Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών

# Μεθοδολογίες και εργαλεία διαχείρισης πόρων και παραμετροποίησης εφαρμογών κατά το χρόνο εκτέλεσης σε πολυπύρηνες ενσωματωμένες πλατφόρμες

**Διδακτορική Διατριβή**

Ηρακλής Αναγνωστόπουλος
Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών

Αθήνα, Ιανουάριος 2014

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Μικροϋπολογιστών & Ψηφιακών Συστημάτων

# Μεθοδολογίες και εργαλεία διαχείρισης πόρων και παραμετροποίησης εφαρμογών κατά το χρόνο εκτέλεσης σε πολυπύρηνες ενσωματωμένες πλατφόρμες

Διδακτορική Διατριβή
του
Ηρακλή Αναγνωστόπουλου
Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών

Υποβληθείσα στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής & Υπολογιστών
Εργαστήριο Μικροϋπολογιστών & Ψηφιακών Συστημάτων

# Μεθοδολογίες και εργαλεία διαχείρισης πόρων και παραμετροποίησης εφαρμογών κατά το χρόνο εκτέλεσης σε πολυπύρηνες ενσωματωμένες πλατφόρμες

Τριμελής Συμβουλευτική Επιτροπή

| Δημήτριος Σούντρης | Κιαμάλ Πεκμεστζή | Γεώργιος Οικονομάκος |
|---|---|---|
| Επ. Καθηγητής Ε.Μ.Π. | Καθηγητής Ε.Μ.Π. | Επ. Καθηγητής Ε.Μ.Π. |

Επταμελής Εξεταστική Επιτροπή

| Δημήτριος Σούντρης | Κιαμάλ Πεκμεστζή | Γεώργιος Οικονομάκος |
|---|---|---|
| Επ. Καθηγητής Ε.Μ.Π. | Καθηγητής Ε.Μ.Π. | Επ. Καθηγητής Ε.Μ.Π. |

| Νεκτάριος Κοζύρης | Ιωάννης Παπαευσταθίου |
|---|---|
| Καθηγητής Ε.Μ.Π. | Αν. Καθηγητής Π. Κρήτης |

| Δημήτριος Γκιζόπουλος | Axel Jantsch |
|---|---|
| Αν. Καθηγητής Ε.Κ.Π.Α. | Καθηγητής ΚΤΗ |

.....
Ηρακλής Ν. Αναγνωστόπουλος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός & Μηχανικός Υπολογιστών Ε.Μ.Π.



Η παρούσα Διδακτορική Διατριβή συγχρηματοδοτήθηκε απο τα ευρωπαϊκά ερευνητικά προγράμματα FP7-248716 2PARMA, FP7-215244 MOSART και ENIAC-2010-1 TOISE. Επίσης, μέρος της παρούσας Διδακτορικής Διατριβής συγχρηματοδοτήθηκε απο εθνικά κεφάλαια και απο το Εθνικό Στρατηγικό Πλαισίο Αναφοράς (ΕΣΠΑ) 2007-2013 για το έργο "Next Generation Millimeter Wave Backhaul Radio".

# Περίληψη

Στην παρούσα διδακτορική διατριβή, παρουσιάζουμε (i) μεθολογίες επιτάχυνσης και παραμετροποίησης της διαχείρισης μνήμης σε επίπεδο middleware για την εφαρμογή προσαρμοσμένων Δυναμικών Διαχειριστών Μνήμης (ΔΔΜ) και (ii) μεθοδολογίες κατανεμημένης διαχείρισης πόρων σε πολυπύρηνες ενσωματωμένες πλατφόρμες. Αρχικά, η παραμετροποίηση επιτυγχάνεται με την εφαρμογή προσαρμοσμένων ΔΔΜ σε μικροκώδικα (microcode). Επιπλέον, η διαχείριση των πόρων της πλατφόρμας κατά τη φάση εκτέλεσης επιτυγχάνεται με τα προτεινόμενα μεθοδολογικά πλαίσια με την χρήση πολλών πυρήνων σε διαφορετικούς ρόλους και την ανάπτυξη επικοινωνιακών πλαισίων για τη μείωση του συνολικού φορτίου στο ολοκληρωμένο. Οι προτεινόμενες μεθοδολογίες έδειξαν ότι η προσέγγιση σε μικροκώδικα αποτελεί μια καλή εναλλακτική λύση για να ξεπεραστεί το δίλημμα απόδοσης και ευελιξίας, προσφέροντας μια προγραμματιζόμενη και ευέλικτη λύση για την επιτάχυνση μεγάλου φάσματος εφαρμογών. Επιπλέον, προσφέρουν ευελιξία στο θέμα της κατανεμημένης απεικόνισης των εφαρμογών στη φάση εκτέλεσης καθώς βασίζονται στο γεγονός ότι μπορούν να πετύχουν διαφορετικά επίπεδα αξιοποίησης των πόρων της πλατφόρμας ανάλογα με τις ανάγκες των εφαρμογών και χωρίς να υπάρχει κάποιο κεντρικό σημείο αποτυχίας. Όσον αφορά τις υπηρεσίες διαχείρισης μνήμης σε επίπεδο μικροκώδικα, τα πειραματικά αποτελέσματα δείχνουν ότι το κέρδος της προτεινόμενης προσέγγισης, για το σχεδιασμό εξατομικευμένων ΔΔΜ, ήταν περίπου 7× μεγαλύτερο με μια μικρή αύξηση, της τάξεως του 14%, στην καταναλισκόμενη ισχύ. Το πλαίσιο για την απεικόνιση εφαρμογών στη φάση εκτέλεσης προσαρμόζεται στις ανάγκες και στους περιορισμούς των εφαρμογών και παράγει κατά μέσο όρο 21% και 10% καλύτερο κόστος επικοινωνίας για ομογενείς και ετερογενείς πλατφόρμες αντίστοιχα. Τέλος, όσον αφορά τις πράλληλες εφαρμογές το προτεινόμενο πλαίσιο έχει κατά μέσο όρο 70% λιγότερα μηνύματα, 64% μικρότερο μέγεθος μηνυμάτων και 20% κέρδος στην επιτάχυνση των εφαρμογών.

**Λέξεις Κλειδιά**

Δυναμική διαχειρίση μνήμης, πολυπύρηνα ενσωματωμένα συστήματα, δυναμική διαχείριση τάσης και συχνότητας, απεικόνιση στο χρόνο εκτέλεσης, Δίκτυο-σε-Ψηφίδα, διαχείριση πόρων.

# Abstract

In this Ph.D. Thesis, we present (i) memory management middleware acceleration and customization methodologies for applying customized dynamic memory managers (allocators) and (ii) frameworks for distributed run-time resource management on many-core platforms. Firtsly, the customization is achieved by applying, on the middleware level, custom microcoded memory allocators. Secondly, the run-time resource management on the platform is achieved by using cores in different roles and by applying a distributed on-chip communication scheme. The proposed methodologies showed that the microcode approach is a good alternative to overcome the performance-flexibility dilemma, offering a programmable and flexible solution for accelerating a wide range of applications. Thus, we adopt the microcoded approach to address memory management issues on Distributed Shared Memory (DSM) many-core embedded platforms, aiming for hardware performance but maintaining the flexibility of programs. Also, the developed framework provides a flexible solution in the run-time mapping problem offering different levels of platform utilization according to application's needs and without a central point of failure. Concerning microcoded memory management services, experimental results show that the gain, of the proposed approaches for designing customized microcoded memory managers, was approximately 7× for served allocation requests with a small increase of approximately 14% to average energy consumption per allocation. The run-time resource management framework adapts to application's needs and application's execution restrictions by using the matching factor parameter and produces on average 21% and 10% better on-chip communication cost for homogeneous and heterogeneous platforms respectively. Last, concerning the malleable parallel applications, the developed framework has on average 70% less messages, 64% smaller message size and 20% application speed-up gain.

**Keywords**

Dynamic memory management, many-core embedded systems, dynamic voltage and frequency scaling, run-time mapping, Network-onChip, resource management

# Περιεχόμενα

# Κατάλογος σχημάτων

# Κατάλογος πινάκων

# Κεφάλαιο 1

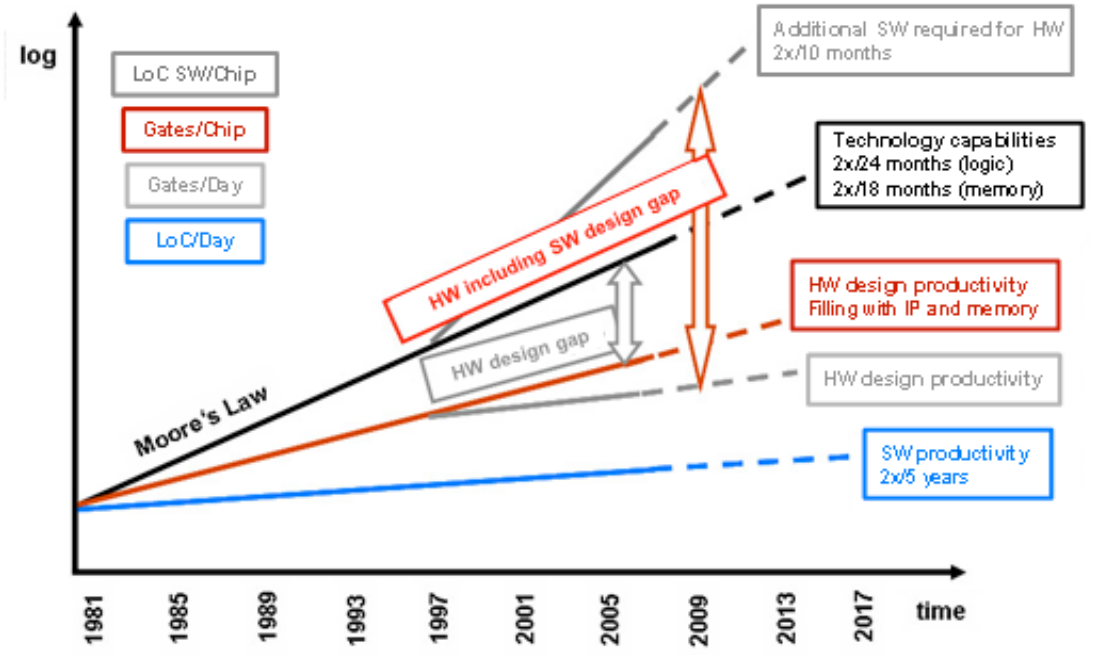
# Εισαγωγή

## 1.1 Επιτάχυνση και παραμετροποίηση διαχείρισης μνήμης σε ενσωματωμένα συστήματα

Τα μελλοντικά ολοκληρωμένα συστήματα θα περιέχουν δισεκατομμυρία τρανζίστορ [76], συνθέτοντας δεκάδες έως εκατοντάδες συστήματα πυρήνων. Οι σύγχρονες ενσωματωμένες πλατφόρμες επωφελούνται από αυτή την κατασκευαστική πρόοδο της τεχνολογίας και κινούνται προς πολυπύρηνα συστήματα. Η Intel έχει ήδη δημιουργήσει πλατφόρμες με 80 και 48 πυρήνες γενικού σκοπού [49, 75, 88], ενώ Δίκτυα-σε Ψηφίδα (Networks-on-Chip, NoC) υποστηρίζονται ήδη από τη βιομηχανία (π.χ. Æthereal NoC [42] από NXP και το STNoC [83] από την STMicroelectronics). Επίσης, η ανάπτυξη αυτών των πολυπύρηνων αρχιτεκτονικών οδηγείται και από την ανάπτυξη ιδιαίτερα απαιτητικών παράλληλων/πολυνηματικών εφαρμογών.

Το χάσμα της παραγωγικότητας των σχεδιαστών υλικού και λογισμικού για ενσωματωμένα συστήματα παρουσιάζεται στο Σχήμα 1.1. Βλέπουμε ότι οι ανάγκες για λογισμικό διπλασιάζονται κάθε 10 μήνες ενώ οι επιδόσεις των πλατφορμών υλικού διπλασιάζονται κάθε 24 μήνες. Η παραγωγικότητα των σχεδιαστών υλικού βελτιώθηκε τα τελευταία χρόνια καθώς έχουν τη δυνατότητα της δημιουργίας πολυπλοκότερων συστημάτων με πολλούς πυρήνες. Οι επιπλέον λειτουργίες και υπηρεσίες παρέχονται μόνο από το λογισμικό. Η παραγωγικότητα των σχεδιαστών λογισμικού, ειδικά του λογισμικού που αλληλεπιδρά συχνά με το υλικό (π.χ., device drivers) έχει παραμείνει σχεδόν στάσιμη. Το κόκκινο βέλος στο Σχήμα 1.1 δείχνει το μέγεθος αυτού του χάσματος.

Η μνήμη αποτελεί σημαντικό παράγοντα για την απόδοση και την κατανάλωση ενέργειας των ενσωματωμένων συστημάτων. Καθώς ο αριθμός των on-chip πυρήνων αυξάνεται, το περιεχόμενο της ενσωματωμένης μνήμης αυξήθηκε επίσης από 20% πριν από δέκα χρόνια στο 85% σήμερα και θα συνεχίσει να αυξάνεται στο μέλλον. Οι μνήμες είναι κατά προτίμηση κατανεμημένες για μεσαίας και μεγάλης κλίμακας μεγεθών συστήματα, καθώς η κεντρικοποιημένη μνήμη έχει ήδη γίνει εμπόδιο στις επιδόσεις, την ισχύ και το κόστος. Οι παραδοσιακές τεχνικές βελτιστοποίησης μνήμης χρησιμοποιούν πληροφορίες κατα τη φάση της μεταγλώττισης και εστιάζουν στην στατική κατανομή σε σχέση με την ιεραρχία μνήμης [29]. Για τις σύγχρονες δυναμικές εφαρμογές που χρησιμοποιούν πολυπύρηνα αρχιτεκτονικά πρότυπα, αυτό δεν είναι πλέον

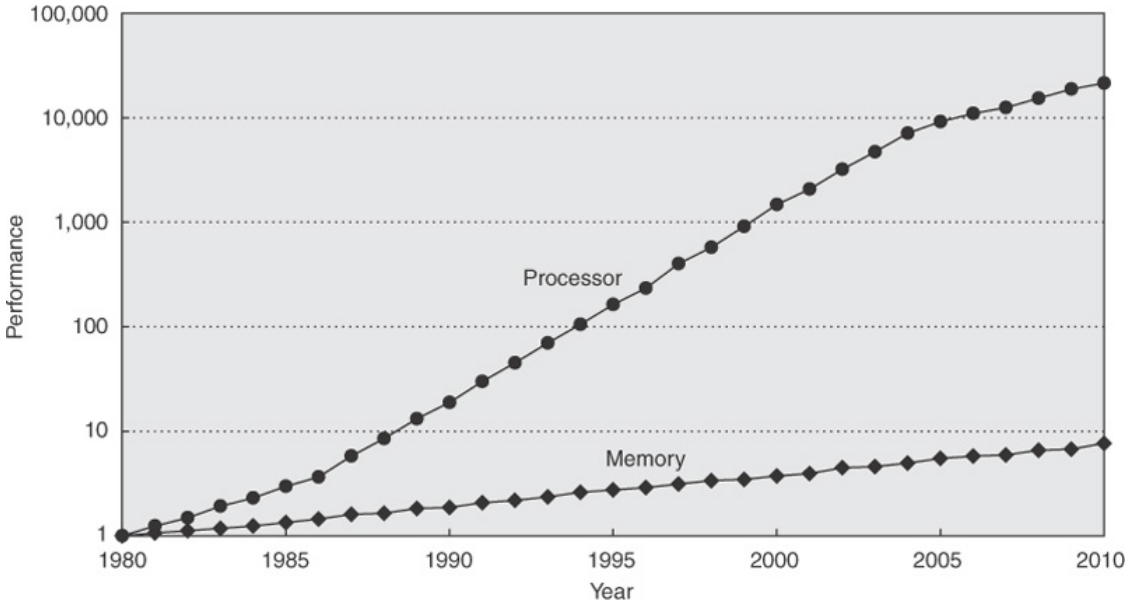



Σχήμα 1.1: Χάσμα παραγωγικότητας των σχεδιαστών υλικού και λογισμικού

δυνατό, διότι η δυναμικότητα των εφαρμογών είναι πολύ μεγάλη οδηγώντας το μέγεθος της μνήμης που χρησιμοποιείται σε απροσδόκητες διακυμάνσεις, άγνωστες κατά το χρόνο σχεδίασης. Η ανεπαρκής διαχείριση μνήμης οδηγεί σε συνολική υποβάθμιση των επιδόσεων, μεγάλο αποτύπωμα μνήμης και αυξημένη κατανάλωση ισχύος, όπως φαίνεται στην Εικόνα 1.2.

## 1.2 Σύγκλιση συστημάτων γενικού σκοπού και ενσωματωμένων συστημάτων

Τα υπολογιστικά συστήματα έχουν τεράστιο αντίκτυπο σε όλους τους τομείς της καθημερινής ζωής, από το διαδίκτυο και τα καταναλωτικά ηλεκτρονικά προϊόντα μέχρι τις μεταφορές, την ιατρική και την ενέργεια. Επίσης, η παραδοσιακή προοπτική μονοπύρηνων συστημάτων αλλάζει καθώς μαγαλύτερη απόδοση-ταχύτητα και λειτρουγικότητα προστίθεται από τη βιομηχανία. Η σύγκλιση, τόσο στο λογισμικό όσο και στο υλικό, είναι ραγδαία στον κόσμο των ενσωματωμένων συστημάτων και των συστημάτων γενικού σκοπού και επίσης οι εφαρμογές κινούνται απο τα τοπικά συστήματα σε συστήματα cloud. Για παράδειγμα, αρχικά τα κινητά τηλέφωνα εφευρέθηκαν μόνο για την ευκολία της επικοινωνίας και οι απαιτήσεις του σε υλικό ήταν ελάχιστες. Σήμερα, τα κινητά τηλέφωνα φιλοξενούν μέχρι και τέσσερις πυρήνες, με κάμερες υψηλής ευκρίνειας και επιταχυντές υλικού και είναι διαθέσιμα στους καταναλωτές σε προσιτές τιμές. Samsung Galaxy S4 , HTC One X , Huawei Ascend P2 κλπ. θεωρούνται από τα καλύτερα κινητά τηλέφωνα από άποψη απόδοσης, με




Σχήμα 1.2: Χάσμα επιδόσεων μεταξύ των επεξεργαστικών στοιχείων και της μνήμης [46]

υπολογιστική ικανότητα μεγαλύτερη απο παλαιότερα συστήματα υψηλής υπολογιστικής ισχύος. Η βιομηχανία των συστημάτων πληροφορικής βιώνει επίσης μια σειρά από διασπαστικές τάσεις. Ο Δρ Doug Burger, διευθυντής του τομέα Client and Cloud Applications της Microsoft Research, στην ομιλία του στο HiPEAC2013 δήλωσε ότι υπάρχει μεγαλύτερη αβεβαιότητα στον τομέα της πληροφορικής σήμερα από οποιαδήποτε άλλη φορά τα τελευταία 40 χρόνια.

Μπορούμε να δούμε εταιρείες που δραστηριοποιούνται ως προμηθευτές υλικού για συστήματα πληροφορικής (Apple, Samsung, κλπ.), να επεκτείνονται στην παραγωγή λογισμικού στοχεύοντας και διεκδικώντας ένα μεγαλύτερο μερίδιο της αγοράς (iOS, Samsung Store, κλπ.). Ομοίως, εταιρείες που κινούνταν παραδοσιακά στον χώρο του λογισμικού (Microsoft, Google, Canonical, κλπ.) κατευθύνονται πλέον προς την κατασκευή κινητών και ενσωματωμένων συστημάτων γενικά (MS Surface, Kinect, Google Glass και Ubuntu Tablet/Phone). *Οι τάσεις αυτές ασκούν πιέσεις στις εταιρείες για την αποτελεσματική ενσωμάτωση στοιχείων υλικού και λογισμικού και το κύριο αποτέλεσμα αυτής της πίεσης είναι η εμφάνιση ετερογενών συστημάτων και η ανάπτυξη και χρήση επιταχυντών υλικού (π.χ., GPU επιταχυντές).*

### 1.2.1 Ετερογενή υπολογιστικά συστήματα

Είναι γεγονός ότι οι υπολογιστές τύπου desktop δεν οδηγούν πλεον τη βιομηχανία, δεδομένου ότι τα smart phones και tablets προσελκύουν όλο και περισσότερους καταναλωτές και παραδοσιακές εφαρμογές, όπως η επικοινωνία και η ανταλλαγή εικόνων και βίντεο, προσφέρονται από τα κινητά συστήματα που μπορούν να χρησιμοποιηθούν οπουδήποτε, οποτεδήποτε. Ως αποτέλεσμα, η αγορά βλέπει μεγάλη ανάπτυξη

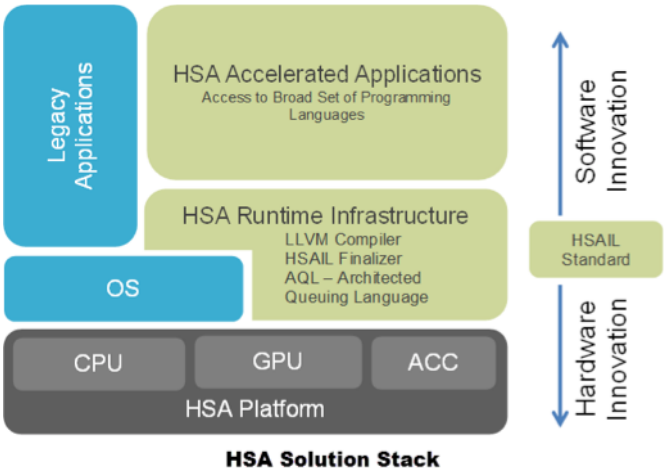
στα ενσωματωμένα συστήματα, η οποία ωθεί τους πόρους σχεδιασμού του υλικού σε αυτές τις κατευθύνσεις. Έτσι, τα σύγχρονα ενσωματωμένα συστημάτα περιλαμβάνουν προγραμματιζόμενους επιταχυντές για τη βελτίωση της απόδοσης και της ενεργειακής κατανάλωσης. Ωστόσο, αυτοί οι επιταχυντές, ακόμη και αν επικρατήσουν, δεν έχουν ακόμη αξιοποιηθεί πλήρως, λόγω της έλλειψης κατάλληλου λογισμικού και προτύπων προγραμματισμού (middleware, APIs).

Έτσι, η ετερογένεια και οι επιταχυντές υλικού είναι το κυρίαρχο χαρακτηριστικό των σύγχρονων υπολογιστικών συστημάτων. Για να αντιμετωπιστει το πρόβλημα της ορθής αξιοποίησης του υλικού, πολλές εταιρείς ομαδοποιήθηκαν και ίδρυσαν το Heterogeneous System Architecture (HSA) Foundation. Το HSA είναι μια μη κερδοσκοπική κοινοπραξία εταιρειών κατασκευής SoC, OEMs, πανεπιστημίων, OSVs και ISVs η οποία στοχεύει στην καλύτερη αξιοποίηση των CPUs, GPUs, DSPs και γενικά των διαθέσιμων επιταχυντών [5]. Μέλη του HSA είναι κορυφαίες παγκοσμίως εταιρείες, όπως η AMD, η ARM, η Samsung, η Qualcomm κλπ. Για να γίνει ευκολότερα ο προγραμματισμός των ετερογενων αρχιτεκτονικών, το HSA επιτρέπει στους προγραμματιστές να προγραμματίζουν σε ένα υψηλότερο επίπεδο αφαίρεσης χρησιμοποιώντας βιβλιοθήκες του HSA. Το Σχήμα 1.3 δείχνει μια αφηρημένη υλοποίηση του HSA Solution Stack. Ο βασικός παράγοντας που επιτρέπει μια γλώσσα προγραμματισμού να εκμεταλλευτεί την ετερογένεια, είναι η ύπαρξη ενός ενδιάμεσου στρώματος στο χρόνο εκτέλεσης που κρύβει τις ιδιαιτερότητες του υλικού, αφήνοντας τον προγραμματισμό του στις εταιρείες που το κατασκευάζουν.

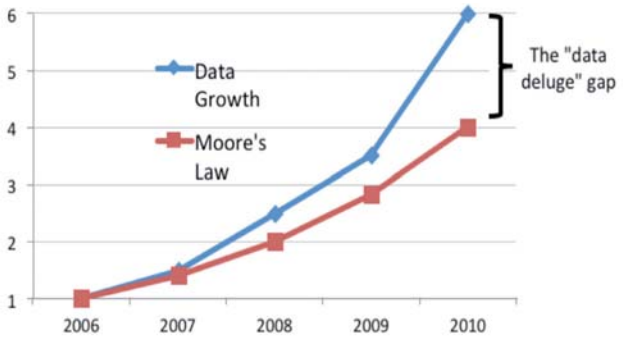
## 1.3 Δυναμικές εφαρμογές

Η σημερινή εποχή χαρακτηρίζεται απο την όλο και αυξανόμενη δημιουργία δεδομένων απο όλων των ειδών τα υπολογιστικά συστήματα, επιβεβαιώνοντας τον όρο Big Data για τις σύγχρονες εφαρμογές: εφαρμογές πολυμέσων και ασφάλειας, ιατρικές και cloud εφαρμογές κτλ. Το Σχήμα 1.4 παρουσιάζει τη σύγκριση της αυξησης των δεδομένων με το νόμο του Moore. Είναι σαφές ότι η αυξηση των δεδομένων ειναι εκθετική με ρυθμό μεγαλύτερο απο την ικανότητα υπολογισμού. Αυτή η άνευ προηγουμένου αύξηση των δεδομένων οδηγεί τις βιομηχανίες να επανεκτιμήσουν το κόστος της επικοινωνίας και την παροχη υπηρεσιών διαχειρισης μνήμης σε επίπεδο middleware χρησιμοποιώντας επιταχυντές υλικού για βελτίωση της απόδοσης και της ενεργειακής κατανάλωσης.

Η επόμενη γενιά των ενσωματωμένων συστημάτων θα κυριαρχείται από φορητές, έξυπνες συσκευές, οι οποίες θα είναι ικανές να προσφέρουν ένα μεγάλο εύρος υπηρεσιών και εφαρμογών επικοινωνίας και πολυμέσων σε οποιοδήποτε σημείο βρίσκονται. Η ενσωμάτωση εφαρμογών πολυμέσων και δικτύων, όπως H.264/AVC/SVC, JPEG 2000 και WiMax, δημιουργεί πολύπλοκες και δυναμικές εφαρμογές οι οποίες έχουν σημαντικές ανάγκες σε πόρους και απαιτήσεις πραγματικού χρόνου. Το χαρακτηριστικό σημείο αυτών των εφαρμογών είναι οι αυξημένες ανάγκες για αποθήκευση και μεταφορά δεδομένων καθώς και για αποδοτική διαχείριση της μνήμης. Η διαχείριση πόρων κατά τη φάση εκτέλεσης, αποτελεί κλειδί για την επιτυχή χρήση αυτών των




Σχήμα 1.3: Στοίβα πρωτόκολλου HSA [6]



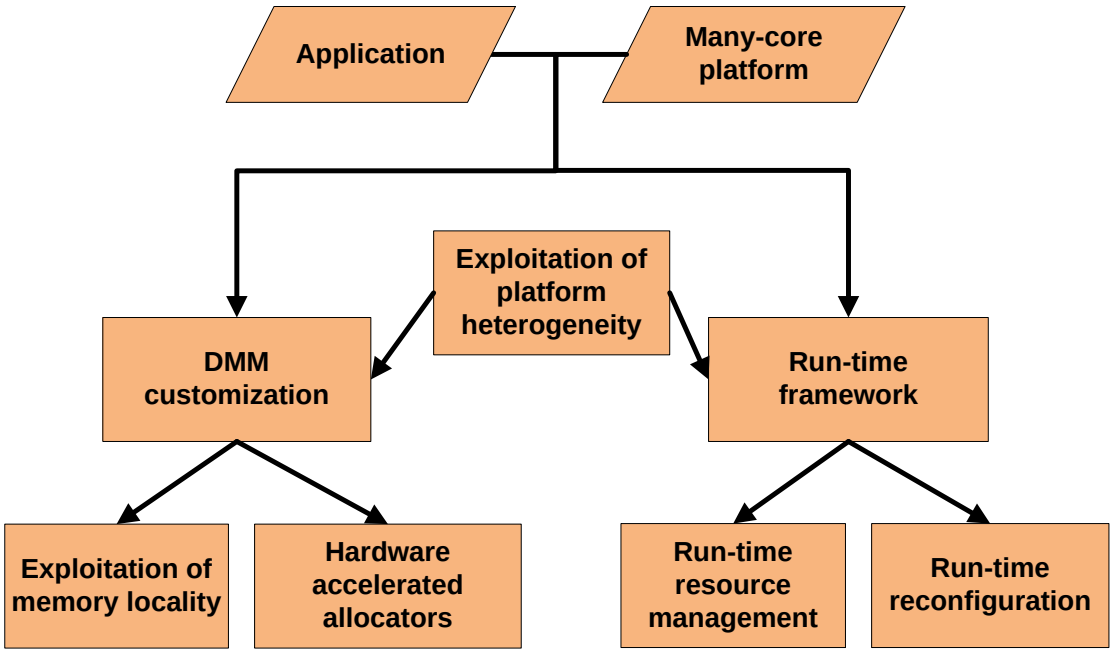

Σχήμα 1.4: Αύξηση δεδομένων σε σύγκριση με το νόμο του Moore τα τελευταία χρόνια [4]


Σχήμα 1.5: Διαχείριση κίνησης για κινητές εφαρμογές [3]

ενσωματωμένων συστημάτων. Το μοντέλο διαχείρισης πόρων κατά τη φάση εκτέλεσης έχει γίνει πρόσφατα εμφανές, επειδή μπορεί να αντιμετωπίσει τη δυναμικότητα των εφαρμογών και των πλατφορμών. Έτσι, η αποτελεσματική διαχείριση πόρων της εφαρμογής κατά τη φάση εκτέλεσης, επιτρέπει την αποτελεσματική χρήση των πόρων της πλατφόρμας.

Η διαχείριση μνήμης μπορεί να επιτευχθεί σε διάφορα επίπεδα αφαίρεσης: διαχείριση της εικονικής/δυναμικής μνήμης της εφαρμογής, σε αντίθεση με τη διαχείριση της φυσικής μνήμης. Επίσης, η διαχείριση της μνήμης μπορεί να εφαρμοστεί με μεγαλύτερη λεπτομέρεια/ακρίβεια στο επίπεδο ιεραρχίας μνήμης/αρχιτεκτονικής. Η αναποτελεσματική διαχείριση μνήμης οδηγεί σε μειωμένη απόδοση (κυρίως λόγω του χάσματος επιδόσεων μεταξύ των επεηεργασών και των μονάδων μνήμης), σε αυξημένες απαιτήσεις μνήμης και υψηλή κατανάλωση ενέργειας.

Υιοθετώντας τις πολυπήρηνες αρχιτεκτονικές, εφαρμογές που παλαιότερα εμφανίζονταν μόνο σε συτήματα υψηλών επιδόσεων, τώρα αρχίζουν να εμφανίζονται και σε ενσωματωμένα συστήματα. Από εφαρμογές υψηλών απαιτήσεων (αποκωδικοποίηση βίντεο, παιχνίδια, κλπ.) μέχρι το διαδικτύο, οι νέες εφαρμογές κατακλύουν τα ενσωματωμένα συστήματα. Ωστόσο, οι προγραμματιστές εφαρμογών δεν είναι πρόθυμοι να αλλάξουν τον τρόπο ανάπτυξης του πηγαίου κώδικα των εφαρμογών που οδηγεί σε ανεπαρκή χρήση της μνήμης. Οι περιορισμοί μνήμης που υπάρχουν στα ενσωματωμένα συστήματα δεν λαμβάνονται υπόψη με αποτέλεσμα την υποβάθμιση των επιδόσεων. Διατηρώντας τις ίδιες προγραμματιστικές διεπαφές (APIs) και τον πηγαίου κώδικα της εφαρμογής, μεγιστοποιώντας ταυτόχρονα τη χρήση της μνήμης φαίνεται να είναι πρόκληση για τους σχεδιαστές των συγχρονων ενσωματωμένων συστήματων.




Σχήμα 1.6: Επισκόπηση διδακτορικής διατριβής

## 1.4 Επισκόπηση Διδακτορικής Διατριβής

Η πρόκληση κατά το σχεδιασμό των ενσωματωμένων συστημάτων είναι η δυσκολία στην επίτευξη της συνεργασίας μεταξύ των εφαρμογών λογισμικού και των ετερογενών αρχιτεκτονικών. Η παρούσα διδακτορική διατριβή, επικεντρώνεται στη διαχειριση πόρων και την παραμετροποίηση εφαρμογών κατά το χρόνο εκτέλεσης σε ενσωματωμένες πολυπύρηνες πλατφόρμες . Μια επισκόπηση των αναπτυγμένων τεχνικών και πλαισίων παρουσιάζεται στο Σχήμα 1.6. Αφετηρία των αναπτυγμένων μεθοδολογιών είναι η εφαρμογή και η πολυπύρηνη πλατφόρμα πάνω στην οποία η εφαρμογή θα εκτελεστεί. Οι προτεινόμενες τεχνικές επικεντρώνονται στην ανάπτυξη Δυναμικών Διαχειριστών Μνήμης (ΔΔΜ) με χρήση επιταχυντών υλικού και διαχείριση πόρων στο χρόνο εκτέλεσης. Βασικό σημείο σε όλες τις τις τεχνικές είναι η εκμετάλλευση της ετερογένειας της πλατφόρμας και των διαθέσιμων επιταχυντών υλικού [11, 13]. Στον τομέα της επιτάχυνσης της διαχειρισης μνήμης στο επίπεδο middleware, αρχικά πραγματοποιείται προσαρμογή του ΔΔΜ με βάση την εφαρμογή και ακολουθείται η προσαρμογή με βάση την ετερογένεια της πλατφόρμας. Αυτό έχει ως αποτέλεσμα την καλύτερη χρήση της μνήμης, προσφέροντας κατανεμημένη λειτουργικότητα, και αξιοποίηση της ετερογένειας της πλατφόρμας. Η υλοποίηση σε επίπεδο middleware αποδεικνύεται να είναι μια λύση η οποία διατηρεί την ευελιξία του λογισμικού, ενώ εκμεταλλεύεται την παρουσία επιταχυντών υλικού. Στο τομέα της διαχείρισης πόρων στη φάση εκτέλεσης, ένα πλαίσιο απεικόνισης λαμβάνει υπόψη την ετερογένεια κάθε πλατφόρμας και χρησιμοποιεί καλύτερα την παρουσία επιταχυντών υλικού [10]. Επίσης, ακολουθώντας τις τάσεις σε διάφορα πεδία εφαρμογών, παρουσιάζεται ένα πλαίσιο διαχείρισης πόρων για παράλληλες εφαρμογές το οποίο επικεντρώνεται στις διαδικασίες αυτο-βελτίωσης [14].

Η διατριβή οργανώνεται σε έξι κεφάλαια ως εξής:

- Στο Κεφάλαιο 1, παρουσιάζονται οι τάσεις της βιομηχανίας για το λογισμικό και το υλικό στα ενσωματωμένα συστήματα. Επίσης, παρουσιάζεται και μια αφηρημένη επισκόπηση της διατριβής και διευκρινίζεται το επίκεντρο των αναπτυγμένων τεχνικών.

- Στο Κεφάλαιο 2, παρουσιάζεται και αναλύεται σε θέματα η συνεισφορά της διατριβής. Επίσης, αναφέρονται οι λύσεις για τα συγκεκριμένα προβλήματα και οι διαφοροποιήσεις με προηγούμενες προσεγγίσεις.

- Το Κεφάλαιο 3 εισάγει την έννοια της επιτάχυνσης της δυναμικής διαχείρισης μνήμης στο επίπεδο middleware. Προτείνεται ένα πλαίσιο για την δημιουργια προσαρμοσμένων και κατανεμημένων ΔΔΜ. Τέλος, παρουσιάζεται ένα πλαίσιο για τη σύζευξη της έννοιας της δυναμικής διαχείρισης μνήμης με DVFS τεχνικές, στοχεύοντας στη χαμηλή κατανάλωση ενέργειας.

- Στο Κεφάλαιο 4 αναλύονται τα πλαίσια για τη διαχείριση πόρων στο χρονο εκτέλεσης. Αρχικά, παρουσιάζεται ένα κατανεμημένο πλαίσιο απεικόνισης για ετερογενείς πλατφόρμες και στη συνέχεια παρουσιάζεται ένα πλαισιο για τη γενική διαχείριση πόρων.

- Το Κεφάλαιο 5 παρουσιάζει μια υψηλού επιπέδου μεθοδολογία προσαρμογής για τη διαχείριση των πόρων σε αρχιτεκτονικές, κανονικές και ακανόνιστες, τυπου Δίκτυο-σε-Ψηφιδα (Network-on-Chip, NoC).

- Τέλος, το Κεφάλαιο 6 συνοψίζει τα καινοτομικά στοιχεία και τα ευρήματα της παρούσας διατριβής και τονίζει μελλοντικές επεκτάσεις.

# Κεφάλαιο 2

# Συνεισφορά

Στο κεφάλαιο αυτό, παρουσιάζεται μια επισκόπηση της ερευνητικής περιοχής και τα προβλήματα αυτής, καθώς επίσης αναφέρεται και η συμβολή των προτεινόμενων λύσεων.

## 2.1 Στόχοι και Συνεισφορά

Η ερευνητική εργασία έχει πολλαπλούς στόχους. Έχουμε επικεντρωθεί στη διαχείριση πόρων και στην παραμετροποίηση εφαρμογών κατά τη φάση εκτέλεσης για πολυπύρηνα ενσωματωμένα συστήματα. Το ερευνητικό έργο επικεντρώθηκε ειδικά στον τομέα της (i) επιτάχυνσης διαχείρισης δυναμικής μνήμης στο επίπεδο του middleware, (ii) διαχείριση πόρων και (iii) παραμετροποίησης των πλατφορμών με βάση τις εφαρμογές κατά το χρόνο εκτέλεσης.

Αρχικά, παρουσιάζεται το πλαίσιο για επιτάχυνση και παραμετροποίηση διαχείρισης δυναμικής μνήμης σε ενσωματωμένα συστήματα. Οι τεχνικές αυτές εφαρμόζονται για να αντιμετωπιστούν αποτελεσματικά τα προβλήματα:

1. Της παροχής εξατομικευμένων Δυναμικών Διαχειριστών Μνήμης (ΔΔΜ) σε πολυπύρηνες ενσωματωμένες πλατφόρμες.

2. Της καλύτερης αξιοποίησης των χαρακτηριστικών των πλατφορμών.

3. Της σύζευξης των ΔΔΜ με DVFS τεχνικές στοχεύοντας σε χαμηλή κατανάλωση ισχύος.

Υιοθετώντας την προσέγγιση σε μικροκώδικα για την υποστήριξη προσαρμοσμένων ΔΔΜ για πολυπύρηνα συστήματα, επιτυγχάνουμε την αξιοποίηση του υλικού ταυτόχρονα με τη διατήρηση της ευελιξίας των εφαρμογών λογισμικού. Προκειμένου να διασφαλιστούν οι υψηλές επιδόσεις, οι προτεινόμενες υπηρεσίες ΔΔΜ αναπτύχθηκαν πάνω σε ένα επιταχυντή υλικού (Dual-Microcoded Controller, DMC) [33] ο οποίος: α) λειτουργεί ανεξάρτητα απο το σύστημα δρομολόγησης, β) είναι υπεύθυνος για τον χειρισμό κατανεμημένης μνήμης και γ) μετριάζει το φόρτο εργασίας του επεξεργαστή.

Οι κύριες συνεισφορές του προτεινόμενου πλαισίου για επιτάχυνση και παραμετροποίηση διαχείρισης δυναμικής μνήμης είναι:

1. **Διαχειριστής μνήμης σε μικροκώδικα [11]**: Προτείνουμε ένα πλαίσιο για τη δημιουργία ΔΔΜ σε μικροκώδικα (microcode) χρησιμοποιώντας τον επιταχυντή υλικού DMC [33] και αξιοποιώντας τα χαρακτηριστικά της πλατφόρμας με Κατανεμημένη Κοινή Μνήμη (Distributed Shared Memory, DSM). Η χρήση του DMC για ΔΔΜ διευκολύνει τους επεξεργαστές της πλατφόρμας καθώς πάβουν να είναι υπεύθυνοι για την εξυπηρέτηση των αιτήσεων και την παρακολούθηση της κατάστασης του σωρού.

2. **Κατανεμημένη λειτουργικότητα και επεκτασιμότητα**: Προτείνουμε ένα ευέλικτο, κλιμακούμενο και κατανεμημένο microcoded ΔΔΜ, που ονομάζεται MAD-DMM. Ο MAD-DMM, ο οποίος είναι υλοποιημένος σε μικροκώδικα, παρέχει κατανεμημένη λειτουργικότητα σε DSM περιβάλλον, διατηρώντας παράλληλα τις προγραμματιστικές διεπαφές (Application Interfaces, API) C-API (`malloc()/free()`). Ο MAD-DMM χειρίζεται το σωρό σαν ένα συνεχή χώρο χρησιμοποιώντας ένα χάρτη που ονομάσαμε Heap Space Map (HSM) και όλοι οι κόμβοι της πλατφόρμας είναι ενήμεροι για την κατάσταση του σωρού. Σε αντίθεση με ΔΔΜ υψηλού επιπέδου (C), οι πληροφορίες σχετικά με τα μεταδεδομένα του διαχειριστή δεν αποθηκεύονται σε δομές υψηλού επιπέδου αλλά σε μικροκώδικα σαν ένας τοπικός πίνακας.

3. **Επόπτευση κατανάλωσης ενέργειας [13]**: Προτείνουμε μια νέα στρατηγική σχεδιασμού για την υλοποίηση μιας αποτελεσματικής μεθοδολογίας για τη συνδυασμένη χρήση τεχνικών DVFS μαζί με ΔΔΜ με στόχο την χαμηλή κατανάλωση ισχύος.

4. **Αξιολόγηση**: Η αξιολόγηση των προτεινόμενων τεχνικών και μεθοδολογιών έγινε μέσω εκτεταμένων και εξερευνητικών πειραματισμών.

Όσον αφορά τους ΔΔΜ σε μικροκώδικα, τα πειραματικά αποτελέσματα δείχνουν ότι η προτεινόμενη προσέγγιση για το σχεδιασμό εξατομικευμένων ΔΔΜ σε μικροκώδικα, οι οποίοι γνωρίζουν την κατανομή μνήμης [11], (α) μπορεί να εξυπηρετήσει περισσότερα αιτήματα με τη χρήση όλων των διαθέσιμων σορών της πλατφόρμας, (β) αυξάνει τη διάρκεια ζωής του σωρού, (γ) είναι πλήρως παραμετροποιήσιμη και εύκολη στη χρήση (προσφέροντας πρότυπα σε μικροκώδικα), (δ) επιτυγχάνει καλύτερη απόδοση αξιοποιώντας την παρουσία του DMC για το χειρισμό κατανεμημένων αιτημάτων και (ε) έχει μικρή επιβάρυνση όσον αφορά την κατανάλωση ενέργειας. Συγκεκριμένα, το κέρδος είναι περίπου 7× όσο αφορά τον αριθμό των εξυπηρετούντων αιτημάτων για δυναμική παραχώρηση μνήμης με μια μικρή αύξηση της τάξεως του 14% σε μέση κατανάλωση ισχύος. Επίσης, ο ΔΔΜ σε μικροκώδικα αποδείχτηκε κατά μέσο όρο 25% ταχύτερος από τον αντίστοιχο ΔΜΜ σε C επαληθεύοντας την επιλογή μας για την χρήση ενός επιταχυντή υλικού. Επιπλέον, ο ΔΔΜ MAD-DMM αποδείχθηκε κατά μέσο όρο 25% πιο αργός από τον ΔΔΜ σε μικροκώδικα [11] και 10% ταχύτερος από τον ΔΔΜ υψηλού επιπέδου [61]. Παρ'όλα αυτά, ο MAD-DMM αποδείχθηκε πιο επεκτάσιμος, αφού ο ΔΔΜ σε μικροκώδικα χρειάζεται κατά μέσο όρο 29% περισσότερους κύκλους για να εξυπηρετήσει ένα αίτημα δυναμικής παραχώρη-

σης μνήμης κάθε φορά που αυξάνεται το μέγεθος της πλατφόρμας, ενώ ο MAD-DMM μόνο 20%. Τέλος, το προτεινόμενο πλαίσιο για την σύζευξη DVFS τεχνικών μαζί με ΔΔΜ έδειξε ότι με τη χρήση της προτεινόμενης μέθοδου για την παρακολούθηση και την εφαρμογή DVFS μηχανισμών, η κατανάλωση ενέργειας σχετικά με τη διαχείριση του σωρού μειώθηκε κατά 37% περίπου. Επιπλέον, με την προτεινόμενη μέθοδο μπορούμε να επιτύχουμε χαμηλή κατανάλωση ισχύος με χαμηλό κατακερματισμό του σωρού.

Στη συνέχεια, παρουσιάζονται μεθοδολογιες για την διαχείριση πόρων στο χρόνο εκτέλεσης σε πολυπύρηνες πλατφόρμες. Συγκεκριμένα, (i) χρησιμοποιείται η ιδέα του *Διαίρει και Βασίλευε* για την παροχή κατανεμημένων υπηρεσιών χαρτογράφησης στο χρόνο εκτέλεσης τόσο για ομογενείς όσο και για ετερογενείς πολυπύρηνες πλατφόρμες, (ii) αναπτύχθηκε μια μεθοδολογία για την διαχείρηση εύπλαστων εφαρμογών με βάση το κόστος ανα πυρήνα και (iii) παρουσιάζεται ένα πλαίσιο και μια μεθοδολογία υψηλού επιπέδου για τη διαχείριση των πόρων σε αρχιτεκτονικές τύπου Δίκτυο-σε-Ψηφίδα, τόσο για κανονικές (regular) όσο και για ακανόνιστες (irregular) τοπολογίες. Τα προτεινόμενα πλαίσια βασίζονται στην ιδέα της χρήσης πολλαπλών πυρήνων σε διαφορετικούς ρόλους και σε κάθε περίπτωση ένα σύστημα διασυνδεσης εξασφαλίζει την κατανεμημένη λειτουργικότητα. Τα προτεινόμενα πλαίσια αξιολογήθηκαν στην πειραματική πολυπύρηνη πλατφόρμα που παρουσιάζεται στο [33] και στην πλαφόρμα Intel Single Chip Cloud (SCC) [49].

Οι κύριες συνεισφορές των προτεινόμενων μεθοδολογιών για την διαχείριση πόρων στο χρόνο εκτέλεσης σε πολυπύρηνες πλατφόρμες είναι [10, 14]:

1. **Υποστήριξη ετερογενών πλατφορμών**: Το προτεινόμενο πλαισιο για κατανεμημένη απεικόνιση στο χρόνο εκτέλεσης υποστηρίζει τόσο ομογενείς όσο και για ετερογενείς πολυπύρηνες πλατφόρμες.

2. **Αξιοποίηση διαθέσιμων πόρων**: Οι προτεινόμενες κατανεμημένες μεθοδολογίες στο χρόνο εκτέλεσης μπορούν να επιτύχουν διαφορετικά επίπεδα αξιοποίησης των διαθέσιμων πόρων, ανάλογα με τις ανάγκες των εφαρμογών, σε σύγκριση με άλλες λύσεις [9, 54]

3. **Εξυπηρέτηση των αναγκών των εφαρμογών**: Οι προτεινόμενες μεθοδολογίες εξασφαλίζουν ότι οι εφαρμογές θα πάρουν το βέλτιστο αριθμό πυρήνων αποφεύγοντας ταυτόχρονα φαινόμενα αποκλεισμών.

4. **Εκμετάλλευση ετερογένειας**: Οι προτεινόμενες μεθοδολογίες λαμβάνουν υπόψη το είδος των επεξεργαστών (ετερογένεια) με μια μικρή επιβάρυνση στο συνολικό κόστος επικοινωνίας.

5. **Αξιολόγηση**: Η αξιολόγηση των προτεινόμενων τεχνικών και μεθοδολογιών έγινε μέσω εκτεταμένων και εξερευνητικών πειραματισμών τόσο σε πειραματικές [33] όσο και σε βιομηχανικές [49] πολυπύρηνες πλατφόρμες.

Τα πειραματικά αποτελέσματα για την προτεινόμενη μεθοδολογία απεικόνισης στο χρόνο εκτέλεσης, με τη μέθοδο του Διαίρει και Βασίλευε, παράγει κατά μέσο όρο

21% και 10% καλύτερο κόστος επικοινωνίας για ομοιογενείς και ετερογενείς πλατφόρμες αντίστοιχα, σε σύγκριση άλλους κατανεμημένους αλγόριθμους [9] με σχεδόν την ίδια υπολογιστική προσπάθεια. Τα τυχαία σενάρια έδειξαν ότι ο προτεινόμενος αλγόριθμος μπορεί να έχει διαφορετική συμπεριφορά και αποτέλεσμα της αξιοποίησης των πόρων της πλατφόρμας. Επίσης, το πλαίσιο για την κατανεμημένη διαχείριση εύπλαστων εφαρμογών έχει κατά μέσο όρο 70% λιγότερα μηνύματα, 64% μικρότερο μέγεθος μηνυμάτων και 20% κέρδος στην επιτάχυνση των εφαρμογών συγκρινόμενο με το κατανεμημένο πλαίσιο που παρουσιάζεται στο [54]

## 2.2 Συναφή βιβλιογραφία

Ολοένα και περισσότερο, η σημερινή τάση σχεδιασμού Συστημάτων-σε-Ψηφίδα (System-on-Chip, SoC) βασίζεται στη χρήση πολλαπλών επεξεργαστών. Αυτή η τάση σχεδιασμού αντιπροσωπεύει τόσο τις αρχιτεκτονικές γενικού σκοπού [81] όσο και τις αρχιτεκτονικές των ενσωματωμένων συστημάτων [8, 47]. Για να αξιοποιηθεί αυτός ο υψηλός αριθμός πυρήνων οι σύγχρονες εφαρμογές γίνονται ολοένα και περισσότερο πολυνηματικές.

### 2.2.1 Επιτάχυνση διαχείρισης μνήμης σε middleware επίπεδο

Υπάρχουν τρεις τρόποι για την παροχή υπηρεσιών διαχείρισης μνήμης σε DSM πλατφόρμες. Αρχικά, οι ΔΔΜ σε λογισμικό αποτελούν την τρέχουσα πρακτική, είναι ευέλικτοι αλλά καταναλώνουν πολλούς κύκλους επεξεργαστή, οι οποίοι περιορίζουν την απόδοση του συστήματος. Εκτεταμένη έρευνα έχει διεξαχθεί για ΔΔΜ γενικής χρήσης που έχουν σαν στόχο είτε τους μονοπύρηνες [90, 91], είτε τους πολυπύρηνους [20, 52, 58, 59, 94] επεξεργαστές. Ωστόσο, το ποσό της εργασίας που εκτελούν οι ΔΔΜ δεν είναι πάντα το ίδιο, αλλάζει κατά το χρόνο εκτέλεσης καθω η κατάσταση του σωρού και ο αριθμός των αιτημάτων ποικίλλει, καθώς επίσης εξαρτάται και από τον τύπο των κλήσεων και των παραμέτρων αυτών. Η ανάπτυξη πολυνηματικών δυναμικών εφαρμογών, χρησιμοποιώντας εκτιμήσεις χειρότερης απόδοσης με στατικό τρόπο, έχουν αντίκτυπο στον κατακερματισμό της μνήμς και στην κατανάλωση ισχύος. Για την αποφυγή αυτών των καταστάσεων οι προγραμματιστές οφείλουν να χρησιμοποιούν αποτελεσματικά τη δυναμική μνήμη του συστήματος.

Οι υλοποιήσεις ΔΔΜ σε υλικό μπορούν να επιτύχουν υψηλή απόδοση, αλλά κάθε μικρή αλλαγή στη λειτουργία τους οδηγεί και σε επανασχεδιασμό του συνόλου της μονάδας. Ένας ΔΔΜ που ευνοεί την τοπικότητα της μνήμης cache για ειδικά SMP συστήματα προτείνεται στο [74]. Οι συγγραφείς του [16] μελέτησαν τη διαχείριση του σωρού στον επεξεργαστή Cell αλλά δεν χειρίζονται την κοινή μνήμη. Αντιθέτως, οι μονάδες επεξεργασίας πρέπει να χειρίζονται την δικιά τους, αποκλειστική μνήμη και να επικοινωνούν με το σύστημα μέσω ρητών DMA κλήσεων, ένας περιορισμός ο οποίος θέτεται απο το υλικό.

Η προσέγγιση με μικροκώδικα αποτελεί μια καλή εναλλακτική λύση για να ξεπεραστεί το δίλημμα επιδόσεων-ευελιξίας και την επιτάχυνση ενός μεγάλου φάσματος εφαρμογών [89]. Για αυτόν τον λόγο, *υιοθετούμε την προσέγγιση με μικροκώδικα για την αντιμετώπιση DSM προβλημάτων σε πολυπύρηνες πλατφόρμες, στοχεύοντας στην καλύτερη αξιοποίηση του υλικού διατηρώντας όμως την ευελιξία των προγραμμάτων*. Προσεγγίσεις με μικροκώδικα έχουν χρησιμοποιηθεί σε προηγούμενα πολυπύρηνα συστήματα για την επίλυση προβλημάτων που σχετίζονται με τη διαχείριση μνήμης σε DSM πλατφόρμες. Το σύστημα Alewife [7] αντιμετωπίζει το πρόβλημα της παροχής ενιαίου χώρου διευθύνσεων με τη χρήση περάσματος μηνυμάτων (message passing). Ωστόσο, αποτελεί πολύ εξειδικευμένη λύση και δεν υποστηρίζει εικονική μνήμη. Το FLASH [56] και το Typhoon [70] χρησιμοποιούν ένα προγραμματιζόμενο συν-επεξεργαστή για την υποστήριξη πρωτοκόλλου συνεκτικότητας της μνήμης cache. Ωστόσο, τόσο το FLASH όσο και το Typhoon έχουν μόνο έναν προγραμματιζόμενο συνεπεξεργαστή για να εξυπηρετήσει τα αιτήματα από το δίκτυο και τη CPU. Αν δύο ή περισσότερα αιτήματα καταυθάσουν ταυτόχρονα, μόνο ένα μπορεί να εξυπηρετηθεί με αποτέλεσμα την αύξηση της καθυστέρησης. Σε σύγκριση με την ειδικές λύσεις υλικού (hardware), ο τοπικός επεξεργαστής ξοδεύει πολύ περισσότερο χρόνο στην πρόσβαση των δεδομένων, ακόμη και όταν χρησιμοποιείται τοπικά. Στην δικιά μας υλοποίηση η μνήμη χωρίζεται σε 2 τμήματα: (i) το ιδιωτικό (private) και (ii) το κοινό (shared) τμήμα. Η πρόσβαση στην ιδιωτική μνήμη είναι τοπική και γρήγορη, έτσι ώστε να βελτιώνει τις επιδόσεις. Το SMTp [32] εκμεταλλεύεται το SMT σε συνδυασμό με έναν πρότυπο ενσωματωμένο ελεγκτή μνήμης για να καταστεί δυνατή η συνεκτικότητα της μνήμης cache σε DSM πολυπύρηνα συστήματα. Η δυνατότητα προγραμματισμού του πρωτοκόλλου προσφέρεται από ένα νήμα του συστήματος παρά απο έναν επιπλέον προγραμματιζόμενο συνεπεξεργαστή.

### 2.2.2 Κατανεμημένη δυναμική διαχείριση μνήμης σε μικροκώδικα

Οι παραδοσιακές τεχνικές βελτιστοποιήσεις, για SoC συστήματα, χρησιμοποιούν πληροφορίες κατά τη φάση της μεταγλώττισης και επικεντρώνονται στην στατική παραχώρηση μνήμης με βάση τη σύνθεση της ιεραρχίας μνήμης [29]. Όμως, για τις σύγχρονες δυναμικές εφαρμογές αυτό δεν είναι πλέον δυνατό, καθώς η δυναμική συμπεριφορά των εφαρμογών, λόγω των δυναμικών εισόδων, δεν μπορεί να συλληφθεί από την ανάλυση του πηγαίου κώδικα και μόνο. Όπως παρουσιάζεται στο [85], η διαχείριση της δυναμικής μνήμης για το MapReduce [37] διαδραματίζει ουσιαστικό ρόλο στην συνολική απόδοση του συστήματος. Αυτό δημιουργείται και απο το γεγονός ότι οι ΔΔΜ δεν είναι επεκτάσιμοι [95]. Επιπλέον, η στατική κατανομή της μνήμης οδηγεί σε αναποτελεσματική αξιοποίησή της [15]. Έτσι, η συμπεριφορά των εφαρμογών και οι απαιτήσεις της μνήμης διαφέρουν σημαντικά κατά τη διάρκεια του χρόνου εκτέλεσης. Η δυναμική διαχείριση μνήμης (ΔΔΜ) είναι ένα κρίσιμο συστατικό στα σύγχρονα ενσωματωμένα συστήματα, δεδομένου ότι συχνά αποτελεί τον κύριο λόγο περιορισμού της απόδοσης και της επεκτασιμότητας των πολυνηματικών εφαρμογών [20]. Επίσης, η μνήμη επηρεάζει σε μεγάλο βαθμό την κατανάλωση ισχύος-ενέργειας του συνολικού συστήματος [15]. Επεκτάσιμοι ΔΔΜ γενικού σκοπού έχουν προταθεί για

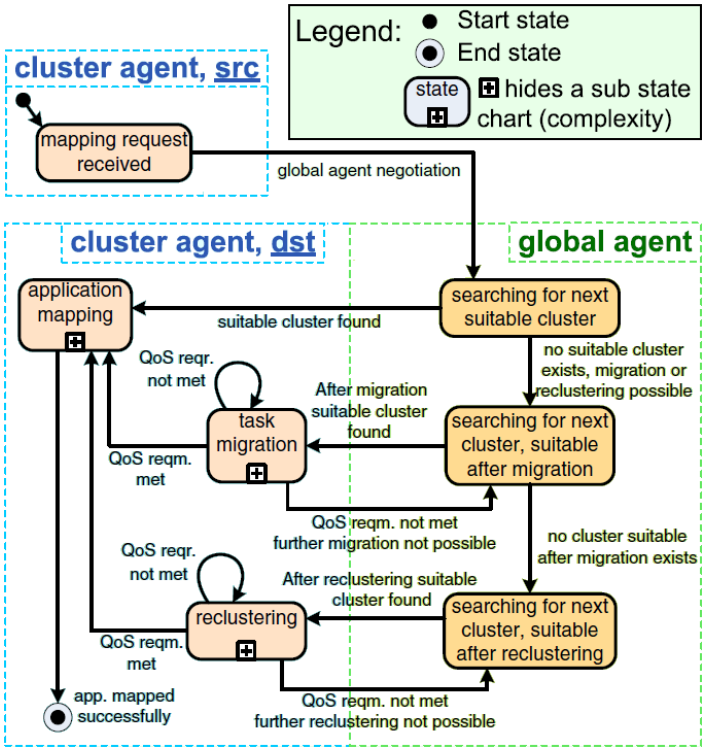
πολυπύρηνα συστήματα [72], αλλά επικεντρώνονται κυρίως στα θέματα συγχρονισμού και δεν μελετούν κατανεμημένα συστήματα μνήμης. Στον τομέα των υπολογιστών υψηλών επιδόσεων, η γενική διευθυνσιοδότηση αποτελεί σημαντικό ζήτημα και η παροχή κατανεμημένων λύσεων για δυναμική διαχείριση μνήμης έχει προταθεί στο [60]. Παρ'όλα αυτά, η συγκεκριμένη ερευνητική εργασία στοχεύει συστήματα τύπου cluster τα οποία δεν εκμεταλλεύονται την τοπικότητα και τα πλεονεκτήματα των NoC συστημάτων.

Επιταχυντές υλικού για δυναμική διαχείριση μνήμης έχουν προταθεί από πολλές ερευνητικές εργασίες [31, 91]. Μια μονάδα υλικού διαχείρισης μνήμης, με το όνομα SoCDMMU, υπεύθυνη για τη δυναμική κατανομή της μνήμης παρουσιάζεται στο [78]. Ωστόσο, αποτελεί μια κεντρικοποιημένη μονάδα και θα μπορούσε να αποτελέσει πιθανό σημείο συμφόρησης σε πολυπύρηνα συστήματα. Επιπλέον, το SoCDMMU είναι σε θέση να παρέχει μόνο σελίδες μνήμης και η διαχείριση των δεδομένων δεν γίνεται απο τους επεξεργαστές. Η μονάδα υλικού, με το όνομα HwMMU, προσφέρει δυναμική διαχείριση δεδομένων σε DSM πλατφόρμες και προτείνεται στο [63]. Παρ'όλα αυτά το HwMMU υποστηρίζει τη δυναμική διαχείριση δεδομένων σε ένα κομμάτι πλήρους μνήμης και η κλήση γίνεται με νέες προγραμματιστικές διεπαφές.

### 2.2.3 Διαχείριση πόρων στο χρόνο εκτέλεσης

Οι συγγραφείς στο [68] παρουσιάζουν μια στατηγική απεικόνισης και προγραμματισμού για ενσωματωμένα συστήματα αυστηρά πραγματικού χρόνου, στην οποία η επικοινωνία γίνεται μέσω ενός διαμοιραζομενου μέσου σοτχευοντας στον ελαχιστοποίηση του κόστους αλλαγής. Μια μεθοδολογία απεικόνισης στο χρόνο εκτέλεσης για αρχιτεκτονικές τύπου Δικτυο-σε-Ψηφίδα με πολλαπλά επίπεδα τροφοδοσίας παρουσιάζεται στο [34]. Η τεχνική αυτή συνίσταται από ένα αλγόριθμο επιλογής περιοχής και έναν υβριστικό αλγοριθμο απεικόνισης στο χρόνο εκτέλεσης. Στο [25], οι συγγραφείς προτείνουν τον αλγόριθμο MinWeight για την επίλυση την επίλυση του πρόβληματος ελάχιστου βάρους, αλλά μόνο για γράφους εργασιών (task graphs) με μέγιστο βαθμό δύο. Οι συγγραφείς στο [82] επέκτειναν τον προηγούμενο αλγόριθμο με την επίλυση του προβλήματος της ανάθεσης, στο χρόνο εκτέλεσης, διεργασιών σε ετερογενείς επεξεργαστές με γράφους εργασιών οι οποίοι περιορίζονται σε ένα μικρό αριθμό κορυφών ή ένα μεγάλο αριθμό κορυφών με βαθμό όχι περισσότερο από δύο [25]. Ένας ενοποιημένος αλγόριθμος μοναδκού σκοπού με το όνομα UMARS, συνδυάζει την επιλογή μονοπατιού, απεικόνισης πυρήνων και TDMA κατανομής τέτοια ώστε το δίκτυο που πρέπει να ανταποκρίνεται στις απαιτήσεις της εφαρμογής να ελαχιστοποιείται [43].

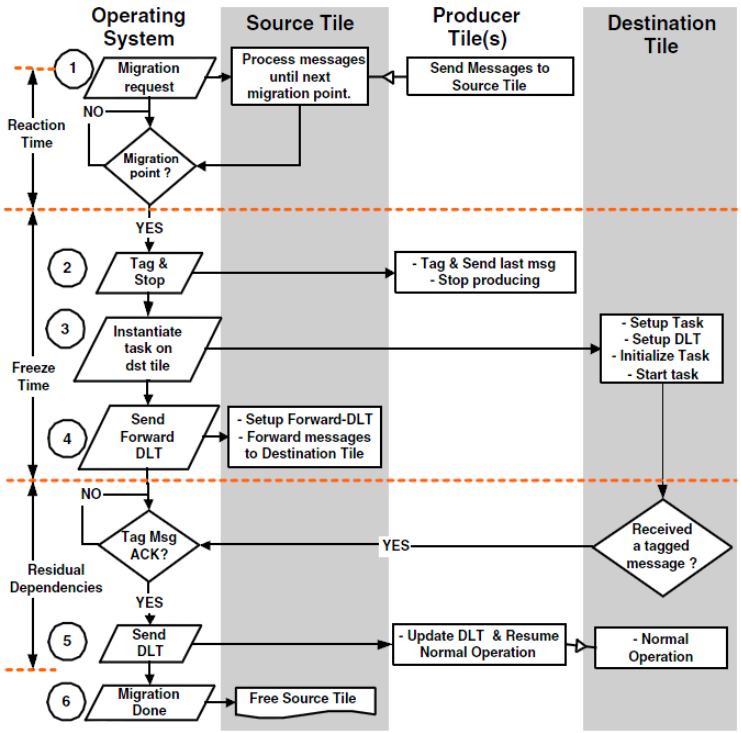
Στο [9], παρουσιάζεται μια μεθοδολογία κατανεμημένης απεικόνισης στο χρόνο εκτέλεσης χρησιμοποιώντας μέσα διασποράς (agents) για προσμαρμοζόμενες αρχιτεκτονικές τύπου Δίκτυο-σε-Ψηφίδα. Η βασική ιδέα είναι ότι, προκειμένου να επιτευχθεί ο κατανεμημένος υπολογισμός της διαδικασίας απεικόνισης, η πλατφόρμα είναι χωρισμένη σε εικονικές ομάδες (cluster) και ο υπολογισμός της απεικόνισης σε κάθε ομάδα γίνεται ανεξάρτητα. Πιο συγκεκριμένα, μία ομάδα είναι ένα υποσύνολο του συνόλου




Σχήμα 2.1: Σχεδιάγραμμα ροής του αλγόριθμου ADAM [9]

των διαθέσιμων πυρήνων. Τα όριά της δεν είναι καθορισμένα και μπορεί να αλλάξουν ανά πάσα στιγμή, συμπεριλαμβάνοντας περισσότερους πυρήνες ή αποκλείοντας άλλους που προηγουμένως άνηκαν σε αυτή. Ένας απο τους πυρηνες κάθε ομάδας δρά ώς μέσο διασποράς (agent). Αυτός ο πυρήνας είναι μια υπολογιστική οντότητα, η οποία ενεργεί για λογαριασμό τρίτων. Ο agent κάθε ομάδας είναι υπεύθυνος για την διαδικασία απεικόνισης μέσα στην ομάδα (Σχήμα 2.1).

Οι συγγραφείς υποστηρίζουν ότι μια κεντρικοποιημένη διαχείριση πόρων στο χρόνο εκτέλεσης μπορεί να έχει μια σειρά από προβλήματα, όπως ένα ενιαίο σημείο αποτυχίας και απαιτεί μεγάλο όγκο για την παρακολούθηση της κίνησης. Ωστόσο, στο [67] παρουσιάζεται ένα κεντρικοποιημένο σύστημα διαχείρισης που είναι σε θέση να διαχειριστεί αποτελεσματικά ένα Δίκτυο-σε-Ψηφίδα με επαναδιαμορφώσιμα στοιχεία και δύο αλγόριθμους για μετακίνηση εργασιών (task migration). Η υβριστική διαχείριση πόρων αποτελείται από ένα βασικό αλγόριθμο πάνω σε επαναδιαμορφούμενα στοιχεία. Ο υβριστικός αλγόριθμος περιέχει ιδέες από πολλαπλές προσεγγίσεις για τη διαχείριση των πόρων. Ο προτεινόμενος μηχανισμός βασίζεται στην υπόθεση ότι πολλοί αλγόριθμοι χρησιμοποιούν σύστημα διοχέτευσης (pipeline) και όχι σταθερά στοιχεία. Αυτά τα μη σταθερά στοιχεία είναι στιγμές όπου καινούρια και ανεξάρτητα δεδομένα εισέρχονται στη διοχέτευση. Η υπόθεση αυτή επιτρέπει στο μηχανισμό μετακίνησης εργασιών να μετακινεί πολλαπλές διοχετευμένες εργασίες ταυτόχρονα χωρίς να υπάρχει ανησυχία για τη μεταφορά των καταστάσεων. Αυτός ο μηχανισμός είναι χρήσιμος όταν νέες απαιτήσεις QoS επηρεάζουν μια εφαρμογή και πρέπει να




Σχήμα 2.2: Γενικός μηχανισμός μετακίνησης εργασιών [67]

γίνει μετακίνηση εργασιών. Ο αλγόριθμος απεικόνισης που προτείνεται στο [67] δεν είναι ο πιο αποτελεσματικός δεδομένου ότι αντιμετωπίζει τους περιορισμούς της κεντρικοποιημένης διαχερίσης. Παρ 'όλα αυτά, οι προτεινόμενοι μηχανισμοί (Σχήμα 2.2) μετακίνησης εργασιών είναι πολύ χρήσιμοι ως τμήματα κάθε διαχειριστή που χρησιμοποιεί τέτοιες στο χρόνο εκτέλεσης. *Ωστόσο, ακόμη και αν αυτές οι προσεγγίσεις χειρίζονται την διαδικασία απεικόνισης εφαρμογων κατά το χρόνο εκτέλεσης πολύ καλά, έχουν σχεδιαστεί για εφαρμογές σταθερού μεγέθους χωρίς οποιαδήποτε μορφή ευπλασίας (malleability).* Έτσι, δεν πραγματοποιείται επαναδιαμόρφωση των εφαρμογών σε τυχόν δυναμικές αλλαγές των διαθέσιμων πόρων της πλατφόρμας και καμία αλλαγή διαστάσεων ή ανακατάταξη επιτρέπεται.

Στο τομέα των αυτο-οργανωμένων και δυναμικών συστημάτων και από την πλευρά των παράλληλων εφαρμογών, οι συγγραφείς στο [71] παρουσιάζουν μια άπληστη κεντρικοποιημένη στρατηγική προγραμματισμού και αποδείκνύουν ότι η σημασία της αποδοτικότητας ποικίλλει σε σχέση με τα χαρακτηριστικά του φόρτου εργασίας που έχει ο χρονοπρογραμματιστής (scheduler). Η κύρια ιδέα είναι ότι ο χρονοπρογραμματιστής ορίζει ένα μέγιστο επιτρεπόμενο μέγεθος διαμέρισης για τα στοιχεία επεξεργασίας ανά εφαρμογή και αργότερα χρησιμοποιεί μια άπληστη στρατηγική για την επιλογή ένα μεγέθους διαμέρισης με στόχο την ελαχιστοποίηση του χρόνου απόκρισης. Το θεμελιώδες πρόβλημα της απεριόριστης άπληστης προσέγγισης για την επιλογή μεγέθους διαμέρισης για όλες τις εφαρμογές είναι ότι οι περισσότερες εργασίες τείνουν να επιλέγουν πολύ μεγάλα μεγέθη διαμέρισης όσον αφορά την χρήση των

επεξεργαστικών στοιχειων. Σαν αποτέλεσμα, οι συγγραφείς στο [71] παρουσιάζουν ένα δίκαιο αλγόριθμο διαχείρισης λαμβάνοντας υπόψη το βάρος κάθε εργασίας ώστε να αξιοποιήσουν τους πόρους με έναν πιο δίκαιο τρόπο. Στο [38], παρουσιάζεται ότι οι εύπλαστες (malleable) εφαρμογές παρέχουν μέχρι 15% επιτάχυνση σε σύγκριση με τη μετακίνηση εργασιών σε ένα δυναμικό περιβάλλον. Για την εξέταση και την αξιοποίηση της ευπλαστότητας, δύο αντιπροσωπευτικές εφαρμογές επιλέχθηκαν και τροποποιήθηκαν. Η πρώτη είναι μία εφαρμογή αστρονομίας, ενώ η δεύτερη είναι μια εφαρμογή που προσομοιώνει τη διάχυση θερμότητας. Και οι δύο έχουν μοναδική ανάγκη στο χειρισμό των δεδομένων και της ευπλαστότητας. Η διαθεσιμότητα των πόρων μεταβάλλεται κάνοντας ομάδες (clusters) να είναι διαθέσιμες ή μη στις εφαρμογες. Στο [54], παρουσιάζεται μια μεθοδολογία, με χρήση διαχειριστών (agents), για κατανεμημένη απεικόνιση εύπλαστων εφαρμογών υποστηρίζοντας επίσης και αυτοοργάνωση. Ο διαχειριστής (agent) ορίζεται κατά το χρόνο εκτέλεσης σε έναν τυχαίο πυρήνα, όταν μια νέα εφαρμογή εισέρχεται στο σύστημα, με το μειονέκτημα της πιθανής συμφόρησης επικοινωνίας όταν ένας τυχαία ήδη κατειλημμένος πυρήνας καλείται να εξυπηρετήσει τη νέα εφαρμογή. Ο πράκτορας έχει ανατεθεί κατά το χρόνο εκτέλεσης σε έναν τυχαίο πυρήνα όταν μια νέα εφαρμογή έρχεται με το μειονέκτημα της πιθανής συμφόρησης επικοινωνίας, όταν ένα τυχαία επιλεγμένο ήδη κατειλημμένη πυρήνα εξυπηρετεί τη νέα αίτηση. Για τη διαχείριση των περιφερειακών πληροφοριών με κατανεμημένο τρόπο, οι συγγραφείς προτείνουν μια υπηρεσία καταλόγου κατανεμημένη σε ολόκληρη την πλατφόρμα. Αυτή η υπηρεσία καταλόγου δίνει τη δυνατότητα στους διαχειριστές (agents) να επικοινωνούν με άλλους, χωρίς την ανάγκη της αναμετάδοσης των μηνυμάτων. Όλοι οι διαθέσιμοι πυρήνες χωρίζονται σε ομοιόμορφα κατανεμημένες ομάδες και κάθε ομάδα περιέχει μία υπηρεσία καταλόγου που εκτελείται σε έναν από τους πυρήνες. Οι διαχειριστές (agents) εγγράφονται στους καταλόγους που αντιστοιχούν στους πυρήνες που εκτελούν τη δική τους εφαρμογή. *Οι διαφοροποιήσεις των προτεινόμενων μεθόδων της διδακτορικής διατριβής έγκειται στο γεγονός ότι (i) οι παραπάνω προσεγγίσεις δεν λαμβάνουν υπόψη ειδικά χαρακτηριστικά ετερογένειας αφήνοντας αναξιοποίητες τις δυνατότητες της πλατφόρμας και (ii) οι προτεινόμενες προσέγγισες έχουν μικρή επιβάρυνση στο συνολικό κόστος επικοινωνίας.*

# Κεφάλαιο 3

# Επιτάχυνση και παραμετροποίηση διαχείρισης μνήμης

## 3.1 Εισαγωγή

Η τρέχουσα τάση στην πληροφορική και στα ενσωματωμένα συστήματα είναι η αντικατάσταση πολύπλοκων superscalar αρχιτεκτονικών με πολλές μονάδες επεξεργασίας οι οποίες συνδέονται μεταξύ τους με ένα on-chip δίκτυο. Τα μελλοντικά ολοκληρωμένα συστήματα θα περιέχουν δισεκατομμύρια τρανζίστορ [81], συνθέτοντας δεκάδες έως εκατοντάδες συστήματα πυρήνων και ο αριθμός των πυρήνων που ενσωματώνονται σε ένα ενιαίο chip αναμένεται να αυξηθεί ραγδαία στα επόμενα χρόνια. Οι σύγχρονες ενσωματωμένες πλατφόρμες επωφελούνται από αυτή την πρόοδο της τεχνολογίας και κινούνται προς πολυπύρηνα συστήματα επεξεργασίας. Από βιομηχανικής άποψης, το όραμα των χιλίων πυρήνων σε ένα ολοκληρωμένο έρχεται όλο και πιο κοντά [24]. Επιπλέον, η ανάπτυξη τέτοιων πολυπύρηνων συστημάτων οδηγείται και από την ανάπτυξη εξαιρετικά απαιτητικών πολυνηματικών εφαρμογών.

Οι Δυναμικοί Διαχειριστές Μνήμης (ΔΔΜ), γνωστοί και ως διαχειριστές σωρού, είναι υπεύθυνοι για την οργάνωση των δυναμικών δεδομένων στη μνήμη και την εξυπηρέτηση των αιτημάτων της εφαρμογής, που αφορούν τη δυναμική διαχειρίση, κατά το χρόνο εκτέλεσης [91]. Οι δυναμικοί διαχείριστες μνήμης, οι οποίοι μπορούν να είναι υλοποιημένοι είτε σε λογισμικό (software) είτε σε υλικό (hardware), διαδραματίζουν σημαντικό ρόλο στην απόδοση των εφαρμογών και στην κατανάλωση ισχύος στην πλατφόρμα. Όπως απεικονίζεται στο [20], οι απλοί ΔΔΜ γενικού σκοπού αποτελούν συχνά εμπόδιο στην απόδοση και στην επεκτασιμότητα των πολυπύρηνων αρχιτεκτονικών επηρεάζοντας τη μνήμη και την κατανάλωση ενέργειας στο συνολικό σύστημα. Έτσι, προσαρμοσμένες λύσεις ΔΔΜ απαιτούνται κατά τη φάση σχεδιασμού των σύγχρονων συστημάτων τόσο για την βελτίωση του κατακερματισμού της μνήμης όσο και για την μείωση της καταναλισκόμενης ισχύος.

## 3.2 Περιγραφή της αρχιτεκτονικής

Το σύστημα που χρησιμοποιήθηκε για την ανάπτυξη και αξιολόγηση των ΔΔΜ σε επίπεδο middleware καθώς επίσης και των τεχνικών που αναπτύχθηκαν, αποτελείται από επεξεργαστικούς κόμβους (Prosecing Modules, PM) οι οποίοι διασυνδέονται μέσω ενός δικτύου σε ψηφίδα. (Σχήμα 3.1). Ένας επεξεργαστικός κόμβος αποτελείται από έναν LEON3 επεξεργαστή με τη δικιά του μνήμη εντολών (I-Cache) και δεδομένων (D-Cache), ένα διπύρηνο μικροπορργραμματίζομενο ελεγκτή (Dual microcoded Controller, DMC) και μνήμη η οποία μπορεί να μοιραστεί μεταξύ των κόμβων (shared memory).

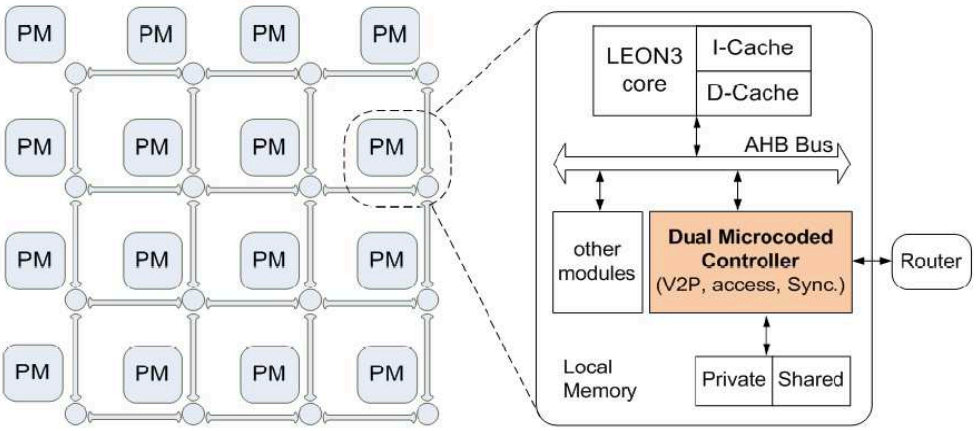
Η βασική μονάδα, στην οποία βασίσονται οι ανεπτυγμένες τεχνικές για ΔΔΜ σε επίπεδο midleware, είναι ο διπύρηνος μικροπορργραμματίζομενος ελεγκτής DMC, ο οποίος είναι σε θέση να εξυπηρετεί ταυτόχρονα αιτήματα τόσο από τον τοπικό πυρήνα όσο και απο το δίκτυο. Το Σχημα 3.2 δείχνει την δομή του DMC. Περισσότερες πληροφορίες σχετικά με τα τεχνικά χαρακτηριστικά του μπορούν να βρεθούν στο [33]. Η πλατφόρμα υποστηρίζει βασικές υπηρεσίες Κατανεμημένης Κοινής Μνήμης (Distributed Shared Memory, DSM), όπως:

- Εικονική-σε-φυσική μετάφραση διευθύνσεων (virtual-to-physical, V2P)
- Συγχρονισμό
- Συνεκτικότητα μνήμης cache
- Συνέπεια μνήμης
- Κοινόχρηστη πρόσβαση στην μνήμη

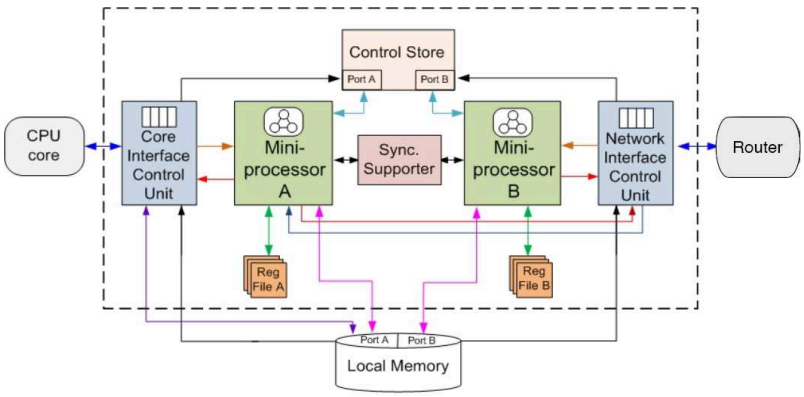
Οι υπηρεσίες αυτές είναι υλοποιημένες σε επίπεδο μικροκώδικα και εκτελούνται από τον DMC ανεξάρτητα απο την εφαρμογή υψηλού επιπέδου (C) που απασχολεί τον LEON3 επεξεργαστή. Με αυτό τον τρόπο, ο DMC ειναι υπεύθυνος για τον χειρισμό της μνήμης και όχι ο επεξεργαστής. *Η προσέγγιση αυτή προσφέρει το πλεονέκτημα της ευελιξίας του λογισμικού (λόγω της διαφάνειας των υπηρεσιών), εκμεταλλευόμενη και την απόδοση του επιταχυντή υλικού.*

*Η χρησιμοποιούμενη πλατφόρμα δεν υποστηρίζει λειτουργικό σύστημα.* Όμως, οι LEON3 επεξεργαστές είναι σε θέση να εκτελούν C εφαρμογές χρησιμοποιώντας τον Bare-C μεταγλωττιστή (BCC). Στη γλώσσα προγραμματισμού C, η δυναμική διαχείριση μνήμης δεν απαιτεί την παρουσία ενός λειτουργικού συστήματος. Αντ 'αυτού, οι συναρήσεις αυτές ορίζονται σε μία πρότυπη βιβλιοθήκη, η οποία μεταγλωττίζεται και συνδέεται με τις εφαρμογές του χρήστη.

Για να επιταχυνθεί η συχνή πρόσβαση στη μνήμη καθώς και για να διατηρηθεί ένας ενιαίος λογικός χώρος διευθύνσεων, η τοπική μνήμη κάθε κόμβου είναι χωρισμένη σε δύο μέρη: στο ιδιωτικό (private) και το κοινό (shared). Κατά συνέπεια, υπάρχουν και δύο τρόποι διευθυνσιοδότησης : φυσική και εικονική. *Ο τοπικός πυρήνας, χρησιμοποιώντας τη φυσική διευθυνσιοδότηση μπορεί να έχει πρόσβαση μόνο στην ιδιωτική*

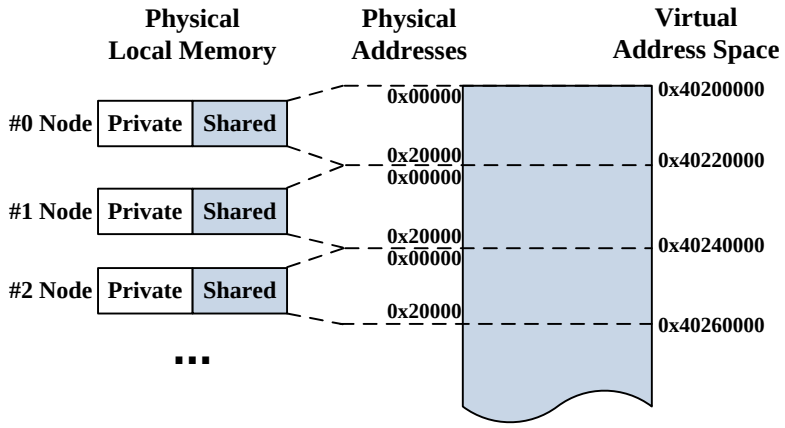



Σχήμα 3.1: Δίκτυο-σε-Ψηφιδα τύπου πλέγματος με 16 κόμβους [33]




| 90 nm synthesis results | | |
|---|---|---|
| | **Optimized for speed** | **Optimized for area** |
| **Frequency** | 481 MHz (2.08 ns) | 472 MHz (2.12 ns) |
| **Area** | 57K NAND gates | 50K NAND gates |

Σχήμα 3.2: Αρχιτεκτονική και αποτελέσματα σύνθεσης του DMC ελεγκτή [33]




Σχήμα 3.3: Η οργάνωση DSM και η V2P μετάφραση διευθύνσεων

*μνήμη. Όλες οι κοινές μνήμες είναι ορατές σε όλους τους κόμβους και οργανώνονται ως ένας ενιαίος εικονικός χώρος χρησιμοποιώντας την εικονική-προς-φυσική (V2P) μετάφραση διευθύνσεων.*

Το Σχήμα 3.3 δείχνει την DSM οργάνωση και την V2P μετάφραση διευθύνσεων. Στα αριστερά, βρίσκονται οι κόμβοι της πλατφόρμας καθένας από τους οποίους διαθέτει την ιδιωτική και την κοινή μνήμη τους. Οι φυσικές διευθύνσεις της κοινής μνήμης ξεκινούν απο την τιμή 0x00000 και τελειώνουν στην 0x20000. Σύμφωνα με την V2P μετάφραση διευθύνσεων, στο DSM περιβάλλον όλες οι κοινές μνήμες οργανώνονται ως ένας ενιαίος εικονικός χώρος. Η εφαρμογή χρησιμοποιεί τις εικονικές διευθύνσεις (0x40200000, κ.λπ.), προκειμένου να έχει πρόσβαση στην κοινή μνήμη. Η ενεργοποίηση των αντίστοιχων φυσικών κόμβων καθώς και η V2P μετάφραση διευθύνσεων γίνεται απο τον DMC ελεγκτή.

### 3.2.1 Συνεκτικότητα μνήμης cache και συνέπεια μνήμης

Η αποδοτική διατήρηση συνεκτικότητας της μνήμης cache σε πολυπύρηνα συστήματα είναι μία αναγνωρισμένη πρόκληση, ειδικά όταν η μνήμη cache και η τοπική μνήμες είναι κατανεμημένες. Για να είναι επεκτάσιμη, η πλατφόρμα υποστηρίζει συνεκτικότητα της μνήμης cache μέσω ενός είδους καταλόγου (directory-based) στο οποίο για κάθε διεύθυνση κοινής μνήμης υπάρχει δηλωμένη η κατάσταση (uncached, cached) και ένας συνδεόμενος κατάλογος ο οποίος καταγράφει τους εμπλεκόμενους κόμβους. Αντί για μια επίπεδη οργάνωση, οι κατάλογοι οργανώνεται ιεραρχικά με στόχο την ελαχιστοποίηση του μεγέθους της μονάδας και της επιβάρυνσης της επικοινωνίας του δικτύου. Όλες οι λειτουργίες συνεκτικότητας της μνήμης cache είναι υλοποιημένες σε επίπεδο μικροκώδικα, η ανάγνωση, η εγγραφή και η ακύρωση. Η υλοποίηση των υπηρεσιών συνεκτικότητας της μνήμης cache σε μικροκώδικα επιτρέπει την ευέλικτη υποστήριξη διαφορετικών πρωτόλλων με διαφορετικούς καταλόγους και πιθανά διαφορετικά ιεραρχικά επίπεδα, ανάλογα με το μέγεθος του συστήματος.

Για τη σωστή εκτέλεση των εφαρμογών, ένα μοντέλο συνέπειας μνήμης (memory consistency) λειτουργεί σαν συνδετικός κρίκος μεταξύ της αρχιτεκτονικής και της εφαρμογής. Λόγω των περιορισμών που επιβάλλονται από το σειριακό μοντέλο συνέπειας, ένα μοντέλο χαλαρής συνοχής είναι συχνά ευνοϊκότερο για υπολογιστικές αρχιτεκτονικές υψηλής απόδοσης. Η συγκεκριμένη πλατφόρμα υποστηρίζει δύο μοντέλα χαλαρής συνέπειας μνήμης (weak, release) τα οποία θέτουν σημεία ελέγχου συγχρονισμού πριν από την εκτέλεση του επόμενου τμήματος κώδικα. Ενώ το πρώτο μοντέλο (weak) χρησιμοποιεί ένα ενιαίο σημείο ελέγχου συγχρονισμού, το δεύτερο (release) διαφοροποιείται επιτρέποντας περισσότερα. Η πλατφόρμα υποστηρίζει και τα δύο μοντέλα χρησιμοποιώντας έναν μετρητή ανάλογα με την εκάστοτε προσέγγιση.

## 3.3 Επιτάχυνση διαχείρισης μνήμης σε επίπεδο middleware

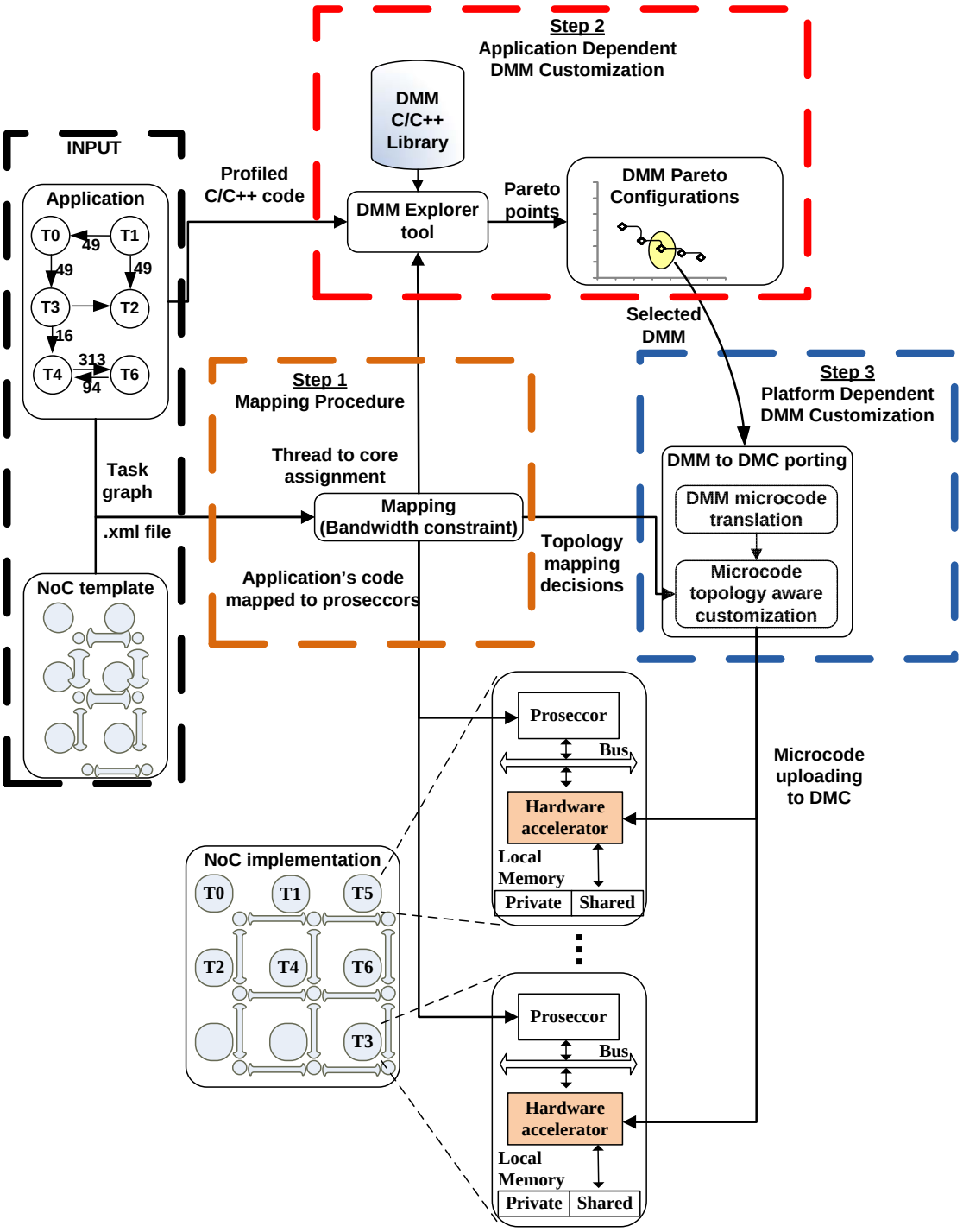
Αυτή η ενότητα εισάγει την έννοια της επιτάχυνσης των λειτουργιών δυναμικής διαχείρισης μνήμης στο επίπεδο middleware. Υιοθετούμε την προσέγγιση σε μικροκώδικα για την υποστήριξη προσαρμοσμένων ΔΔΜ σε πολυπύρηνες πλατφόρμες, με στόχο την αποδοτικότερη αξιοποίηση του υλικού, και παράλληλα διατήρηση της ευελιξίας των εφαρμογών λογισμικού. Συγκεκριμένα, θα αντιμετωπίσουμε το πρόβλημα της παροχής εξατομικευμένων ΔΔΜ σε μικροκώδικα για πολυπύρηνες πλατφόρμες τύπου δικτύου-σε-ψηφίδα (NoC) μέσα σε ένα περιβάλλον κατανεμημένης κοινής μνήμης (DSM). Για να εξασφαλιστεί η υψηλή απόδοση, οι προτεινόμενες υπηρεσίες ΔΔΜ αναπτύχθηκαν πάνω στον επιταχυντή υλικού DMC [33] ο οποίος α) λειτουργεί ανεξάρτητα απο την μηχανή δρομολόγησης, β) είναι υπεύθυνος για το χειρισμό αιτημάτων σε κατανεμημένη μνήμη και γ) μετριάζει το φόρτο εργασίας του επεξεργαστή. Για την περαιτέρω αξιοποίηση των DSM χαρακτηριστικών έχουμε αναπτύξει έναν ευέλικτο, κλιμακούμενο και κατανεμημένο ΔΔΜ, που ονομάζεται MAD-DMM (Microcode-Accelerated Dynamic Memory Manager). Ο MAD-DMM παρέχει κατανεμημένη λειτουργικότητα σε ένα DSM περιβάλλον , διατηρώντας παράλληλα τις προγραμματιστικές διεπαφές C-API (`malloc()/free()`). Τέλος, παρουσιάζουμε έναν αποτελεσματικό τρόπο για την ενσωμάτωση DVFS μηχανισμών σε οποιαδήποτε υψηλού επιπέδου ΔΔΜ χωρίς αλλαγή των εφαρμογών.

### 3.3.1 Προσαρμοσμένη δυναμική διαχείριση μνήμης σε μικροκώδικα

Το προτεινόμενο μεθοδολογικό πλαίσιο για την υποστήριξη προσαρμοσμένων ΔΔΜ σε μικροκώδικα για πολυπύρηνες κατανεμηνές πλατφόρμες παρουσιάζεται στο Σχήμα 3.4 [11].

Παίρνοντας τον πηγαίο κώδικα μιας πολυνηματικής δυναμικής εφαρμογής και ένα πρότυπο χαρακτηρισμού πολυπύρηνης αρχιτεκτονικής, εκτελούμε τα ακόλουθα τρία βήματα:

1. Απεικόνιση (mapping) της εφαρμογής στους πυρήνες της πλατφόρμας.




Σχήμα 3.4: Μεθοδολογικό πλαίσιο για την υποστήριξη προσαρμοσμένων ΔΔΜ σε μικροκώδικα για πολυπύρηνες κατανεμημένες πλατφόρμες [11]

2. Προσαρμογή των ΔΔΜ με βάση την εφαρμογή.

3. Παραμετροποίηση των ΔΔΜ με βάση τα χαρακτηριστικά της πλατφόρμας.

#### 3.3.1.1 Απεικόνιση της εφαρμογής στους πυρήνες της πλατφόρμας

Οι αποφάσεις απεικόνισης της εφαρμογής έχουν μεγάλο αντίκτυπο στην απόδοση και στην κατανάλωση ενέργειας του συστήματος και γι'αυτό το λόγο οι απαραίτητες αποφάσεις πρέπει να λαμβάνονται στο στάδιο του σχεδιασμού. Η εφαρμογή αποτελείται από έναν αριθμό εργασιών (οι κόμβοι στον γράφο της εφαρμογής). Οι ακμές μεταξύ των εργασιών χαρακτηρίζουν το κόστος επικοινωνίας και τις εξαρτήσεις των δεδομένων. Έχουμε διατυπώσει το πρόβλημα απεικόνισης (mapping) ως ένα-προς-ένα αντιστοίχιση μεταξύ δύο γράφων. Ο πρώτος γράφος αντιπροσωπεύει την εφαρμογή, ενώ ο δεύτερος την πολυπύρηνη πλατφόρμα.

- **Γράφος της εφαρμογής**: Ορίζουμε ως γράφο αναπάραστασης της εφαρμογής έναν κατευθυνόμενο γράφο $AppG(V, E)$, όπου η κάθε κορυφή $v_i \in V$ αντιπροσωπεύει μια εργασία και οι κατευθυνόμες ακμές $e_{i,j} \in E$ αντιπροσωπεύουν την επικοινωνία μεταξύ των εργασιών $v_i$ και $v_j$. Το βάρος των ακμών $e_{i,j}$ συμβολίζεται σαν $w_{i,j}$, και αντιπροσωπεύει το φορτίο της επικοινωνίας απο $v_i$ στον $v_j$.

- **Γράφος της πλατφόρμας**: Ορίζουμε ως γράφο αναπάραστασης της πλατφόρμας έναν κατευθυνόμενο γράφο $PlatformG(N, L)$, όπου η κάθε κορυφή $n_i \in N$ αντιπροσωπεύει έναν κόμβο της αρχιτεκτονικής. Ορίζουμε σαν $N_{Proc} \in N$ τους υπολογιστικούς κόμβους της πολυπύρηνης αρχιτεκτονικής και σαν $N_{Mem} \in N$ τους κόμβους που αποτελούν μνήμη. Οι κατευθυνόμενες ακμές $l_{i,j} \in L$ αντιπροσωπεύουν την άμεση επικοινωνία μεταξύ των κόμβων $n_i$ and $n_j$. Το διαθέσιμο έυρος ζώνης μεταξύ αυτών των κόμβων περιγράφεται απο το $bw_{i,j}$.

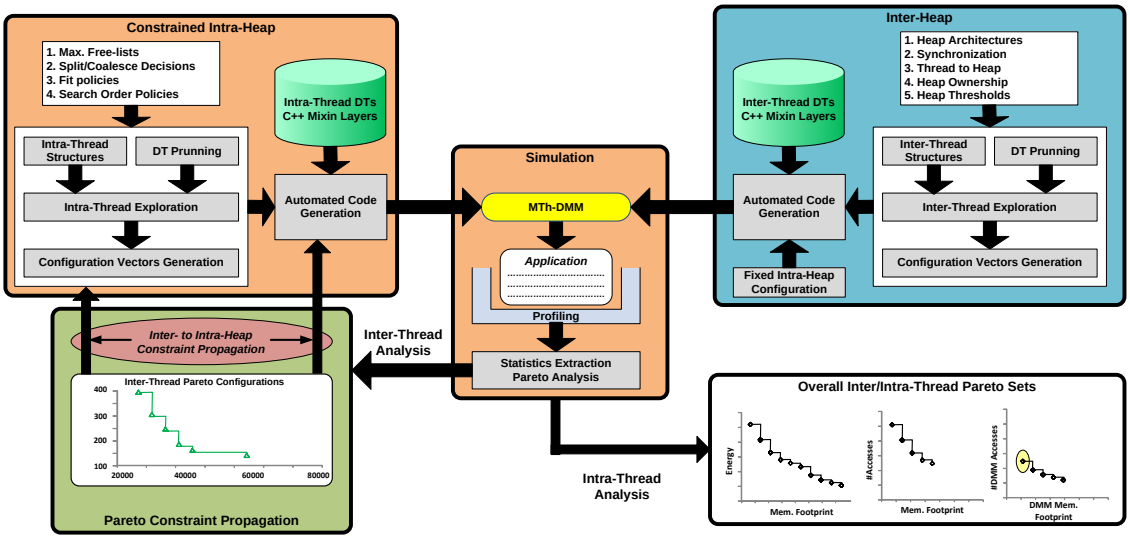
Στόχος της απεικόνισης (mapping) είναι η ελαχιστοποίηση του συνολικού κόστους επικοινωνίας ($C_{total}$) πάνω στην πολυπύρηνη πλατφόρμα, το οποίο υπολογίζεται απο την Εξίσωση 3.1:

$$C_{total} = \sum_i \sum_j D_{i,j} \times W_{i,j} \tag{3.1}$$

όπου $D_{i,j}$ έναι η απόσταση (μετρούμενη σε βήματα) μεταξύ των $i$ και $j$ κόμβων του γράφου της αρχιτεκτονικής, ενώ $W_{i,j}$ είναι το βάρος της επικοινωνίας. Για την απεικόνιση των εφαρμογών, χρησιμοποιούμε τον αλγόριθμο που στοχεύει στην αξιοποίηση του διαθέσιμου εύρους ζώνης και παρουσιάζεται στο [64].

#### 3.3.1.2 Προσαρμογή των ΔΔΜ με βάση την εφαρμογή

Σε αυτό το βήμα, δημιουργούμε ένα Pareto σύνολο πολυνηματικών ΔΔΜ, προσαρμοσμένων στους περιορισμούς και στις ανάγκες της εφαρμογής. Χρησιμοποιήθηκε το




Σχήμα 3.5: Το εργαλείο MTh-DMM Explorer [92]

εργαλείο MTH-DMM Explorer [92] το οποίο εξερευνεί τις λύσεις με βάση δέντρα απόφασης όπως απεικονίζεται στο Σχήμα 3.5.

Το εργαλείο MTh-DMM Explorer αναζητά να βρει τους βέλτιστους συνδυασμούς κατα τη φάση σχεδίασης για τη δυναμική εφαρμογή. Η διαδικασία εξερεύνησης αναλύει αποτελεσματικά το χώρο λύσεων με την αξιοποίηση των εξαρτήσεων μέσα στο σωρό. Μετά τον προσδιορισμό των παραμέτρων εξερεύνησης, αρχικοποιούνται οι δομές που αντιπροσωπεύουν τα αντίτοιχα δέντρα αποφάσεων και παράγονται οι αντίστοιχοι ΔΔΜ. Στη συνέχεια, ακολουθεί μια Pareto ανάλυση για να εξαχθούν οι αποδοτικότεροι ΔΔΜ. Κάθε Pareto λύση δίνεται σαν είσοδο για περαιτέρω εξερεύνηση με βάση τους περιορισμούς και τις αλληλεξαρτήσεις σε επίπεδο σωρού αυτή τη φορά αντί σε επίπεδο νημάτων. Οι παράμετροι για την εξαγωγή των Pareto ΔΜΜ είναι οι εξής: α) το αποτύπωμα μνήμης (memory footprint) και β) ο αριθμός των προσβάσεων.

#### 3.3.1.3 Παραμετροποίηση των ΔΔΜ με βάση τα χαρακτηριστικά της πλατφόρμας

Ο κύριος στόχος του τρίτου βήματος είναι η προσαρμογή των ΔΔΜ με βάση και τα χαρακτηριστικά της πλατφόρμας, με στόχο την αύξηση της απόδοση και την αξιοποίηση των χαρακτηριστικών της πλατφόρμας. Αυτό γίνεται μέσα από την μεταφορά του προσαρμοσμένου κατά την εφαρμογή ΔΔΜ στον επιταχυντή υλικού DMC. Απαιτούνται δύο βήματα. Στο πρώτο, οι υλοποιήσεις των ΔΔΜ σε C/C++ μεταφράζονται σε μικροκώδικα πάνω στον DMC. Στο δευτερο βήμα, ο μικροκώδικας επεκτείνεται ώστε να ληφθούν υπόψη τοπολογικά χαρακτηριστικά , όπως η κατανομή της μνήμης και το κόστος επικοινωνίας.

**3.3.1.3.1 Μετάφραση των ΔΔΜ σε μικροκώδικα** Έχοντας ως είσοδο τους C/C++ ΔΔΜ, μεταμορφώνουμε τον κώδικα υψηλού επιπέδου σε ισοδύναμες λειτουργίες μικροκώδικα. Έχουμε κατασκευάσει γενικά πρότυπα μικροκώδικα (microcode templates) με βάση τις αντίστοιχες C/C++ δυναμικές δομές δεδομένων ώστε να εκμεταλλευτούν τα χαρακτηριστικά της πλατφόρμας. Οι λειτουργίες που παράγονται σε μικροκώδικα είναι πλήρως παραμετροποιήσιμες. Το Σχήμα 3.6 δείχνει ένα παράδειγμα της μετάφρασης μικροκώδικα για τον First Fit [91] αλγόριθμο. Πιο συγκεκριμένα, σύμφωνα με τα χαρακτηριστικά της πλατφόρμας ο σχεδιαστής μπορεί να οργανώσει τον σωρό με πολλούς τρόπους επιλέγοντας κάθε φορά διαφορετικά χαρακτηριστικά στους ΔΔΜ:

- **Αριθμός και τύπος σταθερών λιστών (Fixed Lists)**: Αυτό ο τύπος σωρού εξυπηρετεί αιτήσεις σταθερού μεγέθους με γρήγορο τρόπο. Μπορούν να θεωρηθούν ως μηχανισμοί γρήγορης διαχείρισης δυναμικών δεδομένων που έχουν μεγάλη επίδραση στην απόδοση των ΔΔΜ και στον κατακερματισμό της μνήμης.

- **Μέγεθος σωρού**: Ο μέγιστος χώρος μνήμης που ο ΔΔΜ μπορεί να χρησιμοποιήσει για διαχείριση των δεδομένων. Σε περίπτωση που το μέγεθος δεν είναι επαρκές, απαιτείται πρόσθετη μνήμη από το σύστημα διαφορετικά όλα τα αιτήματα για παραχώρηση δυναμικής μνήμης θα αποτύχουν.

- **Τοποθεσία σωρού**: Αναφέρεται στην απεικόνιση της συνολικής οργάνωσης του σωρού πάνω στην κατανεμημένη μνήμη της πλατφόρμας. Μπορούμε να δημιουργήσουμε διάφορες οργανώσεις σωρού σε μικροκώδικα οι οποίες μπορούν να απεικονιστούν πάνω στην πολυπύρηνη πλατφόρμα. Για παράδειγμα, μπορούμε να χαρακτηρίσουμε σωρούς ανάλογα με τη θέση τους, είτε ως τοπικούς (σωροί που βρίσκονται σε επεξεργαστικούς κόμβους) είτε ως γενικούς (σωροί που βρίσκονται σε κόμβους μνήμης). Το μέγεθος του σωρού είναι στενά συνδεδεμένο με την απεικόνισή λόγω των διαφορετικών μεγεθών των μνημών στα πολυπύρηνα συστήματα.

*Με το φόρτωμα στη μνήμη εντολών του DMC των ΔΔΜ σε μικροκώδικα, οι επεξεργαστές της πλατφορμάς σταματούν να είναι υπεύθυνοι για την εξυπηρέτηση αιτημάτων σχετικά με τη διαχείριση δυναμικών δεδομένων ή για την παρακολούθηση της κατάστασης του σωρού ελαφρύνοντας έτσι το φορτίο τους.*

**3.3.1.3.2 Προσαρμογή σύμφωνα με την κατανομή της μνήμης** Η διαχείριση πρόσβασης στη φυσική μνήμη είναι κυρίαρχο πρόβλημα σε πολυπύρηνες αρχιτεκτονικές λόγω της κατανομής των μνημών πάνω στην πλατφόρμα. Για μια δεδομένη κατανομή μνήμης, αυξάνουμε περαιτέρω τις επιδόσεις των επιλεγμένων ΔΔΜ με την προσθήκη λειτουργιών σε μικροκώδικα αναθέτοντας στον επιταχυντή υλικού DMC να εξετάσει: i) ποιός έιναι ο κατάλληλος τοπικός (Local) σωρός για να εξυπηρετήσει ένα (απομακρυσμένο) αίτημα διαχείρισης μνήμης και ii), ποιός γενικός (Global) σωρός είναι πιο κοντά.

Κατά τη φάση σχεδίασης με βάση *τοπολογικά κριτήρια* δημιουργούμε πίνακες προτεραιοτήτων $PT_{s,d}$, $(s, d \in N)$ για κάθε κόμβο $N$ της πολυπύρηνης πλατφόρμας.

**C/C++ code**

```
inline void * first_fit_free_list (size_t sz) {
      // Check the free list first.
      freeObject * prev = &head;
      while (prev->next != &tail) {
        if (Header::getSize(prev->next) >= sz) {
          freeObject * ptr = prev->next;
          prev->next = ptr->next;
          if (prev->next == &tail) {
            tail.next = prev;
          }
          return (void *) ptr;
        }
        prev = prev->next;
      }
      return 0
}
```

**C/C++ to DMC Microcode**

```
FIRSTFIT:
set A1 {N_FREES}            ; If there are no empty blocks perform an INSERT
set A6 {G_HEAP_START}
beqz A1 INSERT
LOOP:
set A4 0       ; Check the indexes of the single linked list,
               ;to check where is the       empty block
add A6 A6 4
lw *A6 A2
lrs A2 A4 1
beqz A4 FOUND_EMPTY_BLOCK    ; Empty block found?
add A6 A6 A2
jmp LOOP
FOUND_EMPTY_BLOCK:    ;As soon anempty block is found we check
               ;whether we can accommodate the allocation request

set A5 DATA
bleq A5 A2 ADD ;If so, INSERT (place) the element to that address
sub A1 A1 1
beqz A1 ADD           ;If not, check other free blocks (if any), else INSERT
                      ;as a new element to the end of the Allocated List
Add A6 A6 5
add A6 A6 A2
jmp LOOP
```

Σχήμα 3.6: Μετάφραση πηγαιου κώδικα. Απο C/C++ σε μιρκοκώδικα. Ο αλγόριθμος First Fit.

Το $P \in N$ αντιπροσωπεύει τους επεξεργαστικούς κόμβους και το $M \in N$ αντιπροσωπεύει τους κόμβους μνήμης. Το $PT_{s,d}$ περιγράφει το βάρος της προτεραιότητας του κόμβου $s$ ώστε να έχει πρόσβαση στον προορισμό $d$. Οι προτεραιότητες $PT_{s,d}$ χρησιμοποιούνται απο τον ελεγκτή DMC κατά τη φάση εκτέλεσης ώστε να γίνει πρόσβαση στους κόμβους ανάλογα με την τιμή των προτεραιοτήτων στον πίνακα $PT_{s,d}$. Οι τιμές του πίνακα προτεραιοτήτων $PT_{s,d}$ εξάγονται απο την Εξίσωση 3.2.

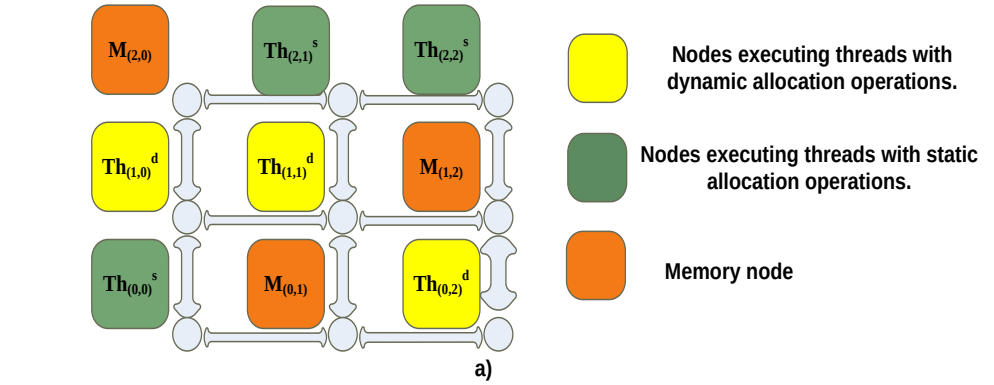
$$PT_{s,d} = \frac{w_1 P_{s,d} + (1 - w_1)(w_2 ML_d + (1 - w_2)(w_3 MP_d + (1 - w_3 D_{s,d})))}{\sum\limits_{\forall i}^{(i \neq d)} \{w_1 Ps.i + (1 - w_1)(w_2 ML_i + (1 - w_2)(w_3 MP_i + (1 - w_3 D_{s,i})))\}} \quad (3.2)$$

όπου $i \in M$, $P_{s,d}$ και $D_{s,d}$ είναι η κατανάλωση ισχύος και η καθυστέρηση της $(s, d)$ σύνδεσης αντίστοιχα. Ως $ML_d$ και $MP_d$ ορίζουμε την καθυστέρηση της μνήμης και την κατανάλωση ισχύος της μνήμης $d$ ανά πρόσβαση. Επίσης, $\sum\limits_{i=1}^{3} w_i = 1, w_i \geq 0$ είναι τα βάρη για την συνάρτηση κόστους.

Ο DMC χρησιμοποιεί την πολιτική ανταλλαγής μηνυμάτων (message passing) για την μετάδοση των πληροφοριών στους γειτονικούς κόμβους. Με τον τρόπο αυτό, η εκτέλεση μικροκώδικα σε διαφορετικό κόμβο επιτρέπεται ακόμη και αν ο απομακρυσμένος DMC δεν έχει λάβει κανένα σήμα από τον τοπικό πυρήνα του. Αυτό που κυρίως επηρεάζει την επεκτασιμότητα του ΔΔΜ είναι: α) η απεικόνιση μεταξύ του σωρού και των νημάτων και β) η απεικόνιση μεαξύ του σωρού και των μνημών.

Το Σχημα 3.7 παρουσιάζει ένα εικονογραφημένο παράδειγμα παραμετροποίησης σύμφωνα με την κατανομή της μνήμης. Υποθέτουμε μία $3 \times 3$ NoC αρχιτεκτονική η οποία αποτελείται απο τρείς κόμβους μνήμης ($M_{(0,1)}$, $M_{(1,2)}$ και $M_{(2,0)}$). Υπάρχουν τρείς επεξεργαστικοί κόμβοι οι οποίοι εκτελούν νήματα εφαρμογών ($\text{Th}^d_{(0,2)}$, $\text{Th}^d_{(1,0)}$, $\text{Th}^d_{(1,1)}$) με δυναμικά δεδομένα και οι υπόλοιποι εκτελούν το στατικό κομμάτι της εφαρμογής ($\text{Th}^s_{(0,0)}$, $\text{Th}^s_{(2,1)}$, $\text{Th}^s_{(2,2}$). Όλοι οι επεξεργαστικοί κόμβοι διαθέτουν τη δικιά τους τοπική μνήμη (Local Memory, LM). Σύμφωνα με την Εξίσωση 3.2 κατασκευάζουμε τους πίνακες προτεραιοτήτων όπως φαίνονται στο Σχημα 3.7(b) (0 = υψηλότερη προτεραιότητα, 8 = χαμηλότερη προτεραιότητα).

Σύμφωνα με τον πίνακα προτεραιοτήτων (Σχήμα 3.7(b)) για κάθε $\text{Th}^d_{(i,j)}$ κατσκευάζουμε μια απλά συνδεδεμένη λιστα (Single Linked List, SLL) η οποία περιέχει πληροφορίες σε μικροκώδικα για τον επόμενο κόμβο που θα πρέπει να ερωτηθεί για διαθεσιμότητα του σωρού. Η δομή της απλά συνδεδεμένης λίστας παρουσιάζεται στο Σχήμα 3.7(c). Έχουμε επιλέξει την πολιτική ανταλλαγής μηνυμάτων (message passing) για την διάδοση των πληροφοριών στους γειτονικούς κόμβους. Τα πρότυπα σε μικροκώδικα, υπεύθυνα για την ενεργοποίηση των απομακρυσμένων κόμβων, παρουσιάζονται στο Σχήμα 3.7(d).




a)

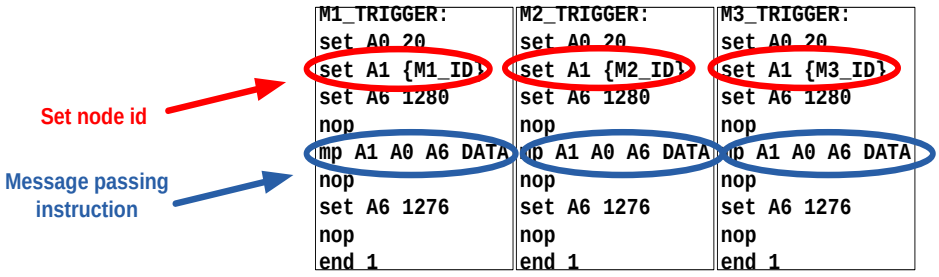
Thread to Memory Priority Table

| | $M_{(2,0)}$ | $M_{(1,2)}$ | $M_{(0,1)}$ | $LM_{(0,0)}$ | $LM_{(0,2)}$ | $LM_{(1,0)}$ | $LM_{(1,1)}$ | $LM_{(2,1)}$ | $LM_{(2,2)}$ |
|---|---|---|---|---|---|---|---|---|---|
| $Th_{(1,0)}^{d}$ | 1 | 5 | 4 | 2 | 8 | 0 | 3 | 6 | 7 |
| $Th_{(0,2)}^{d}$ | 8 | 2 | 1 | 5 | 0 | 7 | 3 | 6 | 4 |
| $Th_{(1,1)}^{d}$ | 5 | 2 | 1 | 7 | 8 | 4 | 0 | 3 | 6 |

0: Highest priority, 8: Lowest priority

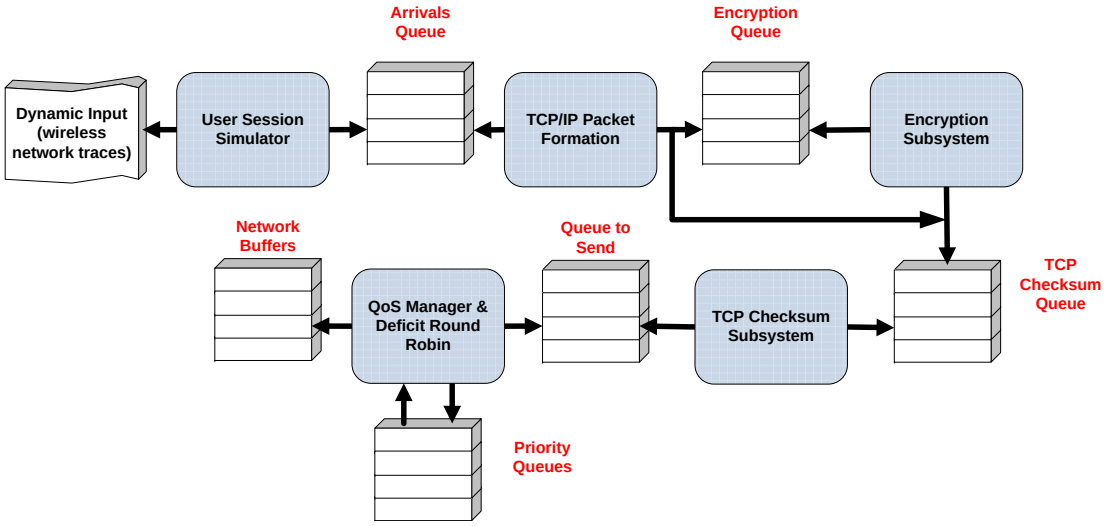
b)


c)




d)

Σχήμα 3.7: Παράδειγμα προσαρμογής ΔΔΜ με βάση την κατανομή της μνήμης σε πολυπύρηνη αρχιτεκτονική: a)Τοπολογία και απεικόνιση πυρήνων, b) Πίνακας προτεραιοτητων, c) Δομή απλά συνδεδεμένης λιστας, d) Πρότυπα συναρτήσεων επικοινωνίας σε μικροκώδικα.




Σχήμα 3.8: Η πολυνηματική εφαρμογή που χρησιμοποιείται. Τα τετράγωνα είναι τα διαφορετικά νήματα εκτέλεσης τα οποία επικοινωνούν μεταξύ του με ασύγχρονες ουρές FIFO [17].

#### 3.3.1.4 Αξιολόγηση των ειδικά προσαρμοσμένων ΔΔΜ

Για να αξιολογήσουμε την προτεινόμενη μεθοδολογία, μοντελοποιήσαμε τα ακόλουθα νήματα εκτέλεσης σε μια ενιαία εφαρμογή, της οποίας η είσοδος είναι δικτυακά ίχνη (network traces) ασύρματων δικτύων [17]. Η εφαρμογή παρουσιάζεται στο Σχήμα 3.8. Κάθε πυρήνας της εφαρμογής εκτελείται στο δικό του ανεξάρτητο νήμα εκτέλεσης και επικοινωνεί ασύγχρονα με τα υπόλοιπα νήματα. Το μέσο επικοινωνίας είναι ασύγχρονες ουρές επικοινωνίας, οι οποίες έχουν ενσωματωμένους μηχανισμούς που επιτρέπουν το συγχρονισμό μεταξύ των νημάτων. Καθώς το σύστημα είναι πολυνηματικό οι προσβάσεις στη μνήμη δεν συμβαίνουν με ένα σειριακό τρόπο. Πολλά πακέτα είναι ζωντανά κατά τη διάρκεια εκτέλεσης της εφαρμογής έτσι οι προσβάσεις στη μνήμη έχουν σαν προέλευση διαφορετικά νήματα.

- Δικτυακή κίνηση η οποία αντιστοιχεί σε δραστηριότητες τύπου VoIP, FTP και πλοήγησης στο διαδίκτυο, και οι οποίες έχουν μετρηθεί στην εργασία [44]. Αυτό το νήμα εκτέλεσης τροφοδοτεί το όλο σύστημα με κάθε πακέτο δεδομένων που βρίσκεται μέσα στο αρχείο που περιέχει την δικτυακή κίνηση. Η διαθέσιμη πληροφορία για κάθε πακέτο είναι: χρόνος, διευθύνσεις IP της προέλευσης και του προορισμού, οι θύρες προέλευσης και προορισμού καθώς και το μέγεθος του πακέτου. Αυτό το νήμα δεσμεύει την απαραίτητη μνήμη για τα δεδομένα του πακέτου (χωρίς να περιλαμβάνει την πληροφορία που υπάρχει στην επικεφαλίδα).

- Δημιουργία ενός TCP/IP πακέτου. Αυτό το νήμα είναι υπεύθυνο για τη συναρμολόγηση ενός πλήρους πακέτου TCP/IP συμπληρώνοντας τη πληροφορία που χρειάζεται στην επικεφαλίδα του πακέτου. Μπορούμε να παρομοιάσουμε τη λειτουργία αυτού του νήματος με την κλήση της `write()` (μια κλήσης του συ-

Πίνακας 3.1: Περιγραφή των διαμορφώσεων των ειδικά προσαρμοσμένων ΔΔΜ

| | | Code size (# microcode instructions) | | | |
|---|---|---|---|---|---|
| DMM | Description | Conf. 1 | Conf. 2 | Conf. 3 | Conf. 4 |
| DMM 1 | $\text{FixList}_0(block = 40B)$<br>$\text{FixList}_1(block = 1460B)$<br>$\text{FixList}_2(block = 1500B)$<br>Generic heap | 1407 | 485 | 1736 | 1859 |
| DMM 2 | $\text{FixList}_0(block \in [0B, 40B])$<br>$\text{FixList}_1(block \in [1280B, 1460B])$<br>$\text{FixList}_2(block \in (1460B, 1500B])$<br>$\text{FixList}_3(block = 92B)$<br>Generic heap | 1467 | 485 | 1796 | 1919 |

στήματος), και με αυτόν τον τρόπο κτίζεται το ολόκληρο το πακέτο. Το συνολικό μέγεθος του πακέτου αυξάνεται κατά 40 Byte (όσο είναι το μέγεθος της επικεφαλίδας). Το ολοκληρωμένο πλέον πακέτο εγγράφεται στις αντίστοιχες ουρές προς κρυπτογράφηση ή προς έλεγχο λαθών (TCP checksum) ανάλογα με το αν η σύνδεση είναι κρυπτογραφημένη ή όχι.

- Κρυπτογράφηση (τα πακέτα που είναι μέρος μια κρυπτογραφημένης σύνδεσης κρυπτογραφούνται σύμφωνα με τον αλγόριθμο DES). Το νήμα αυτό διαβάζει τα δεδομένα (payload) του πακέτου σε μπλοκ μεγέθους 8 Byte, και μετά το τέλος της κρυπτογράφησης μεταφέρει το πακέτο στην ουρά όπου τα πακέτα έτοιμα για έλεγχο λαθών περιμένουν.

- Δημιουργία του αθροίσματος TCP checksum. Το συγκεκριμένο νήμα υπολογίζει το άθροισμα εφαρμόζοντας τη διαδικασία που περιγράφεται στο [51]. Τα περιεχόμενα του πακέτου διαβάζονται ανά 16 bit και σε αυτά εφαρμόζεται η αντίστοιχη διαδικασία. Με το που δημιουργηθεί το άθροισμα αυτό γράφεται στο πεδίο CRC της επικεφαλίδας του πακέτου, και στη συνέχεια το πακέτο οδηγείται προς την επόμενη ουρά και το επόμενο νήμα εκτέλεσης.

- Ο διαχειριστής ποιότητας παροχής υπηρεσιών (QoS manager) χτίζει μια λίστα με τους προορισμούς των διάφορων πακέτων και χρησιμοποιεί προτεραιότητες για τη διαχείρισή της. Όταν ένα πακέτο εισέρχεται στο σύστημα τοποθετείται σε μια από τις διαφορετικές ουρές με βάση την προτεραιότητά του. Τα πακέτα εξάγονται από αυτές τις ουρές και προωθούνται στην δικτυακή έξοδο σύμφωνα με έναν αλγόριθμο Deficit Round Robin (DRR). Όταν ένα πακέτο προωθείται στην έξοδο τότε μειώνεται το βάρος της συγκεκριμένης ουράς σύμφωνα με το μέγεθος του πακέτου.

Βασισμένοι στην συμπεριφορά της εφαρμογής και με τη χρήση του εργαλείου MTh-DMM Explorer, δημιουργήσαμε ένα Pareto σύνολο απο προσαρμοσμένους ΔΔΜ. Ο Πίνακας 3.1 δείχνει τις επιλεγμένες διαμορφώσεις των ειδικά προσαρμοσμένων ΔΔΜ:

Η τοπολογία που χρησιμοποιήθηκε για την αξιολόγηση των ΔΔΜ σε μικροκώδικα παρουσιάζεται στο Σχήμα 3.9. Σύμφωνα με τις αποφάσεις απεικόνισης, οι κόμβοι $(0,0),(0,1),(1,0)$ είναι επεξεργαστικοί κόμβοι με την δικιά τους τοπική μνήμη. Πιο

συγκεκριμένα οι κόμβοι $(0,1),(1,0)$ εκτελούν απο 2 νήματα ο καθένας, ένα που ασχολείται με δυναμικά δεδομένα και ενα μόνο με στατικά.Ο κόμβος $(0,0)$ εκτελεί νήματα με μόνο δυναμικά δεδομένα. Ο κόμβος $(1,1)$ ειναι κόμβος μνήμης ο οποίος εξυπηρετεί όλα τα αιτήματα που δεν ήταν δυνατόν αν εξυπηρετηθούν απο τις τοπικές μνήμες. Για τις τοπικές μνήμες, το μέγεθος του σωρού είναι $4KB$ ενώ για τον κόμβο μνήμης, το μέγεθος του σωρού είναι $32KB$.

Για την επιλεγμένη τοπολογία (Σχήμα 3.9a) υλοποιήσαμε τέσσερις διαμορφώσεις με βάση την κατανομή της μνήμης πάνω στην πλατφόρμα. Η υλοποίηση των ΔΔΜ παρουσιάζεται στο Σχημα 3.9b-e. Οι κατευθυνόμενες ακμές επισημαίνουν ότι είναι δυνατή η εξυπηρέτηση ενός αιτήματος δυναμικής διαχείρισης μνήμης ενώ τα βάρη, $PT_{s,d}$, δείχνουν την προτεραιότητα για την επιλογή του κόμβου (0 = υψηλότερη προτεραιότητα, 3 = χαμηλότερη προτεραιότητα).

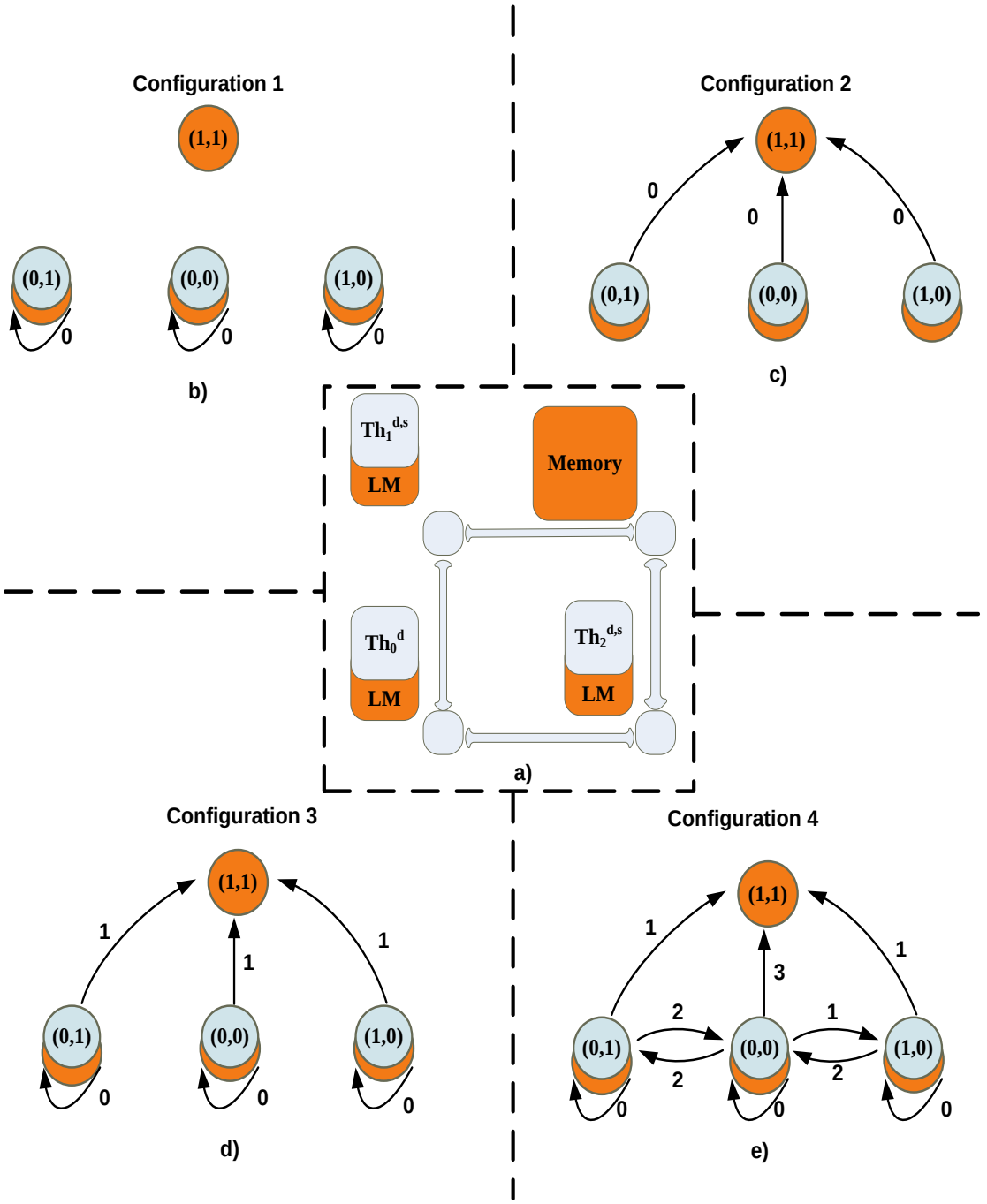
- **Διαμόρφωση 1**: Πλήρως κατανεμημένη μνήμη. Στην πλήρως κατανεμημένη μνήμη (Σχήμα 3.9b), κάθε κόμβος στέλνει τα αιτηματά του μόνο στην τοπική μνήμη (LM).

- **Διαμόρφωση 2**: Κεντρικοποιημένος μοναδικός σωρός. Στον κεντρικοποιημένο μοναδικό σωρό (Σχήμα 3.9c), κάθε κόμβος στέλνει τα αιτήματά του μόνο στον γενικο σωρο $(1,1)$. Δεν υπάρχουν τοπικοί σωροι.

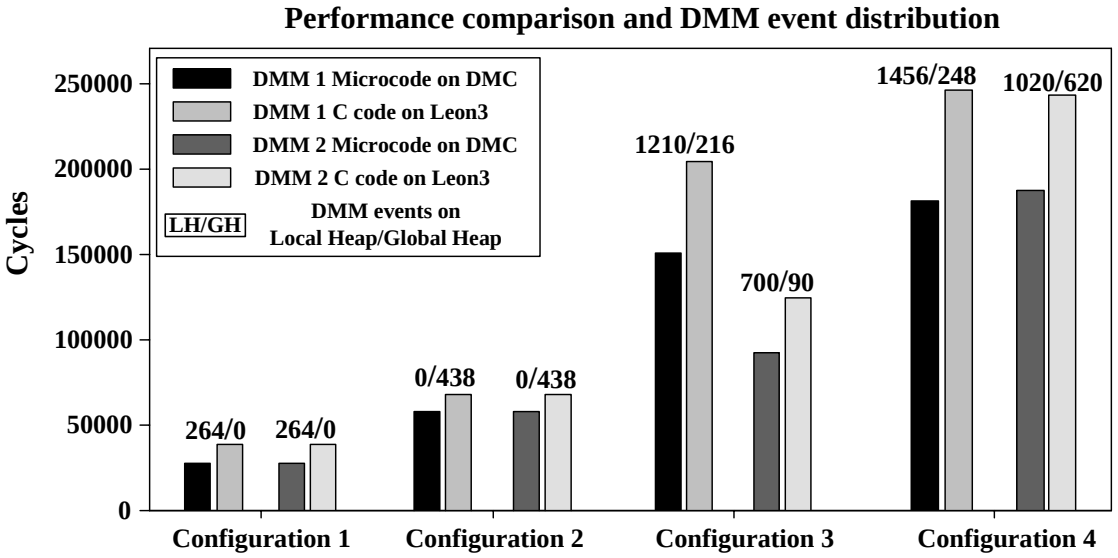
- **Διαμόρφωση 3**: Κατανεμημένοι πολλαπλοί σωροί με επιπλεόν γενικό σωρό. Στην διαμόρφωση κατανεμημένοι πολλαπλοί σωροι με επιπλεόν γενικό σωρό (Σχήμα 3.9d) κάθε κόμβος στέλνει τα αιτήματά του αρχικά στον τοπικό σωρό. Αν δεν είναι η δυνατη η εξυπηρέτηση αυτών των αιτημάτων τοπικά, τότε στέλνει τα αιτήματά του στον γενικό σωρό $(1,1)$

- **Διαμόρφωση 4**: Κατανεμημένοι πολλαπλοί σωροι με βάση την τοπικότητα της μνήμης και ένας επιπλεόν γενικός σωρός. Στην διαμόρφωση Κατανεμημένοι πολλαπλοί σωροι με βάση την τοπικότητα της μνήμης και ένας επιπλεόν γενικός σωρός (Σχήμα 3.9d) κάθε κόμβος στέλνει τα αιτήματά του αρχικά στον τοπικό σωρό. Αν δεν είναι η δυνατή η εξυπηρέτηση αυτών των αιτημάτων τοπικά, τότε σύμφωνα *με τον πίνακα προτεραιοτήτων* δοκιμάζει είτε τον γενικό σωρο $(1,1)$ είτε τους γειτονικούς του τοπικούς σωρούς.

Για τους ΔΔΜ που επιλέχθηκαν (Πίνακας 3.1) και για κάθε διαμόρφωση το Σχήμα 3.10 παρουσιάζει: (i) τους κύκλους μέχρι να υπάρξει αδυναμία του σωρού να εξυπηρετήσει άλλα αιτήματα, (ii) την κατανομή των αιτημάτων δυναμικής διαχειρισης μνήμης και (iii) την απόδοση των ΔΔΜ σε μικροκώδικα σε αντιστοιχία με τις `C` στον LEON3 επεξεργαστή. Πάνω απο κάθε μπάρα παρουσιάζεται ο αριθμός των εξυπηρετηθέντων αιτημάτων ***(Τοπικός Σωρός/ Γενικός Σωρός)***. Σύμφωνα με το Σχημα 3.10, όταν οι ΔΔΜ είναι ενήμεροι για την κατανομή της μνήμης πάνω στην πολυπύρηνη πλατφόρμα, τότε αυξάνεται ο αριθμός των εξυπηρετηθέντων αιτημάτων. Η Διαμόρφωση 4 εξυπηρέτησε $7\times$ περισσότερα αιτήματα απο την αντίστοιχη Διαμόρφωση 1. Επιπλέον, τα αιτήματα δυναμικής διαχείρισης μνήμης που εξυπηρετήθηκαν από τον επιταχυντή υλικού DMC είναι κατά μέρο όρο 25% γρηγορότερα απο τις αντίστοιχες `C` υλοποιήσεις. Αυτό συμβαίνει επειδή ο DMC είναι υπεύθυνος για το χειρισμό των



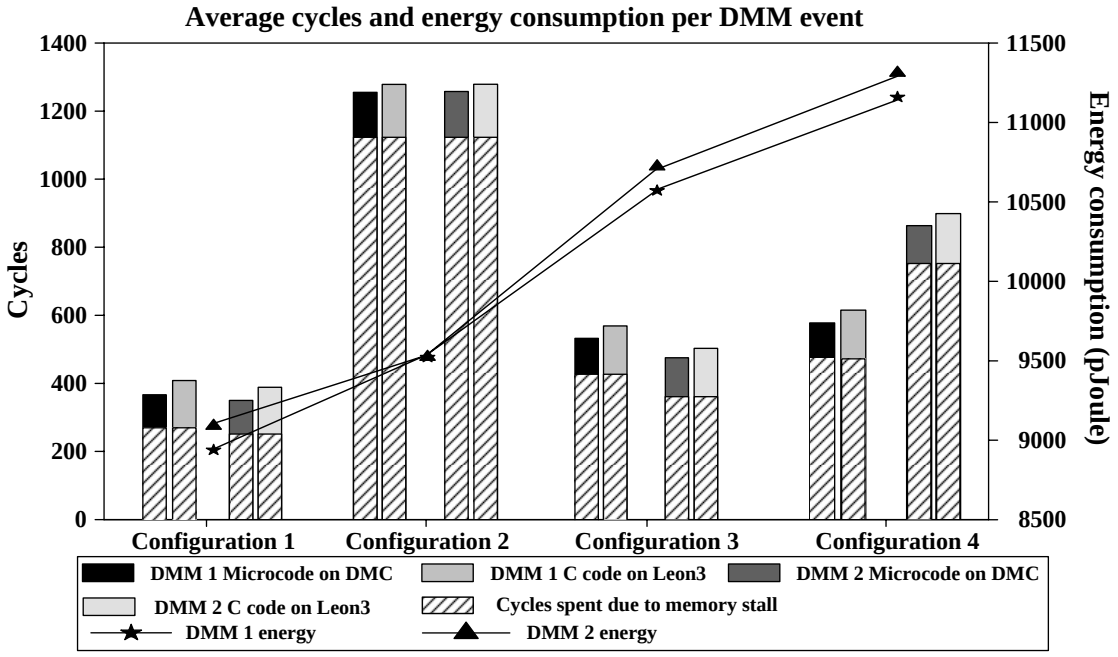

Σχήμα 3.9: a) Μία 2 × 2 πολυπύρηνη NoC αρχιτεκτονική με 3 επεξεργαστικούς κόμβους με τοπική μνήμη (local memory, LM) και 1 κόμβο μνήμης, b) Πλήρως κατανεμημένη μνήμη c) Κεντρικοποιημένος μοναδικός σωρός d) Κατανεμημένοι πολλαπλοί σωροι με επιπλεόν γενικό σωρό e) Κατανεμημένοι πολλαπλοί σωροι με βάση την τοπικότητα της μνήμης και ένας επιπλεόν γενικός σωρός




Σχήμα 3.10: Σύγκριση απόδοσης και κατανομή των αιτημάτων δυναμικής διαχείρισης μνήμης [11].

αιτήσεων στην κατανεμημένη μνήμη και έτσι κάθε φορά που ο LEON3 θέλει να έχει πρόσβαση στη μνήμη, ο DMC είναι υπεύθυνος για την εγκαθίδρυση της επικοινωνίας.

Το Σχήμα 3.11 δείχνει: (i) τον μέσο όρο των κύκλων του επιταχυντή, (ii) τους κύκλους λόγω καθυστέρησης της μνήμης (memory stall) και (iii) τη μέση κατανάλωση ενέργειας (pJoule) ανά αίτημα διαχείρισης δυναμικών δεδομένων. Η Διαμόρφωση 1, εμφανίζεται ώς η πιο γρήγορη αν και είναι αυτή η οποία καταρέει πρώτη. Η Διαμόρφωση 2, είναι η πιο αργή καθώς απαιτούται πολλοί κύκλοι για συγχρονισμό, καθώς όλοι οι κόμβοι προσπαθούν να έχουν προσβαση στον Γενικό Σωρό. Η Διαμόρφωση 4 απαιτεί λίγο περισσότερους κύκλους απο την Διαμόρφωση 3, αλλά είναι μια μικρή ποινή σε σχέση με το γεγονός ότι είναι η καλύτερη λύση όσον αφορά την υπερχείλιση του σωρού. Η ενέργεια υπολογίστηκε με βάση τα αποτελέσματα σύνθεσης του επιταχυντή DMC [33] και τις προσβάσεις στη μνήμη (Cacti [86]). Η Διαμόρφωση 2 καταναλώνει 6% περισσότερη ενέργεια σε σχέση με τη Διαμόρφωση 1, αφού όλα τα αιτήματα εξυπηρετούνται απο τον Γενικό Σωρό. Η Διαμόρφωση 3, καταναλώνει περίπου 18% περισσότερη ενέργεια σε σύγκριση με τη Διαμόρφωση 1. Αυτό προκαλείται από το γεγονός ότι η Διαμόρφωση 3 αποτελείται από περισσότερες εντολές και επιπλέον υπάρχει η κατανάλωση ενέργειας για την επεξεργασία του πίνακα προτεραιοτήτων. Έτσι η Διαμόρφωση 4, καταναλώνει περίπου 25% περισσότερη ενέργεια σε σύγκριση με Διαμόρφωση 1. Αυτό οφειλεται επίσης και λογω της επαυξημένης και συχνής επικοινωνίας για την ανίχνευση του πιο διαθέσιμου σωρου.



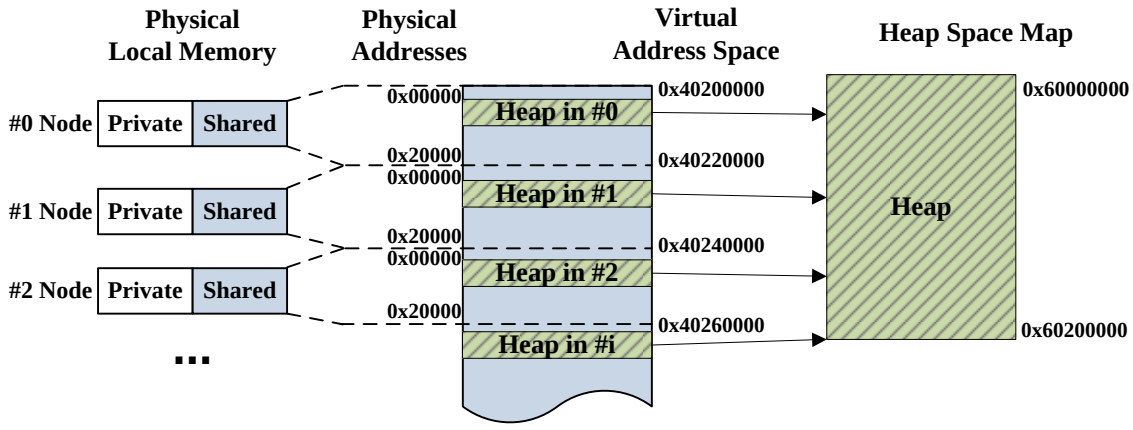

Σχήμα 3.11: Μέσο όρος κυκλων και κατανάλωση ενέργειας ανά αίτημα δυναμικής διαχείρισης μνήμης [11].

## 3.4 Κατανεμημένη επιτάχυνση διαχείρισης μνήμης σε μικροκώδικα

Προηγούμενες ερευνητικές εργασίες σχετικά με τη διαχείριση δυναμικής μνήμης έδειξαν ότι οι απλές υλοποιήσεις ΔΔΜ συχνά αποτελούν εμπόδιο στην απόδοση και στην επεκτασιμότητα των πολυνυματικών εφαρμογών [20].

Οι υπάρχουσες προσεγγίσεις ΔΔΜ βασίζονται κυρίως σε λύσεις λογισμικού που ακόμα κι αν προσφέρουν ευελιξία, απαιτούν πολλούς επεξεργαστικούς κύκλους με αποτέλεσμα την μείωση της απόδοσης. Από την άλλη πλευρά, οι διαθέσιμες λύσεις υλικού (hardware), ακόμη κι αν είναι πιο γρήγορες, δεν μπορούν να εκμεταλλευτούν πλήρως τα χαρακτηριστικά της πλατφόρμας και ακολουθούν μια πιο κεντρικοποιημένη προσέγγιση. Ωστόσο, μια τέτοια κεντρικοποιημένη προσέγγιση έχει αρκετά μειονεκτήματα. Πρώτον, δημιουργείται ένα κεντρικό σημείο αποτυχίας, το οποία καθιστά το όλο σύστημα άχρηστο όταν ο κεντρικός πυρήνας καταρρεύσει. Δεύτερον, η παρουσία ενός και μόνου κεντρικού πυρήνα μπορεί να παρεμποδίσει την επεκτασιμότητα, καθώς σε μεγάλα συστήματα αποτελεί εμπόδιο στην επικοινωνία.

Σε αυτή την ενότητα, παρουσιάζουμε έναν ευέλικτο και επεκτάσιμο κατανεμημένο ΔΔΜ, με το όνομα MAD-DMM (Microcoded-Accelerated Distributed Dynamic Memory Management), ο οποίος αναπτύχθηκε πάνω στον επιταχυντή υλικού DMC [33]. Οι συνεισφορές του MAD-DMM είναι: (i) Εκμεταλλεύεται την παρουσία του επιταχυντή υλικού DMC, με τη χρήση εξατομικευμένων λειτουργιών σε μικροκώδικα, για




Σχήμα 3.12: Η υλοποίηση του Heap Space Map πάνω απο την V2P μετάφραση διευθύνσεων.

την επιτάχυνση της διαχείρισης των δυναμικών δεδομένων, (ii) παρέχει κατανεμημένη λειτουργικότητα σε πολυπύρηνα συστήματα, (iii) χρησιμοποιεί τις ίδιες προγραμματιστικές διεπαφές με την C (`malloc()/free()`) και (iv) μειώνει το φόρτο εργασίας του επεξεργαστή, αφήνοντας όλες τις πράξεις διαχείρισης μνήμης στον DMC.

### 3.4.1 Ο χάρτης απεικόνισης Heap Space Map

Όπως αναφέρθηκε και στην Ενότητα 3.2, η τοπική μνήμη κάθε κόμβου είναι χωρισμένη σε δύο μέρη: στο ιδιωτικό (private) και το κοινό (shared). Κατά συνέπεια, υπάρχουν και δύο τρόποι διευθυνσιοδότησης: φυσική και εικονική. Ο τοπικός πυρήνας, χρησιμοποιώντας τη φυσική διευθυνσιοδότηση μπορεί να έχει πρόσβαση μόνο στην ιδιωτική μνήμη. Όλες οι κοινές μνήμες είναι ορατές σε όλους τους κόμβους και οργανώνονται ως ένας ενιαίος εικονικός χώρος χρησιμοποιώντας την εικονική-προς-φυσική (V2P) μετάφραση διευθύνσεων (Σχήμα 3.3). Προκειμένου να προσφέρουμε κατανεμημένη επιτάχυνση διαχείρισης μνήμης σε μικροκώδικα εισάγουμε την έννοια *Heap Space Map (HSM)* η οποία λειτουργεί πάνω απο την V2P μετάφραση διευθύνσεων.

Το Σχήμα 3.12 παρουσιάζει την υλοποιηση του HSM πάνω απο την V2P μετάφραση διευθύνσεων. Κάθε κόμβος προσφέρει ένα μέρος της κοινής μνήμης του, ως μέρος του σωρού. Στην αριστερή πλευρά, φαίνονται οι κόμβοι της πλατφόρμας καθένας από τους οποίους διαθέτει ιδιωτική και κοινή μνήμη. Οι φυσικές διευθύνσεις της κοινής μνήμης εκτίνονται απο 0x00000 μέχρι 0x20000. Σύμφωνα με τη V2P μετάφραση, όλες οι κοινές μνήμες οργανώνονται ως ένας ενιαίος εικονικός χώρος. Η εφαρμογή χρησιμοποιεί τις εικονικές διευθύνσεις (0x40200000, κ.λπ.), προκειμένου να έχει πρόσβαση στην κοινή μνήμη. Η ενεργοποίηση των αντίστοιχων φυσικών κόμβων καθώς και η V2P μετάφραση διευθύνσεων γίνεται απο τον DMC ελεγκτή.

Στον MAD-DMM, το HSM αποτελείται από το σύνολο των διαθέσιμων σωρών που

προσφέρονται από κάθε κόμβο της πλατφόρμας και είναι διαθέσιμο στις εφαρμογές ως ένας συνεχής ενιαίος χώρος. Κάθε κόμβος προσφέρει ένα μέρος της κοινής μνήμης του, ως μέρος του σωρού. Το HSM, χρησιμοποιώντας την υπηρεσία V2P, δημιουργεί ένα άλλο σχήμα διευθυνσιοδότησης (π.χ. ξεκινώντας από `0x60000000`) που θα χρησιμοποιηθεί ως σωρός. Όλοι οι κόμβοι μπορούν τώρα να δούν το σωρό ως έναν ενιαίο πίνακα διευθύνσεων, ακόμη και αν ο σωρός στην πραγματικότητα αποτελείται από ξεχωριστά μέρη της κοινής μνήμης. Η μετάφραση των διευθύνσεων γίνεται σε δύο επίπεδα: (i) Αρχικά από το HSM (σε επίπεδο `C`) και (ii) στη συνέχεια από τον αντίστοιχο DMC (V2P υπηρεσία σε μικροκώδικα).

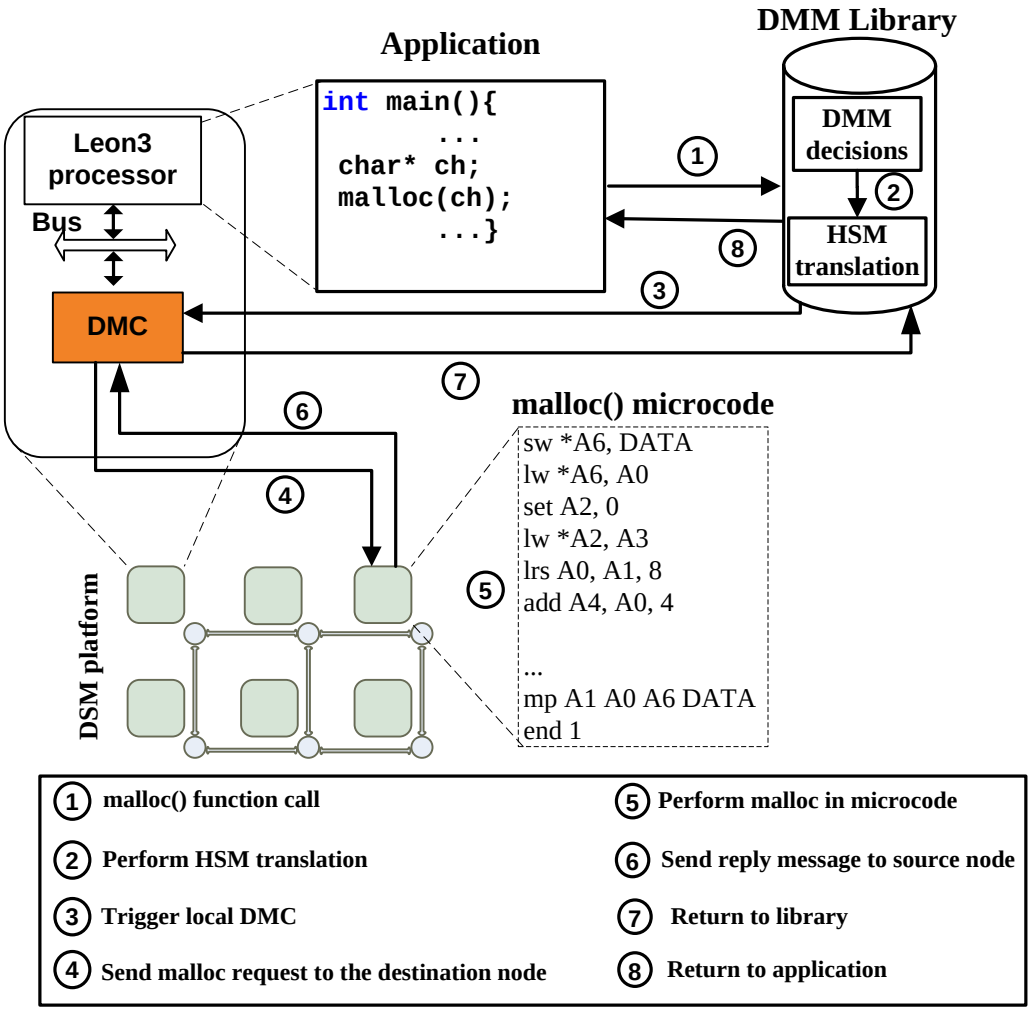
### 3.4.2 Υλοποίηση του MAD-DMM

*Ο MAD-DMM εκμεταλλεύεται την επιτάχυνση των διαδικασιών διαχείρισης μνήμης, που προσφέρονται από την παρουσία του DMC, εκτελώντας σε μικροκώδικα όλες τις απαραίτητες ενέργειες. Με άλλα λόγια, οι λειτουργίες των ΔΔΜ (π.χ. `malloc()/free()`) έχουν κατασκευαστεί σε μικροκώδικα και εκτελούνται απο τον επιταχυντή υλικού.*

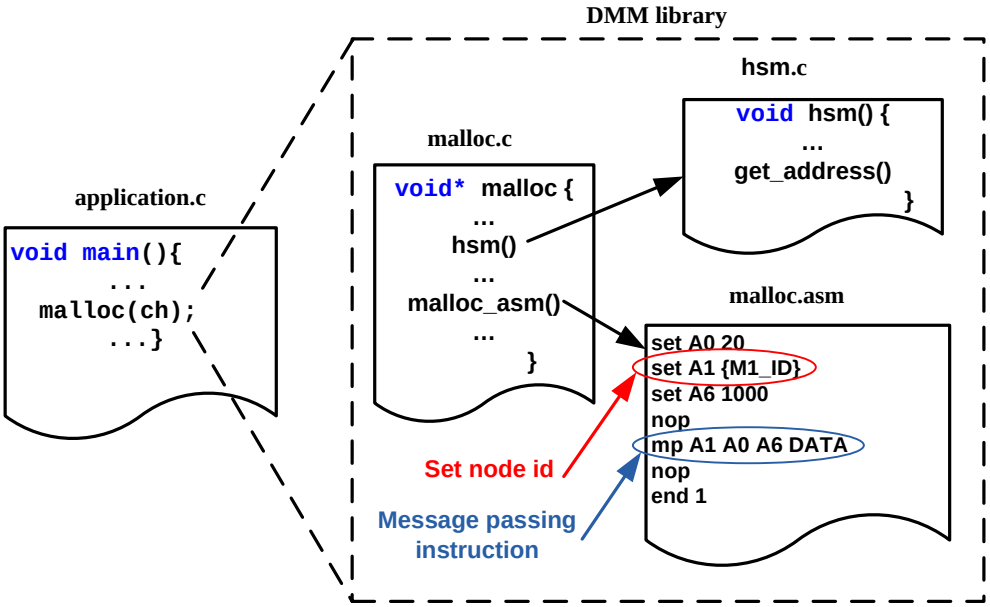
Το Σχήμα 3.13 παρουσιάζει την ροή του MAD-DMM. Το σημείο εισόδου είναι οποιαδήποτε `C` εφαρμογή. Όταν υπάρχει μία κλήση συνάρτησης `malloc()/free()`, ο ΔΔΜ προσπαθεί να διαβάσει τον τοπικό πίνακα σωρού του πρώτου κόμβου στο HSM. Εάν ο κόμβος δεν είναι διαθέσιμος ή δεν έχει άλλο ελεύθερο χώρο, ο MAD-DMM, σύμφωνα με την HSM μετάφραση διευθύνσεων, προσπαθεί στον επόμενο κόμβο. Όταν ο κόμβος βρεθεί (*προορισμός*), ο τοπικός DMC (*πηγή*) ενεργοποιείται και οι ακόλουθες ενέργειες λαμβάνουν χώρα: (i) ο κόμβος πηγή στέλνει ένα μήνυμα με την κλήση της συνάρτησης και το `node id`, (ii) όταν ο κόμβος προορισμού λάβει το μήνυμα, εκτελεί την συνάρτηση σε μικροκώδικα και (iii) ο κόμβος προορισμού επιστρέφει ένα μήνυμα επιτυχίας ως απάντηση στον κόμβο πηγή.

Ο MAD-DMM προσφέρει κατανεμημένη επιτάχυνση διαχείρισης μνήμης σε μικροκώδικα για κατανεμημένα συστήματα μνήμης. Παρόμοια με άλλους ΔΔΜ, ο MAD-DMM έχει τις ίδιες προγραμματιστικές διεπαφές με την `C`, προσφέροντας ευκολία στη χρήση. Τα κύρια χαρακτηριστικά και οι παράγοντες διαφοροποίησης του MAD-DMM είναι τρεις: (i) υποστήριξη κατανεμημένου κοινού σωρού, (ii) επιτάχυνση σε μικροκώδικα λειτουργιών διαχείρισης μνήμης και (iii) επεκτασιμότητα. Ο MAD-DMM χειρίζεται το σωρό σαν ένα συνεχή χώρο με τη βοήθεια του DMC. Επιπλέον, οι κόμβοι στις πολυπύρηνες πλατφόρμες είναι απόλυτα ενήμεροι για την κατάσταση του σωρού. *Σε αντίθεση με άλλους state-of-the-art ΔΔΜ υψηλού επιπέδου [20, 61, 92], οι πληροφορίες σχετικά με τα μετα-δεδομένα (meta-data) δεν αποθηκεύονται σε υψηλό επίπεδο, αλλά σε επίπεδο μικροκώδικα ως μέρος του κάθε τοπικού σωρού.* Κάθε φορά που υπάρχει μια κλήση συνάρτησης, οι πληροφορίες κατάστασης του σωρού διαβάζονται και ενεργοποιείται η αντίστοιχη λειτουργικότητα. Με τον τρόπο αυτό, δεν απαιτείται επιπλέον εσωτερική επικοινωνία μεταξύ των κόμβων.

Το Σχήμα 3.14 παρουσιάζει τις προγραμματιστικές διεπαφές του MAD-DMM μεταξύ `C` και μικροκώδικα. Κάθε φορά που μια εφαρμογή καλεί μια σύναρτηση, π.χ `malloc()`,




Σχήμα 3.13: Επισκόπηση της λειτουργίας του MAD-DMM.



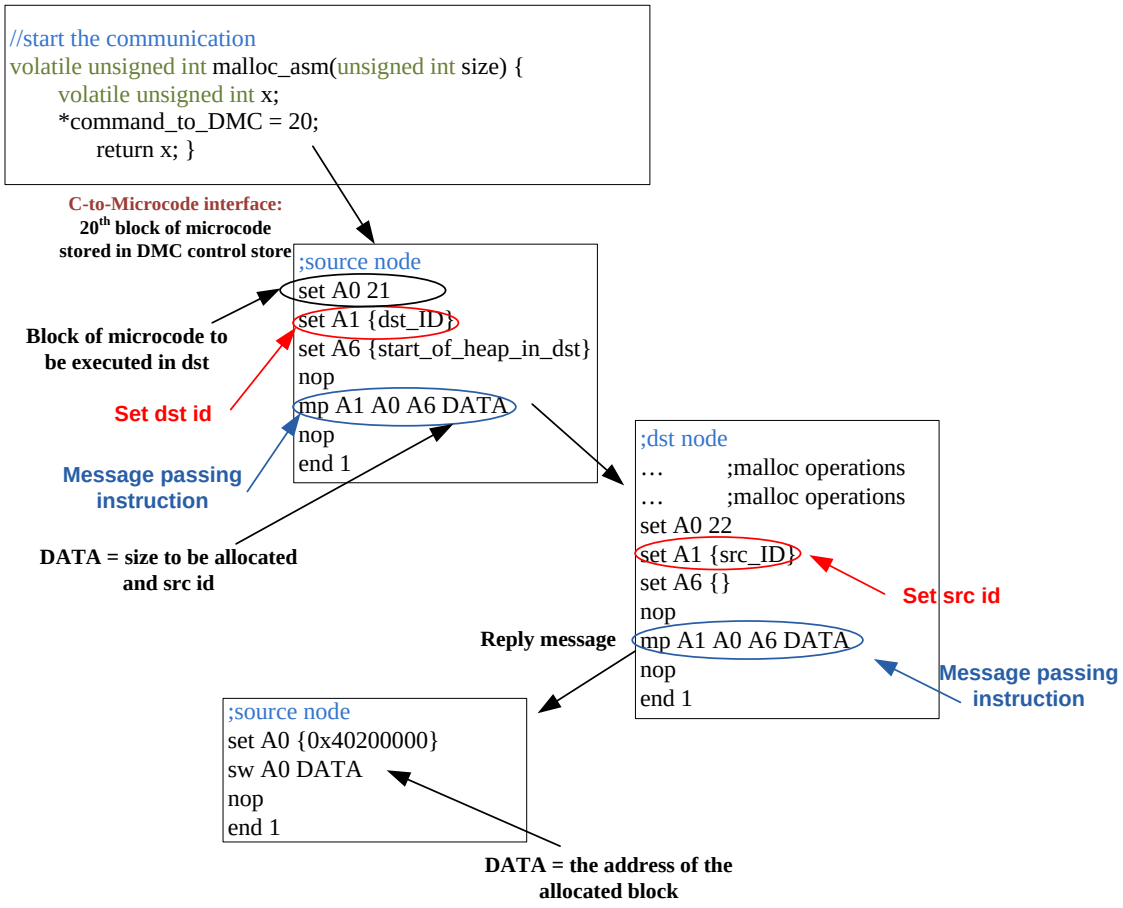

Σχήμα 3.14: Παρουσίαση των προγραμματιστικών διεπαφών του MAD-DMM.

ο MAD-DMM ενεργοποιείται. Μια από τις πρώτες λειτουργίες που πραγματοποιούνται είναι η `hsm()`. Ο στόχος αυτής της λειτουργίας είναι να ελέγξει το σωρό και να βρει τον κατάλληλο κόμβο για να εξυπηρετήσει το αίτημα. Όταν η συνάρτηση `hsm()` ολοκληρωθεί, η συνάρτηση `malloc_asm()` πραγματοποιεί τις κατάλληλες ενέργειες σε επίπεδο μικροκώδικα. Ένα από τα βασικά πράγματα που πρέπει να εκτελεστούν από την `malloc_asm()` είναι να στείλει μήνυμα στον αντίστοιχο κόμβο. Αυτό γίνεται χρησιμοποιώντας την τεχνική μετάδοσης μηνυμάτων.

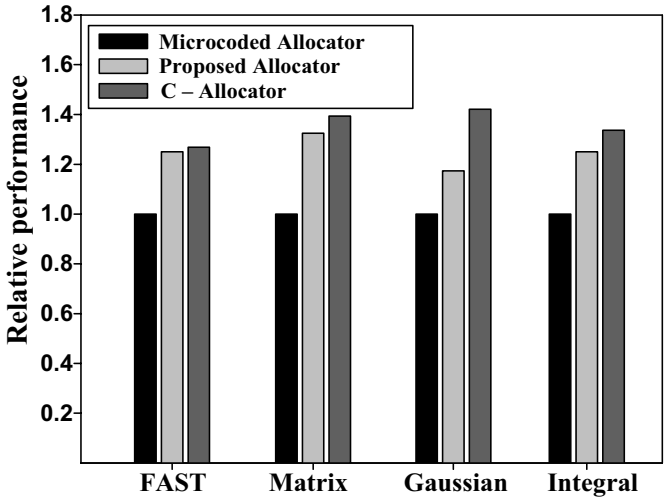
Το Σχήμα 3.15 παρουσιάζει την επικοινωνία μεταξύ των κόμβων με τη χρήση της τεχνικής μετάδοσης μηνυμάτων όταν ο κόμβος προορισμός έχει βρεθεί. Στην περίπτωση αυτή, ένα σήμα από τον κόμβο (i,j) στέλνεται στο τοπικό DMC (πηγή). Στη συνέχεια, ο DMC του κόμβου πηγή, χρησιμοποιώντας την τεχνική μετάδοσης μηνυμάτων, στέλνει ένα μήνυμα με το μέγεθος του αιτήματος και τον αριθμό του κόμβου στον προορισμό. Ο ελεγκτής DMC του προορισμού ενεργοποιείται από διασύνδεση του δικτύου (Σχήμα 3.2) και εκτελεί τις ενέργειες σε μικροκώδικα. Όταν η εξυπηρέτηση του συμβάντος τελειώσει, ο κόμβος προορισμός επιστρέφει τη διεύθυνση με ένα μήνυμα απάντησης στον κόμβο πηγή και η διεύθυνση διαδίδεται στο υψηλό στρώμα του ΔΔΜ και τέλος στην εφαρμογή.

### 3.4.3 Αξιολόγηση του MAD-DMM

Για την αξιολόγηση του MAD-DMM, χρησιμοποιήθηκαν τα ίχνη από τέσσερις εφαρμογές: (i) FAST (Features from an Accelerated Segment Test), (ii) Gaussian, (iii) Integral και (iv) Matrix Multiplication. Η FAST εφαρμογή είναι ένας αλγόριθμος εντοπισμού ακμών, που χρησιμοποιείται ευρέως στην όραση υπολογιστών. Στην Gaussian εφαρμογή, ένα εφέ blur Gaussian δημιουργείται σε μια εικόνα της οποίας τα εικονοστοιχεία




Σχήμα 3.15: Επικοικωνία μεταξύ των κόμβων χρησιμοποιώντας την τεχνική μετάδοσης μηνυμάτων (message passing). Ο κόμβος πηγή ενεργοποιεί την `malloc()` σε μικροκώδικα στον κόμβο προορισμό και κατόπιν ο κόμβος προορισμού επιστρέφει ένα μήνυμα επιτυχίας ως απάντηση στον κόμβο πηγή.



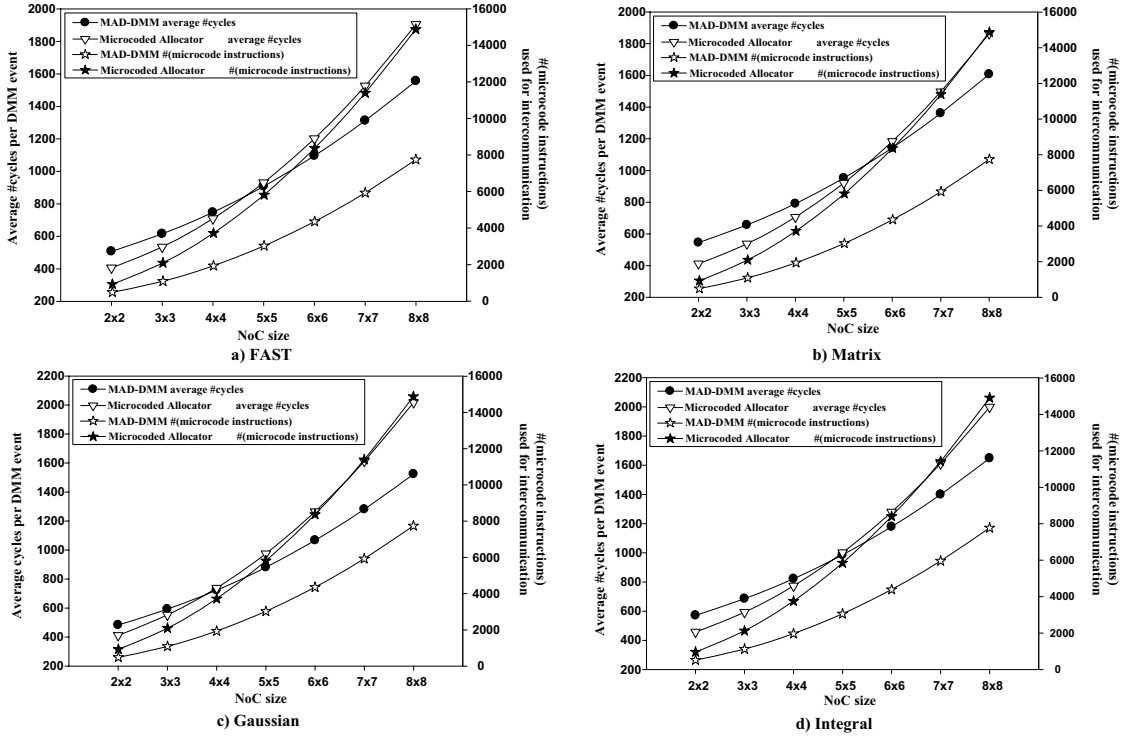

Σχήμα 3.16: Σύγκριση των επιδόσεων του MAD-DMM με τους ΔΔΜ [11] και [61].

δίνονται σε έναν πίνακα. Η εφαρμογή Integral υπολογίζει το ολοκλήρωμα των στοιχείων μιας μήτρας. Τέλος, η εφαρμογή Matrix Multiplication εκτελεί πολλαπλασιασμό πίνακα επί πίνακα.

Όλες οι εφαρμογές ακολουθούν το master-slave μοντέλο: Ο ένας κόμβος, ενεργεί ως ελεγκτής της πλατφόρμας υπεύθυνος για τη διαχείριση μνήμης, ενώ οι υπόλοιποι είναι διαθέσιμοι για την εκτέλεση της εφαρμογής. Πριν από την εκτέλεση οποιασδήποτε εργασίας, ο ελεγκτής θα πρέπει να διαθέσει μνήμη δυναμικά και στη συνέχεια να επιστρέψει ένα δείκτη της χορηγούμενης διεύθυνσης μνήμης στην εκτελεσθείσα εργασία. Αντίστοιχα, ο ελεγκτής είναι υπεύθυνος για την απελευθέρωση της μνήμης μετά από κάθε ολοκλήρωση διαδικασίας. Οι κύκλοι που δαπανούνται για δυναμική διαχείριση δεδομένων στις συγκεκριμένες εφαρμογές είναι κατά μέσο όρο 18,8% σε σχέση με τους απαιτούμενους συνολικούς κύκλους.

Προκειμένου να αξιολογηθεί η απόδοση του MAD-DMM, τον συγκρίναμε σε μία $2\times2$ πολυπύρηνη πλατφόρμα εναντίον (i) ενός ΔΔΜ σε μικροκώδικα ο οποίος λαμβάνει υπόψην την κατανομή της μνήμης [11] και (ii) έαν ΔΔΜ υψηλού επιπέδου [61] (C) ο οποίος στοχεύει στην απόδοση του συστήματος.

Το Σχήμα 3.16 παρουσιάζει την κανονικοποιημένη απόδοση των τριών ΔΔΜ. Όπως ήταν αναμενόμενο, ο ΔΔΜ σε μικροκώδικα είναι ο ταχύτερος όλων, δεδομένου ότι όλες οι πράξεις και οι αποφάσεις εκτελούνται σε χαμηλό επίπεδο. Ωστόσο, ο ΔΔΜ δεν διαθέτει προγραμματιστικές διεπαφές υψηλού επιπέδου καθιστώντας την ενσωμάτωση με C εφαρμογές δύσκολη. Ο MAD-DMM είναι κατά μέσο όρο 25% πιο αργός από τον ΔΔΜ σε μικροκώδικα και 10% γρηγορότερος από τον ΔΔΜ υψηλού επιπέδου. Αυτό οφείλεται στο γεγονός ότι το μεγαλύτερο μέρος των διαδικασιών πραγματοποιούνται σε επίπεδο μικροκώδικα και μόνο ο χειρισμός των διευθύνσεων πραγματοποιείται σε υψηλό επίπεδο (C). Στόχος του MAD-DMM είναι όλοι οι κόμβοι να είναι ενήμεροι σχετικά με την κατάσταση του σωρού έχοντας σαν συνέπεια μια μικρή μείωση της απόδοσης.



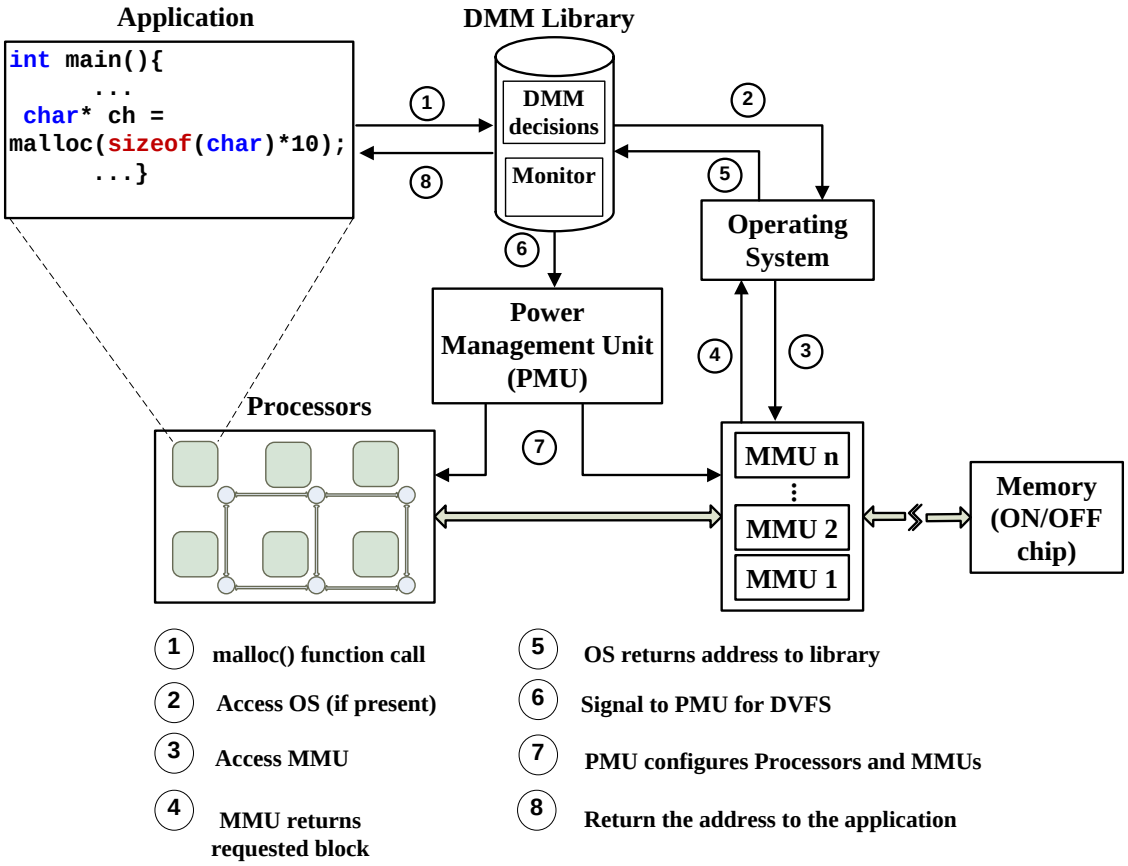

Σχήμα 3.17: Μέσος όρος κύκλων ανά `malloc/free` και αριθμός εντολών μικροκώδικα που χρειάζονται για την επικοινωνία. (a) FAST, (b) Matrix, (c) Gaussian και (d) Integral.

Για να επαληθεύσουμε την κατανεμημένη συμπεριφορά και την επεκτασιμότητα του MAD-DMM, συγκρίναμε (i) τους κύκλους ανά διαδικασία διαχείρισης (`malloc()`/`free()` κλήση) και (ii) τον αριθμό των εντολών σε μικροκώδικα που απαιτούνται για την επικοινωνία για διάφορα μεγέθη πλατφόρμας, απο 4 μέχρι και 64 κόμβους. Όπως φαίνεται στο Σχήμα 3.17 , ο ΔΔΜ σε μικροκώδικα [11] απαιτεί κατά μέσο όρο λιγότερους κύκλους απο τον MAD-DMM όταν η πλατφόρμα είναι μικρότερη από 5×5. Ωστόσο, όταν το μέγεθος της πλατφόρμας αυξηθεί πάνω από τους 25 κόμβους, ο MAD-DMM χρειάζεται λιγότερους κύκλους. Αυτό συμβαίνει επειδή ο ΔΔΜ σε μικροκώδικα [11] βασίζεται σε πίνακες προτεραιοτήτων. Με τη χρήση των πινάκων προτεραιότητας, κάθε κόμβος αποθηκεύει στην τοπική μνήμη όλους τους πιθανούς κόμβους. Για ένα 2×2 σύστημα ο πίνακας περιέχει 4 εγγραφές, ενώ για ένα 8× 8 το μέγεθος αυξάνεται σε 64 και πολλοί κόμβοι έχουν κοινές εγγραφές με αποτέλεσμα να εμφανίζεται το φαινόμενο του ανταγωνισμού των πόρων. Ο MAD-DMM χρησιμοποιεί ένα ελαφρύτερο και πιο γενικό σύστημα επικοινωνίας μεταξύ των κόμβων, το οποίο βοηθά στην κλιμάκωση της πλατφόρμας. Το Σχήμα 3.17 δείχνει ότι ο ΔΔΜ σε μικροκώδικα χρειάζεται κατά μέσο όρο 29% περισσότερους κύκλους για να εξυπηρετήσει ένα αίτημα κάθε φορά που αυξάνεται η πλατφόρμα, ενώ ο MAD-DMM χρειάζεται αντίστοιχα 20%. Επίσης, ο MAD-DMM χρησιμοποιεί κατά μέσο όρο 1.9× λιγότερο μικροκώδικα για την επικοινωνία των κόμβων.

## 3.5 Σχεδίαση ΔΔΜ με βάση την κατανάλωση ισχύος χρησιμοποιώντας DVFS

Όπως προαναφέρθηκε, η μνήμη αποτελεί σημαντικό παράγοντα στην απόδοση και στην κατανάλωση ισχύος των πολυπύρηνων ενσωματωμένων συστημάτων. Οι καινούριες εφαρμογές πολυμέσων βασίζονται σε μεγάλο βαθμό στη διαχείριση της δυναμικής μνήμης λόγω του απρόβλεπτου χαρακτήρα των δεδομένων εισόδου και της αλληλεπίδρασης των χρηστών. Επιπλέον, σύγχρονα υπολογιστικά πρότυπα έχουν δημιουργήσει νέες προκλήσεις για την ανάπτυξη κατανεμημένων εφαρμογών και υπηρεσιών, όπως το MapReduce [37]. Όπως παρουσιάζεται στο [85], η δυναμική διαχείριση μνήμης για τον αλγόριθμο MapReduce διαδραματίζει ουσιαστικό ρόλο στην συνολική απόδοση του συστήματος και την επεκτασιμότητά του. Επιπλέον, ο αυξημένος δυναμισμός στην αποθήκευση των δεδομένων οδηγεί σε απροσδόκητες διακυμάνσεις του αποτυπώματος μνήμης κάτι το οποίο είναι άγνωστο κατά τη φάση σχεδίασης.

Επιπλέον, η κατανάλωση ενέργειας στα ενσωματωμένα συστήματα είναι ένας σημαντικός παράγοντας τον οποίο οι σχεδιαστές πάντα προσπαθούν να μειώσουν όσο περισσοτερο γινεται κρατώντας όμως ανέπαφες τις απαιτήσεις των εφαρμογών. Αφού οι ΔΔΜ γίνονται όλο και πιο κυρίαρχα συστατικά των σύγχρονων συστημάτων, το θέμα της κατανάλωσης ισχύος πρέπει να λαμβανεται υπόψη, επηρεάζοντας έτσι και τον σχεδιασμό τους. Μια τεχνική σχεδιασμού που έχει ως στόχο την εξοικονόμηση ενέργειας και χρησιμοποιείται στις σύγχρονες ενσωματωμένες πλατφόρμες είναι η Δυναμική Κλιμάκωση Τάσης Συχνότητας (Dynamic Voltage Frequency Scaling, DVFS) [48], η οποία δίνει στους επεξεργαστές καθώς και σε και άλλες, εντός του ολοκληρωμέ-




Σχήμα 3.18: Μεθοδολογια σχεδίασης ΔΔΜ με βάση την κατανάλωση ισχύος.

νου, μονάδες να λειτουργούν σε πολλαπλές συχνότητες κάτω από διαφορετικές τάσεις τροφοδοσίας κατά το χρόνο εκτέλεσης.

Σε αυτή την Ενότητα, συνδυάζουμε την έννοια της δυναμικής διαχείρισης μνήμης με την τεχνική DVFS στοχεύοντας στην χαμηλή κατανάλωση ισχύος σε πολυπύρηνες αρχιτεκτονικές. Στόχοι της αναπτυχθείσας μεθοδολογίας είναι: (i) Η εύρεση ενός αποτελεσματικού τρόπου για την ενσωμάτωση DVFS μηχανισμών σε οποιαδήποτε ΔΔΜ υψηλού επιπέδου και (ii) η εκχώρηση της ικανότητας σε οποιαδήποτε ΔΔΜ να αλλάξει κατά το χρόνο εκτέλεσης την τάση τροφοδίας και συχνότητα του συστήματος, ανεξάρτητα από τις επιλεγμένες πολιτικές διαχειρισης του σωρού. Οι καινοτομίες της προτεινόμενης μεθοδολογίας είναι η ανάπτυξη, εντός του ΔΔΜ, ενός γρήγορου μηχανισμού επόπτευσης υψηλού επιπέδου βασισμένο σε παράθυρα παρατήρησης και η ολοκλήρωση υψηλού επιπέδου προγραμματιστικών διεπαφών για τη χρήση DVFS. Επιπλέον, αναπτύχθηκε ένας μηχανισμός για τη λήψη αποφάσεων για να πραγματοποιεί τις κατάλληλες αποφάσεις σχετικά με τη σωστή συχνότητα και τάση τροφοδοσίας [13].

Στο καλύτερο της γνώσης μας, αυτή είναι η πρώτη ερευνητική εργασία στην οποία DVFS τεχνικές έχουν ενσωματωθεί ως μέρος ενός ΔΔΜ, προσφέροντας έτσι ΔΔΜ βασισμένους στην κατανάλωση ισχύος ανεξάρτητους απο την εφαρμογη και τις διαθέσιμες

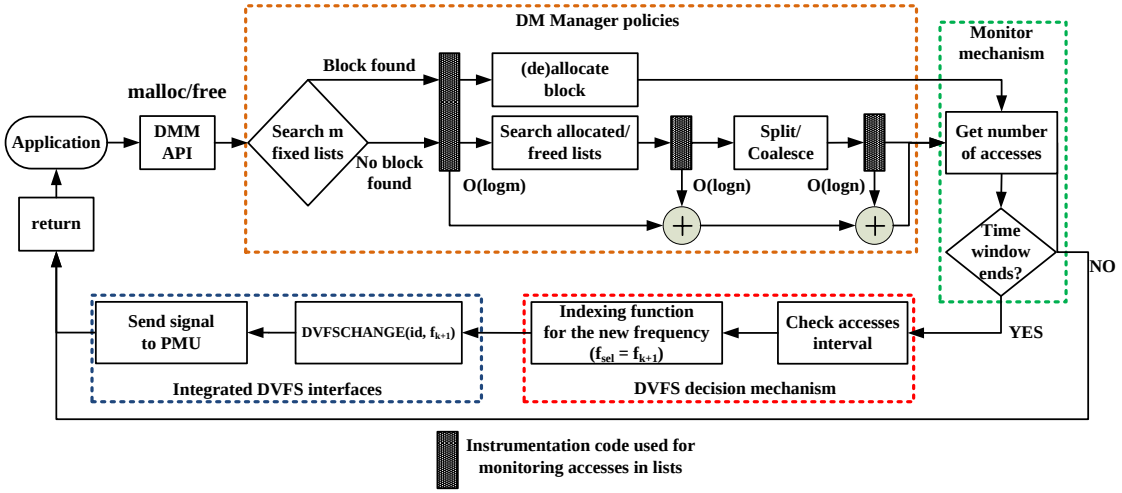
πολιτικές διαχειρισης του σωρού [13].

### 3.5.1 Ολοκλήρωση μηχανισμών DVFS σε ΔΔΜ

Το Σχήμα 3.18 δείχνει ένα παράδειγμα του πώς μια εφαρμογή αλληλεπιδρά με τους βελτιωμένους ΔΔΜ και πώς οι μηχανισμοί DVFS ενεργοποιούνται μέσω των αντίστοιχων προγραμματιστικών διεπαφών. Το σημείο εκκίνησης της μεθοδολογίας είναι μια εφαρμογή γραμμένη σε `C` η οποία εκτελείται σε έναν επεξεργαστή. Η παρουσία ενός λειτουργικού συστήματος δεν είναι απαραίτητη καθώς σε πολλές περιπτώσεις η διαθέσιμη μνήμη δεν επαρκεί ώστε να φιλοξενήσει ένα λειτουργικό σύστημα. Όταν εμφανίζεται μία κλήση δυναμικής διαχειρίσης μνήμης (`malloc()`/`free()`) ενεργοποιείται ο ΔΔΜ ο οποίος χειρίζεται το σωρό με βάση τις προεπιλεγμένες πολιτικές διαχείρισης. Στη συνέχεια, ζητά από το λειτουργικό σύστημα (αν υπάρχει) να του επιστρέψει έναν εικονικό χώρο μνήμης αν χρειάζεται. Στο επόμενο βήμα, η μονάδα διαχείρισης μνήμης (Memory Management Unit, MMU), η οποία είναι αρμόδια για τη διεκπεραίωση των αιτημάτων εκχώρησης μνήμης, αποθηκεύει τις απαραίτητες πληροφορίες σε ειδικά μπλόκ που λέγονται "chunk's" και η διεύθυνση αυτων των μπλόκς επιστρέφεται στο λειτουργικό σύστημα και στον ΔΔΜ. Οι μηχανισμοί επόπτευσης που αναπτύχθηκαν (Ενότητα 3.5.1.1) παρακολουθούν το απαιτούμενο υπολογιστικό κόστος για την εκτέλεση των αιτημάτων και ειδοποιούν με ένα ειδικό σήμα τη μονάδα διαχειρίσης ισχύος του επεξεργαστή (Power Management Unit, PMU), η οποία είναι υπεύθυνη για τη ρύθμιση της συχνότητας του ρολογιού του επεξεργαστή και της τάσης τροφοδοσίας κατά το χρόνο εκτέλεσης. Τέλος η διεύθυνση του μπλοκ επιστρέφεται στην εφαρμογή η οποία συνεχίζει να εκτελείται κάτω από την καινούρια επιλεγμένη συχνότητα ρολογιού και τάση τροφοδοσίας.

Σε αυτή την ενότητα, παρουσιάζουμε τη σύζευξη ΔΔΜ με DVFS στοχεύοντας στην χαμηλή κατανάλωση ισχύος των διαχειριστών. Οι ΔΔΜ έχουν ενισχυθεί με (i) μηχανισμό επόπτευσης, (ii) μηχανισμό απόφασης DVFS και (iii) αντίστοιχες προγραμματιστικές διεπαφές. Το Σχήμα 3.19 παρουσιάζει μια επισκόπηση της βελτίωσης των ΔΔΜ σε τέσσερα στάδια.

Τα μαύρα ορθογώνια κουτιά που απεικονίζονται στο Σχήμα 3.19 αντιπροσωπεύουν τις προσθήκες για τη μέτρηση του υπολογιστικού κόστους των προσβάσεων στις ελεύθερες-λίστες (free-lists) (Ενότητα 3.5.1.1) του ΔΔΜ. Μόλις ο ΔΔΜ τελειώσει την εξυπηρέτηση ενό αιτήματος, υπολογίζεται το υπολογιστικό κόστος απο τους κατάλληλους μηχανισμούς και προωθείται στο δεύτερο μέρος, τον μηχανισμό επόπτευσης. Για την επόπτευση χρησιμοποιούμε ένα μηχανισμό παρατήρησης βασισμένο σε παράθυρα, ώστε να αποφευχθούν μεγάλες απώλειες στην απόδοση του συστήματος (Ενότητα 3.5.1.1). Στη συνέχεια, ο μηχανισμός απόφασης DVFS (Ενότητα 3.5.1.2) είναι υπεύθυνος για την αντιπροσώπευση του μετρούμενου υπολογιστικού κόστους στις διαθέσιμες συχνότητες και τάση τροφοδοσίας του συστήματος. Τέλος, οι επιλεγμένες τιμές συχνότητας-τάσης, διαβιβάζονται στην PMU (Ενότητα 3.5.1.3) και οι αντίστοιχες αλλαγές πραγματοποιούνται.



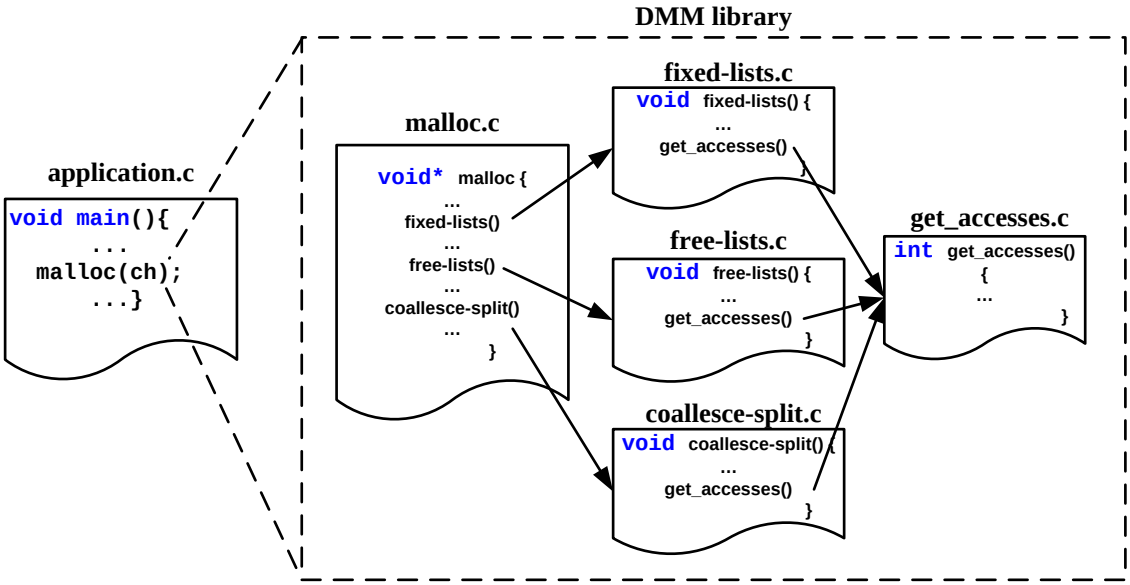

Σχήμα 3.19: Η διαδικασία επόπτευσης και ο μηχανισμός αποφάσεων DVFS [13]

#### 3.5.1.1 Μηχανισμός επόπτευσης

Η παρακολούθηση των ενσωματωμένων αρχιτεκτονικών στη φάση εκτέλεσης αποτελεί προϋπόθεση για τη δοκιμή, αποσφαλμάτωση, καθώς και για την επικύρωση των παραδοχών σχεδιασμού όσον αφορά τη συμπεριφορά του συστήματος και του περιβάλλοντός του. Η κλασική προσέγγιση παρακολούθησης κατά το χρόνο εκτέλεσης είναι η εφαρμογή μηχανισμών επόπτευσης για την ανίχνευση, τη συλλογή και την έκθεση πληροφοριών σχετικά με τη συμπεριφορά του συστήματος. Οι συνηθισμένες τεχνικές που χρησιμοποιούνται για την επόπτευση περιλαμβάνουν ανιχνευτές υλικού και λογισμικού. Στον τομέα του υλικού, οι μηχανισμοί αναπτύσσονται συνήθως είτε σε επίπεδο επεξεργαστή είτε σε επίπεδο συστήματος και απαιτείται επανασχεδιασμός της πλατφόρμας γεγονός το οποίο δεν προσφέρει ευελιξία και προσαρμοστικότητα. Από την άλλη πλευρά, οι μηχανισμοί επόπευσης σε επίπεδο λογισμικού προσφέρουν τη φθηνότερη και πιο ευέλικτη λύση για τη συλλογή πληροφοριών.

Στην παρούσα προτεινόμενη μεθοδολογία, αναπτύξαμε μηχανισμούς επόπτευσης και απόφασης DVFS στο επίπεδο του λογισμικού οι οποίοι στοχεύουν συγκεκριμένα στην διαχείριση του σωρού και ενεργούν ανεξάρτητα απο κάθε άλλη αλλαγή DVFS που εκτελείται από την εφαρμογή. Οι παραγόμενοι ΔΔΜ έχουν εμπλουτιστεί με μηχανισμούς επόπτευσης υπεύθυνοι για τη συλλογή πληροφοριών και την παρακολούθηση του συστήματος σχετικά με το κόστος διάσχισης των λιστών των διαχειριστών.

Τα πλεονεκτήματα του προτεινόμενου μηχανισμού επόπτευσης είναι τα εξής: (i) η επόπτευση σε επίπεδο λογισμικού μπορεί να ενσωματωθεί χωρίς κανένα κόστος με οποιοδήποτε ΔΔΜ, (ii) είναι ανεξάρτητος της πλατφόρμας, (iii) έχει εύκολες προς τον χρήστη προγραμματιστικές διεπεαφές και (iv) οι τροποποιήσεις μπορούν να γίνουν εύκολα σε `C` επίπεδο. Ωστόσο, η χρήση αυτού του τύπου υψηλού επιπέδου μηχανισμών, έχουν ως αποτέλεσμα την μείωση της απόδοσης του συστήματος σε σχέση με λύσεις υλικού και middleware.




Σχήμα 3.20: Αφηρημένη άποψη του μηχανισμού επόπτευσης για τη συλλογή του αριθμού προσβάσεων. Ο συνολικός αριθμός των προσβάσεων διαδίδεται στην DVFS μηχανισμό απόφασης DVFS (Ενότητα 3.5.1.2).

**Κώδικας καθοδήγησης**: Μια σημαντική πτυχή των μηχανισμών επόπτευσης είναι η ελαχιστοποίηση ή η αποφυγή, της παρεμβατικότητας των μηχανισμών πάνω στον συγχρονισμό και στην συμπεριφορά του συστήματος. Όπως προαναφέρθηκε, η αποτυχία γενικά οδηγεί σε μεγάλη μείωση της απόδοσης του συστήματος. Οι λύσεις επόπτευσης σε επίπεδο λογισμικού βασίζονται είτε μέσω κώδικα καθοδήγησης, ο οποίος αναφέρει στη φάση εκτέλεσης διάφορα στατιστικά στοιχεία, είτε μέσω στατιστικών κατανομών [22, 80].

Όπως απεικονίζεται στο Σχήμα 3.19, κώδικας καθοδήγησης έχει προστεθεί μετά από κάθε διακριτή απόφαση του ΔΔΜ. Μια αφηρημένη άποψη υλοποίησης του μηχανισμού επόπτευσης παρουσιάζεται στο Σχήμα 3.20. Οι ΔΔΜ, όπως προαναφέρθηκε, αποτελούνται από ορθογώνια δέντρα απόφασης [92] ανεξάρτητα μεταξύ τους τα οποία συνδυάζονται για την δημιουργία της συμπεριφοράς του διαχειριστή. Για κάθε διακριτή πολιτική διαχείρισης, ένας νέος κώδικας καθοδήγησης (π.χ. `get_accesses()`) έχει εφαρμοστεί και εισάγεται στις αντίστοιχες λειτουργίες. *Ο κώδικας καθοδήγησης είναι υπεύθυνος για την αναφορά του αριθμού των προσβάσεων στις λιστες του ΔΔΜ*. Στο τέλος εξυπηρέτησης κάθε αιτήματος όλες οι απαραίτητες πληροφορίες σχετικά με το συνολικό υπολογιστικό κόστος συλλέγονται και διαδίδονται στον μηχανισμό απόφασης DVFS (Ενότητα 3.5.1.2).

Με τη χρήση αυτής της μεθόδου για τη μέτρηση της απόδοσης του ΔΔΜ μπορούμε να έχουμε έναν γρήγορο και ακριβή μηχανισμό επόπτευσης, σε σύγκριση με τις συμβατικές κλήσεις συστήματος. Ένα σημαντικό ζήτημα που πρέπει να αναφερθεί είναι ότι ο κώδικας καθοδήγησης που ενσωματώσαμε στους ΔΔΜ, δεν χρησιμοποιεί κλήσεις συστήματος (π.χ., `clock()` απο `time.h`). Ο λόγος είναι ότι αυτες οι κλήσεις "παγώνουν" την εκτέλεση της εφαρμογής και προσθέτουν μια ακόμη πιο αισθητή καθυστέρηση

στο σύστημα (μέχρι 15% περισσότερους κύκλους) καθιστώντας τους ακατάλληλους για ελαφριά παρακολούθηση. Τέλος, *η προτεινόμενη προσέγγιση δεν απαιτεί οποιαδήποτε αλλαγή στον κώδικα της εφαρμογής*, δεδομένου ότι όλοι οι μηχανισμοί έχουν ενταχθεί στο ΔΔΜ.

**Επόπτευση με χρήση παραθύρων**: Προκειμένου να αποφευχθεί μεγάλη επιβάρυνση στην επίδοση του συστήματος, οι μηχανισμοί επόπτευσης χρησιμοποιούν μια προσέγγιση βασιζόμενη σε παράθυρα γεγονότων. Ορίζουμε σαν $WS$ το μέγεθος του παραθύρου και σαν $W = \{W_1, W_1, ..., W_k\}$ το σύνολο των παραθύρων $W_i$. Το συνολο $WS$ αποτελείται από αλληλοαποκλειόμενα υποσύνολα $W_i$ καθένα απο τα οποία περιέχει τις απραίτητες πληροφορίες για την εξυπηρέηση των ΔΔΜ-γεγονότων. Σαν ΔΔΜ-γεγονός $e_j$ ορίζουμε είτε μια κλήση `malloc()` είτε μια κλήση `free()`. Αξίζει να τονίσουμε οτι το μέγεθος του υποσυνόλου $W_i$ ειναι το ίδιο για όλα τα υποσύνολα. Για πράδειγμα, αν $WS = 4$, τότε $W_1 = \{e_1, e_2, e_3, e_4\}$, $W_2 = \{e_5, e_6, e_7, e_8\}$ κ.τ.λ. Ορίζουμε σαν $c_{e_j}$ και σαν $c_{W_i}$ το υπολογιστικό κόστος της διάσχισης των λιστών απο το $e_j$ ΔΔΜ-γεγονός στο $W_i$ παράθυρο αντίστοιχα. Το $c_{W_i}$ και το $c_{e_j}$ συνδέονται μέσω της Εξίσωσης 3.3

$$c_{W_i} = \sum_{\forall e_j \in W_i} c_{e_j} \tag{3.3}$$

Στο τέλος του κάθε παραθύρου, ο αριθμός των προσβάσεων στις λίστες του ΔΔΜ συγκεντρώνεται απο τον κώδικα καθοδήγησης και διαδίδεται στον μηχανισμό απόφασης DVFS, προκειμένου να ελεγχθεί εάν υπάρχει ανάγκη για να αλλάξει η συχνότητα λειτουργίας και το επίπεδο της τάσης τροφοδογσίας. Σύμφωνα με τις επιλεγμένες πολιτικές διαχείρισης του σωρού (η ύπαρξη σταθερών λιστων ή μη, πολιτικές ταιριάσματος, διάσπαση και ένωση, κ.λπ.), το κόστος για τη διάσχιση των λιστών και την επιστροφή της διεύθυνσης του μπλοκ ποικίλλει και δεν είναι γνωστό εκ των προτέρων. Επίσης, είναι δύσκολο να προβλεφθεί εκ των προτέρων ο αριθμός των προσβάσεων καθώς αλλάζει κατά το χρόνο εκτέλεσης και εξαρτάται από τα ακόλουθα:

- $\{L_{Fix}\}$: Το σύνολο $L_{Fix}$ περιέχει όλες τις σταθερές λίστες του ΔΔΜ. Οι σταθερές λίστες ταξινομούνται στο $L_{Fix}$ με βάση το μεγεθος που χειρίζονται και η πολυπλοκότητα σε αυτές όσον αφορά την πρόσβαση είναι $T_{S_{sz_m}} = O(1)$. Έτσι το κόστος δέσμευσης ενός μπλοκ σε $m$ διαθέσιμες σταθερές λίστες είναι $T_{L_{Fix}} = O(logm)$.

- $\{L_{Dynamic}\}$: Το σύνολο $L_{Dynamic}$ περιλαμβάνει τις λίστες για τα δεσμευμένα και ελεύθερα μπλοκς. Η πολυπλοκότητα για δέσμευση ή ελευθέρωση ενός μπλοκ στο $L_{Dynamic}$ είναι $T_{L_{Dynamic}} = O(logn)$ όπου $n$ είναι το μέγεθος των λιστών.

- Για ένωση και διάσπαση μπλοκς η πολυπλοκότητα βασίζεται στο μέγεθος του $L_{Dynamic}$ και έτσι η πολυπλοκότητα είναι $T_{coal\|split} = O(logn)$

Έτσι, το συνολικό κόστος πολυπλοκότητας για κάθε παράθυρο $W_i$ είναι το άθροισμα όλων των παραπάνω. Η συνολική πολυπλοκότητα περιγράφεται απο την Εξί-

σωση 3.4.

$$T_{c_{W_i}} = T_{L_{Fix_{W_i}}} + T_{L_{Dynamic_{W_i}}} + T_{coal\|split_{W_i}} \tag{3.4}$$

#### 3.5.1.2 Μηχανισμός απόφασης DVFS

Τα δύο βασικά ερωτήματα που πρέπει να απαντηθούν από κάθε μηχανισμό απόφασης DVFS είναι: (i) πότε πρέπει να αλλάξει η συχνότητα και το επίπεδο τάσης και (ii) ποίες είναι οι κατάλληλες τιμές. Μια πρώιμη ή καθυστερημένη απόφαση για την αλλαγή επιπέδου συχνότητας-τάσης μπορεί να οδηγήσει σε αύξηση της κατανάλωσης ισχύος ή μείωση της απόδοσης αντίστοιχα.

Στο τέλος κάθε παραθύρου $W_i$, ο μηχανισμός απόφασης DVFS ελέγει την τιμή $c_{W_i}$ και την ταιριάζει σε προκαθορισμένα διαστήματα. Ο αριθμός αυτων των διαστημάτων εξαρτάται απο τον αριθμό των διαθέσιμων συχνοτήτων της πλατφόρμας και ο γενικός τύπος είναι $[0, a^K WS]_{K_{0,...,l-1}}$ όπου $l$ είναι ο αριθμός των διαθέσιμων συχνοτήτων και $a$ ειναι μια μεταβλητή, οριζόμενη απο τον σχεδιαστή, που συνδέεται με τον αριθμό των προσβάσεων στις λίστες του ΔΔΜ. Τέλος, ορίζουμε ως $F = \{f_1, f_2, ..., f_l\}$, $f_l > ... > f_2 > f_1$ το σύνολο των $l$ διαθέσιμων συχνοτήτων και $I$ μία 1-1 συνάρτηση ανιστοίχησης $I : c_{W_i} \rightarrow F$ η οποία απεικονίζει το κόστος πολυπλοκότητας στην αντίστοιχη συχνότητα λειτουργίας ($K_0 \rightarrow f_1, K_1 \rightarrow f_2, ..., K_{l-1} \rightarrow f_l$). Για παράδειγμα, υποθέτουμε οτι έχουμε μια πλατφόρμα με 4 επίπεδα συχνοτήτων $F = \{f_1, f_2, f_3, f_4\}$, $f_4 > f_3 > f_2 > f_1$. Στο τέλος κάθε παραθύρου με μέγεθος $WS$ ο μηχανισμός απόφασης DVFS ελέγχει το $c_{W_i}$ και αποφασίζει την καινούρια συχνότητα $f_{new}$ σύμφωνα με την Εξίσωση 3.5:

$$f_{new} = \begin{cases} f_1, & c_{W_i} \in [0, a^0 WS] \\ f_2, & c_{W_i} \in (a^0 WS, a^1 WS] \\ f_3, & c_{W_i} \in (a^1 WS, a^2 WS] \\ f_4 & c_{W_i} > a^2 WS \end{cases} \tag{3.5}$$

Η μεταβλητή $a$ μπορεί να θεωρηθεί ως ένας τρόπος για να δηλώσει πόσο ευαίσθητο είναι το σύστημά μας σε σχέση με τον αριθμό των προσβάσεων στις λίστες του ΔΔΜ και συνεπώς πόσο συχνές θα είναι οι DVFS αλλαγές. Μικρή τιμή του $a$ σημαίνει μικρά $c_{W_i}$ χρονικά διαστήματα και έτσι η πιθανότητα να πάμε από το ένα διάστημα στο άλλο, σύμφωνα με τη συνάρτηση $I$, είναι μεγαλύτερη και οι DVFS αλλαγές θα εμφανίζονται περισσότερες φορές. Όσον αφορά το επίπεδο τάσης, αν η τάση τροφοδοσίας του συστήματος είναι $V$, η μέγιστη συχνότητα $f_{max}$ κατά την οποία η μονάδα μπορεί να λειτουργήσει δίνεται από την προσέγγιση του μοντέλου Alpha [73].

#### 3.5.1.3 Ενσωμάτωση διεπαφών DVFS

Σε αυτή την ενότητα θα περιγράψουμε τις ανεπτυγμένες διεπαφές για την χρήση των παρεχόμενων DVFS μηχανισμών. Οι αναπτυγμένες διεπαφές έχουν ενσωματω-

Πίνακας 3.2: Εντολές διαχειρίσης ισχύος [30]

| Command | Description |
|---|---|
| SETOPTION | χρησιμοποιείται για να ορίσει μια επιλογή διαμόρφωσης της PMU. Η εντολή παίρνει ως παράμετρο μια τιμή αναφέροντας τον κωδικό της επιλογής διαμόρφωσης, και την τιμή στην οποία θα καθοριστεί. |
| DVFSCHANGE | χρησιμοποιείται για να αλλάξει το DVFS μιας περιοχής. Η εντολή παίρνει ως παράμετρο μια τιμή διαιρεσης της γενικής συχνότητας. |
| POWERDOWN | χρησιμοποιείται για να θέσει μια περιοχη σε κατάσταση μειωμένης ενέργειας κλείνοντας τα παρεχόμενα ρολόγια. |
| WAKEUP | χρησιμοποιειται για να ενεργοποιήσει μια περιοχη που ήταν σε κατάσταση POWERDOWN. |

θεί στον ΔΔΜ και μπορούν να ενεργοποιούνται όταν ειναι αναγκαίο. Οι διεπαφές έχουν την παρακάτω μορφή: `COMMAND(id, opt)`, όπου `id` είναι το αναγνωριστικό του επεξεργαστή και `opt` είναι συγκεκριμένες εντολές. Τέσσερις διεπαφές αναπτύχθηκαν που μπορούν να είναι άμεσα προσβάσιμες απο `C` επίπεδο και ενεργοποιούν την PMU: `SETOPTION`, `DVFSCHANGE`, `POWERDOWN`, και `WAKEUP`. Ο Πίνακας 3.2 παρουσιάζει αναλυτικότερα τις εντολές διαχείρισης ενέργειας.

### 3.5.2 Διαμόρφση της πειραματικής διαδικασίας

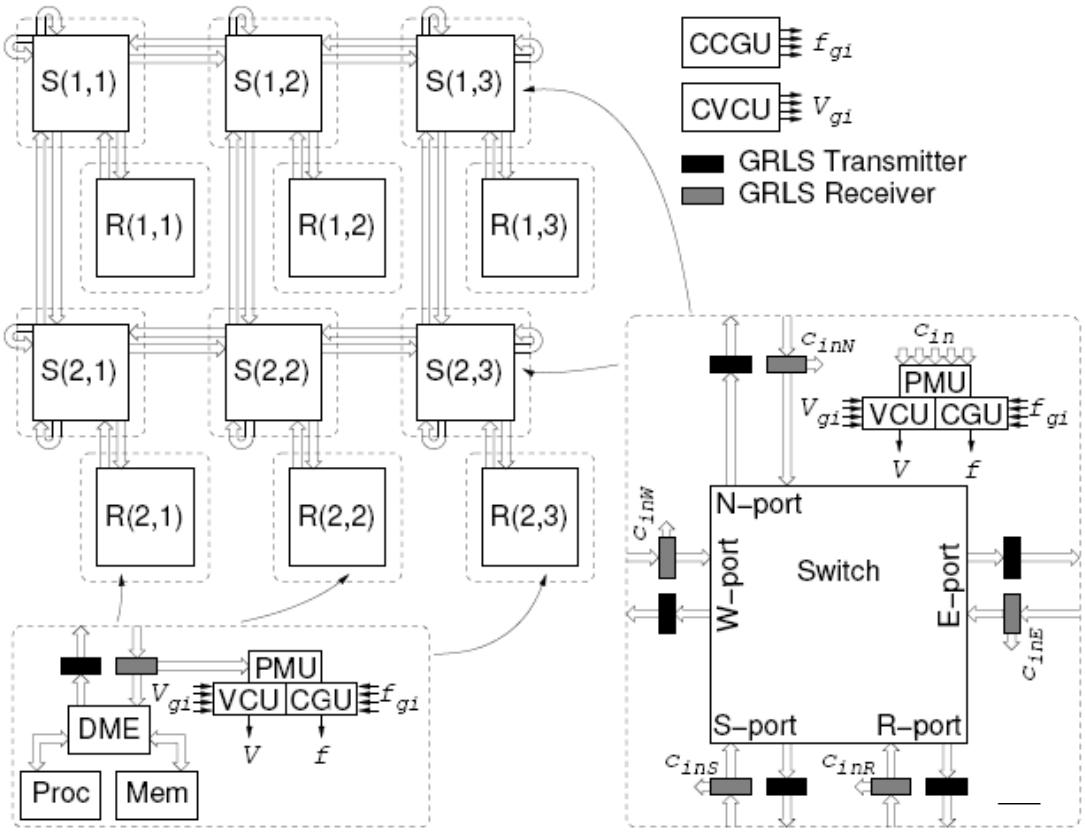
#### 3.5.2.1 Επισκόπηση DVFS

Ένα σύστημα διαχείρισης ενέργειας έχει δημιουργηθεί πάνω στην πλατφόρμα με την εισαγωγή μιας GRLS μονάδας σε κάθε κόμβο. Το σημείο πρόσβασης το οποίο παρέχει τις υπηρεσίες ενέργειας δίνεται από την PMU η οποία ελέγχει την τάση (Voltage Control Unit, VCU) και τη συχνότητα (Clock Generation Unit, CGU) σε κάθε κόμβο. Η δομή της πλατφόρμας δινεται στο Σχημα 3.21.

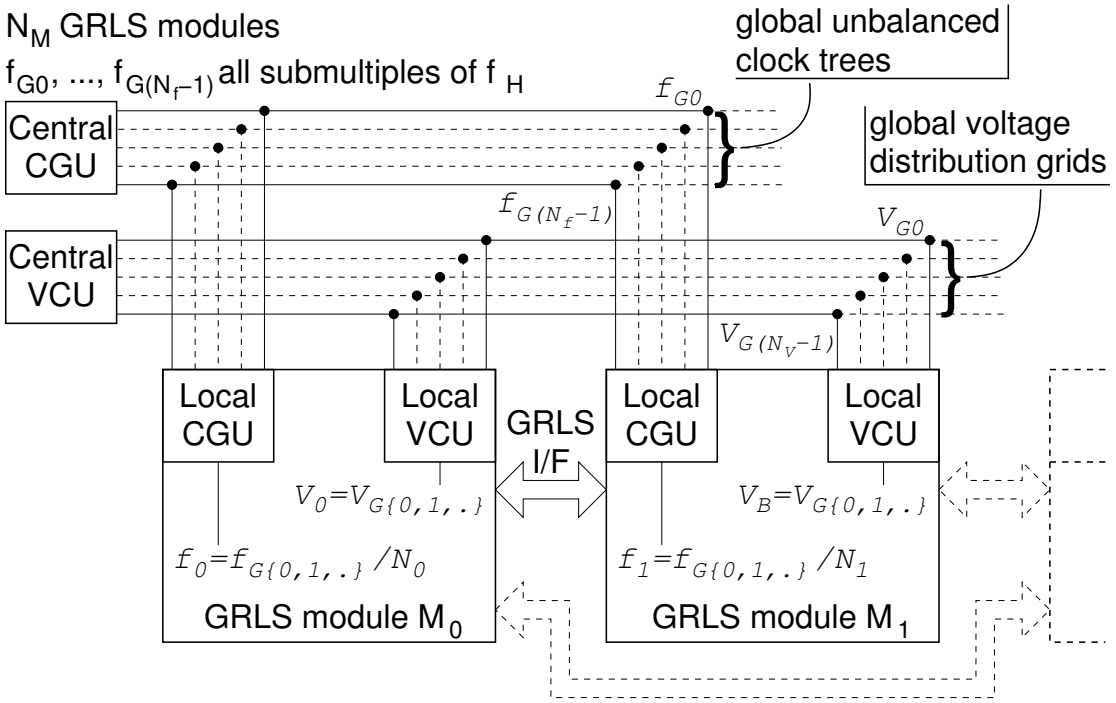
Η CGU παρέχει 4 γενικά ρολογια τα οποία έχουν τις συχνότητες $f_{G0}, f_{G1}, f_{G2}, f_{G3}$, υποπολλαπλάσια μιας γενικής συσχνότητας $f_H$. Επίσης, η πλατφόρμα υποστηρίζει μέχρι τέσσερις τάσεις τροφοδοσίας $V_{G0}, V_{G1}, V_{G2}, V_{G3} (V_{G0} > V_{G1} > V_{G2} > V_{G3})$ μέσω της VCU. Η δομή της πλατφόρμας απο άποψη διαχείρισης DVFS προυσιάζεται στο Σχήμα 3.22.

#### 3.5.2.2 Εφαρμογές και μοντέλο εκτέλεσης

Για την αξιολόγηση, χρησιμοποιήθηκαν τα ίχνη από τέσσερις εφαρμογές: (i) FAST (Features from an Accelerated Segment Test), (ii) Gaussian, (iii) Integral και (iv) Matrix Multiplication. Η FAST εφαρμογή είναι ένας αλγόριθμος εντοπισμού ακμών, που χρησιμοποιείται ευρέως στην όραση υπολογιστών. Στην Gaussian εφαρμογή, ένα εφέ blur Gaussian δημιουργείται σε μια εικόνα της οποίας τα εικονοστοιχεία δίνονται σε έναν πίνακα. Η εφαρμογή Integral υπολογίζει το ολοκλήρωμα των στοιχείων μιας μήτρας.




Σχήμα 3.21: Επισκόπηση GRLS [30]




Σχήμα 3.22: Αρχιτεκτονική της διαχείρισης ισχύος στην πολυπυρήνη πλατφόρμα [30]

Τέλος, η εφαρμογή Matrix Multiplication εκτελεί πολλαπλασιασμό πίνακα επί πίνακα.

Όλες οι εφαρμογές ακολουθούν το master-slave μοντέλο: Ο ένας κόμβος, ενεργεί ως ελεγκτής της πλατφόρμας υπεύθυνος για τη διαχείριση μνήμης, ενώ οι υπόλοιποι είναι διαθέσιμοι για την εκτέλεση της εφαρμογής. Πριν από την εκτέλεση οποιασδήποτε εργασίας, ο ελεγκτής θα πρέπει να διαθέσει μνήμη δυναμικά και στη συνέχεια να επιστρέψει ένα δείκτη της χορηγούμενης διεύθυνσης μνήμης στην εκτελεσθείσα εργασία. Αντίστοιχα, ο ελεγκτής είναι υπεύθυνος για την απελευθέρωση της μνήμης μετά από κάθε ολοκλήρωση διαδικασίας.

#### 3.5.2.3 Επιλεγμένοι ΔΔΜ

Για να επαληθεύσουμε και να αξιολογήσουμε τη συμπεριφορά της προτεινόμενης μεθοδολογίας χρησιμοποιήσαμε πέντε διαφορετικούς ΔΔΜ οι οποίοι ενσχύθηκαν με το μηχανισμό επόπτευσης και μηχανισμό απόφασης DVFS. Οι ΔΔΜ που χρησιμοποιήθηκαν είναι:

- **ΔΔΜ 1 (DM manager 1)**: Ο υψηλού επιπέδου ΔΔΜ με κριτήριο την ελαχιστοποίηση της ισχυος [61].
- **ΔΔΜ 2 (DM manager 2)**: Ο υψηλού επιπέδου ΔΔΜ με κριτήριο την ελαχιστοποίηση της ισχυος [61] με ενεργοποιημένες τις πολιτικές ένωσης και διάσπασης.

- **ΔΔΜ 3 (DM manager 3)**: Ένας ΔΔΜ με best-fit αλγόριθμο ταιριάσματος χωρίς ενεργοποιημένες τις πολιτικές ένωσης και διάσπασης.
- **ΔΔΜ 4 (DM manager 4)**: Ένας ΔΔΜ με best-fit αλγόριθμο ταιριάσματος και ενεργοποιημένες τις πολιτικές ένωσης και διάσπασης.
- **ΔΔΜ 5 (DM manager 5)**: Ένας ΔΔΜ με σταθερές λίστες.

### 3.5.3 Αξιολόγηση

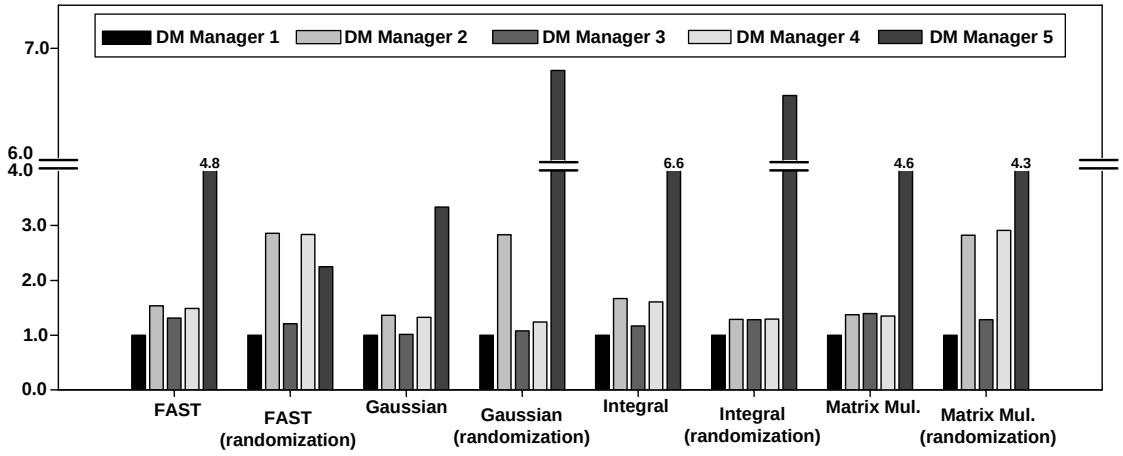
Για να επαληθεύουμε την προσέγγισή μας, πραγματοποιήσαμε εκτεταμένες προσομοιώσεις [13] του προτεινόμενου πλαισίου στοχεύοντας την κατανάλωση ισχύος της πλατφόρμας, τον κατακερματισμό του σωρού και γενικά την απόδοση για κάθε έναν από τους προαναφερθέντες ΔΔΜ.

#### 3.5.3.1 Κατανάλωση ισχύος και κατακερματισμός του σωρού των επιλεγμένων ΔΔΜ

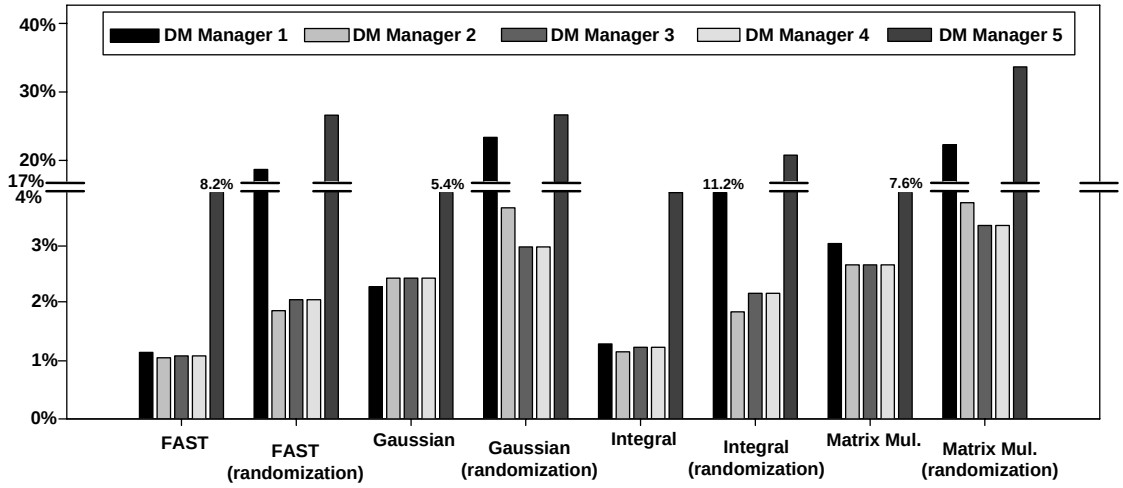
Το μοντέλο ισχύος που χρησιμοποιήσαμε για την εκτίμηση της ισχύος βασίζεται στο γεγονός ότι η κατανάλωση ισχύος είναι ανάλογη προς τη συχνότητα και το τετράγωνο της τάσης τροφοδοσίας στην οποία τρέχει μία μονάδα. Το μοντέλο, σύμφωνα με τα αποτελέσματα σύνθεσης, που προέρχονται από το εργαλείο Synopsys Design Compiler Compiler, μπορεί να καθορίσει ποια είναι η κατανάλωση ισχύος του μπλοκ, όταν λειτουργεί σε μια ορισμένη συχνότητα και τάση τροφοδοσίας.

Τα Σχήματα 3.23 και 3.24 παρουσιάζουν την κανονικοποιημένη κατανάλωση ισχύος και τα αποτελέσματα του κατακερματισμού του σωρού για τους πέντε επιλεγμένους ΔΔΜ αντίστοιχα, χωρίς οποιοδήποτε μηχανισμό επόπτευσης ή αλλαγής DVFS. Ορίζουμε ως κατακερματισμό του σωρού το ποσοστό του μέγιστου ποσού της μνήμης που εκχωρείται από το σύστημα διαιρούμενο με το μέγιστο ποσό μνήμης που απαιτείται από την εφαρμογή [20]. Ο κατακερματισμός του σωρού είναι ένα σημαντικό μετρικό στο πεδίο της δυναμικής διαχείρισης μνήμης το οποίο κατά την υπερβολική κατάτμηση μπορεί να μειώση την απόδοση του συστήματος λόγω προβλημάτων σελιδοποίησης (paging problems).

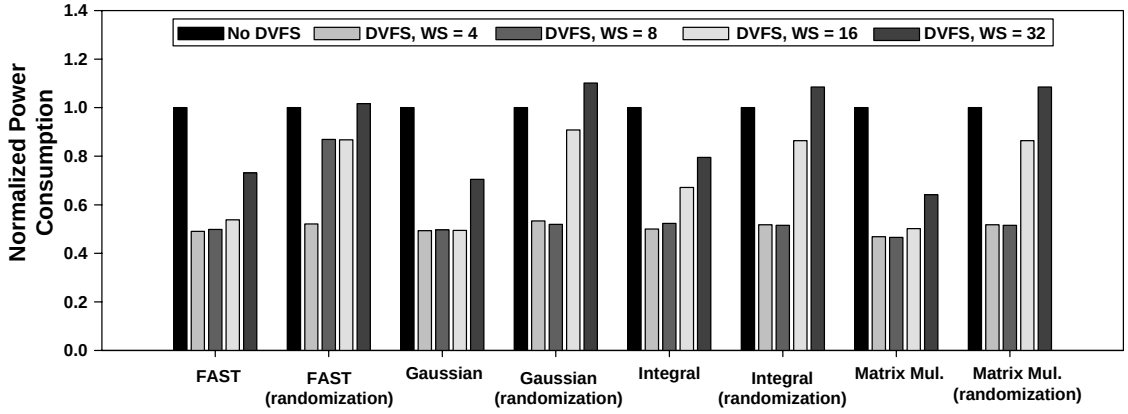
Το Σχήμα 3.23 παρουσιάζει την κανονικοποιημενή κατανάλωση ισχύος των πέντε επιλεγμένων ΔΔΜ για τις επιλεγμένες εφαρμογές. Ο ΔΔΜ 1 [61] αποτελεί τη βάση μέτρησης για όλους τους υπόλοιπους ΔΔΜ. Όπως ήταν αναμενόμενο, ο ΔΔΜ 1 έχει τη χαμηλότερη κατανάλωση ισχύος, επειδή δεν έχει σταθερές λίστες, χρησιμοποιεί first-fit αλγόριθμο τοποθέτησης και δεν διαθέτει πολιτικές διάσπασης και ένωσης μπλοκς. Ο ΔΔΜ 2 καταναλώνει κατά μέσο όρο περίπου 1,9 $\times$ περισσότερη ενέργεια σε σχέση με το ΔΔΜ 1 λόγω της ενεργοποίησης των πολιτικών διάσπασης και ένωσης μπλοκς. Οι συγκεκριμένες πολιτικές στοχεύουν στον καλύτερο κατακερματισμό του σωρού με αποτέλεσμα να έχουν αυξημένη κατανάλωση ισχύος. Ο ΔΔΜ 3 καταναλώνει κατά μέσο όρο 1,2$\times$ περισσότερη ισχύ σε σύγκριση με τον ΔΔΜ 1. Η μόνη διαφορά μεταξύ




Σχήμα 3.23: Κανονικοποιημένη κατανάλωση ισχύος για τους επιλεγμένους ΔΔΜ, χωρίς οποιονδήποτε μηχανισμό επόπτευσης ή αλλαγής DVFS




Σχήμα 3.24: Κατακερματισμος του σωρού για τους επιλεγμένους ΔΔΜ.



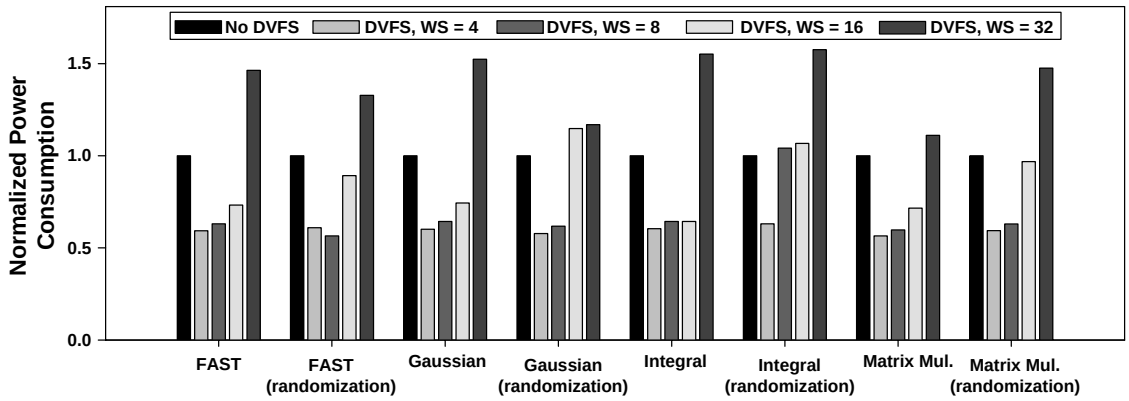

Σχήμα 3.25: Κανονικοποιημένη κατανάλωση ισχύος για τον ΔΔΜ 1 σε σύγκριση με την ενσωμάτωση των DVFS μηχανισμών για διαφορετικό μέγεθος παραθύρων ($WS$).

αυτών των δύο είναι η διαφορετική πολιτική τοποθέτησης που χρησιμοποιούν.Ο ΔΔΜ 4 καταναλώνει κατά μέσο όρο 1,7× περισσότερη ενέργεια σε σχέση με το ΔΔΜ 1. Όπως προαναφέρθηκε οι πολιτικές διάσπασης και ένωσης μπλοκς είναι ο κύριος λόγος για την υψηλή κατανάλωση ισχύος. Τέλος, ο ΔΔΜ 5 είναι ο χειρότερος από όλους έχοντας κατά μέσο όρο 4,6× μεγαλύτερη κατανάλωση ισχύος σε σύγκριση με τον ΔΔΜ 1. Αυτό συμβαίνει επειδή ο ΔΔΜ αναζητά σε λίστες σταθερού μεγέθους τα κατάλληλα μπλόκς.

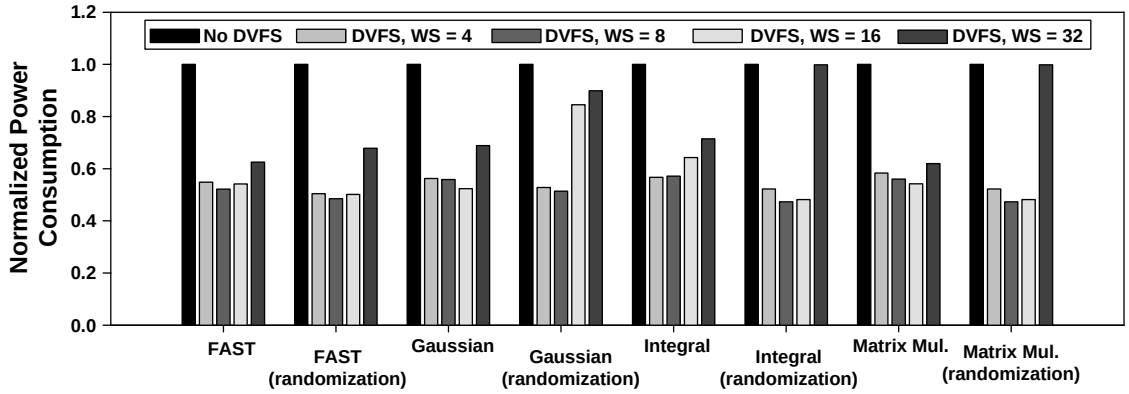
Επιπλέον, το Σχήμα 3.24 παρουσιάζει τον κατακερματισμό του σωρό για τους επιλεγμένους ΔΔΜ. Όπως προαναφέρθηκε, ο κατακερματισμός σωρού είναι η μέγιστη ποσότητα μνήμης που εκχωρείται από το σύστημα διαιρούμενη με το μέγιστο ποσό μνήμης που απαιτείται από την εφαρμογή. Οι ΔΔΜ 1, 2, 3, 4 και 5 έχουν κατά μέσο όρο 10%, 2,3%, 2,3%, 2,3% και 17% κατακερματισμό του σωρού αντίστοιχα. Όπως ήταν αναμενόμενο, ο ΔΔΜ 1 έχει το μεγαλύτερο ποσοστο λόγω της πολιτικής ταιριάσματος first-fit. Οι ΔΔΜ 2, 3 και 4 έχουν τις χαμηλότερες τιμές κατακερματισμού διότι τόσο η πολιτική ταιριάσματος (best-fit) όσο και οι οι πολιτικές διάσπασης και ένωσης μπλοκς συμβάλλουν στο χαμηλό κατακερματισμό του σωρού. Τέλος, ο ΔΔΜ 5 έχει το χειρότερο κατακερματισμό σωρού γεγονός που οφείλεται στο ότι χειρίζεται συγκεκριμένα μεγέθη μπλοκ, πράγμα το οποίο δημιουργεί τρύπες στο σωρό.

#### 3.5.3.2 Κατανάλωση ισχύος

Τα Σχήματα 3.25-3.29 παρουσιάζουν την κανονικοποιημένη κατανάλωση ισχύος για τους πέντε επιλεγμένους ΔΔΜ αντίστοιχα, με την ενσωμάτωση του προτεινόμενου πλαισίου. Για κάθε ΔΔΜ εξάγουμε την κατανάλωση ισχύος όταν (i) δεν υπάρχει κανένα σύστημα ρύθμισης συχνότητας και τάσης (no DVFS mechanism) και (ii) όταν υπάρχει μηχανισμός ρύθμισης της συχνότητας και της τάσης με διαφορετικό μέγεθος παραθύρου ($WS$) (απο $WS = 4$ έως $WS = 32$).

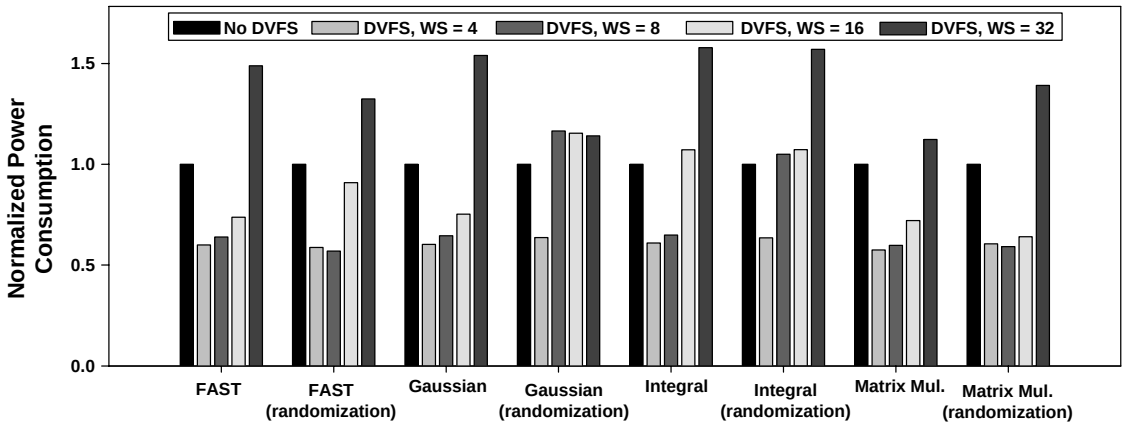



Σχήμα 3.26: Κανονικοποιημένη κατανάλωση ισχύος για τον ΔΔΜ 2 σε σύγκριση με την ενσωμάτωση των DVFS μηχανισμών για διαφορετικό μέγεθος παραθύρων ($WS$).




Σχήμα 3.27: Κανονικοποιημένη κατανάλωση ισχύος για τον ΔΔΜ 3 σε σύγκριση με την ενσωμάτωση των DVFS μηχανισμών για διαφορετικό μέγεθος παραθύρων ($WS$).

Τα πειραματικά αποτελέσματα δείχνουν ότι με τη χρήση της προτεινόμενης μεθόδου για την παρακολούθηση και την εφαρμογή DVFS, η κατανάλωση ισχύος που αφορά τη διαχείριση του σωρού μειώθηκε κατά 37% περίπου. Συγκεκριμένα, όταν το μέγεθος του παραθύρου είναι μικρό (π.χ. $WS = 4$) το κέρδος είναι ακόμα μεγαλύτερο φθάνοντας περίπου στο 45% σε σύγκριση με την περίπτωση όπου δεν υπάρχει παρών κάποιος μηχανισμός DVFS. Ωστόσο, όταν το μέγεθος του παραθύρου αυξάνει, το κέρδος στην κατανάλωση ισχύος μειώνεται και ποικίλλει από 33% ($WS = 8$) σε 21% ($WS = 16$) σε σύγκριση με την περίπτωση όπου δεν εφαρμόζεται μηχανισμός DVFS. Στην περίπτωση όπου το μέγεθος του παραθύρου είναι ίσο με 32 ($WS = 32$) υπάρχει γενικά μεγαλύτερη κατανάλωση ισχύος με περίπου 16%. Αυτό μπορεί να εξηγηθεί από το γεγονός ότι όταν αυξάνεται το μέγεθος του παραθύρου, οι μηχανισμοί απόφασεις DVFS δεν μπορούν να λάβουν τη σωστή απόφαση τη σωστή χρονική στιγμή καθώς οι μηχανισμοί επόπτευσης δεν προωθούν τα σωστα δεδομένα.

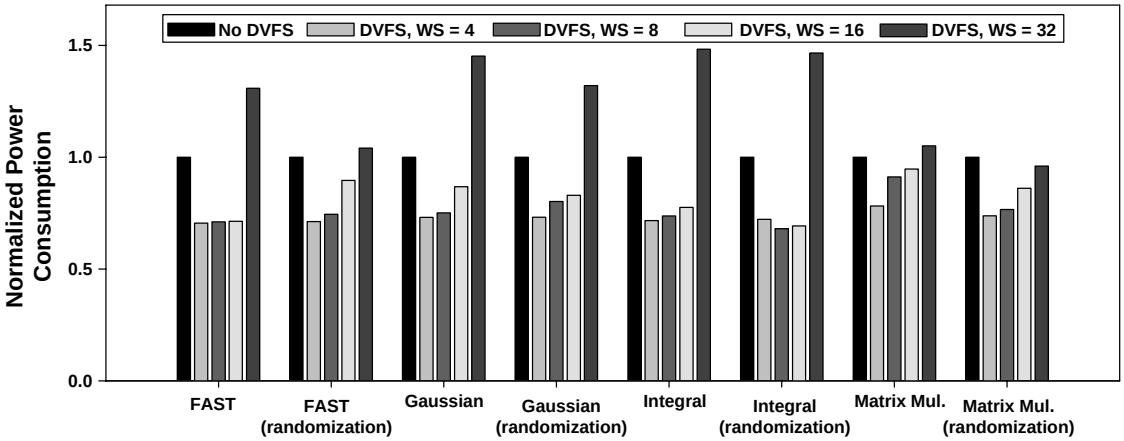



Σχήμα 3.28: Κανονικοποιημένη κατανάλωση ισχύος για τον ΔΔΜ 4 σε σύγκριση με την ενσωμάτωση των DVFS μηχανισμών για διαφορετικό μέγεθος παραθύρων ($WS$).

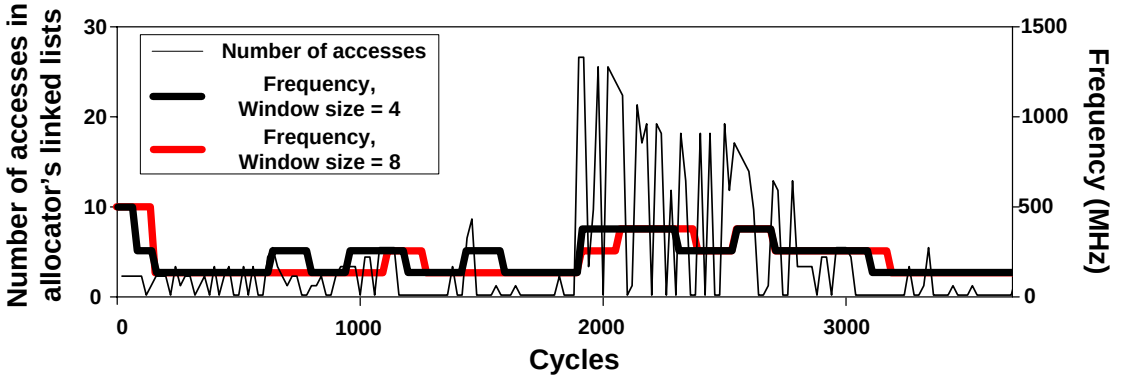
Το Σχήμα 3.25 παρουσιάζει την κανονικοποιημένη κατανάλωση ενέργειας για τον ΔΔΜ 1 με την ενσωμάτωση του προτεινόμενου πλαισίου. Με τη χρήση του προτεινόμενου πλαισίου, η κατανάλωση ισχύος μπορεί να μειωθεί μέχρι και 50% ($WS = 4$) σε σύγκριση με την περίπτωση όπου κανένας μηχανισμός DVFS δεν εφαρμόζεται. Όπως προαναφέρθηκε, όταν αυξάνεται το μέγεθος του παραθύρου, τα κέρδη στην κατανάλωση ισχύος μειώνονται και σε ορισμένες περιπτώσεις, όταν $WS = 32$, υπάρχει μία μικρη αυξηση. Στο Σχήμα 3.23 ο ΔΔΜ 1 απεδείχθη ότι είναι ο πιο κερδοφόρος στην κατανάλωση ισχυος και επομένως βελτιώνουμε περαιτέρω αυτή την ιδιότητα.

Τα Σχήματα 3.26-3.28 παρουσιάζουν την κανονικοποιημένη κατανάλωση ισχύος για τους ΔΔΜ 2, 3 και 4, αντίστοιχα, με την ενσωμάτωση του προτεινόμενου πλαισίου. Ο μέσος όρος μείωσης της καταναλισκώμενης ισχύος είναι 48% και μειώνεται όσο αυξάνεται το μέγεθος του παραθύρου (σε σύγκριση με την περίπτωση όπου δεν υπάρχει μηχανισμός DVFS). Στο Σχήμα 3.24, οι ΔΔΜ 2, 3 και 4 αποδείχθηκαν ότι είναι οι πιο αποτελεσματικοί από άποψη κατακερματισμού του σωρού. Με την εφαρμογή της προτεινόμενης μεθοδολογιας καταφέραμε να πετύχουμε ΔΔΜ με χαμηλή κατανάλωση ισχύος και χαμηλό κατακερματισμό του σωρού. Τέλος, το Σχήμα 3.29 παρουσιάζει την κανονικοποιημένη κατανάλωση ισχύος του ΔΔΜ 5 με την ενσωμάτωση του προτεινόμενου πλαισίου. Και σε αυτή την περίπτωση, όταν το μέγεθος του παραθύρου είναι μικρό υπάρχει σημαντικό κέρδος στην κατανάλωση ισχύος.

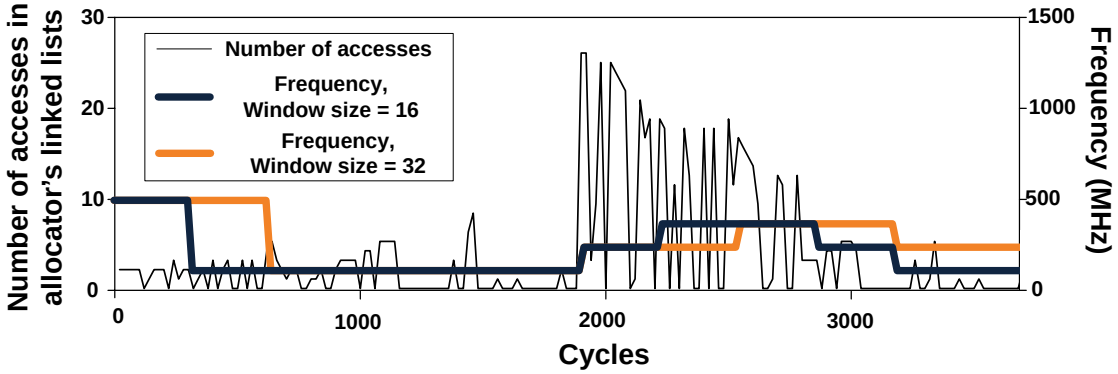
Τα Σχήματα 3.30 και 3.31 δείχνουν τον αριθμό των προσβάσεων στις συνδεδεμένες λίστες των ΔΔΜ, σε σύγκριση με την απόφαση του διαχειριστή για DVFS αλλαγές. Όπως ήταν αναμενόμενο, όταν το μέγεθος του παραθύρου είναι μικρό, οι μηχανισμοί επόπτευσης μπορούν να ακολουθήσουν τις ανάγκες της εφαρμογής και να ενεργοποιήσουν σωστά τους κατάλληλους μηχανισμούς DVFS. Από την άλλη πλευρά, όταν το μέγεθος του παραθύρου γίνεται μεγάλο, οι μηχανισμοί επόπτευσης δυσκολεύονται να ακολουθήσουν τις ανάγκες της εφαρμογής και παίρνονται λανθασμένες αποφάσεις σε λάθος στιγμή.




Σχήμα 3.29: Κανονικοποιημένη κατανάλωση ισχύος για τον ΔΔΜ 5 σε σύγκριση με την ενσωμάτωση των DVFS μηχανισμών για διαφορετικό μέγεθος παραθύρων ($WS$).




Σχήμα 3.30: Αριθμός των προσβάσεων στις συνδεδεμένες λίστες του ΔΔΜ 2 για την εφαρμογή FAST σε σύγκριση με την απόφαση του διαχειριστή για DVFS αλλαγές για $WS = 4$ και $WS = 8$




Σχήμα 3.31: Αριθμός των προσβάσεων στις συνδεδεμένες λίστες του ΔΔΜ 2 για την εφαρμογή FAST σε σύγκριση με την απόφαση του διαχειριστή για DVFS αλλαγές για $WS = 16$ κα $WS = 32$

#### 3.5.3.3 Αντίκτυπο στην απόδοση

Μέχρι τώρα, δείξαμε ότι η προτεινόμενη μέθοδος μπορεί να επιτύχει μείωση της ισχύος έως 37% σε σύγκριση με την περίπτωση όπου δεν εφαρμόζεται κάποιος μηχανισμός DVFS. Επίσης, δείξαμε ότι όταν το μέγεθος του παραθύρου είναι μικρό οι μηχανισμοί επόπτευσης μπορεί να ακολουθήσουν τις ανάγκες της εφαρμογής και να ενεργοποιήσουν τους κατάλληλους μηχανισμούς DVFS όταν χρειάζεται. Ωστόσο, η συνεχής παρακολούθηση των μηχανισμών επόπτευσης έχει ένα αντίκτυπο στην απόδοση. Στον Πίνακα 3.3 παρουσιάζεται η επιβάρυνση στην απόδοση όσον αφορά τους επεξεργαστικούς κύκλους, για καθε μία εφαρμογή. Ιδιαίτερα, η επιβάρυνση στην απόδοση έχει χωριστεί σε δύο μέρη ανάλογα με το αν υπήρχε απόφαση για αλλαγή DVFS ή όχι: (i) η επιβάρυνση προστίθεται μόνο από τους μηχανισμούς επόπτευσης στις περιπτώσεις όπου δεν υπάρχει κάποια αλλαγή DVFS (Idle) και (ii) η επιβάρυνση προστίθεται από τους μηχανισμούς επόπτευσης, συμπεριλαμβανομένων των DVFS αλλαγών (DVFS).

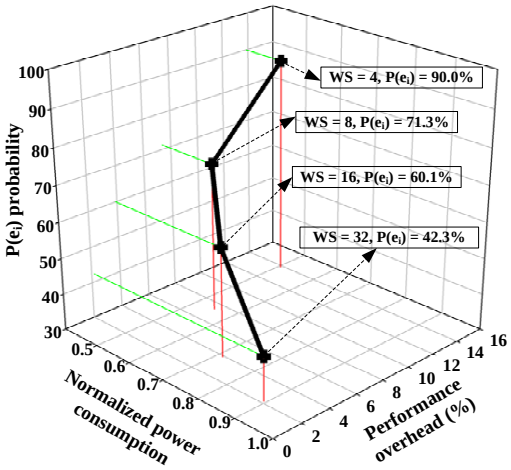
Όπως ήταν αναμενόμενο, όταν το μέγεθος του παραθύρου είναι μικρό η επιβάρυνση στην απόδοση αυξάνεται κατά μέσο όρο με 14% περισσότερους κύκλους εκτέλεσης γεγονός που οφείλεται στο ότι οι μηχανισμοί επόπτευσης ενεργοποιούνται πιο συχνά. Από την άλλη πλευρά, όταν αυξάνει το μέγεθος του παραθύρου, οι μηχανισμοί επόπτευσης δεν ενεργοποιούνται τόσο συχνά και γενικά η επιβάρυνση μπορει να θεωρηθεί αμελητέα σε ορισμένες περιπτώσεις (κάτω από 5%). Μια άλλη αξιοσημείωτη παρατήρηση είναι το γεγονός ότι, όταν το μέγεθος του παραθύρου αυξάνει, η επιβάρυνση των αλλαγων λόγω DVFS παραμένει σχεδόν η ίδια. Αυτό εξηγείται από το γεγονός ότι σε αυτές τις περιπτώσεις υπάρχει εντολή αλλαγής DVFS κάθε φορά που ένα παράθυρο παρατήρησης τελειώνει.

Πίνακας 3.3: Επιβάρυνση στην απόδοση του συστήματος σε επεξεργαστικούς κύκλους

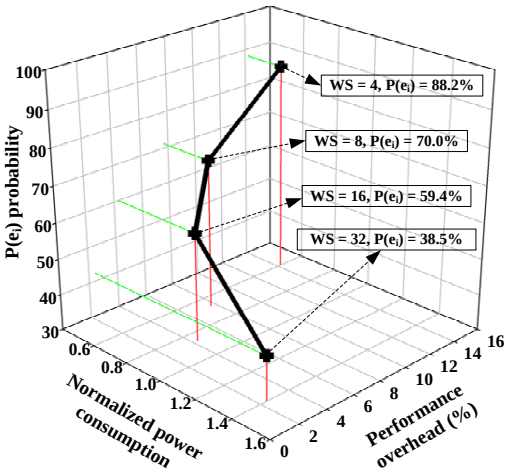
| **Window size** | **Cycles** | **FAST** | **FAST (random.)** | **Gaussian** | **Gaussian (random.)** | **Integral** | **Integral (random.)** | **Matrix Mul.** | **Matrix Mul. (random.)** |
|---|---|---|---|---|---|---|---|---|---|
| **4** | Idle | 10.16% | 9.45% | 11.52% | 8.26% | 12.02% | 5.47% | 6.12% | 5.99% |
| | DVFS | 3.56% | 4.74% | 3.63% | 9.38% | 3.79% | 10.51% | 2.36% | 3.79% |
| | **Total** | **13.72%** | **14.19%** | **15.15%** | **17.65%** | **15.82%** | **15.98%** | **8.48%** | **9.78%** |
| **8** | Idle | 3.88% | 3.88% | 4.27% | 4.51% | 4.59% | 3.92% | 2.32% | 2.63% |
| | DVFS | 3.52% | 3.52% | 3.39% | 4.64% | 4.17% | 4.56% | 2.39% | 2.71% |
| | **Total** | **7.40%** | **7.40%** | **7.66%** | **9.14%** | **8.76%** | **8.49%** | **4.72%** | **5.35%** |
| **16** | Idle | 1.10% | 1.42% | 1.75% | 2.58% | 1.55% | 1.67% | 0.69% | 1.07% |
| | DVFS | 2.62% | 2.10% | 2.59% | 2.34% | 2.90% | 3.12% | 1.63% | 2.01% |
| | **Total** | **3.73%** | **3.52%** | **4.34%** | **4.91%** | **4.45%** | **4.78%** | **2.31%** | **3.08%** |
| **32** | Idle | 0.16% | 0.33% | 0.19% | 1.01% | 0.43% | 0.95% | 0.35% | 0.50% |
| | DVFS | 1.90% | 1.63% | 2.17% | 1.69% | 2.14% | 1.57% | 1.40% | 1.83% |
| | **Total** | **2.06%** | **1.95%** | **2.36%** | **2.70%** | **2.56%** | **2.52%** | **1.75%** | **2.33%** |

#### 3.5.3.4 Ισορροπία μεταξύ κατανάλωσης ισχύος και επιβάρυνσης απόδοσης

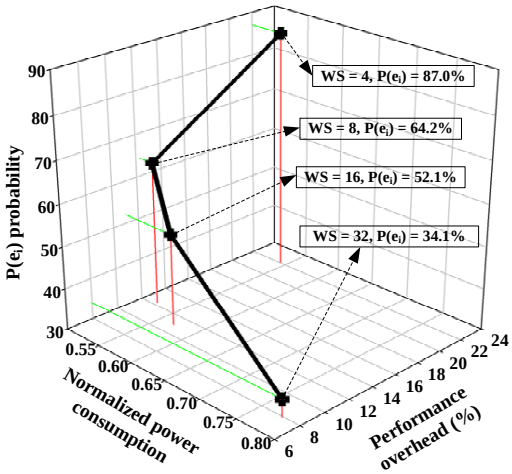
Προκειμένου να αξιολογηθεί η ισορροπία μεταξύ της μείωσης της κατανάλωσης ισχύος, της επιβάρυνσης στην απόδοση και το μέγεθος του παραθύρου, ορίζουμε ως $P(e_i)$ την πιθανότητα το $e_i$ γεγονός να εξυπηρετηθεί στη σωστή συχνότητα και τάση τροφοδοσίας. Το Σχήμα 3.32 δείχνει το $P(e_i)$, για τους επιλεγμένους ΔΔΜ (Σχήματα 3.32a-e), σε σύγκριση με την κανονικοποιημένη κατανάλωση ισχύος (κανένας μηχανισμός DVFS δεν εφαρμόζεται), επιβάρυνση στην απόδοση και μέγεθος παραθύρου ($WS$). Όπως ήταν αναμενόμενο, όταν το μέγεθος του παραθύρου είναι μικρό ($WS = 4$) το $P(e_i)$ είναι κατά μέσο όρο 90,1%, ενώ όταν το μέγεθος του παραθύρου είναι μεγάλο ($WS = 32$) το ποσοστό αυτό πέφτει στο 43,3%. Τα διαφορετικά χαρακτηριστικά των ΔΔΜ κατανεμητών επηρεάζουν τη συμπεριφορά τους και αυτό αντικατοπτρίζεται στις διάφορες τιμές του $P(e_i)$ για το ίδιο το μέγεθος παραθύρου.



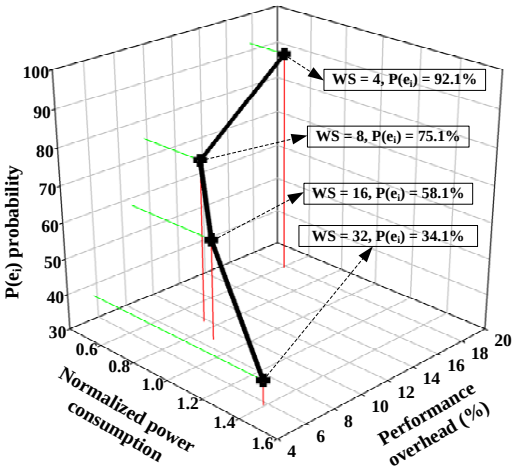

**a) DM Manager 1**



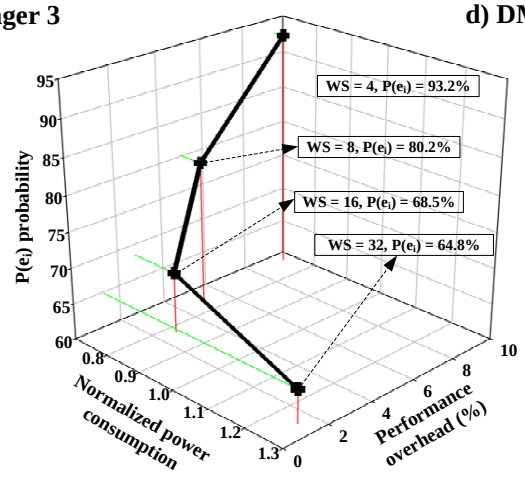

**b) DM Manager 2**




**c) DM Manager 3**




**d) DM Manager 4**




**e) DM Manager 5**

Σχήμα 3.32: $P(e_i)$ τιμές, για όλους τους ΔΔΜ, σε σύγκριση με την κανονικοποιημένη κατανάλωση ισχύος (κανένας μηχανισμός DVFS δεν εφαρμόζεται), επιβάρυνση στην απόδοση και μέγεθος παραθύρου ($WS$).

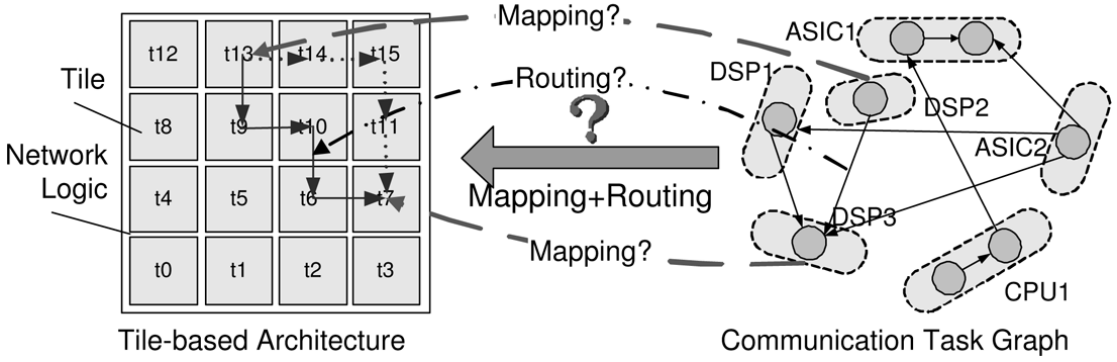
# Κεφάλαιο 4

# Κατανεμημένη διαχείριση πόρων κατά τη φάση εκτέλεσης

## 4.1 Εισαγωγή

Η έννοια των πολυπύρηνων αρχιτεκτονικών θέτει ένα νέο πρόβλημα για τον σχεδιαστή υψηλού επιπέδου, το οποίο αναφέρεται στην κατανομή των πόρων μεταξύ των πυρήνων. Μέχρι τώρα, υπήρχε το πρόβλημα του προγραμματισμού των εργασιών σε έναν πυρήνα ή σε ένα πολύ μικρό σύνολο πυρήνων. Σε μια πολυπύρηνη πλατφόρμα, η κατανομή των πόρων έχει επίσης και μια χωρική διάσταση, σε ποιόν πυρήνα θα ανατεθεί-απεικονιστεί το κάθε έργο λαμβάνοντας υπόψη και τη θέση του επιλεγμένου πυρήνα. Αυτός είναι ο λόγος για τον οποίο αναφερόμαστε σε αυτή τη διαδικασία σαν απεικόνιση (mapping). Ένα παράδειγμα διαδικασίας απεικόνισης παρουσιάζεται στο Σχήμα 4.1.

Πολύ σημαντικό στοιχείο αποτελεί ο υπολογισμός μιας αποδοτικής απεικόνισης για μία δεδομένη εφαρμογή σε ένα σύντομο χρονικό διάστημα. Ο σχεδιασμός μπορεί να έχει πολλές πτυχές. Στόχοι μπορει να είναι η μεγιστοποίηση της απόδοσης, η ελαχιστοποίηση της απόστασης μεταξύ των εργασιών ή η ελαχιστοποίηση της κατανάλωσης ενέργειας σε περίπτωση ετερογενών αρχιτεκτονικών. Δεδομένου ότι η διαδικασία απεικόνισης πραγματοποιείται πριν την την έναρξη της εφαρμογης, η διαδικασία πρέπει να είναι όσο το δυνατόν γρηγορότερη.



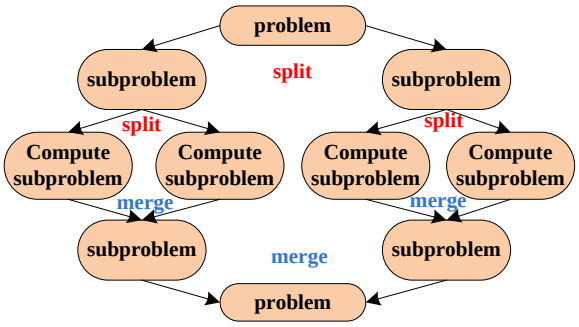

Σχήμα 4.1: Παρουσίαση τους προβλήματος απεικόνισης [50]

Η διαχείριση πόρων είναι μια βασική λειτουργία για την επιτυχή χρήση των υπολογιστικών πλατφορμών. Το μοντέλο διαχείρισης πόρων κατά τη φάση εκτέλεσης έχει γίνει εμφανές πρόσφατα, επειδή μπορεί να ασχοληθεί με τη δυναμική συμπεριφορά των εφαρμογών και των πλατφορμών. Έτσι, η αποτελεσματική απεικόνιση των εφαρμογών κατά τη φάση εκτέλεσης επιτρέπει την αποτελεσματική χρήση των πόρων, ελαχιστοποιώντας το χρόνο απεικόνισης, το φορτίο της επικοινωνίας του δικτύου και την κατανάλωση της ενέργειας.

Στην περίπτωση της απεικόνισης κατα τη φάση σχεδίασης, οι αποφάσεις εξερευνούν όλες τις δυνατές λύσεις. Ωστόσο, η απόφαση αυτή δεν μπορεί να τροποποιηθεί κατά την διάρκεια εκτέλεσης και σε πολλές περιπτώσεις απαιτείται πολύς χρόνος για να υπολογισθεί εκ νέου. Έτσι, αυτή η τεχνική μπορεί να χρησιμοποιηθεί μόνο για λίγες εφαρμογές και μόνο για συγκεκριμένα σενάρια. Προφανώς αυτό δεν αποτελεί περίπτωση στα σύγχρονα συστήματα.

Οι υπάρχουσες προσεγγίσεις για αλγοριθμους απεικόνισης σε πολυπύνες πλατφόρμες, είναι συνήθως κεντρικοποιημένες [27]. Παραδοσιακά, ένας κεντρικός πυρήνας αναλύει περιοδικά τις διαθέσιμες πληροφορίες και υπολογίζει μια γενική διαμόρφωση για ολόκληρη την πλατφόρμα. Ωστόσο, μια τέτοια κεντρικοποιημένη προσέγγιση έχει αρκετά μειονεκτήματα. Πρώτον, δημιουργεί ένα κεντρικό σημείο αποτυχίας, το οποίο καθιστά το όλο σύστημα άχρηστο όταν ο κεντρικός πυρήνας καταρρεύσει [18]. Δεύτερον, ένας κεντρικός πυρήνας δεν είναι επεκτάσιμος, και αποτελεί εμπόδιο για την επεξεργασία και τις λειτουργίες επικοινωνίας, ειδικά σε περιβάλλοντα που απαιτούνται συχνές αλλαγές [71]. Τέλος, οι κεντρικοποιημένοι διαχειριστές δεν έχουν την έννοια της αυτο-προσαρμογής και αυτο-οργάνωσης, ενέργειες που αποτελούν λύση στις σύγχρονες πλατφόρμες [55].

Από την άλλη πλευρά, στο κατανεμημένο σύστημα απεικόνισης, ο υπολογισμός κατανέμεται σε πολλούς πυρήνες μέσα στο ολοκληρωμένο [84]. Με τον τρόπο αυτό, τα προβλήματα της κεντρικοποιημένης απεικόνισης λύνονται ως εξής: (i) η παρακολούθηση της κυκλοφορίας μειώνεται σε όγκο. Τα στοιχεία επεξεργασίας χρειάζεται να στείλουν δεδομένα μόνο στον πλησιέστερο υπεύθυνο πυρήνα, μειώνοντας το κόστος διασύνδεσης, (ii) οι κατανεμημένοι διαχειριστές χρειάζεται να εκτελέσουν τον αλγόριθμο απεικόνισης μόνο για την ειδική περιοχή που ελέγχουν. Με τον τρόπο αυτό, ο απαιτητικός υπολογισμός διαιρείται σε μικρότερα ίδια προβλήματα, (iii) δεν υπάρχει ένας κεντρικός πυρήνας ή ενιαίο σημείο αποτυχίας, δεδομένου ότι ο αλγόριθμος μπορει να εκτελεστεί σε οποιοδήποτε πυρήνα και (iv) η κατανεμημένη απεικόνιση κλιμακώνεται πολύ καλά με μεγάλες πολυπύρηνες πλατφόρμες, δεδομένου ότι το μόνο που χρειάζεται είναι κατανεμημένα στελέχη, των οποίων η ατομική προσπάθεια υπολογισμού δεν αυξάνεται.



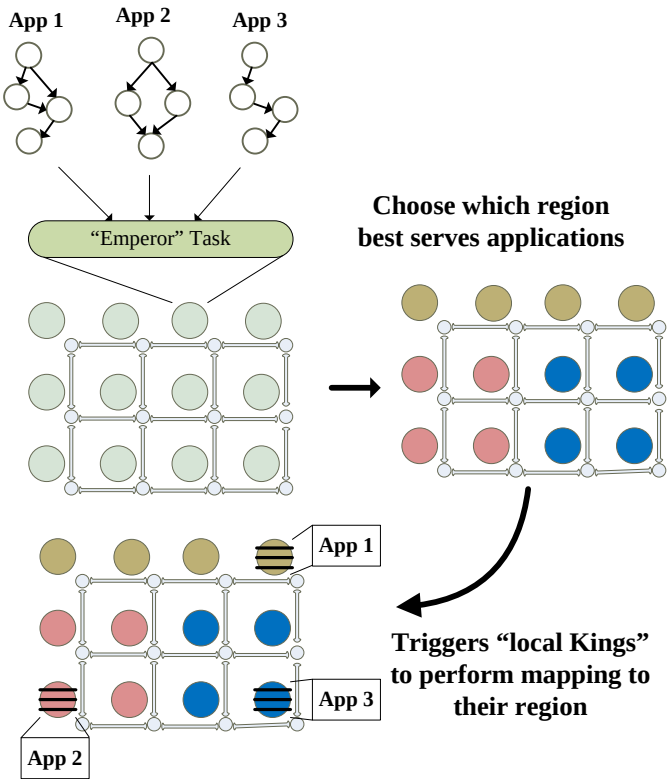

Σχήμα 4.2: Διαίρει και Βασίλευε στην επιστήμη των υπολογιστών

## 4.2 Διαίρει και Βασίλευε κατανεμημένη απεικόνιση για πολυπύρηνες πλατφόρμες

Διαίρει και Βασίλευε (Divide-and-Conquer, D&C) είναι μια στρατηγική που στόχο έχει να διατηρήσει τον έλεγχο με τη διάλυση μεγαλύτερων συγκεντρώσεων σε μικρότερες. Ιστορικά, έχει χρησιμοποιηθεί από μεγάλες αυτοκρατορίες, όπως η αρχαία Ρώμη, διαιρώντας τη γη σε μικρότερες περιοχές και τον καθορισμό των τοπικών αρχόντων. Στην επιστήμη των υπολογιστών, η D&C στρατηγική συνίσταται στο σπάσιμο ενός προβλήματος σε απλούστερα υποπροβλήματα του ίδιου τύπου. Κατοπιν, ακολουθεί η επίλυση αυτών των επιμέρους προβλημάτων και τέλος τα αποτελέσματα συγχωνεύονται σε μια λύση στο αρχικό πρόβλημα. Ως εκ τούτου, αποτελεί κατά κύριο λόγο μια επαναληπτική μέθοδο για την εξεύρεση λύσεων σε ένα μεγάλο πρόβλημα, όπως απεικονίζεται στο Σχήμα 4.2.

Η προτεινόμενη μεθοδολογία υιοθετεί την ιδέα του D&C και το χρησιμοποιεί ώστε να εκτελέσει *κατανεμημένη απεικόνιση κατά τη φάση εκτέλεσης* και σε ομογενείς και σε ετερογενείς πολυπύρηνες πλατφόρμες. Ένα απλουστευμένο παράδειγμα της προτεινόμενης προσέγγισης D&C παρουσιάζεται στο Σχήμα 4.3 [10].

Η χαρτογράφηση διεξάγεται με κατανεμημένο τρόπο. Για να επιτευχθεί αυτό, η πλατφόρμα χωρίζεται σε περιοχές, δηλαδή υποσύνολα του συνόλου όλων των πυρήνων. Αυτές οι περιοχές δεν έχουν σταθερά όρια, και μπορούν να αναδιαμορφωθούν, να δημιουργηθούν ή να καταργηθούν όταν είναι απαραίτητο. Κάθε νέα αίτηση απεικόνισης εξετάζεται κατ 'αρχάς από έναν καθορισμένο πυρηνα, όπου εκτελείται η διεργασία System-Wide Controller. Αυτή η διεργασία είναι ένα ελαφρύ κομμάτι κώδικα, που μπορεί να εφαρμοσεί σε κάθε τύπο επεξεργαστή στις πολυπύρηνες πλατφόρμες, προκειμένου να διατηρηθεί το σύστημα προστατευμένο από ένα πιθανό μοναδικό σημείο αποτυχιας. Σκοπός αυτής της διεργασίας είναι να βρει μια περιοχή κατάλληλη για να εκτελεστεί η νέα εφαρμογή. Ο System-Wide Controller καταγράφει πληροφοριες σχετικά με την κατάσταση της περιοχής που ελέγχει και ενεργεί κατάλληλα. Εκτός από το System-Wide ελεγκτή, υπάρχει ένας επιπρόσθετος ρόλος για κάθε περιοχη που ονομάζεται Regional Controller. Όπως υποδηλώνει το όνομα, οι Regional Controllers είναι υπεύθυνοι για κάθε ενέργεια που αφορά την απεικόνιση της εφαρμογης στην αντίστοιχη περιοχή τους. Πιο συγκεκριμένα είναι υπεύθυνοι




Σχήμα 4.3: Διαίρει και Βασίλευε σε πολυπύρηνη πλατφόρμα

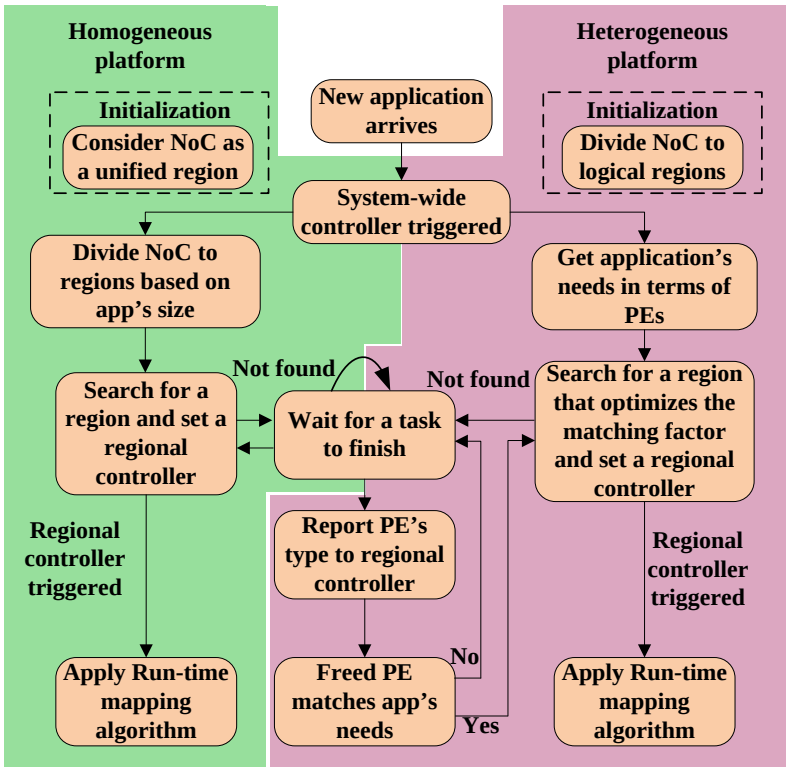
για:

- Τον υπολογισμό της απεικόνισης για την περιοχή για την οποία ο ελεγκτής είναι υπεύθυνος.
- Την συγκέντρωση των δεδομένων σε αυτη την περιοχή.
- Την επικοινωνία και ανταλλαγή δεδομένων με τον System-Wide Controller.

Με τον ίδιο τρόπο όπως ο System-Wide Controller, οι Regional Controllers μπορούν να εκτελεστούν σε οποιοδήποτε πυρηνα της περιοχής, έτσι ώστε η λειτουργικότητα της πλατφόρμας να μην εξαρτάται από ένα μόνο σημείο. Όταν μια περιοχή έχει επιλεγεί από τον System-Wide Controller για την απεικόνιση μιας νέας εφαρμογής, οι Regional Controller της περιοχής ενεργοποιούνται και τα δεδομένα που περιγράφουν την εφαρμογή αποστέλλονται σε αυτόν. Στη συνέχεια, εκτελείται η απεικόνιση και τα αποτελέσματα επιστρέφονται πίσω στο System-Wide Controller.

Οι καινοτόμες ιδέες της προτεινόμενης μεθοδολογίας είναι:

- Προτείνεται μία ευέλικτη και κατανεμημένη μεθοδολογία απεικόνισης κατά τη φαση εκτέλεσης για ομογενείς αλλά και για ετερογείς πολυπύρηνες πλατφόρμες.
- Η ευελιξία της μεθόδου βασιζεται στο γεγονός ότι μπορεί να πετύχει διαφορε-




Σχήμα 4.4: Ροή της προτεινόμενης μεθοδολογιας Διαίρει και Βασίλευε [10]

τικά επίπεδα αξιοποίησης των πόρων της πλατφόρμας ανάλογα με τις ανάγκες των εφαρμογών σε αντίθεση με άλλους αντίστοιχους κατανεμημένους αλγόριθμους [9].

- Χρησιμοποιείται μια γρήγορη διαδικασία αλλαγής κόμβων στο τελικό στάδιο του αλγορίθμου έτσι ώστε να παραχθούν ακόμα καλύτερα αποτελέσματα όσον αφορά το κόστος επικοινωνίας μέσα στο ολοκληρωμένο.

### 4.2.1 Προτεινόμενο μεθοδολογικό πλαίσιο απεικόνισης κατά τη φάση εκτέλεσης

Στόχος του προτεινόμενου μεθοδολογικού πλαισίου είναι η πραγματοποίηση απεικόνισης κατά τη φάση εκτέλεσης, εφαρμογών που περιγράφονται απο γράφους, με κριτήριο την ελαχιστοποίηση του εύρους ζώνης για επικοινωνία μέσα στο ολοκληρωμένο. Μια επισκόπηση του πλαισίου μεθοδολογίας παρουσιάζεται στο Σχήμα 4.4. Μόλις εισέλθει μία νέα εφαρμογή, ο System-Wide Controller ενεργοποιείται και σύμφωνα με το προτυπο της πλατφόρμας εκτελείται η ροή για ομογενείς ή ετερογενείς αρχιτεκονικές.

#### 4.2.1.1 Ορισμοί

Ένας γράφος (Application Task Graph, ATG) χρησιμοποιείται για να περιγράψει την ροή της επικοινωνίας. Ο ATG $G(T, D)$ είναι ένας κατευθυνομενος ακυκλικός γράφος, όπους κάθε κορυφή $t_i$ ανιπροσωπεύει ένα υπολογιστικό φορτίο της εφαρμογής. $K[t]$ $\forall t_i \in T$ χαρακτηρίζει τον τύπο κλάσης κάθε εργασίας. Κάθε κατευθυνόμενη ακμή $d_{i,j} \in D$ μεταξύ των εργασιών $t_i$ και $t_j$ χαρακτηρίζει τις εξαρτήσεις στην επικοινωνία και στα δεδομένα. Κάθε $d_{i,j}$ ακολουθείται απο μία τιμή $b(d_{i,j})$, ηοποία αντιπροσωπεύει τον όγκο επικοινωνίας που ανταλλάσσονται μεταξύ των εργασιών $t_i$ και $t_j$.

Μία πολυπύρηνη τοπολογία μπορει να περιγραφεί μοναδικά απο ένα συνδεδεμένο κατευθυνόμενο γράφο $A(I, N)$. Το σύνολο των κορυφών $N$ αποτελείται απο δύο αμοιβαιώς αποκλειόμενα υποσύνολα $N_{PE}$ και $N_C$ τα οποία περιλαμβάνουν τα διαθέσιμα επεξεργαστικά στοιχεία και τα στοιχεια διασύνδεσης αντάστοιχα. $C[pe_i]$ $\forall pe_i \in N_{PE}$ ορίζει την κλάση του στοιχείου $pe_i$. Το σύνολο των ακμών $I$ περιλαμβάνει τις πληροφορίες διασύνδεσης για το σύνολο $N$.

Οι πυρήνες που έχουν απεικονιστεί ορίζουν το σύνολο $M_{PE}$. Ορίζουμε την συνάρτηση απεικόνισης $map : T \to N_{PE}$ η οποία απεικονίζει τις εργασίες της εφαρμογής (σύνολο $T$) στα διαθέσιμα επεξεργαστικά στοιχεία (σύνολο $N_{PE}$). Επίσης, ορίζουμε το σύνολο των μη απεικονισθέντων εργασιών ως $\overline{M_{PE}}$ έτσι ώστε $pe \in \overline{M_{PE}}$ αν $pe \notin M_{PE}$. Απο τον οριμό προκύπτει ότι $M_{PE} \cap \overline{M_{PE}} = \emptyset$.

Ορίζουμε το σύνολο $R$ το οποίο περιγράφει τις λογικές περιοχές της πλατφόρμας. Το σύνολο $R$ αποτελείται απο $k$ $(k \geqslant 1)$ υποσύνολα $R_1, R_2, ..., R_i, ..., R_k$ τέτοια ώστε $\bigcap_{i=1}^{k} R_i = \emptyset$ και $\bigcup_{i=1}^{k} R_i = R$. $M_{R_i}[]$ είναι μια λίστα η οποία ορίζει το ένα προς ένα αποτέλεσμα της συνάρτησης απεικόνισης $map$ στην περιοχή $R_i$. Μια περιοχή θεωρειται κατειλημμένη αν και μόνο αν $\exists pe_i \in R_i : pe_i \in M_{PE}$.

Σε μία ετερογενή πλατφόρμα, το $C[pe_i]$ ειναι διαφορετικό για κάθε επεξεργαστικό στοιχείο. Επίσης, κάθε $t_i$ μπορεί να απαιτεί μια ειδική κλάση απο επεξεργαστικά στοιχεία για να εκτελεστεί. Προκειμένου να καλυφθούν οι ανάγκες τις εφαρμογής και της πλατφόρμας και προκειμένου να έχουμε την καλύτερη απόδοση $T \to N_{PE}$, ορίζουμε $\forall t_i \in T$ την παράμετρο Matching Factor ($MF$) τέτοια ώστε:

$$1 \geq \frac{C[pe_i]}{K[t_i]} \geq MF_{t_i}, \forall pe_i \in N_{PE} \tag{4.1}$$

Το $MF$ είναι μια παράμετρος που ορίζεται απο τον σχεδιαστή και ορίζει την κλάση των επεξεξεργαστικών πάνω στα οποία μία εργασία $t_i$ μπορεί να εκτελεστεί. Το $MF$ υπονοεί πόσο καλά ταιριάζει η κλάση $C[pe_i]$ με την εργασία $t_i$ και ορίζει ένα είδος προτεραιότητας που αφορά σε ποιον πυρήνα θα πρέπει να απεικονιστεί η συγκεκριμένη εργασία. Διαφορετικές τιμές και διαφορετικές αποφάσεις για το $MF$ έχουν ως αποτέλεσμα διαφορετικές λίστες $M_{R_i}[]$. Σε μία ομογενή πλατφόρμα ισχύει $MF_{t_1} = ... = MF_{t_N}, \forall t_i \in T$ καθώς $C[pe_1] = ... = C[pe_N], \forall pe \in N_{pe}$.

#### 4.2.1.2 Ομογενείς πλατφόρμες

**Algorithm 1** Ομογενείς πλατφόρμες

```
   // Βήμα 1:
 1: If |T| ⩽ |M_PE|
 2:    define new R_i ∈ R|∀pe_i ∈ R_i, pe_i ∈ M_PE
 3:    signal(R_i)
 4:    jump(Βήμα 2)
 5: Else
 6:    wait() // for a task to release its PE
 7:    jump(Βήμα 1)
   // Βήμα 2:
 8: ∀d_{i,j} ∈ D
 9:    ∀pe_i ∈ R_i
10:      src = min{F_HOM(d_i, pe_i)} // εξίσωση 4.3
11:      dst = min{F_HOM(d_j, pe_i)}
12:    M_PE, M_{R_i}[] ← src
13:    M_PE, M_{R_i}[] ← dst
   // Βήμα 3:
14: bestCost = bwCost{M_{R_i}[]} // εξίσωση 4.4
15: ∀t_i ∈ R_i
16:    ∀t_j ∈ R_i, t_j ≠ t_i
17:      If (MD(t_i, t_j) ⩽ MAX_MANH_DST)
18:        swap(t_i, t_j)
19:        tmpCost = bwCost{M_{R_i}[]}
20:        If tmpCost < bestCost
21:          bestCost = tmpCost
22:          M_{R_i}[] ⇐ new M_{R_i}[]
23:        Else
24:          swap(t_i, t_j)
```

Το σημείο εκκίνησης της μεθοδολογίας είναι οι γράφοι $A(I, N)$ και $G(T, D)$. Ο αλγόριθμος απεικόνισης χρησιμοποιεί αυτές τις πληροφορίες και προτείνει τη λύση χωρίς να παραβιάζονται οι περιορισμοί εύρους ζώνης της πλατφόρμας και οι απαιτήσεις της εφαρμογής. Η διαδικασία απεικόνισης παρουσιάζεται στον Αλγόριθμο 1.

- Βήμα 1 (γραμμές 1-7): Στο πρώτο βημα ελέγχουμε τον συνολικό αριθμό των εργασιών $|T|$. Αν η πλατφόρμα ειναι ικανή να εξυπηρετήσει την εφαρμογή, μια καινούρια περιοχή $R_i$ δημιουργείται και το αντίστοιχο σήμα στέλνεται σε ενα πυρήνα μέσα σε αυτή την περιοχη. Αυτός ο πυρήνας παίζει τον ρόλο του Regional Controller και εκτελεί τον αλγόριθμο απεικόνισης. Αν δεν είναι δυνατόν αυτό, περιμένουμε να ολοκληρωθεί κάποια άλλη εργασια και να αποδεσμεύσει επεξεργαστικά στοιχεία. Ο Regional Controller είναι υπεύθυνος για την περιοχή η οποία είναι ίση ή μικρότερη απο την τιμή $search\_distance$, η οποία ορίζεται στην Εξίσωση 4.2.

$$search\_distance = \begin{cases} \sqrt{|T|}, & |T| < 9 \\ \sqrt{|T|} - 1, & |T| \geq 9 \end{cases} \tag{4.2}$$

- Βήμα 2 (γραμμές 8-13): Για κάθε ροή επικοινωνίας ($d_{i,j}$) στον γράφο $G$, βρίσκουμε για την πηγή ($i$) και τον προορισμό ($j$) την τιμή $min$ της Εξίσωσης 4.3

$$F_{HOM} = \sum_{j} (b(d_{i,j}) + MD_{i,j}) + \sum_{i} \sum_{j} b(d_{i,j}) \times MD_{i,j} \tag{4.3}$$

  όπου $MD_{i,j}$ είναι η απόσταση μεταξύ του $pe_i$ και and $pe_j$. Αυτή η συνάρτηση κόστους συνδυάζει το κόστος επικοινωνίας των γειτονικών κόμβων του $pe_i$ και του συνολικού κόστους πάνω στην πλατφόρμα.

- Βήμα 3 (γραμμές 14-24): Μετά την ολοκλήρωση της αρχικής απεικόνισης χρησιμοποιούμε μια επαναληπτική διαδικασία αλλαγής γειτονκών κόμβων (σχετική με αυτή που χρησιμοποιείται στο [65]) για την περαιτέρω μείωση του συνολικού κόστους επικοινωνίας. Κατά τη διάρκεια αυτής της διαδικασίας, ένα ζεύγος ήδη απεικονισθέτων εργασίων ανταλλάζει θέσεις και το συνολικό κόστος επικοινωνίας ξαναυπολογίζεται (Εξίσωση 4.4). Ο μέγιστος αριθμός των αλλαγών κατα τη διάρκεια αυτης της διαδικασίας ορίζεται απο την τιμή $MAX\_MANH\_DST$.

$$bwCost = \sum_{M_{R_i}[]} b(d_{i,j}) * (MD_{i,j}) \tag{4.4}$$

#### 4.2.1.3 Ετερογενείς πλατφόρμες

Η διαδικασία απεικόνισης σε μια ετερογενή πλατφόρμα είναι πιο περίπλοκη. Αυτό οφείλεται στο γεγονός είναι απαραίτητος ο υπολογισμός του πιο αποδοτικού τύπου επεξεργαστικού στοιχείου για κάθε εργασία της εφαρμογής, ελαχιστοποιώντας παράλληλα το κόστος επικοινωνίας. Το σημείο εκκίνησης της μεθοδολογίας για ετερογενείς πλατφόρμες είναι οι γράφοι $A(I, N)$ και $G(T, D)$ και η τιμή $MF$. Ο αλγόριθμος αναζητά περιοχές οι οποίες είναι σε θέση να εξυπηρετήσουν τις κλάσεις εργασιών της εφαρμογής. Η διαδικασία απεικόνισης παρουσιάζεται στον Αλγόριθμο 2.

- Βήμα 1 (γραμμές 1-8): Στο πρώτο βήμα, προσπαθούμε να βρούμε μια περιοχή μέσα στην οποία οι τύποι των πυρήνων ταιριάζουν απόλυτα με την κλάση των εργασιών της εφαρμογής ($\frac{C[pe_i]}{K[t_i]} = 1$). Αν υπάρχει μια τέτοια περιοχή, το αντίστοιχο σήμα στέλνεται και ορίζεται ο Regional Controller.

- Βήμα 2 (γραμμές 9-13): Αν η πρώτη προσπάθεια δεν φέρει αποτέλεσμα και οι απαιτησεις της εφαρμογής δεν είναι αυστηρές ($MF < 1$) αλλάζουμε το ταίριασμα σύμφωνα με την τιμή $MF_{t_i}$ και προσπαθούμε να βρούμε περιοχές όσο πιο κοντά γίνεται στο ζητούμενο $MF_{t_i}$ $\forall t_i \in T$.

**Algorithm 2** Heterogeneous platform

// Βήμα 1:
1: $\forall t_i \in T$
2: $\quad \forall pe_i \in N_{PE}$
3: $\quad\quad$ sort$\{MF_{t_i}\}$
4: $\forall R_i \in R$
5: $\quad$ If ($|T| \leqslant |\overline{M_{PE}}|$) && ($\forall t_i \in T$, $\exists pe_i \in \overline{M_{R_i}} : \frac{C[pe_i]}{K[t_i]} = 1$)
6: $\quad\quad$ select($R_i$)
7: $\quad\quad$ jump(Βήμα 5)
8:
// Βήμα 2:
9: $\forall R_i \in R$
10: $\quad$ If ($|T| \leqslant |\overline{M_{PE}}|$) && ($\forall t_i \in T$, $\exists pe_i \in \overline{M_{R_i}} : 1 > \frac{C[pe_i]}{K[t_i]} \geq MF_{t_i}$ )
11: $\quad\quad$ select($R_i$)
12: $\quad\quad$ jump(Βήμα 5)
13:
// Βήμα 3:
14: $\forall$ unoccupied $R_i \in R$
15: $\quad \forall pe_i \in \overline{M_{PE}}$, $pe_i \notin R_i$
16: $\quad\quad \{R_i\} = \{R_i\} + pe_i$
17: $\quad$ repeat(Steps 1-2) for $R_i$
18: $\quad$ If $R_i$ not selected
19: $\quad\quad \{R_i\} = \{R_i\} - pe_i$, restore($R_i$)
20:
// Βήμα 4:
21: define new $R_i = \emptyset \in R$
22: $\forall pe_i \in \overline{M_{PE}}$
23: $\quad \{R_i\} = \{R_i\} + pe_i$
24: repeat(Steps 1-2) for $R_i$
25: If $R_i$ not selected
26: $\quad \{R_i\} = \{R_i\} - pe_i$, restore($R_i$)
27: wait()
28: jump(Βήμα 1)
// Βήμα 5:
29: $\forall K[t_k] \in G$
30: $\quad \{S\} = d_{i,j}$ iff $(K[t_k] = K[t_i])||(K[t_k] = T[t_j])$
31: sort($S$) //by $b(d_{i,j})$
32: $\forall d_{i,j} \in S$
33: $\quad \forall pe_i \in R_i$
34: $\quad\quad src = min\{F_{HET}(d_i, pe_i)\}$ // εξίσωση 4.5
35: $\quad\quad dst = min\{F_{HET}(d_j, pe_i)\}$
36: $\quad M_{PE}, M_{R_i}[] \leftarrow src$
37: $\quad M_{PE}, M_{R_i}[] \leftarrow dst$
// Step 6: Swapping procedure
38: $bestCost = bwCost\{M_{R_i}[]\}$ // εξίσωση 4.4
39: $\forall t_i \in R_i$
40: $\quad \forall t_j \in R_i, t_j \neq t_i$
41: $\quad\quad$ If $(MD(t_i, t_j) \leqslant MAX\_MANH\_DST)$
42: $\quad\quad\quad swap(t_i, t_j)$
43: $\quad\quad\quad tmpCost = bwCost\{M_{R_i}[]\}$
44: $\quad\quad\quad$ If $tmpCost < bestCost$
45: $\quad\quad\quad\quad bestCost = tmpCost$
46: $\quad\quad\quad\quad M_{R_i}[] \Leftarrow$ new $M_{R_i}[]$
47: $\quad\quad\quad$ Else
48: $\quad\quad\quad\quad swap(t_i, t_j)$

- Βήμα 3 (γραμμές 14-20): Αν ακόμα δεν έχει βρεθεί κάποια περιοχή, ψάχνουμε για κάποιο ελεύθερο $R_i$ και προσθέτουμε σε αυτό οποιοδήποτε $pe_i \in \overline{M_{PE}}$. Αν η καινούρια περιοχή $R_i^{'}$ δεν ειναι ικανή να εξυπηρετήσει την εφαρμογή, όλα τα επεξεργαστικά στοιχεία που προσαρτήθηκαν επαναφέρονται στις αρχικές περιοχές τους.

- Βήμα 4 (γραμμές 21-28): Αν ακόμα δεν έχει βρεθεί κάποια περιοχή ή όλες οι περιοχές είναι κατειλημμένες, μία καινούρα περιοχή $R_i$ δημιουργείται και κάθε $pe_i \in \overline{M_{PE}}$ που προστίθεται σε αυτή. Και σε αυτή την περίπτωση, αν η καινούρια περιοχή $R_i$ δεν ειναι ικανή να εξυπηρετήσει την εφαρμογή, όλα τα επεξεργαστικά στοιχεία που προσαρτήθηκαν επαναφέρονται στις αρχικές περιοχές τους και περιμένουμε να τελειώσει κάποια εργασια και να ελευθερωθούν τα επεξεργαστικά στοιχεία που απασχολούσε.

- Βήμα 5 (γραμμές 29-37): Σε αυτό το βήμα, ορίζουμε το σύνολο $S$ το οποίο περιέχει όλες τις ροές $d_{i,j}$ των οποίων οι κλάσεις των πηγών ($K[t_i]$) ή των προορισμών ($K[t_j]$) είναι σύμφωνες με το $K[t_k]$. Αυτό το σύνολο ταξινομείται σύμφωνα με την τιμή $b(d_{i,j})$. Κατόπιν, για κάθε ροή επικοινωνίας ($d_{i,j}$) μέσα στο σύνολο $S$, βρίσκουμε για την πηγή ($i$) και τον προορισμό ($j$) την τιμή $min$ της Εξίσωσης 4.5 για το επιλεγμένο $pe_i$.

$$F_{HET} = F_{HOM} + Q(C[pe_i]) \tag{4.5}$$

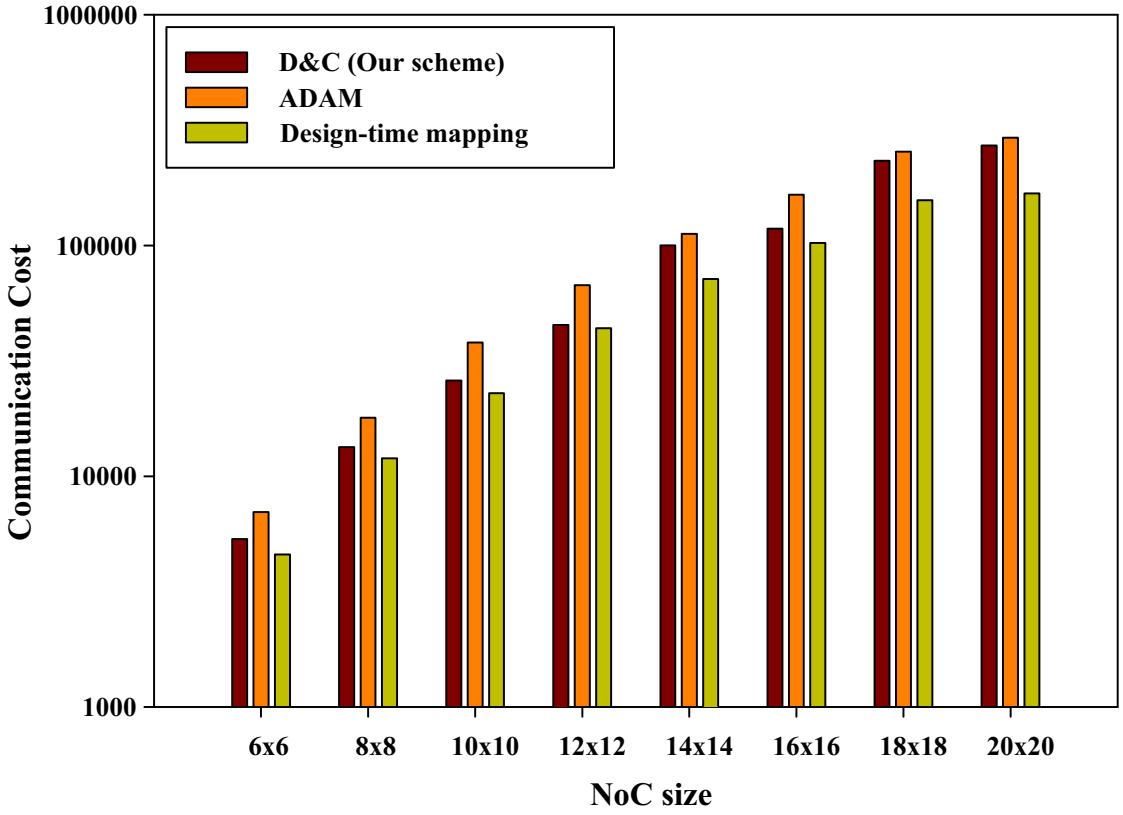
  όπου $Q(C[pe_i])$ είναι οι απαιτήσεις της κλάσης $C[pe_i]$ σχετικά με την χρησιμοποίηση εκτέλεσης και ορίζει την χρησιμοποίηση του επεξεργαστικού στοιχείου στο οποίο θα απεικονιστεί η εργασία.

- Βήμα 6 (γραμμές 38-48): Μετά την ολοκλήρωση της αρχικής απεικόνισης χρησιμοποιούμε μια επαναληπτική διαδικασία αλλαγής γειτονκών κόμβων για την περαιτέρω μείωση του συνολικού κόστους επικοινωνίας. Κατά τη διάρκεια αυτής της διαδικασίας, ένα ζεύγος ήδη απεικονισθέτων εργασίων ανταλλάζει θέσεις και το συνολικό κόστος επικοινωνίας ξαναυπολογίζεται (Εξίσωση 4.4).

### 4.2.2 Αξιολόγηση

Για να επαληθεύουμε την προσέγγισή μας, πραγματοποιήσαμε εκτεταμένες προσομοιώσεις του προτεινόμενου πλαισίου χρησιμοποιώντας τις εφαρμογές (i) MPEG-4, (ii) Multi-Window Display (MWD) [21], (iii) Picture-In-Picture (PIP) [21] (iv) MultiMedia System (MMS) [50], (v) Digitale Radio Mondiale (DRM) [82] και (vi) εφαρμογές απο TGFF [39]. Μια αρχιτεκτονική που είναι σε θέση να φιλοξενήσει έναν μεγάλο αριθμό πυρήνων, ικανοποιώντας την ανάγκη για μεγάλη ποσότητα επικοινωνίας στο ολοκληρωμένο είναι η αρχιτεκτονική Δίκτυα-σε-Ψηφίδα (Network-on-Chip, NoC).

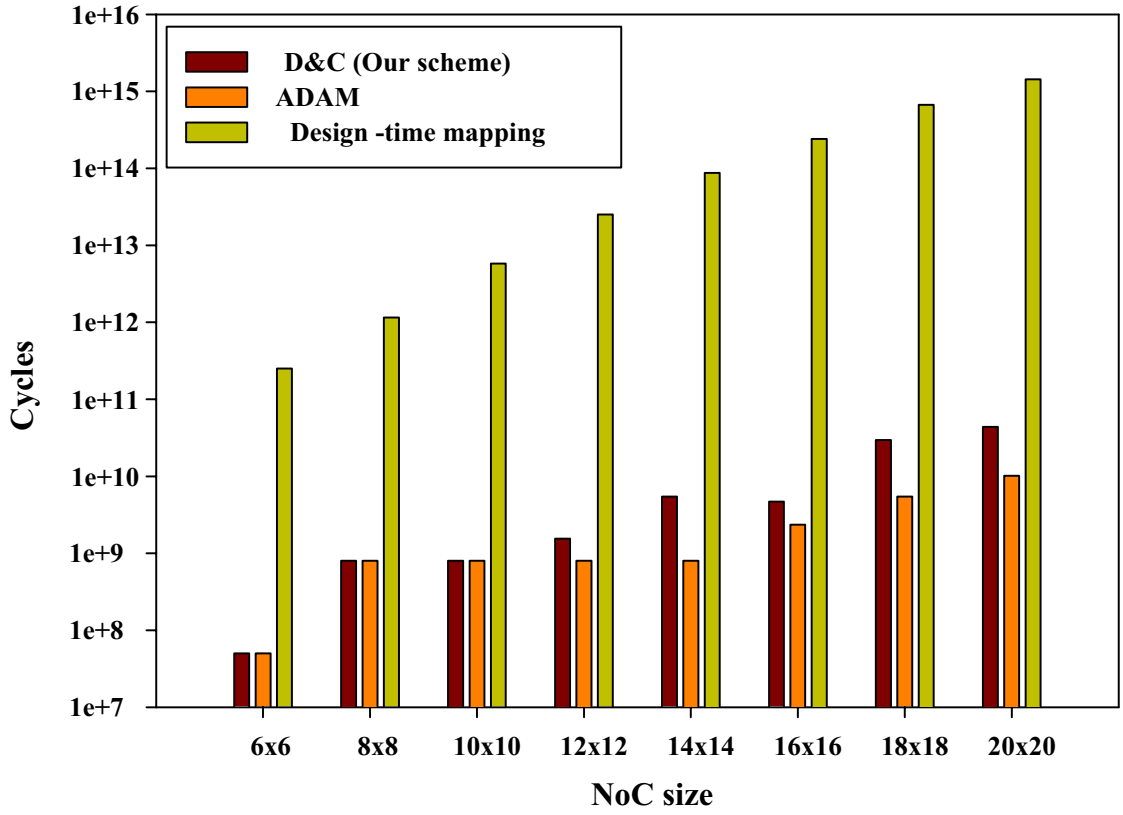
Στο Σχήμα 4.5 και 4.6 παρουσιάζεται η σύγκριση του προτεινόμενου πλαισίου, για διάφορα μεγέθη πλατφόρμας χρησιμοποιώντας γράφους εφαρμογών απο TGFF, με




Σχήμα 4.5: Σύγκριση του κόστους επικοινωνίας σε ομογενείς πλατφόρμες για ADAM [9] και απεικόνιση στη φάση σχεδίασης [50]

(i) τον κατανεμημένο αλγόριθμο απεικόνισης κατά τη φάση εκτέλεσης ADAM [9] και (ii) της εξαντλητικής εξερεύνησης λύσεων κατα τη φάση σχεδίασης [50], σε σχέση με το τελικό κόστος επικοινωνίας. Στο Σχήμα 4.5 παρουσιάζεται ότι η προτεινόμενη κατανεμημένη μεθοδολογία επιτυγχάνει 21% καλύτερο τελικό κόστος επικοινωνίας στο ολοκληρωμένο σε σύγκριση με τον επίσης κατανεμημένο αλγόριθμο κατά τη φαση εκτέλεσης ADAM [9]. Παρόλο που το βέλτιστο αποτέλεσμα επιτυγχάνεται με την εξαντλητική μέθοδο απεικόνισης κατά τη φάση σχεδίασης, η προτεινόμενη μέθοδος απαιτεί σημαντικά λιγότερους κύκλους (βελτιωμένη επεκτασιμότητα) προκειμένου να λάβουμε τις αποφάσεις απεικόνισης, όπως φαίνεται στο Σχήμα 4.6.

Για την αξιολόγηση της προτεινόμενης μεθόδου σε ετερογενείς αρχιτεκτονικές χρησιμοποιήσαμε την πλατφόρμα που παρουσιάστηκε στο Κεφάλαιο 3.2. Η πλατφόρμα αποτελείται από επεξεργαστικούς κόμβους (Prosecing Modules, PM) οι οποίοι διασυνδέονται μέσω ενός δικτύου σε ψηφίδα. Ένας επεξεργαστικός κόμβος αποτελείται από έναν LEON3 επεξεργαστή με τη δικιά του μνήμη εντολών (I-Cache) και δεδομένων (D-Cache), ένα διπύρηνο μικροπορργραμματίζομενο ελεγκτή (Dual microcoded Controller, DMC) και μνήμη η οποία μπορεί να μοιραστεί μεταξύ των κόμβων (shared memory). Οι κόμβοι μπορουν επίσης να περιέχουν μόνο μνήμη ή μονο επεξεργαστική δυνατότητα. Όποτε υπάρχει ανάγκη για τον System-Wide Controller να ενεργοποιήσει

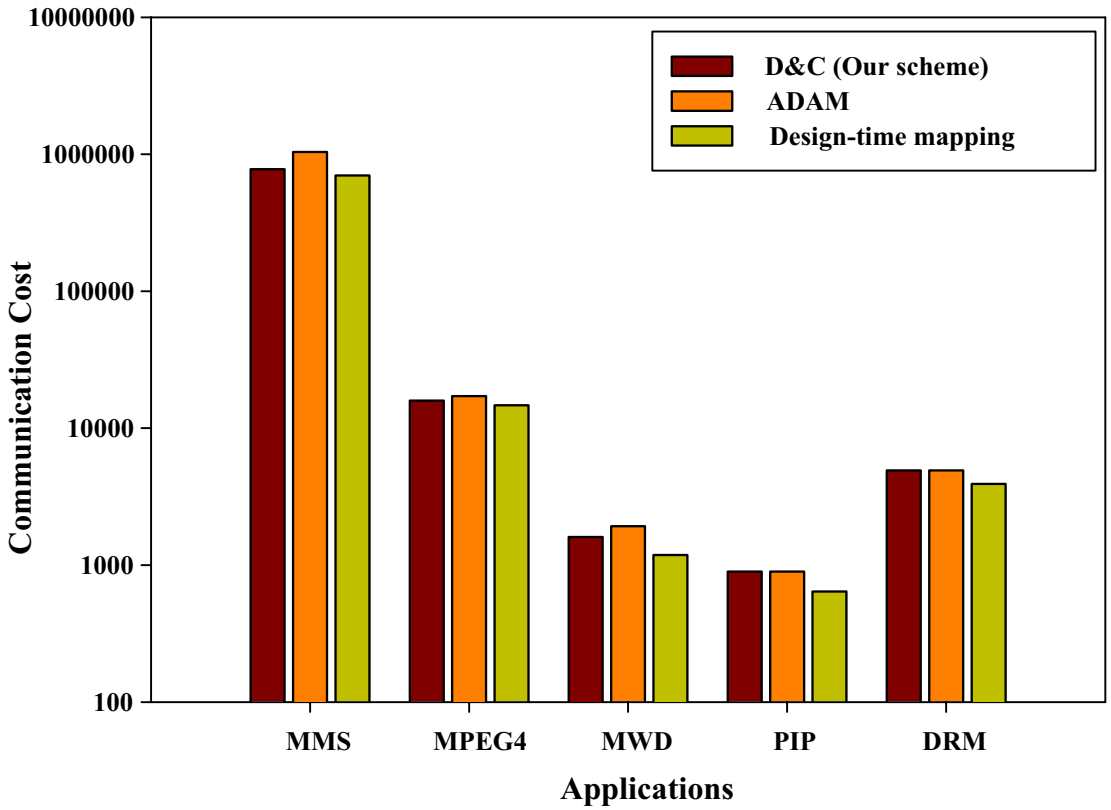



Σχήμα 4.6: Σύγκριση του κόστους υπολογισμου της απεικόνισης σε ομογενείς πλατφόρμες για ADAM [9] και απεικόνιση στη φάση σχεδίασης [50]

κάποιον άλλον πυρηνα, αυτό επιτυγχάνεται με τη χρηση τν μηχανισμών κλειδώματος που υπάρχουν στην πλατφόρμα. Το κλειδωμα επιχειρείται απο τον System-Wide Controller και τα μηνύματα προωθούνται μέσω της κοινής μνήμης.

Το Σχήμα 4.7 παρουσιάζει τη σύγκριση του κόστους επικοινωνίας για τις πέντε επιλεγμένες εφαρμογές μεταξύ του προτεινόμενου πλαισίου, του αλγορίθμου ADAM [9] και της εξαντλητικής εξερεύνησης λύσεων κατα τη φάση σχεδίασης [50]. Σύμφωνα με το Σχήμα 4.7, η προτεινόμενη μεθοδολογία έχει 10% καλύτερα αποτελέσματα όσον αφορά το κόστος επικοινωνίας πάνω στο ολοκληρωμένο σε σύγκριση με τον αλγόριθμο ADAM. Ωστόσο, το καλύτερο αποτέλεσμα επιτυγχάνεται, όπως αναμενόταν, με τον εξαντλητικό αλγόριθμο απεικόνισης κατά τη φάση σχεδίασης. Οι κύκλοι που απαιτούνται για την παραγωγή του αποτελέσματος είναι το ίδιο και για τις αλγορίθμους που αφορούν τη φάση εκτέλεσης αλλά είναι $100\times$ για την εξαντλητική μέθοδο.

Για τη δοκιμή της συμπεριφοράς της μεθοδολογίας στη φάση εκτέλεσης, χρησιμοποιήθηκαν πολλά σενάρια πάνω στην πλατφόρμα. Η διαφορά μεταξύ των διαφόρων σεναρίων είναι ο αριθμός των εφαρμογών και η ώρα άφιξης αυτών η οποία ήταν τυχαία. Πραγματοποιήθηκε σύγκριση του προενόμενου πλαισίου με τον αλγόριθμο ADAM [9]. Το Σχήμα 4.8 παρουσιάζει όλα τα υλοποιημένα σενάρια.

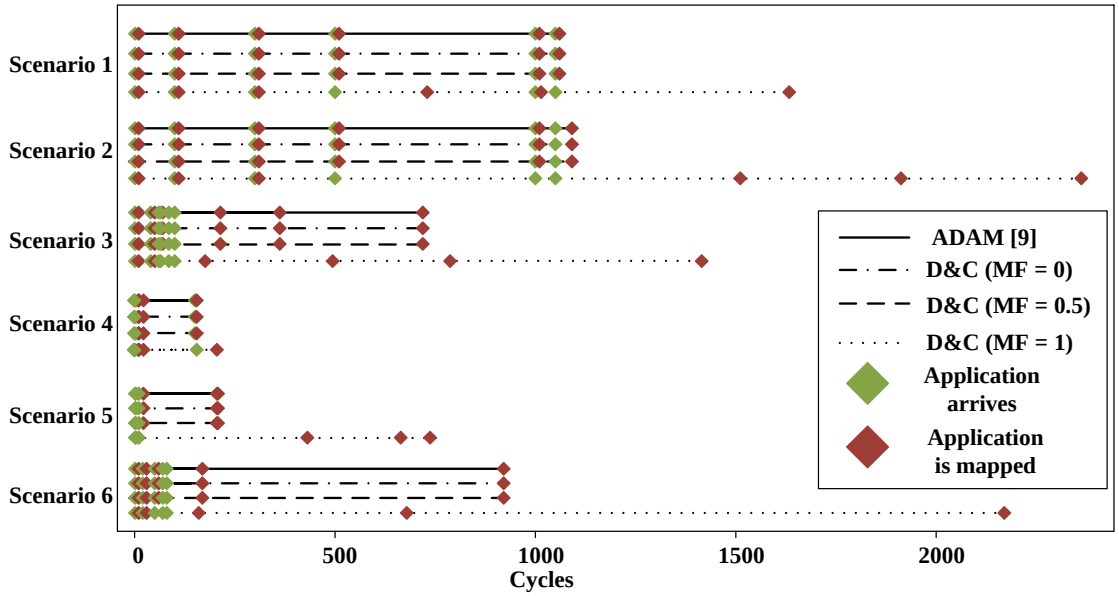



Σχήμα 4.7: Σύγκριση του κόστους επικοινωνίας για τις πέντε επιλεγμένες εφαρμογές για ADAM [9] και απεικόνιση στη φάση σχεδίασης [50]

Το πράσινο διαμάντι αντιπροσωπεύει την ώρα άφιξης της εφαρμογής, ενώ το κόκκινο αντιπροσωπεύει το χρόνο που παράχθηκε το αποτέλεσμα της απεικόνισης. Τρείς τιμές $MF$ χρησιμοποιήθηκαν για να συμβαδιζουν με τις κλάσεις της πλατφόρμας [11, 33].

Το σχήμα δείχνει ότι τοσο η προτεινόμενη μέθοδος (για $MF = 0$ και $MF = 0.5$) και ο ADAM έχουν την ίδια συμπεριφορά. Για $MF = 1$ υπάρχει διαφορετική συμπεριφορά καθώς οι εργασίες της εφαρμογής πρέπει να απεικονιστούν σε πυρήνες συγκεκριμένης κλάσης και μόνο με αποτέλεσμα να αυξάνεται ο χρόνος αναμονής. Όμως, η προτεινόμη μέθοδος έχει το καλύτερο ποσοστό χρησιμοποίησης της πλατφόρμας όπως φαίνεται στον Πίνακα 4.1. Σύμφωνα με τον Πίνακα 4.1, όταν έχουμε $MF = 1$, επιτυγχάνουμε 100% αξιοποίηση των πόρων της πλατφόρμας σε σχέση με τις απαιτήσεις τις εφαρμογής με ένα κόστος στη συνολική απόδοση.

Συνοψίζοντας, παρουσιάστηκε ένα μαθοδολογικό πλαίσιο για κατανεμημένη απεικόνιση εφαρμογών κατά τη φάση εκτέλεσης σε πολυπύρηνες πλατφόρμες, είτε ομογενείς είτε ετερογενείς. Το πλαίσιο προσαρμόζεται στις ανάγκες και στους περιορισμούς των εφαρμογών, χρησιμοποιώντας την παράμετρο $MF$ και παράγει κατά μέσο όρο 21% και 10% καλύτερο κόστος επικοινωνίας για ομοιογενείς και ετερογενείς πλατφόρμες αντίστοιχα, σε σύγκριση με τον κατανεμημένο αλγόριθμο ADAM [9]




Σχήμα 4.8: Σενάρια αξιολόγησης σε ετερογενή πλατφόρμα

Πίνακας 4.1: Αξιοποίηση των πόρων της πλατφόρμας

| | D&C ($MF = 1$) | D&C ($MF = 0.5$) | D&C ($MF = 0$) | ADAM [9] |
|---|---|---|---|---|
| Scenario 1 | 100% | 92% | 92% | 91% |
| Scenario 2 | 100% | 88% | 88% | 87% |
| Scenario 3 | 100% | 87% | 87% | 88% |
| Scenario 4 | 100% | 86% | 86% | 84% |
| Scenario 5 | 100% | 78% | 78% | 79% |
| Scenario 6 | 100% | 87% | 87% | 88% |

με σχεδόν την ίδια υπολογιστική προσπάθεια. Τα τυχαία σενάρια έδειξαν ότι ο προτεινόμενος αλγόριθμος μπορεί να έχει, ανάλογα με το επιλεγμένο $MF$, διαφορετική συμπεριφορά και αποτέλεσμα της αξιοποίησης των πόρων της πλατφόρμας.

## 4.3 Κατανεμημένη διαχείριση πόρων για εύπλαστες παράλληλες εφαρμογές

Η διαχείριση πόρων σε πολυπύρηνες πλατφόρμες κατα τη φάση εκτέλεσης έχει αποκαλυφθεί ως μια βασική πρόκληση για τα σύγχρονα συστήματα και έχει αρχίσει να επικρατεί, λόγω της δυναμικότητας των σύγχρονων πλατφορμών και παράλληλων εφαρμογών. Στα σύγχρονα συστήματα η διαθεσιμότητα των πόρων μπορεί να μεταβάλλεται λόγω του δυναμισμού του συστήματος, καθώς οι πόροι μπορούν να προστεθούν ή να αφαιρεθούν ανά πάσα στιγμή. Σύμφωνα με την ταξινόμηση των παράλληλων εφαμρογών που παρουσιάζεται στο [41], μπορούμε να διακρίνουμε τους εξής τύπους σύμφωνα με τα χαρακτηριστικά τους:

- **Αποτυπώσιμες (Moldable) εφαρμογές**: Παράλληλες εφαρμογές που μπορούν να διακοπούν σε οποιοδήποτε σημείο, αλλά ο αριθμός των επεξεργαστών για τις εφαρμογές αυτές δεν μπορεί να αλλάξει κατά τη διάρκεια της φάσης εκτέλεσης.
- **Εύπλαστες (Malleable) εφαρμογές**: Αυτές είναι παράλληλες εφαρμογές που μπορούν να διακοπούν σε οποιοδήποτε σημείο της εκτέλεσης και έχουν τη δυνατότητα να αλλάξουν τον αριθμό των εκχωρημένων επεξεργαστών κατά την φάση εκτέλεσης. Αυτές οι εφαρμογές ονομάζονται επίσης και επαναδιαμορφούμενες.
- **Μεταναστευτικές (Migratable) εφαρμογές**: Αυτές είναι παράλληλες εφαρμογές που μπορούν να διακοπούν σε οποιοδήποτε σημείο της εκτέλεσης και μπορούν να ξαναρχίσουν σε διαφορετικό χώρο, cluster ή τομέα.

Όπως περιγράφεται στο [54] η ευπλαστότητα (malleability) χρησιμοποιείται για την αυτόνομη αναδιάρθρωση εφαρμογών ως ανταπόκριση στις δυναμικές αλλαγές στη διαθεσιμότητα των πόρων της πλατφόρμας, επιτρέποντας έτσι στις εφαρμογές να βελτιστοποιούν τη χρήση των χαρακτηριστικών της (π.χ., τον αριθμό των επεξεργαστών). Με άλλα λόγια, οι εύπλαστες (malleable) εφαρμογές καθορίζουν τον ελάχιστο και τον μέγιστο αριθμό των επεξεργαστών που χρειάζονται για την εκτέλεσή τους.

### 4.3.1 Μαθοδολογικό πλαίσιο

Ο στόχος του προτεινόμενου μεθοδολογικού πλαισίου [14] είναι η κατανεμημένη διαχείριση πόρων σε πολυπύρηνες πλατφόρμες κατα τη φάση εκτέλεσης, τόσο ομοιογενείς όσο και ετερογενείς, για παράλληλες εφαρμογές. Το προτεινόμενο πλαίσιο

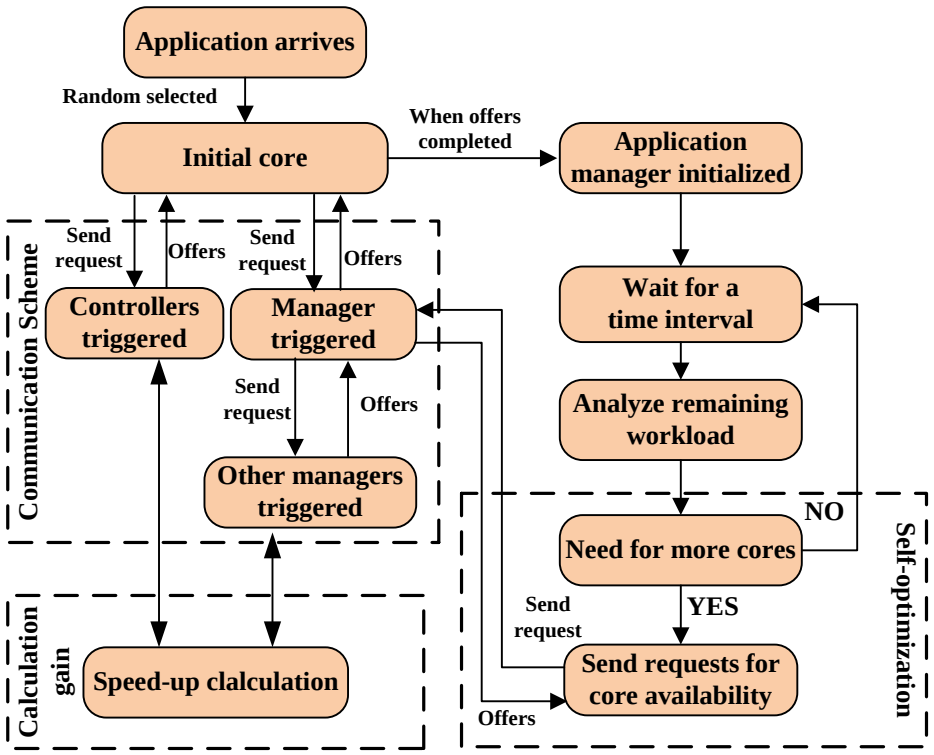
έχει σχεδιαστεί για εύπλαστες (malleable) εφαρμογές. Αν και μπορεί να υποστηρίξει και τους άλλους δύο τυπους παράλληλων εφαρμογών, αξιοποιεί καλύτερα τους διαθέσιμους πόρους της πλατφόρμας για τις εύπλαστες (malleable) εφαρμογές. Στο προτεινόμενο μεθοδολογικό πλαίσιο διακρίνονται τρεις ρόλοι που μπορεί να έχει ένας πυρήνας: *(i) αρχικός πυρήνας (initial core)*, *(ii) πυρήνας ελεγκτής (controller core)*, και *(iii) πυρήνας διαχειριστής (manager core)*.

Ο *αρχικός πυρήνας (initial core)* επιλέγεται τυχαία και ενεργοποιείται όταν μια νέα εφαρμογή εισέλθει στο σύστημα. Σκοπός του είναι να βρει το αρχικό σύνολο των πυρήνων στο οποίο η εφαρμογή θα αρχίσει να εκτελείται. Ζητά πυρήνες από τους πυρήνες ελεγκτή και διαχειριστή, συγκεντρώνει τις προσφορές τους και καθορίζει εάν τουλάχιστον ένας πυρήνας είχε προσφερθεί. Ένας αρχικός πυρήνας δεν μπορεί να χειριστεί δύο εφαρμογές ταυτόχρονα. Ως εκ τούτου, εάν επιλεχτεί να αρχικοποιήσει μια εφαρμογή, ενώ παράλληλα χειρίζεται μια προηγούμενη, η καινούρια θα πρέπει να περιμένει.

Ο *πυρήνας ελεγκτής (controller core)* είναι υπεύθυνος για το χειρισμό όλων των ελεύθερων πυρήνων μιας προκαθορισμένης περιοχής της πλατφόρμας. Ορίζεται στην αρχικοποίηση της πλατφόρμας και δεν μπορεί να αλλάξει κατά το χρόνο εκτέλεσης. Την ίδια στιγμή, ο πυρήνας ελεγκτής διατηρεί έναν κατάλογο όλων των πυρήνων διαχειριστών που καταλαμβάνουν έναν πυρήνα στην περιοχή του. Αυτό είναι ένα ουσιαστικό μέρος του κατανεμημένου σχεδιασμού, δεδομένου ότι οι πληροφορίες των δεσμευμένων πυρήνων σε μια περιοχή δεν διατηρούνται σε ένα κεντρικό σημείο.

Τέλος, ο *πυρήνας διαχειριστής (manager core)* διαχειρίζεται μια εφαρμογή που ψάχνει για νέους πυρήνες και καθορίζει τον ανασχεδιασμό της εφαρμογής κάθε φορά που έχει περισσότερους ή λιγότερους πυρήνες για να τρέξει. Αυτός ο πυρήνας δεν εκτελεί κανένα μέρος της εφαρμογής. Παρά το γεγονός ότι ένας αρχικός πυρήνας (initial core) μπορεί να είναι πυρήνας διαχειριστής (manager core) και αντιστρόφως, ένας πυρήνας ελεγκτής (controller core) δεν μπορεί να αλλάξει τη λειτουργικότητά του.

Μια επισκόπηση του προτεινόμενου μεθοδολογικού πλαισίου παρουσιάζεται στο Σχήμα 4.9. Μόλις μια νέα εφαρμογή φτάνει σε τυχαίο πυρήνα (αυτό είναι ο αρχικός πυρήνας), αυτός ο πυρήνας στέλνει μηνύματα στους πυρήνες ελεγκτές κοντά του και ζητά ένα διαθέσιμο πυρήνα για να χρησιμοποιηθεί ως διαχειριστής της εφαρμογής. Στη συνέχεια, οι πυρήνες ελεγκτές αναζητούν στο χώρο τους τυχόν ελεύθερους πυρήνες στέλνοντας παράλληλα αιτήματα σε πυρήνες διαχειριστές. Ο αρχικός πυρήνας δέχεται όλες τις προσφορές και καθορίζει το νέο διαχειριστή που ξεκινά από το αντίστοιχο σήμα. Στη συνέχεια, ο νέος διαχειριστής κατανέμει ομοιόμορφα τον φόρτο εργασίας που διαχειρίζεται. Μετά από ένα προκαθορισμένο χρονικό διάστημα, και όταν η εφαρμογή δεν είναι κοντά στην ολοκλήρωσή της, ο διαχειριστής ελέγχει αν υπάρχει ανάγκη για αναδιοργάνωσή της με στόχο την βελτιστοποίηση της απόδοσης.




Σχήμα 4.9: Επισκόπηση του προτεινόμενου μεθοδολογικού πλαισίου [14]

#### 4.3.1.1 Ορισμοί

Η μοντελοποίηση των εφαρμογών και του μοντέλου επιάχυνσης που χρησιμοποιήθηκε για τις εύπλαστες εφαρμογές, παρουσιάζεται στο [35, 40] και χρησιμοποιείται και απο άλλες κατανεμημένες μεθόδους [54]. Κάθε εφαρμογή περιγράφεται απο τέσσερις παραμέτρους $W$, $var$, $A$ και $Q$, όπου $W$ είναι ο φόρτος εργασίας, $var$ είναι η διακύμανση της παραλληλίας, $A$ είναι η μέση τιμή της παραλληλίας και $Q$ είναι το πιο προτιμόμενο επεξεργαστικό στοιχείο στο οποίο η εφαρμογη μπορει να εκτελεστεί.

Μία πολυπύρηνη τοπολογια μπορει να περιγραφει μοναδικά απο ένα συνδεδεμένο κατευθυνόμενο γράφο $P(I, N)$. Το σύνολο των κορυφών $N$ αποτελείται απο δύο αμοιβαιώς αποκλειόμενα υποσύνολα $N_{PE}$ και $N_C$ τα οποία περιλαμβανουν τα διαθέσιμα επεξεργαστικά στοιχεία και τα στοιχεία διασύνδεσης αντίστοιχα. Κάθε επεξεργαστικό στοιχείο της πλατφόρμας μπορεί να είναι ενός ειδικού τύπου. Στο προτεινόμενο πλαίσιο το $T_{pe_i} \forall pe_i \in N_{PE}$ ορίζει τον τύπο του επεξεργαστικού στοιχείου $pe_i$. Στις ετερογενείς πλατφόρμες, το $T_{pe_i}$ διαφέρει για κάθε στοιχείο ενώ στις ομογενείς είναι συνέχεια το ίδιο. Προκειμένου να καλυφθούν οι ανάγκες τις εφαρμογής και της πλατφόρμας και προκειμένου να έχουμε την καλύτερη απόδοση $Q \rightarrow T_{pe_i}$, ορίζουμε την παράμετρο $Util[pe_i] \in [0, 1]$ που δηλώνει πόσο καλά η εφαρμογή $app(W, var, A, Q)$ εξυπηρετείται απο το $pe_i$ με τύπο $T_{pe_i}$. Η παράμετρος $Util[pe_i]$ μπορεί να θεωρηθεί σαν ένας παράγοντας προτεραιότητας όταν μία εφαρμογή ζητά επιπλέον πυρήνες.

Έτσι, όταν $Util[pe_i] = 1$ θέλουμε την καλυτερη αντιστοίχηση ενώ όταν $Util[pe_i] = 0$ η εφαρμογή δεν απαιτεί κάποιο συγκεκριμένο $T_{pe_i}$. Τέλος, ορίζουμε τα σύνολα $F$ και $offers[]$, τα οποία περιγράφουν όλους τους κόμβους που μπορούν να εξυπηρετήσουν σωστά την εφαρμογη με βάση τον τύπο τους και το σύνολο των πυρήνων που προσφέρονται στην εφαρμογή, αντίστοιχα.

#### 4.3.1.2 Επικοινωνιακό πλαίσιο

Για λόγους συνέπειας στην ανάλυση των Αλγορίθμων 3 και 4, οριζεται ότι ο δείκτης $dst$ αντιπροσωπεύει τον πυρήνα διαχειριστή ο οποίος ζητά περισσότερους πόρους ενώ ο ο δείκτης $src$ είναι ο πυρήνας διαχειριστής που τους προσφέρει. Επίσης, το σύνολο $R$ περιέχει, για κάθε πυρήνα ελεγκτή, τα επεξεργαστικά στοιχεία που βρίσκονται εντός της περιοχής που ορίζεται απο την Εξίσωση 4.6 όπου $size$ είναι το μέγεθος της πλατφόρμας και $num\_controllers$ είναι ο αριθμός των πυρίνων ελεγκτών κατά την $X$ διάσταση.

$$distance = size/num\_controllers_X \tag{4.6}$$

**Algorithm 3** Αλγόριθμος για το επικοινωνιακό πλαίσιο

```
    // Αρχικός πυρήνας
 1: analyze(W, var, A, Q)
 2: req_send(control[], core_id, app, R)
 3: start_timer()
 4:    offers[] = receive_offers
 5: end_timer()
 6: sel_pe = best{offers[]}
 7: initialize_manager(sel_pe, offer, R)

    // Πυρήνας ελεγκτής
 8: analyze(W, var, A, Q, R)
 9: for each (N_PE ∈ F && N_PE ∈ R)
10:    If calculate(gain(app)) > 0 // Αλγόριθμος 4
11:       offers[] = offers[] + new_offer
12: send_offers(offers, core_id)

    // Πυρήνας διαχειριστής
13: // Προσφορά πυρήνων
14: analyze(W, var, A, Q, R)
15: If calculate(gain(app)) > 0 // Αλγόριθμος 4
16:    offers[] = offers[] + new_offer
17:    send_offers(offers, core_id)
18: // Αυτοβελτίωση
19: while (app(W, var, A, Q)! = finished) {
20:    analyze(W, var, A, Q, offers)
21:    timer()
22:    If ((app.threshold = max) || (app.left_time < timer()))
23:       continue
24:    else
25:       req_send(control[], R)
26:       start_timer()
27:          offers[] = offers[] + receive_offers
28:       end_timer()
29: end while
```

Όπως προαναφέρθηκε, όταν μία νέα εφαρμογή φτάσει σε έναν πυρήνα, ορίζεται ο αρχικός πυρηνας και η επικοινωνία στο εσωτερικό της πλατφόρμας πραγματοποιείται με σκοπό να δημιουργηθεί ένας πυρήνας διαχειριστής για την εφαρμογή. Η επικοινωνία μεταξύ των πυρήνων περιγράφεται στον Αλγόριθμο 3.

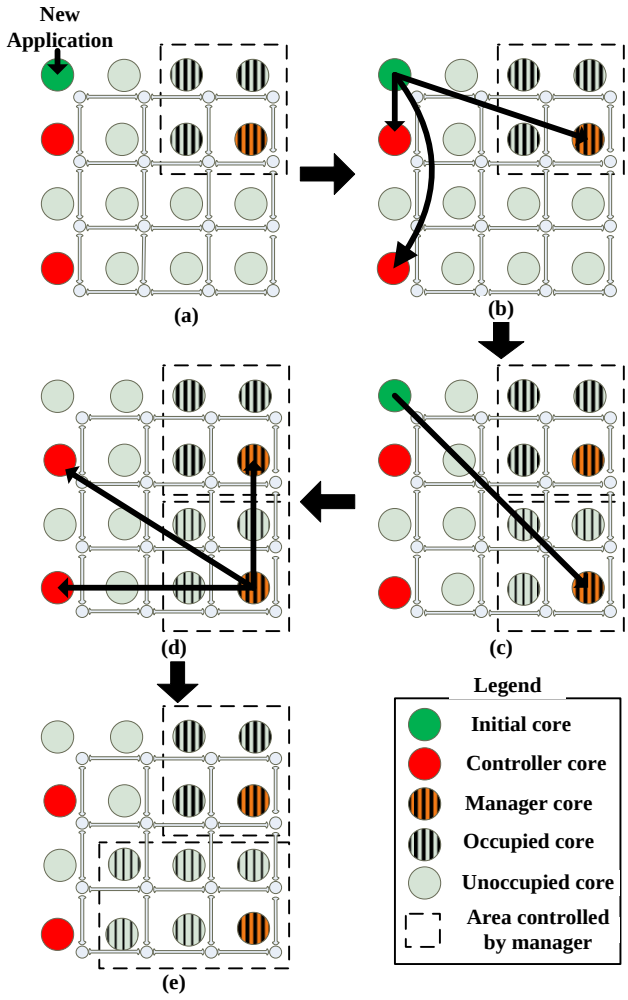
**Αρχικός πυρήνας (initial core) (γραμμές 1-7)**: Όταν μια νέα εφαρμογή φτάσει σε έναν αρχικό πυρήνα (Σχήμα 4.10a), αυτός αναλύει τα χαρακτηριστικά της, στέλνει ένα μήνυμα προς τους πυρήνες ελεγκτές και διαχειριστές (Σχήμα 4.10b) που βρίσκονται εντός της περιοχής του $R$ και πυροδοτεί ένα χρονόμετρο προκειμένου να ελέγξει για τις απαντήσεις τους. Μετά το τέλος του χρονοδιακόπτη, ο αρχικός πυρήνας επιλέγει την καλύτερη προσφορά και στέλνει ένα σήμα σύμφωνα με το το οποίο ξεκινά την προετοιμασία του διαχειριστή που θα χειριστεί την εφαρμογή (Σχήμα 4.10c).

**Πυρήνας ελεγκτής (controller core) (γραμμές 8-12)**: Ένας πυρήνας ελεγκτής εκτός από τη διατήρηση των πληροφοριών σχετικά με τους διαχειριστές που υπάρχουν στην περιοχή, θα πρέπει να παρέχει τις εν λόγω πληροφορίες σε οποιαδήποτε πυρήνα τις ζητά. Ενημερώνει, επίσης, αυτούς τους πυρήνες για τη θέση των πυρήνων ελεγκτών και σε άλλες περιοχές. Όταν ο ελεγκτής λάβει το σήμα απο τον αρχικό πυρήνα, αναλύει την εφαρμογή και αρχίζει να βρίσκει προσφορές. Ο πυρήνας ελεγκτής μπορεί να προσφέρει κάθε ελεύθερο πυρήνα που διαθέτει, μέσα στην περιοχή $R$ του αρχικού πυρήνα, υπό την προϋπόθεση ότι εξυπηρετεί τα χαρακτηριστικά της εφαρμογής.

**Πυρήνας διαχειριστής (manager core) (γραμμές 13-29)**: Ο πυρήνας διαχειριστής έχει δύο καθήκοντα. Αρχικά, ο διαχειριστής ελέγχει αν μπορεί να προσφέρει έναν πυρήνα στη νέα εφαρμογή χωρίς να χάνει περισσότερο απ'ότι θα ειναι το κέρδος επιτάχυνσης στη νέα εφαρμογή. Το δεύτερο έργο έχει να κάνει με την διαδικασία αυτο-βελτιστοποίησης της εφαρμογής που εκτελείται ήδη. Συγκεκριμένα, υπάρχει ένα χρονικό όριο στο οποίο ο διαχειριστής ελέγχει αν η εφαρμογή έχει όλα τα απαραίτητα μέσα που χρειάζεται ή είναι κοντά στην ολοκλήρωσής. Αν κριθεί απαραίτητο, η εφαρμογη εισέρχεται σε μία φάση αυτο-βελτιστοποίησης και ο διαχειριστής ενημερώνει τους πυρηνες ελεγκτές (Σχήμα 4.10d) που είναι μέσα στην περιοχή $R$. Μετά το τέλος ενός χρονικού ορίου, ο πυρήνας διαχειριστής ελέγχει τις προσφορές και αρχίζει η διαδικασία (Σχήμα 4.10e). Ο υπολογισμός του κέρδους για την εφαρμογή παρουσιάζεται στον Αγλόριθμο 4.

#### 4.3.1.3 Υπολογισμός κέρδους

Ο Αλγόριθμος 4 περιγράφει τα απαιτούμενα βήματα που τόσο ο ελεγκτής όσο και ο διαχειριστής εκτελούν προκειμένου να αποφασίσευν ποιοί πυρήνες θα προσφερθούν κατά την εκκίνηση μιας εφαρμογής ή κατα τη φάση της αυτο-βελτιστοποίησης. Υπάρχουν τρία διακριτά μέρη: (i) οι δράσεις που αφορούν τον υπολογισμό της επιτάχυνσης του προορισμού (γραμμές 4-8), (ii) οι δράσεις όσον αφορά τον υπολογισμό της επιτάχυνσης της πηγής (γραμμές 9-13) και (iii) ο υπολογισμός του συνολικού κέρδους της ανταλλαγής (γραμμές 14-21). Κατά τη διάρκεια των δράσεων που αφο-




Σχήμα 4.10: Παράδειγμα του επικοινωνιακού πλαισίου

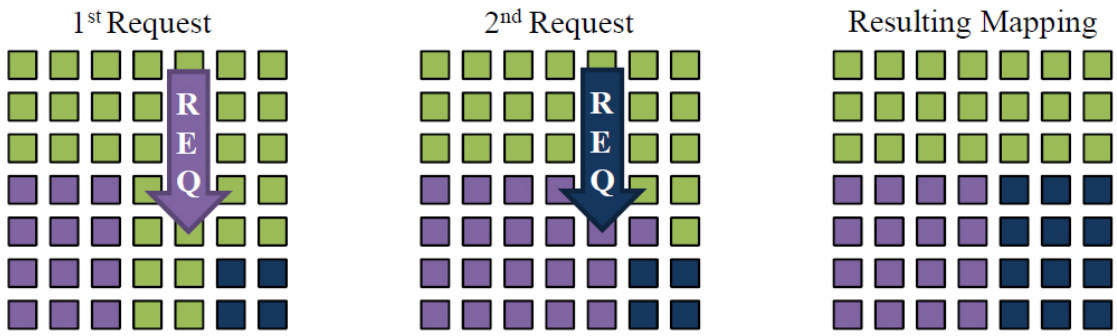
ρούν τον κόμβο προορισμό, για κάθε $N_{PE} \in ((PE_{src} \cap R \cap F) - offers[])$ υπολογίζεται η επιτάχυνση της εφαρμογίς λαμβάνοντας υπόψη την παράμετρο $Util[N_{PE}]$. Μόλις υπολογιστεί η επιτάχυνση υπολογίζεται το συνολικό τελικό κέρδος. Από την άλλη πλευρά, οι δράσεις που αφορούν τον κόμβο πηγή ελέγχουν αν η απώλεια ενός πυρήνα οδηγεί σε μεγαλύτερη υποβάθμιση των επιδόσεων σε μια εφαρμογή που ήδη εκτελείται.

**Algorithm 4** Αλγόριθμος υπολογισμού κέρδους

```
 1: offers[] = ∅
 2: while (gain > 0) {
 3:    for each N_PE ∈ ((PE_src ∩ R ∩ F) − offers[]) {
    // Δράσεις που αφορούν τον κόμβο προορισμό
 4:       PE_dst = PE_dst ∪ N_PE ∪ offers[]
 5:       ord_PE_dst = order{PE_dst}
 6:       for pos = 1 to ord_PE_dst.length() {
 7:          SP_dst = Util{ord_PE_dst[pos]} ∗ (SP[pos] − SP[pos − 1])
 8:       gain_dst = SP_dst − previous_SP_dst
    // Δράσεις που αφορούν τον κόμβο πηγή
 9:       PE_src = PE_src − offers[] − N_PE
10:       ord_PE_src = order{PE_src}
11:       for pos = 1 to ord_PE_src.length() {
12:          SP_src = Util{ord_PE_src[pos]} ∗ (SP[pos] − SP[pos − 1])
13:       loss_src = previous_SP_src − SP_src
    // Υπολογισμός τελικού κέρδους
14:       gain_temp = gain_dst − loss_src
15:       if ((gain_temp > gain) || ((gain_temp = gain) && D(manger, N_PE) < D(manger, prev_N_PE)))
16:          gain = gain_temp
17:          prev_N_PE = N_PE
18:    end for
19:    if (gain > 0)
20:       offers[] = offers[] ∪ N_PE
21: end While
```

#### 4.3.1.4 Διαδικασία αυτο-βελτιστοποίησης

Ένα σημαντικό μέρος του προτεινόμενου πλαισίου μας είναι ότι ο πυρήνας διαχειριστής μπορεί να αποφασίσει να ψάξει για περισσότερους πυρήνες, προκειμένου να αυξηθεί ο αριθμός των πυρήνων εργασίας του. Η διαδικασία αυτή ονομάζεται αυτο-βελτιστοποίηση και στόχος της είναι να αυξήσει περαιτέρω την επιτάχυνση των εφαρμογών που ήδη εκτελούνται. Όλοι οι πυρήνες διαχειριστές μπορούν να κινήσουν τη διαδικασία αυτή με αποτέλεσμα την καλύτερη κατανομή μεταξύ τους. Με τη χρήση της διαδικασίας αυτο-βελτιστοποίησης, η καινούρια εφαρμογή θα αποκτήσει κάποιους πυρήνες απο κάποια άλλη εφαρμογη η οποία ηδη εκτελείται με τέτοιο τρόπο έτσι ώστε το κέρδος της απόκτησης των νέων πυρήνων να είναι μεγαλύτερο από την απώλεια στην εφαρμογή που ήδη εκτελείται. Επιπλέον, η διαθεσιμότητα των πόρων της πλατφόρμας αλλάζει δυναμικά καθώς κάποιες εφαρμογές ολοκληρώνονται. Δεδομένου ότι δεν υπάρχει κεντρικό σύστημα διαχείρισης των πόρων, οι υπάρχουσες εφαρμογές πρέπει να διεκδικήσουν τους νέους πυρήνες χωρίς κάποιο αρχική πληροφορία σχετικά με τη διαθεσιμότητά τους. Αυτή η διαδικασία αυτο-βελτιστοποίησης έχει επίσης χρησιμοποιηθεί σε άλλες κατανεμημένες προσεγγίσεις [54].




Σχήμα 4.11: Δίκαιη κατανομή πόρων μέσω αυτο-βελτιστοποίησης [54]

### 4.3.2 Αξιολόγηση

Για να επαληθεύουμε την προσέγγισή μας, πραγματοποιήσαμε εκτεταμένες προσομοιώσεις του προτεινόμενου πλαισίου σε δύο στάδια: (i) χρησιμοποιώντας έναν προσομοιωτή σε C (Κεφάλαιο 4.3.2.1) και (ii) χρησιμοποιώντας την πολυπύρηνη πλατφόρμα Intel Single Cloud Chip (SCC) [49] (Κεφάλαιο 4.3.3)

#### 4.3.2.1 Αξιολόγηση στον προσομοιωτή (C)

Για πιο ακριβή προσομοίωση, κάθε κόμβος αντιπροσωπεύεται από μια διαφορετική διεργασία. Η επικοινωνία μεταξύ των κόμβων υλοποιείται με τη χρήση παραδοσιακών σημάτων Linux, ενώ τα μηνύματα ανταλάσονται με τη χρήση pipes και ο συγχρονισμός επιτυγχάνεται με χρηση σημαφόρων. Ο στόχος αυτού του βήματος ήταν να έχουμε μια γρήγορη και αφηρημένη άποψη της μεθοδολογίας μας και να εκτιμηθεί το κόστος επικοινωνίας. Επίσης, η μέθοδος προσομοίωσης προσφέρει τη δυνατότητα της λειτουργικής διόρθωσης σφαλμάτων και πιθανών αδιεξόδων στην επικοινωνία. Ο προσομοιωτής υποστηρίζει πολλές τοπολογίες (μέχρι $32\times32$) με πολλές εισόδους εφαρμογών.

Ο Πίνακας 4.2 παρουσιάζει τα αποτελέσματα της σύγκρισης της προτεινόμενης τεχνικής με τον DistRM [54]. Οι δυο μέθοδοι αξιολογήθηκαν για διάφορα μεγέθη πλατφορμών, απο $6\times6$ μέχρι και $32 \times 32$, και για 32 και 64 εφαρμογές. Τα μεγέθη σύγκρισης είναι: (i) ο αριθμός των μηνυμάτων (*Msg. cnt.*), που είναι ο συνολικός αριθμός των μηνυμάτων που αποστέλλονται από όλους τους κόμβους, (ii) το μέγεθος των μηνυμάτων (*Msg. size*), το οποίο είναι το συνολικό μέγεθος των απεσταλμένων μηνυμάτων, (iii) η μέση επιτάχυνση της εφαρμογής (*Avg. sp.*) και (iv) υπολογιστική προσπάθεια (*Comp. eff.*). Ο Πίνακας 4.2 παρουσιάζει τα ποσοστιαία κέρδη (*percentage gains*) του πλαισίου σε σύγκριση με τον DistRM. Όσον αφορά τον αριθμό των μηνυμάτων, τα αποτελέσματα της προσομοίωσης έδειξαν ένα μέσο κέρδος της τάξης του 41% για την πρτεινόμενη μεθοδολογία που παρουσιάζεται λόγω του ότι τα μηνύματα αποστέλλονται μόνο μέσα στην περιοχή $R$, ενώ ο DistRM στέλνει μηνύματα σε πολλές μικρότερες πειοχές. Επίσης, το συνολικό μέγεθος αυτών των μηνυμάτων, σε *bytes*, είναι κατά μέσο όρο 37% μικρότερο απο το μέγεθος που απαιτείται από τον DistRM.

Πίνακας 4.2: Σύγκριση του προτεινόμενου πλαισίου με τον αλγόριθμο DistRM [54] στον C προσομοιωτή

| | Msg. cnt. | | Msg. size | | Avg. sp. | | Comp. eff. | |
|---|---|---|---|---|---|---|---|---|
| Plat. | Number of applications | | | | | | | |
| sizes | 32 | 64 | 32 | 64 | 32 | 64 | 32 | 64 |
| 6x6 | 72.3 | 71.4 | 73.2 | 72.4 | 3.8 | 13.8 | 86.8 | 86.6 |
| 8x8 | 64.3 | 63.4 | 64.8 | 63.6 | 4.5 | 9.3 | 83.3 | 83.0 |
| 12x12 | 49.1 | 52.3 | 44.8 | 48.7 | -1.4 | 1.5 | 66.0 | 68.3 |
| 16x16 | 42.6 | 45.8 | 40.0 | 41.4 | 0.5 | 3.1 | 61.3 | 64.9 |
| 20x20 | 32.6 | 36.1 | 27.7 | 30.5 | 3.1 | 2.7 | 53.4 | 57.8 |
| 24x24 | 25.1 | 27.2 | 18.5 | 19.2 | 2.0 | 2.4 | 50.1 | 51.7 |
| 28x28 | 20.6 | 22.3 | 13.5 | 14.1 | 1.5 | 2.8 | 42.7 | 48.7 |
| 32x32 | 17.9 | 14.6 | 9.9 | 2.5 | 1.6 | 2.8 | 41.9 | 42.4 |
| Average | 41% | | 37% | | 3% | | 62% | |

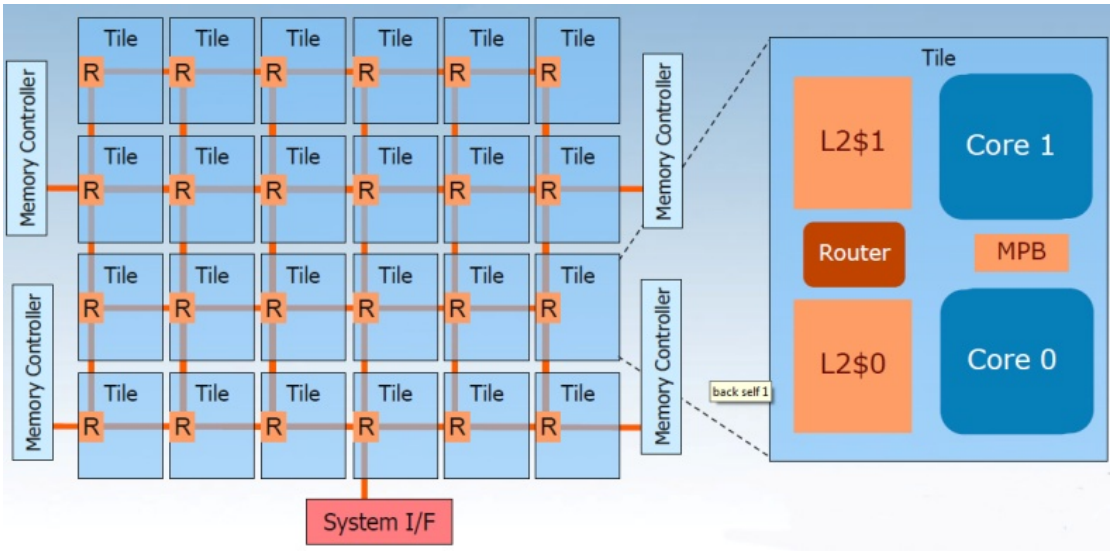
Έτσι, επιβάρυνση του προτεινόμενου πλαισίου στο δίκτυο είναι κατά μέσο όρο 38% μικρότερη. Από την άποψη της μέσης επιτάχυνσης των εφαρμογών, το προτεινόμενο πλαίσιο επιτυγχάνει κατα μέσο όρο 3% καλύτερα αποτελέσματα. Τέλος, το κέρδος της προτεινόμενης μεθοδολογίας σχετικά με τη συνολική υπολογιστική προσπάθεια είναι κατά μέσο όρο 62% σε σύγκριση με τον DistRM.

### 4.3.3 Αξιολόγηση στην πλατφόρμα Intel SCC

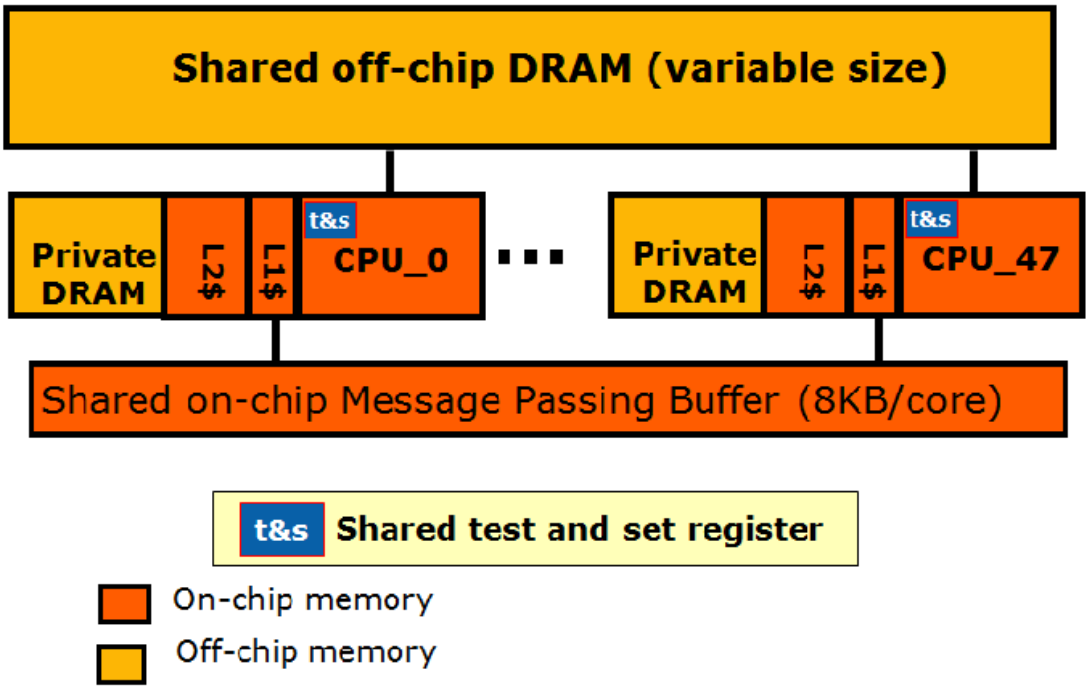
Η πλατφόρμα Intel SCC [49] διαθέτει 48 πυρήνες τύπου x86 με τύπο διασύνδεσης Δίκτυο-σε-Ψηφίδα. Μια ολική επισκόπηση της πλατφόρμας παρουσιάζεται στο Σχήμα 4.12 και αποτελείται απο:

- Δύο μπλόκς, κάθε ένα με επεξεργαστη P54C, 16 KB μνήμη cache εντολών και δεδομένων instruction και μία ενοποιημένη μνήμηε cache L2 256 KB.
- Ένα Mesh Interface Unit (MIU) που επιτρέπει διαφορετικές συχνότητες λειτουργίας.
- 16 KB Message Passing Buffer (MPB).
- Δύο test-and-set καταχωρητές.

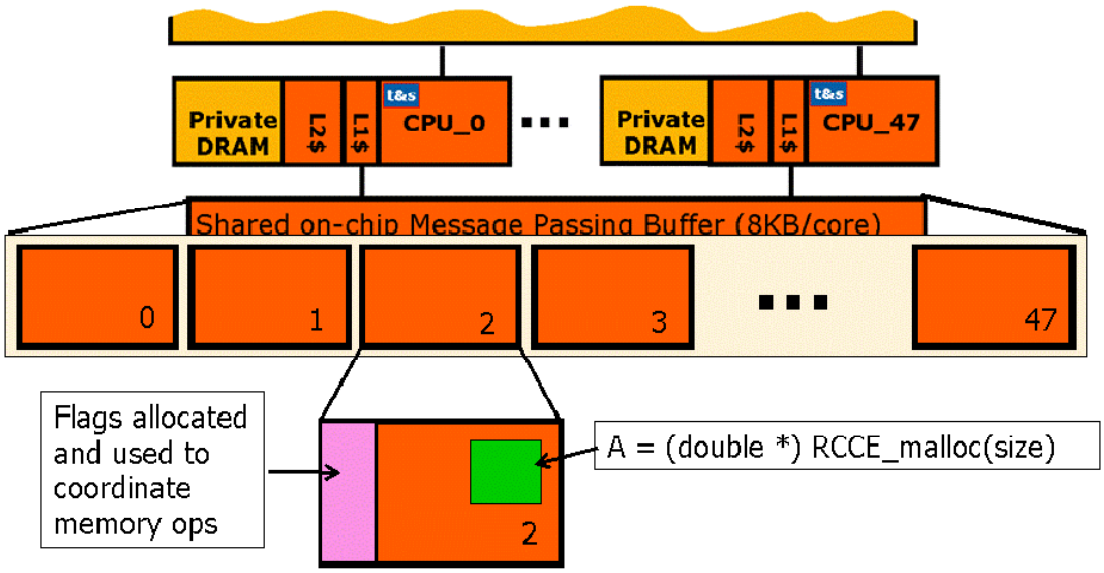
Κάθε μπλοκ συνδέεται με έναν δρομολογητή (router). Αυτός ο δρομολογητής συνδέεται με το MIU για την ενσωμάτωση των μπλοκ σε ένα πλέγμα. Η MIU δημιουργεί πακέτα δεδομένων στο και τα συλλέγει από το πλέγμα χρησιμοποιώντας ένα round-robin σύστημα για τη διαιτησία μεταξύ των πυρήνων. Η μνήμη έξω απο το ολοκληρωμένο κυμαίνεται απο 16 έως 64 GB DDR3 RAM και ελέγχεται από τέσσερις ελεγκτές μνήμης. Ολόκληρη η αρχιτεκτονική μνήμης της πλατφόρμας Intel SCC απεικονίζεται στο Σχήμα 4.13.




Σχήμα 4.12: Επισκόπηση της πλατφόρμας Inter SCC [84]




Σχήμα 4.13: Η αρχιτεκτονική μνήμης της πλατφόρμας Intel SCC [62]

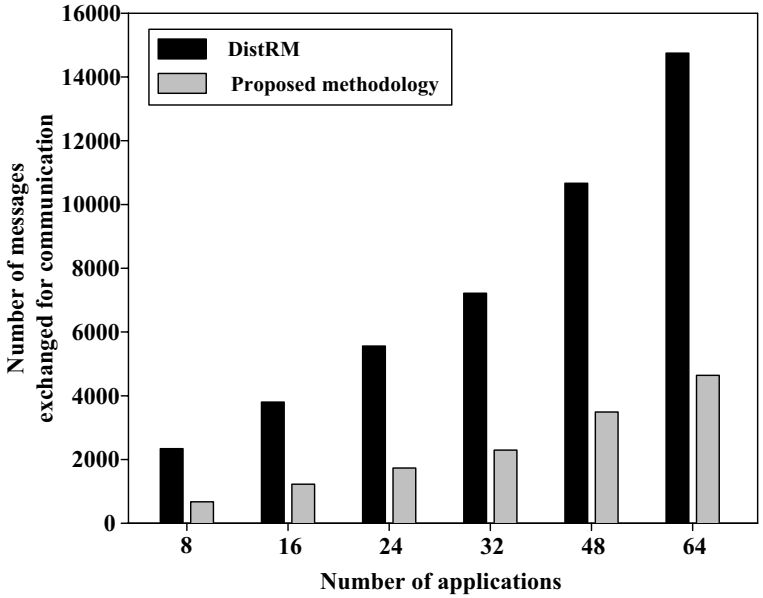



Σχήμα 4.14: Διευθυνσιοδότηση του MPB για την βιβλιοθήκη RCCE [62]

Προκειμένου να καταστεί ευκολότερη στον προγραμματισμό και να αυξήθεί η φορητότητα και η επεκτασιμότητα των προγραμμάτων που γράφονται για την πλατφόρμα SCC, η Intel παρέχει ένα περιβάλλον επικοινωνίας γνωστό ως RCCE. Μάλιστα, SCC και RCCE αναπτύχθηκαν παράλληλα [84]. Το RCCE διανέμει ομοιόμορφα το χώρο διευθύνσεων του MPB στους 48 πυρήνες και ορίζει ότι κάθε πυρήνας θα έχει 8 KB της μνήμης για τον εαυτό του (Σχήμα 4.14). Παρέχει δύο βασικές διεπαφές για επικοινωνία μεταξύ των κόμβων. Η πρώτη ονομάζεται *gory* το οποίο είναι χαμηλού επιπέδου και προσφέρει στον προγραμματιστή μεγαλύτερο έλεγχο πάνω από την SCC με αντάλλαγμα την απαίτηση για συγχρονισμό. Η δεύτερη διεπαφή ονομάζεται *basic* και έχει υλοποιημένες συναρτήσεις `send - recv` για θέματα συγχρονισμού.

Δεδομένου ότι το μέγεθος της πλατφόρμας είναι σταθερό (48 πυρήνες) συγκρίναμε την απόδοση της προτεινόμενης μεθοδολογίας με τον κατανεμημένο αλγόριθμο DistRM [54] για ποικίλο αριθμό εφαρμογών (απο 8 έως 64).

Τα Σχήματα 4.15 και 4.16 παρουσιάζουν το συνολικό αριθμό των μηνυμάτων που αποστέλλονται από όλους τους κόμβους σε όλη τη διάρκεια της προσομοίωσης και το συνολικό μέγεθος αυτών των μηνυμάτων σε *bytes* αντίστοιχα. Τα μηνύματα χρησιμοποιούνται για την αρχικοποίηση της εφαρμογής, την αυτο-βελτιστοποίηση και την αλλαγή μεγέθους σε αριθμό πυρήνων. Το προτεινόμενο πλαίσιο στέλνει κατά μέσο όρο 70% λιγότερα μηνύματα, προκειμένου να εκτελέσει όλες τις απαραίτητες ενέργειες δεδομένου ότι τα μηνύματα αποστέλλονται μόνο μέσα στην περιοχή $R$ ενώ ο DistRM αναζητά διαθέσιμους πυρήνες σε περισσότερες περιοχές, μικρότερες από $R$, αυξάνοντας έτσι τον αριθμό των μηνυμάτων που χρησιμοποιούνται.

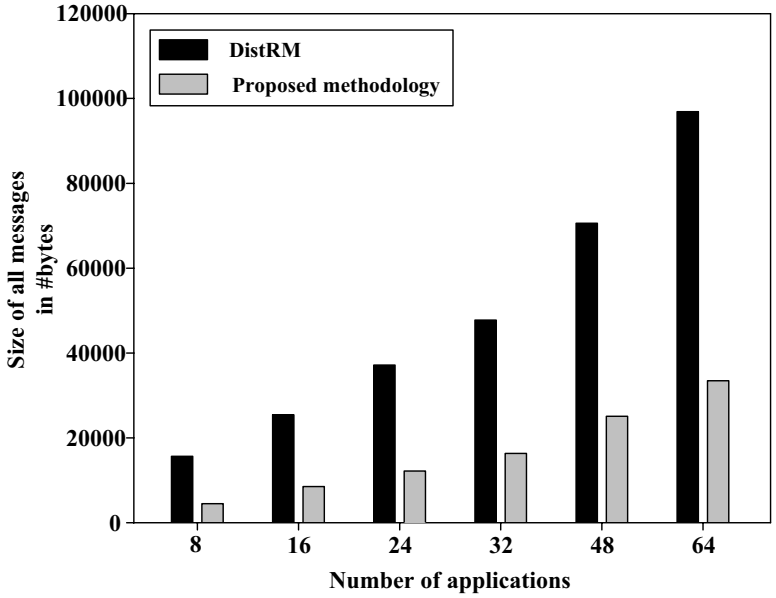



Σχήμα 4.15: Συνολικός αριθμός μηνυμάτων που αποστέλλονται για την επικοινωνία μεταξύ όλων των κόμβων για διάφορες εφαρμογές σε σύγκριση με τον αλγόριθμο DistRM [54]

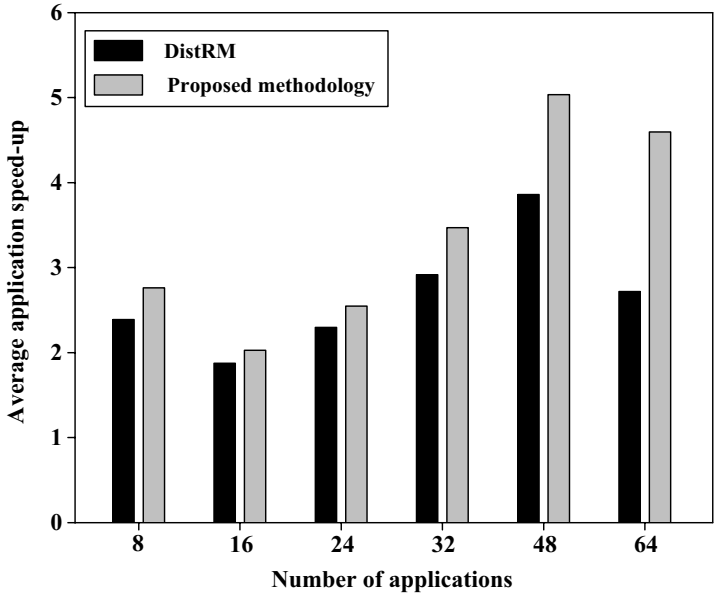
Ένας άλλος λόγος που ο προτεινόμενος αλγόριθμος έχει λιγότερα μηνύματα από τον DistRM, είναι το γεγονός ότι το πλαίσιο εκτελεί την διαικασία αυτο-βελτιστοποίησης κάτω από πολύ συγκεκριμένα κριτήρια και μόνο όταν η εφαρμογη δεν έχει μεγιστοποιήσει την επιτάχυνση της ή δεν είναι κοντά στην ολοκλήρωσή της. Επίσης, το μέγεθος των απεσταλμένων μηνυμάτων είναι κατά μέσο όρο 64% λιγότερο από αυτά που χρησιμοποιούνται από το DistRM. Το μέγεθος δεν είναι ανάλογο με τον αριθμό των μηνυμάτων δεδομένου ότι κάθε μήνυμα ποικίλλει σε μέγεθος. Για παράδειγμα, ένα μήνυμα προσφοράς για περίπου τέσσερις πυρήνες έχει μέγεθος έως και 20 $bytes$, ενώ η απάντηση σε αυτή την προσφορά είναι μόνο 1 $byte$. Τέλος, το προτεινόμενο πλαισιο επιβαρύνει το δίκτυο κατά 66%.

Το Σχήμα 4.17 παρουσιάζει τη μέση επιτάχυνση των εφαρμογών χρησιμοποιώντας τη συνάρτηση που παρουσιάζεται στο [54]. Ως επιτάχυνση ορίζεται ως ο λόγος των συνολικών κύκλων προς το συνολικό φόρτο εργασίας. Το προτεινόμενο πλαίσιο επιτυγχάνει κατά μέσο όρο 20% καλύτερη επιτάχυνση των εφαρμογών από τον DistRM. Αυτό μπορεί να εξηγηθεί από το γεγονός οι πυρήνες δεν διακόπτονται τόσο συχνά από τα μηνύματα κατά την εκτέλεση των εφαρμογών και έτσι οι εφαρμογές ολοκληρώνονται γρηγορότερα. Από την άλλη πλευρά, λόγω του μεγάλου αριθμού των μηνυμάτων που αποστέλλονται στον DistRM, οι πυρήνες σταματούν πιο συχνά καθυστερώντας έτσι την εκτέλεση.

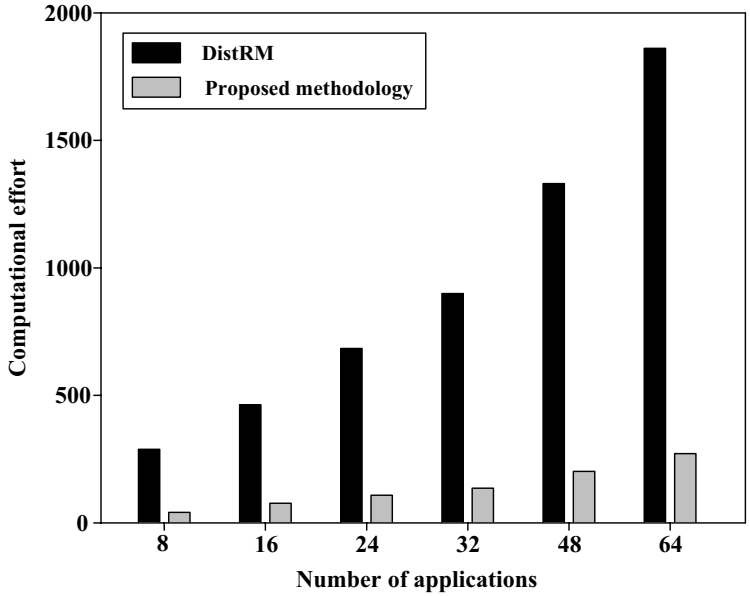
Το Σχήμα 4.18 παρουσιάζει τη σύγκριση του υπολογιστικού προσπάθεια μεταξύ του προεινόμενου πλαισίου και του DistRM. Υπολογιστική προσπάθεια ορίζεται ως ο συνολικός αριθμός κλήσης των συναρτήσεων επιτάχυνσης σε όλη τη διάρκεια της προσομοίωσης. Το πλαίσιο που παρουσιάστηκε έχει κατά μέσο όρο 85% μικρότερο αριθμό




Σχήμα 4.16: Συνολικό μέγεθος μηνυμάτων σε *bytes* σε σύγκριση με τον αλγόριθμο DistRM [54]




Σχήμα 4.17: Μέση επιτάχυνση των εφαρμογων με βάση τη συνάρτηση που παρουσιάζεται στο [54].




Σχήμα 4.18: Σύγκριση υπολογιστικής προσπάθειας σε σύγκριση με τον DistRM [54]

κλήσεων των συναρτησεων επιτάχυνσης από τον DistRM. Αυτό μπορεί να εξηγηθεί από το γεγονός ότι, όπως προαναφέρθηκε, το πλαίσιο που παρουσιάζεται εκτελεί λιγότερες διαδικασίες αυτο-βελτιστοποίησης ανά εφαρμογή λόγω ειδικών κριτηρίων. Έτσι, οι πυρήνες διαχειριστές προχωρούν στον υπολογισμό της συνάρτησης λιγότερο συχνά.

Συνοψίζοντας, σε αυτό το κεφάλαιο παρουσιάστηκε ένα μεθοδολογικό πλαίσιο για κατανεμημένη διαχείριση πόρων σε πολυπύρηνες πλατφόρμες κατα τη φάση εκτέλεσης για εύπλαστες παράλληλες εφαρμογές. Το προτεινόμενο πλαίσιο βασίζεται στην ιδέα των τοπικών ελεγκτών και διαχειριστών, ενώ το σύστημα επικοινωνίας εξασφαλίζει τη κατανεμημένη λειτουργικότητα. Το μεθοδολογικό πλαίσιο είναι υπεύθυνο (i) για την εξυπηρέτηση, κατά το χρόνο εκτέλεσης, των αναγκών των εφαρμογών, (ii) εξασφαλίζει ότι οι εφαρμογες θα πάρουν το βέλτιστο αριθμό πυρήνων αποφεύγοντας φαινόμενα κυριαρχιας, (iii) λαμβάνει υπόψη το είδος των επεξεργαστών αξιοποιώντας καλύτερα την ετερογένεια και (iv) έχει μικρή επιβάρυνση στη συνολική επικοινωνία των πυρήνων. Το πλαίσιο υλοποιήθηκε ως μέρος ενός C προσομοιωτή και, επιπλέον, ως υπηρεσία στο χρόνο εκτέλεσης σε μια πραγματική πολυπύρηνη πλατφόρμα. Συγκρινόμενο με το DistRM [54], τα πειραματικά αποτελέσματα έδειξαν ότι το προτεινόμενο πλαίσιο έχει κατά μέσο όρο 70% λιγότερα μηνύματα, 64% μικρότερο μέγεθος μηνυμάτων και 20% κέρδος στην επιτάχυνση των εφαρμογών.

# Κεφάλαιο 5

# Διαχείριση πόρων σε αρχιτεκτονικές τύπου Δίκτυο-σε-Ψηφίδα

## 5.1 Εισαγωγή

Όπως αναφέρθηκε και προηγουμένως, τα μελλοντικά ολοκληρωμένα συστήματα θα περιέχουν δισεκατομμυρία τρανζίστορ [76], συνθέτοντας δεκάδες έως εκατοντάδες συστήματα πυρήνων εξυπηρετώντας πολύπλοκες και απαιτητικές εφαρμογές. Μια αρχιτεκτονική ικανή να αντιμετωπίσει αυτές τις προκλήσεις είναι η αρχιτεκτονική Δίκτυο-σε-Ψηφίδα (Network-on-Chip, NoC) [19, 53].

Η διαχείριση πόρων έχει διερευνηθεί σε σύγχρονες πολυπύρηνες αρχιτεκτονικές τόσο στα ενσωματωμένα συστήματα όσο και σε συτήματα cloud. Η αποτελεσματική διαχείριση μεταφράζεται συνήθως σε βελτιστοποίηση της απόδοσης και της κατανάλωσης ισχύος και αντιπροσωπευτικά παραδείγματα παρουσιάζονται στα [9, 23, 66, 69, 96].

Στα κατανεμημένα συστήματα οι περιορισμοί είναι σπάνια τόσο αυστηροί. Οι πόροι, ολόκληροι υπολογιστές σε αυτή την περίπτωση, μπορούν να γίνουν διαθέσιμοι ανά πάσα στιγμή [26]. Μηχανισμοί Marketplace διερευνούνται εκτενώς στα [28, 57] σαν ένας τρόπος κατανεμημένης διαχείρισης. Τα προαναφερθέντα άρθρα αντιμετωπίζουν τους υπολογιστικούς πόρους ως παράγοντες που προσπαθούν να διαχειριστουν το φόρτο εργασίας και τους πόρους του συστήματος όσο το δυνατόν αποτελεσματικότερα.

Οι μελλοντικές ηλεκτρονικές συσκευές ευρείας κατανάλωσης αναμένονται να κινηθούν προς την πολυπλοκότητα των προηγούμενων κατανεμημένων συστημάτων λόγω της αδυναμίας να αυξήσουν την επεξεργαστική ισχύ τους με άλλα μέσα. Η Intel έχει ήδη δημιουργήσει πλατφόρμες με 48 επεξεργαστές γενικής χρήσης [49], ενώ η ST-NXP τονίζει επίσης την ανάγκη για πολλές πολυπήρηρνες αχιτεκτονικές σε συμβατικά μέχρι τώρα ενσωματωμένα συστήματα [87].

Τα Δίκτυα-σε-Ψηφίδα (Network-on-Chip, NoC) έχουν αναγνωριστεί ως το νέο πρότυπο για τη διασύνδεση και την οργάνωση μεγάλου αριθμού πυρήνων. Αν και η έννοια των NoCs έχει προέλθει από την παραδοσιακή διασύνδεση των δικτύων, έχουν κάποια ιδιαίτερα χαρακτηριστικά που τα καθιστούν μοναδικά κατά τη φάση σχεδιασμού. Οι

προκλήσεις σχεδιασμού για NoC είναι: (i) η επικοινωνιακή υποδομή, (ii) η επιλογή κατάλληλου μοντέλου επικοινωνίας και (iii) η βελτιστοποίηση απεικόνισης των εφαρμογών. Η επιλογή της τοπολογίας του δικτύου, η απεικόνιση και η δρομολόγηση εντός του ολοκληρωμένου είναι σημαντικά ζητήματα σχεδιασμού που επηρεάζουν δραματικά την απόδοση του συστήματος συνολικά, όπως τη μέση καθυστέρηση δικτύου και την κατανάλωση ενέργειας.

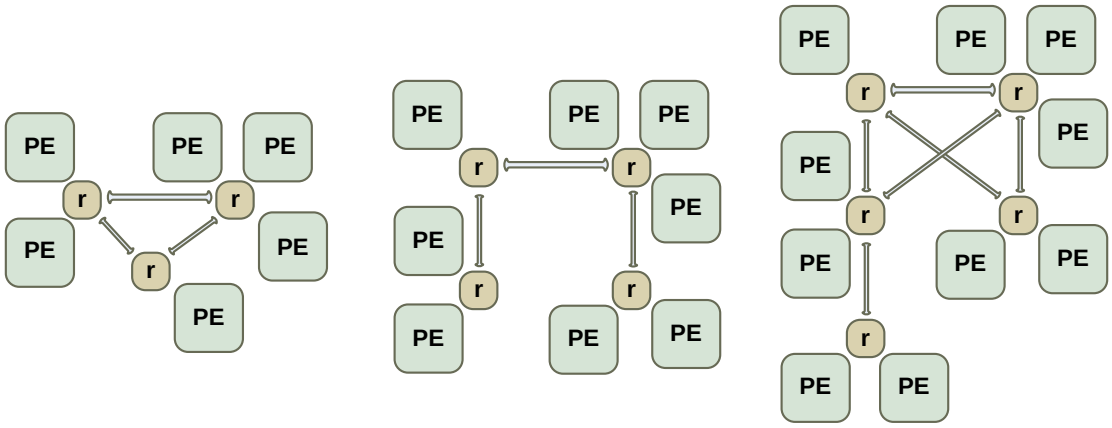
Υπάρχουν δύο βασικά πρότυπα σχετικά με την σχεδίαση NoC αρχιτεκτονικών: (i) Η κανονική (regular) και (ii) η ακανόνιστη (irregular). Η πιο συνηθισμένη κανονική αρχιτεκτονική είναι η τοπολογία πλέγματος στην οποία κάθε δρομολογητής είναι συνδεδεμένος με τέσσερις γειτονικούς δρομολογητές, μέσω ενός αμφίδρομου καναλιού, και έναν πυρήνα. Οι κανονικές τοπολογίες NoC είναι πιο κατάλληλες για πολυπύρηνα συστήματα γενικού σκοπού, τα οποία αποτελούνται κυρίως από ομοιογενή στοιχεία επεξεργασίας και αντίστοιχες συστοιχίες αποθήκευσης και μπορούν να επωφεληθούν από την χωρική τοπικότητα για την επίτευξη υψηλότερων επιδόσεων [36].

Ωστόσο, οι εφαρμογές που εκτελούνται σε κανονικές τοπολογίες NoC δεν αξιοποιούν πλήρως τις δυνατότητες του δικτύου διασύνδεσης. Η ποικιλομορφία της επικοινωνίας στο δίκτυο επηρεάζεται από αρχιτεκτονικά θέματα όπως η σύνθεση του συστήματος και η ομαδοποίηση. Οι ακανόνιστες τοπολογίες NoC εξυπηρετούν καλύτερα τις απαιτήσεις των εφαρμογών, δεδομένου ότι είναι σχεδιασμένες με βάση τις ανάγκες των εκάστοτε εφαρμογών και μεγιστοποιούν τους παράγοντες χρησιμοποίησης όσον αφορά την ισχύ και την καθυστέρηση του δικτύου. Ωστόσο, ο ακανόνιστος σχεδιασμός NoC είναι πιο πολύπλοκος και εξαρτάται από μία ποικιλία παραμέτρων σχεδιασμού σε σχέση με τις κανονικές NoC τοπολογίες.

Στο κεφάλαιο αυτό, παρουσιάζονται ένα πλαίσιο και μεθοδολογίες προσαρμογής υψηλού επιπέδου για (i) την υποστήριξη τόσο κανονικών όσο και ακανόνιστων NoC αρχιτεκτονικών κατά τη φάση σχεδιασμού και (ii) την παραμετροποίηση των εφαρμογών κατά το χρόνο εκτέλεσης. *Ο στόχος είναι να παρασχεθεί στο σχεδιαστή μια ποικιλία από επιλογές, ώστε να δοκιμάσει και να αξιολογήσει διαφορετικές διαμορφώσεις που θα καλύψουν καλύτερα τις απαιτήσεις των εφαρμογών.*

## 5.2 Μεθοδολογία διαχείρισης πόρων σε NoC αρχιτεκτονικές

Το προτεινόμενο πλαίσιο [12] για την υποστήριξη και αξιολόγηση της διαχείρισης των πόρων σε NoC αρχιτεκτονικές, όσον αφορά τις επιδόσεις του δικτύου και την κατανάλωση ενέργειας, βασίζεται σε μια τροποποιημένη έκδοση του προσομοιωτή υψηλού επιπέδου (SystemC) Noxim [2]. Ο προσομοιωτής έχει τροποποιηθεί σε μεγάλο βαθμό προκειμένου να υποστηρίξει τις ακόλουθες υπηρεσίες: (i) προσαρμοσμένες ακανόνιστες τοπολογίες με τους δρομολογητές ως ξεχωριστές οντότητες SystemC, (ii) ένα αρχείο ρυθμίσεων μπορεί να δοθεί ως είσοδος στον προσομοιωτή περιγράφοντας κάθε φορά την NoC τοπολογία, (iii) ο σχεδιαστής μπορεί να καθορίσει το μέγεθος του κάθε καταχωρητή (buffer) σε κάθε δρομολογητή που χρησιμοποιείται στην προσομοίωση με αποτέλεσμα διαφορετική κατανάλωση ενέργειας και καθυστέρηση δικτύου. Αυ-




Σχήμα 5.1: Παραδείγματα ακανόνιστων NoC τοπολογιών. Κάθε δρομολογητής (r) μπορει να εξυπηρετήσει περισσότερα του ενός επεξεργαστικά στοιχεία (PE) έχοντας πολλαπλές πόρτες.

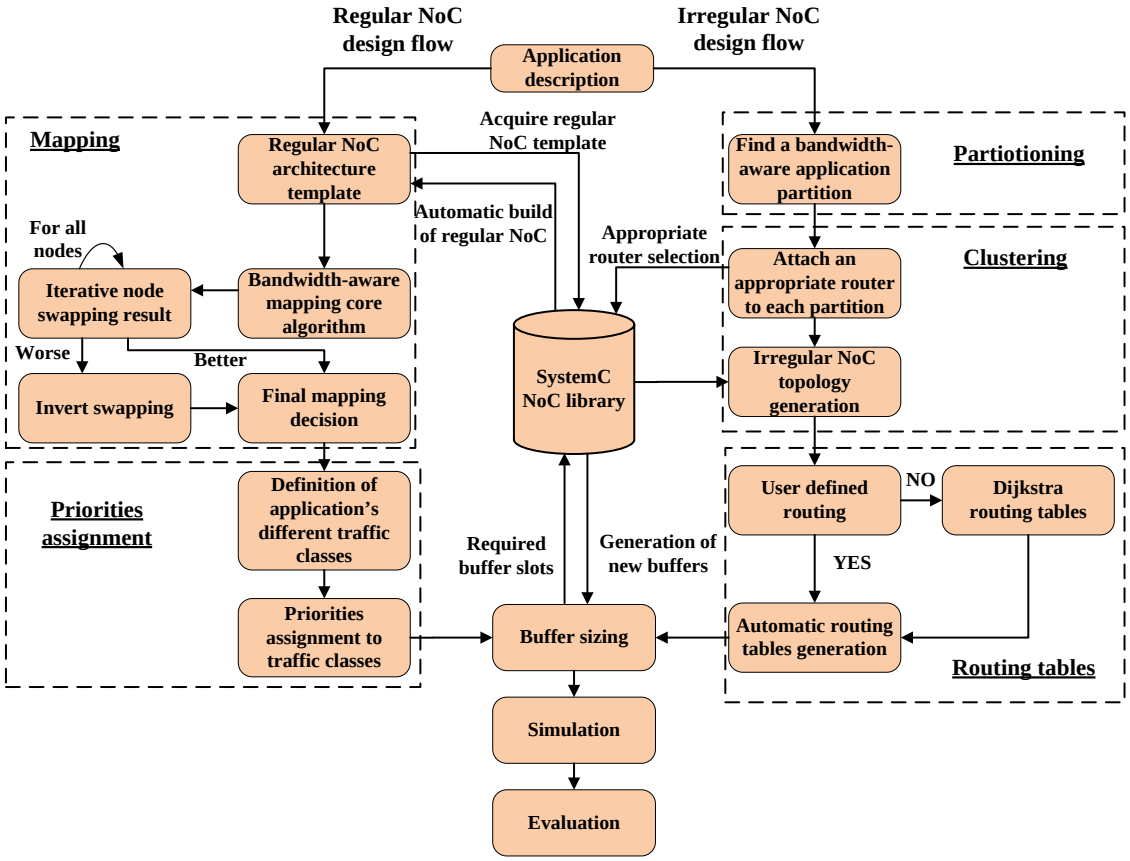
τός ο αριθμός μπορεί να είναι διαφορετικός για διαφορετικούς δρομολογητές στην ίδια προσομοίωση, (iv) το Ebit μοντέλο ενέργειας [93] έχει προστεθεί προκειμένου να γίνει μια αξιολόγηση της κατανάλωσης ενέργειας του δικτύου και (v) προσαρμοσμένους πίνακες δρομολόγησης οι οποίοι ορίζονται από το σχεδιαστή σε ένα ξεχωριστό αρχείο. Σε περίπτωση που ο σχεδιαστής δεν επιλέξει προσαρμοσμένη δρομολόγηση, εφαρμόζεται ο αλγόριθμος μικρότερου μονοπατιού Dijkstra για ακανόνιστες τοπολογιες ενώ για τις κανονικές χρησιμοποιείται ο XY αλγόριθμος δρομολόγησης. Για την διευκρίνηση του όρου ακανόνιστες τοπολογιες NoC, ορίζεται οποαοδήποτε προσαρμοσμένη τοπολογία 2D NoC η οποία χρησιμοποιεί δρομολογητές με πολλαπλές θύρες, εκτός από αυτές που χρησιμοποιούνται για τις τοπολογιες πλέγματος και τόρου. Το Σχήμα 5.1 δείχνει ένα παράδειγμα τέτοιων ακανόνιστων τοπολογιών .

Μια επισκόπηση του αναπτυγμένου πλαισίου για το σχεδιασμό προσαρμοσμένων κανονικών και ακανόνιστων αρχιτεκτονικών NoC παρουσιάζεται στο Σχήμα 5.2. Έχοντας ως είσοδο μια υψηλού επιπέδου περιγραφή της εφαρμογής, ο σχεδιαστής επιλέγει αν θα ακολουθήσει την ροή σχεδιασμού για κανονικές και ακανόνιστες τοπολογίες. Η απόφαση για την επιλογή της κατάλληλης ροής σχεδιασμού εξαρτάται από τους στόχους του σχεδιαστή και τους περιορισμούς και τις απαιτήσεις της εφαρμογής.

Κατά τη ροή σχεδιασμού κανονικών NoC τοπολογιών, το πρώτο βήμα είναι η επιλογή του κατάλληλου πρότυπου αρχιτεκτονικής (πλέγμα, τόρος) και στη συνέχεια εφαρμόζεται ένας αλγόριθμος απεικόνισης με στόχο την ελαχιστοποίηση της επικοινωνίας, ώστε να αξιοποιήσει αποτελεσματικά τα χαρακτηριστικά της πλατφόρμας. Στη συνέχεια, με σκοπό την περαιτέρω προσαρμογή στους περιορισμούς και τις απαιτήσεις της εφαρμογής, ένα συστημα ανάθεσης προτεραιοτήτων παρέχει σε συγκεκριμένες ροές κίνησης την απαιτούμενη μεταχείριση.

Στην ακανόνιστη ροή του σχεδιασμού NoC, το πρώτο βήμα είναι η διαμέριση της εφαρμογής και στη συνέχεια ακολουθεί η διαδικασία της ομαδοποίησης. Στο βήμα

της διαμέρισης, οι εργασίες της εφαρμογής χωρίζονται σε διαφορετικές λογικές κατατμήσεις με βάση το απαιτούμενο εύρος ζώνης επικοινωνίας. Στη συνέχεια, στο πλαίσιο της διαδικασίας ομαδοποίησης, ο σχεδιαστής μπορεί να τοποθετήσει έναν κατάλληλο δρομολογητή σε κάθε διαμέριση, συνδέοντας μια βιβλιοθήκη SystemC παράγοντας έτσι την ακανόνιστη τοπολογία. Επιπλέον, ο σχεδιαστής μπορεί να καθορίσει έναν προσαρμοσμένο αλγόριθμο δρομολόγησης, ορίζοντας τους δικούς τους πίνακες δρομολόγησης, ή να επιλέξει το την αυτόματη δρομολόγηση με χρήση του αλγορίθμου συντομότερου μονοπατιού Dijkstra.




Σχήμα 5.2: Προτεινόμενο πλαισιο για την προσομοίωση και παραγωγή προσαρμοσμένων αρχιτεκτονικών NoC [12]

Τέλος, τόσο για τις κανονικές όσο και για τις ακανόνιστες τοπολογίες, το προτεινόμενο πλαίσιο υποστηρίζει τη μείωση της κατανάλωσης ενέργειας μέσω της μείωσης του αριθμού των περιττών θέσεων στους καταχωρητές των δρομολογητών εξυπηρετώντας πάντα τις ανάγκες της εφαρμογής.

Όπως αναφέρθηκε και προηγουμένως, ένας γράφος (Application Task Graph, ATG) χρησιμοποιείται για να περιγράψει την ροή της επικοινωνίας. Η δομή ATG είναι ένας συνηθισμένος τρόπος για την παρουσιάση των περιορισμών και των απαιτήσεων της εφαρμογής και έχει χρησιμοποιηθεί ευρέως σε εργαλεία σχεδιασμού [21, 50, 65] και για την προσαρμογή κατά τη φάση εκτέλεσης [9, 54] σε NoC αρχιτεκτονικές . Ο ATG $G(T, D)$ είναι ένας κατευθυνομενος ακυκλικός γράφος, όπους κάθε κορυφη $t_i$

ανιπροσωπεύει ένα υπολογιστικό φορτίο της εφαρμογής. Κάθε εργασία $t_i$ παρέχει πληροφορίες σχετικά με τα χαρακτηριστικά της και τις απαιτήσεις της. Κάθε κατευθυνόμενη ακμή $d_{i,j} \in D$ μεταξύ των εργασιών $t_i$ και $t_j$ χαρακτηρίζει τις εξαρτήσεις στην επικοινωνία και στα δεδομένα. Κάθε $d_{i,j}$ ακολουθείται απο μία τιμή $b(d_{i,j})$, ηοποία αντιπροσωπεύει τον όγκο επικοινωνίας που ανταλλάσσονται μεταξύ των εργασιών $t_i$ και $t_j$.

Μία NoC τοπολογία μπορει να περιγραφεί μοναδικά απο ένα συνδεδεμένο κατευθυνόμενο γράφο $A(I, N)$. Το σύνολο των κορυφών $N$ αποτελείται απο δύο αμοιβαιώς αποκλειόμενα υποσύνολα $N_{PE}$ και $N_C$ τα οποία περιλαμβάνουν τα διαθέσιμα επεξεργαστικά στοιχεία και τα στοιχεία διασύνδεσης αντίστοιχα. Το σύνολο των ακμών $I$ περιλαμβάνει τις πληροφορίες διασύνδεσης για το σύνολο $N$.

### 5.2.1 Διαχείριση πόρων σε κανονικές NoC τοπολογίες

Κατά τη ροή σχεδιασμού κανονικών τοπολογιών, το προτεινόμενο πλαίσιο υποστηρίζει τη χρησιμοποίηση ενός συστήματος ανάθεσης προτεραιοτήτων.

Με την παροχή των προτεραιοτήτων σε διαφορετικές εφαρμογές, ροές δεδομένων ή κατηγοριών κίνησης, ο σχεδιαστής μπορεί να εγγυηθεί ένα ορισμένο επίπεδο απόδοσης του συστήματος. Σε πολλές εφαρμογές είναι επιθυμητό ορισμένες κατηγορίες κίνησης δεδομένων να έχουν μια προνομιακή μεταχείριση προκειμένου να εξυπηρετηθούν οι προθεσμίες και προδιαγραφές ποιότητας (Quality of Service, QoS) της εφαρμογής. Σε ένα σύστημα προτεραιοτήτων, οι ροές δεδομένων χωρίζονται σε $K \geq 2$ κλάσσεις $1, 2, \ldots, K$, όπου η χαμηλότερη τιμή δηλώνει και τη μεγαλύτερη προτεραιότητα.

Το προτεινόμενο πλαίσιο διαχειρισης πόρων παρέχει τέσσερις κλάσσεις προτεραιοτήτων. Δύο bit έχουν προστεθεί σε κάθε flit ως πληροφορίες επικεφαλίδας. Αυτά τα δύο bits δηλώνουν το είδος προτεραιότητας κάθε flit (00 είναι η υψηλότερη και 11 είναι η χαμηλότερη προτεραιότητα) και υποδεικνύουν τον τρόπο με τον οποίο το flit θα εξυπηρετηθεί στους καταχωρητές εισόδου των δρομολογητών. Η υλοποίηση μηχανισμού προτεραιοτήτων σε NoC αρχιτεκτονικές επηρεάζεται από (i) τον αλγόριθμο δρομολόγησης wormhole switching και (ii) την in-order παράδοση πακέτων. Το προτεινόμενο πλαισιο χρησιμοποιεί πολιτική διαχείρισης τύπου *non-preemptive* κατά την οποία ένα πακέτο υψηλής προτεραιότητας μπορεί να κινηθεί μπροστά από όλα τα χαμηλής προτεραιότητας που περιμένουν στην ουρά, αλλά τα πακέτα χαμηλής προτεραιότητας που εξυπηρετούνται ήδη δεν επηρεάζονται.

Ένας δρομολογητής στον αλγόριθμο δρομολόγησης wormhole switching μπορει να περιγραφεί σαν μία μεταβλητή δύο καταστάσεων $g \in 1 \geq \Re^2 \geq 0$, όπου το $g(1)$ αντιπροσωπεύει την κατάσταση αναμονής και $g(2)$ την κατάσταση λειτουργίας. Στον αλγόριθμο δρομολόγησης wormhole switching, ο δρομολογητής δεν διακόπτει τη σύνδεση αν δεν έχει εξυπηρετήσει το σύνολο των πακέτων. Ορίζουμε ως $W_{k_m}$ το χρόνο αναμονής του $m$ flit του πακέτου με προτεραιότητα $k$, ως $T_s$ το χρόνο που εξυπηρετείται ένα flit και ως $W_s$ το χρόνο αναμονής ενός flit στο δρομολογητή που είναι

στην κατάσταση λειτουργίας. Αυτός ο χρόνος ειναι ίδιος για όλα τα flits της ίδιας κλάσσης. $n$ ειναι ο αριθμός των flits ανά πακέτο ($n > 1$) και $k$ το id της κλάσσης προτεραιοτήτων ($k \in [1,4]$). Ο χρόνος αναμονής του $m$ flit για τις δύο καταστάσεις περιγράφεται στις Εξισώσεις 5.1 και 5.2.

$$W_{k_m}^{g(1)} = \underbrace{\sum_{i=1}^{k-1} n(W_i + T_s)}_{\text{Flits με υψηλότερη προτεραιότητα}} + \underbrace{\sum_{l=1}^{m-1} l(W_k + T_s)}_{\text{Προπορευόμενα flits του ίδιου πακέτου}} + \underbrace{A \times (W_k + Ts)}_{\text{Flits άλλων πακέτων με την ίδια προτεραιότητα που έφτασαν πρώτα}} \tag{5.1}$$

$$W_{k_m}^{g(2)} = \underbrace{T_s}_{\text{Flit που εξυπηρετείται}} + \underbrace{j(W_s + T_s)}_{\text{Τα εναπομείναντα flits του πακέτου}} + \underbrace{\sum_{i=1}^{k-1} n(W_i + T_s)}_{\text{Flits με υψηλότερη προτεραιότητα}} + \underbrace{\sum_{l=1}^{m-1} l(W_k + T_s)}_{\text{Προπορευόμενα flits του ίδιου πακέτου}} + \underbrace{A \times (W_k + Ts)}_{\text{Flits άλλων πακέτων με την ίδια προτεραιότητα που έφτασαν πρώτα}} \tag{5.2}$$

where $j \in [0, n-1]$ and $A \in Z$.

Κάθε καταχωρητής έχει ένα προκαθορισμένο μέγεθος $|B|$ θέσεων που μπορεί να εξυπηρετήσει. Σύμφωνα με τις Εξισώσεις 5.1 και 5.2 μπορούμε να εξάγουμε, στην Εξίσωση 5.3, ότι προκειμένου οι προτεραιότητες να έχουν αποτέλεσμα, το μέγεθος των καταχωρητών εισόδου πρέπει να είναι μεγαλύτερο από τον αριθμό των flits ανά πακέτο.

$$j + n + \sum_{m-1}^{l=1} l + N < |B| \Rightarrow 0 < n < |B| \tag{5.3}$$

Όπου $n \geq |B|$, $W_{k_m}^{g(1)}$ και $W_{k_m}^{g(2)}$ ακολουθούν την Εξίσωση 5.4.

$$W_{k_m}^{g(1)} = W_{k_m}^{g(2)} = T_s + j(W_s + T_s) \tag{5.4}$$

η οποία ειναι ανεξάρτητη του $K$. Με αυτόν τον τρόπο, ακόμη και αν οι προτεραιότητες υπάρχουν, στην πραγματικότητα δεν χρησιμοποιούνται.

Κατά την υλοποίηση προτεραιοτήτων σε NoC, μπορούν να εμφανιστούν φαινόμενα αποκλεισμού κατά τη διάρκεια των οποίων πακέτα με χαμηλή προτεραιότητα δεν

εξυπηρετούνται ποτέ λόγω της συνεχόμενης άφιξης πακέτων με υψηλότερη προτεραιότητα. Προκειμένου να αποφεύγονται αυτά τα φαινόμενα έχουμε προσθέσει σε κάθε flit ένα πεδίο που ονομάζεται $waiting\_time$ που ξεκινά από το μηδέν και αυξάνει κάθε φορά που το πακέτο δεν επεξεργάζεται απο κάποιον δρομολογητη. Όταν η τιμή αυτή ξεπεράσει ένα όριο, τότε η κλάσση προτεραιότητας του συγκεκριμένου flit μειώνεται μέχρι να φτάσει στο μηδέν που είναι το υψηλότερο είδος προτεραιότητας. Έτσι, τα νέα εισερχόμενα flits θα τοποθετηθούν πίσω απο αυτό.

Ο προτεινόμενος αλγόριθμος διαχείρισης flits με τη χρήση προτεραιοτήτων, παρουσιάζεται στον Αλγόριθμο 5. Αρχικά, ο αλγόριθμος ελέγχει αν το flit έχει τη χαμηλότερη προτεραιότητα (γραμμή 2). Αν ναι, το flit τοποθετείται στην τελευταία θέση της ουράς (γραμμή 3). Εάν αυτό δεν συμβαίνει, τότε ελέγχεται αν κάποιο άλλο flit είναι υπό επεξεργασία (γραμμή 4). Αν αυτή είναι η περίπτωση (γραμμή 5), γίνεται αναζήτηση στον καταχωρητή (και υποθέτοντας ότι τα flits του πακέτου που είναι υπό επεξεργασία έχουν την υψηλότερη προτεραιότητα), το εισερχόμενο flit τοποθετείται στην πρώτη κατάλληλη θέση με βάση την προτεραιότητα του. Στη γραμμή 8 ελέγχουμε αν έχουμε την ίδια προτεραιότητα με το πρώτο flit του καταχωρητή. Μετά από αυτό, στη γραμμή 9, ψάχνουμε τον καταχωρητή με βάση τον αριθμό σειράς των flits και επιβεβαιώνουμε τη θέση του. Σε όλες τις άλλες περιπτώσεις (γραμμές 11-13) ψάχνουμε ολόκληρο τον καταχωρητή και τοποθετούμε σωστά το flit με βάση την προτεραιότητά του. Μετά την τοποθέτηση του flit, ο αλγοριθμος ελέγχει τις τιμές $wating\_time$ των flits ώστε να ενημερώσει, αν χρειάζεται, κάποια τιμή προτεραιότητας (γραμμές 17-24).

**Algorithm 5** Αλγόριθμος διαχείρισης flits με βάση την προτεραιότητα

```
 1: if flit.priority == LOW then
 2:     buffer.put_back(flit)
 3: else
 4:     if buffer.is_processing() then
 5:         buffer.search_by_priority()
 6:         buffer.insert(flit)
 7:     else
 8:         if first == flit.priority then
 9:             buffer.search_by_position()
10:             buffer.insert(flit)
11:         else
12:             buffer.search()
13:             buffer.insert(flit)
14:         end if
15:     end if
16: end if
17: p=buffer.head
18: while p!=NULL do
19:     if p.wating_time > threshold then
20:         p.priotiry−−
21:     else
22:         p.wating_time + +
23:     end if
24:     p=p.next
25: end while
```

### 5.2.2 Διαχείριση πόρων σε ετερογενείς NoC αρχιτεκτονικές

Το προτεινόμενο πλαίσιο για τη διαχείριση πόρων σε ακανόνιστες NoC αρχιτεκτονικές προσφέρει στον σχεδιαστή τη δυνατότητα να εκμεταλλευτεί πλήρως τις δυνατότητες του δικτύου διασύνδεσης κατά την αυτόματη δημιουργία εξατομικευμένων τοπολογιών. Σε αντίθεση με τις κανονικές NoC τοπολογίες, οι ακανόνιστες τοπολογίες εξαρτόνται από την εφαρμογή και η διαδικασία προσαρμογής απαιτεί πολύπλοκες ενέργειες, όπως (i) διαμέριση εφαρμογής, (ii) ομαδοποίηση και (iii) δημιουργία πίνακα δρομολόγησης. Όπως προαναφέρθηκε, στόχος του προτεινόμενου πλαισίου ειναι να δώσει στον σχεδιαστή ένα ευρος επιλογών προκειμένου να δοκιμάσει και να αξιολογήσει μια ποικιλία από διαφορετικές διαμορφώσεις και να επιλέξει αυτή που ταιριάζει καλύτερα στις ανάγκες του.

#### 5.2.2.1 Διαμέριση εφαρμογής

Το πρώτο βήμα αποτελει η διαδικασια διαμέρισης της εφαρμογής με κριτήριο το κόστος επικοινωνίας των εργασιών. Μια προσεκτική και ακριβής ανάλυση των χαρακτηριστικών της εφαρμογής και κυρίως το εύρος ζώνης των εργασιών και της επικοινωνίας είναι απαραίτητη, προκειμένου να καθορίστει ο χώρος σχεδιασμού.

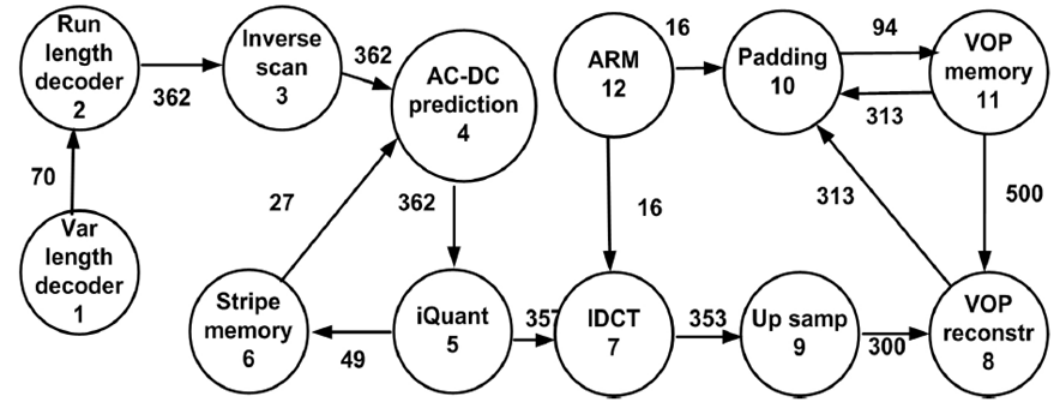
Ορίζουμε ως $R$ το σύνολο που περιγράφει τις λογικές διαμερίσεις της εφαρμογής. Το σύνολο $R$ αποτελείται απο $m$ $(m \geqslant 1)$ υποσύνολα $R_1, R_2, ..., R_i, ..., R_m$ τέτοια ώστε $\bigcap_{i=1}^{m} R_i = \emptyset$ και $\bigcup_{i=1}^{m} R_i = R$. Μία διαμέριση $R_i$ θεωρείτε έγκυρη αν $\exists t_i \in R_i : t_i \in T$. Ορίζουμε ως $K[R_i] = \sum b(d_{i,j}) \ \forall \ t_i, t_j \in R_i$ το κόστος επικοινωνίας μέσα στη διαμέριση $R_i$ και σαν $Q[R_i R_j] = \sum b(d_{i,j}) \ \forall \ t_i \in R_i, \ t_j \in R_j$ το κόστος επικοινωνίας μεταξύ των $R_i$ και $R_j$ διαμερίσεων.

Ο στόχος του σταδίου διαμέρισης της εφαρμογής είναι η δημιουργία του σωστού $R$ ώστε (i) να ελαχιστοποιηθεί το κόστος επικοινωνίας μεταξύ των διαμερίσεων και (ii) η εξισορρόπηση του φορτίου μεταξύ των διαμερίσεων, όπως περιγράφεται στην Εξίσωση 5.5.

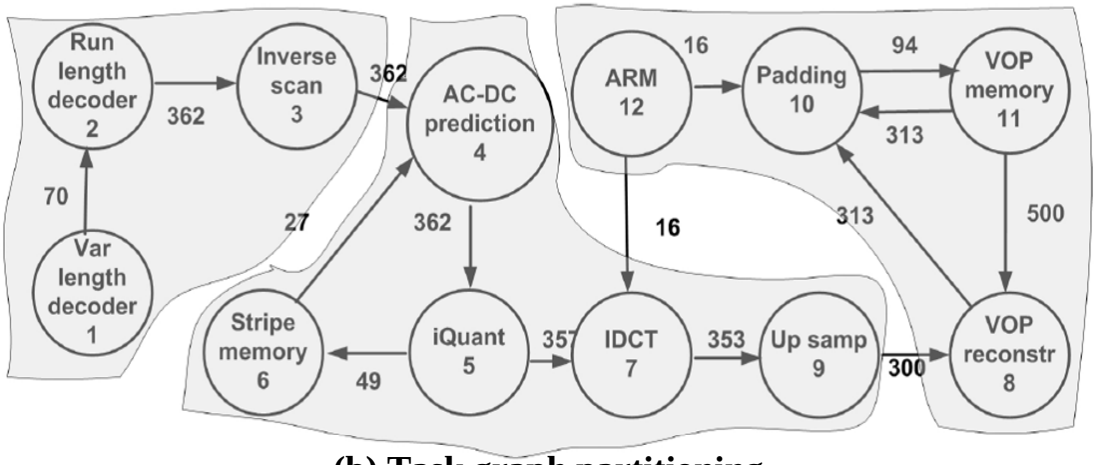
$$\begin{aligned} R = \{R_1, R_2, ..., R_m\} : \{K[R_1] \simeq K[R_2] \simeq ... \simeq K[R_m]\} \wedge \\ \wedge \, min\{Q[R_1 R_2], Q[R_1 R_3], ..., Q[R_1 R_m], ..., Q[R_{m-1} R_m]\} \end{aligned} \tag{5.5}$$

Το πρόβλημα διαμέρισης της εφαρμογής είναι NP-complete, και ως εκ τούτου δεν λύνεται σε πολυωνυμικό χρόνο. Ο αλγόριθμος που χρησιμοποιήθηκε είναι ο $Multilevel-KL$ [45]. Η είσοδος στον αλγόριθμο $Multilevel-KL$ είναι ένα γράφος τύπου ATG, όπως η εφαρμογή VOPD που φαίνεται στο Σχήμα 5.3(a). Ένα παράδειγμα της διαδικασίας διαμέρισης παρουσιάζεται στο Σχήμα 5.3(b).

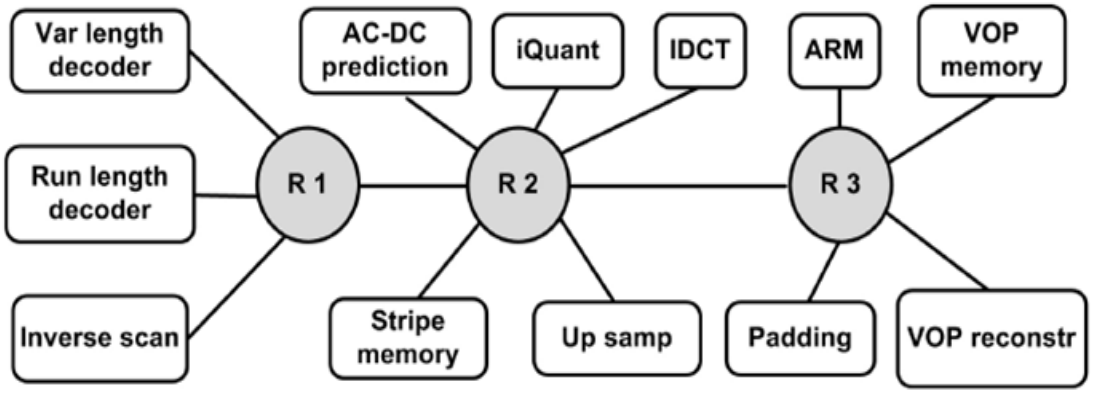
Το προτεινόμενο πλαίσιο υποστηρίζει την αυτόματη διαμέριση εφαρμογών με την ενσωμάτωση του εργαλείου Chaco [1]. Με την αυτόματοποιημένη διαδικασία, το προτεινόμενο πλαίσιο μπορεί να δημιουργήσει πολλαπλές διαμερίσεις ανάλογα με το




**(a) VOPD task graph**




**(b) Task graph partitioning**




**(c) Clustering**

Σχήμα 5.3: VOPD (a) γράφος εφαρμογης, (b) διαμέριση σύμφωνα με τον αλγόριθμο $Multilevel - KL$ [45] και (c) ομαδοποίηση.

```
%Topology table file
%4 Routers, 8 PE's
%port in, port out, buffer in, buffer out, Router, layer
R0: 1,2,4,2,R1,0 2,6,3,4,R2,0 ; 3,9,4,P7,0 ;
R1: 1,3,4,8,R3,0 ; 3,2,8,P5,0 4,2,8,P4,0 5,2,8,P3,0 ;
R2: 9,5,4,5,R3,0 ; 2,4,4,P6,0 ;
R3: ; 6,4,4,P0,0 4,4,4,P1,0 9,5,5,P2,0 ;
```

Σχήμα 5.4: Δομή αρχείου ρυθμίσεων

μέγεθος του ATG και μπορεί να χρησιμοποιηθεί για μια γρήγορη εξερεύνηση του χώρου σχεδιασμού στις ακανόνιστες NoC τοπολογίες.

#### 5.2.2.2 Ομαδοποίηση

Κατά τη διαδικασία της ομαδοποίησης, οι διαμερίσεις της εφαρμογής που παρήχθησαν χρησιμοποιούνται ως είσοδος στο SystemC εργαλείο. Οι στόχοι της διαδικασίας ομαδοποίησης είναι (i) η τοποθέτηση δρομολογητή με κατάλληλο αριθμό θυρών για την εξυπηρέτηση των διαμερίσεων και (ii) η διασύνδεση των δρομολογητών μεταξύ τους (Σχήμα 5.3(γ)). Σύμφωνα με τα καθορισμένα $K[R_i]$ και $Q[R_iR_j]$ για κάθε $R_i$, το κατάλληλο μοντέλο δρομολογητή επιλέγεται έτσι ώστε να είναι σε θέση να εξυπηρετήσει τις ανάγκες επικοινωνίας. Στην αναπτυχθείσα βιβλιοθήκη SystemC κάθε δρομολογητής μπορεί να προσαρμοστεί ανάλογα με (i) τον αριθμό των θυρών εισόδου/εξόδου, (ii) τον έλεγχο σύνδεσης (link control), (iii) το τύπο του διακλαδωτη (switch), (iv) τη δρομολόγηση και (iv) τις μονάδες διαιτησίας. Η εξερεύνηση των ακανόνιστων NoC τοπολογιών απαιτεί διαφορετικό αριθμό συνδέσεων για κάθε δρομολογητή ώστε να μεγιστοποιηθεί η απόδοση του δικτύου. Επίσης, στο στάδιο της ομαδοποίησης, ο σχεδιαστής μπορεί να καθορίσει επιπλέον χαρακτηριστικά του δικτύου, όπως το πλάτος καναλιού, flits ανά πακέτο, μέγεθος καταχωρητή εισόδου και εξόδου κλπ.

Το προτεινόμενο πλαίσιο υποστηρίζει μια ποικιλία μοντέλων απο SystemC δρομολογητές με διαφορετικά χαρακτηριστικά, οι οποίοι, με βάση την προαναφερθείσα διαδικασία διαμέρισης, μπορούν να χρησιμοποιηθούν για μια γρήγορη και αυτόματη εξερεύνηση του χώρου σχεδιασμού. Ο ορισμός της ακανόνιστης τοπολογίας στον προσομοιωτή γίνεται με τη χρήση ενός αρχείου ρυθμίσεων, όπως αυτό παρουσιάζεται στο Σχήμα 5.4. Η τοπολογία που παράγεται από αυτό το αρχείο φαίνεται στο Σχήμα 5.5.

Στο παράδειγμα του Σχήματος 5.4:

1. Ο δρομολογητής 0 (R0) συνδέεται με το δρομολογητή 1(R1), το δρομολογητή 2

και το επεξεργαστικό στοιχειο 7.

2. Η σύνδεση R0-R1 εγκαθιδρύεται μεταξύ των θυρών 1 του R0 και της θύρας 2 του R1.

3. Ο καταχωρητής που εξυπηρετεί την κίνηση απο τον R1 στον R0 έχει 4 θέσεις και για τα flits που φτάνουν απο τον R0 ο R1 έχει 2 θέσεις στον R1.

4. Για τη σύνδεση R0-R2, υπάρχουν 3 θέσεις στην θύρα 2 στην πλευρά του R0 και 4 θέσεις στην θύρα 6 στην πλευρά του R2.

5. Ο R0 συνδέεται με το P7 χρησιμοποιώντας την τρίτη θύρα του και ένα καταχωρητή 9 θέσεων. Τα επεξεργαστικά στοιχεία έχουν μόνο μία θύρα.

6. Ομοίως, ο R1 συνδέεται με τον R3 και τρία επεξεργαστικά στοιχεια (P3, P4, P5), εκτός από τη σύνδεση με τον R0 που δηλώθηκε στην προηγούμενη γραμμή.

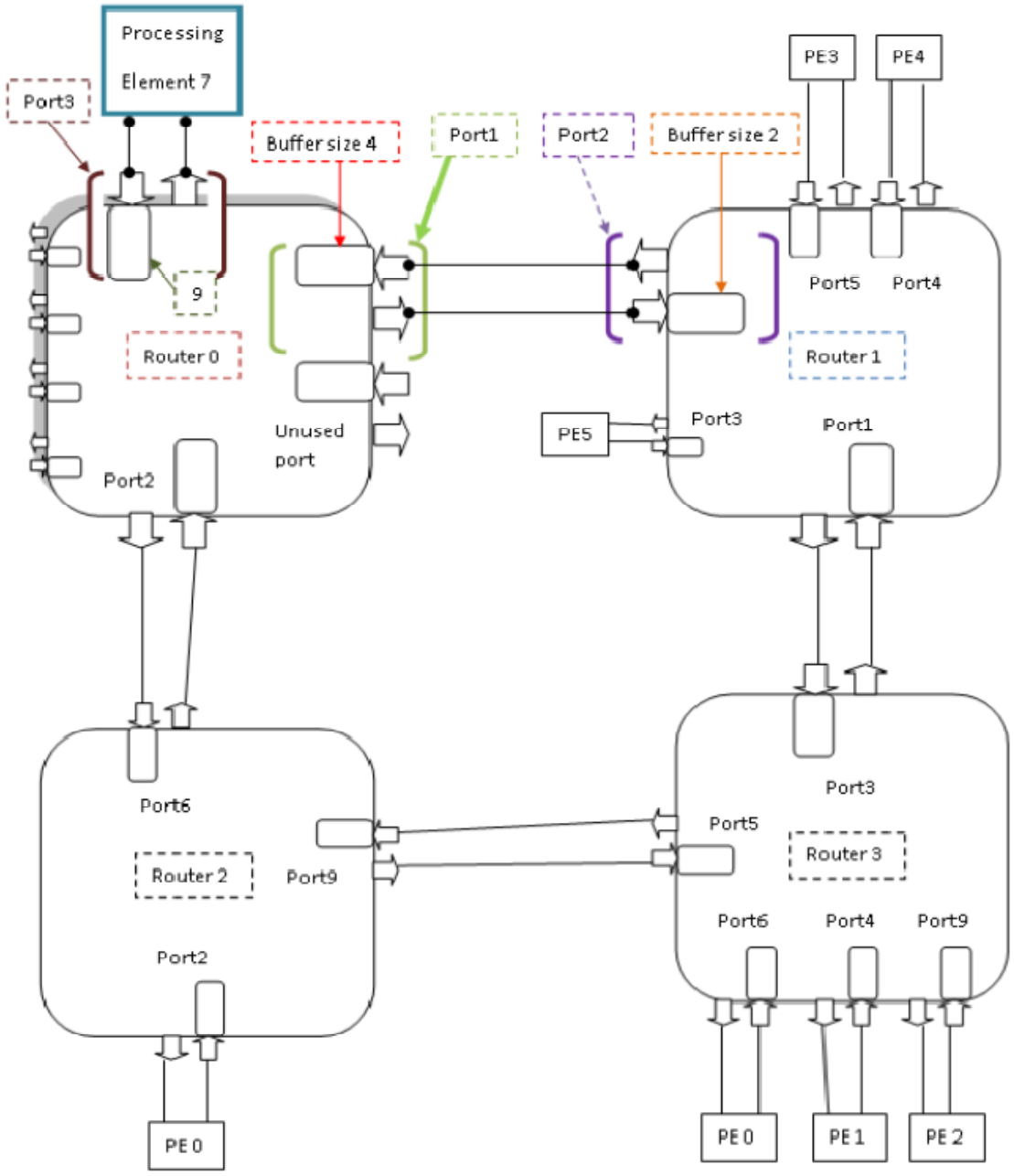
#### 5.2.2.3 Δημιουργία πινάκων δρομολόγησης

Στις ακανόνιστες NoC αρχιτεκτονικές, οι συμβατικοί αλγόριθμοι δρομολόγησης είναι δύσκολο να εφαρμοστούν λόγω των διαφορετικών χαρακτηριστικών και του γεγονότος ότι πρέπει να τροποποιηθούν ειδικά για να εξυπηρετήσουν τη συγκεκριμένη ακανόνιστη τοπολογία. Εντούτοις, μια μικρή αλλαγή στην τοπολογία οδηγεί σε συνολικό επανασχεδιασμό του αλγόριθμου δρομολόγησης. Αυτό, έχει ως αποτέλεσμα μια χρονοβόρα φάση σχεδιασμού και είναι πολύ πιθανό να δημιουργήσει προβλήματα δρομολόγησης, όπως αδιέξοδα και σημεία συμφόρησης.

Το προτεινόμενο πλαίσιο υποστηρίζει δύο τύπους πινάκων δρομολόγησης: (i) οριζόμενους απο το σχεδιαστή και (ii) Dijkstra πίνακες δρομολόγησης. Στην πρώτη περίπτωση, ο σχεδιαστής δημιουργεί, μέσω των αντίστοιχων διεπαφών, τους πίνακες δρομολόγησης. Ωστόσο, η επιλογή αυτή δεν μπορεί να εγγυηθεί την έλλειψη τυχόν λαθών στη διαδικασία δρομολόγησης. Επιλέγοντας τη δεύτερη επιλογή, το προτεινόμενο πλαίσιο δημιουργεί αυτόματα πίνακες δρομολόγησης για ακανόνιστες NoC τοπολογίες με βάση τον αλγόριθμο συντομότερου μονοπατιού Dijkstra. Τα οφέλη της αυτόματης δημιουργίας πινάκων δρομολόγησης είναι: (i) ταχύτερη υλοποίηση του σχεδιασμού, (ii) αποφυγή αδιεξόδων και σημείων συμφόρησης, (iii) απομάκρυνση άχρηστων εγγραφών, (iv) εγγυημένη συντομότερη διαδρομή και (v) εύκολα κατανοητή δομή.

### 5.2.3 Διαχείριση μεγέθους των καταχωρητών στους δρομολογητές

Η κατανάλωση ενέργειας είναι ένας πολύ σημαντικός παράγοντας κατα την αξιολόγηση NoC αρχιτεκτονικών αφού οι καταχωρητές εισόδου,ο τύπος του διακλαδωτή και το συστημα δρομολόγησης κυριαρχούν στην κατανάλωση ισχύος του δικτύου [79]. Επομένως, μια αποτελεσματική διαχείριση των πόρων στις NoC αρχιτεκτονικές απαιτεί χαμηλή κατανάλωση ενέργειας. Μια τεχνική για την αποτελεσματική μείωση της




Σχήμα 5.5: Παράδειγμα της παραγόμενης τοπολογίας χρησιμοποιώντας το αρχείο ρυθμίσεων του Σχήματος 5.4

κατανάλωσης ενέργειας του δικτύου είναι η μείωση του μεγέθους των καταχωρητών στους δρομολογητές χωρίς να επηρεάζονται οι απαιτησεις της εφαρμογής. Σαν τελευταίο βήμα, το προτεινόμενο πλαίσιο χρησιμοποιεί έναν αλγόριθμο μείωσης του μεγέθους των καταχωρητών στους δρομολογητές το οποίο μπορεί να εφαρμοστεί τοτο σε κανονικές όσο και ακανόνιστες τοπολογίες.

Ένας καταχωρητης μπορεί να περιγραφεί σαν μία μεταβλητή δύο καταστάσεων $c \in 1 \geq \Re^2 \geq 0$, όπου το $c(1)$ αντιπροσωπεύει την κατάσταση αναμονής και το $c(2)$ την κατάσταση λειτουργίας. Τη χρονική στιγμή $t$ ο συνολικός αριθμός των flits στον καταχωρητή είναι $x_t$, ο αριθμός των flits που εισέρχονται είναι $\xi_t$, ο αριθμός των flits που εξέρχονται είναι $\nu_t$ και ο καταχωρητής έχει μέγιστη χωρητικότητα $\nu|x^+$. Η ροή των flits σε έναν καταχωρητή δίνεται απο την Εξίσωση 5.6 για $c_t = c(1)$ και απο την Εξίσωση 5.7 για $c_t = c(2)$.

$$x_{t+1} = \begin{cases} x^+ : & \xi_t + x_t > x^+ \\ \xi_t + x_t : & \xi_t + x_t \leq x^+ \end{cases} \tag{5.6}$$

$$x_{t+1} = \begin{cases} x^+ : & \xi_t + x_t - \nu_t > x^+ \\ \xi_t + x_t - \nu_t : & 0 \leq \xi_t + x_t - \nu_t \leq x^+ \end{cases} \tag{5.7}$$

όπου η συνάρτηση $x^+$ ορίζεται στην Εξίσωση 5.8.

$$x^+ = \begin{cases} x : & x \geq 0 \\ 0 : & x < 0 \end{cases} \tag{5.8}$$

Κάθε $t$ κύκλους ο καταχωρητης εισόδου $B_i$ του δρομολογητή $i$ έχει ποσοστό χρησιμοποίησης που περιγράφεται απο την Εξίσωση 5.9.

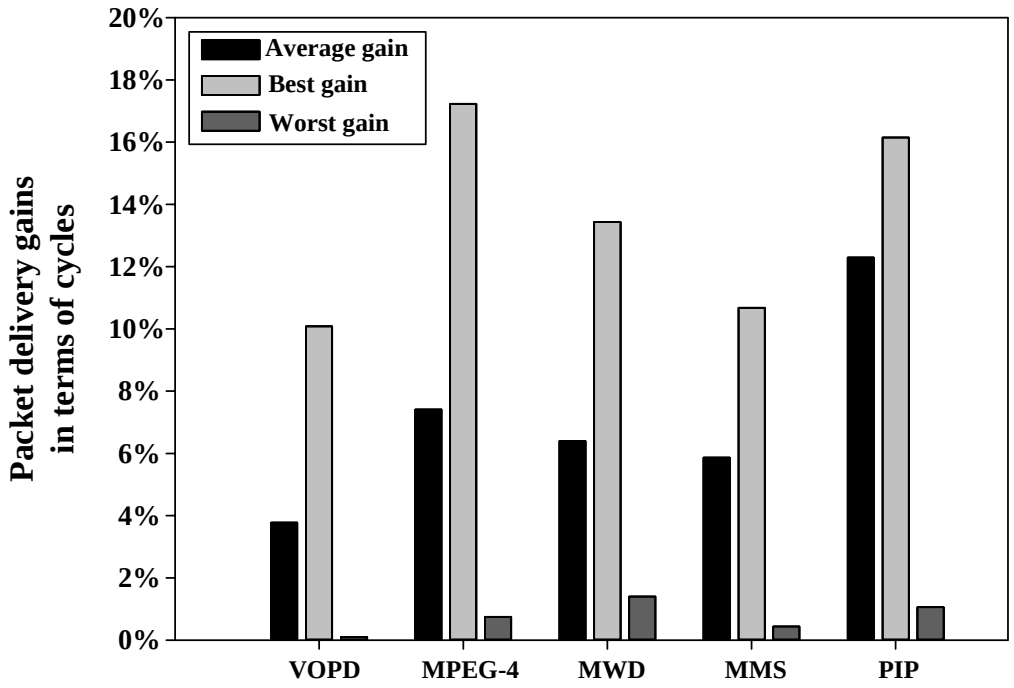
$$u_i^t = \frac{N_i - \mu_i \times E_i}{[B_i]} \tag{5.9}$$

όπου $N_i$ ειναι ο ρυθμός άφιξης πακέτων του δρομολογητή $i$, $\mu_i$ είναι ο ρυθμός εξυπηρέτησης του καταχωρητή, $[B_i]$ είναι ο συνολικός αριθμός των θέσεων σε flits και

$$E_i = \begin{cases} 0, & \text{Αν ο καταχωρηστης είναι άδειος και} \\ 1, & \text{αν ο καταχωρηστης δεν είναι άδειος} \end{cases} \tag{5.10}$$

Ο ελάχιστος αριθμός θέσεων στους καταχωρητές εισόδου, δίνεται απο την Εξίσωση 5.11 μετά απο ένα παράθυρο προσομοίωσης κατά το οποίο για κάθε κύκλο $t$, υπολογίζεται η τιμή $u_i^t$. Οι τιμές που επιστρέφονται χρησιμοποιούνται ως το χαμηλότερο όριο για το μέγεθος των καταχωρητών εισόδου.

$$B_i^t = \begin{cases} 0, & t = 0 \\ u_{k,l}^t, & t > 0, u_i^t > u_i^{t-1} \\ u_{k,l}^{t-1} & t > 0, u_i^t < u_i^{t-1} \end{cases} \tag{5.11}$$




Σχήμα 5.6: Μέσο, καλυτερο και χειρότερο κέρδος σε κύκλους, με τη χρηση προτεραιοτήτων

## 5.3 Αξιολόγηση

Για να επαληθεύουμε την προσέγγισή μας, πραγματοποιήσαμε εκτεταμένες προσομοιώσεις του προτεινόμενου πλαισίου χρησιμοποιώντας πέντε DSP εφαρμογές (i) MPEG-4, (ii) Multi-Window Display (MWD) [21], (iii) Picture-In-Picture (PIP) [21] (iv) MultiMedia System (MMS) [50] και (v) Digitale Radio Mondiale (DRM) [82].

Για τις προαναφερθείσες εφαρμογές αξιολογήθηκε το σύστημα ανάθεσης προτεραιοτήτων, για την περαιτέρω μείωση του χρόνου παράδοσης των πακέτων για ορισμένες κατηγορίες κίνησης, διατηρώντας παράλληλα τη συνολική μέση καθυστέρηση του δικτύου σταθερή. Χωρίσαμε τα πακέτα σε διαφορετικές ροές και σε κάθε ροή ανατέθηκε διαφορετικό είδος προτεραιότητας. Συγκεκριμένα, (i) πακέτα με προορισμό τη μνήμη, (ii) πακέτα που έρχονται από τον κόμβο μνήμης και (iii) πακέτα από κόμβους με υψηλή μέση καθυστέρηση (άνω του 50% της μέσης καθυστέρησης του δικτύου). Ο λόγος για τον οποίο επιλέχθηκε ο κόμβος μνήμης ως κυρίαρχος πάνω στον οποίο εφαρμόζονται τα περισσότερα είδη προτεραιότητας είναι επειδή η επικοινωνία με τη μνήμη κοστίζει λόγω της αργής απόκρισής της. Οι προτεραιότητες εφαρμόστηκαν μόνο σε κανονικές NoC τοπολογίες, καθώς αυτό το πρόβλημα λύνεται στις ακανόνιστες μέσω της διαδικασίας διαμέρισης. Το Σχήμα 5.6 δείχνει το μέσο (7,15% κατά μέσο όρο ), το καλύτερο (13.52% κατά μέσο όρο) και το χειρότερο κέρδος (0,75% κατά μέσο όρο) σε κύκλους, για το χρόνο παράδοσης των πακέτων.

Στις ακανόνιστες τοπολογίες, τα πειραματικά αποτελέσματα επικεντρώθηκαν (i) στη ρυθμαπόδοση (throughput) και (ii) στη μέση καθυστέρηση του δικτύου. Στόχος είναι να πραγματοποιηθούν συγκρίσεις ανάμεσα σε μια ποικιλία απο ακανόνιστες NoC τοπολογίες και στην κανονική τοπολογία πλέγματος. Έτσι:

- Διαμερίσαμε τις εφαρμογές VOPD, MPEG4 και MWD απο 2 μέχρι και 12 διαμερίσεις και την εφαρμογή MMS απο 2 μέχρι και 25 διαμερίσεις σύμφωνα με την προτεινόμενη μεθοδολογία.

- Κάθε διαμέριση, ομαδοποιήθηκε σχηματίζοντας την εκάστοτε τοπολογία.

- Κάθε πακέτο αποτελείται απο 4 flits

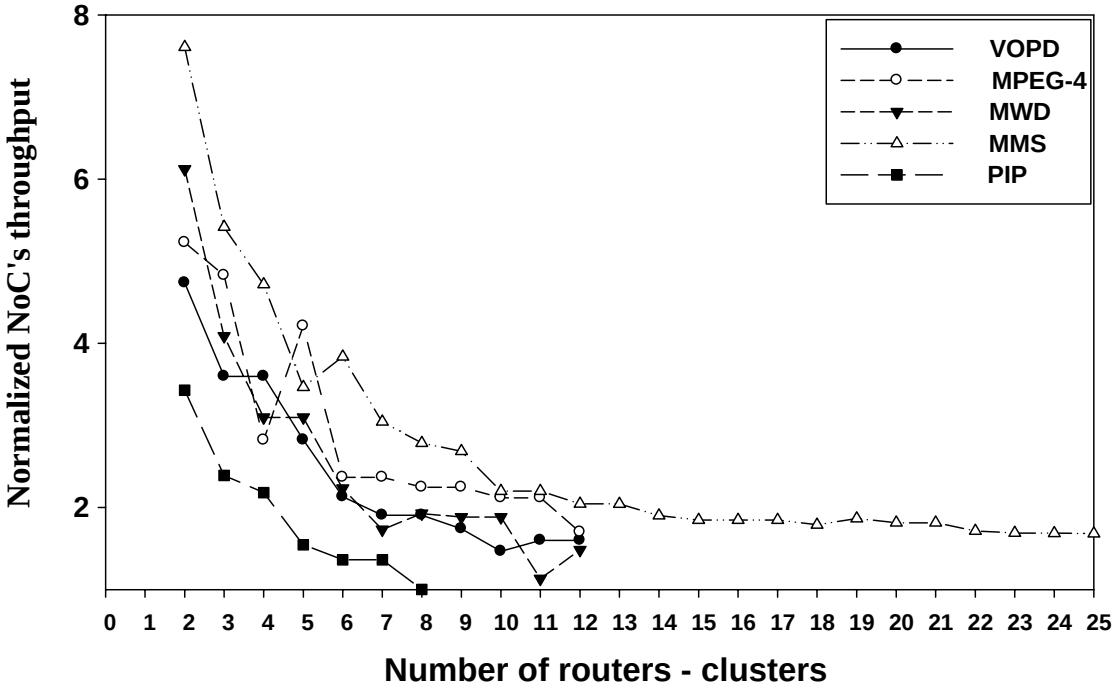
- Οι πίνακες δρομολόγησης δημιουργούνται αυτόματα με τον αλγόριθμο ελάχιστου μονοπατιού Dijkstra.

Ο λόγος για την προσομοίωση τόσο πολλών τοπολογιών είναι ότι δεν μπορεί να προβλεφθεί εκ των προτέρων εάν μία ακανόνιστη τοπολογία με λίγους αλλα μεγάλους δρομολογητές θα είναι πιο αποτελεσματική από μια με περισσότερους αλλά μικρούς. Το σημείο αυτό επιβεβαιώνει το προτεινόμενο πλαίσιο για γρήγορη και αυτόματη εξερεύνηση του χώρου σχεδιασμού. Η βάση σύγκρισης για όλες τις πειραματικές μετρήσεις είναι η κανονική τοπολογία πλέγματος.

#### 5.3.0.1 Ρυθμαπόδοση

Η ρυθμαπόδοση είναι ένα μέτρο της συγκριτικής αποτελεσματικότητας μεγάλων δικτύων και θεωρείται ως ένα από τα πιο σημαντικά χαρακτηριστικά. Ειδικά για τα δίκτυα που εξυπηρετούν εφαρμογές πολυμέσων, υπάρχει μεγάλη ζήτηση για υψηλή ρυθμαπόδοση. Ορίζουμε ως ρυθμαπόδοση σε NoC τοπολογίες το συνολικό αριθμό των flits που παραδίδονται προς όλους τους προορισμούς ανά κύκλο στο εσωτερικό του NoC.

Το Σχήμα 5.7, παρουσιάζει τη ρυθμαπόδοση για τις επιλεγμένες εφαρμογές και όπως είναι εμφανές, η ρυθμαπόδοση σε ακανόνιστες τοπολογιες ειναι καλύτερη σπο την τοπολογία πλέγματος. Πιο συγκεκριμένα, για τις εφαρμογές VOPD και MMS, η ακανόνιστη τοπολογία αυξάνει τη ρυθμαπόδοση του δικτύου, στην καλύτερη περίπτωση, κατά $\times 4.7$ ενώ για τις εφαρμογές MPEG4, PIP και MWD , κατά $\times 8$, $\times 3.4$ και $\times 6.1$ αντίστοιχα. Επιπροσθέτως, το μέσο κέρδος για τις εφαρμογές VOPD, MMS, MWD και MPEG είναι κατα μέσο όρο $\times 2.5$ ενώ για την εφαρμογή PIP είναι $\times 1.9$. Συνοψίζοντας, όλες οι ακανόνιστες τοπολογίες επιτυγχάνουν καλύτερη απόδοση από την κανονική τοπολογία πλέγματος λόγω της ομαδοποίησης η οποία (i) προσφέρει καλύτερα χαρακτηριστικά επικοινωνίας, (ii) αξιοποιεί τα χαρακτηριστικά της τοπολογίας και (iii) διατηρεί χαμηλό τον αριθμό των βημάτων μεταξύ των δρομολογητών.




Σχήμα 5.7: Ρυθμαπόδοση για ακανόνιστες τοπολογίες

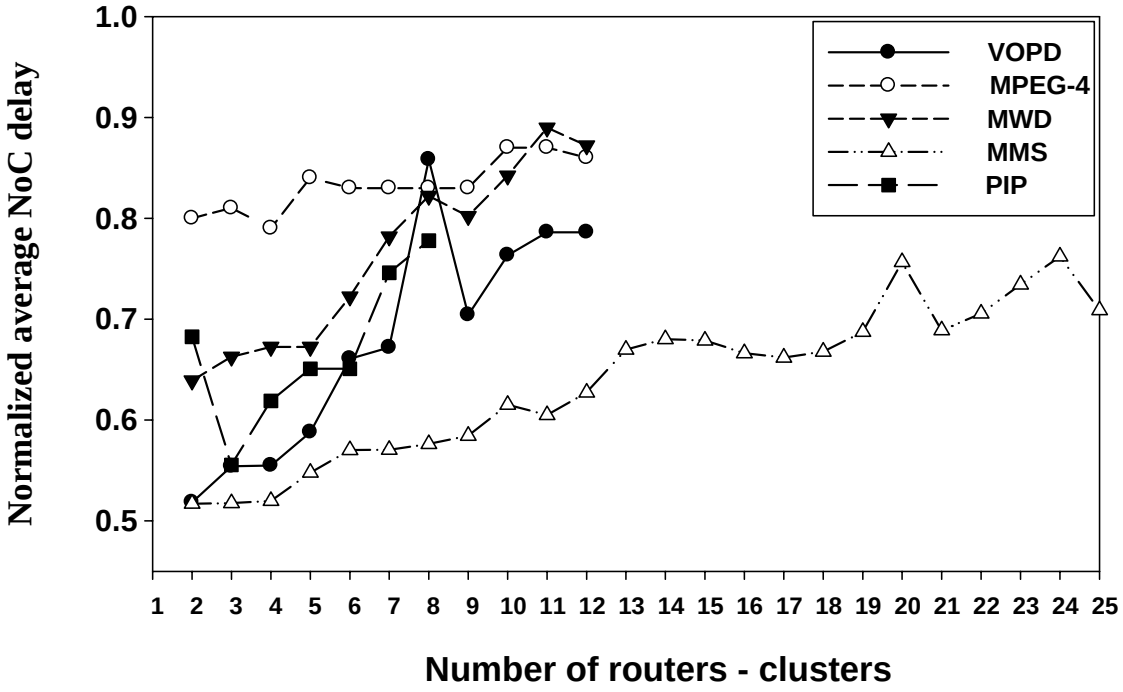
#### 5.3.0.2 Μέση καθυστέρηση δίκτυου

Τα πειραματικά αποτελέσματα για τη μέση καθυστέρηση δίκτυου στις ακανόνιστες τοπολογίες παρουσιάζονται στο Σχήμα 5.8. Όπως φαινεται η μέση καθυστέρηση δίκτυου, μετρούμενη σε κύκλους, είναι καλύτερη για τις ακανόνιστες τοπολογίες σε σύγκριση με την κανονική τοπολογία πλέγματος. Πιο συγκεκριμένα, υπάρχει, στην καλυτερη περίπτωση, μείωση της καθυστέρησης κατά 50% για την εφαρμογή VOPD και 20% για την εφαρμογή MPEG4. Επίσης, για τις εφαρμογές MWD, PIP και MMS η μείωση είναι 40%, 45% και 50% αντίστοιχα. Το μέσο κέρδος για τις εφαρμογές VOPD, MPEG4 και PIP είναι 30%, ενώ για τις εφαρμογές MWD και MMS είναι 17% και 25% αντίστοιχα. Οι ακανόνιστες τοπολογίες μπορούν να εξυπηρετήσουν τα flits πιο γρήγορα λόγω της προσαρμογής και της ομαδοποίησης τους.

### 5.3.1 Κατανάλωση ενέργειας των καταχωρητών

Το Σχήμα 5.9 παρουσιάζει την κανονικοποιημένη κατανάλωση ενέργειας τόσο για κανονικές όσο και για ακανόνιστες τοπολογίες χρησιμοποιώντας τον αλγόριθμο διαχείρισης θέσεων που παρουσιάστηκε στο Κεφάλαιο 5.2.3. Στόχος είναι η ελαχιστοποίηση θέσεων στους καταχωρητές εισόδου, διατηρώντας παράλληλα σταθερή τη μέση καθυστέρηση του δικτύου. Η κατάργηση των περιττών θέσεων οδηγεί σε μεγάλη μείωση της καταναλισκόμενης ενέργειας.

Όπως παρουσιάζεται στο Σχήμα 5.9, ο προτεινόμενος αλγόριθμος διαχείρισης θέσεων

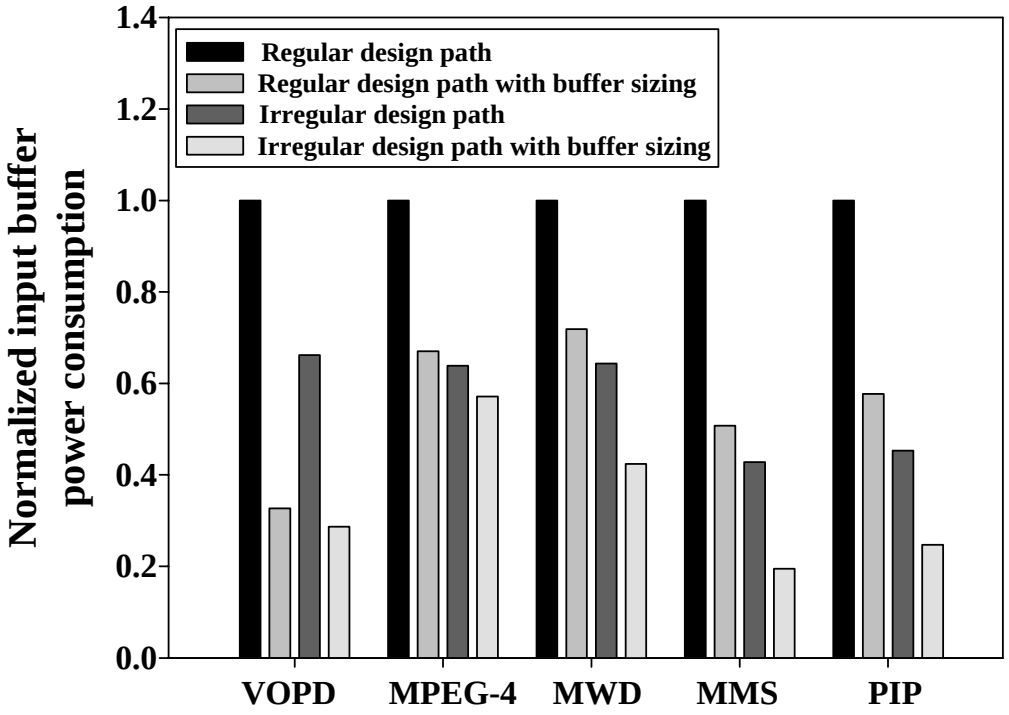



Σχήμα 5.8: Μέση καθυστέρηση δικτύου

επιτυγχάνει, για κανονικές τοπολογίες, 44% κέρδος κατα μέσο όρο ενώ για τις ακανόνιστες το κέρδος φτάνει κατά μέσο όρο στο 45%. Αυτό μπορεί να εξηγηθεί από το γεγονός ότι στις ακανόνιστες τοπολογίες, τα flits ταξιδεύουν τον ελάχιστο αριθμό βημάτων και έτσι δεν απαιτείται επιπλέον χώρος προσωρινής αποθήκευσής τους, σε κάθε ενδιάμεσο δρομολογητή. Τέλος, με την προσθήκη του προτεινόμενου αλγορίθμου διαχείρισης θέσεων, μπορούμε να μειώσουμε περαιτέρω την κατανάλωση ενέργειας 20% κατά μέσο όρο.

*Ο στόχος είναι να παρασχεθεί στο σχεδιαστή μια ποικιλία από επιλογές, ώστε να δοκιμάσει και να αξιολογήσει διαφορετικές διαμορφώσεις που θα καλύψουν καλύτερα τις απαιτήσεις των εφαρμογών.*

Οι αρχιτεκτονικές τύπου NoC έχουν αναγνωριστεί ως το νέο πρότυπο για τη διασύνδεση και την οργάνωση μεγάλου αριθμού πυρήνων. Σε αυτό το κεφάλαιο παρουσιάστηκε ένα πλαίσιο υψηλού επιπέδου για (i) την υποστήριξη τόσο κανονικών όσο και ακανόνιστων NoC αρχιτεκτονικών κατά τη φάση σχεδιασμού και (ii) την παραμετροποίηση των εφαρμογών κατά το χρόνο εκτέλεσης. Το προτεινόμενο πλαίσιο δίνει στον σχεδιαστή μια ποικιλία από δυνατότητες προκειμένου αυτός να δοκιμάσει και να αξιολογήσει έναν μεγάλο αριθμό από διαφορετικές διαμορφώσεις και επιλογές διαχείρισης των πόρων.




Σχήμα 5.9: Κανονικοποιημένη κατανάλωση ενέργειας τόσο για κανονικές όσο και για ακανόνιστες τοπολογίες χρησιμοποιώντας τον προτεινόμενο αλγόριθμο διαχείρισης θέσεων

# Κεφάλαιο 6

# Συμπεράσματα

Στο παρόν κεφάλαιο παρουσιάζονται τα συμπεράσματα που προέκυψαν από την διδακτορική διατριβή καθώς επίσης συνοψίζονται οι καινοτομίες και οι μελλοντικές επεκτάσεις. Οι καινοτομίες που επιτεύχθηκαν στην διδακτορική διατριβή καθιστούν δυνατή την ανάπτυξη νέων μεθοδολογιών σχεδιασμού που μπορούν να βελτιώσουν περαιτέρω τη διαχείριση μνήμης κατά το χρόνο εκτέλεσης σε πολυπύρηνα ενσωματωμένα συστήματα.

## 6.1 Επισκόπηση διδακτορικής διατριβής

Στην παρούσα διδακτορική διατριβή, παρουσιάστηκαν μεθολογίες επιτάχυνσης και παραμετροποίησης της διαχείρισης μνήμης για πολυπύρηνες ενσωμματωμένες αρχιτεκτονικές. Ο κύριος στόχος ήταν να αντιμετωπίστούν αποτελεσματικά τα προβλήματα:

- Της παροχής εξατομικευμένων διαχειριστών δυναμικής μνήμης σε πολυπύρηνες ενσωμματωμένες πλατφόρμες [11].
- Της παροχής κατανεμημένων Δυναμικών Διαχειριστών Μνήμης (ΔΔΜ).
- Της μείωσης της καταναλισκόμενης ισχύος εσνωματώνοντας DVFS μηχανισμούς μέσα στους ΔΔΜ [13].

Για την αντιμετώπιση αυτών των κρίσιμων θεμάτων που επηρεάζουν τα σύγχρονα ενσωματωμένα συστημάτα, αναπτύχθηκαν νέες μεθοδολογίες και πλαίσια διαχείρισης:

- Αναπτύχθηκε ένα πλαίσιο για την παροχή προσαρμοσμένων ΔΔΜ σε μικροκώδικα χρησιμοποιώντας την παρουσία επιταχυντή υλικού. Υιοθετήθηκε η προσέγγιση σε μικροκώδικα καθώς προσφέρει αποδοτικότερη αξιοποίηση του υλικού διατηρώνας παράλληλα την ευελιξία των εφαρμογών λογισμικού [11].
- Παρουσιάστηκε ένας ευέλικτος κλιμακούμενος και κατανεμημένος ΔΔΜ σε επίπεδο μικροκώδικα που ονομάζεται MAD-DMM. Ο MAD-DMM προσφέρει κατανεμημένη λειτουργικότητα, ενώ διατηρεί ίδιες τις προγραμματιστικές διεπαφές (malloc()/free()). Σε αντίθεση με άλλους δυναμικούς διαχειριστές μνήμης

υψηλού επιπέδου, οι πληροφορίες σχετικά με την κατάσταση του σωρού δεν αποθηκεύονται σε υψηλό επίπεδο, αλλά σε μικροκώδικα ως μέρος ενός τοπικού πίνακα.

- Παρουσιάστηκε μια νέα στρατηγική σχεδιασμού για την υλοποίηση μιας αποτελεσματικής μεθοδολογίας για τη συνδυασμένη χρήση τεχνικών DVFS μαζί με ΔΔΜ με στόχο την χαμηλή κατανάλωση ισχύος [13].

Όλες οι προτεινόμενες μεθοδολογίες αξιολογήθηκαν ποιοτικά και ποσοτικά σε μία πολυπύρηνη αρχιτεκτονική η οποία αποτελείται από επεξεργαστικούς κόμβους (Proseccing Modules, PM) οι οποίοι διασυνδέονται μέσω ενός δικτυου σε ψηφίδα. Κάθε επεξεργαστικός κόμβος αποτελείται από έναν LEON3 επεξεργαστή με τη δικιά του μνήμη εντολων (I-Cache) και δεδομένων (D-Cache), ένα διπύρηνο μικροπορογραμματίζομενο ελεγκτή (Dual microcoded Controller, DMC) και μνήμη η οποία μπορεί να μοιραστεί μεταξύ των κόμβων (shared memory) [33]. Τα πειραματικά αποτελέσματα σε διάφορες εφαρμογές έδειξαν σημαντικά οφέλη στις επιδόσεις και στην κατανάλωση ισχύος αναφορικά με τις προτεινόμενες προσαρμογές των ΔΔΜ.

Όπως προαναφέρθηκε, η διαχείριση πόρων είναι μια βασική λειτουργία για την επιτυχημένη χρήση των σύγχρονων υπολογιστικών πλατφορμών και το μοντέλο διαχείρισης πόρων κατά τη φάση εκτέλεσης έχει γίνει εμφανές πρόσφατα, επειδή μπορεί να ασχοληθεί με τη δυναμική συμπεριφορά των εφαρμογών και των πλατφορμών. Ωστώσο, τα σύγχρονα πολύπλοκα πολυπύρηνα συστήματα αντιμετωπίζουν διάφορα προβλήματα στον τομέα της διαχείρισης πόρων στη φάση εκτέλεσης:

- Οι υφιστάμενες προσεγγίσεις για τη διαχείριση πόρων σε πολυπύρηνες πλατφόρμες, ακόμα και αν διαθέτουν κάποιες αυτόνομες ιδιότητες, είναι τυπικά κεντρικοποιημένες δημιουργώντας έτσι ένα κεντρικό πιθανό σημείο αποτυχίας.

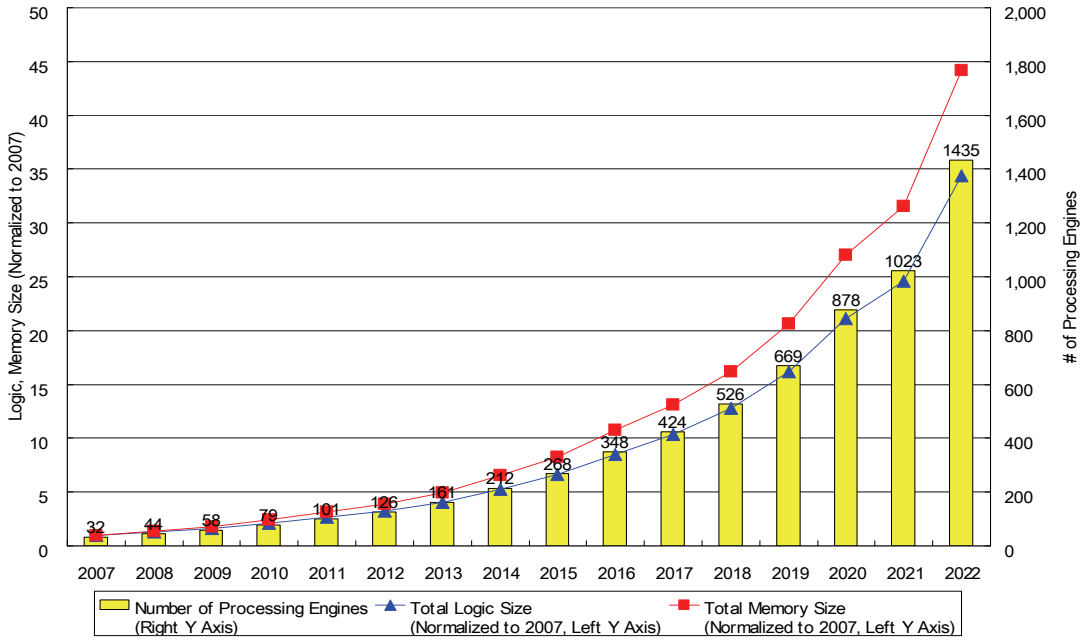
- Ο κεντρικός πυρήνας που αναλυει τα δεδομένα, δεν είναι επεκτάσιμος και ενεργεί ως σημείο συμφόρησης τόσο στην επεξεργασία όσο και στην επικοινωνία.

- Οι περισσότερες προσεγγίσεις στερούνται μιας διαδικασίας αυτο-προσαρμογής δημιουργώντας έτσι προβλήματα αποκλεισμού λόγω του υψηλού αριθμού εισερχόμενων εφαρμογών που δεν είναι σε θέση να εξυπηρετήσουν.

Προκειμένου να αντιμετωπιστούν αυτά τα προβλήματα, αναπτύχθηκαν μεθοδολογίες για τη διαχείριση πόρων σε πολυπύρηνα συστήματα τα οποία στοχεύουν στην επίλυση τους με τους ακόλουθους τρόπους:

- Αναπτύχθηκε ένα κατανεμημένο (*Διαίρει και Βασίλευε*) πλαίσιο για την παροχή κατανεμημένων υπηρεσιών χαρτογράφησης στο χρόνο εκτέλεσης τόσο για ομογενείς όσο και για ετερογενείς πολυπύρηνες πλατφόρμες. Το προτεινόμενο πλαίσιο επιτυγχάνει διαφορετικά επίπεδα χρησιμοποίησης των πόρων της πλατφόρμας ανάλογα με τις ανάγκες της εφαρμογής [10].

- Αναπτύχθηκε μια μεθοδολογία για την διαχείρηση εύπλαστων εφαρμογών με βάση το κόστος ανα πυρήνα [14].




Σχήμα 6.1: Τάσεις της αυξημένης πολυπλοκότητας με στόχο την αγορά των κινητών συσκευών [77]

- Αναπτύχθηκε ένα πλαίσιο και μια μεθοδολογία υψηλού επιπέδου για τη διαχείριση πόρων σε αρχιτεκτονικές τύπου Δίκτυο-σε-Ψηφίδα, τόσο για κανονικές (regular) όσο και για ακανόνιστες (irregular) τοπολογίες [12].

Τα προτεινόμενα πλαίσια βασίζονται στην ιδέα της χρήσης πολλαπλών πυρήνων σε διαφορετικούς ρόλους και σε κάθε περίπτωση ένα σύστημα διασύνδεσης εξασφαλίζει την κατανεμημένη λειτουργικότητα. Τα προτεινόμενα πλαίσια αξιολογήθηκαν στην πειραματική πολυπύρηνη πλατφόρμα που παρουσιάζεται στο [33] και στην πλαφόρμα Intel Single Chip Cloud (SCC) [49].

## 6.2 Προοπτικές και μελλοντικές επεκτάσεις

Οι μεθοδολογίες και τα πλαίσια που περιγράφονται στην παρούσα διδακτορική διατριβή μπορούν να αποτελέσουν τη βάση για την αντιμετώπιση νέων προβλημάτων που θα προκύψουν στα ενσωματωμένα συστήματα τόσο βραχυπρόθεσμα όσο και μεσοπρόθεσμα. Αυτά τα νέα προβλήματα προκύπτουν λόγω της αυξημένης πολυπλοκότητας των πλατφορμών υλικού (Σχήμα 6.1) και των εφαρμογών λογισμικού.

Σύμφωνα με το HiPEAC [4], οι καινούριες προκλήσεις στον χώρο των ενσωματωμένων συστήματων είναι η ενσωμάτωση του κατάλληλου λογισμικού σε πολυπύρηνες πλατφόρμες στοχεύοντας στην διαχείριση κατά τη φάση εκτέλεσης και στην επιτά-

χυνση της επεξεργασίας του τεράστιου όγκου των δεδομένων που παράγονται απο τις σύγχρονες εφαρμογές. Μια πρώτη προσέγγιση, είναι η επέκταση του πλαισίου για τη δημιουργία εξατομικευμένων ΔΔΜ σε μικροκώδικα, έτσι ώστε να υποστηρίζει αλλαγές των πολιτικών κατά το χρόνο εκτέλεσης. Μέχρι τώρα, ο ΔΔΜ λειτουργεί με προκαθορισμένες πολιτικές για το χειρισμό των δυναμικών δεδομένων των εφαρμογών. Οι πολιτικές αυτές έχουν να κάνουν με τη χρήση σταθερών λιστών, τεχνικές συγχώνευσης και διαχωρισμού μπλόκ μνημών κλπ. Ένας ΔΔΜ που είναι σε θέση να ρυθμίσει αυτές τις παραμέτρους κατά το χρόνο εκτέλεσης και σύμφωνα με τις ανάγκες της εφαρμογής, θα είναι πολύ πιο αποτελεσματικός. Όπως προαναφέρθηκε, μεγάλος αριθμός από εφαρμογές έρχεται κάθε στιγμή στα πολυπύρηνα συστήματα. Κάθε μία από αυτές τις εφαρμογές είναι διαφορετική από την άποψη της χρήσης της μνήμης για τα δυναμικά δεδομένα τους. Έτσι, ένας κατανεμημένος σε επίπεδο μικροκώδικα ΔΔΜ που έχει επίσης τη δυνατότητα να προσαρμόζεται κατά το χρόνο εκτέλεσης, θα είναι πολύ πιο αποτελεσματικός από την πλευρά του τελικού αποτυπώματος μνήμης και απόδοσης. Για να επιτευχθεί αυτό, ο ΔΔΜ πρέπει να σχεδιαστει με βάση σενάρια λειτουργίας. Αυτό σημαίνει, ότι ο ΔΔΜ θα έχει κάποια προκαθορισμένα σημεία εργασίας και μπορεί να αλλάξει μεταξύ αυτών κατά το χρόνο εκτέλεσης. Τα κριτήρια για την αλλαγή των σημείων εργασίας, μπορεί να είναι η απόδοση των εφαρμογών, η χρήση μνήμης, η κατανάλωση ενέργειας κλπ.

Μια άλλη επέκταση, στον τομέα της διαχείρισης πόρων στη φάση εκτέλεσης, είναι η ανάπτυξη ενός ολοκληρωμένου μηχανισμού μεταφοράς-μετακίνησης εργασιών. Έτσι, αντί το μεθολογικό πλαίσιο να λαμβάνει την απόφαση μόνο για την απεικόνιση των εργασιών στους πυρηνες, θα είναι πολύ χρήσιμο να αλλάζει, κατά το χρόνο εκτέλεσης, το είδος του επεξεργαστή στον οποίο η εργασία εκτελείται. Για παράδειγμα, μέχρι τώρα, όταν μια νέα εφαρμογή εισέρχεται στο σύστημα, περιμένει για έναν διαθέσιμο πυρήνα να την εξυπηρετήσει. Ωστόσο, αυτός ο πυρήνας μπορεί να μην είναι ο βέλτιστος. Έτσι, κατά τη διαδικασία της αυτο-βελτιστοποίησης, θα βοηθούσε πραγματικά την απόδοση των εφαρμογών, αν ένα έργο "μετανάστευε" σε άλλο τύπο πυρήνα. Αυτό σημαίνει, ότι όλα τα αποθηκευμένα δεδομένα και πληροφορίες πρέπει να ακολουθήσουν την εργασία στο νέο πυρήνα, συμπεριλαμβανομένων των δεδομένων της μνήμης cache. Σε αυτή την περίπτωση, ένας μηχανισμός μεταφοράς-μετακίνησης εργασιών θα φροντίζει ώστε να μην χαθεί χρόνος και πολύτιμοι πόροι.

Σύμφωνα με το HiPEAC [4], τα σημερινά ενσωματωμένα συστήματα είναι σχεδόν καθολικά συνδεδεμένα, και σε σύγκριση με το παρελθόν, πολλές επιθέσεις hacking έχουν αποδείξει ότι η ασφάλεια πρέπει να βελτιωθεί. Η ανάγκη αυτή είναι ιδιαίτερα εμφανής στις σύγχρονες συσκευές καθώς χειρίζονται συνεχώς ιδιωτικά και τα προσωπικά δεδομένα των χρηστών. Οι επιθέσεις που στοχεύουν το σωρό (Heap) αποτελούν συνεχή απειλή. Η κακή διαχείριση μνήμης από τον προγραμματιστή μπορεί να οδηγήσει σε διάφορα λάθη που θα μπορούσαν να επηρεάσουν άλλες διαδικασίες και να επιτρέψουν τέτοιου είδους επιθέσεις. Έτσι, αντί η ασφάλεια να στηρίζεται στις δεξιότητες διαχείρισης της μνήμης απο τον προγραμματιστή, απαιτείται ένα πλαισιο προστασίας. Οι περισσότεροι ΔΔΜ αγνοούν την ασφάλεια. Αντ 'αυτού, επικεντρώνονται στη μεγιστοποίηση της απόδοσης και τον περιορισμό του κατακερματισμού της μνήμης. Ενώ αυτά είναι πολύ σημαντικά ζητήματα για την εκχώρησης μνήμης, στη σημερινή εποχή όπου οι επιθέσεις αυξάνονται, η ασφάλεια δεν μπορεί να αγνοείται.

Έτσι, μια επιπλεόν επέκταση στα προτεινόμενα πλαίσια είναι η ενσωμάτωση μηχανισμών ασφαλείας σε κατανεμημένες μνήμες για πολυπύρηνα συστήματα. Επίσης, η χρήση ετερογένειας και επιταχυντών υλικού σίγουρα θα βελτιώσουν την απόδοση των ΔΔΜ δεδομένου ότι οι περισσότερες τεχνικές ασφάλειας, απαιτούν πολλούς επιπλεόν επεξεργαστικούς κύκλους.

Τέλος, τα μεγάλα κέντρα δεδομένων (data centers) επεξεργάζονται τις τεράστιες ποσότητες δεδομένων που παράγονται από τα σύγχρονα ενσωματωμένα και κινητά υπολογιστικά συστημάτα, ηλεκτρονικές συναλλαγές, και επιστημονικές προσομοιώσεις. Δεδομένου ότι η είσοδος και ο χρόνος της επεξεργασίας για τα συστήματα αυτά είναι άγνωστα, οι ΔΔΜ έχουν ουσιαστικό ρόλο στην συνολική απόδοση. Έτσι, μια επέκταση στα προτεινόμενα πλαίσια, είναι η χρήση επιταχυντών υλικού και προσαρμογή των ΔΔΜ ανά πεδία εφαρμογών στα μεγάλα υπολογιστικά συστήματα. Ακόμα και αν αυτά τα συστήματα δεν περιορίζονται από το μέγεθος της διαθέσιμης μνήμης, το μέγεθος των συναλλαγών είναι τόσο μεγάλο που αποδεικνύεται ότι είναι ένας από τους σημαντικότερους παράγοντες απόδοσης.

# Δημοσιεύσεις

## Κεφάλαια σε βιβλία

1. I. Anagnostopoulos, S. Xydis, A. Bartzas, Z. Lu, D. Soudris and A. Jantsch, "*Chapter 8 Middleware memory management in NoC*", *in "Designing 2D and 3D Network-on-Chip Architectures,"* to appear, Springer

2. B. Candaele, S. Aguirre, M. Sarlotte, I. Anagnostopoulos, S. Xydis, A. Bartzas, D. Bekiaris, D. Soudris, Z. Lu, X. Chen, J.-M. Chabloz, A. Hemani, A. Jantsch, G. Vanmeerbeeck, J. Kreku, K. Tiensyrja, F. Ieromnimon, D. Kritharidis, A. Wiefrink, B. Vanthournout, P. Martin, "*Chapter 11: Mapping Optimisation for Scalable multi-core ARchiTecture: The MOSART approach,"* in "VLSI 2010 Annual Symposium," Springer

3. S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, "*Chapter 3: Application-Specific Multi-Threaded Dynamic Memory Management,"* in "Scalable Multi-core Architectures: Design Methodologies and Tools," 2011, Springer

## Περιοδικά

1. I. Anagnostopoulos, A. Bartzas, I. Filippopoulos, D. Soudris, "*High-level customization framework for application-specific NoC architectures,"* in Springer Design Automation for Embedded Systems, vol.16, no.4, pp.339-361, 2013, doi: 10.1007/s10617-013-9114-5

2. I. Anagnostopoulos, J.M. Chabloz, I. Koutras, A. Bartzas, A. Hemani, D. Soudris, "*Power-aware Dynamic Memory Management on May-core Platfroms utilizing DVFS,"* ACM Transactions on Embedded Computing Systems, vol.13, no.1, pp.40:1–40:25, November 2013

3. I. Anagnostopoulos, S. Xydis, A. Bartzas, Z. Lu, D. Soudris, A. Jantsch, "*Custom Microcoded Dynamic Memory Management for Distributed On-Chip Memory Organizations,"* in IEEE Embedded System Letters, vol.3, no.2, pp.66,69, June 2011

## Συνέδρια

1. I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, D. Soudris, "*Distributed run-time resource management for malleable applications on many-core platforms*," in Proceedings of DAC conference 2013

2. I. Anagnostopoulos, A. Bartzas, G. Kathareios, D. Soudris, "*A Divide and Conquer based Distributed Run-time Mapping Methodology for Many-Core platforms*," in Proceedings of DATE conference 2012

3. K. Siozios, D. Diamantopoulos, I. Kostavelis, E. Boukas, L. Nalpantidis, D. Soudris, A. Gasteratos, M. Aviles, I. Anagnostopoulos, "*SPARTAN project: Efficient implementation of computer vision algorithms onto reconfigurable platform targeting to space applications*," in Proceedings of the 6th InternationalWorkshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011

4. C. Silvano, W. Fornaciari, S. Crespi Reghizzi, G. Agosta, G. Palermo, V. Zaccaria, P. Bellasi, F. Castro, S. Corbetta, E. Speziale, D. Melpignano, JM. Zins, H. Hubert, B. Stabernack, J. Brandenburg, M. Palkovic, P. Raghavan, C. Ykman-Couvreur, I. Anagnostopoulos, A. Bartzas, D. Soudris, T. Kempf, G. Ascheid, J. Ansari, P. Mahonen, B. Vanthournout, "*Parallel programming and run-time resource management framework for many-core platforms: The 2PARMA approach*," in Proceedings of the 6th International Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011

5. A. Bartzas, P. Bellasi, I. Anagnostopoulos, C. Silvano, W. Fornaciari, D. Soudris, D. Melpignano, C. Ykman-Couvreur, "*Runtime Resource Management Techniques for Many-core Architectures: The 2PARMA Approach*," in Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), 2011

6. K. Siozios, I. Anagnostopoulos, D. Soudris, "*Multiple Vdd on 3D NoC Architectures*," in Proceedings of 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS) 2010

7. S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, K. Pekmestzi, "*Custom Multi-Threaded Dynamic Memory Management for Multiprocessor System-on-Chip Platforms*," in Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS) 2010

8. B. Candaele, S. Aguirre, M. Sarlotte, I. Anagnostopoulos, S. Xydis, A. Bartzas, D. Bekiaris, D. Soudris, Z. Lu, X. Chen, J.-M. Chabloz, A. Hemani, A. Jantsch, G. Vanmeerbeeck, J. Kreku, K. Tiensyrja, F. Ieromnimon, D. Kritharidis, A. Wiefrink, B. Vanthournout, P. Martin, "*Mapping Optimisation for Scalable multi-core ARchiTecture: The MOSART approach*," in Proceedings of IEEE Computer Society Annual Symposium on VLSI (IS-VLSI) 2010

9. I. Filippopoulos, I. Anagnostopoulos, A. Bartzas, D. Soudris, G. Economakos,

"*Systematic Exploration of Energy-Efficient Application-Specific Network-on-Chip Architectures*," in Proceedings of IEEE Computer Society Annual Symposium on VLSI (IS-VLSI) 2010

10. K. Siozios, I. Anagnostopoulos, D. Soudris, "*A High-Level Mapping Algorithm Targeting 3D NoC Architectures with Multiple Vdd*," in Proceedings of IEEE Computer Society Annual Symposium on VLSI (IS-VLSI) 2010

11. I. Anagnostopoulos, A. Bartzas, D. Soudris, "*Application-Specific Temperature Reduction Systematic Methodology for 2D and 3D Networks-on-Chip*," in Proceedings of International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2009

12. I. Anagnostopoulos, A. Bartzas, I Vourkas, D. Soudris, "*Node Resource Management for DSP Applications on 3D Network-on-Chip architectures*," in Proceedings of 16th International Conference on Digital Signal Processing (DSP), 2009

# Βιβλιογραφία


[1] Chaco: Software for Partitioning Graphs. URL http://www.cs.sandia.gov/~bahendr/chaco.html.

[2] Noxim: network-on-chip simulator. URL http://sourceforge.net/projects/noxim/.

[3] ECN Magazine. URL http://www.ecnmag.com/articles/2011/01/urgent-need-digital-media-subscriber-modeling.

[4] HiPEAC Roadmap - 2013, . URL http://www.hipeac.net/system/files/hipeac_roadmap1_0.pdf.

[5] HSA: Heterogeneous System Architecture Foundation, . URL http://hsafoundation.com/.

[6] HSA: Heterogeneous System Architecture Foundation, . URL http://chipdesignmag.com/sld/shuler/tag/hsa-foundation/.

[7] A. Agarwal, , et al. The MIT Alewife machine: architecture and performance. In *Proc. of ISCA 1995*, pages 2–13, 1995.

[8] S. Agarwala et al. A 65nm c64x+ multi-core dsp platform for communications infrastructure. In *Proc. of ISSCC*, pages 262 –601, feb. 2007. doi: 10.1109/ISSCC.2007.373394.

[9] Mohammad Abdullah Al Faruque et al. Adam: run-time agent-based distributed application mapping for on-chip communication. In *Proc. of DAC*, pages 760–765. ACM, 2008. ISBN 978-1-60558-115-6.

[10] I. Anagnostopoulos, A. Bartzas, G. Kathareios, and D. Soudris. A divide and conquer based distributed run-time mapping methodology for many-core platforms. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 111–116, March 2012. doi: 10.1109/DATE.2012.6176442.

[11] I. Anagnostopoulos et al. Custom microcoded dynamic memory management for distributed on-chip memory organizations. *Embedded Systems Letters, IEEE*, 2011.

[12] Iraklis Anagnostopoulos, Alexandros Bartzas, Iason Filippopoulos, and Dimitrios Soudris. High-level customization framework for application-specific noc architectures. *Design Automation for Embedded Systems*, 16(4):339–361, 2012. ISSN 0929-5585. doi: 10.1007/s10617-013-9114-5. URL http://dx.doi.org/10.1007/s10617-013-9114-5.

[13] Iraklis Anagnostopoulos, Jean-Michel Chabloz, Ioannis Koutras, Alexandros Bartzas, Ahmed Hemani, and Dimitrios Soudris. Power-aware dynamic memory management on many-core platforms utilizing dvfs. *ACM Trans. Embed. Comput. Syst.*, 13(1s):40:1–40:25, December 2013. ISSN 1539-9087. doi: 10.1145/2536747.2536762. URL http://doi.acm.org/10.1145/2536747.2536762.

[14] Iraklis Anagnostopoulos, Vasileios Tsoutsouras, Alexandros Bartzas, and Dimitrios Soudris. Distributed run-time resource management for malleable applications on many-core platforms. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 168:1–168:6, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2071-9. doi: 10.1145/2463209.2488942. URL http://doi.acm.org/10.1145/2463209.2488942.

[15] David Atienza et al. Systematic dynamic memory management design methodology for reduced memory footprint. *ACM TODAES*, 11(2):465–489, 2006. ISSN 1084-4309. doi: http://doi.acm.org/10.1145/1142155.1142165.

[16] Ke Bai and Aviral Shrivastava. Heap data management for limited local memory (llm) multi-core processors. pages 317–326, 2010.

[17] Alexandros Bartzas et al. Software metadata: Systematic characterization of the memory behaviour of dynamic applications. *Journal of Systems and Software*, In Press, Corrected Proof:–, 2010. ISSN 0164-1212. doi: DOI:10.1016/j.jss.2010.01.001.

[18] Adam Beguelin et al. Application level fault tolerance in heterogeneous networks of workstations. *J. Parallel Distrib. Comput.*, 43(2):147–155, June 1997. ISSN 0743-7315. doi: 10.1006/jpdc.1997.1338. URL http://dx.doi.org/10.1006/jpdc.1997.1338.

[19] L. Benini and G. de Micheli. Networks on chips: a new SoC paradigm. *Computer*, 35(1):70–78, 2002.

[20] Emery D. Berger et al. Hoard: a scalable memory allocator for multithreaded applications. In *Proc. of ASPLOS, Cambridge, MA, USA*, pages 117–128. ACM, 2000. doi: http://doi.acm.org/10.1145/356989.357000.

[21] D. Bertozzi et al. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE TPDS*, 16(2):113–129, Feb 2005.

[22] M.K. Bhatti et al. In *Proc of DASIP*, pages 136 –143, 2010. doi: 10.1109/DASIP.2010.5706257.

[23] R. Bitirgen et al. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proc. of MICRO-41*, pages 318 –329, Nov. 2008.

[24] S. Borkar. Thousand core chips: A technology perspective. In *Proc. of DAC*, pages 746–749, 2007.

[25] Hajo Broersma et al. The computational complexity of the minimum weight processor assignment problem. In *Proc. of WG*, pages 189–200, 2004.

[26] Junwei Cao et al. Arms: An agent-based resource management system for grid computing. *Sci. Program.*, 10:135–148, April 2002. ISSN 1058-9244.

[27] Ewerson Carvalho et al. Heuristics for dynamic task mapping in noc-based heterogeneous mpsocs. In *Proc. of IWRSP*, pages 34–40. IEEE Computer Society, 2007. ISBN 0-7695-2834-1.

[28] T. L. Casavant and J. G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, 14:141–154, February 1988. ISSN 0098-5589.

[29] F. Catthoor et al. *Data access and storage management for embedded programmable processors*. Kluwer Academic Publishers, 2002.

[30] Jean-Michel Chabloz and Ahmed Hemani. Lowering the latency of interfaces for rationally-related frequencies. In *ICCD*, pages 23–30, 2010.

[31] J. Morris Chang and Edward F. Gehringer. A high-performance memory allocator for object-oriented systems. *IEEE Trans. Comput.*, 45(3):357–366, 1996. ISSN 0018-9340. doi: http://dx.doi.org/10.1109/12.485574.

[32] M. Chaudhuri and M. Heinrich. SMTp: an architecture for next-generation scalable multi-threading. In *Proc. of ISCA*, pages 124–135, 2004.

[33] Xiaowen Chen et al. Supporting distributed shared memory on multi-core network-on-chips using a dual microcoded controller. In *Proc. of DATE, Dresden, Germany*, pages 39–44, 2010.

[34] Chen-Ling Chou and Radu Marculescu. Incremental run-time application mapping for homogeneous nocs with multiple voltage levels. In *Proc. of CODES+ISSS*, pages 161–166. ACM, 2007.

[35] D. Feitelson. Parallel Workloads Archive, `http://www.cs.huji.ac.il/labs/parallel/workload`. URL `http://www.cs.huji.ac.il/labs/parallel/workload`.

[36] G. De Micheli. An Outlook on Design Technologies for Future Integrated Systems. *IEEE TCAD*, 28(6):777, 2009.

[37] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL `http://doi.acm.org/10.1145/1327452.1327492`.

[38] Travis Desell et al. Malleable applications for scalable high performance computing. *Cluster Computing*, 10(3):323–337, 2007.

[39] Robert P. Dick et al. Tgff: task graphs for free. In *CODES'98*, pages 97–101, 1998.

[40] Allen B. Downey. A model for speedup of parallel programs. Technical report, 1997.

[41] Dror G. Feitelson and Larry Rudolph. Toward convergence in job schedulers for parallel supercomputers. In *Proc. of JSSPP*, pages 1–26. Springer-Verlag, 1996.

[42] Kees Goossens et al. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test*, 22(5):414–421, 2005. ISSN 0740-7475.

[43] Andreas Hansson et al. A unified approach to constrained mapping and routing on network-on-chip architectures. In *Proc. of CODES+ISSS*, pages 75–80. ACM, 2005. ISBN 1-59593-161-9.

[44] T. Henderson, D. Kotz, and I. Abyzov. The Changing Usage of a Mature Campus-wide Wireless Network. In *Proc. of MobiCom*, 2004.

[45] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing*, volume 95, page 285, 1995.

[46] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2007.

[47] K. Hirata and J. Goodacre. ARM MPCore; The streamlined and scalable ARM11 processor core. In *Proc. of ASP-DAC*, pages 747–748. IEEE Computer Society, 2007.

[48] M. Horowitz et al. Low-power digital design. In *Proc. of SLPD*, pages 8 –11, oct 1994. doi: 10.1109/LPE.1994.573184.

[49] J. Howard et al. A 48-core ia-32 message-passing processor with dvfs in 45nm cmos. In *Proc. of ISSCC*, pages 108 –109, feb. 2010. doi: 10.1109/ISSCC.2010.5434077.

[50] Jingcao Hu and R. Marculescu. Energy- and performance-aware mapping for regular noc architectures. *IEEE TCAD*, 24(4):551–562, 2005. ISSN 0278-0070.

[51] Information Sciences Institute. RFC 793: Transmission Control Protocol. *http://tools.ietf.org/html/rfc793*, 1981.

[52] A. Iyengar. Parallel dynamic storage allocation algorithms. In *Parallel and Distributed Processing, 1993. Proceedings of the Fifth IEEE Symposium on*, pages 82–91, 1993.

[53] Axel Jantsch and Hannu Tenhunen, editors. *Networks on chip*. Kluwer Academic Publishers, 2003. ISBN 1-4020-7392-5.

[54] Sebastian Kobbe et al. DistRM: distributed resource management for on-chip many-core systems. In *Proc. of CODES+ISSS*, pages 119–128. ACM, 2011. ISBN 978-1-4503-0715-4. doi: 10.1145/2039370.2039392. URL http://doi.acm.org/10.1145/2039370.2039392.

[55] Samuel Kounev et al. Towards self-aware performance and resource management in modern service-oriented systems. In *Proc. of SCC*, pages 621–624. IEEE CS, 2010. ISBN 978-0-7695-4126-6. doi: 10.1109/SCC.2010.94. URL http://dx.doi.org/10.1109/SCC.2010.94.

[56] J. Kuskin, , et al. The Stanford FLASH multiprocessor. In *Proc. of ISCA*, pages 302–313, 1994.

[57] Kevin Lai et al. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Syst.*, 1:169–182, August 2005. ISSN 1574-1702.

[58] P. Larson and M. Krishnan. Memory allocation for long-running server applications. In *Proceedings of the 1st international symposium on Memory management*, pages 176–185, 1998.

[59] D. Lea. A memory allocator, 1996. URL `http://g.oswego.edu/dl/html/malloc.html`.

[60] Spyros Lyberis et al. The myrmics memory allocator: hierarchical,message-passing allocation for global address spaces. In *Proc. of Symp. on Principles and practice of parallel programming*. ACM, 2012.

[61] Stylianos Mamagkakis et al. Energy-efficient dynamic memory allocators at the middleware level of embedded systems. In *Proc. of EMSOFT*. ACM, 2006. ISBN 1-59593-542-8.

[62] T.G. Mattson, R.F. Van der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe. The 48-core scc processor: the programmer's view. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pages 1–11, 2010. doi: 10.1109/SC.2010.53.

[63] Matteo Monchiero et al. Exploration of distributed shared memory architectures for NoC-based multiprocessors. *JSA*, 53(10):719–732, 2007. ISSN 1383-7621. doi: http://dx.doi.org/10.1016/j.sysarc.2007.01.008.

[64] Srinivasan Murali and Giovanni De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. *Design, Automation and Test in Europe Conference and Exhibition*, 2:20896, 2004. ISSN 1530-1591. doi: http://doi.ieeecomputersociety.org/10.1109/DATE.2004.1269002.

[65] Srinivasan Murali and Giovanni De Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proc. of DATE*, page 20896. IEEE Computer Society, 2004. ISBN 0-7695-2085-5-2.

[66] K.J. Nesbit et al. Multicore resource management. *Micro, IEEE*, 28(3):6 –16, May-June 2008.

[67] V. Nollet et al. Centralized run-time resource management in a network-on-chip containing reconfigurable hardware tiles. In *Proc. of DATE*, pages 234–239. IEEE Computer Society, 2005. ISBN 0-7695-2288-2.

[68] Paul Pop et al. An approach to incremental design of distributed embedded systems. In *Proc. of DAC*, pages 450–455. ACM, 2001. ISBN 1-58113-297-2.

[69] R. Rajkumar et al. A resource allocation model for qos management. In *Proc. of RTSS*, pages 298 –307, 1997.

[70] S. K. Reinhardt et al. Tempest and Typhoon: user-level shared memory. In *Proc. of ISCA*, pages 325–336, 1994.

[71] Gerald Sabin et al. Moldable parallel job scheduling using job efficiency: an iterative approach. In *Proc. of JSSPP*, pages 94–114. Springer-Verlag, 2007. ISBN 978-3-540-71034-9. URL http://dl.acm.org/citation.cfm?id=1757044.1757049.

[72] Bratin Saha et al. McRT-STM: a high performance software transactional memory system for a multi-core runtime. In *Proc. of Symp. on Principles and practice of parallel programming*. ACM, 2006.

[73] T. Sakurai and A.R. Newton. Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas. *Solid-State Circuits, IEEE Journal of*, 25(2):584 –594, apr 1990. ISSN 0018-9200. doi: 10.1109/4.52187.

[74] Scott Schneider. Scalable locality-conscious multithreaded memor allocation. In *In Proc. of the 2006 ACM SIGPLAN International Symposium on Memory Management*, 2006.

[75] Larry Seiler et al. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27:18:1–18:15, August 2008. ISSN 0730-0301.

[76] Semiconductor Industry Association. International technology roadmap for semiconductors, 2006. URL http://www.itrs.net/Links/2006Update/2006UpdateFinal.htm.

[77] Semiconductor Industry Association. International technology roadmap for semiconductors, 2008. URL http://www.itrs.net/Links/2008Update/2008UpdateFinal.htm.

[78] Mohamed Shalan and Vincent J. Mooney. Hardware support for real-time embedded multiprocessor system-on-a-chip memory management. In *Proc. of CODES, Estes Park, Colorado, USA*, pages 79–84. ACM, 2002. ISBN 1-58113-542-4. doi: http://doi.acm.org/10.1145/774789.774806.

[79] Li Shang, Li-Shiuan Peh, and Niraj K. Jha. Powerherd: dynamic satisfaction of peak power constraints in interconnection networks. In *Proc. of ICS*, pages 98–108. ACM, 2003. ISBN 1-58113-733-8.

[80] Youngsoo Shin et al. Power optimization of real-time embedded systems on variable speed processors. In *Proc. of ICCAD*, pages 365–368. IEEE Press, 2000. ISBN 0-7803-6448-1. URL http://dl.acm.org/citation.cfm?id=602902.602984.

[81] SIA. Semiconductor industry association, international technology roadmap for semiconductors, 2011. URL http://www.itrs.net/Links/2009ITRS/Home2011.htm.

[82] Lodewijk T. Smit et al. Run-time mapping of applications to a heterogeneous soc. In *Proc. of SoC*, 2005.

[83] STMicroelectronics. STNoC: Building a new system-on-chip paradigm. White Paper, 2005.

[84] T. Mattson, Rob van der Wijngaart. RCCE: A small library for many-core communication, `http://www.intel.com/content/www/us/en/research/intel-labs-rcce-single-chip-cloud-brief.html`. URL `http://www.intel.com/content/www/us/en/research/intel-labs-rcce-single-chip-cloud-brief.html`.

[85] Justin Talbot et al. Phoenix++: modular mapreduce for shared-memory systems. In *Proc. of MapReduce*, pages 9–16. ACM, 2011. ISBN 978-1-4503-0700-0. doi: 10.1145/1996092.1996095. URL `http://doi.acm.org/10.1145/1996092.1996095`.

[86] Shyamkumar Thoziyoor and Naveen Muralimanohar. Cacti 5.0, technical report hpl-2007-167, hp labs, 2007.

[87] C. H. (Kees) van Berkel. Multi-core for mobile phones. In *Proc. of DATE*, pages 1260–1265. EDAA, 2009. ISBN 978-3-9810801-5-5.

[88] S. Vangal et al. An 80-Tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS. In *Proc. of ISSCC*, pages 98–589. IEEE, 2007.

[89] S. Vassiliadis et al. Microcode processing: Positioning and directions. *IEEE MICRO*, 23(4):21–30, 2003.

[90] Kiem-Phong Vo. Vmalloc: A general and efficient memory allocator, 1996.

[91] P. R. Wilson et al. Dynamic storage allocation: A survey and critical review. In *Proc. of IWMM, Kinross, Scotland, UK*, pages 1–116. Springer-Verlag, 1995. ISBN 3-540-60368-9.

[92] Sotirios Xydis et al. Custom mutli-threaded dynamic memory management for multiprocessor system-on-chip platforms. In *Proc. of ICSAMOS, Samos Island, Greece*, pages 102–109, jul. 2010.

[93] Terry Tao Ye, Luca Benini, and Giovanni De Micheli. Packetized on-chip interconnect communication analysis for mpsoc. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, 2003.

[94] Voon yee Vee and Wen jing Hsu. A scalable and efficient storage allocator on shared-memory multiprocessors. In *In International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN□99*, pages 230–235, 1999.

[95] Richard M. Yoo et al. Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system. In *Proc. of IISWC*, pages 198–207. IEEE Computer Society, 2009. ISBN 978-1-4244-5156-2. doi: 10.1109/IISWC.2009.5306783. URL `http://dx.doi.org/10.1109/IISWC.2009.5306783`.

[96] Wangyuan Zhang and Tao Li. Managing multi-core soft-error reliability through utility-driven cross domain optimization. In *Proc. of ASAP 2008*, 2008.