties, then in **IDLE_CORE** state, the timer of Initial core will expire and execution flow will transit to **INIT_CORE_CHK_OFFERS** state.

4.5.5 Inter-node Synchronization and Data Exchange

The most challenging part of the proposed framework is the inter-node signal and data exchange. We distinguish all signals described in Section 4.5.4 to two different parts: (i) the part where a signal is sent and (ii) the part where data is exchanged between cores. *DRTRM* employs a three-way handshake communication. As a consequence, whenever core A wants to transmit data to core B, it first sends a signal denoting the semantics of its request and then core B must respond with a signal of acknowledgement (**SIG_ACK**) to declare its availability to receive the data. Only when A receives this acknowledgement, it starts sending the data. However, this protocol does not guarantee deadlock

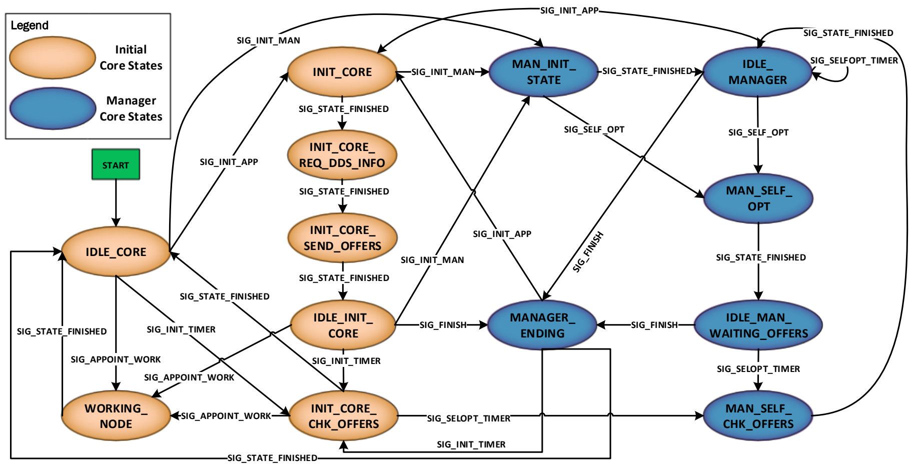


Figure 4.7: Possible internal states of a core under *DRTRM*.

Table 4.3: Detailed information about the states of the Manager core.

State	Description
MAN__INIT__STATE	The Manager is initialized and makes its presence known to its Controller. Application workload is distributed to its workers. Also, the core decides whether a self-optimization is necessary.
MAN__SELF__OPT	A self-optimization process is initialized and requests for cores are sent. Having sent the requests, a timer is set and state is changed to **IDLE__MAN__WAITING__OFFERS** state to wait for offers.
IDLE__MAN__WAITING__OFFERS	The Manager core is idle and waits for core offers. If SIG__FINISH signals are received by all workers owned by the Manager core and application workload is over then execution flow is changed to **MANAGER__ENDING** state in order to finalize the application.
MAN__SELF__CHK__OFFERS	When the timer goes off, the received offers are checked. The best offer is accepted and application resizing commences. Otherwise, a new timer is set for the repetition for the self-optimization phase and state is changed to **IDLE__MANAGER**.
IDLE__MANAGER	The Manager core remains idle in the sense that a self-optimization process is not executed. Incoming requests from other nodes such as requests for cores are serviced. If a timer goes off and criteria for self-optimization are met it proceeds to **MAN__SELF__OPT** state otherwise, it remains idle. If workload execution in completed, it proceeds to **MANAGER__ENDING** state.
MANAGER__ENDING	The Manager informs its Controller core for the its application termination and its managerial duties are over. If a SIG__INIT__MAN signal has been received then execution flow is changed to **INIT__CORE** state to start Initial core duties. Otherwise, the state is changed to **IDLE__CORE**.

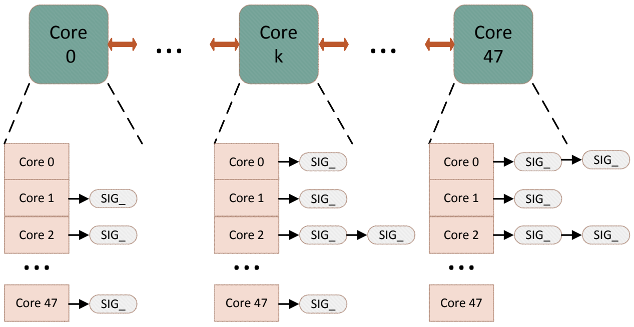


Figure 4.8: Interaction queue.

prevention. For example, assume that core A sends a signal to core B and about the same time core B sends the same signal to core A (e.g. A and B request cores from each other). Then, both will answer with SIG_ACK and both will wait for data, thus creating a deadlock.

Additionally, there is a case where core A sends multiple requests to core B at a certain time. However, due to the network latency, the arrival of signals cannot always be predicted and there is a possibility that a later request is served first. The solution of a transaction ID per request is not considered efficient as it creates extra transmitted information. In addition, it increases the complexity of inter-node communication logic, since extra mechanisms are required to re-arrange the incoming messages so that their semantics are respected, e.g. a core request proceeds a core offer, a rejected message is withdrawn from the incoming messages queue and so on. Therefore, in *DRTRM* a sender node A is allowed to have only one active interaction with node B at a certain moment, by using an interaction queue as depicted in Fig. 4.8. This queue, is responsible for handling the **outgoing** requests of any PE and for guaranteeing that at any given moment a core has only one outgoing request for any other PE on the system. If a need for a second request arrives before the completion of the first one, it is queued up in the interaction queue and the appropriate signal will be sent only when the first request is complete. Using the knowledge of their interaction queue, combined with the information about incoming requests from other cores, receivers can assess when a deadlock will occur and thus reject the necessary incoming requests to avoid it.

Apart from the interaction queue, each PE maintains a First Come First Served (FCFS) queue, in order to handle its **incoming** requests. Specifically, this queue relies on message passing buffers, which are used in a circular manner for incoming signals. Since the sender is not known in advance, the sender must also provide its id along with the signal number. The size of the circular

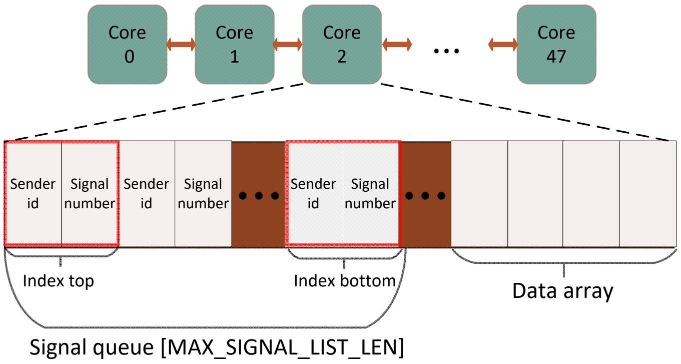


Figure 4.9: Allocated space for signals and data transactions.

buffer for signals is defined by the parameter `MAX_SIGNAL_LIST_LEN`. A typical values of this parameter is 64 and has proven to be adequate for all simulations and configurations of the proposed framework. In the circular buffer, an index called `index_top` points to the current start of the buffer and represents the first signal that was received. The index `index_bottom` represents the signal that was most recently received and will be the last to be served. Regarding synchronization, one mutually excluding lock in every node is sufficient for the signal exchange to be successfully carried out. Fig. 4.9 depicts the circular buffer both for the data transactions and for the signals.

Fig. 4.10 depicts the process of node A sending signal `SIG_X` to node B. First, node A waits until it acquires the lock of node B. Then, it copies the value of `index_bottom` of node B to its memory to avoid multiple operations on the off-chip RAM. The local index is increased by one. The actual new index is the increased value modulo `MAX_SIGNAL_LIST_LEN`. The values of signal type and sender id are written to the signal buffer of B and the `index_bottom` is updated while releasing the lock. Node B checks for the availability of its lock. After the lock has been acquired, node B checks if `index_bottom` and `index_top` have the same value. If this is the case, then no new signals have been received. If the two indexes have different values, node B reads the signal and sender id indicated by `index_top`. Before proceeding to the signal handler function, the lock is released so that B can receive new signals again. After this function is over, `index_top` is increased in a circular way and the process is restarted until `index_bottom` has the same value as `index_top`.

Regarding data exchange, a signal from core A to core B does not involve data transmission, since it cannot be a priori known whether the recipient will acknowledge or reject it. Data is actually transmitted only when the receiver

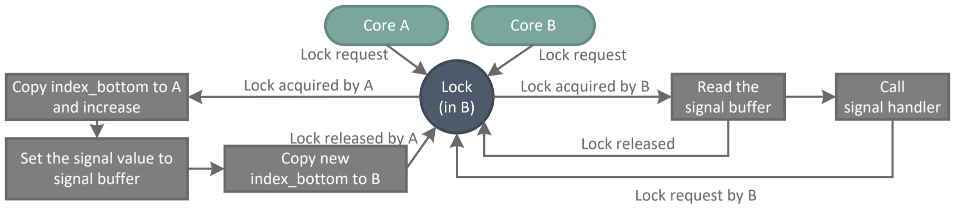


Figure 4.10: Inter-node communication for exchanging signals.

replies with a SIG_ACK signal to the sender. The payload is delivered by writing in the incoming data buffer of each core, which for the case of Intel SCC is allocated in the Message Passing Buffer [114], as shown in Fig. 4.9. The size of the payload varies according to the requirements of the signal, e.g. an offer for resources includes the IDs of offered PEs, while the reply for this offer includes only one integer indicating its acceptance or rejection.

4.6 Experimental Evaluation

4.6.1 Contemporary many-core Systems-on-Chip

The design of DRTRM as presented in Section 4.4, aims at minimizing its dependence on the specifics of the underlying hardware architecture. The requirements with respect to hardware is a large number of Processing Elements inter-connected via a Network-on-Chip architecture, thus refraining from the use of shared memory for inter-core data exchange. With respect to software, the current stack of DRTRM has been designed under the assumption that each PE is managed by a dedicated instance of an OS kernel, as depicted in Fig. 4.3. While this highly simplifies the development of DRTRM, its design is not dependent in the existence of this OS kernel. Therefore, in the case of OS absence, a middleware is required to aid the execution of the high level functionality of DRTRM on the target hardware.

During the writing of this dissertation, there is an ever increasing escalation in the number of PEs integrated on the same System-on-Chip. The highly available commercial processors incorporate up to 28 physical processing cores [8], communicating using a shared memory infrastructure. Furthermore, the cutting edge of PEs incorporates numerous hardware threads within the same processing core, which increases the number of logical cores of up to 56 in Intel Xeon Platinum 8176F Processor [8]. In such designs, there is mesh interconnection for the PEs in order to achieve the shared memory inter-connection requirements.

In all these cases, one OS instance is capable of managing the entirety of logical cores, thus having a centralized view of the system. While DRTRM is capable of being executed on such devices, its basic design principles would be compromised and its distributed nature could result in higher overhead than gain, since the OS oversees the entire system and is capable of performing online, efficient resource allocation.

Intel Many Integrated Core (MIC) architecture, has spawned significant results with respect to providing CPUs with many integrated processing elements. The current highlight of this effort is the Intel Xeon Phi Processor series, based on Knights Landing architecture [211], where the maximum number of processing cores reaches 72, with a total of 288 hardware threads. The connection of the processing elements is performed in a cache coherent manner, achieved via a ring interconnect [211]. Again, an OS instance oversees all hardware threads and thus the management of the system via DRTRM concepts could possibly lead to a run-time overhead.

The aforementioned designs are based on CISC Instruction Set Architectures (ISA). Recently, RISC ISA processors have gained a lot of attention towards the realization of many-core Systems-on-Chip. A well established multi-core architecture is ARM Big.Little, which incorporates up to 8 PEs with asymmetric computing capabilities, i.e. fast (Big) and slower (Small) cores [100]. This approach has been adopted by many commercial designs [62], but the small amount of integrated PEs is prohibitive for the application of distributed resource management on the system.

Kalray MPPA [123] integrates 256 VLIW PEs, divided into computing clusters of 16 elements. Communication inside a cluster is achieved via shared memory, while clusters are inter-connected via a Network-on-Chip. Low-level programming interfaces are exposed for the development of applications, while there is also support for Real-Time Operating Systems and OpenCL [213]. Adapteva has introduced Epiphany-IV [171], based on 64 RISC processors interconnected on a low latency, 2D mesh NoC. The programming of the system is achieved via low-level interfaces implemented using the C programming language. The roadmap of Adapteva is to introduce a chip with 1024 PEs on the same chip, where each PE will comprise of a RISC CPU, local memory and a router for the NoC of the platform [170]. Pezy-SC processor [23], developed by PEZY Computing K.K., incorporates 1024 multi-threaded PEs using an on chip cache architecture of three levels, divided to subsets of PEs. The platform is not cache coherent and thus data need to be explicitly flushed in the different cache levels. The system is programmed using a subset of OpenCL functions [23].

Cavium ThunderX processor series [102] incorporates up to 48 64-bit ARMv8 PEs communicating using a shared memory infrastructure. The system aims

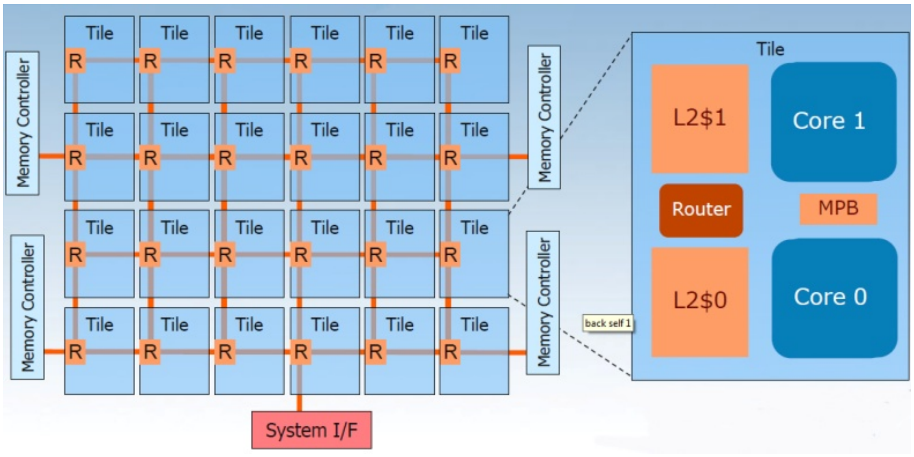


Figure 4.11: Layout of Intel SCC platform [158].

at the Cloud and Data Center market and thus includes accelerators for virtualization, storage and networking. Moreover, its operation is supervised by centralized Linux OS distributions. Last, the SpiNNaker (a contraction of Spiking Neural Network Architecture) project [95] aims at building a massively parallel computing platform, inspired by the function of the human brain. The project has developed a working prototype which incorporates 864 ARM968 PEs in total, grouped into nodes of 18 PEs. The inter-node communication of PEs is achieved by Inter-processor communication is based on an efficient multicast infrastructure inspired by neurobiology.

The features of these novel many-core systems make them well-suited candidates for the deployment of DRTRM. However access to such systems is limited, while the successful porting of DRTRM is dependent of the development of an efficient middleware, which will compensate for the absence of an OS. Taking the above into account, DRTRM was deployed on Intel SCC, which as detailed in Section 4.6.2, meets all the necessary HW and SW design requirements.

4.6.2 Intel SCC: Target NoC based evaluation platform

For the evaluation of DRTRM, Intel Single-chip Cloud Computer (SCC) [114] was used as the driver many-core platform. Intel SCC is a 48-core, single chip platform with a mesh NoC interconnection for on-chip communication. It consists of 24 tiles of 2 PEs each, based on Intel P54C cores, which are second generation in-order Pentium processors, running a Linux operating system per core. Each of the two processors has a dedicated instruction and data L1 cache of 16 KB, while they share 256 KB of L2 cache memory and 16 KB of a fast, on-die shared SRAM called Message Passing Buffer (MPB), which provides a fast and reliable message passing interface for data dispatching amongst cores. Having 24 tiles, Intel SCC provides 384KB of Message Passing Buffer. In addition, 4 memory controllers on the system provide access to off-chip DDR3 DRAM of up to 64 GB, which is visible from all cores. Tiles are connected through a 2D-mesh and a router inside each tile is responsible for forwarding outgoing packets to the correct target, using X-Y routing. When a message is sent from one core to another, data is sent through the Message Passing Buffers on the chip. While the processor does not offer any hardware-managed memory coherence, it features a new memory type to enable efficient communication between cores. This new memory type is called the Message Passing Buffer Type (MPBT) [114]. Fig. 4.11 illustrates the layout of Intel SCC platform.

In order to make programming easier and increase the portability and scalability of programs written for the SCC platform, Intel provides a communication environment known as the RCCE [158]. RCCE distributes evenly the MPB address space to the 48-cores, designating that each processor will have 8KB of memory in this buffer for itself. It provides two basic interfaces for inter-node communication. The first is the *gory* interface, a low level design that offers the programmer greater control over the SCC, in expense of need for explicit synchronization. The second interface is the *basic* one, which offers blocking `send` and `recv` functions.

DRTRM was developed using the gory interface to enable asynchronous send/receive data operations between cores. Each distributed agent allocates at its initialisation phase a memory space in MPB of MPBT data type by using the `RCCE_malloc` function. After this allocation, agents can exchange messages through the `RCCE_put(A, buffer, size, ID)` and `RCCE_get(buffer, A, size, ID)` functions, where `A` is the space allocated in MPB and `buffer` is a memory buffer in the local address space of the agent. The `RCCE_put` function transfers data from buffer to MPB, while the `RCCE_get` function does the opposite operation. In both functions `size` denotes the number of bytes to be transferred, while `ID` is the number of the target node. To achieve mutually exclusive memory operations, a `test&set` register is provided in every core.

4.6.3 Evaluated applications

To alleviate the impact of ideal application characteristics on the analysis of the efficiency of the proposed resource allocation scheme, we developed a set of actual applications, which exhibit behaviour similar to the basic characteristics

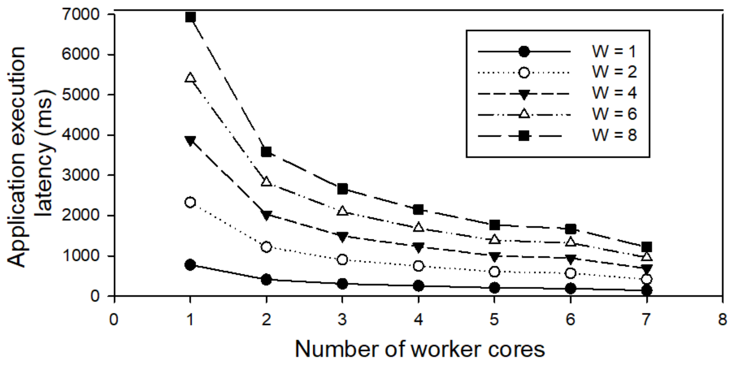


Figure 4.12: Performance of malleability model in respect to #cores and workload size for matrix size M = 4096.

of a malleable application, as dictated by Eq. 4.1. The choice of the applications was in the interest of providing an experimental setup able to represent realistic characteristics of emerging workloads, aligned with the description of IoT applications presented in Chapter 3. At the same time, the patterns of the computations of the following chosen applications allow their parallel execution, while respecting the model of malleable applications.

Matrix-Vector Multiplication (MVM) takes as input a integer matrix $A_{N \times N}$ and an integer vector $V = [v_1, v_2, \cdots v_N]^T$ and produces the outcome of their multiplication. **Support Vector Machines Classifier (SVM)** [109] is a popular pattern recognition algorithm, which is able to tackle complex non-linear classification problems (Section 3.3.3). The utilized models were based on the ECG analysis application presented in Section 3.3, while the actual SVM source code was extracted from libSVM [53], an acclaimed open source implementation of the classifier. The last application is **Fast Fourier Transform (FFT)**, corresponding to the feature extraction stage of IoT applications and derived from the Parsec benchmark suite [39].

To create computationally intensive applications, the core kernel operations (e.g multiplication) is repeated W times and this variable is considered the workload of the application. Resizing is enabled only at specific synchronization points between these repetitions. The speeudp function $S(n, M)$ and remaining execution time T_{rem} of the application are modelled as:

$$S(n, M) = \frac{Exec\_t(1, M)}{Exec\_t(n, M)}; \quad T_{rem} = W_{rem} * Exec\_t(n, M) \tag{4.7}$$

where W_{rem} refers to the remaining multiplication repetitions, n is the number of worker cores and $Exec\_t$ is a function returning the execution time for one instance of the matrix multiplication for n cores. *Exec_t* is dependent both on the input data-set size as well as on the number of processes working in parallel and is derived from extensive profiling on the target platform. Fig. 4.12 shows the execution latency of various workloads of the MVM on Intel SCC in respect to allocated resources. We observe that as dictated by Eq. 4.1 and Eq. 4.2 the scaling of the implemented application is irrelevant to its workload W and respects the linearity between W values and remaining execution time. The same behaviour is observed for the the SVM and FFT applications as well.

To achieve this behavior, we force DRTRM to allocate one parallel process per core, thus eliminating co-scheduling interference effects, since SCC does not support hyper-threading at the core level. At run-time, the Manager communicates to each worker the upper and lower bound of the consecutive rows of the input matrix that it has to multiply. The workload is evenly distributed and the workers do not exchange any information with one another. Each worker writes in the memory of its Manager core the computed results. In order to alleviate the overheads of memory block caching that limits the benefits of dynamic malleability, we carefully tile the workload distributed to each core to fit the SCC node's cache capacity, while also performing data pre-fetching at the initialization of the application.

The input of the implemented parallel applications are (i) for the MVM application, a dense matrix and vector of 4096x4096 and 4096x1 integers, respectively, (ii) an SVM composed of 4096 support vectors, each containing 4096 floating point features and (iii) a signal of 65536 floating point samples for the FFT application. The implemented applications exhibit collective communication during workload distribution and reduction of computed results by the Manager core. Different inter-worker communication patterns can be modelled in the speedup curve of the application and be taken into account by the Manager core at run-time.

As a closing remark, the efficiency of the presented resource management scheme is not directly coupled to the employed application model. However, the efficiency of the employed dynamic, inter-application exchange algorithm is dependent on an reliable way of having an estimate of the remaining execution latency of any running application. Consequently, DRTRM principles can be extended to the use of any parallel application accompanied by a concrete performance and interference model, targeting a many-core System-on-Chip. In the absence of a relevant model, the performance of the applications would be unpredictable and thus the allocation of resources would result in frequent sub-optimal decisions. In general, providing such a model is a very challenging scientific research issue [139, 38], a fact that contributed to the adoption of malleable parallel application model for the development of DRTRM.

4.6.4 Measured Results Overview

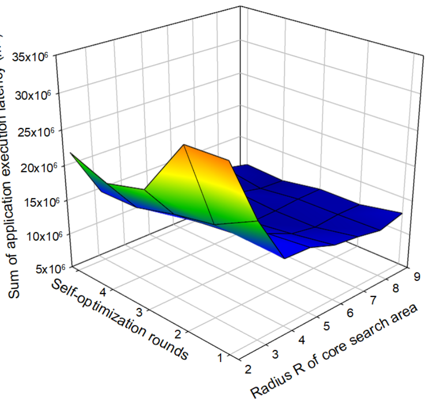
In order to evaluate *DRTRM*, we compared it against: (i) *DistRM* [130], a distributed manager that assigns different run-time agents to cores, which collaboratively perform application mapping via a communication scheme; and (ii) The distributed manager *DRM* presented in [21], which also offers self-optimization and self-organization functions on top of distributed management. The evaluation focuses on quantifying the effectiveness of the different run-time decision making alternatives of the aforementioned frameworks. To achieve that, we alleviate the impact of implementation parameters by making use of the same functional backbone presented in Sections 4.5.4 and 4.5.5 for all different management schemes.

The utilized metrics for the evaluation of the resource allocation schemes are (i) *Total application execution latency*: The total required time for all applications to execute their workload, (ii) *Total number of core negotiation algorithm executions*: This value presents how many times this algorithm was executed in total and quantifies how demanding the resource management process was, (iii) *Total number of exchanged messages*: The total number of all messages exchanged for a scenario to be successfully completed, (iv) *Total size of exchanged messages*: The total size in bytes of the exchanged messages and (v) *Number of hops for exchanged messages*: The total number of hops that the exchanged messages had to travel in order to reach their destination.

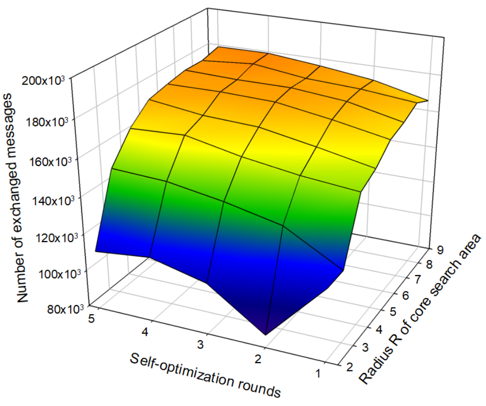
In our effort to create a realistic simulation environment, we examine different scenarios of applications requiring admittance and execution on the many-core system. The scenarios involve applications of all available types and are considered complete only when all incoming applications have successfully completed their workload W. The distribution of W values in each scenario as well as the arrival rate of the applications on the system have been designated according to a the study presented in Chapter 5, aiming at creating input scenarios which can stress the availability of resources on the target platform.

4.6.5 Design Space Exploration on DRTRM Resource Allocation Parameters

The first part of our experimental campaign focuses on identifying the optimum parameters regarding the aggressiveness of the resource allocation search of agents in DRTRM. The examined parameters are (i) the radius R of the area (See Section 4.5.3) inside which a resource search will take place and (ii) the maximum number of Self-optimization rounds that an application can initiate. The presented results in Fig. 4.13 summarize the outcome of scenarios of 128



(a) Sum of application execution latency vs Self-optimization rounds vs Core search radius.



(b) Total number of exchanged messages vs Self-optimization rounds vs Core search radius.

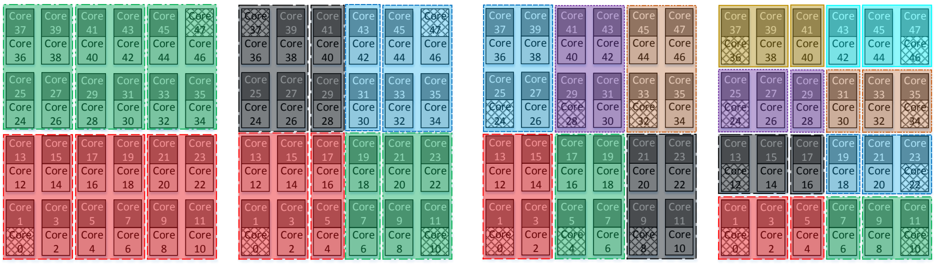
Figure 4.13: Design Space Exploration for DRTRM parameters optimization.

application of all implemented types and the behaviour of DRTRM is quantified in terms of the sum of application execution latency (Fig. 4.13a) and intensity of communication i.e. message count (Fig. 4.13b).

Results indicate a trade-off between the examined metrics, since reduced aggressiveness in search for resources leads to high application execution latency combined with reduced exchanged messages. In other words, Manager cores cannot effectively increase their workers when searching in close vicinity and do not repeat this process often, but this lack of activity results in reduced agent inter-communication. On the contrary, as aggressiveness is heightened we observe higher efficiency in resource management, which requires increased communication between the distributed agents in order to be achieved. However, the reduction of application execution latency reaches a saturation plateau, implying that maximum aggressiveness is not the optimal solution, since the system has reached a point where resources are fully utilized. For the rest of the experiments, we chose Self-optimization search radius equal to 6 and Self-optimization rounds equal to 3.

4.6.6 Evaluations of Resource Allocation Efficiency

Having designated the aforementioned parameters, the conducted experiments focus on quantifying the efficiency of the proposed resource allocation scheme against state-of-the-art distributed resource management frameworks. The first



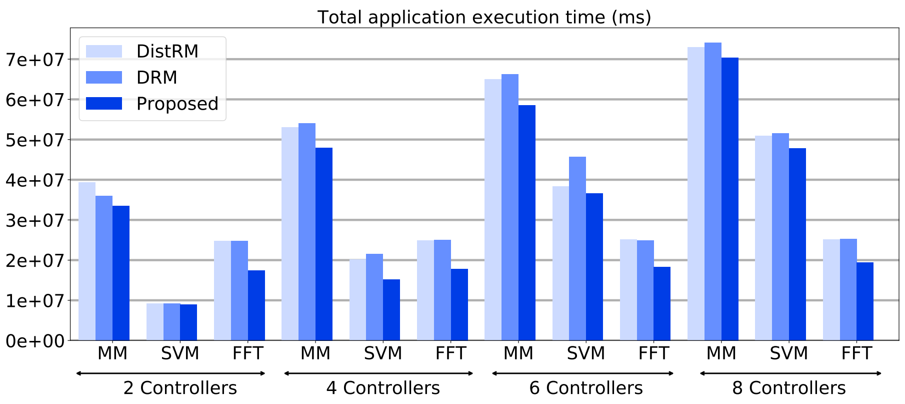
(a) 2 Controllers. (b) 4 Controllers. (c) 6 Controllers. (d) 8 Controllers.

Figure 4.14: Examined cluster topologies. Regions of the same color belong to the same cluster. Controller cores are shaded.

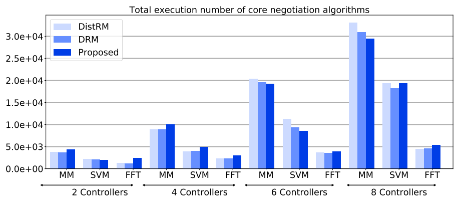
collected data is the outcome of the execution of 128 applications for a varying number of Controller cores on the system. Results are grouped according to this number and the type of the utilized parallel application.

The number of Controller cores creates an inherent design trade-off, taking into account that they are not involved in application workload execution. Increasing their number reduces the amount of cores that each Controller monitors and consequently the intensity of incoming requests to be processed. At the same time, the number of available worker PEs on the system is reduced since more cores have been appointed as Controllers. In this work, we examine and evaluate *DRTRM* configurations of 2, 4, 6 and 8 Controller cores. The case of one Controller is omitted since it creates a centralized way of information storing and communication. Numbers greater than 8 have not been considered for Intel SCC (48 cores in total), as it is considered inefficient. However, the described framework can seamlessly support configurations of any number of Controller cores. For the following experiments, Controller cores have been placed in such a way that they monitor continuous areas on the system, as illustrated in Fig. 4.14.

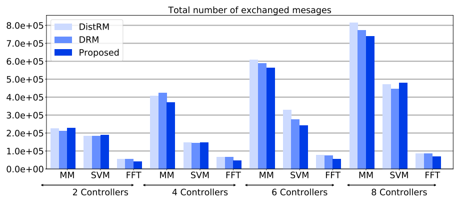
The overview of the results, demonstrates the ability of our proposed scheme to make better decisions regarding the allocations of the PEs of the system and this is reflected in the reduced execution latency of applications (Fig. 4.15a). The small number of available PEs on the system, highlights the efficiency of our proposed workload aware resource negotiation algorithm, which manages to avoid non profitable exchange of resources and allows applications to conclude their life-cycle earlier. Furthermore, by inspection of the comparison of DistRM and DRM we observe a stochastic behaviour with respect to which scheme outperforms the other. This is attributed to their random selection process regarding application initialization and validates our intention to provide an



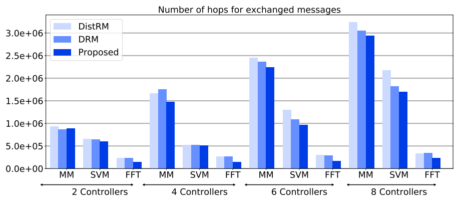
(a) Applications' execution latency.



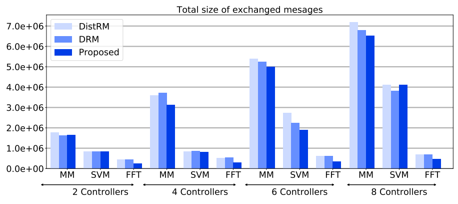
(b) Computational effort of resource exchange negotiations.



(c) Total number of exchanged messages.



(d) Total Manhattan distance of exchanged messages.



(e) Total size of exchanged message.

Figure 4.15: Evaluation of resource allocation efficiency for scenarios of 128 incoming applications.

Initial core designation policy, which minimizes the impact of this selection.

Regarding the required execution latency, we observe that the most lightweight task is SVM classification, while Matrix Multiplication requires the most time to be executed. For these two applications, there is clear correlation between the number of Controllers and increased execution latency. More Controller cores results in less workload per Controller, however this reduction is not enough to compensate for the fact that there are less available worker PEs. Regarding the FFT application, its limited scalability to many workers reduces the effect of the number of Controller cores on its execution latency. The crucial parameter for this application is the spatial distribution of Initial cores, which reduces the congestion of FFT instances on the same region. This sparse distribution is achieved by our proposed Initial core designation strategy, thus resulting to faster FFT application execution. Nevertheless, the exploration in the number of Controller is important in order to validate that the proposed scheme can maintain its efficiency while resource scarcity is aggravated. In summary, our proposed scheme results in 17.5% faster execution in average compared to its rivals, while the maximum can reach up to 30%.

As far are the rest of the presented metrics are concerned, we observe in Fig. 4.15b, that the number of executed instances of the resource negotiation algorithm varies per examined scenario. This variability is due to the activity of the executed agents on the system, which start and finish their execution in a dynamic manner. The combined examination of Fig. 4.15a and Fig. 4.15b reveals that the increased execution effort is correlated with the quicker application workload execution of our proposed scheme. In other words, the extra computational burden for resources negotiation pays off by leading to better resource allocation.

Fig. 4.15c to 4.15e capture the communication aspect of the presented evaluation and show that in the vast majority of cases, our proposed scheme manages to generate less communication traffic compared to the rest, reaching an average reduction of 12.5% in exchanged messages. This is due to the fact that more cores are allocated for workload execution, which is a computation and not communication bound task. In addition, the quantity of exchanged messages is dependent on the required time for applications to be executed since the activity of many applications simultaneously results in increased inter-node communication for resource negotiation purposes. The observed instances where the proposed framework produces more messages compared to DRM are attributed to a marginal case when applications under initialization do not acquire resources, because running applications are close to their conclusion. This does not compromise the effectiveness of the proposed resource negotiation scheme but prolongs the application instantiation process, thus leading to elevated number of exchanged messages.

Apart from the reduced traffic, we observe in Fig. 4.15d that when the proposed scheme is employed, message exchange is more locally contained, i.e. the distance between sender and receiver PEs is in average smaller. This is very important because it reduces the utilization of NoC routers and saves energy, since there are less hops for a message to reach its destination. Last but not least, the total size of exchanged messages (Fig. 4.15e) is in most cases reduced

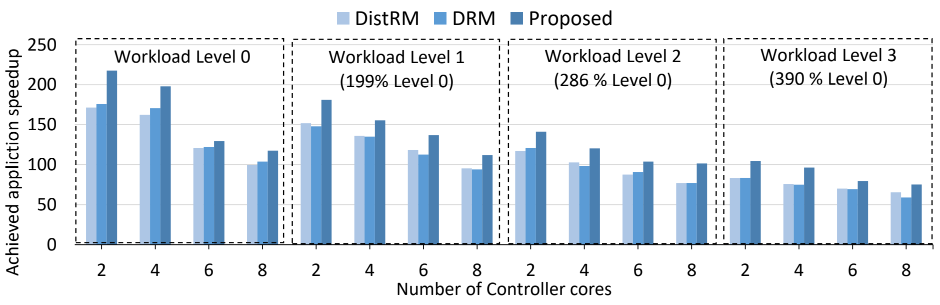


Figure 4.16: Evaluation of resource allocation for applications derived from the malleable application model.

in our proposed scheme, as consequence of the reduced communication traffic. This implies less data exchange on the target NoC and also contributes to reduced congestion of the NoC links.

From a secondary point of view, the results in Fig. 4.15b to 4.15e outline the imposed overhead of the resource management variants on the system, during run-time. The measured data show that the proposed framework manages to reduce the communication aspect of the imposed overhead in comparison to its rivals. Exchanged traffic is less and more locally constrained, leading to less interference and congestion on the communication links. This is achieved at the cost of elevated computational complexity, in order to produce more efficient application mappings. Consequently, as shown in Fig. 4.15b, the imposed computational overhead of the proposed scheme is averagely higher than the investigated alternatives. However, since these computations are distributed, the actual overhead on each distributed agent is minimized.

To further quantify the effectiveness of our proposed framework we employ a more diverse application workload mix, which is derived using the malleable application model presented in Section 4.4. The mix is created by randomly generating applications characteristics according to this model, with values ranging from 0.01 to 100 for parameter σ and from 2 to 16 for parameter A. According to these characteristics and the number of its worker cores, the speedup of an application is calculated at run-time using Eq. 4.2. Low σ values result in applications with steep scaling, which achieve high speedup for almost all values of A. High values of σ, result in applications which exhibit smooth scaling with low speedup values regardless of the values of A.

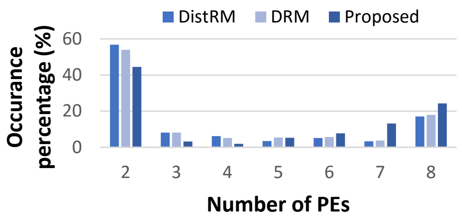
To simulate a workload execution round, we implement it as time delay scaled according to the calculated speedup of the application according to its characteristics and the number of its workers. In order to evaluate our framework under variable workload size, we create four different levels of input workload intensity. The values W of each level are derived using a random number generation function according to a normal distribution. Increasing workload intensity, i.e. higher average W values, is achieved by providing increasing input values to the random generator function. The arrival rate of applications is identical for all workload scenarios, produced according to a random Poisson distribution as presented in Chapter 5.

Fig. 4.16 summarizes the results of the execution of 64 malleable applications, in terms of total achieved speedup, calculated according to Eq. 4.1 as the ratio of the execution latency of applications for a single worker, divided by their measured execution latency. The results are grouped according to the different workload intensity levels. For each level all different Controller core configurations are examined. We observe that our proposed methodology results in higher applications' speedup in all cases, with average gains of more than 20%, reaching up to 30%. The diversity of the scaling characteristics of the input applications maximizes the effect of the resource negotiation algorithm, which allows weak scaling applications to terminate fast without offering resources. Afterwards, these resources are aggregated by highly scaling applications, thus maximizing their speedup. In addition, the experimental evaluation highlights the robustness of the resource allocation efficiency, regardless of the intensity of the input workload. Another important observation is that reducing the number of available worker PEs (i.e. increasing the number of Controller cores), imposes a heavy toll on the achieved speedup since highly scaling applications cannot reach their maximum performance.

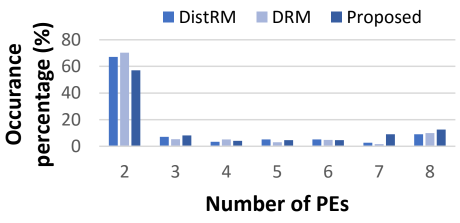
4.6.7 Evaluation of Initial cores' Designation Policy

While the results of the previous Section validate the effectiveness of the resource allocation policy, they do not offer any indication of the correlation of this efficiency to the Initial core designation policy, which is different for the three state-of-the-art implementations. As a means of quantifying this efficiency, we examine the distribution of the number of worker cores of applications throughout our conducted experiments and this information is presented as histograms in Fig. 4.17. This information is used because the ultimate purpose of the Initial core designation algorithm is to aid the DRTRM framework to better disperse application mapping on the target system.

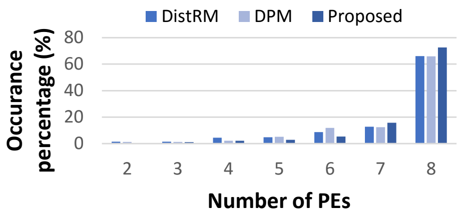
The histograms presented in Fig. 4.17 refer to MVM and SVM applications, as the FFT application scales up to only 4 workers. Results are grouped according to application type and number of Controller cores, as this constraints the amount of available worker PEs. The X axis of each sub-figure refers to the



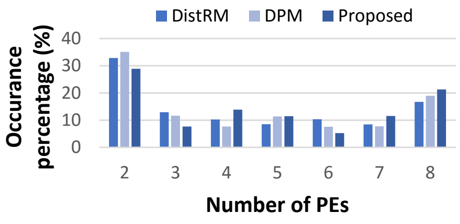
(a) Cores histogram - MM - 2 Controllers.



(b) Cores histogram - MM - 4 Controllers.



(c) Cores histogram - SVM - 2 Controllers.



(d) Cores histogram - SVM - 4 Controllers.

Figure 4.17: Histograms of worker cores' amount for *Cluster* configurations of 2 and 4 Controller cores.

number of workers that executed a workload instance, while the Y axis refers to the occurrence frequency of each number of workers, throughout the execution of the total workload of applications. The maximum number of workers per application was set to 8.

Fig. 4.17a illustrates the aforementioned occurrence percentage for the Matrix Multiplication application when the system operates with 2 Controller cores. We observe that the proposed scheme is able to highly increase the instances of workload execution with maximum or close to maximum worker cores. Due to the aggressiveness of the executed workload there is also a high percentage of instances with only two workers, however this is up to 20% decreased by our proposed scheme compared to the rest of state-of-the-art implementations. The respective gain reaches 300% for near maximum number of workers and 40% for maximum. When the number of Controller cores is increased (Fig. 4.17b) leading to a greater shortage of available PEs for workload execution, our proposed scheme manages to maintain almost the same percentage of low number of workers as well as the same comparative gains in the frequency of the high number of workers, i.e. close to 300% for 7 workers and 32% for 8.

Regarding the execution of the SVM classifier, we observe in Fig. 4.17c that workload requirements are low enough to allow all compared resource management alternatives to provide well balanced and effective resource allocation. Nevertheless, our proposed scheme outperforms its rivals in the instances of higher number of worker cores. For the same application in a system of 4 Controller cores (Fig. 4.17d), there is a high increase in the number of instances with the lowest number of worker cores but our proposed scheme manages to reduce 15% the lowest number of workers, while it increases close to 42% and 20% the instances of 7 and 8 workers, respectively.

4.7 Conclusions

This Chapter presented the motivation and design details of a Distributed Run-Time Resource Manager scheme (DRTRM) for parallel applications targeting many-core, Network-on-Chip based systems. The first Sections provide a thorough description of the various hierarchical roles that the different processing cores of the system can adopt to successfully carry out their resource management tasks. Having described the high level functionality and roles of the proposed software stack, the focus of the manuscript shifted on detailing its low level design details, including a Finite State Machine description of the involved agents as well as the required mechanisms for their efficient communication.

The implementation of DRTRM on an actual many-core system, was bound by two critical parameters. First, efficient run-time resource negotiation is dependent on a reliable mechanism for predicting the remaining execution latency of a running application. To achieve that, the presented work relied on the model of parallel malleable applications, which was replicated via a number of implemented parallel applications on the target many-core platform. Secondly, the design space of available many-core system is large, thus complicating the decision of the target many-core system. The requirements for a meaningful DRTRM implementation are (i) a sufficiently large amount of PEs on the target system, (ii) NoC based interconnection and (iii) an OS instance per PE to support the functionality of DRTRM agents. Taking all these into account, the platform of choice was Intel Single Cloud Chip platform, a 48 many-core system. DRTRM was implemented on top of this system and an extensive experimental campaign was conducted to capture the co-relation and influence of its various design parameters. The results proved that DRTRM can facilitate the run-time requirements of a high number of incoming applications, while optimizing the allocation of resources and diminishing the imposed communication overhead on all running distributed agents.

Chapter 5

Application-Arrival Aware DRTRM

5.1 Introduction

The nature of distributed decision making does come with an increased complexity in the resource management process. The lack of a single point with overview of the platform leads to limited ability to adjust in scenarios that the need for resources is stressed, since numerous distributed agents need to communicate via exchanged messages in order to enforce a global policy. In this Chapter, we identify this limited adaptivity ability by examining resource stressful scenarios, resulting from the arrival rate of incoming applications on many-core systems. More precisely, The evaluation of a very fast and resource hungry scenario of incoming applications shows that it can be the breaking point for the efficiency of DRTRM.

The effects of the arrival rate of incoming execution requests have been widely investigated for different target systems, e.g. Cloud infrastructure [151], Map-Reduce Clusters [231] and many core-systems [179] but all solutions rely on centralized resource management, which allows for effective decision making under heavy input traffic. We differentiate from this concept by analysing and correlating the application arrival rates with the internal mechanisms of DRTRM and propose its extended application-arrival aware version, capable of dynamically adapting to stress-marked scenarios in a distributed manner.

By leveraging the hierarchy of different agents in DRTRM, as well as their different execution profile, this Chapter includes the following contributions:

- The design on an Application-Arrival aware Distributed Run-Time Resource Management framework for many-core systems is presented, which utilizes VFS techniques in order to enforce an application admission regulation policy in a purely distributed manner, requiring the communication of only a small subset of the system's cores.
- A second level of exploitation of the VFS techniques is achieved by meticulously mapping the parts of DRTRM on the system according to their computational requirements in order to maximize the energy consumption gains of the framework.
- The Application-arrival aware features are implemented as an extension of DRTRM presented in Chapter 4 and evaluated on Intel Single Chip Cloud Computer (SCC) [114].
- We provide an exploratory analysis of the different design knobs of the proposed framework in order to fine-tune its parameters, thus maximizing the gains in application execution latency as well as system-wide energy consumption.

More precisely the Chapter is outlined as follows. Section 5.2 focuses on the internal hierarchy of DRTRM agents, while the proposed admission control mechanisms are detailed in Section 5.3. Section 5.4 includes an extensive experimental analysis of the behaviour and efficiency of the proposed DRTRM solution on Intel SCC platform, while Section 5.5 concludes the Chapter.

5.2 Resource allocation hierarchy overview

The presented DRTRM scheme implicitly imposes a hierarchy to the different roles of PEs and the resource allocation flow. This hierarchy, presented in Fig. 5.1, creates dependencies between PEs, which will be the key element of the extended, Application-Arrival Aware DRTRM. At the top level (Level 1), Controller cores are the building blocks of information tracking and idle resources ownership. In Level 2, Manager cores supervise application execution and facilitate inter-application resource exchange, which is critical for the system to reach an optimized state, ensuring no starvation incidence for any application. Initial cores lie in Level 3 and temporarily acquire resources from the upper two levels, in their effort to initialize a new application.

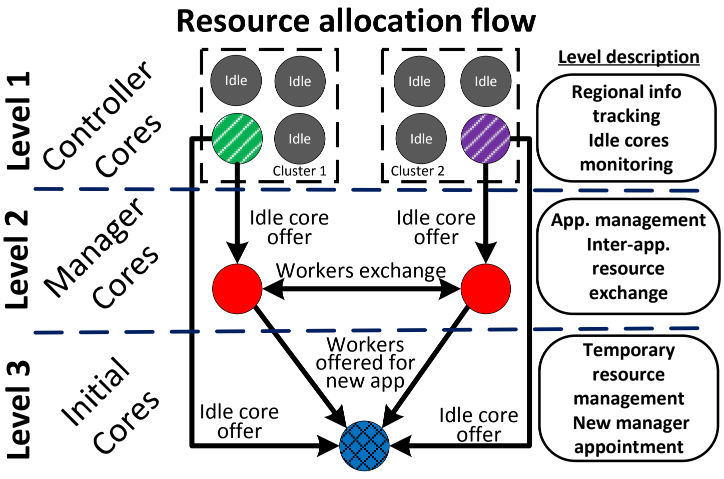

Resource allocation flow
Level 1
Controller Cores
Level 2
Manager Cores
Level 3
Initial Cores
Idle
Cluster 1
Cluster 2
Idle core offer
Workers exchange
Workers offered for new app
Level description
Regional info tracking Idle cores monitoring
App. management Inter-app. resource exchange
Temporary resource management New manager appointment

Figure 5.1: Cores hierarchy and dependencies in DRTRM.

In overall, the resource management hierarchy as well the appointment of dedicated tasks, either at design time, i.e. Controller cores or at run-time i.e. Manager, Initial cores, is mandatory to account for the lack of centralized system management. The inherent trade-off is the reduction of available workers, which we consider tolerable in future many-core systems with hundreds or thousands of PEs. In addition, the presented hierarchical scheme is favored with the intention to provide discrete and encapsulated function of distributed agents to allow them to incorporate custom run-time policies in future designs, tailored to the needs of different applications.

5.3 Adaptive and Distributed Application Admission

The envisioned system structure is presented in Fig. 5.2, consisting of i) the incoming application traffic according to differing arrival rate distributions and application characteristics, ii) an input application queue, iii) the DRTRM module responsible for applications' initialisation, resource management and execution on the many-core system and iv) an admission control module that regulates the dispatching of applications to be mapped onto the targeted platform. The proposed scheme implements a feedback loop approach, in an effort to configure DRTRM at run-time according to the rate of incoming applications and the intensity of system resources utilization by running ones.

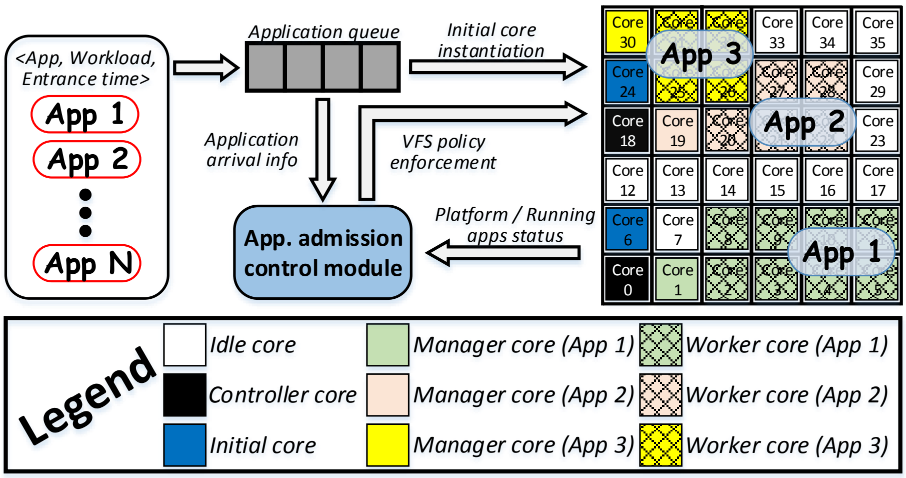


Figure 5.2: The proposed application-arrival aware DRTRM.

We define as $\lambda(t)$ the input rate of new applications on the queue of the processing system and as $\mu(t)$ the total rate of application conclusion and exit from the system. Rate μ_i, that each individual application concludes its operation is:

$$\mu_i = f(R_i, W_{comp_i}, W_{comm_i}) \tag{5.1}$$

ergo a function of its occupying computational resources R_i, the workload W_{comp_i} that the application has to execute and the amount of communication W_{comm_i} which has to perform with other applications on the system (mainly for resource exchange purposes). The correlation between the occupying resources of the application R_i and μ_i is proportional to the speedup of the application:

$$\mu_i \propto speedup(R_i) = \frac{Latency(R_i)}{Latency(1)} \tag{5.2}$$

The speedup of each application is calculated as the ratio of its estimated finish time when it occupies R_i resources and the corresponding values when it occupies only one resource. In the context of this work, these values are derived experimentally (see Section 4.6.3). In total, for the N_{apps} instantiated on the system at a given time t the total output rate μ is defined as:

$$\mu = \sum_{n=1}^{N_{apps}} \mu_i \tag{5.3}$$

The aim of the Application-Arrival Aware DRTRM is to service as many incoming applications as possible or in other words minimize the difference between incoming and outgoing application rate:

$$min\, Q(t) = \lambda(t) - \mu(t) \tag{5.4}$$

5.3.1 Effects of the incoming applications' rate on DRTRM

To motivate the necessity for a DRTRM with application-arrival aware characteristics, we quantify the resource management efficacy of DRTRM against different input application scenarios. A "Stressing" scenario was created, where the interval between successive applications is significantly small. Interval values were derived randomly to provide an unbiased input. This is compared against other application arrival scenarios, where the intervals of consecutive incoming applications derive from Poisson distributions with *Lambda* coefficient values equal to 16, 32, 48 and 64 respectively, as in [179]. The actual interval rate curves of these scenarios are presented in Fig. 5.3. X axis represents application id and Y axis is the interval between the i-th application and its following one. This form of application interval rate describes a system at which applications arrive at an almost steady rate but this rate sharply increases at a given point in time. The difference between the scenarios is the number of already admitted applications, when the arrival spike is observed.

Fig. 5.4 illustrates the application arrival moments of two scenarios, highlighting that applications in the "Stressing" scenario arrive much earlier. In Fig. 5.5, we quantify the impact of $\lambda(t)$ via the aforementioned scenarios, on a DRTRM **without** application-arrival aware characteristics, using the implemented matrix-vector multiplication applications on Intel SCC. Comparison of the "Stressing" to Poisson distribution derived scenarios, shows that total application execution latency exhibits a steep rise of 190% in average, despite the fact that the workload intensity of applications is identical in all scenarios. Additionally, the "Sum of applications' instantiation effort execution latency", which is a quantitative indication of the effort spent in order to introduce all applications into the system is averagely 142% higher. This indicates that despite the shortage of available computational resources, Initial cores kept on frequently searching cores for the new application, even though it was highly probable that no PE was available and would not be available until one of the running applications finished and freed its resources.

In correlation to Eq. 5.4, when $\lambda(t)$ is high, applications pile up on the system leading to a point when $N_{apps} \approx R_{tot}$, i.e. there are no sufficient resources for admission of new applications. Incoming applications are sent back to the queue and an initialisation process is restarted. This involves resources request from active applications, but since resources are scarce, workload execution is interrupted without any gain for any of the involved agents. Inevitably,

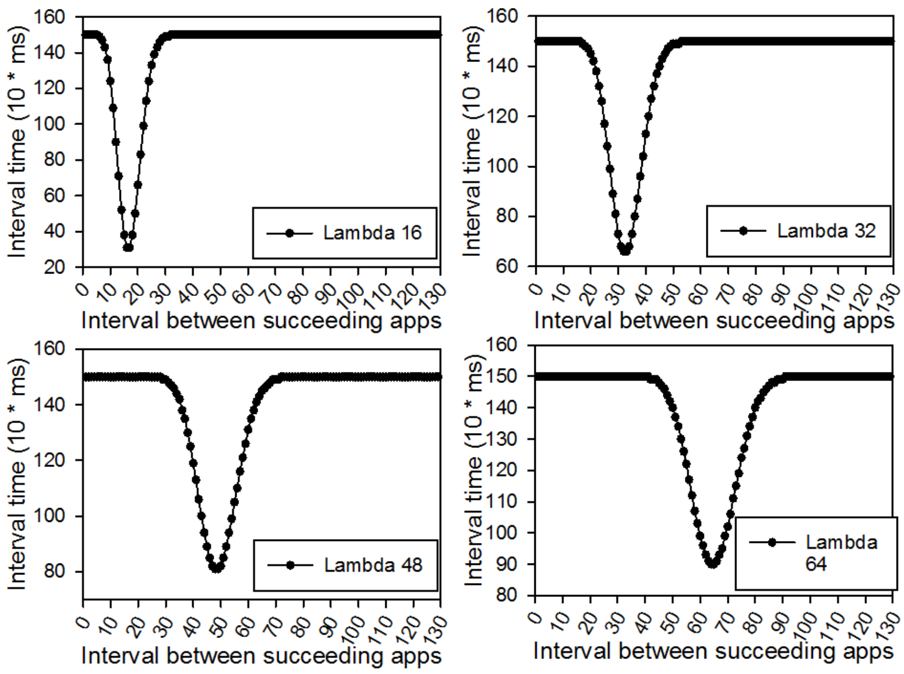


Figure 5.3: Interval rate of incoming application trace.

these interrupts result to an adverse effect on μ, since according to Eq. 5.1 the execution of an application is affected by W_{comm_i}, i.e. its communication with other applications (owned to resource bargaining in this case).

Note that a centralized resource management framework would suffer from similar to the aforementioned inefficiency issues, since the frequent remapping requests due to the high application arrival rates and the lack of distribution of this computational burden, would hinder the run-time adaptation and efficiency of the system.

5.3.2 Proposed Adaptation Scheme

The observed vicious cycle, should be detected and mitigated by a DRTRM able to adapt application admission to the status of resource demand and supply on the system. In a straightforward manner, such an adaptive behavior could be translated to a policy at the Initial core level, e.g. the repetition frequency of the core searching cycle could be adjusted according to system resource

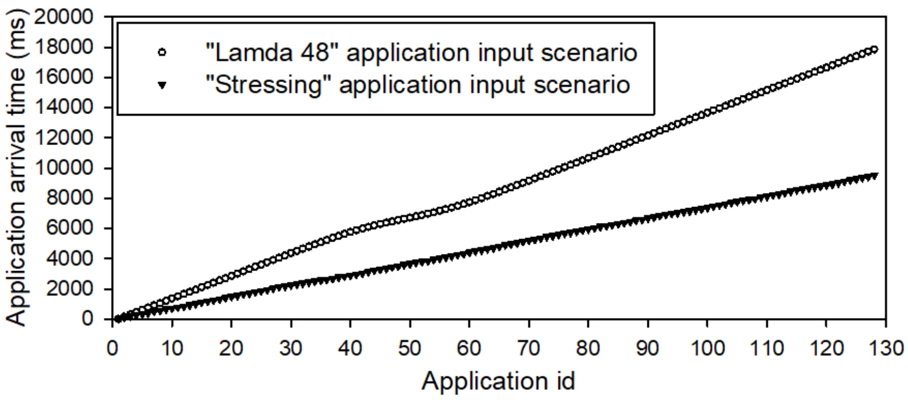


Figure 5.4: Arrival times of application trace.

utilization intensity. This adaptation mechanism stumbles upon the nature of purely distributed resource management, suffering from two major drawbacks:

- It requires a central decision to assure effective adaptation of the framework in respect to the incoming application demands. Consequently, this creates a central point of information acquisition, which is intended to be avoided in a distributed framework.
- It suffers from synchronization latency. Even if the designer decides to gather the necessary information and proceed to an adaptation decision, the gathering process followed by a broadcast to the cores could create a

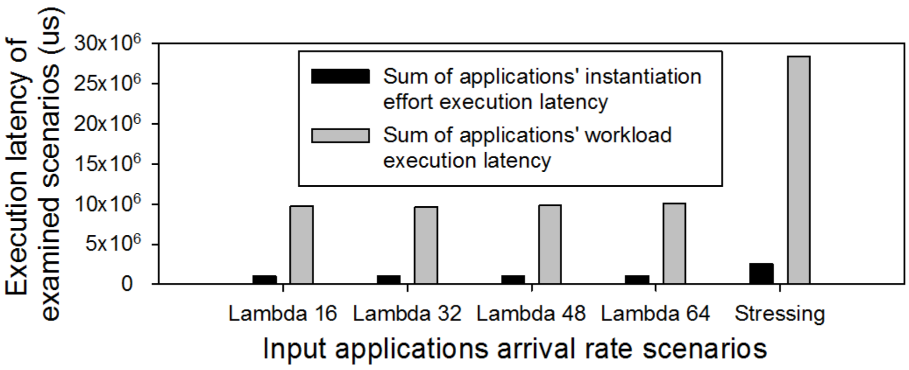


Figure 5.5: DRTRM behaviour for different input application arrival rate scenarios (No application-arrival aware features).

significant latency upon the enforcement of the new policy. Eventually, when the new policy is enforced, the state of the system might be different compared to when the gathering process started.

Our proposed distributed mitigation plan is to reduce the burden of pointless application instantiation efforts indirectly, by taking advantage of the resource allocation hierarchy of DRTRM (Fig. 5.1) in order to slow down the core search efforts of Initial and Manager cores, in case of heavy workload scenarios when available resources are scarce. Our simple yet effective application admission regulation policy, relies on decelerating Controller cores by reducing their operating frequency using Voltage and Frequency Scaling (VFS) techniques. In contrast to a centralized regulation policy, this requires co-ordination of only the Controller cores, which are very limited in number. As a consequence, the required decision making can be implemented in a distributed manner. In our VFS extended DRTRM scheme, application conclusion rate (Eq. 5.1) can be re-written as:

$$\mu_i = f(R_i, W_{comp_i}(f(R_i), V_{dd}(R_i)), W_{comm_i}(f(R_i), V_{dd}(R_i))) \tag{5.5}$$

indicating that both W_{comp_i} and W_{comm_i} are affected by the operational frequency and voltage of the utilized resources R_i of an application. Therefore, for the proposed mitigation plan to be effective, care is taken so that VFS in different PEs is such that:

- W_{comm_i} is reduced by avoiding unnecessary communication between new applications under instantiation requesting resources from running ones.
- W_{comp_i} remains unaffected by not applying VFS on resources executing computational workload. This is highly important and implies that worker cores remain at the highest possible frequency, ensuring that no latency overhead is imposed on applications.

Due to the dependencies of resource allocation hierarchy in DRTRM, the deceleration of Controller cores when system utilization is maximum leads to less frequent execution of temporarily redundant tasks of other agents, such as search for free resources. Thus, significantly less negotiation overhead is imposed on running applications, allowing them to proceed with workload execution uninterrupted. In this way, their execution is concluded faster and resources become available to facilitate the needs of new applications.

Conceptually, the interplay of cores in DRTRM follows/can be modelled as a client-server model, where clients are Initial/Manager cores and the servers are Controllers. Slowing down the servers leads to prolonged waiting times in

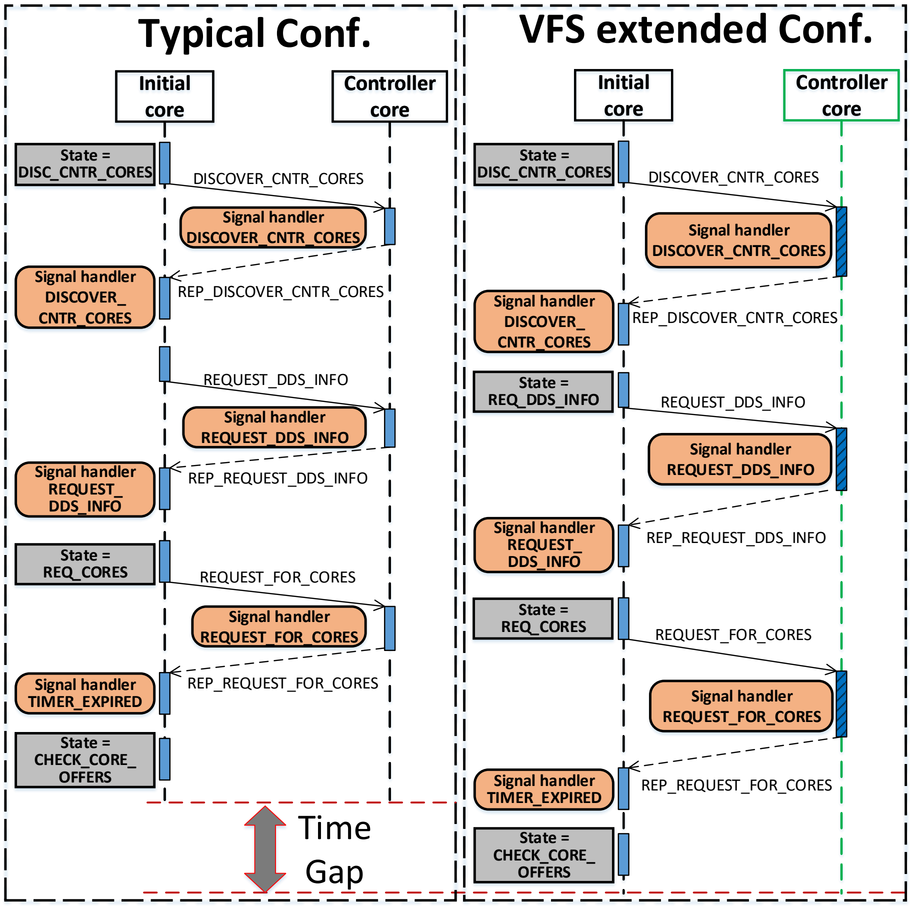

Typical Conf.
Initial core
Controller core
State = DISC_CNTR_CORES
DISCOVER_CNTR_CORES
Signal handler DISCOVER_CNTR_CORES
Signal handler DISCOVER_ CNTR_CORES
REP_DISCOVER_CNTR_CORES
REQUEST_DDS_INFO
Signal handler REQUEST_DDS_INFO
Signal handler REQUEST_ DDS_INFO
REP_REQUEST_DDS_INFO
State = REQ_CORES
REQUEST_FOR_CORES
Signal handler REQUEST_FOR_CORES
Signal handler TIMER_EXPIRED
REP_REQUEST_FOR_CORES
State = CHECK_CORE_ OFFERS
Time Gap
VFS extended Conf.
Initial core
Controller core
State = DISC_CNTR_CORES
DISCOVER_CNTR_CORES
Signal handler DISCOVER_CNTR_CORES
Signal handler DISCOVER_ CNTR_CORES
REP_DISCOVER_CNTR_CORES
State = REQ_DDS_INFO
REQUEST_DDS_INFO
Signal handler REQUEST_DDS_INFO
Signal handler REQUEST_ DDS_INFO
REP_REQUEST_DDS_INFO
State = REQ_CORES
REQUEST_FOR_CORES
Signal handler REQUEST_FOR_CORES
Signal handler TIMER_EXPIRED
REP_REQUEST_FOR_CORES
State = CHECK_CORE_ OFFERS

Figure 5.6: Time gap in RTRM signal exchange.

the client side, leading to increased latency for task completion. In turn, this creates longer control cycles which reduce the pointless application instantiation efforts.

Fig. 5.6 zooms in the communication between a Controller and an Initial/Manager core, showing in more detail how slowing down the operations of the first eventually results in slowing down the function of the others. The Initial core executes its search for resources, which is highly dependent on information acquired by the Controller core. The left side of Fig. 5.6 shows the evolution in time of their communication in a typical operating frequency configuration. The right side illustrates the timing for the execution of the

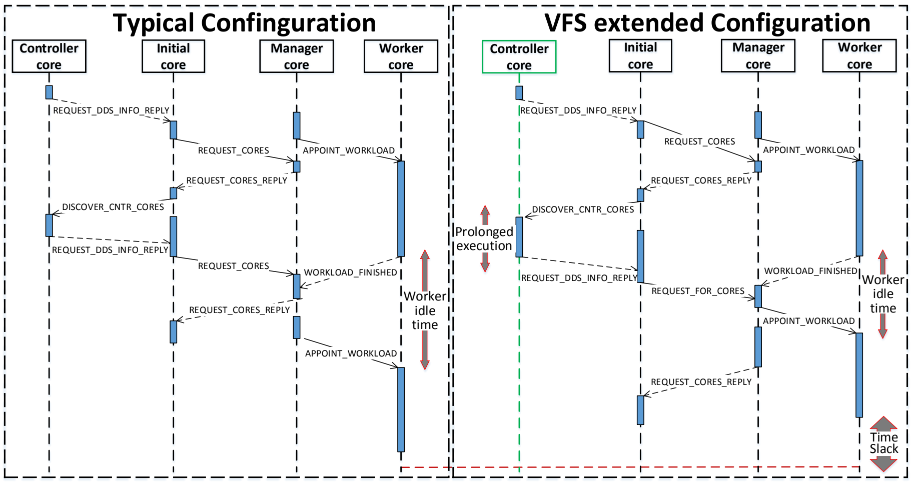


Figure 5.7: VFS induced execution slack.

same operations after the frequency of the Controller core has been scaled down. Since it operates on lower frequency, more time is required for its incoming request to be served. These requests create blocking points in the execution of the Initial core and indirectly its required time to execute a full resource search cycle is prolonged. This induced time gap slows down the instantiation of one new application and by scaling up to all Initial cores, it leads to reduced instantiation rate for all new applications.

Fig. 5.7, presents how all cores categories are affected by the VFS enabled DRTRM. Again, we see the interplay of different cores prior to and after VFS. The left side depicts the internal functionality of a Manager core, which distributes workload to its worker. Whenever the worker core finishes its calculations, it informs the Manager in order to decide how to further allocate application workload. If at that moment the Manager is occupied with serving other incoming requests such as a core request by an Initial core, then the elapsed time until workload re-distribution is increased and the worker remains idle for more time.

The VFS extended DRTRM, achieves more efficient workers utilization by avoiding the overburdening of Manager cores by incoming resource negotiation requests. As described, Controller cores of reduced operation frequency, stall the operation of Initial cores resulting in the generation of less requests for Managers. Consequently, a Manager core is more responsive when its workers notify their workload completion, which significantly reduces their idle time and eventually leads to lower application execution latency. The reader should note that in an actual stressful scenario, the inefficiencies presented in Fig. 5.7 are aggravated since the Manager core is flooded by resource requests from numerous Initial cores. This results to a queue of requests, required to be served before re-allocating workload to the workers.

A system with tunable sleeping intervals of the Controller cores would have comparable results to VFS technique with regard to the admission control mechanisms. However, it requires a fine-grained, run-time tuning mechanism, adaptive to the number of running applications, thus introducing synchronization overheads. In addition, the adoption of tunable sleep calls reduces energy efficiency, since there is no guarantee that the sleep duration is sufficient to trigger a low-power system state.

5.4 Experimental Evaluation

This section, presents the results of our conducted experiments regarding the limits, efficiency, extensions and robustness of our proposed methodology. For consistency purposes, **all experiments have been conducted using the "Stressing" application arrival scenario**, which we consider as the point of interest of our work. Scenarios of lower arrival intensity are sufficiently handled by DRTRM, while scenarios of higher intensity greatly surpass the capabilities of the target platform and are considered unrealistic. **The referenced implemented parallel malleable application corresponds to the Matrix Vector Multiplication application**, which was chosen as the most computationally demanding of all implemented applications. The evaluated metrics quantify the performance of the proposed framework as well as its energy savings resulting from the novel use of VFS techniques.

5.4.1 Implementation details on Intel SCC

Intel Single Chip Cloud Computer [115], described in Section 4.6.2, was utilized as the driver many-core platform for evaluating the proposed framework. The processing tiles of the platform are divided into 6 Voltage Islands (V.I.) and a Voltage Regulator Controller (VRC), provides the ability to regulate the voltage of each island individually. It is also possible to regulate the operating frequency at tile granularity and always within limits dictated by the operation voltage.

The RCCE library [158] of Intel SCC, exposes an API to allow the programmer to safely work with the VFS capabilities of the system, offering the ability to scale the frequency and voltage of an Island. This is achieved by dictating a frequency/voltage divider pair per Voltage Island, ranging from <800MHz, 1.1V> down to <100MHz, 0.7V>. Table 5.1 summarizes the available dividers combined with the supported voltage and frequency levels per Voltage Island.

Table 5.1: Voltage Island VFS configurations on Intel SCC.

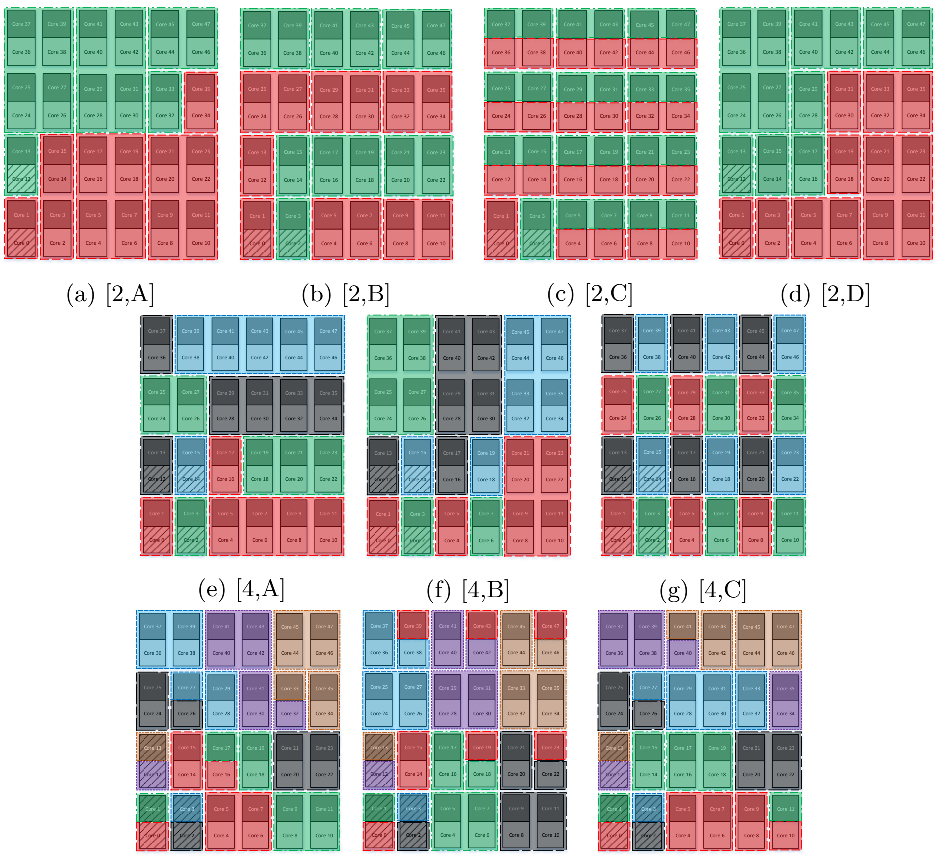
V.I. frequency divider	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
V.I. Voltage	1.1	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
V.I. cores' frequency	800	533	400	320	266	228	200	178	160	145	133	123	114	106	100

The SCC platform incorporates a power metering infrastructure, enabling the reporting of instant voltage and current values drawn by the many-core chip. We developed a custom power metering daemon program, which samples the power metering registers by periodically invoking the *"sccBmc -c status"* command. The gathered values are then numerically integrated over each time interval i, to calculate the dissipated energy: $E = \sum V_i \times I_i \times \Delta t_i$ to acquire the sum of the consumed energy. A similar infrastructure has been used also in [32] to examine the impact of DVFS decisions on the execution of single instances of MPI-based applications mapped onto the SCC.

An instance of DRTRM supervises each PE, operating at the user-space level of its software stack. Taking into consideration the platform's VRC architecture of pre-configured voltage islands, we enforce a grouping of the Controller cores onto a specific voltage island. In this way, frequency scaling can be also combined with voltage scaling to further reduce the power consumption of the DRTRM infrastructure.

As a far as *Clusters* are concerned, we examine all configurations of 2, 4 and 6 Controller cores presented in Fig. 5.8, chosen to include both coarse and fine-grained topologies with only constraint the placement of Controller cores inside V.I. 0, in order to simultaneously regulate their voltage. However, in the general case, DRTRM can support any kind of user-defined topology.

Since the regulation of a Voltage Island affects a group of PEs, the application mapping directives of the VFS enabled DRTRM have been meticulously tweaked to avoid the mapping of a worker core inside a region of reduced frequency in order guarantee that application execution is not hindered. On the contrary, its hierarchical scheme, described in Section 5.2, enables the mapping of Manager cores on low power PEs because they do not execute computational workload but



(a) [2,A] (b) [2,B] (c) [2,C] (d) [2,D]

(e) [4,A] (f) [4,B] (g) [4,C]

(h) [6,A] (i) [6,B] (j) [6,C]

Figure 5.8: Examined cluster topologies [#Controller cores, Cluster conf.]. Regions of the same color belong to the same cluster. Controller cores are shaded.

only orchestrate intra-application workload distribution and inter-application resource bargaining.

5.4.2 Performance-power gains of admission control

The first set of experiments evaluates the efficiency of the proposed admission policy to diminish the congestion created on the many-core system by the "Stressing" application arrival scenario, using the implemented malleable

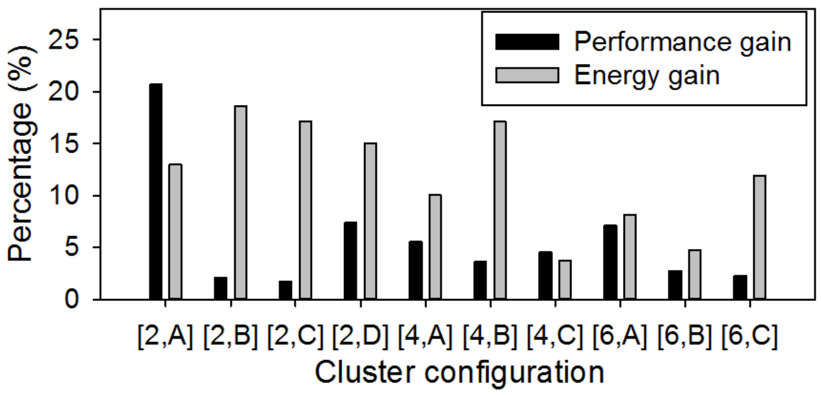


Figure 5.9: Performance-energy gains from application admission control.

applications as input. All Controllers are mapped on V.I. 0 of Intel SCC. Fig. 5.9 presents performance-energy metrics of DRTRM configurations with 2, 4 and 6 Controllers cores. Their operating frequency is dropped from 800 MHz to 533 MHz via a voltage drop from 1.1 V to 0.8 V in V.I. 0.

Results are expressed in normalized gain with respect to the same DRTRM topology without any voltage-frequency scaling of the Controllers. As shown, in all cases performance and energy improvements are reported. The results exhibit a lack of symmetry between the improvement in performance compared to energy. For example, in configuration [2,A] (Fig. 5.8a) there is a 20% improvement in performance accompanied by a 12% reduction in the amount of energy, whereas in configuration [4,B] (Fig. 5.8f) the respective numbers are 3% and 18%. This is because the utilized performance metric does not indicate the degree of concurrent execution of different applications on the system. This degree severely affects the required time for each experiment to be completed and thus its requirements in energy. Therefore, on the one hand in configuration [2,A], applications acquired more working cores, their summed execution time was small but they were executed in a more "serialized" way, thus energy gains are smaller. On the other hand, in configuration [4,B] applications are executed in a more concurrent manner meaning that they possess less cores in average. However, the experiment is concluded faster in total, which accounts for the high energy gains.

A delicate point of our design is the correlation of frequency reduction to the resource management efficiency. We emphasize that the proposed application admission policy is very effective in stressful application service requests, i.e. cases when the system load is consuming almost all hardware resources.

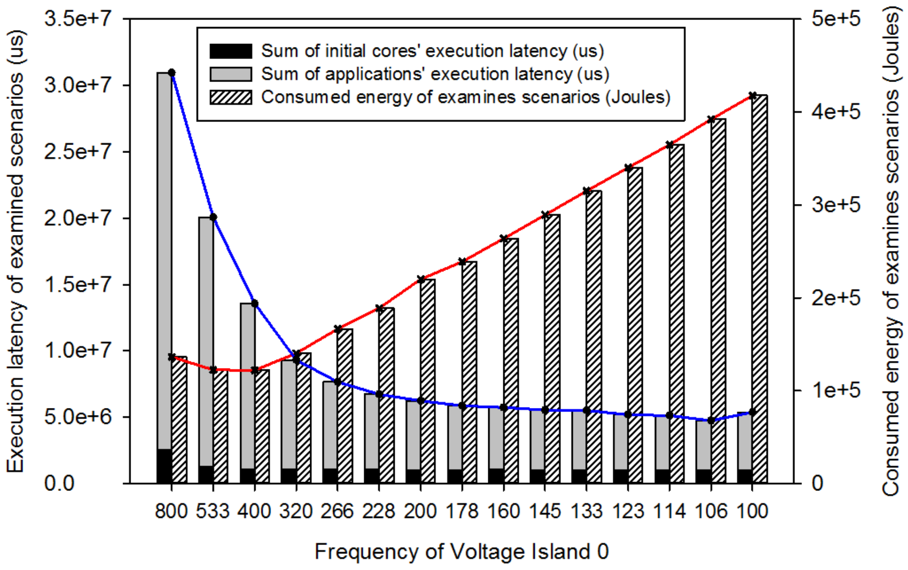


Figure 5.10: DRTRM behaviour for all frequency dividers of Voltage Island 0.

Furthermore, the reduction of the operating frequency of the Controller cores is a critical design parameter and should not be chosen arbitrarily. Based on that, our next experiments evaluates the behavior of the admission policy extended DRTRM for all operating frequencies provided by Intel SCC.

The experiment, involves only *Cluster* topology [2,A] (Fig. 5.8a) of 2 Controller cores. For this DRTRM configuration, all the possible values of Controllers operating frequencies are tested. Fig. 5.10 reports the measured metrics, which expose an interesting performance-energy correlation. Decreasing the operating frequency of Controller cores results in reduced total application execution latency. However, the constant decrease of latency does not always imply more efficient resource allocation. Application initialization on the system is highly related to Controller cores' operation and when their computational capabilities are excessively reduced, this initialization is performed in a slow, almost "serialized" rate. This low rate ensures that when new applications are instantiated, they are offered their maximum number of workers since few other applications occupy resources and thus their execution time is minimized.

Nevertheless, this "serialized" execution profile, results in increased running time for each experiment given that very few applications are executed in parallel. This in turn increases the consumed energy required for the experiment to be concluded. This trend is highlighted on Fig. 5.10 with red line for energy

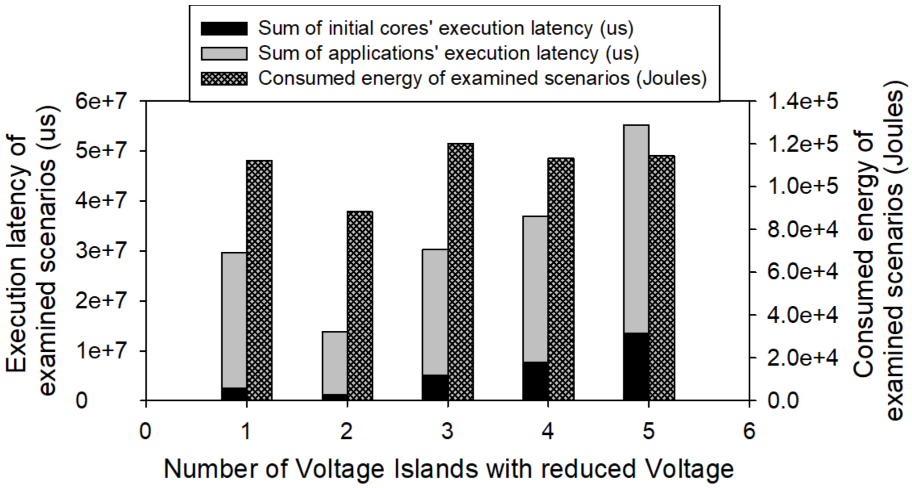


Figure 5.11: DRTRM performance for increasing Islands of reduced Voltage.

consumption and blue line for execution latency. The important outcome of this study is that a careless decision of VFS policy can have an adverse effect on DRTRM efficiency. On the contrary, meticulous choice of operating frequencies can give the system designer the freedom to sacrifice concurrent application execution for small energy savings and vice versa.

Symmetrically to investigating DRTRM behaviour under all available Controller cores' operating frequencies, we experiment with increased number of islands of reduced voltage. This gives the opportunity to the proposed DRTRM to utilize VFS in a more common way, i.e. to reduce power consumption of some parts of the system that are not too computationally intensive, such as Manager cores.

A set of experiments were performed, with more than one V.I. of Intel SCC operating on reduced voltage. Our design constraint is that no worker core will be mapped on these islands, in order not to stall workload execution. Results are presented in Fig. 5.11 including metrics about latency and energy consumption of the various configurations.

We observe that having two islands of reduced voltage leads to high gains, both in execution latency and consumed energy of the system. Further increase in the number of reduced voltage islands leads to prolonged application execution latency. This is attributed mainly to the fact that a large number of cores cannot be utilized to execute workload (worker cores) and thus applications have few resources. This increased latency, results to prolonged activity on the system and consequent increase in its consumed energy. In conclusion, this analysis motivates a further investigation of DRTRM configurations of up to 2 reduced voltage islands aiming at fine tuning the rest of DRTRM design options.

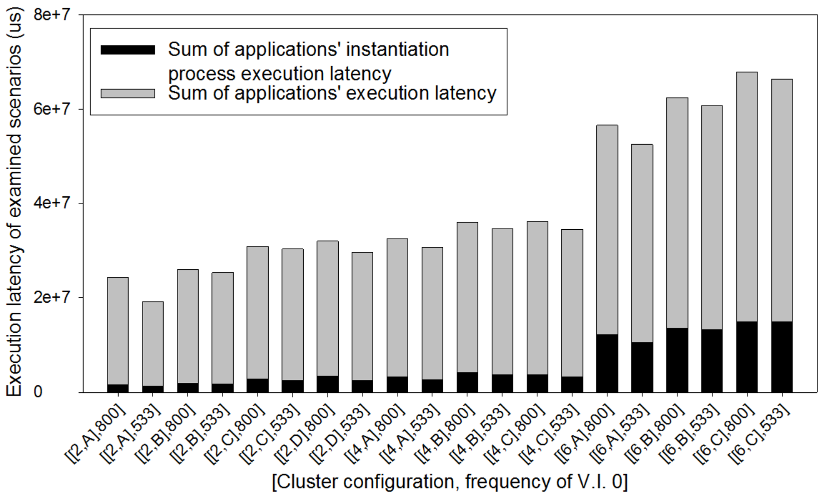
5.4.3 Exploratory analysis of the DRTRM parameters

Configurations of one island with reduced voltage

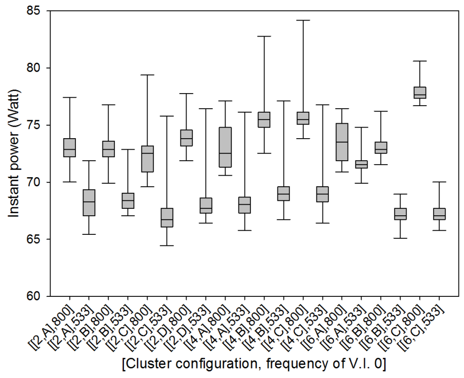
In order to evaluate the combined effects of the design parameters of DRTRM and our proposed methodology, we perform a large exploration campaign over the different *Cluster* topologies. Fig. 5.12 summarizes the results in terms of (a) the total execution latency for all applications to be initiated (Sum of Initial cores execution latency) and executed, (b) distribution of instant power consumption of Intel SCC (c) total consumed energy, (d) total number of exchanged messages, (f) size of exchanged messages. In all diagrams, the X-axis is a tuple of the examined *Cluster* configuration (Fig. 5.8) and the frequency of Controller cores. For example, tuple [[2,A],800] means *Cluster* configuration [2,A] (Fig. 5.8a) with 2 Controller cores operating at 800 MHz. Tuple [[2,A],533] is about the same *Cluster* topology with Controller cores at 533 MHz. Frequency of 800MHz implies that no admission control is performed, while frequency of 533MHz implies that admission control is active and worker cores are mapped outside of the island of reduced voltage.

Regarding system performance, Fig. 5.12a validates that a high number of Controller cores in SCC platform results in increased latency for both applications' instantiation and execution. As far as the proposed admission control policy is concerned, it is shown that it results to lower latency in all cases, enabling performance optimization of 6%, in average. The combined energy consumption gains rise up to 12% (Fig. 5.12c). Additionally, we observe in Fig. 5.12b that the proposed admission control scheme leads to better power distribution both in terms of robustness (the 25- and 75-quantiles in cases of 533MHz are consistently closer than in the case of 800Mhz) and peak power, where gains of averagely 6% are reported. Power distribution for cluster topologies with increased number of Controller cores ([6,B] and [6,C]), exhibits a more robust trace given that the incoming workload is distributed in an even manner across system resources due to the small size of *Clusters*.

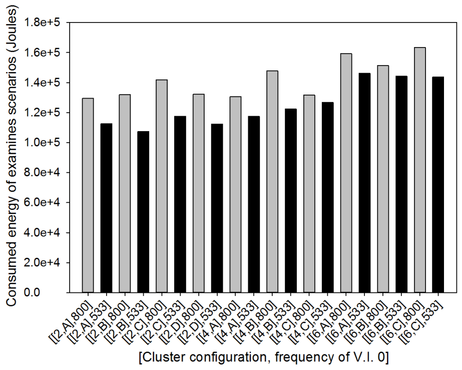
As shown in Fig 5.12d, increased system performance is correlated to the number of exchanged messages. For every *Cluster* topology, the proposed admission control policy results in reduced number of exchanged messages, which validates its intended goal of regulating the intensity of core search operations. Regarding the total size of exchanged messages (Fig. 5.12e), the trend is the same as in their total number, but the actual values are not proportional since the size



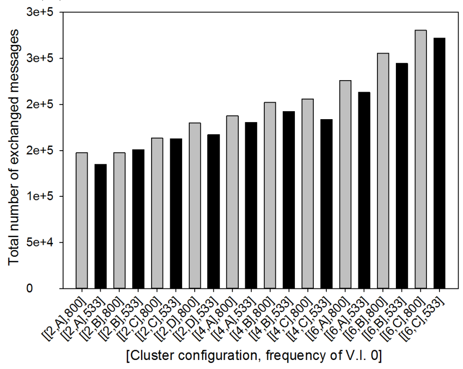
(a) Measured execution latencies.



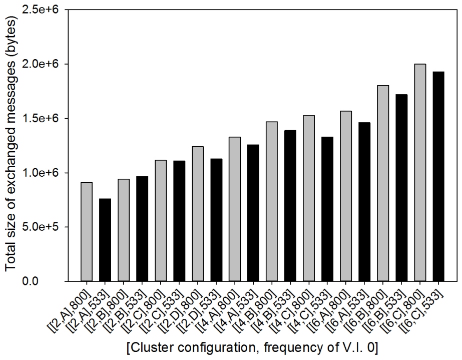
(b) Instant consumed power range.



(c) Total consumed energy.



(d) Total exchanged messages.



(e) Total exchanged message size.

Figure 5.12: Measured values for all configurations (Cluster topology & operation frequency of Voltage Island 0).

of each message varies according to its type. For example, an offer for cores from one Manager to another is a few bytes long since it involves the ids of the offered cores, while the corresponding reply message is only one byte long to indicate the acceptance/rejection of the offer.

To summarize, our experiments provide an experimentally derived proof that the proposed application arrival aware DRTRM is effective in mitigating the "Stressful" system state regardless of the chosen Cluster configuration. Nevertheless, careful choice of the Cluster configuration can maximize the achieved gains, which are not limited to performance metrics but include power and energy consumption as well as communication traffic on the system.

The experiment is repeated for configurations of 2 Controller cores, using the malleable application model presented in Section 4.6.3, to validate the efficiency of our proposed methodology using a more diverse mix of applications. Each application is characterized by its parallelism variance σ and average parallelism A, which are provided as input to DRTRM. The workload values W are maintained identical with the ones used for the Matrix Multiplication application and the experiment is executed on Intel SCC. One workload round equals to a time delay scaled according to the speedup of the application, which is calculated with respect to its characteristics and the number of its worker cores. To create the workload mix, different values of application parameters have been randomly chosen, ranging from 0.01 to 100 for σ and from 2 to 16 for A. Applications with low σ exhibit steep scaling with high speedup for the majority of A values. Applications with high σ have smooth scaling characteristics but do not achieve as great speedup, even for high A values.

The presented results in Fig. 5.13, validate the efficiency of our proposed application arrival aware DRTRM, which manages to reduce the total execution latency of applications by 43.8% and the required latency for application admission by 472% in average, in comparison to the original DRTRM. The higher gains are attributed to the mixed scaling characteristics of applications, enabling the admission control policy to provide many worker cores to high scaling applications and thus conclude their operation much faster. This in turn leads to faster release of their occupied resources, thus allowing the queued incoming applications to locate working cores with highly reduced effort, due to the lack of congestion on the many-core system.

Configurations of two islands with reduced voltage

The experiments illustrated in Fig. 5.14 elaborate on the efficiency of DRTRM in a system with two islands of reduced voltage, inside which Manager cores are mapped. Performance and energy metrics are provided, for configurations

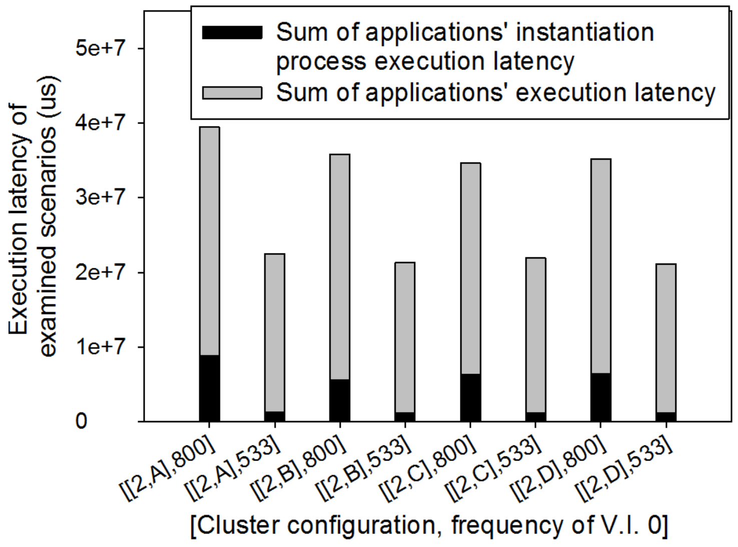
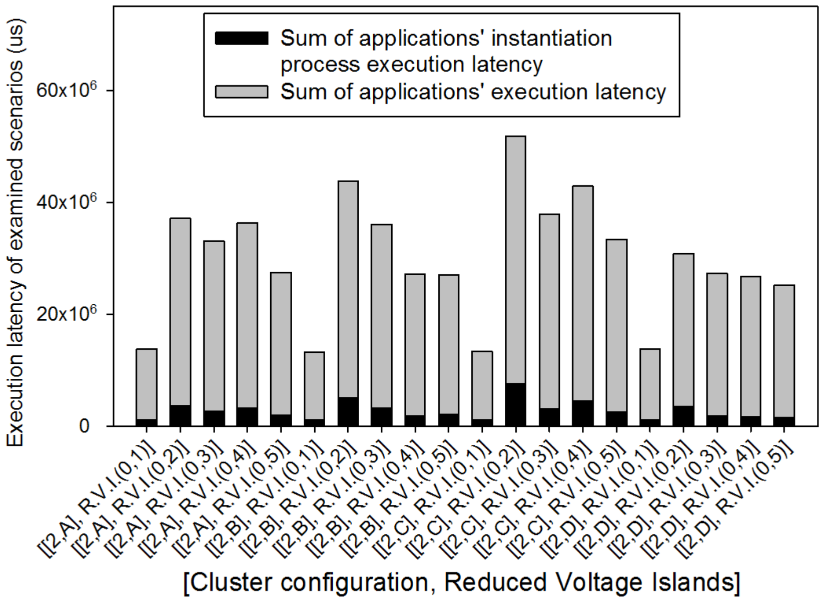


Figure 5.13: Comparison of configurations for model based malleable applications (Cluster topology & frequency of Voltage Island 0).

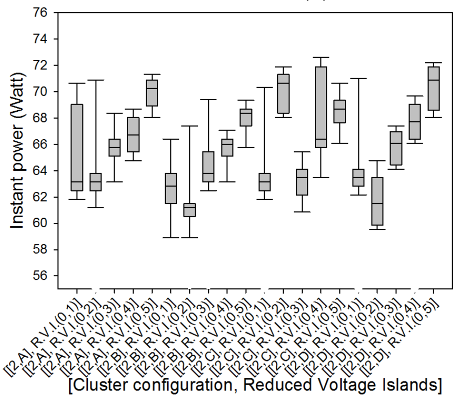
of 2 Controller cores given that they fared better in the previous experiments. The X-axis of the plots includes tuples of the examined *Cluster* configuration (Fig. 5.8) and the ids of the islands of reduced voltage. For example, tuple [[2,A],R.V.I.(0.1)] implies *Cluster* configuration [2,A] (Fig. 5.8a) and Voltage Islands 0 and 1 operating on 0.8V at 533MHz.

The overall trend in results is that all configurations with V.I. 0 and 1 diminished in voltage, lead to significantly better system performance compared to all other combinations of reduced V.I.s of the same *Cluster* configuration. This is because Controller and Manager cores are mapped in close proximity and thus their communication is faster and more efficient. Additionally, V.I. 1 is the access point for communication of Intel SCC with external non-volatile storage, where log files of the incoming applications are stored. As a consequence, I/O operations are sped up leading to highly reduced execution time for Manager cores. In turn, this leads to less energy consumption, compared to all other configurations of the same *Cluster* topology.

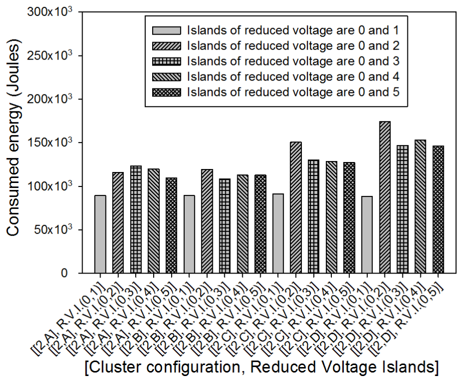
The rest of the combinations of V.I.s do not result in enhanced system performance and significant gains in consumed energy, if any. Amongst them, the one with reduced voltage in islands 0 and 5 fares better, which can be attributed to less communication traffic in the center of the platform, which uses XY-routing in its mesh. This reduced congestion in the center of the platform



(a) Measured execution latency.



(b) Instant consumed power range.



(c) Total consumed energy.

Figure 5.14: Measured values for all configurations (Clusters of 2 Controller cores + Voltage Islands with reduced voltage).

enables better communication between Controllers and the rest of the cores and this in turn leads to small performance improvements.

The best *Cluster* configuration in terms of performance is [2,C] (Fig. 5.8c), which

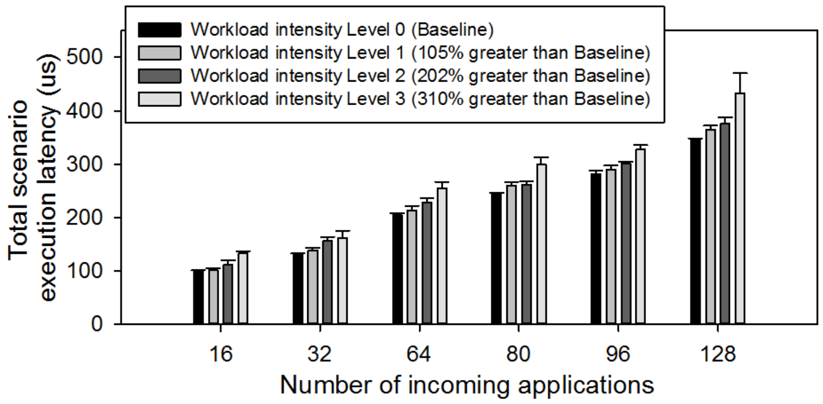


Figure 5.15: Different examined Matrix Multiplication application workloads (Mean value, standard deviation format).

slightly prevails over all others. Interestingly this topology is highly fragmented in the sense that *Clusters* include small areas spread throughout the platform. An example contrary topology is that of configuration [2,D] (Fig 5.8d), where *Clusters* are two big continuous areas. The difference in performance can be explained on two grounds. First, the fragmentation of *Clusters*, leads to highly dispersed application mapping over the platform. Given that the examined workload does not imply communication among worker nodes, this disperse mapping reduces the probability of their memory operations being congested, especially for accesses to off-chip shared DRAM. Secondarily, Initial cores are randomly distributed at run-time in a round robin fashion inside different *Clusters*. As a result, high *Cluster* fragmentation increases the probability of more widespread Initial core assignment and thus more evenly balanced core allocation in applications. In conclusion, the amalgamation of common VFS techniques in our application aware DRTRM, can optimize system performance and lead to reduced energy consumption. However, this can only be achieved by careful choice of DRTRM parameters both at design-time (e.g. Cluster configuration) and run-time (e.g. VFS topology and Managers mapping policy).

Robustness against workload scalability

In this Section the robustness of the proposed DRTRM scheme is evaluated against scaled workloads, using the configuration [2,A] at 533 MHz. In each experiment the number of incoming applications and the intensity of their workload requirements varies. The workload W values were generated using

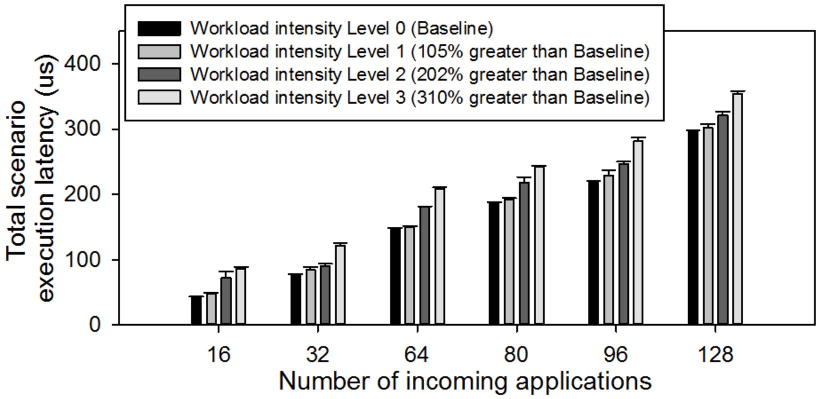


Figure 5.16: Different examined model based application workloads (Mean value, standard deviation format).

a random number generation function based on Poisson distribution. Four different levels of workload intensity were created using values 16, 32, 48, 64 as the mean value of the random generator. The workload escalation of these four levels is provided in the legend of Figs. 5.15 and 5.16. The variation in workload was combined with an ascending number of incoming applications ranging from 16 to 128.

In the first experiment, the implemented malleable application (Section 4.6.3) is utilized and Fig. 5.15 presents the total execution latency of each examined input workload combination. Results show a close to linear scaling in latency for ascending number of incoming applications and workload intensity levels. In addition, in all cases the deviation from the respective mean value is very small and no unexpected behavior is observed, e.g. spikes in the execution latency. It is also important to take into account the noise (expressed through variations in latency) injected in the measured values due to the layered software stack of each SCC core (DRTRM instance - Linux OS - SCC drivers). The above observations qualify the proposed DRTRM scheme as robust in terms of the required average execution latency with respect to the examined workload related parameters.

The experiment is repeated for the mix of applications created using the malleable application model as described in Section 5.4.3. While the characteristics of the input applications differ, their workload requirements are identical to the ones of Fig. 5.15. Fig. 5.16 illustrates the required latency for the execution of the different scenarios of the model based workload mix. We observe that the framework maintains its robustness in handling applications

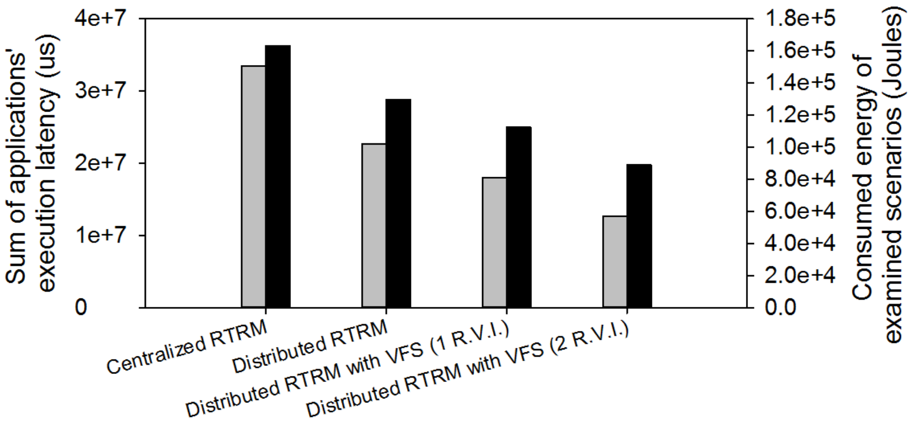


Figure 5.17: Comparison of the different RTRM schemes.

of increasing workload requirements, despite of the higher diversity of their scaling characteristics. Compared to the respective results for the implemented application (Fig. 5.15), the deviation of the measured values is much smaller since the lack of workload computations leads to less latency variations and more deterministic interplay between the distributed agents during the execution of each input scenario.

5.4.4 Comparative evaluation of different RTRM schemes

Finally, we present a combined comparative analysis of the different types of resource management schemes presented throughout this work. More specifically, we compare i) a centralized RTRM, where one agent monitors the entire system and aids inter-application negotiations for resources, ii) a Distributed RTRM without application aware admission control and iii) the proposed VFS extended DRTRM in versions of one or two reduced voltage islands. Examined scenarios involve the execution of 128 application arriving at the rate dictated by the "Stressful" scenario. Results are presented in Fig. 5.17 in terms of total application execution latency and system-wide consumed energy.

The worst RTRM regarding application execution latency is the centralized one, since in SCC a single core is not competent enough to handle the high amount of resource allocation requests generated by the input rate of incoming applications. This inefficiency is translated to elevated consumed energy of the centralized scheme. Increasing the number of RTRM related managerial agents as in the case of no VFS extended DRTRM results in much better resource management performance. However, the arrival rate of applications is still not taken into account, which leaves space for further gains when VFS extended version of DRTRM is applied. Having balanced the resource utilization under this stressful scenario, more gains are acquired in the VFS extended DRTRM with two reduced voltage islands, which maps Manager cores efficiently, taking advantage of their execution profile. In overall, when comparing the centralized approach to the best of the VFS extended one, gains of 62% in total application execution latency and 45% in consumed energy are exhibited.

5.5 Conclusions

This Chapter was focused on the design and implementation of an Application-Arrival Aware Distributed Run-Time Resource Management framework targeting many-core systems. The motivational observations of the work are based on the effect of different application arrival scenarios on the resource allocation efficiency of DRTRM. This analysis showed that under stressing conditions, i.e. a fully occupied system and high application arrival rate, DRTRM engages in futile search for resources, thus pointlessly interfering with running applications. To mitigate this inefficiency, a dynamic application admission regulation policy was proposed based on the internal hierarchical structure of decision making in DRTRM. More precisely, an operating voltage and frequency scaling strategy was proposed, that regulates application admission without degenerating its distributed nature.

The Application-Arrival Aware characteristics were implemented as an integral part of DRTRM on top of Intel SCC platform. The experimental analysis, which focused on the case of stressing application arrival scenarios, validated the efficiency of the proposed regulation policy in alleviating the system and boosting application execution, while reducing the total consumed energy of the platform. A profound outcome of the experimental analysis was that the utilization of VFS as a regulation mechanism is beneficial only under the stressing conditions and a careless configuration of the mechanism can lead to serious system degradation. Last but not least, further investigation showed that the VFS techniques can be utilized in traditional ways, optimizing energy consumption according to execution profile of the distributing agents.

Chapter 6

SoftRM: A Fault Tolerant DRTRM

6.1 Introduction

In the area of Distributed Run-Time Resource Management, the majority of works have targeted the optimization of metrics like system throughput or energy consumption [207, 206]. Nevertheless in order to successfully incorporate kilo-core systems to every-day user experience, the dependability of their operation becomes of critical importance. Projection studies indicate that due to the extreme transistor scaling in nano-scale many-core systems, the probability of operation variability and sub-system failures is highly elevated [202, 111]. This reliability deficiencies are attributed to inherent aging and wear-out mechanisms of the chip such as Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI) and Time Dependent Dielectric Breakdown (TDDB) [143]. Variability issues and faults can also occur and aggravate dynamically due to ambient or workload related thermal fluctuations as well as drops in the supply voltage of an integrated circuit [180].

The distributed nature of the targeted systems on both processing elements (PEs) and Resource Management imposes extra design requirements and increased complexity to provide fault tolerance guarantees in an online and timely manner. Therefore, it has been identified that in order to effectively mitigate variability issues in kilo-core SoCs, it is mandatory to intervene and leverage techniques for increased dependability in all layers of system design ranging from hardware [236] to high level application development [112, 180].

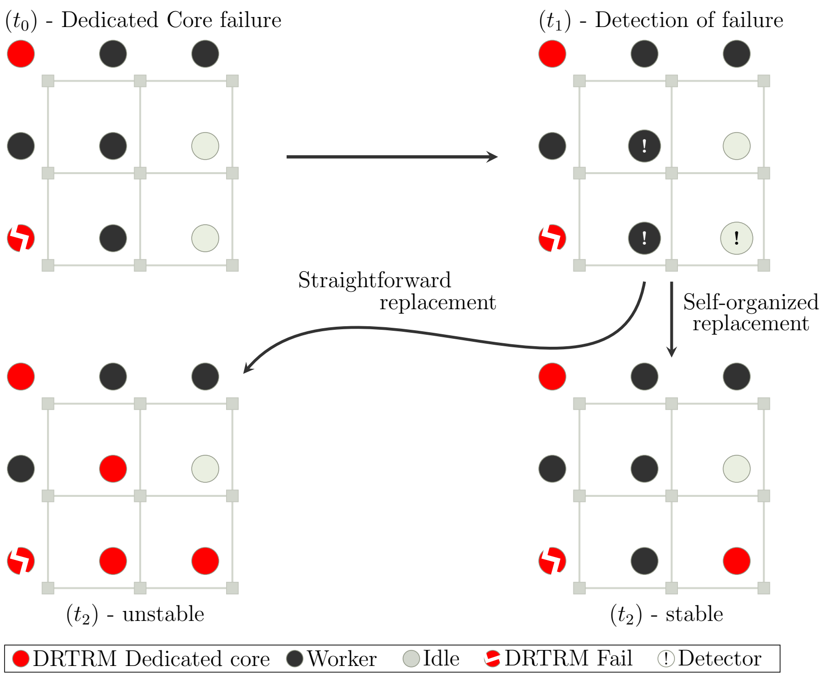

(t_0) - Dedicated Core failure
(t_1) - Detection of failure
Straightforward replacement
Self-organized replacement
(t_2) - unstable
(t_2) - stable
DRTRM Dedicated core
Worker
Idle
DRTRM Fail
Detector

Figure 6.1: Many-core system snapshots.

Fault and lifetime aware application mapping [147, 103, 238] is a first level of dependability extension. Proceeding to system level, fault tolerance is either achieved via centralized decision making [43, 112, 197] or via hierarchical designs which rely on spare PEs provisioning [59, 127, 229, 91]. The latter succeed at concurrent and reliable mapping of many applications but neglect to examine the possibility of failures on the higher level parts of the resource management hierarchy. Consequently, in this work we aim at bridging this design gap by proposing a self-organized, fault-tolerant run-time resource management framework, which integrates mechanisms able to detect and recover faults in every level of its design.

This design aspect is also critical from the point of view of Edge computing systems, as an many-core IoT Gateway facilitates teh execution of numerous IoT applications. Consequently, a possible erroneous operation can have negative impact on many users, while in certain cases, e.g. autonomous systems or medical applications, the degradation of functionality is unacceptable.

Motivation: We examine a NoC based many-core system (Fig. 6.1) governed by a DRTRM framework. The PEs in red indicate a number of dedicated cores, responsible for the correct operation of DRTRM. The rest of them are either idle (grey) or assigned to workload execution of an application (black). The system is at a stable state and all PEs and software stacks are in fine condition.

At a point t_0 in time the DRTRM SW stack of the bottom left core fails. The failure is permanent and can be attributed either to a HW fault or a non-recoverable software error. However, DRTRM cannot operate appropriately without a replacement to the failed core due the fact that its correct operation is vital to resource allocation decision making process. Since the system is fully distributed, there is no central point to determine the replacement core and communicate this decision. *As a consequence, there is the need of a self-organizing process to take place amongst the cores.*

This process should offer a unique outcome and guarantee that this outcome will be received and respected by all cores (as shown in Fig. 6.1 (t_2 - stable). A simple replacement tactic i.e. *"The first core that detects the failed one will act as its replacement"*, is bound to fail due to the distributed nature of the system. There is no guarantee that only one core will detect the failure and volunteer to be the replacement (see '!' at t_1 in Fig. 6.1). In this case, more than one cores will have the same responsibilities which will create collisions and lead to a new unstable system state (t_2 - unstable).

In summary, the focus of this work is to provide a solution and design alternative to the following question. *Given a many-core NoC based system managed in a hierarchical and distributed manner through assignment of distinct roles on different PEs (agents), is there a way to guarantee that a manifested, unrecoverable fault in any agent will be dynamically mitigated in a coordinated way, propagated to all and respected by all healthy PEs on the system?*

Towards this direction, the following contributions are presented in this Chapter:

- SoftRM, a fault tolerant distributed resource management framework is presented, which is able to dynamically react to failures of PEs, in a self-organized way.
- The recovery strategy is based on a consensus agreement algorithm enhanced with workload awareness to achieve an effective replacement policy.
- SoftRM is augmented with a failure detection algorithm, which takes advantage of the resource allocation related communication of PEs in order to significantly reduce the inflicted communication overhead of detection.
- SoftRM is implemented on top of DRTRM presented in Chapter 4 and evaluated on Intel Single Chip Cloud Computer (SCC) [114] in order to

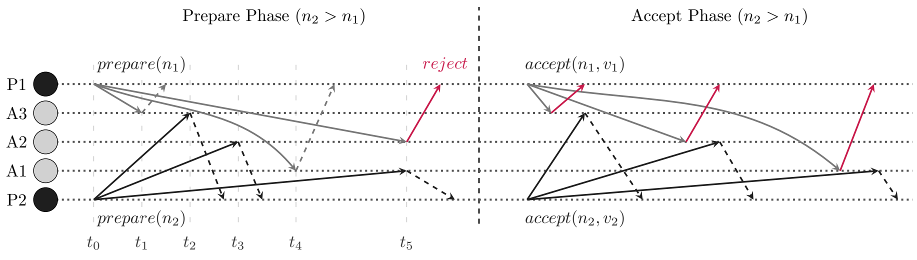


Figure 6.2: Prepare and Accept phases of Paxos Protocol with two Proposers P1 and P2 with proposal numbers n_1 and n_2 ($n_2 > n_1$).

capture the correlation of its design parameters to the efficiency of resource allocation and fault recovery actions. Comparison against state-of-the-art fault tolerance techniques highlights the advantages of self-optimization against spare core provisioning.

SoftRM is designed on a purely distributed concept, implementing a hierarchy of different roles of PEs to achieve resource allocation and negotiation between applications. Self-organization is preferred over PEs provisioning in order to fully utilize system resources and avoid design-time limits on the number of failures that can be tolerated. The hierarchical design of SoftRM allows its high level fault tolerant features to be cooperatively combined with other techniques for increased dependability, such as hardened PEs [121] or customized fault tolerant schemes per application [124].

The Chapter is organised as follows. Section 6.2 introduces PAXOS, the utilized algorithm for consensus agreement, while Section 6.3 details the employed error model. The design of SoftRM is presented in Section 6.4 by describing its key points, i.e. the workload-aware extended version of PAXOS (Section 6.4.1), its error detection infrastructure (Section 6.4.2) and its dynamic recovery mechanisms (Section 6.4.3). The experimental evaluation of SoftRM is presented in Section 6.5, while Section 6.6 concludes the paper.

6.2 PAXOS Consensus Protocol

The core component of the self-organized aspect of the SoftRM is a consensus protocol. It ensures that in a situation where different values are proposed by different processes, only one of them is chosen. Paxos, proposed by Lamport [138], is an algorithm for consensus achievement in a network of unreliable processors. The safety requirements for consensus achievement are (i) only a value that has been proposed may be chosen (*non-triviality*), (ii) only a single value is chosen (*safety*) and (iii) a process never learns that a value has been chosen unless it has actually been (*liveness*). Processes can have any of three different roles in Paxos; *proposers*, who propose values to be chosen, *acceptors*, who accept or reject the proposed values and *learners*, who will eventually learn the chosen value once it has been decided. In a general scenario, a single process may have multiple roles simultaneously. We summarize below the three phases of the complete flow of the protocol.

Prepare Phase: Each proposer picks a unique proposal number n greater than any n_{max} previously sent by any proposer and sends a *prepare* request with number n to all acceptors. On the acceptor side, if a *prepare* request is ever received with number n greater than that of any *prepare* request to which he has already responded ($n > n_{max}$), then he responds with the highest numbered proposal that he has accepted (if any). Additionally, he makes a promise not to accept any more proposals numbered less than n ($n_{max} \leftarrow n$). A request is rejected if the acceptor receives a *prepare* request with n lower than the highest n_{max} proposal number ever received.

Accept Phase: After a proposer has received a response to his *prepare* requests from the majority of acceptors, he sends an *accept* request to those acceptors with a value v. This value is either the highest numbered proposal among the responses received in Prepare phase, or any value if the responses reported no other proposals. On the acceptor side, if an accept message is received with a proposal number n, he accepts the proposal by replying with an *accepted* message, unless a higher proposal number n_{max} has been received in the *Prepare* phase.

Learn Phase: Similarly to the Accept phase, once a proposer has received *accepted* messages from the majority of acceptors, he realizes that his proposed value has been accepted and broadcasts a $learn(v)$ message to all learners.

Fig. 6.2 illustrates an example of the Paxos Protocol with two proposers P1, P2 (black nodes) and three acceptors A1, A2, A3 (grey nodes). At time t_0 ($t_0 < t_1 < ... < t_5$) both P1 and P2 send prepare messages to all acceptors with proposal numbers n_1 and n_2 (grey and black arrows respectively), with $n_1 < n_2$. At times t_1 and t_4 acceptors A3 and A1 receive the $prepare(n_1)$ from P1 whereas at t_3 acceptor A2 receives the $prepare(n_2)$ message from P2 and they promise **not to accept any other proposals with a lower proposal number**. Since this is the first time the acceptors receive a $prepare(n)$ message, they automatically reply with an accept message. Acceptors A3 and A1 receive the $prepare(n_2)$ from P2 at t_2 and t_5 respectively. They reply with an accept message since $n_2 > n_1$. However, when A2 receives the $prepare(n_1)$ from P1 this message is rejected (red arrow), due to the promise to P2 not to accept any messages with proposal number lower than n_2. In the Learn phase, v_2 proposed by P2 has been accepted by the majority of acceptors, so P2 broadcasts this value to all learners. In this work, we propose a version of the Paxos algorithm which takes into account the workload status of different processors when a consensus agreement effort is made.

6.3 Error model

Exhibited errors in many-core systems are broadly categorized as permanent, intermittent and transient. Permanent faults result in sub-component malfunction which is not restored during run-time of the system. On the contrary, intermittent faults regard errors which occur repeatedly, interchanged with periods when the system is fault free, while transient faults are temporary.

Without loss of generality, we demonstrate the properties of our proposed design focusing on permanent faults. We focus on faults manifested on PEs and do not examine faults on the links of the target NoC based system. We model the probability of permanent fault on a PE of the system using a Weibull distribution [131] with Probability Density Function (PDF) defined as:

$$f(t) = \frac{\beta t^{\beta-1}}{\eta^{\beta}} \mathrm{e}^{-(t/\eta)^{\beta}}, t \geq 0 \tag{6.1}$$

where η and β are the scale and shape parameters respectively. According to Eq. 6.1, the reliability of the system is defined as:

$$R(t) = \int_t^{\infty} f(t)\,\mathrm{d}t = \mathrm{e}^{(-t/\eta)^{\beta}} \tag{6.2}$$

Additionally, the failure rate of an individual component is:

$$\lambda(t) = \frac{\beta}{\eta}(t/\eta)^{\beta-1} \tag{6.3}$$

The correlation between the parameters of the PDF is presented in Eq. 6.4, associated with Mean Time To Failure (MTTF) and Γ function. Assuming an MTTF of some years it follows that:

$$\eta = \frac{MTTF}{\Gamma(1+\frac{1}{\beta})} \tag{6.4}$$

To further enhance the utilized error model, we refrain from using a constant error rate but adjust the parameters of the error PDF according to the simulated

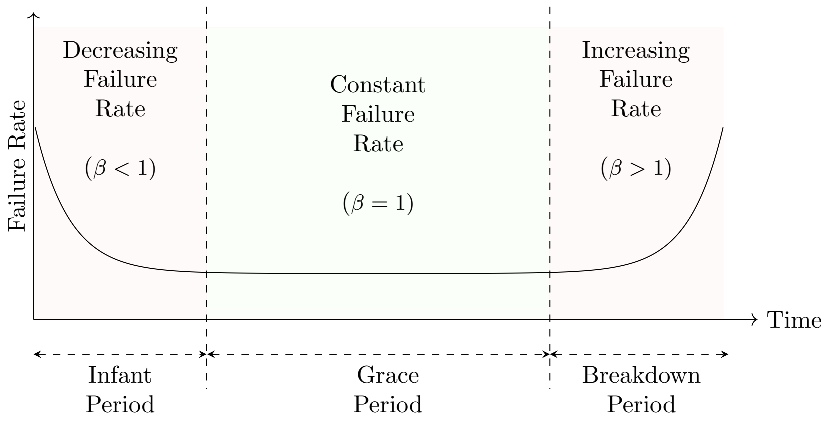


Figure 6.3: Failure rate at different periods of chip's lifetime.

lifetime of the target chip. In more detail, a chip exhibits different error rates in different periods of its lifetime. Without loss of generality, this variability has been modelled as a "bathtub" curve [236], as shown in Fig. 6.3. In its infant period, there is a very high but decreasing failure rate, until a plateau of minimum, constant failure rate is reached at its grace period. After a prolonged period of execution, ageing and wear-out effects are accumulated and aggravated leading to its breakdown period, where there is an ever increasing probability of error manifestation.

Last but not least, we must notice that the fault tolerant capabilities of the proposed framework are not bound by the employed system error characteristics. As presented in Section 6.4.2, our framework incorporates failure detection mechanisms and thus the only limitation of our design is that the manifested errors must affect the high level functionality of a PE. Therefore, intermittent and transient faults can also be mitigated as long as they are manifested long enough to be identified by our detection process. In the opposite case, they are silently masked and might affect only the outcome of the executed application. To mitigate such silent errors, the designer can take advantage of our proposed hierarchical design, in order to employ fault resilient deployment of the application, e.g. redundant execution of its tasks [162, 165].

6.4 Fault Tolerance Infrastructure

SoftRM extends the resource allocation principles of DRTRM, as presented in Chapter 4. The utilized target application model is the one presented in

Section 4.4. In addition, we define $w_i^j = 1$ if core i is a worker of core j and 0 otherwise. Regarding inter-core communication, all PEs maintain their own *neighbourhood* set Π_i with the ids of all cores inside their Cluster. Furthermore, each Manager core j maintains a *workers* set $\mathcal{W}_j = \{\, i \mid w_i^j = 1 \,\}$ which consists of all its Worker cores. Finally, each Controller with id i maintains a set with all the Managers that possess cores inside its Cluster. This is called the *Distributed Directory Service* ($\mathcal{DDS}$), where $\mathcal{DDS}_i = \{\, j \mid p \in \Pi_i \ \wedge \ w_p^j = 1 \,\}$.

At any time t the overall state of framework $\mathcal{F}$ is represented as a tuple $\mathcal{F} = \langle S_{t+1}, A, S_t \rangle$, where $S_t = \begin{bmatrix} s_0^t & s_1^t & \dots & s_n^t \end{bmatrix}$ is a vector of size N with the current states of all cores. A is a set containing all the applications a_i currently executed on the system and S_{t+1} are the next possible states of all cores, which depend on both S_t and A. We define $s_i^t = 3$ for an Idle core, $s_i^t = 2$ for a Worker, $s_i^t = 1$ for a Manager and $s_i^t = 0$ for a Controller.

6.4.1 Workload-Aware Paxos Algorithm

When examining the case of a core failure at run-time, Paxos algorithm can be employed in a straightforward way in order to designate a replacement core. Having identified the failure, the rest of the PEs can volunteer to take its place by proposing themselves as the replacement core (i.e. the proposed value is their id). This process while effective, inherently lacks of the ability to capture the state of the proposer PEs. For example, it is much more efficient to replace the failed PE with an Idle one instead of a Worker core, which is occupied with execution of application workload.

Inspired by this inefficiency, we introduce the *willingness factor* wf_j to capture the suitability of a PE with id j to act as a replacement core under its current workload characteristics as dictated by its role on the system. The willingness factor is defined as:

$$wf_j = s_j^t + c \sum_{k \in \mathcal{I}} w_j^k \sum_{p \in \mathcal{I} \setminus j} w_p^k \tag{6.5}$$

where s_j^t is the current state of core with id j and $\sum_{k \in \mathcal{I}} w_j^k \sum_{p \in \mathcal{I} \setminus j} w_p^k$ is the number of co-workers in an application. The constant c is calculated based on the number of maximum workers of an application, so that $c \sum_{k \in \mathcal{I}} w_j^k \sum_{p \in \mathcal{I} \setminus j} w_p^k < 1$. In our case, the value of c was set equal to 0.1. The formulation of wf favours idle cores (higher s_j^t), to express higher willingness to act as replacement cores. Additionally, in the case of a worker, increased number of co-workers leads to higher willingness in order to prevent resource starvation in applications with few cores.

Once a core with id i detects that a Controller or Manager has failed, it acts as a proposer and sends a *prepare*(n_i) message, where n_i is unique and greater than any n_{max} previously sent by any proposer $(\Pi_i$.

$$\forall j \in \Pi_i \rightarrow \text{send}: \ \mathit{prepare}(n_i) \qquad \text{(Phase } 1^a\text{)}$$

Since each core has a unique identifier i, determining n_i can be achieved by picking the smallest sequence number n_i greater than any previously sent (n_{max}) such that $n_i \bmod n_{max} = i$. Initially, the value of n_{max} is -1. Note that n_{max} and modulo are used for two different purposes: n_{max} guarantees that the accepted proposal numbers increase monotonically, whereas modulo guarantees that all proposal numbers are unique.

Subsequently, when a core with id j receives a *prepare*(n_i) message (Section 6.2), if $n_i > n_{max}$ where n_{max} is the highest numbered proposal received then it replies with its highest accepted proposal (v_{acc}) or -1 if no value has been accepted. The reply is paired with its willingness factor (wf_j). N_{max} becomes n_i and a promise is made not to accept any proposals numbered less than n_{max}.

$$\forall \mathit{prepare}(n_i) \rightarrow \text{send}: \ \mathit{response}(u_{acc}, wf_j),$$

$$n_i \geq n_{max} \qquad \text{(Phase } 1^b\text{)}$$

When the proposer receives an accept from a majority[1] of acceptors, it sends an *accept*(n_i, v_i) where v_i is the value of the highest accepted proposal replied in prepare phase. If no such value has been replied (i.e. all replies were negative), then it proposes the core with the highest willingness factor as the replacement core:

$$v_i = \begin{cases} v_{acc}, & \text{if any} \\ \underset{j}{\operatorname{argmax}}(wf_j), & \text{otherwise} \end{cases} \qquad (6.6)$$

In case that the maximum value is proposed by more than one core, the choice is made in a First Come First Served manner. Afterwards, an *accept*(n_i, v_i) message is sent to all acceptors.

$$\forall j \in \Pi_i \rightarrow \text{send}: \ \mathit{accept}(n_i, v_i) \qquad \text{(Phase } 2^a\text{)}$$

On the contrary, if a value v_{acc} is replied, it is a previously chosen value by another proposer based on willingness factors, since no other value had been

[1]In our case, the majority is any set of cores $S \subseteq \Pi_i$ such that $|S| > \frac{|\Pi_i|}{2}$, where $|S|$ denotes the cardinality of set S.

accepted when the choice was made. This value has already been accepted by the majority of acceptors and is replied so that the *safety* requirement of Paxos is ensured. As mentioned in Section 6.2, when an acceptor receives an $accept(n_i, v_i)$ message it replies with *accepted* iff it has not seen a higher proposal number before.

$$\forall accept(n_i) \rightarrow \text{send}: \; accepted(), \quad n_i \geq n_{max} \tag{Phase 2^b}$$

Eventually, when the proposer receives *accepted* messages from a majority of acceptors, it realises that its proposed value v_i has been accepted and broadcasts the accepted value (i.e. the id of the replacement core) to the platform.

$$\forall j \in \mathcal{I} \rightarrow \text{send}: \; learn(v_i) \tag{Phase 3}$$

It should be noted, that the willingness factor designation builds on top of PAXOS algorithm without interfering with its functionality. More precisely, the requirements of uniqueness and monotonicity of the proposal number (Section 6.2) are not affected and only the proposed value v is customized according to the suitability of one PE to substitute a faulty one. Thus, our proposed technique does not affect the correctness of the employed consensus algorithm.

We highlight that the described infrastructure is proposed to mitigate faults of the administrative tasks of a hierarchical resource management scheme, such as the presented Controllers and Managers. Regarding worker nodes executing the parallel workload, a failed worker is similar to a shrink operation, while the existence of a Manager per application allows the adoption of intra-application error resilient resource management techniques such as Double or Triple Modular Redundancy [165, 162], which can be seamlessly integrated in SoftRM.

6.4.2 Failure Detection

Failure detectors proposed in [52] are responsible for detection of node failures or crashes in distributed systems. A failure detector $\mathfrak{D}$ is classified by its *completeness*, meaning the suspicion of faulty processes and *accuracy*, meaning the suspicion of non-faulty processes. A perfect failure detector $\mathcal{P}$ satisfies both *strong completeness* and *strong accuracy.* Every faulty process is eventually permanently suspected by every non-faulty and no process is suspected by anybody before it has actually crashed. On a synchronous system with a known upper bound communication delay Δ, the simplest Perfect Failure Detector (PFD) [49] can be implemented as follows (HB stands for Heart Beat):

1. Broadcast $\langle HB\_REQUEST \rangle$ message.
2. Wait for the upper bound delay Δ.
3. If node j did not reply with a $\langle HB\_REPLY \rangle$ message within Δ, then detect j as faulty.

Although this approach is tolerable by general purpose systems, it comes with limited scalability and excessive message traffic since $[N-1]^2$ messages are sent every Δ seconds, where N is the number of PEs. We propose a tweaked version of PFD (tPFD), tailored for on-chip communication summarized as follows:

Set a timer to expire in Δ *seconds. If cores i and j from the same Cluster* ($\Pi_i = \Pi_j$) *have exchanged any message until timer expiration then both are considered alive. If not, a heart beat request is sent from i to j and vice versa and timer is reset to* Δ *seconds. If one of i or j has not replied until timer expiration, then it is considered a failed core.*

The algorithm takes advantage of the frequent inter-core communication for resource negotiation and application management on SoftRM to establish that a core is alive. A $\langle HB\_REQUEST \rangle$ is sent from one core to another only if they have not exchanged any messages during a Δ interval. If a $\langle HB\_REPLY \rangle$ has not been received within Δ seconds, then the recipient j core is declared as faulty. Liveness check signals are exchanged only inside a Cluster to provide a scalable detector.

To quantify the inflicted overhead on SoftRM by the presented detectors, we examine the average exchanged messages per second for failure detection in scenarios of 16 incoming applications. Scenarios differ in intensity of average application workload W (See Sections 4.6.3, 6.5). Table 6.1, summarizes the results which indicate that the proposed tPFD imposes more than half the overhead in exchanged messages compared to the original. Interestingly, as application workload increases less messages are sent by the tweaked detector, since inter-core communication is more frequent as applications are active for more time. The trade-off of the proposed detector is that it requires at most 2Δ latency to detect a faulty PE. This behaviour is acceptable for the examined applications since they are not bound by time-critical recovery characteristics.

6.4.3 Recovery

The recovery phase involves all the necessary actions for the system to be restored to a stable state. It takes place after all PEs have received the *learn* signal, which communicates the id of the replacement core as outcome of the execution of Paxos algorithm. All sets presented in Section 6.4 are updated to

Table 6.1: Messages/sec for $\mathcal{P}$ and $\mathcal{P}_t$ for different workloads.

Detector / Workload	Perfect Failure Detector $\mathcal{P}$ [49]	Proposed tweaked Perfect Failure Detector $\mathcal{P}_t$
Light	346.62	122.61
Medium	356.96	115.46
Heavy	388.33	114.84

reflect the new state of the system, omitting the failed core j. In similar manner, the new Controller builds its *Cluster* and *DDS* sets using information gathered by the rest of the system and according to the Cluster region topology of the failed Controller, as this administrative region cannot change dynamically.

In case of a Manager core failure, the replacement Manager signals all its Worker cores to inquire about the portion of workload that has been executed and thus make an assessment about the remaining workload of the application. In addition, by inspection of the logged checkpoints of the application all the portions of the workload under execution when the failure occurred are marked for re-execution to ensure a correct outcome. Then, workload is redistributed to Worker cores and application execution is restarted.

In case of a Worker core failure, application execution is paused and workload is re-distributed to the rest of healthy workers. Erroneous results are not propagated to healthy workers due to the blocking, MPI-like send/receive style of data exchange between workers. If a Worker core has been designated as the replacement core, it concludes its workload execution until the next checkpoint and then proceeds to its new duties. In parallel, its former Manager informs its Controller the worker loss and re-distributes the remaining workload to the rest of its workers.

SoftRM run-time example: Fig. 6.4 illustrates an instance of the operation of SoftRM. Fig. 6.4a presents a point in time where Controller 9 has failed. Core 14 identified the failure and triggered workload-aware Paxos algorithm. The willingness factors of the various PEs have been annotated on Fig. 6.4a (all idle cores have equal willingness factor). To designate the id of the replacement Controller core, an instance of the workload-aware Paxos is executed inside Cluster 2. An idle core has been chosen as replacement since it had the highest willingness factor i.e. the highest suitability to act as a new Controller. Then, the necessary recovery actions are performed (Fig. 6.4b) and Manager cores with workers inside Cluster 2 subscribe to the DDS set of the new Controller.

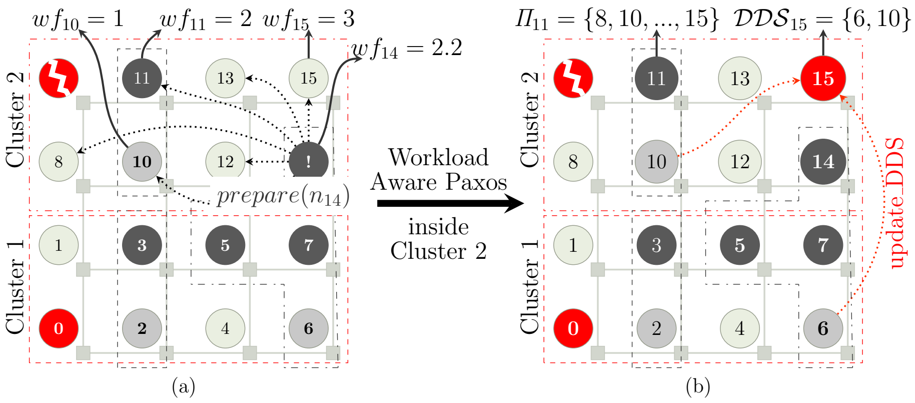


Figure 6.4: Example of workload aware Paxos in SoftRM.

6.5 Experimental Evaluation

SoftRM was developed and deployed on Intel SCC [115], described in Section 4.6.2. An evaluation scenario includes a number of applications requiring to be executed on the system. The following experiments were conducted using the implemented parallel Matrix Vector Multiplication applications presented in Section 4.6.3. This application was chosen as it is the most computationally intensive of the implemented parallel applications.

The distribution of workload values W for 128 incoming applications is presented in Fig. 6.5, for three different scenarios of workload intensity (Low, Medium, High). The values in X axis correspond to the range of possible workload values per application. In addition, the outcome of an examined scenario is affected by the arrival rate of applications on the system. In this experimental evaluation, this rate is generated randomly according to a Poisson distribution with λ equal to 48, as presented in Chapter 5. Each of the examined scenarios explores a different design parameter of SoftRM and is considered complete when all input applications have performed their execution life-cycle.

6.5.1 SoftRM Evaluation

Our first evaluation in Fig. 6.6, shows the correlation between the Δ interval of tPFD (Section 6.4.2) and the required time for SoftRM to recover from a Controller core failure, presented in mean value and standard deviation for a configuration of 4 Clusters. The moment of core failure is designated according to Eq. 6.3 of the presented error model (Section 6.3). In all presented

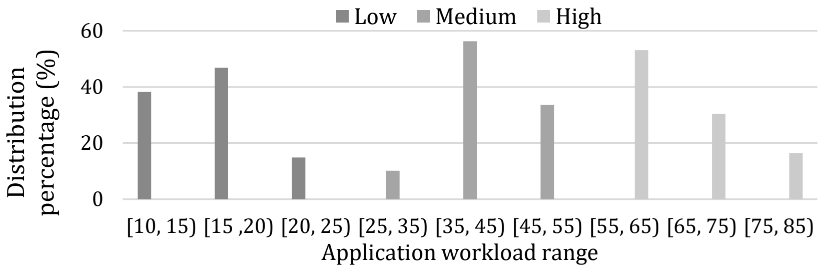


Figure 6.5: Workload distribution of examined scenarios.

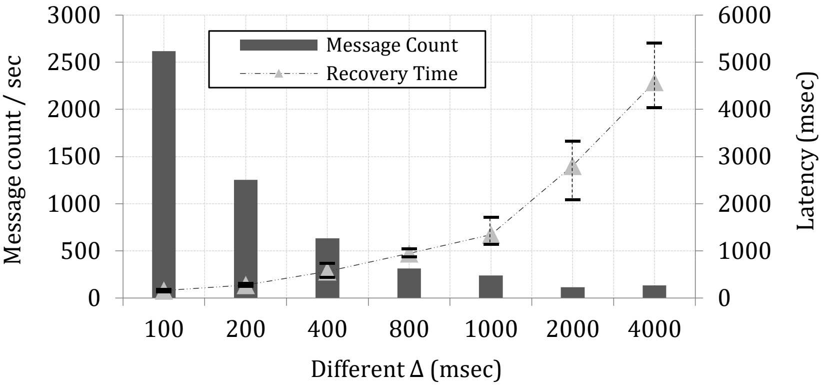


Figure 6.6: Recovery efficiency vs Message traffic overhead.

experiments MTTF is set to 6.56 years as in [74]. For this experiment, β is equal to 1 and all measurements have been repeated 10 times, to filter out the randomness introduced by the parallel execution of framework instances on the target HW. We observe that as Δ increases, both the average value and its deviation rise in absolute numbers. However, in relative numbers, in all cases the deviation is up to 15% of the mean value. The bars present the overhead induced on SoftRM for the detection of failed cores. Increased frequency of liveness check signals results in high toll on exchanged messages. Considering this trade-off, Δ was set to 1000 msec for the rest of the experiments.

The next experiment aims at analyzing the correlation between the granularity of self-organisation and **R**esource **A**llocation (**RA**) effectiveness. The evaluated parameter is the Cluster size, since the cores inside it self-organize to elect a replacement core. The bars in Fig. 6.7 correspond to the overhead in percentage of the **F**ault **T**olerance (**FT**) infrastructure on the rate of exchanged messages for RA per second by SoftRM. Smaller Cluster size leads to overhead reduction,

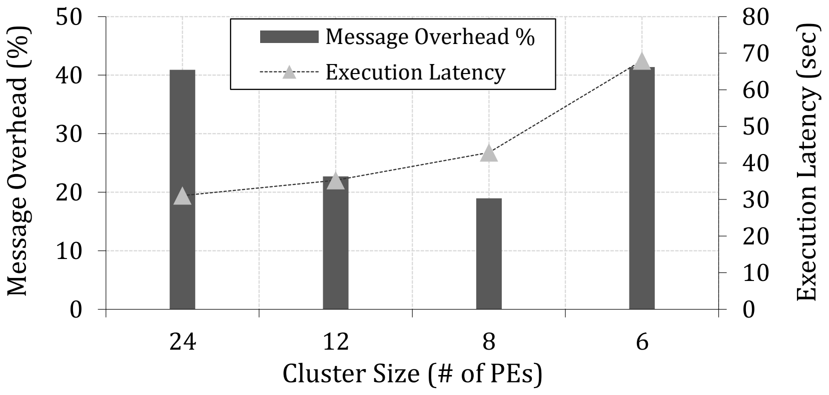


Figure 6.7: FT infrastructure overhead vs RA effectiveness.

Table 6.2: Frequency distribution of Replacement core's state.

Protocol / Previous State	Fixed IDs	Basic Paxos	Workload-aware Paxos
Idle	16.67%	33.33%	91.67%
Worker	58.33%	58.33%	8.33%
Manager	25%	8.34%	0%

since a Paxos instance is executed amongst a reduced number of PEs. However, smaller size equals more Clusters, more Controllers and thus reduced number of available PEs for workload execution. This is reflected in the total execution latency of each scenario (triangles), where a significant increase is observed for small Clusters due to the prolonged execution latency of applications because of the limited PEs assigned to them. For a Cluster of size 6, application slow down is so severe that the overhead of FT infrastructure sky-rockets.

In Table 6.2, an evaluation of the workload-aware Paxos protocol is performed against two other strategies for replacement core designation. More precisely, in Basic Paxos, each proposer volunteers to be the replacement core in step Phase 2^a (Section 6.4.1). In Fixed IDs strategy, the replacement core is the one with id $\min(\Pi_j)$ that resides in the same Cluster as the failed core j. The evaluation quantifies what was the state of the core before being chosen as the replacement one. The proposed workload-aware Paxos protocol manages to highly outperform its rival implementations and achieves to designate an idle core as the replacement with frequency of over 91%. This highly reduces the recovery overhead, compared to when a Worker or Manager core is chosen since

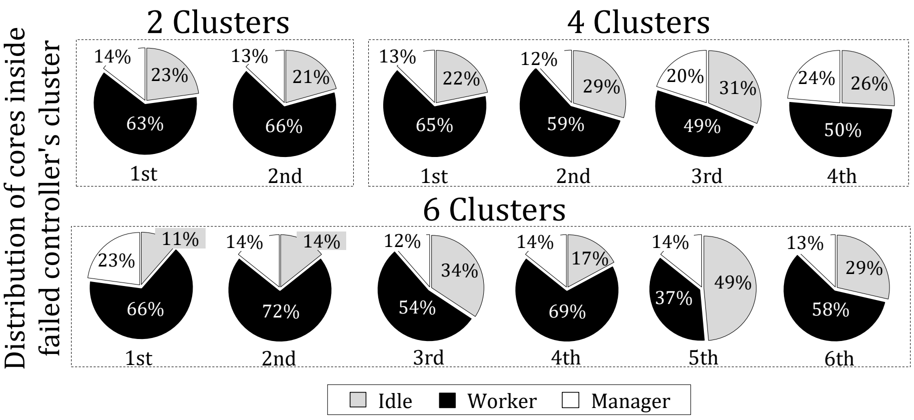


Figure 6.8: Distribution of core states during the execution of different strategies for failed core replacement.

it introduces no interference to the executed applications. Fig. 6.8 illustrates the average distribution of core roles the moment that the various strategies engaged in determining a replacement core. Different examined scenarios involved different numbers of Cluster areas on the system and therefore distributions are grouped according to this variable.

To quantify the efficiency of SoftRM versus state-of-the-art techniques at tolerating multiple errors we present in Fig. 6.11 the overview of system behaviour in a scenario of 64 incoming applications. Errors have been generated according to the utilized error model, taking into account all phases of the chip lifetime (Section 6.3) and affecting any PE of the system. Comparison is performed against two techniques based on spare cores provisioning, either dynamically at run-time [127] or statically at design time [59]. Spare cores are distributed among all clusters (e.g. in a configuration with 4 clusters, 16 spare cores would equal to 4 spare cores per cluster).

The X-axis of Fig. 6.11 represents time, showing the arrival of applications and the time points in which errors occurred (thunder symbols). The different fault tolerance schemes are compared in terms of the number of completed applications within a specific time frame. SoftRM facilitates the completion of a significantly higher number of applications compared to the rest of the techniques, while managing to mitigate all errors. This happens because no core is excluded from workload execution unless it is mandatory for fault recovery and at the same time the overhead of self-optimization is considerably low. The dynamic spare core provisioning scheme [127] as well as the static of 24 cores also manage to mitigate all errors but they impose a heavy toll on

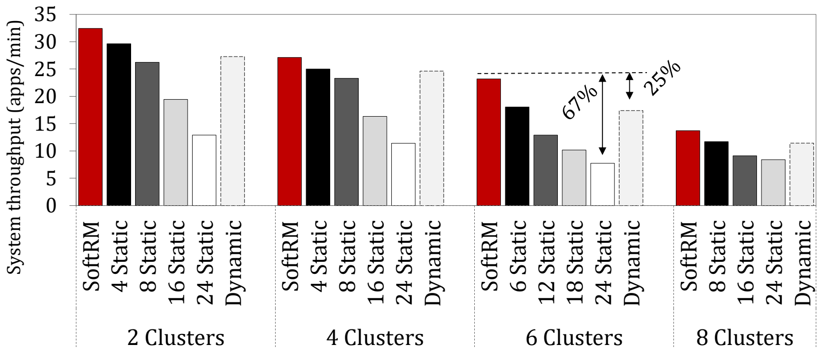


Figure 6.9: Comparison between SoftRM, static and dynamic spare core allocation for fault free execution.

application execution, since a high number of PEs is inactive until an error occurs. Most importantly, all other static spare core allocation configurations fail at mitigating all errors, a fact that we consider a breaking point in the functionality of the system. The less the number of pre-allocated spare cores, the faster this breaking point is reached.

To further quantify the reduced overhead of SoftRM, Fig. 6.9 summarizes the system throughput expressed as successfully completed applications per minute, achieved by SoftRM versus the spare cores provisioning techniques, taking also into account different Cluster configurations. In all cases, our approach results in higher throughput, since no cores are excluded from application execution. As expected, the highest overhead reaching 67%, is observed when 24 cores (half the available ones) are provisioned for fault tolerance. We should note that this high number does not offer higher error resilience as equal number of errors can be mitigated by SoftRM. Dynamic core provisioning leads to overhead of up to 25%, however our implementation of this approach was conservative in the sense that only one spare core is dynamically provisioned per application. This implies that only one error can be tolerated per application and in case of more, its execution fails.

In the last conducted experiment, presented in Fig. 6.10, we quantify the behavior of SoftRM under variable error manifestation time series. By adjusting the value of β parameter of the utilized error model, we are able to customize the error manifestation rate on the system. For fair comparison purposes, we maintain the number of errors constant and equal in all scenarios. The influence of errors is quantified as the percentage of execution penalty on the input applications, using the fault free scenario as a baseline. Results confirm that high execution penalty is observed when more errors are manifested earlier in

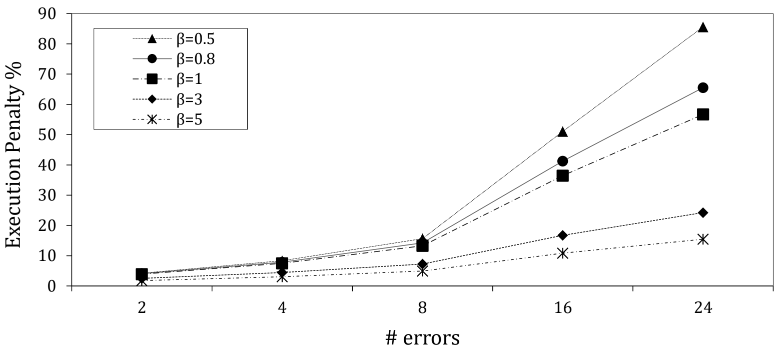


Figure 6.10: SoftRM evaluation for varying β of error model.

the lifetime of a scenario (lower β). This is because a high number of error corresponds to less available PEs for workload execution and the faster these PEs are damaged, the higher is the number of application that are affected thus leading to higher execution latency penalty.

6.6 Conclusions

This Chapter introduced SoftRM, a self-organizing and fault tolerant Distributed Run-Time Resource Management framework for NoC-based many-core systems. Its building blocks were presented i.e. fault detection, replacement core determination and recovery actions for the framework to be restored to a stable state. The replacement policy is based on Paxos consensus reaching algorithm, enhanced with workload-aware features for effective use of free resources on the system. To showcase the effectiveness of SoftRM it was implemented as an extension of DRTRM and evaluated on Intel SCC many-core system.

The key factors for the efficiency of SoftRM is the minimization of the overhead of the fault detection infrastructure on the system, as well as its recovery speed and effectiveness of workload awareness in the case of a manifested error. Towards these goals, the experimental impact analysis of the design parameters of SoftRM enabled their fine-tuning in order to achieve minimum interference on the resource allocation process, during fault-free operation cycles. In addition, it validated the ability its workload-aware features manage to efficiently utilize free resources in more than 90% of the conducted experiments. In total, the optimized version of SoftRM, is able to recover from all manifested errors, while leading to higher system throughput in comparison to both fault tolerance schemes based on static or dynamic spare core provisioning.

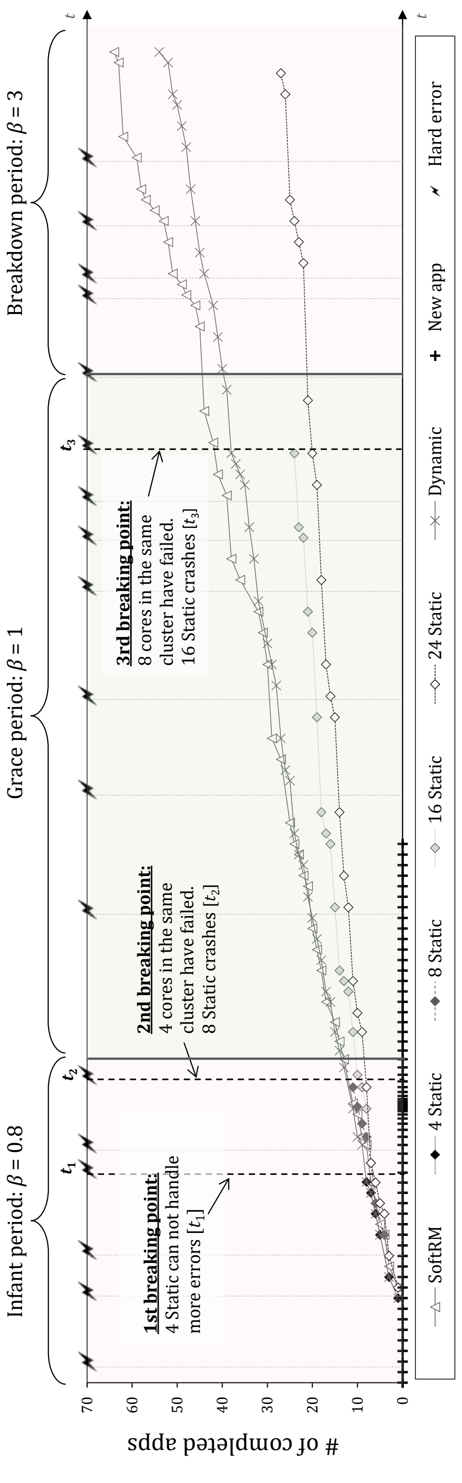


Figure 6.11: Overview of the evolution of the system according to different employed fault tolerant techniques.

Chapter 7

Distributed Trade-based Device Management in Multi-Gateway Edge IoT

7.1 Introduction

A very import feature of the IoT architectures presented in Chapter 1, is that they include mobile nodes which operate on battery supply. These battery-powered devices are characterised by individual lifetime expectancies before their next recharge takes place. To regulate energy consumption under battery lifetime constraints, each IoT device has two control knobs: 1) changing the offered Service Quality (SQ) to the user and 2) changing its on-board processing policy. For instance, in the IoT ECG analysis application (Section 3.3), the device can reduce the sampling rate of input data in case of shortage of battery capacity, thus reducing the required computations. Alternatively (or conjointly), it can stop the execution of the full application pipeline and offload part of it to the IoT Gateway.

Nevertheless, Edge Gateways are also limited in resources both in wireless communication bandwidth and processing for executing the offloaded tasks. It should be noted that although the Gateway might be equipped with a high-bandwidth connection to the Internet (e.g. WiFi), its interface with IoT devices is still a low-power, low bandwidth, wireless connection such as Bluetooth Low Energy (BLE), ZigBee, etc. [251, 106, 19]. Consequently, the resources of the

Gateway should be carefully allocated to the various IoT nodes, taking into account the run-time requirements of all involved nodes.

In such a local network, IoT devices aim at reaching the highest overall SQ while meeting their battery lifetime constraints. Existence of multiple Gateways provides to some IoT nodes more than one option to connect and receive Gateway service. The run-time decision of which IoT node will connect to which Gateway will be referred to as *binding problem*). The importance of efficient binding is to avoid situations where some Gateways are overloaded, while others are underutilized. For instance, if several IoT devices with high data transmission demands are connected to the same Gateway, they must reduce their sampling rates to meet the constraint of the limited shared communication bandwidth, thus reducing the overall offered SQ of applications.. Since in the Edge system users are mobile, the IoT nodes will be able to simultaneously connect to different Gateways at run-time, implying that the binding problem is dynamic and requires an online solution.

The resource allocation problem of the IoT nodes and a single Gateway has already been investigated in [189]. Given a set of IoT nodes already bound to a Gateway, authors of [189] have proposed an optimized dynamic programming solution, which takes as input the acceptable operation modes of the IoT nodes and designates the mode per node in order for the total offered SQ to be maximized. This work, utilizes the work in [189], in order to scale up the decision making logic to a setup of multiple Gateways. The contributions of this Chapter are the following.

- A run-time mechanism is proposed, which is employed by Gateways in order to negotiate their connected IoT nodes, in an effort to maximize the offered SQ of the entire Edge system.
- Taking into account the inherent distributed nature of the target system, the concepts and knowledge derived from Distributed Run-Time Resource Management (Chapter 4) are applied to negotiation mechanisms.
- The proposed scheme is evaluated by means of a developed simulator, able to capture the dynamics of scenarios where multiple IoT nodes and Gateways co-exist.

More precisely, the problem formulation is presented in Section 7.2, followed by a detailed analysis of the proposed distributed solution in Section 7.3. Section 7.4 focuses on the description of required mechanisms for the dynamic negotiation between Gateways. The proposed solution is evaluated experimentally in Section 7.5, via a case study based on the ECG analysis application described in Section 3.3. Last, Section 7.6 concludes the Chapter.

7.2 System Model & Problem Formulation

7.2.1 Application Model

The target application model, assumes IoT applications with a pipelined structure similar to the ECG analysis application presented in Section 3.3. The secondary critical factor is that the application is able to support different levels of service quality, modulating the user's satisfaction at the cost of more resource usage (e.g. energy consumption, bandwidth, etc.). Different Service Qualities usually derive from different quality of input data, since the quality of the the captured signal determines the offered user experience and satisfaction as explained in Section 3.3.6.

The aforementioned pipelined structure, provides the necessary modularity to the application in order to be able to be dynamically configured at run-time. The IoT application has different possible offloading levels, as it may either 1) fully process the captured data on the IoT device, or 2) partially process the data on the IoT device and offload the rest of the computation to the Gateway or 3) offload the whole computation to the Gateway by transmitting the raw data. The combinations of SQ and offloading levels, are the critical dynamic tuning knobs in the effort to meet the run-time constraints of IoT nodes operating on battery supply.

7.2.2 IoT device Model

We consider an IoT network consisting of N portable IoT devices, where each device I_d, $d \in \{1, \cdots, N\}$, is described by a tuple:

$$I_d = \Big(X_d, R_d, B_d, e_d, U_d(\cdot), C_d(\cdot)\Big) \tag{7.1}$$

where:

- X_d denotes the set of possible *input data rates* of device d. They depend on the sensor sampling frequency and data resolution. An IoT device offers its service at M_d different *SQ levels*, with each level having a different input data rate and thus providing a different service quality.

$$X_d = \big\{x_{d_i} \mid i \in [1, M_d]\big\} \tag{7.2}$$

- R_d denotes the set of possible *transmission data rates* of device I_d. They depend on the input data rate x_{d_i} and the computation offloading strategy of the IoT device. Offloading determines the portion of input data that are not processed on the device (on-board processing), but are transmitted to the Gateway instead. An IoT device offers Q_d different *offloading levels*. Each data transmission rate $r_{d_{ij}}$ depends on the SQ level i (input data rate) and the offloading level j.

$$R_d = \left\{ r_{d_{ij}} \mid i \in [1, M_d],\ j \in [1, Q_d] \right\} \tag{7.3}$$

The particular transmission data rates depend on the design of the application and its user preferences, hence are considered as given in this problem formulation.

- B_d denotes the minimum required battery lifetime (i.e. until the next recharge).
- e_d is the remaining energy in the battery of device d.
- $U_d(x_{d_i})$ is the utility function that quantifies the Service Quality (SQ) provided to the user when the device is capturing input data at rate x_{d_i}.
- $C_d(i,j)$ is the total power consumption for sensing and capturing input data at rate x_{d_i}, processing it under offloading level j and transmitting the data at rate $r_{d_{ij}}$. It includes the power consumption for sensing, computation and communication.

The battery lifetime of each IoT device depends on 1) its remaining energy and 2) its power consumption rate:

$$b_{d_{ij}} = \frac{e_d}{C_d(i,j)} \tag{7.4}$$

where $b_{d_{ij}}$ denotes the expected battery lifetime when the device captures input data at rate x_{d_i}, processes it, and then transmits at rate $r_{d_{ij}}$.

7.2.3 Gateway Model

The Gateway bridges local IoT nodes with Cloud infrastructure via the Internet. It receives data from IoT devices, performs local processing and communicates the results back to the IoT devices and the final Cloud destination, accordingly.

We consider a set of G Gateways, where each Gateway g is specified by a tuple:

$$\text{Gateway } g : \ \Big(p_g(\cdot), R_g, P_g \Big)$$

where:

- $p(r_{d_{ij}})$ shows the required processing resources of the Gateway to act upon the received data at rate $r_{d_{ij}}$ and offloading level j,
- R_g is the total available bandwidth of the Gateway for data reception from its connected IoT devices.
- P_g shows the total processing capability of the Gateway.

The effect of the environment and surrounding devices (e.g. wireless interference) on the transmission can be modelled in parameters R_g of the Gateways and $C_d(\cdot)$ of the IoT devices. For instance, if an IoT device observes an increase in re-transmission rate, it can increase the cost of transmission. However, modelling the effect of interference on the transmission parameters is beyond the scope of this work.

7.2.4 Network Model

While the Gateways are assumed to be stationary and non-mobile, the IoT devices are considered *quasi mobile*, i.e. they are mobile but their location does not change significantly and frequently.For instance, the IoT devices used for patient monitoring in a smart hospital or smart home have a mobility pattern of natural movement of patients [148].

Depending on the location of IoT devices and Gateways, each one reaches some Gateways (at least one). Each IoT device should connect to one and only one Gateway and for those devices that reach multiple Gateways, their binding needs to be decided. Let matrix $A[\cdot]_{N\times G}$ denote which IoT devices reach which Gateways, with a value of 1 if device d reaches Gateway g or 0, otherwise.

7.2.5 Problem Statement

The Edge computing system is illustrated in Fig. 7.1. The target problem is composed of 1) the decision of the binding of IoT devices to the Gateways and 2) the designation of the SQ level i and the offloading level j for each IoT device d at run-time, such that the bandwidth, computation, and lifetime constraints

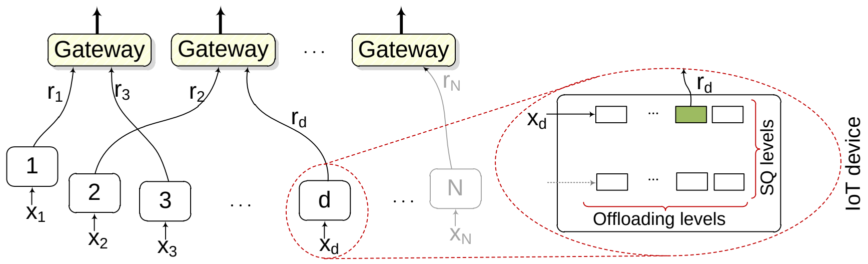


Figure 7.1: Problem model: IoT devices with different SQ and offloading levels resulting in different transmission data rates. Multiple Gateways receive and process the data from multiple IoT devices.

are fulfilled (Eq. (7.5) to (7.7)) and the overall SQ (Eq. (7.8)) is maximized.

$$\text{Bandwidth constraint:} \quad \sum r_{d_{ij}} \leq R_g \qquad \forall d \text{ connected to } g \tag{7.5}$$

$$\text{Computation constraint:} \quad \sum p(r_{d_{ij}}) \leq P_g \qquad \forall d \text{ connected to } g \tag{7.6}$$

$$\text{Lifetime constraint:} \quad \forall\ d:\ \ b_{d_{ij}} \geq B_d \tag{7.7}$$

$$\text{Optimization goal:} \quad \text{maximize} \sum\nolimits_{\forall d} U_d(x_{d_i}) \tag{7.8}$$

7.3 Proposed Solution

7.3.1 Decomposing the Problem

The aforementioned problem has two sets of constraints: one for IoT devices and one for Gateways. The selected configurations for devices d (i.e. x_{d_i} and $r_{d_{ij}}$) should meet the lifetime constraint (Eq. (7.7)). Given the selected configuration for IoT devices, the constraints to meet with respect to the Gateway are the bandwidth and computation resources (Eq. (7.5) and (7.6)).

The individual constraint of each device is affected solely by its parameters. Consequently, to reduce the search space, we can decompose the optimization problem to each IoT device and the network (Gateways) problem. Regarding the IoT device problem, each device excludes those configurations that violate its lifetime constraint to reduce the search space. Then, the network problem is solved by considering only the reduced search space.

7.3.2 Device Problem: Battery Lifetime Constraints

Considering its lifetime constraint, each device finds the efficient feasible configurations (EFC), each of which corresponds to a SQ level with the minimum data transmission rate. Each EFC is a pair containing the utility and the transmission data rate r_{d_i} of this configuration. The Gateway extends each EFC set by including the processing requirement of the associated transmitted data (i.e. $p(r_d)$), resulting to EFC'_d set.

Regarding latency constraints, each IoT device is responsible for excluding from its EFC set the configurations that violate them. The evaluation of the constraints and thus the structure of the EFC set is online and dynamic. Further details on designating the EFC set can be found in [189]. We additionally define as $U_{df} = U_d(x_{d_f})$ the Utility of device I_d for the f-th EFC entry, $r_{df} = r_{d_f}$ as the corresponding transmission rate and $p_{df} = p(r_{d_f})$ as the corresponding Gateway processing requirements. Each IoT device periodically checks its remaining energy e_d and updates the EFC set. In case the EFC set changes, the device sends the new set to its Gateway, where it is used to solve/update the *Network problem*. Note that the EFC set only contains feasible solutions, i.e. the number of entries in the EFC set of a particular device may change over time.

7.3.3 Network of Gateways Problem

Given the EFC sets of IoT devices, the Efficient Multi-Gateway Allocation and Binding (EMGAB) problem can be formulated as:

$$\max \quad \sum_d \sum_f (U_{df} \times w_{df}) \tag{7.9}$$

$$\text{subject to} \quad \forall d: \quad \sum_f w_{df} = 1 \tag{7.10}$$

$$\forall d: \quad \sum_g v_{dg} = 1 \tag{7.11}$$

$$\forall d, g: \quad v_{dg} \leq A[dg] \tag{7.12}$$

$$\forall g: \quad \sum_d \sum_f r_{df} \times w_{df} \times v_{dg} \leq R_g \tag{7.13}$$

$$\forall g: \quad \sum_d \sum_f p_{df} \times w_{df} \times v_{dg} \leq P_g \tag{7.14}$$

where:

$$w_{df} = \begin{cases} 1 & \text{if the } f\text{-th EFC element from the } d\text{-th device is chosen} \\ 0 & \text{otherwise} \end{cases} \tag{7.15}$$

$$v_{dg} = \begin{cases} 1 & \text{if the } d\text{-th device is connected to the } g\text{-th Gateway} \\ 0 & \text{otherwise} \end{cases} \tag{7.16}$$

Eq. (7.10) ensures that one configuration for each device is selected. Eq. (7.11) and (7.12) ensure that each IoT device is connected to one Gateway which is in its range. Finally, Eq. (7.13) and (7.14) ensure that the binding of IoT devices to the Gateways and their selected configurations meet the constraints of each Gateway.

7.3.4 Analysis of the Problem

As stated in Section 7.2.4, some IoT devices have multiple choices to connect to Gateways, while some others reach only one Gateway (i.e. no decision is needed for binding, but they still need to choose the SQ level and offloading policy). Let H_d denote the set of Gateways reachable by device d ($|H_d| \geq 1$). The total number of bindings to investigate is $\Pi_{d=1}^{N}|H_d|$. Then for a possible binding setup, there are G optimization problems to find the optimal SQ level and offloading policy for the IoT devices. The optimal solution for one instance (i.e. a single Gateway) is presented in [189].

Proof. The EMGAB problem (presented in Eq. (7.9) to (7.16)) is strongly NP-complete. □

Proof. The numerical parameters of EMGAB (i.e. N, X_d and R_d) are bounded by a polynomial. The reason is the limitation of practical setup in IoT systems. Now we show that for the bounded parameters, the EMGAB problem remains NP-complete: EMGAB is a generalization of the Multiple Choice Multidimensional Multiple-Knapsack Problem (M^{3}PK). Each Gateway corresponds to a knapsack which has two constraints: bandwidth corresponds to the 'volume' and processing power corresponds to the 'weight'. Each EFC' set of a device corresponds to a class of items among which, one item must be picked. This transformation between EMGAB and M^{3}PK is done in polynomial time. The M^{3}PK problem is a generalization of Multiple Knapsack Problem (MKP), Multiple Choice Knapsack Problem (MCKP), Multiple Choice Multidimensional Knapsack Problem (MMKP) which are proven to be NP-complete. Hence, an NP-complete problem is reduced to the EMGAB in polynomial time. Due to the polynomial bound on the inputs, the EMGAB problem belongs to the class of strongly NP-complete problems. □

The problem is computationally difficult to solve in a centralized fashion. Furthermore, the architecture of the target IoT systems, with multiple devices and multiple Gateways, is naturally distributed. Therefore, distributed or decentralized strategies are promising solutions that take autonomous decisions for IoT devices and Gateways based either on local information, or on an incomplete picture of the global network status. As in the case of DRTRM (Chapter 4), an initial solution is discovered and then the distributed agents iteratively optimize their choices at run-time, taking into account the dynamics of the system and the incoming applications.

Market oriented approaches have been used for distributed resource allocation problems and can be classified into three models: 1) price-based, 2) auction-based, and 3) trade-based [210]. Mechanisms that are based on price and auction usually require a centralized entity with a full picture of network conditions (e.g. an auctioneer to run the auction or a decision-maker to calculate the price) [210, 128, 150]. Due to the nature of the system and problem, a trade-based distributed solution seems more effective.

7.3.5 Distributed Solution and MGAB Protocol

In this subsection, we present a distributed solution to MGAB problem (i.e. multi-Gateway allocation and binding) and the detailed protocol to implement it. Our solution is based on trading, in which Gateways are modelled as intelligent agents that negotiate with each other to *acquire*, *provide* or *exchange* the connected IoT devices.

Definition 7.3.1. *Common vs. Exclusive IoT devices:* From the perspective of Gateways, an IoT device is either reachable by only one Gateway, thus called *exclusive*, or reachable by more than one Gateway, which is then *common* between them. Fig. 7.2 shows an example with two Gateways sharing two common IoT devices (devices 4 and 5) and three additional exclusive IoT devices per Gateway. Each node is characterized by an *established connection*, i.e. IoT nodes 4 and 5 are connected to the Gateways 1 and 2, respectively. However, the figure includes three more *alternative connections* for them.

While the only action for exclusive nodes is to change the SQ and/or offloading level, there are two extra actions for common nodes:

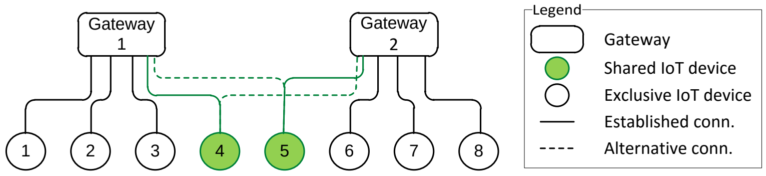


Figure 7.2: An example with two Gateways sharing two IoT devices while each has three exclusive IoT devices.

1. Migration: One common node leaves its current Gateway and joins another one. For instance, IoT device 4 disconnects from Gateway 1 and connects to Gateway 2.
2. Exchange: The two Gateways exchange two common IoT devices with each other. For instance, IoT device 4 connects to Gateway 2 and IoT device 5 connects to Gateway 1.

Given a binding/allocation setup, the following two situations make it inefficient and an *exchange* or *migration* is required.

Definition 7.3.2. *Fragmentation:* Gateways have unused resources that are not enough for increasing the SQ level of the current connected IoT devices, but might be enough for increasing the SQ level of a common IoT device, which is currently connected to another Gateway.

Definition 7.3.3. *Heterogeneity of resource usage:* The IoT devices are different in terms of resource usage, i.e. some require more bandwidth while some others more processing power. This can lead to a situation where one Gateway has much unused bandwidth, and another has much unused processing power. We refer to this case as heterogeneity in resource usage.

7.3.6 Properties of Applications and Problem

A number properties of the applications are introduced and exploited to reduce the complexity of the target problem.

Property 1: Considering device d and two elements of its EFC'_d set, (U_{di}, r_{di}, p_{di}) and (U_{dj}, r_{dj}, p_{dj}): If $U_{di} > U_{dj}$ then at least one of these conditions hold: $r_{di} > r_{dj}$ or $p_{di} > p_{dj}$. The underlying rationale is that otherwise the i-th element dominates the j-th element and is always preferable to it.

Property 2: Considering device d and two elements of its EFC'_d set, (U_{di}, r_{di}, p_{di}) and (U_{dj}, r_{dj}, p_{dj}): Device d can operate in a mode which provides any average utility (SQ) U'_d, $U_{di} \leq U'_d \leq U_{dj}$, from the application's perspective.

Proof. Consider a constant time interval of T. If the device operates at the j-th point for t_1 time and then changes the SQ level and operates at the i-th point for $(T - t_1)$ time, then the average utility, transmission rate, and processing power usages are:

$$U'_d = \frac{t_1 \times U_{dj} + (T - t_1) \times U_{di}}{T} \tag{7.17a}$$

$$r'_d = \frac{t_1 \times r_{dj} + (T - t_1) \times r_{di}}{T} \tag{7.17b}$$

$$p'_d = \frac{t_1 \times p_{dj} + (T - t_1) \times p_{di}}{T} \tag{7.17c}$$

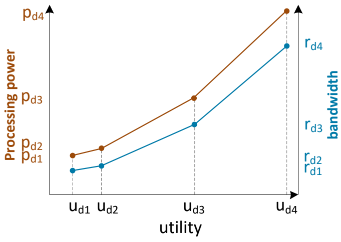
□

When a device is instructed to deliver an intermediate SQ level (e.g. U'_d in Eq. (7.17c)), it uses the following scheme: It starts operating at the j-th configuration (which has higher utility and consumes more resources on the Gateway compared to the i-th configuration). It keeps working at this configuration for t_1 time. However, it transmits its data to the Gateway at the rate of r'_d ($r'_d \leq r_{dj}$). It buffers the rest of produced data (i.e. $r_{dj} - r'_d$) on its memory. After t_1, it switches to the i-th configuration that produces data at r_{di} rate ($r_{di} \leq r'_d$). It still transmits the data to the Gateway at the rate of r'_d, which consists of previously buffered data <u>and</u> newly generated data. It keeps operating at the i-th configuration for $(T - t_1)$ time and then repeats this procedure as long as the requested utility is U'_d.

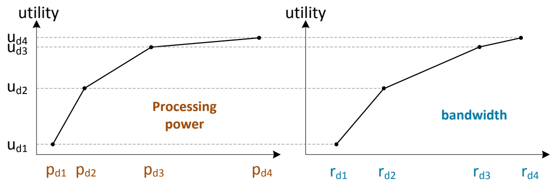
Therefore, while the device is only operating at the i-th and j-th configurations and switching between them, the Gateway sees the device operating at an intermediate configuration mode with (U'_d, r'_d, p'_d). In summary, this new operating point (i.e. configuration) is the application level view of the device. The device still operates only in discrete configurations, but this is transparent to the Gateway and the average effect from the perspective of the application and the Gateway is continuous.

7.3.7 Forming Piecewise-linear Utility Function:

Using Property 2, we can expand the discrete utilities of each device to a piecewise-linear function. However, since each discrete utility value (e.g. U_{di})



(a) Processing power & bandwidth associated with utility values.



(b) Piecewise-linear utility functions with respect to bandwidth and processing power.

Figure 7.3: The extended utility function of device d derived from discrete EFC'_d set. The function is piecewise liner and weakly concave with respect to both variables (i.e. r and p).

corresponds to two variables (e.g. r_{di} and p_{di}), there will be two uni-variable utility functions. These two variables are dependent. Fig. 7.3b shows an example of these two utility functions for device d.

Property 3: *Concavity of Utilities* is a general property of typical applications [150, 37]. It is a natural restriction based on the “law of diminishing returns” from economic concepts [58, 99].

According to Properties 2 and 3, the piecewise-linear utility functions are *weakly concave.* In the example of Fig. 7.3b, the concavity of utility functions is illustrated. The fragmentation problem can be addressed according to Property 2. The problem stems from discrete and coarse-grained configurations (i.e. EFC elements), and a solution can be provided by adopting continuous utility functions, derived using Property 2.

7.4 Details of Agent-based Approach

After addressing the fragmentation problem, we need to deal with the heterogeneity problem. In the following, we present the details of our solution which is a distributed agent-based negotiation mechanism between Gateways, for migration and exchange of IoT devices. It begins with an initial setup and then step-by-step converges to the efficient solution by increasing the overall utility of IoT devices. Each Gateway is modelled as an autonomous intelligent agent, where the interests (or goals) of each agent is consistent with the goals of the whole optimization problem. We use the terms agent and Gateway interchangeably from here on.

Initial Phase

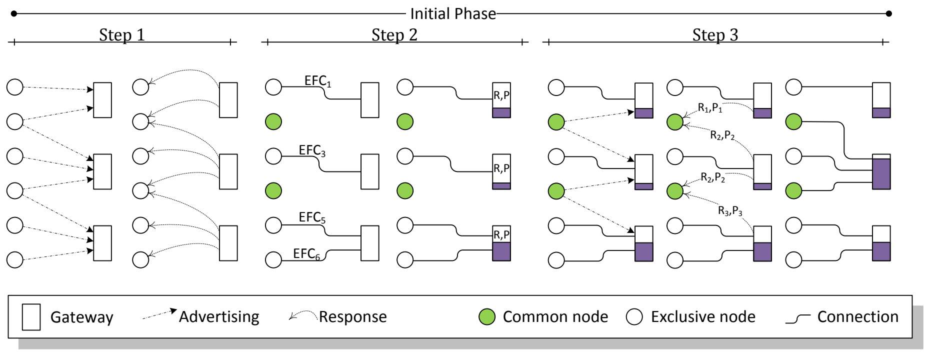


Figure 7.4: Different steps during the initial phase, followed by trade phase.

During the initial phase, Gateways try to establish connections with IoT devices to achieve the first setup. Fig. 7.4 shows an example of the initial phase with 6 IoT nodes and 3 Gateways. It consists of three steps:

1. Advertisement and Discovery: First, the IoT devices broadcast advertising packets to locate Gateways in their vicinity. Each Gateway that received the advertisement packet, replies with a request to establish the connection, one for each IoT device in its range. All exclusive IoT devices receive only one request (as they in reach of only one Gateway), while the rest receive more than one.

2. Exclusive Connections: Then, each exclusive IoT device accepts the request of its Gateway and connects to it. It sends its EFC set to the Gateway. After establishing the connection and receiving the EFC set of devices, the Gateway checks the lowest SQ level of connected devices, and calculates the remaining resources to make sure that it still has enough resources to provide service to new devices. If not, it should stop sending requests to other IoT devices and accepting new devices.

3. Non-Exclusive Connections: In this step, common IoT devices send new advertisement packets. In return, the Gateways in range send updated requests along with the information on their remaining resources. Each common IoT device accepts the first request for connection, which has enough resources to support its lowest SQ level.

The advertisement and discovery mechanism is based on the BLE protocol, where slave nodes (IoT devices) initiate the advertising and master node (Gateway) sends the request in response to the advertisement [219]. However, other protocols or wireless technologies (e.g. ZigBee) can support this mechanism. Once the initial phase is complete, each Gateway executed the algorithm presented in [189, 188] in order to designate the optimal configuration points (SQ level and offloading policy) for its connected IoT devices, based on the received information (EFC sets) from them.

Negotiation & Trade Phase

After the initial phase, each IoT node is bound to one Gateway. This setup is referred to 'initial binding'. Each Gateway designates the optimal SQ level for its connected IoT devices, using the algorithm proposed in [189]. This initial allocation is only optimal for the current binding (i.e. single Gateway level), but still not optimal for the whole network. The initial phase is followed by negotiation phases, where agents communicate to evaluate the possibility of a trade to adapt the binding and advance towards the optimal binding. This is achieved via a *market-based* approach, where agents make offers for exchanging or migrating their connected IoT devices.

In the negotiation phase, two agents are involved in a trade: an agent initiates a trade by sending a request either to migrate one of its shared devices to the other agent or to exchange a shared device with another one. The other agent, after performing some evaluations, may 1) accept the offer, 2) reject the offer, or 3) make another offer in response to the request. This decision is in the direction of increasing the overall SQ of the system. In other words, each trade should be a *Kaldor-Hicks improvement* [66], i.e. the gain of the IoT device that increases its SQ is higher than the loss of the IoT device that decreases its SQ. In this way, the overall SQ of the system is increased.

A. Migration:

The migration can more effectively address the situation of an uneven binding (i.e. overloaded Gateways), but it can address the heterogeneity issue as well. At first, each Gateway g, obtains 2 parameters: $\hat{R}_g$ and $\hat{P}_g$ which denote the amount of the remaining bandwidth and processing power, respectively. Note that these values are non-zero for the Gateways, where connected nodes are at their highest SQ level. The migration is feasible only if one Gateway (i.e. source) has no resources left (i.e. $\hat{R}_g \times \hat{P}_g = 0$), but the other Gateway (i.e. destination) still has available resources.

Fig. 7.5 illustrates an example negotiation of two agents. In this example, nodes 4 and 5 are common between gateway1 and gateway2 and in the initial binding,

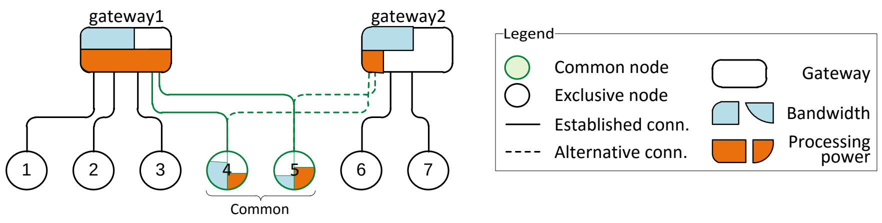


Figure 7.5: Heterogeneity in resource usage of IoT devices and Gateways.

both nodes are connected to gateway1. The negotiation and trade details of a migration are as follows:

1. **Picking the candidate to migrate:** Gateway1 selects one of its common (i.e. non-exclusive) IoT devices for migration. The node with the highest consumption of the scarcest resource (i.e. makes the heterogeneity issue worse) is prioritized. For instance in Fig. 7.5, node 5 is prioritized over node 4, because it consumes more processing power, which is the scarcest resource on gateway1.

2. **Checking the usefulness of migration:** Gateway1 calculates the 3-tuple of $(\overset{+}{u}, \overset{+}{r}, \overset{+}{p})$. This triple describes the best possible improvement among IoT devices *if* the candidate common device is migrated to another Gateway. Therefore, $\overset{+}{u}$ is the gained utility, while $\overset{+}{r}$ and $\overset{+}{p}$ show the remaining bandwidth and processing power of gateway1 after this improvement, respectively. For instance, if node 5 is migrated from gateway1 to gateway2, then other nodes might be able to increase their SQ level at the cost of getting more bandwidth and processing power from gateway1. If the candidate device migrates to another Gateway, it releases its resources on the first Gateway, freeing up r^* and p^* bandwidth and processing power, respectively.

3. **Make an offer:** Gateway1 sends an offer of migration to gateway2, which includes the information about 1) its benefit from improvement (i.e. $\overset{+}{u}$), 2) the EFC set of the candidate node, and 3) the current operating configuration of the candidate node (i.e. (u_5^*, r_5^*, p_5^*) in our example).

Upon receiving this offer (i.e. migration request), the destination Gateway (i.e. gateway2) evaluates the offer and replies to it. The offer is evaluated with respect to its potential of improving the overall SQ of the system. The outcome of this evaluation determines whether to accept or reject the offer. The offer evaluation correspond to two cases:

1. First, the receiving Gateway compares its remaining unused resources (i.e. $\hat{R}_2$ and $\hat{P}_2$) with the requirements of the offered candidate. If it has enough unused resources to host the migrated device, i.e. $\hat{R}_2 \geq r^*$ and $\hat{P}_2 \geq p^*$, it accepts the offer.

2. In the opposite case, it checks if it can reduce the SQ of other devices to be able to host the migrated note, while still improving the overall SQ. Gateway2 calculates the optimal allocation under the assumption that the candidate node is connected to it. Then it calculates $\bar{u}$, which shows the utility loss if node 4 had connected to gateway2. The offer is acceptable if the decrease in the overall utility of gateway2 is less than the gained utility of gateway1, i.e. $\overset{+}{u} > \bar{u}$.

After the offer evaluation, the initiator (i.e. gateway1) is informed the outcome. In case of an acceptance, it notifies the migration candidate (e.g. device 5). The migration candidate receives a request for connection from the new Gateway (i.e. gateway2), and it joins the new network. Last, the Gateways update the rest of the connected nodes for their updated mode of operation, given the trade.

Outcome of Migration. Following a successful trade between two Gateways, these three conditions are possible:

1. For both Gateways, the connected devices have reached their highest SQ level. This means that these two Gateways have achieved their local optimal configuration, and therefore no more trade is beneficial between them.

2. For both Gateways, the SQ level of connected devices can be improved but there are no resources left (i.e. $\hat{R}_1 \times \hat{P}_1 = 0$ and $\hat{R}_2 \times \hat{P}_2 = 0$). In this case, no more migration can help, and the Gateways can only rely on an exchange to improve the overall utility.

3. On one Gateway the devices are operating at their highest SQ level, while on the other Gateway there are not enough resources left to increase the SQ level of devices (e.g. $\hat{R}_2 \times \hat{P}_2 = 0$). In this case, the Gateways can keep migrating devices until either the system converges to cases (1) or (2), or there is no candidate to migrate.

It is worth to emphasize that in migration, the Gateways will not encounter a ping-pong effect in trading (migrating nodes between two Gateways back and forth). The intuitive reason is that a mode is migrated only if the gained utility

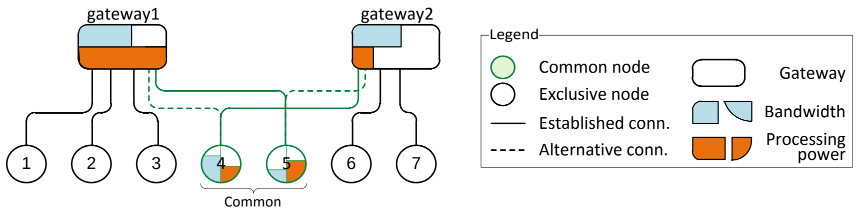


Figure 7.6: An example for exchange trade.

on the source Gateway is strictly greater than the utility loss on the destination Gateway. Hence, the reverse move is certainly disadvantageous and harmful.

B. Exchange:

Prior to proceeding to the details of the exchange move, we introduce some metrics that will be used to evaluate a trade possibility.

Definition 7.4.1. *Marginal value* is the differentiation in utility resulting from a fixed differentiation in the resources allocated to a device [46]. Loosely speaking, it is the derivative of the utility function (i.e. slope of lines in Fig. 7.3b).

The key policy of our trade scheme is: *Having a unit of resource available on the Gateway, allocate it to the device with the highest marginal value.* In other words, we allocate more resources to the IoT device that benefits the most. According to the concavity property (Property 3), the marginal value decreases (or remains constant, finitely) as the allocated resources to a node increase.

Which Gateway & Which IoT Device? If a Gateway has allocated the resources to its connected nodes such that they are all operating at their highest SQ level, it has no incentive to initiate an exchange. Therefore, an exchange is initiated by a Gateway that does not have enough resources left to increase the SQ of its nodes (i.e. $\hat{R}_g \times \hat{P}_g = 0$). In the following, we present the details of an exchange trade using the example shown in Fig. 7.6.

- **Picking the candidate node:** If the initiator Gateway (e.g. gateway1) has more than one candidate to consider for the exchange, it prioritizes the common node with the highest usage of the scarcest resource. For instance, in the example of Fig. 7.6, the scarcest resource of gateway1 is the processing power. Hence, the common node which consumes the most processing power on gateway1 is the candidate to be exchanged. The rationale is that exchanging this node might resolve the heterogeneity issue and free up more resources for other nodes to increase their SQ level (and consequently the overall utility).

- **Offer an exchange:** Once the candidate node is selected (node 5 in the example), the trade-initiating Gateway sends an exchange offer to the other one (i.e. gateway2), which includes the following information:
  - The EFC_5 set.
  - The current configuration of device 5, i.e. (u_5^*, r_5^*, p_5^*).
  - Its remaining unused resources, $\hat{R}_1$ and $\hat{P}_1$, (at least one is zero).

The recipient of the offer, i.e. gateway2, needs to first select one candidate node for exchange and then evaluate whether the acceptance of this offer is in direction of increasing the overall utility of the system (i.e. is an Kaldor-Hicks improvement).

- **Picking the candidate node:** If it has more than one candidate to exchange, gateway2 prioritizes its common node which has an opposite resource usage profile compared to the offered node. This means that if node 5 is processing intensive, gateway2 looks for a node which is bandwidth intensive, and vice versa. In our considered example shown in Fig. 7.6, the candidate node that gateway2 selects for the exchange is node 4.
- **Evaluate the offer:** Since the recipient of the offer (i.e. gateway2) has the overall information of the trade, it can make the decision to accept or reject the offer. Let $V_d(r,p)$ denote the maximum utility that device d can reach if the allocated bandwidth and processing power for it are r and p, respectively. This value can be calculated easily from the piecewise-linear utility function of the device (see Fig. 7.3b).

  Then, gateway2 needs to calculate and compare the possible change in utility of the exchanged devices:

  - First, it calculates $V_5(\hat{R}_2+r_4^*, \hat{P}_2+p_4^*)$, which describes the utility of node 5 if it connects to gateway2, replacing node 4. It considers the amount of resources that it would have after exchange, $\hat{R}_2$ and $\hat{P}_2$ in addition to the resources that node 4 would release (i.e. r_4^* and p_4^*).
  - Then, it similarly calculates $V_4(\hat{R}_1+r_5^*, \hat{P}_1+p_5^*)$.

  Finally, to evaluate the received offer, gateway2 calculates the change in the overall utility and decides about the offer according to:

$$\begin{cases} \text{accept:} & V_5(\hat{R}_2+r_4^*, \hat{P}_2+p_4^*) + V_4(\hat{R}_1+r_5^*, \hat{P}_1+p_5^*) > u_5^* + u_4^* \\ \text{reject:} & \text{otherwise} \end{cases} \tag{7.18}$$

Once the Gateways agree upon a trade (exchange or migration), the IoT device will be disconnected from one Gateway and connect to the other one. Before the disconnection, the original Gateway concludes its remainining execution tasks with respect to the traded node. The re-connection process is usually very fast and takes only a few milliseconds (6 ms for Bluetooth Low Energy). During this time, the IoT device can buffer its data (if any) and transmit it after connecting to the new Gateway. The state of process (i.e. execution context) on the Gateway is very small for the IoT applications (negligible for our ECG processing case-study presented in Section 3.3). Therefore the main communication overhead for exchange/migration is the negotiation of Gateways for trading, while this overhead does not affect the operation of IoT nodes.

7.5 Experimental Evaluation

In order to evaluate the effectiveness of our proposed mechanism, we conducted experiments and case-studies, which include real world measurements on an actual IoT device and trace driven network simulation to investigate the behavior of the system. The core of the application case study is the IoT based ECG analysis application presented in Section 3.3. The presented experimental campaign is based on the configuration of the ECG analysis application for Gateway based Edge computing systems, as it has been detailed in Section 3.3.6.

7.5.1 Experimental Setup

To conduct the experiments, a combination of experimentally derived data enhanced with nominal data from data-sheets of commercial devices is used for the values of our model parameters. Regarding the available energy of the IoT device (e_d), a battery consumption model of each IoT node is composed based on the instrumented CPU utilization. To complete the battery model of the IoT nodes, we choose a rechargeable Lithium-Ion coin cell battery with nominal capacity of up to 420 mAh [82]. We use a realistic discharge model for the battery using [258] for various values of discharge currents to evaluate the available energy for Eq. (7.4).

An ARM Cortex-M3 device is considered as the Gateway [152]. The energy consumption values were acquired by hardware measurements and profiling the execution of the ECG analysis flow for all combinations of SQ levels and processing stages to measure the values of our parameters, e.g. $C_d(\cdot)$, $p(r_{d_{ij}})$, etc. The energy consumption of ECG acquisition was calculated based on [104].

Bluetooth Low Energy (BLE) is used for communication between IoT devices and Gateway. Power consumption value of data transmission is 0.153 μW based on [209] and transmission latency is 4 $\mu s/bit$ [204]. Since BLE exploits adaptive frequency hopping mechanisms, the probability of interference is very low and even if it happens, it would be for a very short period of time. Moreover, the Gateways communicate with each other using either a different medium (wired) or a dedicated frequency band such as Sub-1 GHz (different from the band that IoT devices are transmitting their data). Therefore the interference among Gateways is avoided and would not affect the IoT devices and their battery.

7.5.2 Simulation framework

Our following experimental analysis has been conducted using a simulator, created as a part of this work, which implements all the models of the different devices of the system. The inherent distributed nature of the system under analysis, led us to create a simulator where each one of the previously described agents (Gateways and IoT nodes) corresponds to a different processing entity. In this way, message exchange and execution of our proposed resource allocation scheme can take place in parallel, thus creating a realistic interplay of the involved agents.

A very important goal was to be able to capture the topological characteristics of an examined IoT based system. In this respect, our simulation evolves in a virtual grid of devices, where in each position of the grid there can be either an IoT node or a Gateway. Using this feature, we simulate real-life conditions where communication of different nodes is affected by their distance and interference with the rest of working devices in proximity. Every node is unique and can have different different initial values in its characteristics, such as battery capacity for an IoT node or available bandwidth and processing capabilities in a Gateway.

An example of an instance of the topologies than can be mapped to our simulation framework is illustrated in Fig. 7.7. In this case, there is a 4x4 grid with 2 Gateways and 4 IoT nodes. The rest of the grid points are unoccupied, but are available for use in other scenarios. Each one of the devices, has its own characteristics, e.g. IoT node 3 has increased battery capacity compared to IoT node 2 whose battery is low. The same applies for Gateways, where A is rich in bandwidth while B in processing power.

From the point of view of the implementation, the simulator has been created as a C program running on Linux OS. Each different entity is mapped to a different process of the OS, in order to provide encapsulation and enable parallel execution. Inter-process i.e. inter-node communication is achieved by writing data packets in the shared memory of each entity. These data packets

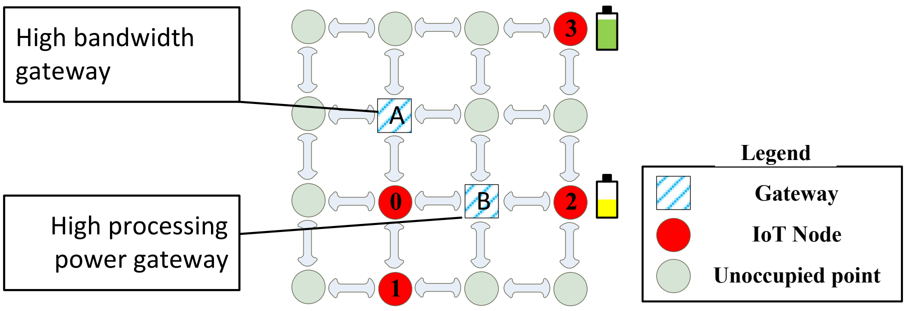


Figure 7.7: An example of an examined IoT based system.

correspond to either signals or data which are transmitted from one node to the other. The employed communication scheme is based on the inter-node signal exchange mechanisms presented in Section 4.5.5, regarding DRTRM. The same three-way handshake communication protocol has been adopted, augmented with the use of semaphores in order to help its orchestration and protect against the violation of the memory space of a process by another one.

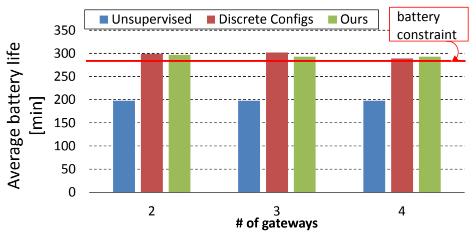
7.5.3 Results

We consider an IoT system which is located inside a big hospital room or a clinic ward. We assume a number of patients whose vital signals and mainly ECG signals are monitored by wearable IoT edge devices. The room is simulated by a 8x8 grid where a different number of Gateways and IoT Edge devices are present per scenario.

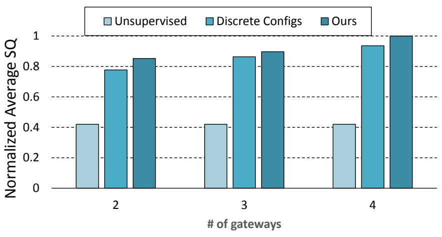
We identify three distinct scenarios where the number of patients inside the room is (i) low, (ii) medium and (iii) high. This affects the requirements of the connection to the available Gateways, leading to increased demand as the number of patients rises. Furthermore, for each scenario we examine a different number of available Gateways. The Gateways are located diagonally in the room as shown in Fig. 7.7.

The characteristics of each IoT node also differ since they are mapped to different patients. Each device has its own initial battery capacity and expected battery lifetime, accompanied by a coefficient indicating the importance of increased SQ level to the specific user.

The last critical design alternative examined in each scenario, is the algorithm employed by the Gateways in order to accomplish the binding and allocation for

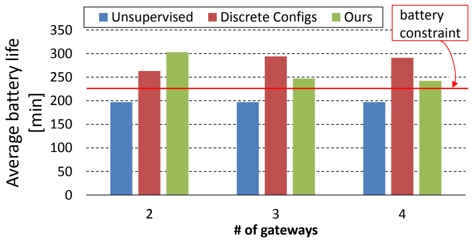


(a) Average battery lifetime.

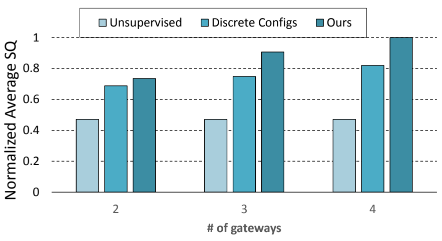


(b) Normalized average SQ level.

Figure 7.8: The scenario of low number of IoT devices.



(a) Average battery lifetime.

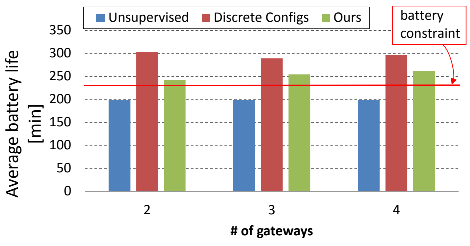


(b) Normalized average SQ level.

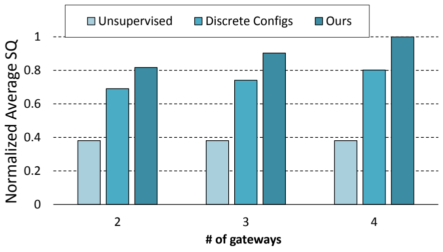
Figure 7.9: The scenario of medium number of IoT nodes.

the operation points of each IoT node. The available options are to execute the resource allocation algorithm either with discrete configuration points [189], or with our proposed scheme. In both cases, the ability for Gateways to migrate or exchange nodes is activated. Another comparison is with the scheme where devices operate in the absence of any resource management scheme, where IoT node operates in a medium SQ level and only communicate their processing outcome to the Gateway.

The results of the three examined scenarios are presented with two metrics: 1) Average battery lifetime of the IoT nodes and 2) normalized average achieved SQ level. The first examined scenario was the one with a low number of patients (i.e. IoT edge devices). Fig. 7.8a illustrates the average achieved battery lifetime of the devices. This metric is chosen since there is no global expected lifetime value for all devices but each one operates under a unique constraint. To visualize the expected battery lifetime, the red line on the figure denotes the average expected battery lifetime, taking into account all devices involved in the scenario. In both techniques for SQ management, all devices meet their battery lifetime constraint. On the contrary, the unsupervised system behaves very poorly and misses the majority of the constraints.



(a) Average battery lifetime.



(b) Normalized average SQ level.

Figure 7.10: The scenario of high number of IoT nodes.

In Fig. 7.8b, the average achieved SQ level is presented, normalized in respect to the highest achieved value in this scenario. We can see that the proposed resource allocation scheme operating on a continuous utility function, achieves better SQ levels compared to the scheme with discrete utility function in all cases. Since the number of devices is low, the gain remains low as in both schemes there are enough available resources. The gain in SQ level of the proposed scheme comes at the price of decreased average battery lifetime of IoT nodes compared to the discrete version. However, this is acceptable since the battery lifetime constraints are still met.

This presented system behavior remains the same throughout the examined scenarios with medium and high number of devices as illustrated in Fig. 7.9 and Fig. 7.10. As expected in these cases, as the number of IoT devices increases, the available resources on the Gateways become more and more scarce. Consequently, the ability of the proposed solution to fully utilize available bandwidth and processing resources of the Gateway, results in improvement in achieved SQ levels. The same overall SQ level cannot be achieved by the discrete counterpart of the resource allocation scheme, since it suffers from the fragmented utilization of its resources. These gains reach up to 22% and 24.6% in the cases of medium and high number of edge devices, respectively. In overall the gains compared to the unsupervised version of the system start from 56% and can reach up to more than 100%.

Fig. 7.11, presents the overhead of the resource management negotiations on the rest of the system communication, which corresponds to the data traffic generated by the applications running on the IoT nodes. Assuming that Tot_Mngmt_Msgs is the total number of messages exchanged for resource negotiations and Tot_App_Msgs is the total number of application related messages sent by the IoT nodes to Gateways, then the overhead in message

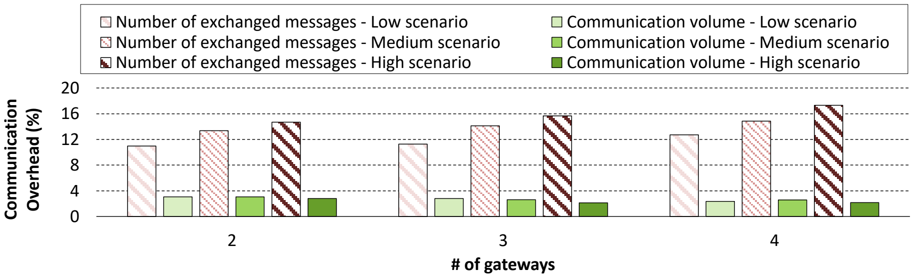


Figure 7.11: Communication overhead of distributed negotiation scheme.

count is defined as

$$\frac{Tot\_Mngmt\_Msgs}{Tot\_App\_Msgs} \times 100\% \tag{7.19}$$

while the overhead in transmitted data volume equals to

$$\frac{Sizeof(Tot\_Mngmt\_Msgs)}{Sizeof(Tot\_App\_Msgs)} \times 100\%. \tag{7.20}$$

Experiments show that the overhead in the number of exchanged messages ranges from 10% to 18% and is higher for more IoT nodes (patients) and Gateways, since more devices need to communicate in order to designate the operating points of the system. Conversely, the overhead on communication volume is almost constant and lower than 3% in all cases. To interpret this result, the differences in the structure of resource management and application related messages should be taken into account. Resource negotiation messages are small and contain only a few values as described in Section 7.4, while application messages transmitted to the Gateway can contain many values according to the SQ and off-loading level of each device, e.g. a complete heartbeat window sent for processing. As a result, the resource negotiation related messages constitute a very small fraction of the total communication volume.

7.6 Conclusions

This Chapter studied the joint problem of binding and resource allocation for IoT systems with multiple Gateways. The main characteristics of the IoT nodes is their ability to dynamically provide different Service Quality (SQ) levels to the user, aided by offloading a portion of the necessary workload to their subscribed Gateway. The binding of IoT nodes to Gateways is also dynamic and can vary according to the physical position of the IoT nodes and the workload of the

Gateway. In total, the binding and resource allocation problem determines the unique binding of IoT devices to Gateways under constraints of battery lifetime in the IoT devices and communication bandwidth and processing capability in the Gateway (for offloading purposes). According to this calculated binding, the operating configuration of the IoT node is determined, i.e. the provided SQ level and the amount of offloaded tasks to the Gateway.

The aforementioned problem is formulated as an Integer Programming problem and is shown to be NP-hard. Taking this into account, by adopting concepts of distributed resource management of DRTRM presented in Chapter 4, a resource negotiation mechanism between Gateways is proposed, which makes use of trading of IoT nodes in order to gradually optimize the provided SQ to the users. The effectiveness of the proposed approach is demonstrated using a case study of ECG processing in a personal healthcare monitoring application, as designed and implemented in Section 3.3.6. In addition, a custom built simulator is used in order to capture the dynamics of an environment where multiple IoT devices and Gateways co-exist. The experiments show that the proposed solution meets the constraints of IoT devices and Gateways, while achieving higher accumulated SQ compared both to an unsupervised system and a resource allocation scheme with discrete utility configurations.

Chapter 8

Conclusions

8.1 Summary of Main Contributions

The presented technical work can be concluded with a summary of the achievements of this thesis. The first part of the work focused on the design of embedded applications stemming from the medical domain. An ecosystem for Smart Wound Management was presented in Section 3.2, at the heart of which there is wearable device which delivers the Negative Pressure Wound Therapy and communicates with other devices in order to acquire sensed parameters of the wound. The timely design of the embedded software of this system, was facilitated by the use of an FPGA based emulation platform, which provides the technical advantage of developing the first prototype of the software, in parallel to the development of the first prototype of the hardware. This strategy laid the foundations of the final embedded software and provided the necessary time margin for its thorough validation, which led to a successful clinical validation of the complete Wound Management system on numerous patients.

Dependable operation was the main challenge of the first developed medical application. The second one, an arrhythmia detection application via the Electrocardiogram signal (Section 3.3), has critical requirements with respect to the accurate assessment of the heart physiology, in real-time. Support Vector Machines (SVM) classifier, a supervised machine learning model, was chosen as the core of the heart beats assessment logic and a methodology for its optimization was developed. The structure of the application was designed in a modular, pipelined manner in order to support the offloading of its tasks to an IoT Gateway. This feature constitutes the primary tuning knob for the run-time configuration of a system with multiple IoT devices and Gateways. More precisely, by assigning different Service Quality (SQ) on each operation mode, an optimization problem of maximizing the aggregated SQ is formulated, under the battery-related lifetime constraints of each device. The ECG analysis application was implemented, evaluated and profiled on top of Intel Galileo, a development board for IoT applications.

This evaluation showed that the execution of the SVM classifier can be the breaking point for the real-time operation of the ECG analysis application. This fact inspired us to design a HW accelerated version of the model in Section 3.4, targeting devices which integrate CPU and FPGA logic on the same Systems on Chip. High Level Synthesis (HLS) was used to speed up the development of the HW component and provide different design options with respect to its size and efficiency. Specific structural modifications of the SVM algorithm are proposed to optimize the outcome of the HLS tool, as well as an automated design space exploration methodology for the optimal designation of the tuning knobs of the tool. This exploration, relied on a set of design space pruning guidelines, which highly reduced the search latency, while achieving the coverage of a large portion of the Pareto optimal configurations of the full search space.

On the level of the IoT Gateway, we proposed that a system of many processing elements can accommodate the computational and communicational requirements of multiple IoT nodes. The allocation of resources on such many-core systems is characterized by increased computational complexity and the distribution of the decision making is required for real-time operation. In Chapter 4 of this thesis, we detailed the design of a Distributed Run-Time Resource Management (DRTRM) framework, targeting many-core systems with Network-on-Chip interconnection of the processing elements. The evaluation of the system was performed on Intel SCC many-core platform, using as input parallel versions of integral components of IoT applications, such as the SVM classifier. The experiments validated the efficiency of DRTRM, as well its capability of more efficient resource management in comparison to relevant state-of-the-art distributed resource allocation frameworks.

Having defined the structure and inter-play of agents in DRTRM, we examined in Chapter 5 the effects of the rate of incoming applications on its efficiency. We showed that a fast and resource hungry scenario of incoming applications can be its breaking point and afterwards analyzed the reasons that complicate the enforcement of a global policy regarding application admission in DRTRM. To overcome this issue, we identified the inherent dependencies of the distributed agents of DRTRM and proposed a mitigation scheme, which takes advantage of these dependencies in order to dynamically enforce policies according to the arrival rate of applications. The key tuning knob of this proposition is the scaling of the Voltage and Frequency of Controller cores, which belong to the highest level of DRTRM hierarchy. The evaluation of the proposed technique for stressing application input scenarios showed its ability to effectively mitigate the congestion on the system, while also providing gains on the overall energy consumption due to the utilized VFS techniques.

Apart from performance, we consider that the role of an IoT Gateway is critical, due to the dependency of IoT devices on its correct operation. In general, the deep integration and long operating cycles of many-core systems are expected to cause errors on the processing elements of the system. In Chapter 6, we modelled this error rate and proposed SoftRM, a fault tolerant version of DRTRM, which refrains from the provisioning of spare cores by using a dynamic, self-organizing approach in order to restore the system to a stable state, in case of manifested permanent HW errors. The evaluation of this technique, showed that self-organization in comparison to spare core provisioning, can lead to higher system throughput, while tolerating all dynamically manifested errors.

Run-time efficiency, flexibility and dependability were the foundations of the run-time management logic of an IoT Gateway. However, the Gateway based IoT system is additionally bound and affected by the interaction of the Gateways themselves. This interaction is critical since, IoT nodes are shared between gateways and this binding can lead to increased Service Quality for the user, if designated effectively. Due to this importance, as well as the fast evolving dynamics of the IoT setup, in Chapter 7 we proposed a distributed, trade based negotiation mechanism, for the Gateways to dynamically re-configure the solution of the binding problem. The evaluation of the proposed scheme, showed its potential for offering increased SQ to the user, while meeting the individual constraints of the devices and maintaining a very low overheard of the required communication between Gateways.

8.2 Future Work

The presented research work has managed to provide a number of design alternatives for the challenges described in Chapter 1, however there are still open issues to be addressed in future efforts. The Edge (or Fog) computing architecture is in its infancy and as a result it has still many aspects to be defined. Beginning from application design, while we introduced a way of application deployment taking into account the distributed architecture, the actual need is for a consistent framework or programming model, which will automate and guide application development for the Edge computing infrastructure. Eventually, this framework should also take into account the parts of the IoT application that are executed in the Cloud and aid the designer to map the application in the full stack. In general, the designation of applications that can fully exploit the multi-tiered local-Edge-Cloud infrastructure is very important in order to fully establish this new infrastructure.

On the level of the IoT Gateway, as presented in Section 1.3, the design choices are overwhelming. Consequently, a systematic methodology is required to help the system designer to determine which is the optimum hardware configuration of the Gateway for a specific IoT application. This decision is complicated by the fact that there is a straightforward trade-off between the number of deployed Gateways, their HW competency and the cost of a specific deployment. In other words, the critical decision factors are not only performance oriented but should include a cost-benefit analysis in order to have a practical gain.

With respect to Distributed Run-Time Resource Management, the key open issue is the efficient support of HW heterogeneity. To a great extent, the effectiveness of the presented DRTRM relies on the ability of the applications to support task resizing and migration without an unacceptable overhead on the performance of the application. This suggests that the application under management should have been developed in a specific manner which supports this run-time re-configuration. This design requirement is complicated when there is a need to migrate a part of the application to processing elements of different Instruction Set Architecture or totally different processing architecture, e.g. from a CPU to a DSP. To facilitate such a migration, both the run-time system and the application should be able to support it.

From the Edge system perspective, a necessity for wide scale deployments is the secure and dependable operation of IoT applications. Security is an essential parameter of any IoT system, since (i) the conveyed information are critical for the user and (ii) a security breach could physically harm the users. Still, the computationally intensive security techniques must be often executed on top of low-end, wearable, battery or intermediately powered devices. This tradeoff emphasizes the merits of Edge computing, as the Gateway can aggregate the functions related to encryption and guarantee the secure operation of the combined IoT nodes and Gateway system. Such an example deployment would be composed of IoT end nodes with no Wide Area Network connectivity, thus fully relying on the Gateway to encrypt and upload data to the Cloud.

Dependability of the Edge system is the second critical, multi-aspect success factor. Both IoT nodes and Gateway contribute to the functionality of an application and hence a consistent manner is required for them to synergistically re-organize task execution, when an error occurs in one of the involved nodes. This implies the definition of the necessary error detection and mitigation mechanisms in all layers of Edge computing as well as the capability for dynamic migration of tasks from malfunctioning to healthy nodes.

Bibliography

[1] Gartner, inc., gartner says 6.4 billion connected "things" will be in use in 2016, up 30 percent from 2015. [Online]. Available: `https://www.gartner.com/newsroom/id/3165317`, 2015.

[2] Mckinsey institute, the internet of things: Mapping the value beyond the hype. [Online]. Available: `https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/The%20Internet%20of%20Things%20The%20value%20of%20digitizing%20the%20physical%20world/Unlocking_the_potential_of_the_Internet_of_Things_Executive_summary.ashx`, 2015.

[3] Tech insights, apple iphone 7 teardown. [Online]. Available: `http://www.techinsights.com/about-techinsights/overview/blog/apple-iphone-7-teardown/`, 2016.

[4] Arm limited, arm cortex processors. [Online]. Available: `https://www.arm.com/-/media/global/products/processors/Arm-Cortex-processors-public-August-2017.pdf?revision=1d5fde81-f9f7-47ae-9c2c-ef151f974909`, 2017.

[5] Ditas consortium. data-intensive applications improvement by moving data and computation in mixed cloud/fog environments. [Online]. Available: `https://www.ditas-project.eu/`, 2018.

[6] Far-edge consortium. factory automation edge computing operating system reference implementation. [Online]. Available: `http://www.faredge.eu/`, 2018.

[7] Fora consortium. fog computing for robotics and industrial automation. [Online]. Available: `http://www.fora-etn.eu/`, 2018.

[8] Intel xeon scalable platform. [Online]. Available: `https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-scalable-platform-brief.pdf`, 2018.

[9] mf2c consortium. towards an open, secure, decentralized and coordinated fog-to-cloud management ecosystem. [Online]. Available: `http://www.mf2c-project.eu/`, 2018.

[10] Prestocloud consortium. proactive cloud resources management at the edge for efficient real-time big data processing. [Online]. Available: `http://prestocloud-project.eu/new/`, 2018.

[11] Smith & nephew, pico single use negative pressure wound therapy. [Online]. Available: `https://www.smith-nephew.com/key-products/advanced-wound-management/pico/`, 2018.

[12] Smith & nephew, renasys negative pressure wound therapy. [Online]. Available: `https://www.smith-nephew.com/key-products/advanced-wound-management/renasys/`, 2018.

[13] Smith & nephew, renasys touch. [Online]. Available: `https://www.smith-nephew.com/key-products/advanced-wound-management/renasystouch/`, 2018.

[14] Tegra x1. [Online]. Available: `https://developer.nvidia.com/content/tegra-x1`, 2018.

[15] Zynq-7000 all programmable soc. [Online]. Available: `https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`, 2018.

[16] Abiodun, A. S., Anisi, M. H., Ali, I., Akhunzada, A., and Khan, M. K. Reducing power consumption in wireless body area networks: A novel data segregation and classification technique. *IEEE Consumer Electronics Magazine 6*, 4 (2017), 38–47.

[17] Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. *Learning from data*, vol. 4. AMLBook New York, NY, USA:, 2012.

[18] Al Faruque, M. A., Krist, R., and Henkel, J. Adam: run-time agent-based distributed application mapping for on-chip communication. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE* (2008).

[19] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials 17*, 4 (2015), 2347–2376.

[20] Anagnostopoulos, I., Bartzas, A., Kathareios, G., and Soudris, D. A divide and conquer based distributed run-time mapping methodology for many-core platforms. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2012), EDA Consortium, pp. 111–116.

[21] Anagnostopoulos, I., Tsoutsouras, V., Bartzas, A., and Soudris, D. Distributed run-time resource management for malleable applications on many-core platforms. In *Proceedings of the 50th Annual Design Automation Conference* (2013), ACM, p. 168.

[22] Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International Workshop on Ambient Assisted Living* (2012), Springer, pp. 216–223.

[23] Aoyama, T., Ishikawa, K.-I., Kimura, Y., Matsufuru, H., Sato, A., Suzuki, T., and Torii, S. First application of lattice qcd to pezy-sc processor. *Procedia Computer Science 80* (2016), 1418–1427.

[24] Apelqvist, J., and Larsson, J. What is the most effective way to reduce incidence of amputation in the diabetic foot? *Diabetes/metabolism research and reviews 16*, S1 (2000).

[25] Azad, S. P., Niazmand, B., Raik, J., Jervan, G., and Hollstein, T. Holistic approach for fault-tolerant network-on-chip based many-core systems. *arXiv preprint arXiv:1601.07089* (2016).

[26] Azariadi, D., Tsoutsouras, V., Xydis, S., and Soudris, D. Ecg signal analysis and arrhythmia detection on iot wearable medical devices. In *Modern Circuits and Systems Technologies (MOCAST), 2016 5th International Conference on* (2016), IEEE, pp. 1–4.

[27] Azimi, I., Anzanpour, A., Rahmani, A. M., Pahikkala, T., Levorato, M., Liljeberg, P., and Dutt, N. Hich: Hierarchical fog-assisted computing architecture for healthcare iot. *ACM Transactions on Embedded Computing Systems (TECS) 16*, 5s (2017), 174.

[28] Baker, S. B., Xiang, W., and Atkinson, I. Internet of things for smart healthcare: Technologies, challenges, and opportunities. *IEEE Access 5* (2017), 26521–26544.

[29] Banerjee, S., and Mitra, M. Application of cross wavelet transform for ecg pattern analysis and classification. *IEEE transactions on instrumentation and measurement 63*, 2 (2014), 326–333.

[30] Barbalace, A., Ravindran, B., and Katz, D. Popcorn: a replicated-kernel os based on linux. In *Proceedings of the Linux Symposium, Ottawa, Canada* (2014).

[31] Barbancho, J., Leon, C., Molina, J., and Barbancho, A. Giving neurons to sensors. qos management in wireless sensors networks. In *IEEE Conference on Emerging Technologies and Factory Automation* (2006), IEEE, pp. 594–597.

[32] Bartolini, A., Sadri, M., Furst, J., Coskun, A. K., and Benini, L. Quantifying the impact of frequency scaling on the energy efficiency of the single-chip cloud computer. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012* (2012), IEEE.

[33] Bartus, C. L., and Margolis, D. J. Reducing the incidence of foot ulceration and amputation in diabetes. *Current diabetes reports 4*, 6 (2004), 413–418.

[34] Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A., and Singhania, A. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 29–44.

[35] Beguelin, A., Seligman, E., and Stephan, P. Application level fault tolerance in heterogeneous networks of workstations. *Journal of Parallel and Distributed Computing 43*, 2 (1997), 147–155.

[36] Ben-Asher, Y., and Rotem, N. Automatic memory partitioning: increasing memory parallelism via data structure partitioning. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (2010), ACM, pp. 155–162.

[37] Bhattacharya, S., Saifullah, A., Lu, C., and Roman, G.-C. Multi-application deployment in shared sensor networks based on quality of monitoring. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium* (2010), pp. 259–268.

[38] Bhimani, J., Mi, N., Leeser, M., and Yang, Z. Fim: performance prediction for parallel computation in iterative data processing applications. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on* (2017), IEEE, pp. 359–366.

[39] Bienia, C., Kumar, S., Singh, J. P., and Li, K. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of*

the 17th international conference on Parallel architectures and compilation techniques (2008), ACM, pp. 72–81.

[40] Bilal, K., Khalid, O., Erbad, A., and Khan, S. U. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks 130* (2018), 94–120.

[41] Bjerregaard, T., and Mahadevan, S. A survey of research and practices of network-on-chip. *ACM Computing Surveys (CSUR) 38*, 1 (2006), 1.

[42] Bohnenstiehl, B., Stillmaker, A., Pimentel, J., Andreas, T., Liu, B., Tran, A., Adeagbo, E., and Baas, B. A 5.8 pj/op 115 billion ops/sec, to 1.78 trillion ops/sec 32nm 1000-processor array. In *VLSI Circuits (VLSI-Circuits), 2016 IEEE Symposium on* (2016), IEEE, pp. 1–2.

[43] Bolchini, C., Carminati, M., and Miele, A. Self-adaptive fault tolerance in multi-/many-core systems. *Journal of Electronic Testing 29*, 2 (2013), 159–175.

[44] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. Fog computing and its role in the Internet of Things. In *workshop on Mobile cloud computing (MCC)* (2012), pp. 13–16.

[45] Bortolotti, D., Mangia, M., Bartolini, A., Rovatti, R., Setti, G., and Benini, L. Energy-aware bio-signal compressed sensing reconstruction on the wbsn-gateway. *IEEE Transactions on Emerging Topics in Computing* (2016).

[46] Boyd, S., and Vandenberghe, L. *Convex optimization.* Cambridge university press, 2004.

[47] Boyd-Wickizer, S., Chen, H., Chen, R., Mao, Y., Kaashoek, M. F., Morris, R., Pesterev, A., Stein, L., Wu, M., Dai, Y.-h., et al. Corey: An operating system for many cores. In *OSDI* (2008), vol. 8, pp. 43–57.

[48] Braojos, R., Beretta, I., Constantin, J., Burg, A., and Atienza, D. A wireless body sensor network for activity monitoring with low transmission overhead. In *IEEE International Conference on Embedded and Ubiquitous Computing (EUC)* (2014), pp. 265–272.

[49] Cachin, C., Guerraoui, R., and Rodrigues, L. *Introduction to reliable and secure distributed programming.* Springer Science & Business Media, 2011.

[50] Cadambi, S., Durdanovic, I., Jakkula, V., Sankaradass, M., Cosatto, E., Chakradhar, S., and Graf, H. P. A massively parallel fpga-based coprocessor for support vector machines. In *Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on* (2009), IEEE, pp. 115–122.

[51] Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Czajkowski, T., Brown, S. D., and Anderson, J. H. Legup: An open-source high-level synthesis tool for fpga-based processor/accelerator systems. *ACM Transactions on Embedded Computing Systems (TECS) 13*, 2 (2013), 24.

[52] Chandra, T. D., and Toueg, S. Unreliable failure detectors for reliable distributed systems. *J. ACM 43*, 2 (1996), 225–267.

[53] Chang, C.-C., and Lin, C.-J. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST) 2*, 3 (2011), 27.

[54] Chen, D., and Varshney, P. K. Qos support in wireless sensor networks: A survey. In *International conference on wireless networks* (2004), vol. 233, pp. 1–7.

[55] Chen, X. Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems 26*, 4 (2015), 974–983.

[56] Chen, Z., and Marculescu, D. Distributed reinforcement learning for power limited many-core system performance optimization. In *Proceedings of the 2015 DATE* (2015), EDA Consortium, pp. 1521–1526.

[57] Chiang, M., and Zhang, T. Fog and iot: An overview of research opportunities. *IEEE Internet of Things Journal 3*, 6 (2016), 854–864.

[58] Cho, S.-w., and Goel, A. Pricing for fairness: distributed resource allocation for multiple objectives. In *ACM symposium on Theory of computing* (2006), pp. 197–204.

[59] Chou, C.-L., and Marculescu, R. Farm: Fault-aware resource management in noc-based multiprocessor platforms. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011* (2011), IEEE, pp. 1–6.

[60] Christin, D., Reinhardt, A., Mogre, P., and Steinmetz, R. *Wireless Sensor Networks and the Internet of Things: Selected Challenges*, 2009.

[61] Chu, P. P. *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version.* John Wiley & Sons, 2011.

[62] Chung, H., Kang, M., and Cho, H.-D. Heterogeneous multi-processing solution of exynos 5 octa with arm® big. little™ technology. *Samsung White Paper* (2012).

[63] Cilardo, A., and Gallo, L. Improving multibank memory access parallelism with lattice-based partitioning. *ACM Transactions on Architecture and Code Optimization (TACO) 11*, 4 (2015), 45.

[64] Cilardo, A., and Gallo, L. Interplay of loop unrolling and multidimensional memory partitioning in hls. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)* (2015), IEEE, pp. 163–168.

[65] Clifford, G. D., Azuaje, F., and McSharry, P. Advanced methods and tools for ecg data analysis. norwood, ma, usa: Artech house, 2006.

[66] Coleman, J. L. Efficiency, utility, and wealth maximization. *Hofstra L. Rev. 8* (1979), 509.

[67] Coley, G. Beagleboard system reference manual. *BeagleBoard. org, December* (2009), 81.

[68] Colmenares, J. A., Eads, G., Hofmeyr, S., Bird, S., Moretó, M., Chou, D., Gluzman, B., Roman, E., Bartolini, D. B., Mor, N., et al. Tessellation: refactoring the os around explicit resource containers with continuous adaptation. In *Proceedings of the 50th Annual Design Automation Conference* (2013), ACM, p. 76.

[69] Cong, J., Jiang, W., Liu, B., and Zou, Y. Automatic memory partitioning and scheduling for throughput and power optimization. *ACM Transactions on Design Automation of Electronic Systems (TODAES) 16*, 2 (2011), 15.

[70] Cong, J., Li, P., Xiao, B., and Zhang, P. An optimal microarchitecture for stencil computation acceleration based on non-uniform partitioning of data reuse buffers. In *Proceedings of the 51st Annual Design Automation Conference* (2014), ACM, pp. 1–6.

[71] Cong, J., Zhang, P., and Zou, Y. Optimizing memory hierarchy allocation with loop transformations for high-level synthesis. In *Proceedings of the 49th Annual Design Automation Conference* (2012), ACM, pp. 1233–1238.

[72] Cortes, C., and Vapnik, V. Support-vector networks. *Machine learning 20*, 3 (1995), 273–297.

[73] Cui, Y., Zhang, W., and Yu, H. Decentralized agent based re-clustering for task mapping of tera-scale network-on-chip system. In *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on* (2012), IEEE, pp. 2437–2440.

[74] Das, A., Kumar, A., and Veeravalli, B. Communication and migration energy aware design space exploration for multicore systems with intermittent faults. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2013), EDA Consortium, pp. 1631–1636.

[75] Das, A., Kumar, A., and Veeravalli, B. Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems. In *Proceedings of the Conference on Design, Automation and Test in Europe* (2013), EDA Consortium, pp. 689–694.

[76] Das, A., Kumar, A., Veeravalli, B., Bolchini, C., and Miele, A. Combined dvfs and mapping exploration for lifetime and soft-error susceptibility improvement in mpsocs. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014* (2014), IEEE, pp. 1–6.

[77] Daubechies, I. The wavelet transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theory 36*, 5 (1990), 961–1005.

[78] Daubechies, I., et al. *Ten lectures on wavelets*, vol. 61. SIAM, 1992.

[79] Dean, J., and Ghemawat, S. Mapreduce: a flexible data processing tool. *Communications of the ACM 53*, 1 (2010), 72–77.

[80] Desell, T., El Maghraoui, K., and Varela, C. A. Malleable applications for scalable high performance computing. *Cluster Computing 10*, 3 (2007), 323–337.

[81] Digilent's ZedBoard Zynq, F. Dev. board documentation.

[82] Dittrich, T., Menachem, C., Herzel, Y., and Lou, A. Lithium batteries for wireless sensor networks. Tech. rep., Tadiran Batteries, 2012.

[83] Donyanavard, B., Mück, T., Sarma, S., and Dutt, N. Sparta: runtime task allocation for energy efficient heterogeneous many-cores. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (2016), ACM, p. 27.

[84] Downey, A. B. A model for speedup of parallel programs. Tech. rep., CALIFORNIA UNIV BERKELEY COMPUTER SCIENCE DIV, 1997.

[85] Du, J., Zhao, L., Feng, J., and Chu, X. Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee. *IEEE Transactions on Communications* (2017).

[86] Dunkels, A., et al. The lwip tcp/ip stack. *lwIP–A LightWeight TCP/IP Stack* (2004).

[87] Dziurzanski, P., Singh, A. K., and Indrusiak, L. S. Feedback-based admission control for hard real-time task allocation under dynamic workload on many-core systems. In *International Conference on Architecture of Computing Systems* (2016), Springer, pp. 157–169.

[88] Ebi, T., et al. Tape: Thermal-aware agent-based power econom multi/many-core architectures. In *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on* (2009), IEEE, pp. 302–309.

[89] Fattah, M., Daneshtalab, M., Liljeberg, P., and Plosila, J. Smart hill climbing for agile dynamic mapping in many-core systems. In *Proceedings of the 50th Annual Design Automation Conference* (2013), ACM, p. 39.

[90] Fattah, M., Palesi, M., Liljeberg, P., Plosila, J., and Tenhunen, H. Shifa: System-level hierarchy in run-time fault-aware management of many-core systems. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE* (2014), IEEE, pp. 1–6.

[91] Fattah, M., Palesi, M., Liljeberg, P., Plosila, J., and Tenhunen, H. Shifa: System-level hierarchy in run-time fault-aware management of many-core systems. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE* (2014), IEEE, pp. 1–6.

[92] Feist, T. Vivado design suite. *White Paper 5* (2012).

[93] Feitelson, D. G., and Rudolph, L. Toward convergence in job schedulers for parallel supercomputers. In *Proc. of JSSPP* (1996), Springer-Verlag, pp. 1–26.

[94] Fleig, T., Mattes, O., and Karl, W. Evaluation of adaptive memory management techniques on the tilera tile-gx platform. In *Architecture of Computing Systems (ARCS), 2014 Workshop Proceedings* (2014), VDE, pp. 1–8.

[95] Furber, S. B., Galluppi, F., Temple, S., and Plana, L. A. The spinnaker project. *Proceedings of the IEEE 102*, 5 (2014), 652–665.

[96] Furey, T. S., Cristianini, N., Duffy, N., Bednarski, D. W., Schummer, M., and Haussler, D. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics 16*, 10 (2000), 906–914.

[97] Gajski, D. D., Dutt, N. D., Wu, A. C., and Lin, S. Y. *High—Level Synthesis: Introduction to Chip and System Design.* Springer Science & Business Media, 2012.

[98] Ge, Y., Malani, P., and Qiu, Q. Distributed task migration for thermal management in many-core systems. In *Proceedings of the 47th Design Automation Conference* (2010), ACM, pp. 579–584.

[99] Goel, A., and Nazerzadeh, H. Price-based protocols for fair resource allocation: Convergence time analysis and extension to leontief utilities. *ACM Transactions on Algorithms (TALG) 10*, 2 (2014).

[100] Greenhalgh, P. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White paper 17* (2011).

[101] Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems 29*, 7 (2013), 1645–1660.

[102] Gwennap, L. Thunderx rattles server market. *Microprocessor Report 29*, 6 (2014), 1–4.

[103] Haghbayan, M.-H., Miele, A., Rahmani, A. M., Liljeberg, P., and Tenhunen, H. A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016* (2016), IEEE, pp. 854–857.

[104] Hann, M. W. Ultra low power, 18 bit precision ecg data acquisition system, June 2013.

[105] Hartman, A. S., and Thomas, D. E. Lifetime improvement through runtime wear-based task mapping. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis* (New York, NY, USA, 2012), CODES+ISSS '12, ACM.

[106] Hassanalieragh, M., Page, A., Soyata, T., Sharma, G., Aktas, M., Mateos, G., Kantarci, B., and Andreescu, S. Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges. In *IEEE International Conference on Services Computing (SCC)* (2015), pp. 285–292.

[107] Haubelt, C., Koch, D., Reimann, F., Streichert, T., and Teich, J. Reconets design methodology for embedded systems consisting of small networks of reconfigurable nodes and connections. In *Dynamically Reconfigurable Systems*. Springer, 2010, pp. 223–243.

[108] He, R., Wang, K., Li, Q., Yuan, Y., Zhao, N., Liu, Y., and Zhang, H. A novel method for the detection of r-peaks in ecg based on k-nearest neighbors and particle swarm optimization. *EURASIP Journal on Advances in Signal Processing 2017*, 1 (2017), 82.

[109] Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., and Scholkopf, B. Support vector machines. *IEEE Intelligent Systems and their Applications 13*, 4 (1998), 18–28.

[110] Heinzelman, W. B., Murphy, A. L., Carvalho, H. S., and Perillo, M. A. Middleware to support sensor network applications. *IEEE Network 18*, 1 (2004), 6–14.

[111] Henkel, J., Bauer, L., Dutt, N., Gupta, P., Nassif, S., Shafique, M., Tahoori, M., and Wehn, N. Reliable on-chip systems in the nano-era: lessons learnt and future trends. In *Proc. of the 50th Annual Design Automation Conference* (2013), ACM.

[112] Henkel, J., Bauer, L., Zhang, H., Rehman, S., and Shafique, M. Multi-layer dependability: From microarchitecture to application level. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)* (2014), IEEE, pp. 1–6.

[113] Hossain, M. S., and Muhammad, G. Cloud-assisted industrial internet of things (iiot)–enabled framework for health monitoring. *Computer Networks 101* (2016), 192–202.

[114] Howard, J., Dighe, S., Hoskote, Y., Vangal, S., Finan, D., Ruhl, G., Jenkins, D., Wilson, H., Borkar, N., Schrom, G., et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International* (2010), IEEE, pp. 108–109.

[115] Howard, J., Dighe, S., Hoskote, Y., Vangal, S., Finan, D., Ruhl, G., Jenkins, D., Wilson, H., Borkar, N., Schrom, G., et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International* (2010), IEEE, pp. 108–109.

[116] Hu, J., and Marculescu, R. Energy-and performance-aware mapping for regular noc architectures. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 24*, 4 (2005), 551–562.

[117] Hu, P., Dhelim, S., Ning, H., and Qiu, T. Survey on fog computing: architecture, key technologies, applications and open issues. *Journal of Network and Computer Applications* (2017).

[118] Huang, C., Leavitt, T., Bayer, L. R., and Orgill, D. P. Effect of negative pressure wound therapy on wound healing. *Current problems in surgery 51*, 7 (2014), 301–331.

[119] Huang, D., Wang, P., and Niyato, D. A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications 11*, 6 (2012), 1991–1995.

[120] Ipek, E., De Supinski, B. R., Schulz, M., and McKee, S. A. An approach to performance prediction for parallel applications. In *European Conference on Parallel Processing* (2005), Springer, pp. 196–205.

[121] Izosimov, V., Polian, I., Pop, P., Eles, P., and Peng, Z. Analysis and optimization of fault-tolerant embedded systems with hardened processors. In *2009 Design, Automation & Test in Europe Conference & Exhibition* (2009), IEEE.

[122] Jeffers, J., and Reinders, J. *Intel Xeon Phi coprocessor high-performance programming.* Newnes, 2013.

[123] Kalray, S. Kalray mppa manycore 256, 2014.

[124] Kanellakis, P. C., and Shvartsman, A. A. *Fault-tolerant parallel computation*, vol. 401. Springer Science & Business Media, 2013.

[125] Kapadia, N., and Pasricha, S. A runtime framework for robust application scheduling with adaptive parallelism in the dark-silicon era. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2017).

[126] Katz, D., Barbalace, A., Ansary, S., Ravichandran, A., and Ravindran, B. Thread migration in a replicated-kernel os. In *Distributed*

Computing Systems (ICDCS), 2015 IEEE 35th International Conference on (2015), IEEE.

[127] Khalili, F., and Zarandi, H. R. A reliability-aware multi-application mapping technique in networks-on-chip. In *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (2013), IEEE.

[128] Kim, S. Nested game-based computation offloading scheme for mobile cloud IoT systems. *Journal on Wireless Communications and Networking 2015*, 1 (2015), 1–11.

[129] Kobbe, S., Bauer, L., and Henkel, J. Adaptive on-the-fly application performance modeling for many cores. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition* (2015), EDA Consortium.

[130] Kobbe, S., Bauer, L., Lohmann, D., Schröder-Preikschat, W., and Henkel, J. Distrm: distributed resource management for on-chip many-core systems. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis* (2011), ACM, pp. 119–128.

[131] Koren, I., and Krishna, C. M. *Fault-tolerant systems.* Morgan Kaufmann, 2010.

[132] Kounev, S., Brosig, F., Huber, N., and Reussner, R. Towards self-aware performance and resource management in modern service-oriented systems. In *Services Computing (SCC), 2010 IEEE International Conference on* (2010).

[133] Kovachev, D., Yu, T., and Klamma, R. Adaptive computation offloading from mobile devices into the cloud. In *International Symposium on Parallel and Distributed Processing with Applications* (2012), pp. 784–791.

[134] Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE 103*, 1 (2015), 14–76.

[135] Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B. A survey of computation offloading for mobile systems. *Mobile Networks and Applications 18*, 1 (2013), 129–140.

[136] Kurian, G., Miller, J. E., Psota, J., Eastep, J., Liu, J., Michel, J., Kimerling, L. C., and Agarwal, A. Atac: a 1000-core cache-coherent processor with on-chip optical network. In *Proceedings of the*

19th international conference on Parallel architectures and compilation techniques (2010), ACM, pp. 477–488.

[137] Kwak, J., Kim, Y., Lee, J., and Chong, S. Dream: dynamic resource and task allocation for energy minimization in mobile cloud systems. *IEEE Journal on Selected Areas in Communications 33*, 12 (2015), 2510–2523.

[138] Lamport, L., et al. Paxos made simple. *ACM Sigact News 32*, 4 (2001), 18–25.

[139] Lattuada, M., Pilato, C., Tumeo, A., and Ferrandi, F. Performance modeling of parallel applications on mpsocs. In *System-on-Chip, 2009. SOC 2009. International Symposium on* (2009), IEEE, pp. 064–067.

[140] Lee, J., Nicopoulos, C., Lee, H. G., Panth, S., Lim, S. K., and Kim, J. Isonet: Hardware-based job queue management for many-core architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 21*, 6 (2013), 1080–1093.

[141] Li, L., Li, S., and Zhao, S. Qos-aware scheduling of services-oriented internet of things. *IEEE Transactions on Industrial Informatics 10*, 2 (2014), 1497–1505.

[142] Li, P., Wang, Y., Zhang, P., Luo, G., Wang, T., and Cong, J. Memory partitioning and scheduling co-optimization in behavioral synthesis. In *Proceedings of the International Conference on Computer-Aided Design* (2012), ACM, pp. 488–495.

[143] Li, X., Qin, J., and Bernstein, J. B. Compact modeling of mosfet wearout mechanisms for circuit-reliability simulation. *IEEE Transactions on Device and Materials Reliability 8*, 1 (2008), 98–121.

[144] Li, Z., Wang, C., and Xu, R. Computation offloading to save energy on handheld devices: a partition scheme. In *International conference on Compilers, architecture, and synthesis for embedded systems* (2001), pp. 238–246.

[145] Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., and Zhao, W. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal 4*, 5 (2017), 1125–1142.

[146] Liu, H., and Carloni, L. P. On learning-based methods for design-space exploration with high-level synthesis. In *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013* (2013), pp. 50:1–50:7.

[147] Liu, L., Wu, C., Deng, C., Yin, S., Wu, Q., Han, J., and Wei, S. A flexible energy-and reliability-aware application mapping for noc-based reconfigurable architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2015).

[148] López, G., Custodio, V., and Moreno, J. I. Lobin: E-textile and wireless-sensor-network-based platform for healthcare monitoring in future hospital environments. *IEEE Transactions on Information Technology in Biomedicine 14*, 6 (2010), 1446–1458.

[149] Luz, E. J. d. S., Schwartz, W. R., Cámara-Chávez, G., and Menotti, D. Ecg-based heartbeat classification for arrhythmia detection: A survey. *Computer methods and programs in biomedicine 127* (2016), 144–164.

[150] Ma, Q., Parkes, D. C., and Welsh, M. D. A utility-based approach to bandwidth allocation and link scheduling in wireless networks. In *International Workshop on Agent Technology for Sensor Networks (ATSN-07)* (2007).

[151] Maguluri, S. T., Srikant, R., and Ying, L. Heavy traffic optimal resource allocation algorithms for cloud computing clusters. *Performance Evaluation 81* (2014), 20–39.

[152] Mainetti, L., Patrono, L., and Vilei, A. Evolution of wireless sensor networks towards the internet of things: A survey. In *International Conference on Software, Telecommunications and Computer Networks (SoftCOM)* (2011), pp. 1–6.

[153] Mao, Y., and Zhang, J. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal of Solid-State Circuits 51*, 3 (2016), 712–723.

[154] Marcus, F. I., Ruskin, J. N., and Surawicz, B. Arrhythmias. *Journal of the American College of Cardiology 10*, 2 (1987), 66A–72A.

[155] Mariani, G., Palermo, G., Zaccaria, V., and Silvano, C. Arte: An application-specific run-time management framework for multi-cores based on queuing models. *Parallel Computing 39*, 9 (2013), 504–519.

[156] Mark, R., and Moody, G. Mit-bih arrhythmia database,. Available online from: `http://www.physionet.org/physiobank/database/mitdb/`.

[157] Martis, R. J., Chakraborty, C., and Ray, A. K. Wavelet-based machine learning techniques for ecg signal analysis. In *Machine Learning in Healthcare Informatics*. Springer, 2014, pp. 25–45.

[158] Mattson, T., and van der Wijngaart, R. Rcce: a small library for many-core communication. *Intel Corporation, May* (2010).

[159] Mehmood, N., Hariz, A., Templeton, S., Voelcker, N. H., et al. *An Innovatinve Sensing Technology for Chronic Wound Monitoring*. PhD thesis, Engineers Australia Pty Limited, 2014.

[160] Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F., and Boutaba, R. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials 18*, 1 (2016), 236–262.

[161] Mohammed, J., Lung, C.-H., Ocneanu, A., Thakral, A., Jones, C., and Adler, A. Internet of things: Remote patient monitoring using web services and cloud computing. In *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE* (2014), IEEE, pp. 256–263.

[162] Munk, P., Alhakeem, M. S., Lisicki, R., Parzyjegla, H., Richling, J., and Heiss, H.-U. Toward a fault-tolerance framework for cots many-core systems. In *Dependable Computing Conference (EDCC), 2015 Eleventh European* (2015), IEEE.

[163] Murphy, A., and Heinzelman, W. Milan: Middleware linking applications and networks. Tech. rep., Jan. 2003.

[164] Nacci, A. A., Rana, V., Bruschi, F., Sciuto, D., Beretta, I., and Atienza, D. A high-level synthesis flow for the implementation of iterative stencil loop algorithms on fpga devices. In *Proceedings of the 50th Annual Design Automation Conference* (2013), ACM, p. 52.

[165] Nahar, B., and Meyer, B. H. Rotr: Rotational redundant task mapping for fail-operational mpsocs. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), 2015 IEEE International Symposium on* (2015), IEEE, pp. 21–28.

[166] Networking, C. V. Cisco global cloud index: Forecast and methodology, 2015-2020. white paper. *Cisco Public, San Jose* (2016).

[167] Nollet, V., Marescaux, T., Avasare, P., Verkest, D., and Mignolet, J.-Y. Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles. In *Design, Automation and Test in Europe, 2005. Proceedings* (2005), IEEE, pp. 234–239.

[168] Oddi, G., Pietrabissa, A., Priscoli, F. D., Facchinei, F., Palagi, L., and Lanna, A. A QoE-aware dynamic bandwidth allocation algorithm based on game theory. In *Mediterranean Conference on Control and Automation (MED)* (2015), pp. 979–985.

[169] Ojha, D. K., and Subashin, M. Analysis of electrocardiograph (ecg) signal for the detection of abnormalities using matlab. *World Academy of Science, Engineering and Technology International Journal of Medical, Health, Biomedical and Pharmaceutical Engineering 8*, 2 (2014).

[170] Olofsson, A. Epiphany-v: A 1024 processor 64-bit risc system-on-chip. *arXiv preprint arXiv:1610.01832* (2016).

[171] Olofsson, A., Nordström, T., and Ul-Abdin, Z. Kickstarting high-performance energy-efficient manycore architectures with epiphany. In *Signals, Systems and Computers, 2014 48th Asilomar Conference on* (2014), IEEE, pp. 1719–1726.

[172] Orrù, G., Pettersson-Yeo, W., Marquand, A. F., Sartori, G., and Mechelli, A. Using support vector machine to identify imaging biomarkers of neurological and psychiatric disease: a critical review. *Neuroscience & Biobehavioral Reviews 36*, 4 (2012), 1140–1152.

[173] Ota, K., Dao, M. S., Mezaris, V., and De Natale, F. G. Deep learning for mobile multimedia: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 13*, 3s (2017), 34.

[174] Papadonikolakis, M., and Bouganis, C.-S. A novel fpga-based svm classifier. In *Field-Programmable Technology (FPT), 2010 International Conference on* (2010), IEEE, pp. 283–286.

[175] Parandeh-Afshar, H., Verma, A. K., Brisk, P., and Ienne, P. Improving FPGA performance for carry-save arithmetic. *IEEE Trans. VLSI Syst. 18*, 4 (2010), 578–590.

[176] Pathania, A., Venkataramani, V., Shafique, M., Mitra, T., and Henkel, J. Distributed fair scheduling for many-cores. In *Proceedings of the 2016 Conference on Design, Automation & Test in Europe* (2016), EDA Consortium, pp. 379–384.

[177] Pathania, A., Venkataramani, V., Shafique, M., Mitra, T., and Henkel, J. Distributed scheduling for many-cores using cooperative game theory. In *Design Automation Conference (DAC), 2016 53nd ACM/EDAC/IEEE* (2016), IEEE, pp. 1–6.

[178] Pathania, A., Venkataramani, V., Shafique, M., Mitra, T., and Henkel, J. Optimal greedy algorithm for many-core scheduling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2016).

[179] Raghunathan, B., and Garg, S. Job arrival rate aware scheduling for asymmetric multi-core servers in the dark silicon era. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis* (2014), CODES '14, pp. 14:1–14:9.

[180] Rahimi, A., Benini, L., and Gupta, R. K. Variability mitigation in nanometer cmos integrated systems: A survey of techniques from circuits to software. *Proceedings of the IEEE 104*, 7 (2016), 1410–1448.

[181] Ramachandran, S., and Mueller, F. Distributed job allocation for large-scale manycores. In *International Conference on High Performance Computing* (2016), Springer, pp. 404–425.

[182] Ramon, M. C. Intel galileo and intel galileo gen 2. In *Intel® Galileo and Intel® Galileo Gen 2*. Springer, 2014, pp. 1–33.

[183] Ramos, S., and Hoefler, T. Modeling communication in cache-coherent smp systems: a case-study with xeon phi. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing* (2013), ACM, pp. 97–108.

[184] Raphael, C. Why edge computing is crucial for the IoT. Online: `http://www.rtinsights.com/why-edge-computing-and-analytics-is-crucial-for-the-iot/`, 2016.

[185] Rhoden, B., Klues, K., Zhu, D., and Brewer, E. Improving per-node efficiency in the datacenter with new os abstractions. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (2011), ACM, p. 25.

[186] Sabin, G., Lang, M., and Sadayappan, P. Moldable parallel job scheduling using job efficiency: An iterative approach. In *Workshop on Job Scheduling Strategies for Parallel Processing* (2006), Springer, pp. 94–114.

[187] Samie, F., Bauer, L., and Henkel, J. IoT Technologies for Embedded Computing: A Survey. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)* (2016).

[188] Samie, F., Tsoutsouras, V., Xydis, S., Bauer, L., Soudris, D., and Henkel, J. Computation Offloading Management and Resource Allocation for Low-power IoT Edge Devices. In *IEEE 3rd World Forum on Internet of Things (WF-IoT)* (2016).

[189] Samie, F., Tsoutsouras, V., Xydis, S., Bauer, L., Soudris, D., and Henkel, J. Distributed QoS Management for Internet of Things under Resource Constraints. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)* (2016), IEEE Press.

[190] Sardellitti, S., Scutari, G., and Barbarossa, S. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks 1*, 2 (2015), 89–103.

[191] Satyanarayanan, M. The emergence of edge computing. *Computer 50*, 1 (2017), 30–39.

[192] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing 8*, 4 (2009).

[193] Schäfer, B. C., and Wakabayashi, K. Divide and conquer high-level synthesis design space exploration. *ACM Trans. Design Autom. Electr. Syst. 17*, 3 (2012), 29.

[194] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural networks 61* (2015), 85–117.

[195] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., et al. Larrabee: a many-core x86 architecture for visual computing. In *ACM Transactions on Graphics (TOG)* (2008), vol. 27, ACM, p. 18.

[196] Semiconductor, D. Da14580 low power bluetooth smart soc. *DA14580 datasheet, February* (2014).

[197] Shafique, M., Axer, P., Borchert, C., Chen, J.-J., Chen, K.-H., Döbel, B., Ernst, R., Härtig, H., Heinig, A., Kapitza, R., et al. Multi-layer software reliability for unreliable hardware. *it-Information Technology 57*, 3 (2015), 170–180.

[198] Shafique, M., and Henkel, J. Agent-based distributed power management for kilo-core processors. In *Proceedings of the International Conference on Computer-Aided Design* (2013), IEEE Press.

[199] Shafique, M., Ivanov, A., Vogel, B., and Henkel, J. Scalable power management for on-chip systems with malleable applications. *IEEE Transactions on Computers 65*, 11 (2016), 3398–3412.

[200] Sheng, Z., Mahapatra, C., Leung, V., Chen, M., and Sahu, P. Energy efficient cooperative computing in mobile wireless sensor networks. *IEEE Transactions on Cloud Computing* (2015).

[201] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* (2016).

[202] Shivakumar, P., Kistler, M., Keckler, S. W., Burger, D., and Alvisi, L. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on* (2002), IEEE, pp. 389–398.

[203] Shoaib, M., Jha, N. K., and Verma, N. Algorithm-driven architectural design space exploration of domain-specific medical-sensor processors. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 21*, 10 (2013), 1849–1862.

[204] Siekkinen, M., Hiienkari, M., Nurminen, J. K., and Nieminen, J. How low energy is bluetooth low energy? comparative measurements with ZigBee/802.15.4. In *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)* (2012), pp. 232–237.

[205] Silvano, C., Fornaciari, W., Reghizzi, S. C., Agosta, G., Palermo, G., Zaccaria, V., Bellasi, P., Castro, F., Corbetta, S., Speziale, E., et al. Parallel paradigms and run-time management techniques for many-core architectures: The 2parma approach. In *2011 9th IEEE International Conference on Industrial Informatics* (2011), IEEE, pp. 835–840.

[206] Singh, A. K., Dziurzanski, P., Mendis, H. R., and Soares Indrusiak, L. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi/many-core systems. *ACM Comput. Surv.* (2017).

[207] Singh, A. K., Shafique, M., Kumar, A., and Henkel, J. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference* (2013), ACM, p. 1.

[208] Singh, I., Shriraman, A., Fung, W. W., O'Connor, M., and Aamodt, T. M. Cache coherence for gpu architectures. In *High Performance Computer Architecture, 2013 IEEE 19th International Symposium on* (2013).

[209] Smith, P. Comparing low-power wireless technologies. Tech Zone, Digikey Online Magazine, Digi-Key Corporation, 2011.

[210] Smolinski, B. A. Approximating the 0-1 Multiple Knapsack Problem with Agent Decomposition and Market Negotiation. In *Intelligent Problem Solving. Methodologies and Approaches.* Springer, 2000, pp. 296–306.

[211] Sodani, A., Gramunt, R., Corbal, J., Kim, H.-S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., and Liu, Y.-C. Knights landing: Second-generation intel xeon phi product. *Ieee micro 36*, 2 (2016), 34–46.

[212] Spanò, E., Di Pascoli, S., and Iannaccone, G. Low-power wearable ecg monitoring system for multiple-patient remote monitoring. *IEEE Sensors Journal 16*, 13 (2016), 5452–5462.

[213] Stone, J. E., Gohara, D., and Shi, G. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering 12*, 3 (2010), 66–73.

[214] Su, J., Yang, F., Zeng, X., and Zhou, D. Efficient memory partitioning for parallel data access via data reuse. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (2016), ACM, pp. 138–147.

[215] Tang, Y. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239* (2013).

[216] Texier, I., Xydis, S., Soudris, D., Marcoux, P., Pham, P., Muller, M., Correvon, M., Dudnik, G., Voirin, G., Kristenssen, J., et al. Swan-icare project: Towards smart wearable and autonomous negative pressure device for wound monitoring and therapy. In *Wireless Mobile Communication and Healthcare (Mobihealth), 2014 EAI 4th International Conference on* (2014), IEEE, pp. 357–360.

[217] Tong, S., and Chang, E. Support vector machine active learning for image retrieval. In *Proceedings of the ninth ACM international conference on Multimedia* (2001), ACM, pp. 107–118.

[218] Tong, S., and Koller, D. Support vector machine active learning with applications to text classification. *Journal of machine learning research 2*, Nov (2001), 45–66.

[219] Townsend, K., Cufí, C., Davidson, R., et al. *Getting started with Bluetooth low energy: Tools and techniques for low-power networking.* " O'Reilly Media, Inc.", 2014.

[220] Tsoutsouras, V., Azariadi, D., Koliogewrgi, K., Xydis, S., and Soudris, D. Software design and optimization of ecg signal analysis and

diagnosis for embedded iot devices. In *Components and Services for IoT Platforms*. Springer, 2017, pp. 299–322.

[221] Tsoutsouras, V., Azariadi, D., Xydis, S., and Soudris, D. Effective learning and filtering of faulty heart-beats for advanced ecg arrhythmia detection using mit-bih database. In *Proceedings of the 5th EAI International Conference on Wireless Mobile Communication and Healthcare* (2015), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 50–53.

[222] Tsoutsouras, V., Koliogeorgi, K., Xydis, S., and Soudris, D. An exploration framework for efficient high-level synthesis of support vector machines: Case study on ecg arrhythmia detection for xilinx zynq soc. *Journal of Signal Processing Systems 88*, 2 (2017), 127–147.

[223] Tsoutsouras, V., Xydis, S., and Soudris, D. A hw/sw framework emulating wearable devices for remote wound monitoring and management. In *Wireless Mobile Communication and Healthcare (Mobihealth), 2014 EAI 4th International Conference on* (2014), IEEE, pp. 369–372.

[224] Tsoutsouras, V., Xydis, S., Soudris, D., and Lymperopoulos, L. Swan-icare project: On the efficiency of fpgas emulating wearable medical devices for wound management and monitoring. In *International Symposium on Applied Reconfigurable Computing* (2015), Springer, pp. 499–510.

[225] Übeyli, E. D. Ecg beats classification using multiclass support vector machines with error correcting output codes. *Digital Signal Processing 17*, 3 (2007), 675–684.

[226] Vadhiyar, S. S., and Dongarra, J. Srs: A framework for developing malleable and migratable parallel applications for distributed systems. *Parallel Processing Letters 13* (2003), 291–312.

[227] Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Iyer, P., Singh, A., Jacob, T., et al. An 80-tile 1.28 tflops network-on-chip in 65nm cmos. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International* (2007), IEEE, pp. 98–589.

[228] Volpi, M., Tuia, D., Bovolo, F., Kanevski, M., and Bruzzone, L. Supervised change detection in vhr images using contextual information and support vector machines. *International Journal of Applied Earth Observation and Geoinformation 20* (2013), 77–85.

[229] Wachter, E., Fochi, V., Barreto, F., Amory, A., and Moraes, F. A hierarchical and distributed fault tolerant proposal for noc-based mpsocs. *IEEE Transactions on Emerging Topics in Computing* (2016).

[230] Wan, C. H., Lee, L. H., Rajkumar, R., and Isa, D. A hybrid text classification approach with low dependency on parameter by integrating k-nearest neighbor and support vector machine. *Expert Systems with Applications 39*, 15 (2012), 11880–11888.

[231] Wang, W., Zhu, K., Ying, L., Tan, J., and Zhang, L. Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality. *IEEE/ACM Transactions on Networking 24*, 1 (2016), 190–203.

[232] Wang, Y., Chen, R., and Wang, D.-C. A survey of mobile cloud computing applications: perspectives and challenges. *Wireless Personal Communications 80*, 4 (2015), 1607–1623.

[233] Wang, Y., Zhang, P., Cheng, X., and Cong, J. An integrated and automated memory optimization flow for fpga behavioral synthesis. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific* (2012), IEEE, pp. 257–262.

[234] Weber, S. A., Watermann, N., Jossinet, J., Byrne, J. A., Chantrey, J., Alam, S., So, K., Bush, J., O'Kane, S., and McAdams, E. T. Remote wound monitoring of chronic ulcers. *IEEE Transactions on Information Technology in Biomedicine 14*, 2 (2010), 371–377.

[235] Wentzlaff, D., and Agarwal, A. Factored operating systems (fos): the case for a scalable operating system for multicores. *ACM SIGOPS Operating Systems Review 43*, 2 (2009), 76–85.

[236] Werner, S., Navaridas, J., and Luján, M. A survey on design approaches to circumvent permanent faults in networks-on-chip. *ACM Computing Surveys* (2016).

[237] Wolf, W. Cyber-physical systems. *Computer 42*, 3 (2009), 88–89.

[238] Wu, C., Deng, C., Liu, L., Han, J., Chen, J., Yin, S., and Wei, S. An efficient application mapping approach for the co-optimization of reliability, energy, and performance in reconfigurable noc architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 34*, 8 (2015), 1264–1277.

[239] Xia, F. Qos challenges and opportunities in wireless sensor/actuator networks. *Sensors 8*, 2 (2008), 1099–1110.

[240] Xia, F., Zhao, W., Sun, Y., and Tian, Y.-C. Fuzzy logic control based qos management in wireless sensor/actuator networks. *Sensors 7*, 12 (2007), 3179–3191.

[241] Xian, C., Lu, Y.-H., and Li, Z. Adaptive computation offloading for energy conservation on battery-powered systems. In *International Conference on Parallel and Distributed Systems* (2007), vol. 2, pp. 1–8.

[242] Xilinx. Vivado design suite user guide: High-level synthesis, v2014.1.

[243] Xu, Z. J. Ls2085/8a freescale's new qorlq layerscape communications processor. In *Hot Chips 27 Symposium (HCS), 2015 IEEE* (2015), IEEE, pp. 1–25.

[244] Xydis, S., Economakos, G., Soudris, D., and Pekmestzi, K. Z. High performance and area efficient flexible DSP datapath synthesis. *IEEE Trans. VLSI Syst. 19*, 3 (2011), 429–442.

[245] Xydis, S., Palermo, G., Zaccaria, V., and Silvano, C. SPIRIT: spectral-aware pareto iterative refinement optimization for supervised high-level synthesis. *IEEE Trans. on CAD of Integrated Circuits and Systems 34*, 1 (2015), 155–159.

[246] Xydis, S., Pekmestzi, K. Z., Soudris, D., and Economakos, G. Compiler-in-the-loop exploration during datapath synthesis for higher quality delay-area trade-offs. *ACM Trans. Design Autom. Electr. Syst. 18*, 1 (2012), 11.

[247] Xydis, S., Triantafyllou, I. S., Economakos, G., and Pekmestzi, K. Z. Flexible datapath synthesis through arithmetically optimized operation chaining. In *NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2009, San Francisco, California, USA, July 29 - August 1, 2009* (2009), pp. 407–414.

[248] Yang, B., Guang, L., Säntti, T., and Plosila, J. Mapping multiple applications with unbounded and bounded number of cores on many-core networks-on-chip. *Microprocessors and Microsystems 37*, 4 (2013), 460–471.

[249] Yang, L., Liu, W., Jiang, W., Li, M., Yi, J., and Sha, E. H.-M. Application mapping and scheduling for network-on-chip-based multiprocessor system-on-chip with fine-grain communication optimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24*, 10 (2016), 3027–3040.

[250] You, C., Huang, K., Chae, H., and Kim, B.-H. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications 16*, 3 (2017), 1397–1411.

[251] Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., and Dutta, P. The internet of things has a gateway problem. In *Mobile Computing Systems and Applications (HotMobile)* (2015), pp. 27–32.

[252] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. Apache spark: a unified engine for big data processing. *Communications of the ACM 59*, 11 (2016), 56–65.

[253] Zhang, B., Mor, N., Kolb, J., Chan, D. S., Goyal, N., Lutz, K., Allman, E., Wawrzynek, J., Lee, E., and Kubiatowicz, J. The cloud is not enough: saving IoT from the cloud. In *USENIX Conf. on Hot Topics in Cloud Computing* (2015), pp. 21–21.

[254] Zhang, H., and Zhang, L.-Q. Ecg analysis based on pca and support vector machines. In *Neural Networks and Brain, 2005. ICNN&B'05. International Conference on* (2005), vol. 2, IEEE, pp. 743–747.

[255] Zhang, Y., Wang, S., Ji, G., and Dong, Z. An mr brain images classifier system via particle swarm optimization and kernel support vector machine. *The Scientific World Journal 2013* (2013).

[256] Zheng, W., and Sakellariou, R. Budget-deadline constrained workflow planning for admission control. *Journal of grid computing 11*, 4 (2013), 633–651.

[257] Zhong, G., Venkataramani, V., Liang, Y., Mitra, T., and Niar, S. Design space exploration of multiple loops on fpgas using high level synthesis. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)* (2014), IEEE, pp. 456–463.

[258] Zhu, C., Li, X., Song, L., and Xiang, L. Development of a theoretically based thermal model for lithium ion battery pack. *Journal of Power Sources 223* (2013), 155–164.

[259] Zong, W., and Moody, G. Wqrs-single-channel qrs detector based on length transform. *Physionet http://wwwphysionetorg/physiotools/wag/wqrs-1htm* (2003).

[260] Zynq, X. 7000 all programmable soc zc702 evaluation kit, 2015.

List of publications

Journals

J6. Samie, F., **Tsoutsouras, V.**, Bauer, L., Xydis, S., Soudris, D. and Henkel, J., Oops: Optimizing Operation-mode Selection for IoT Edge Devices. Accepted for publication in ACM Transactions on Internet Technology: Special issue on Fog, Edge, and Cloud Integration for Smart Environments.

J5. **Tsoutsouras, V.**, Anagnostopoulos, I., Masouros, D. and Soudris, D., 2018. A Hierarchical Distributed Runtime Resource Management Scheme for NoC-Based Many-Cores. ACM Transactions on Embedded Computing Systems (TECS), 17(3), p.65.

J4. **Tsoutsouras, V.**, Xydis, S. and Soudris, D.J., 2018. Application-Arrival Rate Aware Distributed Run-Time Resource Management for Many-core Computing Platforms. IEEE Transactions on Multi-Scale Computing Systems (TMSCS).

J3. **Tsoutsouras, V.**, Masouros, D., Xydis, S. and Soudris, D., 2017. SoftRM: Self-Organized Fault-Tolerant Resource Management for Failure Detection and Recovery in NoC Based Many-Cores. ACM Transactions on Embedded Computing Systems (TECS), 16(5s), p.144.

J2. Samie, F., **Tsoutsouras, V.**, Bauer, L., Xydis, S., Soudris, D. and Henkel, J., 2018. Distributed Trade-Based Edge Device Management in Multi-Gateway IoT. ACM Transactions on Cyber-Physical Systems, 2(3), p.17.

J1. **Tsoutsouras, V.**, Koliogeorgi, K., Xydis, S. and Soudris, D., 2017. An Exploration Framework for Efficient High-Level Synthesis of Support Vector Machines: Case Study on ECG Arrhythmia Detection for Xilinx Zynq SoC. Journal of Signal Processing Systems, pp.1-21.

Book Chapters

B2. **Tsoutsouras, V.**, Azariadi, D., Koliogewrgi, K., Xydis, S. and Soudris, D., 2017. Software Design and Optimization of ECG Signal Analysis and Diagnosis for Embedded IoT Devices. In Components and Services for IoT Platforms (pp. 299-322). Springer International Publishing.

B1. **Tsoutsouras, V.**, Xydis, S. and Soudris, D., 2016. 7 Cooperative Data Fusion. Wireless Medical Systems and Algorithms: Design and Applications, 56, p.163.

Conferences

C11. Masouros, D., Bakolas, I., **Tsoutsouras, V.**, Siozios, K., and Soudris, D. "From Edge To Cloud: Design and Implementation of a Healthcare Internet of Things Infrastructure", in 27th International Symposium on Power and Timing Modeling, Optimization and Simulation, September 2017.

C10. Zhao, Z., **Tsoutsouras, V.**, Soudris, D. and Gerstlauer, A., Network/System Co-Simulation for Design Space Exploration of IoT Applications, International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XVII), July 2017.

C9. Samie, F., **Tsoutsouras, V.**, Bauer, L., Xydis, S., Soudris, D. and Henkel, J., 2016, December. Computation offloading and resource allocation for low-power IoT edge devices. In Internet of Things (WF-IoT), 2016 IEEE 3rd World Forum on (pp. 7-12). IEEE.

C8. Samie, F., **Tsoutsouras, V.**, Xydis, S., Bauer, L., Soudris, D. and Henkel, J., 2016, October. Distributed QoS management for Internet of Things under resource constraints. In Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2016 International Conference on (pp. 1-10). IEEE.

C7. Railis, K., **Tsoutsouras, V.**, Xydis, S. and Soudris, D., 2016, September. Energy profile analysis of Zynq-7000 programmable SoC for embedded medical processing: Study on ECG arrhythmia detection. In Power and Timing Modeling, Optimization and Simulation (PATMOS), 2016 26th International Workshop on (pp. 275-282). IEEE.

C6. Azariadi, D., **Tsoutsouras, V.**, Xydis, S. and Soudris, D., 2016, May. ECG signal analysis and arrhythmia detection on IoT wearable medical devices. In Modern Circuits and Systems Technologies (MOCAST), 2016 5th International Conference on (pp. 1-4). IEEE.

C5. **Tsoutsouras, V.**, Azariadi, D., Xydis, S. and Soudris, D., 2015, December. Effective learning and filtering of faulty heart-beats for advanced ecg arrhythmia detection using mit-bih database. In Proceedings of the 5th EAI International Conference on Wireless Mobile Communication and Healthcare (pp. 50-53). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

C4. **Tsoutsouras, V.**, Xydis, S. and Soudris, D., 2015, October. Job-arrival aware distributed run-time resource management on intel scc manycore platform. In Embedded and Ubiquitous Computing (EUC), 2015 IEEE 13th International Conference on (pp. 17-24). IEEE.

C3. **Tsoutsouras, V.**, Xydis, S., Soudris, D. and Lymperopoulos, L., 2015, April. SWAN-iCARE project: On the efficiency of FPGAs emulating wearable medical devices for wound management and monitoring. In International Symposium on Applied Reconfigurable Computing (pp. 499-510). Springer, Cham.

C2. **Tsoutsouras, V.**, Xydis, S. and Soudris, D., 2014, November. A HW/SW framework emulating wearable devices for remote wound monitoring and management. In Wireless Mobile Communication and Healthcare (Mobihealth), 2014 EAI 4th International Conference on (pp. 369-372). IEEE.

C1. Anagnostopoulos, I., **Tsoutsouras, V.**, Bartzas, A. and Soudris, D., 2013, May. Distributed run-time resource management for malleable applications on many-core platforms. In Proceedings of the 50th Annual Design Automation Conference (p. 168). ACM.

Glossary

Term	Explanation
Application mapping	Η χαρτογράφηση μιας παράλληλης εφαρμογής σε ένα πολυπύρηνο σύστημα, αναφέρεται στην διαδικασία με την όποια καθορίζονται οι υπολογιστικοί πόροι στους οποίους θα εκτελεστεί η εφαρμογή. Σε αντίθεση με την χρονοδρομολόγηση που αφορά την χρονική κατανομή πόρων που βρίσκονται κοντά μεταξύ τους, η χαρτογράφηση μιας εφαρμογής έχει έντονη χωρική συνιστώσα στην κατανομή των πόρων. Δεδομένης της χαρτογράφησης, η χρονοδρομολόγηση χρησιμοποιείται ώστε οι πόροι της εφαρμογής που μοιράζονται να διατίθενται στις διεργασίες της.
Application profiling	Η διαδικασία αυτή αναφέρεται στα απαραίτητα βήματα για την ποσοτικοποίηση των υπολογιστικών απαιτήσεων μιας εφαρμογής σε σχέση με μια συγκεκριμένη υπολογιστική πλατφόρμα. Η συνήθης διαδικασία περιλαμβάνει την εκτέλεση τα εφαρμογής για ένα δείγμα αντιπροσωπευτικών εισόδων ώστε να προσδιοριστούν οι απαιτήσεις των επιμέρους δομικών της τμημάτων.
Central Processing Unit (CPU)	Η κεντρική μονάδα επεξεργασίας ενός υπολογιστικού συστήματος αναφέρεται σε ένα ολοκληρωμένο κύκλωμα το οποίο είναι σε θέση να πραγματοποιεί υπολογισμούς γενικού σκοπού, έλεγχο της εισόδου/εξόδου καθώς και έλεγχο των περιφερειακών του.

Cloud computing	Η έννοια του υπολογιστικού νέφους αναφέρεται σε ένα σύνολο υπολογιστικών πόρων, ένα ποσοστό των οποίων παρέχονται από έναν πάροχο σε ένα πελάτη με χρήση του διαδικτύου. Οι παροχές μπορούν να αφορούν αποκλειστικά του υπολογιστικούς πόρους αλλά και αποθηκευτικό χώρο και πιο σύνθετες εφαρμογές ως υπηρεσίες. Κατά κανόνα η παροχή των ανωτέρω χρεώνεται στον πελάτη.
Edge computing	Ο υπολογισμός στην άκρη του δικτύου αφορά σε μια πρόσφατη τάση στον τομέα του διαδικτύου των πραγμάτων, όπου αποφεύγεται η εκτέλεση των εφαρμογών στο υπολογιστικό νέφος και προτιμάται η εκτέλεση τους στην άκρη του δικτύου, όντας πιο κοντά στον τελικό χρήστη. Με αυτή την επιλογή, ελαχιστοποιείται η εξάρτηση από το υπολογιστικό νέφος, προβλήματα συνδεσιμότητας με αυτό και μειώνεται το κόστος της χρήσης των πόρων του.
Field Programmable Gate Array (FPGA)	Η τεχνολογία αυτή αφορά ολοκληρωμένα κυκλώματα τα οποία έχουν την δυνατότητα να προγραμματίζονται ώστε να επιτελούν διαφορετικούς υπολογισμούς κατά περίπτωση. Η επιλογή αυτή αντιτίθεται στα κλασσικά ολοκληρωμένα κυκλώματα που κατασκευάζονται ώστε να υλοποιούν ένα σύνολο συγκεκριμένων υπολογισμών.
Gateway	Στη τεχνολογία των υπολογιστικών συστημάτων η εν λόγω συσκευή χρησιμοποιείται κατά κανόνα στις τηλεπικοινωνίες, βοηθώντας τις συσκευές δύο διαφορετικών δικτύων να επικοινωνούν. Στα πλαίσια της συγκεκριμένης διατριβής, η συσκευή αυτή διασυνδέσει ενσωματωμένες συσκευές με το διαδίκτυο. Επιπρόσθετα, μπορούν να παρέχουν και υπολογιστικούς πόρους ώστε να διευκολύνουν την εκτέλεση των εφαρμογών στην άκρη του δικτύου.
High Level Synthesis (HLS)	Ο όρος αναφέρεται σε μια αυτοματοποιήμενη διαδικασία σχεδιασμού όπου ένα ολοκληρωμένο κύκλωμα παράγεται από μια αλγοριθμική περιγραφή υψηλού επιπέδου της λειτουργικότητας του.

Internet of Things (IoT)	Με τον όρο διαδίκτυο των πραγμάτων αναφερόμαστε στον στόχο της διασύνδεσης της πλειονότητας των συσκευών που χρησιμοποιούνται σε όλες τις εκφάνσεις της καθημερινότητας, ώστε να μπορεί να γίνεται ζωντανή παρακολούθηση των δραστηριοτήτων τους και συλλογή των δεδομένων αυτών. Τεχνολογικά αυτό επιτυγχάνεται με την κατανομή και την διασύνδεση των ενσωματωμένων συσκευών στο διαδίκτυο. Τα δεδομένα που συλλέγονται μπορούν έπειτα να δώσουν πρωτοφανείς δυνατότητες ανάλυσης και βελτιστοποίησης σύνθετων φαινομένων και διεργασιών.
Linux Operating System	Αφορά μια οικογένεια λειτουργικών συστημάτων γενικού σκοπού που χαρακτηρίζονται από δωρεάν χρήση και ελεύθερη πρόσβαση στον πηγαίο κώδικα τους. Κατά κανόνα, το εν λόγω λειτουργικό σύστημα φτάνει στους χρήστες εν είδει μια διανομής που περιέχει ένα σύνολο προγραμμάτων και λειτουργιών χτισμένων γύρω από τον πυρήνα. Εκατομμύρια χρήστες κάνουν χρήση του Linux, το οποίο ως εκ τούτου έχει προσαρμοστεί για πολύ μεγάλο αριθμό συσκευών διαφορετικών υπολογιστικών δυνατοτήτων και αρχιτεκτονικών.
L1/L2 cache memory	Η κρυφή μνήμη αναφέρεται σε επίπεδα μικρής και γρήγορης μνήμης που βρίσκονται κοντά στον επεξεργαστή, με στόχο να γίνεται επαναχρησιμοποίηση των δεδομένων που έχουν έρθει από την κεντρική μνήμη αποφεύγοντας την χρονοβόρα διαδικασία του να έρθουν ξανά. Αυξανόμενα επίπεδα κρυφής μνήμης έχουν μεγαλύτερη καθυστέρηση παροχής δεδομένων στον επεξεργαστή αλλά ταυτόχρονα έχουν και μεγαλύτερη διαθέσιμη χωρητικότητα.
Machine Learning	Η μηχανική μάθηση αναφέρεται στην μελέτη αλγορίθμων που έχουν στόχο να δώσουν στα υπολογιστικά συστήματα την δυνατότητα να κάνουν χρήση μαθηματικών μοντέλων για την ερμηνεία και κατηγοριοποίηση δεδομένων εισόδου. Μια χαρακτηριστική εφαρμογή της μηχανικής μάθησης είναι η προσπάθεια κατασκευής ενός μοντέλου για διαχωρισμό της εισόδου σε κλάσεις με βάση ένα σετ δεδομένων εκμάθησης που αποτελείται από ήδη κατηγοριοποιημένα δείγματα (επιβλεπόμενη μηχανική μάθηση). Το αποτέλεσμα της εκμάθησης είναι οι παράμετροι του μοντέλου, το οποίο έπειτα είναι σε θέση να αποφανθεί για την κλάση μια καινούριας, άγνωστης εισόδου.

Network on Chip	Η σύνδεση δικτύου σε ψηφίδα αποτελεί μια δημοφιλή επιλογή στα πλαίσια του σχεδιασμού συσκευών με πολύ μεγάλο αριθμό επεξεργαστών. Σε αυτές τις συσκευές, η επιλογή διαμοιραζόμενής μνήμης για την επικοινωνία των επεξεργαστών είναι μη αποδοτική και ως εκ τούτου επιλέγεται η διασύνδεση τους πάνω σε ένα δίκτυο μικρής καθυστέρησης, ώστε να μπορούν να ανταλλάσσουν δεδομένα μέσω μηνυμάτων.
Voltage and Frequency Scaling (VFS)	Η τεχνική αυτή αφορά την δυνατότητα που δίνουν τα σύγχρονα ολοκληρωμένα κυκλώματα να μεταβάλλουν δυναμικά την συχνότητα ή/και την τάση λειτουργίας τους. Η δυνατότητα αυτή είναι πολύ σημαντική ώστε να μπορεί να προσαρμοστεί η κατανάλωση ισχύος του συστήματος σε σχέση με τον φόρτο εργασίας του. Σε περίπτωση μικρού φόρτου εργασίας, το σύστημα είναι σε θέση να λειτουργήσει σε χαμηλότερή τάση και συχνότητα ώστε να μειωθεί η κατανάλωση ισχύος του. Το χαρακτηριστικό αυτό είναι κρίσιμο σε μεγάλο εύρος υπολογιστικών συστημάτων, ξεκινώντας από φορητές συσκευές για την επιμήκυνση του χρόνου λειτουργίας με μπαταρίας και καταλήγοντας σε μεγάλα υπολογιστικά κέντρα για την μείωση της καταναλισκόμενης ενέργειας τους, για οικονομικούς και οικολογικούς λόγους.
Voltage Island	Ο όρος αναφέρεται σε τμήματα ενός ολοκληρωμένου κυκλώματος, τα οποία τροφοδοτούνται με την ίδια τάση. Τα τμήματα αυτά λογίζονται ως νησιά δυναμικού και το κάθε ένα μπορεί να έχει ξεχωριστή τάση, ανεξάρτητα από τα υπόλοιπα.
Pattern Recognition	Η διαδικασία αυτή αφορά την κατασκευή αλγορίθμων για την αυτόματη αναγνώριση μοτίβων και κανονικοτήτων σε ένα σύνολο δεδομένων. Οι αλγόριθμοι αυτοί μπορεί να είναι ευριστικοί, να βασίζονται σε στατιστική ανάλυση ή να κάνουν χρήση της μηχανικής μάθησης και της τεχνητής νοημοσύνης, που αποτελεί μια εξαιρετικά δημοφιλή επιλογή.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
MICROPROCESSORS AND DIGITAL SYSTEMS LAB (MICROLAB)
9 Heroon Polytechneiou, Zographou Campus
157 80 Athens, Greece
billtsou@microlab.ntua.gr