NATIONAL TECHNICAL UNIVERSITY OF ATHENS
School of Electrical & Computer Engineering

Division of Communication, Electronic and Information Engineering
Microprocessors and Digital Systems Lab

Design Methodologies and Tools for Energy-aware IoT-based Applications

Ph.D. Thesis

of

Charalampos C. Marantos

Supervisor: Prof. Dimitrios Soudris

Athens, September 2022

This work is partially supported by European Commission projects that received funding from the European Unions Horizon 2020 Research and Innovation Programme: *FABSPACE2.0*, *SDK4ED* and *EVOLVE*.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΫΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Design Methodologies and Tools for Energy-aware IoT-based Applications

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

Χαράλαμπου Χρήστου Μάραντου

Συμβουλευτική Επιτροπή: Δημήτριος Σούντρης
Κιαμάλ Πεκμεστζή
Κωνσταντίνος Σιώζιος

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 26η Σεπτεμβρίου 2022

.....	.....	.....
Δημήτριος Σούντρης	Βασίλειος Παλιουράς	Κωνσταντίνος Σιώζιος
Καθηγητής Ε.Μ.Π.	Καθηγητής Π.Π.	Αν. Καθηγητής Α.Π.Θ.
.....	.....	.....
Αλέξανδρος Χατζηγεωργίου	Σωτήριος Ξύδης	Γεώργιος Θεοδωρίδης
Καθηγητής ΠΑ.ΜΑΚ.	Επ. Καθηγητής Χ.Π.Α.	Αν. Καθηγητής Π.Π.

.....
Ηλίας Κοσματόπουλος
Καθηγητής Δ.Π.Θ.

Αθήνα, Σεπτέμβριος 2022

Χαράλαμπος Μάραντος
Διδάκτωρ Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Abstract

Green, sustainable and energy efficient computing terms are gaining more and more attention during the last years. As the number of Internet of Things (IoT) computing devices keeps increasing, energy efficiency is becoming an important requirement, imposing new challenges to software developers. Existing works vary significantly, depending on the abstraction level in which the energy efficiency is treated. On the one hand, from software engineering perspective, there are tools that suggest best practices and guidelines based on empirical studies. On the other hand, embedded system practitioners reduce energy either at hardware level or by making custom transformations at source-code level, using custom techniques, DSPs or memory management optimizations. As applications evolve, there is an increasing need to address energy efficiency at application source code level, beyond general guidelines. Therefore, software tools capable of providing energy consumption estimations and identifying optimization opportunities are vital for assisting developers during the phases of application development.

The goal of this dissertation is to introduce the design of application analysis tools that target energy efficiency at the software design level. The introduced tools, coupled with implementation details, are capable of estimating the expected energy consumption of applications running on multiple devices. The proposed tools suggest a number of optimizations to the user with special emphasis on estimating potential gains by acceleration. The presented methodology provides several features, including the combination of static analysis and dynamic instrumentation approaches in order to exploit the advantages of both. The potential use of the proposed methods towards building a tool that focuses on saving energy by suggesting efficient function placements on Edge devices is demonstrated. Finally, a special study of the impact of the suggested optimizations on software development, such as the programming effort, is introduced.

The recent increase in demand for IoT embedded systems, such as the control of Heating Ventilation and Air-Conditioning (HVAC) in buildings, motivated our study of a special use-case. HVAC control systems exhibit increased complexity and their operation relies less on human decision-making and more on computational intelligence. The efficiency of these systems is usually limited by the orchestrators' flexibility to optimize simultaneously multiple, and usually contrary, parameters. Throughout this thesis, we aim to introduce novel solutions for designing model-free orchestrators. Experimental results highlight the superiority of our solutions, as we achieve comparable performance to state-of-the-art relevant controllers without the need of any prior detailed modeling and requiring lower computational and storage

resourced without sacrificing the quality of derived results.

Keywords: Green Computing, Energy Consumption, Sustainable Computing, Software Design, Decision-making, Cyber-Physical Systems, Embedded Systems, Machine Learning, Smart Buildings.

Περίληψη

Η ενεργειακή αποδοτικότητα των υπολογιστικών πόρων που εκφράζεται συχνά με τους όρους Green και Sustainable computing κερδίζει όλο και μεγαλύτερη προσοχή τα τελευταία χρόνια. Καθώς ο αριθμός των υπολογιστικών συσκευών σε εφαρμογές Διαδικτύου των Αντικειμένων (IoT) συνεχίζει να αυξάνεται, η μείωση της ενέργειας των εφαρμογών αποτελεί πλέον μια σημαντική απαίτηση που επιβάλλει νέες προκλήσεις στους προγραμματιστές. Οι υπάρχουσες λύσεις ποικίλλουν σημαντικά, ανάλογα με το επίπεδο στο οποίο εξετάζεται η ενεργειακή αποδοτικότητα. Από τη μία πλευρά, πιο κοντά στο λογισμικό, υπάρχουν ερευνητικές εργασίες που προτείνουν βέλτιστες πρακτικές και δίνουν κατευθυντήριες γραμμές βασιζόμενες σε εμπειρικές μελέτες. Από την άλλη πλευρά, οι ερευνητές στον τομέα των ενσωματωμένων συστημάτων μειώνουν την ενέργεια είτε με βελτιστοποιήσεις στο ίδιο το υλικό, είτε μετασχηματίζοντας τον κώδικα της εφαρμογής, χρησιμοποιώντας εμπειρικές τεχνικές, π.χ. για την βελτιστοποίηση της διαχείρισης μνήμης. Καθώς οι εφαρμογές εξελίσσονται, δημιουργείται η ανάγκη αντιμετώπισης της αυξημένης ενεργειακής κατανάλωσης στο επίπεδο του πηγαίου κώδικα της εφαρμογής από τους ίδιους τους προγραμματιστές. Ως εκ τούτου, η δημιουργία εργαλείων λογισμικού ικανών να παρέχουν εκτιμήσεις κατανάλωσης ενέργειας και να προτείνουν βελτιστοποιήσεις, είναι πλέον εξαιρετικά σημαντική, προκειμένου να παρέχεται βοήθεια στους προγραμματιστές σε όλες τις φάσεις ανάπτυξης IoT εφαρμογών.

Στόχος της παρούσας διατριβής είναι η σχεδίαση εργαλείων ανάλυσης εφαρμογών που στοχεύουν στην ενεργειακή αποδοτικότητα. Οι προτεινόμενες λύσεις, οι οποίες συνδυάζονται και με λεπτομέρειες υλοποίησης, παρέχουν εκτιμήσεις της αναμενόμενης κατανάλωσης ενέργειας των εφαρμογών πριν αυτές εκτελεστούν στις συσκευές. Τα προτεινόμενα εργαλεία παρέχουν προτάσεις βελτιστοποιήσεων στον χρήστη με ιδιαίτερη έμφαση στην εκτίμηση των πιθανών ενεργειακών κερδών από την επιτάχυνση της εφαρμογής σε GPU. Επιπλέον, εξετάζεται η επέκταση τους στην δημιουργία ενός συστήματος αποφάσεων ενεργειακά αποδοτικών τοποθετήσεων των επιμέρους συναρτήσεων των εφαρμογών στις διαθέσιμες συσκευές του δικτύου. Η μεθοδολογία που παρουσιάζεται στην διατριβή, έχει πολλά καινοτόμα χαρακτηριστικά, όπως τον συνδυασμό στατικής και δυναμικής ανάλυσης προκειμένου να αξιοποιούνται τα πλεονεκτήματα και των δύο τεχνικών. Επιπλέον, παρουσιάζεται μια ειδική μελέτη της επίδρασης των προτεινόμενων βελτιστοποιήσεων στην ανάπτυξη λογισμικού, όπως π.χ. η εκτίμηση της προσπάθειας που πρέπει να καταβληθεί για να εφαρμοστούν.

Η πρόσφατη αύξηση της ζήτησης για νέες εφαρμογές IoT, όπως για παράδειγμα για τον έλεγχο της ενέργειας και του κλιματισμού στα σύγχρονα έξυπνα κτίρια, κινητοποίησε τη μελέτη μιας ειδικής περίπτωσης. Τα συστήματα αυτά παρουσιάζουν αυξημένη πολυπλοκότητα και η λειτουργία τους βασίζεται

λιγότερο στην ανθρώπινη λήψη αποφάσεων και περισσότερο στην υπολογιστική νοημοσύνη. Σε αυτή τη διατριβή εισάγονται νέες λύσεις των οποίων τα πειραματικά αποτελέσματα δείχνουν συγκρίσιμες επιδόσεις με παρόμοιους ελεγκτές τελευταίας τεχνολογίας, χωρίς όμως την ανάγκη προηγούμενης λεπτομερούς μοντελοποίησης των κτιρίων και απαιτώντας πολύ χαμηλότερους υπολογιστικούς πόρους.

Λέξεις Κλειδιά: Ενεργειακή Αποδοτικότητα, Ενεργειακή Κατανάλωση, Σχεδίαση Λογισμικού, Ανάλυση Υπολογιστικών Απαιτήσεων, Λήψη Αποφάσεων, Κυβερνο-φυσικά Συστήματα, Ενσωματωμένο Σύστημα, Μηχανική Μάθηση, Έξυπνο Κτίριο

Acknowledgements

First of all, I would like to thank my supervisor Prof. Dimitrios Soudris for the trust, the support and the guidance. Our conversations gave me valuable memories and knowledge and broadened my horizons and my way of thinking, while I will never forget his genuine interest and appreciation.

I probably would not have started this doctorate if I had not met Prof. Kostas Siozios. We met in 2016 when he supervised my diploma thesis. Since then he was always there to give me advice and knowledge, while our cooperation is still active.

Then I would like to thank Dr. Lazaros Papadopoulos with whom we collaborated very actively in the SDK4ED project and the products of our collaboration and his proposals are part of my dissertation.

Afterwards, I would like to thank Christos Lambrakos. We met in 2017 when he did his diploma. Parts of the present dissertation would not be the same without our collaboration that has already lead to five publications.

In the context of research projects that I worked on, I also met people that I would like to thank. I will make a special reference to the partners of the SDK4ED project and specifically to Prof. Alexandros Chatzigeorgiou from University of Macedonia who in collaboration with his team show me the world of software quality and our collaboration led to a best paper award at the IGSC conference as well as the CERTH team under the coordination of Dr. Dionisios Kehagias and especially Dr. Miltiadis Siavvas (CERTH) with whom we often talked, exchanged views, organized new experiments and publications. Other important collaborations include prof. Francky Catthoor (KU Leuven and imec), who organized and supported a research collaboration between Belgium, Japan (prof. Ittetsu Taniguchi and phd student Daichi Watari) and Greece that led to valuable publications, as well as Dr. Iosif Paraskevas who gave me knowledge on management tasks on FabSpace 2.0 project.

Next I would like to refer to people that collaborated with me for their diploma theses. Their work gave me valuable knowledge, and parts of it are included in this dissertation. I would especially like to thank Konstantinos Salapas with whom we also collaborated in SDK4ED and his patience and hard work contributed the most. The very organized work of Nikolaos Maidonis, with whom we collaborated during the difficult period of the coronavirus, gave very useful conclusions. Giannos Gavriilides with his characteristic ease of learning new things helped me to see things that I knew in a new environment and Chrysostomos Karakassis introduced me to the world of robotics.

Nothing would be the same without Microlab's wonderful company. Apart

from partners, Microlab members are also friends. One by one, they made each day pleasant and unique, keeping the fun of research work undiminished. In order not to upset anyone, I will refer only to those that we co-authored publications, namely Achilleas Tzenetopoulos who gave me valuable help in technical matters related to experiments with dockers and Kubernetes and Vasileios Leon.

The feedback of Prof. Sotirios Xydis was also very important for improving the quality of some parts of the present document.

Finally, I would like to thank my friends who have been listening to my joys and concerns all these years. Leonidas Moustakas, for our endless discussions, Savvas Koulepoglou that we started ECE school together back in 2010 and Konstantinos Kontos.

Without my family this thesis would never have existed. My parents have always supported me since my childhood with love and interest and gave me the opportunity to study and realize my goals and dreams. To my mother (Lina), to my father (Christos) and to my sister (Korina), I love you.

Ευχαριστίες

Θα ήθελα να γράψω τις ευχαριστίες και στην γλώσσα μας καθώς είναι πολλά τα συναισθήματα που με κατακλύζουν στο τέλος αυτού του μεγάλου ταξιδιού.

Ένα μεγάλο ευχαριστώ στον επιβλέποντα καθηγητή μου κ. Δημήτριο Σούντρη για την εμπιστοσύνη, την στήριξη και την καθοδήγηση. Πέρα από την άψογη επαγγελματική μας συνεργασία, οι συμβουλές του και οι συζητήσεις μας μου έχουν δημιουργήσει μνήμες και γνώσεις που δεν θα ξεχάσω ποτέ, ενώ παράλληλα μου έδωσαν ένα τρόπο σκέψης και εργασίας που θα με ακολουθεί σε όλα μου τα βήματα. Το αληθινό του ενδιαφέρον σε όλα τα επίπεδα της ζωής, το οποίο διαπιστωνόταν σε όλη την διάρκεια της συνεργασίας μας, είναι συγκινητικό.

Πιθανότατα δεν θα είχα ξεκινήσει ποτέ διδακτορικό αν στον δρόμο μου δεν βρισκόταν ο κ. Κώστας Σιώζιος. Συναντηθήκαμε το 2016 στην διπλωματική μου και έπειτα ήταν εκείνος που μου πρότεινε να δουλέψω ερευνητικά δίνοντας μου κίνητρα και στόχους. Έκτοτε σαν φίλος βρισκόταν εκεί να μου δίνει συμβουλές και γνώσεις, ενώ η συνεργασία μας παρέμεινε πάντα ενεργή.

Έπειτα θα ήθελα να ευχαριστήσω τον διδάκτορα Λάζαρο Παπαδόπουλο με τον οποίο συνεργαστήκαμε πολύ ενεργά στο έργο SDK4ED και προϊόντα της συνεργασίας μας είναι παρόντα σε πολλά κομμάτια της διατριβής μου και δεν θα ήταν ίδια χωρίς τις πολίτιμες προτάσεις και την καθοδήγησή του.

Στην συνέχεια, οφείλω να ευχαριστήσω τον συνεργάτη Χρήστο Λαμπράκο. Συναντηθήκαμε το 2017 όταν έκανε την διπλωματική του. Οι ιδέες και το μεράκι του είναι πολύ χαρακτηριστικά. Είναι εκείνος που μου γνώρισε τον κόσμο του Reinforcement Learning και κομμάτια της παρούσας διατριβής δεν θα ήταν ίδια χωρίς την συνεργασία μας η οποία μετράει ήδη πέντε δημοσιεύσεις.

Με την ευκαιρία των ερευνητικών έργων συνεργαστήκαμε με ανθρώπους που μου έδωσαν πολύτιμες εμπειρίες και γνώσεις. Θα κάνω ιδιαίτερη αναφορά στην ομάδα της Θεσσαλονίκης από το έργο SDK4ED και συγκεκριμένα στον καθηγητή του Πανεπιστημίου Μακεδονίας κ. Αλέξανδρο Χατζηγεωργίου που σε συνεργασία με την ομάδα του γνώρισα τον κόσμο του software quality και η συνεργασία μας οδήγησε σε δημοσίευση για την οποία λάβαμε βραβείο στο συνέδριο IGSC καθώς και την ομάδα του δρ. Διονύσιου Κεχαγιά από το ΕΚΕΤΑ. Ξεχωριστά θα αναφέρω τον δρ. Μιλτιάδη Σιάββα με τον οποίο συχνά μιλούσαμε, ανταλλάζαμε σκέψεις, οργανώναμε νέα πειράματα και δημοσιεύσεις. Άλλες σημαντικές συνεργασίες ήταν με τον καθηγητή Francky Catthoor (KU Leuven και imec) ο οποίος στήριξε μια συνεργασία μεταξύ Βελγίου, Ιαπωνίας (καθ. Ittetsu Taniguchi - ΥΔ Daichi Watari) και Ελλάδας στο ερευνητικό θέμα της διαχείρισης ενέργειας στα έξυπνα κτήρια που έχει οδηγήσει ήδη σε σημαντικές δημοσιεύσεις καθώς και εκείνη με τον δρ. Ιωσήφ Παρασκευά σε διαχειριστικά θέματα κατά το έργο FabSpace 2.0.

Έπειτα θα ήθελα να αναφερθώ σε ανθρώπους που συνεργαστήκαμε για την διπλωματική τους. Η συνεργασία μας μου πρόσφερε πολύτιμες γνώσεις, ενώ κομμάτια από αυτήν αποτελούν μέρος της παρούσας διατριβής. Ιδιαίτερα θα ήθελα να αναφερθώ στον Κωνσταντίνο Σαλάπα με τον οποίο δουλέψαμε μαζί και σε ερευνητικό έργο και ο οποίος με πρωτόγνωρη υπομονή και εργατικότητα συνέβαλε τα μέγιστα. Η πολύ οργανωμένη δουλειά και το κέφι του Νικόλαου Μαϊδώνη, με τον οποίο συνεργαστήκαμε στην δύσκολη περίοδο του κορονοϊού, έδωσε πολύ χρήσιμα συμπεράσματα και μετρήσεις. Ο Γιάννος Γαβριηλίδης με την χαρακτηριστική ευκολία του να μαθαίνει νέα πράγματα βοήθησε να δω πράγματα που γνώριζα μέσα σε ένα νέο περιβάλλον και ο Χρυσόστομος Καρακάσσης με έβαλε στον άγνωστο για μένα κόσμο της ρομποτικής.

Τίποτα στον δρόμο αυτό δεν θα ήταν ίδιο χωρίς την υπέροχη παρέα του Microlab. Πέρα από συνεργάτες είναι και φίλοι. Όλοι ένας ένας ξεχωριστά έκαναν την κάθε μέρα ευχάριστη και μοναδική και το κέφι για δουλειά αμείωτο. Για να μην δυσαρεστήσω κανέναν θα αναφερθώ μόνο σε εκείνους που έχουμε και δημοσιεύσεις μαζί και συγκεκριμένα τον Αχιλλέα Τζενετόπουλο, του οποίου η βοήθεια και η τεχνική υποστήριξη σε πειράματα που εμπεριείχαν Docker και Kubernetes υπήρξε πολύτιμη και τον Βασίλη Λέων.

Οι συμβουλές και οι επισημάνσεις του κ. Σωτηρίου Ξύδη ήταν επίσης πολύ σημαντικές ως προς την ποιοτική βελτίωση του κειμένου ορισμένων κεφαλαίων της διατριβής.

Τέλος θα ήθελα να ευχαριστήσω τους παιδικούς φίλους μου που άκουγαν την γκρίνια μου στις αποτυχίες αλλά και τις χαρές μου στις επιτυχίες όλα αυτά τα χρόνια. Τον Λεωνίδα Μουστάκα, για τις ατέλειωτες συζητήσεις μας, τον Σάββα Κουλέπογλου που μαζί ξεκινήσαμε την σχολή το 2010 στο προπτυχιακό και τον Κων/νο Κοντό.

Χωρίς την οικογένεια μου δεν θα υπήρχε ποτέ αυτό το κείμενο. Οι γονείς μου με στήριξαν από τα παιδικά μου χρόνια πάντα με αγάπη και ενδιαφέρον δίνοντάς μου την ευκαιρία να σπουδάσω, να πραγματοποιήσω τους στόχους και τα όνειρά μου. Στην μητέρα μου (Λίνα) στον πατέρα μου (Χρήστο) και στην αδερφή μου (Κορίνα), σας ευχαριστώ και σας αγαπώ.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC	Analog-to-Digital Converter
AF	Available Funds
ALU	Arithmetic Logic Unit
ANN	Artificial Neural Network
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AST	Abstract Syntax Tree
BCVTB	Building Controls Virtual TestBed
BEM	Building Energy Management
CNN	Convolutional Neural Network
CPS	Cyber-Physical System
CPU	Central Processing Unit
CSV	Comma-Separated Values
DRAM	Dynamic Random-Access Memory
DSP	Digital Signal Processing
DVFS	Dynamic Voltage and Frequency Scaling
EBO	Event -Based Optimization
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
GPGPU	General Purpose Graphics Processing Unit
GPIO	General-Purpose Input/Output
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDD	Hard Disk Drive
HPC	High Performance Computing
HVAC	Heating Ventilation and Air Conditioning
HW	Hardware
I/O	Input/Output
IACA	Intel Architecture Code Analyser
ILP	Instruction Level Parallelism
IoT	Internet of Things
IR	Intermediate Representation
JIT	Just In Time

JVM	Java Virtual Machine
LoC	Lines of Code
LOOCV	Leave-One-Out Cross-Validation
LR	Linear regression
LSS	Large-Scale system
MAE	Mean Absolute Error
MCKP	Multiple-Choice Knapsack Problem
ML	Machine Learning
MOO	Multi-Objective Optimization
MPC	Model Predictive Control
MSE	Mean Squared Error
NN	Neural Network
PC	Personal Computer
PCIe	Peripheral Component Interconnect Express
PMV	Predicted Mean Vote
PPD	Predicted Percentage of Dissatisfied
PV	PhotoVoltaic
QoS	Quality of Service
RAM	Random-Access Memory
RBC	Ruled Based Controller
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
SDK	Software Development Kit
SoC	System-on-Chip
SoTA	State of The Art
SW	Software
TLB	Translation Lookaside Buffer
TPU	Tensor Processing Unit
UI	User Interface
VLIW	Very Long Instruction Word
WCT	Worst-Case execution Time

Chapter 1

1 Introduction

1.1 Internet-of-Things

Internet of Things (IoT) edge devices are now installed in various environments, e.g. factories, hospitals and residential buildings. The interconnected devices are estimated to transmit up to 79.4 zettabytes of data over the Internet, further increasing the demands for electric power and the impact on the CO_2 emissions generated by electricity production [2] [3].

The IoT market is expected to achieve a big global economic value, as the current predictions show a sharp increase in the computing devices revenue in the next years. This big increment is going to be enhanced by the upcoming 5G technology, which is expected to enable the support of more AI applications at the Edge[1]. The continued growth of IoT will lead to 41.6 billion connected devices by 2025 [2]. Another study by IHS (as depicted in Figure 1) predicts that the total number of installed connected computing devices is expected to reach 75.4 billion in 2025[2].

As IoT applications gain increased attention, application developers target devices with constraints in terms of power, memory capacity and energy consumption. On the cloud computing side, the amount of data being processed is also increasing, leading to greater use of accelerators and memory management techniques to reduce energy consumption. As a result, the growing number of IoT applications [4] has created new challenges for software developers. Demanding computations, such as image processing and Neural Network training, were traditionally performed on the Cloud layers leading to an increase of the energy consumed by data-centers. However, in case of applications which are sensitive to latency or when data security is critical, offloading computations to the Cloud may violate application requirements. Therefore, demanding computations are often performed in-place at the Edge

[1] `https://www.iotworldtoday.com/2019/03/18/how-5g-could-help-fuel-the-next-generation-of-iot-projects/`

[2] `https://ihsmarkit.com`

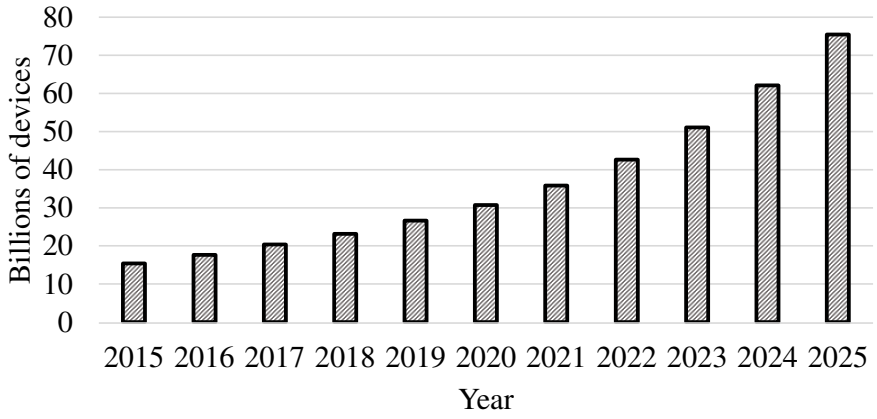


Figure 1: Estimated number of connected IoT devices[2]

or the Fog layers of the network [5].

1.2 Green/Sustainable Software

Since low energy consumption of battery-operated edge devices is a critical design constraint and reducing the energy footprint of High Performance Computing (HPC) infrastructure is mandatory, the development of energy efficient software becomes a significant challenge for application developers. Therefore, bringing the design principles of energy efficient development closer to the software engineering perspective is now an active research topic [6].

Although improvements at the hardware level are typically a key factor to achieve energy savings, software design also significantly affects the consumption of applications. The importance of tools that aim to optimize these designs during the Software Development Life Cycle is highlighted in recent survey studies [6]. Inefficient software in terms of energy consumption can waste the system's energy [7].

Accelerators integrated into edge computing devices target not only performance improvements, but also energy efficiency [8]. Indeed, the growth of image processing and machine learning embedded applications, contributed to the evolution of embedded system architectures towards heterogeneity. Acceleration units, such as GPUs and FPGAs are often integrated in embedded systems to enable efficient execution of computationally intensive algorithms. This method reduces the amount of data transmitted to Fog and Cloud and further improves the energy efficiency of the whole network. Typical examples include CPU-GPU System-on-Chips (SoC) such as Nvidia Tegra, which incorporates an embedded GPU [9], custom VLIW architec-

tures such as Intel/Movidius Myriad [10] [11] and CPU-FPGA SoCs, such as Xilinx Zynq 7000 Series [12].

Green, sustainable and energy efficient computing terms are gaining more and more attention [13]. However, software engineers still either do not take energy into consideration [14], or lack the necessary expertise [15]. Although, from the software engineering perspective, there are tools that propose best practices and guidelines for software development, they are based on empirical studies [16], without specific structure and tool-support. On the other side of the coin, a large number of tools and methods are proposed in the literature for embedded or hardware developers, targeting specific devices and architectures or requiring access to specific hardware, sophisticated equipment and special knowledge.

Therefore, based on the tools and methodologies developed in the context of embedded computing, it is necessary to provide tools that will support software engineers towards the development of energy efficient software. There is still a limited number of tools that can actually support developers to evaluate their software in terms of energy consumption [17]. To make things worse, the techniques and refactorings that are proposed are stated to be too complex for software developers [15] [16]. Although, a large number of techniques that aim to contribute to the energy efficient software development can be found in the literature (especially regarding the implementation phase of a Software Development Life-cycle) such as Parallel Programming [18], Approximate Computing [19], Data Structures [20], and other Practices [21], they are difficult to be adopted by a non-experienced software developer.

More and more tools and services are being introduced in order to properly use modern heterogeneous devices from the software level and without the need for specialized hardware knowledge. Typical examples can be found in the field of machine learning, where libraries such as Tensorflow, Pytorch etc. offer the ability to use GPUs and TPUs easily from the Python source code level. Typical examples include high level languages, like CUDA and OpenCL, which are used by software engineers in order to exploit the acceleration capabilities of the targeted heterogeneous devices. However, software developers still need support about how and when to use these features, while a significant programming effort is required to effectively employ the available accelerators and bring the computational load of such complex algorithms within their power envelope.

Another very important limitation of existing tools is the fact that they target specific architectures and programming environments. Furthermore, they require significant time and effort to be configured and used. Existing tools use hardware-specific performance and energy models, for associating basic software constructs (source code blocks or basic blocks in the interme-

diate representation) with energy consumption [22]. New tools that will offer energy consumption estimation in the source code level (with the minimum possible interaction with the targeted hardware), even less accurate, will enable continuous monitoring and provide feedback to the developers, assisting them to have a view of the energy consumption of their code [6].

Recent survey studies in the field of software engineering highlight the development of tools for energy efficient development targeting software engineers as of great importance [6]. More specifically, they highlight the need of tools that will help inexperienced developers to identify the most energy expensive parts of an application and to adopt techniques that are applicable and have the potential to reduce energy consumption.

A promising solution is the performance prediction: Tools that fall into this category estimate the performance of a piece of CPU code on various devices or the potentiality to exploit an accelerator. In this way, they guide application developers in the early design choice of using a specific device or offloading a piece of code to an accelerator without requiring tedious redevelopment effort for testing and/or access to the actual hardware. These tools, however, only targets performance, while non of the existing solutions considers software engineering metrics such as the effort required to develop the accelerated version of a piece of CPU code.

1.3 Cyber-Physical Systems

Recently, the convergence of embedded computing and information technology became also a key enabler for new generation products. These systems bridge the *cyber* world of computing and communications with the *physical* world and are referred to as Cyber-Physical Systems (CPS) [23]. Specifically, a CPS is a collection of task-oriented, or dedicated, systems that pool their resources and capabilities together to create a new, more complex ecosystem, which offers increased functionality and performance. This new design paradigm has the ability to interact with, and expand the capabilities of physical world through monitoring, computation (i.e., distributed coordination), communication, and decision-making mechanisms [24]. A typical CPS block diagram is depicted in Figure 2. According to this representation, a CPS can be considered as a tight integration of cyber and physical components. The term *cyber* refers to any computing hardware/software resources that perform data processing, computation, communication and decision-making services. Similarly, the *physical* components include any natural or human-made systems such as the environment, appliances, even humans that communicate with the cyber part through sensors that express the physical system state and actuators that receive and apply the cyber part's decision.

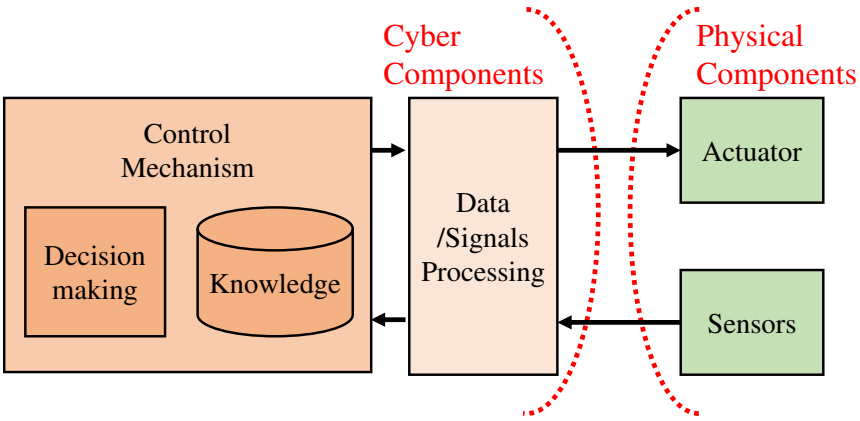


Figure 2: The block diagram of a typical cyber-physical system (CPS)

We might say that cyber part operates in discrete time, while physical part is governed by the laws of physics and operate in continuous time. The goal of a CPS is the effective combination of all these subsystems, while the most critical part is the decision-making mechanism, that controls all the individual modules.

Another definition of CPS includes three main components: *Sensing, Computing* and *Communication*. In this definition, *sensing* belongs to the aforementioned physical components, *computing* in cyber components and *communication* offers the bridge between the two parts [23].

By pooling the system's resources and capabilities together, CPS offers a new, more complex system with additional functionality and a number of sub-systems. This design enables a better resolution of the physical world and thus, advanced mechanisms of detecting the occurrence of an event. As a result, CPS are expected to play a key role in the development of next-generation autonomous systems. CPS are widely used and provide solutions in a large number of state-of-the art fields. They can be found in automotive, in sanitary environments, in data centers, in telecommunication networks, in industry, in robotics etc.

Since the physical world is bounded by unpredictability, it is not prudent to expect the CPS to be operating in a fully-controlled environment; thus, solutions that rely on *robust* and *self-adaptable* to unexpected conditions controllers are of upmost importance: These controllers (also called orchestrators) will be responsible for orchestrating the CPS sub-components and drive their operation automatically. Formally, self-adaptivity refers to systems that adjust their behavior autonomously in response to external events unpredictable, or unforeseen, at design time [25]. The orchestrators'

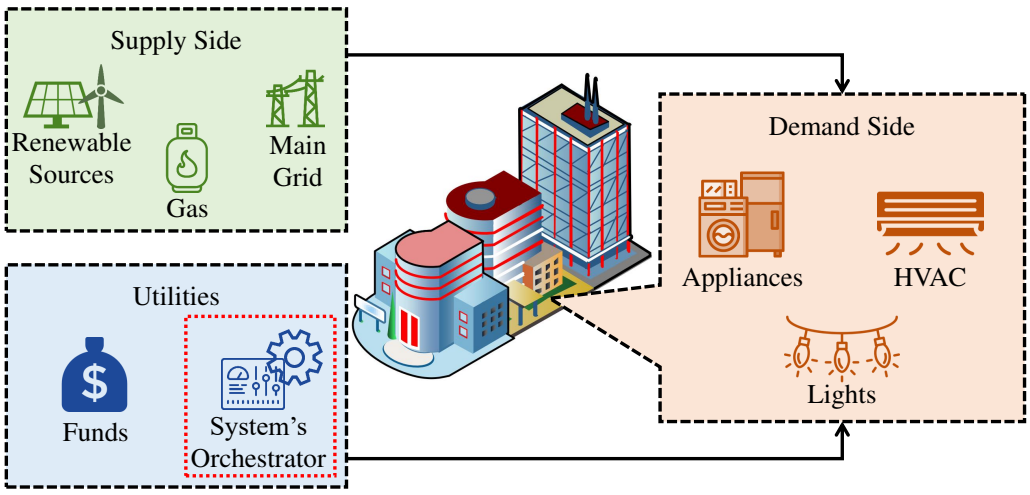


Figure 3: Overview of the a smart-grid environment

selections are usually defined according to the system's cost function. Since multiple (usually conflicting) objectives have to be taken into account simultaneously, the orchestrator relies on advanced decision-making mechanisms for this purpose. The efficiency of a CPS relies mainly on the employed algorithm(s) that perform system's orchestration under real-time constraints. Existing approaches [26] [27] [28] towards this direction rely on computational intensive decision-making algorithms that are executed onto powerful processing core(s).

1.3.1 Smart-grid - HVAC

CPS are widely-used in modern smart-grid environments. A template of a micro-grid environment is depicted at Figure 3. More precisely, for introducing the micro-grid structure and capabilities we assume that it is composed by three entities:

- The *supply side*: It contains the available energy sources (e.g., solar, wind, bio-gas)
- The *demand side*: It contains the energy consuming devices found in buildings such as the appliances and the cooling/heating systems.
- The *utilities*: They refer to the mechanisms that guarantee optimal control of system's components (supply and demand sides) and here is the focus of the present study

Smart-grids include a number of sensors that acquire data (such as weather conditions). Also they are capable of performing energy trading for buying/selling energy depending on the building's requirements and the available funds (AF). Finally, micro-grids are usually connected either directly to other micro-grids, or to the main grid [29].

Buildings are immensely energy-demanding and are expected to consume even more in the near future. European Union's (E.U.) buildings consume around 40-45% of the total energy consumption, whereof two-thirds of which is used in dwellings [30]. Heating, Ventilation and Air-Conditioning (HVAC) is the largest contributor to the buildings's energy cost and carbon emissions.

A large number of studies aim at mitigating this problem and its financial and ecological costs. Next generation materials, innovative design methods and new appliances [31] are used when constructing new buildings including improvements to the insulation and less energy consuming HVAC systems. New green building can be benefit also from the usage of renewable sources, while fine-tuning and intelligent control of HVAC systems is applied to all buildings, regardless of their age. Such a solution is given by Smart Thermostats: Computerized embedded platforms that apply advanced control methods on HVAC systems. These systems promise to reduce energy consumption while improving thermal conditions by using proper configuration of the HVAC system at real-time, based on environmental parameters and occupants preferences. Recent reports highlight the continuously increasing revenue of the global smart thermostat market[3] [32].

This type of devices targets the problem of non-optimal strategies for the building's cooling and heating services that increases the energy consumption even more. The expected energy savings that can be achieved are very high, considering that 20–30% of the building's energy requirements could be reduced just by turning off the HVAC system when residents are sleeping or away [33].

Although promising, the programmable thermostats are too difficult to be used effectively for the majority of people. For example, a lot of households with programmable thermostats have higher energy consumption on average than those with manual thermostat control because users program them incorrectly [34]. All these limitations can partially alleviated by CPS that enable a better resolution of the physical world (weather conditions, renewable resources production, energy prices). Thus, CPS are expected to to play a key role in controlling smart buildings and micro-grids subsystems [33].

However, the CPS technology also faces a number of challenges and con-

[3] `https://www.fortunebusinessinsights.com/u-s-smart-thermostat-market-106393`

straints. The control of HVAC uses sensors and processing nodes that monitor system's dynamic parameters and evaluate multiple operating scenarios. Due to the large problem's scalability, the CPS control mechanisms exhibit increased requirements for the data processing and storage. Additionally, as the physical world is not entirely predictable, the CPS operation is not expected in a fully-controlled environment. So, the CPS orchestrator has to be robust enough to unexpected events at run-time. A continuous demand both in academia and industry for providing higher flexibility during the service and product development of CPS orchestrators is observed, and rapid prototyping of plug&play solutions that can be sufficiently operate onto low-performance (and hence low-cost) embedded devices is always a deign goal [23].

1.4 Contribution and Structure

The contributions of the work presented in the context of this disertation can be summarised as follows:

- *A holistic methodology for designing software analysis tools to be used by developers for estimating energy consumption for running an application on multiple embedded devices is introduced* (Chapter 4).
  - Cross-device energy estimation methods are designed based on both dynamic and static analysis techniques, trading between accuracy and usability. The introduced methods use datasets, popular profiling tools and well-established regression estimators achieving an R2 score of 0.96. By studying the correlation between the various metrics (features) and energy consumption, as well as by comparing predicting methods we aim to contribute on constructing practical and easy-to-develop energy estimation tools.
- *Novel techniques that guide energy optimizations at the software level are designed. More precisely, special emphasis is given on estimating the potential energy gains by offloading parts of the source code to a GPU accelerator of a heterogeneous system* (Chapter 5).
  - The proposed approach combines both static and dynamic analysis approaches and exploits the advantages of both into a single flow. While the static analysis components rely on analysing source code, the dynamic estimation part, extends existing approaches that provide performance predictions, towards estimat-

ing the potential energy gains too achieving a classification accuracy of more than 75%.

  - Impact of energy related optimizations on software design quality is studied, making also a first step towards designing a tool that estimates programming effort prior development. The results are very promising as the proposed solution estimates the programming effort (expressed using the Halstead effort metric) achieving an accuracy of 85%.
  - The potential use of the introduced models to build another type of software analysis tool that gives energy-aware placement solutions of the application's individual functions on Edge devices, is demonstrated. The first results are promising, as the presented solution achieves 33.6% energy savings (on average) for using 4 edge devices to place 25 functions and 8.5% for using 3 edge devices, compared to the default Kubernetes placement.
  - An application analysis framework, aiming to integrate the introduced methods that contribute to the identification of energy consumption issues and propose relevant optimizations is designed. A pre-alpha version that includes individual micro-services, combined with a database, github support and graphical user interface is demonstrated, offering the capability of integrating different kinds of application analysis tools in a convenient and user-friendly way.

- *Novel online decision-making methods for optimal configuration and rapid prototyping of Cyber-Physical Systems applications, such as the HVAC systems in micro-grid environment, are presented. The method exhibits remarkable lower computational and storage complexities without sacrificing the quality of derived results* (Chapter 6).

  - One-way solutions based on Reinforcement Learning as well as a custom solution that incorporates *Linear Regression* (LR) and *Multiple-Choice Knapsack* algorithms are designed and tested. The problem and the objectives of the introduced decision-making algorithm were formulated towards being flexible and supporting multiple operating modes, such as balancing energy consumption with residents' satisfaction, minimizing energy consumption while maintaining a satisfactory level of thermal comfort, and maximizing residents' satisfaction without exceeding the available energy budget. The introduced fast and accurate models for estimating

the residents' thermal comfort and HVAC's energy consumption exhibit negligible complexity compared to relevant implementations without any quality degradation. More precisely, the final version of the proposed orchestrator exhibits comparable efficiency against the initial offline solver (3% worse in terms of total cost reduction) but with significant lower complexity (about 8 orders of magnitude) and without using any prior knowledge or system modeling. As a result, the introduced solutions contribute on targeting low-cost embedded devices (e.g. smart thermostats) and on enabling shorter design times, which in turn alleviates the time-to-market pressure.

The rest of the dissertation is structured as follows: A detailed description of the motivation behind the presented work along with a detailed survey on the related approaches is presented in Chapter 2. Chapter 3 describes the technical background that is needed in order for the reader to understand more easily the rest of the document. Chapter 4 describes the designed solutions towards calculating the expected energy consumption of programs running on different devices and architectures (cross-device), while Chapter 5 presents the proposed methodology with all the design choices for energy optimization along with implementation details and software engineering aspects analysis. Our solutions applied in the special application of the Smartgrid use-case are presented in Chapter 6. Finally, in Chapter 7 we draw conclusions and we discuss a number of observations, potential issues and future directions.

Chapter 2

2 Motivation and Related work

This Chapter gives a brief overview of the related research activities to the present work, aiming at giving the reader an understanding of the motivation behind our study. More detailed and specified related work sections are also found in each of the following Chapters, giving a more detailed view of the relevant approaches to the specific problem on which each Chapter focuses, including also quantitative and qualitative comparisons.

2.1 Green Computing - Existing approaches

Back in 2017, Pinto et al. [15] emphasize software developers' lack of knowledge about energy. More precisely, while 70% of developers participated in a study, thing that energy is important, 50% could not find what consumes energy in their apps. Indeed, 32% tried to improve the consumption unsuccessfully. The same conclusion was made by Pang et al. [17], where a representative number of developers gave very different answers about which part of their application consumes more energy.

In fact, although traditionally energy efficiency was studied on the hardware level, green computing is gaining more attention from the software engineering community [6]. Relative sessions are included in Software Engineering conferences and journals, while new research groups are specialized on this field (e.g. Green Software Lab[4]). However, the relevant work, includes completely different approaches, while recent survey articles highlight the need for tools and practices also influenced by the author's background. As a result, we can conclude that we need approaches that build a "bridge" between the various sectors.

In Figure 4, we make a symbolic representation of the current approaches that can be found in the literature, placing them on an axis between a software perspective and a hardware point of view. We refer to the knowledge of embedded computing as "hardware", while "software" refers to approaches

[4] `https://greenlab.di.uminho.pt/`

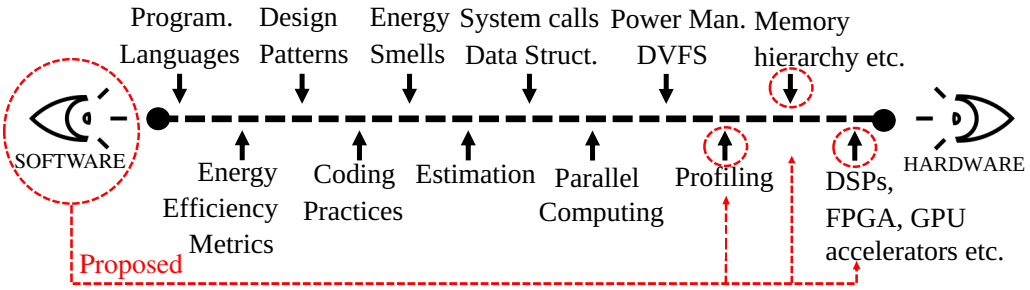


Figure 4: Green computing from different point of views

that mainly focus on the high-level source code and application development. As we point out in the following paragraphs, the red lines indicate the need to provide the software developer with a clear picture of the energy improvement opportunities that are closer to hardware knowledge. The following paragraphs provide a brief overview of some representative approaches about energy efficiency, green computing and sustainability, starting from the software engineering point of view and moving on to the hardware, in order to give a clear picture of what is proposed in this thesis.

Programming Languages: Recent work, even in 2021 [35], ranks the choice of programming languages in terms of energy efficiency. After developing the same algorithms in different languages, they draw conclusions about the energy consumption. For example, according to [35], Python consumes 75 times more energy than C. Many similar works can be found in the literature in recent years (2015-2020) [36]. Most of them, evaluate different implementations on workstations, embedded systems or mobile devices and compare the energy consumption of memory allocation methods, structures, etc.

Software Energy Efficiency Metrics: Approaches that belong to this category propose new Energy Efficiency metrics that assess how efficient a software application is. Capra et al. [37] introduced a metric based on the size of the source code, the application age, the type of internal applications (e.g. web, image processing) and the framework entropy [38] which is a measure of the degree that external libraries and application development environments have been used by the developers. Based on this metric, the authors conclude that intensive use of development environments has a positive impact on energy for small applications, but increases energy for large projects.

Design Patterns: There are a lot of approaches in the software engineering area that study which Software Design Pattern is the most energy efficient [39]. These approaches are based on empirical evaluation only [40]. According to them, for example, Flyweight, Mediator and Proxy [41] design pat-

terns are more energy efficient than Decorator and Abstract Factory [41] [42]. However, these works do not provide clarifications and explanations of the results and we might also argue that there are many inconsistencies in the observations depending on the studied application.

Coding Practices: Another type of green computing approaches, also closer to the software engineering perspective, suggests coding practices and guidelines for energy efficiency, i.e. sets of coding rules that help software practitioners to reduce energy consumption [43] [44]. These practices are also based on empirical evaluation. Some examples include using macros instead of function calls, using variables for loop termination, using effective queries in databases etc.

Source code analysis - Energy Code Smells: The term "code smells"[5] was proposed by Fowler et al. back in 1997 [45] and is widely used in the software engineering communities. Eliminating "smells" through source-to-source refactorings (i.e. code reconstructions) contributes to the maintainability of the application. New kinds of "code smells", such as the energy smells have been recently proposed. By eliminating the energy smells, the application source code is claimed to be energy efficient [46]. Some examples of energy smells include *Loop bug* (repeating same activity usually waiting for an I/O operation), *in-line method* (method in-lining sometimes leads to energy savings), *immortal processes* (processes that keep restarting) etc. Other studies try to evaluate the effect of refactoring existing maintainability smells on energy consumption. For example, refactoring Feature Envy (methods placed in wrong class in an application developed in an object-oriented language) or Long Methods (methods that should be broken in more methods), God Class (a class that controls a large part of the application) [47] is expected to affect energy consumption.

Source code analysis - Estimation of energy consumption: The next category of research works includes the design of tools that aim at estimating energy consumption. These tools usually perform static analysis. For example [48] relies on machine learning methods based on measurement-based dataset targeting a specific microcontroller. There are also tools that make a coarse-grain estimation of expected consumption based on symbolic execution, such as SEEP [49].

System Calls and Data Structures: A large number of works refer to the impact of system calls on energy efficiency. For example, GreenAdvisor [20] tracks the number of system calls during application lifetime, comparing the number of system calls between different versions. Other tools [50] try to

[5]Energy Smell is a source code segment which needs to be restructured to improve the application's quality (e.g. maintainability).

estimate the energy consumption of system calls and use system call traces to estimate energy. Focusing on data structures is a traditional method of reducing energy consumption [51]. Relevant projects propose methods for selecting data structures that are more energy efficient for particular cases. These approaches rely on both general empirical and application specific studies [14].

Parallel computing: Thread management often has significant impact on the energy consumption of applications. A set of the configurations and parameters is selected manually, through experimentation or based on the designer's experience. A number of studies propose methods for energy efficient workload management, such as thread shuffling [52]. According to this method, slow threads are placed together in the same processor core and DVFS (dynamic voltage frequency scaling) is used for their fast execution. Another method is the work-stealing: the thread that completes its tasks, steals work from other threads [18].

Power Management, DVFS: Getting closer to the hardware perspective, a widely used technique is DVFS (dynamic voltage and frequency scaling): defining the clock frequency based on performance and power requirements. Determining voltage/frequency during heavy operations of an application may lead to trade-offs between power and performance and finding suitable settings that take into account energy is a challenging problem. Usually, there are specific sets of selections provided by the hardware (power modes) that can be used properly. For example Awan et al. proposes a method for selecting power mode based on offline analysis [53], while Bhatti et al. introduces an online analysis machine learning approach [54].

Profiling for Energy Modeling: A traditional approach of embedded system developers is to perform dynamic instrumentation manually on the hardware. They use the offered CPU performance counters [55], or popular tools like linux Perf [56] or Valgrind [57], in order to profile their applications and to find the behavior that lead to increased energy consumption as well as potential opportunities for improving. Approaches that try to automate and bring this procedure closer to the software perspective either use hardware-specific performance/energy models in order to link basic software constructs (source code or IR blocks) to energy [22] or perform dynamic instrumentation to collect features like cache misses, cycles etc to train machine learning models or to construct analytical modelling of the energy behaviour [58] [59].

Memory hierarchy, microarchitectural techniques etc.: Embedded systems developers with a hardware background, often use microarchitectural techniques to save energy in specific device components. By leveraging the device properties and according to the application's behavior, a dynamic reconfiguration of the individual components is carried out to optimise energy. A

large number of approaches can be found in the literature that usually rely on exhaustive design space exploration in different combinations of line sizes, associativity and cache-partitioning [60] [61] [62].

Custom DSPs, FPGA accelerators etc: From hardware point of view, optimizing energy consumption is usually associated with the use of custom SoCs, ASICs, DSPs or programmable FPGA and GPU accelerators. A large number of approaches propose application specific architectures and designs. Comparisons between alternative architectures are also presented. For example, using FPGAs may lead to better energy efficiency than GPUs, while GPUs have better performance for some applications [21] [63] [64] [65]. In this category, we can find studies that perform custom design approaches to further improve performance and energy efficiency, such as approximate computing [66].

2.1.1 What is missing?

All the aforementioned techniques are so different from each other and are characterized by custom inherent characteristics, usually perfectly related to the application under analysis or to the background and experience of the designer. From the above analysis it becomes clear that there is a need for tools capable of supporting a wide range of different techniques, assisting software development and providing a clear picture of energy improvement opportunities to the developer, as indicated by red lines in Figure 4. For example, software developers need to be aware about when to offload parts of an application to an accelerator, about the memory utilization, data races, etc. but this information should be offered to them in a way that is familiar to them, accompanied with a set of suggestions.

One could argue that as the years go by, the burden of efficient programming falls more on libraries and service providers. More and more tools and services are being introduced in order to properly use the hardware from the software level without the need for specialized knowledge. Typical examples can be found in recent years in the field of machine learning that is very popular, where libraries such as Tensorflow, Pytorch etc. offer the ability to use GPUs and TPUs easily from the Python code level. In addition, cloud service providers use their algorithms to select the best servers, VMs or physical machines to run client applications consuming less energy. Finally, high-level programming languages offer more libraries with data structures, monitoring functions, etc. However, software developers still need advice about how and when to use all these automated features, the choices and configurations they need to make, and the potential impact on the behavior of their applications. As a result, we may argue that such assisting tools become even

more necessary and useful.

The methods proposed in this dissertation, supported by tools, analyze applications in terms of energy consumption and recommend relevant optimizations, without the need of executing the code on the targeted devices (cross-device). This means that the proposed techniques guide energy optimizations at the software level (as part of an SDK tool). Special emphasis is given on estimating the potential energy gains by offloading parts of the source code to a GPU accelerator of a heterogeneous system. The introduced application analysis framework is designed to be able to integrate different kinds of tools that contribute to the identification of energy consumption issues and propose relevant optimizations.

2.2 Related tools

Designing a software development toolkit that targets energy consumption and optimization is a challenging task. As stated earlier, Pinto and Castor [15] highlight the lack of such tools and consider supporting developers on refactoring their code as a key concept (with emphasis on memory utilization and acceleration optimizations which we target in this thesis), as well as the challenge of designing simple static analysis estimators. The importance of such a tool is also emphasized in a recent survey by Georgiou et.al. [6]. The article concludes that software-based energy monitoring tools are of great importance as most tools are based on custom empirical studies or target a very specific type of application or hardware. Additionally, continuous energy consumption monitoring that provides feedback to the developer is needed for developing energy efficient applications.

The proposed solutions aim to offer energy consumption estimation and optimization, lowering the barrier of access of embedded systems energy optimizations for software engineers. On the contrary, State-of-the-Art approaches either target a very specific domain or are based on empirical results that require specific knowledge by developers. The proposed tools are also extensible in the sense that they can be used in a variety of different architectures.

2.2.1 Monitoring Tools

Tools that fall in this category monitor consumption, helping the designer to have a view of the application's energy. Monitoring tools are based on a variety of different approaches from estimation models and performance counters to direct measurements through hardware energy sensors. A very promising and widely-used solution is the *Running Average Power Limit*

(rapl) [67], which is a dynamic tool, supported from specific Intel architectures, which estimates power consumption from the CPU's performance counters. New tools based on Rapl have been introduced, such as *jRapl* for Java. A popular tool in the Java community is *Jalen*[6], which is a dynamic analysis tool that estimates energy by analyzing JVM. *PowerAPI* [68] performs energy profiling through dynamic instrumentation. There are tools specified to Virtual machines, that estimate energy by profiling CPU, RAM, HDD usage, like *Vmeter* and *BitWatts* [69]. The most mature and well-structured monitoring tools target Smartphones. A large amount of tools offer energy related metrics based on simulation or profiling. Some of them are the following: *Android Energy Profiler, GreenOracle, Petra, Anandroid, PowerBooter, GreenScaler, Trepn* [70]. Finally, tools targeting performance can be extended or considered very useful for improving energy. Such tools are the throughput estimators, like LLVM-mca[7] , the older Intel Architecture Code Analyser (IACA)[8] or Ithemal [71].

Tools aiming at estimating energy consumption of running the code on different devices (cross-device) have also been introduced in the literature. Initial approaches are based on simple regression algorithms [48], adopting measurement-based methods that achieve very accurate results. However, they only target specific microcontrollers. More recent approaches employ machine learning methods [58] [59] and after dividing the application into phases they use dynamic instrumentation features (such as cache misses, CPU cycles etc.) to train the models. The dynamic instrumentation, however, not only requires application execution, but also adds a large time overhead and needs a lot of manual actions by the developer (such as adding annotations in the source code) imposing a lot of problems for being a part of Software Development Tools. Another type of methods extend the Worst Case execution Time tools (WCT) [72] [73] towards energy estimation. These tools are usually criticised because they are very slow, their accuracy is very limited and they can support only specific architecture models.

2.2.2 Optimization Automation Tools

PEEK [74] aims to be used as a plugin on Eclipse or other IDEs and to make energy monitoring on function level, using git as a middleware. It supports a simple search on a set of alternative power mode configurations, compiler flags, libraries executing the app in the backend using a custom energy

[6] `https://gitlab.com/adelnoureddine/jalen`

[7] `https://llvm.org/docs/CommandGuide/llvm-mca.html`

[8] `https://software.intel.com/content/www/us/en/develop/articles/intel-architecture-code-analyzer-download.html.`

measurement device or SEEP profiler [49]. *SEEP* makes coarse-grain estimations of expected energy based on symbolic execution. *SEEDS* [75] targets also Java. This tool transforms the code and profile energy usage trying to make some optimizations in data structures automatically (e.g. substitutes HashSet for a TreeSet or LinkedList for ArrayList and vice versa). Other tools aim at a specific optimization type. For example *Green, Parrot, EnerJ* [76] target approximate computing, offering support to perform function memoization, loop termination, approximate arithmetic computations, but they still require specific knowledge from the user. Tools like *GreenAdvisor* and *Eprof* monitors system calls. Another category of tools used by practitioners to help them improve energy efficiency are the system emulators like Gem5. Finally, tools that target polyhedral optimization, such as MIRA [77] can also improve energy.

There are tools aiming to assist developers by predicting the GPU acceleration gains. These tools analyze CPU code to predict potential performance gains by GPU acceleration (speed-up). XAPP [78] and CGPredict [79] use dynamic instrumentation to analyze CPU code. They leverage either machine learning techniques for prediction, or they use analytical models. Other recent works that belong to the broad category of performance predictors are Compass and Automatch [80, 81]. Compass analyzes C code and generates application performance models, while Automatch detects application characteristics and predicts the performance of execution in various accelerators through the generation of analytical modeling. Most of the aforementioned predictors rely on dynamic instrumentation techniques. Although dynamic instrumentation is a widely-used analysis technique and is supported by mature tools (e.g. Valgrind [57], Pin [82]), it suffers from large execution time overhead and requires manual actions as mentioned before. Also although the existing approaches provide predictions in terms of execution time, they do not consider energy consumption predictions at all. Therefore, in the context of this dissertation, we extend existing approaches towards predicting energy gains by acceleration on heterogeneous embedded devices.

2.3 Optimizing Smart-Grid HVAC Control

The HVAC system's orchestration is a well-established challenge that has been extensively analyzed in literature. For the dynamic HVAC control there exist two major techniques: on-line decision-making and Model Predictive Control (MPC). Each one is characterized by a number of advantages and disadvantages.

On-line algorithms usually require lower design time, while MPC methods are usually more robust and efficient, especially in cases where the models

were designed along with the system under control. The usual "black-box" approach of machine learning on-line methods is sometimes criticized [83]. However, on-line methods are more reactive to real-time conditions, whereas the accuracy of MPC techniques is affected by the precision of weather forecasts and building dynamics models.

While MPC control techniques have been successfully applied in a wide range of non-linear applications [84, 85], including HVAC system control [83], extensive analysis of the system is required, fact that leads to high-dimensional mathematical problems [86]. A large amount of design and customization time for each individual building, through detailed experimental and mathematical analysis, is needed. Taking into account the tight time-to-market pressure, that imposes the need of rapid prototyping solutions, MPC solutions sometimes are difficult to be applied (especially on existing buildings). In general, MPC methods are suitable for controlling components of HVAC systems that have been modeled at their design time.

Simulation-based methods based on popular tools such as the EnergyPlus and Modelica facilitate building modeling but their use can only be part of a Building Management System, included mainly in large buildings (offices, hotels, public buildings) due to their increased computational requirements [28].

Fuzzy rules [87, 88] alleviate the necessity of a detailed mathematical model giving a predefined action plan that selects HVAC configuration based on the information received by the environment. In some cases genetic algorithms are employed to support the fuzzy controller [89, 90]. Supervised machine learning techniques, such as Artificial Neural Networks (ANNs) [91,92], are gaining a lot of attention, as that they do not require a detailed study of the underlying building dynamics. They are trained, using historical data and learn the the building's physics indirectly by estimating their impact. These techniques have the ability to enable model-free controllers. However, they have a number of limitations. Machine learning models usually need long time to be trained and calibrated and are difficult to implement in practice, while fuzzy rules create fuzzy classes of some parameters and as a result they are not able to learn building's behavior in detail, in order to react on real-time dynamics. Therefore a stage of "pre-training" is performed, based on historical data of a target building that they target to or by using building modeling tools (e.g. EnergyPlus, Modelica).

Reinforcement Learning (RL) aims at giving a solution by continuously learning through the results of different HVAC configurations. RL is gaining attention nowadays, while several HVAC control approaches use RL [93] [94] [95]. Commercial thermostats are also rumored to use similar techniques in order to learn occupant's preferences standing on their manual configura-

tion [93]. Usual criticism to this approach is the human factor of defining RL rewards, the instability and inefficiency at the initial system period, as well as prolonged learning periods [83].

Finally, regarding the optimization target, several works focus only on energy consumption. Some approaches try to satisfy a desired threshold set by users [96] and minimize energy cost taking into account the energy market. By using simulations [96] linear regression models that relate energy to temperature difference are created or a full model for estimating energy consumption is pre-assumed [97], based on modeled building parameters and a computationally demanding Monte Carlo approach. On the other hand, a number of related methods attempt to serve occupant's preferences according to their manual modifications on HVAC configuration. These solutions attempt to save energy by avoiding unnecessary re-adjustments, or by turning off the HVAC when while the comfort zone is defined manually by the occupants [98] [99].

2.3.1 What is missing?

The related work highlights the necessity of a plug&play solution that targets existing buildings easily. While there is a large number of relevant approaches, most of them are building-dependent and need a considerable amount of time for modeling the building under control or retrieving historical data to train machine learning models. In the context of this thesis, emphasis was given on designing decision-making techniques that support on-line learning by receiving information from the environment, striving for rapid prototyping.

The work presented in the context of this thesis envisions a controller that takes into account both energy and thermal comfort. More precisely, we build a multi-objective optimization problem (MOO), where energy and comfort give the two objectives. The optimization goal is expressed as a weighted sum of the single objectives, where by setting the weights the user can define the relative importance of optimizing either the energy, or the comfort objective.

Finally, the increased computational complexity of most of the existing techniques makes them affordable only to enterprise environments, e.g., as part of BEM systems. In this thesis, we give special emphasis on designing a solution that exhibits low computational requirements without sacrificing the quality of the delivered results, in order to facilitate its implementation as part of a low-cost embedded system. This necessity has already been identified by the industry, as it is portrayed by the expanding market of smart thermostats.

Chapter 3

3 Background (methods, tools and experimental setup)

This Chapter provides a brief overview of the methods, the tools and the infrastructure used in this study. It is considered useful for the reader in order to understand the rest of the dissertation.

For the reader's convenience, although this Chapter's sections correspond to different subjects, we decided to refer on which of the following information will be useful as a background for which of the next Chapters. Section 3.1 presents the dynamic intstrumentation tools that are extensively used in the work presented in Chapters 4 and 5. The machine learning and statistics techniques presented in Section 3.2 are present in all the chapters of this thesis, while Multi-objective optimization (Section 3.3) is used in Chapters 5 and 6. The first paragraphs of Section 3.4 describe the experimental setup (applications, devices and tools) used in Chapters 4 and 5, while Section 3.4.4 presents the simulation environment and the thermal comfort model used in Chapter 6.

3.1 Dynamic Instrumentation Tools

Dynamic Binary Instrumentation tools add monitoring functions and routines into the application during its runtime (i.e. in its binary file) in order to extract profiling information [100]. This technique allows events, statistics and error detection information to be gathered in fine granularity. On the other hand, it imposes significant overhead in the execution time of the application.

3.1.1 Valgrind Suite

Valgrind is one of the most popular open-source profiling suites. Valgrind suite includes a wide variety of dynamic instrumentation tools that provide a lot of detailed information about the execution of a program, the software and

hardware behaviour, the memory utilization and threading errors and bugs. It also enables the design of new tools that provide profiling information which are not available by the default tools of the suite. The design of new tools is based on Valgrind intermediate representation (IR) [14].

The input of Valgrind tools is the application executable. Valgrind generates a synthetic CPU in which the application is executed. During the runtime, Valgrind adds instrumentation code to the application binary in order to coordinate the execution and the corresponding Valgrind tool retrieves profiling results. Valgrind instrumentation code imposes overhead in the execution time of the application. Experiments show that profiling the application using Valgrind increases the execution time by 10-50 times.

In the following paragraphs we describe each tool of the Valgrind suite, along with the profiling information that it provides.

Memcheck identifies memory errors, including: Overrunning/underrunning heap blocks, Overrunning the top of the stack, Attempts to access freed memory, Attempts to use undefined values, Incorrect allocation/freeing of heap memory, Overlapping source and destination pointers in memcpy() and related functions and Memory leaks.

Cachegrind simulates the way that applications utilize cache memory hierarchies, as well as branch prediction mechanisms. By using Cachegrind, developers can evaluate various cache architectures and gather profiling results. Cachegrind provides profiling information for the cache architecture under evaluation including: Instruction cache reads/misses, Last-level cache instruction write/read misses, Data cache reads/misses, Data cache writes/write misses, Conditional braches executed/mispredicted and Indirect braches executed/mispredicted.

Callgrind monitors function calls during the application execution. The gathered call history is used to generate a call-graph. Profiling information includes the number of instructions, call/calle relationship, number of calls, for example: Call-graph (list of functions and number by which each function is called), Data read, cache misses in each function (gathered using cachegrind).

Although Valgrind is normally used based on command line input/output and text output files, the suite also offers a tool for graphical visualization of the output, called KCachegrind, which is a KDE/Qt based GUI.

Helgrind is used for finding synchronisation errors in applications that use POSIX threads. Helgrind can monitor and report: Mutexes errors/misuses, Inconsistent Lock Orderings (potential deadlocks), Data races, List of locks and their status, location, global variable and Access history recorded for a number of bytes starting at a specified address.

DRD tool detects possible errors with respect to thread synchronization

similar to Helgrind. However, it also monitors lock contention and can be used to identify thread starvation. DRD reports: Data races, Memory accesses that conflict with a past memory accesses and Lock Contention monitoring.

Massif tool profiles heap memory. The profiling information of Massif can be used to improve the utilization of heap memory by the application. An example of its output is shown in Figure 6. Massif profiling information includes the reporting of: Memory footpring, Useful-heap / extra-heap (malloc, calloc, new etc), Size of the stack(s) and Location of memory allocations and percentage of total allocation.

DHAT profiles heap memory and provides detailed information about allocated memory blocks (lifetime and utilisation of each one). DHAT profiling results include the following: Heap allocations: total number of bytes and blocks, Maximum live volume (i.e. maximum total size of blocks that are simultaneously live), Average block lifetime, Average number of reads and writes of each block ("access ratios") and Access frequency of each block.

SGcheck detects overflows of stack and global arrays.

3.1.2 Pin tools

Pin Tools are a different kind of dynamic analysis binary instrumentation tools. Pin offers a C-based development API to be used within the application source code. Developers insert the instrumentation functions in selected places of the application source code and select the metrics that will be monitored. The instrumentation functions are invoked at runtime and gather the selected profiling information [100]. An interesting feature offered by Pin tools is the fact that the instrumentation functions can be customized by developers, enabling, in this way, the monitoring of user-defined metrics. Pin works like a just-in-time (JIT) compiler and execute the application in a Virtual Machine. Pin tools fully support Intel-based architectures, while they provide limited ARM support[9]. Finally, they integrate several models, such as branch predictor, cache memory architectures and timing models. MICA[10] includes a set of metrics developed by Ghent University, which enables the collection of various profiling information [101]. It is built on top of Pin tools and the reported profiling information includes:

- Degree of Instruction-Level Parallelism (ILP)
- Types of Instructions, such as load/store, control, arithmetic (integer and floating point), string operations, bitwise operations, etc.

[9] `https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool`
[10] `https://github.com/boegel/MICA`

- Ratio of branch predictions
- Data stream strides
- Memory Footprint

3.2 Statistics and Machine Learning

3.2.1 Overview

Machine learning (ML) is a very popular sub-field of computer science, with a huge number of applications, which have gained much attention by academia and industry in recent decades. In this section we give a very brief description. The term "learning" refers on the ability of gaining information from the environment without programming and adding new elements and functions as well as the ability of adjustment and improvement of the system without re-programming.

According to Arthur Samuel (1959), machine learning gives computers the ability to learn without being explicitly programmed [102]. Machine learning models can learn a system's behaviour, improve, generalize and extract important features, using feedback from the environment. ML algorithms receive, as input, data retrieved from observations and produce models for making predictions on new data. Therefore, ML is closely related to computational statistics and mathematical optimization [103].

ML algorithms can be divided into the following categories:

- *Supervised learning*: This type of methods receives as input already known observations (input-output pairs) and then produce a general model that is able of finding the output of future inputs. In other words, we might say that the model tries to produce a function that describes the given data.
- *Unsupervised learning:* In this type of methods, there are no output labels on the observations. The algorithms are employed to find patterns and correlations in order to create a structure in the input data.
- *Reinforcement learning:* In this method, the model starts without any data and interacts dynamically with the environment giving reward to each action, in order to learn strategies. More details are given in Section 3.2.1.1.

Another ML categorization is based on the type of the model's output:

- *Classification:* The input data are assigned to two or more classes and the algorithm creates a model that assigns new input data into one of these classes (supervised learning).

- *Regression:* The output is a continuous real number. The algorithm produces a function that approaches the correlation between input and output (supervised learning).

- *Clustering:* The model divides data into groups according to correlations of the input features, as there are no output data (unsupervised learning). In contrast to the classification models the cluster that each observation belongs to is completely uknown and thus the model is responsible of grouping the data by identifying their similarities.

- *Dimensionality Reduction:* These algorithms reduce the number of input features by selecting those that explain the most information and removing those that do not contribute to the models prediction accuracy. Some algorithms of this category are also able of constructing new features from the existing ones.

3.2.1.1 Reinforcement Learning (RL) For a system that has unknown parameters, a deterministic approach for controlling and decision making is limited by approximations regarding the available system states that can be reached at run-time. Similarly, when the parameter space is vast, the definition of deterministic transitions from one state to another can be proven to be infeasible. Such design requirements, gave birth to Reinforcement learning (RL) machine learning approach, which constantly gaining attention. RL has been successful applied in applications including strategic games [104] [105] and supporting decisions for autonomous driving [106]. Tasks of greater difficulty have been tackled as well, in fields like neuroscience and psychology [107].

More precisely, an RL problem consists of a set S of states, a set A of actions, and a reward function $r : S \times A \to R$. At each instance of the problem, an action $a_i \in A$ (we assume that A is finite) has to be chosen, which will lead from $s_i \in S$ to a new state s_{i+1}. The tuple (s_i, a_i, s_{i+1}) is called a *transition*. A real value r_i is assigned to each of the transitions. The agent's goal is a series of transitions $t_1, t_2, \ldots, t_n$ that maximizes the R value (called *the return*) in Equation 1. Finally, the $\gamma \in [0, 1)$ is a discounting factor that controls the importance of future rewards and ensures convergence of the sum in Equation 1 when $n \to \infty$.

$$Maximize\ R = \sum_{i=0}^{n} \gamma^i r_i \tag{1}$$

Although state-values suffice to converge to an optimal solution, it is useful to define action-values. Given a state s and an action a, the action-value of the pair (s, a) is defined by Equation 2, where R now stands for the random return associated with first taking action a in state s, thereafter. Consequently, we might claim that estimating Q plays a key role on the overall performance as it quantifies the efficiency of the candidate CPS orchestrator's selections.

$$Q(s, a) = \mathbb{E}[R|(s, a)] \tag{2}$$

Next we summarize the RL Definitions and terminology:

- *Agent:* The mechanism that learns and acts, having as a goal to maximize the rewards given by the Environment.
- *State:* We call state a vector that describes the scenario/situation that the Agent encounters in the Environment. The Agent performs an action that drives the Environment to the next state and gives a reward. If a terminal state has been reached, and a new Episode begins. A terminal state gives as a reward zero.
- *Actions:* The Agent makes an Action, which influences the Environment and therefore changes the State. Every Action gives a Reward from the Environment to the Agent.
- *Reward:* A value that the Agent receives from the Environment as a response to the Action performed. The Agent's objective is to maximize the total Reward it receives during an Episode. An Episode ends if a terminal State has been approached.

3.2.2 Prediction Models

This paragraph gives a very brief overview of the specific ML regression models that can be found in the rest of the present thesis:

- *Linear regression* is the simplest prediction model used in this dissertation. It uses a linear approach to correlate the features with the target

values. The model training procedure sets the values of the coefficients to minimize the residual sum of squares between the targets of the training set and the targets as calculated by the linear approximation formula [108].

- *Lasso* stands for Least Absolute Shrinkage and Selection Operator. It is a more complex regression analysis method that performs not only coefficients regularization but also feature selection. This is achieved by the "shrinking" the coefficients towards zero to exclude certain features. This model works better on new data-sets [109] that are not pre-processed using another feature selection methods.

- *Random Forests* is a Supervised ML Algorithm capable of performing both classification and regression analysis. It is an ensemble method that is based on the simpler decision tree method. By building multiple decision trees using different samples of the training set, it returns the result of the majority in case of classification or the average value for regression [110].

- *Orthogonal Matching Pursuit* (extension of Matching Pursuit) is a sparse approximation algorithm that uses sparse solutions for systems of linear equations. It is an iterative algorithm that finds the best matching to approximately represent a signal from Hilbert space as a weighted sum of finitely many functions. It is a greedy algorithm as at each stage, the problem is solved based on current information only [111].

- *Neural Networks* is one of the most popular ML methods originally inspired by the human brain. They are systems with interconnected nodes that (by using algorithms) become able to identify hidden patterns and correlations, and to perform clustering, classification or regression analysis. A special type of neural networks is the Convolutional Neural Network (CNN), which is widely used in image processing, sound recognition, and sequence classification because of its inherent ability to identify new features using convolution filters [112].

3.2.3 Correlation

Correlation methods are used to express the degree that two variables are related. They are used to describe relationships and to show which characteristics (features) are the most important in terms of their relationship to the value needed to be predicted. Two correlation models are used in the context of the present study:

- *StepAIC* is an automated method that identifies an optimal set of features by step-wisely adding and removing features in each step and by using regression methods to evaluate the importance of each one. AIC stands for Akaike Information Criteria and estimates the prediction error and thus, the quality of each model [113]. Therefore stepAIC quantifies in each step the amount of information loss when a feature is removed. For a typical analysis, the null hypothesis is that removing a feature from the features vector does not affect the estimation accuracy. The hypothesis is rejected when p-value < 0.05 and not rejected when p-value > 0.05.

- *Spearman* correlation is a widely-used correlation method, which evaluates the monotonic relationship between two continuous variables: A coefficient is assigned to each feature, which varies from -1 to +1. -1 or +1 indicates an exact monotonic relationship. Positive correlations mean that as the feature value increases so does the the targeted value, while negative correlations mean that the increase of the feature indicates decrease in the targeted value. Zero implies no correlation [114].

3.3 Multiobjective optimization

In the concepts presented in this dissertation, we model the core functionality of our targets as a multi-objective optimization problem (MOO) [115], formally defined in Equation 3. Without loss of generality, we refer to a minimization problem, where $F_1, F_2, \ldots, F_n$ are the multiple objectives that have to be minimized simultaneously, while $G_j \leq 0$ denotes the constraints that have to be satisfied. Moreover, x is a vector of input variables, which represent the design parameters of the target system. Finally, especially for the CPS (IoT) case, that we examine in Chapter 6 of this dissertation, the objective functions may also be related to an external vector of (environmental) variables (s).

$$\begin{aligned} Minimize &: (F_1(x,s), F_2(x,s), \ldots, F_n(x,s)) \\ subject\ to &: G_j(x,s) \leq 0,\ j = 1, 2, \ldots, m \end{aligned} \tag{3}$$

3.3.1 Weighted-sum optimization

Equation 3 refers to the general case of multi-objective optimization problems. Since we aim to provide online solutions, the proposed solvers focus to a subset of these problems, where the cost function is expressed in the form of a weighted sum of single objectives, as it is described by Equation

4 [116]. Such an approach enables using the weights of the sum as a way to express how important each system's objective is. According to relevant weighted-sum optimization literature [116] there is a need for normalizing the objective functions in order to achieve *adherence to the preferences of the designer*, expressed through the values of the weights.

$$\begin{aligned} Minimize : Cost = \sum_{i=1}^{n} w_i F_i(x,s),\ \sum_{i=1}^{n} w_i = 1 \\ subject\ to : G_j(x,s) \leq 0,\ j = 1,2,\ldots,m \end{aligned} \tag{4}$$

For the problems described in this dissertation, the increased complexity of this formulation makes it prohibiting, or even infeasible, to derive an accurate analytical description of all the involved objective functions of Equation 4. Furthermore, the number of system parameters can be large and bound by unpredictability with respect to the actual value range of each parameter. Thus, the problem's definition of Equation 4 is described by the following properties:

1. the detailed form of the objective functions $(F_1, F_2, \ldots, F_n)$ is uknown;
2. the objective functions are not only related to the design variables but also to additional variables that retrieve values from the environment of the system
3. the effect of choosing a value for the design variables can be evaluated once per operation time-step;

The ultimate goal of the presented approaches is to provide the building blocks of a framework that solve the optimization problem characterized by the aforementioned properties in specific domains (e.g. Smart-grid presented in Chapter 6). According to the first property the problem *cannot be analytically defined at design time* and as a result a self-adaptive method is needed. Furthermore, the cost is affected from the system's environment and this *enlarges the problem's difficulty*. The third property represents the "penalty" of exploring the solution space as the only way to evaluate a solution is by applying it on the system and receiving the feedback. This means that a *fast solution* is needed because in most of the cases wrong configuration for some time-steps may have a huge impact in its operation.

3.3.2 Algorithms and Methods

Optimization methods are used in various stages of the present dissertation (e.g. Chapter 6). This type of methods belong to the Mathematical Programming field of study. The solution that these methods generate, is the input that minimizes (or maximizes) a targeted function and it is searched in a solution space that is characterized by specific available boundaries. The targeted function is called objective, cost, loss or utility function and the input that minimizes (or maximizes) its value is called an optimal solution. A fast introduction of some methods that will be mentioned in the rest of this dissertation is presented in the next paragraphs.

Dynamic Programming is a problem solving technique that is based on combining sub-problems solutions. Sub-problems depend on each other. Each sub-problem is solved once and the solution is saved in an array, in order to be easily available without needing resolving [117]. Dynamic programming methods start with constructing the problem's solution and finding the optimal solution recursively ("bottom-up").

Interior Point Methods were introduced around 1980 - 1990 and target non-linear problems. Although their results can be considered comparable to the simplex method, simplex is usually faster for small problems but extremely complex for large problems. Interior point algorithms offer solutions with polynomial complexity. While each interior-point iteration is computationally expensive it is progressive towards finding the optimal solution [118]. Interior-point algorithms approach the optimal solution but never actually reach it as they use barrier functions. Barrier functions' values tend to infinity as the point approaches the limits of the feasible region. For example the simplified optimization problem's (Equation 5) inequalities can be handled with a logarithmic barrier function that converts the problem in the way presented in Equation 6) [119], where $\mu > 0$ is a very small parameter.

$$min\ c^T\mathbf{x}\ \ , s.t\ \ A\mathbf{x} = b,\ \mathbf{x} \geq 0 \tag{5}$$

$$min\ c^T\mathbf{x} - \mu\sum_{j=1}^{n} log(x_j)\ \ , s.t\ \ A\mathbf{x} = b \tag{6}$$

Differential Evolution [120] [121] is a metaheuristic method used for function optimization. Due to the fact that it treats the targeted function as a black-box, it is extensively used for multi-objective optimization. Differential Evolution solves the optimization problem by iterative trials, creating new

candidate solutions and combining existing ones. Its stochastic nature enables searching over large areas of candidate solutions, but usually needs large numbers of evaluations. Also, Differential Evolution does not require the targeted function to be differentiable, as it does not rely on gradient methods. While originally Differential Evolution finds solution in the form of vectors of real numbers, recent studies propose its usage also for integer/discrete variables [122].

The Knapsack problem is a special problem of combinatorial optimization that aims at optimization under constraints and is very popular in computer science. It has been mentioned in the literature since 1897, while special emphasis has been given since the 50's due to its application in economics [123]. The definition of the problem is as follows: "*Given a set of items, where each item is associated with a value and a weight, choose which items to include in a collection (sack) so that the total weight is less than or equal to a given capacity and the total value is maximized*".

In the context of this dissertation we will refer to a special subset of the knapsack problems, namely the *Multiple-Choice Knapsack Problem (MCKP)*. Its definition [124] [125] is as follows: "*Given K classes of items $N_1, N_2, \ldots, N_K$, where each item $j \in N_i$ is associated with a profit/value v_{ij} and a weight w_{ij}, choose exactly one item from each class so that the total weight is less than, or equal, to a given capacity W and the total value is maximized*". The problem's mathematical formulation is given by Equation 7.

$$\begin{aligned} & max \sum_{i=1}^{K} \sum_{j \in N_i} v_{ij} x_{ij}, \\ subject\ to & \sum_{i=1}^{K} \sum_{j \in N_i} w_{ij} x_{ij} \leq W, \\ & \sum_{j \in N_i} x_{ij} = 1, i = 1 \ldots K, \\ & x_{ij} \in \{0, 1\}, i = 1 \ldots K,\ j \in N_i \end{aligned} \tag{7}$$

The MCKP algorithm is an NP-hard problem, leading to a high computational complexity. Solutions, found in the literature, optimize the problem by initially solving a simplified linear MCKP problem and then expanding the core solution based on dynamic programming and by adding the necessary classes [124]. In contrast to relevant solvers, such as dynamic programming [125] that exhibits pseudo-polynomial time complexity, the solution proposed by Pisinger [124] minimizes the problem's complexity by considering only a few items. More thoroughly, the computational complexity for

finding a solution is $O(n + W \times \sum_{N_i' \in c} n_i)$, where n is the number of total items, n_i the number of items in class N_i' and c is the core solution that contains only the classes and items that constitute the solution space retrieved from [124].

3.4 Experimental Setup

3.4.1 Applications

For evaluating the methods designed in the context of this dissertation, applications from popular benchmark suites that are widely-used by researchers are chosen. These benchmarks are considered reliable for demonstrating, testing and presenting research results.

- *Rodinia* [126] is a well-known benchmark suite that provides a set of applications for the study of heterogeneous systems. The programs coupled with sets of data are publicly available. Each application's inherent architectural characteristics effect parallelization, data transfers and communication in a different way. Finally, each Rodinia application includes CPU and GPU versions of the same applications.

- *Polybench* [127] is a collection of benchmarks that contains computational intensive kernels. It includes applications from different fields such as linear algebra, stencil computations, image processing and data mining. The latest versions of the suite provide applications both in CPU and GPU versions. These kernels can be part of a large number of both cloud (server) and embedded applications.

3.4.2 Targeted hardware

- *Nvidia Jetson TX1*[11] is a low-power embedded device that includes a Tegra System on Chip (SoC) which combines an ARM Cortex-A57 quad-code CPU and a 256-core NVIDIA Maxwell GPU. A high-level schematic diagram is shown in Figure 5. This embedded platform is used for embedded deep learning, computer vision and graphics applications, offering acceleration by exploiting the inherent heterogeneity of the SoC and the GPU capabilities. Some additional characteristics of the device are the 4GB 64-bit LPDDR4 Memory and the 16GB eMMC 5.1 internal storage. It consumes less than 10W total power and it includes Wi-Fi sensors and Gen 2 1x4 + 1x1 PCIe connection. We

[11] `https://developer.nvidia.com/embedded/jetson-tx1`

selected this device for our experiments due to the fact that it incorporates a widely-used ARM embedded processor, it includes a hardware energy sensor (INA 3221) to measure the actual energy on the chip and it is a typical haterogeneous SoC including an embedded GPU that is used for improving performance and energy consumption in IoT applications [128] [129].

- *Nvidia Jetson Nano* is another device that incorporates a version of the same SoC as Jetson TX1 (Tegra X1) with the same CPU ARM Cortex-A57 at a maximum frequency of 1.48GHz instead of 1.73GHz of TX1 and a smaller 128-core Maxwell GPU at 921MHz instead of TX1's 256-core GPU at 994MHz[12]

- *Nvidia Jetson Xavier NX*[13] is a high performance edge device that focuces on AI applications. It incorporates a different architecture (Nvidia Volta) including 384 CUDA cores and a 6-core NVIDIA Carmel ARMv8.2 CPU, with 2-level cache and 8 GB of 128bit RAM. In the context of this dissertation, this device is mainly used to evaluate the extensibility of the presented methods that were first designed based on the Jetson TX1. The increased capabilities of this device in both CPU and GPU compared to the Tegra X1 offer better CPU performance and much faster but more power consuming GPU usage.

- *Intel Xeon Gold 6138 - Nvidia Tesla V100 server* is used only to evaluate the potential extensibility of the presented methods that target embedded systems to partially support and assist developers in designing HPC (server/cloud/data center) applications. It is a high-end Supermicro server that incorporates a 20-core Intel Xeon Gold 6138 CPU (40 threads), 128GB RAM and an Nvidia Tesla V100 GPU. Intel Xeon Gold processor delivers improved performance, enhanced memory capabilities, advanced security technologies optimized for data centers, cloud computing, network and storage workloads. NVIDIA V100 Tensor Core [14] is one of the most advanced data center GPUs that targets AI, high performance computing (HPC), data science and graphics applications. It is based on NVIDIA Volta architecture and has a 32GB memory size.

[12] `https://docs.nvidia.com/jetson/l4t/` Clock Frequency and Power Management
[13] `https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit`
[14] `https://www.nvidia.com/en-us/data-center/v100/`

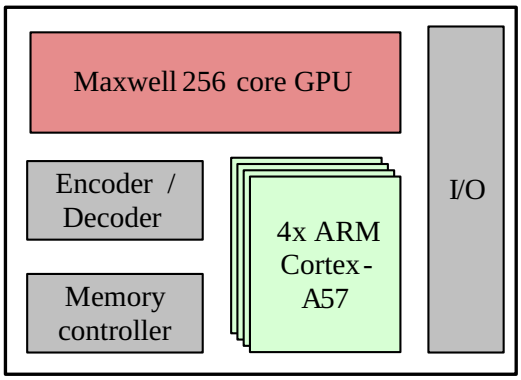


Figure 5: High-level schematic diagram of Tegra X1

3.4.3 Container orchestration - Kubernetes

Kubernetes[15] is an open-source platform for orchestrating containerized (e.g. Docker[16] containers) services on Cloud clusters. It has became the most popular solution for facilitating the configuration and automation of managing very large number of workloads (even in huge clusters) and it is supported by a large ecosystem.

Kubernetes introduces the pod concept, a group of one or more containers with shared storage and network. A Container Management Platform (CMP) offers the capabilities for building and managing a Kubernetes infrastructure, making a distributed and scalable system that contains (micro)services, supporting resilience and elasticity. In the Kubernetes environment, a cluster includes of a set of worker machines, namely *Nodes*, that run the containerized services. The components of the application workload are called *Pods* and are hosted by the Nodes. Nodes and Pods are controlled by the Control Plane. Kubernetes also offers fault-tolerance and high availability, by keeping replicas of pods and running the control plane on multiple machines.

While Kubernetes (or k8s) can be used for large configurations (up to 5000 nodes), it is not operable in Edge computing devices due to its' large computational requirements. For this type of applications, a lightweight distribution of Kubernetes, namely the k3s was developed by Rancher Labs.

Kubernetes was originally designed by Google, but now is maintained by the Cloud Native Computing Foundation. We will refer to Kubernetes in Section 5.4.

[15]https://kubernetes.io/
[16]https://www.docker.com/

3.4.4 Building Simulation

For simulating the building dynamics and the data of the micro-grid sensors, in the context of this thesis (Chapter 6), we use the well-known EnergyPlus suite [130]. EnergyPlus is a building simulation program that is used to study the energy consumption of building's HVAC systems, lights and appliances. It offers a complete solution that takes into account detailed models of the components of the building's infrastructure, as well as weather conditions and occupants loads (through schedules). It uses advanced air flow models to achieve high accuracy. EnergyPlus also offers interaction with the user between timesteps in order to test real-time systems as they were operating in the actual building.

For the communication and the update of the EnergyPlus schedules during the experiment, there are external tools that make data exchanges between EnergyPlus and MATLAB scripts, such as the BCVTB (Building Controls Virtual TestBed) [131].

3.4.4.1 Thermal Comfort The definition of thermal comfort as defined in the ANSI/ASHRAE Standard [132] is: "The state of the brain that expresses satisfaction with the thermal environment". This concept is extensively used in Chapter 6, where it is a design goal of the proposed system. Retaining its value, within some satisfactory limits (according to the standards) is a goal of HVAC systems (heating, air conditioning, ventilation). In addition to ANSI/ASHRAE, there are other standards such as EN 15251 and ISO 7730.

Thermal comfort is calculated based on factors that affect the uptake and loss of heat by the human skin [133]. The most important factors are the following:

- Air temperature
- Mean Mean radiant temperature
- Air speed
- Relative humidity
- Metabolic rate (Tables with various activities along with approximated values of metabolic rates are offered for helping the calculation [132] - ISO 8996)
- Clothing insulation (It is calculated through detailed tables that include clothes along with the insulation factors [134].)

- Psychological factors may also be involved

Predicted Mean Vote (PMV) is one of the most popular thermal comfort models and it is introduced by Fanger [1]. It is based on experiments performed in a room by controlling the environment conditions. The people who participated in the experiments had to rank their feel on a scale from -3 (very cold) up to +3 (very hot). The Predicted Mean Vote (PMV) (as calculated by Fanger's equations) can range from -0.5 to 0.5 according to ASHRAE standard [132] in order to have acceptable thermal conditions in a building. Zero is the ideal value. Another way to express the same value is by using the predicted Percentage of Dissatisfied (PPD), which is a number that corresponds to how many people (out of 100) would be dissatisfied by the thermal conditions in the room.

Chapter 4

4 Cross-device Energy Estimation

Modern low-energy IoT devices should be driven by energy-aware software. The ever-increasing attention to IoT applications pose new challenges to software developers as they now target devices where energy is a critical design constraint, with great social and economic impact. A promising solution to support developers in this direction is provided by energy estimation tools. The design of such tools, for bringing energy efficiency closer to the software engineering perspective during all the phases of the Software Development Life Cycle, is now an active research topic [6]. This Chapter presents a methodology for designing cross-device energy consumption estimation tools.

The introduced energy estimation tools rely on two types of analysis: Dynamic or Static analysis of the source code in order to get the advantages of both:

- Dynamic analysis (instrumentation) is extensively used by embedded system developers. It usually leads to accurate results through detailed profiling of the application's execution and it is employed by the most of existing approaches. However, this kind of solutions not only requires applications to be executed but also adds a large time overhead. In addition, manual actions are usually needed by the developers (e.g. adding special comments or compiler annotations).

- Static analysis is more difficult to develop and thus it is not so widely used. The accuracy is expected to be rather limited, however, as it provides energy estimations by using the source code as the only input, it alleviates the need for real hardware devices and it doesn't require the developer to have specific knowledge about how to measure energy consumption. Therefore, we might claim that static analysis offers a more user-friendly solution that can be part of a software analysis toolbox. Such a solution introduces similar approaches to tools used

in software engineering community that target other quality attributes (e.g. maintainability).

4.1 Problem definition

It is considered very important to state and define the specific problem that the methodology presented in the next paragraphs aims to solve: The energy estimation tool runs on the programmer's workstation (PC), as part of the SDK tool used by the software developer. When the final application or part of it is ready, the developer uses the energy estimation tool to get an idea of the energy consumption that the application's code will consume if executed on a list of embedded devices (cross-device). This idea is presented schematically in Figure 6.

Although measuring energy directly on the targeted device would give the most accurate results, not all hardware alternatives are accessible to developers and such a process may involve sophisticated equipment (e.g. special sensors) or expertise. These extra efforts would not only increase development time and cost, but may also not be feasible in very complex applications that involve a large number of different devices.

4.2 Related work

A large variety of approaches for solving the aforementioned problem has been introduced in the literature. Initial approaches towards performance and/or energy predictions at instruction-level, include works based on simple regression algorithms [48]. Work in [48] adopts a measurement-based method

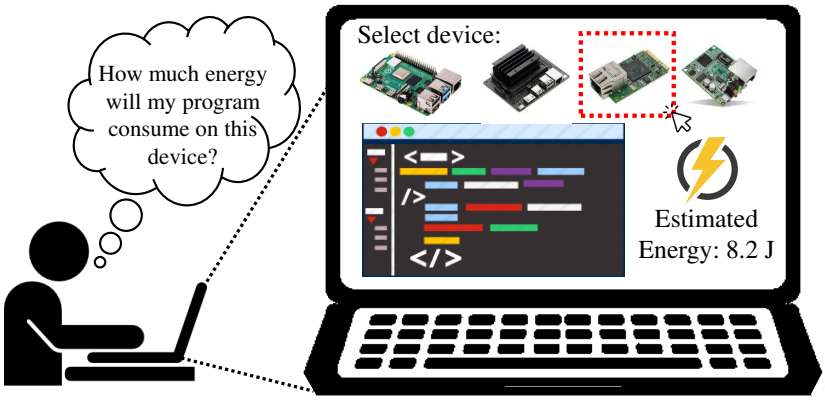


Figure 6: Cross-device energy estimation tools as part of software development

and achieves very accurate results. However, it targets only a specific microcontroller and it is based on its specific Instruction Set. Furthermore, this approach uses a very limited dataset (60 programs).

More recent approaches that achieve increased accuracy, employ machine learning methods for predicting the performance and power consumption of running the code on different devices (cross-device) [58] [59]. These methods divide the application into phases and use dynamic instrumentation to retrieve the features that feed the introduced estimation models such as cache misses, CPU cycles etc. The dynamic part of the method presented in the context of this dissertation is inspired by these methods. We focus mainly on loops (the usual most energy-intensive phases). In addition, in this study we provide features-energy correlation results and we select a small subset of the features provided, to make simpler models and reduce the profiling phase time overhead. Also, we focus on designing a method for adding new targeted platforms easily. However, the dynamic instrumentation used in this type of approaches, not only requires application execution, but also adds a large time overhead. Therefore, despite its very accurate estimations, it imposes a lot of problems for being a part of a Software Development Toolbox. Furthermore, a lot of manual actions are needed by the developer (such as adding annotations in the source code). These problems inspired us to also investigate the alternative solution of using simple static analysis features.

Mira tool [77] leverages the capabilities of the ROSE compiler and estimates the numbed of floating-point operations per block, based on a user input (i.e. the number of loop iterations) and an analytical hardware architecture description file. Although Mira does not estimate energy consumption directly, it is worth including it in the related studies as it gave us a lot of ideas for designing the static component of the presented methodology. For example, the analysis of the Abstract Syntax Tree and the special focus on the loops combined with the requirement of some dynamic information by the user (such as the number of loop iterations) are inspired by Mira.

Another type of estimation methods extend the Worst Case execution Time tools (WCT) [72] [73] towards energy estimation. These methods aim to estimate the applications worst case performance based on an iterative approach without receiving any information about the input. These tools are usually criticised because they are estremely slow, their accuracy is very limited and they can support only specific architecture models. Some approaches that aim to provide improvements are also proposed [135].

Another type of tools that predict the throughput (or the number of cycles), include Ithemal [71], LLVM-mca[17] and Intel Architecture Code Anal-

[17]https://llvm.org/docs/CommandGuide/llvm-mca.html

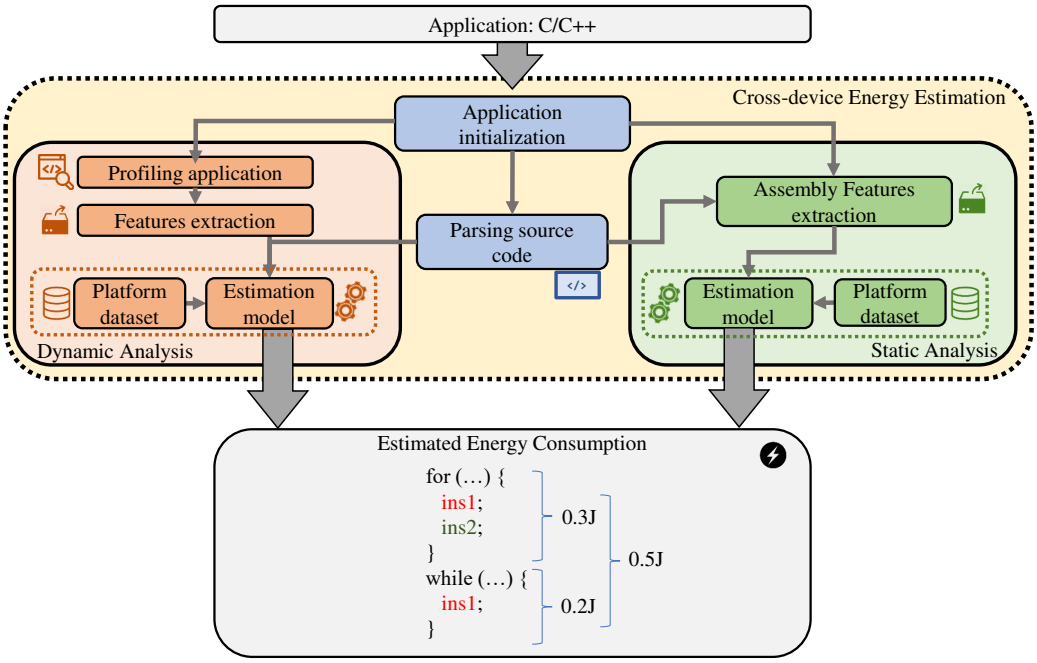


Figure 7: Overview of the studied energy estimation method

yser (IACA)[18]. Ithemal tool is based on machine learning techniques, while LLVM-mca and Intel IACA use architecture models. These tools are very useful as they give as an output the estimated throughput of executing a basic block of source code on a specific architecture. In the static part of the methodology presented in this dissertation, the information retrieved by these tools is also combined with additional features to construct an input for the designed energy consumption estimation models.

4.3 Design approach

Figure 7 depicts the estimation tool that is ready to be used by the final user. Static and dynamic analysis tools work in a similar way. The two key components (included in the two boxes with the dashed line) are an *Estimation model* that receives *Platform dataset* as input. More specifically, the estimation model refers to the final model that estimates energy consumption, while the platform dataset is a set of data retrieved from the targeted platform for which we want to estimate the applications energy. The difference between the two approaches lies in the way they analyze applications

[18]https://software.intel.com/content/www/us/en/develop/articles/intel-architecture-code-analyzer-download.html

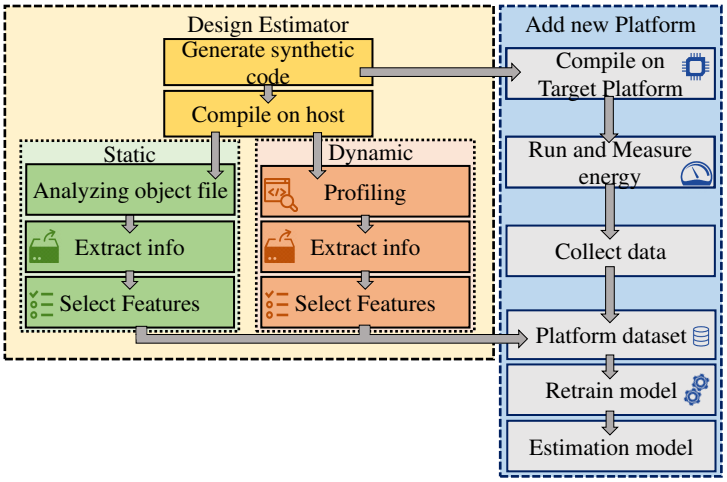


Figure 8: Create platform dataset and estimation models

to obtain the features that feed the estimation models. To illustrate these differences, Figure 8 shows the detailed procedure for creating the datasets and the energy estimation models, detailing both the Design Estimator phase and the addition of a new platform.

As shown on Figure 8, the idea of designing our practical and flexible estimators is based on two phases: The first one is the back-end phase, called *the Design Estimator phase* in the context of this dissertation, which contains the central mechanism for designing energy estimators. The second (*Add new Platform*) gives the process for adding new target hardware platforms (devices) to be supported by the introduced tool, contributing to the desired extensibility of the presented method.

4.4 Design Estimator

The first component is responsible for generating synthetic code: the data-set used for training the estimation models. It is very important to describe the way the dataset is structured, as it is crucial for supporting the energy estimating methods.

The dataset used in the context of this work consists of random synthetic C/C++ loops. The loops are generated by a set of python scripts and include operations between random-sized vectors and matrices [136]. More specifically, the generator creates a random number of integer arrays and a random number of floating-point arrays, where the sizes and the number of dimensions are also random. The limits can be selected by the user. For example, for the experiments performed in this work we selected a maximum num-

Table 1: Generated Loop example

```
for(i=0; i<18; i++){
    for(j=0; j<250; j++){
        for(k=0; k<167; k++){
            A0[i][j][k] = A1[i][j][k]*A1[i][j][k]/A1[i][j][k];
            B0[i][j][k] = B1[i][j]-B1[i][j]/B1[i][j]*B1[i][j];
        }
    }
}
```

ber of 10 arrays, with maximum size of 1000000 elements and a maximum dimension of 4. Then, the generator creates a program that initialize the arrays with random numbers and makes a for loop code segment where the profiling and the energy measurements are performed. In this loop random operations (additions, subtractions, multiplications or divisions) between array elements are selected. A representative example of a generated loop is given in Table 1.

In order to keep the most representative data-points, we employ an additional k-means clustering pre-processing. By selecting one data-point from each cluster, we avoid keeping very similar data-points which can cause model over-fitting.

The next step performs the compilation of the random codes on the host machine (the workstation that the estimation tool is running on). The synthetic codes are compiled using the same optimization flags (O0): Due to the inherent random characteristics of the loops, the loop bodies usually involve calculations that are not affected by the loop iterations. As a result, choosing other optimization flags would place the body's functionality out of the loop to avoid the meaningless overhead, fact that is not desired in our case.

4.4.1 Dynamic analysis approach

The dynamic approach analyzes applications through dynamic instrumentation (profiling). This step uses the popular Valgrind [57] and Pin [100] tools and the profiling process runs on the host (developer's workstation) to generate the information to be used by the estimation models.

Cachegrind tool (from the Valgrind's suite) simulates the cache memory architecture. It gathers information about *Instruction and Data cache reads, writes and misses* and it monitors *branching* behavior. Valgrind's Massif tool measures *heap and stack memory usage* (see Section 3).

Pin tools, as mentioned in Section 3, is a flexible tool that offers the ca-

pability of designing custom monitoring metrics through C++. It can be considered as a just in time compiler that performs the manual defined calculations each time the next assembly instruction is fetched. Our custom designed measurements capture *branching divergence* information, the number of *single-point and double-point operations* and the *types of arithmetic operations (additions, subtractions, multiplications, divisions)*. Moreover, we used Pin metrics designed by MICA[19] [101]. The provided metrics include *instruction-level parallelism, instruction types* (categories include *memory reads, memory writes, control flow, arithmetic operations, etc.), instruction and data memory footprint, memory reuse distances, conditional branching predictability* and *memory stride* (distances between subsequent memory accesses). Table 2 shows the full list of the gathered metrics.

The information provided by the *Profiling* step is analyzed in the next components in order to extract useful information for feeding the energy estimation models. These steps are considered very important for identifying relation between the metrics and energy. For this purpose, the component uses correlation and cross-validation techniques to produce the final features. These features are forwarded to the next component, which is responsible for comparing and selecting the most accurate models. The relevant analysis is presented in detail in Section 4.7.

4.4.2 Static analysis approach

The alternative static analysis approach comes as a mitigation of the constraints of dynamic analysis. Dynamic analysis, as already mentioned, adds a large time overhead and requires the execution of the programs under analysis. The static analysis mechanism aims to make the estimation framework more user-friendly and easier to use.

The *Design Estimator* (shown in Figure 8 green boxes) first analyzes the object file created by the compiler and identifies blocks of code. Due to the fact that only the source code is analysed (without dynamic information), static analysis focuses on code blocks: Blocks of code executed as they are (e.g. loop bodies, function bodies, etc.) and do not contain iterations, branches or calls.

The code block identification is based on the analysis of the Abstract Syntax tree (AST) produced by the compiler front end (in this work, CLANG was used) and splits the given application source code into blocks. The object file (generated by the compiler) is analysed for extracting information to be used as input to the estimation model. It is worth mentioning that, especially

[19] `https://github.com/boegel/MICA`

Table 2: List of examined dynamic analysis features

Dynamic Analysis Feature
Instructions
Instruction reads
Instruction cache Level 1 miss rate
Instruction cache Last Level miss rate
Data reads
Data cache Level 1 miss rate
D cache Last Level miss rates
Data writes
Data cache Level 1 write misses
Data cache Last Level write misses
Conditional Branches
Branch prediction misses
Indirect Branches executed
Indirect Branches misspredicted
Stack bytes used
Heap bytes allocated as "padding"
Heap bytes used
Execution time
Instruction Level Parallelism rate
Parallel instructions for different instruction window sizes
Branch predictability rate
Register traffic
Average number of register operands
Average degree of register use
Local memory stride
Global memory stride
Total memory stride
Total memory accesses with stride 0 (local memory)
Total memory accesses with stride 0 (global memory)
Control flow instructions
Arithmetic operations
Integer operations
Floating-point operations
Shift operations
Single precision Floating-point operations
Double precision Floating-point operations
Memory footprint (blocks and pages) different sizes
Memory reuse distances (count for different distance sizes)
Division operations
Control operations
Total memory operations
Branch divergence rate
Branch divergences (count in branch window sizes)

Table 3: List of examined static analysis features

Static Analysis Feature
Estimated throughput (by LLVM-mca)
Number of instructions
Number of LOAD instructions
Number of STORE instructions
Number of OP (operations) instructions
Number of class 1 instructions (add, sub, shift, mul)
Number of class 2 instructions (conv, arrays, div)
OP, LOAD and STORE instructions order

regarding the loops, dynamic information (e.g. the number of iterations) is also considered necessary. The presented framework supports gathering this information as an input from the user in order to combine the individual code blocks estimations in a proper way that corresponds to the actual execution of the total application.

As we aim to provide a cross-device and cross-architecture solution, the presented solution uses general features that model and characterize the application's behavior and computational requirements that directly affect the performance and energy consumption. The selected features are based on the application's assembly code as well as the output of the LLVM-mca[20] tool analysis.

Assembly instructions are categorized, building a small number of generalized features. Instructions belonging to the same category have similar execution time. This choice supports the applicability of the estimation method to different architectures and instruction sets. The selected features are presented in Table 3.

Section 4.7 describes in detail the methodology followed for selecting these features, as well as their importance.

Not only the kind but also the order of the assembly instructions has a large impact on the performance and energy. In order to model the instructions order and build simple and generic features capable of being used to estimate the performance and energy for various platforms and architectures, we adopted a sliding window approach. More precisely, a window runs through the assembly instructions of the block under analysis and each combination of instructions types corresponds to a new feature, as it is depicted schematically in the example presented in Figure 9 [137]. Beyond the type of instructions, further information, such as the registers being used and the location of the accessed data affects the consumption. However, we keep the

[20] `https://llvm.org/docs/CommandGuide/llvm-mca.html`

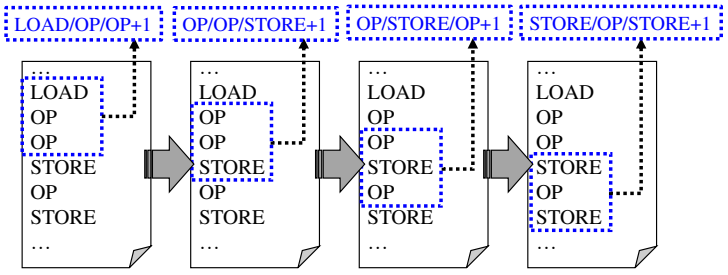


Figure 9: The sliding window method for modeling the assembly instructions order

features simple and relatively few to support adding new devices and retraining models easily, resulting to a generic solution that offers flexibility perhaps with a small effect on the quality of the results. The basic instruction categories are 3 (LOAD, STORE and OP), as the rest of them are subcategories of the OP instructions. Our final design choice is a 3-instructions-size sliding window leading up to $3^3 = 27$ new features.

Similarly to the aforementioned dynamic analysis based procedure, the selected features are forwarded to the next component for comparing and selecting the best models that estimate energy. The analysis of the features importance is presented in detail in Section 4.7.

4.5 Add new platform

One of the main goals of the designed energy consumption estimators is to be extensible: New devices (hardware platforms) can be added by the user easily, following three steps:

- Run the synthetic codes on the new device in combination with a call to the energy metering script. The user adds commands to get energy information from external monitors, specific paths in the device tree or, if no sensors are available, user-defined metrics (e.g. power-delay product).

- The energy measurements received from the previous step are processed and form an extra dataset file (csv).

- The estimation model is retrained using the new data as energy values (y values), maintaining the same feature values (x values) retrieved from the already analysed applications of the dataset (statically or dynamically).

A presentation of the extension to support an additional platform based on these steps is provided in Section 4.8.

4.6 Energy Estimation model

The heart of energy estimation is the model, while nothing is possible without the right features. Before proceeding to the very important analysis of our feature selection method and the feature-significance study, we must refer to the models used for the energy estimation mechanism described in this thesis.

The procedure is rather straightforward: We compare the accuracy of different models expressed as the Mean Absolute or Square Error between the actual energy value corresponding to a subset of the synthetic code blocks of the data set running on the targeted device and the predicted values produced by each alternative model. The device we chose for the construction of the models is the Nvidia Tegra TX1 platform, which incorporates an integrated ARM-Cortex A57 processor as as well as a built-in energy sensor (INA 3221) (see Section 3).

4.6.1 Dynamic analysis

The features gathered by dynamic instrumentation analysis, described in Section 4.4.1 are more than 100 and related to CPU and memory behavior, as well as operations types, branches etc. The full list of the examined features is presented in Table 2 (please note that some lines of Table 2 include multiple features).

As mentioned before, all the tests are based on our generated dataset, splitted into training and test sets, for performing cross-validation. In the next step, we gave the complete list of features, which were collected by analyzing the synthetic data set, as an input to the estimation models. Figure 10 presents the results of the top six models. We might observe that the Lasso model is far superior to the competing models. However, this is due to its inherent feature selection capabilities, which gives a comparative advantage when we have a very large number of features, where many are proportional and express similar features. More specifically, the Mean Absolute Error appears to be 85 times smaller compared to using the Linear regressor and 2.2 times compared to the results of using the Random Forest regressor. It is worth mentioning that the error refers to the execution of the entire loop (including the total number of iterations), as the dynamic analysis process aims to estimate the total execution energy.

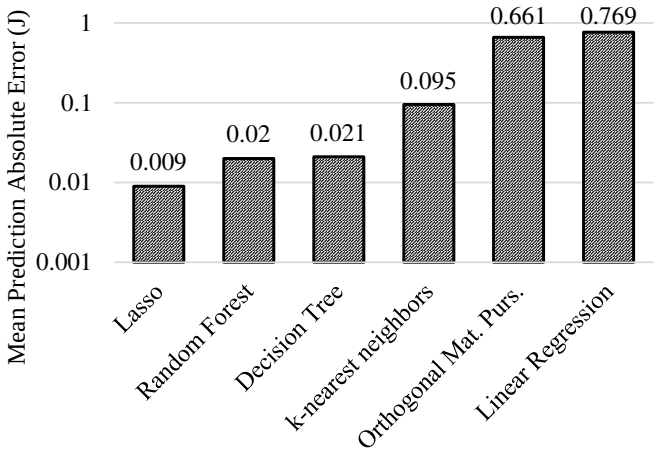


Figure 10: Alternative dynamic analysis based energy estimation models comparison

4.6.2 Static analysis

Similarly to the case of dynamic analysis, Figure 11 shows the accuracy of the most suitable models for static analysis. For each model, the Mean Absolute Error between the actual values (that correspond to the basic blocks) and the predicted values is presented. Based on these results, we can conclude that the Orthogonal Matching Pursuit model makes better predictions. It is worth noting that the error refers to the execution of just one loop iteration, as the core static analysis process aims to estimate the basic block execution energy, without the need of dynamic information such as the number of iterations.

As mentioned in Section 4.4, a k-means clustering pre-processing process is used in order to avoid potential over-fitting problems due to to the great similarities between data points in the synthetic (generated) data set. As a result, the number of selected clusters, corresponding to the final number of data points in the data set, is also a design option to explore in this analysis.

In this direction, Figure 12 shows the mean absolute prediction error, for estimating the energy of a synthetic loop iteration belonging to the testset, using the above methods. According to these results, we might conclude that selecting 200 clusters, that correspond to the most representative 200 generated blocks of code, reduces the final error.

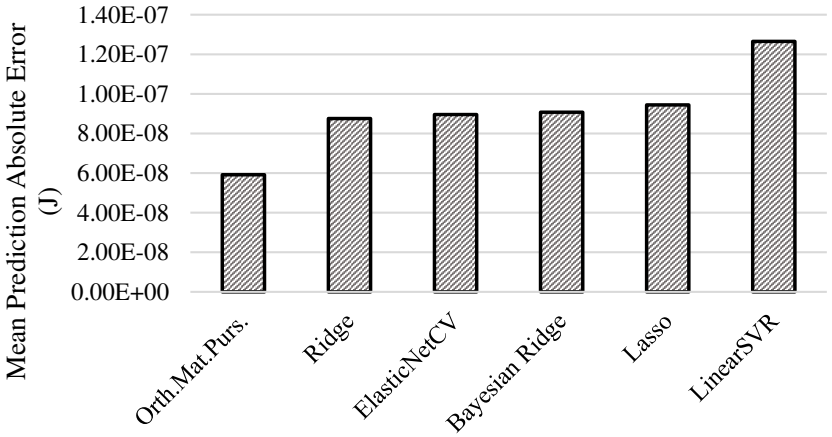


Figure 11: Alternative static analysis based energy estimation models comparison

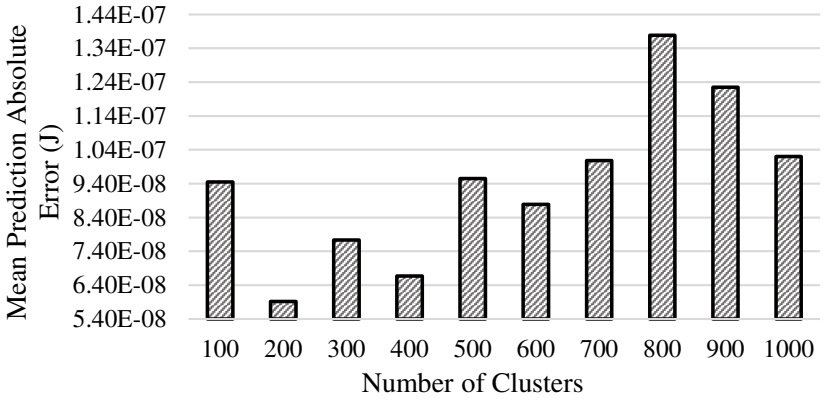


Figure 12: Impact of number of k-means dataset pre-process

4.7 Feature analysis: Selection and Correlation study

A key concept of the work presented in this Chapter is to study the correlation between alternative features (collected in the developer's workstation - host machine) and the energy consumption (measured on the targeted device).

4.7.1 Dynamic analysis features

For this study, we employed widely-used correlation methods. The list of the examined features is presented in Table 2. A first analysis is based on the weights calculated by the Lasso Regressor (see Section 4.6.1). The inherent feature selection mechanism of the Lasso Regressor relies on shrinking

Table 4: Most important dynamic features

Feature	Lasso importance	Feature	Spearman Correlation
Instruction level parallelism	2.481	# Instruction Reads	0.979
# Data writes	0.688	# Instructions	0.979
# Memory blocks/pages	0.438	# Data reads	0.978
Heap memory size	0.379	# Data LL misses	0.977
# Arithmetic operations	0.324	# Data writes	0.977
Memory stride	0.240	# Data L1 misses	0.975
# Conditional branches	0.187	Data L1 miss rate	0.975
# Branch prediction misses	0.185	Instruction types	0.974
Data LL miss rate	0.119	Instruction level parallelism	0.974
Stack usage	0.099	Memory stride	0.974
Data L1 miss rate	0.053	Memory reuse distances	0.974
# Memory reads	0.014	Register traffic	0.974

the weights of non-used features to zero, while assigning a non-zero weight value to the rest of the features. Larger weights are given to the most important features. An alternative method is the Spearman correlation analysis. Spearman correlation evaluates the monotonic relationship between two continuous variables. More specifically, a coefficient is assigned to each feature, which varies from -1 to +1. The correlations of -1 or +1 indicate an exact monotonic relationship. Positive correlations mean that as the feature value increases, so does the energy consumption, while negative correlations mean that as the value of the feature increases, the energy decreases. Zero values imply no correlation. The most important characteristics according to this analysis are presented in Table 4 alongside with the values of their coefficients [136].

According to the correlation analysis we came to the expected following conclusions:

- *Instruction Level Parallelism* indicates how many instructions can be processed in parallel affecting the performance and respectively the energy consumption.

- Each *memory access* (especially *writes*) imposes a cost in terms of energy consumption. The amount of cost is affected by various parameters, such as the layer of the memory hierarchy in which data are accessed. When a *cache miss* occurs, the CPU fetches requested data from the main memory. The cache miss rate describes how effectively the application uses cache memories. Apparently, cache misses impose energy consumption overhead. The memory reuse distances (*stride*) heavily affect the potential cache performance.

- Modern CPUs use *branch prediction* mechanisms to guess the branch that will be selected and fetch the corresponding instructions in the

CPU pipeline. When branch prediction fails, the CPU pipeline is flushed, which has a negative impact in application's energy consumption.

- Arithmetic operations are managed by the ALU and often need a lot of energy, especially in case of division. As a result, this feature can be used as an indicator of the energy that a piece of code consumes.

One might claim that the conclusions seem rather obvious. However, the total metrics we collected from the profiling tools gave us a list of more than 100 different metrics possibly related to energy consumption, as mentioned in Section 4.6.1. According to the correlation results, instruction miss rate, some types of operations, branch divergence rates, etc. are not included in the most important features. For example, shift operations seem to have a similar overhead as addition operations. Similarly, multiplications, although they have a higher overhead as they involve multiple additions, seem to affect the energy less than the metrics presented in Table 4 and are included coupled with the rest of arithmetic operations in a single feature metric. The selection of fewer metrics serves our purpose of building a method for designing simple estimators, as it reduces the time overhead costs of the profiling phase (making the integration in SDK tools easier) and produces an hierarchical analysis of the alternative features.

4.7.2 Static analysis features

The static analysis features are presented in Section 4.6.2, Table 3. In this section, we will analyse the selection and the importance of each of the aforementioned features.

As mentioned in Section 4.6.2 the execution time of instructions also inspired our choices to include them or not in the feature list and to group them as similar features. Figure 13 shows the average execution time of each assembly instruction, on Nvidia Jetson TX1 (ARM-Cortex A-57). Exactly the same behavior were observed in other devices too (such as Raspberry Pi 4 - ARM-Cortex A-72).

Load and *Store* operations highly affect the program performance and energy consumption because they require access to the system's memory. The average execution time of these operations as measured on the ARM Cortex-A57-based device (Nvidia Tegra TX1) is from 2.2×10^{-7} up to 4.4×10^{-7} ms for *Store* and from 1.2×10^{-7} up to 3.2×10^{-7} ms for *Load* operations. The observed time variations depend on the type and the size of data, as well as the cache memory behavior (also analysed in the beginning of this Section).

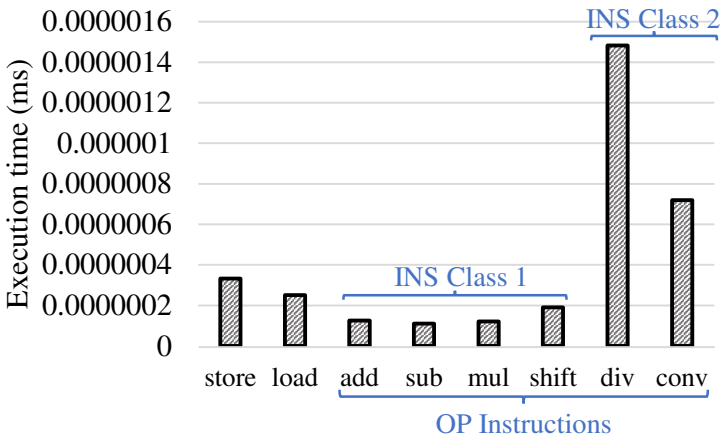


Figure 13: Average execution time of each instruction category feature

The operation type instructions (OP) are categorized into two more categories. The first one contains the *add, sub, shift*, and *mul* instructions. These instructions are the majority of the arithmetic operations performed in the applications included in our dataset and analyzed in the context of this dissertation. *Shift* and addition operations seem to have similar time overhead. Also, *mul* operations seem to have similar or sometimes lager overhead, fact that makes perfect sense considering that they include multiple additions. For example, add, sub, and shift operation's time does not exceed 2.2×10^{-7} ms, while multiplications need up to 2.4×10^{-7} ms to be executed. However, this additional overhead is not considered significantly large leading us to include them also in the same category. One of the most usual operations are between arrays and matrices. Arrays processing is very important for the overall performance and energy consumption of the application and is accompanied by memory functions. When an array element if accessed for the first time, a cache miss occurs and a transaction of data from the main memory is required. If the specific block is already cached, the energy consumption and time overhead is relatively smaller. For the purpose of the presented static analysis method, we assume that the cache memory is ideal and we do not take into account any hardware information about the cache architecture, as we rely on the source code only. The impact of array operations is measured indirectly by including all the instructions that perform array operations into a single category. Usually, especially in 64-bit systems, when an array operation is performed, the compiler also produces conversion instructions (conv) (such as cltq, cdqe, and movslq in x86 assembly) to convert the index value from a 32 to 64 bits for the pointer to access the 64-bit memory addresses. So, by monitoring this type of instructions, we indirectly monitor the array

operations. *Division* operations use pipeline and include multiple stages and this increases time and energy. The array operations and the conversion execution time in ARM-Cortex A-57 varies from 2×10^{-7} ms up to 1.3×10^{-6} ms, while division operations sometimes need up to $1.6 \times 10^{-}6$ ms to be executed.

Jump, *comparison* and *move* instructions are included only in the OP instruction category. *Jump* and *comparison* operations are not present in the basic blocks analyzed by the static analysis part of the tool (loop bodies and function bodies) because they change the program flow. As stated in Section 4.4.2, due to the fact that this component is based on static analysis (similarly to the relevant literature, e.g., [71]), we focus only on basic blocks. The estimation of the total application's energy is calculated based on a combination of these blocks. Therefore, these types of operations are just treated as part of the operations category. Finally, *move* instructions are used both in memory operations and in arithmetic operations (data exchanges between registers) and thus, further classification is not needed.

The importance of the selected features is highlighted in Figure 14. More precisely, we re-build the estimation model using different sets of features in the training set and we evaluate the efficiency of the tool on estimating energy consumption of a random testset on Nvidia Tegra TX1 (ARM-Cortex A57). The reduction of the average absolute error indicates the importance of each new feature added. In the first case, only the number of instructions is used as a feature. Afterwards, we added the estimated throughput (by LLVM-mca), as well as the OP feature, corresponding to arithmetic operations (the rest of the instructions are memory loads and stores). Then, we made the features more special (LOADS, STORES, OP classes), and then, the order of the instructions is added in the way described in Section 4.4.2.

We should mention again here that the selected features must be quite specific in order to increase estimation accuracy but also quite general to support a wide range of devices and architectures, even different instruction sets.

4.8 Evaluation of Energy Estimation

4.8.1 Experimental Setup

- Applications: Applications from Rodinia and Polybench suites, presented in Section 3 are used.

- Targeted hardware: The methods were first designed and tested on the widely-used *Nvidia Jetson TX1* embedded platform (see Section

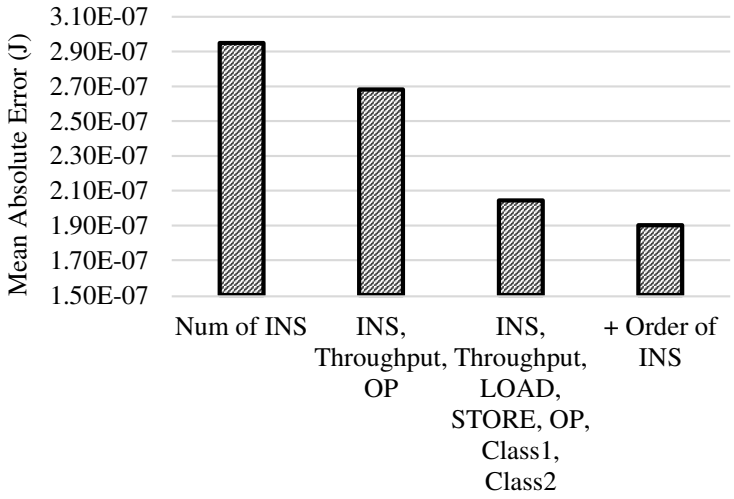


Figure 14: Impact of the selected features on Energy Estimation error

3). For testing the extensibility, another embedded device was used, namely the *Nvidia Jetson Xavier NX*. Finally, to test if such a method can be extended towards supporting also high-end computers/services, *Intel Xeon Gold 6138 server*, also presented in Section 3, was used.

4.8.2 Evaluation Results

Figure 15 presents the static analysis estimation results for running the most important basic blocks of the Polybench and Rodinia benchmark applications. The code blocks that we selected to analyse correspond to the bodies of the most computationally intensive loops of the applications. These loops have a large impact on the total program's performance and energy. The applications are executed on ARM Cortex A57 and the energy is calculated on the Nvidia Tegra TX1 platform through an integrated energy sensor. As shown in this Figure, most of the points are very close to the ideal diagonal line. According to these results we concluded that the estimation accuracy can be considered acceptable (R^2 score = 0.92). The largest divergences are observed for blocks that consume the lowest energy (less than 10^{-8} J) such *bigc*, *syr2k* and *doitgen* applications from the Polybench benchmark suite.

Similarly, Figure 16 presents the estimated (through dynamic analysis) and the actual energy of the application loops for the default input data sizes on Nvidia Tegra TX1. Actual energy ranges from less than 0.01 to 10 Joules. Based on these results we might claim that the estimation can be considered accurate. More specifically, the Mean Absolute Error (MAE) is 0.17 Joules, while the estimations follow the real values very well, as the final R^2 score

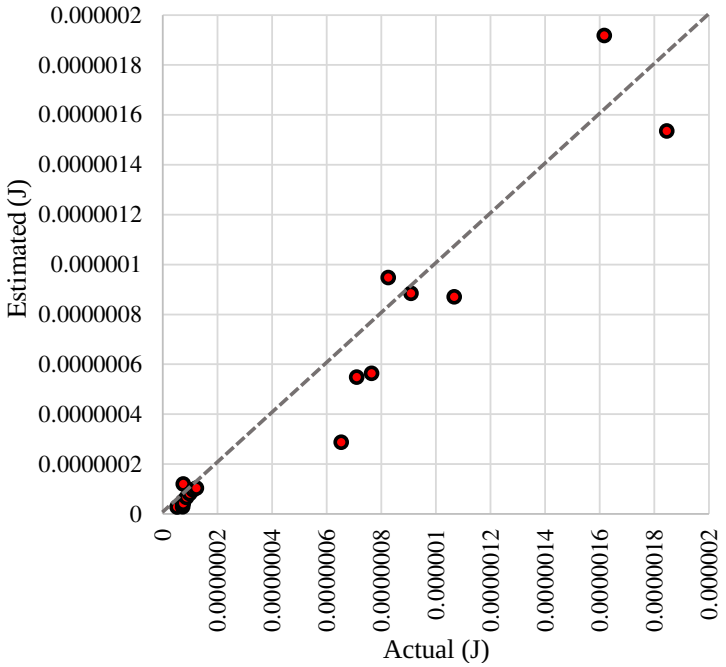


Figure 15: Actual vs Estimated energy of Rodinia/Polybench basic blocks (Static analysis) - Test on ARM Cortex A57

exceeds 0.96.

After comparing the results presented in Figures 15 and 16, we can conclude that the static analysis approach is (as expected) less accurate ($R^2 \approx 0.92$), while it requires additional user input that is not always easy to obtain (e.g. the number of iterations of each loop body) for making predictions for the entire execution of the loop. A more detailed comparison is depicted in Figure 17, where the results for 6 representative applications from the Polybench suite are presented. The average error is 0.14 Joules for the proposed approach, while the static method has an average error of 0.24 Joules.

4.8.3 Extensibility evaluation

The results of estimating the Polybench/Rodinia applications energy vs the actual values using the dynamic analysis approach for the case of using Nvidia Xavier NX are presented in Figure 18. As shown in this Figure, for the applications that consume less than 0.2 Joules of energy the predictions were mostly overestimating the actual energy. Additionally, there are three applications for which we have a relatively large error (more than 0.3 Joule).

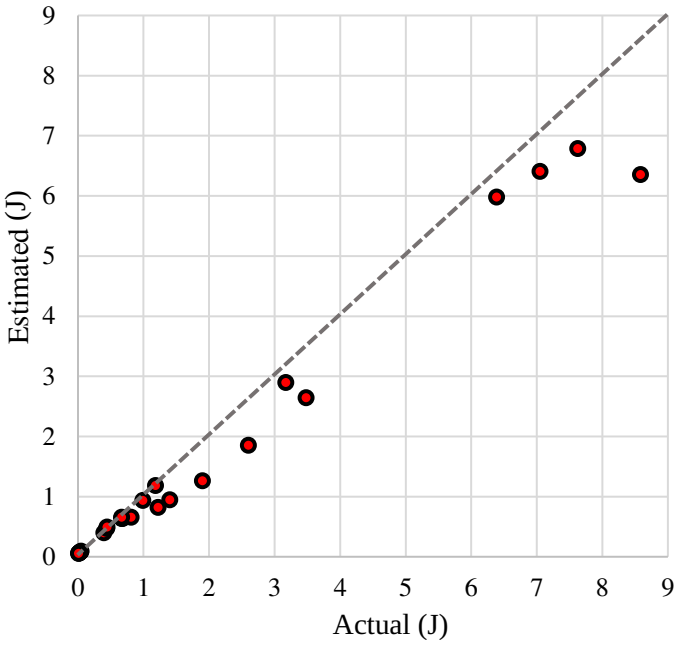


Figure 16: Actual vs Estimated energy of Rodinia/Polybench loops (Dynamic analysis) - Test on ARM Cortex A57

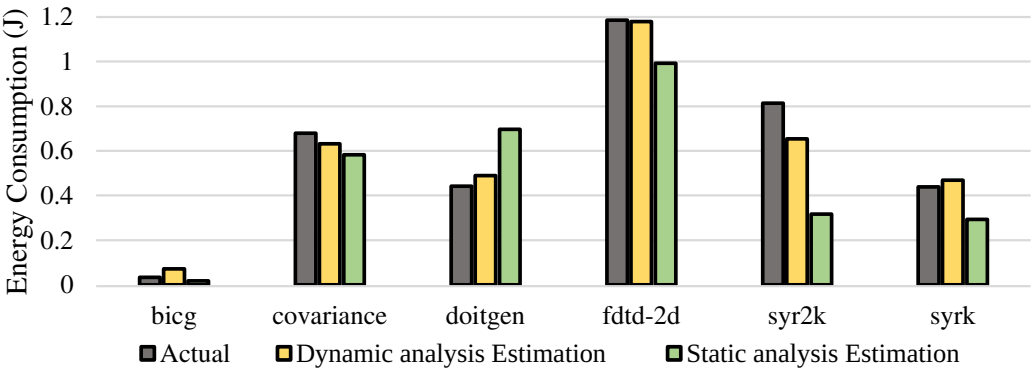


Figure 17: Dynamic against static analysis approach - Representative Polybench benchmarks

However, we might conclude that these miss-predictions do not affect the overall quality of the presented methodology. The study of their characteristics will give a future direction for improvements, while a refinement of the model's parameters is expected to lead to better results. In the context of this work we focused on showing the results of using exactly the same model.

Finally, we extended the proposed energy estimator by adding the HPC system that incorporates a high-end CPU, namely the Intel Xeon Gold 6138.

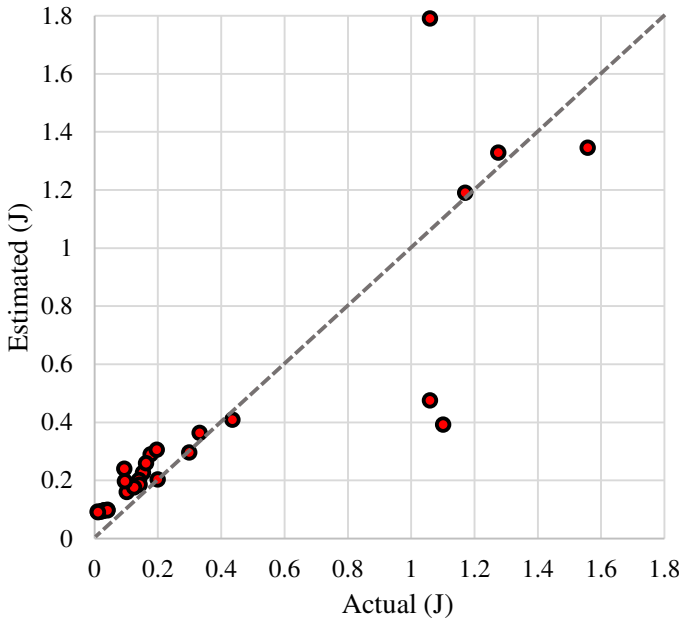


Figure 18: Actual vs Estimated energy of Rodinia/Polybench loops (Dynamic analysis) - Test on Xavier NX (ARM v8.2)

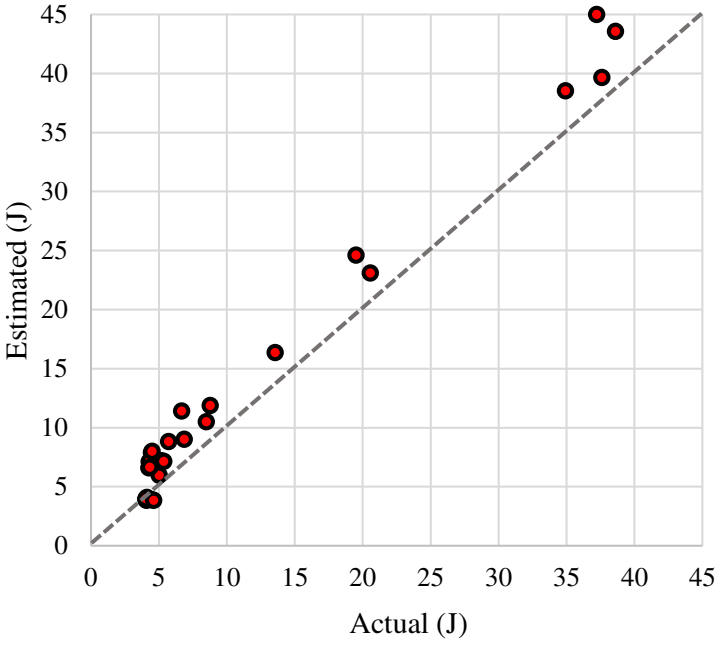


Figure 19: Actual vs Estimated energy of Rodinia/Polybench loops (Dynamic analysis) - Test on Intel Xeon Server

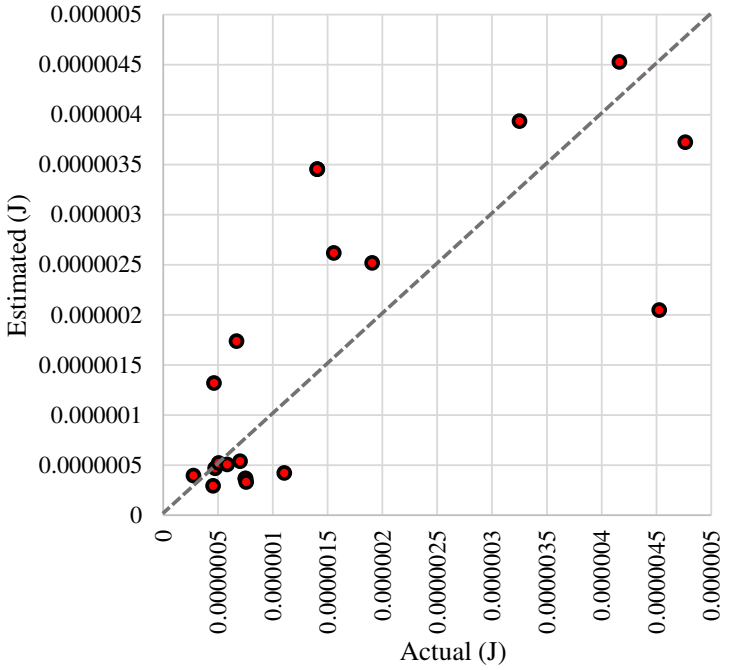


Figure 20: Actual vs Estimated energy of Rodinia/Polybench loops (Static analysis) - Test on Intel Xeon Server

Following the introduced procedure (see 4.5), we estimated the server CPU and DRAM energy. The results for estimating the energy consumption using the dynamic analysis method are presented in Figure 19, while the static analysis based estimation for the most important basic blocks of the Rodinia and Polybench benchmark suite are presented in Figure 20.

The loops consume up to 45 Joules on the server. Dynamic analysis offers accurate results as it achieves an R^2 score of 0.9 and an average error of 2.4 Joules. According to the results presented in Figure 19, we might observe that the model overestimates the energy for the most cases. However, this observation can only be a promising direction towards improving the model and a validity of its extensibility capabilities as no refinement of the model's parameters was performed.

For static analysis, the average error is high compared to the estimations made for embedded systems. A server is much more complex and the energy is affected by many parameters and peripheral systems. The processor itself incorporates 20 cores (40 threads) making the correlation of a simple instruction block with energy consumption more complex. Finally, the energy measurements required for building the proposed model, as well as the measurements for making the final evaluation, can be very noisy. However,

we might conclude that the results are not prohibitively inaccurate. We can clearly see that the order of magnitude of energy consumption is approached correctly: Polybench blocks consume about 5 times less energy compared to the Rodinia ones and the prediction is also close to the actual values. Additionally, in the Rodinia benchmark suite the more energy expensive blocks are also predicted to consume more energy than the others leading to accurate results in terms of comparison (R^2 score = 0.85).

According to these results we might conclude that the proposed method can partially support importing CPU-based servers easily.

4.9 Conclusion

A complete methodology for designing practical analysis tools to be used by developers for estimating energy consumption for running an application on different embedded devices was presented. The introduced framework uses random synthetic loops and regression methods. The approach offers both static analysis and dynamic analysis based solutions, in order for the user to exploit the advantages of both. The proposed solutions achieve similar effectiveness compared to related state-of-the-art tools but focuses on building an easy-to-use and extensible solution that can be part of SDK tools. Particular emphasis was placed on studying the correlation between alternative features and energy as well as the tool's capabilities to add new targeted platforms in an easy and convenient way.

Chapter 5

5 Energy Optimization

The present work introduces an energy optimization methodology that consists of two basic parts. The first one is similar to the logic in Chapter 4 which presented our methodology for designing cross-device energy consumption estimation tools applied at source code level. This component not only estimates the consumption of individual code blocks but also monitors indicators helping developers to understand the application behavior that consumes more energy and identifies the most consuming parts of the code (hot-spots), where developers should focus.

The second part suggests optimizations to reduce the energy of the application under analysis, focusing on the identified energy hot-spots. The current version of the introduced framework suggests various categories of optimizations, namely the data-flow optimizations, the concurrency-related optimizations, platform selection, function placement on Edge resources and the acceleration optimizations. In the context of this thesis, platform selection is simply based on the cross-device energy estimations presented in Chapter 4, while data-flow and concurrency related optimizations are partially supported by making suggestions based on the energy indicators values (see Section 5.2). Our main research focus is on estimating the potential energy gains by acceleration (Section 5.3), while we also investigate the use of our estimation models for supporting an energy-aware placement on the Edge (Section 5.4). Also, a study of the impact of the proposed source-code optimizations on design-time quality (software development) is presented in Section 5.5.

The flow of the proposed methodology is depicted in Figure 21 and is presented in detail in the rest of this Chapter. Our optimization strategy follows three key criteria:

- Optimizations can be applied at the application source code level.
- Optimizations should be applicable in a wide range of embedded devices. In other words, we focus on optimizations applied to families of

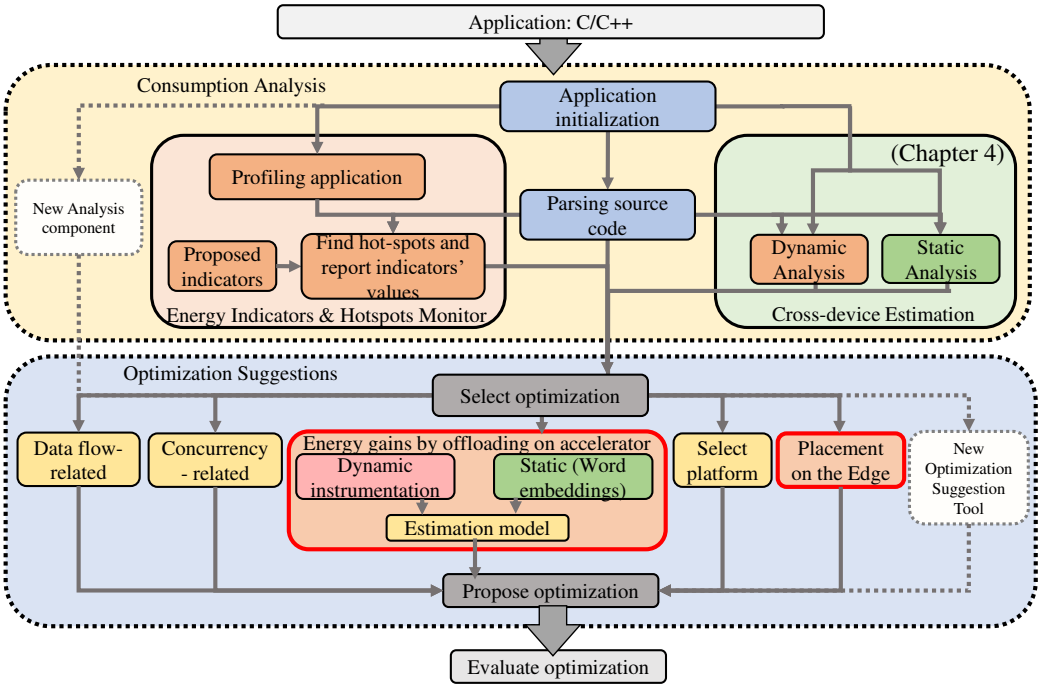


Figure 21: Overview of the energy optimization framework

embedded architectures, rather than to a specific embedded platform.

- Optimizations will have an impact on other software qualities such as maintainability. This will be discussed in Section 5.5.

For the design of the introduced solution, we focused on meeting two goals:

- Usability: The introduced solution offers an analysis that is closer to the software developers point of view. The consumption analysis and the optimization suggestions are identified and presented at the source code level. Although measuring energy (through sensors) on the actual hardware leads to more accurate estimations, targeted hardware alternatives are not always accessible. Also, measurement may require special equipment and knowledge.

- Extensibility: The proposed tools are designed based on specific computing architectures and offer energy optimizations for embedded systems. Our design, however, aims to be generic enough and able to be extended for other types of architectures by following the provided step-by-step guidelines. These extensibility capabilities are presented and demonstrated in evaluation section.

Similarly to Chapter 4, the introduced optimization tools rely on both dynamic and static analysis of the application in order to exploit the advantages of both.

5.1 Energy Indicators and Hotspots Monitoring

There exist various metrics that can be considered valuable indicators that monitor the energy consuming application behaviour. Their values represent particular aspects of an application executed on specific hardware architecture. Some of them can be used to show how effectively an application exploits features of the underlying architecture, such as the memory hierarchy. Their optimization potentially reduces the energy consumption.

In this Section we focus on indicators that can be monitored using tools that are mature, actively supported by developers and widely used by the embedded systems community. The selection of energy indicators is based on the following three criteria:

- The indicators should be directly related to the source code of the application. Indicators that are mainly controlled by OS or hardware architecture-level techniques are not selected. For instance, TLB misses are mainly controlled by the memory management unit and the CPU clock frequency is very important but it is controlled by the dynamic frequency scaling.

- There should exist source-to-source transformations that may improve their values. More specifically, the values of the indicators are expected to indicate corresponding optimizations.

- Some of the selected indicators should indicate the efficiency of assigning the execution of a part of CPU source code on an accelerator. We selected indicators from the literature that can be used to estimate the energy gains of offloading a piece of application source code on an accelerator (see Section 5.3).

The selected energy indicators that are integrated in the current version of the framework are summarized below. The introduced indicators will be used to indicate source-to-source optimizations that improve energy efficiency. For each indicator, we describe its impact on energy consumption as well as the tools that can be used to monitor its value.

- *CPU cycles:* A CPU cycle generally refers to a clock cycle (a single tick of the internal clock of the CPU). Some instructions on a CPU take

multiple cycles to be executed. In most cases, we consider as an optimization goal, the ability to execute multiple instructions in a single cycle (pipelining, parallelization). Therefore, the number of instructions executed in a given cycle may vary across different application implementations. The number of CPU cycles is directly related with the performance as well as with the energy that the application consumes. This indicator is generic and applicable in every CPU-based application. A large number of optimizations can reduce the CPU cycles (loop transformations for better memory utilization, acceleration, etc.) [138].

- *Ratio of branch misses:* Modern CPUs use branch prediction mechanisms that attempt to guess the branch that will be selected and fetch the assumed instructions in the CPU pipeline. However, when CPU branch prediction fails, a branch miss occurs and the CPU pipeline is flushed, which has a negative impact in application's energy consumption. Removing of recursion may reduce branch misses.

- *Number of memory accesses:* Each memory access imposes a cost in terms of energy consumption and execution time. The amount of cost is affected by various parameters, such as the layer of the memory hierarchy in which the access occurs. Reducing memory accesses improves the execution time and the energy consumption. There is a lot of work in the literature towards this direction by applying transformations at the source-code level (e.g. using data-flow optimizations) [139].

- *Ratio of D-Cache and I-Cache misses:* Cache utilization directly affects energy consumption. A cache miss occurs when a CPU fetches requested data from the main memory of the system, because they are not available in cache. There is a lot of work in the literature that proposes source-to-source optimizations to reduce the ratio of cache misses (e.g. loop optimizations that improve locality).

- *Data races:* A data race occurs when two or more threads access the same memory address simultaneously, with at least one thread altering the data stored in this memory location. The absence of mechanisms that coordinate the memory accesses by multiple threads to shared data (mutexes, atomic operations, etc.) generate data races. Possible effects of data races are the erroneous results leading to wasted energy, since restarting the execution may be required.

As the proposed framework aims to be continuously extending and evolving more indicators may be added in the future for serving the indication

of more potential source-to-source optimizations. The next indicators that we suggest to add, which are able to be reported using the already integrated tools, are the *the (Ratio of) CPU stalls*, *the number of page faults* and *the number of heap memory blocks that are lost* (or memory leaks). CPU stalls are often a result of a cache miss, when a CPU waits for a cache line to be fetched from another layer of the memory hierarchy. Mitigation techniques include out of order execution and hyper-threading. Apparently, CPU stalls contribute to increase application execution time and energy consumption [140]. A page fault is an exception raised by the system when a program tries to access a non-mapped memory address. The Memory Management Unit detects this fault and raises an exception, which is handled by the operating system or the page fault handler. Page faults have negative impact on energy consumption and execution time [141]. C and C++ developers are responsible for allocating and deallocating heap memory within the application source code. Failing to deallocate a block of memory that is no longer used is a common bug called memory leak. Memory leaks are a cause of wasted heap memory. Ineffective management of the heap memory may cause its exhaustion, yielding application crashes and wasted energy consumption. In programming languages that utilize a Garbage Collector, such as Java, memory leaks are not as frequent as in C/C++, in which developers are responsible to deallocate heap memory. However, there are still cases that unused memory cannot be deallocated by the Garbage Collector, causing memory leaks: Objects and data that are no longer used but they are still referenced [142]. For example, data that are defined as static or originate from external data streams and connections may cause memory leaks, increasing the risk of running out of memory, leading to wasted energy consumption.

With regard to multi-threading applications, *Lock contention* monitors the time that a thread waits for a lock that is currently held by another thread. As a result, this metric indicates thread starvation, where CPU cores are active (consuming energy) without performing useful work for the application progress. This indicator can help the developer use better locking mechanisms. *Lock order violation* monitors situations where using multiple layers of locks is required. Lock order violation is a common bug that may lead to deadlocks, causing wasted energy consumption [143].

Our framework starts with the initialization of the application. This stage includes retrieving information about the targeted source files by the user and building/running the application. Afterwards, the tool proceeds to the application profiling, supported by dynamic instrumentation tools, such as Valgrind and Linux Perf and returns the energy indicators (after processing the generated results).

The definition of an energy hotspot (in the context of this thesis) is as follows: *A block of CPU source code, in which significant number of CPU cycles are spent, compared to the application's total.* A hotspot is a candidate place for applying energy optimizations.

By generating the Abstract Syntax tree (AST) of the application using CLANG, *for* and *while* blocks are identified. Then, the application is dynamically analysed to monitor CPU cycles, by leveraging a widely used dynamic binary instrumentation profiler: Callgrind by the Valgrind suite. By combining the information generated by the dynamic analysis (i.e. Callgrind output) and the statements identified by the AST processing, the number of CPU cycles spent in each statement is calculated. The code blocks in which the number of CPU cycles spent is above a threshold (e.g. 3% of the total applications cycles) are considered as hotspots. For each hot-spot, the corresponding values of energy indicators such as CPU cycles and cache misses are provided. All this information is forwarded to the next component, which is responsible for suggesting suitable optimizations.

5.2 Data flow-related optimizations

The first category of energy optimization techniques, at application level, aims to improve the memory hierarchy utilization. Since the energy consumed by memory operations depends on whether the access hits or misses in the cache memory, we can claim that the cache behaviour is very important for optimizing energy.

Typical examples of these techniques are the loop transformations that aim to improve the cache performance, by improving data locality and to reduce the overhead of the loops, that are often the most computationally intensive parts of an application. Due to the fact that each memory access has a cost in terms of energy and performance, this kind of transformations aims also to improve the memory utilization and to reduce memory allocation and memory accesses.

Data-flow optimizations are proposed in the case that the hotspot under analysis includes nested loops that have a number of cache misses that is beyond a threshold, which can be set by developers (default: 3%). Cache misses are directly related to energy consumption, as mentioned in Section 4.4.1. The Spearman correlation between Data cache misses and Energy consumption is realy strong (above 0.97). Developers can either use tools that implement the proposed optimizations automatically (e.g. Pluto, Orio [144] [145]) or perform them manually. In the context of the present work, we expensively used these two tools with very promising results in terms of the reduction of programming effort (see also Section 5.5). These tools alleviate the need of

Table 5: Indicative loop transformations for improving energy/performance

Before	After
Loop Merge:	

for (i=0; i<N; i++) {
  //do something...
}
for (i=0; i<N; i++) {
  //do something else...
}
	for (i=0; i<N; i++) {
	  //do something...
	  //do something else..
	}

Loop Interchange:	

for (j=0; j<N; j++) {
  for (i=0; i<N; i++) {
  sum += a[i][j];
  }
}
	for (i=0; i<N; i++) {
	  for (j=0; j<N; j++) {
	  sum += a[i][j];
	  }
	}

Loop Tiling:	

for (i=0; i<MAX; i++) {
  for (j=0; j<MAX; j++) {
  A[i][j] = A[i][j] + B[i][j];
  }
}
	for (i=0; i<MAX; i+=BLOCKSIZE) {
	  for (j=0; j<MAX; j+=BLOCKSIZE) {
	  for (ii=i; ii<i+BLOCKSIZE; ii++) {
	  for (jj=j; jj<j+BLOCKSIZE; jj++) {
	  A[ii][jj] = A[ii][jj] + B[ii][jj];
	  }
	  }
	  }
	}

Loop Unrolling:	

for (i=0; i<100; i++) {
  A[i] = B[i];
}
	for (i=0; i<100; i+=4) {
	  A[i] = B[i];
	  A[i+1] = B[i+1];
	  A[i+2] = B[i+2];
	  A[i+3] = B[i+3];
	}

manual code refactorings and help to easily investigate if loop transformation has a positive impact on the performance and energy consumption. An automatic estimation of the impact of Loop Transformations on energy is a future work (more details are provided in Section 7.2).

Typical loop transformation examples are illustrated in Table 5:

- Loop merge (or Loop fusion) is the technique that combines two loop bodies into one. This method is applicable when two loops iterate over the same range and do not reference each other's data.

- Loop interchange, is a typical data reuse transformation that, when properly applied, reduces execution time and energy consumption by improving memory hierarchy utilization.

- Loop tiling method, breaks the iteration space of a loop into smaller blocks, in order to ensure that more data used in the loop will stay in the cache memory until they are reused. As a result, this method aims to increase the loop depth to reduce D-cache misses. Breaking the loop space leads to partitioning of large arrays into smaller blocks, aiming to fit the accessed array elements into the cache size, exploiting spatial and temporal locality of data accesses. The partition block size is defined as a parameter by the developer with a goal to maximize the reuse of data at a specific level of the memory hierarchy [146].

- Loop unrolling aims to increase the program's speed by eliminating loop control instruction and loop test instructions, as well as by exposing and enabling parallelization (if the statements in the loop are independent of each other). The loop body is repeated a number of times and consequently the loop iteration space is reduced. This type of transformation is safe to apply as the order in which the operations are executed remains unchanged, however if the number of iterations is not known there is the need to include code to check if the number of iterations is greater or equal than the unrolling factor [146].

5.3 Energy Gains by Acceleration

A massive improvement of the performance and reduction of energy consumption is achieved by using accelerators [147]. During the recent years, a plethora of heterogeneous computing architectures are introduced providing increased performance at limited energy consumption. A complementary infrastructure, which is part of modern System-on-Chip (SoC) embedded devices installs a GPU or an FPGA on the same chip (along with the CPU). This trend provides even increased performance. However, using this type of device, i.e. offloading computationally intensive parts of the application to the acceleration unit, can not be considered as trivial due to the large number of source code refactorings that have to be performed manually. While modern libraries and tool-flows aim to make this process as easy as possible, different programming tools or languages must be used properly. As a result,

the software developer still needs to know how, when and where to use these enhanced programming features.

In this direction, this subsection presents a flow for predicting the potential energy gains of offloading a hotspot to a GPU accelerator. The overview of the introduced flow is depicted in Figure 22. The hotspots identified using the aforementioned methodology (see Section 5.1) are further analyzed for monitoring the values of features (acceleration specific indicators), which will be the inputs of the prediction models for estimating energy consumption gains. Similarly to the work presented in Chapter 4, the techniques are first built and tested on NVidia Jetson Tegra X1 that incorporates a MAXWELL embedded GPU. Energy consumption was measured using the integrated power monitor.

The presented tool-flow (similarly to the techniques presented in Section 4) combines static source code analysis and dynamic instrumentation techniques to exploit the advantages of both approaches. While the developer can use both techniques for the application under analysis, one suggestion for combining static and dynamic methods in a single flow is the following: Initially, the hotspots are statically analyzed (Section 5.3.2) using text analytics methods, that provide a coarse grained estimation of potential energy gains. Throught this procedure, the hotspots are classified into one of the following two categories: "High-gains" or "Moderate/no gains". The hotspots that are predicted to belong into the "Moderate/no gains" category are further analyzed using dynamic instrumentation methods for fine grained predictions, as described in Section 5.3.3. In this way, the overhead of dynamic instrumentation can be avoided in cases in which high energy gains by acceleration are predicted. Predicting the cases for which relatively high gains are expected using only static analysis gives a significant advantage as the time required to make a prediction by using dynamic analysis can reach many hours for some applications. Static analysis, on the other hand, generates results almost instantly. More information about this process and the motivation behind its implementation will be given in Section 5.3.5. But first of all, we should analyze the process we followed for building the dataset that is used for training the estimation models of both the approaches. Afterwards, the details of each approach will be described in a detailed manner.

5.3.1 Building the dataset

A major challenge in designing estimation models, both in general and in the case of the source code analysis on which this study focuses, is to create a high quality dataset. This is also highlighted by several related studies [78, 79]. Especially in the case described in this section, the main difficulty relies

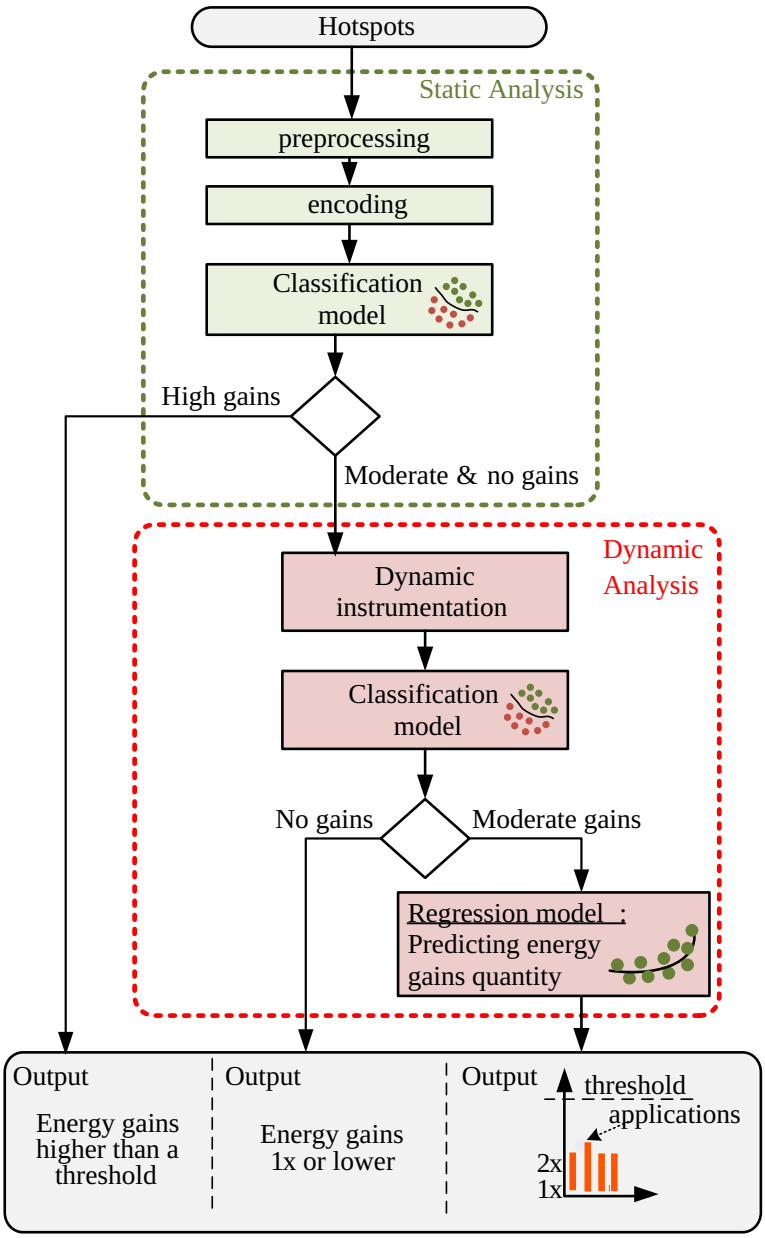


Figure 22: Estimation of energy gains by acceleration

on the need to include CPU (source code) applications, together with the corresponding GPU kernels (i.e. the accelerated GPU-version of the specific CPU code). Apparently, we could find a relatively small number of available benchmark suites that provide both CPU and the exactly corresponding accelerated code. As a result, in the work presented in the context of this

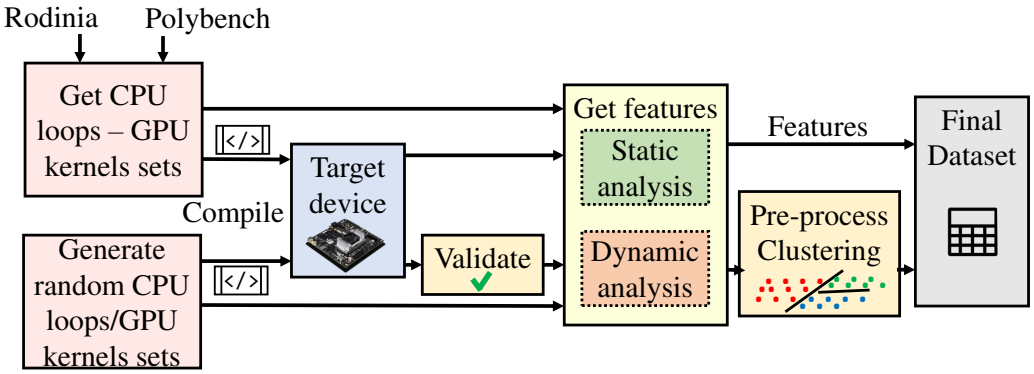


Figure 23: Building dataset for estimationg energy gains by acceleration

dissertation, we used a dataset that combines an equal number of synthetic benchmarks and real-world applications. This choice aims to reduce the danger of over-fitting that could be caused by the small dataset size that could lead to biased results. Figure 23 shows the steps we followed to build the dataset, which are detailed in the next paragraphs.

Regarding the synthetic part of the dataset, we used the same procedure described in Section 4 for energy estimation. The major difference, as mentioned before, is that we also need to include the corresponding GPU code. However, due to the fact that our Python scripts generate simple *for* loops that perform random matrix and vector operations, adding also an extra check that we don't write the same matrices that are also read in the loop to avoid data conflicts, the corresponding CUDA kernels can include the same loop body. Finally, to keep only valid data points in the data set, we run both CPU and GPU versions, and a test phase follows that performs cross-checking to all the resulting data structure values. This validation method checks that no errors were occurred due to data conflicts in the parallel execution because of the inherent random characteristics of the generated code.

The real-world application data-points of the dataset include kernels from the already mentioned and extensively used in the literature and in the context of this thesis, Polybench [127] and Rodinia benchmark suites [126]. These suites include a set of CPU applications, as well as the corresponding GPU version of each application. Following a similar approach for dataset building with other approaches [78], the size of the dataset is increased by modifying the input of some kernels (i.e. the amount of data processed). So, the final number of real-world data-points equals to 100. We should notice here that since the dataset includes same kernels with different inputs or a single application may include very similar kernels, we made sure that for all the experiments that we performed in the context of this study, data-points

that belong to the same application will not be in training and test sets consequently.

A further refinement to the synthetic part of the dataset is achieved through eliminating very similar data-points that may cause overfitting. This process is performed in the same way employed in Section 4, through a k-means clustering pre-processing of the data-points. After applying the clustering method we select one data-point from each cluster. The number of generated and selected data-points is configured to be equal to the real-world data-points (100), resulting in a final training dataset of 200 data points in total. The method for gathering the utilised dataset, consisting of 50% of synthetic benchmarks and 50% of real-world applications is publicly available[21].

A reasonable question one might ask here is how the quality of CUDA code can affect results? Obviously, the quality of the CUDA code is expected to affect the quality of the dataset and, subsequently, the accuracy of the predictions. Since the CUDA synthetic part is relatively simple, CUDA quality concern us for the real-world part of the dataset. The utilized benchmark suites are widely used in many research works, both in the embedded and HPC domains. So they are constantly maintained and improved. As a result, we might claim that it makes sense to assume that the CUDA code quality is relatively high, or at least close to the code quality created by experienced developers. Although studying the impact of the code quality on the accuracy of predictions is an interesting issue, it is beyond the scope of this work.

5.3.2 Static analysis approach

As presented in Figure 22, the static analysis mechanism first receives the C/C++ CPU source code that corresponds to the hotspot that is a candidate to be offloaded to a GPU. This component classifies the received blocks of source code into one of the two categories, with respect to the expected energy savings: "High gains" and "Moderate/No gains". As mentioned above, the goal of this analysis is to identify with a very small time overhead, the hotspots that will benefit a lot by GPU acceleration ("High gains"). The threshold between "Moderate/No gains" and "High gains" classes is selected so that the accuracy of the static analysis based classification is maximized. This is demonstrated in the evaluation section. The accuracy of the static analysis significantly decreases when more that two output classes are defined. As a result, discrimination between "Moderate gains" and "No gains"

[21] `https://git.microlab.ntua.gr/hmar/Decision_Support_for_GPU_Accelerator_dataset`

is not possible by using static analysis. As a result, for identifying hotspots for which "No gains" are expected and to provide fine-grained predictions for "Moderate gains", dynamic analysis must be used.

From a technical point of view, the static analysis method is based on text analytic approaches. It is inspired by existing work in the literature that focuses on analysing source code but for different purposes [148]. Figure 24 presents the static analysis procedure. After pre-processing the source code to remove information that is irrelevant to the code's structure (e.g. comments), the source code is encoded into sequences of integers. A vocabulary is constructed including all the source code that can be derived from the training dataset. A tokenization mechanism is then employed to convert each character into a vocabulary item. Common language words such as *if*, *for*, *while* etc are special vocabulary items. The encoded source code (in the form of word embeddings) is then given as input to the classification model. The utilized prediction model is a Convolutional Neural Network (CNN) for sequence classification [149]. The input sequences have a maximum length of 500. The CNN includes 250 hidden neurons and uses filter sizes of 12. Related studies that leverage static analysis techniques for speed-up prediction (e.g. [150]) use also dynamic data as input (such as the number of loop iterations or the direction of branches). In the context of this study, we decided to test the use of a more sophisticated approach that relies only on source code without any dynamic information. This choice may reduce the prediction accuracy, as the actual gains in absolute values are affected a lot by such execution-context metrics. Considering, however, that the orders of magnitude of the gains are not usually affected , we preferred to design a coarse-grain classification based on static analysis, having the source code as the only input of the prediction model. Of course, the enhancement of the static analysis approach with the dynamic information obtained earlier by dynamic instrumentation is a worth investigating direction to improve the classification accuracy of the static analysis step. For example, during the hotspot identification, the CPU cycles are calculated for each code block that is a candidate for acceleration. However, we decided to use a purely static analysis approach for coarse grained prediction and to provide as input only the application source code. The reason is the fact that developers familiar with the application under analysis are already aware of the hotspots, therefore, they are expected to skip the entire hotspot identification step.

5.3.3 Dynamic instrumentation

The dynamic instrumentation based method for predicting the potential energy savings by acceleration is described in this paragraph. The goals of

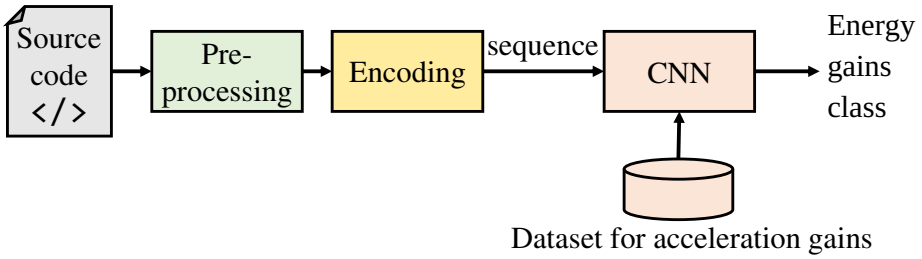


Figure 24: Flow of prediction of energy savings by acceleration based on static analysis

the dynamic analysis are the following:

- identifying the hotspots for which "No gains" are expected using a classification model
- performing a fine grained analysis to the hotspots classified into the "Moderate gains" using a regression model

In case we follow the suggested combination between the two technices, then as shown in Figure 22, the dynamic analysis component of the methodology will receive as input the output of the static analysis component and more specifically the hotspots for which "Moderate/No gains" are predicted.

The CPU code blocks (provided by the hotspots identification), are analyzed using profiling tools presented in Section 3 and already used in Section 4.4.1 for CPU to CPU cross-platform energy estimation. These tools include Intel Pin (custom designed metrics) and Valgrind and monitor the values of accelerator-specific indicators that capture the extent by which the code behaviour can exploit the architectural features of the GPU accelerator. The selected indicators are a subset of the most widely used features defined in the literature [78,151] for speed-up prediction, after studying their correlation to energy consumption. The final feature set contains the following metrics:

- Total number of instructions in the code block
- Instruction level parallelism
- Number of cold memory references
- Number of single precision floating point operations
- Number of integer operations
- Number of control/branching operations

- Number of memory operations
- Number of memory accesses with zero stride
- Branch divergence
- Data reuse percentage
- Number of data blocks that belong to the same memory page

The selection of the features was performed by investigating the suitability of each candidate metric (retrieved from a set of metrics already used in the literature for speed-up) for predicting potential energy savings. For this purpose, we employed the stepAIC method. This method is used for identifying an optimal set of features by selectively adding and removing features and using regression methods to evaluate the importance of each one. In this way, AIC (which stands for Akaike Information Criteria) quantifies the amount of information loss when a feature is removed. AIC re-estimates the prediction error and thus, the quality of each model based on different subsets of the examined features.

Table 6: Importance of features in terms of relation to GPU version energy

Features	p-value
Instruction Level Parallelism	2.12e-06
Number of instructions	1.44e-07
Number of cold memory references	0.033269
Number of single precision floating point operations	0.035817
Number of integer operations	0.006868
Number of control operations	0.039069
Number of memory operations	5.14e-05
Number of memory accesses with zero stride	1.41e-06
Branch divergence	2.84e-11
Number of division operations	0.061380
Number of blocks accessed in the same page	2.75e-05

Table 6 presents the result of the StepAIC analysis. More precisely, the features with the highest relation to the energy consumption of the accelerated version, which are the ones for which the p-value is less or close to 0.05, are presented. For our statistical analysis, the null hypothesis is that we can remove a feature from the features vector as it is not related with energy consumption. This hypothesis is rejected when $p < 0.05$ and not rejected when $p > 0.05$. It should be noted here that the 0.05 value is a standard

choice and it is worth keeping in the features-set metrics for which the p value is close to 0.05, such as the Number of division operations for which $p = 0.061380$, according to the results presented in the Table 6.

For analysing a hotspot in terms of energy gains, the values of the features presented in Table 6 are first forwarded to the classification model. If the hotspot is classified into the "No gains" class, no further analysis is required. If the hotspot is classified into the "Moderate gains", then the values are forwarded to a regression model for predicting the expected gains as accurate as possible. For this proposes, the acceleration gains are predicted as a percentage of corresponding CPU energy consumption. For example, energy gains of 3× means that the accelerated version of the code will consume 3 times less energy, compared to the corresponding CPU code.

5.3.4 Experimental setup and methodology

The evaluation of the methodology presented in this Chapter is based on the level of prediction accuracy of each of the two mechanisms (the static analysis based and the dynamic analysis based). As mentioned before, the evaluation results are first analysed for the NVidia Jetson Tegra X1 that incorporates a MAXWELL embedded GPU. Energy was measured using the installed power monitor (INA3221). Then we test the extensibility of the method to support and provide results for two more Nvidia SoC embedded devices, namely Nvidia Jetson Nano and Nvidia Xavier NX as well as for a high-end server (Section 5.3.7).

After training the estimation models using the dataset described in Section 5.3.1, we first defined the boundary between the "High gains" and the "Moderate/no gains" classes (the boundary used by the static analysis based component). To select this boundary, we performed a k-means clustering on the energy gains degree of the data to include hotspots with similar energy gains in each class. For setting the number of clusters into 2, k-means makes a cluster that includes codes with energy-gains $\leq 83\times$ and another with energy-gains $> 83\times$. However, in this scenario only 11 of the 100 hotspots are placed in the "High-gains" class, while we consider such a boundary very high. So, we tried to increase the number of k-means clusters. The results are the following:

- clusters = 2:
  - cluster 1 (gains $\leq 83\times$) - Data-points: 89
  - cluster 2 (gains $> 83\times$) - Data-points: 11
- clusters = 3:

  - cluster 1 (gains $\leq 49\times$) - Data-points: 88
  - cluster 2 (gains $\leq 128\times$) - Data-points: 5
  - cluster 3 (gains $> 128\times$) - Data-points: 7

- clusters = 4:
  - cluster 1 (gains $\leq 16\times$) - Data-points: 68
  - cluster 2 (gains $\leq 49\times$) - Data-points: 18
  - cluster 3 (gains $\leq 128\times$) - Data-points: 7
  - cluster 4 (gains $> 128\times$) - Data-points: 7

- clusters = 5:
  - cluster 1 (gains $\leq 16\times$) - Data-points: 68
  - cluster 2 (gains $\leq 49\times$) - Data-points: 18
  - cluster 3 (gains $\leq 128\times$) - Data-points: 7
  - cluster 4 (gains $\leq 180\times$) - Data-points: 5
  - cluster 5 (gains $> 128\times$) - Data-points: 2

- clusters = 6:
  - cluster 1 (gains $\leq 16\times$) - Data-points: 68
  - cluster 2 (gains $\leq 49\times$) - Data-points: 18
  - cluster 3 (gains $\leq 128\times$) - Data-points: 7
  - cluster 4 (gains $\leq 180\times$) - Data-points: 5
  - cluster 5 (gains $\leq 253\times$) - Data-points: 1
  - cluster 6 (gains $> 253\times$) - Data-points: 1

- clusters = 7:
  - cluster 1 (gains $\leq 16\times$) - Data-points: 68
  - cluster 2 (gains $\leq 49\times$) - Data-points: 18
  - cluster 3 (gains $\leq 83\times$) - Data-points: 1
  - cluster 4 (gains $\leq 128\times$) - Data-points: 6
  - cluster 5 (gains $\leq 180\times$) - Data-points: 5
  - cluster 6 (gains $\leq 253\times$) - Data-points: 1
  - cluster 7 (gains $> 253\times$) - Data-points: 1

According to these results, we conclude that k-means does not split the cluster of energy gains $\leq 16\times$, which includes 68 data-points. This means that the hotspots in this cluster have very similar gains. Indeed, if we place the gains in an ascending order, for each subsequent hotspot the gains increase by $0.23\times$ on average for hotspots of energy gains $\leq 16\times$ and by $10.43\times$ for hotspots of energy gains $> 16\times$. Therefore, we choose a class boundary of $16\times$. This means that, hotspots for which energy gains above $16\times$ are predicted, are classified into the "High gains", while the rest of the hotspots are classified into the "Moderate/No gains" class. This threshold is recalculated each time the dataset changes in order to support additional platforms or in order to optimize the accuracy for the specified platform. Therefore, this procedure takes place rarely, only when the models are retrained.

For the evaluation purposes we select to highlight the proposed methodology effectiveness on the Rodinia and Polybench benchmark suites. It is worth summarizing here the experimental methodology details:

- *Models*: The static and dynamic models were presented in Sections 5.3.2 and 5.3.3, respectively. Detailed model selection experiments will be presented in the next paragraphs.

- *Features*: The input of the static model is encoded C/C++ source code (5.3.2). The input of the dynamic model is the features gathered by dynamic instrumentation, presented on Table 6.

- *Test dataset and evaluation process*: The test dataset consists of a total number of 100 applications hotspots from Rodinia and Polybench benchmark suites [126, 127]. For the evaluating the accuracy of the introduced models, we followed the same approach with related works in the literature (e.g. [78,150]): When predicting the energy gains for a specific hotspot (i.e. single datapoint), we train the models from scratch using a training dataset that includes all the datapoints that belong to the rest of the applications. This approach, which can be termed as a modified leave-one-out cross-validation (LOOCV) is widely used in the analysis of small dataset [78, 150, 152].

- *Evaluation platform*: NVidia Tegra X1 with an integrated power monitor (INA3221 sensor) [9] (see Section 3).

5.3.5 Accuracy evaluation

In this paragraph we evaluate the accuracy of the prediction models of the methodology presented in Sections 5.3.2 and 5.3.3.

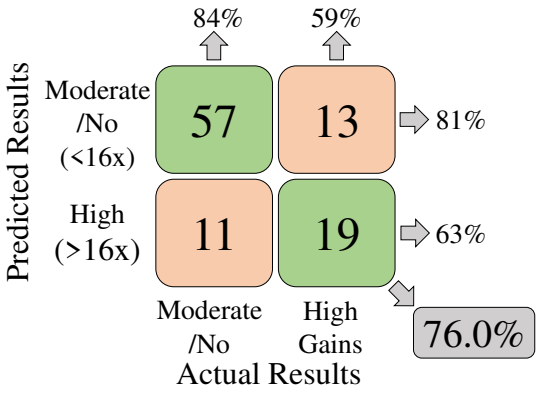


Figure 25: Predicted vs. actual energy gains class: Static analysis

The accuracy of the *static analysis component* (Section 5.3.2) equals to 76%. In other words, the probability that a hotspot that actually belongs to the "High gains" class will be classified into the "Moderate/no gains" or vice versa, is 24%. The accuracy results are further demonstrated in the confusion matrix presented in Figure 25. In this Figure the rows represent the instances in a predicted class and the columns represent the instances in an actual energy gains class. It is shown that the model correctly classifies 76 datapoints (i.e. hotspots) out of 100. Considering the fact that the static analysis receives as input only the hotspot source code, this level of accuracy is reasonable. The accuracy can be potentially improved by receiving as input, apart from the source code, information obtained by dynamic analysis. However, as stated earlier, in this work we developed a purely static analysis component and traded classification accuracy for user friendliness.

As mentioned in Section 5.3.3, the implemented dynamic analysis component includes a classification step, to identify the hotspots for which no gains are expected and a regression step to make fine-grained predictions of the energy gains by acceleration for the hotspots classified into the "Moderate gains" category. In order to select a suitable classification model, we compare the accuracy of alternative models. The accuracy of the best 7 models is depicted schematically in Figure 26. After extensive testing, we conclude that by utilizing an Ensemble Voting technique that incorporates the best 3 models (Extra Trees, Bagging Trees and Gradient Boosting) we can reach an energy gains prediction accuracy level of 85.3%. The final results are shown in the confusion matrix of Figure 27, highlighting that the misclassification probability is lower than 15%.

The input features are the values of the acceleration specific indicators used for the dynamic analysis based component (see Section 5.3.3). It is

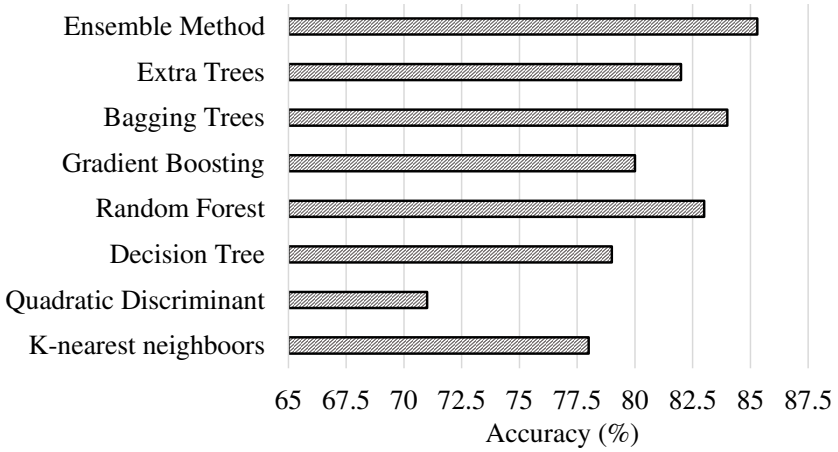


Figure 26: Comparison of accuracy of various classification models for dynamic analysis based estimation

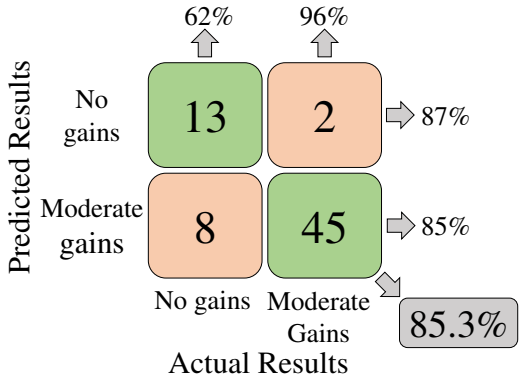


Figure 27: Predicted vs. actual energy gains class: Dynamic analysis

worth noting again that the models predict how many times the energy consumption is lower when the code is offloaded to the GPU, compared to the corresponding CPU-only execution. As a result, the prediction model assigns each piece of CPU (hotspot) code to the appropriate category.

For the regression step, we evaluated several alternative methods in terms of accuracy, as shown in Figure 28, which depicts the Mean Absolute Error of the 7 most accurate models. Based on this analysis, we selected the Random Forest regression method, which provides the highest accuracy.

The actual and the predicted energy gains regarding the hotspots classified into the "Moderate gains" category are shown in Figure 29. Each hotspot is denoted as $\{application\ name\}\_\{hotspot\ id\}\_\{input\ size\}$. The average error, which is defined as the difference between actual and predicted energy gains gains is 2.6×. However, mispredictions can be more costly for

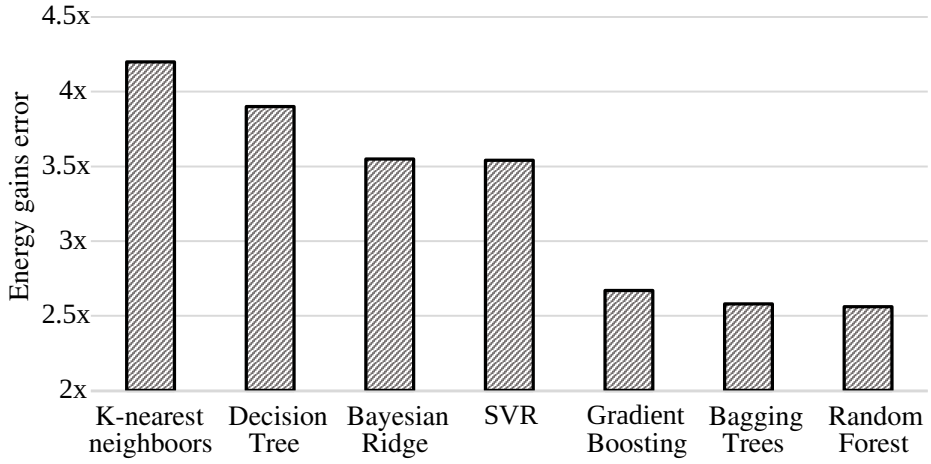


Figure 28: Energy gains prediction accuracy comparison of various regression models

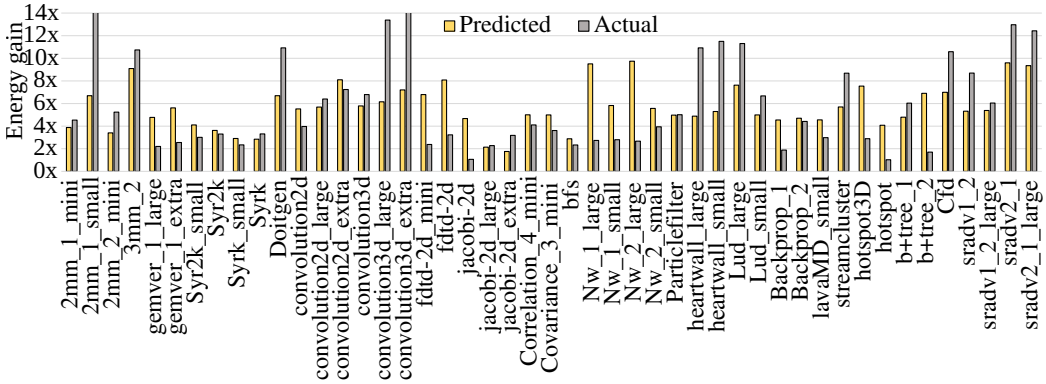


Figure 29: Predicted vs. actual energy gains using regression analysis for hotspots assigned into the "Moderate gains" category (Nvidia TX1)

low energy gains, than for higher. As an example, predicting energy gains 4×, while the actual is 2× may affect decision making to a higher degree, than in the scenario of predicting 10×, while the actual is 8×. Based on the results presented in Figure 29, there are 28 hotspots with actual energy gains below 6×. 22 out of 28 predictions can be considered accurate, since the average error is 1.4× only.

The few mispredictions observed can be attributed to the following reasons:

- The presence of a relatively small instruction level parallelism may lead the models to overestimate the potential energy gains. As an example,

Nw (for large input) falls into this category, where actual energy gains are around 3×, while the model predicts more than 9×.

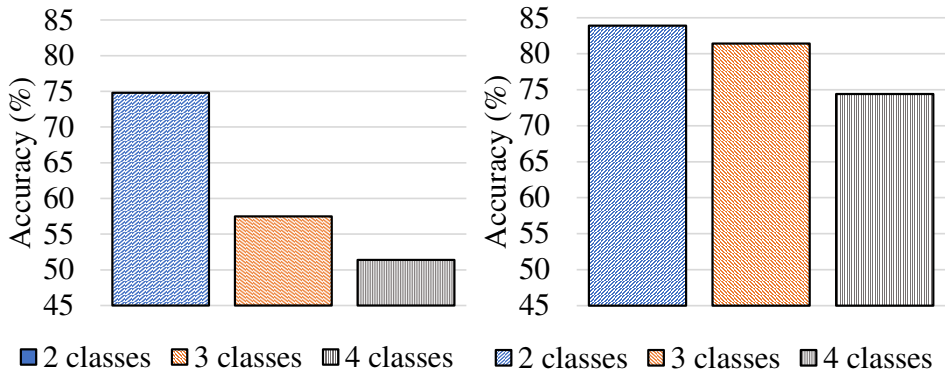
- When we have a high branch divergence we also have mispredictions. This was observed for example in *Heartwall*: the model predicts a energy gains below 6×, while the actual gains are more than 10×.

In fact, very few real-world applications and synthetic benchmarks used in the dataset had similar behavior with respect to branch divergence and instruction level parallelism. As a result, the models were not trained to provide accurate predictions for such cases. We might claim that enhancement of the dataset with more real-life and synthetic applications that have a similar behavior to the above is expected to improve the energy gains predictions.

5.3.6 Motivation for Combining Dynamic and Static Analysis

In this paragraph we present what motivated us to select a hybrid approach. If we increase the number of classes in the classification step to three or four, the prediction accuracy results of static and dynamic analysis based approaches are shown in Figure 30. For the purpose of this experiment, the classes divide the number of hotspots into equal parts. For example, in the case of having 3 classes, the boundaries are set so that each 1/3 of the dataset is classified into a different class. Based on these results, we conclude that the classification accuracy of the static analysis approach significantly decreases for more than 2 classes. This shows that the static analysis can be effectively used for coarse-grained predictions (i.e. up to 2 classes). However, for more fine grained predictions, using three or more classes, analysis based on dynamic instrumentation should be employed, as shown in Figure 30b. Indeed, predictions based on the dynamic instrumentation technique provide correct classification with 75% accuracy even for 4 classes.

The overhead of the dynamic instrumentation approach in terms of execution time is shown in Figure 31. The dynamic instrumentation is performed by analyzing each one of the applications of the dataset with Valgrind and Intel Pin tools, as mentioned in 5.3.3, in order to extract the acceleration specific indicators. Figure 31 presents the large time overhead that the dynamic instrumentation adds, making the execution more than 2000 times slower in some cases, compared to running the application without dynamic instrumentation profiling. The reason, is the fact that instrumentation process adds extra instructions in application binaries in order to extract the required information. The static-only version, on the other hand, takes only the source code as input without running the application. This means that the analysis time is constant as opposed to the dynamic analysis which is



(a) Accuracy of static analysis approach

(b) Accuracy of dynamic instrumentation approach

Figure 30: Energy gains classification accuracy (Nvidia Jetson TX1)

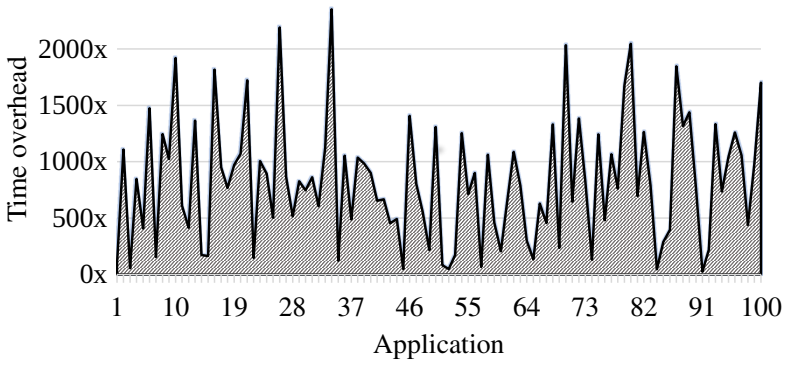


Figure 31: Execution time overhead of the dynamic instrumentation

proportional to the execution time of the application. As a result, we can argue that we can not refer to "overhead" of the static analysis component. The static analysis process takes less than 1 second to make a prediction (using an average CPU of a personal computer, e.g. Intel i7-6400). The biggest argument in favor of using static analysis in the presented solution is that it can produce fast results for hotspots belonging to the "High Gains" category. However, the overall cost of the dynamic analysis cannot be avoided in cases where the static approach is not adequate or when the developer needs fine-grained results. Nevertheless, dynamic analysis overhead can motivate even partial static-only analysis support.

5.3.7 Extensibility Evaluation

The experiments, presented earlier, were performed on Nvidia Jetson TX1 device. In this paragraph, we evaluate the extensibility of the presented tool and its ability to support more devices and systems. A prerequisite for being able to support the new device is that it must include an energy sensor (or at least a way to define the actual energy). The step-by-step guidelines that must be followed in order to add more platforms in the tool are provided below [153]:

- Step 1: Download the dataset from git repository
- Step 2: Build the dataset in the new device.
- Step 3: Run the dataset. The developer has to update the paths of power sensors of the new device.
- Step 4: The final dataset is added in the tool. The models are re-trained in the new data (transfer learning).

After retraining the models, we investigated the support of the following additional devices:

- *Nvidia Jetson Nano*: Nvidia Jetson Nano is very similar to TX1. as stated in Section 3. It should be noticed here, that we used the default power modes for all the platforms (without custom frequency scaling) and the dataset was build with keeping this configuration constant in all measurements, as the methodology aims to use the application software as input.

  For Nano, Rodinia/Polybench hotspots are classified slightly differently. More specifically, we have 4 more hotspots in the "Moderate gains" category and 1 more in the "No gains" category, while the values of energy gains differ in most cases. The accuracy of each estimation component is similar to the results for TX1. The static analysis classification reaches 78% accuracy, while the accuracy of the dynamic analysis classification is around 82%. The fine-grain regression results for "Moderate gains" hotspots are presented in Figure 32. We observe a similar performance, as the average error is 2.9×, while the proposed model miss-predicts the energy gains of the same benchmarks described in Section 5.3.5 (e.g. the Nw app).

- *Nvidia Jetson Xavier NX*: For Xavier NX (see Section 3), Rodinia/Polybench hotspots are classified differently. 28 hotspots show no gains, while 39 hotspots are expected to have high Energy gains (more than $16x$). The increased capabilities of this device in both CPU and GPU compared to the Tegra X1 offer better CPU performance and

much faster but more more power consuming GPU usage. These characteristics increase the energy gains of some hotspots, such as 2mm and 3mm from the Polybench suite due to the higher efficiency that exceeds the higher power consumption and reduces the energy gains for kernels that do not benefit from the increased number of GPU cores. Classification accuracy decreases to 78% for dynamic analysis and 75% for static analysis. This is due to the fact that we maintain the same boundary between the "Moderate Gains" and "High Gains" classes, which is based on accuracy optimization for TX1.

Figure 33 presents the regression results for "Moderate gains" hotspots. Based on these results, we observe a similar performance to the other devices, as the average error is 2.7×.

- *Intel Xeon Gold 6138 - Nvidia Tesla V100 server:* The studied application hotspots show completely different behavior, when executed on the server infrastructure. The hospots classified in the no gains class are much more (18 instead of 11 for the Tegra X1). This is mainly due to the fact that the data transfers between the main memory and the GPU have a large impact compared to the embedded device where both CPU and GPU are in the same chip. More precisely, data transfers increase energy by 22× on average for running the kernels belonging to the utilized dataset presented in Section 5.3.1, while the overhead is only 3× more energy in Tegra X1.

  The prediction results of the proposed tool are presented in Figure 34. Here, we evaluate the 50 Polybench/Rodinia hotspots (without the applied increments based on input data changes). We use just two classes "No gains" and "Gains". As depicted in Figure 34a the accuracy reaches 82% for the dynamic analysis, while static analysis clasifies around 72% of the hotspots correctly (Figure Figure 34b). Especially in the case of the server, more fine-grain results are not provided due to the very large prediction errors of the models. The biggest disadvantage of static analysis is the fact that it cannot model the volume of data required for transmission. For example, for some hotspots (e.g. in the Polybench suite correlation application), if we reduce the input data using the mini dataset provided by the benchmark, no gains are observed as the kernel also runs very fast only on the CPU and the additional energy overhead for data transfer and GPU utilization exceeds the energy consumed by the CPU-only version. This case is correctly classified by the dynamic analysis tool, while static analysis cannot work due to the fact that the source code remains the same.

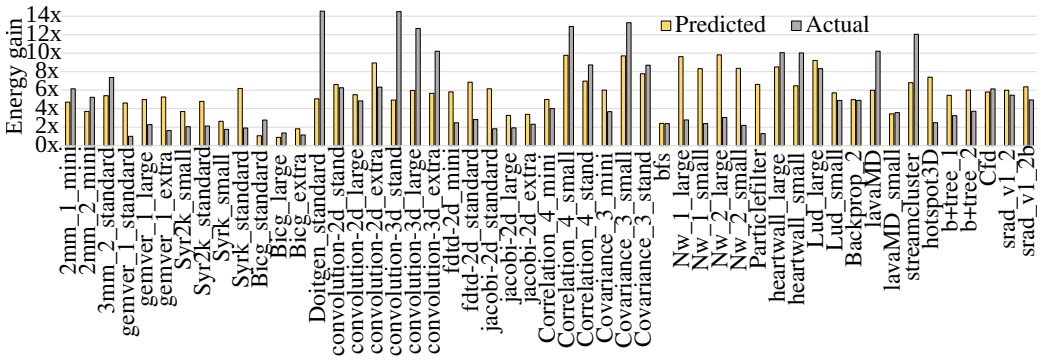


Figure 32: Energy Gains Prediction results for Nvidia Jetson Nano

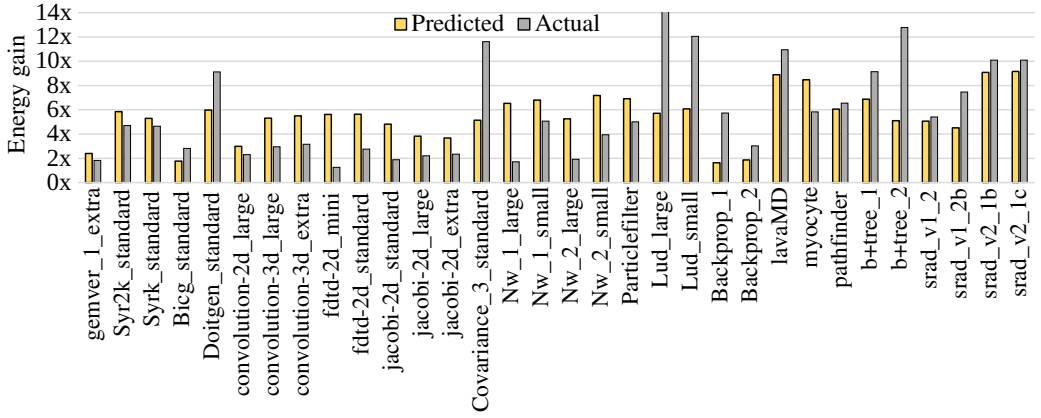
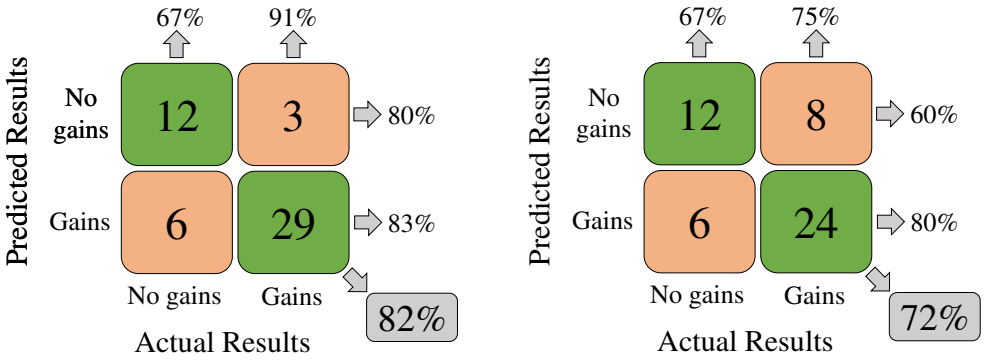


Figure 33: Energy Gains Prediction results for Nvidia Jetson Xavier NX



(a) Dynamic Analysis

(b) Static Analysis

Figure 34: Predicted vs. actual energy consumption gains class - Server (Xeon Gold 6138 - Tesla V100)

5.4 Energy-aware Placement on Edge resources

Resource management is a key concept of modern IoT application development, that hat has been attracting the interest of many researchers over the years. While there is a plethora of provided solutions, selecting the best resources at the application design time, can lead to significant energy savings. Exploiting the information retrieved by the proposed energy estimation models (see Chapter 4) can be very beneficial towards an energy-efficient application functions placement that also exhibits high performance. In this Chapter, we demonstrate a proof-of-concept of the potential use of the methods introduced in the context of this thesis on the use-case of building an energy-aware function placement tool for IoT applications.

The majority of approaches that can be found in the literature focus mostly on performance, often neglecting the impact on energy consumption. While more recent approaches consider multi-objective optimization solutions, most of them formulate and solve a problem using only simulations, abstractions or by utilizing traditional IoT communication frameworks and exploring candidate solutions on the actual targeted devices [154], [155]. Nowadays, serverless computing introduces the Function-as-a-Service (FaaS) paradigm, where short-lived functions are also placed on Edge computing resources [156]. A large number of recent research works propose custom frameworks to optimize function placement [157], [158] [159].

In this Section we present our vision towards using the introduced models to build a software analysis tool to be used by software developers in order to select the placement of the application's individual functions on Edge environments. The resulted placement will take into account both energy consumption and quality-of-service (QoS) at design-time, before running the code on the targeted devices.

The overview of the presented solution is illustrated in Figure 35. The introduced tool consists of the following steps: Initially, the tool parses the source code, which corresponds to the user's application code, as well as the list of the available devices and estimates the potential energy and time of executing each function on each of the devices, using the methods described in Chapter 4. The output of this task is forwarded to the multi-objective optimization stage, where the function-to-device mapping, that minimizes both energy consumption and execution time is resulted. A pre-request for this step is a user input that corresponds to a value ($[0, 1]$) that defines the preferences (focusing more on energy or on execution time).

In the context of this study, the output of the aforementioned procedure will be given (by the developers) to the Kubernetes container orchestrator system of the IoT/Edge cluster (see Section 3), which is responsible for

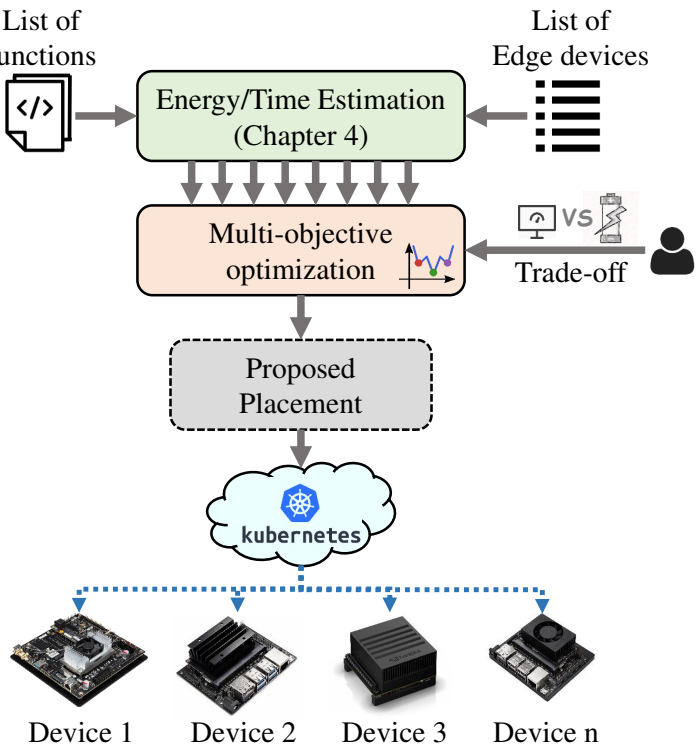


Figure 35: Overview of Energy-aware function placement on Edge resources

placing the functions on the resources that were determined by the multi-objective optimization heuristic.

5.4.1 Cross-device Energy/Time estimation

The methods described in Chapter 4 were used in this stage. More specifically, we followed the Extensibility procedure to add all the devices of the Edge network. We used the dynamic analysis methods due to their increased accuracy, their capabilities of estimating the energy of the entire application (without needing to give additional information such as the number of loop iterations) and because they are based on dynamic instrumentation of the execution and thus they give us the opportunity to analyse more languages such as Python which is used in modern serverless environments. Finally, we extended the introduced methods to make predictions also for execution latency, by changing the energy values of the datasets with execution time values. We do not provide analytical results for this procedure because the purpose of this Section is to show the importance and the potential use (as a proof-of-concept) of the methods introduced in this thesis on building an

energy-aware function placement tool that can be used by developers at application design time.

5.4.2 Multi-objective optimization

After predicting the energy consumption and the execution time of each function (f_i), per candidate device host (d), a multi-objective solver based on differential evolution (see Section 3) follows. This solver is responsible of calculating a function placement strategy ($\mathbf{p}$, where p_i determines the device where function f_i is placed) that optimizes the total energy consumption (E) and the total time (T). Finally, a weighing factor tr defines the relative importance of optimizing either the energy cost or the execution time according to the designer's (user's) priorities. Based on this value our method's solution might lead to the maximum energy savings ($tr = 1.0$), the lowest execution latency ($tr = 0.0$), or any trade-off between these two border scenarios as described in Equation 8.

$$Minimize : tr \times E(\mathbf{p}) + (1 - tr) \times T(\mathbf{p}) \tag{8}$$

The aforementioned total energy cost is formulated in Equation 9. More thoroughly, we assume a factor α (0,1) that makes a simple model of the functions interference. The total energy is expected to be less than the consumption of a fully serial execution of all the functions (E_{serial}) that would be estimated by summing the individual estimations. On the other hand, the total energy is expected to be much higher than the energy of the ideal case (E_{ideal}) of running all the functions in parallel consuming the same amount of energy as the most consuming function (without additional energy).

$$\begin{aligned} E(\mathbf{p}) &= \alpha \times E_{serial}(\mathbf{p}) + (1 - \alpha) \times E_{ideal}(\mathbf{p}), \\ &\qquad where \\ E_{serial}(\mathbf{p}) &= \sum_{i=1}^{i=n} E_{est}(f_i, p_i), \\ E_{ideal}(\mathbf{p}) &= \sum_{\forall d} (E_d),\ E_d = max(E_{est}(f_i, d)),\ where\ p_i = d \end{aligned} \tag{9}$$

In order to calculate the total execution time, we followed a similar procedure as it can be shown in Equation 10. For the worst case scenario, corresponding to the fully serial execution per device, total time (T_{serial})

equals to the time of the device that completes the execution last (because we have only the parallelization between devices). For the ideal case, all the functions are executed in parallel perfectly and the total time corresponds to the latency of the slowest function.

$$
\begin{aligned}
T(\mathbf{p}) &= \alpha \times T_{serial}(\mathbf{p}) + (1-\alpha) \times T_{ideal}(\mathbf{p}), \\
&\qquad\qquad where \\
T_{serial}(\mathbf{p}) = max(T_d),\ T_d &= \sum_{\forall i,\ where\ p_i=d} T_{est}(f_i, d) \\
T_{ideal}(\mathbf{p}) &= max(T_{est}(f_i, p_i))
\end{aligned}
\tag{10}
$$

5.4.3 Placement example

In order to demonstrate the solution presented in this Section, Figure 36 illustrates four examples of our solver's outputs for analysing an application decomposed into ten individual functions. In those four scenarios, we are gradually decreasing the trade-off value to get better execution latency.

Starting from the first scenario, by setting trade-off to 0.9 we focus more on energy and, as a result, the proposed placement utilizes only two of the available devices, to achieve a lower energy consumption. In the second scenario (Figure 36b), our solver alters the function placement to achieve better execution latency. More precisely, device #1 is utilized, while functions are more equally balanced between devices #2 and #3. The additional computing power increases the consumed energy. By further reducing the trade-off (Figure 36c), the heuristic solver changes the mapping of functions to the available devices, e.g., selecting a function with higher computing demands to be executed on the device #1. Finally, in the scenario with the most demanding execution time limit (Figure 36d), there are two functions mapped to device #1, and the energy consumption is the highest amongst all four scenarios [160].

5.4.4 Experimental Results

In this paragraph, we present some first experimental results that validate our assumptions and illustrate that our vision towards using the introduced models and tools for supporting energy-aware function placement can be successful.

5.4.4.1 Experimental Setup The introduced solution is designed to serve the needs of developing energy-aware IoT applications targeting power-

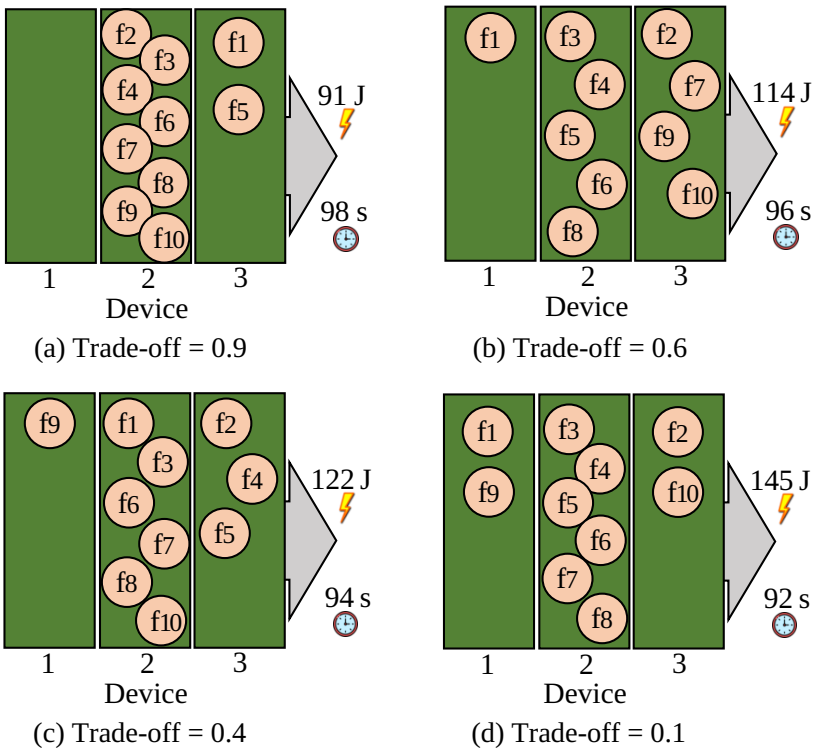


Figure 36: Output example: Different function placement scenarios

constrained Edge clusters. Therefore, while the proposed method runs on the user's personal computer (as also mentioned in Section 4.1), for the evaluation we used four low-powered embedded devices (see also Section 3.4). More precisely, we place CPU functions on NVIDIA Jetson TX1, NVIDIA Jetson Nano, NVIDIA Xavier NX and NVIDIA Xavier AGX. All the devices include a built-in power monitoring sensor that we used for measuring energy consumption. Regarding the applications, we used 25 benchmarks/functions from the widely-used *scikit-learn* python library, that is very popular in machine learning applications.

5.4.4.2 Evaluation and Discussion Figure 37 depicts the total time and total energy results of the placements proposed by the introduced method for various trade-off values, as well as the average results of letting Kubernetes decide the placement (denoted with the blue Kubernetes logo). According to this analysis we conclude that our solution that takes into account the estimated energy and time can achieve up to 43% Energy Savings, by setting trade-off value to 1.0 ($tr = 1.0$) with a cost on the QoS and up to 32%

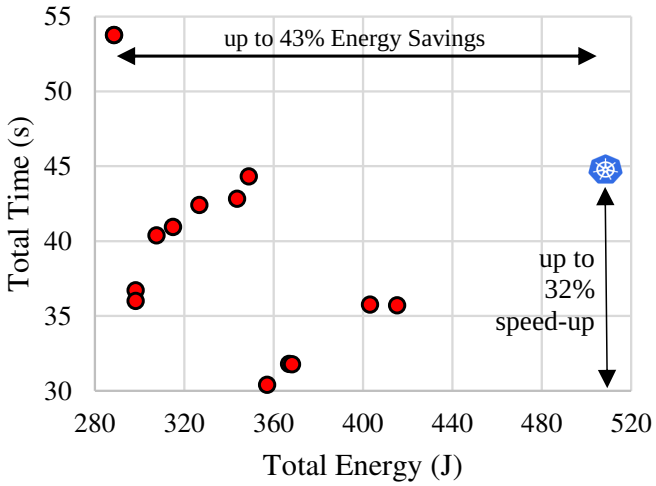
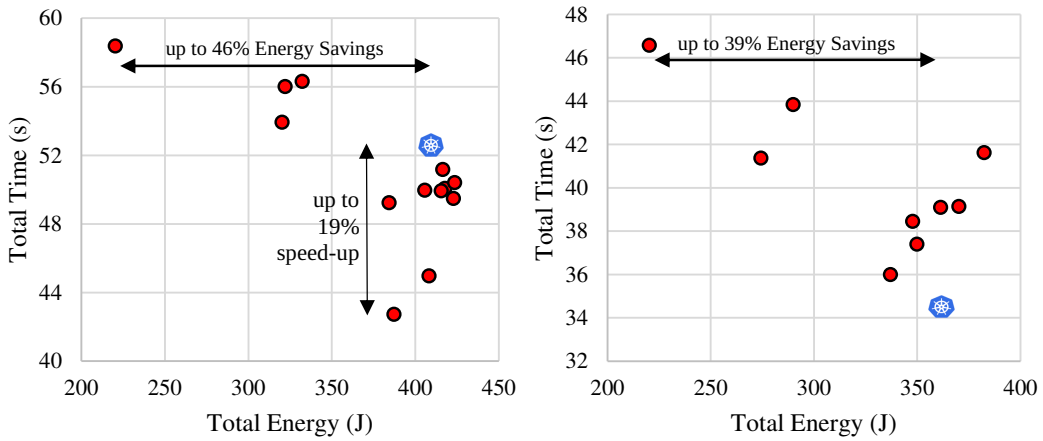


Figure 37: Placement results using 4 Edge devices against the average results of default Kubernetes placement

less latency for $tr = 0.0$, while also consuming less energy. On average the proposed solution reduces the energy consumption by 33.6%, achieving also a 11% of speed-up.

The results are very promising and are expected to improve further for using more Edge devices. We observe that our method does not use some devices especially in the cases that energy is of most importance. For example, TX1, while consuming less energy, is the slowest device and porting heavy functions on that may lead to larger energy overhead. On the other hand, Kubernetes default strategy tries to distribute the functions as much as possible on the available resources. Sometimes placing more functions on Nvidia NX and less to AGX leads to better energy results and although such a placement selection doesn't exploit as much as possible the available computing capabilities, it doesn't have a large overhead on the execution time. Therefore, in the following experiments we try to stress our solution even more by using only 3 devices, forcing (this way) also our method to use all devices for most cases (trade-off values).

Figure 38a illustrates the results of using Jetson TX1, Jetson Nano and Xavier AGX. Based on these results, we might claim that by using the introduced method, the users can choose between a number of available solutions, according to their preferences determined by the trade-off value. By selecting to optimize only energy, up to 46% energy savings can be achieved, while focusing only on execution time offers a speed-up of 19% compared to the Kubernetes placement. Also, the provided solutions consume on average 7.5% less energy compared to the Kubernetes placement.



(a) TX1, Nano, Xavier AGX (b) TX1, Nano, Xavier NX

Figure 38: Placement results using three devices against the average results of default Kubernetes placement

In the experiment presented in Figure 38b, we replaced the Nvidia Xavier AGX, with the Nvidia Xavier NX that incorporates a similar CPU with 4 cores instead of 6. Here the Kubernetes placement achieves the faster results, while we can observe three placements provided by the introduced framework that give sub-optimal results, meaning that both energy and time are worse than in the pure Kubernetes case. However, we can see that changing the trade-off value still provides a compromise between time and energy, achieving up to 39% energy savings for $tr = 1.0$, while the provided solutions reduce the energy 9.6% on average compared to the Kubernetes placement.

According to the presented results, we conclude that the methods proposed in this thesis can be very beneficial for building a solution that provides the software developer with placement solutions that minimize the energy consumed and/or the execution time of Edge computing clusters, subject to a user-defined trade-off value.

5.4.5 Limitations

As mentioned in the beginning of this Section, the proposed placement solution employs the methods proposed in the rest of this thesis to perform function-level profiling and, by using the energy consumption and execution latency predictions, to propose efficient function placement on Edge devices. So, we give the directions for the development of another type of energy optimization tools that can be easily integrated in the overall framework proposed in this dissertation. The first results are very promising giving

a proof-of-concept for this solution. However, there is still a number of limitations summarized bellow:

- The proposed solution does not take into account data exchanges between the function calls, meaning that there is no modelling of the communications that have a large impact on the execution time of the application and the energy of the Edge cluster.
- The proposed method assumes that the Edge devices are dedicated to run only the application under analysis. This means that it does not consider the case when other applications are running in parallel (traffic).
- One might claim that the comparisons in the presented experiments are not fare enough due to the fact that the pure Kubernetes solution also runs a scheduling algorithm on the run-time to select the final placement.

5.5 Software Engineering Perspective - Design time quality

The rapid evolution of the embedded systems and IoT applications increases the requirements for long lifetime expectancy and the demand for maintainable software products [6]. However, the source code optimizations that developers apply to improve performance and energy may affect the maintainability [161] [162]. In other words, employing techniques (presented in Chapter 5) to improve the energy consumption (runtime quality) may have a negative impact on the design time quality, such as maintainability, reusability and programming effort [163] [164]. In this section we investigate the impact of the proposed transformations for improving energy efficiency on software quality metrics and we make a first step towards estimating the programming effort required to develop optimized code (and more specifically CUDA acceleration) based only on the analysis of the initial corresponding CPU code, by using well-established indicators. We would like to inform the reader that for the work presented in Section 5.5 we closely collaborated with the Department of Applied Informatics, University of Macedonia[22], Thessaloniki, Greece that suggested metrics and tools for measuring design quality, with the author's contributions being on applying and studying the impact of energy transformations on maintainability metrics (Section 5.5.1), designing

[22]We would like to thank Prof. Alexander Chatzigeorgiou, Prof. Apostolos Ampatzoglou and Angeliki-Agathi Tsintzira

and testing (dataset, experiments) models to predict the programming effort (Section 5.5.2).

5.5.1 Impact on Software Maintainability

Cache blocking and acceleration optimization presented in Sections 5.2 and 5.3 correspondingly, are applicable in the selected applications from the Polybench and Rodinia benchmark suites. After applying the relevant refactorings proposed by the previously described energy optimization methods, we evaluate the impact on Software Maintainability. More precisely, for this analysis we chose to use the Lines of Code and the Cyclomatic Complexity as the simplest and oldest Maintainability indicators used in the literature [165] [166] [167]. We should mention here that the focus of this work is not based on the use of the most accurate maintainability metrics, but rather on the study of the impact of the aforementioned energy optimization techniques on maintainability and (as we will discuss in Section 5.5.2) on the design of tools that predict maintainability metrics that express the programming effort required to develop the new version of the code, which is optimized in terms of energy.

Typical examples of cache blocking (or data-flow) optimizations are the loop transformations that aim to improve the cache performance, by improving data locality to reduce the overhead of the loops. Loop tiling method is the most important and widely-used loop transformation. It ensures that more data are reused by the cache, by breaking the iteration space of a loop into smaller blocks. Thus, loop tilling increases the loop depth and partitions large arrays into smaller blocks, in order to fit the accessed array elements into the cache size and reduce D-cache misses (see Section 5.2).

In the context of this analysis, we analysed several applications from the Rodinia and Polybench benchmark suite, using tools designed in the SDK4ED EU project. In cases where the energy optimization tools propose the use of the Cache blocking refactoring, we applied the optimization manually. The majority of the applied optimizations are loop tiling. Then, the updated applications were analysed in terms of Maintainability, measuring the total Lines of Code as well as the Cyclomatic Complexity. Figure 39 shows the increase of the two maintainability indicators between the two versions of the application. As expected, in all cases the Lines of Code as well as the total Cyclomatic Complexity increase. The increase varies from 10% to 105% for Complexity and from 7% to 50% for Lines of Code.

Offloading the heavy parts of the applications to accelerators is a process that needs a lot of source code transformations and additional source code, a fact that usually has a negative impact on Maintainability. Following

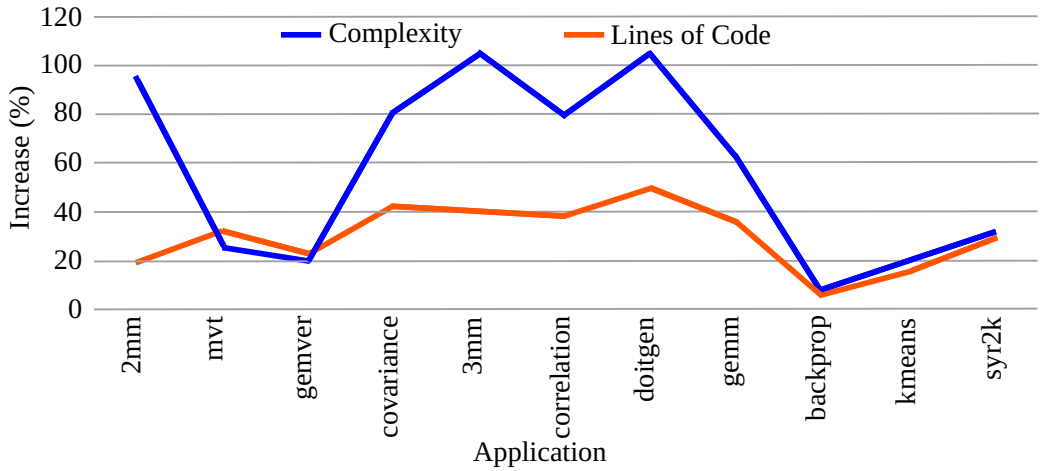


Figure 39: Cache blocking impact on Maintainability (applied on applications from Rodina and Polybench suite)

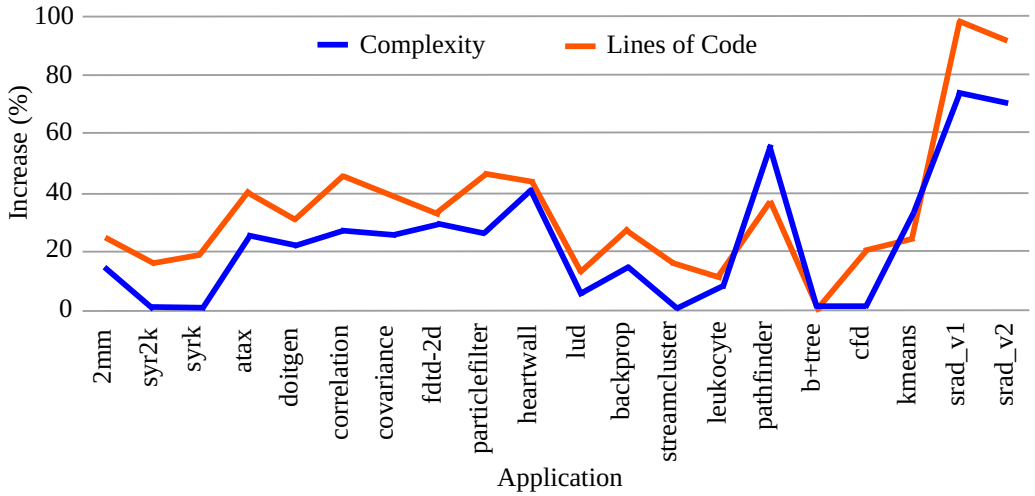


Figure 40: Lines of code and Cyclomatic complexity increase after applying GPU acceleration

the same procedure, we measure and compare the Lines of Code as well as the Cyclomatic Complexity of the CPU-only vs CPU+CUDA version of applications from Rodinia and Polybench benchmark suites. The results are shown in Figure 40.

As shown in Figure 40, Lines of Code are increased from 1% up to 100% while there is an increase of 1% up to 78% for Cyclomatic Complexity. The range of the impact is from slightly to severely negative. The large number of changes (even to the algorithm itself) for applying acceleration in some

applications leads to high increase while other projects require simple restructuring (for example, in the case that the for-loop is already written in a way that is parallelizable and the kernel is easy to be developed). For a few cases, the CPU-only version of the application was already very complex and the refactorings did not lead to more complex code.

The presented results give us a coarse-grained analysis of the impact of cache blocking and acceleration energy optimizations on software Maintainability. However, the experiments show very different effects (from small up to severe impact). Therefore, although we might conclude that there is a clear negative impact on maintainability, we also think that further analysis is needed.

In this direction, we will try to make a more fine-grained characterization of the trade-off between energy and maintainability and, more specifically, to highlight the project-related metrics and characteristics that may affect the observed impact. For this purpose, we divide the Rodinia and Polybench cases into two classes. The first includes applications in which the refactorings seem to have a slightly negative impact: less than 25% increase in Cyclomatic Complexity and less than 30% increase in the total number of Lines of Code. The second category includes all the applications that lead to a severe trade-off for applying the two energy-related optimizations. One could argue that 30% cannot be considered as a slight impact, but we decided to split the two sets of applications in the middle. In addition, all these applications are relatively small and therefore more "sensitive" to small changes in the number of lines. Figure 41 illustrates the number of the initial Lines of Code and the number of times the refactoring is applied to the application. Based on these results, we might claim that a low impact (slightly negative) is observed when the application has more lines of code or if the refactorings are applied 1 or 2 times. On the other hand, when the initial number of Lines of Code is relatively small or the refactorings are applied many times, we have a greater impact.

These results highlight that the size of the impact is related to the size of the application under analysis and the number of the times that a refactoring is applied. However, we can find some applications that belong to the first class that may have a larger number of applicable refactorings or some others that, although only one instance of the refactoring is applied, it leads to a high impact. Figure 42 shows the number of the specific lines in the initial application that are going be refactored (or the number of lines of the energy hotspot according to the definition presented in Section 5.1). This analysis indicates that the number of lines that will be changed during the optimization application is also related to the size of the impact that this refactoring will have on Maintainability. More lines are refactored in the

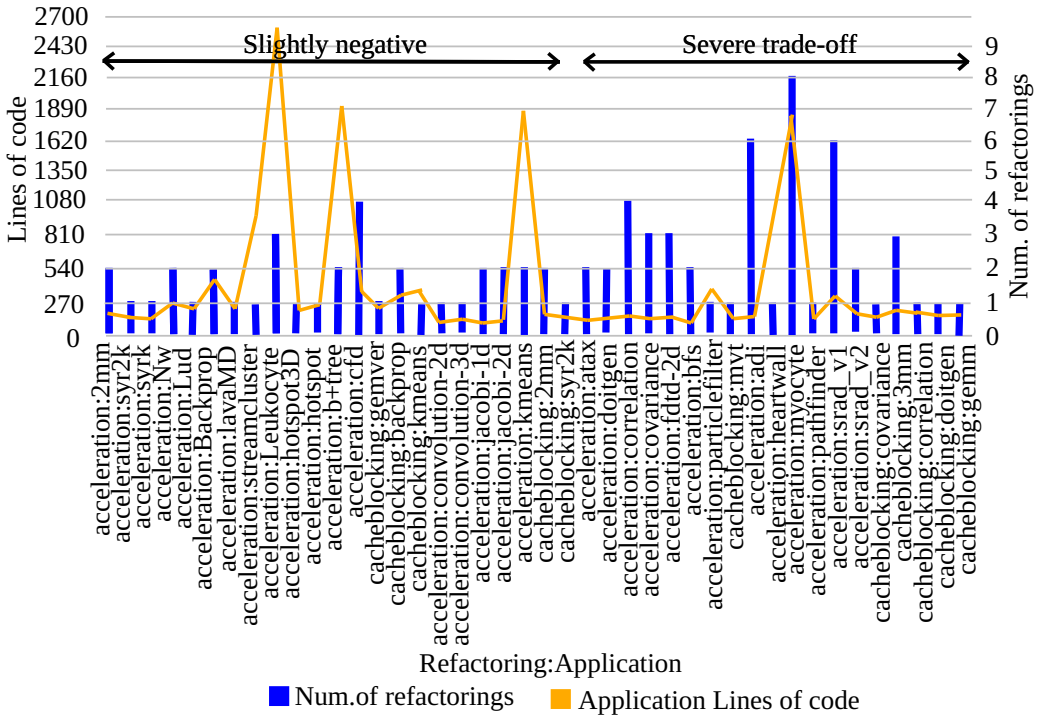


Figure 41: Impact on maintainability: Number of applied optimizations and initial application lines of code

cases that although we applied more than one refactorings, a small impact is observed. On the other hand, when the initial version has fewer lines in the hotspot that will be rewritten, the lines are expected to be increased, making the updated version less maintainable.

In order to study the way that the aforementioned characteristics affect the final maintainability and to make a more formal study, Figure 43 illustrates the correlation (scatter plot) between Lines of Code of the final refactored application with the initial Cyclomatic Complexity, the number of refactorings, the initial Lines of Code as well as the number of lines belonging to the hotspots that will be refactored. Scatter plots show how much one variable is affected by another (linearly). The R value also presented in the figure quantifies how strong the linear relationship between the two variables is:

- $R = 1$ perfect positive linear relationship, $R = -1$ perfect negative linear relationship
- $R > 0.7$ strong positive linear relationship, $R < -0.7$ strong negative linear relationship

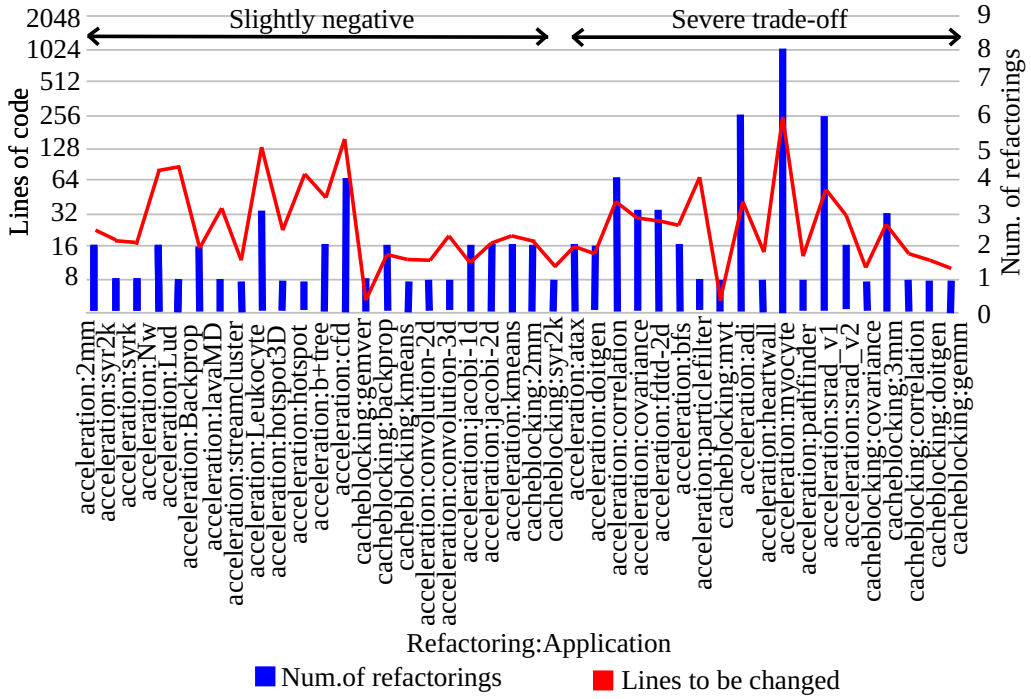


Figure 42: Impact on maintainability: Number of applied optimizations and lines of code to be changed

- $R > 0.5$ moderate positive linear relationship, $R < -0.5$ moderate negative linear relationship
- $R > 0.3$ weak positive linear relationship, $R < -0.3$ weak negative linear relationship

Based on these results, we can observe a clear correlation, concluding that the impact of the energy-related refactorings depends on the application under analysis. More specifically, the relationship of the impact with the initial Complexity and Lines of Code is very strong positive ($R > 0.7$), while there is a considerable relationship with the number of refactorings and the lines that will be refactored. Note that we do not present a more detailed analysis regarding the Cyclomatic Complexity (in the way presented in Figure 41 and Figure 42) as the results are very similar.

5.5.2 Programming Effort Estimation

Although the energy optimizations proposed in the context of the present dissertation are widely used (usually for targeting performance), existing

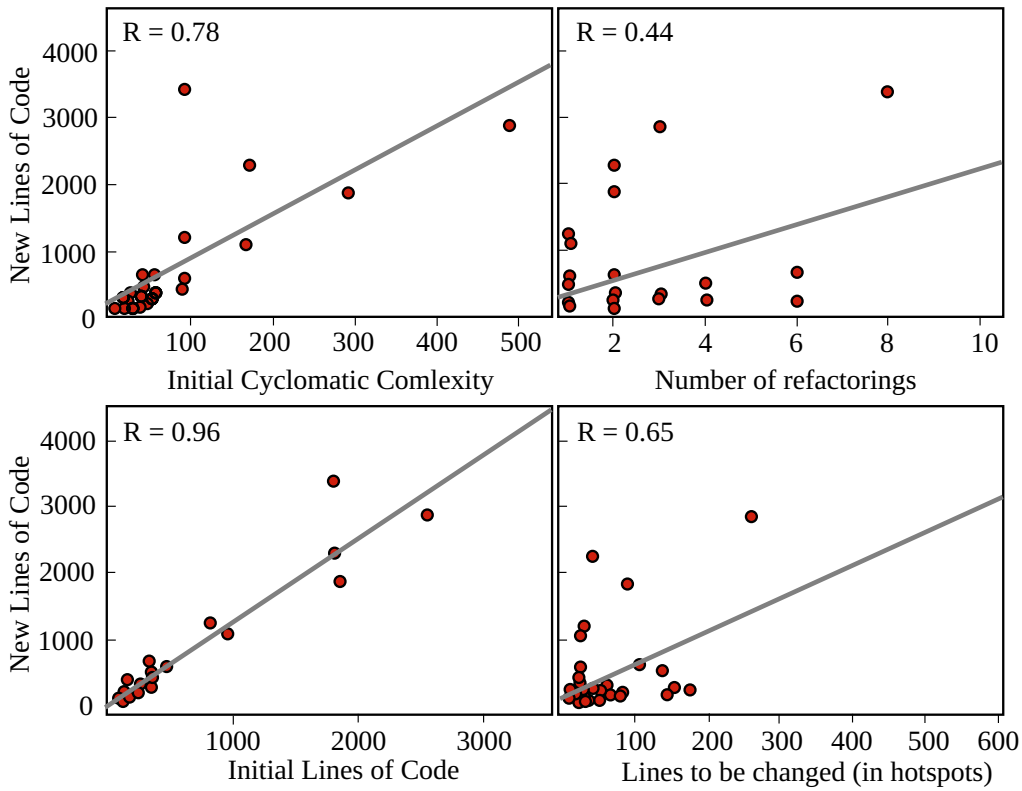


Figure 43: Correlation of initial program characteristics with the final lines of code

work in the literature does not take into consideration the programming effort that is required to refactor existing code in order to achieve the desired performance or energy gains. There are only a few attempts to quantify the effort of accelerating applications. These are either based on empirical investigations [168], or rely on relatively simple metrics, such as the Lines-of-Code (LoC) of accelerator-specific code (i.e. CUDA, OpenCL) versus the LoC of the corresponding CPU code [169].

Aside from the utilized metric, the most important thing here is that, to the best of the author's knowledge, *there is no approach towards designing a tool that estimates programming effort of developing GPU code using only the CPU code as input*. Existing work only measures the effort of existing code, while it is very important to know the effort required before writing the new code.

The first type of optimization presented in Section 5.2 concerns *Data-flow* optimizations with particular emphasis on loop transformations. So, one might wonder why we are only talking about GPU acceleration. There are

two reasons: First of all, loop transformations usually require small changes inside the loop. If a complete loop reconstruction is not required, then by following the examples in Table 5, the developer can easily become familiar with this type of optimizations. The second reason is that there are tools that help the developer in this direction. In our study, we used Pluto[23] extensively. Pluto uses the polyhedral model and transfers the compiler optimizations for data locality and parallelization up to the software level (source to source) [144] [145]. It is considered easy to use, while the user can choose which optimization to perform in which loop (with special *pragmas* in the source code and flags in the command line) or perform an automatic optimization. Other tools, such as Orio[24], also based on the LLVM infrastructure, offer heuristic optimization methods (e.g. for selecting the best tile size). Regarding *Platform Selection* and *Placement* optimizations, they do not require direct changes to the application software.

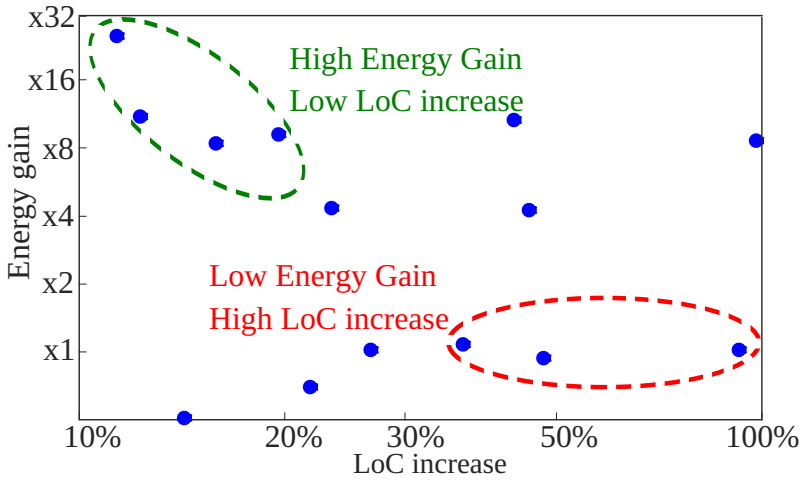


Figure 44: Energy gains vs LoC increase by GPU acceleration in applications of the Rodinia Benchmark Suite

Figure 44 shows how GPU acceleration on a set of applications of the Rodinia suite [126] affects the final energy consumption, as well as the Lines of Code. The vertical axis corresponds to the energy gains by GPU acceleration measured on NVidia Jetson TX1 embedded heterogeneous platform [9] (see Section 3), while the horizontal is the percentage of LoC increase. According to these results, we might notice that some applications require minor effort to be ported on GPU, while the energy gains are very high. Also, there are applications that need significant effort to be accelerated, but with relatively

[23] `http://pluto-compiler.sourceforge.net/`
[24] `http://brnorris03.github.io/Orio/.`

small gains in energy. Especially for the applications belonging to the last category, developers may decide not to invest in GPU acceleration. As a result, tool support for predicting not only energy gains (Section 5.3) but also programming effort is expected to significantly contribute to an effective investment of programming time and effort of IoT applications.

As mentioned before, the first thing we have to do is to choose the metric that we will use for the quantification of programming effort. Beyond the old LoC metric that sometimes may underestimate the amount of programming effort [170] we also used the Halstead's metrics [171] [172]. These metrics quantify both the number of distinct operators and distinct operands (their sum corresponds to the program's *Length* (N)), as well as the total number of occurrences of operators and operands (their sum corresponds to the program's *Vocabulary* (n)). The *Volume* of a Program ($V = N * log_2 n$) expresses the minimum number of bits required to represent all operators and operands [173]. Any piece of code, could be theoretically written in its most abstract form as a hypothetical function that has the same functionality. In this sense, the minimal volume for any program can be defined as $V^* = (2+n_2^*)log_2(2+n_2^*)$ and the ratio of the minimal volume over the actual volume is called *Level* $L = V^*/V$, expressing the abstractness of a program. *Difficulty* is the inverse of the program level, while the ratio of volume over the level represents a metric called *Effort* ($E = L/V$). This metric quantifies the effort needed to write or understand a program as it considers both its actual size and its abstractness [174] [171]. Halstead metrics have been used to measure both maintainability [175] and the development effort in high-level parallel programming approaches [176]. More specifically, according to these studies, there is evidence that they are capable of estimating the time spent on writing an existing source code.

It is worth mentioning here that the main focus in the context of the present dissertation is not based on the use and monitoring of the Halstead metric itself, but rather on the design of a tool that predicts the effort required to develop a new (accelerated) version of the code, *before development*. In contrast to related approaches that quantify the programming effort of already written code, in the presented analysis, the *LoC* and the *Halstead's Effort* is estimated using only the initial CPU code as input, in order to support design-decision for developers with respect to acceleration. Other metrics that capture progamming effort can be also supported by the introduced method.

This type of analysis is very important for energy hotspots (see Section 5.1) that are predicted to have energy gains if offloaded on a GPU accelerator (see Section 5.3). Capturing the required effort to develop the new GPU code (e.g. CUDA, OpenCL) would be useful for the evolution of a software project.

The programming effort prediction is based on the following procedure: Having as a baseline the effort (expressed using the Halstead's *Effort* or the simpler LoC) required to develop a part of the CPU code, we predict the percentage of extra effort (the increase of the value) required to develop the accelerator-specific code. This metric will serve as indicator of the programming effort required to develop the accelerated version of the code. The programming effort predictions are made by a regression model. The model receives as input the values of the features listed below (captured on the initial CPU-version of the application):

- The number of hotspots
- LoC (Lines of Code) of the application
- The number of hotspots' LoC
- The number of hotspots' statements
- The number of distinct and total operators
- The application's Complexity
- The application's Volume
- The application's Length
- The application's Difficulty

Table 7: Importance of the selected features in terms of relation to Programming Effort to produce the accelerated version (stepAIC criterion)

Features	p-value
Number of hotspots	0.0707
LoC of application	0.0219
Number of hotspot's LoC	0.0219
Number of hotspot's statements	0.0129
Distinct operations	0.0238
Total operations	0.0308
Complexity	0.069
Volume	0.0362
Length	0.0231
Difficulty	0.0289

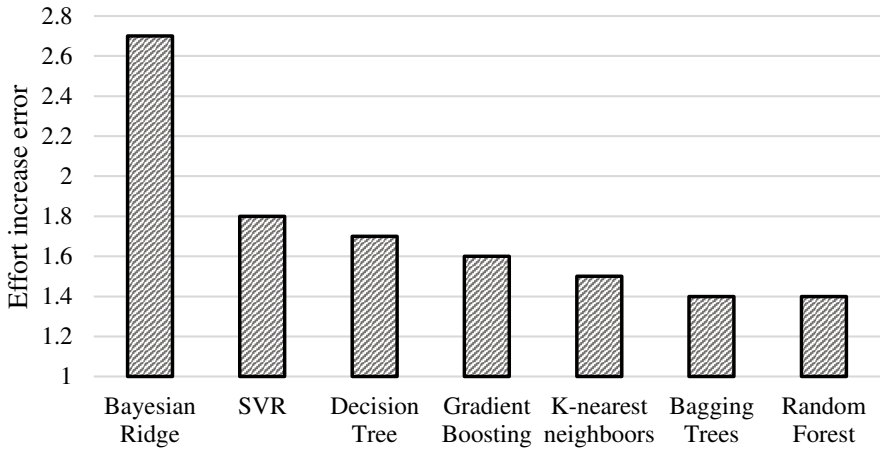


Figure 45: Effort prediction accuracy comparison of various regression models

Table 7 quantifies the importance of each of the selected features regarding the programming effort, after applying the stepAIC criterion similarly to the method used in Section 5.3 for estimating the energy gains. The feature metrics are collected by analysing the source code. Open-source tools can be used for this purpose such as the SonarQube platform[25] and the Halstead Metrics tool[26].

The dataset used for programming effort includes hotspots from Polybench and Rodinia benchmark suites. We did not use any synthetic applications, since only for applications developed by programmers it makes sense to evaluate the programming effort. As regards the selected estimation model, Figure 45 presents the 7 best models. Based on these results, we can conclude that that Random forest regression achieves the highest accuracy. The prediction accuracy of the programming effort increase of developing CUDA code compared to the corresponding CPU is shown in Figure 46. The average absolute error (i.e. *Effort*-increase difference between predicted and actual values) is 1.4×. The accuracy is very high for the applications from the Polybench benchmark suite (around 93%). However, few inaccurate predictions are observed in some applications from Rodinia. The reason is the fact that these use-cases are more complex compared to the Polybench applications, including more changes from version to version than the necessary. Interestingly, we can notice that for 3 applications the effort increase seems to be less than 1×. The reason is the fact that the CPU versions of these applications provide functionality that is not present in the corresponding GPU version. Therefore, the GPU version required less effort to be developed than the

[25]https://www.sonarqube.org/

[26]https://sourceforge.net/projects/halsteadmetricstool/

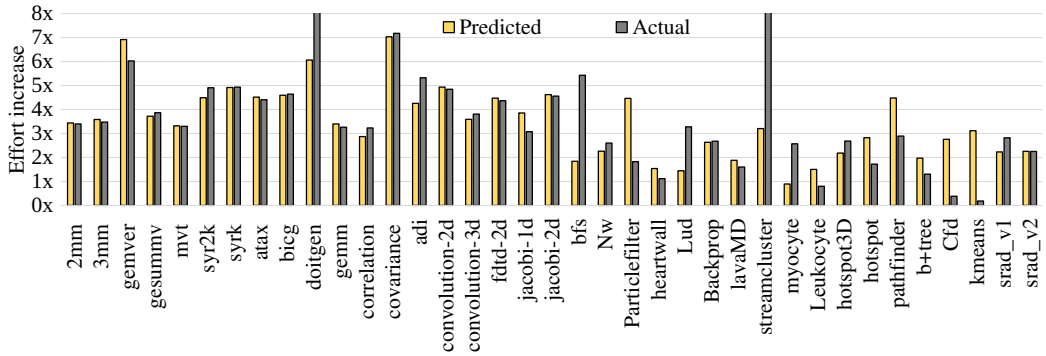


Figure 46: Predicted vs. actual programming effort increase using regression analysis

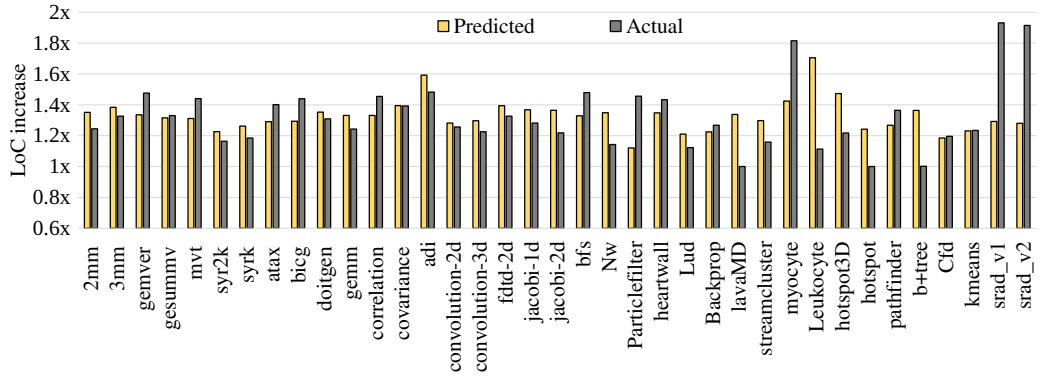


Figure 47: Predicted vs. actual LoC increase using regression analysis

CPU version. Nevertheless, we decided to maintain the original structure of the applications, without making any modifications. Due to these cases, we employ the median absolute percentage error (MdAPE) approach, which is less sensitive to outliers. According to this analysis, we have an error less than 15% (85% of accuracy).

Although we mentioned that the LoC metric may not be a good estimator of the programming effort, it would be interesting to use it in our predictors as a more traditional metric. In this regard, Figure 47 shows the accuracy of using the proposed models to predict the CUDA code development effort expressed in LoC increment. The prediction accuracy is considered high (87.3%).

After comparing the actual values of the effort expressed using the Halstead's Effort metric as well as the simple LoC increase (Figures 46 and 47), we might confirm that by using advance metrics, such as the Halstead's *Effort*, the programming effort required for developing CUDA code may be

captured more accurately than using LoC. We notice that *Effort* ranges from $\times 1$ to $\times 8$, while LoC ranges from $\times 1$ to $\times 2$. However, in many cases the increase in LoC can be safely attributed to statements such as definitions and variable initialization, which cannot be directly related to programming effort. On the other hand, *Effort* is mainly affected by the number of operands, operators and function calls. Therefore, programming effort is more accurately expressed through *Effort*. Indeed, in Figure 47 we observe that there are some applications in which there is a very small LoC increase, but the effort increases by more than $\times 4$. This observation is in line with Shihab et al. that state that relying in LoC often leads to underestimation of programming effort [170].

5.5.3 Combination of Programming Effort Estimation and Acceleration Gains Prediction

The analysis of the programming effort presented above, inspired us to propose a final methodology that combines the effort with the prediction of energy gains described in Section 5.3 in a single tool-flow that aims to help developers investigate the GPU acceleration of their applications [172].

The proposed methodology is depicted in Figure 48. The input of the methodology is the source code of the CPU application and the output is the predicted energy gains by acceleration, as well as the programming effort needed to develop the CUDA version of the CPU application code. The methodology consists of the following steps:

- Hotspot identification (Section 5.1)
- Prediction of acceleration gains by static analysis (Section 5.3.2)
- Prediction of acceleration gains by dynamic analysis (Section 5.3.3)
- Programming effort prediction (Section 5.5.2)

The output of the methodology is shown in Figure 48. A CPU application under analysis is classified into one of the three categories: "No gains", "Moderate" or "High gains" with respect to the predicted energy consumption gains by GPU acceleration. Fine-grained energy consumption prediction, along with programming effort prediction is provided for the applications classified into the "Moderate gains" category.

The methodology reports the prediction of the programming effort required to develop CUDA code, only for the applications classified into the "Moderate gains" category. We make the assumption that for applications

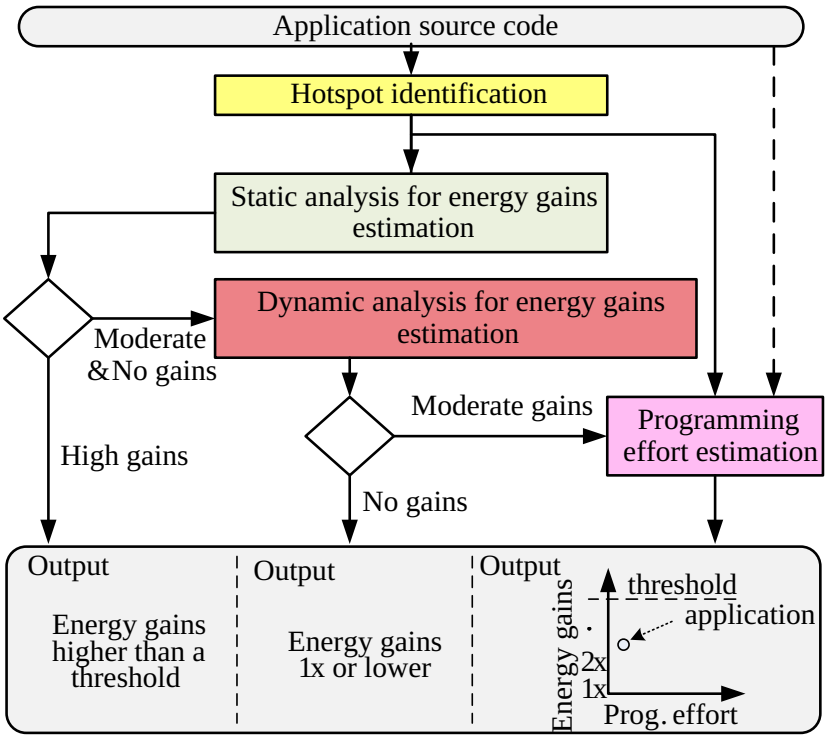


Figure 48: Overview of the proposed methodology

classified into "High gains", the CUDA version will be developed no matter how much programming effort is required. However, for applications classified into the "Moderate gains", programming effort may affect developers' decision about developing CUDA or not, especially if the predicted energy gains by the regression step are relatively low.

For the moderate gains, the output of the proposed methodology is depicted in Figure 49. The Figure shows the predicted energy gains on Tegra X1 (see Section 3) vs. the predicted *Effort* required to develop the GPU version of each application in comparison to the corresponding CPU version. Each point corresponds to a single hotspot. 55 hotspots were classified into the "Moderate gains" category, while 30 were classified into "High gains" and for 15 hotspots "No gains" were predicted.

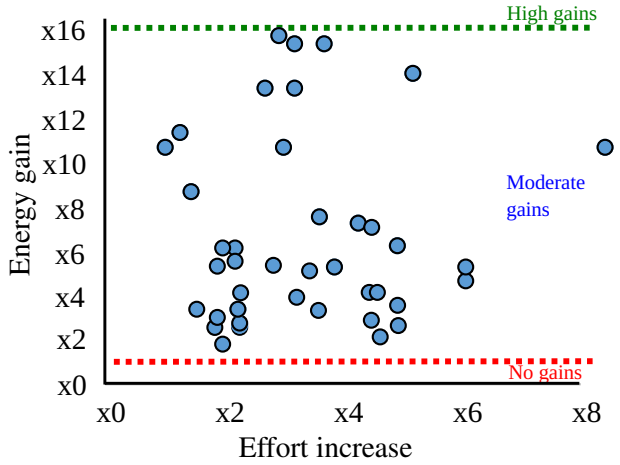


Figure 49: Output of combining the programming effort estimation with the acceleration gains prediction methodology for the Polybench and Rodinia hotspots that are classified into the "Moderate gains" category.

5.6 Implementation of Energy-aware Software Analysis Tools

In this Section, we describe the implementation of an Energy-aware Software Analysis Toolbox that incorporates the methods described in Chapter 4 as well as in Sections 5.2 and 5.3. We clarify that this work is part of the European Union's Project SDK4ED[27] [177] [178]. The thesis author's contributions are on the implementation of the Energy Toolbox backend's services[28], as well as on the structure of the Energy Toolbox front-end based on the template of the project. The usability of the introduced solution is demonstrated using a real-life application designed by Neurasmus[29].

5.6.1 Back-end micro-services

The architecture of the framework back-end is depicted in Figure 50. More specifically, the central (Manage Energy Consumption) component consists of two basic parts that correspond to the sub-components of the methodology presented in Figure 21. The first one is responsible for monitoring and

[27] `https://sdk4ed.eu/`, funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 780572

[28] The design of these components were done together with Christos P. Lamprakos and Kostantinos Salapas from National Technical University of Athens - Microlab

[29] `http://www.neurasmus.nl/`, the author would like to thank Assoc. Prof. Christos Strydis (Erasmus MC) for providing the application

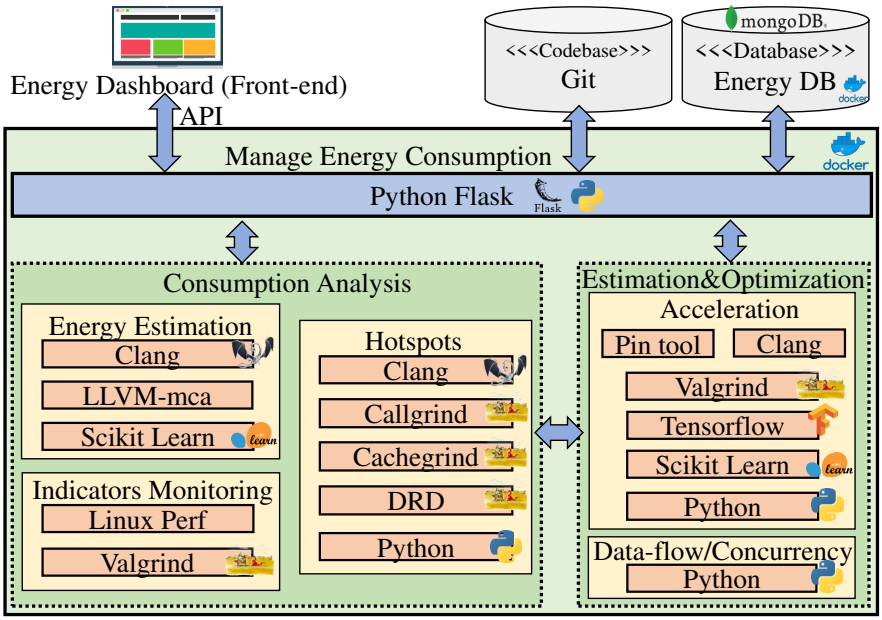


Figure 50: Energy analysis framework back-end tools

estimating the Energy Consumption, reporting indicators and identifying the most energy consuming parts of the code (hotspots). Developers should focus on these code blocks to potentially reduce the energy consumption or improve the performance of the executed operations. The latter (Estimation & Optimization) aims at suggesting the proper source-code transformations that will potentially reduce the energy consumption of the operations executed in this block. The functionality of these mechanisms is described in Chapter 4 and the rest of Chapter 5.

The framework is implemented as a Docker Container, while the entry point that receives requests through APIs and returns results in a json format is a RESTful web server that uses the Python Flask framework. The Consumption Analysis sub-component first checks the database for previous analysis results on the same project. If none exists, it proceeds with the code profiling task. All this information is passed to the next step, which is responsible for suggesting a proper optimization.

5.6.1.1 Consumption Analysis Component back-end Both the energy estimation and hotspots identification functionalities require a source-code-parsing task. This procedure is performed for compiling and breaking the application source file in blocks, such as for and while statements or functions. More specifically, the AST tree produced by the compiler front-end

is analysed. For those purposes, the Clang compiler is used coupled with Python libraries (libclang). The aforementioned mechanism is implemented in a Python module.

For the hotspots identification, Valgrind and Linux Perf are used for fine-grain profiling of the applications execution (hprof is used for partially supporting Java applications). The generated output of the application profiling is recorded into log files. For each produced statement by the previously described procedure, the profiling results are analysed in order to characterize the statement as a hot-spot or not. For each hot-spot, the corresponding values of performance/energy indicators such as CPU cycles and cache misses are provided. All this information is forwarded to the Optimization component, which is responsible for suggesting suitable optimizations.

Although this approach produces accurate results, it requires the developer to use the appropriate structure that profiling tools need. Also, as mentioned in Chapter 4 and Section 5.3.3, the profiling procedure adds execution time overhead. Therefore, an alternative static analysis process is also introduced. For the energy estimation through static analysis, gcc and CLANG are used for compiling, while all the analysis is written in Python. As referred in Section 4.4.2, LLVM-mca is used for getting the features presented before. Python Scikit Learn is used for implementing the estimation models.

5.6.1.2 Optimization Component back-end functionality The hotspots identified by the aforementioned component (Consumption Analysis), are further analyzed for monitoring the values of features (described in Section 5) in order to suggest optimizations based on the potential energy gains estimation.

A dynamic-analysis-based classification is applied in order to identify the hotspots for which energy gains by acceleration are expected (see Section 5.3.3). The CPU code of each hotspot is analyzed, to monitor the accelerator-specific indicators. These indicators capture the extent by which the architectural features of the accelerator can be exploited. The values of these indicators are forwarded to the classification model (implemented in Python Scikit-learn). It is worth mentioning that we used only GPU accelerators but other types of accelerators can also be supported if the corresponding dataset is provided.

Finally the "Optimization" part (written in Python) of the component proposes energy consumption refactorings/optimizations: Data-flow optimizations are proposed in the case that the hot-spot under analysis includes nested loops that have a number of cache misses that is beyond a thresh-

old (3%), concurrency-related optimizations are suggested when we observe data races between the application threads and acceleration optimizations are recommended if the potential energy gains prediction phase estimates that offloading the hot-spot code on an acceleration will lead to energy consumption gains.

5.6.1.3 Language Support With respect to programming language support, the proposed tools are fully functional for C/C++ (indicators monitoring, hotspot identification and energy estimation). Some first steps for supporting also Java (indicators monitoring and hotspot identification) have been done using the hprof tool for profiling the applications. While not fully integrated in the platform, the majority of the tools are tested and work also for Python. Of course, in order to perform an analysis, the application must meet the standards and requirements set out on the SDK4ED Wiki page[30].

5.6.1.4 Git Support The introduced framework analyses projects that are stored in git repositories. For supporting this functionality, there is a special API (or parameters on the API) and the back-end includes all the mechanisms that are necessary for cloning a git repository (from a user-provided URL), using a user-name and a token for credentials in order to get the source files and all the information needed for performing the analysis. This capability makes the proposed framework more useful, following the same direction with popular software developing tools, making the introduced tools capable of beeing a part of the entire applications development lifecycle. In Section 5.6.4, the importance of git Support and the database (described in the next paragraph), will be highlighted through the use of the proposed tool during the developement of a real application.

5.6.1.5 Database implementation In order to support the proposed framework and store the analysis results, a MongoDB database is used. The results are retrieved from the database, when the API requests an analysis for a project commit that was analysed before. The database is also very useful for providing the results and services to other tools, as well as for supporting possible future extensions, that need access to historical analysis data. The database is implemented as a separate Docker container that communicates with the main toolbox Docker container via local port 27017. The data is stored in JSON format and the project name is the key used for searching the required results.

[30] `https://gitlab.seis.iti.gr/sdk4ed-wiki/wiki-home/wikis/Energy-Toolbox`

5.6.1.6 API implementation The Energy Toolbox API includes both the Consumption Analysis API and the Energy Optimization API. A call to this API tests the whole functionality of the Energy Toolbox. The URL given in Table 8 represents the call to the Energy Toolbox API including the requested information (details about each parameter can be found in SDK4ED Project Wiki page).

Table 8: Energy Toolbox API call

`http://<energy_toollbox_endpoint>:3002/analysis?new=T&user=<github-user>&token=<github-token>&url=<github-url>&commit=<github_commit>&type=<analysis-type>`	
url	The URL of the git repository to be analyzed
user	The user name of the repository owner
token	Private access token in the case of private repo
type	whether to run a full, hotspots-only, static-only, acceleration or history analysis. Supported values: ['full', 'hotspots', 'static', 'acceleration', 'history']
new	run a new analysis, or return existing results (if any)

5.6.2 Front-end

React is a JavaScript library for building user interfaces for applications. React can be used as a base for the development of web applications, as it is very effective for fetching changing data. React applications require the use of additional libraries for state management, routing, and interaction with APIs. The GUI of the introduced framework is implemented in React Framework [177]. The Energy dashpage is responsible for sending requests to the back-end APIs, according to the user input, as well as for visualizing and presenting the results of the analysis to the user.

The provided user interface offers the user the ability to select the applications for analysis and see the results as well as the recommendations that are produced by the proposed framework. Requests from the front-end are sent to the back-end in order to specify the microservices that will be executed. The UI is part of the SDK4ED platform [178], which can be found online *just for demo purposes* on `platform.sdk4ed.eu` (some components provide limited functionality on the online version of the tool for security reasons and due to proprietary back-end support tools, but the tool can be easily installed locally). A walk-through scenario coupled with some screenshots of the Front-end will be presented in Section 5.6.4, where the use

of the proposed framework on a real-life applications development cycle is presented.

5.6.3 Extensibility

One of the main goals of the proposed platform is to be extensible: The user can add new devices and systems. The GUI and backend include reports and guidelines, as well as useful scripts and tools. In this section, we present the main steps that must be followed to add a new system to the two introduced components, namely energy estimation and acceleration prediction.

5.6.3.1 Energy Estimation The main backend python script receives as input the source files of an application and performs the energy analysis for all the functions and loop statements. There is a specific call on an energy estimation function that takes as input the data collected from a specific platform. In order to add a new platform, the energy consumption of the programs of the training-set in the specific new platform must be computed and included in the file.

- *Step 1 (Back-End):* Collect the data for the new device and provide an additional call to the energy estimation function, giving the new data as an argument. For the case that the new platform runs a linux OS, the proposed platform provides a script that automates the procedure. The user has to add the commands needed for the specific platform (energy data received from external monitors, power monitors, specific paths from the device tree etc). If energy sensors are not available, a user defined estimation (such as a power-delay product) can be given.

- *Step 2 (Back-End):* In order for the user to add the new platform, two extra dictionaries to store the results need to be added. All the dictionaries are stored in a json file that is loaded when a specific event in the main platform page is triggered. Comments in the source code guide the user.

- *Step 3 (Front-End):* The prediction results are shown in the static analysis panel and are taken from the json file sent by the back-end services. The user can choose a specific platform from a drop-down list to show the prediction results. There is a special method that loads the results into the page. If the user wants to add a platform, an additional condition needs to be added in the method, including the name of the platform, which must be the same as it is defined in the back-end.

5.6.3.2 Acceleration Prediction This sub-component calculates a number of Acceleration Specific indicators based on which, a classification model is used to estimate the potential energy gains of using a specific acceleration unit. The step-by-step guidelines provided show how to add predictions for new GPU devices. A prerequisite for being able to achieve this is that the new device must include an energy sensor:

- *Step 1:* Download the dataset benchmarks[31]
- *Step 2:* Compile the dataset in the new device/system.
- *Step 3:* Run the dataset in the new device. The developer has to update the scripts providing the monitoring of power sensors of the new device.
- *Step 4:* The final dataset must be added in the back-end and more precisely in specific places given in the Wiki page of the platform. This process is guided by comments.

5.6.4 Energy Toolbox demonstration

This Section demonstrates the use of energy toolbox by presenting screenshots of the SDK4ED platform with analysis results of analysing an embedded healthcare application developed by Neurasmus (IMD) [179].

The *IMD application* is developed by Neurasmus and it is one of the pilot use-cases presented in the context of the SDK4ED EU project. It is an embedded application that runs on standalone battery-powered devices. These devices are equipped with wireless transceivers, able to communicate with external reader/programmer for remote monitoring, testing and updating. The application includes functionality for supporting receiving data from (ECoG/EEG) sensors via ADC periodically, performing FIR filtering on the input samples to approximate a Morlet wavelet, deciding whether a seizure is detected or not based on the FIR output and applying optogenetic or electrical stimulus via GPIO in order to suppress the seizure.

The first up-and-running version of the proposed framework was employed for assisting the development of the IMD-application by Neurasmus.

The first version of the application was an emulator of the final system that was running on a linux environment, written in C/C++ (imd-v1.3 version) [180]. The implementation on the hardware that follows is based on this first version that was used for experiments. Figure 51 presents the results

[31] `https://gitlab.seis.iti.gr/hmar/energy_toolbox_new_platform_addition.git`

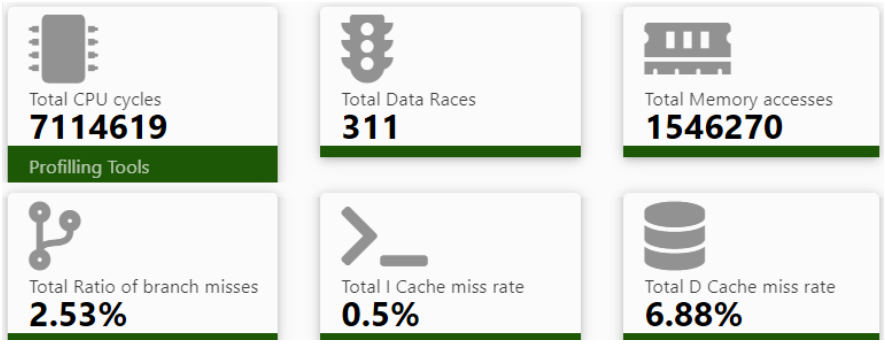


Figure 51: Indicators results panel

of Energy indicators monitoring. We see that the total number of Memory accesses is 1546270, which is considered a relatively small number, while the data cache misses (6.88%) are mainly cold misses at the beginning of the application, that do not lead to any cache-blocking optimization. Another important indicator is the Data Races. The imd-v1.3 version of the Neurasmus use-case consists of 4 threads, simulating a realistic healthcare wearables scenario. The analysis of the application based on the introduced tool revealed a significant number of recommendations by the Valgrind DRD tool. DRD is the thread error detector of the Valgrind suite, which is integrated in the introduced framework. DRD provides recommendations for eliminating data races and thread starvation. In the Neurasmus use case, DRD detected a large number of data races in the use of I/O C libraries and more specifically, in the use of printf(), scanf() and similar I/O functions. Since these functions are not implemented in a thread-safe manner, they may provide erroneous output results, leading to wasted energy consumption. DRD recommended the use of mutexes to eliminate this issue. With regard to the identified energy hotspots, the tool can report hotspots both on functions and in statements level by using the "Hotspot Granularity" button. It is worth mentioning that most of the identified hotspots correspond to the locks waiting for the rest of the application threads, while there is just one for loop statement in the energy hotspots.

The framework proposes also a for loop hotspot as a candidate block for acceleration due to the fact that there is a large instruction parallelism combined with a few cold references and control operations, while a significant part of operations are memory operations that do not refer to the same memory address. Indeed, this is a parallelizable loop, however this loop performs a very small number of iterations and therefore the use of acceleration does not make sense as we will have more delays on the transmission of the data and thus no such an optimization was applied.

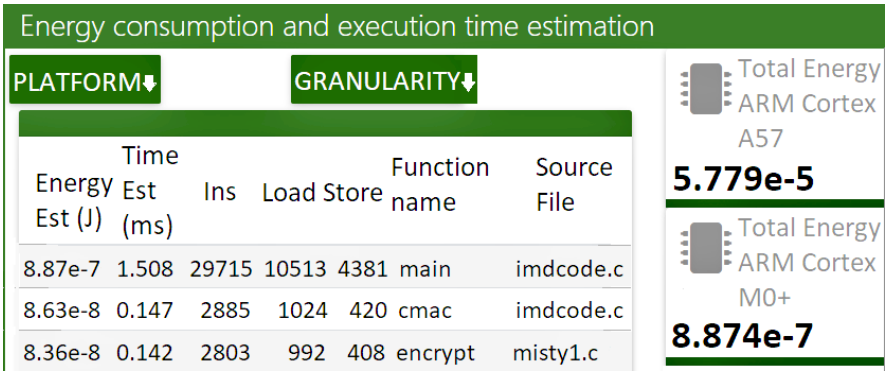


Figure 52: Static analysis results panel

The next versions of this use-case were implemented on the hardware platform and were continuously monitored through the proposed framework. The introduced static analysis energy estimation was extensively used in these versions. As an example, Figure 52 shows the results of the static analysis for the last version of the application. The static analysis provides a fast and easy way to have estimations of the energy and to compare the efficiency of using different platforms, assisting the developer to select the most energy efficient CPU architecture, giving also the trade-off between execution time and energy consumption. The panel consists of two parts: Profiling results per function or loop on the left presented table. Users can configure this selecting, by using the "Granularity" button. By setting different granularity levels the results in the table below are updated accordingly. Energy is estimated for each loop, while additional information is given. For example, in Figure 52 the estimated time and energy per function for ARM Cortex M0+, finally used in the use-case, are presented. Also, the user can select the platform for which Time and Energy is estimated. Results for running the entire application are also given for different embedded systems. For example, in this particular use-case, selecting the microprocessor ARM Cortex M0+ is more energy efficient but with a penalty on the response time and the quality of service. Using this microcontroller can achieve energy reduction up to 98% compared to using the more complex ARM Cortex A57. We should remind here that the introduced framework provides the option of adding more platforms. There are relevant guidelines in SDK4ED Project's Wiki page.

The static-analysis estimations for all the revisions (including minor Git commits) are presented in Figure 53 (as it is depicted in the GUI). We can see that the estimated energy consumption is lower when using ARM Cortex M0+. There is also an increase in energy consumption in the middle of

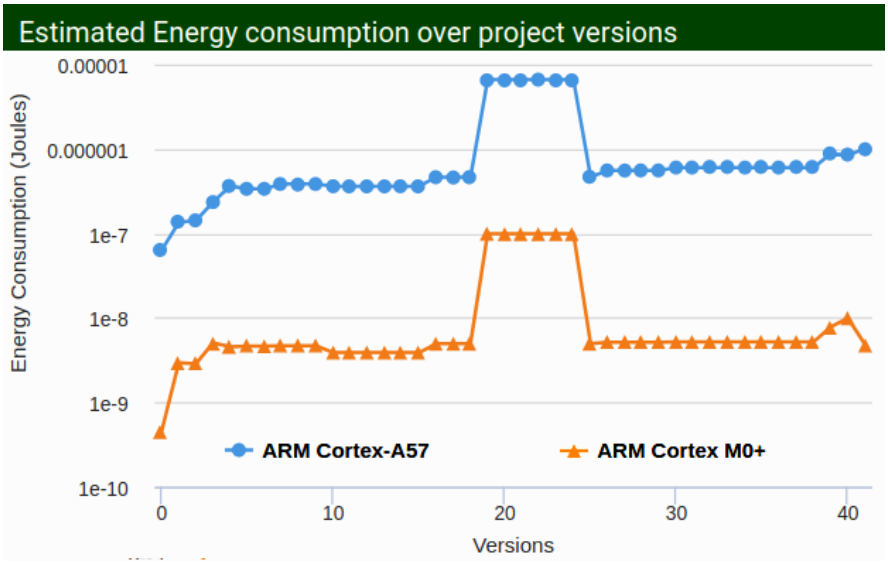


Figure 53: History results panel (IMD use-case analysis)

the project's history. A cryptographic function was added in these versions. After measuring the energy overhead in the next versions a special hardware crypto peripheral was employed for this purpose and the cpu energy was reduced by 95%. Note that the proposed framework only analyse the software part of the applications. These results were considered very useful during the development phase as they provide fast estimations of how the new code or refactorings affect the energy consumption [153].

5.7 Related work and comparison

Regarding the prediction of the acceleration gains, a detailed comparison of the method presented in this dissertation against the most recent performance prediction tools (i.e. after 2015) is depicted in Table 9. These tools analyze CPU code to predict potential performance gains by GPU acceleration (speed-up). XAPP [78] and CGPredict [79] use dynamic instrumentation to analyze CPU code. XAPP leverages machine learning techniques for prediction, while CGPredict uses analytical models. A first step towards a static analysis approach using random forest classification with two output classes has also been proposed recently [150]. Other recent works that belong to the broad category of performance predictors are the Compass and Automatch tools [80, 81]. Compass analyzes C code and generates application performance models, while Automatch detects application characteristics and through the generation of analytical modeling predicts the performance of

Table 9: Comparison against related recently designed approaches

Approach	**[78]**	**[79]**	**[150]**	**Proposed**
Analysis Type	Dynamic instrumentation	Dynamic instrumentation	Static Analysis	Combination (advantages of both)
Predicted value	Speed-up	Speed-up	Speed-up	Energy Gain & Progr. Effort
Targeted platform	CPU-GPGPU	CPU-GPGPU	CPU-GPGPU	Embedded
Provide results only based on static analysis	No	No	No, dynamic info needed	Yes
Training Dataset	Benchmarks	Benchmarks	Benchmarks	Benchmarks & Synthetic
Ability to reproduce the analysis	No	No	No	Dataset and models provided
Accuracy	76% - 88% regression	81% regression	62-85% classification	Static classification: 76% Dynamic classification: 85.3% Dynamic Regr: 63% (median) Effort Regression: 85%
Extensibility/ Usability	Medium	Medium	Medium	High (guidelines scripts provided)

execution in various accelerators.

Most of the aforementioned predictors rely on dynamic instrumentation techniques. After profiling the applications, they extract features that feed models that generate predictions. Although dynamic instrumentation is a well established analysis and profiling technique and is supported by widely-used tools (e.g. Valgrind [57], Pin [82]), it suffers from a large execution time overhead, as mentioned in Section 5.3.6. In addition, the predictors that entirely rely on dynamic instrumentation inherit the limitations and the constraints of the instrumentation tools that integrate, such as the need for specific configurations and the requirement of source code modifications. As a result, some first approaches of designing predictors that rely on static analysis of source code have recently been proposed [150]. However these approaches still require some dynamic information, such as the number of loop iterations by the user. Static analysis methods trade prediction accuracy or granularity level for user friendliness and short analysis time overhead. The methodology described in this work proposes a combination of static and dynamic analysis approaches to a single tool-flow, to exploit the advantages of both approaches, to provide flexibility to application developers, offering

also the capability of using static analysis only, when feasible, to avoid large time overhead.

Although the existing approaches provide predictions in terms of execution time, they do not consider energy consumption predictions. Indeed, energy efficiency is a critical design constraint, especially in the embedded systems domain. Therefore, in the context of this thesis, we extend existing approaches towards the prediction of energy consumption gains by acceleration on heterogeneous embedded devices. This is performed by investigating the potential use of machine learning features already proposed in the literature, but for building energy consumption (instead of performance) prediction models.

Finally, the existing tools predict only execution time gains without considering the programming effort that is required to achieve the predicted gains. Few attempts to quantify the programming effort of accelerating applications can be found in the literature, which are either based on empirical investigations [168], or rely on relatively simple metrics. A typical example of such a metric, is the Lines-of-Code (LoC) of accelerator-specific code (i.e. CUDA, OpenCL) versus the LoC of the corresponding CPU code [169]. However, using LoC may not be a good indicator of programming effort [170]. In addition, these approaches measure the effort on existing code, while it is very important to know the effort required before writing the new code.

We should mention that our main purpose is to provide an extensible tool that makes the prediction of energy gains by accelerating in heterogeneous embedded devices as well as estimating the effort of developing GPU-accelerated code prior development feasible and not to increase the accuracy of models as much as possible. The advantages of the work presented in this dissertation over the relevant approaches are summarized below:

- Combining both static and dynamic analysis approaches and exploiting the advantages of both of them into a single tool-flow.
- Designing a static analysis component relies entirely on analysing source code using text analytics techniques.
- Extending existing approaches ([78] [151]), that provide speedup predictions, towards estimating the potential energy gains too by studying the correlation of the used features with energy consumption.
- To the best of the author's knowledge this is a first approach towards designing a tool that estimates programming effort (expessed using the Halstead's effort metric) of developing GPU code using the CPU code as input.

5.8 Conclusion

A framework for analyzing applications for improving energy consumption has been introduced. The proposed method enables optimization suggestions, with particular emphasis on the investigation of potential energy savings for individual blocks of the application by using GPU accelerators. The introduced techniques rely on both static and dynamic analysis of CPU source code to provide a compromise between prediction granularity and time overhead. We concluded that approaches focusing on GPGPU speed-up can be extended towards Energy Consumption predictions for heterogeneous embedded systems. Accuracy evaluation performed in well-known platforms, where energy measurements are enabled by sensor, shows that the proposed methodology is able to provide acceptable predictions and that a future increase of the size of the dataset has the potential to further improve the accuracy of the models.

In addition, the impact of the proposed optimizations on design time software quality metrics was studied and a method for predicting programming effort prior developing the optimized code was proposed.

As a result, this Chapter presents a methodology of designing new tools that assist application developers to decide whether to invest in optimizing CPU application (mostly by using GPU accelerating), considering not only speed-up but energy consumption and programming effort for developing CUDA code, as well.

Moreover, the introduced models were used to build another type of software analysis tool that suggests energy-aware placement solutions of the application's individual functions on Edge devices.

Finally, a tool-flow that implements the proposed methods was integrated as a number micro-services, combined with a database, github support and graphical user interface. After evaluating the proposed tools in applications from well-known benchmark suites, a walk-through scenario using a real-life application was demonstrated.

Chapter 6

6 The Smart-grid HVAC Control use-case

The constant need for energy efficiency and the design of new renewable energy and smart-grid technologies, foretell that in the near future building facilities will be active participants in the energy market. This trend is expected to lead to autonomous micro-grids that incorporate energy trading capabilities. The challenges posed by the smart-grid concept can be addressed by the new CPS - IoT technologies. More specifically, the real-time monitoring and action capabilities of these technologies allow the system to be continuously refined to an optimal state. Utilities and information, such as market-driven pricing, are available even to the end users [30] [181]. Energy pricing policies (expressed as a price per kilowatt hour) are now beginning to abandon fixed pricing (24/7) and follow variable pricing mechanisms that make changes based on day, time of day or more dynamic events, such as weather conditions or expected load requirements.

6.1 Problem Definition

This section introduces the template of our case study. In order for the reader to understand the rest of this Chapter easily, Table 10 summarizes the symbols used in this analysis. The targeted study corresponds to a micro-grid environment, depicted schematically in Figure 54. In detail, our template includes multiple energy sources (e.g., solar, wind, bio-gas) and nodes that are in need of energy, such as the HVAC systems. Throughout this Chapter we focus on the orchestration task (highlighted with red color). These mechanisms aim to offer optimal control of system's components (energy sources) in terms of computing the HVAC thermostat set-points per thermal zone. In order to support the HVAC control task, a number of sensors acquire data related to weather (temperature, humidity, solar radiation and forecasts), building conditions (indoor temperature/humidity) and residents activity. This data is transferred to the main controller in order to compute optimal actions that co-optimize thermal comfort and energy cost metrics.

Table 10: The smart-grid HVAC control use-case Symbols

Symbols	
k	Number if buildings in micro-grid environment
t	Time-step
a_t	Control actions (thermostat set-points) at time-step t
$E_i(t, a_t^i)$	Energy consumption for building i at time-step t
$E_i^G(t, a_t^i)$	Energy purchased from the grid for building i at time-step t
$E_i^{PV}(t)$	Renewable energy for building i at time-step t
$P(t)$	Trading price for buying/selling energy at time-step t
tr	Trade-off between energy and thermal comfort optimization
AF	Available funds for buying energy budget from the grid
$C_i(t, a_t^i)$	PPD for building i at time-step t
C_{limit}	Maximum acceptable PPD (based on ASHRAE standard [132])
C_{est}	Estimated thermal comfort per thermal zone
C_{real}	Actual thermal comfort per thermal zone
E_{est}	Estimated energy consumption per thermal zone
E_{real}	Actual energy consumption per thermal zone
r_t	Reward at time-step t
c_t	Cost at time-step t (a combination of energy and comfort)
s_t	State of the building at time-step t
T_t^{out}	Outdoors temperature at time-step t
R_t	Solar Radiation at time-step t
T_t^{in}	Thermal Zone Indoors temperature at time-step t
H_t	Thermal Zone Indoors Humidity at time-step t

Throughout this Section we discuss the analytical form of the HVAC optimization problem. For this purpose, we build a multi-objective optimization problem (MOO), formally defined with Equation 11, where E and C give the two objectives (Energy consumption and occupants' thermal Comfort) under minimization. C corresponds to the Predicted Percentage of Dissatisfied occupants, while $C \leq C_{limit}$ gives the thermal comfort constraints that have to be satisfied. At this notation, α_i corresponds to the input variables that refer to the temperature set-point of the target HVAC system in time-step i. Finally, we consider that objective functions are also related to an external vector of environmental variables s_i.

The proposed framework focuses to a subset of the general MOO problem, where the cost function is expressed as a weighted sum of the single objectives [116]. However, the increased complexity of contemporary buildings makes it prohibiting, or even impossible, to consider at Equation 11 an

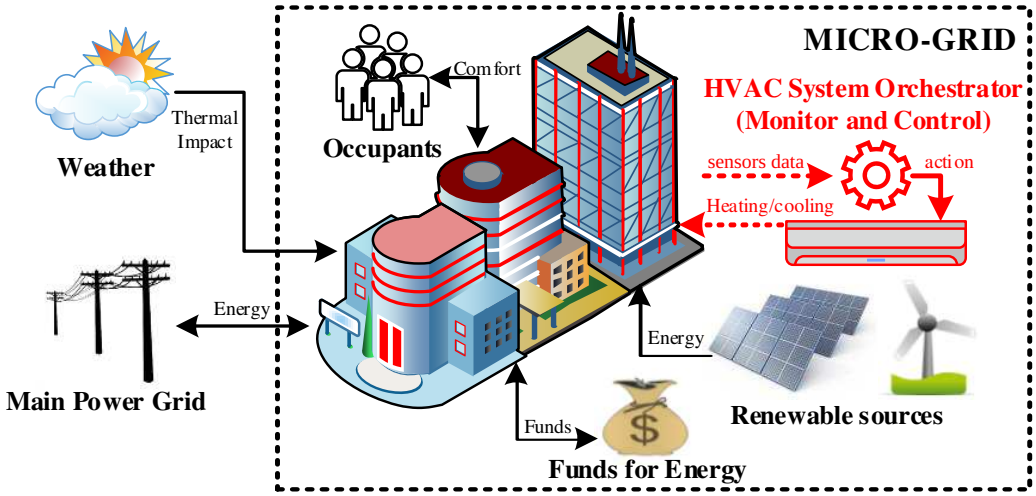


Figure 54: Overview of the employed Smart-Grid case study (emphasis on the HVAC control).

accurate analytical description, while providing a Plug&Play solution; thus, the definition of the problem that this chapter aims to solve is given by the Equation 11 coupled with the following properties:

1. the detailed form of the objective functions (E (Energy) and C (thermal Comfort)) is unknown;

2. the objective functions are not only related to the temperature set-point, but also to the buildings environment, the occupants behaviour etc.;

3. by considering discrete time/events (time-steps), the controllers actions are evaluated once per time-step;

4. No prior information is given.

$$\begin{aligned} Minimize : tr \times E(\alpha_t, s_t) + (1 - tr) \times C(\alpha_t, s_t) \\ subject\ to : C(\alpha_t, s_t) \leq C_{limit} \end{aligned} \tag{11}$$

The target case study considers five buildings with multiple thermal zones (summarized in Table 11), while the efficiency of the proposed orchestrator is evaluated with the usage of the well-established EnergyPlus suite [182]. The

Table 11: Summary of building properties.

Building	Surface area	Thermal zones	Operating hours	Warm-up phase	Random occupancy
#1	$350m^2$	8	6:00am–9:00pm	No	Yes
#2	$525m^2$	10	8:00am–9:00pm	Yes	Yes
#3	$420m^2$	10	8:00am–5:00pm	Yes	Yes
#4	$280m^2$	6	7:00am–8:00pm	Yes	Yes
#5	$228m^2$	4	6:00am–6:00pm	No	Yes

buildings' modeling was performed in detailed manner[32] [130]. The weather and energy pricing data, used in the experiments, correspond to publicly available information for 2010 [181].

By appropriately configuring the temperature set-point it is possible to improve the residents' thermal comfort and also reduce the energy consumption [183]. Assuming a grid consisted of k buildings, the quality of proposed solution is quantified with the (already mentioned) two orthogonal metrics, namely the *energy cost* and the *thermal comfort level*. A more detailed form of Equation 11 is given in Equation 12. Factors $E_i^G(t, S_t^i)$ and $C_i(t, S_t^i)$ denote the Energy purchased from the main-grid and the average thermal Comfort, respectively, for building i during the time-step t. a_t^i is a vector with the actions of our controller and more specifically the temperature set-points for the building i during time-step t. We should menthion that the building's energy cost per time-step $E_i^G(t, a_t^i)$ differs from the total energy consumption $E_i(t, a_t^i)$, since it takes also into consideration the power saving from PV panels ($E^{PV}(t)$). Additionally, by definition the comfort metric is improved whenever its value is reduced (as the Predicted Percentage of Dissatisfied - PPD metric was used).

$$Cost(t) = \sum_{\forall t} \left(tr \times \sum_{i=1}^{i=k} E_i^G(t, a_t^i) + (1 - tr) \times \sum_{i=1}^{i=k} C_i(t, a_t^i) \right) \tag{12}$$

The aforementioned energy cost is formulated by Equation 13: In case that the i-th building's energy requirements ($E_i(t, a_t^i)$) exceed the sum of energy provided by the PV panels ($E_i^{PV}(t)$), the additional demand is satisfied by purchasing energy from the main-grid at the current price ($P(t)$). On the contrary, if the energy budget for the desired HVAC operation is available

[32]The modeling of the buildings was part of the PEBBLE FP7 project funded by the European Commission under the grand agreement 248537.

from renewable sources ($E^{PV}(t)$) the energy cost is assumed to be 0. Extra energy can be stored in batteries, but this analysis goes beyond the scope of this dissertation. The proposed methods, however, could easily take this case into account.

$$E_i^G(t, a_t^i) = \begin{cases} \left(E_i(t, a_t^i) - E_i^{PV}(t) \right) \times P(t), \text{ if } E(t, a_t^i) \geq E^{PV}(t) \\ \\ 0, \textit{ if } E^{PV}(t) \geq E(t, a_t^i) \end{cases} \tag{13}$$

The energy consumption can be measured with metering devices, while the residents' thermal comfort can only be estimated. For our study, we employ the Fanger thermal comfort [1]. This model relates environmental and physiological factors in conjunction to the thermal sensation in order to estimate the Predicted Percentage of Dissatisfied (PPD) people in a room. We have to mention that both the energy metering device, as well as the PPD metric, are not applicable to the online control algorithms since they report results only for previous time-steps (up to $t-1$). To overcome this limitation, we estimate the impact of candidate temperature set-points in HVAC's energy and resident's thermal comfort.

Finally, the factor tr in Equations 11 and 12 defines the relative importance of optimizing either the energy, or the comfort objective (trade-off). A proper normalization of the objectives is necessary. For this purpose, the $E_i^G(t, a_t^i)$ is normalized over the nominal energy of HVAC system, whereas the $C_i(t, a_t^i)$ is normalized over its maximum observed value. Usually, the improvement of the residents' thermal comfort imposes additional energy consumption for cooling/heating the corresponding thermal zone. As it will be shown in Section 6.4, this formulation enables the study of alternative operating scenarios:

- *Scenario 1*: The goal is to achieve a compromise between energy consumption $\sum_{\forall i}\left(E_i^G(t, a_t^i)\right)$ and thermal comfort $\sum_{\forall i}\left(C_i(t, a_t^i)\right)$ metrics. This operation mode considers that $0< tr <1$, while regarding our implementation we study the case where both energy consumption and resident's thermal comfort are of equal importance ($tr = 0.5$).

- *Scenario 2*: Minimize energy consumption $\sum_{\forall i}\left(E_i^G(t, a_t^i)\right)$ while respecting a minimum threshold for the PPD metric. According to standards there are specific ranges for PPD in order for the thermal conditions to be acceptable for residents. For example according ASHRAE

standard [132], the range is 0–10%, while according EN15251 European standard PPD can be up to 15%. Hence, we consider that $tr = 1$, while PPD is up to the limit (Equation 14).

- *Scenario 3*: Optimize thermal comfort ($tr = 0$) without exceeding a predefined energy budget for the experiment's duration (mentioned as available funds or AF), as it is formulated by Equation 15.

$$Min \sum_{\forall t} \left(\sum_{i=1}^{i=k} E_i^G(t, a_t^i) \right) \tag{14}$$
$$s.t.\ \forall i \in 1 \dots k : C_i(t, a_t^i) \leq C_{limit}$$

$$Min \sum_{\forall t} \left(\sum_{i=1}^{i=k} C_i(t, a_t^i) \right) \tag{15}$$
$$s.t.\ \sum_{\forall t} \left(\sum_{i=1}^{i=k} E_i^G(t, a_t^i) \right) \leq AF$$

The problem sescribed in this Section cannot be considered trivial due to the intermittent behaviour of the solar energy, the uncertain building' dynamics, as well as the constant requirement to meet a desired residents' thermal comfort level. Section 6.1.1 describes the Experimental Framework, while Section 6.2 gives an overview of the related approaches to the problem. The proposed approach that offers a plug& play solution based on the Knapsack algorithm and Linear regression models is presented in Section 6.4. Finally, Section 6.5 offers experimental results that highlight the superiority of the proposed method.

6.1.1 Experimental Setup - Simulation testbed

To evaluate the effectiveness of the solutions presented in the next Sections, we use a well-known simulation testbed presented in [184, 185]. Figure 55 depicts an overview of the utilized testbed, which has also been used in a variety of works [186, 187]. The building dynamics and the data of the microgrid sensors are produced by the EnergyPlus suite [130] (see also Section 3).

The designed smart termostat controller gathers this data and calculates the set-points through MATLAB. Data exchanges are facilitated through BCVTB (Building Controls Virtual TestBed) [131]. As mentioned above, the

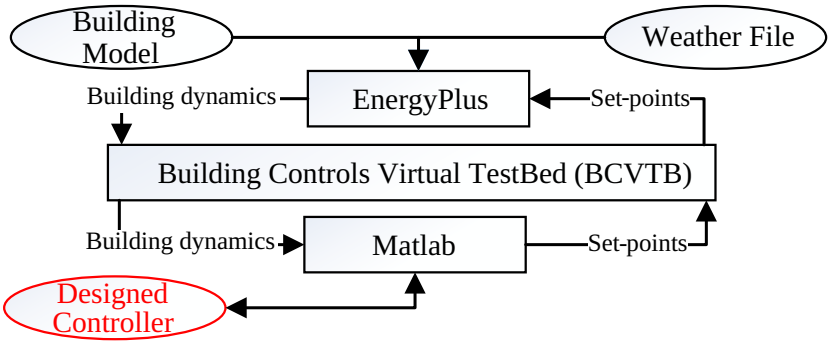


Figure 55: Simulation testbed of proposed controller

employed building models correspond to actual buildings located in Crete, Greece and their design is part of the PEBBLE FP7 EU project. All data required for the thermostat's control (energy consumption, room humidity, etc) are outputs of the EnergyPlus suite, ensuring realistic results.

To evaluate the thermostat's performance, the results are compared with rule-based control set-points (RBC's). This is a typical function found in all cooling/heating devices for setting a "static" temperature set-point for some periods (e.g. winter). In the context of this study, a setup with "RBC 23" means that during the experiment the temperature set-point at a given thermal zone is constantly equal to 23°C. This can be considered as a reasonable comparison because most manual thermostats tend to operate in a single heating set-point in winter, and a respective cooling set-point in summer [188]. Similarly, alternative smart thermostat solutions also produce set-points in the range from 20^oC to 27^oC, which is deemed reasonable in regard of thermal comfort. Fluctuations in these set-points do exist, since other factors affecting room temperature and user activities could be at play (open windows, cooking, etc), but still the result of these fluctuations would be a trajectory varying between the RBC set-points. So, by including a set of RBC values for comparison, a meaningful assessment against typical user or smart control can be ensured. In Section 6.5 comparisons against more sophisticated and well-established solvers are also provided.

6.2 Related work

The problem of deciding upon the HVAC configuration in smart buildings is a well-established challenge that has been attracting the interest of many researchers over the years [189] [190] [191] [192]. There are two mainstream ways for the HVAC configuration according to literature. The first way aims on designing systems that provide online decision-making [28], while the lat-

ter method relies on Model Predictive Control (MPC) [193]. Each method's inherent characteristics offer different advantages and characteristics.

More precisely, the online algorithms exhibit limited efficiency compared to MPC but they are reactive to real-time constraints (i.e., climatic conditions, occupants behaviour, etc). Although MPC for nonlinear systems has been extensively analyzed and successfully applied in various domains during the recent decades [84] [85] [83] due to the significant progress made in optimal nonlinear control theory [194] [195], they likewise encounter dimensionality issues: in most cases, predictive control computations for nonlinear systems amount to numerically solving a non-convex high-dimensional mathematical problem [86], whose solution may require formidable computational power for supporting online solutions.

However the typical "black-box" approach of on-line methods (e.g. based on machine learning techniques) is often criticized [83]. MPC solutions include controllers that were usually designed along with the system under control. However, as mentioned before, MPC algorithms cannot support real-time decisions because their efficiency relies on moving forward in time to simulate the impact of alternative control strategies. In addition to that, online solvers exhibit lower computational and storage complexities compared to the corresponding MPC solutions (especially when the buildings' modelling relies on simulation); thus, they are candidate for being implemented onto embedded devices

The increased computational requirements of MPC-based algorithms makes their implementation feasible only on enterprise environments, e.g., as part of a Building Energy Management (BEM) systems. However, recently there is an emerging need for solutions that are applicable to residential buildings as well. This necessity is being clear considering the expanding market of smart thermostats [196].

In order to alleviate the overhead of the increased computational complexity, meta-heuristics are employed. This type of approaches includes stochastic dynamic programming [26] and genetic algorithms [197]. Furthermore, methods that rely on empirical models [198], simulation optimization [27] and event-based optimization (EBO) [190] have also studied. Although these algorithms trade-off quality of derived solutions with the associated computational and storage complexities, they are rarely employed as orchestrators for large-scale systems, such as the one discussed throughout this Chapter, because the algorithm's customization phase is firmly tied to the selected cost function. In addition to that, the adoption of cost function in (meta-)heuristic solvers impose mentionable effort for fine-tuning algorithm's parameters and weights.

An alternative way for trading quality (accuracy) with problem's com-

plexity relies on empirical models, such as fuzzy logic [87] [88] [89] [90], that enable CPS customization without prior detailed modeling of the underline ecosystem. Specifically, the controller employ a fuzzy approximation scheme (i.e. predefined action plan) according to the feedback acquired by sensors. Supervised learning techniques, such as Artificial Neural Networks (ANNs) [92] [91], are also recently gaining a lot of attention because they do not require detailed study of the micro-grid dynamics. More specifically, by training these models with history data, the ANN learns the behavior of the CPS. This enables solutions that rely on ANN to be employed as model-free approaches at the decision-making task of CPS platforms by acquiring input data from sensors [199].

Although the aforementioned techniques are in line with the model-free controller concept, they exhibit limited flexibility. Specifically, both supervised and unsupervised machine learning models usually impose excessive computational/storage complexities for training, which cannot be tackled with low-cost embedded platform under real-, or run-time, constrains. Similarly, fuzzy rules create fuzzy classes regarding some of the system's parameters; hence, they cannot "*learn*" in detail the CPS's behavior at unexpected operating conditions. To overcome this limitation, a "pre-training" phase has to be performed based either on historical data (pairs of input from sensors and their associated score from the objective function), or on simulation software that estimates overall system's performance. Consequently, both machine learning and fuzzy rules solvers cannot tackle efficiently the task of self-adaptive orchestration without any prior knowledge of the problem's statement that we indent to address throughout this dissertation.

To address this challenge, another class of decision-making algorithms have been proposed that "learns" continuously by quantifying the quality (impact) of applied actions. These algorithms, also known as Reinforcement Learning (RL), apply a rewarding scheme that accrues by the evaluation of the CPS output. Thus, based on RL decisions, the orchestrator takes actions in an environment so as to maximize some notion of cumulative reward. Various RL approaches have been proposed to address the HVAC control according to a strategy for simultaneous maximization of occupant's thermal comfort and energy savings [93] [95] [94]. Apart from the control algorithm itself, these approaches do not give any information about the physical implementation, fact that doesn't contribute towards a rapid prototyping solution. This challenge becomes more important by taking into consideration the development cost of an embedded system.

Table 12 provides a qualitative overview among alternative algorithms discussed so far. According to this analysis, the designed method has to be able of taking into account constraints posed at execution phase. However,

Table 12: Qualitative comparison of system's orchestrators.

	Optimi-zation	Design Time	Plug& Play	Real Time	Adaptive	Complex.	Accuracy
On/Off [200]	No	Low	Yes	No	No	Low	Low
PID [201]	No	Low	Yes	No	No	Low	Low
MPC [191] [193] [83]	Yes	High	No	No	No	High	High
Approx. MPC [189] [192]	Yes	High	No	No	Partially	Med.	Med.
Event-based Opt. [190]	Yes	Med.	No	Yes	No	Med.	Med.
Fuzzy [88] [202]	Partially	Med.	Partially	Partially	No	Low	Med.
Stochastic [97]	Yes	Med.	No	Partially	No	High	Med.
ML [92] [186]	Yes	Med.	Partially	Yes	Yes	Low	High
RL [93] [94]	Yes	Low	Yes	Yes	Yes	Med.	Med.
Proposed	Yes	Low	Yes	Yes	Yes	Low	High

our target is on exhibiting lower computational and storage complexities. Note that the absence of lightweight decision-making solutions in relevant literature able to be executed onto embedded platforms is not due to neglect, but rather due to its difficulty. Additionally, the plug&play functionality is also crucial for this type of orchestrators, as they have to support the model-free feature. In contrast to MPC and online methods, we focus on designing modular components (e.g. history windows, energy/thermal models, heuristic solver, etc.) that can be adapted even at run-time in order to respect the resident's requirements. Furthermore, the mainstream methods usually require prior knowledge, while we focus on learning the system's behavior through receiving information form its output. Finally, regarding the hardware implementation phase, the presented framework exhibits remarkable lower design complexity since it is not necessary to model system's dynamics, while the availability of software tools for supporting the implementation phase is also a crucial parameter.

6.2.1 Simulation-based approaches and objectives

Current approaches to the specific problem described in Section 6.1.1 require simulation-based loops. For example, the solution by Korkas et. al. [28] makes about 700 loops of using a detailed simulation of the micro-grid which needs about 20 hours to calculate the results for a 3-days building operation, while relative solutions (fmincon and genetic algorithms) need about 4 days to complete.

Another simulation-based approach uses the well-established *fmincon* solver [203]. Fmincon solves the problem by performing iterative simulations based on the interior-point algorithm, while the objectives (energy and comfort) are known a priori through detailed modeling of the buildings dynamics

by using the EnergyPlus suite. Consequently, the efficiency of this method is limited by the accuracy of modeling each particular setup, which is an ineffective online solution. Moreover, the fact that Fmincon relies on exhaustive design space exploration, leads to high latency (arround 5×10^{14} execution cycles, or more than 12 hours execution time, even for simulating just 1-day building's operation in Intel i7-6700K@4GHz), which makes its usage prohibitive for an embedded controller.

The main objective of the proposed solution is to support the task of decision-making for HVAC control but with significant lower computational and storage complexities as compared to the existing simulation-based implementations. For this purpose, different contributions of the proposed method (e.g., coarse-grain window, fine-grain window, low-complexity energy and thermal comfort models etc) are applied in a complementary manner.

A major differentiator of the introduced solution compared to similar approaches found in relevant literature can be summarized as follows: Rather than applying a detailed modeling either of building or HVAC dynamics, which is a time-consuming procedure, throughout this study we propose a Plug&Play framework that can be applied to any building in order to control the HVAC system. The data acquired by sensors enable the proposed orchestrator to improve itself through a "learning procedure", without considering any prior knowledge (i.e. buildings' modeling) and constant definition of cost function(s).

We strongly believe that by enabling "smart components" to apply an efficient pre-processing step of the acquired data (such as the sliding windows), it is feasible to minimize both the data communication problem, as well as to enable more efficient data analysis. For this purpose, rather than proposing a "typical" control algorithm with enormous computational and storage complexities, which execution pre-requests an HPC (High-Performance Computing) platform, we study a solution capable to be executed onto low-cost embedded devices similar to those that tackle the data acquisition in the majority of "smart sensors".

6.3 Reinforcement Learning Approach (RL)

Throughout this Section we aim to design a rapid solution to the model-free HVAC optimization problem, which is based on Reiforcement Learning (see Section 3). For this purpose, we should remind the multi-objective optimization problem (MOO), formally defined in Equation 11, where E and C are the two objectives (energy and comfort) under minimization, α_i corresponds to the input variables (controller's actions) that refer to the temperature set-points in time-step i and s_i is an external vector of environmental

variables.

Reinforcement Learning is a very popular method that targets our need for self-adaptive solutions. As a Reinforcement Learning problem consists of a set of states, a set of actions, and a reward function. Each time-step, the dynamic control of the HVAC system is performed by the Reinforcement Learning Agent (see Section 3), by sampling the state of the building and computing the next *set-point*, i.e. a vector designating the values of all the configuration parameters of the HVAC system.

In order to address the inherent unpredictability of the unknown underline system, the Reinforcement Learning (RL) based controller (i) navigates between available system's states, and (ii) models and predicts the cost function according to the already acquired (from the CPS/building's sensors) data. The studied RL algorithm models the target system by means of a set of *states*, a set of *actions* and a reward function. Since the reward maximization is equal to the minimization of cost function (see Equation 16), for the rest of our analysis we will refer to c as the *cost* function. At this notation, a real value c_i is assigned per transition (per time-step). Estimating the action-value function Q (see Equation 2 in Section 3) is of high-importance for evaluating the overall orchestrator's performance. Moreover, as the employed costs c_i are unknown, a function approximation (by means of data-driven supervised learning) is applied.

$$r_t = max\_cost - c_t \tag{16}$$

The Reinforcement Learning method is supported by a framework depicted in Figure 56 in order to be used to control the targeted HVAC system operation. The next paragraphs describe the individual components of the framework.

6.3.1 Pre-processing tasks

The functionality of these tasks is to guarantee that the preceding decision-making logic can dynamically respond to the building and weather related events, constructing the introduced self-adaptive mechanism.

State retrieval: This task retrieves the current buildings' state once per time-step. The state is formed by the subset of acquired data that influence system's functionality. At this notation, a system's state (Equation 17) is defined as a tuple of (*Outdoor temperature (T_i^{out}), Solar radiation (R_i), Indoor temperature (T_i^{in}), Indoor humidity (H_i)*), since they effectively capture the building's dynamics for both energy consumption and thermal comfort metrics.

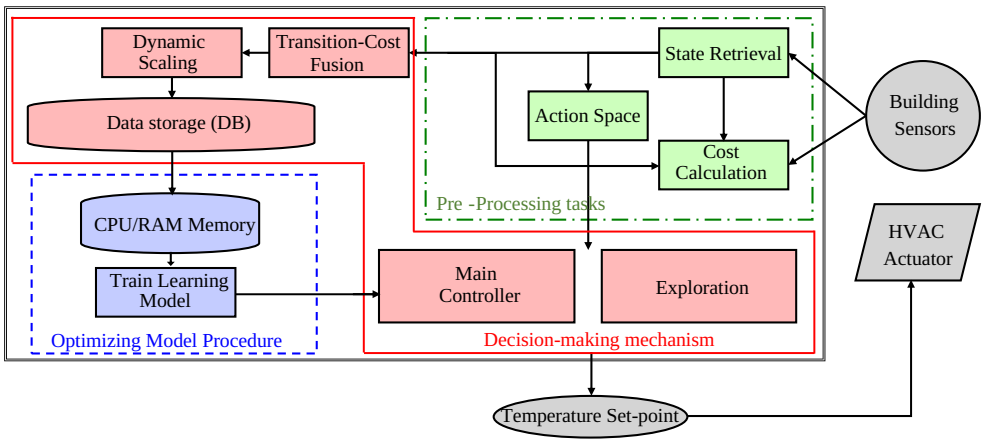


Figure 56: Reinforcement Learning method for design and customization of the targeted model-free HVAC orchestrator

$$s_t = [T_t^{out}, R_t, T_t^{in}, H_t] \tag{17}$$

Action Space: For each time-step, an action refers to the assignment of a temperature set-point per thermal zone. Then, the action space accrue by dividing the range of HVAC's set-points (temperature) into discrete segments. A step-by-step search approach is applied for this purpose. Regarding the studied problem formulation, the required action space a_i includes five candidate options with respect to the current temperature, i.e. maintaining current temperature, increase/decrease it by a step of $\pm 0.5°$C or $\pm 1.0°$C degrees (Equation 18).

$$\alpha_t \in \{T_t^{in} - 1, T_t^{in} - 0.5, T_t^{in}, T_t^{in} + 0.5, T_t^{in} + 1\} \tag{18}$$

Cost calculation: The cost function is formed by a weighted sum of the HVAC objectives (energy cost and thermal comfort). An action is deemed *terminal* (i.e., resulting in a terminal cost of value *max_cost* that corresponds to zero reward (see Equation 16)) if it leads to prohibitive results regarding the constraints (thermal comfort), or the constraints were already unsatisfied and the action further increases the dissatisfaction value [106]. Our framework evaluates the efficiency of applied, or candidate actions, through the cost function given at Equation 19. The weighting factor ($tr \in [0, 1]$) gives the relative importance between the two orthogonal metrics, namely the *energy cost* and the occupants' *thermal comfort level*.

$$c(s_t, \alpha_t) = \begin{cases} tr \times E(s_t, \alpha_t) + (1 - tr) \times C(s_t, \alpha_t), & C(s_t, \alpha_t) \leq C_{limit} \\ max\_cost, & \text{else} \end{cases} \quad (19)$$

Regarding the energy cost (E), if the expected energy loads of the buildings at time-step i exceed the energy availability from the photovoltaic panels (PVs), the excessive demand is met by purchasing additional energy from the main-grid at price. Otherwise, the energy requirements are met with microgrid's renewable sources.

Given that the value of thermal comfort (C) cannot exceed a limit (15% according to EN15251 European standard or 10% according to ASHRAE standard [132]), the cost metric at Equation 19 does not equal to max_cost whenever $|\mathrm{C}| \leq C_{limit}$. On the other hand, an action is deemed terminal (i.e., corresponds to a terminal cost) if:

- the thermal comfort metric is out of acceptable range ($|C| > C_{limit}$);
- the thermal comfort metric was already out of the acceptable range and the action further increases its value.

6.3.2 Decision-making mechanism

In this paragraph, we descibe the functionality of the presented RL algorithm. By taking advantage of its acquired knowledge, the method increases the efficiency of the predicted actions. In principle, this is a leap towards self-adaptive orchestrators, as any unforeseen and manifested condition will be considered at the supervised machine learning model.

Transition-cost fusion: The controller's efficiency is based on the history of all encountered states, taken actions, calculated costs and the preceding states as a result of these actions. A batch of data in the form of concatenated tuples (s_i, a_i, c_i), one per transition (s_i, a_i, s_{i+1}), is created and stored to the database to support the learning procedure.

Dynamic scaling: An accurate scaling (normalization) of the objective functions must be accomplished especially for the energy consumption, where the range is unknown without prior information about the buildings' dynamics. Our framework lies to uses an unsupervised dynamic scaling method [204], where running average and standard deviation are calculated for all the objectives. As new data acquired from building's sensors, the scaling parameters are re-calculated, ensuring that the scaling is up-to-date.

Data storage: The presented framework considers that both transitions and costs are stored in databases to enable model's refinement task.

Main controller: Given the current state and the available actions, the orchestrator designates the configuration for the next set-point, which minimizes the expected return (i.e. the cumulative future costs in the time frame defined by γ in Equation 1 of Section 3).

An Artificial Neural Network (ANN) is used for supporting the task of machine learning. The training of this ANN is performed with the database in the form of (s_i, α_i) tuples. The targets for this training are computed based on Equation 20, where Q_i denotes the output of the *current* ANN. Finally, the term $c_i(s_i, \alpha_i)$ is a linearly scaled value of the actual cost as a function of cost definition and dynamic scaling adjustment.

$$target = c_i(s_i, \alpha_i) + \gamma \cdot min_{\alpha_i} Q_i(s_{i+1}, \alpha_i) \tag{20}$$

When terminal cost is exceeded, the following two procedures take place:

- An action, which exhibits increased probability to improve the overall solution in term of thermal comfort constraints satisfaction is selected. This choice merely rectifies the previous "sub-optimum" action and based on previous observations it computes a set-point that ensures thermal comfort constraint satisfaction. More precisely, this action increases or decreases the indoor temperature by 1°C, according to the residents estimated thermal comfort.
- The ANN model is retrained

Exploration: In order for the proposed framework to solve non-convex problems, an exploration mechanism for avoiding being trapped in a local minimum is necessary. In this work, an ϵ-greedy exploration mechanism is employed. The main controller operates with possibility $1 - \epsilon$, where ϵ is self-regulated [205]. The value of ϵ is calculated based on the orchestrator's success rate and its update is performed by Equation 21, where λ and k are the success rate and learning rate, respectively.

$$\epsilon' = f(1-\lambda)\Big[k(1-\lambda) + (1-k)\epsilon\Big], where\ f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{21}$$

The λ parameter is based on the orchestrator's consecutive positive outcomes, which are determined by the validity of the learning model's prediction. Inspired by [206] the definition of this validity is selected to correlate with the Temporal Difference (TD) error formulated by Equation 22. Specifically, the controller's decision is deemed positive if $|TD| \leq TD\_limit$, where

Table 13: Summary of the introduced RL solution parameters.

State	$s_t = [T_t^{out}, R_t, T_t^{in}, H_t]$	
Action Space	$\alpha_t \in \{T_t^{in} - 1, T_t^{in} - 0.5, T_t^{in}, T_t^{in} + 0.5, T_t^{in} + 1\}$	
Objective	$Maximize\ \sum_{i=0}^{n} \gamma r_t$	
Reward	$r_t = max\_cost - c_t$ $c_t(s_t, \alpha_t) = \begin{cases} tr \times E(s_t, \alpha_t) + (1 - tr) \times C(s_t, \alpha_t), & C(s_t, \alpha_t) \leq 15 \\ max\_cost, & \text{else} \end{cases}$	
Algorithm	Neural Fitted Q Iteration (NFQ)	
Exploration	Strategy	ϵ-greedy exploration
	Exploration rate (ϵ)	Based on model's prediction (Eq. 21 and 22)
Discounting factor - γ	0.98	
Learning Model	Type	Multilayer Perceptron (MLP)
	Activation Function	tanh
	#Layers	4
	#Nodes	16
Validity Definition	Based on	Temporal Difference (TD Eq. 22)
	Criterion for positive decision	$\lvert TD \rvert \leq 0.15$

TD_limit has to be small enough to represent an accurate approximation and elastic enough to encourage early stages of learning [207].

$$TD = c_t(s_t, \alpha_t) + \gamma \cdot min_{\alpha_t} Q(s_{t+1}, \alpha_t) - Q(s_t, \alpha_t) \tag{22}$$

Table 13 summarizes the parameters employed within our RL-based solution. The values of these parameters regarding the studied case study (i.e. efficient configuration of HVAC system at agnostic buildings towards improving thermal comfort and energy savings metrics) are also included.

6.3.3 Optimizing Model Procedure

This task performs the iterative model optimization. Specifically, it deals with the data transfer from system's database to RAM or GPU memory, the ANN's targets calculation according to Equation 20, as well as the ANN retraining.

Train Learning Model: Two different approaches for ANN retraining are considered. The first of them assumes that one training epoch is performed once per time-step, whereas the latter approach concerns a full ANN retrain in case a terminal-cost is reported. Since an ANN retraining includes many epochs, it exhibits increased computational complexity and execution run-time overhead. During the initial time-steps, ANN retrain is performed more frequently due to the limited orchestrator's "knowledge"; however, the reduced amount of training data imposes negligible overhead for this task. On the other hand, the retrains for an already trained RL algorithm are lim-

ited; hence, the associated overhead (due to the increased database size) is also limited.

6.3.4 Experimental Results

Without loss of generality we consider that both energy consumption and thermal comfort metrics are of equal importance; thus, the selected weights at Equation 11 are equal to 0.5.

The targets of the proposed data-driven machine learning method aim to derive temperature set-points that not only lead to lower costs for the current time-step, but also optimize future orchestrator's decisions. For this purpose, careful selection of the γ parameter (Equation 1) is necessary. Based on our exploration, we conclude that the performance of proposed orchestrator is retrieved for γ equals to 0.98. The estimation of objective functions is performed via a Multiple Layers Perceptron (MLP) ANN. The activation function for all the nodes except those of the output layer is the hyperbolic tangent sigmoid, whereas the output layer incorporates the logistic sigmoid to produce values in the range $[0, 1]$. Similarly, the λ parameter for the exploration component refers to the number of successful actions of the controller. For the scopes of this manuscript, according to the required characteristics of the TD_limit presented in 6.3.2, we consider that an action is valid if the MLP's $|TD|$ is less or equal to 0.15 (Equation 22). With regard to data manipulation for local database storage, we selectively save the data that refer to the last N days in order to reduce the incremental batch of data. Such a technique has proven to be very efficient as it will be discussed in Section 6.4.

Initially, we quantify the efficiency of the proposed framework in terms of *episodes*. An episode is a sequence of control iterations, that ends if the current state fulfills a termination condition (e.g. the system reached its goal state, or a failure occurred) [208]. In the context of this case study, an episode ends either when the orchestrator results to thermal comfort values out of the acceptable limits, or the HVAC system is turn off at the end of a day.

The results of this analysis are depicted in Figure 57. For demonstration purposes, the horizontal axis plots the id of consecutive episodes for an operation period of three months (January–March), while the vertical one gives the duration (in term of operating time-steps) of each episode. Without affecting the generality of our analysis we considered time-steps of 20 minutes duration. Hence, the controller can be involved up to 45 times per day (based on operating hours depicted in Table 11). At this figure, an episode with duration 45 indicates that the Main Controller component computes

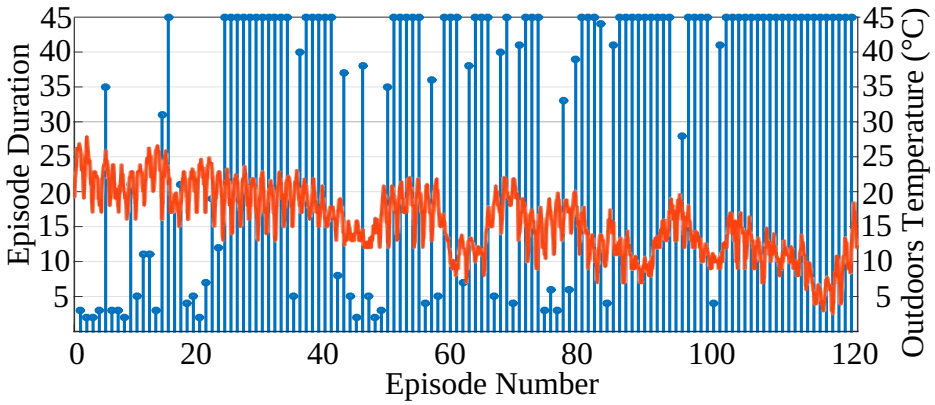


Figure 57: Evaluation of the orchestrator's performance over time: Episodes duration regarding the January–March experiment.

successfully the temperature set-points for the entire day without any failure. Based on our experimentation, the proposed orchestrator achieves an average episode's duration equals to 32 regarding the 3 months (90 days) experiment. In addition to that, if we exclude the training phase during the first 20 episodes, then the corresponding average episode's duration is 37. This figure highlights also that the performance of introduced orchestrator is improved over time, since there is no failure for the last 22 consecutive days. In order to study more thoroughly this efficiency, we also plot with red color line the outdoor temperature. This analysis indicates that the proposed orchestrator exhibits increased episodes' duration until an unexpected change in weather conditions that has never been encountered before occurs (i.e. at episode 40 the outdoors temperature remains under $15°C$ for the whole day). Similar behavior is reported until the proposed model-free orchestrator to be robust to weather changes (last 22 days).

The mean TD error (Equation 22) for each of the first 90 days is plotted in Figure 58. These results confirm previous evidence about improving the controller's efficiency over time, as the machine learning part of the controller leads to lower error values. More specifically, the majority of these values is less than 0.15 (average error value is 0.07), which has been defined in as the threshold value for a successful.

Finally we should evaluate the RL based controller's efficiency against typical RBC values (see Section 6.1.1). Figure 59 summarizes the results for a typical summer day. According to these results, we might conclude that the proposed solution actually verges on the ideal comfort level for $tr = 0$ and leads to less consumption for $tr = 1$. Results concerning for the entire year and more precisely two 3-month periods, one for testing heating (January to

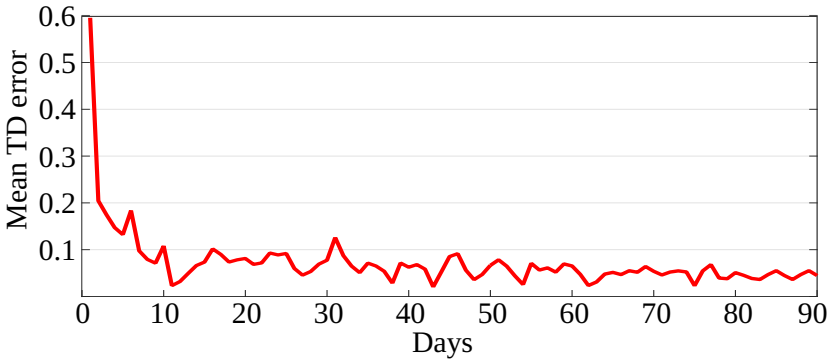


Figure 58: Evaluation of the Machine Learning model: Daily mean MLP TD-error

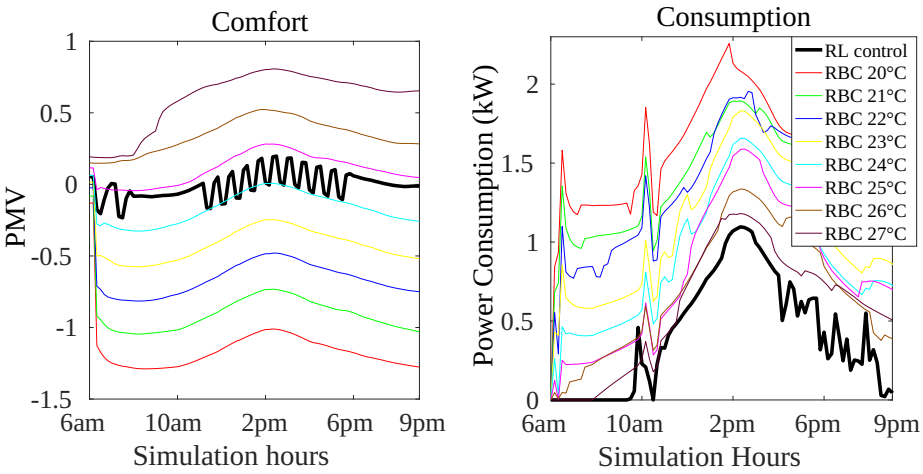
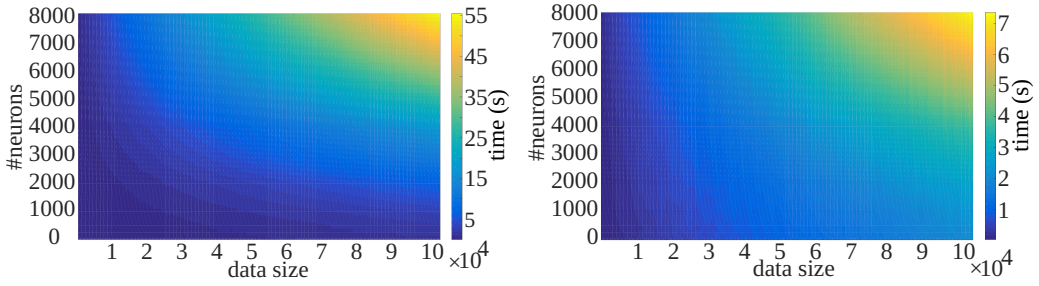


Figure 59: Daily performance vs RBCs (left $tr = 0$, right $tr = 1$)

Trade-off	Mean energy savings (vs RBCs)	Mean comfort savings (vs RBCs)
0 (Optimize Comfort)	-7.8% / 10.8%	**11.9%** / **41.8%**
0.5 (Optimize both)	**28.4%** / **32.4%**	**-3.9%** / **27.4%**
1 (Optimize Energy)	**59.2%** / **48.3%**	-23.3% / 5.8%

Table 14: Evaluation of RL control performance

March) and one for testing cooling (June to August), are presented in Table 14. The RL based controller achieves up to 59.2% mean energy savings (for $tr = 1$) and up to 41.8% comfort savings (for $tr = 0$) on average.



(a) Embedded CPU (ARM Cortex A57) (b) Embedded GPU (Nvidia Tegra X1)

Figure 60: Time needed for model optimization (data transfers + building targets + 1 epoch re-training)

6.3.5 Orchestrator's Performance on Embedded Devices

While the framework that supports the Reinforcement Learning solution consists of numerous components, the task of model optimization is performance dominant. Hence, improving the execution time of this component will improve the overall CPS orchestrator's performance. In order to study the run-time requirements for the proposed framework, the model optimization test was implemented onto a low-cost embedded system. The target platform for this analysis is an 4-core ARM Cortex A57 operating at 1900 MHz with 4 GB of system memory. Figure 60a plots the execution latency for different input data set sizes and number of neurons per ANN. These results refer to the ANN retraining task for ANN architectures that consist of 2 up to 18 layers. Each of the solution has an average of 4 input features (i.e. state) that correspond to environmental variables that are acquired by CPS's sensors.

According to this analysis, the proposed framework can perform model refinement (through ANN's 1-epoch retrain) with a maximum execution run-time overhead of 55 seconds for 10^4 data and 8197 neurons. In case a more complex model is considered, our decisions have to be performed in tighter manner. In such a case, an architectural template with accelerator hardware (such as FPGA or GPU cores) is necessary. Figure 60b evaluates the corresponding ANN performance regarding the same experiment for the Nvidia Tegra Jetson TX1 embedded many-core platform (with 256 cores). Based on this analysis, the model's retraining is performed in less than 8 seconds, which is sufficient for the majority of on-line orchestrators.

Taking onto account that for the case-study of smart thermostats we utilize a MLP with 4 input nodes, corresponding to the 4 features of the state (see Section 5) and 2 hidden layers with total number of nodes 16, and

a data window size that is less than 3000, we might conclude that around 0.03 second delay is added to each time-step (the decision-making has to be done in less than 10 minutes) and as a result an embedded CPU can support the proposed method. However, if a more complex model is needed, more data are required and the time-step restrictions are more tight, a more complex embedded system with a GPU can be used.

The second optimizing approach refers to the total neural network training when optimizing model. It is important to note that according to this approach, this full re-training is not required in every time-step but only when the Main Controller fails. For example as shown in Figure 57 for our case-study, for the first 3 months of operation only 55 re-training instances are required. As a result the time delay is acceptable and allows the training phase to take place with the continuous operation. For demonstrating the performance in this approach we would repeat the previous experiments (same data transfers and target preperation) but for 100 epochs NN training. This procedure would lead to around 100 times larger overhead. For the case-study of the smart thermostat, we calculated an overhead of around 1 second for ARM Cortex A57.

6.4 Proposed LR-Knapsack based Orchestrator

In this section we propose a more detailed low-complexity decision-making mechanism for the specific targeted CPS optimization problem. Contrary to relevant methods, that rely either on detailed building modeling, iterative and time consuming simulation or on analyzing an excessive amount of data, our approach delivers close to optimal results by taking into account only a small subset of information without applying any simulation or considering any prior knowledge. This solution aims to improve the results even more than the "black-box" RL approach described in Section 6.3, being also more flexible as it supports multiple operating modes easily, such as balancing energy consumption with residents' satisfaction, minimizing energy consumption while maintaining a satisfactory level of thermal comfort, and maximizing residents' satisfaction without exceeding the available energy budget.

The introduced LR-Knapsack orchestrator consists of three basic components as depicted schematically in Figure 61. Initially, the method manipulates the raw data aquired from various sensors (e.g., temperature, humidity, solar radiation etc). Additional details are provided in Section 6.4.1. Afterwards, the proposed linear regression based models for estimating energy and thermal comfort are appropriately refined (Section 6.4.2). Finally, the orchestrator proceeds to the third component (the decision-making itself),

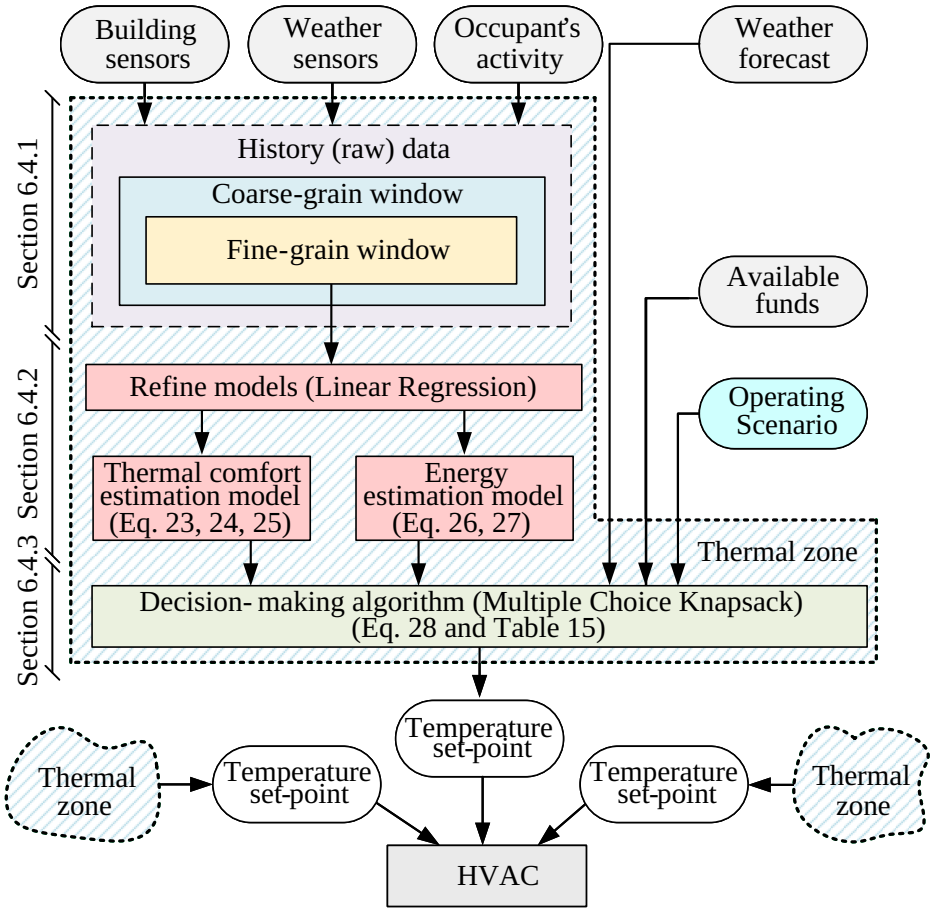


Figure 61: Proposed LR-Knapsack decision-making orchestrator.

where the temperature set-points per thermal zone are computed using a Multiple-Choice Knapsack algorithm. As described in Section 6.4.3 the algorithm can support different operation scenarios. The operation scenario as well as the available funds for heating/cooling are additional inputs to the decision making component.

6.4.1 Efficient manipulation of history data - Sliding window approach

The effective selection of data subset used for the analysis purposes is a very important part of our framework. Although one might expect that additional data improves the models' accuracy, this is not the case because this improvement degrades with non-correlated data and imposes constraints to the refinement procedure. Moreover, since the amount of data (stored to

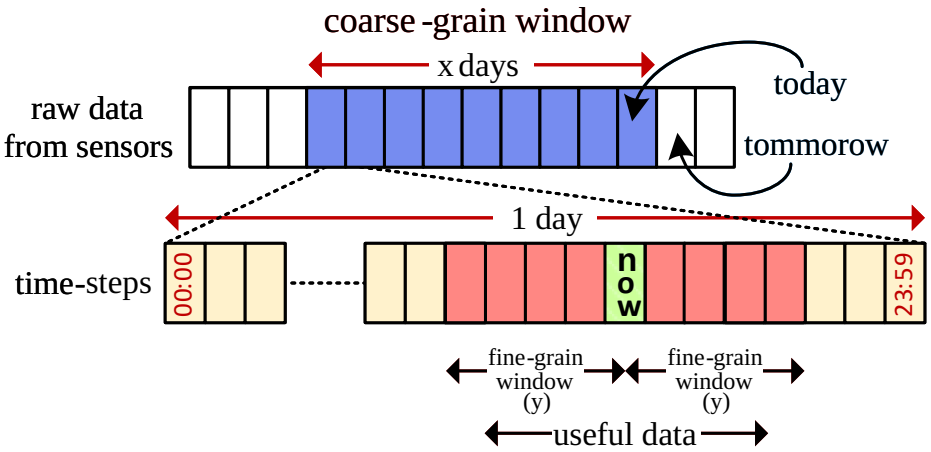


Figure 62: The concept of *coarse-* and *fine-grain* sliding windows

support the models) increases linearly with the execution time, data storage and processing are becoming challenging aspects especially at the embedded domain. In order to overcome these drawbacks, our framework employs two complementary mechanisms, namely the *coarse-* and *fine-*grain sliding windows, respectively.

This combination of sliding windows is used to determine and keep the "useful" data that will be considered for energy and thermal comfort prediction. Figure 62 depicts the functionality of sliding windows schematically. By utilizing this mechanism, instead of using the raw historical data acquired from sensors, the *coarse-grain* sliding window considers only data that refer to the last x days. A further improvement is achieved by using the *fine-grain* window, which selects a subset of this data referring to a specific time-slot (time period) per day (within the last x days configured by the coarse-grain window). As a result, instead of manipulating the entire history data, the combination of coarse-/fine-grain windows selects only a small subset for the model's refinement task. In other words, we keep only the data from the last x days with timestamps from $Now - y$ to $Now + y$, where Now is the timestamp that we invoke the framework [187].

We should mention that sliding windows are part of the solution and not an optimization step. Without the use of the slding windows, the excessive amount of data would lead to sub-optimal solutions. Both the coarse-grain window size, as well as the amount of data per day (fine-grain window), are adaptive in order to balance the problem's scalability and the availability of hardware resources. Finally, we have to state that the coarse-grain window's size defines the storage requirements for the proposed controller.

The sizes of these windows are defined after a detailed exploration un-

der typical weather conditions and residents' activity and the results of this analysis along with additional details about the efficiency of this selection in term of improving model's accuracy are provided in upcoming paragraphs.

6.4.2 Energy and Thermal comfort estimation (Linear Regression)

In these paragraphs, we describe the construction as well as the procedure for refining the thermal comfort and HVAC energy consumption estimation models. As mentioned above, both models rely on *linear regression* (LR) techniques. By using LR we compute the relationship between the dependent variable y that we desire to estimate (energy or thermal comfort) and a number of explanatory variables that correspond to the values of buildings sensors. The input to this procedure is the subset of historical data, as they are retrieved from the fine-grain sliding window, described in Section 6.4.1. Refinement is applied periodically to the employed models.

6.4.2.1 Thermal Comfort Estimation Model Since thermal comfort depends among others on air temperature, humidity, radiant temperature, air velocity, metabolic rate, and clothing levels, while each person experiences these sensations a bit differently based on his/her physiology and state, it is assessed by subjective evaluation.

For the scope of the work presented in the context of this dissertation, we propose a new model to quantify the residents' satisfaction by correlating the impact of thermal zone's temperature to the PPD (Predicted Percentage of Dissatisfied) value computed by the widely-used Fanger model [1]. The presented model is a function of the thermal zone's temperature and this choice was selected since the rest parameters (e.g. metabolic rate, clothing, etc) are difficult to be measured and they can be considered as constant for a relatively short period of time [209].

Figure 63 plots the result of this analysis for a representative winder day. Although this information refers to the winter period, similar results are also observed during summer. Based on this analysis, the reference estimation solution (blue color curve) can be approximated by the square function (red color) given at Equation 23, where the T^{in} parameter refers to the indoor temperature of the thermal zone. T^{in} is configured by the smart thermostat's set-point and, for the majority of cases, the difference in temperature set-points between consequent time-steps is less than 1^oC. So, we can claim that T^{in} for time-step t is assumed to be equal to the selected temperature set-point for time-step $t-1$ (as the HVAC system follows the smart thermostat's configuration). Hence, by refining the θ_1 and θ_2 weights iteratively, the

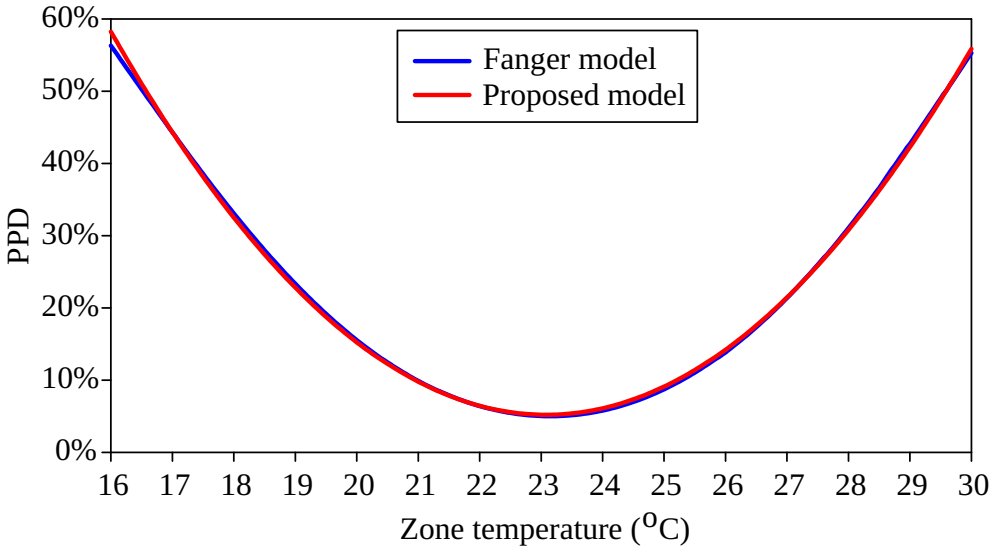


Figure 63: PPD estimation with the proposed and the reference Fanger [1] models as a function of thermal zone's air temperature (T^{in})

estimator minimizes the error (J) for the optimization problem presented in Equation 24.

$$C_{est} = \theta_0^c + \theta_1^c \times T^{in} + \theta_2^c \times T^{in^2} \tag{23}$$

$$J_c(\theta^c) = \sum \Big(C_{real} - C_{est} \Big)^2 = \Big\| C_{real} - X_c \times \theta^c \Big\|^2 , \tag{24}$$

where

$$\theta^c = \begin{bmatrix} \theta_0^c \\ \theta_1^c \\ \theta_2^c \end{bmatrix}$$

$$X_c = \begin{bmatrix} 1 & T_1^{in} & (T_1^{in})^2 \\ 1 & T_2^{in} & (T_2^{in})^2 \\ \vdots & \vdots & \vdots \\ 1 & T_m^{in} & (T_m^{in})^2 \end{bmatrix}$$

$$C_{real} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_m \end{bmatrix}$$

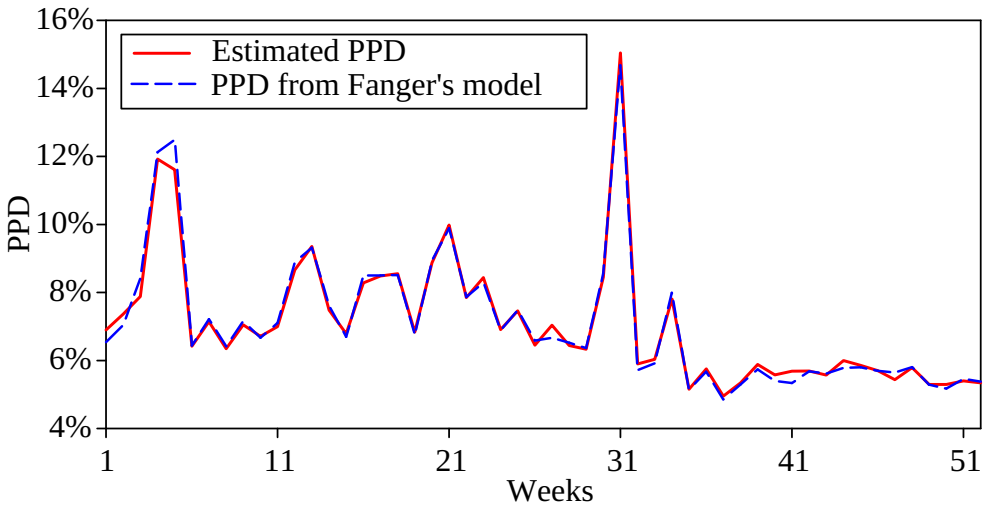


Figure 64: Evaluate the accuracy of the proposed thermal comfort model.

As mentioned above, the concept of the coarse- and fine-grain sliding windows enables the proposed thermal comfort estimation model, due to the fact that the majority of Fanger model's factors (e.g., metabolic rate, clothing, etc) are almost constant for the last few days, while the weather (e.g., solar radiation) and the residents'-related metrics (e.g., number of people, activity, etc) can be considered similar for a given time-slot between consecutive days.

The number of features used by the linear regression model is relative small. More specifically, in our case, the features are 1, T^{in} and T^{in^2}. As a result, it is possible to calculate the weight vector θ^c with the normal equation formulated by Equation 25. The computational complexity of the employed method is $O(n^2 \times m)$, where m and n refer to the rows (number of history data after applying the sliding windows data manipulation) and the columns (number of features) of Table X_c, respectively.

$$\theta^c = \left(X_c^T \times X_c\right)^{-1} \times X_c^T \times C_{real} \tag{25}$$

The accuracy of the proposed thermal comfort estimation model is evaluated against the corresponding results from the Fanger reference solution [1] and the results are plotted in Figure 64. The outcome of this analysis indicates that the average error between these two models is about 0.02% for the entire year. As a result, we might conclude, that the proposed thermal comfort model can be considered accurate enough for the smart thermostat use-case.

6.4.2.2 Energy Consumption Estimation Model This section introduces the proposed method for performing building-agnostic HVAC energy predictions and quantifying the impact of the selected temperature set-points on energy consumption. We focus on providing a Plug&Play solution that accomplish self-adaptive customization based on a limited amount of data acquired from building's sensors at run-time. Since metering devices measure only already consumed energy, such a prediction model is absolutely necessary. We assume that each smart thermostat constructs its own model in order to consider the parameters from its associated thermal zone.

As inputs for our model we consider the data acquired by building's sensors together with the hourly weather forecast regarding the next hours (for predictions far in the future). The sensors data include *Building's outdoor conditions* and more precisely the current weather conditions and weather forecast (i.e., temperature and solar radiation), as well as *Building's indoor conditions*, that include zone's temperature, thermostat's configuration set-point, etc. Additionally, our method relies on a small dataset that includes the "*useful data*" extracted from the sliding windows data management mechanism introduced in Section 6.4.1. The data are fed as inputs to the linear regression task in order to predict HVAC's energy consumption. It is worth highlighting that apart from the selected linear regression, any other model with similar functionality could also be applied for this task. However, the selection of linear regression was based on our objective for deriving a low-complexity solution, which can be implemented onto low-cost embedded devices.

The energy consumption estimation model is described by Equation 26, while (similarly to the thermal comfort model) its formulation with normal equations is described in Equation 27. The $\theta_0^e \dots \theta_4^e$ are the weights that are (re-)calculated, while T^{out}, T^{in}, R and S refer to the outdoors temperature, the indoors temperature, the solar radiation and the thermostat's set-point respectively. For performing predictions for the next hours, future values of T_{out} and R parameters are acquired from the weather forecasts. Regarding the indoor temperature T^{in} for time-step t, similarly to the thermal comfort model, we assume that it is equal to the HVAC's configuration set-point for the time-step $t-1$.

$$E_{est} = \theta_0^e + \theta_1^e \times T^{out} + \theta_2^e \times R + \theta_3^e \times T^{in} + \theta_4^e \times S \tag{26}$$

$$\theta^e = \left(X_e^T \times X_e\right)^{-1} \times X_e^T \times E_{real} \ , \tag{27}$$

where

$$\theta^e = \begin{bmatrix} \theta_0^e \\ \theta_1^e \\ \theta_2^e \\ \theta_3^e \\ \theta_4^e \end{bmatrix}$$

$$X_e = \begin{bmatrix} 1 & T_1^{out} & R_1 & T_1^{in} & S_1 & \\ 1 & T_2^{out} & R_2 & T_2^{in} & S_2 & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & T_m^{out} & R_m & T_m^{in} & S_m & \end{bmatrix}$$

$$E_{real} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_m \end{bmatrix}$$

The energy estimation model, which is designed for the purposes of the present dissertation follows a linear approach, similarly to relevant literature [191] [97] [96] [210]. For building this model, we must make the assumption that energy consumption is a linear combination of the selected features (Equation 26). To test the validity of this assumption, two diagnostic plots (namely the Q-Q and the Residuals-vs-Fitted plots) were used, as they are depicted in Figures 65(a) and 65(b). Based on the QQ plot, we observe that the residuals of energy model (red color dots) are normally distributed, a trend that is pre-requested for constructing a linear model [211] [212]. The vertical axis at the Residuals-vs-Fitted graph gives the residuals of energy consumption, while the horizontal one corresponds to the fitted values. Since the values at this graph are almost equally distributed around the horizontal line and there are no patterns, we might claim that our assumption about energy model's linearity can be confirmed.

The accuracy of the proposed model as compared to the actual energy reported from metering sensors is evaluated in Figure 66. Based on this study, the presented model exhibits a very small average error for the entire year (about 2.5%), which is considered acceptable for the scopes of our analysis. The overall complexity of the energy prediction method is $O(n_e^2 \times m)$, where n_e and m refer to the columns and rows of table X_e, respectively. As a result, by configuring the selected window sizes, the associated complexity can be sufficiently supported by the majority of existing low-cost embedded devices.

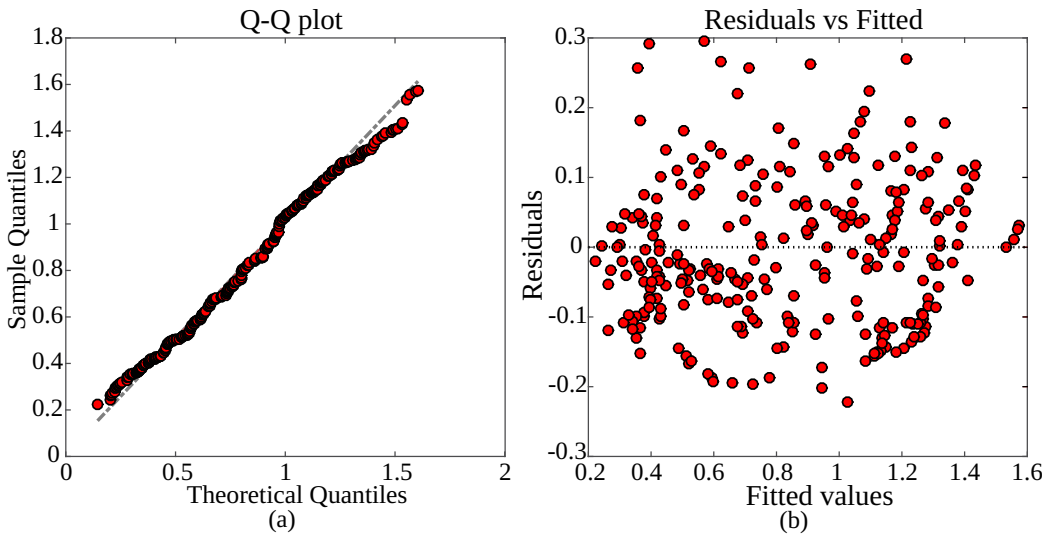


Figure 65: Energy consumption model's linearity selection: (a) based on Q-Q plot, and (b) based on the Residuals-vs-Fitted graph.

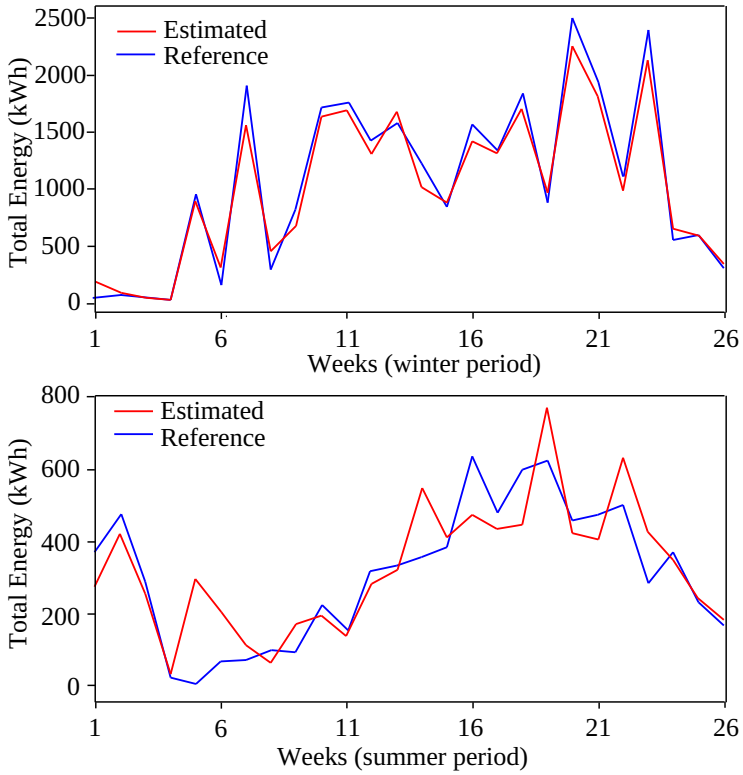


Figure 66: Evaluation of the proposed energy consumption model accuracy

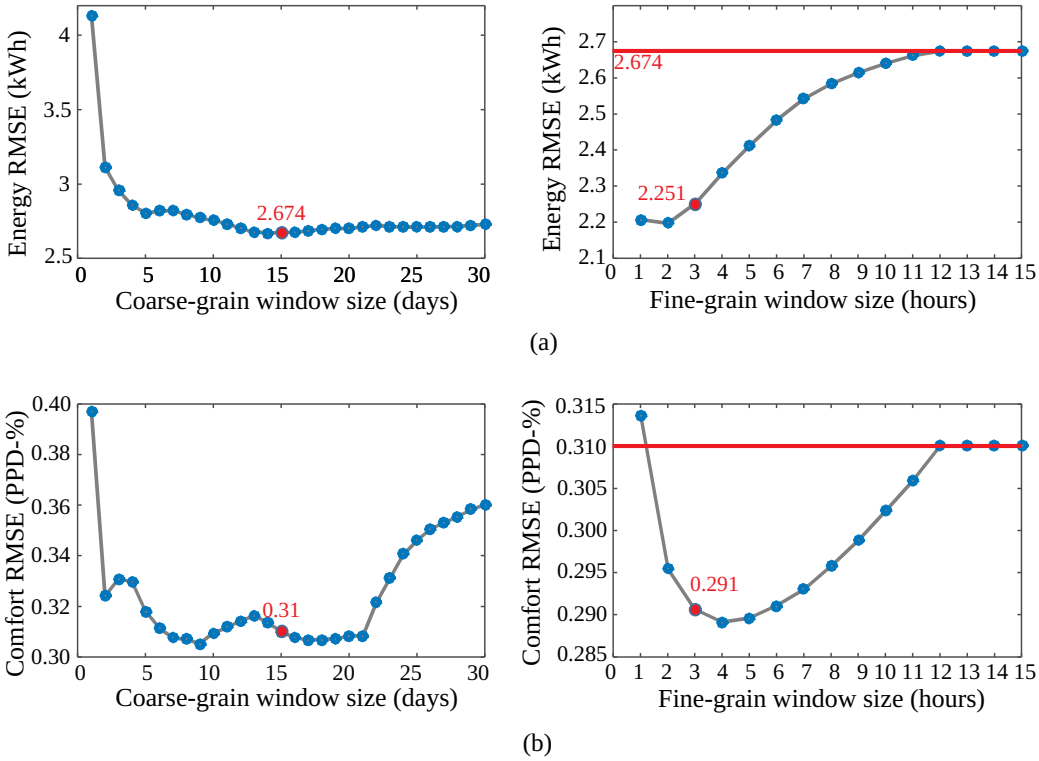


Figure 67: RMSE analysis for quantifying the impact of coarse- and fine-grain window sizes to: (a) the energy estimation model accuracy, and (b) thermal comfort estimation model accuracy

6.4.2.3 Sliding window data management impact on estimation models The previously discussed results regarding the thermal comfort and energy consumption estimation (Sections 6.4.2.1 and 6.4.2.2) are retrieved for the selected sliding window sizes discussed in Section 6.4.1.

The size of coarse and fine-grain windows are selected in a way that maximizes the final control results (as we will discuss later). The efficiency of the comfort and energy estimation models is a critical part of our solution, since the results of these models guide the decisions of our controller. Therefore, in order to validate the efficiency of our selection, we have to evaluate the accuracy of the estimation models for different window sizes. Figures 67(a) and 67(b) plot the average value (among buildings' thermal zones) of Root-Mean-Square Error (RMSE) for the thermal comfort and energy consumption models. For demonstration purposes, we highlight the corresponding values for the selected window sizes. The selected window sizes for coarse- and fine-grain windows must reduce the RMSE for both models. Based on this

analysis, slightly different results are observed for the two metrics, however, we might claim that a selection that leads to very good accuracy for both models is achieved for coarse-grain and fine-grain window sizes equal to 15 and 3, respectively.

6.4.3 Decision-making algorithm (Knapsack)

The outcomes from the comfort and energy estimation models feed the last phase of the HVAC control framework, which is responsible for making the system's decisions. The functionality of the system's orchestrator is based on a modified Multiple-Choice Knapsack Problem (MCKP) [124] [125]: "*Given K classes of items $N_1, N_2, \ldots, N_K$, where each item $j \in N_i$ is associated with a profit/value v_{ij} and a weight w_{ij}, the orchestrator aims to choose exactly one item from each class so that the total weight is less than, or equal, to a given capacity W and the total value is minimized*". The mathematical formulation of our problem is given in Equation 28. Our goal is minimizing (instead of maximizing) the total value and this is the main differentiation compared to conventional Knapsack problem.

$$\begin{aligned} & min \sum_{i=1}^{K} \sum_{j \in N_i} v_{ij} x_{ij}, \\ subject\ to & \sum_{i=1}^{K} \sum_{j \in N_i} w_{ij} x_{ij} \leq W, \\ & \sum_{j \in N_i} x_{ij} = 1, i = 1 \ldots K, \\ & x_{ij} \in \{0,1\}, i = 1 \ldots K,\ j \in N_i \end{aligned} \tag{28}$$

At the Knapsack problem's notation, each class refers to a time-step of the smart thermostat's operation. An item represents the temperature set-point regarding the aforementioned time-step, which is associated with a PPD value and an energy cost. Table 15 summarizes the correspondences between the proposed MCKP algorithm solution and the HVAC control problem's parameters. For the sake of completeness, three different operating scenarios are presented in order to show the capabilities of the employed solution (Equation 12) as presented also in Section 6.1:

- *Scenario 1*: Achieve a compromise between energy consumption and thermal comfort metrics ($tr = 0.5$).

Table 15: Applying the knapsack formulation to the HVAC control case-study

	Knapsack formulation	Case-study	
	Class	Time-step	Objective
	Item	Temperature set-point	
Scenario 1	Item's value (v_{ij})	tr× Energy + (1-tr)× PPD	$\min(\sum_{\forall t}\left(tr \times \sum_{i=1}^{i=k} E_i^G(t,a_t^i) + (1-tr) \times \sum_{i=1}^{i=k} C_i(t,a_t^i)\right))$
	Item's weight (w_{ij})	0	
	Knapsack's capacity (W)	∞	
Scenario 2	Item's value (v_{ij})	Energy	$\min(\sum_{\forall t}(\sum_{i=1}^{i=k} E_i^G(t,Sa_t^i))),$ while $\sum_{i=1}^{i=k} C_i(t,a_t^i) < C_{limit}$
	Item's weight (w_{ij})	PPD	
	Knapsack's capacity (W)	PPD_{lim}	
Scenario 3	Item's value (v_{ij})	PPD	$\min(\sum_{\forall t}(\sum_{i=1}^{i=k} C_i(t,a_t^i))),$ while $\sum_{\forall t}(\sum_{i=1}^{i=k} E_i^G(t,a_t^i))) < AF$
	Item's weight (w_{ij})	Energy	
	Knapsack's capacity (W)	AF	

- *Scenario 2*: Minimize energy consumption while respecting a minimum threshold for the PPD metric ($tr = 1$).

- *Scenario 3*: Optimize thermal comfort ($tr = 0$) without exceeding a predefined energy budget for the experiment's duration (mentioned as available funds or AF).

Figure 68 visualizes the proposed version of MCKP algorithm for the operating Scenario 3. For demonstration purposes (at this figure) we assume that each time-step corresponds to a day. In addition, we highlight the parameters discussed in Table 15, namely the "*classes*" (time-steps), "*items*" (set-points), "*item's value*" (the PPD value per item), "*item's weight*" (the corresponding energy per item), as well as the "*Knapsack's capacity*" that refers to the available energy budget (AF). We also assume that we want to calculate the temperature set-points one week ahead. The decision-making procedure is applied once per day in order to refine the thermostat's selections based on the available funds (AF).

The MCKP algorithm is an NP-hard problem. The employed solution initially solves the simplified linear MCKP problem and then expands the core solution based on dynamic programming and by adding the necessary classes [124]. In contrast to relevant state-of-the-art solvers, such as the dynamic programming [125] that exhibits pseudo-polynomial time complexity, the solution discussed throughout this Chapter minimizes the problem's complexity by considering only a few items. More precisely, the computational complexity for the decision-making algorithm is $O(n+W \times \sum_{N_i' \in c} n_i)$, where

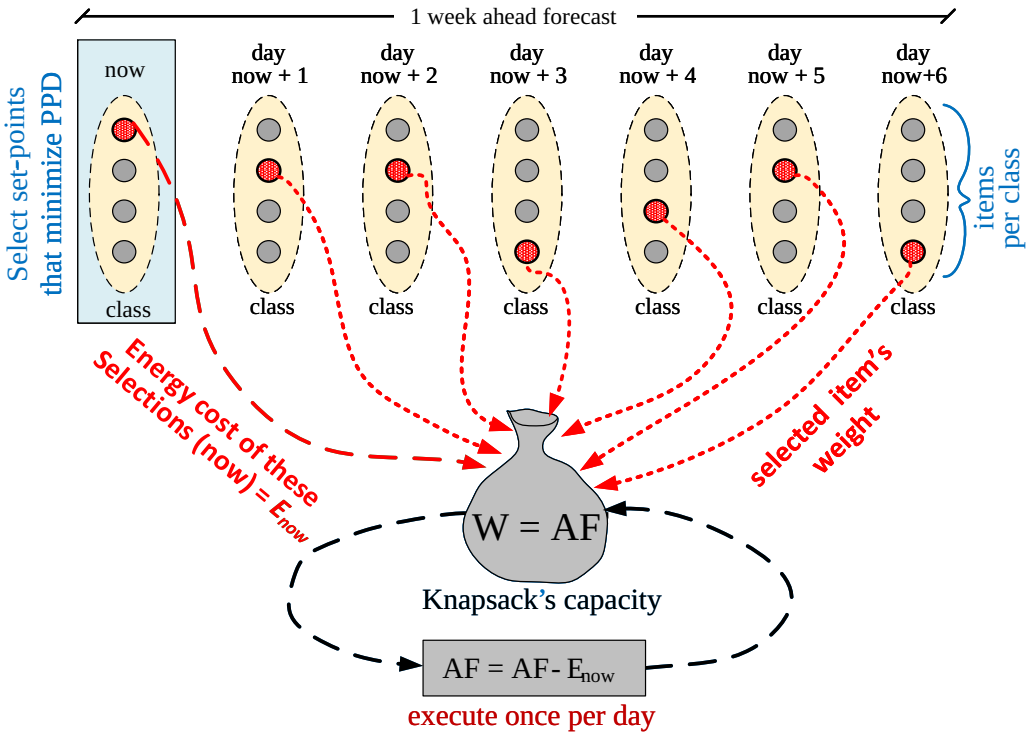


Figure 68: Virtualization example of the proposed Multiple-Choice Knapsack Problem (MCKP) approach to the HVAC control problem

n is the number of total items, n_i the number of items in class N_i' and c is the core solution that contains only the classes and items that constitute the solution space retrieved from [124].

6.5 Experimental Results

6.5.1 Results on minimizing cost

As mentioned above, the proposed sliding windows data manipulation is part of the introduced solution and not an optimization step, as otherwise the excessive amount of data leads to sub-optimal solutions for the model's refinement problem. The sizes of the coarse and fine-grain windows were defined with a detailed exploration under typical weather and residents' activity, in a way that optimizes the energy and thermal comfort model accuracy (see Section 6.4.1 and Figure 67). Figure 69 presents the impact of the selected windows sizes on the overall cost computed by Equation 12 (in normalized manner). Based on this analysis, the selected sizes (equal to 15 for coarse-grain and 3 for fine-grain window) lead to optimal performance

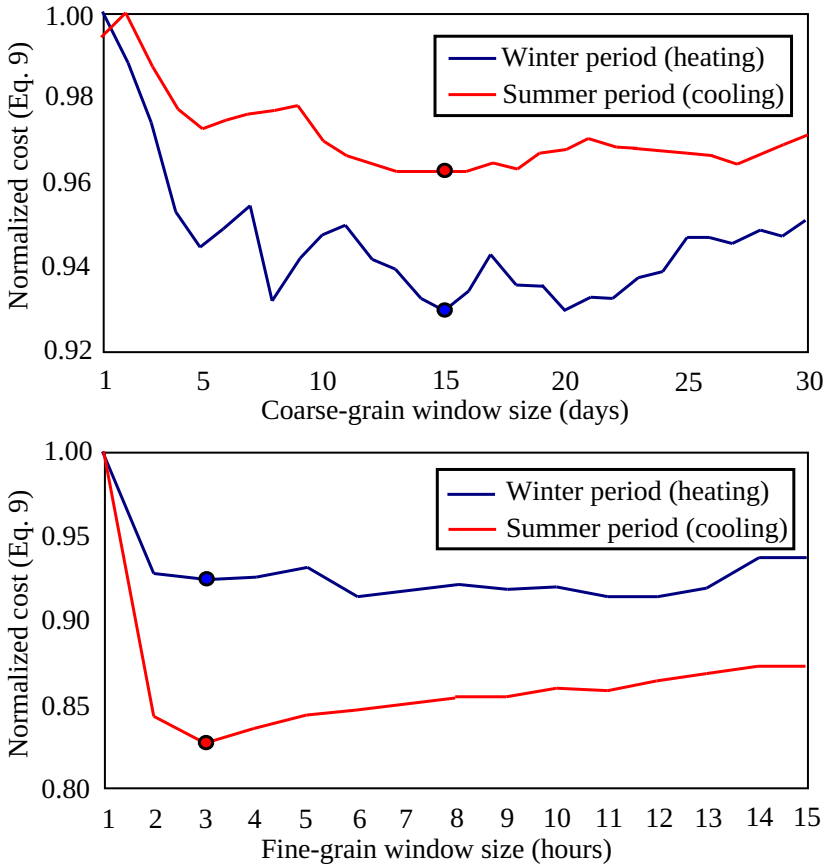
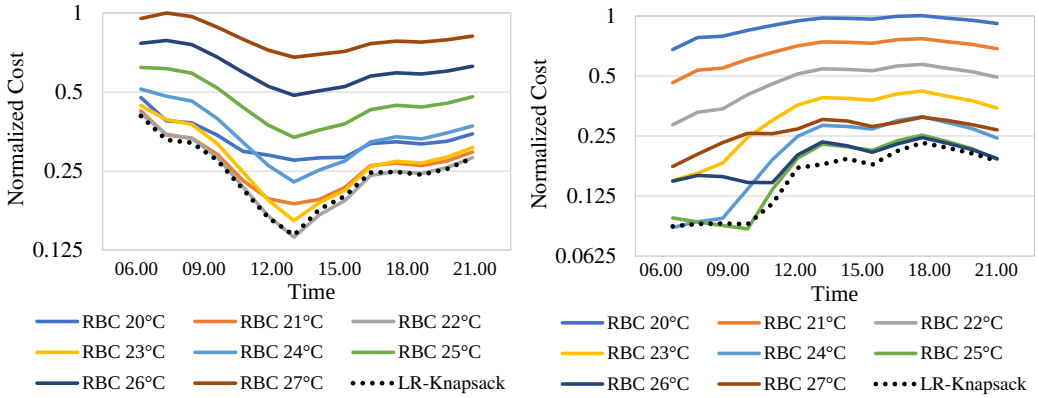


Figure 69: Evaluate (based on Equation 12) the efficiency of alternative coarse- and fine-grain sliding windows.

(minimization of total cost).

A number of quantitative comparisons that highlight the superiority of introduced solution are provided in this section. The proposed decision-making mechanism computes temperature set-points for the micro-grid case study discussed in Table 11. The reference solutions to this analysis are the Ruled Based Configurations (RBCs). As mentioned earlier, RBC is the typical approach for thermostat configuration. The RBC configuration aims to achieve a constant temperature to the target thermal zone. The majority of commercially available thermostats are configured with consecutive RBCs that differ 0.5°C or 1.0°C. ranging from 20°C up to 27°C. The experimentation for this analysis refers to a 52-weeks duration (winter, spring, summer and autumn) in order to evaluate the stability of the proposed orchestrator.

Figure 70 quantifies the quality of decisions derived from the introduced decision-making algorithm as compared to the reference RBC solutions. Ver-



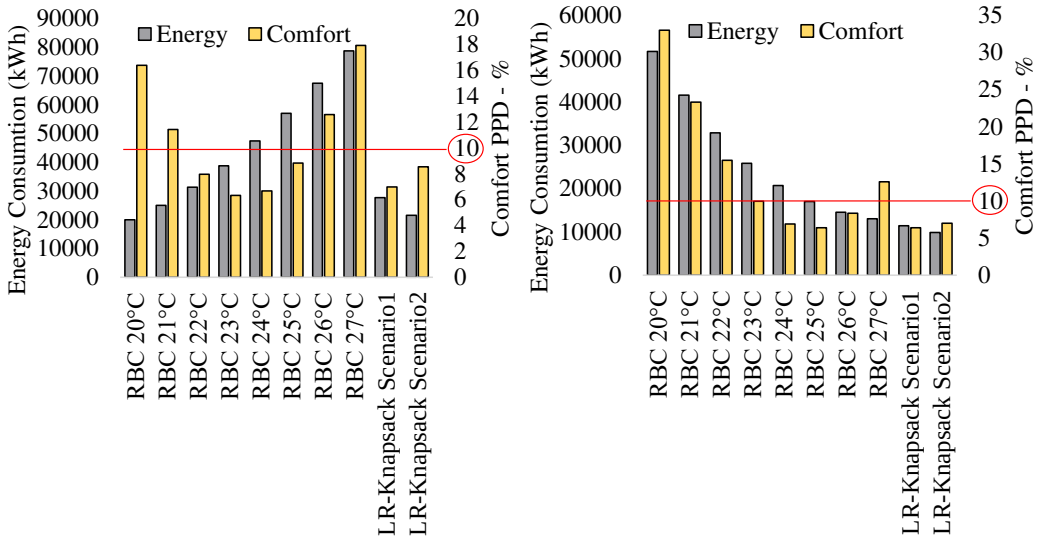
(a) Winter day (b) Summer day

Figure 70: LR-Knapsack efficiency against RBCs (balanced scenario)

tical axis corresponds to the normalized cost (according to Equation 12) for a representative winter and a representative summer day, in order to consider both building's cooling and heating. According to this analysis, we conclude that the proposed framework achieves to minimize the overall cost as compared to RBC values. Moreover, the temperature set-points computed by the proposed methodology improve the cost metric as compared to RBCs for the presented winter and summer days by 31% and 46%, on average, respectively.

In the next experiments, we quantify the efficiency of the presented orchestrator to minimize energy cost while respecting a minimum threshold for PPD metric, presented as Scenario 2 in Table 15. For this purpose, we set the ASHRAE standard limit, according to which temperatures leading to PPD values up to 10% are acceptable by residents [132]. So, the optimization goal for this experiment is to minimize the energy consumption ($tr = 1$) under the respect of ASHRAE standard (PPD value less or equal to 10%). This scenario enables considerable flexibility to the controller's selections, since further reduction of PPD's value imposes increased energy consumption for heating/cooling.

Figure 71 evaluates the efficiency of this operating mode as compared to the corresponding selections from the operating Scenario 1 and the RBC configurations. Also, for demonstration purposes, we highlight the 10% PPD threshold, as it is defined by the ASHRAE standard. Based on these results, our orchestrator achieves an average reduction at energy consumption by 48% compared to the RBC selections that achieve an average PPD value under 10%. More precisely, for the winter period, we achieve almost equal energy consumption to keeping the set-point constant on 20°C but with a



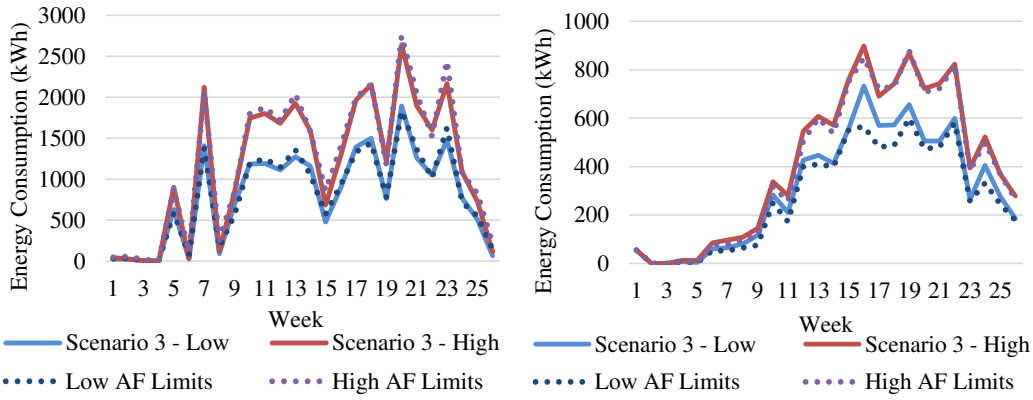
(a) Winter (b) Summer

Figure 71: Evaluate Energy and PPD variation for the alternative Scenario 2 (optimize energy keeping acceptable PPD) vs RBC values

thermal comfort value of 8.5% instead of the unacceptable 16.4% of RBC 20^{o}C. Note that Scenario 2 does not lead to a PPD value of exactly 10% but even better, due to the weather conditions. For example in summer if the weather conditions lead to PPD under 10% without the need of HVAC cooling, which is a case observed especially in the night or early in the morning, then our orchestrator keeps the HVAC system off, maintaining good thermal conditions and saving energy. According to the previous findings, we claim that the presented decision-making algorithm can address the goal of maintaining affordable indoor thermal conditions with the minimum possible energy cost efficiently.

The third Scenario tested in the context of this dissertation, considers a case where the smart thermostat optimizes the thermal comfort (PPD) without exceeding the available energy budget (available funds - AF) ($tr = 0$ at Equation 12). As it has already mentioned in Section 6.1, if the energy from renewable sources is not enough to meet the residents' demand, additional electricity is purchased from the main-grid (by decreasing the value of AF parameter).

This problem is addressed with the Multiple-Choice Knapsack algorithm discussed in Section 6.4.3. For our analysis, and without affecting the generality of the introduced solution, we consider random weekly reference values for the AF metric in a range that partially takes into account the weather



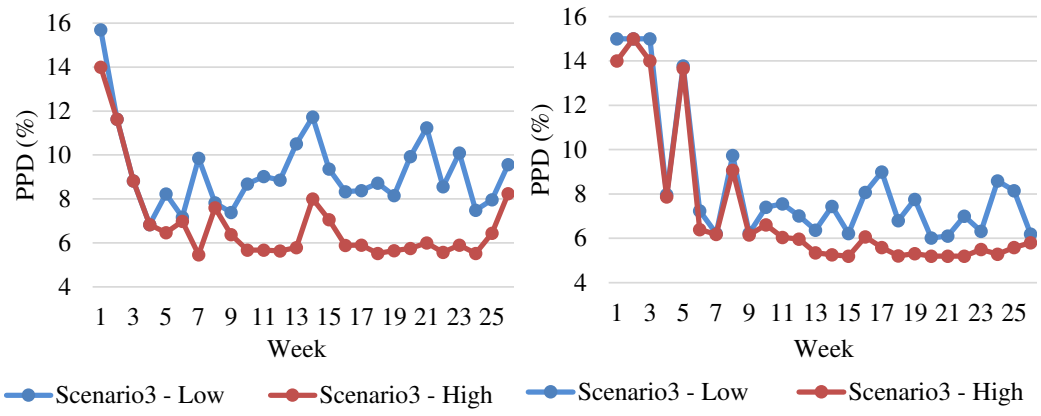
(a) Winter (b) Summer

Figure 72: Evaluate Energy variation for alternative Scenario 3 (optimize energy keeping total energy under AF limits)

conditions (more funds for the coldest weeks in winter and the warmest weeks in summer) and around the nominal HVAC heating and cooling energy values respectively. Two instantiations of Scenario 3 are evaluated throughout this paragraph, where the initial energy budget (AF) is set to "Low" (underestimate scenario) and "High" (overestimate scenario).

In order to evaluate the efficiency of this scenario, Figure 72 evaluates energy for the aforementioned operating scenarios. The results indicate that the proposed orchestrator respects the AF limit in most cases expect only some weeks for the "Low AF" case in summer. The reason is that in these weeks the weather was too hot at noon to meet the PPD limits without exceeding the energy limits. So the controller had to consume a little above AF to satisfy the comfort constraints. Apart from the variation of energy metric, we also evaluate the PPD metric for the studied operating scenarios. These results, depicted in Figure 73, indicate that the underestimated version of our algorithm exhibits on average 36% higher PPD value compared to the "High AF" solution. The few spikes where the PPD metric exceed the 10% threshold defined by ASHARE standard are on periods that the HVAC system is off (Octomber and April-May as also can be seen in Figure 72 that the energy is zero).

More experiments will be presented in Section 6.5, where the proposed solution will be also compared to the current state-of-the-art simulation-based method.



(a) Winter (b) Summer

Figure 73: Evaluate PPD variation for alternative Scenario 3 (optimize energy keeping total energy under AF limits)

Table 16: Execution run-time for computing temperature set-points per thermal zone regarding the proposed orchestrator.

	ARM Cortex-A57 @2 GHz	Intel Quarkx1000 Intel @400MHz	ARMx STM32F103 @72MHz
Scenario 1 & Scenario 2	0.004 sec	0.006 sec	0.082 sec
Scenario 3	0.03 sec	0.035 sec	0.576 sec

6.5.2 Orchestrator's Performance on Embedded Devices

Finally, we present a number of experimental results regarding the physical implementation of the introduced LR-Knapsack orchestrator. For evaluation purposes we evaluate the previously mentioned controller to various architectures, including micro-controllers (ARMx STM32F103@72MHz) and embedded processors (ARM Cortex-A57@2GHz - see Section3), Intel Quarkx1000@400MHz).

Table 16 summarizes the execution time results of our orchestrator for the three operating scenarios presented in Table 15. These results highlight that our decision-making framework is able to compute a temperature set-point per building's thermal zone in less than a second, even in the case where the target platform is a low-performance micro-controller (ARMx STM32F103@72MHz). We have to mention that the execution time for the Scenario 3 is higher because temperature set-points are computed for all the thermal zones a week ahead. To sum up, this analysis indicates that one low-cost platform can be used to control in optimal way indoor temperatures

Table 17: Evaluation of yearly results against other methods.

Method	Energy (kWh)	Avg. PPD (%)	Cost (Equation 4)
RBC 20°C	66,967	24.99	0.89
RBC 21°C	62,939	17.46	0.72
RBC 22°C	61,223	11.66	0.59
RBC 23°C	61,955	7.94	0.52
RBC 24°C	65,191	6.46	0.51
RBC 25°C	70,467	7.23	0.56
RBC 26°C	77,359	10.22	0.66
RBC 27°C	85,680	15.31	0.81
Fmincon	34,936	6.17	0.33
RL	34,601	7.71	0.36
LR-Knapsack	36,399	6.47	0.34

for multiple thermal zones [213].

6.6 Comparison to state-of-the-art methods

Table 17 quantifies the impact of the orchestrator's selections in terms of the total energy, the average comfort and total weighted cost. The reference solutions to this comparison are the Ruled Based Configurations (RBCs) ranging from 20°C up to 27°C (already presented in Section 6.4), as well as the well-established *Fmincon* solver [203] (see Section 6.2.1). We assume that both the building's model as well as the weather forecasts used by the fmincon solver are 100% accurate, due to the fact that we evaluate our controller at the same "simulated" model. This means that the results of the Fmincon solver can be considered as *the optimal results* in the context of this study.

According to Table 17, the introduced orchestrators achieve superior performance against RBCs, as it leads to overall cost reduction ranging from 40% up to 150%. Additionally, the introduced solutions exhibit comparable efficiency against relevant implementations (e.g. Fmincon), but without the limitations of these solutions that restrict their general-purpose applicability. More thoroughly, Fmincon solves the problem using iterative simulations, while the objectives (namely the energy consumption and thermal comfort) are known a priori through detailed modeling of the buildings' dynamics. Consequently, the efficiency of existing solvers is firmly tied to the assumption that CPS modeling was performed in advance. In addition, our orchestrator does not rely on such an iterative approach (it is executed only once), without considering any information about the employed objective

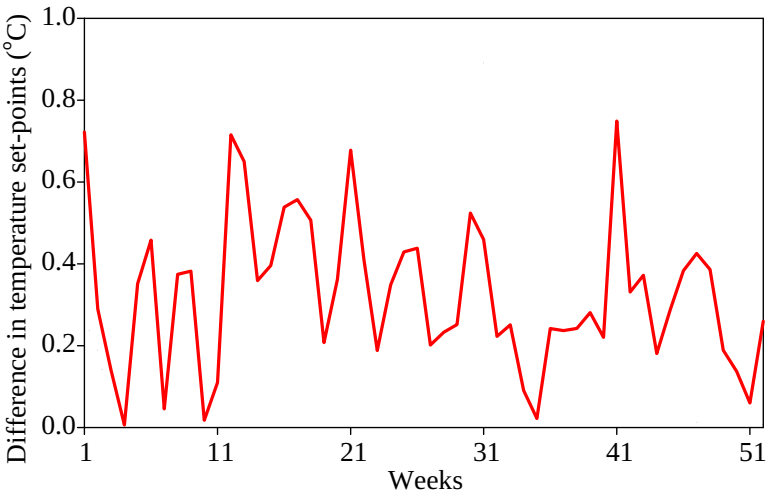


Figure 74: Difference (in absolute manner) between the LR-Knapsack orchestrator versus the Fmincon solver

functions.

The LR-Knapsack approach improves the overall cost by 5.5% compared to the "black-box" RL approach presented in Section 6.3. In addition, LR-Knapsack solution gives the capability of quickly changing operating modes as it was highlighted in Section 6.5 based on its individual energy and thermal comfort models, while the RL approach would be restarted from the beginning because the cost/reward definition would have been changed.

Figure 74 plots the mean absolute difference between temperature set-points computed from the LR-Knapsack based system's orchestrator compared to the reference solutions of the fmincon solver. In particular, the decisions from our orchestrator exhibit an average variation about 0.32°C compared to the Fmincon solver.

Among others, the exhaustive design space exploration performed by existing solvers (such as the Fmincon), leads to increased execution runtime (more than 10^{14} execution cycles, or equivalently 12 hours execution time, for simulating an 1-day building's operation experiment in Intel i7-6700K@4GHz), which makes its usage prohibitive for an embedded controller. Moreover, objective functions have to be known a priori. On the other hand, the proposed solution achieves to compute close to optimal operating set-points without any prior information about the problem's instantiation.

Figure 75 quantifies the number of execution cycles in order to compute temperature set-points per thermal zone regarding the operating Scenario 1. For this analysis, we consider that the underline device is a ARM-based embedded processor (ARM Cortex-A8) and a BEM system (Intel i7-

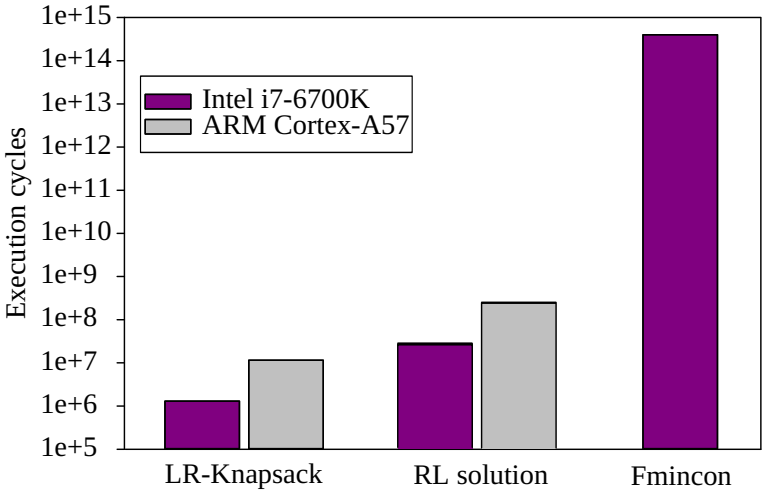


Figure 75: Minimum number of execution cycles for computing temperature set-points per thermal zone among alternative controllers.

6700K@4GHz). Based on this analysis, the proposed LR-Knapsack orchestrator exhibits significant lower complexity (about 8 orders of magnitude) compared to the Fmincon offline solver. For sake of completeness we have to mention that Fmincon cannot be executed onto an embedded platform (low-cost device), as it imposes EnergyPlus and Matlab-based simulations.

6.7 Conclusion

A framework for supporting the design of a low-cost CPS orchestrator targeting HVAC systems, was introduced. This orchestrator was applied to solve a multi-objective problem related to the simultaneous enhancement of buildings' energy consumption and the occupant's thermal comfort metric.

Based on our experimentation, we validate the superiority of the proposed solution against relevant solvers without the necessity of accurate prior system modeling, as both functions that describe energy consumption and thermal comfort are agnostic. Moreover, the introduced solution exhibits significant lower computational/storage complexities, which in turn enables its execution onto low-cost embedded devices.

Chapter 7

7 Discussion

In this Chapter we summarize the conclusions we have reached from the present study. The presented conclusions are accompanied by discussions and explanations of the methods and experiments that led us to them as well as possible limitations of the proposed solutions. Finally, we suggest extensions of the present work.

7.1 Conclusions

The construction of practical tools capable of analysing the source code of applications and estimating the expected energy consumption, without the need of executing the code on the targeted devices, is feasible.

The proposed energy estimation tools run on the programmer's workstation (PC). So, when the final application or part of it is ready, the developer can use the proposed tools to estimate the energy consumption that the application's code will consume, if executed on a list of embedded devices (cross-device). Of course, measuring energy directly on the targeted device would give the most accurate results. However, not all hardware alternatives are accessible to developers and such a process may involve sophisticated equipment (e.g. sensors). Furthermore, such a solution would need hardware expertise, increasing the development time and cost. Finally, this procedure is not feasible in very complex applications that involve a large number of devices/architectures.

Throughout Section 4, we presented a method for designing practical analysis tools to be used by developers for estimating energy consumption of applications running on CPU-based embedded devices. We provide a systematic methodology to create estimation models for various platforms.

The introduced framework uses random synthetic loops, popular profiling tools and regression methods. Static and dynamic analysis methods were designed and compared. Dynamic analysis leads to more accurate results

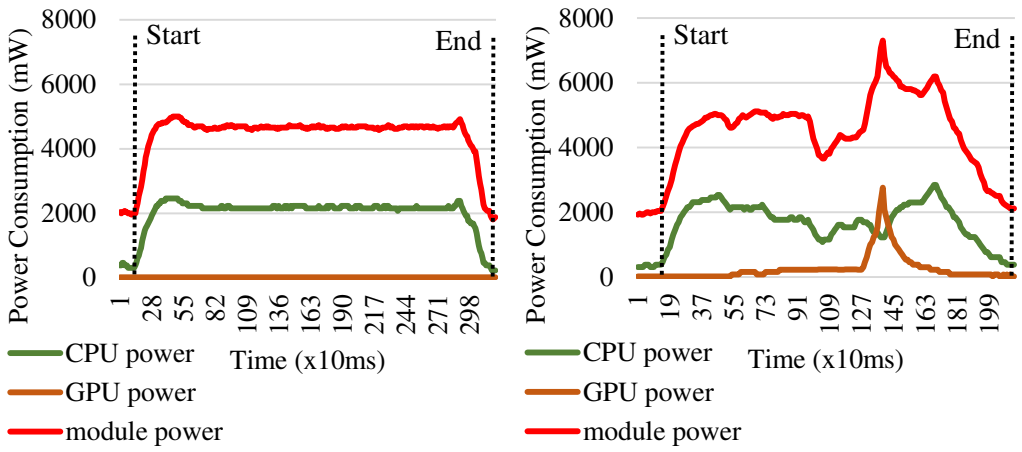
as expected ($R^2 = 0.96$). Its limitations are summarized in the fact that it requires application execution, adding also a large time overhead. On the other hand, static analysis, although less accurate ($R^2 = 0.92$), estimates energy in a fast, convenient, and user-friendly way. However, it also needs additional user input that is not always easy to obtain (e.g. the number of iterations of each loop body).

For demonstration purposes, each sub-component was evaluated on a widely used ARM-based device using well-known benchmark suites, while some results of the extension of the tool for partially supporting HPC systems is presented. Based on our experimentation, we evaluated the capabilities of the proposed solutions, concluding that they achieve acceptable results without requiring accurate prior hardware modeling. The proposed approaches achieve similar effectiveness compared to related state-of-the-art tools but focus on building an extensible solution that can be part of SDK tools. Our special emphasis on studying the correlation between alternative metrics and energy is expected to contribute to the creation of new software analysis metrics for energy consumption.

The prediction of energy consumption gain by acceleration in heterogeneous embedded devices is feasible.
Existing works focus on the prediction of speedup by offloading a piece of CPU code on GPGPUs [78, 79]. However, in the area of embedded heterogeneous systems, energy efficiency is equally important and affects design decisions. Based on the results presented in Section 5.3, we conclude that the existing approaches that analyze CPU code to provide speedup predictions can be extended towards predicting the energy gains by acceleration.

An important contribution of this thesis is the proposed combination of static and dynamic approaches in the same tool-flow. In addition, for the dynamic analysis component, a clarification of the difference between speed-up and energy gain estimation will highlight the importance of the presented extension of existing (speed-up) predictors towards estimating energy gains. One may argue that energy consumption can usually be estimated based on the prediction of execution time by using the power delay product or an analytical model. This is a reasonable question considering that if the average power consumed by the CPU and GPU does not vary and change dynamically, a performance gain will lead to a similar energy gain in most cases.

Although performance gains are usually similar to energy gains, this is not always the case, especially when we have heterogeneous systems such as CPU-GPU devices that we target in the context of this study. Indeed, power consumption is not constant because it depends on a wide range of parame-



(a) CPU version power consumption (b) GPU version power consumption

Figure 76: 3d convolution power consumption on Nvidia TX1

ters (memory transactions, cache behavior, number of cores used, etc.). This is also indicated in Figure 76. Figure 76a shows the power consumption of the CPU, GPU, and the entire module unit to run the 3d convolution application from the polybench benchmark suite on Nvidia Jetson TX1, while Figure 76b shows the same power consumption modules for the GPU version. Based on this experiment, we can observe that the additional power of the GPU leads to a higher instantaneous module power consumption compared to the CPU-only version, while the power consumption can not be considered stable at all. Based on these results, we can expect that the performance gains should be more than the energy savings in most cases and not with a fixed additional gain.

Studying and designing a model of the correlation between energy and performance for the specific platform would be also interesting but it is beyond the scope of this work. However, in the proposed study we wanted to show that it is easy to extend existing approaches towards estimating energy gains following the same procedure. Also presenting the importance of features from existing approaches in terms of relation to energy consumption (see Table 6) is also something new (to the best of the author's knowledge). Finally, a first method that is based entirely on static analysis and a tool-flow that combines the two methods are designed. We also consider this study as perfectly related to our focus on green, sustainable and energy efficient computing.

The overhead of dynamic instrumentation in cross-device energy prediction is large and it can be avoided by using static analysis

when maximum accuracy is not required.
As it is mentioned in the first conclusion, dynamic instrumentation although more accurate, adds a large time overhead that is a big disadvantage of making the proposed solutions a part of an SDK tool. In the case of applying static analysis, prediction results are instantly generated.

The time required to reach a prediction by using dynamic analysis reached even 10 hours for some applications included in our datasets. Special emphasis on this overhead was given in Section 5.3.6, where we presented the motivation behind combining static and dynamic analysis in predicting energy gains by acceleration.

As a result, the support of making some estimations using only static analysis is a significant advantage for application developers. However, the accuracy of static analysis is limited. Therefore, the combination of both techniques can compromise granularity and time overhead to reach a prediction, making the integration of energy estimation methods as part of a software development toolkit easier. For example in estimating potential gains by acceleration, the overhead of dynamic instrumentation can be avoided in cases in which high energy gains are predicted. In contrast to existing works ([78, 79]), the fact that the proposed methodology, predicts the cases for which relatively high gains are expected using only static analysis is a significant advantage for application developers.

The quantification of programming effort is important and its prediction is feasible.

Quantifying the programming effort is a very interesting research topic that is expected to support the design of software applications. Expressing programming effort in actual development time (e.g. hours) requires feedback from many developers working on the same projects, in order to make a clear and representative dataset. The purpose of Section 5.5.2 is to show that metrics, that have already been mapped with programming effort in the literature, can be estimated before the new version of the code is written. The main contribution of this analysis is not based on the use and monitoring of the Halstead metric itself, but rather on the design of a tool that predicts the effort required to develop a new (accelerated) version of the code, before development. To the best of the author's knowledge, this is a first attempt of using this metric (the increase of Halstead effort) to create a model that predicts GPU-accelerated programming effort using the CPU version of the code as the only input. In this direction, we observed that it is feasible to predict the effort required to develop the acceleration-specific code using only the CPU code as input, achieving a very high accuracy (85%). We might claim that such a solution can contribute significantly to support developers

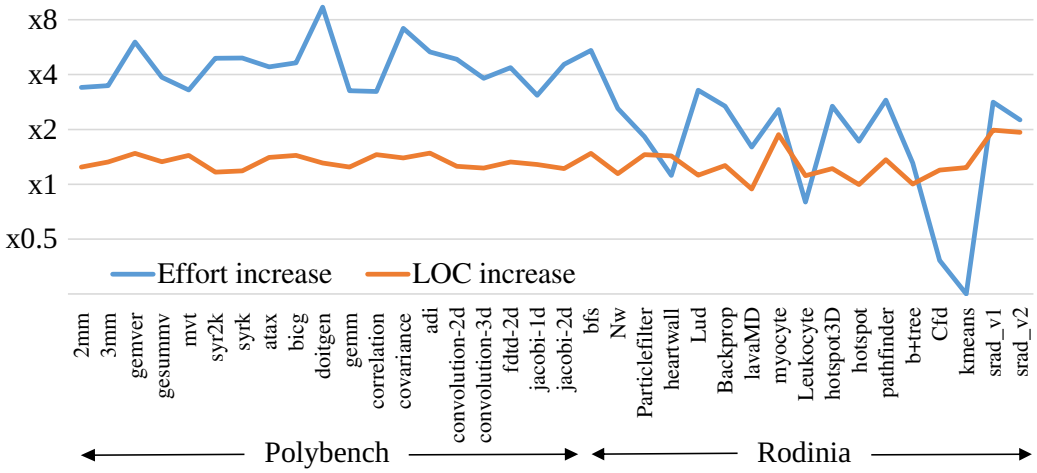


Figure 77: LoC and *Effort* actual increase of GPU version in comparison to CPU for Polybench and Rodinia benchmarks

on deciding whether to invest in acceleration. Results using the traditional LoC metric are also provided.

Figure 77 shows the actual LoC and *Effort* increase between the CPU vs. GPU versions for the Polybench and Rodinia applications. We notice that *Effort* ranges from x1 and x8, while LoC ranges from x1 to x2. However, in several cases the increase in LoC can be safely attributed to statements such as definitions and variable initialization, which cannot be directly related to programming effort. On the other hand, *Effort* is mainly affected by the number of operands, operators and function calls. Therefore, programming effort is more accurately expressed through *Effort*. Indeed, in Figure 77 we notice that there are applications in which there is a very small increase in LoC, however the effort increases by more than x4. This observation is inline with Shihab et al. that state that relying on LoC often leads to underestimation of programming effort [170]. Interestingly, there are some applications for which the effort seems to decrease. These applications belong to the Rodinia benchmark suite, as shown in Figure 77. The reason, as explained in Section 5.5.2, is the fact that the CPU versions of these applications provide functionality that is not present in the corresponding GPU version.

Extending the proposed energy estimation and optimization methods towards supporting additional devices is straightforward. There are no constraints in the presented methods with respect to portability. When accurate energy consumption measurement is feasible, the designed methods can be easily applied to other devices.

To apply the methodology to other platforms the following steps should be followed:

i) Downloading the datasets and performing energy consumption measurements for each application to generate datapoints

ii) Training of the static and dynamic analysis regression models.

iii) Design choices and methods calibration (e.g. definition of class boundaries between "high gains" and "moderate/no gains" for the prediction of gains by acceleration in Section 5.3).

Cross-device energy estimation methods support Energy-aware Function Placement on Edge devices (Resource management).

More and more applications are deployed on power-constrained Edge-computing devices. The minimization of the energy consumption of Edge resources becomes very important but any transformation towards saving energy may violate the Quality of Service (QoS), which is the main design goal. As a result, effective decomposing of applications into fine-grained functions and energy-aware placement on a cluster of edge devices that also guaranties QoS is a very important problem.

Nowadays, serverless computing offers a large flexibility and user-friendliness by utilizing fine-grained functions and can be easily adopted in edge computing environments. Developers can use a large variety of available frameworks to apply the serverless concept on an Edge infrastructure without needing deep knowledge of network protocols etc. These frameworks allow many optimizations from the resource management point of view.

As tested and demonstrated in Section 5.4, methods that exploit the output of the proposed software analysis tools (in this thesis) alongside with multi-objective optimization heuristics enable a more efficient resource management on the edge. The introduced tools presented in Chapters 4 and 5 provide function-level profiling, making cross-device energy consumption predictions, while it is easy to be used also for estimating execution latency on various platforms. The two objectives are the energy consumption and the QoS. As shown in Section 5.4, formulating the optimization goal as a multi-objective problem can help developers minimize energy consumption while also satisfying the QoS guarantees for latency-critical applications. The results are very promising showing an average 33.6% of energy savings for using 4 edge devices to place 25 functions, or 8.5% for using 3 edge devices, giving the directions for future research and the development of tools that can be easily integrated in the overall framework proposed in this dissertation.

Online CPS orchestrators are very important for controlling existing systems. Low-complexity statistical models and multi-objective optimization algorithms can contribute towards this direction. Designing smart thermostats capable of controlling heating/cooling in existing buildings without requiring prior modelling is a typical example.

In the context of this thesis, a framework for rapid prototyping and implementation of low-cost orchestrators for Cyber-physical systems, was introduced. More specifically, for demonstration purposes, the framework was employed to control HVAC configuration in a micro-grid environment. The introduced system is a model-free, plug&play solution of the problem of HVAC thermostat's set-point scheduling. The user can set the preferred balance between energy savings and thermal comfort. The proposed controller is lightweight and can be implemented on low-cost embedded devices. Experimental results for representative weather conditions shown the superiority of proposed solution against state-of-the-art relevant algorithms as it achieved comparable performance but with significant lower computational and storage complexities, which in turn enables its execution onto low-cost embedded devices.

Although simulation-based MPC methods can produce a solution, they require a time-consuming procedure and rely on the quality and the accuracy of the building model. Thus, this type of methods is suitable for controlling components of HVAC systems that have been modeled at their design time. The problem of energy prediction becomes far more challenging in case we consider a scenario without any prior knowledge about consumers (i.e. energy consumption profiles, occupants behavior) and buildings' modeling, for example controlling existing systems.

Some approaches try to handle the buildings' energy consumption as a time-series data consisting of linear and nonlinear components based on heuristics algorithms, statistical analysis, and machine learning. However, the efficiency of these algorithms is firmly tied to their increased computational complexity, as well as to excessive training periods in order to improve model's accuracy.

In contrast to relevant implementations, the proposed solution requires only a limited amount of historical data. We build individual models for estimating thermal comfort and energy consumption based on linear regression. The selection of linear regression was based solely on our objective for deriving a low-complexity solution, which can be implemented onto an embedded low-cost device. These models feed a knapsack algorithm that is responsible for delivering the optimal solutions, depending on the problem objectives as

set by the residents.

Efficient data manipulation can enable the design of low-complexity embedded CPS orchestrators. Sliding window approaches are a good choice for data based on weather or building's residents behaviour.

To address the challenge of the increased complexitiy, our framework incorporates a combination of sliding windows to determine the "useful" data that will be considered for energy prediction. The functionality of these sliding windows is presented in Section 6.4.1. More thoroughly, instead of analyzing raw historical data acquired from sensors, the *coarse-grain* sliding window considers only the data that refer to the last days. A further improvement is achieved with the *fine-grain window*, which selects a subset of this data that refers to a specific time period per day within the days selected by the coarse-grain window.

This data manipulation can enable the use of much simpler models (e.g. the linear regression that was used in our case). Such an enhancement of linear regression's output is feasible, since both weather and occupants' energy consumption profiles, usually exhibit similarities for a given time period within consecutive days. Note that this time frame affects only the subset of historical data that is stored to the coarse-grain window. The fact that the coarse- and fine-grain windows approach is an enabler for our solution, is highlighted in Figure 67 and Figure 69. According to these results it is clear that its efficiency dominates the overall framework's performance.

Both the number of previous days, as well as the time period per day can be customized depending on the requirement for model's accuracy and the availability of hardware resources. Part of the framework's superiority is due to the effective selection of data subset used for the analysis purposes. Although one might expect that additional data improves the model's accuracy, this is not the case because this improvement degrades with non-correlated data. Moreover, since the amount of data increases linearly with the execution time, data storage, manipulation and processing are becoming challenging aspects, especially at the embedded domain.

We have to state that the coarse-grain window's size defines the storage requirements for the proposed framework. In this sense, Figure 78 plots the number of execution cycles without and with the combination of selected coarse-/fine-grain windows. This analysis highlights the superiority of introduced sliding windows, which is (as mentioned above) a prerequisite for performing models retraining, and hence minimizing the system's output error, onto an embedded device. More specifically, in contrast to the typical solution, the employed data manipulation technique leads to an almost con-

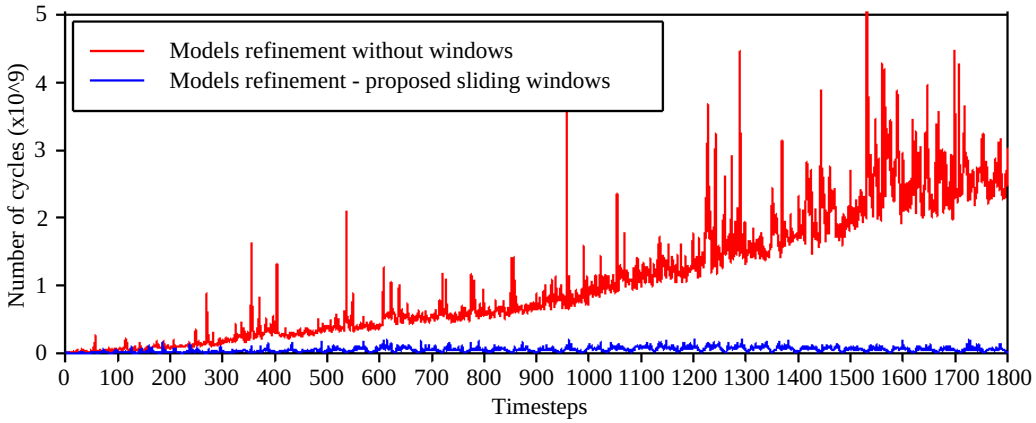


Figure 78: Number of execution cycles for energy and thermal comfort models retraining without and with the proposed coarse and fine-grain sliding windows.

stant demand of execution cycles ignoring the experiment's duration. This mainly occurs because the windows stores only selective data, whereas the conventional way stores (on-chip) raw history files.

7.2 Future work

There is a large number of potential extensions of the presented work, fact that also highlights its importance and applicability. The proposed framework aims to guide developers in designing energy-aware IoT applications. Of course the energy consumption optimizations that such a tool can offer to developers are really many. In this section we present some ideas for future work.

Increase proposed models accuracy by either enlarging the datasets and using more complex models or change the analysis and the features type.

As observed in Section 5.3 the larger errors of the energy gains by acceleration estimation models where due to the fact that very few benchmarks, included in the dataset, had similar behavior. As a result, we might conclude that the models are not trained to provide accurate predictions for such cases and although the first results can be considered acceptable, an enlargement of the dataset with more real-world applications and more representative synthetic applications is expected to also improve the estimations.

An improvement of the way that static estimator analyses the code and more precisely the type of the features it uses, can contribute to improv-

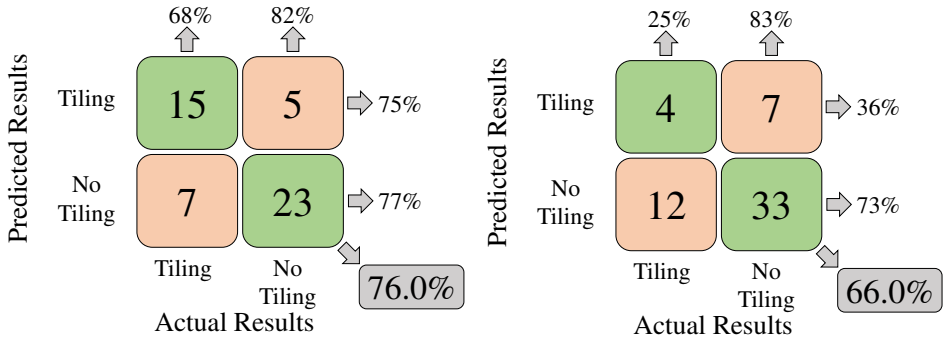
ing the quality of results. We should mention that our main purpose is to provide an extensible tool that makes the prediction of energy and energy gains in heterogeneous embedded devices prior development feasible and not to increase the accuracy of models as much as possible. The suggested future directions presented here are not easy to implement as they require, first of all, to solve the problem of creating a large and reliable data-set containing CPU (source code) applications, together with the corresponding accelerated GPU version of the specific CPU code. The data-points must be executable on the targeted devices (thus synthetic code solutions found in the literature usually are not applicable) and in a second time we must change all the analysis tools and estimation models, performing calibration and cross-validation. One solution towards this direction is to retrieve features based on the compiler's Intermediate Representation. Approaches that estimate the number of cycles based on LLVM-IR and deep neural networks are promising in capturing the behavior of modern processors [71]. In addition, more complex methods [214] are recently proposed offering a (derived from IR) graph-based representation of programs, capable of capturing the semantics of the application's statements that can be used by source code analysis tools and estimators.

Support additional hardware architectures.

The ability of the proposed solution to support not only standard CPU-GPU architectures, but also specific domain accelerators, as well as data flow architectures or even CPU-FPGA SoCs can be easily tested. However, this requires a very difficult process, which is to create a sufficient and accurate dataset. As mentioned in Section 5.3.1 the main difficulty is based on the need to include CPU (source code) applications, along with the corresponding CPU-FPGA accelerated version of the specific CPU code. Obviously, we could find a relatively small number of available benchmarks that provide both the CPU and the corresponding accelerated code. HLS (High Level Synthesis) can be very useful in creating a synthetic dataset because of its similarities to the C language. However, we can not expect very accurate results as the quality of the FPGA accelerator code is expected to affect the quality of the dataset and, subsequently, the accuracy of the predictions. Some first steps to support energy-aware placement and HW/SW partitioning just by analyzing the CPU code version can be found in the literature [215].

Support additional source-to-source optimizations or make more fine-grain suggestions.

As mentioned above, the number of source-to-source optimizations that



(a) Synthetic testset (b) Rodinia/Polybench

Figure 79: First try to make a model that suggests Loop tiling by predicting the energy gain

the proposed framework can support is not limited.

Our motivation for designing the proposed methods, was the lack of tools to assist developers in estimating the energy on multiple devices without needing to get access to them and in addressing the heterogeneity challenges of modern embedded systems. The proposed framework is designed to be extensible in the sense that it can be used in a variety of different architectures but also that the user can add new tools. The GUI and back-end include reports and guidelines, as well as useful scripts and tools to incorporate them into our framework.

For example, the proposed framework suggests testing data-flow optimizations when cache misses in nested loops occur. Developers can either use tools that implement the proposed optimizations automatically (e.g. Pluto, Orio) or perform them manually. In the context of the present work, we used extensively these two tools with very promising results in terms of the reduction of programming effort. These tools alleviate the need of manual code refactorings and can help to easily investigate if loop transformation has a positive impact on the performance and energy consumption. However, this process is still slow and needs the final targeted device to test the actual results of loop transformation. It would be interesting if the tool also suggested this type of optimizations more accurately by predicting the potential energy gains.

Towards this direction, we tried to estimate if Loop tilling is expected to improve energy or not. To achieve this we used a similar procedure to the method for predicting energy gains by acceleration 5.3. More specifically, after studying the correlation between profiling features and energy gain after tilling, we built estimation models. Figure 79 presents these preliminary

results. However, we might conclude that such a solution leads to unsatisfactory, inaccurate results. Although in the synthetic dataset the results might be acceptable (76% accuracy), in the real-life applications (Rodinia/Polybench benchmark suite) the model suggests tiling only for 4 out of the total 16 cases that offer actual energy savings.

Improve, extend and fully-integrate new optimization tools that use the proposed cross-device estimation methods, such as the Energy-aware function placement on Edge devices.

In Section 5.4 we introduced a placement decision-making solution that employs the methods proposed in Chapters 4 and 5 to propose efficient function placement on Edge devices. This study gives the directions for the development of another type of energy optimization tools that can be easily integrated in the overall framework proposed in this dissertation. The first results are very promising giving a proof-of-concept for this solution. However, as mentioned in Section 5.4, some limitations still exist. We claim that focusing on their mitigation is expected to provide a map for future research directions.

The placement suggestion method should take into account the data exchanges between the function calls. Modeling communication costs is a challenging task that is expected to have a large impact on the quality of the results produced in real IoT applications. Furthermore, making a more fine-grain modeling of the interference between functions in the same node is expected to give more accurate energy and time predictions. Moreover, the introduced method should also be able of partially adapting its initial placement suggestion, by considering scenarios where other applications are running in parallel on the same Edge devices. Finally, increasing the experiments by adding comparisons with other function placement strategies found in the literature is expected to increase the added value of the presented research study.

Generalize the targeted CPS control problem by taking into account more components (Batteries, Electric Vehicles) as well as a more complex energy market (grid-to-grid transactions).

Nowadays, energy market starts to make traditional energy consumers active prosumers, meaning that they can both dissipate and generate energy in modern smart-grid environments [216]. As a result, a dynamic pricing scheme is adopted, depending on building-to-microgrid, as well as microgrid-to-grid transactions. Demand and price forecasts are becoming crucial components for estimating energy cost and for scheduling loads. Taking also into account the intensively increased energy entities in the micro-grid, such as the use

of bateries to store energy produced by the renewable sources, even the use of Electrical Vehicles charging stations, the orchestration of such a CPS is expected to be even more complex [217].

The solution presented in the context of this thesis is generic enough and an extension towards covering these aspects can be tested in the future. Of course, such a problem remains difficult to be approached due to its complexity and the absence of a testbed (simulation framework or grid) that incorporates all these components.

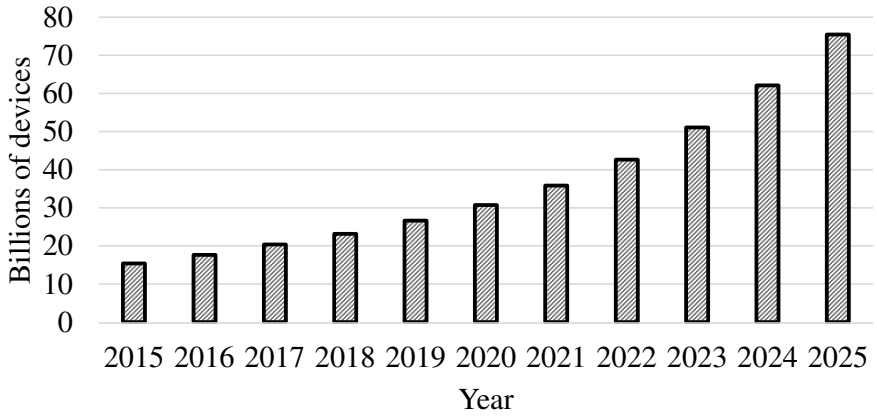
ΠΑΡΑΡΤΗΜΑ Α

Εκτεταμένη Περίληψη

Εισαγωγή

Η ενεργειακή αποδοτικότητα των υπολογιστικών πόρων που εκφράζεται συχνά με τους όρους Green και Sustainable computing κερδίζει όλο και μεγαλύτερη προσοχή τα τελευταία χρόνια. Καθώς ο αριθμός των υπολογιστικών συσκευών σε εφαρμογές Διαδικτύου των Αντικειμένων (IoT) συνεχίζει να αυξάνεται (Σχήμα 80 - Πηγή: `https://ihsmarkit.com`), η μείωση της ενέργειας των εφαρμογών αποτελεί πλέον μια σημαντική απαίτηση που επιβάλλει νέες προκλήσεις στους προγραμματιστές. Οι υπάρχουσες λύσεις ποικίλλουν σημαντικά, ανάλογα με το επίπεδο στο οποίο εξετάζεται η ενεργειακή αποδοτικότητα. Από τη μία πλευρά, πιο κοντά στο λογισμικό, υπάρχουν ερευνητικές εργασίες που προτείνουν βέλτιστες πρακτικές και δίνουν κατευθυντήριες γραμμές βασιζόμενες σε εμπειρικές μελέτες [37, 42, 43, 47]. Από την άλλη πλευρά, οι ερευνητές στον τομέα των ενσωματωμένων συστημάτων μειώνουν την ενέργεια είτε με βελτιστοποιήσεις στο ίδιο το υλικό, είτε μετασχηματίζοντας τον κώδικα της εφαρμογής, χρησιμοποιώντας εμπειρικές τεχνικές, π.χ. για την βελτιστοποίηση της διαχείρισης μνήμης [14, 21, 51, 53, 62]. Καθώς οι εφαρμογές εξελίσσονται, δημιουργείται η ανάγκη αντιμετώπισης της αυξημένης ενεργειακής κατανάλωσης στο επίπεδο του πηγαίου κώδικα της εφαρμογής από τους ίδιους τους προγραμματιστές. Ως εκ τούτου, η δημιουργία εργαλείων λογισμικού ικανών να παρέχουν εκτιμήσεις κατανάλωσης ενέργειας και να προτείνουν βελτιστοποιήσεις, είναι πλέον εξαιρετικά σημαντική, προκειμένου να παρέχεται βοήθεια στους προγραμματιστές σε όλες τις φάσεις ανάπτυξης IoT εφαρμογών.

Στόχος της παρούσας διατριβής είναι η σχεδίαση εργαλείων ανάλυσης εφαρμογών που στοχεύουν στην ενεργειακή αποδοτικότητα. Οι προτεινόμενες λύσεις, οι οποίες συνδυάζονται και με λεπτομέρειες υλοποίησης, παρέχουν εκτιμήσεις της αναμενόμενης κατανάλωσης ενέργειας των εφαρμογών πριν αυτές εκτελεστούν στις συσκευές. Τα προτεινόμενα εργαλεία παρέχουν προτάσεις βελτιστοποιήσεων στον χρήστη με ιδιαίτερη έμφαση στην εκτίμηση των πιθανών ενεργειακών κερδών από την επιτάχυνση της εφαρμογής σε GPU, ενώ εξετάζεται η επέκταση τους στην δημιουργία ενός συστήματος αποφάσεων ενεργειακά αποδοτικών τοποθετήσεων των επιμέρους συναρτήσεων των εφαρμογών στις διαθέσιμες συσκευές του δικτύου. Η μεθοδολογία που παρουσιάζεται στην διατριβή, έχει πολλά καινοτόμα χαρακτηριστικά, όπως τον συνδυασμό στατικής και δυναμικής ανάλυσης προκειμένου να αξιοποιούνται τα πλεονεκτήματα και των δύο. Επιπλέον, παρουσιάζεται μια ειδική μελέτη της επίδρασης των προτεινόμε-



Σχήμα 80: Εκτιμώμενος αριθμός συνδεδεμένων IoT συσκευών

νων βελτιστοποιήσεων στην ανάπτυξη λογισμικού, όπως π.χ. η εκτίμηση της προσπάθειας που πρέπει να καταβληθεί για να εφαρμοστούν.

Η πρόσφατη αύξηση της ζήτησης για εφαρμογές IoT, όπως για παράδειγμα ο έλεγχος της ενέργειας και του κλιματισμού στα κτίρια [26–28], κινητοποίησε τη μελέτη μιας ειδικής περίπτωσης. Τα συστήματα αυτά παρουσιάζουν αυξημένη πολυπλοκότητα και η λειτουργία τους βασίζεται λιγότερο στην ανθρώπινη λήψη αποφάσεων και περισσότερο στην υπολογιστική νοημοσύνη [23, 24]. Σε αυτή τη διατριβή εισάγονται νέες λύσεις των οποίων τα πειραματικά αποτελέσματα δείχνουν συγκρίσιμες επιδόσεις με παρόμοιους ελεγκτές τελευταίας τεχνολογίας, χωρίς όμως την ανάγκη προηγούμενης λεπτομερούς μοντελοποίησης των κτηρίων και απαιτώντας πολύ χαμηλότερους υπολογιστικούς πόρους.

Οι συνεισφορές της διατριβής μπορούν να συνοψιστούν στα εξής:

- Προτείνεται μια γενική μεθοδολογία για την δημιουργία εργαλείων ανάλυσης πηγαίου κώδικα τα οποία θα χρησιμοποιούνται από τους προγραμματιστές με στόχο την εκτίμηση της ενέργειας των εφαρμογών τους χωρίς την ανάγκη εκτέλεσης τους στις τελικές συσκευές.
  - Η εκτίμηση ενέργειας, οι μέθοδοι και τα εργαλεία βελτιστοποίησης έχουν σχεδιαστεί χρησιμοποιώντας τόσο τεχνικές δυναμικής όσο και στατικής ανάλυσης κώδικα, πετυχαίνοντας ένα συμβιβασμό μεταξύ ακρίβειας και χρηστικότητας. Οι προτεινόμενες μέθοδοι χρησιμοποιούν συνθετικά προγράμματα ως σύνολα δεδομένων, δημοφιλή εργαλεία ανάλυσης και στατιστικές τεχνικές πετυχαίνοντας υψηλή ακρίβεια πρόβλεψης ($R2 = 0.96$). Μελετώντας τη συσχέτιση μεταξύ των διαφόρων μετρήσεων (χαρακτηριστικών) και της κατανάλωσης ενέργειας, καθώς και συγκρίνοντας μεταξύ μεθόδων πρόβλεψης, η

διατριβή συμβάλει στην κατασκευή πρακτικών και εύκολων στην ανάπτυξη εργαλείων.

- Οι νέες τεχνικές και τα εργαλεία επεκτείνουν την υπάρχουσα βιβλιογραφία, δίνοντας τις κατευθύνσεις για ενεργειακή βελτιστοποίηση από το επίπεδο του λογισμικού. Πιο συγκεκριμένα, οι μέθοδοι αφορούν εκτίμηση δυνητικών ενεργειακών κερδών μέσω μετασχηματισμών για εκτέλεση τμημάτων του πηγαίου κώδικα σε έναν επιταχυντή GPU ενός ετερογενούς συστήματος.
  - Η προτεινόμενη προσέγγιση συνδυάζει μεθόδους στατικής και δυναμικής ανάλυσης και εκμεταλλεύεται τα πλεονεκτήματα και των δύο σε ένα ενιαίο εργαλείο. Ενώ η στατική ανάλυση βασίζεται στην ανάλυση του πηγαίου κώδικα, η δυναμική ανάλυση επεκτείνει τις υπάρχουσες προσεγγίσεις που παρέχουν προβλέψεις χρόνου, προς την εκτίμηση και των πιθανών ενεργειακών κερδών. Οι εφαρμογές κατηγοριοποιούνται με ακρίβεια μεγαλύτερη του 75%.
  - Μελετάται ο αντίκτυπος των βελτιστοποιήσεων που σχετίζονται με την ενέργεια στην ποιότητα σχεδιασμού λογισμικού, κάνοντας επίσης ένα πρώτο βήμα προς τη σχεδίαση ενός εργαλείου που εκτιμά την προσπάθεια που χρειάζεται να καταβληθεί για τον προγραμματισμό της βελτιστοποιημένης έκδοσης. Τα αποτελέσματα έδειξαν ακρίβεια πρόβλεψης της προγραμματιστικής προσπάθειας (εκφρασμένης μέσω των μετρικών του Halstead) της τάξεως του 85%.
  - Εξετάστηκε η χρήση των προτεινόμενων μεθοδολογιών και μοντέλων στην δημιουργία ενός άλλου είδους εργαλείου ανάλυσης εφαρμογών που στοχεύει σε ενεργειακά αποδοτικές επιλογές τοποθέτησης των συναρτήσεων των εφαρμογών στους διαθέσιμους υπολογιστικούς πόρους (συσκευές) του δικτύου. Τα πρώτα αποτελέσματα είναι ενθαρρυντικά καθώς επιτυγχάνεται 33.6% μέση ενεργειακή εξοικονόμηση για χρήση τεσσάρων συσκευών για την τοποθέτηση 25 συναρτήσεων ενώ για τρεις συσκευές η εξοικονόμηση που πετυχαίνουμε είναι περίπου στο 8.5% εν συγκρίσει με την τοποθέτηση που κάνει ο Κυβερνήτης.
  - Σχεδιάστηκε ένα πλαίσιο ανάλυσης εφαρμογών, με στόχο την ενσωμάτωση διαφορετικών ειδών εργαλείων που συμβάλλουν στον εντοπισμό ζητημάτων κατανάλωσης ενέργειας και στην πρόταση σχετικών βελτιστοποιήσεων. Μια σειρά εργαλείων υλοποιήθηκαν για να στηρίξουν την προτεινόμενη μεθοδολογία περιέχοντας τόσο τις λεπτομέρειες υλοποίησης για την δημιουργία υπηρεσιών όσο και γραφικό περιβάλλον.

- Προτείνονται νέες μέθοδοι λήψης αποφάσεων για βέλτιστη απόδοση και ταχεία δημιουργία εφαρμογών λήψης αποφάσεων για Cyber-Physical Systems, όπως τα συστήματα διαχείρισης ενέργειας και κλιματισμού σε περιβάλλον έξυπνων κτηρίων. Η προτεινόμενη μέθοδος εμφανίζει αξιοσημείωτα χαμηλότερη πολυπλοκότητα χωρίς να θυσιάζει την ποιότητα των παραγόμενων αποτελεσμάτων.

  – Παρουσιάζονται προσαρμοσμένες λύσεις που ενσωματώνουν γραμμική παλινδρόμηση και αλγόριθμους Σακιδίου, ενώ προτείνονται νέα, γρήγορα και ακριβή μοντέλα για την εκτίμηση της θερμικής άνεσης και της κατανάλωσης ενέργειας. Τα μοντέλα αυτά έχουν πολύ μικρή πολυπλοκότητα σε σύγκριση με τις άλλες υλοποιήσεις, χωρίς καμία υποβάθμιση της ποιότητας. Συγκεκριμένα η προτεινόμενη λύση πετυχαίνει μόλις 3% χειρότερα αποτελέσματα από την βέλτιστη αλλά σε 8 τάξεις μεγέθους λιγότερους κύκλους μηχανής και χωρίς προηγούμενη μοντελοποίηση ή γνώση. Το πρόβλημα έχει μοντελοποιηθεί με κατάλληλο τρόπο, ούτως ώστε να υποστηρίζονται πολλαπλές λειτουργίες, όπως η εξισορρόπηση της κατανάλωσης ενέργειας με την ικανοποίηση των κατοίκων, η ελαχιστοποίηση της κατανάλωσης ενέργειας διατηρώντας παράλληλα ένα ικανοποιητικό επίπεδο θερμικής άνεσης και η μεγιστοποίηση της ικανοποίησης των κατοίκων χωρίς υπέρβαση του διαθέσιμου ενεργειακού προϋπολογισμού.

Μέθοδοι εκτίμησης της ενέργειας

Ορισμός του προβλήματος και σχετική εργασία

Είναι σημαντικό αρχικά να ορίσουμε το πρόβλημα που λύνει η προτεινόμενη μεθοδολογία. Το εργαλείο που παράγεται με χρήση της μεθοδολογίας μας, θα τρέχει στον υπολογιστή στον οποίο εργάζεται ο προγραμματιστής, ως μέρος του περιβάλλοντος ανάπτυξης εφαρμογών που χρησιμοποιεί. Με αυτόν τον τρόπο, οι προγραμματιστές μπορούν να έχουν ανά πάσα στιγμή εικόνα του πόση ενέργεια πρόκειται να καταναλώσουν τμήματα της εφαρμογής τους σε διάφορες συσκευές χωρίς την ανάγκη εκτέλεσης του κώδικα σε αυτές.

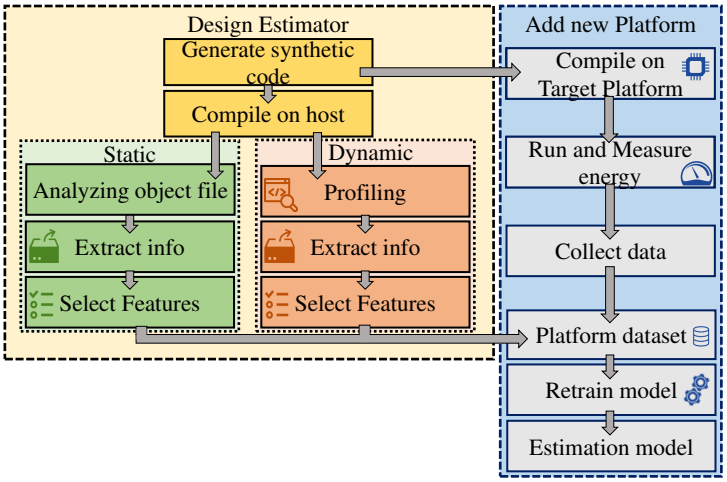
Αυτό είναι πολύ χρήσιμο, καθώς οι προγραμματιστές δεν έχουν πάντα στην διάθεσή τους όλες τις εναλλακτικές συσκευές. Επίσης η μέτρηση της ενέργειας πάνω στο πραγματικό υλικό μπορεί να χρειάζεται ειδικό εξοπλισμό (π.χ. αισθητήρες) και γνώσεις. Ακόμα, αν επιλεγόταν η πραγματική μέτρηση, τότε η παρεμβολή μιας τέτοιας διαδικασίας (σε τακτά διαστήματα) κατά την διάρκεια σχεδίασης των εφαρμογών θα αύξανε πολύ τον χρόνο ολοκλήρωσης του προϊόντος. Τέλος, μπορεί να μην είναι πάντα εφικτό η εφαρμογή να τρέχει στις συσκευές του δικτύου για πειραματισμό.

Όπως είναι φυσικό υπάρχουν σχετικές εργασίες που αποσκοπούν στο να προσφέρουν λύσεις στο πρόβλημα που περιγράψαμε. Κάποιες προσεγγίσεις στοχεύουν μόνο σε μικροελεγκτές και σε συγκεκριμένα σύνολα εντολών προσφέροντας πολύ μεγάλη ακρίβεια [48]. Άλλες εργασίες, οι οποίες μάλιστα επηρέασαν αρκετά την δική μας προσέγγιση, ειδικά στο κομμάτι της δυναμικής ανάλυσης, χρησιμοποιούν τεχνικές μηχανικής μάθησης πάνω σε δεδομένα που έχουν εξάγει από την προς ανάλυση εφαρμογή μέσω τεχνικών ανάλυσης υπολογιστικών απαιτήσεων (profiling και dynamic instrumentation) [58, 59]. Τα εργαλεία αυτά έχουν μεγάλη ακρίβεια αλλά χρειάζονται πολύ χρόνο για να ολοκληρώσουν την ανάλυση τους και μερικές φορές απαιτούν κάποια προετοιμασία του προς ανάλυση κώδικα από τους προγραμματιστές. Αυτές οι παρατηρήσεις μας έδωσαν κίνητρα να εξετάσουμε επίσης την δημιουργία εργαλείων που βασίζονται σε στατική ανάλυση. Το εργαλείο Mira [77], παρόλο που δεν εκτιμά την ενέργεια αλλά τον αριθμό πράξεων κινητής υποδιαστολής, μας επηρέασε πολύ στην σχεδίαση της λύσης μας μέσω στατικής ανάλυσης. Συγκεκριμένα, η έμφαση που δίνει στους βρόχους (loops), η αναμονή πρόσθετων πληροφοριών όπως ο αριθμός των επαναλήψεων στους βρόχους από τον προγραμματιστή κ.α. είναι σχεδιαστικές επιλογές που γίνονται και από μας. Άλλα εργαλεία που υπάρχουν στην βιβλιογραφία επεκτείνουν μεθόδους που στοχεύουν στην εκτίμηση του χειρότερου δυνατού χρόνου (worst case execution time – WCT) [72, 73, 135]. Συνήθη προβλήματα τους είναι ότι είναι εξαιρετικά αργά, όχι πολύ ακριβή και έχουν περιορισμένη εφαρμογή σε αρχιτεκτονικές και συσκευές. Τέλος πολύ σημαντικά είναι τα εργαλεία εκτίμησης του throughput μιας εφαρμογής όπως το LLVM-mca, το Intel Architecture Code Analyser (IACA) ή το Ithemal [71]. Το LLVM-mca μάλιστα ενσωματώθηκε και στην προτεινόμενη στατική μέθοδο.

Προτεινόμενη μέθοδος

Η προσέγγισή μας παρουσιάζεται συνοπτικά στο Σχήμα 81. Αρχικά παράγονται συνθετικοί κώδικες για να στηθεί το σύνολο δεδομένων. Οι κώδικες αναλύονται με τα προτεινόμενα εργαλεία στατικής και δυναμικής ανάλυσης ώστε να παραχθούν τα χαρακτηριστικά που θα δίνονται σαν είσοδοι στα μοντέλα εκτίμησης της ενέργειας. Προκειμένου να προσθέτουμε νέες συσκευές, οι οποίες στο εξής θα είναι διαθέσιμες για εκτιμήσεις ενέργειας, χρειάζεται μια φορά να τρέξουν οι συνθετικοί κώδικες και να μετρηθεί η ενέργεια. Τα δεδομένα αυτά ενέργειας μαζί με τα χαρακτηριστικά που παράχθηκαν προηγουμένως συνθέτουν το σύνολο δεδομένων το οποίο χρησιμοποιείται για την εκπαίδευση των μοντέλων τα οποία στην συνέχεια είναι έτοιμα για χρήση.

Οι συνθετικοί κώδικες παράγονται μέσω Python scripts και αποτελούνται από τυχαίους βρόχους σε γλώσσα C, οι οποίοι εμπεριέχουν τυχαίο αριθμό επα-



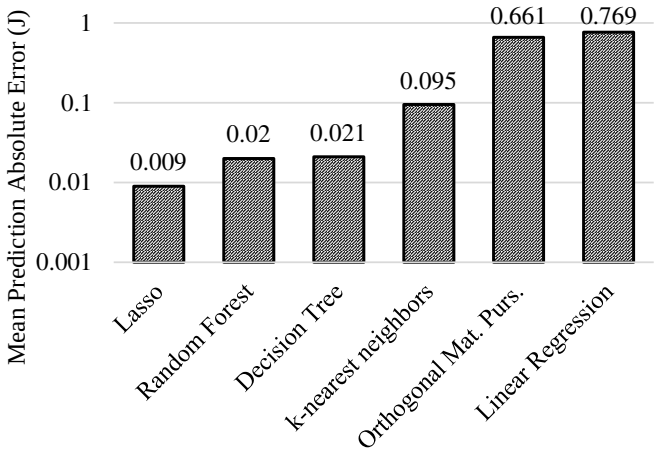
Σχήμα 81: Προτεινόμενη μέθοδος σχεδίασης εργαλείων εκτίμησης της ενέργειας

Πίνακας 18: Παράδειγμα τυχαία παραγόμενου κώδικα

```
for(i=0; i<18; i++){
    for(j=0; j<250; j++){
        for(k=0; k<167; k++){
            A0[i][j][k] = A1[i][j][k]*A1[i][j][k]/A1[i][j][k];
            B0[i][j][k] = B1[i][j]-B1[i][j]/B1[i][j]*B1[i][j];
        }
    }
}
```

ναλήψεων και εκτελούν τυχαίες πράξεις (πρόσθεσης, αφαίρεσης, πολλαπλασιασμού και διαίρεσης) μεταξύ τυχαίου αριθμού και μεγέθους πινάκων ακεραίων και αριθμών κινητής υποδιαστολής. Ένα παράδειγμα τυχαία παραγόμενου κώδικα φαίνεται στον Πίνακα 18.

Δυναμική ανάλυση Η δυναμική ανάλυση βασίζεται κυρίως σε μεθόδους ανάλυσης υπολογιστικών απαιτήσεων (profiling και dynamic instrumentation). Χρησιμοποιεί τα δημοφιλή εργαλεία Valgrind και Pin. Με αυτόν τον τρόπο παράγουμε περισσότερες από 100 μετρικές από την εκτέλεση της εφαρμογής. Προκείμενου να επιλέξουμε τις πιο σχετικές με την ενέργεια, χρησιμοποιούμε μεθόδους συσχέτισης και τέλος συγκρίνουμε και επιλέγουμε τα καταλληλότερα μοντέλα εκτίμησης. Υπενθυμίζουμε ότι οι μετρικές/χαρακτηριστικά παράγονται στον υπολογιστή που δουλεύει ο προγραμματιστής, ενώ οι εκτιμήσεις μας



Σχήμα 82: Εναλλακτικά μοντέλα εκτίμησης της ενέργειας βασισμένα σε δυναμική ανάλυση

αφορούν την ενέργεια που η εφαρμογή θα καταναλώσει στην τελική συσκευή.

Αρχικά το σύνολο δεδομένων μας περιέχει όλα τα χαρακτηριστικά, όλων των συνθετικών προγραμμάτων και την ενέργεια που αυτά καταναλώνουν στον ARM Cortex-A57 της κάρτας Nvidia TX1. Δίνοντας σαν είσοδο μέρος των συνθετικών δεδομένων σε διάφορα εναλλακτικά μοντέλα και προβλέποντας τις ενέργειες των υπολοίπων (τεχνική cross-validation), τα αποτελέσματα φαίνονται στο Σχήμα 82. Παρατηρούμε ότι το μοντέλο Lasso υπερτερεί των υπολοίπων. Ένας λόγος που συμβαίνει αυτό είναι η δυνατότητα που έχει να επιλέγει χαρακτηριστικά, γεγονός πολύ σημαντικό ειδικά σε περιπτώσεις σαν την δική μας, όπου έχουμε πολλά χαρακτηριστικά που εκφράζουν παρόμοια πληροφορία ή έχουν κάποια αναλογική σχέση αναμεταξύ τους.

Το ίδιο το μοντέλο Lasso μας παρέχει έναν τρόπο επιλογής χαρακτηριστικών δίνοντας βάρη κοντά στο μηδέν σε όσα χαρακτηριστικά δεν χρησιμοποιεί. Τα πιο σημαντικά χαρακτηριστικά σύμφωνα με το Lasso, φαίνονται στον Πίνακα 19. Παρατηρούμε ότι οι λειτουργίες μνήμης, η συμπεριφορά της μνήμης cache, η πρόβλεψη των branches, ο αριθμός και ο τύπος των αριθμητικών πράξεων κ.α. έχουν κυρίαρχη συνεισφορά στην ενέργεια που καταναλώνει η τελική συσκευή. Παρόλο που τα αποτελέσματα θα μπορούσαν να χαρακτηριστούν αναμενόμενα, η μελέτη αυτή είναι πολύ σημαντική γιατί μας επιτρέπει να μειώσουμε τον αριθμό των χρησιμοποιούμενων χαρακτηριστικών κατά πολύ, μειώνοντας έτσι και το πολύ μεγάλο (όπως θα δούμε παρακάτω) κόστος της δυναμικής ανάλυσης υπολογιστικών απαιτήσεων.

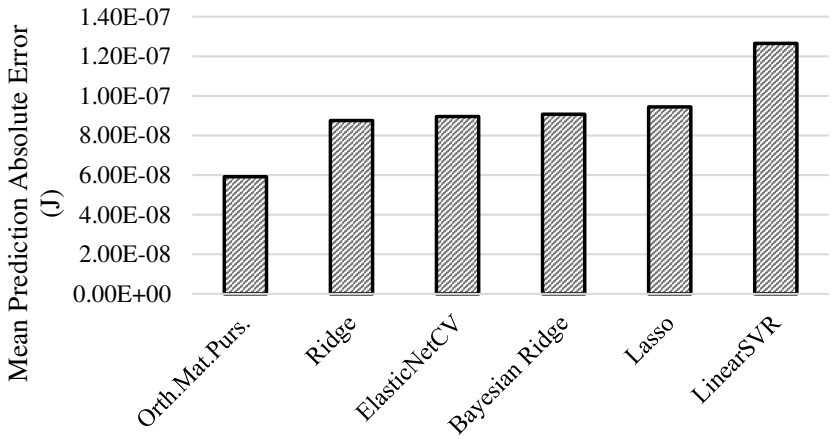
Στατική ανάλυση Σε αυτήν την παράγραφο παρουσιάζουμε μια εναλλακτική προσέγγιση προκειμένου να αποφεύγουμε την αργή εκτέλεση των εφαρμο-

Πίνακας 19: Πιο σημαντικά χαρακτηριστικά δυναμικής ανάλυσης

Χαρακτηριστικό	Βάρος Lasso (σημαντικότητα)
Παραλληλοποίηση σε επίπεδο εντολών	2.481
Εγγραφές δεδομένων στη μνήμη	0.688
Σελίδες μνήμης που προσπελαύνονται	0.438
Μέγεθος μνήμης σωρού	0.379
Αριθμητικές πράξεις	0.324
Βήμα προσβάσεων στην μνήμη (stride)	0.240
Υπο συνθήκη διακλαδώσεις	0.187
Αποτυχίες πρόβλεψης διακλαδώσεων	0.185
Ρυθμός αστοχίας στο τελευταίο επίπεδο μνήμης	0.119
Χρήση στοίβας	0.099
Ρυθμός αστοχίας στο πρώτο επίπεδο μνήμης	0.053
Αναγνώσεις μνήμης	0.014

γών που επιφέρει η δυναμική ανάλυση. Σε αυτή την προσέγγιση η ανάλυση μας βασίζεται στον πηγαίο κώδικα της εφαρμογής και μόνο. Ο στόχος μας είναι και πάλι μια λύση που δίνει προσεγγίσεις ενέργειας για πολλές και διαφορετικές συσκευές και αρχιτεκτονικές. Αυτό μας οδηγεί στην επιλογή απλών και γενικών χαρακτηριστικών για να τροφοδοτήσουμε τα μοντέλα εκτίμησης της ενέργειας, τα οποία θα εξάγονται από τον κώδικα και θα έχουν εφαρμογή και νόημα σε πληθώρα αρχιτεκτονικών βασισμένων σε CPU. Τα χαρακτηριστικά αυτά λαμβάνονται από την ανάλυση της assembly την οποία παράγει ο μεταγλωττιστής καθώς και με χρήση του εργαλείου LLVM-mca. Τα χαρακτηριστικά που τροφοδοτούν τα μοντέλα μας είναι τα εξής:

- Ο αριθμός των εντολών (INS)
- Η εκτίμηση του throughput (μέσω LLVM-mca)
- Ο αριθμός των εντολών ανάγνωσης μνήμης (LOAD)
- Ο αριθμός των εντολών εγγραφής μνήμης (STORE)
- Ο αριθμός των εντολών αριθμητικών λειτουργιών (OP)
- Υποκατηγορία αριθμητικών εντολών 1 (πρόσθεση, αφαίρεση, πολλαπλασιασμός, ολίσθηση)
- Υποκατηγορία αριθμητικών εντολών 2 (μετατροπή δείκτη, εντολές σε πίνακες, διαίρεση)
- Η σειρά των εντολών OP, LOAD και STORE



Σχήμα 83: Εναλλακτικά μοντέλα εκτίμησης της ενέργειας βασισμένα σε στατική ανάλυση

Η κάθε κατηγορία επιλέχθηκε με βάση τον χρόνο εκτέλεσης και την ενέργεια των εντολών αυτών σε συσκευές βασισμένες σε ARM. Συγκεκριμένα κάθε χαρακτηριστικό περιλαμβάνει εντολές με κοντινό κόστος σε ενέργεια/χρόνο. Η σειρά των εντολών μοντελοποιήθηκε με μια τεχνική sliding window. Ένα παράθυρο ολισθαίνει μεταξύ των εντολών του προγράμματος και κάθε συνδυασμός εντολών LOAD, STORE και OP δημιουργεί ένα επιπρόσθετο χαρακτηριστικό οδηγώντας σε $3^3 = 27$ νέα χαρακτηριστικά.

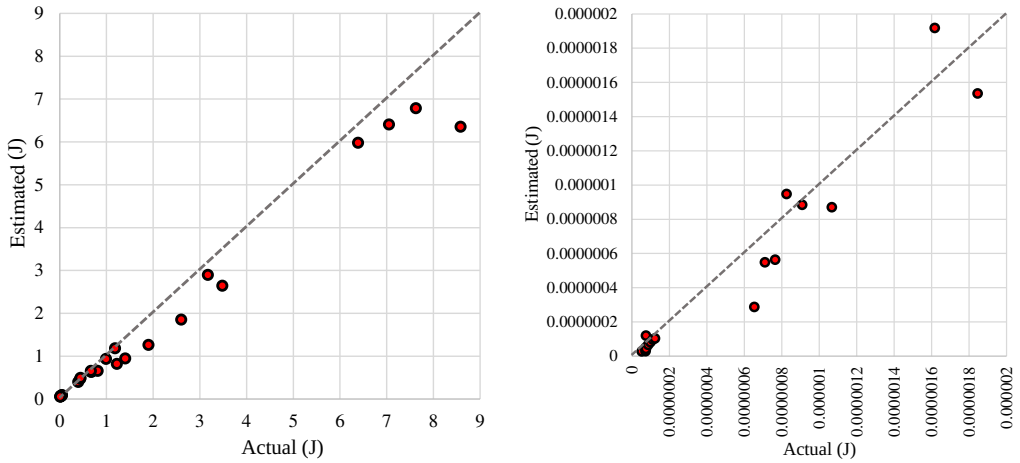
Κατόπιν σύγκρισης μεταξύ διαφόρων μοντέλων εκτίμησης, η ακρίβεια των έξι καλύτερων συνοψίζεται στο Σχήμα 83. Σύμφωνα με την μελέτη αυτή, το μοντέλο Orthogonal Matching Pursuit επιλέχθηκε.

Πειραματικά αποτελέσματα

Για τα πειράματα που θα παρουσιάσουμε παρακάτω χρησιμοποιήσαμε αρχικά την συσκευή Nvidia Jetson TX1 και συγκεκριμένα επειδή τρέχουμε CPU εφαρμογές, τον ARM Cortex A-57 που περιέχει το τσιπ. Προκειμένου να ελέγξουμε την επεκτασιμότητα της προτεινόμενης μεθόδου, θα εκτιμήσουμε την ενέργεια που καταναλώνει ο Intel Xeon Gold 6138 ενός server συστήματος.

Οι εφαρμογές που χρησιμοποιούμε για την αξιολόγηση των προτεινόμενων μεθόδων περιλαμβάνονται στις σουίτες εφαρμογών Polybench και Rodinia [126], που περιέχουν μεγάλη γκάμα υπολογισμών από στατιστική, μηχανική μάθηση, επεξεργασία εικόνας, βιοπληροφορική κ.α. και χρησιμοποιούνται ευρέως στην βιβλιογραφία.

Τα πρώτα αποτελέσματα φαίνονται στο Σχήμα 84. Χρησιμοποιώντας σαν σύνολο δεδομένων εκπαίδευσης μόνο τους συνθετικούς κώδικες, προβλέπουμε την ενέργεια των πιο σημαντικών βρόχων των εφαρμογών του Polybench και

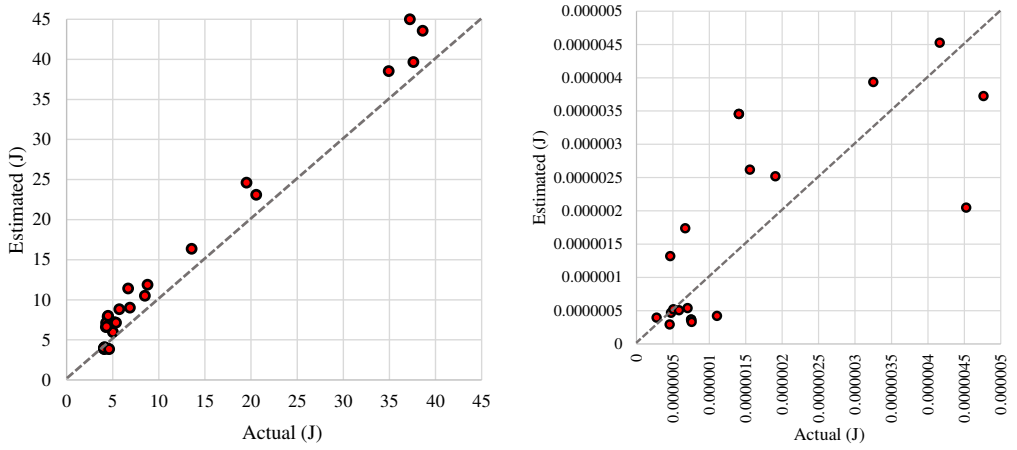


(α′) Δυναμική ανάλυση (β′) Στατική ανάλυση

Σχήμα 84: Σύγκριση εκτιμόμενης και πραγματικής ενέργειας στον ARM-Cortex A-57

του Rodinia χρησιμοποιώντας δυναμική (Σχήμα 84α′) και στατική (Σχήμα 84β′) ανάλυση. Όπως είναι φυσικό η δυναμική ανάλυση οδηγεί σε πιο ακριβή αποτελέσματα ($R^2 = 0.96$) ενώ η στατική ανάλυση πετυχαίνει ένα R^2 της τάξης του 0.92. Η δυναμική ανάλυση χρειάζεται (όπως αναφέρθηκε και προηγουμένως) αρκετό χρόνο για να ολοκληρωθεί, ενώ απαιτεί την εκτέλεση των εφαρμογών και αυτές οι ιδιότητες της αποτελούν μειονεκτήματα κατά την ενσωμάτωσή της σε ένα περιβάλλον σχεδίασης εφαρμογών ενώ η στατική ανάλυση παράγει στιγμιαία αποτελέσματα αναλύοντας μόνο τον πηγαίο κώδικα. Η στατική ανάλυση από την άλλη, κάνει εκτίμηση μόνο της ενέργειας εκτέλεσης ενός μπλοκ κώδικα χωρίς να λαμβάνει υπόψη δυναμική πληροφορία όπως για παράδειγμα τον αριθμό των επαναλήψεων στους βρόχους. Αυτό φαίνεται και στο Σχήμα 84β′ όπου βλέπουμε ότι οι ενέργειες αντιστοιχούν σε μια μόνο επανάληψη των βρόχων. Η πληροφορία αυτή πρέπει να δοθεί από τον προγραμματιστή κάτι που δεν είναι πάντα εύκολο, ειδικά στην περίπτωση που έχουμε εμφωλευμένους βρόχους, κλήσεις συναρτήσεων μέσα σε βρόχους ή υπό συνθήκη κλάδους.

Στα επόμενα πειράματα που παρουσιάζονται στο Σχήμα 85, χρησιμοποιούμε την μέθοδο προσθήκης νέων συσκευών προκειμένου να υποστηρίξουμε τον Intel Xeon Gold 6138. Εδώ η στατική ανάλυση χάνει πολύ σε ακρίβεια, κάτι που οφείλεται στο γεγονός ότι οι ενέργειες της μιας επανάληψης είναι πολύ μικρές και ο θόρυβος που περιλαμβάνει στις μετρήσεις ένας μεγάλος server με 20 πυρήνες και 40 νήματα δυσκολεύει ακόμα περισσότερο τα πράγματα. Χαρακτηριστικό είναι ότι στους πιο μεγάλους βρόχους, όπως για παράδειγμα αυτούς που περιλαμβάνει η σουίτα Rodinia, το R^2 αγγίζει το 0.85. Αντιθέτως, για την



(α′) Δυναμική ανάλυση (β′) Στατική ανάλυση

Σχήμα 85: Σύγκριση εκτιμόμενης και πραγματικής ενέργειας στον Server (Xeon Gold 6138)

χρήση δυναμικής ανάλυσης τα αποτελέσματα είναι πολύ καλά ($R^2 = 0.9$), λαμβάνοντας μάλιστα υπόψιν ότι δεν έγιναν αλλαγές στο μοντέλο εκτίμησης παρά μόνο στις τιμές ενέργειας του συνόλου δεδομένων που χρησιμοποιείται για την εκπαίδευση.

Μέθοδοι βελτιστοποίησης της ενέργειας

Στόχος της ενότητας αυτής είναι η ανάπτυξη μεθόδων ανάλυσης του λογισμικού IoT εφαρμογών με σκοπό την ενεργειακή βελτιστοποίηση. Οι μέθοδοι αυτές θα είναι μέρος του περιβάλλοντος ανάπτυξης εφαρμογών των προγραμματιστών και θα προτείνουν τρόπους μετασχηματισμού των εφαρμογών προκειμένου να μειώνεται η ενέργεια που θα καταναλώνουν. Οι ενεργειακές βελτιστοποιήσεις οι οποίες υποστηρίζονται από το προτεινόμενο εργαλείο εφαρμόζονται αποκλειστικά στο λογισμικό (δεν περιλαμβάνουν αλλαγές στο υλικό) και είναι οι εξής:

- Βελτίωση της ροής δεδομένων στην ιεραρχία μνήμης μέσω βελτιστοποιήσεων εντός βρόχου.
- Επιλογή πλατφόρμας/συσκευής στην οποία θα εκτελείται η εφαρμογή.
- Εκτίμηση των πιθανών ενεργειακών κερδών μέσω επιτάχυνσης.
- Ενεργειακά αποδοτική τοποθέτηση συναρτήσεων της εφαρμογής στους διαθέσιμους υπολογιστικούς πόρους.

Η πρώτη κατηγορία βασίζεται στην παρακολούθηση των αστοχιών της μνήμης cache (cache misses), η οποία είναι μέρος των μετρικών σχετικών με την ενέργεια που προβάλλονται στον χρήστη (όπως δείξαμε στον Πίνακα 19). Στην περίπτωση που οι αστοχίες σε κάποιο βρόχο περνούν το 3%, προτείνεται στον χρήστη η δοκιμή κάποιων μετασχηματισμών βελτιστοποίησης βρόχου. Παρ' όλο που στο πλαίσιο της διατριβής δοκιμάσαμε εργαλεία αυτοματοποίησης όπως το Pluto και το Orio και διαπιστώσαμε ότι επιταχύνουν πολύ τις δοκιμές και την όλη διαδικασία, οπότε και τα προτείνουμε στον χρήστη, απαιτούν και πάλι την εκτέλεση της εφαρμογής στην τελική συσκευή. Μια περαιτέρω αυτοματοποίηση και πρόβλεψη των εκτιμώμενων κερδών θα επιτάχυνε ακόμα περισσότερο την διαδικασία καθώς ο μετασχηματισμός θα γινόταν μόνο στο περιβάλλον ανάπτυξης της εφαρμογής. Αυτό όμως αποτελεί μελλοντική μελέτη.

Η δεύτερη κατηγορία βασίζεται στις μεθόδους που είδαμε στις προηγούμενες παραγράφους. Με βάση τις προβλέψεις για διάφορες συσκευές, ο χρήστης μπορεί να επιλέγει την συσκευή που πληροί τις προϋποθέσεις που θέτει η εφαρμογή από πλευράς ενέργειας.

Η τρίτη κατηγορία είναι αυτή που μας απασχόλησε περισσότερο στην παρούσα διατριβή. Η ουσία αυτού του είδους βελτιστοποίησης συνοψίζεται στα εξής: Στις προηγούμενες παραγράφους προτείναμε μια μέθοδο δημιουργίας εργαλείων για την εκτίμηση της ενέργειας που θα καταναλώσει μια εφαρμογή αν εκτελεστεί σε μια άλλη συσκευή που περιλαμβάνεται στο IoT δίκτυο, καθαρά μέσω ανάλυσης της εφαρμογής στο περιβάλλον ανάπτυξης που χρησιμοποιεί ο προγραμματιστής. Σήμερα, οι συσκευές IoT περιλαμβάνουν πολλές φορές ετερογενή τσιπ που εμπεριέχουν επιταχυντές μαζί με τον κεντρικό επεξεργαστή, όπως για παράδειγμα GPUs ή FPGAs. Αυτό δημιουργεί νέες δυνατότητες βελτίωσης τόσο του χρόνου εκτέλεσης της εφαρμογής όσο και της ενέργειας που καταναλώνει. Αν οι προγραμματιστές γνωρίζουν εκ των προτέρων αν η χρήση των δυνατοτήτων αυτών πρόκειται να επιφέρει σημαντικά κέρδη, μπορούν να επενδύουν στον ανασχηματισμό του κώδικα της εφαρμογής προς αυτήν την κατεύθυνση. Μέσω της προτεινόμενης λύσης, θα δίνεται μια εκτίμηση της τελικής ενέργειας που μπορεί να καταναλώσει η εφαρμογή του αφού μετασχηματιστεί κατάλληλα.

Η τέταρτη κατηγορία αφορά μια προέκταση των μεθόδων που αναπτύξαμε στην παρούσα διατριβή. Η χρήση των προτεινόμενων εργαλείων εκτίμησης της ενέργειας στην δημιουργία ενός επιπλέον εργαλείου ενεργειακά βέλτιστης τοποθέτησης των συναρτήσεων της εφαρμογής στους διαθέσιμους υπολογιστικούς πόρους με τρόπο ώστε να πληρούνται επίσης οι απαιτήσεις της εφαρμογής σε χρόνο εξετάζεται και παρουσιάζονται τα πρώτα αποτελέσματα.

Εκτίμηση των πιθανών κερδών μέσω επιτάχυνσης

Στην μελέτη μας και πάλι συμπεριλαμβάνουμε τόσο την αναζήτηση μεθόδων δυναμικής όσο και στατικής ανάλυσης. Συγκεκριμένα, για λόγους που θα παρουσιαστούν στην συνέχεια, προτείνουμε ένα συνδυασμό των δύο μεθόδων. Αρχικά, πρέπει να αναφέρουμε ότι όλοι οι μετασχηματισμοί προτείνονται για τα τμήματα κώδικα που παρουσιάζουν ενδιαφέρον και αποτελούν υποψήφια σημεία για να εφαρμόσουμε ενεργειακές βελτιστοποιήσεις. Πιο συγκεκριμένα, θεωρούμε σημεία ενδιαφέροντος τα κομμάτια κώδικα για τα οποία οι μετρικές του Πίνακα 19 που ορίζει ο χρήστης του εργαλείου μας είναι μεγάλες. Πρώτα με στατική ανάλυση εκτιμάται αν θα έχουμε μεγάλα ενεργειακά κέρδη μέσω επιτάχυνσης. Αυτήν την κλάση ταξινόμησης των τμημάτων κώδικα, θα την ονομάζουμε «Μεγάλα κέρδη». Στην συνέχεια, στην περίπτωση που το κομμάτι κώδικα δεν ανήκει στην κλάση «Μεγάλα κέρδη», με δυναμική ανάλυση εκτιμούμε την περίπτωση να μην έχουμε κανένα κέρδος από επιτάχυνση (Κλάση «Όχι κέρδη»). Οι περιπτώσεις που δεν ανήκουν ούτε στην κλάση «Μεγάλα κέρδη» ούτε στην «Όχι κέρδη» θα αποτελούν την κλάση «Μεσαία κέρδη». Για αυτήν θα εφαρμόζεται μέθοδος παλινδρόμησης βασισμένη σε δυναμική ανάλυση προκειμένου να δίνονται πιο λεπτομερείς προσεγγίσεις των αναμενόμενων κερδών από επιτάχυνση.

Εξέχουσας σημασίας στην μελέτη αυτή, είναι η δημιουργία ενός καλού και χαρακτηριστικού συνόλου δεδομένων που θα υποστηρίξει τα προτεινόμενα μοντέλα. Η δυσκολία ενός τέτοιου εγχειρήματος έγκειται στο γεγονός ότι το σύνολο πρέπει να περιέχει κώδικες που εκτελούνται σε CPU μαζί με τις ακριβώς αντίστοιχες εκδοχές τους για GPU. Για τους σκοπούς της παρούσας έρευνας, χρησιμοποιήσαμε ένα σύνολο δεδομένων που αποτελείται κατά 50% από συνθετικούς κώδικες (όπως αυτούς που περιγράφηκαν παραπάνω – Πίνακας 18) και κατά 50% από κώδικες που περιλαμβάνονται στις σουίτες Rodinia και Polybench.

Η στατική ανάλυση χρησιμοποιεί μεθόδους αναγνώρισης κειμένου από την βιβλιογραφία [148]. Αφού γίνει μια προ-επεξεργασία του κώδικα όπου αφαιρούνται σχόλια κλπ., ένας μηχανισμός αντιστοιχεί κάθε γράμμα καθώς και γνωστές λέξεις που αντιστοιχούν σε εντολές π.χ. (while, if, for) σε ξεχωριστούς αριθμούς, οπότε και το τμήμα κώδικα γίνεται ένα διάνυσμα αριθμών. Αυτά τροφοδοτούν ένα συνελικτικό νευρωνικό δίκτυο ειδικό για την κατηγοριοποίηση ακολουθιών [149].

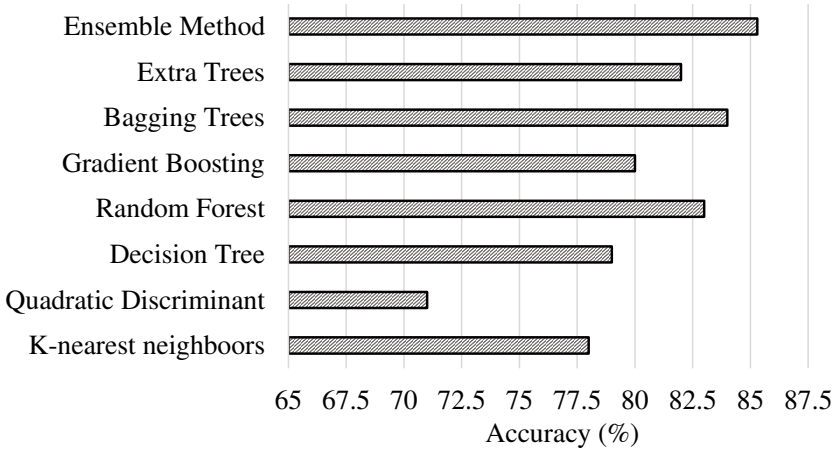
Η δυναμική ανάλυση βασίζεται στην πληθώρα μετρικών που παίρνουμε με τα εργαλεία ανάλυσης υπολογιστικών απαιτήσεων που περιγράψαμε παραπάνω. Έπειτα γίνεται μια μελέτη συσχέτισης για να δούμε ποιες από αυτές τις μετρικές είναι κατάλληλες για την εκτίμηση της ενέργειας στην GPU με χρήση του μοντέλου stepAIC. Από αυτήν την μελέτη προκύπτει ότι οι πιο σχετικές

μετρικές, που θα αποτελέσουν και τα χαρακτηριστικά για την τροφοδοσία των μοντέλων δυναμικής ανάλυσης είναι οι εξής:

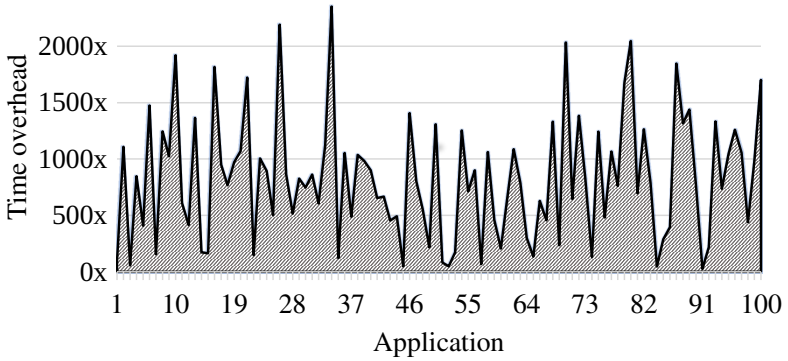
- Ο βαθμός παραλληλοποίησης εντολών
- Ο αριθμός των εντολών
- Ο αριθμός των προσβάσεων στην μνήμη για πρώτη φορά
- Ο αριθμός των πράξεων κινητής υποδιαστολής
- Ο αριθμός των ακεραίων πράξεων
- Ο αριθμός των εντολών ελέγχου
- Ο αριθμός των εντολών μνήμης
- Ο αριθμός των προσβάσεων στην μνήμη με μηδενικό βήμα (zero stride)
- Οι αλλαγές κατεύθυνσης στις διακλαδώσεις (branch divergence)
- Ο αριθμός των πράξεων διαίρεσης
- Ο αριθμός των προσβάσεων σε μπλοκ μνήμης στην ίδια σελίδα

Για την επιλογή του μοντέλου προσέγγισης των ενεργειακών κερδών κάναμε και πάλι συγκρίσεις. Η ακρίβεια των καλύτερων μοντέλων φαίνεται στο Σχήμα 86. Ως ενεργειακό κέρδος ορίζουμε το πηλίκο της ενέργειας που καταναλώνει η εκδοχή του κώδικα που χρησιμοποιεί μόνο CPU προς την ενέργεια του κώδικα που κάνει χρήση και της GPU, ή διαφορετικά το πόσες φορές μειώνεται η ενέργεια. Ως τελικό μοντέλο χρησιμοποιούμε έναν συνδυασμό των τριών καλύτερων με μια μέθοδο ψηφοφορίας (ensemble voting).

Πριν προχωρήσουμε στα πειραματικά αποτελέσματα αξίζει να δείξουμε τι μας οδήγησε στον συνδυασμό στατικής και δυναμικής ανάλυσης κατά τον τρόπο που παρουσιάσαμε. Όπως προαναφέραμε η δυναμική ανάλυση επιφέρει μεγάλο κόστος χρόνου για να ολοκληρωθεί. Στο Σχήμα 87 φαίνεται η επαύξηση που δημιουργεί στην εκτέλεση 100 εφαρμογών από το σύνολο δεδομένων μας, η οποία σε ορισμένες περιπτώσεις είναι πάρα πολύ υψηλή. Από την άλλη, η στατική ανάλυση εκτελείται στιγμιαία αλλά έχει χαμηλή ακρίβεια συγκριτικά με την δυναμική. Αυτό φαίνεται στο Σχήμα 88. Η στατική ανάλυση έχει ικανοποιητική ακρίβεια μόνο για κατηγοριοποίηση μεταξύ δύο κλάσεων ενεργειακών κερδών, ενώ η δυναμική κρατάει την ακρίβεια στο 75% ακόμα και για τέσσερις κλάσεις. Άρα αυτό μας οδήγησε στην χρήση της στατικής ανάλυσης για μια πρώτη κατηγοριοποίηση μεταξύ δύο κλάσεων και έπειτα την χρήση δυναμικής ανάλυσης για πιο λεπτομερή αποτελέσματα.



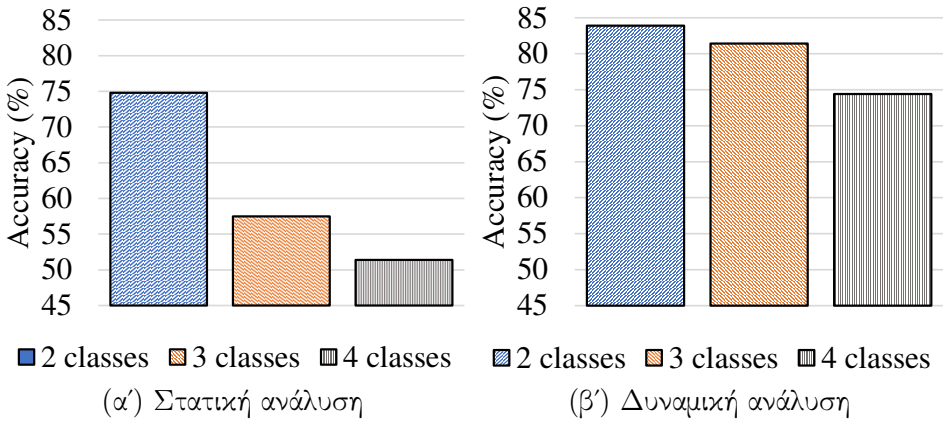
Σχήμα 86: Σύγκριση εναλλακτικών μοντέλων για την εκτίμηση ενεργειακών κερδών μέσω δυναμικής ανάλυσης



Σχήμα 87: Επαύξηση του χρόνου εκτέλεσης λόγω παρεμβολής της δυναμικής ανάλυσης

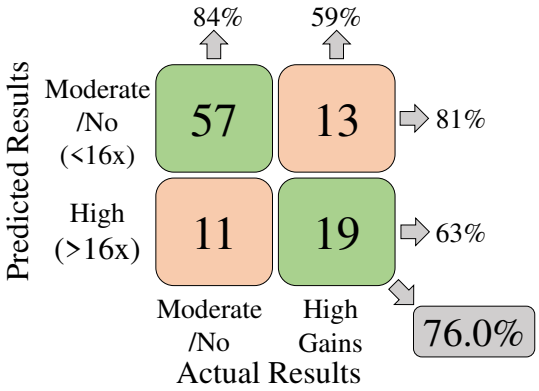
Πειραματικά αποτελέσματα Για τα πειράματα που παρουσιάζονται στην συνέχεια, αρχικά και πάλι χρησιμοποιήθηκε η συσκευή Nvidia TX1, αυτή την φορά και η CPU και η GPU από το τσιπ και οι πραγματικές ενεργειακές καταναλώσεις μετρήθηκαν με τον ενσωματωμένο αισθητήρα ενέργειας. Όσον αφορά τις εφαρμογές χρησιμοποιήσαμε και πάλι βρόχους από τις σουίτες Polybench και Rodinia. Δόθηκε προσοχή ώστε ποτέ να μην συμπεριλαμβάνονται στο σύνολο δεδομένων εκπαίδευσης και στους κώδικες που χρησιμοποιούμε για την αξιολόγηση των μοντέλων, βρόχοι από την ίδια εφαρμογή!

Τα αποτελέσματα φαίνονται στο Σχήματα 89 και 90. Το συγκεκριμένο είδος διαγράμματος λέγεται confusion plot και οι γραμμές του αντιστοιχούν στις προβλέψεις μας ενώ οι στήλες του στις πραγματικές κλάσεις. Στην διαγώνιο που φαίνεται με πράσινο χρώμα έχουμε τις περιπτώσεις που οι προβλέψεις μας



(α′) Στατική ανάλυση (β′) Δυναμική ανάλυση

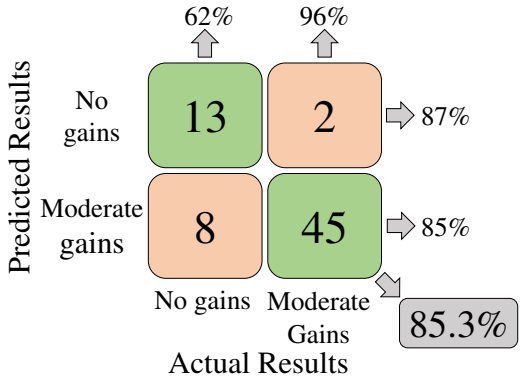
Σχήμα 88: Ακρίβεια κατηγοριοποίσης ενεργειακών κερδών μέσω επιτάχυνσης στην Nvidia Jetson TX1



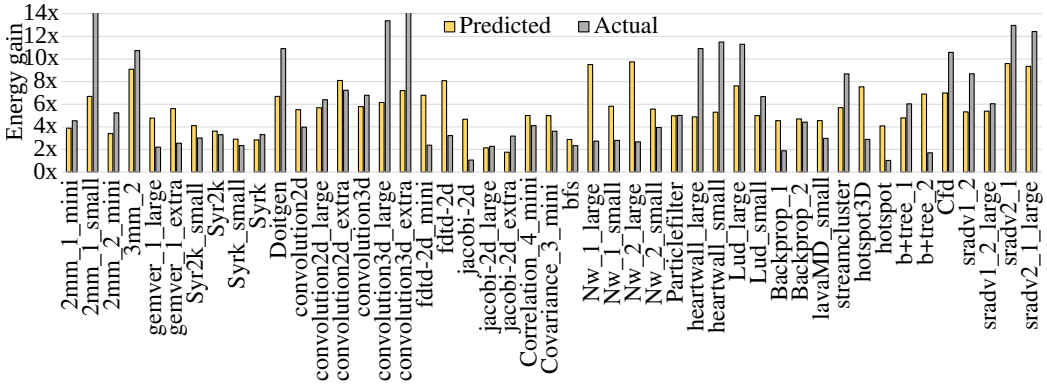
Σχήμα 89: Σύγκριση προβλέψεων με στατική ανάλυση και πραγματικών κερδών (κλάση 'Μεγάλα κέρδη')

και η πραγματική τοποθέτηση των υπό εξέταση κωδίκων συμπίπτουν ενώ στα πορτοκαλί τετράγωνα έχουμε τις περιπτώσεις που οι προβλέψεις μας είναι λάθος. Σύμφωνα με το πείραμα αυτό, η στατική ανάλυση πετυχαίνει μια ακρίβεια της τάξης του 76% στην εύρεση των περιπτώσεων με «Μεγάλα κέρδη». Η κλάση «Μεγάλα κέρδη» για τους υπό εξέταση βρόχους ορίζεται σε κέρδη ενέργειας πάνω από 16 φορές. Η δυναμική ανάλυση πετυχαίνει ακρίβεια κατηγοριοποίησης άνω του 85%.

Έπειτα, για τις περιπτώσεις που ανήκουν στην κατηγορία «Μεσαία κέρδη», κάνουμε μια πιο λεπτομερή εκτίμηση μέσω παλινδρόμησης, χρησιμοποιώντας το μοντέλο Τυχαία Δάση (Random forest). Τα αποτελέσματα φαίνονται στο Σχήμα 91. Το μέσω σφάλμα αντιστοιχεί σε ενεργειακά κέρδη 2.6×. Τα μεγαλύτερα σφάλματα παρατηρούνται σε περιπτώσεις που δεν υπήρχαν στο υπόλοιπο



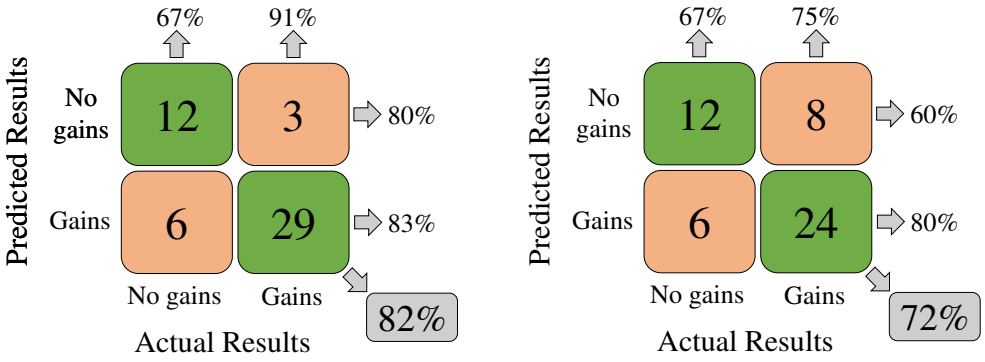
Σχήμα 90: Σύγκριση προβλέψεων με στατική ανάλυση και πραγματικών κερδών (κλάσεις 'Μεσαία κέρδη'/'Όχι κέρδη')



Σχήμα 91: Σύγκριση προβλέψεων με παλινδρόμηση βασισμένη σε δυναμική ανάλυση και πραγματικών κερδών για τις περιπτώσεις που ανήκουν στην κλάση 'Μεσαία κέρδη' στην Nvidia Jetson TX1

σύνολο δεδομένων. Για παράδειγμα, η εφαρμογή Heartwall έχει μεγάλο επίπεδο αλλαγών κατεύθυνσης στις διακλαδώσεις, γεγονός που οδηγεί το μοντέλο μας να προβλέψει μικρά κέρδη. Παρ' όλα αυτά, οι βρόχοι της εφαρμογής τελικά πετυχαίνουν πάνω από 10 φορές μείωση της ενέργειας στην GPU. Οι παρατηρήσεις αυτές μας οδηγούν στο συμπέρασμα ότι μια μελλοντική επαύξηση του συνόλου δεδομένων με περισσότερες πραγματικές εφαρμογές θα βελτιώσει ακόμα περισσότερο τα αποτελέσματα.

Η επέκταση της μεθόδου σε άλλες ενσωματωμένες συσκευές όπως στην Nvidia Jetson Nano και Nvidia Xavier NX έδωσε παρόμοια αποτελέσματα. Η προσπάθεια επέκτασης στην υποστήριξη ενός server με Intel Xeon Gold 6138 και Nvidia Tesla V100, ο οποίος μπορεί πλέον να είναι μέρος ενός IoT δικτύου, έδωσε πιο μέτρια αποτελέσματα όπως φαίνεται στο Σχήμα 92 όπου



(α′) Δυναμική ανάλυση (β′) Στατική ανάλυση

Σχήμα 92: Σύγκριση προβλέψεων και πραγματικών κερδών ενέργειας στον Server (Xeon Gold 6138 - Tesla V100)

κάναμε μια κατηγοριοποίηση των βρόχων σε «Κέρδη»/«Όχι κέρδη». Αυτό οφείλεται στο γεγονός ότι στο σύστημα αυτό έχουμε πολύ μεγάλο κόστος μεταφοράς δεδομένων από την κεντρική μνήμη στην μνήμη της GPU, κάτι που οδηγεί σε επαύξηση του χρόνου ακόμα και 22 φορές για μικρούς βρόχους, την στιγμή που στις ενσωματωμένες συσκευές είναι περίπου 3 φορές. Η προσέγγιση στατικής ανάλυσης ζημιώνεται πολύ από το γεγονός αυτό καθώς δεν έχει τρόπο να μοντελοποιεί την ποσότητα των δεδομένων που μεταφέρονται.

Σχετική Εργασία Οι πιο σημαντικές σχετικές προσεγγίσεις που βρίσκονται στην βιβλιογραφία [78–81] εστιάζουν στην επιτάχυνση και όχι στα ενεργειακά κέρδη. Επίσης στοχεύουν γενικού σκοπού συστήματα CPU-GPGPU και όχι ενσωματωμένες συσκευές. Επιπλέον χρησιμοποιούν μόνο δυναμική ανάλυση, ενώ ελάχιστες προσπάθειες έχουν βρεθεί να δοκιμάζουν στατική ανάλυση ζητώντας και πάλι όμως πολλές δυναμικές πληροφορίες από τον χρήστη [150]. Τέλος δεν εξετάζουν την επεκτασιμότητα, ενώ δεν περιλαμβάνουν μετρικές που αφορούν το κόστος σχεδίασης του νέου κώδικα όπως την προγραμματιστική προσπάθεια που θα δούμε παρακάτω.

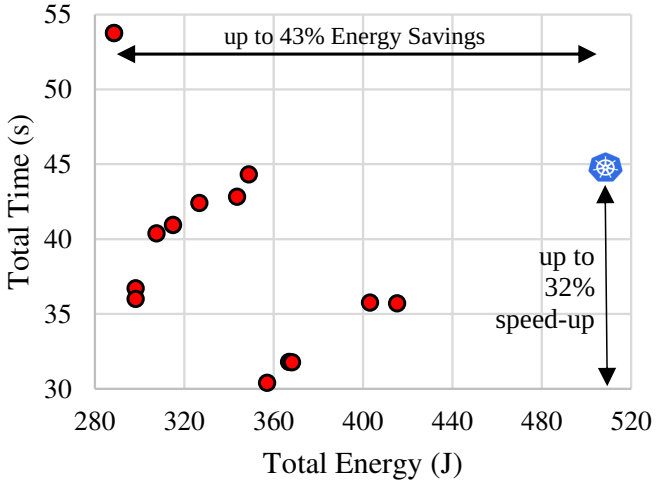
Προεκτάσεις στην ενεργειακά αποδοτική τοποθέτηση συναρτήσεων στους διαθέσιμους υπολογιστικούς πόρους

Στην παράγραφο αυτή θα δείξουμε μια περίπτωση χρήσης των προαναφερθέντων μοντέλων εκτίμησης ενέργειας σε έναν μηχανισμό λήψης αποφάσεων για την τοποθέτηση συναρτήσεων στους διαθέσιμους υπολογιστικούς πόρους του δικτύου IoT. Όπως αναφέραμε προηγουμένως, τα προτεινόμενα μοντέλα βοηθούν τον προγραμματιστή να διαλέγει την συσκευή στην οποία θα εκτελεί

τα προγράμματά του. Στην περίπτωση που μια εφαρμογή IoT αποτελείται από πολλές ανεξάρτητες συναρτήσεις και οι διαθέσιμες συσκευές είναι περισσότερες από μια, η εύρεση κατάλληλης στρατηγικής τοποθέτησης κάθε συνάρτησης προς εκτέλεση στις συσκευές, είναι ένα κλασικό πρόβλημα που έχει τραβήξει εδώ και χρόνια το ενδιαφέρον των ερευνητών [154, 155]. Το να λαμβάνεται η απόφαση αυτή ένα στάδιο πίσω, δηλαδή κατά την δημιουργία της εφαρμογής και χωρίς τις δοκιμές πάνω στα πραγματικά συστήματα, μπορεί να δώσει στον προγραμματιστή την δυνατότητα να βελτιώνει την ενέργεια της εφαρμογής του ορίζοντας μια διαφορετική τοποθέτηση των εφαρμογών εξ αρχής. Οι σύγχρονες μάλιστα τάσεις του υπολογισμού χωρίς διακομιστή (serverless computing) προσφέρουν εργαλεία που χρησιμοποιούν Docker containers, Κυβερνήτη (`https://kubernetes.io/`) κ.λ.π., και με τα οποία ο προγραμματιστής μπορεί πολύ εύκολα να ορίζει και να αλλάζει την τοποθέτηση χωρίς να έχει γνώσεις δικτύων και επικοινωνιών [156]. Πολλές ερευνητικές προσπάθειες στοχεύουν στην πρόταση λύσεων βασισμένων στα εργαλεία αυτά [158, 159].

Αρχικά χρησιμοποιούμε τα προτεινόμενα μοντέλα για να προβλέψουμε την ενέργεια όλων των συναρτήσεων της εφαρμογής, σε όλες τις συσκευές. Επεκτείναμε τα μοντέλα, επίσης, για την εκτίμηση του χρόνου αλλάζοντας τις τιμές ενέργειας στα σύνολα δεδομένων με τιμές χρόνου. Έπειτα, όλες οι πληροφορίες που παράγονται δίνονται σαν είσοδος σε μια μέθοδο βελτιστοποίησης που είναι βασισμένη στον αλγόριθμο differential evolution. Μια τιμή από 0 έως 1 (trade-off – tr) δίνεται από τον χρήστη ανάλογα με το αν θέλει να βελτιστοποιεί περισσότερο τον χρόνο εκτέλεσης ή περισσότερο την ενέργεια. Άρα η μέθοδος ελαχιστοποιεί την τιμή που δίνεται από την παρακάτω εξίσωση, όπου E, είναι η ενέργεια και T ο χρόνος που προβλέπεται για κάθε τοποθέτηση: $Minimize : tr \times E(\mathbf{p}) + (1 - tr) \times T(\mathbf{p})$

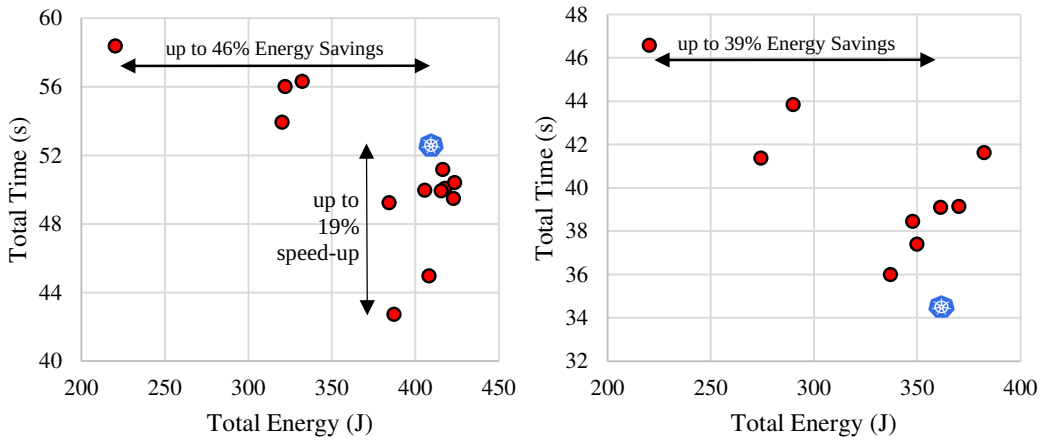
Για τα παρακάτω πειράματα χρησιμοποιήσαμε 25 συναρτήσεις από την βιβλιοθήκη scikit-learn της Python, η οποία χρησιμοποιείται ευρέως σε εφαρμογές μηχανικής μάθησης. Επιλέξαμε την Python λόγω της μεγάλης χρήσης της σε εφαρμογές υπολογισμού χωρίς διακομιστή και της μεγάλης υποστήριξής της από τα σύγχρονα εργαλεία. Στο πρώτο πείραμα χρησιμοποιήθηκαν 4 συσκευές (Nvidia TX1, Nvidia Jetson Nano, Nvidia Xavier NX, Nvidia Xavier AGX). Όπως βλέπουμε στο Σχήμα 93 για μέγιστη βελτιστοποίηση ενέργειας πετυχαίνουμε 43% λιγότερη κατανάλωση συγκριτικά με το να αφήσουμε τον Κυβερνήτη να αποφασίσει μόνος του την τοποθέτηση και κατά μέσο όρο 33.6% λιγότερη ενέργεια και 11% λιγότερο χρόνο. Οι αποφάσεις όμως της μεθόδου μας δεν χρησιμοποιούν σε κάποιες περιπτώσεις κάποια από τις συσκευές, ενώ ο Κυβερνήτης προσπαθεί να ισομοιράζει τις συναρτήσεις στους διαθέσιμους πόρους. Γι' αυτό στα επόμενα πειράματα (Σχήμα 94) αποφασίσαμε να χρησιμοποιήσουμε 3 συσκευές, πιέζοντας ακόμα περισσότερο την λύση μας. Στην πρώτη περίπτωση και πάλι βλέπουμε ότι η μέθοδος μας οδηγεί σε διάφορους συμβιβασμούς μεταξύ



Σχήμα 93: Σύγκριση των αποτελεσμάτων τοποθέτησης σε 4 συσκευές με την τοποθέτηση του Κυβερνήτη

ενέργειας και χρόνου ανάλογα με την τιμή (tr) που ορίζει ο χρήστης οδηγώντας σε μέχρι και 46% μείωση της ενέργειας και μέχρι 19% μείωση του χρόνου συγκριτικά με τον Κυβερνήτη. Στην δεύτερη περίπτωση έχουμε 3 λύσεις οι οποίες είναι χειρότερες από τον Κυβερνήτη καθώς πετυχαίνουν μεγαλύτερο χρόνο καταναλώνοντας όμως και μεγαλύτερη ενέργεια ενώ οι υπόλοιπες μειώνουν την ενέργεια σε σχέση με τον Κυβερνήτη με κόστος όμως στον χρόνο. Κατά μέσο όρο οι λύσεις μας πετυχαίνουν εξοικονόμηση ενέργειας κατά 8.5% σε σχέση με τον Κυβερνήτη.

Τα αποτελέσματα αυτά είναι αρκετά ενθαρρυντικά. Παρ' όλα αυτά έχουμε ακόμα κάποιους περιορισμούς για την πλήρη ενσωμάτωση της λύσης μας στο προτεινόμενο πλαίσιο, οι οποίοι δίνουν και τις κατευθύνσεις για μελλοντική έρευνα: Πρώτον, η λύση μας δεν λαμβάνει υπόψιν τυχόν ανταλλαγές δεδομένων μεταξύ των συναρτήσεων. Η μοντελοποίηση των επικοινωνιών αυτών είναι μια πρόκληση η οποία όμως αναμένεται να αυξήσει κατά πολύ την εφαρμογή της μεθόδου μας σε πραγματικά συστήματα IoT. Δεύτερον, θεωρούμε ότι οι συσκευές προορίζονται μόνο για την εκτέλεση της εφαρμογής μας χωρίς να λαμβάνουμε υπόψιν την περίπτωση να τρέχουν παράλληλα και άλλες εφαρμογές. Τέλος, τα πειράματά μας πρέπει να επεκταθούν μέσω συγκρίσεων με άλλα εργαλεία τοποθέτησης συναρτήσεων από την βιβλιογραφία.



(α′) TX1, Nano, Xavier AGX (β′) TX1, Nano, Xavier NX

Σχήμα 94: Σύγκριση των αποτελεσμάτων τοποθέτησης σε 3 συσκευές με την τοποθέτηση του Κυβερνήτη

Επιρροή μετασχηματισμών σε μετρικές σχεδίασης λογισμικού - Εκτίμηση προγραμματιστικής προσπάθειας

Στις προηγούμενες παραγράφους αναπτύξαμε τεχνικές οι οποίες προτείνονται στους προγραμματιστές προκειμένου να βελτιώνουν την κατανάλωση ενέργειας κάνοντας αλλαγές στον πηγαίο κώδικα. Όμως δεν αναφερθήκαμε καθόλου σε επιπτώσεις που θα έχουν οι αλλαγές αυτές κατά την σχεδίαση του κώδικα όπως για παράδειγμα στην προσπάθεια που πρέπει να καταβάλουν οι σχεδιαστές προκειμένου να τις υλοποιήσουν. Μια τέτοια επιπλέον πληροφορία εκτιμάται ότι θα παίξει ρόλο στις επιλογές των σχεδιαστών των εφαρμογών, καθώς θα μπορούν πλέον να λαμβάνουν υπόψιν και τον χρόνο που θα πρέπει να καταβληθεί ώστε να σχεδιαστεί η νέα έκδοση του λογισμικού.

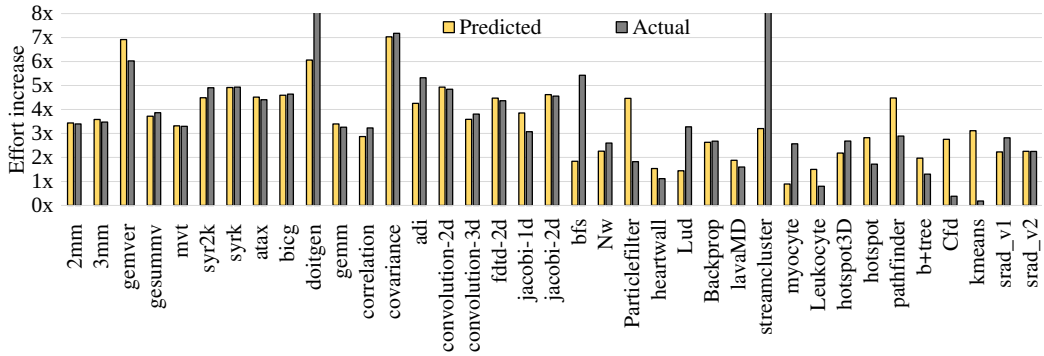
Οι μεγαλύτεροι μετασχηματισμοί χρειάζονται στην περίπτωση της επιτάχυνσης καθώς ο κώδικας πρέπει να ξαναγραφτεί για να εκτελείται πλέον στην GPU. Στις περιπτώσεις των μετασχηματισμών βελτιστοποίησης βρόχου υπάρχουν εργαλεία όπως το Pluto που μειώνουν την προσπάθεια, ενώ στις περιπτώσεις επιλογής συσκευής ή τοποθέτησης των συναρτήσεων, οι όποιες αλλαγές στον πηγαίο κώδικα θα αφορούν και πάλι την περίπτωση που επιλέγεται η αλλαγή μιας συσκευής CPU σε μια που περιλαμβάνει GPU που θέλουμε να χρησιμοποιηθεί για επιτάχυνση, ενώ οι αλλαγές της τοποθέτησης από μόνες τους δεν επιφέρουν ουσιαστικές αλλαγές στον πηγαίο κώδικα.

Για την εκτίμηση της προγραμματιστικής προσπάθειας, στην βιβλιογραφία βρίσκουμε την παλιά μετρική του αριθμού των γραμμών κώδικα (Lines of Code – LOC) [169] που όμως πλέον δέχεται κριτική για την ακρίβειά της [170]. Στην

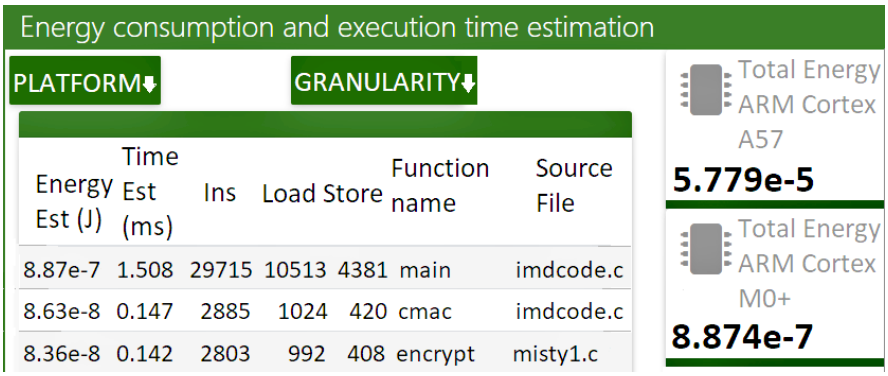
παρούσα εργασία εξετάζουμε την χρήση της μετρικής προσπάθειας του Halstead (Halstead effort) [171] [172]. Η μετρική αυτή επηρεάζεται από το πλήθος συναρτήσεων, τα ορίσματα, τους τελεστές κ.α., εκφράζοντας έτσι πιο καλά την προγραμματιστική προσπάθεια [173]. Η πρόταση μας δεν έγκειται απλά στην χρήση της μετρικής αυτής αφού γραφτεί ο νέος κώδικας. Από όσο γνωρίζουμε κατά την συγγραφή της παρούσας διατριβής, γίνεται για πρώτη φορά η προσπάθεια της πρόβλεψης της πριν την συγγραφή του νέου κώδικα. Συγκεκριμένα, θα χρησιμοποιούμε χαρακτηριστικά και μοντέλα πρόβλεψης στην CPU έκδοση του κώδικα προκειμένου να προβλέψουμε την προσπάθεια που θα χρειαστεί να καταβάλουν οι σχεδιαστές προκειμένου να γράψουν την εκδοχή που εμπεριέχει την επιτάχυνση σε GPU. Ως μετρική της προσπάθειας θεωρούμε τον αριθμό των φορών που αυξάνεται η μετρική του Halstead στην CPU-GPU έκδοση εν συγκρίσει με την CPU έκδοση του κώδικα. Οι μετρικές που τροφοδοτούν το μοντέλο πρόβλεψης (στην περίπτωσή μας ένα μοντέλο Τυχαίου Δάσους όπως προέκυψε από πειράματα και συγκρίσεις), λαμβάνονται αναλύοντας μόνο την CPU έκδοση του κώδικα και είναι οι εξής:

- Ο αριθμός των σημείων ενδιαφέροντος
- Ο αριθμός των γραμμών της εφαρμογής
- Ο αριθμός των γραμμών στα σημεία ενδιαφέροντος
- Ο αριθμός των δηλώσεων (statements) στα σημεία ενδιαφέροντος
- Ο αριθμός των διακριτών πράξεων (distinct operations)
- Οι συνολικές πράξεις
- Η πολυπλοκότητα (complexity)
- Ο όγκος (Volume)
- Το μήκος (Length)
- Η δυσκολία (Difficulty)

Τα αποτελέσματα φαίνονται στο Σχήμα 95. Η ακρίβεια πρόβλεψης αγγίζει το 85%. Οι περιπτώσεις που χάνουμε ανήκουν κυρίως στην σουίτα Rodinia όπου μάλιστα βλέπουμε 3 εφαρμογές με κάτω του 1× αύξηση του Halstead effort. Αυτό βέβαια δεν σημαίνει ότι έχουμε αρνητική προσπάθεια, απλά στις συγκεκριμένες εφαρμογές η CPU έκδοση περιείχε λειτουργικότητα (π.χ. εκτύπωση κάποιων μηνυμάτων) που δεν μεταφέρθηκε στην GPU. Προτιμούμε παρ' όλα αυτά να μην επέμβουμε στα προγράμματα.



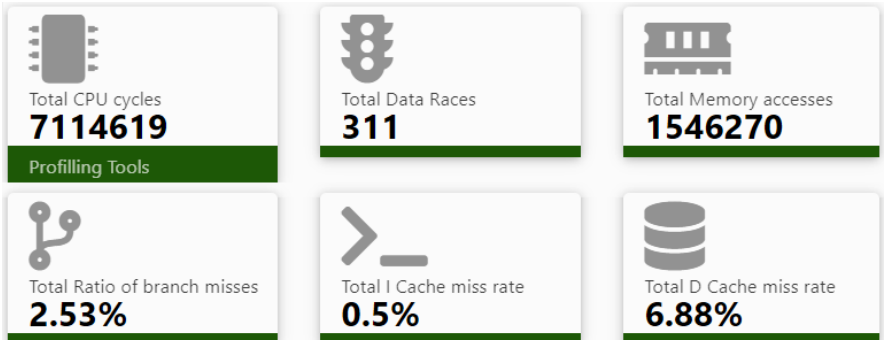
Σχήμα 95: Ακρίβεια εκτίμησης της προγραμματιστικής προσπάθειας για την επιτάχυνση των εφαρμογών από τις σουίτεςPolybench και Rodinia



Σχήμα 96: Πάνελ εκτίμησης ενέργειας

Υλοποίηση των προτεινόμενων μεθόδων

Οι μέθοδοι που παρουσιάσαμε παραπάνω ενσωματώνονται σε ένα πλήρες περιβάλλον ανάλυσης λογισμικού. Το περιβάλλον αυτό διαθέτει γραφικό μέρος επικοινωνίας με τον χρήστη, λαμβάνει τον κώδικα προς ανάλυση από Git repositories, διαθέτει βάση δεδομένων αποθήκευσης των αποτελεσμάτων και υλοποιείται ολόκληρο μέσα σε Docker containers προκειμένου να εγκαθίστανται εύκολα και με ασφάλεια. Σχεδιάστηκε στα πλαίσια του ευρωπαϊκού έργου SDK4ED και στα Σχήματα 96 και 97 παρουσιάζουμε δύο στιγμιότυπα, το πρώτο από την εκτίμηση της ενέργειας σε διάφορες συσκευές και το δεύτερο από την παρουσίαση μετρικών σχετικών με την ενέργεια στον χρήστη.



Σχήμα 97: Πάνελ παρουσίασης μετρικών σχετικών με την ενέργεια

Εφαρμογή ελέγχου θέρμανσης/ψύξης σε έξυπνα κτήρια

Τις τελευταίες δεκαετίες, οι απαιτήσεις ενέργειας στον πλανήτη έχουν αυξηθεί σημαντικά και αυτό έχει τόσο οικονομικές όσο και περιβαλλοντικές επιπτώσεις. Ως εκ τούτου, η ανάγκη για την εύρεση «έξυπνων» συστημάτων ικανών να μειώνουν την κατανάλωση ενέργειας γίνεται πλέον επιτακτική. Σύμφωνα με στατιστικές έρευνες, τα κτήρια καταναλώνουν περίπου το 40% της συνολικής κατανάλωσης ενέργειας στην Ευρωπαϊκή Ένωση. Μάλιστα, μεγάλο μέρος αυτής οφείλεται στα συστήματα θέρμανσης/ψύξης (περίπου το 45% της κατανάλωσης).

Ένας τρόπος εξοικονόμησης είναι η χρήση νέων τεχνολογιών κλιματιστικών ή θερμαντικών σωμάτων που έχουν πολύ καλύτερη ενεργειακή απόδοση. Μια δεύτερη λύση, άμεσα εφαρμόσιμη και στα υπάρχοντα κτήρια, παρέχουν οι σύγχρονες τεχνολογίες Κυβερνο-φυσικών συστημάτων (Cyber-physical) μέσω συνεχής επίβλεψης, ελέγχου και αλληλεπίδρασης με το περιβάλλον. Σε αυτό βοηθούν οι σύγχρονες δυνατότητες των υπολογιστικών συστημάτων, των αισθητήρων και του Διαδικτύου των αντικειμένων. Στον τομέα των έξυπνων κτηρίων, οι τεχνολογίες αυτές παρέχουν άμεσες λύσεις και ένα τυπικό παράδειγμα χρήσης τους είναι οι έξυπνοι θερμοστάτες. Οι πωλήσεις τους τα τελευταία χρόνια παρουσιάζουν ραγδαία αύξηση και η βελτίωση της απόδοσής τους αποτελεί αντικείμενο των σύγχρονων ερευνών.

Σε αυτή την ενότητα της διατριβής προτείνουμε ένα σύστημα διαχείρισης ενέργειας που στοχεύει την εφαρμογή IoT των έξυπνων κτηρίων. Πιο συγκεκριμένα, προτείνουμε μια μέθοδο λήψης αποφάσεων για την αυτόματη ρύθμιση των επιλογών του συστήματος θέρμανσης ή κλιματισμού μέσω έξυπνων θερμοστατών και με στόχο την ελαχιστοποίηση του κόστους. Ως «κόστος» θεωρούμε έναν συνδυασμό τόσο της κατανάλωσης ενέργειας του συστήματος κλιματισμού, όσο και της θερμικής δυσαρέσκειας των ανθρώπων στο κτήριο που υπόκεινται

στις συνθήκες που δημιουργούνται ως αποτέλεσμα των επιλογών του συστήματός μας. Οι συνεισφορές της μελέτης αυτής συνοψίζονται ως εξής.

- Προτείνεται ένας νέος αλγόριθμος λήψης αποφάσεων για τον βέλτιστο έλεγχο Κυβερνοφυσικών συστημάτων, με εξειδίκευση στα συστήματα θέρμανσης/ψύξης των έξυπνων κτηρίων. Η λύση εμφανίζει αξιοσημείωτα χαμηλότερη υπολογιστική πολυπλοκότητα χωρίς να θυσιάζει την ποιότητα των παραγόμενων αποτελεσμάτων.

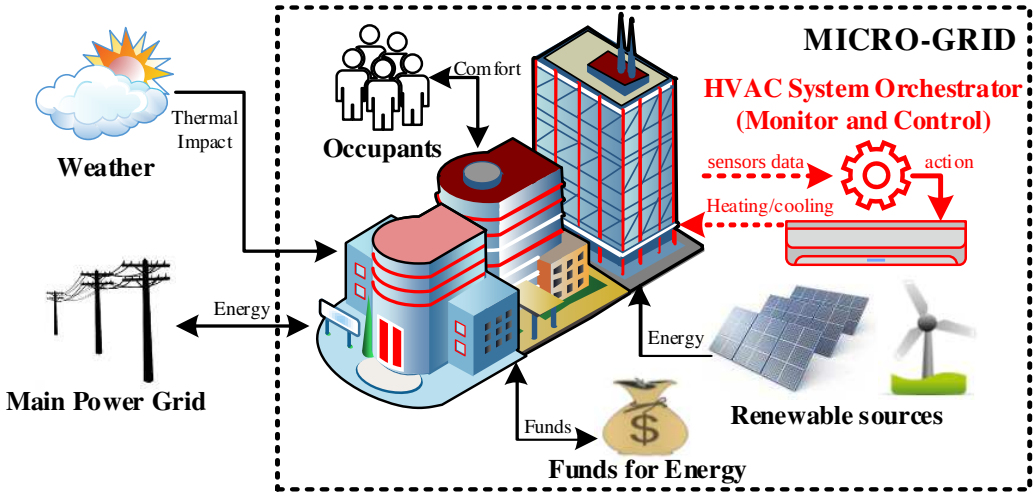
- Παρουσιάζουμε δύο γρήγορα και ακριβή μοντέλα για την εκτίμηση της θερμικής άνεσης και της ενεργειακής κατανάλωσης των κλιματιστικών συστημάτων βασισμένα σε γραμμική παλινδρόμηση.

- Η προτεινόμενη λύση υποστηρίζει βελτίωση τόσο της ενέργειας όσο και της θερμικής άνεσης προσφέροντας διάφορες δυνατότητες στους χρήστες χωρίς την ανάγκη προηγούμενου σχεδιασμού, αναλυτικής μοντελοποίησης ή γνώσης (plug&play λύση).

Ορισμός του προβλήματος

Η μελέτη μας γίνεται σε ένα περιβάλλον κτηρίων, σαν αυτό που απεικονίζεται στο Σχήμα 98. Πιο συγκεκριμένα, το κτήριο διαθέτει πηγές ενέργειας (π.χ. ηλιακή, αιολική, φυσικό αέριο), ενώ τα συστήματα ψύξης/θέρμανσης καταναλώνουν ενέργεια. Ένας αριθμός αισθητήρων συγκεντρώνει δεδομένα που σχετίζονται με τον καιρό (θερμοκρασία, υγρασία και ηλιακή ακτινοβολία), τις συνθήκες του κτηρίου (εσωτερική θερμοκρασία και υγρασία), καθώς και τη δραστηριότητα των ανθρώπων.

Το συγκεκριμένο πρόβλημα που αντιμετωπίζουμε σε αυτήν την ενότητα αφορά τον έλεγχο των συστημάτων θέρμανσης/ψύξης (δηλαδή, τις αποφάσεις των έξυπνων θερμοστατών) προκειμένου να βελτιωθεί η θερμική άνεση των κατοίκων με την ελάχιστη κατανάλωση ενέργειας. Τα κτήρια στα οποία έγιναν τα πειράματα προσομοιώθηκαν με λεπτομερή τρόπο στο πρόγραμμα προσομοίωσης EnergyPlus [182], ενώ τα δεδομένα καιρού και τιμολόγησης ενέργειας αντιστοιχούν σε δεδομένα που συλλέχθηκαν το 2010 για την Αθήνα. Στα πειράματά μας, υποθέτουμε ότι οι άνθρωποι βρίσκονται στα κτήρια μόνο κατά τις ώρες λειτουργίας τους (6.00 - 21.00) και ο αριθμός των ατόμων ανά δωμάτιο ποικίλλει κατά τη διάρκεια της ημέρας σύμφωνα με μια ψευδοτυχαία κατανομή. Όπως αναφέραμε, η ποιότητα της προτεινόμενης λύσης ποσοτικοποιείται με δύο μετρήσεις και πιο συγκεκριμένα το κόστος ενέργειας που αγοράζεται από το δίκτυο και το επίπεδο θερμικής άνεσης.



Σχήμα 98: Παρουσίαση του περιβάλλοντος έξυπνων κτηρίων στο οποίο εφαρμόζεται η λύση μας

Μοντελοποίηση κόστους - Λειτουργίες του προτεινόμενου συστήματος

Η συνάρτηση κόστους του συστήματός μας μοντελοποιήθηκε με τέτοιο τρόπο ώστε το σύστημα έξυπνων θερμοστατών να υποστηρίζει τρία εναλλακτικά σενάρια λειτουργίας:

Λειτουργία 1) Επίτευξη συμβιβασμού μεταξύ της κατανάλωσης ενέργειας και των μετρήσεων θερμικής άνεσης.

Λειτουργία 2) Ελαχιστοποίηση της κατανάλωσης ενέργειας τηρώντας παράλληλα ένα ελάχιστο όριο για την θερμική άνεση. Σύμφωνα με το πρότυπο ASHRAE [132], οποιαδήποτε τιμή θερμικής δυσαρέσκειας (PPD) στο εύρος 0% -10% είναι αποδεκτή.

Λειτουργία 3) Βελτιστοποίηση της θερμικής άνεσης χωρίς να υπερβούμε έναν προκαθορισμένο προϋπολογισμό ενέργειας για τη διάρκεια του πειράματος.

Η συνάρτηση κόστους δίνεται παρακάτω:

$Cost(t) = tr \times E(\alpha_t, s_t) + (1 - tr) \times C(\alpha_t, s_t)$, όπου t το βήμα χρόνου, $E(\alpha_t, s_t)$, η ενέργεια στον χρόνο t και $C(\alpha_t, s_t)$ η θερμική δυσαρέσκεια στον χρόνο t, για α_t αποφάσεις του έξυπνου θερμοστάτη και s_t συνθήκες περιβάλλοντος. Το tr αποτελεί μια τιμή που ορίζεται κατάλληλα από 0 έως 1 προκειμένου να δίνεται ανάλογο βάρος στην βελτιστοποίησης της ενέργειας ή της άνεσης π.χ. στην Λειτουργία 1 έχουμε $tr = 0.5$, στην Λειτουργία 2 έχουμε $tr = 1$ και στην Λειτουργία 3 $tr = 0$.

Στο κτήριο υπάρχουν αισθητήρες που μας δίνουν τιμές σε κάθε χρονική στιγμή (βήμα). Το μέγεθος του χρονικού βήματος το καθορίζει ο χρήστης.

Αυτές οι τιμές αποτελούν την είσοδο στο σύστημά μας. Σε κάθε δωμάτιο υπάρχουν αισθητήρες που μας δίνουν την θερμοκρασία, την υγρασία και την κατανάλωση ενέργειας του κλιματιστικού συστήματος, ενώ στην οροφή του κτηρίου υπάρχουν αισθητήρες που μας ενημερώνουν για την εξωτερική θερμοκρασία, την εξωτερική υγρασία και την ηλιακή ακτινοβολία.

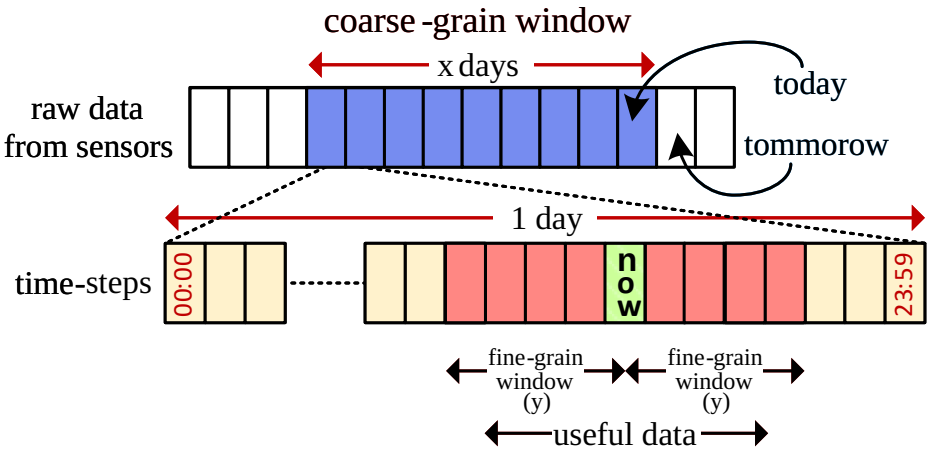
Προτεινόμενη Λύση

Αυτή η ενότητα περιγράφει τον προτεινόμενο μηχανισμό λήψης αποφάσεων. Σε αντίθεση με σχετικές μελέτες, που η αποτελεσματικότητά τους βασίζεται στην πλήρως αναλυτική μοντελοποίηση του κτηρίου [28, 203] ή στην ανάλυση ενός μεγάλου όγκου ιστορικών δεδομένων [92, 186], η προτεινόμενη λύση παρέχει πολύ καλά αποτελέσματα λαμβάνοντας υπόψη μόνο ένα μικρό υποσύνολο πληροφοριών. Επιπλέον, η προσέγγισή μας δεν εφαρμόζει χρονοβόρες προσομοιώσεις, ενώ επίσης δεν χρειάζεται προηγούμενη μοντελοποίηση για τη δυναμική του κτιρίου και των κλιματιστικών. Έτσι, παρουσιάζει σημαντικά χαμηλότερη υπολογιστική πολυπλοκότητα και στοχεύει να δίνει άμεση λύση.

Ο ελεγκτής, αρχικά, λαμβάνει ανεπεξέργαστα δεδομένα από αισθητήρες (εσωτερική και εξωτερική θερμοκρασία και υγρασία, ηλιακή ακτινοβολία κλπ.) και στη συνέχεια, τα μοντέλα ενέργειας και θερμικής άνεσης ενημερώνονται κατάλληλα για να προσεγγίσουν τη δυναμική του κτηρίου.

Τα μοντέλα στηρίζονται σε τεχνικές γραμμικής παλινδρόμησης. Η γραμμική παλινδρόμηση είναι μια τεχνική της στατιστικής με την οποία ερευνάται η συσχέτιση μεταξύ μιας εξαρτώμενης μεταβλητής (μεταβλητής εξόδου) και ενός συνόλου ανεξάρτητων μεταβλητών. Οι μέθοδοι αυτοί έχουν ως στόχο την εύρεση του καταλληλότερου μοντέλου που προσαρμόζεται καλύτερα στα δεδομένα και στις παρατηρήσεις. Σε κάθε χρονικό βήμα, τα βάρη των μοντέλων γραμμικής παλινδρόμησης υπολογίζονται εκ νέου. Τέλος, προχωράμε στην βελτιστοποίηση πολλαπλών κριτηρίων που αποτελεί την καρδιά της λήψης αποφάσεων. Ο μηχανισμός αυτός υπολογίζει οι τιμές θερμοκρασίας που θα δίνονται ανά θερμική ζώνη, χρησιμοποιώντας μια τροποποιημένη έκδοση του αλγορίθμου σακιδίου (Knapsack).

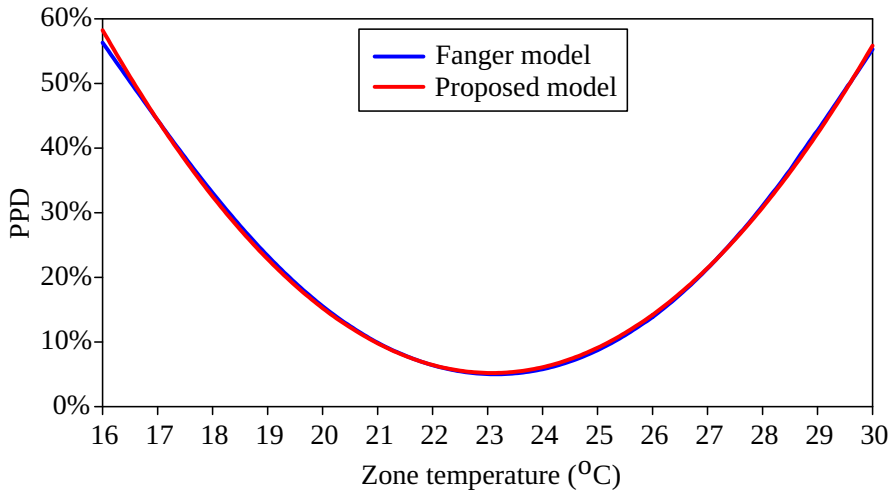
Παράθυρα διαχείρισης δεδομένων Οι τιμές των αισθητήρων αποθηκεύονται προσωρινά για να υποστηρίξουν την ανανέωση των μοντέλων. Εδώ χρησιμοποιούνται δύο συμπληρωματικοί μηχανισμοί, που στηρίζονται στην ιδέα του συρόμενου παραθύρου (sliding window) προκειμένου να μειώνουν την ποσότητα δεδομένων. Με αυτόν τον τρόπο, αντί να αποθηκεύονται όλα τα δεδομένα που έρχονται από τους αισθητήρες, το ένα παράθυρο κρατάει τα δεδομένα που αντιστοιχούν στις τελευταίες ημέρες, ενώ μια περαιτέρω βελτίωση επιτυγχάνεται δίνοντας έμφαση μόνο σε μια συγκεκριμένη διάρκεια ανά ημέρα.



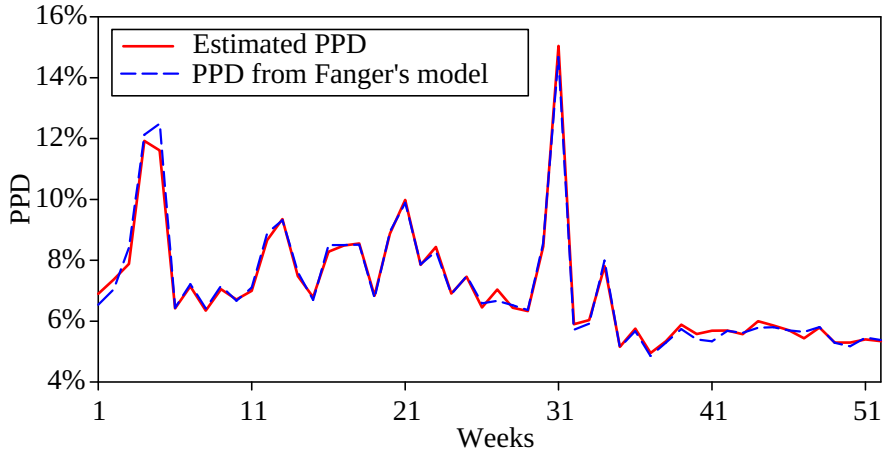
Σχήμα 99: Η προτεινόμενη διαχείριση δεδομένων μέσω παραθύρων

Οπότε τελικά αποθηκεύεται μόνο ένα μικρό υποσύνολο δεδομένων όπως φαίνεται στο Σχήμα 99. Τόσο το μέγεθος του παραθύρου που ορίζει τον αριθμό των προηγούμενων ημερών, όσο και το μέγεθος του παραθύρου που ορίζει την διάρκεια ανά ημέρα, μπορούν να προσαρμόζονται προκειμένου να επιτυγχάνεται η καλύτερη ποιότητα μοντέλων αλλά και η χρήση των διαθέσιμων υπολογιστικών πόρων. Με αυτήν την τεχνική επιτυγχάνουμε γρήγορα αποτελέσματα και μειώνουμε σημαντικά τις απαιτήσεις επεξεργαστικής ισχύος και αποθήκευσης. Η πρόταση χρήσης αυτής της τεχνικής διευκολύνει την ενσωμάτωση του προτεινόμενου μηχανισμού σε χαμηλού κόστους ενσωματωμένα συστήματα (π.χ. έξυπνος θερμοστάτης).

Μοντέλο προσέγγισης θερμικής άνεσης Η θερμική άνεση είναι η συνθήκη του νου που εκφράζει την ικανοποίησή του ανθρώπου για το θερμικό περιβάλλον. Στην παρούσα μελέτη χρησιμοποιούμε το μοντέλο PMV/PPD, που αναπτύχθηκε από τον P.O Fanger [1] και εκφράζει την δυσαρέσκεια με τις θερμικές συνθήκες. Αυτή η τιμή εξαρτάται, μεταξύ άλλων, από την εσωτερική θερμοκρασία, την υγρασία, την ταχύτητα του αέρα, τους μεταβολικούς ρυθμούς (δραστηριότητα) και το είδος ένδυσης. Προτείνουμε ένα μοντέλο που συσχετίζει την δυσαρέσκεια μόνο με την θερμοκρασία του δωματίου, καθώς οι υπόλοιπες παράμετροι (μεταβολικός ρυθμός, ρούχα κ.λπ.) είναι δύσκολο να μετρηθούν αλλά μπορούν να θεωρηθούν σταθερές για σχετικά σύντομο χρονικό διάστημα. Το αποτέλεσμα αυτής της ανάλυσης για μια χειμερινή ημέρα απεικονίζεται στο Σχήμα 100 (αντίστοιχα αποτελέσματα βλέπουμε επίσης και για το καλοκαίρι). Με βάση το πείραμα αυτό, οι τιμές του Fanger (μπλε χρώμα) προσεγγίζονται από ένα τετραγωνικό μοντέλο (κόκκινο χρώμα), όπου η μεταβλητή μας θα είναι μόνο η εσωτερική θερμοκρασία.



Σχήμα 100: Η εκτίμηση της δυσαρέσκειας σαν συνάρτηση της εσωτερικής θερμοκρασίας



Σχήμα 101: Ακρίβεια πρόβλεψης θερμικής δυσαρέσκειας

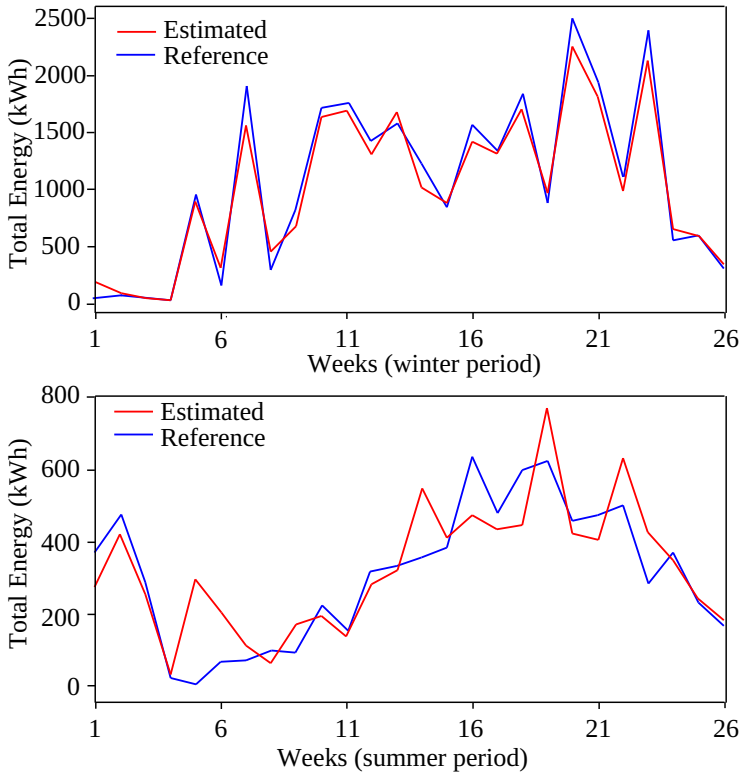
Τα συρόμενα παράθυρα που παρουσιάσαμε παραπάνω ενεργοποιούν αυτήν την προσέγγιση, καθώς η πλειοψηφία των παραμέτρων του Fanger που αναφέραμε είναι σχεδόν σταθερές για τις τελευταίες λίγες ημέρες, ενώ ο καιρός της περιοχής και οι παράμετροι που σχετίζονται με τους κατοίκους (π.χ. ρούχα, η δραστηριότητά τους κ.λπ.) μπορούν να θεωρηθούν παρόμοιες για τις ίδιες ώρες μεταξύ διαδοχικών ημερών. Η ακρίβεια της προτεινόμενης λύσης σε σχέση με τα αντίστοιχα αποτελέσματα της λύσης του Fanger, φαίνεται στο Σχήμα 101, όπου παρατηρούμε ότι το μέσο σφάλμα μεταξύ των δύο μετρήσεων είναι μόλις 0,02% για ολόκληρο το έτος.

Μοντέλο προσέγγισης ενεργειακής κατανάλωσης Αντίστοιχα με την μέθοδο της εκτίμησης της θερμικής άνεσης, σχεδιάσαμε μια μέθοδο για την προσέγγιση της επίδρασης των θερμοκρασιών που επιλέγει ο μηχανισμός ελέγχου στην ενεργειακή κατανάλωση. Καθώς οι αισθητήρες μέτρησης μετρούν την ενέργεια που ήδη έχει καταναλωθεί, ένα τέτοιο μοντέλο (πρόβλεψης) καθίσταται απολύτως απαραίτητο. Η λύση μας υποθέτει ότι κάθε έξυπνος θερμοστάτης κατασκευάζει το δικό του μοντέλο για να εξετάσει τις παραμέτρους από το δωμάτιο/θερμική ζώνη στο οποίο βρίσκεται. Τα δεδομένα που λαμβάνουμε από τους αισθητήρες του κτιρίου περιλαμβάνουν: 1) Συνθήκες εξωτερικού χώρου: Τρέχουσες καιρικές συνθήκες και πρόγνωση καιρού (δηλ. Θερμοκρασία και ηλιακή ακτινοβολία), 2) Εσωτερικές συνθήκες κτιρίου: Θερμοκρασία δωματίου, αριθμός ατόμων ανά δωμάτιο και ούτω καθεξής. Χρησιμοποιούμε ένα μοντέλο γραμμικής παλινδρόμισης, τα βάρη του οποίου υπολογίζονται εκ νέου σε κάθε χρονικό βήμα. Η γραμμική προσέγγιση της ενέργειας, χρησιμοποιείται ευρέως στη βιβλιογραφία [96, 191, 210]. Στο Σχήμα 102 αξιολογείται η ακρίβεια του προτεινόμενου μοντέλου σε σύγκριση με την πραγματική κατανάλωση ενέργειας που δίνουν οι συσκευές μέτρησης (στην προσομοίωση του EnergyPlus). Τα αποτελέσματα δείχνουν ότι το προτεινόμενο μοντέλο εμφανίζει ένα μέσο σφάλμα για ολόκληρο το έτος της τάξεως του 2,5%.

Μέθοδος λήψης αποφάσεων Τα μοντέλα θερμικής άνεσης και ενεργειακής κατανάλωσης δίνονται ως είσοδος στην τελευταία φάση της λύσης, όπου γίνεται η βελτιστοποίηση του κόστους και υπολογίζονται οι αποφάσεις του συστήματος. Για την λύση μας χρησιμοποιούμε μια τροποποίηση του προβλήματος του σακιδίου με σύστημα πολλαπλών επιλογών: 'Έχουμε K σετ αντικειμένων όπου κάθε αντικείμενο έχει μια αξία και ένα βάρος. Σκοπός είναι να επιλέξουμε να βάλουμε στο σακίδιο μας ένα ακριβώς στοιχείο από κάθε σετ, έτσι ώστε το συνολικό βάρος να είναι μικρότερο ή ίσο με το μέγιστο βάρος που μπορεί να μπει στο σακίδιο μας και να μεγιστοποιείται η συνολική μας αξία'. Στο πρόβλημα μας στόχος είναι η ελαχιστοποίηση της συνολικής τιμής (κόστους) αντί για την μεγιστοποίηση της αξίας. Κάθε σετ αναφέρεται σε ένα χρονικό βήμα της λειτουργίας του συστήματός μας και ένα στοιχείο αντιπροσωπεύει το σημείο θερμοκρασίας σχετικά με το προαναφερθέν χρονικό βήμα, το οποίο με τη σειρά του σχετίζεται με την τιμή δυσαρέσκειας και την ενεργειακή του κατανάλωση.

Πειραματικά αποτελέσματα

Στην ενότητα αυτή παρέχουμε μια σειρά ποσοτικών συγκρίσεων προκειμένου να δείξουμε την ποιότητα της λύσης που προτείνουμε, σε σύγκριση με τις σχετικές προσεγγίσεις. Οι λύσεις με τις οποίες συγκρινόμαστε είναι:



Σχήμα 102: Ακρίβεια πρόβλεψης ενεργειακής κατανάλωσης

- Σταθερή θερμοκρασία: 8 λύσεις από 20^oC έως 27^oC.
- Μια αναλυτική μέθοδος που κάνει πλήρη προσομοίωση του κτηρίου και λύνει το πρόβλημα με ευριστικό τρόπο (Fmincon+EnergyPlus). Μάλιστα θεωρούμε την λύση αυτή ως βέλτιστη λύση (λύση αναφοράς) καθώς υποθέτουμε ότι έχουμε 100% ακριβή προσομοίωση και πρόβλεψη καιρού [203] [182].

Τα πειράματά μας έχουν χρονική διάρκεια ενός χρόνου. Στον Πίνακα 20 ποσοτικοποιείται το τελικό κόστος (κανονικοποιημένο) που προέρχεται από τον προτεινόμενο αλγόριθμο λήψης αποφάσεων σε σύγκριση με τις λύσεις που προαναφέραμε για την Λειτουργία 1 (συμβιβασμός μεταξύ ενέργειας και δυσαρέσκιας). Αυτή η ανάλυση δείχνει ότι η λύση μας έχει ανώτερη απόδοση έναντι στις στατικές επιλογές. Επιπλέον, επιτυγχάνουμε συγκρίσιμη απόδοση (3% χειρότερη) με την αναλυτική-ευριστική λύση, η οποία όμως είναι βασισμένη σε πλήρη προσομοίωση. Ακόμα και αν παρακάμψουμε τους περιορισμούς που επιφέρει η ανάγκη για ακριβή προσομοίωση των κτηρίων, μια τέτοια λύση παρουσιάζει σημαντικά υψηλότερες υπολογιστικές απαιτήσεις συγκριτικά με την

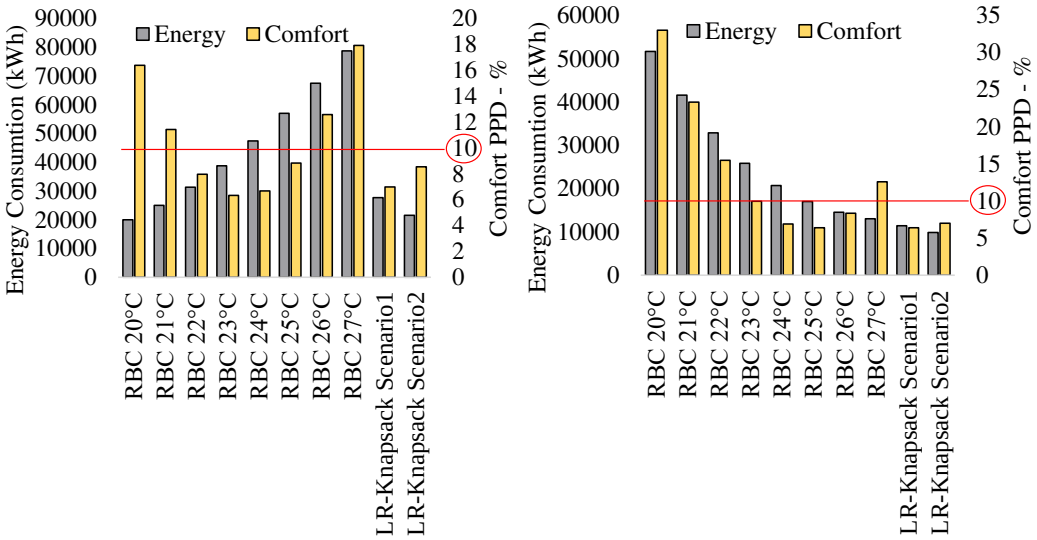
Πίνακας 20: Σύγκριση κόστους με άλλες μεθόδους

Μέθοδος	Ενέργεια (kWh)	PPD (%)	Κόστος
Σταθερά $20°C$	66,967	24.99	0.89
Σταθερά $21°C$	62,939	17.46	0.72
Σταθερά $22°C$	61,223	11.66	0.59
Σταθερά $23°C$	61,955	7.94	0.52
Σταθερά $24°C$	65,191	6.46	0.51
Σταθερά $25°C$	70,467	7.23	0.56
Σταθερά $26°C$	77,359	10.22	0.66
Σταθερά $27°C$	85,680	15.31	0.81
Fmincon	34,936	6.17	0.33
Προτεινόμενη λύση	36,399	6.47	0.34

δική μας.

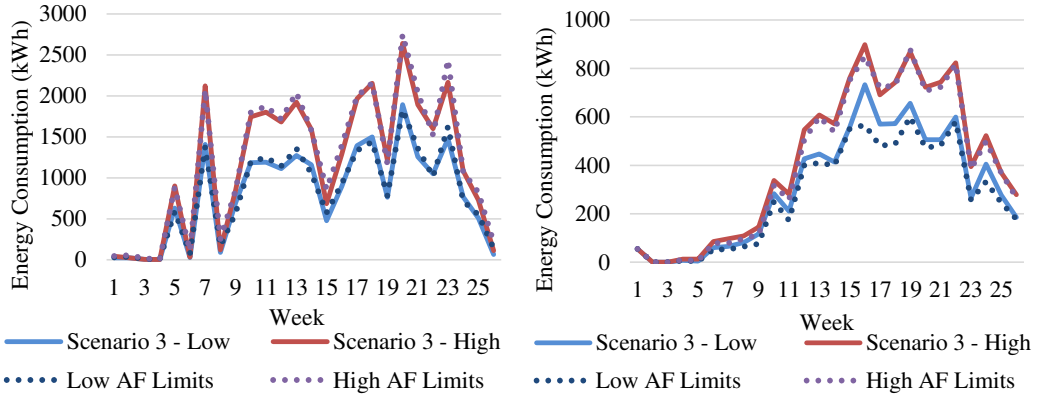
Η Λειτουργία 2 έχει σκοπό την ελαχιστοποίηση της ενεργειακής κατανάλωσης, διατηρώντας πάντα την τήρηση των ορίων του προτύπου ASHRAE όσον αφορά την θερμική άνεση (10%). Τα αποτελέσματα, τα οποία φαίνονται στο Σχήμα 103, δείχνουν μικρές διακυμάνσεις σε αυτήν τη μέτρηση έως και 1%. Αυτό συμβαίνει λόγω διαφόρων μη προβλέψιμων συνθηκών, όπως η δυναμική του κτηρίου και η πρόβλεψη του καιρού. Για τον χειμώνα, η μέση βελτίωση στην κατανάλωση ενέργειας που πετυχαίνουμε σε σχέση με την συμβιβαστική Λειτουργία 1 και τις προηγούμενες λύσεις είναι 27%. Επίσης πετυχαίνουμε περίπου ίδια κατανάλωση ενέργειας με τους 20^oC αλλά με δυσαρέσκεια 8.5% αντί του μη αποδεκτού 16.4%. Το καλοκαίρι η ενέργεια είναι χαμηλότερη από όλες τις άλλες περιπτώσεις. Σύμφωνα με τα αποτελέσματα αυτά, ισχυριζόμαστε ότι ο προτεινόμενος αλγόριθμος λήψης αποφάσεων μπορεί να πετυχαίνει τον στόχο της διατήρησης αποδεκτών θερμικών συνθηκών εσωτερικού χώρου με το ελάχιστο δυνατό κόστος ενέργειας.

Προκειμένου να αξιολογηθεί η αποτελεσματικότητα της Λειτουργίας 3, το Σχήμα 104 απεικονίζει την Ενέργεια και την θερμική άνεση για 2 διαφορετικά σενάρια διαθέσιμων πόρων για αγορά ενέργειας. Αυτά τα αποτελέσματα δείχνουν ότι η όταν δίνουμε μικρότερους πόρους (κατά 20% συγκριτικά με την λύση αναφοράς Fmincon+EnergyPlus) για αγορά ενέργειας, η λύση μας πετυχαίνει κατά μέσο όρο 28% υψηλότερη τιμή PPD σε σχέση με την λύση αναφοράς και 36% από αυτή που πετυχαίνει με την επιπλέον προσθήκη διαθέσιμων πόρων κατά 20%, ενώ είναι εμφανής η τήρηση του ορίου πόρων που δώσαμε στο σύστημα λήψης αποφάσεων. Οι μόνες φορές που το υπερβαίνει, είναι όταν οι διαθέσιμοι πόροι που δίνουμε είναι πολύ μικροί για να καλύψουν το όριο το PPD που δίνεται από το πρότυπο ASHRAE.



(α′) Χειμώνας (β′) Καλοκαίρι

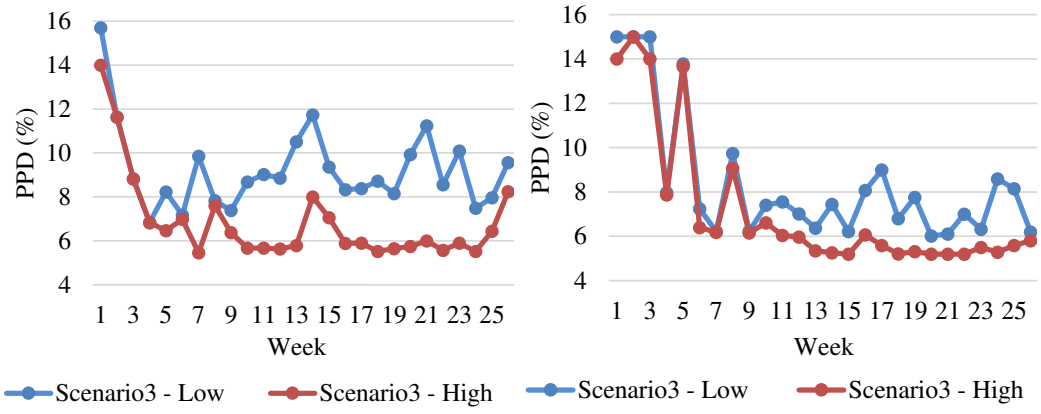
Σχήμα 103: Ενέργεια και θερμική άνεση για την Λειτουργία 2 (Βελτίωση ενέργειας με όριο 10% για την δυσαρέσκεια)



(α′) Χειμώνας (β′) Καλοκαίρι

Σχήμα 104: Αξιολόγηση της ενεργειακής κατανάλωσης για την λειτουργία 3 (βελτίωση άνεσης με όριο για τους διαθέσιμους πόρους ενέργειας)

Η προτεινόμενη λύση είναι επίσης αποδεκτή από τους ανθρώπους στο κτήριο όπως φαινεται στο Σχήμα 105, καθώς η τιμή PPD σπανίως υπερβαίνει το όριο του 10% που ορίζεται από το πρότυπο ASHRAE. Οι ελάχιστες περιπτώσεις που το υπερβαίνει είναι το φθινόπορο και την άνοιξη. Τις εποχές αυτές οι θερμοκρασίες είναι μεσαίες ενώ το κλιματιστικό μόνιμα ρυθμισμένο ή σε θέρμανση ή σε ψύξη. Οπότε αν για παράδειγμα τον Οκτώβριο κάνει ζέστη και το κλιματι-



(α′) Χειμώνας (β′) Καλοκαίρι

Σχήμα 105: Αξιολόγηση της θερμικής δυσαρέσκιας για την λειτουργία 3 (βελτίωση άνεσης με όριο για τους διαθέσιμους πόρους ενέργειας)

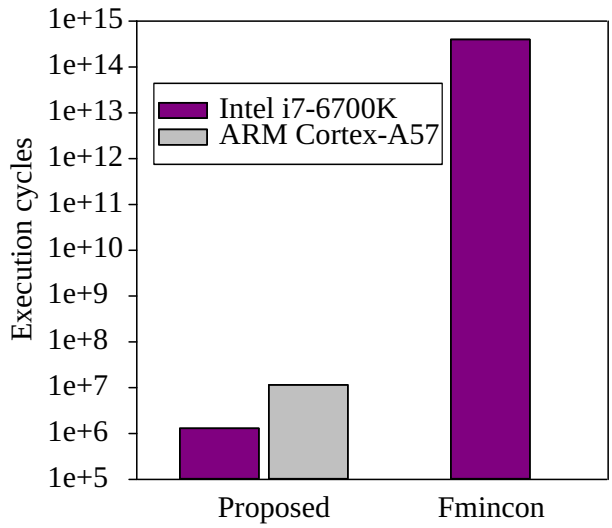
Πίνακας 21: Χρόνος εκτέλεσης προτεινόμενης λύσης σε διάφορες συσκευές

	ARM Cortex-A57 @2 GHz	Intel Quarkx1000 Intel @400MHz	ARMx STM32F103 @72MHz
Λειτουργία 1 & Λειτουργία 2	0.004 sec	0.006 sec	0.082 sec
Λειτουργία 3	0.03 sec	0.035 sec	0.576 sec

στικό μπορεί να προσφέρει μόνο θέρμανση, παραμένει κλειστό χωρίς να μπορεί να μειώσει το PPD.

Υλοποίηση σε ενσωματωμένο σύστημα Προκειμένου να μελετήσουμε τον χρόνο εκτέλεσης της λύσης μας υλοποιήσαμε τον μηχανισμό σε διάφορες ενσωματωμένες συσκευές που θα μπορούσαν εύκολα να αποτελέσουν το υλικό ενός έξυπνου θερμοστάτη. Τα αποτελέσματα των μετρήσεών μας φαίνονται στον Πίνακα 21. Αυτά τα αποτελέσματα υπογραμμίζουν ότι ο προτεινόμενος μηχανισμός είναι σε θέση να υπολογίσει τις θερμοκρασίες σε λιγότερο από ένα δευτερόλεπτο, ακόμη και αν χρησιμοποιείται ένας χαμηλής απόδοσης μικροελεγκτής (ARMx STM32F103 στα 72 MHz). Ο πρόσθετος χρόνος εκτέλεσης της Λειτουργίας 3 οφείλεται στο ότι οι αποφάσεις θερμοκρασίας υπολογίζονται για όλες τις ζώνες μια εβδομάδα μπροστά (προκειμένου να τηρούνται τα όρια των διαθέσιμων πόρων για αγορά ενέργειας).

Το Σχήμα 106 δείχνει τον αριθμό των κύκλων μηχανής που χρειάζεται η λύση μας συγκριτικά με την αναλυτική μέθοδο. Για αυτό το πείραμα χρησιμοποιούμε ένα ενσωματωμένο σύστημα ARM Cortex-A57 και ένα κεντρικό server που έχει έναν Intel i7-6700K@4GHz. Με βάση τα αποτελέσματα αυτά,



Σχήμα 106: Αριθμός κύκλων μηχανής για τον υπολογισμό των θερμοκρασιών για την προτεινόμενη και την αναλυτική λύση αναφοράς

η λύση μας είναι γρηγορότερη κατά 8 τάξεις μεγέθους. Κατά συνέπεια, μία πλατφόρμα χαμηλού κόστους μπορεί να υποστηρίξει την προτεινόμενη λύση και να χρησιμοποιηθεί για τον έλεγχο των θερμοκρασιών εσωτερικού χώρου, κάτι που δεν είναι εφικτό για τον μεθόδους που χρησιμοποιούν προσομοίωση (λόγω της εγγενούς αυξημένης υπολογιστικής πολυπλοκότητας).

Συμπεράσματα

Τα συμπεράσματα που απορρέουν από αυτήν την διατριβή συνοψίζονται παρακάτω:

- Η δημιουργία πρακτικών εργαλείων ικανών να αναλύουν τον πηγαίο κώδικα εφαρμογών και να εκτιμούν την ενέργεια που αυτές θα καταναλώσουν, χωρίς την ανάγκη εκτέλεσής τους στις τελικές συσκευές είναι εφικτή. Η ενσωμάτωση αυτών των εργαλείων μέσα σε περιβάλλοντα ανάπτυξης εφαρμογών θεωρούμε ότι θα επιφέρει σημαντικά οφέλη κατά τον σχεδιασμό IoT εφαρμογών.

- Η πρόβλεψη του αναμενόμενου κέρδους σε ενέργεια από την επιτάχυνση των εφαρμογών σε ετερογενείς συσκευές είναι εφικτή.

- Το κόστος της δυναμικής ανάλυσης υπολογιστικών απαιτήσεων εφαρμογών είναι μεγάλο και μπορεί να αποφευχθεί με χρήση στατικής ανάλυσης όταν δεν επιθυμείται η μέγιστη δυνατή ακρίβεια αποτελεσμάτων.

- Η ποσοτικοποίηση της προσπάθειας που χρειάζεται να καταβληθεί για τον μετασχηματισμό εφαρμογών με στόχο την βελτίωση της επίδοσης και της ενέργειας είναι σημαντική και η πρόβλεψη της είναι εφικτή.

- Οι προτεινόμενες μεθοδολογίες είναι εύκολα επεκτάσιμες προκειμένου να υποστηρίζουν περισσότερες συσκευές και αρχιτεκτονικές.

- Τα εργαλεία εκτίμησης της ενέργειας που προτείνονται μπορούν να υποστηρίξουν τον σχεδιασμό μεθόδων ενεργειακά αποδοτικής τοποθέτησης συναρτήσεων στους διαθέσιμους υπολογιστικούς πόρους του δικτύου.

- Η δημιουργία online συστημάτων λήψης αποφάσεων για εφαρμογές Κυβερνο-φυσικών (Cyber-Physical) συστημάτων είναι πολύ σημαντική για τον έλεγχο υπαρχόντων συστημάτων. Στατιστικά μοντέλα χαμηλής πολυπλοκότητας και αλγόριθμοι βελτιστοποίησης πολλών κριτηρίων μπορούν να συμβάλουν προς αυτή την κατεύθυνση. Η σχεδίαση έξυπνων θερμοστατών για τον έλεγχο της θέρμανσης/ψύξης σε υπάρχοντα κτήρια χωρίς προηγούμενη μοντελοποίηση που προτείναμε είναι ένα χαρακτηριστικό παράδειγμα.

- Η αποδοτική διαχείριση των δεδομένων μπορεί να συμβάλει στην δημιουργία ελεγκτών για εφαρμογές Κυβερνο-φυσικών συστημάτων χαμηλής πολυπλοκότητας. Η μέθοδος των συρόμενων παραθύρων (sliding windows) αποτελεί μια καλή επιλογή για διαχείριση δεδομένων βασισμένων σε καιρικές συνθήκες ή συμπεριφορές κατοίκων σε κτήρια.

Πίνακας 22: Γλωσσάριο Αντιστοίχισης Αγγλικών-Ελληνικών Όρων

Branch Divergence	Αλλαγή Κατεύθυνσης Διακλαδώσεων
Central Processing Unit	Κεντρική Μονάδα Επεξεργασίας
Classification	Κατηγοριοποίηση
Compiler	Μεταγλωττιστής
Computational Complexity	Υπολογιστική Πολυπλοκότητα
Control Operation Instructions	Εντολές Ελέγχου
Correlation	Συσχέτιση
CPU cycles	Κύκλοι Μηχανής

Cyber-Physical System	Κυβερνο-φυσικό Σύστημα
Decision-making Algorithm	Αλγόριθμος Λήψης Αποφάσεων
Embedded System	Ενσωματωμένο Σύστημα
Energy Efficiency	Ενεργειακή Αποδοτικότητα
Ensemble Voting Method	Συνδυαστική Μέθοδος Ψηφοφορίας
Feature Selection	Επιλογή Χαρακτηριστικών
Floating-point Operations	Πράξεις Κινητής Υποδιαστολής
Function Placement	Τοποθέτηση Συναρτήσεων
Graphics Processing Unit	Κάρτα Γραφικών
Heterogeneous System	Ετερογενές Σύστημα
Insrtuction Level Parallelism	Βαθμός Παραλληλοποίησης Εντολών
Internet of Things	Διαδίκτυο των Αντικειμένων
Knapsack algorithm	Αλγόριθμος Σακιδίου
Kubernetes (software)	Κυβερνήτης
Loop	Βρόχος
Memory Management	Διαχείριση Μνήμης
Microcontroller	Μικροελεγκτής
Model Training	Εκπαίδευση Μοντέλου
Neural Network	Νευρωνικό Δίκτυο
Profiling	Ανάλυση Υπολογιστικών Απαιτήσεων
Programming Effort	Προγραμματιστική Προσπάθεια
Random Forest Method	Μέθοδος Τυχαίων Δασών
Regression Analysis	Ανάλυση Παλινδρόμησης
Resource Management	Διαχείριση Πόρων
Serverless Computing	Υπολογισμός χωρίς Διακομιστή
Sliding window technique	Τεχνική συρόμενου Παραθύρου
Smart Building	Έξυπνο Κτήριο
Smart Thermostat	Έξυπνος Θερμοστάτης
Software Development Kit	Περιβάλλον Ανάπτυξης Εφαρμογών
Source Code	Πηγαίος Κώδικας
Thermal Comfort	Θερμική Άνεση

List of Publications

Book Chapters

B1. **Charalampos Marantos**, Christos Lamprakos, Kostas Siozios, and Dimitrios Soudris. "Towards Plug&Play smart thermostats for building's heating/cooling control." In IoT for Smart Grids, pp. 183-207. Springer, Cham, 2019.

Journals

J6. **Charalampos Marantos**, Lazaros Papadopoulos, Christos P. Lamprakos, Konstantinos Salapas, and Dimitrios Soudris. "Bringing Energy Efficiency closer to Application Developers: An Extensible Software Analysis Framework." IEEE Transactions on Sustainable Computing, *under minor revision*

J5. **Charalampos Marantos**, Lazaros Papadopoulos, Angeliki-Agathi Tsintzira, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Dimitrios Soudris. "Decision support for GPU acceleration by predicting energy savings and programming effort." Sustainable Computing: Informatics and Systems 34 (2022): 100631.

J4. Christos P. Lamprakos, **Charalampos Marantos**, Miltiadis Siavvas, Lazaros Papadopoulos, Angeliki-Agathi Tsintzira, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Dionysios Kehagias, and Dimitrios Soudris. "Translating quality-driven code change selection to an instance of multiple-criteria decision making." Information and Software Technology 145 (2022): 106851.

J3. **Charalampos Marantos**, Konstantinos Salapas, Lazaros Papadopoulos, and Dimitrios Soudris. "A flexible tool for estimating applications performance and energy consumption through static analysis." SN Computer Science 2, no. 1 (2021): 1-11.

J2. **Charalampos Marantos**, Kostas Siozios, and Dimitrios Soudris. "Rapid prototyping of low-complexity orchestrator targeting cyberphysical systems: The smart-thermostat usecase." IEEE Transactions on Control Systems Technology 28, no. 5 (2019): 1831-1845.

J1. **Charalampos Marantos**, Kostas Siozios, and Dimitrios Soudris. "A flexible decision-making mechanism targeting smart thermostats." IEEE Embedded Systems Letters 9, no. 4 (2017): 105-108.

Conferences - Workshops

C14. **Charalampos Marantos**, Nikolaos Maidonis, and Dimitrios Soudris. "Designing Application Analysis Tools for Cross-Device Energy Consumption Estimation" In 2022 11th International Conference on Modern Circuits and Systems Technologies (MOCAST), pp. 1-4. IEEE, 2022.

C13. **Charalampos Marantos**, Miltiadis Siavvas, Dimitrios Tsoukalas, Christos P. Lamprakos, Lazaros Papadopoulos, Paweł Boryszko, Katarzyna Filus, Joanna Domanska, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Erol Gelenbe, Dionysios Kehagias and Dimitrios Soudris. "SDK4ED: One-click platform for Energy-aware, Maintainable and Dependable Applications." In 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 981-986. IEEE, 2022.

C12. Achilleas Tzenetopoulos, **Charalampos Marantos**, Giannos Gavrielides, Sotirios Xydis, and Dimitrios Soudris. "FADE: FaaS-inspired application decomposition and Energy-aware function placement on the Edge." In Proceedings of the 24th International Workshop on Software and Compilers for Embedded Systems, pp. 7-10. 2021.

C11. Daichi Watari, Ittetsu Taniguchi, Francky Catthoor, **Charalampos Marantos**, Kostas Siozios, Elham Shirazi, Dimitrios Soudris, and Takao Onoye. "Thermal Comfort Aware Online Energy Management Framework for a Smart Residential Building." In 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 535-538. IEEE, 2021.

C10. Christos P. Lamprakos, **Charalampos Marantos**, Lazaros Papadopoulos, Dimitrios Soudris. (2021). The Known Unknowns: Discovering Trade-Offs Between Heterogeneous Code Changes. In: Orailoglu, A., Jung, M., Reichenbach, M. (eds) Embedded Computer Systems: Architectures, Modeling, and Simulation. SAMOS 2021. Lecture Notes in Computer Science, vol 13227. Springer, Cham.

C9. Chrysostomos Karakasis, Konstantinos Machairas, **Charalampos Marantos**, Iosif S. Paraskevas, Evangelos Papadopoulos, and Dimitrios Soudris. "Exploiting the SoC FPGA Capabilities in the Control Architecture of a Quadruped Robot." In 2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), pp. 501-507. IEEE, 2020.

C8. **Charalampos Marantos**, Angeliki-Agathi Tsintzira, Lazaros Papadopoulos, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Dimitrios Soudris. "Technical Debt Management and Energy Consumption Evaluation in Implantable Medical Devices: The SDK4ED Approach." In International Conference on Embedded Computer Systems, pp. 348-358. Springer, Cham, 2020.

C7. Miltiadis Siavvas, Dimitrios Tsoukalas, **Charalampos Marantos**, Angeliki-Agathi Tsintzira, Marija Jankovic, Dimitrios Soudris, Alexander Chatzigeorgiou, and Dionysios Kehagias. "The sdk4ed platform for embedded software quality improvement-preliminary overview." In International Conference on Computational Science and Its Applications, pp. 1035-1050. Springer, Cham, 2020.

C6. Miltiadis Siavvas, **Charalampos Marantos**, Lazaros Papadopoulos, Dionysios Kehagias, Dimitrios Soudris, and Dimitrios Tzovaras. "On the relationship between software security and energy consumption." In 15th China-Europe International Symposium on software engineering education. 2019.

C5. **Charalampos Marantos**, Christos P. Lamprakos, Vasileios Tsoutsouras, Kostas Siozios, and Dimitrios Soudris. "Towards plug&play smart thermostats inspired by reinforcement learning." In Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications, pp. 39-44. 2018.

C4. Lazaros Papadopoulos, **Charalampos Marantos**, Georgios Digkas, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Dimitrios Soudris. "Interrelations between software quality metrics, performance and energy consumption in embedded applications." In Proceedings of the 21st International Workshop on software and compilers for embedded systems, pp. 62-65. 2018.

C3. **Charalampos Marantos**, Nikolaos Karavalakis, Vasileios Leon, Vasileios Tsoutsouras, Kiamal Pekmestzi, and Dimitrios Soudris. "Efficient support vector machines implementation on Intel/Movidius Myriad 2."

In 2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST), pp. 1-4. IEEE, 2018.

C2. Helbert Arenas, Aurélie Baker, Damian Bargiel, Matthias Becker, Anna Bialczak, Francesco Carbone, Véronique Gaildrat, Sascha Heising, Md Bayzidul Islam, Philippe Lattes, **Charalampos Marantos**, Colette Menou, Josiane Mothe, Aude Nzeh Ngong, Iosif S. Paraskevas, Miguel Penalver, Paulina Sciana, Dimitrios Soudris "FabSpaces at ImageCLEF 2017-Population Estimation (Remote) Task." In International Conference of the CLEF Association, CLEF 2017 Labs Working Notes (CLEF 2017), vol. 1866, pp. pp-1. 2017.

C1. **Charalampos Marantos**, Iosif S. Paraskevas, Kostas Siozios, Josiane Mothe, Colette Menou, and Dimitrios Soudris. "FabSpace 2.0: A platform for application and service development based on Earth Observation data." In 2017 6th International Conference on Modern Circuits and Systems Technologies (MOCAST), pp. 1-4. IEEE, 2017.

Curriculum Vitae

Charalampos Marantos received the Diploma and the PhD degrees from the Department of Electrical and Computer Engineering, National Technical University of Athens, Greece, in 2016 and 2022. From 2016 to 2022, he worked as a Research Assistant and Freelance Computer Engineer in National Technical University of Athens (NTUA) and Institute of Communication and Computer Systems (ICCS). His research interests include Embedded Systems Programming, Decision-making mechanisms targeting IoT and Cyber-Physical systems applications, Machine Learning, Energy/Performance optimization for embedded systems applications. He worked and collaborated on many EU research projects such as the SDK4ED, which was about designing a Software Development Toolkit for Energy Optimization and Technical Debt Elimination, targeting Embedded Systems applications and the FABSPACE 2.0, which created an open-innovation network for geodata-driven information. He is a co-author of more than 20 international journal and conference research papers. During his Ph.D. he co-supervised multiple master theses in the fields of Cyber-Physical Systems, Machine Learning and Embedded Systems Developing, while offering assistant teaching in Microprocessors courses in the fields of Microprocessors architecture and embedded systems development (C and Assembly). Finally, he offered System Administration services such as Linux server management (OS, installations, users, dockers, SW licenses), purchase, installation and maintenance of equipment.

References

[1] P. O. Fanger *et al.*, "Thermal comfort. analysis and applications in environmental engineering." *Thermal comfort. Analysis and applications in environmental engineering.*, 1970.

[2] A. Hamm, A. Willner, and I. Schieferdecker, "Edge computing: a comprehensive survey of current initiatives and a roadmap for a sustainable edge computing development," *15th International Conference on Wirtschaftsinformatik*, 2019.

[3] B. Ramprasad, A. da Silva Veith, M. Gabel, and E. de Lara, "Sustainable computing on the edge: A system dynamics perspective," in *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, 2021, pp. 64–70.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE communications surveys & tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[6] S. Georgiou, S. Rizou, and D. Spinellis, "Software development lifecycle for energy efficiency: techniques and tools," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–33, 2019.

[7] K. Georgiou, S. Xavier-de Souza, and K. Eder, "The iot energy challenge: A software perspective," *IEEE Embedded Systems Letters*, vol. 10, no. 3, pp. 53–56, 2017.

[8] Y. Chen, T. Chen, Z. Xu, N. Sun, and O. Temam, "Diannao family: energy-efficient hardware accelerators for machine learning," *Communications of the ACM*, vol. 59, no. 11, pp. 105–112, 2016.

[9] "Specifications of Nvidia Tegra X1:," https://shield.nvidia.com/blog/tegra-x1-processor-and-shield, 2017.

[10] "Specifications of Intel/Movidius Myriad:," https://www.movidius.com/myriadx, 2019.

[11] C. Marantos, N. Karavalakis, V. Leon, V. Tsoutsouras, K. Pekmestzi, and D. Soudris, "Efficient support vector machines implementation on intel/movidius myriad 2," in *2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2018, pp. 1–4.

[12] "Xilinx zynq 7000," https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html, 2022.

[13] E. Kern, "Green computing, green software, and its characteristics: Awareness, rating, challenges," in *From Science to Society*. Springer, 2018, pp. 263–273.

[14] I. Manotas, C. Bird, R. Zhang, D. Shepherd, C. Jaspan, C. Sadowski, L. Pollock, and J. Clause, "An empirical study of practitioners' perspectives on green software engineering," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 237–248.

[15] G. Pinto and F. Castor, "Energy efficiency: a new concern for application software developers," *Communications of the ACM*, vol. 60, no. 12, pp. 68–75, 2017.

[16] G. Procaccianti, H. Fernández, and P. Lago, "Empirical evaluation of two best practices for energy-efficient software development," *Journal of Systems and Software*, vol. 117, pp. 185–198, 2016.

[17] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, "What do programmers know about software energy consumption?" *IEEE Software*, vol. 33, no. 3, pp. 83–89, 2015.

[18] H. Ribic and Y. D. Liu, "Energy-efficient work-stealing language runtimes," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 513–528, 2014.

[19] V. Leon, G. Zervakis, D. Soudris, and K. Pekmestzi, "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 421–430, 2017.

[20] K. Aggarwal, A. Hindle, and E. Stroulia, "Greenadvisor: A tool for analyzing the impact of software evolution on energy consumption," in *2015 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 2015, pp. 311–320.

[21] J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*, 2012, pp. 47–56.

[22] K. Eder, J. P. Gallagher, G. Fagas, L. Gammaitoni, and D. Paul, "Energy-aware software engineering," *ICT-energy concepts for energy efficiency and sustainability*, pp. 103–127, 2017.

[23] E. A. Lee, "Cps foundations," in *Design automation conference*. IEEE, 2010, pp. 737–742.

[24] K. Siozios, D. Soudris, and E. Kosmatopoulos, *Cyber-Physical Systems: Decision Making Mechanisms and Applications*. River Publishers, 2017.

[25] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7267 – 7279, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417413005125

[26] B. Sun, P. B. Luh, Q.-S. Jia, Z. Jiang, F. Wang, and C. Song, "Building energy management: Integrated control of active and passive heating, cooling, lighting, shading, and ventilation systems," *IEEE Transactions on automation science and engineering*, vol. 10, no. 3, pp. 588–602, 2012.

[27] K. F. Fong, V. I. Hanby, and T.-T. Chow, "Hvac system optimization for energy management by evolutionary programming," *Energy and Buildings*, vol. 38, no. 3, pp. 220–231, 2006.

[28] C. D. Korkas, S. Baldi, I. Michailidis, and E. B. Kosmatopoulos, "Intelligent energy and thermal comfort management in grid-connected microgrids with heterogeneous occupancy schedule," *Applied Energy*, vol. 149, pp. 194–203, 2015.

[29] N. Lu, "An evaluation of the hvac load potential for providing load balancing service," *IEEE Transactions on Smart Grid*, vol. 3, no. 3, pp. 1263–1270, 2012.

[30] Eurostat, "Energy balance sheets," Data 2002-–2003, Luxemburg, 2005.

[31] Y. He, T. Kvan, M. Liu, and B. Li, "How green building rating systems affect designing green," *Building and Environment*, vol. 133, pp. 19–31, 2018.

[32] D. C. Khedekar, A. C. Truco, D. A. Oteyza, and G. F. Huertas, "Home automation—a fast-expanding market," *Thunderbird International Business Review*, vol. 59, no. 1, pp. 79–91, 2017.

[33] K. Rathouse and B. Young, "Domestic heating: Use of controls," *Defra Market Transformation Programme*, 2004.

[34] E. P. Agency, "Summary of research findings from the programmable thermostat market," Washington, DC: Office of Headquarters, 2004.

[35] R. Pereira, M. Couto, F. Ribeiro, R. Rua, J. Cunha, J. P. Fernandes, and J. Saraiva, "Ranking programming languages by energy efficiency," *Science of Computer Programming*, vol. 205, p. 102609, 2021.

[36] M. Couto, R. Pereira, F. Ribeiro, R. Rua, and J. Saraiva, "Towards a green ranking for programming languages," in *Proceedings of the 21st Brazilian Symposium on Programming Languages*, 2017, pp. 1–8.

[37] E. Capra, C. Francalanci, and S. A. Slaughter, "Is software "green"? application development environments and energy efficiency in open source applications," *Information and Software Technology*, vol. 54, no. 1, pp. 60–71, 2012.

[38] E. Capra and F. Merlo, "Green it: Everything starts from the software," 2009.

[39] C. Bunse and S. Stiemer, "On the energy consumption of design patterns," *Softwaretechnik-Trends: Vol. 33, No. 2*, 2013.

[40] C. Sahin, F. Cayci, I. L. M. Gutierrez, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage," in *2012 First International Workshop on Green and Sustainable Software (GREENS)*. IEEE, 2012, pp. 55–61.

[41] H. Mu and S. Jiang, "Design patterns in software development," in *2011 IEEE 2nd International Conference on Software Engineering and Service Science*. IEEE, 2011, pp. 322–325.

[42] A. Noureddine and A. Rajan, "Optimising energy consumption of design patterns," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 623–626.

[43] D. Li and W. G. Halfond, "An investigation into energy-saving programming practices for android smartphone app development," in *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, 2014, pp. 46–53.

[44] J. Grossschadl, S. Tillich, C. Rechberger, M. Hofmann, and M. Medwed, "Energy evaluation of software implementations of block ciphers under memory constraints," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[45] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[46] A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, 2011, pp. 1–6.

[47] R. Pérez-Castillo and M. Piattini, "Analyzing the harmful effect of god class refactoring on power consumption," *IEEE software*, vol. 31, no. 3, pp. 48–54, 2014.

[48] M. Bazzaz, M. Salehi, and A. Ejlali, "An accurate instruction-level energy estimation model and tool for embedded systems," *IEEE transactions on instrumentation and measurement*, vol. 62, no. 7, pp. 1927–1934, 2013.

[49] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat, "Seep: exploiting symbolic execution for energy-aware programming," *ACM SIGOPS Operating Systems Review*, vol. 45, no. 3, pp. 58–62, 2012.

[50] A. Pathak, Y. C. Hu, and M. Zhang, "Where is the energy spent inside my app? fine grained energy accounting on smartphones with eprof," in *Proceedings of the 7th ACM european conference on Computer Systems*, 2012, pp. 29–42.

[51] L. Papadopoulos, C. Baloukas, and D. Soudris, "Exploration methodology of dynamic data structures in multimedia and network applications for embedded platforms," *Journal of Systems Architecture*, vol. 54, no. 11, pp. 1030–1038, 2008.

[52] Q. Cai, J. González, G. Magklis, P. Chaparro, and A. González, "Thread shuffling: Combining dvfs and thread migration to reduce

energy consumptions for multi-core systems," in *IEEE/ACM International Symposium on Low Power Electronics and Design*. IEEE, 2011, pp. 379–384.

[53] M. A. Awan and S. M. Petters, "Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems," in *2011 23rd Euromicro Conference on Real-Time Systems*. IEEE, 2011, pp. 92–101.

[54] K. Bhatti, C. Belleudy, and M. Auguin, "Power management in real time embedded systems through online and adaptive interplay of dpm and dvfs policies," in *2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. IEEE, 2010, pp. 184–191.

[55] K. Hazelwood and A. Klauser, "A dynamic binary instrumentation engine for the arm architecture," in *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, 2006, pp. 261–270.

[56] A. C. De Melo, "The new linux'perf'tools," in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.

[57] N. Nethercote and J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation," in *ACM Sigplan notices*, vol. 42, no. 6. ACM, 2007, pp. 89–100.

[58] X. Zheng, L. K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[59] X. Zheng, H. Vikalo, S. Song, L. K. John, and A. Gerstlauer, "Sampling-based binary-level cross-platform performance estimation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1709–1714.

[60] W. Wang, P. Mishra, and S. Ranka, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2011, pp. 948–953.

[61] R. Reddy and P. Petrov, "Cache partitioning for energy-efficient and interference-free embedded multitasking," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 9, no. 3, pp. 1–35, 2010.

[62] H. Hajimiri, K. Rahmani, and P. Mishra, "Synergistic integration of dynamic cache reconfiguration and code compression in embedded systems," in *2011 International Green Computing Conference and Workshops*. IEEE, 2011, pp. 1–8.

[63] D. Llamocca, C. Carranza, and M. Pattichis, "Separable fir filtering in fpga and gpu implementations: Energy, performance, and accuracy considerations," in *2011 21st International Conference on Field Programmable Logic and Applications*. IEEE, 2011, pp. 363–368.

[64] C. Karakasis, K. Machairas, C. Marantos, I. S. Paraskevas, E. Papadopoulos, and D. Soudris, "Exploiting the soc fpga capabilities in the control architecture of a quadruped robot," in *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2020, pp. 501–507.

[65] V. Leon, I. Stamoulias, G. Lentaris, D. Soudris, D. Gonzalez-Arjona, R. Domingo, D. M. Codinachs, and I. Conway, "Development and testing on the european space-grade brave fpgas: Evaluation of ng-large using high-performance dsp benchmarks," *IEEE Access*, vol. 9, pp. 131 877–131 892, 2021.

[66] V. Leon, K. Pekmestzi, and D. Soudris, "Exploiting the potential of approximate arithmetic in dsp & ai hardware accelerators," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 263–264.

[67] K. N. Khan, M. Hirki, T. Niemi, J. K. Nurminen, and Z. Ou, "Rapl in action: Experiences in using rapl for power measurements," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 3, no. 2, pp. 1–26, 2018.

[68] R. E. Grant, M. Levenhagen, S. L. Olivier, D. DeBonis, K. Pedretti, and J. H. Laros, "Overcoming challenges in scalable power monitoring with the power api," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 1094–1097.

[69] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe, "Process-level power estimation in vm-based systems," in *Proceedings of the Tenth European Conference on Computer Systems*, 2015, pp. 1–14.

[70] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Petra: a software-based tool for estimating the energy profile of android applications," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2017, pp. 3–6.

[71] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks," in *International Conference on machine learning*. PMLR, 2019, pp. 4505–4515.

[72] C. Ferdinand and R. Heckmann, "ait: Worst-case execution time prediction by static program analysis," in *Building the Information Society*. Springer, 2004, pp. 377–383.

[73] X. Li, Y. Liang, T. Mitra, and A. Roychoudhury, "Chronos: A timing analyzer for embedded software," *Science of Computer Programming*, vol. 69, no. 1-3, pp. 56–67, 2007.

[74] T. Hönig, H. Janker, C. Eibel, O. Mihelic, and R. Kapitza, "Proactive energy-aware programming with {PEEK}," in *2014 Conference on Timely Results in Operating Systems ({TRIOS} 14)*, 2014.

[75] I. Manotas, L. Pollock, and J. Clause, "Seeds: A software engineer's energy-optimization decision support framework," in *Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 503–514.

[76] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, pp. 8–22, 2015.

[77] K. Meng and B. Norris, "Mira: A framework for static performance analysis," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2017, pp. 103–113.

[78] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, "Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance," in *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 2015, pp. 725–737.

[79] S. Wang, G. Zhong, and T. Mitra, "Cgpredict: Embedded gpu performance estimation from single-threaded applications," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 146, 2017.

[80] S. Lee, J. S. Meredith, and J. S. Vetter, "Compass: A framework for automated performance modeling and prediction," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 405–414.

[81] A. E. Helal, W.-c. Feng, C. Jung, and Y. Y. Hanafy, "Automatch: An automated framework for relative performance estimation and workload distribution on heterogeneous hpc systems," in *2017 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2017, pp. 32–42.

[82] V. J. Reddi, A. Settle, D. A. Connors, and R. S. Cohn, "Pin: a binary instrumentation tool for computer architecture research and education," in *Proceedings of the 2004 workshop on Computer architecture education: held in conjunction with the 31st International Symposium on Computer Architecture*, 2004, pp. 22–es.

[83] A. Afram and F. Janabi-Sharifi, "Theory and applications of hvac control systems–a review of model predictive control (mpc)," *Building and Environment*, vol. 72, pp. 343–355, 2014.

[84] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[85] L. Magni, G. De Nicolao, L. Magnani, and R. Scattolini, "A stabilizing model-based predictive control algorithm for nonlinear systems," *Automatica*, vol. 37, no. 9, pp. 1351–1362, 2001.

[86] A. I. Dounis and C. Caraiscos, "Advanced control systems engineering for energy and comfort management in a building environment—a review," *Renewable and Sustainable Energy Reviews*, vol. 13, no. 6-7, pp. 1246–1261, 2009.

[87] J. Singh, N. Singh, and J. Sharma, "Fuzzy modeling and control of hvac systems–a review," 2006.

[88] F. Calvino, M. La Gennusa, G. Rizzo, and G. Scaccianoce, "The control of indoor thermal comfort conditions: introducing a fuzzy adaptive controller," *Energy and buildings*, vol. 36, no. 2, pp. 97–102, 2004.

[89] D. Kolokotsa, G. Stavrakakis, K. Kalaitzakis, and D. Agoris, "Genetic algorithms optimized fuzzy controller for the indoor environmental management in buildings implemented using plc and local operating

networks," *Engineering Applications of Artificial Intelligence*, vol. 15, no. 5, pp. 417–428, 2002.

[90] P. P. Angelov and R. A. Buswell, "Automatic generation of fuzzy rule-based models from data by genetic algorithms," *Information Sciences*, vol. 150, no. 1-2, pp. 17–31, 2003.

[91] A. E. Ben-Nakhi and M. A. Mahmoud, "Energy conservation in buildings through efficient a/c control using neural networks," *Applied Energy*, vol. 73, no. 1, pp. 5–23, 2002.

[92] R. Kumar, R. Aggarwal, and J. Sharma, "Energy analysis of a building using artificial neural network: A review," *Energy and Buildings*, vol. 65, pp. 352–358, 2013.

[93] E. Barrett and S. Linder, "Autonomous hvac control, a reinforcement learning approach," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2015, pp. 3–19.

[94] T. Wei, Y. Wang, and Q. Zhu, "Deep reinforcement learning for building hvac control," in *Proceedings of the 54th annual design automation conference 2017*, 2017, pp. 1–6.

[95] K. Dalamagkidis, D. Kolokotsa, K. Kalaitzakis, and G. S. Stavrakakis, "Reinforcement learning for energy conservation and comfort in buildings," *Building and environment*, vol. 42, no. 7, pp. 2686–2698, 2007.

[96] J. H. Yoon, R. Baldick, and A. Novoselac, "Dynamic demand response controller based on real-time retail price for residential buildings," *IEEE Transactions on Smart Grid*, vol. 5, no. 1, pp. 121–129, 2014.

[97] D. Menniti, F. Costanzo, N. Scordino, and N. Sorrentino, "Purchase-bidding strategies of an energy coalition with demand-response capabilities," *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1241–1255, 2009.

[98] D. P. Chassin, J. Stoustrup, P. Agathoklis, and N. Djilali, "A new thermostat for real-time price demand response: Cost, comfort and energy impacts of discrete-time control without deadband," *Applied Energy*, vol. 155, pp. 816–825, 2015.

[99] S. Behboodi, D. P. Chassin, N. Djilali, and C. Crawford, "Transactive control of fast-acting demand response based on thermostatic loads in real-time retail electricity markets," *Applied Energy*, vol. 210, pp. 1310–1320, 2018.

[100] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *Acm sigplan notices*, vol. 40, no. 6, pp. 190–200, 2005.

[101] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *IEEE micro*, vol. 27, no. 3, pp. 63–72, 2007.

[102] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," *IBM Journal of research and development*, vol. 11, no. 6, pp. 601–617, 1967.

[103] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[104] G. Tesauro, "Td-gammon: A self-teaching backgammon program," in *Applications of Neural Networks*. Springer, 1995, pp. 267–285.

[105] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[106] M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *Frontiers in the Convergence of Bioscience and Information Technologies, 2007. FBIT 2007.* IEEE, 2007, pp. 645–650.

[107] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[108] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[109] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[110] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[111] S. G. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *IEEE Transactions on signal processing*, vol. 41, no. 12, pp. 3397–3415, 1993.

[112] S. Haykin, *Neural networks and learning machines, 3/E.* Pearson Education India, 2009.

[113] H. Akaike, "A new look at the statistical model identification," *IEEE transactions on automatic control*, vol. 19, no. 6, pp. 716–723, 1974.

[114] L. Myers and M. J. Sirois, "Spearman correlation coefficients, differences between," *Encyclopedia of statistical sciences*, vol. 12, 2004.

[115] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and multidisciplinary optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[116] ——, "The weighted sum method for multi-objective optimization: new insights," *Structural and multidisciplinary optimization*, vol. 41, no. 6, pp. 853–862, 2010.

[117] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms.* MIT press, 2022.

[118] I. J. Lustig, R. E. Marsten, and D. F. Shanno, "Interior point methods for linear programming: Computational state of the art," *ORSA Journal on Computing*, vol. 6, no. 1, pp. 1–14, 1994.

[119] M. Kojima, S. Mizuno, and A. Yoshise, "A primal-dual interior point algorithm for linear programming," in *Progress in mathematical programming.* Springer, 1989, pp. 29–47.

[120] R. Storn, "On the usage of differential evolution for function optimization," in *Proceedings of north american fuzzy information processing.* Ieee, 1996, pp. 519–523.

[121] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[122] D. Datta and J. R. Figueira, "A real–integer–discrete-coded differential evolution," *Applied Soft Computing*, vol. 13, no. 9, pp. 3884–3893, 2013.

[123] G. B. Dantzig, "Discrete-variable extremum problems," *Operations research*, vol. 5, no. 2, pp. 266–288, 1957.

[124] D. Pisinger, "A minimal algorithm for the multiple-choice knapsack problem," *European Journal of Operational Research*, vol. 83, no. 2, pp. 394–410, 1995.

[125] K. Dudziński and S. Walukiewicz, "Exact methods for the knapsack problem and its generalizations," *European Journal of Operational Research*, vol. 28, no. 1, pp. 3–21, 1987.

[126] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE international symposium on workload characterization (IISWC)*. Ieee, 2009, pp. 44–54.

[127] L.-N. Pouchet *et al.*, "Polybench: The polyhedral benchmark suite," *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, 2012.

[128] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices," in *Proceedings of the 2015 international workshop on internet of things towards applications*, 2015, pp. 7–12.

[129] X. Qi and C. Liu, "Enabling deep learning on iot edge: Approaches and evaluation," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 367–372.

[130] D. B. Crawley, L. K. Lawrie, F. C. Winkelmann, W. F. Buhl, Y. J. Huang, C. O. Pedersen, R. K. Strand, R. J. Liesen, D. E. Fisher, M. J. Witte *et al.*, "Energyplus: creating a new-generation building energy simulation program," *Energy and buildings*, vol. 33, no. 4, pp. 319–331, 2001.

[131] M. Wetter, "Co-simulation of building energy and control systems with the building controls virtual test bed," *Journal of Building Performance Simulation*, vol. 4, no. 3, pp. 185–203, 2011.

[132] ASHRAE, "Thermal environmental conditions for humman occupancy," ANSI/ASHRAE Standard 55-2013, 2013.

[133] R. De Dear and G. S. Brager, "Developing an adaptive model of thermal comfort and preference," 1998.

[134] A. AC08024865, *Ergonomics of the thermal environment-Analytical determination and interpretation of thermal comfort using calculation of the PMV and PPD indices and local thermal comfort criteria*. ISO, 2005.

[135] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, and P. Cunha, "Energy consumption and execution time estimation of embedded system applications," *Microprocessors and Microsystems*, vol. 35, no. 4, pp. 426–440, 2011.

[136] C. Marantos, N. Maidonis, and D. Soudris, "Designing application analysis tools for cross-device energy consumption estimation," in *2022 11th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2022, pp. 1–4.

[137] C. Marantos, K. Salapas, L. Papadopoulos, and D. Soudris, "A flexible tool for estimating applications performance and energy consumption through static analysis," *SN Computer Science*, vol. 2, no. 1, pp. 1–11, 2021.

[138] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," in *Mobile Computing*. Springer, 1994, pp. 449–471.

[139] F. Catthoor, S. Wuytack, G. de Greef, F. Banica, L. Nachtergaele, and A. Vandecappelle, *Custom memory management methodology: Exploration of memory organisation for embedded multimedia system design*. Springer Science & Business Media, 2013.

[140] D. A. Patterson and J. L. Hennessy, *Computer organization and design ARM edition: the hardware software interface*. Morgan kaufmann, 2016.

[141] A. S. Tanenbaum and A. S. Woodhull, "Operating systems, design and implementation, 1997."

[142] J. Bloch, *Effective java (the java series)*. Prentice Hall PTR, 2008.

[143] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. pearson education, 2005.

[144] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan, "Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model," in *International Conference on Compiler Construction*. Springer, 2008, pp. 132–146.

[145] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan, "A practical automatic polyhedral parallelizer and locality optimizer," in *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2008, pp. 101–113.

[146] J. M. P. Cardoso, J. G. de Figueired Coutinho, and P. C. Diniz, *Embedded computing for high performance: Efficient mapping of computations using customization, code transformations and compilation.* Morgan Kaufmann, 2017.

[147] J. Fowers, G. Brown, J. Wernsing, and G. Stitt, "A performance and energy comparison of convolution on gpus, fpgas, and multicore processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, pp. 1–21, 2013.

[148] C. Cummins, P. Petoumenos, Z. Wang, and H. Leather, "End-to-end deep learning of optimization heuristics," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2017, pp. 219–232.

[149] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[150] N. Ardalani, U. Thakker, A. Albarghouthi, and K. Sankaralingam, "A static analysis-based cross-architecture performance prediction using machine learning," *arXiv preprint arXiv:1906.07840*, 2019.

[151] I. Baldini, S. J. Fink, and E. Altman, "Predicting gpu performance from cpu runs using machine learning," in *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*. IEEE, 2014, pp. 254–261.

[152] P. A. Lachenbruch and M. R. Mickey, "Estimation of error rates in discriminant analysis," *Technometrics*, vol. 10, no. 1, pp. 1–11, 1968.

[153] C. Marantos, L. Papadopoulos, C. P. Lamprakos, K. Salapas, and D. Soudris, "Bringing energy efficiency closer to application developers: An extensible software analysis framework," *IEEE Transactions on Sustainable Computing*, 2022, *under minor revision.*

[154] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4738–4755, 2014.

[155] F. Samie, V. Tsoutsouras, D. Masouros, L. Bauer, D. Soudris, and J. Henkel, "Fast operation mode selection for highly efficient iot edge devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 3, pp. 572–584, 2019.

[156] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[157] A. Hall and U. Ramachandran, "An execution model for serverless functions at the edge," in *Proceedings of the International Conference on Internet of Things Design and Implementation*, 2019, pp. 225–236.

[158] T. Pfandzelter and D. Bermbach, "tinyfaas: A lightweight faas platform for edge environments," in *2020 IEEE International Conference on Fog Computing (ICFC)*. IEEE, 2020, pp. 17–24.

[159] T. Achilleas et al., "Faas and curious: Performance implications of serverless functions on edge computing platforms," in *International Conference on High Performance Computing*. Springer, 2021.

[160] A. Tzenetopoulos, C. Marantos, G. Gavrielides, S. Xydis, and D. Soudris, "Fade: Faas-inspired application decomposition and energy-aware function placement on the edge," in *Proceedings of the 24th International Workshop on Software and Compilers for Embedded Systems*, 2021, pp. 7–10.

[161] L. Papadopoulos, C. Marantos, G. Digkas, A. Ampatzoglou, A. Chatzigeorgiou, and D. Soudris, "Interrelations between software quality metrics, performance and energy consumption in embedded applications," in *Proceedings of the 21st International Workshop on software and compilers for embedded systems*, 2018, pp. 62–65.

[162] C. P. Lamprakos, C. Marantos, L. Papadopoulos, and D. Soudris, "The known unknowns: Discovering trade-offs between heterogeneous code changes," in *International Conference on Embedded Computer Systems*. Springer, 2022, pp. 342–353.

[163] C. P. Lamprakos, C. Marantos, M. Siavvas, L. Papadopoulos, A.-A. Tsintzira, A. Ampatzoglou, A. Chatzigeorgiou, D. Kehagias, and D. Soudris, "Translating quality-driven code change selection to an instance of multiple-criteria decision making," *Information and Software Technology*, vol. 145, p. 106851, 2022.

[164] M. Siavvas, C. Marantos, L. Papadopoulos, D. Kehagias, D. Soudris, and D. Tzovaras, "On the relationship between software security and energy consumption," in *15th China-Europe International Symposium on software engineering education*, 2019.

[165] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[166] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

[167] G. K. Gill and C. F. Kemerer, "Cyclomatic complexity density and software maintenance productivity," *IEEE transactions on software engineering*, vol. 17, no. 12, pp. 1284–1288, 1991.

[168] X. Li, P.-C. Shih, J. Overbey, C. Seals, and A. Lim, "Comparing programmer productivity in openacc and cuda: an empirical investigation," *International Journal of Computer Science, Engineering and Applications (IJCSEA)*, vol. 6, no. 5, pp. 1–15, 2016.

[169] S. Memeti, L. Li, S. Pllana, J. Kołodziej, and C. Kessler, "Benchmarking opencl, openacc, openmp, and cuda: programming productivity, performance, and energy consumption," in *Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, 2017, pp. 1–6.

[170] E. Shihab, Y. Kamei, B. Adams, and A. E. Hassan, "Is lines of code a good measure of effort in effort-aware models?" *Information and Software Technology*, vol. 55, no. 11, pp. 1981–1993, 2013.

[171] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[172] C. Marantos, L. Papadopoulos, A.-A. Tsintzira, A. Ampatzoglou, A. Chatzigeorgiou, and D. Soudris, "Decision support for gpu acceleration by predicting energy savings and programming effort," *Sustainable Computing: Informatics and Systems*, vol. 34, p. 100631, 2022.

[173] E. M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, M. Galster, and P. Avgeriou, "A mapping study on design-time quality attributes and metrics," *Journal of Systems and Software*, vol. 127, pp. 52–77, 2017.

[174] T. Amanatidis and A. Chatzigeorgiou, "Studying the evolution of php web applications," *Information and Software Technology*, vol. 72, pp. 48–67, 2016.

[175] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2009, pp. 367–377.

[176] J. Legaux, F. Loulergue, and S. Jubertie, "Development effort and performance trade-off in high-level parallel programming," in *2014 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 2014, pp. 162–169.

[177] C. Marantos, M. Siavvas, D. Tsoukalas, C. P. Lamprakos, L. Papadopoulos, P. Boryszko, K. Filus, J. Domańska, A. Ampatzoglou, A. Chatzigeorgiou *et al.*, "Sdk4ed: One-click platform for energy-aware, maintainable and dependable applications," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022, pp. 981–986.

[178] M. Siavvas, D. Tsoukalas, C. Marantos, A.-A. Tsintzira, M. Jankovic, D. Soudris, A. Chatzigeorgiou, and D. Kehagias, "The sdk4ed platform for embedded software quality improvement-preliminary overview," in *International Conference on Computational Science and Its Applications*. Springer, 2020, pp. 1035–1050.

[179] C. Marantos, A.-A. Tsintzira, L. Papadopoulos, A. Ampatzoglou, A. Chatzigeorgiou, and D. Soudris, "Technical debt management and energy consumption evaluation in implantable medical devices: The sdk4ed approach," in *International Conference on Embedded Computer Systems*. Springer, 2020, pp. 348–358.

[180] C. Strydis, R. M. Seepers, P. Peris-Lopez, D. Siskos, and I. Sourdis, "A system architecture, processor, and communication protocol for secure implants," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 4, pp. 1–23, 2013.

[181] "Wholesale electricity and natural gas market data," November 2015. [Online]. Available: http://www.eia.gov/electricity/wholesale/#history

[182] U. S. Department of Energy, "Energyplus energy simulation software," http://apps1.eere.energy.gov/buildings/energyplus/, 2015.

[183] C. Marantos, C. Lamprakos, K. Siozios, and D. Soudris, "Towards plug&play smart thermostats for building's heating/cooling control," in *IoT for Smart Grids*. Springer, 2019, pp. 183–207.

[184] C. Sagerschnig, D. Gyalistras, A. Seerig, S. Prívara, J. Cigler, and Z. Vana, "Co-simulation for building controller development: The case study of a modern office building," in *Proc. CISBAT*, 2011, pp. 14–16.

[185] G. Kontes, G. Giannakis, E. B. Kosmatopoulos, and D. Rovas, "Adaptive-fine tuning of building energy management systems using co-simulation," in *Control Applications (CCA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1664–1669.

[186] P. Danassis, K. Siozios, C. Korkas, D. Soudris, and E. Kosmatopoulos, "A low-complexity control mechanism targeting smart thermostats," *Energy and Buildings*, vol. 139, pp. 340–350, 2017.

[187] C. Marantos, K. Siozios, and D. Soudris, "A flexible decision-making mechanism targeting smart thermostats," *IEEE Embedded Systems Letters*, vol. 9, no. 4, pp. 105–108, 2017.

[188] B. Urban and K. Roth, "A data-driven framework for comparing residential thermostat energy performance," 2014.

[189] Y. Ma, J. Matuško, and F. Borrelli, "Stochastic model predictive control for building hvac systems: Complexity and conservatism," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 1, pp. 101–116, 2014.

[190] Z. Wu, Q.-S. Jia, and X. Guan, "Optimal control of multiroom hvac system: An event-based approach," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 2, pp. 662–669, 2015.

[191] S. A. Vaghefi, M. A. Jafari, J. Zhu, J. Brouwer, and Y. Lu, "A hybrid physics-based and data driven approach to optimal control of building cooling/heating systems," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 600–610, 2014.

[192] X. Zhang, W. Shi, X. Li, B. Yan, A. Malkawi, and N. Li, "Decentralized temperature control via hvac systems in energy efficient buildings: An approximate solution procedure," in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2016, pp. 936–940.

[193] J. Clarke, S. Conner, G. Fujii, V. Geros, G. Jóhannesson, C. Johnstone, S. Karatasou, J. Kim, M. Santamouris, and P. Strachan, "The role of simulation in support of internet-based energy services," *Energy and Buildings*, vol. 36, no. 8, pp. 837–846, 2004.

[194] W.-M. Lu and J. C. Doyle, "/spl hscr//sub/spl infin//control of nonlinear systems: a convex characterization," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1668–1675, 1995.

[195] T. Başar and P. Bernhard, *H-infinity optimal control and related minimax design problems: a dynamic game approach*. Springer Science & Business Media, 2008.

[196] S. Research, "Global smart thermostats market 2015-2019," pp. 1–66, Oct. 2015. [Online]. Available: http://www.sandlerresearch.org/global-smart-thermostats-market-2015-2019.html/

[197] W. Huang and H. Lam, "Using genetic algorithms to optimize controller parameters for hvac systems," *Energy and Buildings*, vol. 26, no. 3, pp. 277–282, 1997.

[198] Y. Yao, Z. Lian, Z. Hou, and X. Zhou, "Optimal operation of a large cooling system based on an empirical model," *Applied Thermal Engineering*, vol. 24, no. 16, pp. 2303–2321, 2004.

[199] J. Liang and R. Du, "Thermal comfort control based on neural network for hvac application," in *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*. IEEE, 2005, pp. 819–824.

[200] M. Harrold and D. Lush, "Automatic controls in building services," in *IEE Proceedings B (Electric Power Applications)*, vol. 135, no. 3. IET, 1988, pp. 105–133.

[201] G. Levermore, *Building energy management systems: An application to heating, natural ventilation, lighting and occupant satisfaction*. Routledge, 2013.

[202] R. Alcala, J. Casillas, O. Cordon, A. Gonzalez, and F. Herrera, "A genetic rule weighting and selection process for fuzzy control of heating, ventilating and air conditioning systems," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 3, pp. 279–296, 2005.

[203] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Mathematical programming*, vol. 89, no. 1, pp. 149–185, 2000.

[204] D. Bollegala, "Dynamic feature scaling for online learning of binary classifiers," *Knowledge-Based Systems*, vol. 129, pp. 97–105, 2017.

[205] T.-H. Teng, A.-H. Tan, and Y.-S. Tan, "Self-regulating action exploration in reinforcement learning," *Procedia Computer Science*, vol. 13, pp. 18–30, 2012.

[206] C. Gehring and D. Precup, "Smart exploration in reinforcement learning using absolute temporal difference errors," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1037–1044.

[207] C. Marantos, C. P. Lamprakos, V. Tsoutsouras, K. Siozios, and D. Soudris, "Towards plug&play smart thermostats inspired by reinforcement learning," in *Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications*, 2018, pp. 39–44.

[208] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *European Conference on Machine Learning*. Springer, 2005, pp. 317–328.

[209] D. Watari, I. Taniguchi, F. Catthoor, C. Marantos, K. Siozios, E. Shirazi, D. Soudris, and T. Onoye, "Thermal comfort aware online energy management framework for a smart residential building," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 535–538.

[210] N. Fumo and M. R. Biswas, "Regression analysis for prediction of residential energy consumption," *Renewable and sustainable energy reviews*, vol. 47, pp. 332–343, 2015.

[211] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis*. Wiley-Interscience; 3 edition, July 2003.

[212] R. D. Cook and S. Weisberg, *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[213] C. Marantos, K. Siozios, and D. Soudris, "Rapid prototyping of low-complexity orchestrator targeting cyberphysical systems: The smart-thermostat usecase," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 5, pp. 1831–1845, 2019.

[214] C. Cummins, Z. V. Fisches, T. Ben-Nun, T. Hoefler, M. F. O'Boyle, and H. Leather, "Programl: A graph-based program representation for data flow analysis and compiler optimizations," in *International Conference on Machine Learning*. PMLR, 2021, pp. 2244–2253.

[215] G. Führ, S. H. Hamurcu, D. Pala, T. Grass, R. Leupers, G. Ascheid, and J. F. Eusse, "Automatic energy-minimized hw/sw partitioning for

fpga-accelerated mpsocs," *IEEE Embedded Systems Letters*, vol. 11, no. 3, pp. 93–96, 2019.

[216] K. Siozios and S. Siskos, "A low-complexity framework for distributed energy market targeting smart-grid," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 878–883.

[217] C. D. Korkas, M. Terzopoulos, C. Tsaknakis, and E. B. Kosmatopoulos, "Nearly optimal demand side management for energy, thermal, ev and storage loads: An approximate dynamic programming approach for smarter buildings," *Energy and Buildings*, vol. 255, p. 111676, 2022.