Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων
Εργαστηριο Υπολογιστικων Συστηματων

Μελέτη επίδοσης εφαρμογών υψηλής έντασης σε ετερογενείς αρχιτεκτονικές

Διπλωματικη Εργασια

του

ΑΘΑΝΑΣΙΟΥ Γ. ΜΑΡΚΟΥ

Επιβλέπων: Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής

Αθήνα, Ιούνιος 2021

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομεας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

Μελέτη επίδοσης εφαρμογών υψηλής έντασης σε ετερογενείς αρχιτεκτονικές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΑΘΑΝΑΣΙΟΥ Γ. ΜΑΡΚΟΥ

Επιβλέπων: Γεώργιος Γκούμας
Αναπληρωτής Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 23η Ιουνίου 2021.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Διονύσιος Πνευματικάτος
Αναπληρωτής Καθηγητής	Καθηγητής	Καθηγητής

Αθήνα, Ιούνιος 2021

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομεας Τεχνολογίας Πληροφορικής και Υπολογιστών
Εργαστήριο Υπολογιστικών Συστημάτων

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Αθανάσιος Μάρκου

23 Ιουνίου 2021

Περίληψη

Κατά την ανάπτυξη των παράλληλων εφαρμογών υψηλής έντασης οι ερευνητές συναντούν ένα κρίσιμο δίλημμα. Το δίλημμα αυτό συνίσταται στο αν η εφαρμογή τους πρέπει να υλοποιηθεί για μια CPU αρχιτεκτονική ή για μια GPU αρχιτεκτονική. Μια κοινή στρατηγική για τους ερευνητές είναι να αναπτύξουν αρχικά την εφαρμογή τους στο προγραμματιστικό μοντέλο OpenMP για να έχουν μια ποσοτική εκτίμηση της επίδοσης του αλγορίθμου τους σε έναν πολυπύρηνο CPU.

Αν η επίδοση δεν είναι ικανοποιητική ο ερευνητής πρέπει να επανεξετάσει την προσέγγισή του. Η βελτιστοποίηση του OpenMP κώδικα για να αξιοποιήσει απτελεσματικά τα ιδιαίτερα αρχιτεκτονικά ενός δεδομένου CPU κόμβου ενδεχομένως να μην είναι η λύση που εξασφαλίζει την καλύτερη δυνατή επίδοση, είναι πιθανό μια άλλη αρχιτεκτονική να είναι καταλληλότερη για να εκμεταλλευτεί την κλιμακωσιμότητα της εφαρμογής και επομένως να αποφέρει καλύτερη επίδοση. Ίσως μια GPU είναι καταλληλότερη για την συγκεκριμένη εφαρμογή.

Είναι επίσης σημαντικό να λάβουμε υπόψιν την ποικιλομορφία του διαθέσιμου hardware. Χάρις τους παρόχους του υπολογιστικού νέφους (Amazon, Googlem Microsoft, κλπ) το σκηνικό στους κλάδους του HPC και του Datacenter Computing έχει αλλάξει ριζικά. Πλέον οι ερευνητές έχουν στην διάθεσή του μια μεγάλη ποικιλία τόσο από CPUs όσο και από GPUs. Έτσι το παραπάνω δίλημμα μπορεί να εκφραστεί και ως εξής: ″Δοθείσας μιας OpenMP υλοποίησης κάποιας εφαρμογής και δοθέντων των διαθέσιμων CPUs και GPUs, αξίζει να προχωρήσουμε σε μία μετάβαση από την OpenMP υλοποίηση σε μια CUDA υλοποίηση; Τι τάξη μεγέθους βελτίωση της απόδοσης πρέπει να αναμένουμε από μια τέτοια τροποποίηση;″

Σκοπός αυτής της διπλωματικής εργασίας είναι η δημιουργία ενός μοντέλου που παρέχει προβλέψεις για την απόδοση και για την κατανάλωση ισχύος μιας εφαρμογής. Πιο συγκεκριμένα, το μοντέλο μας προβλέπει για μια δεδομένη εφαρμογή (α) την τάξη μεγέθους της σχετικής επιτάχυνσης που προσφέρει η υλοποίηση σε μια συγκεκριμένη GPU ως προς μια αντίστοιχη υλοποίηση σε έναν δεδομένο CPU, (β) το ποια εκ των δύο αρχιτεκτονικών παρέχει την μικρότερη κατανάλωση ισχύος.

Λέξεις Κλειδιά

Ετερογενείς αρχιτεκτονικές, Εφαρμογές υψηλής έντασης, High Performance Computing, Cloud Computing, Performance Modeling, Power Efficiency

Abstract

While developing their parallel high-intensity applications researchers come across a crucial dilemma. The dilemma is whether their application should be targeted for a CPU-architecture or a GPU-architecture. A common strategy for researchers is to first implement an OpenMP version of their application to have a quantitative estimation of how well their algorithm leverages from the use of a multicore CPU-node.

If the performance attained is not sufficient one has to reconsider his/her approach. Tuning perfectly your OpenMP code to utilize special architectural features of the target CPU-node might not always be the solution that yields the optimal performance, maybe another architecture is more suitable for leveraging from the application's scalability and thus yielding the optimal performance. Maybe one GPU is more suitable for this applicaton.

An additional concern one has to take into consideration is the overall diversity of the available hardware. The modern landscape of HPC and Datacenter Computing has drastically changed thanks to cloud server vendors (Amazon, Google, Microsoft, etc), researchers are now able to develop and run their applications on a wide variety of different CPUs and different GPUs. So the above question/dilemma can be stated as: "Given an OpenMP implementation of an application and the available CPUs and GPUs, is it worth proceeding to an OpenMP-to-CUDA transformation? What order of magnitude of performance improvement should one expect from such a transformation?"

This diploma thesis aims to the development of a model that provides a performance and power prediction. More specifically, for an application our model predicts (a) the order of magnitude of the relative speedup that a specific GPU offers relatively to a specific CPU, (b) the most power efficient architecture between the two.

Keywords

Heterogeneous architectures, High-Intensity Applications, High Performance Computing, Cloud Computing, Performance Modelling, Power Efficiency

στους γονείς μου

Ευχαριστίες

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων. Θα ήθελα να ευχαριστήσω τον επίκουρο καθηγητή κ. Γεώργιο Ι. Γκούμα για την εμπιστοσύνη που μου έδειξε αναθέτοντάς μου αυτή τη διπλωματική εργασία και για τις γνώσεις που αποκόμισα από τα μαθήματά του. Ιδιαίτερα θα ήθελα να τον ευχαριστήσω για την αγάπη που μου μετέδωσε για το αντικείμενο των συστημάτων παράλληλης επεξεργασίας. Επίσης ευχαριστώ την μεταδιδακτορική ερευνήτρια Νικέλα Παπαδοπούλου για την καθοδήγησή της και την εξαιρετική συνεργασία που είχαμε. Οφείλω να ευχαριστήσω από καρδιάς όλα τα φιλικά πρόσωπα με τα οποία κάναμε τόσες και τόσες μακροσκελείς και ουσιώδεις συζητήσεις όλα αυτά τα χρόνια, που διαμορφώσαμε ο ένας τον άλλον και αναπτύξαμε ένα σπάνιο δέσιμο ζωής. Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου για την καθοδήγηση, την ηθική και υλική υποστήριξη που μου προσέφεραν όλα αυτά τα χρόνια.

Αθήνα, Μάιος 2021

Αθανάσιος Μάρκου

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο Εργαστήριο Υπολογιστικών Συστημάτων του Εθνικού Μετσόβιου Πολυτεχνείου. Ο μεγάλος όγκος της έρευνας πραγματοποιήθηκε λόγω της νόσου COVID-19 από απόσταση. Το εργαστήριο μας παρείχε το σύνολο τους multicore CPUs και τις GPUs που χρειαστήκαμε για τις μετρήσεις και τα πειράματά μας.

Κεφάλαιο 1

Εισαγωγή

1.1 Σύγχρονα Datacenters

Τις δύο τελευταίες δεκαετίες η εδραίωση του υπολογιστικού νέφους έχει αλλάξει ριζικά την ανάπτυξη των εφαρμογών υψηλής έντασης. Οι εφαρμογές υψηλής έντασης συνιστούν ένα ευρύ σύνολο εφαρμογών από διαφορετικούς επιστημονικούς κλάδους όπως η Βιοπληροφορική, η Τεχνητή Νοημοσύνη, η Φυσική, η Ανάλυση Δεδομένων κ.λπ. Η κοινή συνισταμένη όλων αυτών των ανομοιογενών μεταξύ τους εφαρμογών είναι ότι χαρακτηρίζονται από παραλληλία, δοθέντος δηλαδή ενός κατάλληλου παράλληλου υπολογιστικού συστήματος η εκτέλεσή τους μπορεί να επισπευθεί δραματικά.

Μέχρι πρότινος, η έλλειψη της απαραίτητης υπολογιστικής ισχύος αποτελούσε ένα από τα κύρια εμπόδια για τις ερευνητικές ομάδες. Πλέον όμως μπορούν μέσω του νέφους να χρησιμοποιήσουν διάφορους υπολογιστικούς κόμβους από μια ευρεία ποικιλία ετερογενών παράλληλων αρχιτεκτονικών. Πιο συγκεκριμένα, παρέχεται πρόσβαση σε πληθώρα από multicore-CPUs και GPUs. Αποδείχτηκε πώς ένα μεγάλο μέρος των εφαρμογών που εκτελούνται στα Datacenters επωφελούνται δραστικά και κλιμακώνουν μέσω εξειδικευμένων hardware accelerators. Η απήχηση και η χρησιμότητα του hardware acceleration ήταν τέτοια που ώθησε τους παρόχους στο να συμπεριλάμβουν στις υπηρεσίες τους την πρόσβαση και σε αρχιτεκτονικές όπως τα FPGAs και τα ASICs. Κάθε μία από αυτές τις αρχιτεκτονικές έχει σημαντικές διαφορές από τις υπόλοιπες, συγκεκριμένη στόχευση, διαφορετικά πλεονεκτήματα και γνωσιακές απαιτήσεις για την ανάπτυξη μιας εφαρμογής πάνω σε αυτή.

Από την πλευρά των παρόχων, η διάθεση και αποδοτική χρήση των διαθέσιμων υπολογιστικών πόρων ενός Datacenter είναι ένα από τα κύρια θέματα που τους απασχολούν. Για αυτό το λόγο πραγματοποιούνται έρευνες για την επίτευξη μεγαλύτερου ποσοστού αξιοποίησης των πόρων και επομένως περιορισμού του χρόνου που μένουν αδρανείς. Η αποδοτική αξιοποίηση των υπολογιστικών πόρων από τους χρήστες οδηγεί σε μειωμένους χρόνους εκτέλεσης των εφαρμογών και άρα δύναται να συνεισφέρει στη μείωση της ευρύτερης υπολογιστικής κίνησης στους κόμβους που διαθέτει το Datacenter. Όλα αυτά αναδεικνύουν την ανάγκη παροχής ενός συνόλου εργαλείων και μεθοδολογιών προς τις ερευνητικές ομάδες, προκειμένου να είναι σε θέση να παράξουν το καλύτερο δυνατό αποτέλεσμα δεδομένων των επιλογών που έχουν στη διάθεσή τους.

1.2 Παράλληλα προγραμματιστικά μοντέλα

Η ραγδαία στροφή προς τον προγραμματισμό παράλληλων υπολογιστικών συστημάτων οδήγησε στην ανάπτυξη ενός ευρύτατου συνόλου από προγραμματιστικά μοντέλα. Τα μοντέλα αυτά επιτρέπουν τον προγραμματισμό παράλληλων εφαρμογών. Δύο από τα πιο διαδεδομένα παράλληλα προγραμματιστικά μοντέλα είναι το OpenMP για τον προγραμματισμό multicore-CPUs και η CUDA για τον προγραμματισμό GPUs.

Το OpenMP αποτελεί ένα πρότυπο [1], είναι δηλαδή ένα σύνολο λεπτομερών κανόνων αναφορικά με τα compiler directives, τα library routines και τα environmental variables. Με βάση το OpenMP ο προγραμματιστής μπορεί να εκφράσει από ένα υψηλό επίπεδο αφαίρεσης τον παραλληλισμό της εφαρμογής του σε γλώσσες όπως η Fortran και η C/C++. Προκειμένου να παραλληλοποιησεί μία εφαρμογή, ο προγραμματιστής εντοπίζει τα τμήματα του κώδικα στα οποία οι εντολές μπορούν να διαμοιραστούν μεταξύ των διαθέσιμων πυρήνων του επεξεργαστή. Η πιο συνήθης περίπτωση είναι ο διαμοιρασμός του υπολογιστικού φόρτου σε τμήματα του κώδικα όπου συναντώνται loops. Παράλληλα το πρότυπο παρέχει ένα σύνολο από διαφορετικούς τρόπους να εφαρμόσει ο προγραμματιστής συγχρονισμό όπου αυτό κρίνεται αναγκαίο.

Η CUDA είναι το προγραμματιστικό μοντέλο και η κατεξοχήν πλατφόρμα παράλληλου προγραμματισμούπου των GPUs της Nvidia [2]. Παρέχει ένα προγραμματιστικό μοντέλο για την δημιουργία εφαρμογών που εκτελούνται και εκμεταλλεύονται τα ιδιαίτερα αρχιτεκτονικά χαρακτηριστικά που προσφέρει η GPU. Στις CUDA εφαρμογές το σειριακό μέρος εκτελείται στον επεξεργαστή του συστήματος ενώ το βαρύ υπολογιστικά κομμάτι που χαρακτηρίζεται από παραλληλία εκτελείται στα CUDA cores που προσφέρει η GPU. Η CUDA μπορεί να χρησιμοποιηθεί σε συνδυασμό με γλώσσες όπως οι C/C++, Fortran, Python και MATLAB.

1.3 Αντικείμενο της διπλωματικής

Στους τομείς του HPC και του Cloud Computing ένα από τα πρωταρχικά διλήμματα που συναντούν οι ερευνητικές ομάδες είναι το κατά πόσο οι παράλληλες εφαρμογές τους θα επωφελειθούν από την υλοποίηση τους σε μία αρχιτεκτονική έναντι μιας άλλης. Στην πλειοψηφία τους τα σύγχρονα Datacenters παρέχουν υπολογιστικούς κόμβους που αποτελούνται από δύο ετερογενείς αρχιτεκτονικές. Η πρώτη αρχιτεκτονική είναι οι πολυπύρηνοι επεξεργαστές (multicore-CPUs) ενώ η δεύτερη αρχιτεκτονική είναι οι κάρτες γραφικών (GPUs). Μάλιστα ο χρήστης μπορεί να επιλέξει από μία ποικιλία διαφορετικών multicore-CPUs και μια ποικιλία διαφορετικών GPUs.

Η επιλογή της κατάλληλης αρχιτεκτονικής για την ανάπτυξη μιας εφαρμογής επηρεάζει καθολικά την ερευνητική διαδικασία. Συγκεκριμένα καθορίζει:

- coding effort
  Προγραμματίζοντας μια εφαρμογή για μια δεδομένη αρχιτεκτονική απαιτείται βαθιά γνώση τόσο του προγραμματιστικού μοντέλου με το οποίο αυτή είναι συμβατή, όσο και των αρχιτεκτονικών/λειτουργικών της χαρακτηριστικών. Μια εξειδικευμένη αρχιτεκτονική για hardware acceleration όπως η GPU κατά κοινή ομολογία οδηγεί σε αυξημένες χρονικές

απαιτήσεις για την υλοποίηση μιας εφαρμογής σε αυτή.

- Άνω φράγμα στο speedup
  Σύμφωνα με το Roofline Model [3] οι CPUs και οι GPUs οδηγούν σε διαφορετικά άνω φράγματα του επιτεύξιμου speedup ανάλογα με το Operational Intensity έκαστης εφαρμογής.

- Οικονομικό κόστος
  Η πρόσβαση στους διαθέσιμους υπολογιστικούς κόμβους ενός υπολογιστικού νέφους έρχεται με διαφορετική χρέωση ανάλογα την αρχιτεκτονική και την γενιά που επιλέγει ο χρήστης.

- Κατανάλωση ισχύος
  Οι μεγάλες αρχιτεκτονικές διαφορές μεταξύ των CPUs και των GPUs μπορούν να οδηγήσουν σε αξιοσημείωτες διαφορές στην κατανάλωση ισχύος κατά την εκτέλεση μίας εφαρμογής. Σε συγκεκριμένες περιπτώσεις η υψηλή κατανάλωση ισχύος μπορεί να είναι απαγορευτική ως προς τους στόχους μίας εφαρμογής.

Κατά κύριο λόγο οι εφαρμογές που σχεδιάζονται και εκτελούνται στα Datacenter προέρχονται από διαφορετικούς επιστημονικούς κλάδους και επομένως έχουν σημαίνουσες διαφορές ως προς τη δομή τους και τις υπολογιστικές τους απαιτήσεις. Αν η εφαρμογή προς υλοποίηση δεν εμπίπτει σε μία κατηγορία προβλημάτων για τα οποία είναι γνωστό ότι θα κλιμακώσουν καλύτερα σε μια δεδομένη αρχιτεκτονική, τότε το ερώτημα ″Ποια αρχιτεκτονική είναι η κατάλληλη για την συγκεκριμένη υπό ανάπτυξη εφαρμογή;″ δεν επιδέχεται μια απλοϊκή και εμπειρική απάντηση.

Ως εκ τούτου, μια συνήθης τακτική που ακολουθείται είναι η αρχική ανάπτυξη της παράλληλης εφαρμογής με το προγραμματιστικό μοντέλο OpenMP. Εν αντιθέσει με το προγραμματιστικό μοντέλο CUDA, η απλότητα του OpenMP καθιστά τον προγραμματισμό μιας παράλληλης εφαρμογής προσιτό και επιτρέπει την σχετικά άμεση περάτωση σου. Έτσι, μπορούμε άμεσα να έχουμε μια ποσοτική εκτίμηση της απόδοσης ή της ενεργειακής συμπεριφοράς της εφαρμογής όταν αυτή εκτελείται σε έναν multicore-CPU. Αν τα μετρούμενα μεγέθη δεν είναι ικανοποιητικά τότε πρέπει να επανεξεταστεί η αρχική προσέγγιση.

Η διαδικασία βελτιστοποίησης της παράλληλης εφαρμογής με το OpenMP μπορεί να αποδειχτεί μη επαρκής για να αποφέρει τελικά το επιθυμητό αποτέλεσμα. Ενδεχομένως, το διαθέσιμο πλήθος των πυρήνων του multicore-CPU που έχουμε στη διάθεσή μας να μην είναι ικανό να εκμεταλλευτεί την παραλληλία της εφαρμογής. Είναι πιθανό ένας άλλος multicore-CPU με περισσότερους πυρήνες να είναι καταλληλότερος ή ακόμη είναι πιθανό μια άλλη αρχιτεκτονική με περισσότερους και απλούστερους ″πυρήνες″ να λειτουργήσει ευεργετικά και να επιτρέψει την κλιμάκωση αυτής της παράλληλης εφαρμογής. Εδώ διαφαίνεται λοιπόν ο σημαντικός ρόλος που δύναται να επιτελέσει κάποια ή κάποιες από τις διαθέσιμες GPUs.

Οι δύο αυτές ετερογενείς αρχιτεκτονικές εξελίσσονται με την πάροδο του χρόνου. Υπάρχουν πολλές διαφορετικές γενιές και εκδόσεις multicore-CPUs, αντίστοιχα το ίδιο συμβαίνει και για τις GPUs. Και στις δύο οικογένειες αρχιτεκτονικών, από γενιά σε γενιά πραγματοποιούνται σημαντικές αρχιτεκτονικές αλλαγές κατά την σχεδίαση, αναθεωρούνται και βελτιώνονται

τρέχουσες υλοποιήσεις ενώ επίσης προστίθενται νέα χαρακτηριστικά. Μέσα σε όλο αυτό το γεμάτο πληροφορία σκηνικό ερχόμαστε να αναζητήσουμε απαντήσεις.

Στα πλαίσια της παρούσας διπλωματικής πραγματοποιήθηκε μια ενδελεχής μελέτη της επίδοσης ενός ευρύτατου συνόλου εφαρμογών που εκτελέσαμε σε ένα πλήθος διαφορετικών γενιών των ετερογενών αρχιτεκτονικών multicore-CPUs και GPUs. Με βάση αυτή τη μελέτη, διαμορφώσαμε ένα μοντέλο που προβλέπει (α) την τάξη μεγέθους της σχετική επιτάχυνση που θα προσφέρει μια δεδομένη GPU έναντι ενός δεδομένου multicore-CPU, (β) ποια από τις συγκρινόμενες αρχιτεκτονικές θα έχει την μικρότερη ενεργειακή κατανάλωση. Θεωρούμε ότι η προσέγγισή μας προσφέρει στους ερευνητές ένα ποσοτικό κριτήριο για το πόσο θα επωφελειθούν από τη μετάβαση της εφαρμογής τους από μια OpenMP υλοποίηση σε μια CUDA υλοποίηση. Με το κριτήριο που τους παρέχουμε έχουν την δυνατότητα να αξιολογήσουν αν η προαναφερθείσα μετάβαση θα αποδώσει καρπούς.

1.4 Οργάνωση του τόμου

Η εργασία αυτή είναι οργανωμένη ως ακολούθως:

Στο πρώτο μέρος παρουσιάζουμε λεπτομερώς το απαραίτητο θεωρητικό υπόβαθρο για την εκπόνηση της διπλωματικής εργασίας. Εξετάζουμε κριτικά προηγούμενες προσεγγίσεις επί του θέματος. Παρουσιάζουμε την σουίτα των εφαρμογών που χρησιμοποιήσαμε για να εξετάσουμε το software diversity των εφαρμογών που υποβάλλονται στα σύγχρονα υπολογιστικά συστήματα. Εν συνεχεία, εκθέτουμε τα εργαλεία που χρησιμοποιήσαμε για να λάβουμε μετρήσεις σε CPU και GPU. Επίσης, παρουσιάζουμε τις μεθόδους μηχανικής μάθησης που χρησιμοποιήσαμε για την υλοποίηση των μοντέλων μας.

Στο δεύτερο μέρος εξετάζουμε την πειραματική υποδομή που χρησιμοποιήσαμε και επισημαίνουμε κάποιες κύριες αρχιτεκτονικές διαφορές. Έπειτα, παρουσιάζουμε τις μετρικές που χρησιμοποιούμε για την εκπαίδευση των μοντέλων απόδοσης και κατανάλωσης ισχύος. Στην συνέχεια, παρουσιάζουμε την ακριβή μεθοδολογία που ακολουθούμε και τελικά παραθέτουμε τα αποτελέσματα που πήραμε από την εφαρμογή της.

Στο τρίτο και τελευταίο μέρος παραθέτουμε τα συμπεράσματα στα οποία οδηγηθήκαμε από την μεθοδολογία που ακολουθήσαμε. Επίσης παρουσιάζουμε κάποιες μελλοντικές επεκτάσεις για τα μοντέλα που προτείνουμε.

Μέρος I

Θεωρητικό Μέρος

Κεφάλαιο 2

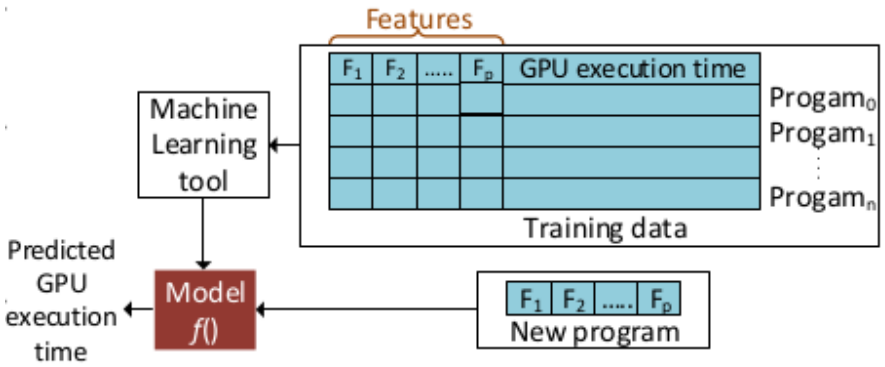
Υπάρχουσες Λύσεις

Στο κεφάλαιο αυτό παρουσιάζονται συνοπτικά δύο υπάρχουσες λύσεις. Οι μελέτες αυτές επιδιώκουν να προβλέψουν την επίδοση μιας εφαρμογής κατά την εκτέλεσή της σε μια GPU αρχιτεκτονική. Η βάση αυτής της πρόβλεψης έγκειται και στις δύο περιπτώσεις στην υπάρχουσα CPU έκδοση του κώδικα της εφαρμογής. Και στις δύο περιπτώσεις παρουσιάζονται πολύ ενδιαφέρουσες ιδέες και λεπτομερείς αναλύσεις από τις οποίες αντλήσαμε αρκετές ιδέες για την εκπόνηση της παρούσας διπλωματικής εργασίας. Μολονότι ενδελεχείς οι δύο αυτές έρευνες παρουσιάζουν κάποια μειονεκτήματα τα οποία φιλοδοξούμε να ξεπεράσουμε.

Η πρώτη δουλειά με τίτλο "**Should I Use a GPU? Predicting GPU Performance from CPU Runs**" προτείνει μια μεθοδολογία πρόβλεψης του speedup που δύναται να επιτύχει μια GPU υλοποίηση της εφαρμογής του ενδιαφέροντός μας δεδομένης μιας υπάρχουσας CPU υλοποίησης [4]. Συγκεκριμένα γίνεται χρήση δύο supervised μεθόδων για την δημιουργία του κατάλληλου μοντέλου πρόβλεψης. Οι μέθοδοι που χρησιμοποιούνται είναι οι Nearest neighbor with generallized exemplars (NNGE) και Support Vector Machine (SVM). Για να εξαλειφθεί το variation εφαρμόστηκε η μέθοδος Cross-Validation. Τα δεδομένα με τα οποία εκπαιδεύεται το μοντέλο πρόβλεψης συλλέγονται δυναμικά με χρήση του Pin στον κώδικα της εφαρμογής που προορίζεται για εκτέλεση σε CPU, εν συνεχεία στο μοντέλο παρέχονται τα speedups που επιτυγχάνονται τόσο για την εκτέλεση σε CPU όσο και σε GPU έκαστης εφαρμογής. Οι εφαρμογές που χρησιμοποιούνται για την εκπαίδευση του μοντέλου προέρχονται από benchmarks των Parboil 2.0 και Rodinia 2.2 σουιτών. Εφόσον το μοντέλο έχει εκπαιδευτεί, ο χρήστης είναι σε θέση να πραγματοποιήσει τις απαραίτητες δυναμικές μετρήσεις στον CPU κώδικά του, να παραθέσει αυτές τις μετρήσεις στο μοντέλο και να αποφανθεί για το όφελος που θα αποκομίσει από μια μετάβαση σε μια υλοποίηση για GPU.

Το κύριο μειονέκτημα της παραπάνω μεθοδολογίας για την εκπαίδευση του μοντέλου χρησιμοποιεί microarchitecture-dependent μετρικές. Κατά την εκπαίδευση του μοντέλου, η δυναμική λήψη των απαραίτητων μετρικών οδηγεί σε μετρήσεις η οποίες εξαρτώνται σε μεγάλο βαθμό από την μικροαρχιτεκτονική του συστήματος. Μια σημαντική παράληψη είναι ότι δεν ποσοτικοποιούνται οι κύριες αρχιτεκτονικές διαφορές μεταξύ των συγκρινόμενων αρχιτεκτονικών. Έτσι, όταν στα πλαίσια μιας συγκεκριμένης εφαρμογής η σύγκριση θα περιλαμβάνει ένα σύνολο από διαφορετικές CPUs και ένα σύνολο από διαφορετικές GPUs, δεν μπορεί να υπάρξει εγγύηση για την ακρίβεια της πρόβλεψης.

Η δεύτερη μελέτη με τίτλο **"Cross-Architecture Performance Prediction (XAPP) Using CPU Code to Predict GPU Performance"** [5] είναι η πιο πλήρης μεταξύ των δύο. Καλύπτει περισσότερα σενάρια πιθανής εφαρμογής της μεθοδολογίας που προτείνει, ενώ μάλιστα δεν έχει στο ακέραιο τα μειονεκτήματα της προηγούμενης δουλειάς που παρουσιάσαμε. Οι δύο μεγάλες συνεισφορές των ερευνητών μέσω αυτής της μελέτης είναι (α) ότι διακρίνει την απόδοση της εφαρμογής σε GPU ως συνάρτηση αφενός των ιδιοτήτων της εφαρμογής και αφετέρου των GPU hardware χαρακτηριστικών της GPU που εξετάζουμε με τη μορφή coefficients, (β) ένα σύνολο τεχνικών για να καταδειχθεί η αποτελεσματικότητα του εργαλείου που προτείνουν. Εν αντιθέσει με την προηγούμενη δουλειά τα μετρούμενα χαρακτηριστικά έκαστης εφαρμογής είναι microarchitecture-independent. Στην παρακάτω εικόνα βλέπουμε την κεντρική ιδέα η οποία είναι παρεμφερής με την τακτική που ακολουθούμε για την διπλωματική μας εργασία.



Σχήμα 2.1: *Μοντέλο του XAPP*

Για το μοντέλο του XAPP ακουλουθείται μια τεχνική δύο επιπέδων. Στο πρώτο επίπεδο χρησιμοποιείται regression ενώ στο δεύτερο επίπεδο διαμορφώνεται η πρόβλεψη μέσω bootstrap aggregating (bagging). Για την συλλογή των απαραίτητων δεδομένων αξιοποιούνται benchmarks από τις σουίτες Lonestar, Parsec, Parboil, Rodinia, NAS κλπ.

Η δουλειά αυτή παραλείπει να παρουσιάσει κάποια λύση για την μείωση της καθυστέρησης που εισάγει στην διαμόρφωση του μοντέλου η συλλογή των microarchitecture-independent ιδιοτήτων της εφαρμογής μέσω του εργαλείου MICA. Επιπρόσθετα, με τον τρόπο που διαμορφώνεται η πρόβλεψη δεν μπορεί να ελεγχθεί με αναλυτικό ή έστω διαισθητικό τρόπο, το ποιος είναι ο σημαίνων παράγοντας για μια παρατηρούμενη συμπεριφορά.

Παρατήρηση: κοινό μειονέκτημα των παραπάνω είναι η σιωπηλή υπόθεση ότι οι εφαρμογές που χρησιμοποιήθηκαν για την εκπαίδευση των μοντέλων είναι βελτιστοποιημένες. Επομένως η ποσοτική εκτίμηση του GPU-speedup για μια δεδομένη εφαρμογή είναι επιρρεπής σε σφάλματα. Επίσης, δεν παρέχουν μια εκτίμηση του ποια αρχιτεκτονική προσφέρει την χαμηλότερη κατανάλωση για την υπό εξέταση εφαρμογή.

Κεφάλαιο 3

Μελέτες και Εργαλεία στα οποία στηριχτήκαμε

Στο κεφάλαιο αυτό περιγράφεται το σύνολο των ερευνών και εργαλείων στο οποίο στηριχτήκαμε για να εκπονήσουμε την διπλωματική. Αρχικά παρουσιάζεται η σουίτα Rodinia από την οποία αντλούμε τις εφαρμογές πάνω στις οποίες στηρίζουμε την έρευνά μας συλλέγοντας τα δεδομένα μας. Εν συνεχεία, παρουσιάζεται το εργαλείο MICA και πώς αυτό συνεισφέρει στην έρευνά μας. Επίσης παρουσιάζουμε τα εργαλεία μέσω των οποίων μπορεί να μετρηθεί η κατανάλωση ισχύος στο σύνολο των ετερογενών αρχιτεκτονικών που μελετάμε. Στο επόμενο κεφάλαιο παρουσιάζουμε τα supervised μοντέλα μηχανικής μάθησης στα οποία στηριχτήκαμε για να παράξουμε τα δικά μας μοντέλα πρόβλεψης.

3.1 Rodinia Benchmark Suite

Η σουίτα των Rodinia benchmarks είναι ένα σύνολο από εφαρμογές ειδικά υλοποιημένες για ένα σύνολο από διαφορετικούς accelerators που συναντώνται στα σύγχρονα ετερογενή υπολογιστικά συστήματα [6, 7]. Στοχοθεσία της σουίτας είναι να αποτελέσει ένα χρήσιμο εργαλείο για την μελέτη των σύγχρονων υπολογιστικών συστημάτων που απαρτίζονται από πολλές ετερογενείς αρχιτεκτονικές. Οι εφαρμογές έχουν υλοποιηθεί αφενός για multicore CPUs με το προγραμματιστικό μοντέλο OpenMP και αφετέρου για GPUs με το προγραμματιστικό μοντέλο CUDA. Με την εξέλιξη της σουίτας οι περισσότερες εφαρμογές έγιναν διαθέσιμες και σε OpenCL, δεδομένου όμως ότι η Nvidia σταμάτησε την υποστήριξη στο προγραμματιστικό μοντέλο OpenCL επιλέξαμε να εστιάσουμε στις υλοποιήσεις OpenMP και CUDA.

Οι εφαρμογές που παρέχονται στην σουίτα αναπαριστούν και διαφορετικούς τύπους συμπεριφοράς σύμφωνα με τα Berkeley dwarves [8]. Με τον όρο dwarf εννοείται μια αλγοριθμική μέθοδος που αποτυπώνει ένα μοτίβο υπολογισμών και επικοινωνίας. Τα Berkeley dwarves στοχεύουν στην αξιολόγηση τόσο των παράλληλων προγραμματιστικών μοντέλων όσο και των παράλληλων αρχιτεκτονικών. Το γεγονός ότι στην παρούσα σουίτα συναντάμε εφαρμογές από πολύ διαφορετικούς επιστημονικούς κλάδους, σημαίνει ότι έχουμε στην διάθεσή μας πολλές ανομοιογενείς εφαρμογές με διαφορετικές υπολογιστικές απαιτήσεις. Όλα αυτά μας επιτρέπουν μέσω του software diversity των εφαρμογών της σουίτας να μελετήσουμε το hardware diversity των διαθέσιμων υπολογιστικών πόρων, των διαθέσιμων ετερογενών αρχιτεκτονικών.

Μέσω των εφαρμογών εκφράζονται πολλοί διαφορετικοί τύποι παραλληλισμού, διαφορετι-

κά μοτίβα προσβάσεων στη μνήμη καθώς επίσης και περιπτώσεις διαμοιρασμού των δεδομένων. Τα dwarves που καλύπτονται είναι Structured Grid, Unstructured Grid, Dynamic Programming, Dense Linear Algebra, Graph Traversal κλπ. Συνολικά οι εφαρμογές, τα αντίστοιχα dwarves και οι επιστημονικοί κλάδοι από τους οποίους αυτές προέρχονται φαίνονται στον Πίνακα 3.1. Η σουίτα αυτή ενδείκνυται για τις εξής αναλύσεις:

- Diversity Analysis
  Δυνατότητα να εξετάσουμε αν τα benchmarks που παρέχει η σουίτα είναι επαρκή για να καλύψουν τις ερευνητικές μας ανάγκες.

- Parallelization and Speedup
  Οι εφαρμογές της Rodinia σουίτας έχουν παραλληλοποιηθεί και βελτιστοποιηθεί με ποικίλους τρόπους για να προσφέρουν μια ικανοποιητική απόσοδη. Επομένως μπορούμε να έχουμε μια ποσοτική εκτίμηση του πόσο καλά αποδίδουν σε μια δεδομένη αρχιτεκτονική.

- Computation vs Communication
  Πολλοί accelerators όπως οι GPUs χρησιμοποιούν ένα co-processor model όπου το υπολογιστικό φορτίο των εφαρμογών ανατίθεται στον accelerator από τον CPU του συστήματος. Πολλές φορές το χρονικό κόστος που επιφέρει η επικοινωνία μεταξύ GPUs και CPUs είναι καταλυτικής σημασίας για την τελική απόδοση μιας εφαρμογής.

- Synchronization
  Ένα άλλο μεγάλο εμπόδιο για την επίτευξη υψηλής απόδοσης ενδέχεται να είναι το χρονικό κόστος που οφείλεται στον συγχρονισμό.

- Power Consumption
  Ένα πιθανό πλεονέκτημα που προσφέρουν οι accelerators είναι η μειωμένη κατανάλωση ισχύος για μια δεδομένη εφαρμογή, έναντι της κατανάλωσης ισχύος που επιφέρει μια αντίστοιχη CPU υλοποίηση.

Τέλος, αξίζει να επισημάνουμε ότι με τις εφαρμογές της σουίτας παρέχονται αρχεία που λειτουργούν ως δεδομένα εισόδου για τις εφαρμογές. Επίσης, σε πολλές περιπτώσεις παρέχονται και ειδικά scripts για την δημιουργία νέων δεδομένων εισόδου. Όλα αυτά καθιστούν στην σουίτα Rodinia ιδανική για τη μελέτη μας.

Πίνακας 3.1: *Rodinia Benchmarks*

Applications	Dwarves	Domains	Parallel Model
Leukocyte	Structured Grid	Medical Imaging	CUDA, OMP, OCL
Heart Wall	Structured Grid	Medical Imaging	CUDA, OMP, OCL
MUMmerGPU	Graph Traversal	Bioinformatics	CUDA, OMP
CFD Solver	Unstructured Grid	Fluid Dynamics	CUDA, OMP, OCL
LU Decomposition	Dense Linear Algebra	Linear Algebra	CUDA, OMP, OCL
Hotspot	Structured Grid	Physics Simulation	CUDA, OMP, OCL
Back Propagation	Unstructured Grid	Pattern Recognition	CUDA, OMP, OCL
Needleman-Wunsch	Dynamic Programming	Bioinformatics	CUDA, OMP, OCL
Kmeans	Dense Linear Algebra	Data Mining	CUDA, OMP, OCL
Breadth-First Search	Graph Traversal	Graph Algorithms	CUDA, OMP, OCL
SRAD	Structured Grid	Image Processing	CUDA, OMP, OCL
Streamcluster	Dense Linear Algebra	Data Mining	CUDA, OMP, OCL
Particle Filter	Structured Grid	Medical Imaging	CUDA, OMP, OCL
PathFinder	Dynamic Programming	Grid Traversal	CUDA, OMP, OCL
Gaussian Elimination	Dense Linear Algebra	Linear Algebra	CUDA, OCL
k-Nearest Neighbors	Dense Linear Algebra	Linear Algebra	CUDA, OMP, OCL
LavaMD2	N-Body	Molecular Dynamics	CUDA, OMP, OCL
Myocyte	Structured Grid	Biological Simulation	CUDA, OMP, OCL
B+Tree	Graph Traversal	Search	CUDA, OMP, OCL
GPUDWT	Spectral Method	Image/Video Compression	CUDA, OCL
Hybrid Sort	Sorting	Sorting Algorithms	CUDA, OCL
Hotspot3D	Structured Grid	Physics Simulation	CUDA, OMP, OCL
Huffman	Finite State Machine	Lossless data compression	CUDA, OCL

3.2 MICA: Microarchitecture-Independent Characterization of Applications

Το MICA αποτελεί ένα Pin εργαλείο που επιτρέπει στον χρήστη να συλλέξει ένα σύνολο από μετρικές που αφορούν την εφαρμογή του και εκφράζουν την συμπεριφορά της [9], [10], [11]. Αναπτύχθηκε μετά την δημοσίευση του ″Microarchitecture-Independent Workload Characterization″ όπου οι συγγραφείς αναζητούν τις κατάλληλες μετρικές που είναι σε θέση να αποδώσουν πιστά την συμπεριφορά ενός προγράμματος [12]. Κεντρικό σημείο στην συλλογιστική τους είναι η συλλογή δεδομένων για τις μετρικές να γίνει με ένα τρόπο που η μικροαρχιτεκτονική του συστήματος δεν επηρεάζει τα μετρούμενα μεγέθη, έχουμε δηλαδή ένα σύνολο από microarchitecture-independent μεγέθη. Αξίζει να σημειωθεί ότι οι μετρικές παραμένουν ISA-dependent(εξαρτώνται δηλαδή από την Αρχιτεκτονική του συνόλου εντολών), παρόλα αυτά όμως είναι σε θέση να αποδώσουν με έναν αφαιρετικό τρόπο τα χαρακτηριστικά αυτά που καθορίζουν την συμπεριφορά ενός προγράμματος. Πιο σύγχρονες μελέτες ξεπερνάνε αυτήν την εξάρτηση από την αρχιτεκτονική του συνόλου εντολών, επειδή όμως στηρίζονται στο προγραμματιστικό μοντέλο OpenCL δεν θα τις εξετάσουμε [13].

Για να γίνει πιο κατανοητή η σημασία που έχουν τα microarhitecture-independent μεγέθη αρκεί να σκεφτούμε το εξής παράδειγμα:
Έστω ότι έχουμε μια εφαρμογή Α για την οποία επιθυμούμε να πάρουμε τις επιθυμητές μετρήσεις. Επίσης έχουμε στην διάθεσή μας ένα πλήθος από διαφορετικούς CPUs που όμως υπάγονται στο ίδιο ISA. Αυτό που αναμένουμε είναι ότι αν η μέτρηση της εφαρμογής πραγματοποιηθεί σε οποιαδήποτε εκ των διαθέσιμων CPUs, τότε τα αποτελέσματα που θα πάρουμε θα είναι τα ίδια.
Αυτό ασφαλώς αναδεικνύει την ευελιξία αλλά και την χρησιμότητα του εργαλείου για μια έρευνα σαν τη δική μας.

Στο MICA γίνεται διάκριση σε επτά διαφορετικές οικογένειες microarhitecture-independent μετρικών κάθε μία εκ των οποίων αποσκοπεί και στην ανάδειξη διαφορετικών χαρακτηριστικών του κώδικα μας. Ο διαμοιρασμός αυτός των μετρικών επιτρέπει να γίνει αντιστοίχιση μιας παρατηρούμενης συμπεριφοράς με κάποια συγκεκριμένη αρχιτεκτονική διαφορά του ετερογενούς συστήματος που εξετάζουμε. Επομένως είναι πλέον εφικτό να καταλάβουμε και σε κάποιες περιπτώσεις να προβλέψουμε το ποια αρχιτεκτονική είναι ιδανική για την κλιμάκωση κάποιας συγκεκριμένης εφαρμογής. Στο πίνακα 3.2 παρουσιάζονται συγκεντρωμένες οι οικογένειες μετρήσεων.

Πίνακας 3.2: *Οικογένειες microarchitecture-independent μετρικών του MICA*

Characteristic Category	Description
Instruction mix	Ποσοστά των εντολών loads, stores, branches, arithmetic, multiplies, floating-point.
Instruction-level parallelism	Το IPC που επιτυγχάνεται για διαφορετικά window sizes με την υπόθεση ιδανικών caches και ιδανικού branch predictor.
Register traffic	Μέσο πλήθος καταχωρητών ως input operands ανά εντολή, μέσος αριθμός register read ανά register write, κατανομή των dependency distances για τους καταχωρητές όπου ως dependency distance νοείται το πλήθος των εντολών μεταξύ παραγωγής και κατανάλωσης μιας δεδομένης τιμής ενός καταχωρητή.
Working-set size	Πλήθος από ξεχωριστά 32-byte blocks και 4-Kbyte σελίδες μνήμης που χρησιμοποιούνται για τις εντολές αλλά και τα δεδομένα.
Data stream strides	Κατανομές για global και local strides. Ως strides νοούνται οι διαφορά διευθύνσεων μεταξύ δύο διαδοχικών προσβάσεων.
Branch predictability	Ένα σύνολο μετρικών που εκφράζουν τις ακρίβειες που πετυχαίνουν διαφορετικές υλοποιήσεις από branch predictors.
Memory stack distance	Ένα σύνολο μετρικών για την μελέτη της εικόνας της cache κατά την εκτέλεση μιας εφαρμογής.

Ένα από τα κύρια προτερήματα του MICA είναι ότι παρέχει μεγάλη ευελιξία στους τύπους ανάλυσης που επιθυμούμε να πραγματοποιήσουμε για μια δεδομένη εφαρμογή. Ανάλογα με τις

ανάγκες μας μπορούμε να επιλέξουμε έναν, περισσότερους ή όλους τους διαθέσιμους τύπους ανάλυσης. Μάλιστα για κάθε έναν εκ των τύπων ανάλυσης μπορούμε να επιλέξουμε να πραγματοποιήσουμε τη μέτρηση ενός υποσυνόλου των διαθέσιμων μετρικών που πραγματεύεται το εργαλείο. Ασφαλώς, έχουμε τη δυνατότητα να ρυθμίσουμε συγκεκριμένες παραμέτρους της δειγματοληψίας που πραγματοποιεί το εργαλείο. Αξίζει να σημειωθεί ότι πέραν των παραπάνω εφτά οικογενειών μέσω του MICA μπορούμε να μετρήσουμε το πλήθος εντολών που δεν αντιστοιχίζονται σε αυτές τις οικογένειες. Όλα αυτά συνιστούν ένα πολύ εύχρηστο εργαλείο από το οποίο μπορούμε να αντλήσουμε τις πληροφορίες που αποδίδουν καλύτερα την συμπεριφορά της υπό εξέταση εφαρμογής.

Πίνακας 3.3: *microarchitecture-independent μετρικές του MICA*

Characteristic Category	Metrics
Instruction mix	mem-read, mem-write, control-flow, arithmetic, floating-point, stack, shift, string, sse, system, nop, other
Instruction-level parallelism	ILP32, ILP64, ILP128, ILP256
Register traffic	reg_age_cnt1, reg_age_cnt2, reg_age_cnt8, ... , reg_age_cnt64, total_reg_age, total_num_ops, instr_reg_cnt
Working-set size	DataFootprint64, DataFootprint4K, InstrFootprint64, InstrFootprint4K
Data stream strides	mem_read_local_stride0 , ... , mem_read_local_stride256k, mem_read_global_stride0 , ... , mem_read_global_stride256k mem_write_local_stride0 , ... , mem_write_local_stride256k mem_write_global_stride0, ..., mem_write_global_stride256k
Branch predictability	GAg, PAg, GAs, PAs predictors' accuracies for 4bit, 8bit and 12bit implementations
Memory stack distance	cold-references, memReuseDist0-2, memReuseDist2-4, memReuseDist4-8, ... , memReuseDist256k-∞

Στις πρώτες εκδόσεις του MICA η μέτρηση μιας εφαρμογής επιβάλλεται να πραγματοποιηθεί για την μονονηματική έκδοση της συγκεκριμένης εφαρμογής. Αυτό για παράλληλες εφαρμογές αυτομάτως μεταφράζεται σε σημαντικό χρόνο ολοκλήρωσης της εφαρμογής. Σε νεότερες εκδόσεις δόθηκε η δυνατότητα μέτρησης πολυνηματικών εφαρμογών.

Ένα κρίσιμο σημείο που δεν αναφέρεται επαρκώς στην βιβλιογραφία τόσο του MICA όσο και των ερευνών που αξιοποιούν το MICA είναι το overhead που επιφέρει. Ανάλογα με το πόσο εκτενής είναι η ανάλυση ή οι αναλύσεις που επιλέγουμε να πραγματοποιήσουμε, και ανάλογα του χρόνου εκτέλεσης της εφαρμογής στην οποία εφαρμόζουμε το εργαλείο έχουμε και μια συγκεκριμένη χρονική καθυστέρηση. Εμπειρικά αυτό που διαπιστώσαμε μέσα από τα δικά μας πειράματα είναι ότι η καθυστέρηση που επιφέρει το MICA κυμένεται από έναν συντελεστή x10 ο οποίος όμως για συγκεκριμένες εφαρμογές ξεπερνά το x2000. Επειδή ασφαλώς η καθυστέρηση αυτή είναι απαγορευτική για την διεκπεραίωση συγκεκριμένων πειραμάτων, είναι λογικό να ωθούμε προς πρόωρο τερματισμό τις αντίστοιχες μετρούμενες εφαρμογές. Η ευελιξία που προσφέρεται ωστόσο από το MICA, αναφορικά με τον τρόπο δειγματοληψίας, μας επιτρέπει να περισώσουμε τα μετρούμενα μεγέθη που έχουν προκύψει ως την στιγμή του τερματισμού. Αυτό ασφαλώς εισάγει σφάλμα στις μετρήσεις όμως δίνει μια καλή εκτίμηση των

κεντρικών χαρακτηριστικών της εφαρμογής.

Προκειμένου να έχουμε μια πιο πλήρη εικόνα του MICA αξίζει να σκιαγραφήσουμε τι είναι το εργαλείο Pin στο οποίο στηρίζεται. Το Pin είναι έναν binary instrumentation framework που επιτρέπει την δημιουργία ενός πλήθους από εργαλεία για δυναμική ανάλυση των προγραμμάτων που εκτελούνται στον χώρο χρήστη. Τα εργαλεία αυτά αποκαλούνται pintools. Η δυναμική ανάλυση πραγματοποιείται κατά την εκτέλεση των προγραμμάτων και βασίζεται στα compiled binary αρχεία (εκτελέσιμα). Ένα από τα κύρια προτερήματα που του Pin είναι ότι παρέχει ένα αφαιρετικό πλαίσιο που υποκρύπτει ιδιοσυγκρασίες που μπορεί να έχει το σύνολο εντολών που εξετάζουμε, και επιτρέπει να αντλήσουμε χρήσιμες και κατανοητές πληροφορίες για την εφαρμογή μας. Τέλος, η απήχηση αυτού του framework φαίνεται από το μέγεθος της ενεργούς ερευνητικής κοινότητας που ασχολείται με αυτό.

3.3 Εργαλεία για την μέτρηση της κατανάλωσης ισχύος σε ετερογενείς αρχιτεκτονικές

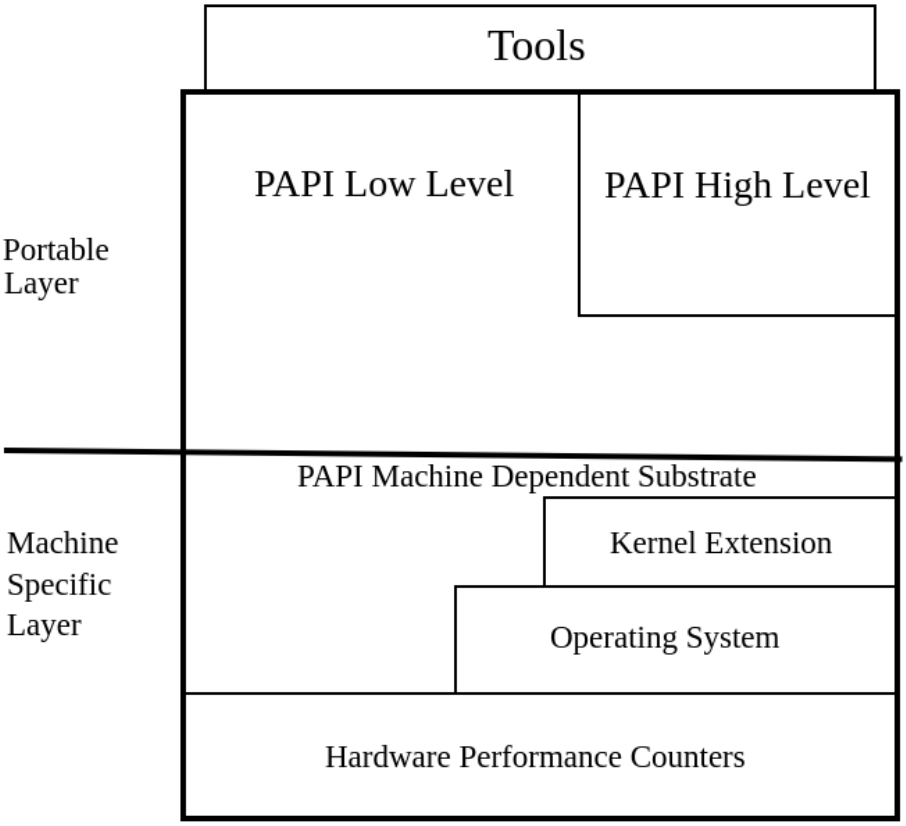
Όπως αναφέραμε και προηγουμένως, μια σημαντική συνεισφορά της παρούσας διπλωματικής είναι ότι μέσω των μοντέλων που προτείνει προσφέρει μια πρόβλεψη σχετικά με το ποια από τις συγκρινόμενες ετερογενείς αρχιτεκτονικές έχει την χαμηλότερη κατανάλωση ισχύος για μια συγκεκριμένη εφαρμογή. Για να εκπαιδευτούν και τελικά να διαμορφωθούν όμως τα κατάλληλα μοντέλα απαιτούνται δεδομένα αναφορικά με την παρατηρούμενη συμπεριφορά των ετερογενών αρχιτεκτονικών στις διάφορες εφαρμογές. Χρειαζόμαστε δηλαδή εργαλεία από τα οποία μπορούμε να μετρήσουμε την κατανάλωση ισχύος σε κάθε περίπτωση. Στο παρόν υποκεφάλαιο παρουσιάζουμε τα δύο βασικά εργαλεία που αξιοποιήσαμε, το εργαλείο PAPI για τις αρχιτεκτονικές των multicore CPUs και το NVML API για τις GPU αρχιτεκτονικές [14], [15].

Το **PAPI (Performance Application Programming Interface)** δημιουργήθηκε για την σχεδίαση και υλοποίηση ενός αποδοτικού API (Application Programming Interface) για την εύκολη πρόσβαση στους hardware performance counters (μετρητές επίδοσης υλικού) που παρέχονται στους σύγχρονους multicore CPUs. Οι hardware counters αποτελούν πλέον ένα σταθερό χαρακτηριστικό των σύγχρονων multicore CPUs, σε αυτούς στηρίζονται πολλοί ερευνητές και προγραμματιστές για την δημιουργία εργαλείων που επιτηρούν την συμπεριφορά των εφαρμογών πάνω σε μια συγκεκριμένη CPU αρχιτεκτονική. Η στοχοθεσία του PAPI είναι:

- να αξιοποιηθεί για την ανάλυση της επίδοσης για διαφορετικές CPU πλατφόρμες
- να παρουσιάσει ένα σύνολο καλά ορισμένων μετρικών για την επίδοση
- να παρέχει ένα κοινό API μεταξύ των χρηστών, κατασκευαστών, ακαδημαϊκών
- να είναι εύχρηστο, πλήρως τεκμηριωμένο και διαθέσιμο δωρεάν

Κεντρικής σημασίας είναι να κατανοήσουμε την αρχιτεκτονική του PAPI. Αυτή συνίσταται σε δύο επίπεδα. Το πρώτο επίπεδο είναι το Portable Layer το οποίο με την σειρά του

διακρίνεται αφενός στο API (low level και high level) το οποίο εκτίθεται στον χρήστη και παρέχει λειτουργίες που είναι κοινές για τις διαφορετικές πλατφόρμες. Το δεύτερο επίπεδο είναι το Machine Specific Layer το οποίο εξαρτάται όπως μαρτυρά το όνομά του από την συγκεκριμένη πλατφόρμα στην οποία χρησιμοποιούμε το PAPI. Στο επίπεδο αυτό ορίζονται και αντιστοιχίζονται οι κλήσεις που γίνονται στο πρώτο επίπεδο σε κλήσεις προσαρμοσμένες ακριβώς στην πλατφόρμα που χρησιμοποιούμε. Στην εικόνα 3.1 βλέπουμε ένα διάγραμμα που οπτικοποιεί αυτήν την περιγραφή.



Σχήμα 3.1: *Αρχιτεκτονική του PAPI*

Το PAPI στηρίζεται στην χρήση και διαχείριση μιας σειράς από events. Τα events αποτελούν signals (σήματα) που συνδέονται με συγκεκριμένες συναρτήσεις του CPU. Οι hardware performance counters αποτελούν ουσιαστικά ένα μικρό σύνολο από καταχωρητές οι οποίοι είναι υπεύθυνοι για την αρίθμηση συγκεκριμένων events που πραγματοποιούνται κατά την εκτέλεση του προγράμματος που επιθυμούμε να εξετάσουμε. Κάθε CPU παρέχει events τα οποία είναι native για την αρχιτεκτονική στην οποία εκτελούμε το πρόγραμμα. Το PAPI παρέχει ένα επίπεδο αφαίρεσης για αυτά τα native events και επιτρέπει την πρόσβαση σε αυτά μέσω των preset events τα οποία είναι διαθέσιμα στον χρήστη μέσω του PAPI. Στα πλαίσια μιας εφαρμογής που θέλουμε να μελετήσουμε το PAPI μας δίνει τη δυνατότητα να ορίσουμε ένα EventSet (σύνολο από events), για τα events αυτά στο τέλος θα λάβουμε τις τιμές που προέκυψαν από την μέτρηση κατά την εκτέλεση.

Τι ακριβώς είναι όμως τα native events; Τα native events είναι όλα τα events που μπορούμε

να μετρήσουμε για μία CPU. Σε πολλές περιπτώσεις είδαμε ότι μπορούμε να έχουμε πρόσβαση σε αυτά μέσω αντίστοιχων preset events, όμως ακόμα και όταν δεν παρέχεται αυτή η διεπαφή έχουμε τη δυνατότητα να κάνουμε ρητή αναφορά σε αυτά απευθείας. Από την άλλη, τα preset events συναντώνται σε πολλούς CPUs όπου λειτουργούν ως συμβολικά ονόματα (PAPI preset name) για συγκεκριμένα native events που αφορούν συγκεκριμένους hardware πόρους.

Παρά τη μεγάλη ποικιλία από μετρικές που μπορούμε να μετρήσουμε με το PAPI, οι ανάγκες για τις οποίες θα το χρησιμοποιήσουμε είναι πολύ συγκεκριμένες και αφορούν εξ ολοκλήρου την συμπεριφορά έκαστης εφαρμογής από την σκοπιά της κατανάλωσης ισχύος. Ειδικότερα τα τελευταία χρόνια το PAPI τείνει να ταυτιστεί με ένα εργαλείο για την μέτρηση της ενέργειας και της ισχύος [16]. Συγκεκριμένα, με το ντεμπούτο της αρχιτεκτονικής οικογένειας Sandy Bridge επεξεργαστών της Intel έγινε εισαγωγή της διεπαφής RAPL (Running Average Power Limit). Σκοπός αυτής της διεπαφής είναι να μπορεί ο χρήστης να ορίσει ένα παράθυρο αποδεκτών τιμών ισχύος για την λειτουργία του επεξεργαστή. Τα αποτελέσματα αυτού του μοντέλου είναι προσβάσιμα από τον χρήστη μέσω των καραχωρητών MSR (Model Specific Register).

Μέσω της διεπαφής των RAPL counters έχουμε πρόσβαση σε μέγεθη όπως η μετρούμενη ισχύς ή ενέργεια, σε συγκεκριμένο hardware του συστήματός μας. Παραδείγματος χάριν, μπορούμε να λάβουμε διακριτές μετρήσεις για την κατανάλωση ισχύος της DRAM (κύριας μνήμης), των chip των CPUs συνολικά, των πυρήνων έκαστου CPU chip, κλπ.

Για την μέτρηση της κατανάλωσης ισχύος σε GPU αρχιτεκτονικές της NVIDIA το πιο συχνά χρησιμοποιούμενο εργαλείο είναι το **NVIDIA Management Library (NVML)** API. Πρόκειται για μια προγραμματιστική διεπαφή που βασίζεται στην γλώσσα προγραμματισμού C. Η διεπαφή παρέχει στους προγραμματιστές πρόσβαση σε βασικές πληροφορίες που αφορούν την χρησιμοποιούμενη GPU, οι πληροφορίες αυτές αφορούν το ποσοστό χρήσης της GPU, την τρέχουσα θερμοκρασία, την τρέχουσα ισχύ που καταναλώνει η GPU. Το NVML API επιτρέπει την δημιουργία πολυνηματικών εφαρμογών για να εποπτεύουν διαφορετικά χαρακτηριστικά και συμπεριφορές μιας δεδομένης εφαρμογής που εκτελείται σε μια συγκεκριμένη GPU. Αποτελεί την βιβλιοθήκη στην οποία στηρίζεται το εργαλείοNVIDIA System Management Interface (nvidia-smi) [17]. Μια σημαντική δυνατότητα που παρέχει το NVML API είναι μεταξύ άλλων ότι σε συστήματα με πολλές διαθέσιμες GPUs μπορούμε να καθορίσουμε για ποια θα λάβουμε τις μετρήσεις που έχουμε ορίσει.

Σε επόμενο κεφάλαιο θα δούμε εξονυχιστικά πώς αξιοποιούνται αυτά τα εργαλεία, καθώς και θα παρουσιάσουμε αφαιρετικά κάποια ειδικά προγράμματα που κατασκευάσαμε για να επεξεργαστούμε τις πληροφορίες που αποκομίσαμε από την χρήση τους.

Κεφάλαιο 4

Μέθοδοι Μηχανικής Μάθησης

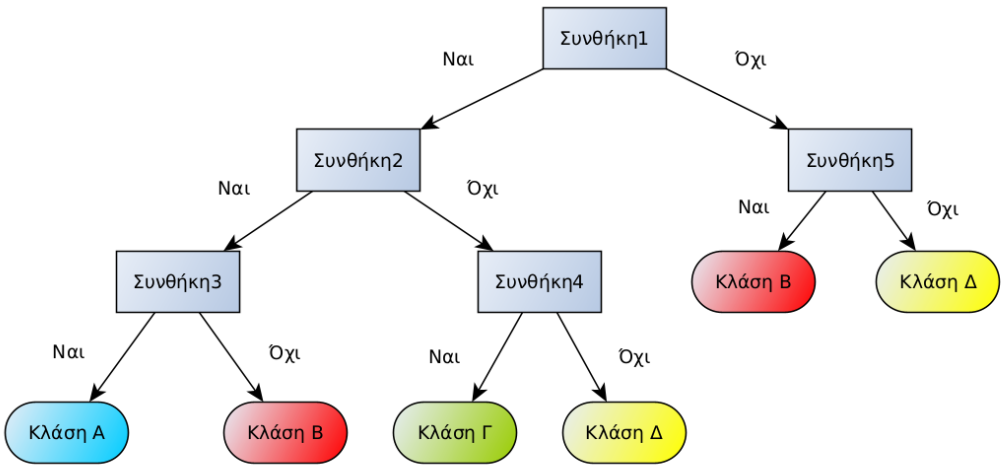
Στο κεφάλαιο αυτό παρουσιάζονται δύο μέθοδοι μηχανικής μάθησης που αξιοποιήσαμε για την δημιουργία των μοντέλων πρόβλεψης για την απόδοση και την ισχύ. Σκιαγραφούμε τα βασικά θεωρητικά θεμέλια στα οποία στηρίζονται. Πρόκειται για μεθόδους που χρησιμοποιούνται σε προβλήματα classification (ταξινόμησης). Από την σκοπιά της απόδοσης τις χρησιμοποιούμε για να διαμορφώσουμε ένα multiclass classification των εφαρμογών σε κατηγορίες ανάλογα με το σχετικό speedup που προσφέρει μια GPU έναντι ενός CPU. Από την σκοπιά της κατανάλωσης ισχύος τις χρησιμοποιούμε για να διαμορφώσουμε ένα binary classification των εφαρμογών ανάλογα με το ποια αρχιτεκτονική είναι η κατάλληλη με γνώμονα την χαμηλότερη δυνατή κατανάλωση ισχύος.

Το sci-kit learn είναι ένα σύνολο υλοποιήσεων μεθόδων μηχανικής μάθησης σε Python. Χάρις σε αυτό μπορούμε άμεσα και εύκολα να βγάλουμε χρήσιμα μοντέλα και πορίσματα πάνω σε μεγάλο πλήθος δεδομένων που έχουμε για την περιγραφή ενός συγκεκριμένου προβλήματος. Υπάρχει μεγάλη ποικιλία σε υλοποιήσεις μεθόδων μηχανικής μάθησης που επιτελούν διαφορετικούς σκοπούς όπως classification και regression προβλήματα. Για την περίπτωση που εξετάζουμε στην διπλωματική μας θα εστιάσουμε στην πρώτη κατηγορία. Στο πρακτικό μέρος, σε επόμενα κεφάλαιο, θα γίνει κατανοητό γιατί δεν κάναμε αναγωγή του προβλήματος σε regression πρόβλημα. Οι μέθοδοι στις οποίες στηριχτήκαμε είναι το **Decision Tree Classifier** και το **Random Forest Classifier** όπως αυτές έχουν υλοποιηθεί στο scikit-learn της γλώσσας προγραμματισμού Python [18, 19, 20].

Και οι δύο αυτές μέθοδοι μηχανικής μάθησης είναι **supervised**. Ας δούμε περιγραφικά τι ακριβώς σημαίνει supervised μέθοδος. Για προβλήματα στα οποία έχουμε εποπτεία των φυσικών μεγεθών και του ρόλου που αυτά παίζουν για μία δεδομένη συμπεριφορά είναι λογικό να επιδιώξουμε το μοντέλο που εκπαιδεύουμε να εκμεταλλευτεί αυτήν την εποπτεία. Δηλαδή δεν αντιμετωπίζουμε το πρόβλημα που πάμε να μελετήσουμε αγνωστικιστικά αλλά αντίθετα σταδιακά χτίζεται εμπειρία κατά την εκπαίδευση του μοντέλου. Τα δεδομένα που παρέχουμε απαρτίζουν ένα σύνολο από χαρακτηριστικά με συγκεκριμένη ονομασία (labeled features). Τα χαρακτηριστικά αυτά θεωρούμε ότι συνδέονται μεταξύ τους από μία συγκεκριμένη σχέση, αυτή η σχέση συνήθως λόγω αυξημένης πολυπλοκότητας (λόγου χάριν μεγάλο πλήθος χαρακτηριστικών) είναι δύσκολο να παραχθεί με έναν αναλυτικό τρόπο. Αντ'αυτού μπορούμε με κάποιο κόστος στην ακρίβεια να εκφράσουμε αυτήν την σχέση μέσω του μοντέλου, οπότε το εκπαιδευμένο μοντέλο μπορεί να προβλέψει την τιμή ενός χαρακτηριστικού μέσω των υπόλοιπων χαρακτηριστικών που έχει στην διάθεσή του.

Η μέθοδος **Decision Tree Classifier** στηρίζεται στα Decision Trees (Δέντρα Αποφάσεων). Τα Decision Trees είναι μια supervised μέθοδος μηχανικής μάθησης όπου οι προσεγγίσεις προς το ζητούμενο μέγεθος γίνονται στη βάση ενός συνόλου από if-then-else κανόνων. Η μέθοδος των Decision Trees χρησιμοποιείται τόσο για classification όσο και για regression προβλήματα. Τα δεδομένα που παρέχουμε στην μέθοδο και εκφράζουν το πρόβλημα που εξετάζουμε κάθε φορά σπάνε σε υποσύνολα καθώς σταδιακά χτίζεται το Decision Tree. Η διαδικασία συνεχίζεται έως ότου φτάσουμε στα φύλλα του δέντρου όπου εκτίθεται η τελική απόφαση, δηλαδή η πρόβλεψη που δίνει η μέθοδος για το μονοπάτι από την ρίζα προς το συγκεκριμένο φύλλο. Στην περίπτωση που εξετάζουμε στη μελέτη μας, η απόφαση είναι σε ποια κλάση εντάσσεται το πρόβλημα του οποίου τα χαρακτηριστικά έχουν οδηγήσει στο συγκεκριμένο φύλλο.

Για την διαμόρφωση του Decision Tree για ένα πρόβλημα υπάρχουν ένα σύνολο από βήματα που διενεργούνται αναδρομικά επί των δεδομένων. Το σημαντικότερο είναι το splitting (διαχωρισμός), κατά το splitting τα δεδομένα εισόδου του προβλήματος χωρίζονται σε δύο κατηγορίες ανάλογα με την τιμή μίας μεταβλητής, δηλαδή ενός χαρακτηριστικού του προβλήματος. Ο διαχωρισμός αυτός στο Decision Tree αποτυπώνεται ως ένας κόμβος απόφασης όπου διατυπώνεται μία συνθήκη για την μεταβλητή διαχωρισμού. Κάθε συνθήκη αναφέρεται αποκλειστικά σε μία μεταβλητή. Κατά σύμβαση στο αριστερό υποδέντρο εντάσσονται τα προβλήματα που επαληθεύουν την συνθήκη που περιέχει ο κόμβος ενώ στο δεξί υποδέντρο εντάσσονται τα προβλήματα που δεν επαληθεύουν την συνθήκη. Αυτή η διαδικασία γίνεται αναδρομικά έως ότου φτάσουμε στην επιθυμητή απόφαση που δίνεται στα φύλλα. Υπάρχουν διάφορα κριτήρια για να καθοριστεί η σειρά των μεταβλητών με τις οποίες θα διαμορφωθεί η συνθήκη σε κάθε κόμβο απόφασης, αυτό συνεπώς καθορίζει και το πώς θα είναι το παραχθέν Decision Tree.



Σχήμα 4.1: *Παράδειγμα ενός Decision Tree*

Το κριτήριο με βάσει το οποίο επιλέγεται ένα χαρακτηριστικό των προβλημάτων με τα οποία εκπαιδεύουμε το μοντέλο και τελικά παράγουμε το Decision Tree είναι συνήθως ένα εκ των Gini και Entropy. Στα πλαίσια της δικής μας μελέτης καταλήξαμε στο κριτήριο Gini,

ωστόσο η ακρίβεια του μοντέλου μας και για τα δύο κριτήρια ήταν στα ίδια επίπεδα.

Το μεγάλο πλεονέκτημα του Decision Tree Classification είναι ότι πέραν της πρόβλεψης παρέχει στο χρήστη μια αναπαράσταση για να καταλάβει ποια είναι τα κύρια χαρακτηριστικά του προβλήματος που καθορίζουν την κλάση στην οποία αυτό θα ενταχθεί. Συγκεκριμένα για το πρόβλημα το οποίο θέλουμε να εντάξουμε σε μια κλάση παίρνουμε τον πίνακα των χαρακτηριστικών του και εξετάζουμε μία-μία τις συνθήκες, αποφασίζουμε σε κάθε περίπτωση σε ποιο υποδέντρο θα συνεχίσουμε την διάσχιση έως ότου φτάσουμε σε ένα φύλλο όπου δίνεται η πρόβλεψη. Από το μονοπάτι που ακολουθήσαμε μπορούμε να πάρουμε μια σύνθετη συνθήκη η οποία προκύπτει ως συνάρτηση των συνθηκών που συναντήσαμε. Η σύνθετη αυτή συνθήκη εμπεριέχει λοιπόν τα χαρακτηριστικά του προβλήματος που αξιολογήσαμε διασχίζοντας κάθε απλή συνθήκη. Αυτή η αναπαράσταση μπορεί λοιπόν να αξιολογηθεί από την εποπτεία και την διαίσθηση που έχουμε και τελικά να κρίνουμε αν το Decision Tree που παράγεται παρέχει κάποια χρήσιμη πληροφορία ή απλά χρησιμοποιεί φωτογραφικές συνθήκες για να ″σώσει τα φαινόμενα″.

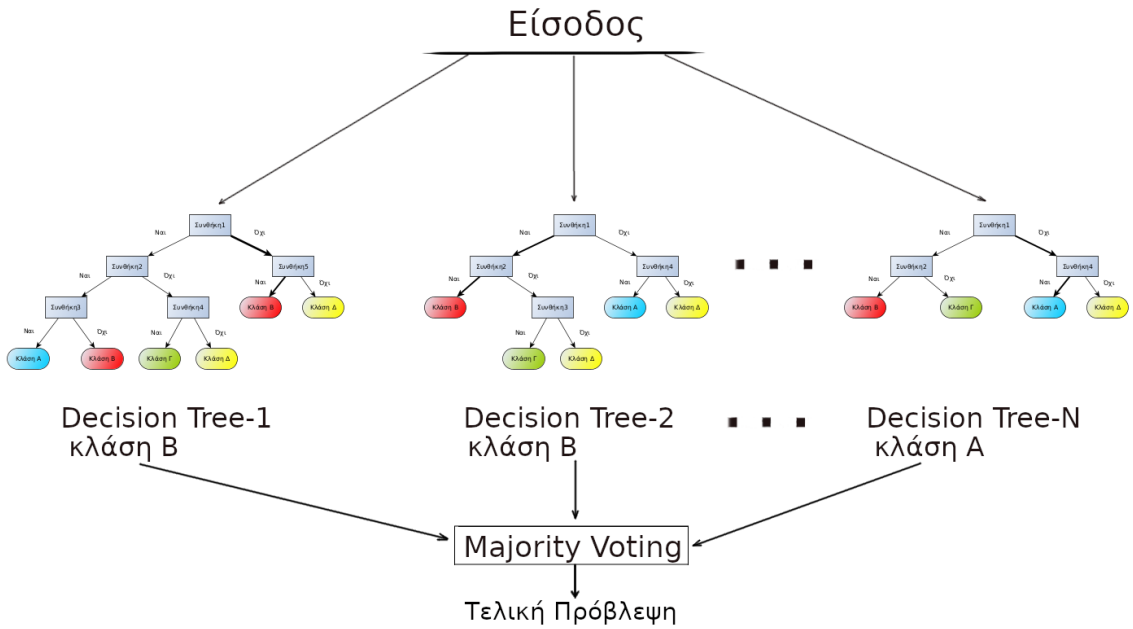
Υπάρχουν μια σειρά από μειονεκτήματα όσον αφορά τη χρήση του Decision Tree Classifier. Αρχικά είναι πολύ ευαίσθητο στα δεδομένα, δηλαδή μια πολύ μικρή αλλαγή στα δεδομένα με τα οποία εκπαιδεύουμε το μοντέλο μπορεί να οδηγήσει σε σημαντικές διακυμάνσεις στη μορφή και άρα στην πληροφορία του Decision Tree που παράγουμε. Ένα άλλο μειονέκτημα είναι το πρόβλημα που αναφέραμε προηγουμένως. Το επίσημο όνομα αυτού του προβλήματος είναι overfitting. Η χρήση φωτογραφικών κριτηρίων σημαίνει ότι το Decision Tree αντί να προσπαθεί να εκπαιδευτεί κατάλληλα από τα δεδομένα εισόδου, επιχειρεί να ″απομνημονεύσει τις λύσεις″ για κάθε ένα από τα προβλήματα με τα οποία το εκπαιδεύουμε. Ως εκ τούτου, το παραχθέν Decision Tree μπορεί να είναι πραγματικά πολύπλοκο στη μορφή του και εξαιρετικά ανακριβές για τα προβλήματα με τα οποία ελέγχουμε την ακρίβειά του μετά την διαδικασία της εκπαίδευσης.

Μια συνήθης τακτική για να ξεπεραστεί το overfitting είναι η μέθοδος pruning (κλάδεμα). Η μέθοδος αυτή οδηγεί σε ένα μικρότερο, πιο κατανοητό και κατά περιπτώσεις πιο ακριβές Decision Tree. Θα εξηγήσουμε επαρκώς την διαδικασία αυτή στο πρακτικό μέρος της διπλωματικής σε επόμενο κεφάλαιο.

Η δεύτερη μέθοδος μηχανικής μάθησης που χρησιμοποιούμε στα πλαίσια της διπλωματικής μας εργασία είναι ο **Random Forest Classifier** που παρέχει το sci-kit learn. Η μέθοδος αυτή στηρίζεται στον αλγόριθμο Random Forest. Το όνομα του αλγορίθμου δεν είναι τυχαίο, στην ουσία πρόκειται για μία μέθοδο που στηρίζεται σε ένα δάσος από Decision Trees για να υπηρετήσει τον σκοπό της. Σε αυτήν την μέθοδο για τα δεδομένα εισόδου που χρησιμοποιούνται για την εκπαίδευση του μοντέλου διαμορφώνεται ένα καθορισμένο πλήθος από Decision Trees, ένα δάσος. Εν συνεχεία, όταν δοθεί ένα πρόβλημα για το οποίο θέλουμε να γίνει μια πρόβλεψη από το μοντέλα, αυτό που γίνεται είναι ότι κάθε ένα από τα Decision Trees κάνει μία πρόβλεψη για το πρόβλημα αυτό. Έπειτα ανάλογα με το κριτήριο που έχει επιλεγεί δίνεται ως τελική πρόβλεψη μία συνάρτηση των προβλέψεων που έχουν δώσει τα Decision Trees. Η πιο συνήθης προσέγγιση είναι το majority vote, δηλαδή η πρόβλεψη που παρέχει το μοντέλο για μια συγκεκριμένη είσοδο είναι η πρόβλεψη που πλειοψηφικά δίνουν τα διαθέσιμα Decision

Trees.

Επειδή ακριβώς έχουμε ένα δάσος από Decision Trees σε αυτή την μέθοδο δεν υπάρχει μία μοναδική γραφική αναπαράσταση για το μοντέλο. Επομένως συγκριτικά με την πρώτη μέθοδο έχουμε περιορισμένη εποπτεία στο ποια χαρακτηριστικά διαμορφώνουν μια δεδομένη συμπεριφορά. Ωστόσο, επειδή ακριβώς έχουμε ένα δάσος από Decision Trees το μοντέλο είναι απαλλαγμένο από το πρόβλημα overfitting καθώς για ένα μεγάλο πλήθος από Decision Trees μπορούμε να θεωρήσουμε ότι οι overfitted λύσεις δεν επικρατούν πλειοψηφικά. Τέλος, όπως θα δούμε και σε επόμενο κεφάλαιο παρέχεται μερική εποπτεία στο ποια είναι τα χαρακτηριστικά που ρυθμίζουν κατά κύριο λόγο την τελική πρόβλεψη.



Σχήμα 4.2: *Παράδειγμα ενός Random Forest Classifier*

Στο πρακτικό μέρος που ακολουθεί θα δούμε κάποια ακόμα εργαλεία και πώς αυτά μπορούν να μας βοηθήσουν να βγάλουμε χρήσιμα πορίσματα για τη μελέτη μας.

Μέρος II

Πρακτικό Μέρος

Κεφάλαιο 5

Πειραματική υποδομή

Για την εκπόνηση της παρούσας διπλωματικής εργασίας αξιοποιήθηκε ένα σύνολο πόρων που έχει στη διάθεσή του το Εργαστήριο Υπολογιστικών Συστημάτων. Φροντίσαμε η μελέτη μας να καλύπτει ένα πλήθος από διαφορετικές γενιές αρχιτεκτονικών από multicore CPUs και GPUs. Αυτή η ποικιλομορφία στην πειραματική μας υποδομή επιλέχθηκε ακριβώς για να δώσει απάντηση στο ζήτημα του hardware diversity. Όπως είπαμε στερείται νοήματος η μονολιθική αντιμετώπιση των ετερογενών αρχιτεκτονικών, δεν είναι όλοι οι CPUs ίδιοι και δεν είναι όλες οι GPUs ίδιες.

Το πόσο βασική είναι η μη μονολιθική αντιμετώπιση των συγκρινόμενων ετερογενών αρχιτεκτονικών διαφαίνεται από το ακόλουθο παράδειγμα:
Ας υποθέσουμε ότι έχουμε στην διάθεσή μας μία εφαρμογή. Εκτελούμε και μετράμε την απόδοση της εφαρμογής σε ένα σύνολο ετερογενών αρχιτεκτονικών. Ένα πολύ πιθανό σενάριο είναι η εφαρμογή αυτή να εκτελείται πιο αργά σε έναν CPU_A από ότι σε μία GPU_A. Έπειτα αν εξετάσουμε αυτήν την εφαρμογή σε έναν CPU_B (που έστω ότι διαθέτει περισσότερους λογικούς πυρήνες) παρατηρούμε ότι εκτελείται γρηγορότερα από ότι στην προαναφερθείσα GPU_A. Σε αυτήν την περίπτωση για ποια αρχιτεκτονική θα έπρεπε να σχεδιάσουμε την εφαρμογή μας; Ένα αντίστοιχο παράδειγμα μπορεί να δοθεί αν το κριτήριο είναι η χαμηλότερη δυνατή κατανάλωση ισχύος.

Αυτό που αναδεικνύει το συγκεκριμένο παράδειγμα είναι ότι παρά τις μεγάλες αρχιτεκτονικές διαφορές των ετερογενών αρχιτεκτονικών, υπάρχουν περιπτώσεις που δεν μπορούμε να αποτιμήσουμε εύκολα το ποια είναι η κατάλληλη αρχιτεκτονική για την υλοποίηση της εφαρμογής μας. Σε τέτοιες περιπτώσεις δεν μπορούμε να αποτιμήσουμε εύκολα ποια συγκεκριμένη αρχιτεκτονική υλοποίηση από τις προσφερόμενες είναι η επωφελέστερη για την εφαρμογή μας.

Οι προηγούμενες λύσεις επί του ζητουμένου όπως είδαμε δεν απαντάνε επαρκώς στο ζητούμενο του hardware diversity. Συγκεκριμένα βλέπουμε ότι η πειραματική τους υποδομή περιορίζεται σε ένα εξεταζόμενο multicore CPU και σε δύο εξεταζόμενες GPUs. Αν και στο XAPP παρατίθεται μια ανάλυση για μια γενίκευση της σύγκρισης μεταξύ περισσότερων ετερογενών αρχιτεκτονικών, αυτή η ανάλυση δεν επικυρώνεται επαρκώς και πειραματικά. Επίσης στις σύγχρονες υπηρεσίες υπολογιστικού νέφους προσφέρεται μεγάλη ποικιλία για κάθε μία από τις οικογένειες των ετερογενών αρχιτεκτονικών. Για αυτούς τους λόγους επιλέξαμε να εξετάσουμε το σύνολο των εφαρμογών μας σε τρεις διαφορετικές γενιές από multicore CPUs της Intel, και τρεις διαφορετικές γενιές από GPUs της NVIDIA.

5.1 Πειραματική υποδομή των multicore CPUs

Για την εκτέλεση και μελέτη των εφαρμογών της Rodinia σουίτας αξιοποιήσαμε τρεις διαφορετικές γενιές επεξεργαστών της Intel. Οι γενιές που επιλέξαμε είναι οι Sandy Bridge, Haswell και Broadwell. Κάθε υπολογιστικός κόμβος του εργαστηρίου που αξιοποιήσαμε αποτελείται από ένα πλήθος από πανομοιότυπους multicore CPUs έκαστης γενιάς. Συγκεκριμένα για την γενιά Sandy Bridge χρησιμοποιήσαμε έναν από τους τέσσερις διαθέσιμους Intel(R) Xeon(R) CPU E5-4620 @ 2.20GHz, ενώ στους άλλους δύο χρησιμοποιήσαμε ολόκληρο το dual socket σύστημα που στην πρώτη περίπτωση απαρτιζόταν από Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.6 GHz και στην δεύτερη περίπτωση από Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz [21, 22, 23].

Στο παρακάτω πίνακα παραθέτουμε κάποιες βασικές διαφορές μεταξύ των συγκεκριμένων υλοποιήσεων [24, 25, 26, 27, 28]. Σε καμία περίπτωση η μετάβαση από τη μία γενιά στην επόμενη δεν εκφράζεται μόνο από τα μεγέθη που φαίνονται εδώ, υπάρχουν πολλές ακόμη διαφοροποιήσεις για τις οποίες πρέπει κανείς να κάνει ενδελεχή μελέτη ειδικά όταν το διακύβευμα είναι το αποδοτικότερο tuning μιας εφαρμογής σε μια συγκεκριμένη υλοποίηση. Παρ'όλα αυτά κρίνουμε ότι ο κάτωθι πίνακας δείχνει μεταξύ άλλων κάποιες σημαίνουσες διαφορές στις οποίες θα βασιστούμε εν συνεχεία για την εισαγωγή κάποιων coefficients στο μοντέλο μας.

Πίνακας 5.1: *Χαρακτηριστικά μεγέθη των χρησιμοποιούμενων γενιών από multicore CPUs*

	Sandy Bridge (E5-4620)	Haswell (E5-2697 v3)	Broadwell (E5-2630 v4)
Lithography	32nm	22nm	14nm
cores/threads	8c/16t	14c/28t	10c/20t
Base Frequency	2.2GHz	2.6GHz	2.2GHz
Max Turbo Frequency	2.6GHz	3.6GHz	3.1GHz
L1 cache size	32KB Instr., 32KB Data	32KB Instr., 32KB Data	32KB Instr., 32KB Data
L1 associative	8-way	8-way	8-way
L2 cache size	256KB	256KB	256KB
L2 associative	8-way	8-way	8-way
L3 cache size	16MB	35MB	25MB
L3 associative	16-way	20-way	20-way
SIMD width SP	4	8	8
SIMD width DP	2	4	4
Memory Bandwidth	42.6GB/s	68GB/s	68.3GB/s
Issue width	16B/cycle	16B/cycle	16B/cycle
OoO window ROB entries	168	192	192
TDP	95W	145W	85W

5.2 Πειραματική υποδομή των GPUs

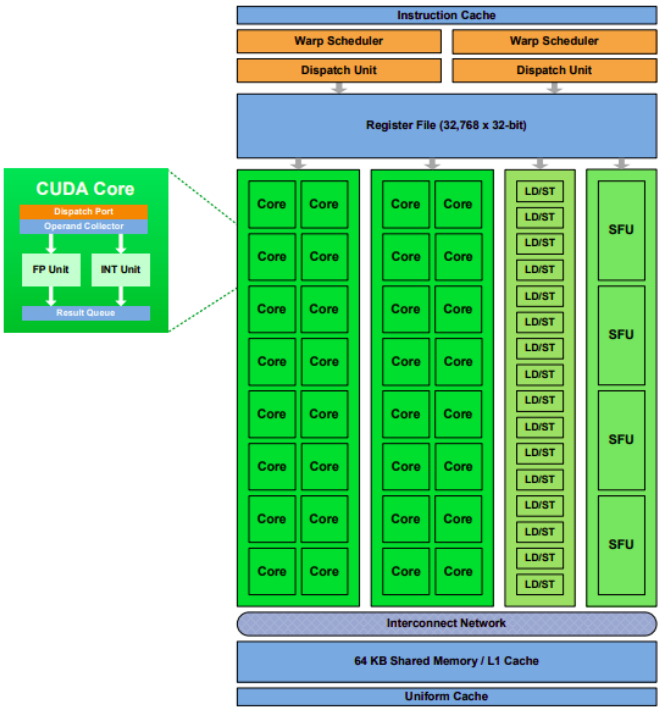
Για την εκτέλεση των αντίστοιχων CUDA υλοποιήσεων των εφαρμογών της Rodinia σουίτας επιλέξαμε υλοποιήσεις των αρχιτεκτονικών οικογενειών της NVIDIA: Fermi, Kepler και Maxwell. Για να είμαστε πιο σαφείς οι υλοποιήσεις αυτών των αρχιτεκτονικών οικογενειών που είχαμε στη διάθεσή μας ήταν οι Tesla m2050(Fermi), Tesla K40c(Kepler) και Quadro M4000(Maxwell) [29, 30, 31]. Σε αντίθεση με τις CPU αρχιτεκτονικές που παραθέσαμε προηγουμένως εδώ οι αρχιτεκτονικές διαφορές από γενιά σε γενιά οπτικοποιούνται πιο εύκολα. Έχουμε σημαντικές σχεδιαστικές και ποσοτικές αλλαγές που πρέπει να λάβουμε υπόψιν για να πραγματοποιήσουμε μια επιτυχημένη σύγκριση στα πλαίσια έκαστης εφαρμογής.

Ας δούμε πιο συγκεκριμένα τα χαρακτηριστικά των υλοποιήσεων που χρησιμοποιήσαμε.

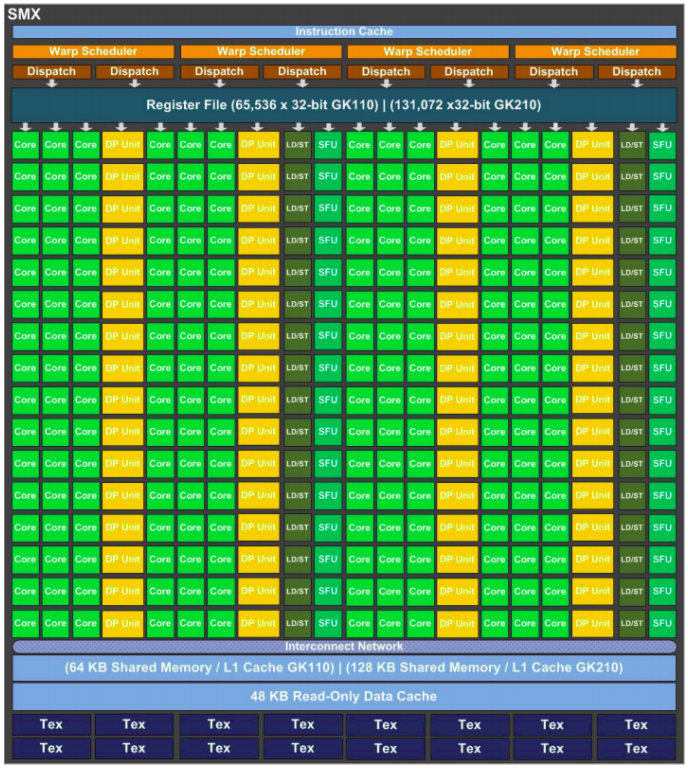
Πίνακας 5.2: *Χαρακτηριστικά μεγέθη των χρησιμοποιούμενων γενιών από GPUs*

	Fermi (Tesla m2050)	Kepler (Tesla K40c)	Maxwell (Quadro m4000)
Lithography	40nm	28nm	28nm
CUDA cores	448	2880	1664
number of SMs	14	15	13
CUDA cores per SM	32	192	128
Max Clock Rate	1147MHz	745MHz	773MHz
Memory Bus Width	384-bit	384-bit	256-bit
Constant Memory	64KB	64KB	64KB
Register Memory per block	32KB	64KB	64KB
Shared Memory per block	48KB	48KB	48KB
L2 cache size	768KB	1.5MB	2MB
Memory Bandwidth	148.4GB/s	288.4GB/s	192.3GB/s
Memory size	3GB	12GB	8GB
TDP	225W	235W	120W

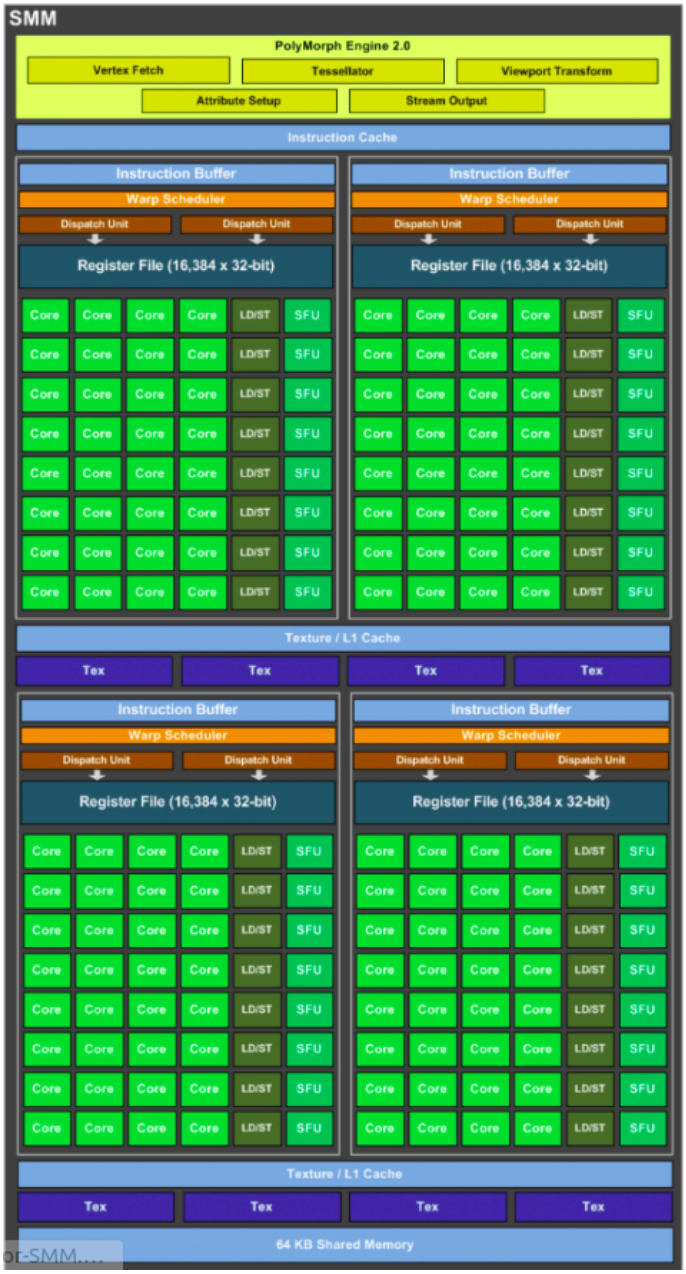
Το γεγονός ότι δεν μπορούμε να αντιμετωπίζουμε με μονολιθικό τρόπο τις GPUs ίσως δεν εκφράζεται επαρκώς από τον παραπάνω πίνακα. Τα θεμελιώδη αρχιτεκτονικά στοιχεία παραμένουν σταθερά όμως η σχεδίαση γίνεται ολοένα και πολυπλοκότερη. Σχηματικά μπορούμε να δούμε ένα σύνολο από διαφορετικές προσεγγίσεις που έγιναν από γενιά σε γενιά σε επίπεδο Streaming Multiprocessor όπου συναντάμε ριζικές αλλαγές στην οργάνωση του υλικού.



Σχήμα 5.1: *Streaming Multiprocessor της αρχιτεκτονικής Fermi*



Σχήμα 5.2: *Streaming Multiprocessor της αρχιτεκτονικής Kepler*



Σχήμα 5.3: *Streaming Multiprocessor της αρχιτεκτονικής Maxwell*

5.3 Κύριες αρχιτεκτονικές διαφορές μεταξύ multicore CPUs και GPUs

Αυτό που διαπιστώνει κανείς από τα προηγούμενα υποκεφάλαια είναι ότι καθώς εξελίσσονται οι αρχιτεκτονικές πραγματοποιούνται άλλοτε μικρές και άλλοτε σημαντικές αλλαγές σε κάθε αρχιτεκτονική οικογένεια. Σε μια πρώτη φάση όμως για να απαντηθεί το ερώτημα του ποια αρχιτεκτονική οικογένεια είναι κατάλληλη για την εφαρμογή μου, πρέπει να δοθούν κάποια οριζόντια κριτήρια και αυτά να επικεντρωθούν στα σημεία όπου κρίνεται τελικώς η απόδοση. Η εστίαση στα συγκεκριμένα σημεία θα παρουσιαστεί στη συνέχεια. Προς το παρόν, ας εξετάσουμε κάποιες κεντρικές αρχιτεκτονικές διαφορές, το πού αποσκοπούν και πού

μπορούμε να αναζητήσουμε συσχετίσεις μεταξύ των multicore CPUs και GPUs.

Πολλές φορές η ανομοιογενής ορολογία που χρησιμοποιείται από τους κατασκευαστές των GPUs μπορεί να οδηγήσει σε παρανοήσεις και άρα σε εσφαλμένη κατανόηση κεντρικών αρχιτεκτονικών εννοιών. Ο όρος ″CUDA core″ ενώ παραπέμπει σε μία έννοια κοντά στον υπολογιστικό πυρήνα των CPUs, πρακτικά αποτελεί ένα single precision ALU (Arithmetic Logic Unit). Στο σύνολο των CUDA cores ανατίθεται υπολογιστικός φόρτος μέσω της ομαδοποίησης, οργάνωσης και επίβλεψής τους από έναν Streaming Multiprocessor. Στην περίπτωση της AMD οι Streaming Multiprocessors αποκαλούνται Compute Units ενώ αντί του επιρρεπή σε παρερμηνείες όρου ″CUDA core″, τα διαθέσιμα ALUs ανά Compute Unit ομαδοποιούνται σε 16-wide ή 32-wide SIMD Vector Machines. Προφανώς, υπάρχουν μεγάλες σχεδιαστικές αποκλίσεις μεταξύ των δύο κατασκευαστών, θεωρούμε όμως ότι η παραπάνω παρατήρηση είναι καίριας σημασίας για να αντιληφθούμε καλύτερα τη βασική λογική πάνω στην οποία οργανώνονται και λειτουργούν οι GPUs.

Από άποψη παράλληλου υπολογιστικού φόρτου σε επίπεδο υλικού παρατηρούμε ότι για την εκτέλεση των εντολών:

- Οι multicore CPUs διαθέτουν πολλαπλούς λογικούς πυρήνες ο καθένας εκ των οποίων έχει πρόσβαση σε SIMD Vector Machines με ένα περιορισμένο εύρος.

- Οι GPUs διαθέτουν ένα πλήθος από Streaming Multiprocessors με ένα πλήθος από CUDA cores που μπορούμε να θεωρήσουμε ομοδοποιημένα ως pipelined 32-wide SIMD Vector Machines.

Μια σημαντική διαφορά έγκειται στο ILP (παραλληλισμός σε επίπεδο εντολών). Σε κάθε warp μιας GPUs έχει ανατεθεί η εκτέλεση μίας εντολής. Για να επιτευχθεί αυτό χρησιμοποιούνται τα Dispatch Units που διαθέτει ο κάθε Warp Scheduler. Το πλήθος των Dispatch Units είναι αυτό που καθορίζει το issue-width (πλήθος των εντολών που δεσμεύονται για εκτέλεση). Αντίστοιχα στους multicore CPUs για κάθε πυρήνα υπάρχει το ειδικό κομμάτι του υλικού για να επιτευχθεί το issue των εντολών.

Αναφορικά με τις προσβάσεις στη μνήμη οι αρχιτεκτονικές διαφορές των δύο αυτών ετερογενών αρχιτεκτονικών οικογενειών είναι σημαντικές. Κάποιες από τις πιο σημαντικές αρχιτεκτονικές διαφορές είναι:

- Οι GPUs δεν διαθέτουν L3 cache όπως οι multicore CPUs. Αυτό δεν πρέπει να μας ξαφνιάζει καθώς σχεδιάστηκαν με σκοπό να παρέχουν μεγάλη απόδοση για παράλληλες εφαρμογές όπου το Operational Intensity είναι υψηλό. Επομένως η σχεδιαστική σύμβαση αυτή στοχεύει στο σενάριο κατά το οποίο πραγματοποιείται μεγάλος υπολογιστικός φόρτος στα δεδομένα.

- Οι GPUs προσφέρουν γρηγορότερη πρόσβαση στην κύρια μνήμη. Για να καλυφθεί η έλλειψη της L3 cache οι GPUs χρησιμοποιούν ευρύτερο κανάλι επικοινωνίας με τη μνήμη και ως εκ τούτου το Memory Bandwidth που πετυχαίνουν είναι πολύ μεγαλύτερο από ότι οι multicore CPUs για την αντίστοιχη πρόσβαση στην DRAM.

- Οι GPUs διαθέτουν cache lines περισσότερων bytes από ότι οι multicore CPUs.

Σε όλο αυτό το πλαίσιο του hardware diversity πρέπει λοιπόν να βρούμε ένα κοινό έδαφος, συγκεκριμένες συνθήκες, συγκεκριμένα μεγέθη έτσι ώστε η σύγκριση των αρχιτεκτονικών που πραγματοποιούμε να έχει βάση και όντως να δίνει πληροφορίες στο οποίο θα μπορέσουμε να συγκρίνουμε αφενός τις ετερογενείς αρχιτεκτονικές και αφετέρου τις υλοποιήσεις κάθε οικογένειας αρχιτεκτονικών. Ο τρόπος με τον οποίο το επιτυγχάνουμε αυτό θα γίνει σαφής στο επόμενο κεφάλαιο όπου παρουσιάζουμε τις μετρικές του μοντέλου που προτείνουμε.

Κεφάλαιο 6

Μετρικές του μοντέλου

Προκειμένου να διαμορφώσουμε ένα μοντέλο που να απαντάει στο ερώτημα που έχουμε θέσει πρέπει να το εκπαιδεύσουμε με το κατάλληλο πλήθος και είδος μετρικών. Εφόσον οι μέθοδοι μηχανικής μάθησης που χρησιμοποιούμε είναι supervised οφείλουμε να έχουμε αντίληψη του τι ακριβώς πραγματεύεται η κάθε μετρική και του πώς μία συγκεκριμένη τιμή της μεταφράζεται σε μια συγκεκριμένη συμπεριφορά. Δεν μπορούμε να παραβλέψουμε ότι δεν έχουμε μία μονοσήμαντη αντιστοίχιση τιμών και συμπεριφορών, ίσα ίσα αρκετά από τα μετρούμενα μεγέθη είναι σε διαλεκτική σχέση μεταξύ τους. Αυτό είναι που καθιστά το πρόβλημα πολύπλοκο και μας αποτρέπει από το να αναζητήσουμε μία αναλυτική λύση, αντάυτού καταφεύγουμε στις μεθόδους. Μια άλλη δυναμική των μετρικών είναι ότι σε ορισμένες περιπτώσεις βασισμένοι στη διαίσθηση και στη λογική μας μπορούμε να καταλάβουμε ποια είναι τα αρχιτεκτονικά χαρακτηριστικά που θα παίξουν καθοριστικό ρόλο στο ποια εκ των συγκρινόμενων αρχιτεκτονικών τελικά θα είναι η επικρατέστερη για μια εφαρμογή.

Στο κεφάλαιο αυτό θα δούμε αναλυτικά τρεις κατηγορίες μετρικών που χρησιμοποιήθηκαν έμμεσα ή άμεσα για την εκπαίδευση του μοντέλου μας. Στην πρώτη κατηγορία συμπεριλαμβάνονται οι μετρικές που παρέχει το MICA οι οποίες με την σειρά τους χωρίζονται σε υποκατηγορίες, θα δούμε αναλυτικά τις υποκατηγορίες αυτές, θα παρουσιάσουμε ποιες από τις υποκατηγορίες είναι χρήσιμες για το μοντέλο μας. Στην κατηγορία αυτή θα δούμε κάποιες ομαδοποιήσεις ορισμένων μεταβλητών που κάνουμε και κάποιες ιδιαίτερες συνθήκες που διαμορφώνουν ορισμένες εκ των μετρικών. Στη δεύτερη κατηγορία θα δούμε ένα σύνολο από coefficients που εισάγουμε και χρησιμοποιούμε έμμεσα για το μοντέλο μας. Σε αυτήν την κατηγορία επιχειρούμε να διαμορφώσουμε ένα σύνολο ποσοτικοποιημένων συγκρίσεων μεταξύ των αρχιτεκτονικών χαρακτηριστικών των ετερογενών αρχιτεκτονικών που συγκρίνουμε. Τέλος, στην τρίτη κατηγορία εισάγουμε ένα σύνολο από fused μετρικές που προκύπτουν από τον συνδυασμό συγκεκριμένων μετρικών της πρώτης κατηγορίας με τις κατάλληλες coefficients της δεύτερης κατηγορίας.

6.1 Microarchitecture-Independent μετρικές

Εδώ θα παρουσιάσουμε αναλυτικά τις κατηγορίες μετρικών που μπορούμε να μετρήσουμε για μία εφαρμογή μέσω του MICA. Θα παραθέσουμε τις συνθήκες υπό τις οποίες γίνονται οι μετρήσεις, θα εξετάσουμε πιθανές παραλλαγές ορισμένων μετρικών ή ακόμα και κάποιες ομαδοποιήσεις. Σε κάθε περίπτωση θα εξηγήσουμε το σκεπτικό κάθε επιχειρούμενης τρο-

ποποίησης. Θα δούμε αναλυτικά ποιες κατηγορίες χρησιμοποιήθηκαν και ποιες όχι για το μοντέλο μας. Σε αυτό το σημείο επισημαίνουμε απλώς ότι τα μετρούμενα μεγέθη δεν είναι κανονικοποιημένα, δεν έχουν δηλαδή τη μορφή ποσοστών, αντίθετα αποτελούν πλήθη. Η κανονικοποίηση των μεγεθών γίνεται σε δεύτερο χρόνο αφού λάβουμε τις τιμές τους και διαμορφώσουμε κατάλληλα τα αντίστοιχα ποσοστά.

6.1.1 Instruction-Level Parallelism (ILP)

Για την εκτίμηση του παραλληλισμού σε επίπεδο εντολών που χαρακτηρίζει την εφαρμογή μας το MICA μας παρέχει τέσσερις μετρικές. Με την υπόθεση ότι η χρησιμοποιούμενη αρχιτεκτονική έχει ιδανικές caches, ιδανικό branch prediction κτλ., το MICA υπολογίζει το IPC (Instructions Per Cycle) για τέσσερα διαφορετικά instruction window sizes (32, 64, 128, 256). Μέσω αυτών των μετρικών μπορούμε να διαπιστώσουμε πόσο καλά μπορεί να εκμεταλλευτεί μια εφαρμογή ένα μεγαλύτερο instruction window size, δηλαδή την αρχιτεκτονική υλοποίηση που επιτρέπει σε μεγαλύτερο πλήθος διαδοχικών εντολών να γίνουν issue ταυτόχρονα. Αυτό που καθορίζει την παρατηρούμενη συμπεριφορά και ως εκ τούτου το πόσο καλά θα κλιμακώσει η εφαρμογή χάρις το αυξημένο instruction window size δεν είναι κάτι άλλο πέραν των data dependencies (εξάρτηση των δεδομένων) μεταξύ διαδοχικών εντολών.

Οι τέσσερις μετρήσεις που λαμβάνουμε αποτυπώνουν το πλήθος των κύκλων που απαιτούνται για την ολοκλήρωση της εφαρμογής ενώ επίσης δίνεται και το πλήθος εντολών που απαρτίζουν την εφαρμογή. Διαιρώντας σε κάθε περίπτωση το πλήθος των εντολών με το αντίστοιχο πλήθος των κύκλων παίρνουμε τελικά τις μετρικές ILP_32, ILP_64, ILP_128, ILP_256. Για να συμπυκνώσουμε την πληροφορία αυτών των μετρικών και να ποσοτικοποιήσουμε την κλιμάκωση ή τη μη κλιμάκωση της εφαρμογής που συνιστά η αύξηση του instruction window size εισάγουμε μία νέα μετρική με το όνομα **ILP_rate**. Το μέγεθος αυτό προκύπτει ως ακολούθως:

$$ILP\_rate = \frac{ILP\_256}{ILP\_64} \tag{6.1}$$

6.1.2 Instruction Mix

Το σύνολο των μετρικών αυτής της κατηγορίας αποσκοπεί στον να εκφράσει την σύσταση της εφαρμογής υπό την οπτική των εντολών. Οι εντολές διαχωρίζονται σε δεκατρείς ομάδες ανάλογα με το είδος τους. Κάποιες από αυτές τις ομάδες πριμοδοτούνται περισσότερο από τις GPUs αρχιτεκτονικές, ενώ κάποιες άλλες πριμοδοτούνται περισσότερο από τους multicore CPUs. Για παράδειγμα ένα υψηλό ποσοστό sse (Streaming SIMD Extensions) εντολών σηματοδοτεί πιθανή εκμετάλλευση της παραλληλίας που αυτές εκφράζουν από τις GPUs. Επισημαίνουμε πως αυτές οι εντολές εκτελούνται παράλληλα από κατάλληλα units του υλικού που αποκαλούνται SIMD Vector Machines. Όπως είδαμε στο προηγούμενο κεφάλαιο το σύνολο των CUDA cores μπορεί να ομαδοποιηθεί και να θεωρηθεί ως ένα σύνολο από 32-wide SIMD Vector Machines. Το εύρος αυτών των units στις αρχιτεκτονικές που εξετάσαμε αλλά και κατά κανόνα είναι μεγαλύτερο στις GPUs από ότι στους multicore CPUs. Στον αντίποδα ένα υψηλό ποσοστό από εντολές τύπου control flow, ενδέχεται να σηματοδοτεί πολύπλοκες συνθήκες και ελέγχους της ροής εκτέλεσης της εφαρμογής. Δεδομένου ότι οι multicore CPUs

διαθέτουν σύνθετους και ακριβείς branch predictors είναι λογικό να περιμένουμε ότι θα διαχειριστούν πιο αποδοτικά αυτές τις εντολές έναντι των GPUs που δεν διαθέτουν αντίστοιχους branch predictors.

Δεδομένου ότι οι x86 δεν είναι load-store αρχιτεκτονικές, οι αναγνώσεις και οι εγγραφές στη μνήμη δεν εμπίπτουν στην ίδια κατηγορία. Αναλυτικά οι τύποι των εντολών που προσμετρώνται μέσω του MICA είναι οι ακόλουθοι:

- **memory read**: αναγνώσεις από τη μνήμη
- **memory write**: εγγραφές στη μνήμη
- **control flow**: εντολές ελέγχου της ροής εκτέλεσης της εφαρμογής
- **arithmetic**: αριθμητικές εντολές απλής ακρίβειας
- **floating point**: εντολές δεκαδικών αριθμών διπλής ακρίβειας
- **stack**: εντολές στοίβας
- **shift**: bitwise πράξεις ολίσθησης
- **string**: εντολές για συμβολοσειρές
- **sse**: εντολές Streaming SIMD Extensions
- **system**: κλήσεις συστήματος
- **nop**: idle εντολές
- **other**: εντολές που δεν εμπίπτουν σε κανέναν από τους παραπάνω τύπους, οι τύποι τους αναγράφονται σε ένα ειδικό αρχείο εξόδου (itypes_other_group_categories.txt) του MICA το οποίο στην περίπτωση των εφαρμογών που εξετάσαμε περιείχε τις εντολές CONVERT, SETCC, XSAVE.

Για να κανονικοποιήσουμε τις παραπάνω μετρήσεις κάθε εφαρμογής τις μετατρέπουμε σε ποσοστά ως προς το συνολικό πλήθος των εντολών.

6.1.3 Branch Predictability

Εδώ το MICA παρέχει κάποιες μετρικές που αφορούν την ακρίβεια συνολικά δώδεκα διαφορετικών branch predictors που στηρίζονται στη μέθοδο prediction-by-partial-match(PPM). Συγκεκριμένα οι branch predictors που εξετάζονται στηρίζονται σε:

- global/local branch history
- shared/separate prediction table(s)
- history length των (4, 8 ή 12bits)

Όπως αναφέραμε προηγουμένως οι GPUs δεν διαθέτουν τους εκλεπτυσμένους branch predictors που έχουν οι multicore CPUs. Επομένως, θεωρούμε ότι η μετρική control flow του προηγούμενου συνόλου μετρικών είναι επαρκής για να διαφοροποιήσουμε τις συγκρινόμενες ετερογενείς αρχιτεκτονικές για μία δεδομένη εφαρμογή. Ως εκ τούτου, δεν συμπεριλαμβάνουμε αυτήν την ομάδα μετρικών στο τελικό μοντέλο μας.

6.1.4 Register Traffic

Αυτό το σύνολο μετρικών αποτελείται από (α) το μέσο πλήθος από register operands των εντολών που εκτελούνται, (β) το μέσο βαθμό χρήσης έκαστου register και (γ) τις πιθανότητες για ένα σύνολο από αποστάσεις μεταξύ της εγγραφής και ανάγνωσης από ένα register.

Αυτό το σύνολο μετρικών μολονότι πλούσιο σε πληροφορία δεν προσθέτει κάτι στην ακρίβεια του μοντέλου. Για να είμαστε ακριβείς δοκιμάσαμε να εκπαιδεύσουμε το μοντέλο με και χωρίς τις κανονικοποιημένες τιμές αυτού του συνόλου. Αυτό που διαπιστώσαμε είναι ότι η ακρίβεια του μοντέλου δε βελτιώθηκε. Πιθανότατα αυτό που συμβαίνει είναι ότι επειδή ακριβώς οι εφαρμογές δεν είναι βελτιστοποιημένες για τις GPU υλοποιήσεις που εξετάζουμε, δε γίνεται πλήρης εκμετάλλευση των δυνατοτήτων του υλικού όσον αφορά τους registers. Συνεπώς, δεν μπορούμε ούτε διαισθητικά αλλά ούτε και σε επίπεδο μοντέλου να αντιστοιχήσουμε τις μετρήσεις αυτές με μία παρατηρούμενη συμπεριφορά.

6.1.5 Data Stream Strides

Σε αυτήν την ομάδα μετρικών εκτίθενται οι αποστάσεις για διαδοχικές προσβάσεις στη μνήμη. Ως απόσταση θεωρούμε το απόλυτο μέγεθος που προκύπτει από τη διαφορά της διεύθυνσης στην κύρια μνήμη της τρέχουσας πρόσβασης με τη διεύθυνση της ακριβώς προηγούμενης πρόσβασης. Άρα πρόκειται για διαφορά διευθύνσεων η οποία υπολογίζεται σε bytes. Έχουμε τέσσερις γενικές κατηγορίες τέτοιων αποστάσεων:

- local load (memory read) strides
- global load (memory read) strides
- local store (memory write) strides
- global store (memory write) strides

Παραπάνω ο όρος local απευθύνεται σε στατικές εντολές ενώ ο όρος global απευθύνεται σε όλες τις εντολές που τελικά καταλήγουν να εκτελεστούν. Ας κάνουμε μια σημαντική διάκριση σε αυτό το σημείο μεταξύ των στατικών και δυναμικών εντολών. Οι στατικές εντολές είναι οι εντολές που συνιστούν το εκτελέσιμο αρχείο, το πλήθος τους είναι γνωστό κατά την παραγωγή του εκτελέσιμου αρχείου, δεν είναι το σύνολο των εντολών που τελικά εκτελούνται από τον υπολογιστικό πόρο που έχουμε στην διάθεσή μας (για παράδειγμα ας σκεφτούμε μια περιοχή του κώδικα στο οποίο μεταβαίνουμε αν πληρείται μία συνθήκη, αν δεν πληρείται τότε δεν εκτελούνται οι εντολές αυτής της περιοχής). Στον αντίποδα έχουμε το σύνολο των δυναμικών εντολών δηλαδή τις εντολές που καταλήγουν να εκτελούνται κατά το runtime της εφαρμογής. Επειδή θέλουμε να μελετήσουμε πώς ακριβώς συμπεριφέρεται η εφαρμογή κατά την εκτέλεση θα αξιοποιήσουμε τις κατηγορίες global load και global store strides που αφορούν τις δυναμικές εντολές. Οι παραπάνω κατηγορίες παρέχουν μετρικές που αντιστοιχούν σε αποστάσεις που εκφράζονται ως δυνάμεις του οχτώ (0, 8 ,64, 512, 4096, 32768, 262144).

Για κάθε μία από τις παραπάνω κατηγορίες έχουμε μετρικές που εκφράζουν την πιθανότητα η εν λόγω πρόσβαση να εμπίπτει στην απόσταση αυτή. Συνολικά λοιπόν οι μετρικές που χρησιμοποιούμε από αυτό το σύνολο είναι:

- mem_read_global_stride_0, mem_read_global_stride_64, mem_read_global_-stride_512, mem_read_global_stride_4096, mem_read_global_stride_32768, mem_-read_global_stride_262144.

- mem_write_global_stride_0, mem_write_global_stride_64, mem_read_write_-stride_512, mem_write_global_stride_4096, mem_read_write_stride_32768, mem_-write_global_stride_262144.

Σε αυτήν την κατηγορία μετρικών πραγματοποιούμε μία ομαδοποίηση η οποία σε συνδυασμό με το κατάλληλο coefficient βοηθά σε μεγαλύτερη ακρίβεια και άρα μια επιβεβαίωση της διαίσθησής μας. Συγκεκριμένα αντικαθιστούμε τις μετρικές mem_read_global_stride_-32768, mem_read_global_stride_262144 με μια νέα μετρική **mrgs32k-∞** η οποία είναι ίση με το άθροισμά τους. Θεωρούμε ότι οι αποστάσεις αυτές οδηγούν όλες τις αρχιτεκτονικές που εξετάζουμε σε cache miss οπότε επιβάλλεται να υπάρξει επικοινωνία με την κύρια μνήμη για να βρεθεί το ζητούμενο block. Σε αυτήν την περίπτωση η αρχιτεκτονική με το μεγαλύτερο Memory Bandwidth υπερισχύει. Στην περίπτωση του Decision Tree Classification η ομαδοποίηση αυτή οδηγεί σε ένα πιο ευσύνοπτο και κατανοητό δέντρο αποφάσεων.

Μια τελευταία σημαντική παρατήρηση η οποία θα φανεί χρήσιμη κατά την ανάλυση του μοντέλου μας είναι ότι οι GPUs διαθέτουν μεγαλύτερο cache line από ότι οι multicore CPUs (64 bytes). Συνεπώς ένα αυξημένο ποσοστό για την μετρική mem_read_global_stride_64 μπορεί να μεταφραστεί ως πιθανό προβάδισμα για τις GPUs καθώς εμπίπτει σε πρόσβαση σε υπάρχον για αυτή cache line, ενώ για τον multicore CPU σημαίνει πρόσβαση σε νέο cache line.

6.1.6 Memory Footprint

Εδώ το MICA παρέχει τέσσερις μετρικές που αφορούν το αποτύπωμα της εφαρμογής στη μνήμη για τις εντολές και για τα δεδομένα τις εφαρμογής. Σε κάθε μία από τις δύο περιπτώσεις γίνεται μια περαιτέρω διάκριση, υπολογίζονται τόσο το πλήθος των blocks (64 bytes) όσο και το πλήθος των σελίδων (4KB) που χρησιμοποιούνται από την εφαρμογή κατά την εκτέλεση.

Αυτή η κατηγορία μετρικών μολονότι χρήσιμη δεν περιλαμβάνεται στα δεδομένα που χρησιμοποιούμε για την εκπαίδευση του μοντέλου για πρακτικούς λόγους. Σε πολλές περιπτώσεις οι εφαρμογές στις οποίες εφαρμόσαμε το MICA δεν ήταν δυνατό να εκτελεστούν σε κάποιο λογικό χρονικό διάστημα οπότε μετά από ένα εύλογο χρόνο τις διακόπταμε. Τα υπόλοιπα σύνολα μετρικών επειδή ακριβώς τα μετατρέψαμε σε ποσοστά αυτή η διακοπή πέρα από ένα ποσοστό σφάλματος δεν εισέφερε κάποιο άλλο πρόβλημα. Στην περίπτωση όμως του memory footprint μας αφορά το απόλυτο μέγεθος και όχι κάποια κανονικοποιημένη μορφή αυτού. Εκεί λοιπόν η διακοπή της εκτέλεσης προκαλούσε ένα πρόβλημα που δεν μπορούμε να παραβλέψουμε.

Προκειμένου να μετρήσουμε με ακρίβεια το memory footprint κάθε εφαρμογής χρησιμοποιήσαμε το valgrind massif tool. Αναζητήσαμε το μεγαλύτερο heap που προκύπτει κατά την εκτέλεση της εφαρμογής και επομένως είχαμε μια καλή εκτίμηση του memory footprint.

6.1.7 Memory Reuse Distances

Το τελευταίο σύνολο μετρικών του MICA είναι και ένα από τα πιο χρήσιμα καθώς βοηθάει να αντιληφθούμε το πώς συμπεριφέρεται η cache κατά την εκτέλεση της εφαρμογής που εξετάζουμε. Εδώ οι αποστάσεις νοούνται ως το πλήθος των εντολών πρόσβασης στη μνήμη που μεσολαβούν έως ότου επαναπραγματοποιηθεί πρόσβαση σε ένα συγκεκριμένο cache block. Πρόκειται για αποστάσεις επαναχρησιμοποίησης. Ας το δούμε όμως αυτό λίγο πιο αναλυτικά. Για κάθε ανάγνωση στη μνήμη καθορίζεται και το αντίστοιχο cache block (64 bytes) το οποίο επιχειρούμε να αναγνώσουμε. Για κάθε τέτοια πρόσβαση το πλήθος των ξεχωριστών cache blocks τα οποία έχουμε αναγνώσει καθορίζεται με τη χρήση μίας LRU (Least Recently Used) στοίβας.

Οι μετρικές αντιστοιχούν έκαστη σε ένα bucket. Ο πρώτος αφορά τα cold references, ο δεύτερος αφορά αποστάσεις του διαστήματος [0, 2), οι επόμενοι αφορούν αποστάσεις επαναχρησιμοποίησης της μορφής $[2^n, 2^{n+1})$ όπου $n = 1, 2, 3, ..., 18$, ο τελευταίος αφορά το διάστημα $[2^{18}, \infty)$. Έχουμε 21 μετρικές. Οι μετρικές είναι λοιπόν της μορφής:

- cold references
- memReuseDistance0-2
- memReuseDistance2-4
- memReuseDistance4-8
- memReuseDistance8-16

  ...

- memReuseDistance32k-64k
- memReuseDistance64k-128k
- memReuseDistance128k-256k
- memReuseDistance256k-∞

Σε αυτό το σύνολο κάνουμε δύο ομαδοποιήσεις από τις οποίες προκύπτουν δύο μετρικές οι **mRD[1k, 32k)** και **mRD[32k, ∞)**. Η πρώτη ομαδοποίηση γίνεται με σκοπό να εξετάσουμε πώς τα διαφορετικά μεγέθη των L2 caches των GPUs επηρεάζουν την τελική απόδοση. Για να καταλάβουμε καλύτερα ας κάνουμε στον εαυτό μας την αφελή ερώτηση ″Πόσα CPU cache lines (64 bytes) χωράνε στην L2 cache κάθε μίας GPU;″, για την Tesla m2050 η απάντηση είναι 12288, για την Tesla K40c η απάντηση είναι 24576 ενώ για την Quadro m4000 είναι 32768 CPU cache lines. Αυτό σημαίνει ότι για μια μεγάλη τιμή της ομαδοποιημένης μετρικής mRD[1k, 32k) που εισάγουμε η Quadro m4000 θα υπερισχύσει των υπόλοιπων δύο, η Tesla K40c θα υπερισχύσει της Tesla m2050. Αυτό συμβαίνει διότι εκεί που η GPU με τη μεγαλύτερη L2 cache θα έχει ένα cache hit, οι άλλες θα αναγκαστούν να αναζητήσουν στην κύρια μνήμη το αντίστοιχο block. Η δεύτερη ομαδοποίηση μπορεί να μεταφραστεί ως σχεδόν βέβαιο cache miss και άρα επακόλουθη πρόσβαση στην κύρια μνήμη κάθε GPU. Εκεί καταλυτικό ρόλο θα παίξει το memory bandwidth. Όλα αυτά θα γίνουν σαφή στο κεφάλαιο των fused μετρικών.

6.2 Hardware Coefficients

Όπως αναδείξαμε προηγουμένως, για να μην αντιμετωπίζουμε με μονολιθικό τρόπο τις ετερογενείς αρχιτεκτονικές που συγκρίνουμε οφείλουμε να θέσουμε ένα κοινό έδαφος στο οποίο θα ποσοτικοποιούνται οι κύριες αρχιτεκτονικές διαφορές. Για να το πετύχουμε αυτό εξ αρχής θα παραβλέψουμε συγκεκριμένες σχεδιαστικές αλλαγές που πραγματοποιούνται από γενιά σε γενιά και θα εστιάσουμε στις διαφορές που φαίνεται να είναι οι πιο κρίσιμες για την απόδοση. Αυτή η αφαιρετική προσέγγιση είναι που καθιστά εφικτή την σύγκριση ανόμοιων στην ουσία αρχιτεκτονικών. Για να είναι όμως και έγκυρη αυτή η σύγκριση πρέπει να γίνει υπό τις κατάλληλες συνθήκες, αυτό θα το δούμε αναλυτικά στο επόμενο υποκεφάλαιο.

Ιδανικά επιθυμούμε να καλύψουμε με τις hardware coefficients όλες τις μορφές του παραλληλισμού που συναντάμε στις εφαρμογές. Οι μορφές αυτές είναι:

- Instruction-Level Parallelism (ILP)
- Thread-Level Parallelism (TLP)
- Data-Level Parallelism (DLP)

Η πρώτη hardware coefficient που προτείνουμε είναι η **coef_IW**. Με αυτή τη μετρική ποσοτικοποιούμε την σύγκριση του συνολικού issue width (IW) που προσφέρει η υπό εξέταση GPU έναντι του υπό εξέταση multicore CPU. Για να το πετύχουμε αυτό αρχικά πολλαπλασιάζουμε το πλήθος των διαθέσιμων Streaming Multiprocessors (SMs) με το πλήθος των αντίστοιχων Warp Dispatch Units που διαθέτει. Υπενθυμίζουμε ότι κάθε Dispatch Unit είναι υπεύθυνο για τη δέσμευση μιας εντολής προς εκτέλεση για ένα δεδομένο warp. Ακολουθούμε την αντίστοιχη λογική για τον multicore CPU, πολλαπλασιάζουμε το πλήθος των πυρήνων με το issue width (IW) ανά πυρήνα που δίνει ο κατασκευαστής. Τελικά, ο λόγος των δύο γινομένων συνιστά τη μετρική μας.

$$coef\_IW = \frac{(number\ of\ SMs) \cdot IW_{GPU}}{(number\ of\ CPU\ cores) \cdot IW_{CPU}} \tag{6.2}$$

Έπειτα, για τις προσβάσεις στην κύρια μνήμη εισάγουμε την μετρική **coef_BW**. Η μετρική αυτή είναι ο λόγος των Memory Bandwidth των δύο ετερογενών αρχιτεκτονικών που συγκρίνουμε.

$$coef\_BW = \frac{Mem\_BW_{GPU}}{Mem\_BW_{CPU}} \tag{6.3}$$

Για να εκφράσουμε μερικώς τις δυνατότητες μιας GPU να εκμεταλλευτεί την παραλληλία που εκφράζουν οι sse εντολές χρησιμοποιούμε τη μετρική **coef_SIMD**. Εδώ συγκρίνουμε ουσιαστικά το εύρος των SIMD Vector Machines των ετερογενών αρχιτεκτονικών.

$$coef\_SIMD = \frac{warp\_size_{GPU}}{vector\_width_{CPU}} \tag{6.4}$$

Προφανώς, η πληροφορία που αποκομίζουμε από αυτόν το λόγο δεν παρέχει κάποια διαφοροποίηση μεταξύ των GPUs μιας και όλες έχουν σταθερό εύρος. Αυτήν την αδυναμία την προσπερνάμε με κάποιες από τις ακόλουθες μετρικές.

Όπως είδαμε στο προηγούμενο κεφάλαιο το διαφορετικό μέγεθος των L2 caches των GPUs μπορεί να οδηγήσει σε διαφορετικές εκβάσεις για την απόδοση μίας εφαρμογής. Για το λόγο αυτό εισάγουμε την μετρική **coef_L2_gpu** όπου βλέπουμε πόσο μεγαλύτερη είναι η L2 cache της τρέχουσας GPU από την μικρότερη L2 cache του συνόλου των διαθέσιμων GPUs.

$$coef\_L2\_gpu = \frac{L2\_size_{testedGPU}}{L2\_size_{reference}} \tag{6.5}$$

Τέλος, για να καλύψουμε το Thread-Level Parallelism (TLP) εισάγουμε τις μετρικές **coef_TLP_min** και **coef_TLP_max**. Η πρώτη εστιάζει στο ελάχιστο πλήθος από δρομολογημένα και υπό εκτέλεση warps που απαιτούνται ώστε να έχουμε πλήρη αξιοποίηση όλων των CUDA cores που διατίθενται σε κάθε SM της GPU. Ας έχουμε υπόψιν το μέγεθος αυτό ως ένα πλήθος από warps που εκτελούνται ταυτόχρονα. Το ελάχιστο πλήθος το συμβολίζουμε ως mifw (minimum in-flight warps). Για την δεύτερη μετρική εστιάζουμε στο μέγιστο δυνατό πλήθος ενεργών warps, δηλαδή δρομολογημένων warps. Οι μετρικές αυτές υπολογίζονται ως ακολούθως:

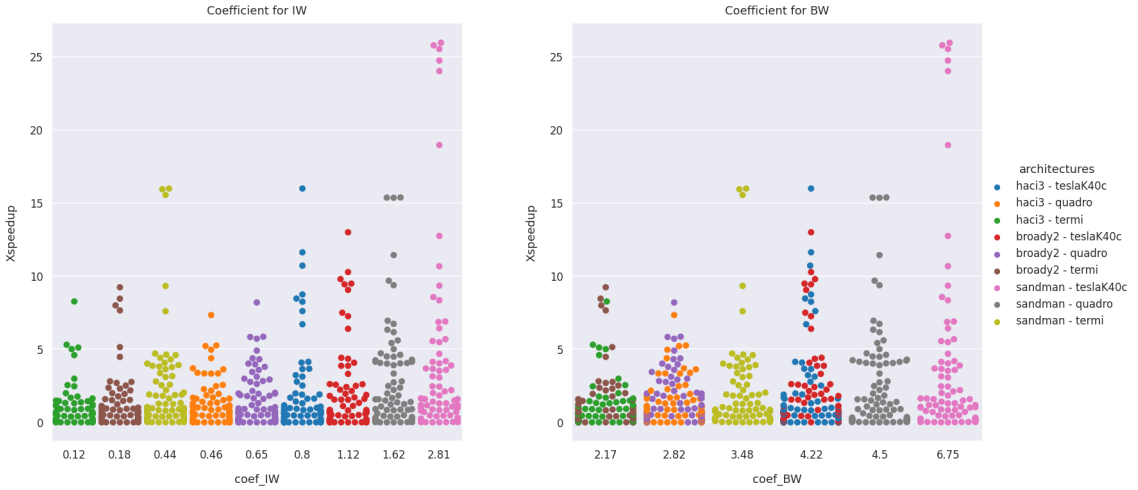
$$coef\_TLP\_min = \frac{(number\ of\ SMs) \cdot mifw}{(number\ of\ CPU\ cores) \cdot (hyperthreading\ degree)} \tag{6.6}$$

$$coef\_TLP\_max = \frac{(number\ of\ SMs) \cdot (max\ number\ of\ active\ warps)}{(number\ of\ CPU\ cores) \cdot (hyperthreading\ degree)} \tag{6.7}$$

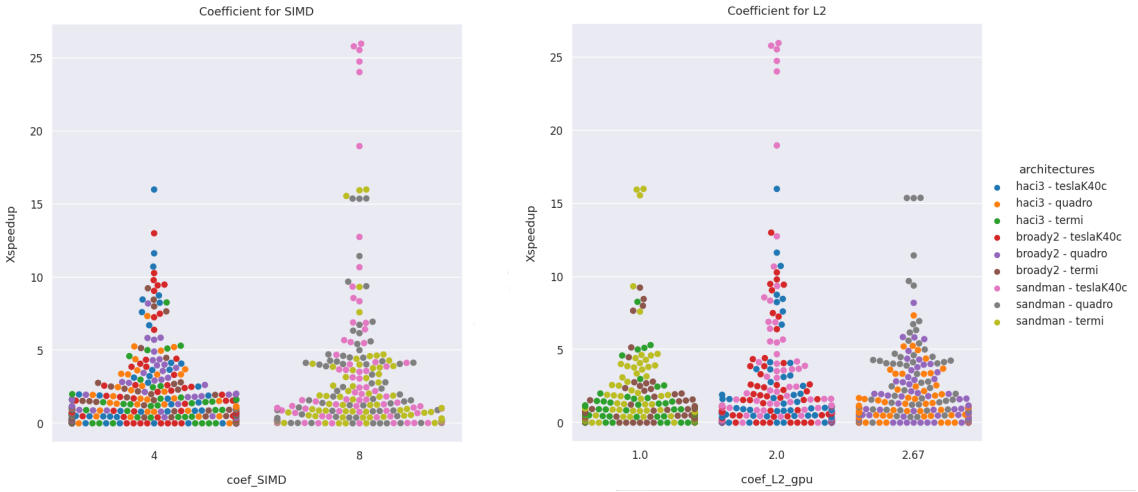
6.3 Αξιολόγηση των Hardware Coefficients

Είναι ιδιαίτερα σημαντικό να καταδείξουμε την χρησιμότητα των μετρικών που εισάγαμε στο προηγούμενο υποκεφάλαιο. Χρειαζόμαστε χειροπιαστά επιχειρήματα που να εξηγούν το πώς η κάθε μία από τις προτεινόμενες hardware coefficients προσφέρει μεγαλύτερη εποπτεία για την επίδραση που έχουν οι αρχιτεκτονικές διαφορές στην τελική απόδοση. Για να πετύχουμε τα παραπάνω κάνουμε χρήση των **Categorical Scatterplots** [32]. Τα διαγράμματα αυτά επιτρέπουν να διαμοιράσουμε τα προβλήματα που εξετάζουμε σε κατηγορίες και να δούμε πώς συμπεριφέρονται ως προς ένα ορισμένο μέγεθος το οποίο υποδηλώνει μια ξεχωριστή κατηγορία. Στην περίπτωσή μας σχεδιάζουμε ένα διάγραμμα για κάθε μία από τις hardware coefficients που προτείναμε, διακρίνουμε τις αρχιτεκτονικές η σύγκριση των οποίων αποφέρει την τιμή που συνιστά την κάθε κατηγορία για την υπό εξέταση μετρική. Μελετούμε την συμπεριφορά βλέποντας πώς οι διαφορετικές της μετρικής (δηλαδή οι διαφορετικές κατηγορίες) οδηγούν σε διαφορετική απόδοση.

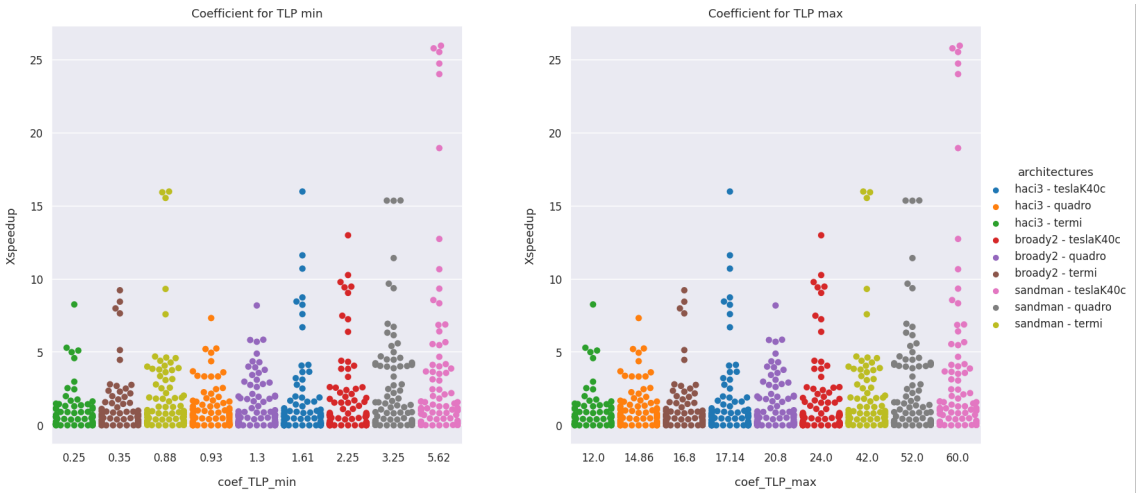
Πρέπει να επισημάνουμε για άλλη μία φορά ότι τα μεγέθη που παρέχουμε στο μοντέλο για την εκπαίδευσή του είναι σε διαλεκτική σχέση μεταξύ τους. Αυτό σημαίνει ότι και από τα κάτωθι διαγράμματα δεν αναμένουμε να δούμε κάποια καθολική συμπεριφορά. Αυτό που επιδιώκουμε να επικυρώσουμε είναι ότι έστω κατά τόπους επαληθεύονται τα λογικά πορίσματα και η διαίσθησή μας. Μια ανοδική τάση στο σύνολο των δειγμάτων για μεγαλύτερες τιμές της hardware coefficient που εξετάζουμε είναι μια καλή ένδειξη για το πόσο ωφέλιμη είναι η μετρική που προτείνουμε σε κάθε περίπτωση. Ας δούμε τα διαγράμματα:



Σχήμα 6.1: *Categorical Scatterplots για τις coefficients που αφορούν το issue-width και το memory bandwidth*



Σχήμα 6.2: *Categorical Scatterplots για τις coefficients που αφορούν το SIMD-width και το L2 cache size των GPUs*



Σχήμα 6.3: *Categorical Scatterplots για τις coefficients που αφορούν το TLP*

Στα παραπάνω διαγράμματα βλέπουμε την μετρική **Xspeedup** την οποία ορίζουμε ως το σχετικό speedup που προσφέρει για μια εφαρμογή μία συγκεκριμένη GPU ως προς τη γρηγορότερη εκτέλεση του συγκρινόμενου multicore CPU. Για να είμαστε πιο σαφείς, η γρηγορότερη εκτέλεση μιας παράλληλης εφαρμογής σε έναν multicore CPU συνήθως επιτυγχάνεται για ένα πλήθος από OpenMP threads. Επειδή η σύγκριση με την σειριακή εκτέλεση της εφαρμογής δεν είναι αρκετή για το κεντρικό ερώτημα που επιχειρούμε να απαντήσουμε με την παρούσα διπλωματική, επεκτείναμε την έννοια του speedup ώστε να εκφράζει την σχέση της γρηγορότερης GPU-υλοποίησης, με την γρηγορότερη CPU-υλοποίηση (η οποία κατά κανόνα είναι πολυνηματική). Η νέα αυτή μετρική είναι ο λόγος του χρόνου της γρηγορότερης εκτέλεσης της εφαρμογής στο multicore CPU προς τον αντίστοιχο χρόνο για την εκτέλεση στην GPU. Να επισημάνουμε επίσης ότι κάθε κουκίδα αναπαριστά την σύγκριση μίας εφαρμογής σε δύο δεδομένες ετερογενείς αρχιτεκτονικές. Επίσης, οι ονομασίες που χρησιμοποιούνται για τις κουκίδες αναπαριστούν:

- haci3: κόμβος για τους multicore CPUs της γενιάς Haswell.
- broady2: κόμβος για τους multicore CPUs της γενιάς Broadwell.
- sandman: κόμβος για τους multicore CPUs της γενιάς Sandy Bridge.
- Tesla K40c: GPU Tesla K40c της γενιάς Kepler.
- quadro: GPU Quadro m4000 της γενιάς Maxwell.
- termi: κόμβος για τη GPU Tesla m2050 της γενιάς Fermi.

Ας ξεκινήσουμε από το Σχήμα 6.1 με το διάγραμμα για το $coef\_IW$. Όσον αφορά την αρχιτεκτονική οικογένεια Sandy Bridge βλέπουμε ότι το αυξανώμενο issue-width των GPUs όντως αντικατοπτρίζεται σε μια αισθητή αύξηση του Xspeedup. Το ίδιο παρατηρούμε πώς συμβαίνει τόσο για την αρχιτεκτονική οικογένεια Haswell όσο και για την Broadwell. Σημαντικό είναι να έχουμε υπόψιν ότι δεν εξετάζουμε τις κορυφές των στηλών που φαίνονται στο διάγραμμα, αλλά την ευρύτερη τάση που ακολουθούν οι κουκίδες της κάθε στήλης. Στο διάγραμμα του ίδιου σχήματος για το $coef\_BW$, όσον αφορά την σύγκριση μίας CPU αρχιτεκτονικής με όλες τις διαθέσιμες GPU αρχιτεκτονικές, παρατηρούμε ότι όσο μεγαλύτερη είναι η τιμή του $coef\_BW$ (και συνεπώς το Memory Bandwidth της GPU που εξετάζουμε) τόσο μεγαλύτερη είναι και η κατακόρυφη διάχυση των κουκίδων στην αντίστοιχη στήλη. Έπειτα στο σύνολο των συγκρίσεων των ετερογενών αρχιτεκτονικών βλέπουμε μια γενικευμένη τάση αύξησης του επιτεύξιμου Xspeedup όσο αυξάνεται η εξεταζόμενη μετρική. Συνεπώς, οι δύο αυτές hardware coefficients έχουν την προοπτική να μας φανούν ιδιαιτέρως χρήσιμες για το μοντέλο μας.

Στο Σχήμα 6.2 έχουμε τα διαγράμματα για τις hardware coefficients $coef\_SIMD$ και $coef\_L2\_gpu$. Στο πρώτο διάγραμμα βλέπουμε ότι το μειωμένο εύρος των SIMD Vector Machines της γενιάς Sandy Bridge επιτρέπει στις GPUs να εκμεταλλευτούν καλύτερα το σύνολο των sse εντολών. Ως εκ τούτου η δεύτερη στήλη διαθέτει πιο διακεχυμένες κουκίδες στον κατακόρυφο άξονα, δηλαδή μεγαλύτερο Xspeedup. Για την $coef\_L2\_gpu$ η πληροφορία δεν είναι τόσο διαυγής, δεν παρατηρούμε μία αύξηση του $Xspeedup$ ανάλογη της αύξησης του

μεγέθους της L2 cache των GPUs. Αυτό οφείλεται σε δύο παράγοντες. Αφενός δεν αποτελεί καταλυτικής σημασίας αρχιτεκτονική διαφορά για το σύνολο των εφαρμογών που εξετάσαμε, αφετέρου η μικρότερη L2 cache της Tesla K40c αντισταθμίζεται από την ταχύτερη πρόσβαση στη μνήμη. Βλέπουμε ωστόσο ότι η μετρική αυτή οδηγεί σε διαφοροποίηση της Tesla m2050 από τις άλλες δύο GPUs οπότε θα την αξιοποιήσουμε.

Τέλος, στο Σχήμα 6.3 εξετάζουμε τις μετρικές που αφορούν τις δυνατότητες εκμετάλλευσης του παραλληλισμού TLP. Εδώ τα αποτελέσματα είναι αρκετά ξεκάθαρα, βλέπουμε ότι η αύξηση της τιμής και των δύο μετρικών οδηγεί (έστω και κατά τμήματα) σε αύξηση του επιτεύξιμου Xspeedup.

6.4 Fused μετρικές

Στην ενότητα αυτή παρουσιάζουμε αναλυτικά ένα σύνολο μετρικών που αποκαλούμε fused μετρικές. Οι μετρικές αυτές έχουν προκύψει ως γινόμενα των ομαδοποιημένων ή παράγωγων μετρικών που παρουσιάσαμε στην ενότητα 6.1 με τις κατάλληλες hardware coefficients της ενότητας 6.2. Μέσω του πολλαπλασιασμού καταφέρνουμε να δώσουμε στο μοντέλο νέες μετρικές οι οποίες συνεκτιμούν μία συνθήκη και την αντίστοιχη αρχιτεκτονική σύγκριση για την συνθήκη αυτή. Επομένως οι μετρικές αυτές εκφράζουν την συσχέτιση μιας παρατηρούμενης συμπεριφοράς με ένα συγκεκριμένο αρχιτεκτονικό χαρακτηριστικό ανάλογα με το οποίο αναμένουμε να επικρατήσει η μία αρχιτεκτονική έναντι της άλλης. Αυτό θα γίνει πιο σαφές στη συνέχεια.

Οι fused μετρικές που εισάγουμε στο μοντέλο μας είναι οι ακόλουθες:

- ILP_{fused}
- $mrgs_{fused}$
- $mRD\infty_{fused}$
- $mRD[1k, 32k)_{fused}$
- SSE_{fusedA}
- SSE_{fusedB}

Μέσω του ILP_{fused} ποσοτικοποιούμε την υπεροχή (ή μη υπεροχή) μιας δεδομένης GPU έναντι ενός δεδομένου multicore CPU αναφορικά με την μετρική ILP_rate. Εκφράζουμε δηλαδή το πόσο καλά μπορεί να εκμεταλλευτεί το αυξημένο instruction window size. Στην περίπτωση που έχουμε μια μεγάλη τιμή σε κάθε έναν από τους δύο παράγοντες προκύπτει ένα διακριτά μεγάλο γινόμενο (συγκριτικά με τις τιμές αυτής της μετρικής για άλλες εφαρμογές) που σηματοδοτεί για το μοντέλο μας καλή εκμετάλλευση του συγκεκριμένου είδους παραλληλισμού από την GPU. Αναλυτικά το μέγεθος προκύπτει ως εξής:

$$ILP_{fused} = (coef\_IW) \cdot (ILP\_rate) \tag{6.8}$$

Οι fused μετρικές οι οποίες αφορούν προσβάσεις στην κύρια μνήμη είναι οι $mrgs_{fused}$, $mRD\infty_{fused}$. Ανάλογα με την λογική που εκθέσαμε παραπάνω υπολογίζουμε τις τιμές τους

όπως φαίνεται παρακάτω. Ασφαλώς το κατάλληλο hardware coefficient σε αυτήν την περίπτωση είναι το $coef\_BW$.

$$mrgs_{fused} = (coef\_BW) \cdot (mrgs[32k, \infty)) \tag{6.9}$$

$$mRD\infty = (coef\_BW) \cdot (mRD[32k, \infty)) \tag{6.10}$$

Για την διαφοροποίηση των L2 cache των GPUs εισάγουμε την μετρική $mRD[1k, 32k)_{fused}$:

$$mRD[1k, 32k)_{fused} = (coef\_L2\_gpu) \cdot (mRD[1k, 32k)) \tag{6.11}$$

Τέλος για να μελετήσουμε τις sse εντολές ακολουθούμε δύο προσεγγίσεις:

$$SSE_{fusedA} = (coef\_TLP\_min) \cdot (coef\_SIMD) \cdot (sse) \tag{6.12}$$

$$SSE_{fusedB} = (coef\_TLP\_max) \cdot (coef\_SIMD) \cdot (sse) \tag{6.13}$$

Συνοψίζοντας οι μετρικές βάσει των οποίων εκπαιδεύουμε το μοντέλο είναι:

- Memory Footprint
- memory read, memory write, control flow, arithmetic, floating point, stack, shift, string, sse, system, nop, other
- ILP rate
- cold references, memReuseDist0-2, memReuseDist2-4, ..., memReuseDist512-1k, $mRD[1k, 32k)$, $mRD[32k, \infty)$
- mem_read_global_stride[8, 64, 512, 4096], $mrgs[32k, \infty)$
- mem_write_global_stride[8, 64, 512, 4096, 32768, 262144]
- ILP_{fused}
- $mrgs_{fused}$
- $mRD\infty_{fused}$
- $mRD[1k, 32k)_{fused}$
- SSE_{fusedA}
- SSE_{fusedB}

Κεφάλαιο 7

Μεθοδολογία

Στο κεφάλαιο αυτό παρουσιάζουμε αναλυτικά τη μέθοδο που ακολουθήσαμε για να δημιουργήσουμε το μοντέλο μας[33]. Όπως είπαμε με τη διπλωματική μας εργασία επιχειρούμε να δώσουμε απαντήσεις για το ποια εκ των συγκρινόμενων ετερογενών αρχιτεκτονικών ενδείκνυται για μια δεδομένη εφαρμογή με γνώμονα αφενός την απόδοση και αφετέρου την κατανάλωση ισχύος. Αν και οι δύο άξονες βάσει τον οποίων παρέχουμε την αντίστοιχη πρόβλεψη είναι διακριτοί μεταξύ τους, η μεθοδολογία που ακολουθούμε είναι παρεμφερής. Έτσι στην πραγματικότητα καταλήγουμε σε δύο μοντέλα το πρώτο για την απόδοση το δεύτερο για την κατανάλωση ισχύος. Στην συνέχεια θα παραθέσουμε την κεντρική μεθοδολογία που ακολουθήσαμε για κάθε μία από τις δύο περιπτώσεις.

Ας δούμε αναλυτικά τα βήματα της μεθοδολογίας που ακολουθήσαμε για να διαμορφώσουμε τα μοντέλα μας:

1. Συλλογή χρονικών δεδομένων και συλλογή δεδομένων για την κατανάλωση ισχύος
2. Συλλογή μετρήσεων για τις microarchitecture-independent μετρικές
3. Πρώτη φάση προεπεξεργασίας των δεδομένων για την εκπαίδευση του μοντέλου
4. Δεύτερη φάση προεπεξεργασίας των δεδομένων
5. Εκπαίδευση του μοντέλου με επιλογή των κατάλληλων classifiers
6. Πρώτα αποτελέσματα και δυνατότητα βελτίωσης της ακρίβειας
7. Τελικές προβλέψεις των μοντέλων

7.1 Συλλογή δεδομένων για τον χρόνο εκτέλεσης και την κατανάλωση ισχύος

Εδώ θα παρουσιάσουμε αναλυτικά το πρώτο βήμα για να διαμορφώσουμε το μοντέλο που αφορά την πρόβλεψη της σχετικής επιτάχυνσης που προσφέρει μία συγκεκριμένη GPU έναντι ενός συγκεκριμένου multicore CPU για μια δεδομένη εφαρμογή. Στην κατηγορία multicore CPU εμπίπτουν και οι dual socket υλοποιήσεις όπου έχουν δύο multicore CPU chips στον ίδιο υπολογιστικό κόμβο και στις περιπτώσεις που εξετάσαμε στην ίδια μητρική. Αξίζει να διευκρινίσουμε ότι ο όρος "σχετική επιτάχυνση" δεν είναι κάτι άλλο πέραν του σχετικού speedup

(Xspeedup) που ορίσαμε και προηγουμένως στο Κεφάλαιο 6. Πρόκειται για το λόγο του χρόνου της γρηγορότερης εκτέλεσης της εφαρμογής στο multicore CPU ως προς το χρόνο της γρηγορότερης εκτέλεσης της εφαρμογής στην GPU. Αυτό το μέγεθος προσφέρει μια αμεσότερη και ποσοτική σύγκριση της απόδοσης που επιτυγχάνουν οι ετερογενείς αρχιτεκτονικές που συγκρίνουμε για την εκάστοτε εφαρμογή.

7.1.1 Μέτρηση του χρόνου εκτέλεσης

Το πρώτο βήμα είναι να συγκεντρώσουμε τις μετρήσεις χρόνου για τις εφαρμογές της σουίτας Rodinia για κάθε διαφορετικό dataset που χρησιμοποιούμε ως είσοδο της κάθε εφαρμογής. Αυτό το κάνουμε για το σύνολο των αρχιτεκτονικών που εξετάζουμε τις οποίες παραθέσαμε αναλυτικά στο Κεφάλαιο 5. Από το σύνολο των εφαρμογών που παρέχει η σουίτα Rodinia και τις οποίες παραθέσαμε στον Πίνακα 3.1 εστιάσαμε στις εφαρμογές που έχουν υλοποιηθεί τόσο σε OpenMP (OMP) όσο και σε CUDA. Στον παρακάτω πίνακα παραθέτουμε αναλυτικά τις εφαρμογές και το πλήθος διαφορετικών datasets που εξετάσαμε για κάθε μία από αυτές. Επιπρόσθετα εφαρμογές οι οποίες έχουν διαφορετικές αλγοριθμικές υλοποιήσεις έχουν εξεταστεί κανονικά και τις αναγράφουμε αναλυτικά παρακάτω.

Η αρίθμηση σε κάποιες από τις παραπάνω εφαρμογές υποδηλώνει διαφορετικές αλγοριθμικές εκδοχές της αντίστοιχης εφαρμογής. Οι εκδοχές αυτές συναντώνται είτε με ρητά διαφορετικά εκτελέσιμα (και κώδικες) ή με διαφοροποιήσεις των τιμών εισόδου συγκεκριμένων παραμέτρων που καθορίζουν την ροή εκτέλεσης των εφαρμογών. Συνολικά έχουμε λάβει δεδομένα για 93 ζεύγη εφαρμογών-datasets για κάθε μία από τις υπό εξέταση αρχιτεκτονικές. Λόγω της περιορισμένη μνήμης της Tesla m2050 αλλά και της μη συμβατότητας της έκδοσης της CUDA στην κάρτα αυτή δεν ήταν εφικτό να λάβουμε μετρήσεις χρόνου για τα δύο μεγαλύτερα datasets της εφαρμογής Breadth-First Search και για όλα τα ζεύγη της εφαρμογής Leukocyte. Κρίνουμε όμως ότι το σύνολο των δεδομένων που συλλέξαμε είναι επαρκές για τον σκοπό μας.

Για τις χρονικές μετρήσεις στους multicore CPUs χρησιμοποιήσαμε συναρτήσεις όπως η $omp\_get\_wtime()$ της βιβλιοθήκης <omp.h> [34] και η $gettimeoftheday()$ της βιβλιοθήκης <sys/time.h> [35]. Για τις αντίστοιχες GPU υλοποιήσεις χρησιμοποιήσαμε κατά κανόνα την συνάρτηση $gettimeoftheday()$ της βιβλιοθήκης <sys/time.h>. Οι κλήσεις των συναρτήσεων αυτών έγιναν πριν και μετά το παράλληλο τμήμα του κώδικα (kernel) της εφαρμογής, τα αποτελέσματα αποθηκεύτηκαν στις κατάλληλες μεταβλητές και πριν το τέλος της κάθε εφαρμογής οι μεταβλητές αυτές χρησιμοποιήθηκαν για να υπολογίσουμε και να τυπώσουμε στην έξοδο τον χρόνο της εκτέλεσης. Για περαιτέρω διερεύνηση σε πολλές από τις CUDA υλοποιήσεις υπολογίσαμε ξεχωριστά τους χρόνους που απαιτούνταν για τις δεσμεύσεις των απαραίτητων buffers, την αντιγραφή των δεδομένων εισόδου και των δεδομένων εξόδου του kernel και τέλος της απελευθέρωσης των διευθύνσεων των buffers. Αυτό το σύνολο λειτουργιών δεν αντιστοιχεί άμεσα στην παράλληλη εκτέλεση της εφαρμογής στην GPU είναι όμως ένα χρονικό overhead που δεν μπορούμε να παραβλέψουμε, επομένως το λαμβάνουμε υπόψιν στον συνολικό χρόνο εκτέλεσης του παράλληλου τμήματος του κώδικα. Απλουστευμένα η διαδικασία που περιγράφουμε παραπάνω υλοποιείται όπως φαίνεται στους ψευδοκώδικες στο Παράρτημα Α΄.

Αξίζει να επισημάνουμε ότι για τις εφαρμογές στις οποίες τα εξεταζόμενα datasets οδη-

Πίνακας 7.1: *Rodinia Benchmarks που χρησιμοποιήθηκαν για το μοντέλο απόδοσης*

Applications	No. of Datasets
B+Tree	3
Back Propagation	5
Breadth-First Search	3
CFD Solver 1	3
CFD Solver 2	3
CFD Solver 3	3
CFD Solver 4	3
Heart Wall	4
Hotspot	3
Hotspot3D	3
k-Nearest Neighbors	5
Kmeans	4
LavaMD2	4
Leukocyte	4
LU Decomposition	4
Myocyte 1	3
Myocyte 2	3
Needleman-Wunsch	6
Particle Filter	5
PathFinder	4
SRAD 1	4
SRAD 2	5
SRAD 3	5
Streamcluster	4

γούσαν σε αμελητέους χρόνους εκτέλεσης τροποποιήσαμε τους κώδικες ώστε το παράλληλο τμήμα να εκτελείται επαναληπτικά. Έπειτα ανάλογα με το πλήθος των επαναλήψεων υπολογίζουμε τον μέσο όρο και έχουμε τον αντίστοιχο χρόνο εκτέλεσης. Αυτή η διαδικασία επιτρέπει η μέτρησή μας να μην είναι επιρρεπής σε σφάλματα. Για τα ζεύγη που ο χρόνος εκτέλεσης ήταν μεγάλος δεν ακολουθήθηκε η ίδια διαδικασία. Αρκεστήκαμε στο να επικυρώσουμε εμπειρικά ότι δεν είχαμε μεγάλες χρονικές αποκλίσεις από εκτέλεση σε εκτέλεση.

Οι χρόνοι εκτέλεσης χρησιμοποιούνται στο επόμενο πρώτο βήμα της προεπεξεργασίας, στηριζόμαστε σε αυτές τις μετρήσεις για να υπολογίσουμε το σχετικό speedup βάσει του οποίου γίνεται τελικώς το classification. Τα ζεύγη για τα οποία πήραμε αυτές τις χρονικές μετρήσεις είναι και τα ζεύγη για τα οποία στο επόμενο βήμα θα συλλέξουμε μετρήσεις για τις microarchitecture-independent μετρικές μέσω του MICA. Επιπρόσθετα, η συλλογή των χρονικών μετρήσεων είναι απαραίτητη και για τα δύο μοντέλα. Αυτό θα επεξηγηθεί επαρκώς στην συνέχεια. Στο Παράρτημα Α΄ παρουσιάζονται ψευδοκώδικες για τη μέτρηση του χρόνου εκτέλεσης τόσο σε OpenMP όσο και σε CUDA.

7.1.2 Μέτρηση της κατανάλωσης ισχύος

Για το μοντέλο πρόβλεψης της επωφελέστερης αρχιτεκτονικής για την επίτευξη της μικρότερης κατανάλωσης ισχύος οι εφαρμογές για οποίες συλλέξαμε μετρήσεις είναι αυτές που αναγράφονται στον Πίνακα 7.2.

Πίνακας 7.2: *Rodinia Benchmarks που χρησιμοποιήθηκαν για το μοντέλο κατανάλωσης ισχύος*

Applications	No. of Datasets
B+Tree	3
Back Propagation	5
Breadth-First Search	3
CFD Solver 1	3
CFD Solver 2	3
CFD Solver 3	3
CFD Solver 4	3
Heart Wall	4
Hotspot	3
Hotspot3D	3
Kmeans	4
LavaMD2	4
LU Decomposition	3
Needleman-Wunsch	6
PathFinder	4
SRAD 1	4
SRAD 2	5
SRAD 3	5
Streamcluster	4

Αυτό το βήμα πραγματοποιείται μόνο για το μοντέλο που αφορά την πρόβλεψη για το ποια εκ των ετερογενών αρχιτεκτονικών είναι ενεργειακά η πιο ωφέλιμη για την εφαρμογή που εξετάζουμε. Όπως αναλύσαμε Κεφάλαιο 3 για τη μέτρηση της κατανάλωσης ισχύος έχουμε στη διάθεσή μας το εργαλείο PAPI για τους multicore CPUs και τη βιβλιοθήκη <nvml.h> για τη μέτρηση της ισχύος στις GPUs.

Θα ξεκινήσουμε με την **παρουσίαση των μετρήσεων της κατανάλωσης ισχύος στου multicore CPUs**. Προηγουμένως είδαμε ότι κάθε επεξεργαστής Intel από τη γενιά Sandy Bridge και έπειτα παρέχει ένα σύνολο από RAPL counters οι οποίοι χρησιμοποιούνται για την επίβλεψη αλλά και για την ρύθμιση της ενεργειακής συμπεριφοράς του multicore CPU στον οποίον αντιστοιχούν. Για να πάρουμε το σύνολο των διαθέσιμων counters μπορούμε να εκτελέσουμε στην τερματική κονσόλα του αντίστοιχου υπολογιστικού κόμβου το εκτελέσιμο *papi_native_available* με το όρισμα $-i$ *rapl*. Στην περίπτωση των κόμβων για τις αρχιτεκτονικές οικογένειες Haswell και Broadwell οι RAPL counters στους οποίους θα εστιάσουμε είναι αυτοί που αναγράφονται στον Πίνακα 7.3. Για την αρχιτεκτονική Sandy Bridge που είχαμε στη διάθεσή μας αντιμετωπίσαμε κάποια τεχνικά προβλήματα όσον αφορά την ανάγνω-

ση των μετρήσεων, συγκεκριμένα τα μετρούμενα μεγέθη ξεπέρναγαν το θεωρητικό άνω όριο TDP του κατασκευαστή. Παρά τις προσπάθειές μας δεν κατορθώσαμε να καταλήξουμε σε κάποια λύση, οπότε δεν θα συμπεριλάβουμε το κόμβο για Sandy Bridge στο μοντέλο μας το αφήνουμε ως εκκρεμότητα που ιδανικά θα καλύψουμε σε μελλοντικές επεκτάσεις.

Πίνακας 7.3: *RAPL counters που αξιοποιήσαμε για τις Haswell και Broadwell*

Architectural Family	RAPL counters (in Joules)
Haswell	PACKAGE_ENERGY:PACKAGE0, PACKAGE_ENERGY:PACKAGE1, DRAM_ENERGY:PACKAGE0, DRAM_ENERGY:PACKAGE1
Broadwell	PACKAGE_ENERGY:PACKAGE0, PACKAGE_ENERGY:PACKAGE1, DRAM_ENERGY:PACKAGE0, DRAM_ENERGY:PACKAGE1

Με τον όρο ″PACKAGE_ENERGY″ εννοούμε την ενέργεια που καταναλώθηκε κατά την εκτέλεση της εφαρμογής από ολόκληρο το chip του multicore CPU, με την αρίθμηση υποδεικνύεται σε ποιον από τους δύο διαθέσιμους multicore CPUs αντιστοιχεί το μετρούμενο μέγεθος. Με τον όρο ″DRAM_ENERGY″ εννοούμε την ενέργεια που καταναλώθηκε από την κύρια μνήμη για αυτήν την εφαρμογή. Εκτός από αυτές τις μετρικές συχνά παρέχονται και μετρικές που αφορούν την μέγιστη ή ελάχιστη ισχύ που παρατηρείται κατά την εκτέλεση της εφαρμογής, επειδή αυτές οι μετρήσεις είναι επιρρεπείς σε σφάλματα εμείς κατευθυνόμαστε προς τον υπολογισμό της μέσης ισχύος. Για να το πετύχουμε αυτό κατά την εκτέλεση της εφαρμογής με τις κατάλληλες συναρτήσεις του PAPI υπολογίζουμε και τον αντίστοιχο χρόνο εκτέλεσης. Προσοχή, επειδή το PAPI προκαλεί με την σειρά του μια σχετική καθυστέρηση της εκτέλεσης, όπως διαπιστώσαμε εμπειρικά, επιλέγουμε να διαιρέσουμε την συνολική ενέργεια που εκφράζουν τα παραπάνω μεγέθη με τον αντίστοιχο χρόνο. Έτσι προκύπτει η μέση ισχύς για την συγκεκριμένη εφαρμογή. Το μέγεθος αυτό είναι χρήσιμο όπως θα φανεί στο πρώτο στάδιο της προεπεξεργασίας των δεδομένων εισόδου για το μοντέλο, περισσότερα όμως για αυτό θα δούμε αναλυτικά στην συνέχεια.

Εξετάσαμε επίσης το ποσοστό κατανάλωσης ισχύος που επιφέρει η χρήση αυτών των μετρητών και διαπιστώσαμε ότι ήταν αμελητέα, έτσι παραλείπουμε να επεκταθούμε περαιτέρω σε αυτήν την πτυχή του PAPI. Είναι εξαιρετικά κρίσιμο να διευκρινήσουμε πώς για τα ζεύγη εφαρμογών-dataset όπου ο χρόνος εκτέλεσης ήταν εξαιρετικά μικρός, επιβάλαμε μέσω της προσθήκης βρόχων (loops) το παράλληλο τμήμα του κώδικα να εκτελεστεί αρκετές τάξεις μεγέθους παραπάνω και τελικά υπολογίσαμε το μέσο όρο της ισχύος ώστε να μην έχουμε μετρήσεις ευαίσθητες σε σφάλμα. Τέλος, όπως εξηγήσαμε ενδελεχώς κατά την παρουσίαση του PAPI, μια κεντρικής σημασίας δυνατότητα που μας παρέχει αυτό το εργαλείο είναι να ορίσουμε ένα σύνολο από γεγονότα για τα οποία θέλουμε να συλλέξουμε μετρήσεις. Το σύνολο αυτών τον γεγονότων αποκαλείται Event Set. Στο Παράρτημα Α′ παραθέτουμε ένα ψευδοκώδικα για να καταλάβουμε ακριβώς πώς γίνεται η μέτρηση των RAPL counters για κάθε εφαρμογή.

Για την **μελέτη της κατανάλωσης ισχύος στις GPUs** χρησιμοποιούμε κατάλλη-

λα την βιβλιοθήκη <nvml.h>. Η κεντρική ιδέα είναι να δημιουργήσουμε ένα API η χρήση του οποίου θα συνεπάγεται την δημιουργία και εκτέλεση ενός POSIX thread το οποίο πραγματοποιεί δειγματοληψία και καταγράφει την μετρούμενη τιμή της ισχύος για το συγκεκριμένο βήμα της δειγματοληψίας. Το συγκεκριμένο API θα έχει και μια κλήση που θα σηματοδοτεί τον τερματισμό της δειγματοληψίας. Όποιος επιθυμεί να δει αναλυτικά την υλοποίηση μπορεί να αναζητήσει αναλυτικά το σύνδεσμο στο GitLab όπου έχουμε παραθέσει όλες τις χρησιμοποιούμενες υλοποιήσεις αναλυτικά [33]. Με την Tesla m2050 αντιμετωπίσαμε το πρόβλημα ότι η version της CUDA η οποία ήταν διαθέσιμη στον κόμβο δεν υποστήριζε τη βιβλιοθήκη <nvml.h>, στο Παράρτημα Γ′ εξηγούμε την εναλλακτική προσέγγιση που ακολουθήσαμε για να εμπλουτίσουμε το μοντέλο με βάση την εκτιμώμενη τιμή της κατανάλωσης ισχύος της GPU αυτής για κάθε εφαρμογή.

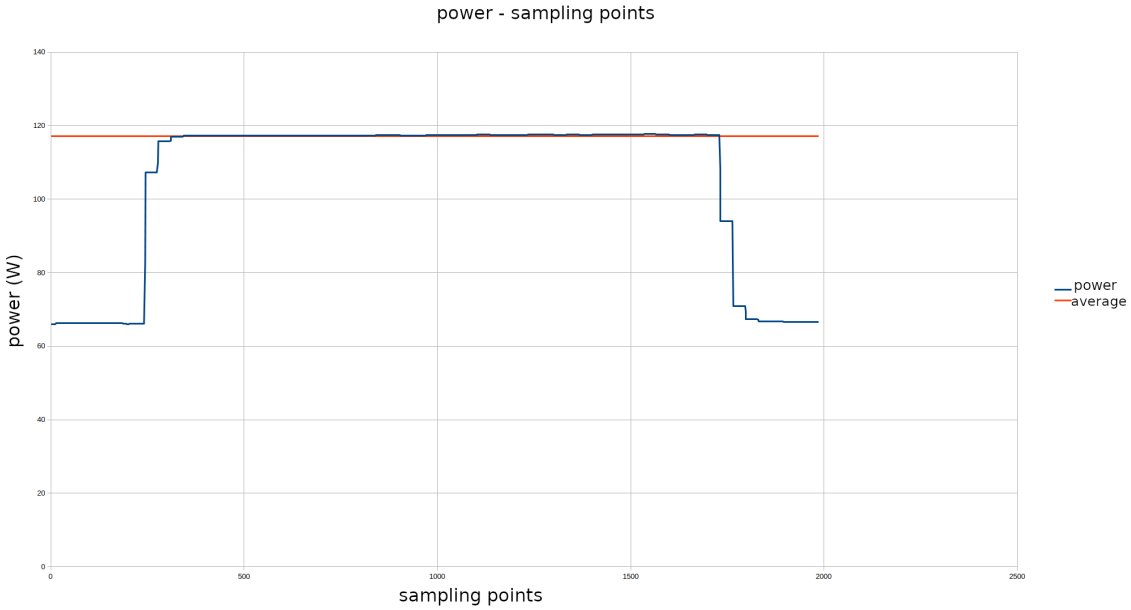
- **nvmlAPIRun():**
  Κατά την εκτέλεση της εφαρμογής η συνάρτηση αυτή καλείται ακριβώς πριν αρχίσουν οι πρώτες λειτουργίες που αφορούν την προετοιμασία για την εκτέλεση του CUDA kernel. Αρχικά, εντός αυτής της συνάρτησης αρχικοποιείται το nvml ώστε να μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις που εμπεριέχονται στη βιβλιοθήκη. Έπειτα, επιλέγεται η GPU για την οποία θέλουμε να πραγματοποιηθεί η δειγματοληψία. Τέλος, δημιουργείται και εκτελείται ένα POSIX thread [36], το οποίο ανοίγει ένα αρχείο εξόδου και περιοδικά καταγράφει στο αρχείο την τρέχουσα μετρούμενη ισχύ.

- **nvmlAPIEnd():**
  Η κλήση αυτής της συνάρτησης πραγματοποιείται αφού ολοκληρωθεί η εκτέλεση του CUDA kernel, γίνει μεταφορά από την GPU στον Host των δεδομένων εξόδου και ελευθερωθεί η μνήμη που δεσμεύτηκε για τους buffers. Μέσω αυτής της κλήσης το POSIX thread που δημιουργήθηκε τερματίζει τη λειτουργία του και απενεργοποιείται το nvml.

Στο Παράρτημα Α′ παρουσιάζουμε ένα παράδειγμα χρήσης του API αυτού.

Για άλλη μια φορά προκειμένου η δειγματοληψία να μπορεί να ανταποκριθεί στην πραγματικότητα για τα ζεύγη εφαρμογών-datasets που ο χρόνος εκτέλεσης είναι ιδιαίτερα μικρός επεμβαίνουμε και επιβάλουμε την επανάληψη της εκτέλεσής τους δεκάδες, εκατοντάδες ή και χιλιάδες φορές. Δεδομένου ότι το αρχείο εξόδου απαρτίζεται από χιλιάδες ή και εκατομμύρια μετρήσεις οφείλουμε να βρούμε το μέσο όρο της ισχύος ειδικότερα στο πιο έντονο ενεργειακά κομμάτι που κατά κανόνα είναι η εκτέλεση του kernel. Για να πετύχουμε αυτό δημιουργήσαμε ένα πρόγραμμα (average.c)[33] το οποίο διαβάζει το αρχείο στο οποίο το POSIX thread έχει καταχωρήσει όλες της μετρήσεις για την ισχύ. Εν συνεχεία για να αποφύγουμε φαινόμενα υπερχείλισης, ανά αυστηρά καθορισμένα διαστήματα που ορίζουμε (ανάλογα με το μέγεθος του αρχείου) υπολογίζουμε το μέσο όρο της ισχύος για το κάθε διάστημα. Το πρώτο βήμα είναι να παράξουμε έναν γενικό μέσο όρο. Επειδή όπως εξηγήσαμε θέλουμε να εστιάσουμε στην συμπεριφορά της εφαρμογής κυρίως κατά την εκτέλεση του kernel ξαναδιαβάζουμε το αρχείο και προσθέτουμε μόνο τις μετρήσεις που είναι πάνω από τον πρώτο μέσο όρο. Από το άθροισμα που προκύπτει υπολογίζουμε εκ νέου το μέσο όρο. Το μέγεθος αυτό το θεωρούμε ως τη μέση ισχύ που καταναλώνει η εφαρμογή στην συγκεκριμένη GPU.

Για να οπτικοποιήσουμε το αποτέλεσμα της διαδικασίας αυτής παραθέτουμε το ακόλουθο διάγραμμα που πήραμε για την εκτέλεση της εφαρμογής Hotspot για το τρίτο διαθέσιμο dataset στην Tesla K40c. Βλέπουμε ότι η μέση ισχύς απεικονίζεται με την οριζόντια κόκκινη γραμμή.



Σχήμα 7.1: *Παράδειγμα υπολογισμού μέσης κατανάλωσης ισχύος κατά την εκτέλεση του kernel*

7.2 Μέτρηση microarchitecture-independent μετρικών και προεπεξεργασία δεδομένων εκπαίδευσης

Για να λάβουμε αυτές τις μετρήσεις χρησιμοποιήσαμε το εργαλείο MICA. Το πρώτο βήμα ήταν να καθορίσουμε το είδος και τις παραμέτρους της ανάλυσης. Για να το πετύχουμε αυτό διαμορφώσαμε κατάλληλα το αρχείο mica.conf ορίζοντας:

- **analysis_type: all**, μετρήσεις για όλες τις οικογένειες μετρικών.
- **interval_size: 10000000**, το παράθυρο δειγματοληψίας (σε πλήθος εντολών).
- **itypes_spec_file: itypes_default.spec**, ορισμός του αρχείου όπου θα καταχωρηθούν οι τύποι των εντολών που δεν εμπίπτουν σε κάποια υπάρχουσα ομάδα.

Για την λήψη των μετρήσεων δεν έχουμε παρά να εκτελέσουμε σε έναν οποιονδήποτε υπολογιστικό κόμβο με multicore CPU την ακόλουθη εντολή:
`$ /path-to-pin/pin -follow_execv 1 -t /path-to-mica.so/mica.so - ./executable_name arguements`
Προσοχή: έχουμε φροντίσει το εκτελέσιμο να εκτελείται σε ακριβώς ένα λογικό πυρήνα (δηλαδή για ένα νήμα), όπως ακριβώς απαιτεί το MICA. Σε περιπτώσεις που η δειγματοληψία ξεπέρναγε ένα συγκεκριμένο χρονικό πλαίσιο διακόπταμε την εκτέλεση, επειδή όμως σώζουμε ανά βήματα τις μετρήσεις μας έχουμε τελικά μια καλή εκτίμηση της συμπεριφοράς της εφαρμογής.

Για να πάρουμε το Memory Footprint του κάθε ζεύγους εφαρμογής-dataset χρησιμοποιούμε το valgrind massif tool πάλι με ακριβώς ένα νήμα, η εντολή που εκτελούμε είναι η εξής:
`$ valgrind -tool=massif ./executable_name arguements`
Έπειτα αναζητούμε το μεγαλύτερο heap που προκύπτει από την εκτέλεση και έχουμε το ζητούμενο μέγεθος.

Η προεπεξεργασία των δεδομένων που χρησιμοποιούνται για την εκπαίδευση των δύο μοντέλων πραγματοποιείται σε δύο φάσεις και είναι κατά κύριο λόγο κοινή. Στο παρόν υποκεφάλαιο θα δούμε αναλυτικά τις δύο αυτές φάσεις προεπεξεργασίας και θα επισημάνουμε διαφοροποιήσεις. Η πρώτη φάση πραγματοποιείται (προς το παρόν) με σχεδόν χειροκίνητο τρόπο και σκοπεύει στη διαμόρφωση του αρχείου που θα χρησιμοποιήσουμε για την εκπαίδευση των μοντέλων. Η δεύτερη φάση πραγματοποιείται κατά την εκτέλεση του κώδικα κάθε μοντέλου και αποτελεί ουσιαστικά ένα φιλτράρισμα που μας βοηθά να εστιάσουμε στις πληροφορίες όπου θα στηριχθούν τα μοντέλα.

Η **πρώτη φάση** πραγματοποιείται τοπικά. Αρχικά πραγματοποιούμε την σύμπτυξη και επεξεργασία των δεδομένων που έχουμε συλλέξει από το MICA, τα αποτελέσματα αυτά σώζονται τελικά σε ένα συγκεκριμένο αρχείο το οποίο αποτελεί τη βάση μας. Όταν πια όλα τα microarchitecture-independent δεδομένα είναι συγκεντρωμένα τα μετατρέπουμε σε ποσοστά ώστε κάθε χαρακτηριστικό να είναι κανονικοποιημένο για όλες τις εφαρμογές. Το επόμενο βήμα είναι να παρέχουμε τις hardware coefficients που αντιστοιχούν στην σύγκριση των ετερογενών αρχιτεκτονικών για έκαστη υπό εξέταση εφαρμογή. Έπειτα, υπολογίζουμε τις κατάλληλες fused μετρικές. Επόμενο βήμα όσον αφορά το μοντέλο πρόβλεψης για την απόδοση,

είναι δεδομένων των γρηγορότερων χρόνων εκτέλεσης της εκάστοτε εφαρμογής για έναν-έναν multicore CPU και για μία-μία GPU να υπολογίσουμε το σχετικό speedup (Xspeedup) σε κάθε σύγκριση. Βάσει αυτού θα γίνει το classification. Για το μοντέλο που αφορά την κατανάλωση ισχύος εξετάζουμε ανά εφαρμογή τις υλοποιήσεις (πλήθος από νήματα και μέγεθος thread block αντίστοιχα για multicore CPUs και GPUs) που προσφέρει το χαμηλότερο EDP (Energy Delay Product), προκύπτουν δύο EDPs ανά εφαρμογή. Τελευταίο βήμα είναι ανάλογα με το μοντέλο να κάνουμε το classification των εφαρμογών.

Ας δούμε πιο αναλυτικά τα παραπάνω βήματα:

1. **Σύμπτυξη δεδομένων από το MICA:**
   Η ανάλυση που έχουμε επιλέξει γίνεται με δειγματοληψία ανά συγκεκριμένο πλήθος εντολών. Το παραχθέν αρχείο για κάθε οικογένεια μετρήσεων αποτελείται από πολλές γραμμές οι οποίες αντιστοιχούν σε διαφορετικά βήματα της δειγματοληψίας. Αθροίζουμε λοιπόν τα πλήθη που αναγράφονται σε κάθε στήλη των αρχείων αυτών ανά στήλη και έπειτα συμπτύσσουμε τα αποτελέσματα σε ένα τελικό αρχείο με το σωστό labeling. Για να πετύχουμε αυτήν τη διαδικασία έχουμε υλοποιήσει δύο scripts τα οποία ο αναγνώστης μπορεί να μελετήσει στο αρχείο της διπλωματικής[33].

2. **Μετατροπή microarchitecture-independent δεδομένων σε ποσοστά:**
   Για να μετατρέψουμε τα δεδομένα στα κατάλληλα ποσοστά που ανταποκρίνονται στο όνομα του κάθε χαρακτηριστικού ακολουθήσαμε τις υποδείξεις του README.md που περιέχεται στο αρχείο του MICA-1.0.1[10].

3. **Υπολογισμός των τιμών για τις hardware coefficients:**
   Βασισμένοι στους τύπους που παραθέσαμε στο Κεφάλαιο 6 και στα datasheets των κατασκευαστών υπολογίζουμε τις τιμές αυτών των μετρικών. Έπειτα ανάλογα με τις υπό σύγκριση ετερογενείς αρχιτεκτονικές καταχωρούμε τις κατάλληλες τιμές.

4. **Υπολογισμός των fused μετρικών:**
   Για να διαμορφώσουμε τις fused μετρικές ομαδοποιούμε κατάλληλα τις μετρικές του δεύτερου βήματος όπως υποδείξαμε στο Κεφάλαιο 6, αυτό το επιτυγχάνουμε αθροίζοντας τα ποσοστά των αντίστοιχων μετρικών που ομαδοποιούμε. Για τη νέα μετρική που προκύπτει δίνουμε την κατάλληλη ονομασία. Τέλος, με τον κατάλληλο πολλαπλασιασμό με τις hardware coefficients παίρνουμε μια ξεχωριστή στήλη με τα fused μεγέθη.

5. **Υπολογισμός σχετικού speedup και EDP:**
   Εδώ η διαδικασία που ακολουθείται είναι η προφανής, για κάθε ξεχωριστή σύγκριση των ετερογενών αρχιτεκτονικών ως προς ένα ζεύγος εφαρμογής-dataset υπολογίζουμε τα μεγέθη σύμφωνα με τους τύπους:

$$Xspeedup = \frac{t_{CPU[i]}}{t_{GPU[j]}} \tag{7.1}$$

$$EDP_{CPU[i]} = Power \cdot t'_{CPU[i]} \cdot t'_{CPU[i]} \tag{7.2}$$

$$EDP_{GPU[j]} = Power \cdot t'_{GPU[j]} \cdot t'_{GPU[j]} \tag{7.3}$$

Όπου $t_{CPU[i]}$ είναι ο χρόνος που αντιστοιχεί στην γρηγορότερη εκτέλεση. Υπενθυμίζουμε πώς έχουμε πάρει χρονικές μετρήσεις για διαφορετικά πλήθη από OpenMP threads. Με ανάλογο τρόπο προκύπτει το $t_{GPU[j]}$. Στις επόμενες δύο σχέσεις τα αντίστοιχα τονούμενα μεγέθη αναπαριστούν τους χρόνους εκτέλεσης των υλοποιήσεων που αποφέρουν το μικρότερο EDP για την εκάστοτε αρχιτεκτονική. Το EDP όπως μαρτυρά το όνομά του είναι μια μετρική που λαμβάνει υπόψιν τόσο τον χρόνο εκτέλεσης όσο και την ενέργεια που καταναλώθηκε. Κρίνουμε ότι είναι η ιδανική μετρική για να στηρίξουμε το classification του δεύτερου μοντέλου.

6. **Classifications:**
   Για το μοντέλο που αφορά την απόδοση επιθυμούμε να διακρίνουμε τέσσερις κατηγορίες:

   - class 0: $Xspeedup \in [0, 1)$
   - class 1: $Xspeedup \in [1, 2)$
   - class 2: $Xspeedup \in [2, 10)$
   - class 3: $Xspeedup \in [10, +\infty)$

   Για το μοντέλο της κατανάλωσης ισχύος έχουμε binary classification:

   - class 0: $EDP_{CPU[i]} \leq EDP_{GPU[j]}$
   - class 1: $EDP_{CPU[i]} > EDP_{GPU[j]}$

Η **δεύτερη φάση** της προεπεξεργασίας των δεδομένων εκπαίδευσης πραγματοποιείται ήδη από την αρχή του κώδικα του κάθε μοντέλου. Σε πρώτο χρόνο διατηρούμε από το αρχείο εισόδου μόνο τις στήλες/χαρακτηριστικά που θα χρειαστούμε.Έπειτα, στρογγυλοποιούμε τα μεγέθη του μοντέλου. Με αυτόν τον τρόπο πιο εμφανώς στην περίπτωση του Decision Tree Classifier αποτρέπουμε σε μεγάλο βαθμό φωτογραφικές συνθήκες και έχουμε ένα αποτέλεσμα που ανταποκρίνεται περισσότερο στην πραγματικότητα. Όλα αυτά φαίνονται πιο καθαρά στο αρχείο της διπλωματικής[33].

7.3 Εκπαίδευση των μοντέλων

Για κάθε ένα από τα δύο μοντέλα έχουμε δύο προσεγγίσεις. Η πρώτη είναι η χρήση του Decision Tree Classifier για να έχουμε εποπτεία, και η δεύτερη είναι η χρήση του Random Forest Classifier για να έχουμε ένα πιο ακριβές μοντέλο. Για την εκπαίδευση του κάθε μοντέλου το πρώτο βήμα είναι να χωρίσουμε τα δεδομένα εισόδου σε δύο ομάδες, το training set και το test set. Το πρώτο σύνολο είναι αυτό βάσει του οποίου γίνεται η εκπαίδευση του μοντέλου ενώ το δεύτερο είναι για να ελέγξουμε την ακρίβεια που επιτυγχάνουμε για το παραχθέν μοντέλο. Από το sci-kit learn έχουμε τη δυνατότητα να ορίσουμε το ποσοστό των δεδομένων εισόδου που θα αποτελέσουν το test set και κατά επέκταση το ποσοστό δεδομένων που θα αποτελέσουν το training set. Για τον σκοπό της διπλωματικής σε κάθε προσέγγιση του κάθε μοντέλου έχουμε επιλέξει τα ποσοστά 25% και 40%.

Ο διαμοιρασμός αυτός γίνεται με τις εντολές που ακολουθούν:

```
1.from sklearn.model_selection import train_test_split
2.X_tr, X_t, y_tr, y_t = train_test_split(X,y,test_size=0.40,random_state=42)
```

Ο διαχωρισμός των δεδομένων εισόδου γίνεται με την εντολή train_test_split() [37]. Το βήμα είναι να εκπαιδεύσουμε το μοντέλο μας. Αυτό γίνεται με την χρήση του κατάλληλου Classifier, ως εκ τούτου έχουμε τις ακόλουθες δύο προσεγγίσεις:

Για τον **Decision Tree Classifier**:

```
3.from sklearn.tree import DecisionTreeClassifier
4.clf_dt = DecisionTreeClassifier(random_state=42)
5.clf_dt = clf_dt.fit(X_tr, y_tr)
```

Για τον **Random Forest Classifier**:

```
3.from sklearn.ensemble import RandomForestClassifier
4.classifier = RandomForestClassifier(n_estimators=128,random_state=42)
5.classifier.fit(X_tr,y_tr)
```

Συνοπτικά, αυτό που γίνεται στους παραπάνω κώδικες οι οποίοι εκτελούνται σε cells του Jupyter Notebook[38], είναι ότι φορτώνεται ο εκάστοτε learner (γρ. 3), δημιουργείται ένα instance αυτού για να μπορούμε να χρησιμοποιήσουμε τις ιδιότητες (γρ. 4) του και τέλος εκπαιδεύουμε το μοντέλο μας μέσω της συνάρτησης fit (γρ. 5). Στην περίπτωση του Random Forest Classifier έχουμε επιλέξει να δημιουργηθούν 128 συνολικά δέντρα αποφάσεων από όπου θα διαμορφωθεί η τελική πρόβλεψη. Αυτό έγινε διότι παρατηρήσαμε εμπειρικά ότι παραπάνω ο υπολογιστικός κόπος που επιφέρει το μεγαλύτερο πλήθος δέντρων απόφασης δεν αποφέρει τελικά κάποια αισθητή αλλαγή στην ακρίβεια του μοντέλου.

7.4 Cost Complexity Pruning & Accuracies

Κατά την ανάπτυξη των δύο μοντέλων, τόσο για τον Decision Tree Classifier όσο και για τον Random Forest Classifier, προσθέταμε διαδοχικά μία-μία τις fused μετρικές παρατηρώντας αν η ακρίβεια ανέβαινε ή όχι. Αυτό που παρατηρήσαμε είναι ότι η ακρίβεια που προέκυψε με την συμμετοχή όλων των fused μετρικών ήταν η μεγαλύτερη από όσες είχαμε δει ως τώρα. Για να επιβεβαιώσουμε ότι η σύμπτηξη των μετρικών του MICA με τις hardware coefficients δεν έπασχε αλλά αντίθετα ήταν εύστοχη δοκιμάσαμε αντί των fused μετρικών να συμπεριλάβουμε στο μοντέλο τις hardware coefficients. Εν τέλει, είδαμε ότι η ακρίβεια που πετύχαμε σε κάθε περίπτωση για αυτό το σενάριο ήταν μικρότερη από ότι με τις fused μετρικές, πράγμα που ενισχύει την ορθότητα της υπόθεσης εργασίας μας.

Αφού λοιπόν έχει ολοκληρωθεί όλη η διαδικασία μέχρι την συμμετοχή όλων των fused μετρικών στο μοντέλο, όπου και επιτυγχάνεται η μέγιστη ακρίβεια, πρέπει να εξετάσουμε αφενός την εγκυρότητα των αποτελεσμάτων και αφετέρου αν μπορούμε να επιφέρουμε κάποιες

βελτιώσεις. Για τον Random Forest Classifier, λόγω της σχεδίασης του δεν υπάρχουν βελτιώσεις που μπορούμε να κάνουμε. Στην περίπτωση του Decision Tree Classifier υπάρχει το πρόβλημα του overfitting δηλαδή της φωτογραφικής προσαρμογής του μοντέλου στο training set που συνεπώς οδηγεί σε μεγάλο ποσοστό αστοχίας για το μοντέλο. Επίσης, το δέντρο αποφάσεων που προκύπτει είναι ευμέγεθες και άρα περιορίζεται η εποπτεία λόγω της οποίας καταφύγαμε σε αυτόν τον classifier.

Για να αντιμετωπίσουμε αυτό το πρόβλημα η τακτική που αξιοποιήσαμε είναι η μέθοδος Cost Complexity Pruning (CCP) για να προσδιορίσουμε κατάλληλα την παράμετρο ccp_alpha βάσει της οποίας εν συνεχεία πραγματοποιούμε pruning, δηλαδή το κλάδεμα του δέντρου[39]. Την πλήρη διαδικασία που ακολουθήσαμε ο αναγνώστης μπορεί να την αναζητήσει στο online υλικό της διπλωματικής [33]. Στο επόμενο κεφάλαιο θα παρουσιάσουμε αναλυτικά τα αποτελέσματα που λάβαμε, θα εξηγήσουμε τις πληροφορίες που μπορούμε να εξάγουμε καθώς και τα τελικά μοντέλα.

Μετά από όλα τα παραπάνω βήματα το μοντέλο τόσο για την απόδοση όσο και για την κατανάλωση ισχύος είναι έτοιμο προς χρήση. Ο χρήστης έχει τη δυνατότητα μέσω έκαστου μοντέλου να πάρει για την εφαρμογή του ενδιαφέροντός του μια πρόβλεψη. Για να το πετύχει αυτό οφείλει να δώσει στο μοντέλο τα χαρακτηριστικά της δικής του εφαρμογής (μέσω του MICA, valgrind massif, κτλ.), να διαμορφώσει κατάλληλα το σύνολο των hardware coefficients και να συμπεριλάβει τελικά τις fused μετρικές.

Πέραν της πρόβλεψης, ένα σημαντικό πλεονέκτημα των μοντέλων που διαμορφώσαμε είναι ότι παρέχουν στον χρήστη τη δυνατότητα να μελετήσει τον Confusion Matrix. Ο πίνακας αυτός εξ ορισμού είναι δισδιάστατος στον κατακόρυφο άξονα έχει τις πραγματικές τιμές για την κλάση στην οποία ανήκει κάθε εφαρμογή, ενώ στον οριζόντιο άξονα είναι οι τιμές που προβλέφθηκαν από το μοντέλο για έκαστη εφαρμογή. Μπορεί να εκφραστεί ως πίνακας, μπορεί όμως να εκφραστεί και ως διάγραμμα. Γενική ιδέα είναι ότι όσο πιο ακριβές είναι ένα μοντέλο τόσο μεγαλύτερα πλήθη θα βλέπουμε στη διαγώνιο του πίνακα αυτού. Η συνεισφορά αυτού του πίνακα στο να μπορούμε να εξάγουμε συμπεράσματα δεν σταματά όμως εδώ. Συγκεκριμένα, αν παρατηρήσουμε ότι το μοντέλο παρεκκλίνει στις προβλέψεις του με έναν συστηματικό τρόπο μπορούμε να εκτιμήσουμε αν με βάση τις προσδοκίες και την πρόβλεψη που λάβαμε αξίζει να επιχειρήσουμε τη μετατροπή του κώδικα από μια OpenMP υλοποίηση σε μία CUDA υλοποίηση.

Κεφάλαιο 8

Μελέτη των αποτελεσμάτων

Στο παρόν κεφάλαιο θα παρουσιάσουμε αναλυτικά τα αποτελέσματα που λάβαμε από τη μεθοδολογία που ακολουθήσαμε. Θα προσπαθήσουμε να ερμηνεύσουμε τα αποτελέσματα και θα οδηγηθούμε σε κάποια σημαντικά συμπεράσματα. Όπως θα γίνει αντιληπτό μπορούμε να βγάλουμε κάποια πορίσματα σχετικά με το πώς συγκεκριμένες πτυχές μιας εφαρμογής μπορεί να παίξουν τον καθοριστικό ρόλο για την τελική της απόδοση σε μια εκ των δύο ετερογενών αρχιτεκτονικών. Θα δούμε πώς ο χρήστης των δύο μοντέλων μπορεί να οδηγηθεί στην απάντηση του ερωτήματος ″CPU ή GPU ″ στο πλαίσιο μιας συγκεκριμένης εφαρμογής.

Για το μοντέλο απόδοσης έχουμε δύο προσεγγίσεις τις οποίες πρέπει να εξετάσουμε, τον Decision Tree Classifier και τον Random Forest Classifier. Όπως είπαμε για κάθε μία από τις προσεγγίσεις προσθέσαμε σταδιακά μία μία τις μετρικές για να εξακριβώσουμε ότι όντως πριμοδοτούν την ακρίβεια. Σε κάθε ένα από τα ακόλουθα υποκεφάλαια θα παραθέσουμε πίνακες που δείχνουν αυτήν την αύξηση, θα δούμε τη μορφή των αποτελεσμάτων, θα τα ερμηνεύσουμε και θα αξιολογήσουμε την πληροφορία που αντλούμε σε κάθε περίπτωση.

8.1 Αποτελέσματα μοντέλου απόδοσης

Στο μοντέλο απόδοσης έχουμε ορίσει τέσσερις κλάσεις. Σε κάθε βήμα μετρήσαμε την ακρίβεια του μοντέλου για $test\_size = 25\%$ και για $test\_size = 40\%$, τόσο για την αμιγή ακρίβεια επί του test set όσο και για την ακρίβεια ολόκληρου του αρχικού set. Οι ακρίβειες αναγράφονται αναλυτικά στου πίνακες 8.1 και 8.2. Μια πληρέστερη εικόνα των αποτελεσμάτων την έχουμε από τους Confusion Matrix. Για την περίπτωση της προσέγγισης με τον Decision Tree Classifier διαμορφώνεται και ένα δέντρο αποφάσεων για κάθε υλοποίηση. Στον Random Forest Classifier επειδή τα δέντρα αποφάσεων στα οποία στηρίζεται η πρόβλεψη είναι 128 δεν έχουμε κάποια μοναδική γραφική απεικόνιση του μοντέλου. Μπορούμε όμως να δούμε την σημασία των παραμέτρων του μοντέλου στον καθορισμό της τελικής πρόβλεψης μέσω ενός barplot.

8.1.1 Random Forest Classifier

Η ακρίβειες για κάθε βήμα της διαδικασίας που περιγράψαμε στην περίπτωση του Random Forest Classifier είναι αυτές που αναγράφονται στον Πίνακα 8.1. Με τις διαδοχικές προσθήκες των παραμέτρων παρατηρούμε ότι η ακρίβεια αυξάνεται σταδιακά, γεγονός που επικυρώνει την ανάλυση που κάναμε. Για το test set πετυχαίνουμε αύξηση της ακρίβειας είναι σχεδόν 9

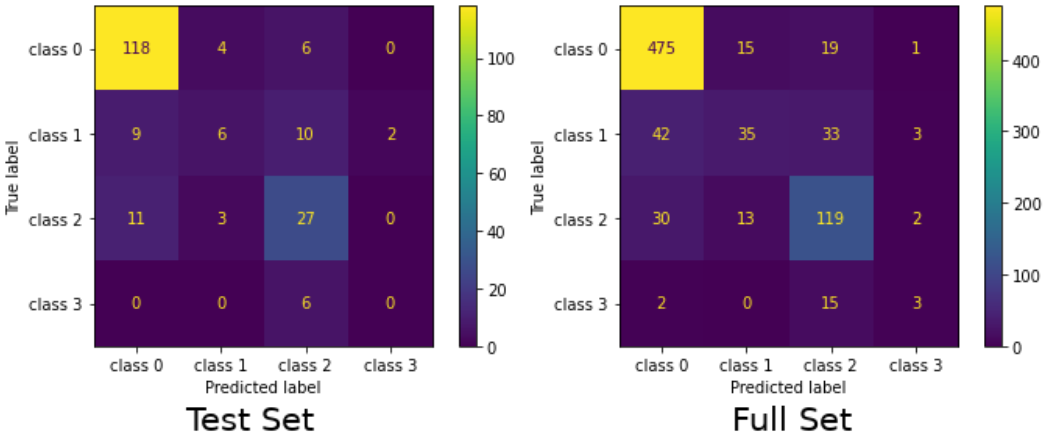
μονάδες, για το full set η αντίστοιχη αύξηση είναι πάνω από 15 ποσοστιαίες μονάδες. Η τελική ακρίβεια αυτής της υλοποίησης είναι:

- Για $test\_size = 25\%$:
  - Η ακρίβεια επί του test set είναι **84.16%**.
  - Η ακρίβεια επί του full set είναι **96.03%**.
- Για $test\_size = 40\%$:
  - Η ακρίβεια επί του test set είναι **82.04%**.
  - Η ακρίβεια επί του full set είναι **92.81%**.

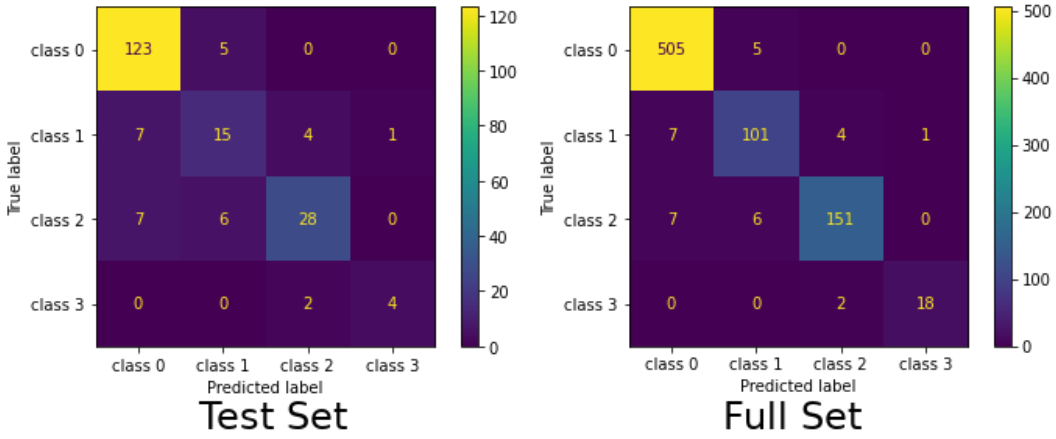
Πίνακας 8.1: *Random Forest Classifier ακρίβειες για το μοντέλο απόδοσης*

Model Version	Parameters	Test Set Size	Test Set Acc.	Full Set Acc.
1	MICA-only	25%	74.75%	78.31%
1	MICA-only	40%	72.76%	77.82%
2	+memory grouped metrics	25%	74.75%	78.31%
2	+memory grouped metrics	40%	72.76%	77.82%
3	+memory fused metrics	25%	80.20%	92.69%
3	+memory fused metrics	40%	78.95%	90.33%
4	+ILP fused	25%	81.68%	95.42%
4	+ILP fused	40%	79.26%	91.70%
5	+SSE fused metrics	25%	84.16%	96.03%
5	+SSE fused metrics	40%	82.04%	92.81%

Ας εξετάσουμε τα διαγράμματα της Εικόνας 8.1, εκεί παρατίθενται οι Confusion Matrix που πήραμε για το μοντέλο που της πρώτης υλοποίησης του μοντέλου που στηριζόταν αποκλειστικά στην χρήση των microarchitecture-independent μετρικών με $test\_size = 25\%$. Έπειτα, ας δούμε τους αντίστοιχους Confusion Matrix της Εικόνας 8.2, αυτοί αντιστοιχούν στην τελευταία υλοποίηση του μοντέλου όπου επιτυγχάνουμε την υψηλότερη ακρίβεια (πάλι για $test\_size = 25\%$). Χάριν πληρότητας στο Παράρτημα Β΄ παραθέτουμε τους Confusion Matrix που αντιστοιχούν σε όλες τις υλοποιήσεις που αναγράφονται στον Πίνακα 8.1.



Σχήμα 8.1: *Confusion Matrix για το Model Version 1*



Σχήμα 8.2: *Confusion Matrix για το Model Version 5*

Μέσω της μετάβασης από την πρώτη στην τελευταία υλοποίηση είναι εμφανής η αύξηση της ακρίβειας του μοντέλου. Η βελτίωση όμως δεν έγκειται μόνο στην απόλυτη τιμή της ακρίβειας. Μέσω των Confusion Matrix παρατηρούμε μια άλλη σημαντική βελτίωση. Ιδανικά θα επιθυμούσαμε όλες οι τιμές να βρίσκονται στη διαγώνιο του πίνακα. Αντ΄ αυτού βλέπουμε ότι πολλές από τις τιμές είναι διασκορπισμένες. Είναι όμως εξίσου διασκορπισμένες;

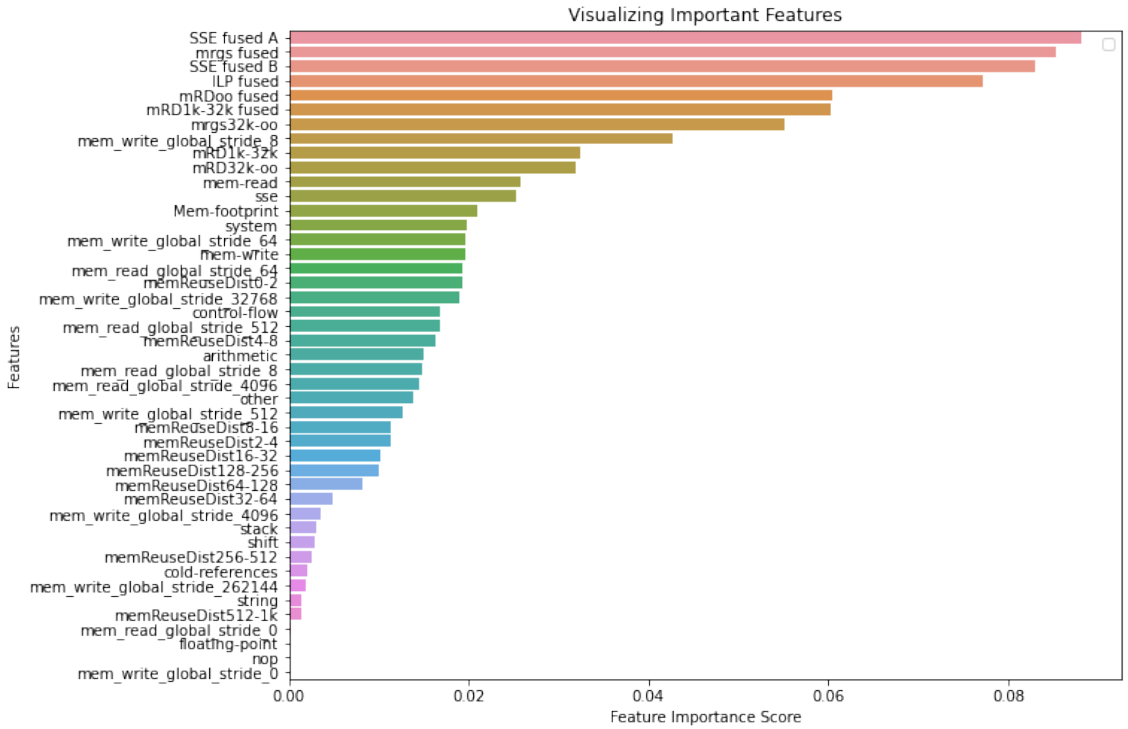
Είναι γεγονός ότι με την πέμπτη υλοποίηση έχουμε μεγαλύτερη συγκέντρωση στη διαγώνιο του πίνακα τόσο για το test set όσο και για το full set. Επιπρόσθετα για τις εφαρμογές της κάθε κλάσης παρατηρούμε μια σοβαρή μείωση στο πόσο διασκορμπισμένες είναι. Συγκεκριμένα στο full set:

- Για την πρώτη κλάση μόλις το **0.98%** των εφαρμογών προβλέπεται εσφαλμένα ότι ανήκει στη δεύτερη κλάση.
- Για την δεύτερη κλάση το **6.19%** των εφαρμογών προβλέπεται εσφαλμένα ότι ανήκει στην πρώτη κλάση, το **3.54%** προβλέπεται εσφαλμένα ότι ανήκει στην τρίτη κλάση και το **0.88%** ότι ανήκει στην τέταρτη κλάση.
- Για την τρίτη κλάση το **4.27%** των εφαρμογών προβλέπεται εσφαλμένα ότι ανήκει στην πρώτη κλάση, το **3.66%** προβλέπεται εσφαλμένα ότι ανήκει στην δεύτερη κλάση.
- Για την τέταρτη κλάση το **10%** των εφαρμογών προβλέπεται εσφαλμένα ότι ανήκει στην τρίτη κλάση.

Όλα αυτά καταδεικνύουν ότι ακόμα και στις περιπτώσεις που το μοντέλο μας αστοχεί δίνει μια πρόβλεψη η οποία κατά κανόνα δεν απέχει πολύ από την πραγματικότητα. Πρακτικά αυτό σημαίνει ότι αν για μια δεδομένη εφαρμογή η πρόβλεψη είναι ότι ανήκει στην τρίτη κλάση (δηλαδή το σχετικό speedup κυμαίνεται στο διάστημα $[2, 10)$), τότε ακόμα και σε περίπτωση αστοχίας της πρόβλεψης υπάρχει πιθανότητα 3.66% να πετύχουμε σχετικό speedup στο διάστημα [1,2) κοκ. Αυτή η παρατήρηση είναι ιδιαιτέρως χρήσιμη καθώς ο χρήστης θα έχει τη δυνατότητα να συνεκτιμήσει δεδομένης της κατανομής που εκφράζεται από τον Confusion Matrix αλλά και από την ακρίβεια του μοντέλου κατά πόσο αξίζει να προβεί για τους σκοπούς της δικής του εφαρμογής στην μετατροπή της υλοποίησης του από OpenMP

σε CUDA. Για κάποιες εφαρμογές μια επιτάχυνση του διαστήματος της δεύτερης κλάσης παραμένει ικανοποιητική, και η ίδια λογική ισχύει και για τις επόμενες κλάσεις.

Τελευταία πληροφορία προς τον χρήστη του μοντέλου είναι το διάγραμμα που απεικονίζει το ποιες εκ των παραμέτρων που χρησιμοποιούνται για την εκπαίδευση του μοντέλου έχουν τη μεγαλύτερη σημασία στη διαμόρφωση της τελικής πρόβλεψης. Δεδομένου του διαγράμματος αυτού ο χρήστης δύναται να καταλάβει ποια είναι τα χαρακτηριστικά της εφαρμογής βάσει των οποίων μια εφαρμογή θα πριμοδοτηθεί ή όχι από την εκτέλεση σε μια δεδομένη GPU αρχιτεκτονική.



Σχήμα 8.3: *Διάγραμμα σημασίας των παραμέτρων για το Model Version 5*

Όπως παρατηρούμε στην κορυφή της λίστας αναγράφονται οι fused μετρικές που εισάγαγαμε, πράγμα που δείχνει ότι η ανάλυσή μας σχετικά με τις μετρικές απέφερε καρπούς.

8.1.2 Decision Tree Classifier

Για την υλοποίηση του μοντέλου απόδοσης με τον Decision Tree Classifier κάναμε διαδοχικές απόπειρες για να διαπιστώσουμε αν όντως οι ομαδοποιήσεις και οι fused μετρικές που εισάγουμε συνεισφέρουν ενεργά στην αύξηση της ακρίβειας των προβλέψεων. Οι ακρίβειες για κάθε βήμα της διαδικασίας που περιγράψαμε φαίνονται αναλυτικά στον Πίνακα 8.2. Παρατηρούμε ότι σταδιακά κατορθώνουμε να αυξήσουμε την ακρίβεια και να μειώσουμε την διαφορά της ακρίβειας του test set και του full set. Αυτό είναι επιθυμητό καθώς το μοντέλο μας θέλουμε να παρέχει έγκυρες προβλέψεις για εφαρμογές πάνω στις οποίες δεν έχει εκπαιδευτεί.

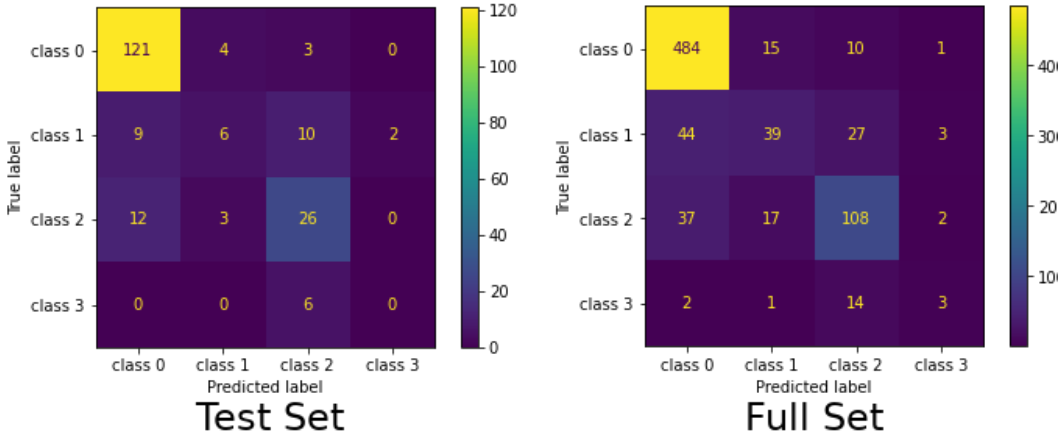
Σε αντίθεση με την υλοποίηση με τον Random Forest Classifier εδώ θα εξετάσουμε τρεις εκδοχές του μοντέλου. Έχει νόημα να δούμε και να συγκρίνουμε την πρώτη, την πέμπτη και την έκτη εκδοχή στην οποία έχει πραγματοποιηθεί το pruning. Η σύγκριση αυτή θα γίνει

για $test\_size = 25\%$ σε τρία επίπεδα αρχικά με τις ακρίβειες, έπειτα με του αντίστοιχους Confusion Matrix και τέλος με τα δυο τελευταία προκύψαντα δέντρα αποφάσεων.

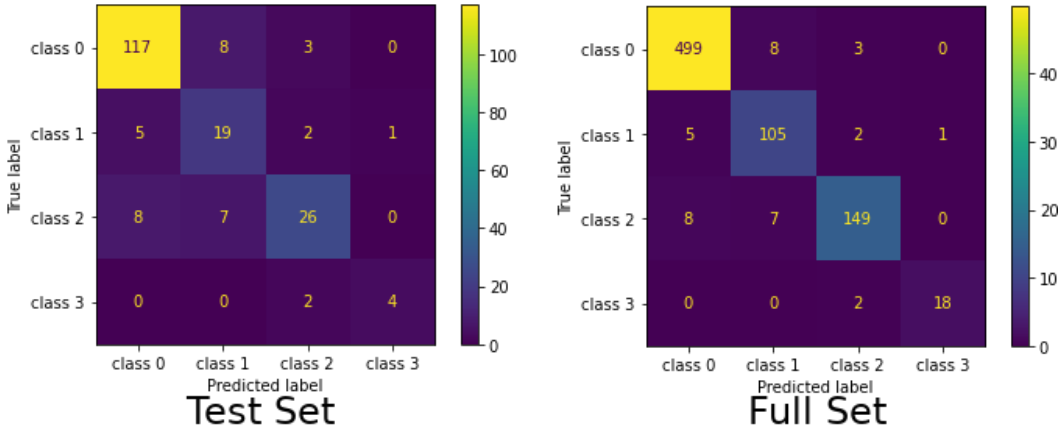
Πίνακας 8.2: *Decision Tree Classifier ακρίβειες για το μοντέλο απόδοσης*

Model Version	Parameters	Test Set Size	Test Set Acc.	Full Set Acc.
1	MICA-only	25%	75.74%	78.56%
1	MICA-only	40%	72.45%	77.70%
2	+memory grouped metrics	25%	75.74%	78.56%
2	+memory grouped metrics	40%	72.45%	77.70%
3	+memory fused metrics	25%	78.22%	92.19%
3	+memory fused metrics	40%	77.40%	89.71%
4	+ILP fused	25%	85.69%	95.17%
4	+ILP fused	40%	79.57%	91.82%
5	+SSE fused metrics	25%	82.18%	95.54%
5	+SSE fused metrics	40%	80.80%	92.32%
6	pruned	25%	81.68%	86.74%
6	pruned	40%	80.80%	92.32%

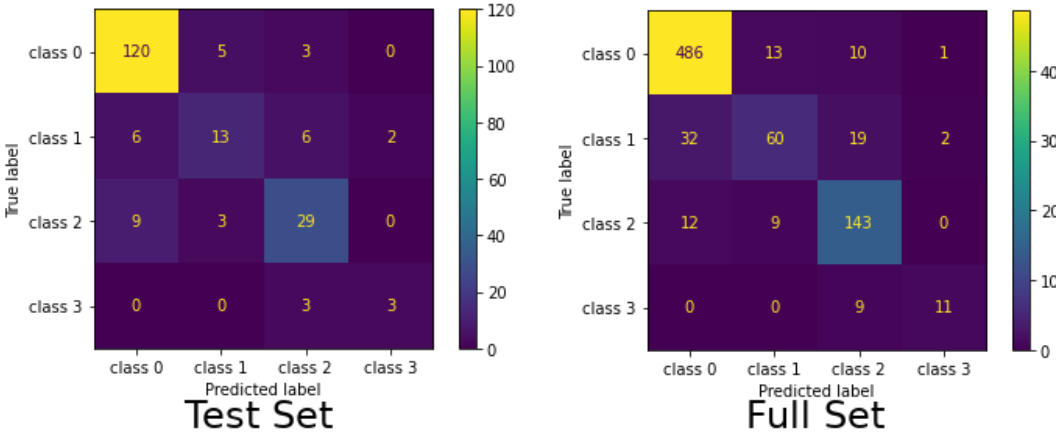
Βλέπουμε ότι από υλοποίηση σε υλοποίηση, με την επιλογή των κατάλληλων παραμέτρων καταφέρνουμε να αυξήσουμε την ακρίβεια. Οι Confusion Matrix που προέκυψαν είναι οι ακόλουθοι:



Σχήμα 8.4: *Confusion Matrix Model Version 1*

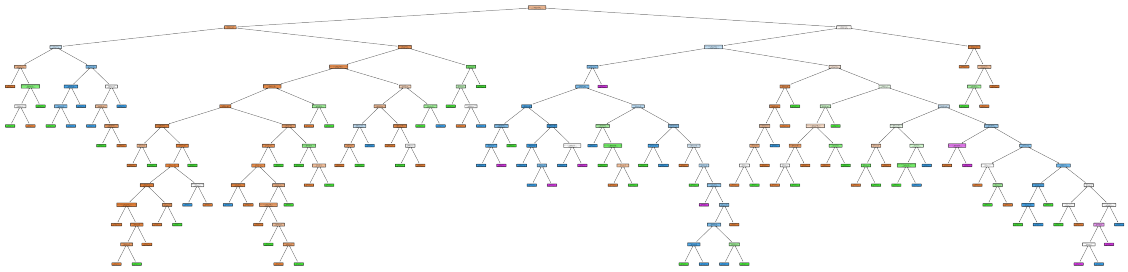


Σχήμα 8.5: *Confusion Matrix Model Version 5*



Σχήμα 8.6: *Confusion Matrix Model Version 6 (pruned)*

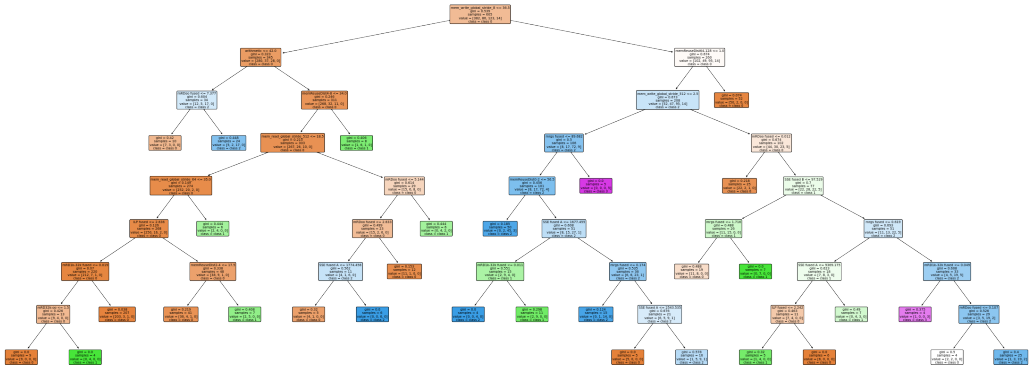
Μια εύλογη απορία που ανακύπτει από τα παραπάνω αποτελέσματα είναι το γιατί προχωράμε σε pruning του δέντρου αποφάσεων την στιγμή που δεν επωφελούμαστε από άποψη ακρίβειας. Όπως είπαμε επιλέξαμε τον Decision Tree Classifier προκειμένου να έχουμε εποπτεία σχετικά με το ποια χαρακτηριστικά της εφαρμογής είναι αυτά τα οποία κρίνουν την καταλληλότητα της μία αρχιτεκτονική έναντι μίας άλλης. Για να αναδείξουμε την αναγκαιότητα του pruning παραθέτουμε τα δέντρα αποφάσεων που παρήχθησαν αφενός για την υλοποίηση με την μεγαλύτερη ακρίβεια (Model Version 5) και αφετέρου για την pruned υλοποίηση. Τα δέντρα αποφάσεων που παρήχθησαν εκ των πραγμάτων δεν είναι ευκρινή κατά τη διατύπωσή τους λόγω του μεγέθους τους. Αξίζει όμως να εστιάσουμε στη τεράστια διαφορά μεγέθους τους και στη μικρή διαφορά τους από άποψη ακρίβειας.



Σχήμα 8.7: *Decision Tree Model Version 5*

Το δέντρο αυτό παρότι προσφέρει υψηλή ακρίβεια προβλέψεων, έχει μεγάλη διαφορά μεταξύ της ακρίβειας του test set και του full set. Το σημαντικότερο όμως είναι ότι λόγω του μεγέθους του δεν είναι ευανάγνωστο προς τον χρήστη, η πληροφορία που παρέχει είναι πολύπλοκη καθώς μέχρι να φτάσουμε σε μία πρόβλεψη πρέπει να προσπελαστούν πολλοί κόμβοι και άρα διαμορφώνεται μια σύνθετη και πολύπλοκη συνθήκη. Έτσι, ο χρήστης δεν έχει την δέουσα εποπτεία ώστε να διαπιστώσει ποια είναι τα χαρακτηριστικά του κώδικα της εφαρμογής του που θα παίξουν τον καθοριστικό ρόλο. Ας θυμηθούμε ότι κάθε ένας από τους κόμβους εκτός των φύλλων του δέντρου περιλαμβάνει και από μία συνθήκη για μία παράμετρο του μοντέλου. Με ελάχιστο κόστος στην ακρίβεια προχωράμε σε pruning. Η ακριβής διαδικασία εξηγείται πλήρως στο Παράρτημα Δ΄ του παρόντος τόμου. Το pruned δέντρο αποφάσεων που προκύπτει

είναι το ακόλουθο.



Σχήμα 8.8: *pruned Decision Tree (Model 6)*

Ο διαφορετικός χρωματισμός αντιστοιχεί και σε διαφορετική κλάση. Μια τελευταία παρατήρηση που πρέπει να έχουμε υπόψιν είναι ότι τα δέντρα αποφάσεων που έχουν προκύψει από αυτήν την προσέγγιση δεν είναι μοναδικά, αν κάποιος δεν ορίσει αυστηρά το random_state σε ένα σταθερό αριθμό τότε μια επανεκτέλεση του κώδικα θα οδηγήσει και σε μια διαφορετική απεικόνιση. Παρόλα αυτά όπως παρατηρήσαμε και εμπειρικά υπάρχουν υποκλάδοι του δέντρου οι οποίοι παραμένουν σταθεροί και επαληθεύουν την διαίσθησή μας. Ο αναγνώστης μπορεί να ανατρέξει στο υλικό της διπλωματικής που διατηρούμε στο GitLab [33].

8.2 Αποτελέσματα μοντέλου κατανάλωσης ισχύος

Στο μοντέλο που αφορά την κατανάλωση ισχύος έχουμε ορίσει δύο κλάσεις. Η πρώτη κλάση είναι αυτήν στην οποία οι εφαρμογές παρουσιάζουν μικρότερο EDP κατά την εκτέλεση σε CPU από ότι σε GPU. Όπως είπαμε στο μοντέλο αυτό αξιοποιήθηκαν οι μετρήσεις από τις ετερογενείς αρχιτεκτονικές Haswell, Broadwell, Kepler, Maxwell. Για να εμπλουτίσουμε το μοντέλο και δεδομένου ότι η Tesla m2050 έχει CUDA version 7.5 και άρα δεν μπορούμε να πραγματοποιήσουμε μετρήσεις με τη βιβλιοθήκη $< nvml.h >$, κάναμε μία εκτίμηση η οποία περιγράφεται στο Παράρτημα Γ′. Για να είμαστε πλήρεις στο online αρχείο της διπλωματικής ο αναγνώστης μπορεί να βρει τα δεδομένα και με ελάχιστες τροποποιήσεις να λάβει το αντίστοιχο μοντέλο χωρίς να έχουν συμπεριληφθεί οι ″μετρήσεις″ της Tesla m2050.

Σε κάθε βήμα μετρήσαμε την ακρίβεια του μοντέλου για μόνο για $test\_size = 40\%$, για το test set όσο και για το full set. Η επιλογή αυτού του ποσοστού διαμοιρασμού έγινε μιας και το αρχείο εισόδου για αυτό το μοντέλο είναι μικρότερο από ότι για το μοντέλο της απόδοσης. Έτσι καλύτερα ο έλεγχος της ακρίβειας να γίνει σε μια πιο αυστηρή βάση. Τα αποτελέσματα αναγράφονται αναλυτικά στου πίνακες 8.3 και 8.4 καθώς επίσης και στους Confusion Matrix που ακολουθούν. Όσον αφορά τον Decision Tree Classifier παραθέτουμε μόνο το τελευταίο pruned δέντρο αποφάσεων. Για τον Random Forest Classifier βλέπουμε την σημασία των παραμέτρων του μοντέλου στον καθορισμό της τελικής πρόβλεψης μέσω ενός barplot.

Η συλλογιστική που ακολουθείται στο συγκεκριμένο μοντέλο είναι εντελώς παρεμφερής με την συλλογιστική που παρουσιάστηκε στο μοντέλο απόδοσης.

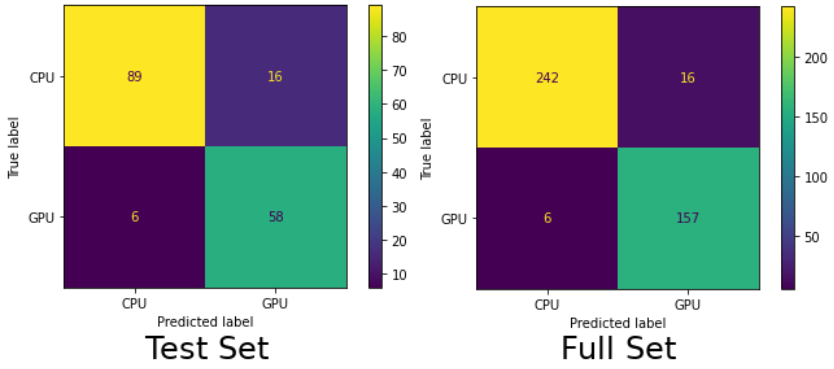
8.2.1 Random Forest Classifier

Για την υλοποίηση του μοντέλου με τον Random Forest Classifier οι υλοποιήσεις και οι αντίστοιχες ακρίβειες που πήραμε είναι αυτές που αναγράφονται στον Πίνακα 8.3.

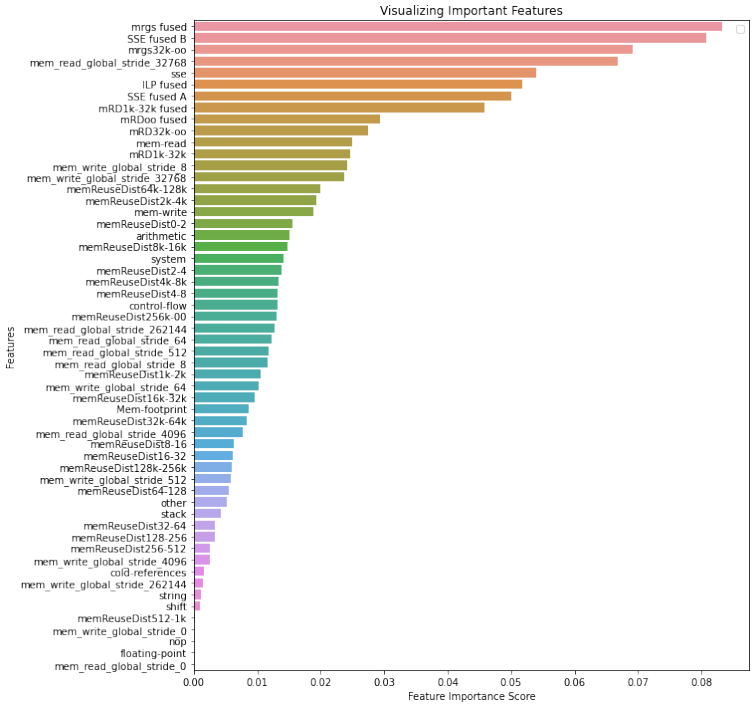
Πίνακας 8.3: *Random Forest Classifier ακρίβειες για το μοντέλο κατανάλωσης ισχύος*

Model Version	Parameters	Test Set Size	Test Set Acc.	Full Set Acc.
1	MICA-only	40%	81.56%	85.99%
2	+memory grouped metrics	40%	81.66%	85.99%
3	+fused metrics	40%	86.98%	94.77%

Οι Confusion Matrix και το διάγραμμα των σημαντικότερων παραμέτρων του μοντέλου έχουν ως εξής:



Σχήμα 8.9: *Confusion Matrix για το μοντέλο κατανάλωσης ισχύος (Model Version 3)*



Σχήμα 8.10: *Διάγραμμα σημασίας των παραμέτρων για το μοντέλο κατανάλωσης ισχύος (Model Version 3)*

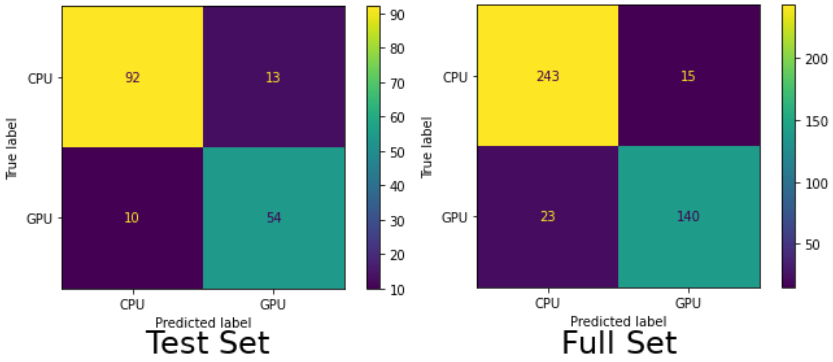
8.2.2 Decision Tree Classifier

Για την υλοποίηση του μοντέλου απόδοσης με τον Decision Tree Classifier κάναμε τρεις διαδοχικές απόπειρες όπως φαίνεται στον Πίνακα 8.4. Και εδώ επιλέγουμε $test\_size = 40\%$.

Πίνακας 8.4: *Decision Tree Classifier ακρίβειες για το μοντέλο κατανάλωσης ισχύος*

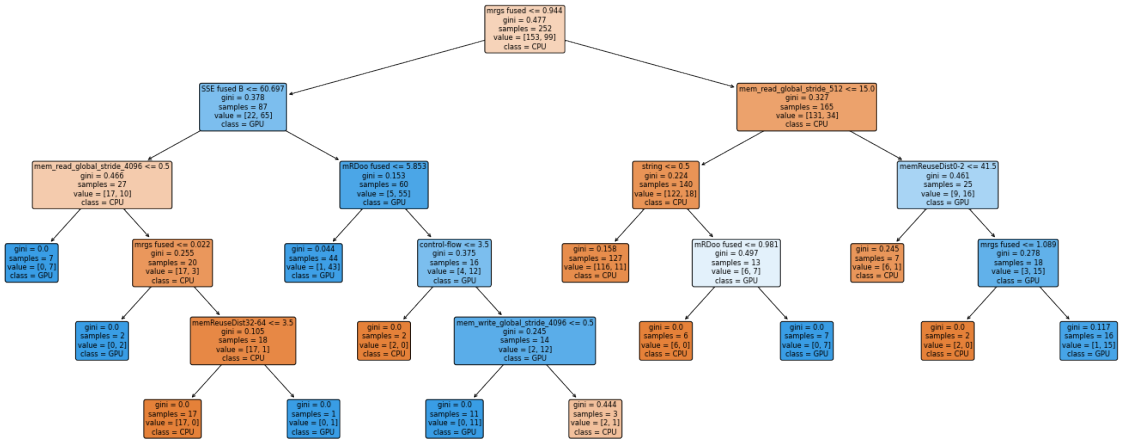
Model Version	Parameters	Test Set Size	Test Set Acc.	Full Set Acc.
1	MICA-only	40%	79.29%	84.56%
2	+fused metrics	40%	81.66%	92.64%
3	pruned	40%	86.39%	90.97%

Βλέπουμε ότι από υλοποίηση σε υλοποίηση, με την επιλογή των κατάλληλων παραμέτρων καταφέρνουμε να αυξήσουμε την ακρίβεια. Οι Confusion Matrix που προέκυψαν είναι οι ακόλουθοι:



Σχήμα 8.11: *Confusion Matrix για το pruned Decision Tree (Model 3)*

Τέλος το pruned δέντρο αποφάσεων είναι το ακόλουθο:



Σχήμα 8.12: *pruned Decision Tree (Model 3)*

Μέρος III

Επίλογος

Κεφάλαιο 9

Επίλογος

9.1 Συμπεράσματα

Η αποδοτική αξιοποίηση των πόρων ενός σύγχρονου υπολογιστικού συστήματος είναι ένα από τα κύρια ζητούμενα τόσο για τους παρόχους των υπηρεσιών υπολογιστικού νέφους όσο και για τους χρήστες του. Ως εκ τούτου, η δημιουργία εργαλείων τα οποία συνεισφέρουν σε κεντρικές σχεδιαστικές αποφάσεις των χρηστών είναι καίριας σημασίας. Δεδομένων του hardware diversity των σύγχρονων υπολογιστικών συστημάτων και του software diversity των εφαρμογών που σχεδιάζονται και εκτελούνται σε αυτά, τα μοντέλα που προτείνουμε δίνουν χειροπιαστές ενδείξεις για το ποια εκ των διαθέσιμων ετερογενών αρχιτεκτονικών θα πρέπει να αξιοποιηθεί για την εκάστοτε εφαρμογή.

Συγκεκριμένα, το μοντέλο απόδοσης που προτείνουμε:

- Ορίζει τέσσερις κλάσεις τιμών για την επιτάχυνση που δύναται να προσφέρει για μία εφαρμογή η χρήση μιας GPU έναντι ενός multicore CPU. Οι κλάσεις αυτές έχουν διαμορφωθεί με τέτοιον τρόπο ώστε ο προγραμματιστής μίας εφαρμογής ανάλογα με τις ανάγκες του να κρίνει κατά πόσο η πρόβλεψη που λαμβάνει από το μοντέλο δικαιολογεί τον χρόνο που απαιτείται για τη μετάβαση σε μια GPU-υλοποίηση της εφαρμογής του.

- Η ακρίβεια της πρόβλεψης του μοντέλου είναι υψηλή.

- Ακόμα και στις περιπτώσεις εσφαλμένων προβλέψεων ο προγραμματιστής μπορεί μέσω των Confusion Matrix που παρέχονται να κρίνει αν η αμέσως επόμενη πιθανή κλάση τιμών στην οποία μπορεί να εμπίπτει η εφαρμογή του, δικαιολογεί τη μετάβαση της εφαρμογής σε μία GPU-υλοποίηση.

- Το μοντέλο απόδοσης προσφέρει μέσω του δέντρου αποφάσεων αλλά και του διαγράμματος σημασίας των παραμέτρων επαρκή εποπτεία στον προγραμματιστή. Μέσω αυτών ο προγραμματιστής μπορεί να αντιληφθεί ποια είναι τα κύρια χαρακτηριστικά της εφαρμογής που θα κρίνουν την αρχιτεκτονική για την οποία πρέπει να σχεδιάσει την εφαρμογή του.

Χάρις το μοντέλο κατανάλωσης ισχύος που προτείνουμε ο προγραμματιστής έχει τη δυνατότητα με μεγάλη ακρίβεια να προβλέψει αν μια GPU-υλοποίηση της εφαρμογής του θα οδηγήσει σε μικρότερη κατανάλωση ισχύος.

9.2 Μελλοντικές Επεκτάσεις

Κατά την εκπόνηση της παρούσας διπλωματικής εργασίας ήρθαμε σε επαφή με ένα σύνολο από πραγματικά ενδιαφέρουσες θεματικές. Ασφαλώς η δουλειά που παρουσιάστηκε είναι πλήρης, είναι όμως και επεκτάσιμη:

- Αρχικά θα ήταν δυνατό να επεκταθεί το σύνολο των εφαρμογών που χρησιμοποιήσαμε για να συλλέξουμε τα δεδομένα μας. Έτσι τα μοντέλα θα εκφράζουν καλύτερα το software diversity των εφαρμογών που εκτελούνται στα σύγχρονα Datacenters.

- Μια σημαντική μελλοντική επέκταση θα ήταν να πραγματοποιήσουμε πειράματα σε περισσότερες ετερογενείς αρχιτεκτονικές, τόσο σε περισσότερους multicore CPUs όσο και σε GPUs.

- Το πιο καίριο βήμα θα ήταν να προβούμε σε επέκταση των μοντέλων που προτείναμε με την προσθήκη των μετρήσεων μιας τρίτης αρχιτεκτονικής που δεν εξετάσαμε, των FPGAs. Αυτό θα σήμαινε αναθεώρηση των προτεινόμενων μετρικών υλικού (hardware coefficients) έτσι ώστε να εκφράζουν τις κύριες αρχιτεκτονικές διαφορές για το σύνολο των τριών αυτών ετερογενών αρχιτεκτονικών.

- Για την παρούσα υλοποίηση θα ήταν χρήσιμο να ποσοτικοποιήσουμε την μείωση του χρόνου εκτέλεσης που έχουμε για την στοχευμένη ανάλυση μέσω του MICA, εν αντιθέσει με την πλήρη ανάλυση η οποία όπως διαπιστώσαμε για συγκεκριμένες εφαρμογές γίνεται απαγορευτική.

Κατά την εκτέλεση συγκεκριμένων εφαρμογών μας στου multicore CPUs παρατηρήσαμε ότι η χρήση μεγαλύτερων datasets δεν οδηγούσε σε καλύτερη αξιοποίηση των πυρήνων της αρχιτεκτονικής αυτής. Για να είμαστε σαφείς για τις εν λόγω εφαρμογές παρατηρήσαμε το μέγιστο speedup για τις εκτελέσεις όπου μόνο ένα υποσύνολο των υπολογιστικών πυρήνων αξιοποιούταν. Η εκτίμησή μας είναι ότι αυτό το φαινόμενο είναι άρρηκτα συνδεδεμένο με το κόστος της επικοινωνίας μεταξύ των υπολογιστικών πόρων. Θα ήταν λοιπόν πολύ ενδιαφέρον αλλά και χρήσιμο (στο πλαίσιο δέσμευσης πόρων ενός cluster) να αναπτυχθεί ένα μοντέλο που να εκτιμά πόσοι πυρήνες είναι αρκετοί για μια δεδομένη εφαρμογή, έτσι οι διαθέσιμοι υπολογιστικοί πόροι ενός συστήματος δε θα μένουν ανενεργοί αλλά αντίθετα θα μένουν ελεύθεροι προς εκμετάλλευση για άλλες εφαρμογές.

Παραρτήματα

Παράρτημα Α′

Αλγόριθμοι Κεφαλαίου 7

Στο Παράρτημα Α′ παραθέτουμε 4 ψευδοκώδικες που υλοποιούν τις μετρήσεις χρόνου και κατανάλωσης ισχύος αφενός για τους multicore CPUs και αφετέρου για τις GPUs.

Αλγοριθμος Α′.1: *Παράδειγμα μέτρησης του χρόνου εκτέλεσης για OpenMP*

```
// include all necessary libraries
#include <omp.h>
define N = 10000;
// forward declarations of functions, variables etc.

int main()
{
    int i;
    double start, finish, total;

    start = omp_get_wtime(); // time measured in seconds
    // Kernel
    #pragma omp parallel for
    for (i = 0; i < N; i++){
        function_A(arguements); // parallel workload
    }

    finish = omp_get_wtime();
    total = finish - start;
    printf("Kernel-execution-time=%lf(sec)\n", &total);

    return 0;
}
```

Αλγοριθμος Α΄.2: *Παράδειγμα μέτρησης της εφαρμογής για CUDA*

```
// include all necessary libraries (and kernels' code)
#include <sys/time.h>
#include <cuda.h>
// forward declarations of functions, variables etc.
double gettime() {  //returns time measured in seconds
    struct timeval t;
    gettimeofday(&t,NULL);
    return t.tv_sec+t.tv_usec*1e-6;
}

int main()
{
    int i;
    double allocations, host_to_gpu, kernel_start;
    double gpu_to_host, free_t, end;
    //Setup grid and threads parameters for the CUDA kernel

    allocations = gettime();
    // Allocations of GPU's buffers with cudaMalloc(...)

    host_to_gpu = gettime();
    // Copy data from host CPU to GPU using the allocated buffers
    //                  with cudaMemcpy(...)

    kernel_start = gettime(); // parallel workload
    kernel_function_A_CUDA<<< grid, threads>>>(arguments);

    gpu_to_host = gettime(); //kernel endpoint
    // Copy data from GPU to host CPU using the allocated buffers
    //                  with cudaMemcpy(...)

    free_t = gettime();
    // Free the allocated GPU's buffers with cudaFree(...)

    end = gettime();

    printf("Total-time(sec):%lf\n", end - start);
    printf("CUDA-Mallocs(sec):%lf\n", host_to_gpu - allocations);
    printf("Host-to-GPU(sec):%lf\n", kernel_start - host_to_gpu);
    printf("Kernel-time(sec):%lf\n", gpu_to_host - kernel_start);
    printf("GPU-to-Host(sec):%lf\n", free_t - gpu_to_host);
    printf("Free-Time(sec):%lf\n", end - free_t);

    return 0;
}
```

Αλγοριθμος Α΄.3: *Παράδειγμα μέτρησης του ισχύος για τους multicore CPUs*

```
// include all necessary libraries
#include <papi.h>
#include <omp.h>
define N = 10000;
// forward declarations of functions, variables etc.
int main()
{
    double start, finish, total, power;
    int i, ret;
    int ec[4]; //event code
    long long values[4] = { 0, 0, 0, 0};
    int event_set = PAPI_NULL;
    if ((ret=PAPI_library_init(PAPI_VER_CURRENT))<0) {
        fprintf(stderr,"PAPI_ERROR=%d,%s\n",ret,PAPI_strerror(ret));
        exit(1);
    }
    //Energy on chips and DRAMs//
    PAPI_event_name_to_code("rapl:::PACKAGE_ENERGY:PACKAGE0",&ec[0]);
    PAPI_event_name_to_code("rapl:::PACKAGE_ENERGY:PACKAGE1",&ec[1]);
    PAPI_event_name_to_code("rapl:::DRAM_ENERGY:PACKAGE0",&ec[2]);
    PAPI_event_name_to_code("rapl:::DRAM_ENERGY:PACKAGE1",&ec[3]);

    PAPI_event_info_t einfo;
    PAPI_get_event_info(ec[0], &einfo);
    if ((ret = PAPI_create_eventset(&event_set)) != PAPI_OK) {
        fprintf(stderr, "Error %d creating event set\n", ret);
        exit(-1);
    }
    if ((ret = PAPI_add_events(event_set, ec, 4)) != PAPI_OK) {
        fprintf(stderr, "Error %d adding event to set\n", ret);
        exit(-1);
    }
    PAPI_start(event_set);
    start = omp_get_wtime(); // time measured in seconds
    // Kernel
    #pragma omp parallel for
    for (i = 0; i < N; i++){
        function_A(arguements); // parallel workload
    }
    total = omp_get_wtime() - start;
    PAPI_stop(event_set, values);
    printf("PACKAGE_ENERGY:PACKAGE0(J)=%lf\n", values[0]*(1e-9));
    printf("PACKAGE_ENERGY:PACKAGE1(J)=%lf\n", values[1]*(1e-9));
    printf("DRAM_ENERGY:PACKAGE0(J)=%lf\n", values[2]*(1e-9));
    printf("DRAM_ENERGY:PACKAGE1(J)=%lf\n", values[3]*(1e-9));
    power=(values[0]+values[1]+values[2]+values[3])*(1e-9)/(time);
    printf("Power(W)=%lf\n", power);
    return 0;
}
```

Αλγοριθμος Α΄.4: *Παράδειγμα μέτρησης ισχύος με χρήση του nvml*

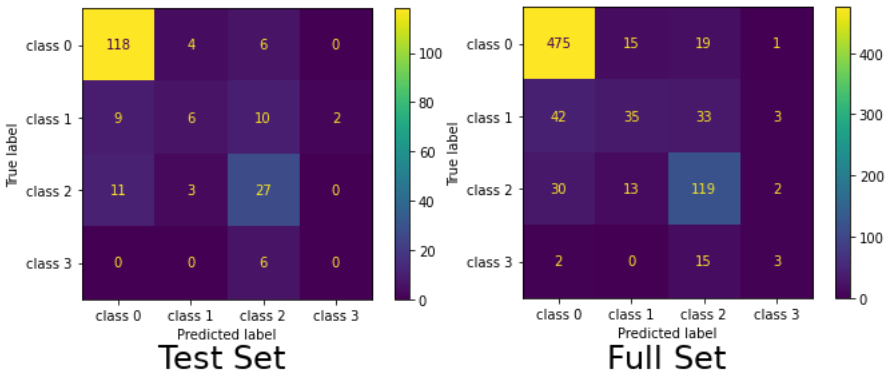
```
// include all necessary libraries (and kernels' code)
#include "./nvmlPower.h"
#include <cuda.h>
// forward declarations of functions, variables etc.

int main()
{
    //Setup grid and threads parameters for the CUDA kernel
    nvmlAPIRun();
    // Allocations of GPU's buffers with cudaMalloc(...)
    // Copy data from host CPU to GPU with cudaMemcpy(...)
    kernel_function_A_CUDA<<< grid, threads>>>(arguments);
    // Copy data from GPU to host CPU with cudaMemcpy(...)
    // Free the allocated GPU's buffers with cudaFree(...)
    nvmlAPIEnd();
    return 0;
}
```

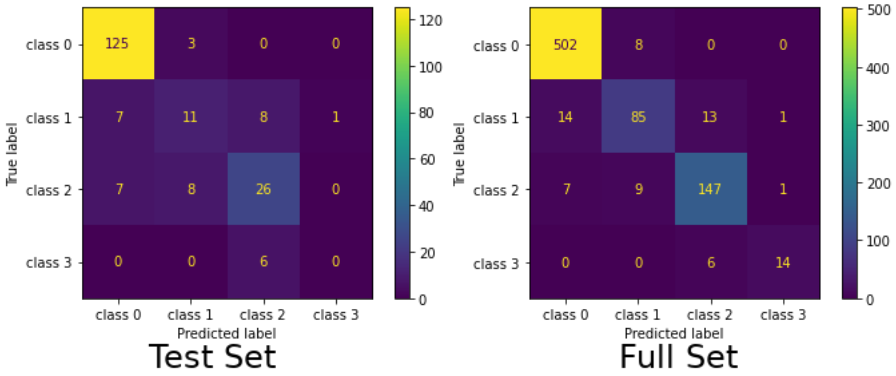
Παράρτημα Β′

Confusion Matrix ενδιάμεσων υλοποιήσεων

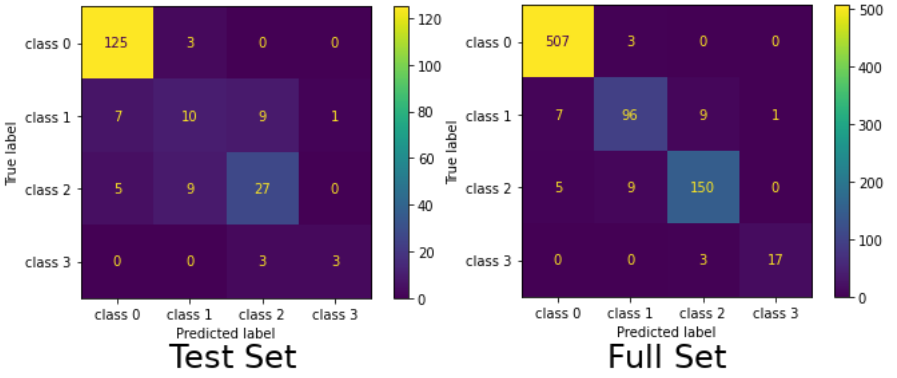
Στο κεφάλαιο αυτό παρουσιάζονται οι Confusion Matrix που προέκυψαν για το μοντέλο απόδοσης για τις υλοποιήσεις 2, 3, 4 (Πίνακες 8.1, Πίνακας 8.2) τόσο για τον Random Forest Classifier όσο και για τον Decision Tree Classifier. Τα διαγράμματα αυτά έχουν παραχθεί για $test\_size = 25\%$.



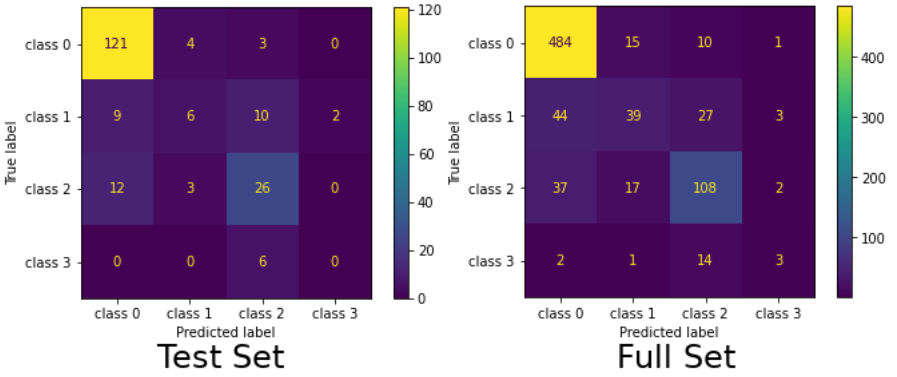
Σχήμα Β′.1: *Confusion Matrix για το Model 2 (RFC)*



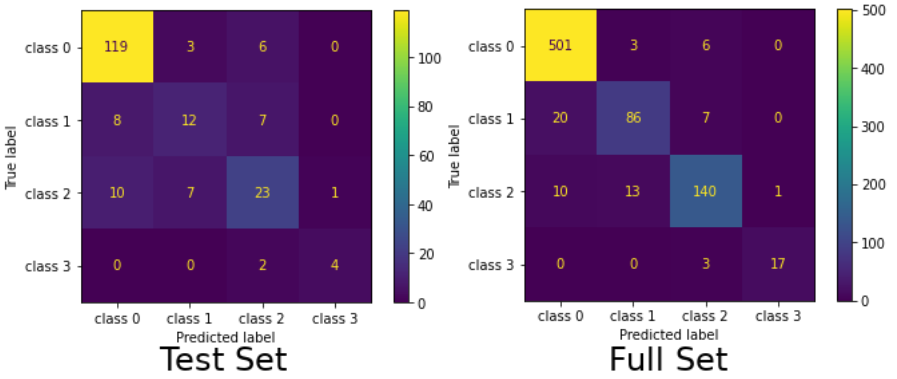
Σχήμα Β′.2: *Confusion Matrix για το Model 3 (RFC)*



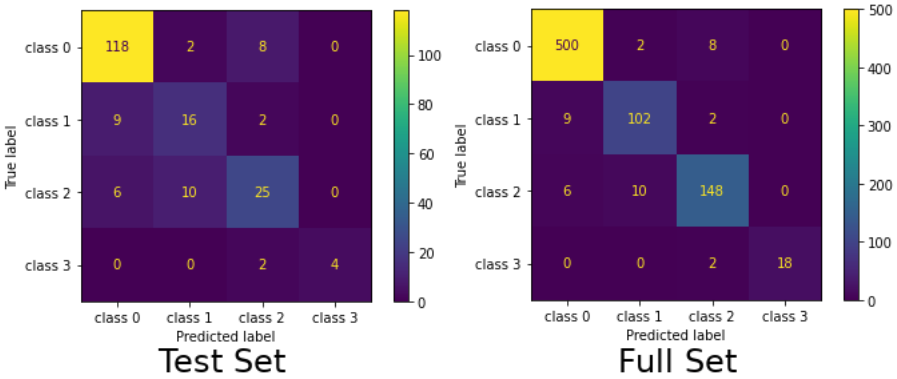
Σχήμα Β΄.3: *Confusion Matrix για το Model 4 (RFC)*



Σχήμα Β΄.4: *Confusion Matrix για το Model 2 (DT)*



Σχήμα Β΄.5: *Confusion Matrix για το Model 3 (DT)*



Σχήμα Β΄.6: *Confusion Matrix για το Model 4 (DT)*

Παράρτημα Γ′

Μέτρηση EDP για την Tesla m2050

Εξέτασα τις μετρούμενες τιμές της ισχύος για το σύνολο των εφαρμογών τόσο για την Tesla K40c(Kepler), όσο και για την Quadro m4000(Maxwell). Τα TDP ήταν 245W για την Tesla K40c και 120W για την Quadro m4000. Το TDP για την παλιότερη κάρτα (Tesla m2050) ήταν 225W. Ο κόμβος της Tesla m2050 έχει cuda 7.5 οπότε δεν υπάρχει υποστήριξη για την <nvml.h> (NVIDIA Management Library) για να πάρουμε μετρήσεις ισχύος. Επίσης, να επισημάνω ότι η Fermi είναι παλιότερη εκ των τριών εξεταζόμενων οικογενειών και είναι στα 40 nm έναντι των 28 nm (Kepler και Maxwell). Αυτό λοιπόν δικαιολογεί μια πιο πεσιμιστική προσέγγιση για την εκτίμηση της κατανάλωσής της.

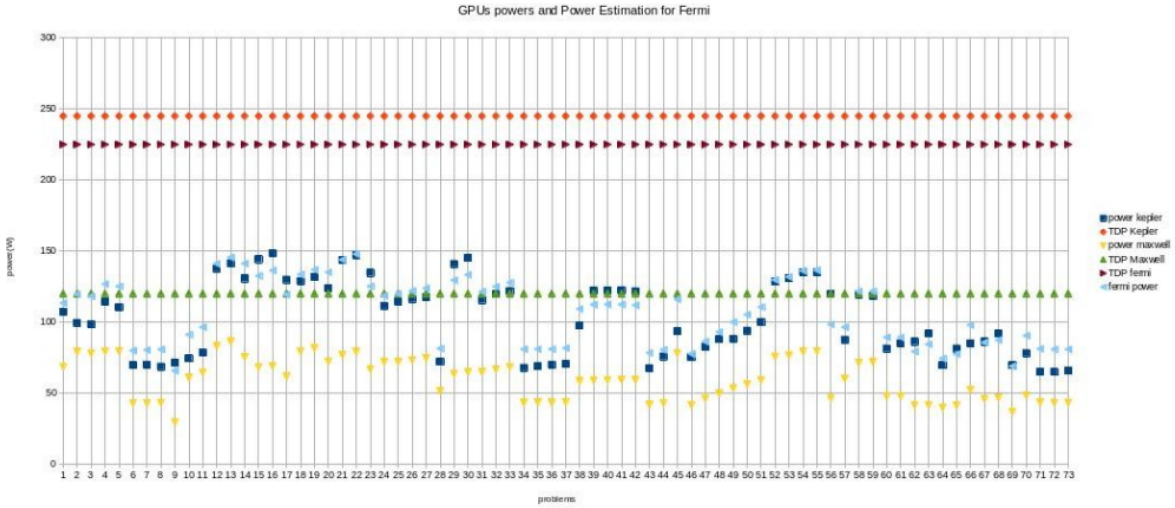
Η εικόνα που έχουμε για τις ΓΠΥ που έχουμε λάβει μετρήσεις είναι η παρακάτω. Όπου οι οριζόντιες γραμμές είναι τα TDP μεγέθη, ενώ με κίτρινο απεικονίζεται η συμπεριφορά της Quadro m4000 και με μπλε η συμπεριφορά της Tesla K40c. Στον οριζόντιο άξονα έχουμε αρίθμηση των διαφορετικών application-dataset ζευγών.

Σχήμα Γ′.1: *Κατανάλωση ισχύος για τις Tesla K40c και Quadro m4000*

Η προσέγγιση είναι η εξής:

$$PowerPercentage_{GPU[i]} = MeasuredPower_{GPU[i]} over TDP_{GPU[i]}$$

Αν για τις δύο κάρτες που έχουμε μετρήσεις τα παραπάνω ποσοστά διαφέρουν λιγότερο από 10% τότε θέτουμε το PowerPercentage της Fermi ίσο με το μεγαλύτερο εκ των δύο ποσοστών. Αν η διαφορά είναι μεγαλύτερη του 10% επειδή έχουμε σημαντική διαφορά στα TDP των δύο καρτών θέτουμε το PowerPercentage της Fermi ίσο με το μέσο όρο των δύο ποσοστών. Δεδομένων λοιπόν του PowerPercentage και του TDP της Fermi μπορούμε να παράξουμε την ισχύ (και την ενέργεια και το EDP). Συνολικά για τις τρεις GPUs η εικόνα είναι η ακόλουθη:



Σχήμα Γ΄.2: *Κατανάλωση ισχύος για όλες τις GPUs*

Παράρτημα Δ΄

Cost Complexity Pruning

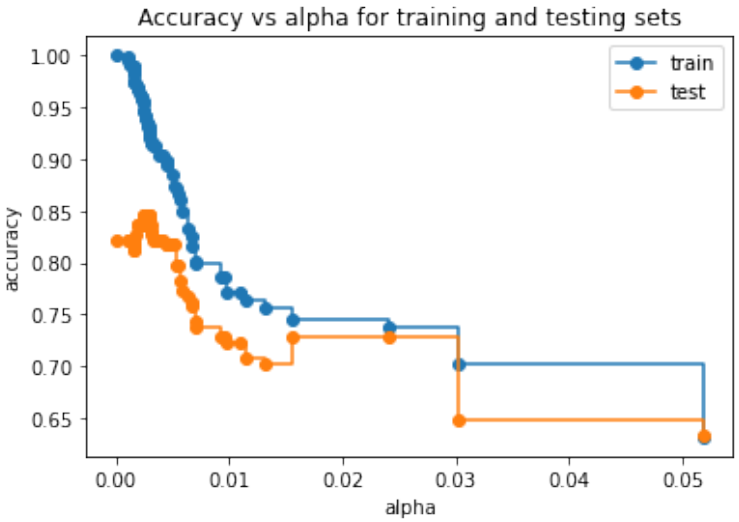
Για να προχωρήσουμε στο κλάδεμα του δέντρου αποφάσεων που προκύπτει από την εκπαίδευση και χρήση του Decision Tree Classifier ακολουθούμε την απλή διαδικασία που περιγράφουμε παρακάτω. Χρησιμοποιούμε Cost Complexity Pruning (CCP) το οποίο στηρίζεται στη λογική του Cross Validation [39, 40]. Η πλήρης υλοποίηση μπορεί να μελετηθεί στο online αρχείο της διπλωματικής [33]. Εδώ θα αρκεστούμε στο να παραθέσουμε τις κεντρικές ενέργειες που πραγματοποιούμε για να πραγματοποιήσουμε το pruning για παράδειγμα του Model Version 5 της υλοποίησης με τον Decision Tree Classifier για το μοντέλο απόδοσης.

```
1.path = clf_dt5.cost_complexity_pruning_path(X5_tr,y5_tr)
2.ccp_alphas = path.ccp_alphas
3.ccp_alphas = ccp_alphas[:-1]
```

Στα βήματα αυτά πραγματοποιούμε μια αναζήτηση των τιμών της παραμέτρου ccp_alpha. Όσο μεγαλύτερες είναι αυτές οι τιμές τόσο περισσότερο pruning επιφέρουμε στο δέντρο αποφάσεων. Για να αποκλείσουμε την ρίζα του δέντρου η οποία φέρει και τη μεγαλύτερη τιμή της παραμέτρου ccp_alpha εκτελούμε την τρίτη εντολή.

```
4.clf_dts = []
5.for ccp_alpha in ccp_alphas:
    clf_dt5 = DecisionTreeClassifier(random_state=42, ccp_alpha=ccp_alpha)
    clf_dt5.fit(X5_tr,y5_tr)
    clf_dts.append(clf_dt5)
6.train_scores = [clf_dt5.score(X5_tr, y5_tr) for clf_dt5 in clf_dts]
  test_scores = [clf_dt5.score(X5_t, y5_t) for clf_dt5 in clf_dts]
7.fig, ax = plt.subplots()
  ax.set_xlabel("alpha")
  ax.set_ylabel("accuracy")
  ax.set_title("Accuracy vs alpha for training and testing sets")
  ax.plot(ccp_alphas, train_scores, marker="o",label="train",drawstyle="steps-post")
  ax.plot(ccp_alphas, test_scores, marker="o",label="test",drawstyle="steps-post")
  ax.legend()
  plt.show()
```

Με τις παραπάνω εντολές υπολογίζουμε τις ακρίβειες (test, full set) που παίρνουμε για τα διαφορετικά δέντρα αποφάσεων που προκύπτουν για τις διαφορετικές τιμές της παραμέτρου ccp_alpha. Τέλος, σχεδιάζουμε ένα διάγραμμα των ακριβειών για τα δύο set.



Σχήμα Δ′.1: *Ακρίβειες για διαφορετικές τιμές της παραμέτρου ccp_alpha*

Τελικά, επιλέγουμε την τιμή του ccp_alpha που φαίνεται να δίνει την αφενός την καλύτερη δυνατή ακρίβεια για το test set και αφετέρου μια κοινή συμπεριφορά (από αυτήν και έπειτα) για τις ακρίβειες των test και training sets. Με την τιμή αυτή επανασχεδιάζουμε το δέντρο αποφάσεων.

Βιβλιογραφία

[1] *OpenMP API*. `https://www.openmp.org/`. Ημερομηνία πρόσβασης: 10-5-2021.

[2] *CUDA*. `https://developer.nvidia.com/cuda-zone`. Ημερομηνία πρόσβασης: 10-5-2021.

[3] Andrew Waterman Samuel Williams και David Patterson. *Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures. Communications of the ACM*, 52(4):–, 2008.

[4] Erik Altman Ioana Baldini, Stephen J. Fink. *Should I Use a GPU? Predicting GPU Performance from CPU Runs.* Τεχνική Αναφορά με αριθμό RC25487 (WAT1408-021), IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598 USA, 2014.

[5] Karthikeyan Sankaralingam Xiaojin Zhu Newsha Ardalani, Clint Lestourgeon. *Cross-Architecture Performance Prediction (XAPP) Using CPU Code to Predict GPU Performance. MICRO-48: Proceedings of the 48th International Symposium on Microarchitecture*, Waikiki, HI, USA, 2015.

[6] J. Meng D. Tarjan J. W. Sheaffer S. H. Lee S. Che, M. Boyer και K. Skadron. *Rodinia: A Benchmark Suite for Heterogeneous Computing. In Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 52(4):44–54, 2009.

[7] *Rodinia Benchmark Suite 3.1*. `http://www.cs.virginia.edu/rodinia/doku.php`. Ημερομηνία πρόσβασης: 13-5-2021.

[8] K. Asanovicet al. *The landscape of parallel computing research: A view from Berkeley.* Τεχνική Αναφορά με αριθμό UCB/EECS-2006-183, EECS Department, University of California, Berkeley, California, 2006.

[9] *MICA: Microarchitecture-Independent Characterization of Applications.* `https://users.ugent.be/~kehoste/ELIS/mica/`. Ημερομηνία πρόσβασης: 14-5-2021.

[10] *MICA: Microarchitecture-Independent Characterization of Applications, implementaion.* `https://github.com/boegel/MICA`. Ημερομηνία πρόσβασης: 14-5-2021.

[11] *Pin - A Dynamic Binary Instrumentation Tool.* `https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html`. Ημερομηνία πρόσβασης: 15-5-2021.

[12] Kenneth Hoste και Lieven Eeckhout. *Microarchitecture-Independent Workload Characterization. IEEE Micro Hot Tutorials*, 27(3):63–72, 2007.

[13] B. Johnston και J. Milthorpe. *AIWC: OpenCL-Based Architecture-Independent Workload Characterization. 2018 IEEE/ACM 5th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC)*, σελίδες 81–91, 2018.

[14] *PAPI User's Guide*. `http://icl.cs.utk.edu/projects/papi/files/documentation/PAPI_USER_GUIDE_23.htm#INTRODUCTION_TO_PAPI`. Ημερομηνία πρόσβασης: 15-5-2021.

[15] *NVML API Reference*. `https://docs.nvidia.com/deploy/nvml-api/index.html`. Ημερομηνία πρόσβασης: 15-5-2021.

[16] V. M. Weaveret al. *Measuring Energy and Power with PAPI. 2012 41st International Conference on Parallel Processing Workshops*, σελίδες 262–268, 2012.

[17] *NVIDIA System Management Interface*. `https://developer.nvidia.com/nvidia-system-management-interface`. Ημερομηνία πρόσβασης: 15-5-2021.

[18] *Decision Tree*. `https://scikit-learn.org/stable/modules/tree.html#classification`. Ημερομηνία πρόσβασης: 16-5-2021.

[19] *Forests of randomized trees*. `https://scikit-learn.org/stable/modules/ensemble.html#forests-of-randomized-trees`. Ημερομηνία πρόσβασης: 16-5-2021.

[20] *sci-kit learn, Machine Learning in Python*. `https://scikit-learn.org/stable/`. Ημερομηνία πρόσβασης: 16-5-2021.

[21] *Intel(R) Xeon(R) CPU E5-4620*. `https://ark.intel.com/content/www/us/en/ark/products/64607/intel-xeon-processor-e5-4620-16m-cache-2-20-ghz-7-20-gt-s-intel-qpi.html`. Ημερομηνία πρόσβασης: 22-5-2021.

[22] *Intel(R) Xeon(R) CPU E5-2697 v3*. `https://ark.intel.com/content/www/us/en/ark/products/81059/intel-xeon-processor-e5-2697-v3-35m-cache-2-60-ghz.html`. Ημερομηνία πρόσβασης: 22-5-2021.

[23] *Intel(R) Xeon(R) CPU E5-2630 v4*. `https://ark.intel.com/content/www/us/en/ark/products/92981/intel-xeon-processor-e5-2630-v4-25m-cache-2-20-ghz.html`. Ημερομηνία πρόσβασης: 22-5-2021.

[24] *Sandy Bridge*. `https://en.wikichip.org/wiki/intel/microarchitectures/sandy_bridge_(client)`. Ημερομηνία πρόσβασης: 22-5-2021.

[25] *Haswell*. `https://en.wikichip.org/wiki/intel/microarchitectures/haswell_(client)`. Ημερομηνία πρόσβασης: 22-5-2021.

[26] *Broadwell*. `https://en.wikichip.org/wiki/intel/microarchitectures/broadwell_(client)`. Ημερομηνία πρόσβασης: 22-5-2021.

[27] *The microarchitecture of Intel, AMD and VIA CPUs.* https://www.agner.org/optimize/microarchitecture.pdf. Ημερομηνία πρόσβασης: 23-5-2021.

[28] *Intel Xeon.* https://www.cpu-world.com/CPUs/Xeon/index.html. Ημερομηνία πρόσβασης: 23-5-2021.

[29] *Tesla m2050.* https://www.techpowerup.com/gpu-specs/tesla-m2050.c1534. Ημερομηνία πρόσβασης: 23-5-2021.

[30] *Tesla K40c.* https://www.techpowerup.com/gpu-specs/tesla-k40c.c2505. Ημερομηνία πρόσβασης: 23-5-2021.

[31] *Quadro m4000.* https://www.techpowerup.com/gpu-specs/quadro-m4000.c2757. Ημερομηνία πρόσβασης: 23-5-2021.

[32] *Plotting with categorical data.* https://seaborn.pydata.org/tutorial/categorical.html. Ημερομηνία πρόσβασης: 30-5-2021.

[33] *Diploma Thesis of Athanasios Markou online repository.* https://gitlab.com/athamark96/diploma.git. Ημερομηνία πρόσβασης: 1-6-2021.

[34] *omp_get_wtime.* https://www.openmp.org/spec-html/5.0/openmpsu160.html. Ημερομηνία πρόσβασης: 31-5-2021.

[35] *gettimeofday(2) - Linux manual page.* https://man7.org/linux/man-pages/man2/gettimeofday.2.html. Ημερομηνία πρόσβασης: 31-5-2021.

[36] *POSIX Threads (pthread) libraries.* https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html. Ημερομηνία πρόσβασης: 1-6-2021.

[37] *sklearn's train_test_split.* https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html. Ημερομηνία πρόσβασης: 1-6-2021.

[38] *Jupyter Notebook.* https://jupyter.org/. Ημερομηνία πρόσβασης: 14-6-2021.

[39] *Post pruning decision trees with cost complexity pruning.* https://scikit-learn.org/stable/auto_examples/tree/plot_cost_complexity_pruning.html#post-pruning-decision-trees-with-cost-complexity-pruning. Ημερομηνία πρόσβασης: 4-6-2021.

[40] *Cross-validation: evaluating estimator performance.* https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation-evaluating-estimator-performance. Ημερομηνία πρόσβασης: 14-6-2021.

Συντομογραφίες - Αρκτικόλεξα - Ακρωνύμια

κλπ.	και λοιπά
κ.ο.κ.	και ούτω καθεξής
γρ.	γραμμή
TDP	Thermal Design Power
EDP	Energy Delay Product
CPU	Central Processing Unit
GPU	Graphics Processing Unit
HPC	High Performance Computing
RFC	Random Forest Classifier
DT	Decision Tree
RAPL	Running Average Power Limit
MSR	Model Specific Register
SM	Streaming Multiprocessor

Απόδοση ξενόγλωσσων όρων

Απόδοση	Ξενόγλωσσος όρος
μεταβλητές περιβάλλοντος	environmental variables
βρόχος επανάληψης	loop
πολυπύρηνος επεξεργαστής	multicore CPU
επεξεργαστής	CPU
κάρτα γραφικών	GPU
προγραμματιστικός κόπος	coding effort
επιτάχυνση	speedup
εξαρτώμενος από την μικροαρχιτεκτονική	microarchitecture-dependent
ανεξάρτητος από την μικροαρχιτεκτονική	microarchitecture-independent
επιταχυντής υλικού	hardware accelerator
ποικιλομορφία	diversity
κατανάλωση ισχύος	power consumption
καθυστέρηση	overhead
συμβάν	event
μετρητής	counter
κλάση	class
δυαδικό	binary
υπό επίβλεψη	supervised
δέντρο απόφασης	decision tree
κλάδεμα	pruning
δρομολογητής	scheduler
εξάρτηση των δεδομένων	data dependency
τμήματα, μονάδες	units
αδρανής	idle
καταχωρητής	register
χρόνος εκτέλεσης	runtime
κουβάς	bucket