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

ACUTE: Δυναμική Ομαδοποίηση Εφαρμογών για τον Αποδοτικό Διαμοιρασμό της Κρυφής Μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεωργία Θ. Πανουτσακοπούλου

Επιβλέπων : Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2018

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

ACUTE: Δυναμική Ομαδοποίηση Εφαρμογών για τον Αποδοτικό Διαμοιρασμό της Κρυφής Μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Γεωργία Θ. Πανουτσακοπούλου

Επιβλέπων : Γεώργιος Γκούμας
Επ. Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 14η Μαρτίου 2018.

.....	.....	.....
Γεώργιος Γκούμας Επ. Καθηγητής Ε.Μ.Π.	Νεκτάριος Κοζύρης Καθηγητής Ε.Μ.Π.	Δημήτριος Σούντρης Αν. Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2018

Γεωργία Θ. Πανουτσακοπούλου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Η βελτίωση της απόδοσης των σύγχρονων πολυεπεξεργαστικών συστημάτων έχει αποτελέσει πηγή έρευνας την τελευταία δεκαετία. Η ταυτόχρονη εκτέλεση πολλών εφαρμογών στο ίδιο chip, έχει οδηγήσει στη μείωση της απόδοσης του συστήματος λόγω της διαμάχης που δημιουργείται μεταξύ των εφαρμογών για τους κοινόχρηστους πόρους, όπως το τελευταίο επίπεδο της cache (LLC). Γι' αυτόν τον λόγο, πολλές μελέτες έχουν επικεντρωθεί στον δυναμικό διαμοιρασμό της κοινόχρηστης cache μεταξύ των συνεκτελούμενων εφαρμογών.

Στην παρούσα διπλωματική εργασία, αφού παρουσιαστούν ορισμένες από τις παραπάνω μελέτες, αναλύονται χαρακτηριστικά εφαρμογών σε σχέση με το ποσοστό της LLC που έχουν διαθέσιμο και έπειτα πραγματοποιείται συνεκτέλεση εφαρμογών σε πραγματικό μηχάνημα για την ανάδειξη του προβλήματος που αναφέρθηκε. Στη συνέχεια παρουσιάζεται ο ACUTE (Adaptive Clustering for UTility-based cachE partitioning), ένας χαμηλού κόστους, αποδοτικός μηχανισμός που πραγματοποιεί δυναμική ομαδοποίηση εφαρμογών για τον διαχωρισμό της cache με βάση τη χρησιμότητα και όχι τη ζήτησή της. Επιπλέον ο μηχανισμός ACUTE κλιμακώνεται με την αύξηση του αριθμού των πυρήνων και κατ' επέκταση την αύξηση των συνεκτελούμενων εφαρμογών. Τέλος, για την αξιολόγηση του μηχανισμού ACUTE, αλλά και την πραγματοποίηση όλων των πειραμάτων της εργασίας, χρησιμοποιήθηκαν οι νέες τεχνολογίες της Intel, Cache Monitoring Technology (CMT) και Cache Allocation Technology (CAT), οι οποίες προσφέρουν την απαραίτητη υποστήριξη στο υλικό για την παρακολούθηση και την κατανομή της LLC στις εφαρμογές αντίστοιχα.

Λέξεις κλειδιά

Πολυεπεξεργαστικά συστήματα, Κοινόχρηστη κρυφή μνήμη, LLC, Διαμοιρασμός κρυφής μνήμης, Συνεκτέλεση εφαρμογών, UMON, Ομαδοποίηση εφαρμογών, Συνδυασμένες καμπύλες αστοχιών, ACUTE, Intel CMT-CAT

Abstract

Improving the performance of current chip multiprocessors has been a source of research over the last decade. The concurrent execution of multiple applications on a single chip can significantly degrade system performance due to inter-application interference in shared resources, such as the last level cache (LLC). To address this problem, many studies have focused on dynamic partitioning of the shared cache among co-running applications.

In this diploma thesis, once the above studies are presented, we venture to investigate application characteristics according to the utility of the available LLC and then execute concurrently multiple applications in a real system in order to verify the existence of the aforementioned problem. Furthermore, we propose Adaptive Clustering for UTility-based cachE partitioning (ACUTE), a low-overhead, efficient mechanism that allocates the shared cache to applications based on benefit rather than demand. Moreover, ACUTE is scalable to a large number of cores and hence a large number of competing applications. Finally, for the purpose of this thesis and especially the evaluation of ACUTE, Intel's new technologies, Cache Monitoring Technology (CMT) and Cache Allocation Technology (CAT), have been used to provide the necessary hardware support for monitoring and allocation of LLC respectively.

Key words

Chip multiprocessors CMPs, Shared cache, LLC, Cache partitioning, Concurrently executing applications, UMON, Application clustering, Combined miss curves, Adaptive clustering for utility-based cache partitioning, ACUTE, Intel CMT-CAT

Ευχαριστίες

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας έλαβε τέλος ένα σημαντικό κεφάλαιο της ζωής μου και της ακαδημαϊκής μου πορείας. Σε αυτό το σημείο θα ήθελα να ευχαριστήσω όλα τα μέλη του Εργαστηρίου Υπολογιστικών Συστημάτων του Εθνικού Μετσόβιου Πολυτεχνείου, που ήταν δίπλα μου σε όλη τη διάρκεια της εργασίας μου.

Στον επιβλέποντα καθηγητή μου, κ. Γεώργιο Γκούμα, θα ήθελα να απευθύνω τις θερμές μου ευχαριστίες για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα σύγχρονο, ερευνητικό θέμα μεγάλου ενδιαφέροντος αλλά και τις ουσιώδεις γνώσεις που μου μετέδωσε σε όλη τη διάρκεια των σπουδών μου. Στον Δρ. Κωνσταντίνο Νίκα και Δρ. Βασίλη Καρακώστα χρωστάω ένα μεγάλο ευχαριστώ για τη συνεχή παρουσία τους, την ακατάπαυστη επιστημονική καθοδήγησή τους και την αμέριστη βοήθεια που μου προσέφεραν σε όλα τα στάδια της εργασίας μου.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου και όλους τους κοντινούς μου ανθρώπους, φίλους και συμφοιτητές, για την έμπρακτη και αδιάκοπη υποστήριξή τους όλον αυτόν τον καιρό.

Γεωργία Θ. Πανουτσακοπούλου,

Αθήνα, 14η Μαρτίου 2018

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Προκλήσεις στα συστήματα πολλαπλών επεξεργαστών

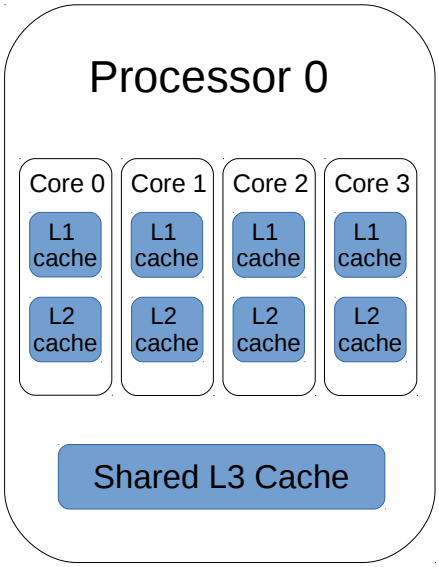
Οι τεχνολογικές επιτεύξεις στο πεδίο των υπολογιστικών συστημάτων έχουν οδηγήσει στο σχεδιασμό και την υλοποίηση συστημάτων πολλαπλών επεξεργαστών. Οι πλέον σύγχρονοι επεξεργαστές περιλαμβάνουν πολλαπλούς πυρήνες σε ένα τσιπ (chip), το οποίο τους επιτρέπει να εκτελούν διαφορετικές εφαρμογές ταυτόχρονα. Με αυτόν τον τρόπο ξεπεράστηκαν οι περιορισμοί του παραλληλισμού σε επίπεδο εντολών και ενισχύθηκε η επίδοση των επεξεργαστών. Καθώς ο αριθμός των πυρήνων σε ένα chip αυξάνεται εκθετικά σύμφωνα με το νόμο του Moore, αυξάνεται και ο αριθμός των εφαρμογών που εκτελούνται σε αυτό. Μία πρόκληση για τα συστήματα αυτά αποτελεί το γεγονός πως οι συνεκτελούμενες εφαρμογές αναπόφευκτα θα μοιραστούν ορισμένους πόρους στο υλικό (hardware), όπως την προσωρινή μνήμη, το δίκτυο διασυνδέσεων και τον προϋπολογισμό ισχύος στο chip αλλά και το bandwidth της κύριας μνήμης. Εάν οι πυρήνες έχουν ελεύθερη πρόσβαση στους κοινούς πόρους χωρίς κάποιον περιορισμό, τότε η διαμάχη για τους διαμοιραζόμενους πόρους μπορεί να επηρεάσει την απόδοση του συστήματος και την ποιότητα παροχής υπηρεσιών. Επομένως το κλειδί για την αντιμετώπιση αυτού του προβλήματος είναι ο αποδοτικός διαχωρισμός των κοινόχρηστων πόρων μεταξύ των συνεκτελούμενων εφαρμογών.

Ένας βασικός διαμοιραζόμενος πόρος μεταξύ των πυρήνων ενός chip είναι το τελευταίο επίπεδο της προσωρινής μνήμης (last level cache, LLC). Οι σύγχρονοι πολυπύρηνοι επεξεργαστές συνήθως υλοποιούν πολύ μεγάλου μεγέθους LLC προκειμένου να περιορίσουν την καθυστέρηση που προκαλείται από την πρόσβαση στην εκτός chip κύρια μνήμη. Λόγω αυτού του μεγάλου μεγέθους δεν είναι εφικτή η υλοποίηση μίας ιδιωτικής LLC για κάθε πυρήνα και έτσι οι συνεκτελούμενες εφαρμογές ανταγωνίζονται μεταξύ τους για την απόκτηση χώρου στην κοινόχρηστη LLC. Επομένως οι εφαρμογές μπορούν να αντικαταστήσουν δεδομένα της LLC που ανήκουν σε διαφορετικές εφαρμογές, γεγονός που μπορεί να έχει σοβαρό αντίκτυπο στην απόδοση των τελευταίων. Επιπλέον η πρόβλεψη της αλληλεπίδρασης των εφαρμογών αυτών είναι δύσκολη, αφού εξαρτάται από τα χαρακτηριστικά της κάθε μίας εφαρμογής. Γι’ αυτόν τον λόγο πολλές έρευνες έχουν πραγματοποιηθεί για την εύρεση ενός τρόπου διαχωρισμού της LLC ώστε να αντιμετωπίσουν το παραπάνω πρόβλημα.

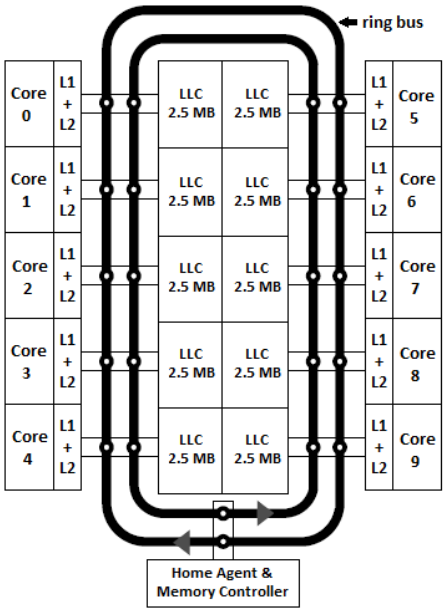
Όλα τα παραπάνω αποτελούν κίνητρο της παρούσας διπλωματικής, η οποία επικεντρώνεται στην ανάλυση των χαρακτηριστικών των εφαρμογών και την πρόταση ενός μηχανισμού για τον διαχωρισμό της LLC στις συνεκτελούμενες εφαρμογές, με στόχο τη βελτίωση της συνολικής απόδοσης ενός πολυπύρηνου συστήματος. Επιπλέον κίνητρο της εργασίας αποτελεί η πρόσφατη εισαγωγή νέων τεχνολογιών υλικού CMT-CAT [1] σε επεξεργαστές της Intel για την παρακολούθηση και τον πρακτικό περιορισμό της μνήμης που αποδίδεται σε κάθε εφαρμογή. Οι τεχνολογίες αυτές χρησιμοποιήθηκαν σε όλη τη διάρκεια της εργασίας και αποτελεί μέρος του προτεινόμενου μηχανισμού που θα παρουσιαστεί.

1.2 Ιεραρχία των caches και Ορολογία

Ένα από τα πιο σημαντικά συστατικά των υπολογιστών αποτελεί η μνήμη τους. Η μνήμη των υπολογιστών είναι ιεραρχικά δομημένη, δηλαδή αποτελείται από πολλά επίπεδα μνήμης με διαφορετικές ταχύτητες, χωρητικότητες και κόστη. Η ταχύτερη μνήμη βρίσκεται κοντά στον επεξεργαστή ενώ η πιο αργή βρίσκεται ιεραρχικά κάτω από αυτή. Η ταχύτερη και μικρότερη σε μέγεθος μνήμη ονομάζεται κρυφή-προσωρινή μνήμη (cache) και αντιπροσωπεύει το επίπεδο ιεραρχίας της μνήμης μεταξύ του επεξεργαστή και της κύριας μνήμης. Η μνήμη cache ξεκίνησε να υλοποιείται σε πολλαπλά επίπεδα όπως φαίνεται στην Εικόνα 1.1, όταν οι επεξεργαστές άρχισαν να αποτελούνται από πολλούς πυρήνες.



Εικόνα 1.1: Ιεραρχία της cache



Εικόνα 1.2: Μπλοκ διάγραμμα του επεξεργαστή Intel Xeon E5-2630 v4

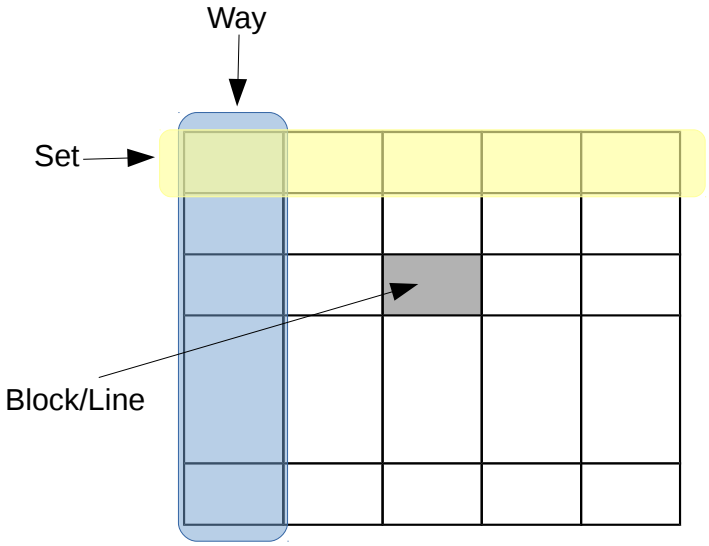
Ο επεξεργαστής αποθηκεύει προσφάτως χρησιμοποιούμενα δεδομένα στην ιεραρχία των caches προκειμένου να μειώσει το χρόνο αναζήτησής τους. Στους σύγχρονους πολυπύρηνους επεξεργαστές η μνήμη cache αποτελείται από τρία επίπεδα, όπου τα δύο πρώτα, L1 και L2, έχουν συνήθως μικρό μέγεθος και είναι ιδιωτικά για κάθε πυρήνα. Αντίθετα, το τρίτο επίπεδο, το οποίο ονομάζεται και τελευταίο επίπεδο της cache (last level cache, LLC), είναι διαμοιραζόμενο μεταξύ των πυρήνων και έχει μεγάλη χωρητικότητα της τάξης των Megabyte. Επιπλέον η LLC είναι inclusive, δηλαδή αποτελεί ένα υπερσύνολο των δεδομένων των προηγούμενων επιπέδων της cache. Πιο συγκεκριμένα, όταν ένας πυρήνας ζητά δεδομένα από την κύρια μνήμη, ένα αντίγραφο αυτών τοποθετείται σε κάθε επίπεδο της cache. Επομένως, αν ένα αίτημα για την απόκτηση δεδομένων αστοχήσει (πραγματοποιήσει ένα miss) στις L1 και L2 caches, τότε ελέγχεται και η LLC. Στην περίπτωση που αστοχήσει και στην LLC το αίτημα μεταφέρεται στην κύρια μνήμη, ενώ αν ευστοχήσει (πραγματοποιήσει hit) πρέπει να διατηρηθεί η συνοχή με τους υπόλοιπους πυρήνες που πιθανώς χρησιμοποιούν αυτά τα δεδομένα. Από την άλλη πλευρά, όταν απομακρυνθούν δεδομένα ενός πυρήνα από την LLC, π.χ. λόγω μεγαλύτερης χρήσης της LLC από άλλους πυρήνες, πρέπει αυτά να καταστούν άκυρα εφόσον υπάρχουν στην L1 και L2 cache του πυρήνα [2].

Παρακάτω αναφέρονται επιγραμματικά ορισμένοι όροι που χρησιμοποιούνται για την περιγραφή των caches και παρουσιάζονται στην Εικόνα 1.3:

- **Block/Line:** Το block ή line αποτελεί τη μικρότερη μονάδα μνήμης που μεταφέρεται μεταξύ της κύριας μνήμης και των caches. Συνήθως έχει μέγεθος 64 bytes και αποτελείται από ένα σύνολο δεδομένων και από μία ετικέτα (tag) που χρησιμοποιείται για την εύρεση του συγκεκριμένου block/line.

- **Set:** Το set είναι ένα σύνολο από blocks, των οποίων τα tags ελέγχονται ταυτόχρονα κατά την διάρκεια μιας πρόσβασης στην cache.

- **Way:** Ο αριθμός των ways ισούται με τον αριθμό των blocks που περιέχονται σε κάθε set και καθορίζει το βαθμό συσχετιστικότητας (associativity) της cache.



Εικόνα 1.3: Παράδειγμα 5-way associative cache

Επομένως, οι caches είναι οργανωμένες σε blocks (συνήθως μεγέθους 64 bytes) και είναι n-way associative, που σημαίνει πως κάθε block εισάγεται σε ένα καθορισμένο set ανάλογα με συγκεκριμένα bits της διεύθυνσής του και καταλαμβάνει μία από τις n θέσεις blocks αυτού του set. Όταν χρησιμοποιούνται όλες οι θέσεις ενός set τότε η πολιτική αντικατάστασης (replacement policy) αποφασίζει ποιο block θα απομακρυνθεί από την cache ώστε να υπάρχει κενή θέση για την εισαγωγή του νέου. Μία ευρέως χρησιμοποιούμενη πολιτική αντικατάστασης είναι η LRU (Least Recently Used), η οποία επιλέγει προς απομάκρυνση το block που έχει μείνει αχρησιμοποίητο για το μεγαλύτερο χρονικό διάστημα. Με την πάροδο του χρόνου, η πολιτική αυτή εξελίσσεται ενώ οι εταιρίες κατασκευής των επεξεργαστών δεν αποκαλύπτουν την ακριβή πολιτική που χρησιμοποιούν.

Ένα ακόμη σημαντικό χαρακτηριστικό των caches είναι η δομή του τελευταίου κοινόχρηστου επιπέδου τους. Στους τελευταίους πολυπύρηνους επεξεργαστές της Intel, όπως ο Intel Xeon E5-2630 v4 που χρησιμοποιήθηκε στην παρούσα διπλωματική εργασία, η LLC δεν είναι απλά δομημένη όπως η L1 και L2 με τον τρόπο που περιγράφηκε παραπάνω. Η LLC είναι χωρισμένη σε ”κομμάτια”, τα οποία ονομάζονται slices και αποτελούνται από sets όπως ακριβώς και τα υπόλοιπα επίπεδα της cache. Ο αριθμός των slices ισούται με τον αριθμό των πυρήνων, χωρίς όμως να υπάρχει απόλυτη αντιστοίχηση πυρήνα-slice, και συνδέονται μεταξύ τους αλλά και με τους πυρήνες μέσω ενός ring bus, όπως φαίνεται στην Εικόνα 1.2. Για την αντιστοίχιση μίας διεύθυνσης με ένα slice χρησιμοποιείται ένας μη δημοσιευμένος αλγόριθμος κατακερματισμού (hashing algorithm) ούτως ώστε να μειωθεί ο συνωστισμός

στην LLC αλλά και να αποφευχθούν κακόβουλες επιθέσεις μεταξύ των πυρήνων μέσω της κοινόχρηστης cache [3].

1.3 Οργάνωση κειμένου

Στο κεφάλαιο αυτό αναλύθηκαν οι προκλήσεις που αντιμετωπίζουν τα πολυεπεξεργαστικά συστήματα, δηλαδή το κίνητρο της παρούσας διπλωματικής εργασίας, και αναλύθηκαν βασικές έννοιες για τη δομή των caches. Τα επόμενα κεφάλαια δομούνται ως εξής:

- Στο Κεφάλαιο 2 παρουσιάζονται οι τεχνικές διαμοιρασμού της κοινόχρηστης LLC που έχουν προταθεί από την ερευνητική κοινότητα τα τελευταία χρόνια.
- Στο Κεφάλαιο 3 παρουσιάζονται οι νέες τεχνολογίες CMT-CAT της Intel για την παρακολούθηση και τον πρακτικό διαχωρισμό της LLC στο hardware.
- Στο Κεφάλαιο 4 αναλύεται η πειραματική μεθοδολογία που ακολουθήθηκε για την εύρεση χαρακτηριστικών των εφαρμογών και τη συνεκτέλεση εφαρμογών στο σύστημα πειραματικής αξιολόγησης.
- Στο Κεφάλαιο 5 περιγράφεται και αξιολογείται ο προτεινόμενος μηχανισμός, ACUTE, για τον διαμοιρασμό της κοινόχρηστης LLC κατά τη συνεκτέλεση εφαρμογών στο σύστημα.
- Τέλος, στο Κεφάλαιο 6 συγκεντρώνονται τα συμπεράσματα από την παρούσα εργασία και προτείνονται μελλοντικές επεκτάσεις της.

Κεφάλαιο 2

Μέθοδοι διαμοιρασμού της κοινόχρηστης cache

Με την αύξηση του αριθμού των πυρήνων στο chip αλλά και των απαιτήσεων μνήμης των εφαρμογών, η ανάγκη για ορθή διαχείριση των πόρων μνήμης έχει γίνει επιτακτική. Οι σύγχρονοι πολυπύρηνοι επεξεργαστές έχουν το τελευταίο επίπεδο της κρυφής τους μνήμης (last level cache, LLC) κοινόχρηστο μεταξύ των πυρήνων. Επομένως όταν εφαρμογές με διαφορετικές απαιτήσεις μνήμης εκτελούνται ταυτόχρονα στο ίδιο chip και διεκδικούν την κοινόχρηστη μνήμη, είναι πιθανή η μείωση της συνολικής απόδοσης του συστήματος. Συγκεκριμένα, ένας πυρήνας με υψηλό ρυθμό πρόσβασης στην LLC μπορεί να απομακρύνει από αυτή δεδομένα που χρησιμοποιούνται από άλλους πυρήνες, γεγονός που έχει αντίκτυπο στην επίδοση των πυρήνων αυτών αλλά και στη συνολική απόδοση του συστήματος. Η ευρέως χρησιμοποιούμενη πολιτική αντικατάστασης της cache LRU (Least Recently Used) κατανέμει τη μνήμη στις συνεκτελούμενες εφαρμογές με βάση το ρυθμό ζήτησης και για τον λόγο αυτό η LRU συχνά κατανέμει πόρους μνήμης σε εφαρμογές που δεν επωφελούνται από αυτούς. Επομένως μία τεχνική διαμοιρασμού βασισμένη στο όφελος και όχι στο ρυθμό ζήτησης της cache μπορεί να βελτιώσει αισθητά την απόδοση του συστήματος. Ως αποτέλεσμα, πολλοί ερευνητές [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18] έχουν προτείνει ένα μεγάλο εύρος τεχνικών διαχείρισης της LLC με σκοπό τη βελτίωση της απόδοση, δικαιοσύνης και ποιότητας υπηρεσιών του συστήματος.

Οι τεχνικές διαμοιρασμού της LLC στοχεύουν στο συνδυασμό των πλεονεκτημάτων των ιδιωτικών επιπέδων της cache, όπως η απομόνωση των δεδομένων, και των κοινόχρηστων, όπως η μεγάλη χωρητικότητα. Είναι γνωστό πως διαφορετικές εφαρμογές, ή ακόμη και διαφορετικά νήματα (threads) μιας πολυνηματικής εφαρμογής, μπορεί να επιδείξουν ανόμοια συμπεριφορά ως προς τη ζήτηση μνήμης ή την ευαισθησία στην επίδοσής τους. Οι τεχνικές διαμοιρασμού της cache εστιάζουν στις ανάγκες των εφαρμογών και προσπαθούν να κατανείμουν αποτελεσματικά την κοινόχρηστη μνήμη ώστε να μεγιστοποιήσουν την απόδοση του συστήματος. Επιπλέον της απόδοσης, οι τεχνικές αυτές μπορεί να στοχεύουν στη βελτιστοποίηση της παροχής δικαιοσύνης και ποιότητας υπηρεσιών (quality of service, QoS), χαρακτηριστικά που είναι ιδιαίτερα σημαντικά στους servers. Με την αποφυγή των αλληλοεπεμβάσεων στα δεδομένα της cache μεταξύ των πυρήνων, οι τεχνικές διαμοιρασμού πετυχαίνουν καλύτερη εκμετάλλευση του μέρους της cache που κατανέμεται σε κάθε εφαρμογή. Επιπροσθέτως, μπορούν να μειώσουν τη διαμάχη για το εύρος ζώνης (bandwidth) προς την κύρια μνήμη κι έτσι να επωφεληθούν ακόμη και εφαρμογές που τους αποδόθηκε μειωμένο μέρος της cache.

Όπως είναι κατανοητό, οι τεχνικές διαμοιρασμού της cache αποτελούν ένα πολύπλευρο και ισχυρό εργαλείο για ένα ευρύ φάσμα σεναρίων χρήσης. Ωστόσο υπάρχουν πολλές προκλήσεις που καλούνται να αντιμετωπίσουν. Με τη συνεχή αύξηση των πυρήνων των επεξεργαστών, ο αριθμός των πιθανών τρόπων διαχωρισμού της cache αυξάνεται εκθετικά. Έχει αποδειχθεί πως η εύρεση του βέλτιστου διαχωρισμού της cache (π.χ. με την έννοια του μικρότερου συνολικά ρυθμού αστοχιών) είναι NP-δύσκολο πρόβλημα, ενώ μπορεί η βέλτιστη

λύση να μην είναι δίκαιη [19]. Στις επόμενες ενότητες κατηγοριοποιούνται και παρουσιάζονται ορισμένες τεχνικές διαμοιρασμού της κρυφής μνήμης, οι οποίες έχουν προταθεί τα τελευταία χρόνια.

2.1 Κατηγορίες τεχνικών διαμοιρασμού της κοινόχρηστης cache

Οι τεχνικές διαμοιρασμού της cache μπορούν να χωριστούν σε πολλές κατηγορίες ανάλογα με τη μετρική που προσπαθούν να βελτιστοποιήσουν, την απαίτηση χρήσης επιπλέον hardware ή την εύρεση των προφίλ εφαρμογών, τις αλλαγές στην πολιτική αντικατάστασης της cache, τα τμήματα της cache που κατανέμουν κ.ά.. Πιο συγκεκριμένα, οι τεχνικές διαμοιρασμού της cache μπορούν να διαχωριστούν σύμφωνα με τα τμήματα της cache που κατανέμουν σε way, set και block επιπέδου διαμοιρασμού. Οι τεχνικές που κατανέμουν ways παρέχουν σχετικά απλές υλοποιήσεις, ανακατανομή της μνήμης χωρίς να απαιτείται άδειασμα της cache και την ευκολία εύρεσης προφίλ εφαρμογών σε επίπεδο way. Για τους λόγους αυτούς οι ερευνητές [4, 5, 8, 10, 11, 12, 13] έχουν στραφεί περισσότερο σε αυτόν τον τύπο τεχνικών απ' ότι στους δύο επόμενους. Ωστόσο οι τεχνικές αυτές έχουν νόημα μόνο εφόσον ο αριθμός συσχετιστικότητας (associativity) της cache είναι μεγαλύτερος ή ίσος με το διπλάσιο του αριθμού των πυρήνων, δεδομένου πως τουλάχιστον ένα way πρέπει να κατανεμηθεί σε κάθε πυρήνα. Από την άλλη πλευρά, οι τεχνικές διαμοιρασμού της cache που κατανέμουν sets (ή page colors) [16] παρέχουν υψηλότερη ακρίβεια κατανομής σε σχέση με αυτή των ways και μπορούν να ελεγχθούν μέσω του λογισμικού. Το page color προκύπτει από τα επικαλυπτόμενα bits της φυσικής διεύθυνσης μεταξύ του set-index και του αριθμού της σελίδας. Οι τεχνικές που κατανέμουν sets διαμορφώνουν κατάλληλα αυτά τα bits προκειμένου να αλλάξουν τον αριθμό χρωμάτων, και επομένως τα sets, που κατανέμονται σε έναν πυρήνα. Ωστόσο απαιτούν σημαντικές αλλαγές στο λειτουργικό σύστημα προκειμένου να υλοποιηθούν και όταν απαιτείται ανακατανομή της μνήμης αλλάζουν οι δείκτες των sets πολλών blocks με αποτέλεσμα αυτά τα blocks να πρέπει να αδειάσουν ή να αλλάξει ο δείκτης των sets τους. Τέλος οι τεχνικές που κατανέμουν blocks [9] προσφέρουν τη μέγιστη δυνατή ακρίβεια διαχωρισμού της cache αλλά η απόκτηση προφίλ εφαρμογών σε επίπεδο block είναι δύσκολη και μέχρι στιγμής γίνεται προσεγγιστικά.

Επιπλέον οι τεχνικές διαμοιρασμού της cache μπορούν να χωριστούν σε εκείνες που προσπαθούν να κατανείμουν τη μνήμη τροποποιώντας την ίδια την πολιτική αντικατάστασης της cache και σε εκείνες που ενώ διατηρούν την υπάρχουσα πολιτική αντικατάστασης, περιορίζουν πρακτικά το μέρος της cache που κάθε εφαρμογή έχει πρόσβαση. Οι τεχνικές της πρώτης κατηγορίας που έχουν προταθεί απαιτούν την αλλαγή της πολιτικής αντικατάστασης και για τον λόγο αυτό έχουν υλοποιηθεί και ελεγχθεί μέσω προσομοιώσεων. Όμως οι προσομοιώσεις δεν είναι απόλυτα ακριβείς καθώς δεν λαμβάνουν υπ' όψιν τους την επίδραση της προανάκλησης δεδομένων (prefetching) και του ελεγκτή της κύριας μνήμης ή μοντελοποιούν απλοποιημένες εκδοχές τους αφού δεν παρέχονται ακριβείς πληροφορίες από τους κατασκευαστές τους. Από την άλλη πλευρά οι τεχνικές της δεύτερης κατηγορίας προτείνουν έναν μηχανισμό για την εύρεση του βέλτιστου διαχωρισμού της cache και η επιβολή του γίνεται είτε μέσω του λογισμικού (page coloring) είτε μέσω προσομοιώσεων μέχρι το 2015 λόγω έλλειψης υποστήριξης στο υλικό για τον πρακτικό διαχωρισμό της cache. Ωστόσο μετά το 2015 ξεκίνησε μία νέα εποχή για τις τεχνικές διαμοιρασμού της cache με την εισαγωγή νέων κατάλληλων hardware μηχανισμών από την Intel και την ARM, ούτως ώστε η αξιολόγηση και η επιβολή του εκάστοτε διαχωρισμού της cache να γίνεται σε πραγματικά μηχανήματα. Στις επόμενες ενότητες παρουσιάζονται τεχνικές διαμοιρασμού της κοινόχρηστης cache πριν και μετά την εισαγωγή αυτών των μηχανισμών στους νέους επεξεργαστές των Intel και ARM.

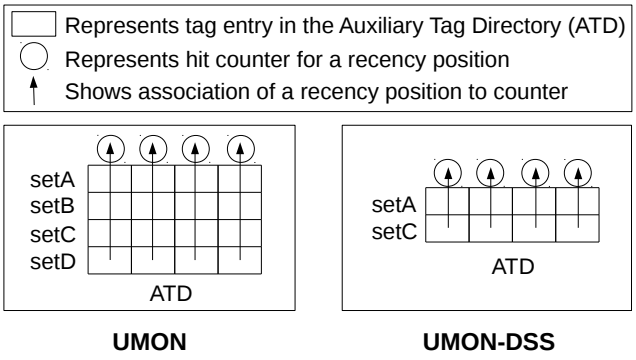
2.2 Τεχνικές διαμοιρασμού της LLC χωρίς τη χρήση αντίστοιχων hardware μηχανισμών των Intel/ARM

Στην ενότητα αυτή παρουσιάζονται τεχνικές διαμοιρασμού των κοινόχρηστων caches, οι οποίες έχουν υλοποιηθεί και ελεγχθεί μέσω προσομοιώσεων. Ορισμένες τεχνικές προτείνουν ένα hardware μηχανισμό για την εύρεση του βέλτιστου διαχωρισμού της cache και λόγω της έλλειψης ενός μηχανισμού στο υλικό που θα επιβάλλει τον εκάστοτε διαχωρισμό της κρυφής μνήμης, ελέγχθηκαν μέσω προσομοιώσεων μεταβάλλοντας αναγκαστικά την πολιτική αντικατάστασης. Από την άλλη πλευρά, άλλες τεχνικές διαμοιρασμού της κοινόχρηστης cache στοχεύουν στην εξολοκλήρου αλλαγή της πολιτικής αντικατάστασης του συστήματος.

2.2.1 Utility-based Cache Partitioning

Στη ενότητα αυτή παρουσιάζεται μία τεχνική ορόσημο για τον διαμοιρασμό της κοινόχρηστης cache καθώς πολλές μετέπειτα τεχνικές βασίστηκαν σε αυτή και χρησιμοποιείται ως μέτρο σύγκρισης για τη βελτίωση της επίδοσης ενός συστήματος. Η τεχνική UCP (utility-based cache partitioning), που προτάθηκε από τους Qureshi και Patt [4] το 2006, είναι ένας χαμηλού κόστους hardware μηχανισμός, ο οποίος διαμοιράζει μία κοινόχρηστη cache σε επίπεδο way μεταξύ πολλών εφαρμογών με βάση τη μείωση στις αστοχίες (misses) της cache που μία εφαρμογή πρόκειται να παρουσιάσει για ένα συγκεκριμένο ποσοστό της cache. Προκειμένου να το πετύχει αυτό, η UCP στηρίζεται στις καμπύλες των misses ανά εφαρμογή, οι οποίες υποδεικνύουν τον αριθμό των misses που θα προκαλέσει μία εφαρμογή για κάθε πιθανό ποσοστό της cache που μπορεί να της αποδοθεί. Για την εύρεση αυτών των καμπυλών, εκμεταλλεύεται την ιδιότητα στοίβας της LRU (θεωρώντας πως είναι η υπάρχουσα πολιτική αντικατάστασης), που σημαίνει πως όταν μία πρόσβαση στην n-way cache ευστοχεί, τότε είναι δεδομένο πως θα ευστοχήσει αν η cache είχε περισσότερα των n ways.

Σημαντικό μέρος της UCP αποτελούν τα κυκλώματα παρακολούθησης της χρησιμότητας της cache (utility monitoring circuits, UMON) ανά πυρήνα, τα οποία παρέχουν την προαναφερθείσα πληροφορία σχετικά με τη χρησιμότητα της cache για όλες τις συνεκτελούμενες εφαρμογές κατά το χρόνο εκτέλεσής τους. Τα κυκλώματα αυτά διαχωρίζονται από την κοινόχρηστη cache και για τον λόγο αυτό είναι δυνατή η παροχή της πληροφορίας χρησιμότητας κάθε εφαρμογής που εκτελείται σε έναν πυρήνα για όλα τα ways της cache, ανεξάρτητα από τις εφαρμογές που εκτελούνται στους άλλους πυρήνες. Τα κυκλώματα UMON (Εικόνα 2.1 [4]) αποτελούνται από έναν βοηθητικό κατάλογο ετικετών (Auxiliary Tag Directory, ATD), ο οποίος υποδεικνύει ποια θα ήταν τα περιεχόμενα της cache αν μόνο ένας πυρήνας είχε πρόσβαση σε ολόκληρη την κοινόχρηστη cache. Επιπλέον κάθε πυρήνας έχει ένα σύνολο από n μετρητές ευστοχιών (hits), όπου n ο αριθμός των ways της cache. Κάθε φορά που ένας πυρήνας ευστοχεί στο i-οστό way του ATD, τότε ο i-οστός μετρητής αυξάνεται. Επομένως ο i-οστός μετρητής καταγράφει τον αριθμό των hits που θα είχαν πραγματοποιηθεί εάν ο πυρήνας ήταν ο μόνος που είχε πρόσβαση σε ολόκληρη την cache και η γραμμή της cache που θα παρείχε το hit ήταν στην i-οστή περισσότερο πρόσφατα χρησιμοποιούμενη θέση. Με άλλα λόγια, οι μετρητές αυτοί προσδιορίζουν *τα επιπλέον hits που θα μπορούσαν να πραγματοποιηθούν για κάθε επιπλέον way που αποδίδεται στον πυρήνα* ή αντιστρόφως *τα επιπλέον misses που θα μπορούσαν να πραγματοποιηθούν με την ελάττωση της cache ανά way*. Για την υλοποίηση του UCP, ο αριθμός των κυκλωμάτων UMON, επομένως και των ATD, είναι ίσος με τον αριθμό των πυρήνων, ενώ κάθε ATD απαιτεί μία επιπλέον είσοδο ετικέτας για κάθε γραμμή της cache. Για παράδειγμα, θεωρώντας πως κάθε είσοδος ετικέτας είναι 4 bytes, ένα σύστημα τεσσάρων πυρήνων χρειάζεται επιπλέον 16 bytes για κάθε μία γραμμή της κοινό-



Εικόνα 2.1: Δομή κυκλωμάτων UMON

χρηστης cache. Επομένως η συνολική μνήμη που απαιτείται για την υλοποίησή των UMON δεν είναι αμελητέα. Για τον λόγο αυτό προτάθηκε η χρήση του μηχανισμού Dynamic Set Sampling (DSS) [20], με τον οποίο δειγματοληπτικά, μόνο ένα υποσύνολο της cache καταγράφεται στον ATD, όπως φαίνεται στην Εικόνα 2.1.

Έχοντας την πληροφορία για τη μείωση των misses ανά πυρήνα από τα κυκλώματα UMON, μπορούν να παρθούν οι αποφάσεις για τον διαχωρισμό της κοινόχρηστης cache. Εάν $miss_a$ και $miss_b$ είναι ο αριθμός των misses που προκαλεί μία εφαρμογή όταν λάβει a και b ways αντίστοιχα (a < b), τότε ορίζεται η χρησιμότητα U_a^b της αύξησης των ways από a σε b ως εξής:

$$U_a^b = miss_a - miss_b$$

Επομένως όσο μεγαλύτερη είναι η χρησιμότητα U_a^b τόσο μεγαλύτερη μείωση στα misses επιτυγχάνεται. Ο αλγόριθμος εξαντλητικής αναζήτησης του βέλτιστου διαχωρισμού της cache προσπαθεί να μεγιστοποιήσει τη συνδυασμένη χρησιμότητα (U_{tot}). Έστω A και B δύο εφαρμογές με συναρτήσεις χρησιμότητας UA και UB αντίστοιχα, τότε για μία cache με 16 ways η συνδυασμένη χρησιμότητα U_{tot} ορίζεται ως εξής:

$$U_{tot} = UA_1^i + UB_1^{(16-i)},\ i = 1 \text{ μέχρι } (16-1)$$

Ο βέλτιστος διαχωρισμός της cache είναι αυτός με τη μεγαλύτερη τιμή U_{tot}. Επομένως με τον αλγόριθμο αυτό υπολογίζεται η συνδυασμένη χρησιμότητα για όλους τους πιθανούς συνδυασμούς διαχωρισμού της cache. Όταν πρόκειται για δύο εφαρμογές, ο εξαντλητικός αλγόριθμος δεν παρουσιάζει κάποιο μειονέκτημα, αφού όλοι οι πιθανοί συνδυασμοί για μία n-way cache είναι n-1 (θεωρώντας πως πρέπει να αποδοθεί σε κάθε εφαρμογή τουλάχιστον ένα way). Ωστόσο με την αύξηση των συνεκτελούμενων εφαρμογών, ο αριθμός των πιθανών τρόπων διαχωρισμού της cache αυξάνεται εκθετικά. Για τον λόγο αυτό προτάθηκε ο αλγόριθμος ”lookahead”, ο ψευδοκώδικας του οποίου παρουσιάζεται στο Παράρτημα A. Ορίζεται η οριακή χρησιμότητα (marginal utility, MU) ως η χρησιμότητα ανά way. Εάν $miss_a$ και $miss_b$ είναι ο αριθμός των misses που προκαλεί μία εφαρμογή όταν λάβει a και b ways αντίστοιχα (a < b), τότε η οριακή χρησιμότητα MU_a^b της αύξησης των ways από a σε b ορίζεται ως εξής:

$$MU_a^b = (miss_a - miss_b)/(b-a) = U_a^b/(b-a)$$

Ο αλγόριθμος lookahead υπολογίζει τη μέγιστη οριακή χρησιμότητα (maximum marginal utility, MMU) και τα ελάχιστα ways τα οποία απαιτούνται για την επίτευξή της για κάθε μία εφαρμογή. Στην εφαρμογή με το μέγιστο MMU αποδίδονται τα ways που απαιτούνται για

την επίτευξή του και ο αλγόριθμος σταματά όταν κατανεμηθούν όλα τα ways. Οι Qureshi και Patt δείχνουν πως η υλοποίηση του μηχανισμού UCP βελτιώνει τη συνολική επίδοση του συστήματος, αν και η επιβολή του διαχωρισμού της cache σε επίπεδο way, λόγω έλλειψης υποστήριξης στο υλικό, έγινε με την τροποποίηση της πολιτική αντικατάστασης LRU μέσω προσομοίωσης.

2.2.2 Adaptive Bloom Filter Cache Partitioning

Στην ίδια λογική με τους Qureshi και Patt [4], οι Nikas κ.ά. [5] παρουσίασαν το 2008 το ABFCP (Adaptive Bloom Filter Cache Partitioning), έναν ακόμη χαμηλότερου κόστους hardware μηχανισμό διαχωρισμού της κοινόχρηστης cache με στόχο τη βελτίωση της συνολικής απόδοσης του συστήματος. Για την επίτευξη αυτού του σκοπού χρησιμοποιούν έναν συνδυασμό από μετρητές και ειδικούς πίνακες, που ονομάζονται Bloom Filters, προκειμένου να προσδιορίσουν πως οι συνεκτελούμενες εφαρμογές θα επωφεληθούν από επιπλέον πόρους της LLC.

Αρχικά ανιχνεύουν το ποσοστό της cache που χρησιμοποιεί κάθε πυρήνας, προσθέτοντας ένα πεδίο για το αναγνωριστικό ID του πυρήνα στο tag κάθε γραμμής της cache. Με αυτόν τον τρόπο, είναι δυνατή η καταγραφή των misses και των hits κάθε πυρήνα. Ωστόσο είναι απαραίτητη η καταγραφή μόνο των far-misses (δυνατή μέσω των Bloom Filters), δηλαδή των misses που θα μετατρέπονταν σε hits εάν είχαν δοθεί περισσότερα ways στον πυρήνα. Για κάθε πυρήνα, προσθέτουν έναν πίνακα Bloom Filter 2^k bits (Bloom Filter Array, BFA) σε κάθε set της cache. Όταν ένα tag απομακρύνεται από την cache, τα k λιγότερο σημαντικά bits (least significant bits, LSBs) χρησιμοποιούνται για την εύρεση του bit του BFA, το οποίο τίθεται σε ”1”. Σε ένα cache miss, εξετάζεται το bit του BFA, το οποίο υπολογίζεται και πάλι από τα k LSBs του tag, και αν αυτό είναι 1 τότε θεωρείται ως far-miss. Ωστόσο υπάρχει η πιθανότητα λάθους, εφόσον σε περισσότερα του ενός tags μπορεί να αποδοθεί το ίδιο bit στον BFA αφού χρησιμοποιούνται μόνο τα k-LSBs για τη εύρεσή του. Επομένως με την υλοποίηση των Bloom Filters ανιχνεύονται τα misses *που πιθανότατα θα ήταν hits αν είχε αποδοθεί στον πυρήνα κάποιος επιπλέον αριθμός ways*. Ακόμη ανιχνεύουν τον αριθμό των hits στη θέση LRU (least recently used) για κάθε set της cache και για κάθε πυρήνα, ο οποίος δείχνει τον αριθμό των hits που θα μετατρέπονταν σε misses με τη μείωση κατά ένα way του τμήματος της cache που έχει αποδοθεί στον πυρήνα.

Για τον διαμοιρασμό της κοινόχρηστης cache χρησιμοποιούν ένα γραμμικό αλγόριθμο. Ο αλγόριθμος αυτός διαβάζει τους μετρητές των hits και misses κάθε πυρήνα και σε κάθε επανάληψη συγκρίνει τη μέγιστη τιμή κέρδους (προκύπτει από τους μετρητές των far-misses) με την ελάχιστη τιμή απώλειας (προκύπτει από τους μετρητές των hits στη θέση LRU) των πυρήνων. Εάν η μέγιστη τιμή κέρδους είναι μεγαλύτερη, τότε το τμήμα της cache του πυρήνα στον οποίο αντιστοιχεί αυτή η μέγιστη τιμή κέρδους αυξάνεται κατά ένα way ενώ το τμήμα της cache του πυρήνα που αντιστοιχεί η ελάχιστη τιμή απώλειας μειώνεται κατά ένα way. Η διαδικασία αυτή συνεχίζεται έως ότου η μέγιστη τιμή κέρδους να είναι μικρότερη από την ελάχιστη τιμή απώλειας ή όλοι οι πυρήνες να έχουν εξεταστεί. Οι Nikas κ.ά. δεν είχαν στη διάθεσή τους κατάλληλους hardware μηχανισμούς για την πρακτική επιβολή του διαχωρισμού της cache σε επίπεδο way και για τον σκοπό αυτό τροποποίησαν την πολιτική αντικατάστασης LRU μέσω προσομοίωσης.

2.2.3 Πολιτικές ψευδο-κατανομής της cache, TADIP και PIPP

Παράλληλα με τους Nikas κ.ά., το 2008 οι Jaleel κ.ά. [7], στοχεύοντας και αυτοί στη βελτίωση της συνολικής απόδοσης του συστήματος, πρότειναν μία τεχνική διαχείρισης των

κοινόχρηστων caches, την TADIP (Thread-Aware Dynamic Insertion Policy), η οποία τροποποιεί την ίδια την πολιτική αντικατάστασης του συστήματος. Πιο συγκεκριμένα, εξελίσσει την τεχνική DIP (Dynamic Insertion Policy) [6] λαμβάνοντας υπ' όψιν τις απαιτήσεις μνήμης κάθε συνεκτελούμενης εφαρμογής.

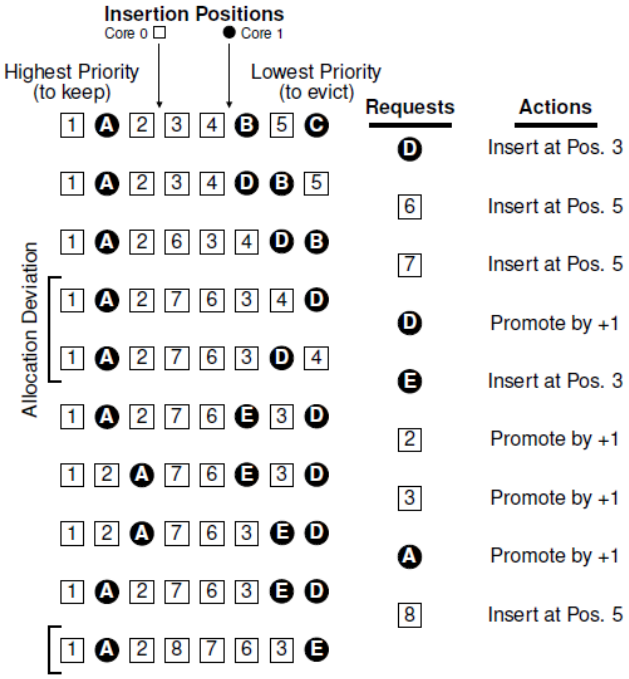
Η τεχνική DIP λειτουργεί βρίσκοντας την καλύτερη πολιτική αντικατάστασης μεταξύ των LRU και BIP (Bimodal Insertion Policy). Η πολιτική BIP εισάγει την πλειονότητα των cache lines στη θέση LRU ενώ τις υπόλοιπες στη θέση MRU (most recently used), εμποδίζοντας έτσι εφαρμογές που δεν επαναχρησιμοποιούν τα δεδομένα τους να καταναλώνουν μεγάλο μέρος της cache. Η τεχνική DIP επιλέγει δυναμικά μεταξύ των πολιτικών LRU και BIP μέσω των μηχανισμών Set Dueling Monitors (SDMs). Κάθε SDM εφαρμόζει μία δοσμένη πολιτική αντικατάστασης (LRU ή BIP) σε έναν μικρό αριθμό sets της cache και υπολογίζει τον αριθμό των misses που προκύπτουν από αυτά τα sets. Η πολιτική του SDM με τον μικρότερο αριθμό σε misses επικρατεί και εφαρμόζεται στα υπολειπόμενα sets της cache. Ωστόσο η τεχνική DIP δεν λαμβάνει υπ' όψιν της τις ανάγκες των διαφορετικών εφαρμογών και εφαρμόζει την νικητήρια πολιτική στα υπολειπόμενα sets της cache ανεξαρτήτως του πυρήνα στον οποίο ανήκουν. Για τον λόγο αυτό η τεχνική TADIP προσπαθεί να αποφασίσει μεταξύ των πολιτικών LRU και BIP για κάθε εφαρμογή που εκτελείται σε ένα ξεχωριστό πυρήνα. Ωστόσο ο αριθμός των πιθανών συνδυασμών πολιτικών και επομένως ο αριθμός των απαιτούμενων SDMs για N συνεκτελούμενες εφαρμογές είναι 2^N, δηλαδή αυξάνεται εκθετικά με την αύξηση του αριθμού των εφαρμογών/πυρήνων. Για τον λόγο αυτό η τεχνική TADIP κάνει χρήση ευριστικών μεθόδων προκειμένου να μειώσει τον αριθμό των SDMs με την αύξηση των πυρήνων.

Στην ίδια λογική με την τεχνική TADIP των Jaleel κ.ά., προτάθηκε το 2009 η τεχνική PIPP (Promotion/Insertion Pseudo-Partitioning) από τους Xie και Loh [8] για την κατανομή των κοινόχρηστων caches, με σκοπό τη βελτίωση της συνολικής επίδοσης του συστήματος. Η τεχνική αυτή τροποποιεί την πολιτική αντικατάστασης του συστήματος και συγκεκριμένα μεταβάλει την πολιτική εισαγωγής και προαγωγής των cache lines. Η πολιτική εισαγωγής των δεδομένων στην cache καθορίζει σε ποια θέση της σειράς προτεραιότητας (όπως η στοίβα LRU), η οποία χρησιμοποιείται για την απομάκρυνση των cache lines, θα εκχωρηθεί αρχικά μία cache line. Από την άλλη πλευρά, η πολιτική προαγωγής καθορίζει ποιες αλλαγές θα γίνουν στη σειρά προτεραιότητας όταν πραγματοποιείται ένα cache hit.

Η τεχνική PIPP χρησιμοποιεί τα κυκλώματα παρακολούθησης της χρησιμότητας της cache UMON της τεχνικής UCP, που παρουσιάστηκαν στην Ενότητα 2.2.1, για την εύρεση του διαχωρισμού της cache σε n τμήματα $\{\pi_1, \pi_2, ..., \pi_n\}$ από ways που θα αποδοθούν στους n πυρήνες. Το τμήμα της cache που αποδίδεται σε κάθε πυρήνα καθορίζει τη θέση εισαγωγής νέων cache lines του πυρήνα, αφού μία νέα cache line του πυρήνα i θα εισαχθεί στη θέση προτεραιότητας π_i. Σε ένα cache hit, η cache line έχει πιθανότητα P_{prom} να αναβαθμιστεί κατά μία θέση ενώ $1 - P_{prom}$ να παραμείνει στην ίδια θέση. Τέλος η πολιτική απομάκρυνσης cache lines παραμένει ίδια με την ευρέως χρησιμοποιούμενη LRU, δηλαδή απομακρύνεται η cache line με τη μικρότερη σειρά προτεραιότητας.

Επιπλέον η τεχνική PIPP ανιχνεύει εφαρμογές που ζητούν μεγάλο όγκο δεδομένων χωρίς να τον επαναχρησιμοποιούν (γνωστές ως streaming εφαρμογές), ελέγχοντας αν ο αριθμός των misses και το ποσοστό των misses σε σχέση με τις συνολικές προσβάσεις στην cache ξεπερνούν συγκεκριμένα κατώφλια. Για όλες αυτές τις εφαρμογές η εισαγωγή των cache lines γίνεται στη θέση π_{stream}, όπου το π_{stream} ισούται με τον αριθμό των streaming εφαρμογών, οπότε αποδίδεται ουσιαστικά μόνο ένα way σε κάθε streaming εφαρμογή. Στην Εικόνα 2.2, όπως αυτή δημοσιεύθηκε στο άρθρο [8], παρουσιάζεται ένα παράδειγμα για μία cache με 8 ways κοινόχρηστη μεταξύ δύο πυρήνων και με επιθυμητό διαχωρισμό της cache $\pi_1 = 5$

και $\pi_2 = 3$. Παρουσιάζεται μια σειρά ενεργειών, με την οποία φαίνεται ο τρόπος εισαγωγής, προαγωγής και απομάκρυνσης δεδομένων από την cache, θεωρώντας πως η πιθανότητα P_{prom} για την αναβάθμιση μιας cache line κατά μία θέση ισούται με 1. Αξίζει να σημειωθεί πως ενώ ο διαχωρισμός της cache στους δύο πυρήνες είναι $\pi_1 = 5$ και $\pi_2 = 3$ ways, υπάρχουν στιγμιότυπα της cache όπου δεν τηρείται απόλυτα αυτός ο διαχωρισμός, γι' αυτό άλλωστε και πρόκειται για μια τεχνική ψευδο-κατανομής της κοινόχρηστης cache.



Εικόνα 2.2: Παράδειγμα λειτουργίας της πολιτικής PIPP

2.2.4 Vantage Cache Partitioning

Το 2011 οι Sanchez και Kozyrakis [9] παρατήρησαν πως οι μέχρι τότε προτεινόμενες τεχνικές διαμοιρασμού των κοινόχρηστων caches περιορίζονταν σε λίγα τμήματα διαχωρισμού (partitions) της cache (συνήθως με μέγιστο αριθμό το σύνολο των ways της cache), μειώνοντας έτσι τη συσχετιστικότητα της cache και πλήττοντας την απόδοση του συστήματος, ιδιαίτερα με την αύξηση των πυρήνων. Για την αντιμετώπιση αυτών των περιορισμών, πρότειναν την τεχνική διαμοιρασμού της κοινόχρηστης cache, Vantage, η οποία διατηρεί την υψηλή συσχετιστικότητα της cache, χωρίζοντάς την σε δεκάδες τμήματα σε επίπεδο block/cache line με την τροποποίηση της πολιτικής αντικατάστασης.

Η τεχνική Vantage δεν πραγματοποιεί πρακτικό διαχωρισμό των blocks της cache, αντίθετα επιτρέπει σε κάθε block να εισαχθεί οπουδήποτε στην cache ανεξάρτητα από το partition στο οποίο ανήκει, και η επιβολή των partitions γίνεται μέσω των αποφάσεων για την επιλογή των blocks που θα απομακρυνθούν από την cache. Αν το block που θα αντικατασταθεί ανήκει σε διαφορετικό partition από αυτό του εισερχομένου, τότε θα εξακολουθεί να υπάρχει διαμάχη μεταξύ των partitions των εφαρμογών. Επιπλέον, αν η επιλογή του block αντικατάστασης γίνει από το ίδιο partition, τότε δε θα μπορεί να κλιμακωθεί με τον αριθμό των partitions, αφού όσο μεγαλύτερος ο αριθμός των partitions τόσο μικρότερος ο αριθμός των blocks που ανήκουν στο ίδιο partition και είναι υποψήφια προς απομάκρυνση. Επομένως οι Sanchez και

Kozyrakis για να ελαττώσουν αυτόν τον περιορισμό, απαιτούν μόνο ο ρυθμός εισαγωγής και απομάκρυνσης των blocks από κάθε partition να είναι κατά μέσο όρο ίδιος. Ωστόσο και αυτή η προσέγγιση δεν εξαλείφει τη διαμάχη μεταξύ των διαφορετικών partitions, γι' αυτό και η τεχνική τους δε διαμοιράζει ολόκληρη την cache αλλά το μεγαλύτερο μέρος της (περίπου το 85%).

Πιο συγκεκριμένα, χωρίζουν την cache σε δύο μέρη, στο διαχειριζόμενο και το μη διαχειριζόμενο τμήμα, και κατανέμουν στις εφαρμογές μόνο το διαχειριζόμενο. Με αυτόν τον τρόπο είναι δυνατόν τα partitions να υπερβούν το ποσοστό της cache που τους αναλογεί χρησιμοποιώντας μέρος του μη διαχειριζόμενου τμήματος, διατηρώντας έτσι υψηλή συσχετιστικότητα για κάθε partition, ανεξαρτήτως του συνολικού αριθμού τους. Ορίζοντας ένα κατάλληλο μέγεθος για το μη διαχειριζόμενο τμήμα, οι εκδιώξεις block από την cache γίνονται σχεδόν αποκλειστικά από το τμήμα αυτό. Για την εύρεση των partitions του διαχειριζόμενου τμήματος, χρησιμοποιείται η τεχνική UCP με τον αλγόριθμο lookahead (Ενότητα 2.2.1). Αρχικά τα blocks εισέρχονται στο διαχειριζόμενο τμήμα, υποβιβάζονται στο μη διαχειριζόμενο τμήμα (με τη αλλαγή της ετικέτας τους) και από εκεί απομακρύνονται ή προβιβάζονται στο διαχειριζόμενο τμήμα σε ένα cache hit. Η υποβίβαση των blocks στο μη διαχειριζόμενο τμήμα γίνεται ταυτόχρονα με την απομάκρυνση ενός block από το τμήμα αυτό και υποβιβάζονται τα blocks με προτεραιότητα υποβίβασης μεγαλύτερη από ένα κατώφλι, ειδικό για κάθε partition. Η τιμή για κάθε κατώφλι εξαρτάται από το ρυθμό εισαγωγής blocks σε κάθε partition αλλά και από το πόσο έχουν υπερβεί το ποσοστό της cache που τους αναλογεί με τη χρήση του μη διαχειριζόμενου τμήματος. Η τεχνική Vantage βελτιώνει κατά μέσο όρο τη συνολική επίδοση του συστήματος, αλλά υπάρχουν περιπτώσεις όπου ο διαμοιρασμός ενός μέρους της cache και όχι ολόκληρης οδηγεί στη μείωση της επίδοσης ορισμένων συνδυασμών εφαρμογών.

2.2.5 Cooperative Partitioning

Το 2012 οι Sundararajan κ.ά. [10] πρότειναν μία ενεργειακά αποδοτική τεχνική διαμοιρασμού της κοινόχρηστης cache, την Cooperative Partitioning (CoP). Οι περισσότερες τεχνικές που είχαν προταθεί μέχρι τότε στόχευαν στη βελτίωση της επίδοσης του συστήματος χωρίς να λαμβάνουν υπ' όψιν τους την κατανάλωση ενέργειας, η οποία είναι υπεύθυνη για ένα σημαντικό μέρος του κόστους του μηχανήματος. Γι' αυτόν τον λόγο παρουσίασαν μία δυναμική τεχνική διαχωρισμού της cache που μειώνει την κατανάλωση ενέργειας ενώ παράλληλα βελτιώνει τη συνολική απόδοση του συστήματος.

Οι μέχρι τότε τεχνικές διαμοιρασμού της cache σε επίπεδο way, περιόριζαν τη μνήμη που αποδίδεται σε κάθε εφαρμογή/πυρήνα επιτρέποντας την εισαγωγή νέων cache lines μόνο σε ένα συγκεκριμένο αριθμό ways, αν και όλοι οι πυρήνες είχαν διαρκώς πρόσβαση σε ολόκληρη την cache για ανάγνωση ή εγγραφή δεδομένων. Από την άλλη πλευρά, η τεχνική CoP εφαρμόζει απόλυτη αντιστοίχηση πυρήνων με ways, δηλαδή κάθε στιγμή όλα τα sets κάθε way ανήκουν εξ' ολοκλήρου σε έναν μόνο πυρήνα. Ωστόσο υπάρχει μία μικρή περίοδος μετά τον επανακαθορισμό του διαχωρισμού της cache, κατά την οποία οι πυρήνες "συνεργάζονται" ώστε να μην αδειάσουν κατευθείαν τα ways που θα μεταβιβαστούν. Για την εύρεση του βέλτιστου διαχωρισμού της cache χρησιμοποιούνται τα κυκλώματα UMON και ο αλγόριθμος lookahead της τεχνικής UCP (Ενότητα 2.2.1), με τη διαφορά πως τα ways αποδίδονται στον πυρήνα μόνο αν η μείωση του ρυθμού των misses είναι σημαντικά μεγάλη και ξεπερνάει ένα κατώφλι. Με αυτόν τον τρόπο μετά την εκτέλεση του αλγορίθμου lookahead μπορεί να υπάρχουν ways τα οποία δεν αποδίδονται σε κανέναν πυρήνα. Για την επιβολή των αποφάσεων διαχωρισμού της cache και για την εξασφάλιση της πρόσβασης ενός μόνο πυρήνα

σε κάθε way, οι Sundararajan κ.ά. πρότειναν την εισαγωγή δύο καταχωρητών ανά way, τον read access permission (RAP), ο οποίος καθορίζει ποιος πυρήνας έχει δικαίωμα ανάγνωσης στο συγκεκριμένο way, και τον write access permission (WAP), ο οποίος καθορίζει ποιος πυρήνας έχει δικαίωμα εγγραφής στο συγκεκριμένο way. Έχοντας αυτούς τους μηχανισμούς, η εξοικονόμηση ενέργειας επιτυγχάνεται με δύο τρόπους. Πρώτον, κάθε πυρήνας σε κάθε πρόσβαση στην cache χρειάζεται να ψάξει τα δεδομένα του μόνο στα ways που του ανήκουν και όχι σε όλα όπως θα γινόταν κανονικά, και δεύτερον, όταν ένα way δε χρησιμοποιείται από κάποιον πυρήνα μπορεί να απενεργοποιηθεί για την εξοικονόμηση ενέργειας.

2.2.6 XChange: Dynamic Multi-resource Allocation

Το 2015 οι Wang και Martínez [12] παρουσίασαν ένα νέο μηχανισμό, τον XChange, με τον οποίο κατανέμουν την κοινόχρηστη cache αλλά και την ισχύ του chip στους πυρήνες πετυχαίνοντας υψηλή επίδοση και δικαιοσύνη στο σύστημα. Ο μηχανισμός XChange μοντελοποιεί το πρόβλημα του διαμοιρασμού των κοινόχρηστων πόρων ως μία δυναμική και κατανεμημένη αγορά, όπου σε κάθε κοινόχρηστο πόρο αποδίδεται μία τιμή που αλλάζει με την πάροδο του χρόνου και κάθε πυρήνας υποβάλει προσφορές γι' αυτούς τους πόρους έχοντας συγκεκριμένο προϋπολογισμό. Οι τιμές των πόρων προσαρμόζονται σύμφωνα με την προσφορά και τη ζήτηση, ενώ οι πυρήνες μαθαίνουν δυναμικά τη δική τους σχέση επίδοσης-πόρων και αναλόγως κάνουν προσφορές. Με αυτόν τον τρόπο αν ένας πόρος έχει υψηλή τιμή λόγω μεγάλης ζήτησης, τότε ο πυρήνας θα αρχίσει να κάνει προσφορές για ένα πιο φθηνό πόρο ακόμη κι αν ο πρώτος του εξασφάλιζε υψηλότερη επίδοση. Τέλος, ανάλογα με τον εκάστοτε στόχο του μηχανισμού, οι προϋπολογισμοί των πυρήνων μπορεί να διαφέρουν. Για παράδειγμα, για τη βελτιστοποίηση της συνολικής απόδοσης του συστήματος, μπορεί να δοθεί μεγαλύτερος προϋπολογισμός σε εφαρμογές/πυρήνες με μεγαλύτερη οριακή χρησιμότητα (χρησιμότητα ανά μονάδα πόρου), ενώ για τη βελτιστοποίηση της δικαιοσύνης μπορούν να δοθούν ίσοι προϋπολογισμοί σε όλους τους πυρήνες.

2.2.7 Optimal Cache Partition-Sharing

Παράλληλα με τους Wang και Martínez, το 2015 οι Brock κ.ά. [14] παρουσίασαν μία τεχνική διαμοιρασμού της κοινόχρηστης cache, αλλά και το θεωρητικό της υπόβαθρο, κατά την οποία ορισμένοι πυρήνες μπορεί να μοιράζονται μέρος της cache ενώ άλλοι να έχουν πρόσβαση σε ιδιωτικά τμήματα. Η τεχνική τους Partition-Sharing (PS) στοχεύει στη βελτίωση της συνολικής απόδοσης του συστήματος αλλά μπορεί να χρησιμοποιηθεί και για τη βελτιστοποίηση της δικαιοσύνης ή της ποιότητας υπηρεσιών του συστήματος. Τα ιδιωτικά μέρη της cache παρέχουν σε κάθε εφαρμογή προστασία από άλλες επιθετικές εφαρμογές, ενώ τα τμήματα της cache στα οποία έχουν πρόσβαση πολλοί πυρήνες αποτρέπουν τους πόρους από το να μείνουν αχρησιμοποίητοι. Οι Brock κ.ά. προσπαθούν να ανάγουν το πρόβλημα του διαχωρισμού της cache σε διαμοιραζόμενα και ιδιωτικά τμήματα (PS), στο πιο απλό πρόβλημα του διαχωρισμού της μόνο σε ιδιωτικά τμήματα. Ορίζουν το "φυσικό τμήμα" έτσι ώστε κάθε εφαρμογή να έχει τον ίδιο ρυθμό αστοχιών (miss rate) στο "φυσικό της τμήμα" με αυτόν που θα είχε στην κοινόχρηστη cache χωρίς κάποιον διαχωρισμό. Επομένως η επίδοση της cache αποτελούμενη από τα "φυσικά τμήματα" είναι ίση με αυτήν της απλής κοινόχρηστης cache. Αυτή η φόρμουλα επιτρέπει την αναγωγή του προβλήματος PS μόνο στο πρόβλημα του διαχωρισμού της cache σε ιδιωτικά τμήματα, και έτσι η βέλτιστη λύση της εύρεσης των ιδιωτικών τμημάτων είναι τουλάχιστον τόσο καλή όσο η βέλτιστη λύση του PS. Για την εύρεση του βέλτιστου διαχωρισμού της cache σε ιδιωτικά τμήματα προτείνουν έναν αλγόριθμο που

χρησιμοποιεί δυναμικό προγραμματισμό για την εξέταση ολόκληρου του χώρου πιθανών λύσεων. Ο αλγόριθμος δυναμικού προγραμματισμού βρίσκει τη βέλτιστη λύση εφαρμογή ανά εφαρμογή για μία cache χωρητικότητας C. Συγκεκριμένα κάθε φορά που προστίθεται μία εφαρμογή P_i, της αποδίδεται c_i τμήμα της cache, το οποίο ελαχιστοποιεί το άθροισμα των misses της αλλά και των misses της βέλτιστης λύσης για τις πρώτες i-1 εφαρμογές με χωρητικότητα cache $C - c_i$. Με αυτόν τον τρόπο βρίσκουν τη βέλτιστη λύση για τη βελτίωση της απόδοσης του συστήματος, αλλά ο αλγόριθμός τους μπορεί να χρησιμοποιηθεί και για τη βελτίωση της δικαιοσύνης και της ποιότητας υπηρεσιών του συστήματος.

2.3 Τεχνικές διαμοιρασμού της LLC με τη χρήση αντίστοιχων hardware μηχανισμών των Intel/ARM

Στην προηγούμενη ενότητα παρουσιάστηκαν πολλές έρευνες που πρότειναν τεχνικές διαμοιρασμού της κοινόχρηστης cache οι οποίες μετά την εύρεση του επιθυμητού διαχωρισμού της cache δεν είχαν τη δυνατότητα της πρακτικής επιβολής του. Επομένως κατέφευγαν σε λύσεις μέσω προσομοιώσεων, συνήθως τροποποιώντας την υπάρχουσα πολιτική αντικατάστασης της cache. Σε αυτή την ενότητα παρουσιάζονται έρευνες που πραγματοποιήθηκαν σε πραγματικά μηχανήματα αφού είχαν πλέον στη διάθεσή τους την απαραίτητη υποστήριξη στο υλικό για τον διαμοιρασμό της LLC στους νέους επεξεργαστές των Intel και ARM.

2.3.1 Τεχνική διαμοιρασμού της κοινόχρηστης cache για τη βελτίωση ποιότητας υπηρεσιών

Οι Papadakis κ.ά. [15] παρουσίασαν το 2017 ένα μηχανισμό στο λογισμικό για τον δυναμικό διαχωρισμό της κοινόχρηστης cache εκμεταλλευόμενοι τις δυνατότητες των νέων μηχανισμών υλικού της Intel CMT-CAT, που παρουσιάζονται στο Κεφάλαιο 3, για την επιβολή του. Ο μηχανισμός τους έχει σκοπό την εξασφάλιση της βέλτιστης ποιότητας υπηρεσιών (QoS) σε εφαρμογές υψηλής προτεραιότητας και στην ταυτόχρονη διατήρηση της υψηλής επίδοσης του συστήματος. Για την επίτευξη αυτού του σκοπού χρησιμοποιούν την τεχνολογία CMT για την καταγραφή της επίδοσης των εφαρμογών μέσω της μετρικής IPC (instructions per cycle, εντολές ανά κύκλο) και την τεχνολογία CAT για την εφαρμογή του εκάστοτε διαχωρισμού της μνήμης σε επίπεδο way.

Πιο συγκεκριμένα, θεωρούν μία εφαρμογή ως την εφαρμογή υψηλής προτεραιότητας και πολλές άλλες ως χαμηλής προτεραιότητας εφαρμογές. Στόχος τους είναι η διατήρηση της επίδοσης της εφαρμογής με υψηλή προτεραιότητα στο 95% της μέγιστης απόδοσής της και έπειτα η παραχώρηση όσο το δυνατόν περισσότερης μνήμης στις εφαρμογές χαμηλής προτεραιότητας. Με την εκκίνηση της εκτέλεσης των εφαρμογών, ο αλγόριθμος αποδίδει N-1 ways στην εφαρμογή υψηλής προτεραιότητας και 1 way στις εφαρμογές χαμηλής προτεραιότητας, θεωρώντας LLC με N ways associativity. Έπειτα αναμένει μέχρι να σταθεροποιηθεί η απόδοση της εφαρμογής υψηλής προτεραιότητας και στη συνέχεια μειώνει σταδιακά τη μνήμη της κατά ένα way, το οποίο αποδίδει στις εφαρμογές χαμηλής προτεραιότητας, έως ότου η απόδοσή της να μην είναι πλέον σταθερή. Σε αυτό το σημείο επαναφέρει το τελευταίο way που της στέρησε και αν η απόδοσή της δεν σταθεροποιηθεί η διαδικασία επαναλαμβάνεται αλλιώς βρέθηκε το σημείο ισορροπίας, δηλαδή ο βέλτιστος διαχωρισμός. Μετά την εύρεση αυτού του σημείου ο αλγόριθμος συνεχίζει να ελέγχει τη σταθερότητα της απόδοσης της εφαρμογής υψηλής προτεραιότητας και σε περίπτωση αλλαγής της πέραν ενός κατωφλίου, θεωρείται πως ξεκίνησε διαφορετική φάση εκτέλεσης και επαναλαμβάνει την προαναφερθείσα διαδικασία. Με αυτόν τον τρόπο οι Papadakis κ.ά. καταφέρνουν να αποδίδουν όσο

το δυνατόν λιγότερο μέρος της cache στην εφαρμογή υψηλής προτεραιότητας διατηρώντας όμως την υψηλή επίδοσή της και παράλληλα οι εφαρμογές χαμηλής προτεραιότητας να δέχονται σημαντικά αυξημένο τμήμα της cache.

2.3.2 Set and Way Cache Partitioning

Παράλληλα με τους Papadakis κ.ά., το 2017 προτάθηκε από τους Wang κ.ά. [16] η τεχνική διαχείρισης της κοινόχρηστης cache SWAP (Set and WAy Partitioning), μία ελαχίστου κόστους τεχνική που στοχεύει στη βελτίωση της συνολικής επίδοσης του συστήματος συνδυάζοντας τον διαχωρισμό της cache σε επίπεδο way και σε set. Η τεχνική SWAP συνδυάζοντας τον διαχωρισμό της cache σε επίπεδο way, χρησιμοποιώντας υπάρχον hardware, με τον διαχωρισμό της cache σε επίπεδο set, μέσω του λειτουργικού συστήματος, καταφέρνει να χωρίσει την cache σε εκατοντάδες περιοχές προσφέροντας έτσι επαρκή κλιμακωσιμότητα καθώς αυξάνεται ο αριθμός των πυρήνων.

Για την επιβολή του επιθυμητού διαχωρισμού της κοινόχρηστης cache σε επίπεδο way χρησιμοποίησαν τον διαθέσιμο hardware μηχανισμό που τους παρείχε ο πολυπύρηνος επεξεργαστής ThunderX της ARM. Συγκεκριμένα ο επεξεργαστής αυτός παρέχει έναν ειδικό καταχωρητή ανά πυρήνα, ο οποίος προσδιορίζει τα cache ways στα οποία μπορεί ο πυρήνας να εισάγει νέες cache lines, αν και συνεχίζει να έχει πρόσβαση σε ολόκληρη την cache για ανάγνωση και εγγραφή δεδομένων. Τους καταχωρητές αυτούς διαμορφώνει η τεχνική SWAP ώστε να επιβληθεί η εκάστοτε ανάθεση ways στους πυρήνες. Για τον περαιτέρω διαχωρισμό της cache σε επίπεδο set εφαρμόζουν την τεχνική page coloring τροποποιώντας το λειτουργικό σύστημα. Με την τεχνική αυτή κάθε σελίδα ανήκει σε ένα ”χρώμα” (”color”), το οποίο προκύπτει από τα επικαλυπτόμενα bits της φυσικής διεύθυνσης μεταξύ του set-index και του αριθμού της σελίδας. Το λειτουργικό σύστημα, περιορίζοντας τον αριθμό των χρωμάτων των σελίδων που ανήκουν σε μία εφαρμογή, μπορεί να περιορίσει και την εφαρμογή στη χρήση μόνο ενός υποσυνόλου των sets της cache. Έτσι μία cache με 16 ways και με τη θέσπιση του αριθμού των page colors σε 16 μπορεί να χωριστεί σε 256 περιοχές (regions).

Αρχικά η τεχνική SWAP υπολογίζει τις καμπύλες των misses ανά εφαρμογή, είτε από offline πληροφορία είτε από online μέσω κάποιων από τους μέχρι τότε προτεινόμενους μηχανισμούς λογισμικού [21] ή υλικού [4], και έπειτα εφαρμόζει τον αλγόριθμο lookahead των Qureshi και Patt (Ενότητα 2.2.1), ο οποίος αποφασίζει το βέλτιστο μέγεθος του region κάθε πυρήνα έτσι ώστε το άθροισμα όλων των regions να ισούται με το συνολικό μέγεθος της cache. Για την απόδοση του μεγέθους κάθε region σε συγκεκριμένα ways και colors υλοποιούν έναν αλγόριθμο πολυπλοκότητας O(NlogN), όπου N ο αριθμός των πυρήνων, με τον οποίο προσπαθούν όσο είναι δυνατόν να μην επικαλύπτονται τα regions μεταξύ τους αλλά και να μη μένει αχρησιμοποίητος χώρος στην cache. Επιπλέον προσπαθούν με τον δυναμικό επανακαθορισμό των μεγεθών των regions στις αλλαγές των φάσεων εκτέλεσης, να επηρεάζονται όσο το δυνατόν λιγότερα μέρη της cache. Εξετάζοντας και βελτιστοποιώντας πολλές πτυχές αυτού του αλγορίθμου, οι Wang κ.ά. καταφέρνουν να βελτιώσουν τη συνολική απόδοση του πραγματικού μηχανήματός τους. Τέλος παρουσιάζουν μία επέκταση του μηχανισμού τους ώστε να μπορεί να χρησιμοποιηθεί και για τη διασφάλιση της ποιότητας υπηρεσίας σε κρίσιμες εφαρμογές με την παράλληλη μεγιστοποίηση της επίδοσης των υπολοίπων εφαρμογών. Συγκεκριμένα η μνήμη cache μοιράζεται σε επίπεδο way στις κρίσιμες εφαρμογές, αυξάνοντας σταδιακά κατά ένα way τη μνήμη που τους αποδίδεται, και όταν αυτές διασφαλίσουν τη ζητούμενη απόδοση, μειώνεται η μνήμη που τους αποδίδεται μέχρι το σημείο όπου διασφαλίζεται ακόμη η ίδια απόδοση. Έπειτα η υπόλοιπη cache διαμοιράζεται στις μη-κρίσιμες εφαρμογές χρησιμοποιώντας την τεχνική SWAP, όπως αυτή παρουσιάστηκε.

2.3.3 Τεχνική διαμοιρασμού της κοινόχρηστης cache με ομαδοποίηση εφαρμογών

Οι Selfa κ.ά. [18] παρουσίασαν το 2017 μία τεχνική διαμοιρασμού της κοινόχρηστης cache μέσω ορισμένων πολιτικών ομαδοποίησης εφαρμογών, με στόχο τη βελτίωση της δικαιοσύνης του συστήματος. Είναι η πρώτη τεχνική που προσπαθεί να βελτιώσει τη δικαιοσύνη του συστήματος σε πραγματικό σύστημα, χρησιμοποιώντας τον hardware μηχανισμό CAT της Intel (Κεφάλαιο 3). Η τεχνική τους χωρίζεται σε δύο μέρη, τον χωρισμό των εφαρμογών σε ομάδες ανάλογα με τον αριθμό καθυστερήσεων (stalls) των πυρήνων λόγω προσβάσεων στην LLC και τον διαχωρισμό των ways της LLC στις ομάδες εφαρμογών σύμφωνα με ένα μαθηματικό μοντέλο. Ο διαχωρισμός των ways στις ομάδες εφαρμογών δεν είναι απόλυτος, με την έννοια ότι επιτρέπεται επικάλυψη μεταξύ ορισμένων ways της LLC.

Για την ομαδοποίηση των εφαρμογών χρησιμοποιείται ως μετρική ο αριθμός των stalls που προκύπτουν λόγω των misses στο δεύτερο επίπεδο της cache (L2 cache), θεωρώντας cache με τρία επίπεδα. Τη μετρική αυτή εξάγουν μέσω του μετρητή παρακολούθησης της επίδοσης του επεξεργαστή STALLS_L2_PENDING, ο οποίος συγκρατεί τον αριθμό των κύκλων κατά τους οποίους η εκτέλεση της εφαρμογής καθυστερεί λόγω των misses στην L2 cache. Ο μετρητής STALLS_L2_PENDING επηρεάζεται από τη διαμάχη των συνεκτελούμενων εφαρμογών για τους κοινόχρηστους πόρους, όπως η LLC, η κύρια μνήμη και οι διασυνδέσεις εντός του chip (π.χ. διασυνδέσεις μεταξύ των slices της LLC). Αν και αυτός ο μετρητής δεν ξεχωρίζει τους κύκλους των stalls που προκαλούνται από τα κανονικά misses και αυτά λόγω της διαμάχης των εφαρμογών, καθώς αυξάνεται ο αριθμός των συνεκτελούμενων εφαρμογών τα δεύτερα επικρατούν και έτσι δεν τίθεται πλέον αυτός ο προβληματισμός.

Οι Selfa κ.ά. παρουσιάζουν τρεις πολιτικές για την ομαδοποίηση των εφαρμογών, τις SFn-mK, mK και Dunn. Και οι τρεις πολιτικές εφαρμόζουν τον αλγόριθμο Kmeans [22], ο οποίος χωρίζει τις εφαρμογές σε m ομάδες με βάση τον αριθμό των stalls κάθε εφαρμογής/πυρήνα λόγω των L2 misses (από τον μετρητή STALLS_L2_PENDING). Μετά την ομαδοποίηση των εφαρμογών σε m ομάδες, η πολιτική SFn-mK δίνει την υψηλότερη προτεραιότητα στην ομάδα εφαρμογών με την μεγαλύτερη επιβράδυνση δίνοντάς της πρόσβαση σε όλα τα ways της LLC. Στην ομάδα με την αμέσως επόμενη προτεραιότητα δίνει έναν μειωμένο κατά n αριθμό ways κ.ο.κ., όπου τα n, m μπορεί να πάρουν τιμές από 2 μέχρι 4 για μία cache με 20 ways. Για παράδειγμα, για n=3 και m=4, οι τέσσερις ομάδες εφαρμογών λαμβάνουν με βάση την προτεραιότητά τους στατικά 20, 17, 14 και 11 ways αντίστοιχα. Η πολιτική mK δεν αποδίδει στατικά τα ways σε κάθε ομάδα εφαρμογών, αλλά τα υπολογίζει δυναμικά σύμφωνα με μία απλή εκθετική συνάρτηση. Η είσοδος στην εκθετική συνάρτηση είναι ο κανονικοποιημένος αριθμός των misses κάθε ομάδας εφαρμογών και η έξοδός της είναι ο αριθμός των ways, με τιμές από 2 μέχρι 20, που θα της ανατεθούν. Τέλος, η πολιτική Dunn ομαδοποιεί τις εφαρμογές και αναθέτει τα ways της cache όπως η πολιτική mK, όμως σε αντίθεση με τις προηγούμενες πολιτικές όπου θεωρούσαν έναν στατικό αριθμό ομάδων εφαρμογών, η πολιτική αυτή καθορίζει δυναμικά κατά το χρόνο εκτέλεσης τον βέλτιστο αριθμό των ομάδων (με βάση τον δείκτη Dunn [23]) ώστε να προσαρμόζεται στις διαφορετικές φάσεις εκτέλεσης των εφαρμογών. Με την τελευταία πολιτική πετυχαίνουν τη μεγαλύτερη αύξηση της δικαιοσύνης σε σχέση με τη δικαιοσύνη που έχει το σύστημα χωρίς κάποια πολιτική διαμοιρασμού της μνήμης.

Κεφάλαιο 3

Οι τεχνολογίες CMT-CAT της Intel

Οι τεχνολογίες παρακολούθησης της κρυφής μνήμης (Cache Monitoring Technology, CMT) και κατανομής της (Cache Allocation Technology, CAT) που υλοποιήθηκαν από την Intel [1], παρέχουν έναν μηχανισμό υλικού (hardware) με τον οποίο είναι δυνατή η διαχείριση ενός διαμοιραζόμενου πόρου μεταξύ των επεξεργαστών, όπως το τελευταίο επίπεδο της cache (last level cache, LLC). Με την ανάπτυξη των πολυπύρηνων αρχιτεκτονικών, εδώ και μία δεκαετία, ξεκίνησε η έρευνα για την εξασφάλιση της υψηλής επίδοσης των εξυπηρετητών (servers), οι οποίοι έπρεπε να αντιμετωπίσουν τη διαμάχη για τους κοινόχρηστους πόρους. Αυτή η διαμάχη επιδρά όχι μόνο στην επίδοση, αλλά και στην ποιότητα παροχής υπηρεσιών (Quality of Service, QoS) του συστήματος όταν οι εφαρμογές εκτελούνται ταυτόχρονα στην ίδια πλατφόρμα. Για τον λόγο αυτό η Intel εισήγαγε τις τεχνολογίες CMT-CAT, ώστε να είναι διαθέσιμοι οι κατάλληλοι hardware μηχανισμοί για την διαχείριση των κοινόχρηστων πόρων.

Η τεχνολογία CMT αποτελεί μία νέα δυνατότητα η οποία επιτρέπει στο λειτουργικό σύστημα (operating system, OS), σε ένα λογισμικό ελέγχου (hypervisor) ή σε μία εικονική μηχανή (virtual machine, VM) να προσδιορίσει τη χρήση της κοινόχρηστης cache από εφαρμογές που εκτελούνται στην πλατφόρμα. Από την άλλη πλευρά η τεχνολογία CAT δίνει τη δυνατότητα στους παραπάνω μηχανισμούς να ελέγχουν την κατανομή του τελευταίου επιπέδου της cache σε έναν επεξεργαστή. Μετά την παραμετροποίηση της CAT, ο επεξεργαστής επιτρέπει την πρόσβαση μόνο στα τμήματα της cache που καθορίζονται από την κλάση υπηρεσιών (class of service, CLOS) με την οποία έχει αντιστοιχηθεί και τηρεί αυτή την πολιτική κατά τη διάρκεια εκτέλεσης μιας εφαρμογής, ενός νήματος ή μιας διεργασίας. Στις ενότητες που ακολουθούν παρουσιάζονται αναλυτικά οι δύο αυτοί μηχανισμοί.

3.1 Η τεχνολογία Cache Monitoring Technology (CMT)

Η τεχνολογία CMT της Intel ήταν διαθέσιμη για πρώτη φορά το 2014 στο σύνολο των προϊόντων Intel Xeon E5 2600 v3 και αποτελεί μέρος μιας μεγαλύτερης σειράς τεχνολογιών, της Intel(r) Resource Director Technology (RDT). Ο μηχανισμός CMT δίνει τη δυνατότητα παρακολούθησης της χρήσης των διαμοιραζόμενων πόρων της πλατφόρμας και επιτρέπει βελτιωμένη εξαγωγή προφίλ εφαρμογών, καλύτερη χρονοδρομολόγηση (scheduling) και βελτιωμένη ανίχνευση εφαρμογών που μπορεί να χρησιμοποιούν υπερβολικά τους κοινούς πόρους και επομένως να μειώνουν την επίδοση άλλων συνεκτελούμενων εφαρμογών.

Η CMT παρέχει τη δυνατότητα παρακολούθησης της διαμοιραζόμενης L3 cache (τελευταίο επίπεδο της cache στους περισσότερους servers) και ειδικά του ποσοστού χρήσης της (occupancy) σε πραγματικό χρόνο. Προκειμένου να παρέχεται η μέγιστη δυνατή ευελιξία παρακολούθησης, η CMT λειτουργεί ανεξάρτητα από άλλες τεχνολογίες, όπως οι μετρητές παρακολούθησης επίδοσης ή οι τεχνολογίες εικονικοποίησης και παρέχει συνεχή παρακολούθηση της χρήσης της κοινόχρηστης cache μέσω ενός συνόλου μηχανισμών, των αναγνω-

ριστικών παρακολούθησης πόρων (Resource Monitoring IDs, RMIDs) που παρέχονται από την πλατφόρμα. Συγκεκριμένα είναι διαθέσιμοι οι εξής μηχανισμοί [24]:

1. Ένας μηχανισμός για την ανίχνευση των υπαρχόντων δυνατοτήτων παρακολούθησης στην πλατφόρμα (μέσω της εντολής CPUID).

2. Μία δομή για την απαρίθμηση των λεπτομερειών κάθε υπολειτουργίας (συμπεριλαμβανομένης της CMT).

3. Ένας μηχανισμός για το λειτουργικό σύστημα ή τον hypervisor για την απόδοση ενός λογισμικά καθορισμένου αναγνωριστικού (ID) σε κάθε ένα από τα νήματα λογισμικού (εφαρμογές, εικονικές μηχανές κτλ.) που είναι δρομολογημένα να εκτελεστούν σε έναν λογικό επεξεργαστή. Αυτά τα αναγνωριστικά είναι γνωστά ως Resource Monitoring IDs (RMIDs).

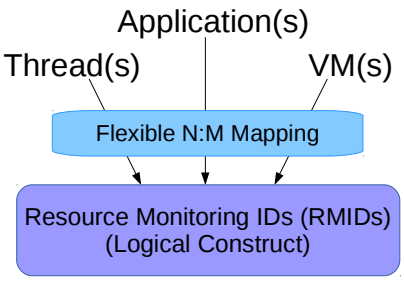
4. Μηχανισμοί στο hardware για τη λήψη στατιστικών χρήσης της cache και του bandwidth για κάθε RMID.

5. Μηχανισμοί για το λειτουργικό σύστημα ή τον hypervisor για να συλλέξει μετρικές όπως το L3 occupancy για ένα συγκεκριμένο RMID οποιαδήποτε στιγμή κατά το χρόνο εκτέλεσης.

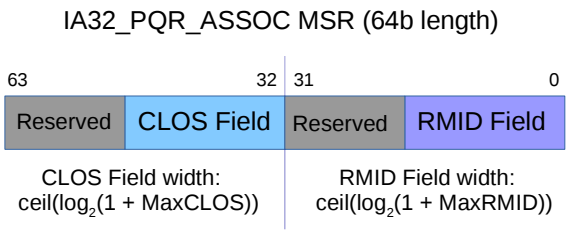
Τέλος, στην υποδομή της CMT βασίζεται ακόμη μία τεχνολογία, η τεχνολογία παρακολούθησης του εύρους ζώνης (Memory Bandwidth Monitoring, MBM), η οποία επιτρέπει τη μέτρηση του bandwidth από το ένα επίπεδο της cache στο ιεραρχικά επόμενό του. Στην περίπτωση της L3 cache, η οποία ακολουθείται από την κύρια μνήμη του συστήματος, μπορεί να καταμετρηθεί το bandwidth από τη μνήμη.

3.1.1 Ο μηχανισμός Resource Monitoring IDs (RMIDs)

Ο μηχανισμός CMT επιτρέπει την ανεξάρτητη και ταυτόχρονη παρακολούθηση πολλών συνεκτελούμενων νημάτων σε έναν πολυπύρηνο επεξεργαστή μέσω ενός μηχανισμού γνωστού ως Resource Monitoring ID (RMID). Κάθε λογικός επεξεργαστής, (hardware) thread, του συστήματος μπορεί να αντιστοιχηθεί με ένα ξεχωριστό RMID ή πολλοί λογικοί επεξεργαστές να αντιστοιχηθούν με το ίδιο RMID (π.χ. για την ανίχνευση μιας εφαρμογής με πολλαπλά νήματα). Επιπλέον, εκτός από λογικούς επεξεργαστές, το λειτουργικό σύστημα μπορεί να αντιστοιχίσει νήματα, εφαρμογές ή VMs με ένα RMID ανάλογα με τις ανάγκες παρακολούθησης, όπως φαίνεται στην Εικόνα 3.1.



Εικόνα 3.1: Αντιστοίχιση των RMIDs



Εικόνα 3.2: Η δομή του καταχωρητή PQR

Για κάθε λογικό επεξεργαστή μόνο ένα RMID είναι ενεργό κάθε φορά. Αυτό επιβάλλεται από τον καταχωρητή IA32_PQR_ASSOC MSR (PQR), ο οποίος καθορίζει το ενεργό RMID

ενός λογικού επεξεργαστή. Γράφοντας μέσω του λογισμικού στο κατάλληλο πεδίο του καταχωρητή PQR, όπως φαίνεται στην Εικόνα 3.2, είναι δυνατή η αλλαγή της παλιάς τιμής του RMID με μία νέα. Επομένως όταν μία εφαρμογή δρομολογείται σε έναν πυρήνα (core), το λειτουργικό σύστημα ενημερώνει τον καταχωρητή PQR με το RMID που έχει αντιστοιχηθεί στην εφαρμογή αυτή. Επιπλέον στην Εικόνα 3.2 φαίνεται το πεδίο CLOS που αντιστοιχεί στην κλάση υπηρεσίας (class of service) και χρησιμοποιείται για την κατανομή των πόρων όπως θα παρουσιαστεί στην Ενότητα 3.2. Τέλος, παρέχεται πλήθος ανεξάρτητων RMIDs που επιτρέπουν την ατομική παρακολούθηση πολλαπλών ανεξάρτητων νημάτων. Ωστόσο ο αριθμός των διαθέσιμων RMIDs ανά επεξεργαστή ποικίλει και είναι μία από τις παραμέτρους που απαριθμούνται με την εντολή CPUID [25].

3.1.2 Ανίχνευση υποστήριξης CMT

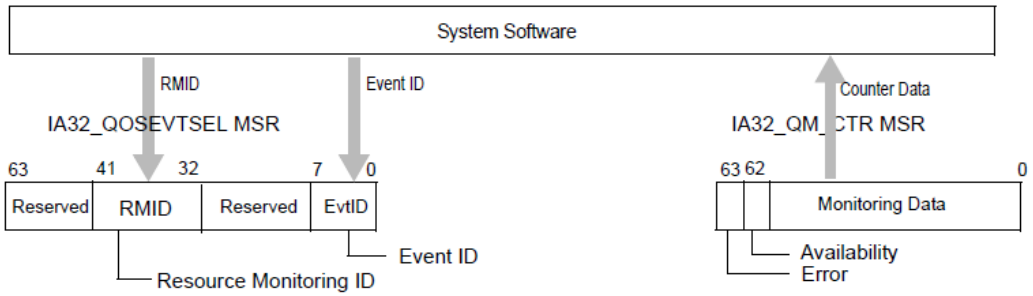
Η εντολή CPUID χρησιμοποιείται για την απαρίθμηση όλων των παραμέτρων της CMT, οι οποίες μπορεί να αλλάζουν ανά γενιά υπολογιστών, συμπεριλαμβανομένου του αριθμού των διαθέσιμων RMIDs. Για να ανιχνευτεί γενικά η παρουσία των λειτουργιών παρακολούθησης στην πλατφόρμα, ελέγχεται το bit 12 στο CPUID.0x7.0 (ένα διάνυσμα που περιέχει bits για να υποδείξει την παρουσία πολλαπλών διαφορετικών τύπων χαρακτηριστικών στον επεξεργαστή).

Μόλις επιβεβαιωθεί η παρουσία της λειτουργίας παρακολούθησης, οι πόροι στους οποίους υποστηρίζεται η λειτουργία αυτή μπορούν να απαριθμηθούν μέσω του CPUID.0xF. Μόλις επιβεβαιωθεί και η υποστήριξη για έναν συγκεκριμένο πόρο, μπορούν να προσδιοριστούν τα χαρακτηριστικά κάθε επιπέδου παρακολούθησης. Για παράδειγμα, οι λεπτομέρειες για την CMT στην L3 cache, όπως ο αριθμός των RMIDs, απαριθμούνται στο CPUID.0xF.1 [25].

3.1.3 Καθορισμός πόρων και Λήψη μετρήσεων

Μετά την επιβεβαίωση της ύπαρξης υποστήριξης CMT και γνωρίζοντας τον αριθμό των RMIDs, κάθε thread μπορεί να συσχετιστεί με ένα RMID μέσω του πεδίου PQR MSR RMID. Έπειτα από ένα χρονικό διάστημα (το οποίο καθορίζεται από το λογισμικό), ακριβείς μετρήσεις της χρήσης της L3 cache ή άλλων γεγονότων μπορούν να συλλεχθούν αυτόματα από το hardware και οι τιμές αυτές να διαβαστούν περιοδικά από το λογισμικό.

Προκειμένου το λογισμικό να λάβει τα αποτελέσματα των μετρήσεων, παρέχονται δύο καταχωρητές, η δομή των οποίων παρουσιάζεται στην Εικόνα 3.3 [24]. Ο πρώτος καταχω-

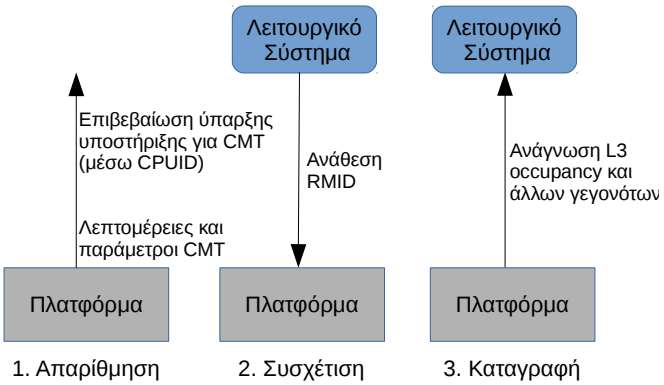


Εικόνα 3.3: Χρήση καταχωρητών IA32_QOSEVTSEL και IA32_QM_CTR

ρητής, IA32_QM_EVTSEL, επιτρέπει στο λογισμικό να προσδιορίσει ένα ζεύγος RMID - Event ID (αναγνωριστικό συμβάντος) για το οποίο θα ανακτηθούν τα δεδομένα. Το RMID έχει αποδοθεί προηγουμένως στην εφαρμογή, thread ή VM από το λειτουργικό σύστημα και

το Event ID προσδιορίζει τον τύπο των δεδομένων που θα ανακτηθούν, όπως το L3 cache occupancy. Μόλις το λογισμικό προσδιορίσει το ζευγάρι RMID - Event ID, το hardware αναζητά τα καθορισμένα δεδομένα και τα επιστρέφει στον καταχωρητή IA32_QM_CTR MSR, ο οποίος περιέχει επιπλέον ορισμένα bits που αντιστοιχούν σε αναφορά σφάλματος για την εξασφάλιση της ορθότητας των μετρήσεων. Στην περίπτωση του L3 cache occupancy, τα δεδομένα που επιστρέφονται από τον IA32_QM_CTR MSR μπορεί προαιρετικά να πολλαπλασιαστούν με ένα συντελεστή (παρέχεται από την εντολή CPUID) για να μετατραπούν σε bytes πριν τη χρησιμοποίησή τους από το λογισμικό.

Συνοψίζοντας, στην Εικόνα 3.4 παρουσιάζονται συγκεντρωμένα τα βήματα για μία ολοκληρωμένη χρήση του μηχανισμού CMT, η οποία περιλαμβάνει τη γνωστοποίηση ορισμένων χαρακτηριστικών του συστήματος μέσω της εντολής CPUID, την ανάθεση των RMIDs στις εφαρμογές και τέλος τον καθορισμό και τη λήψη συγκεκριμένων μετρικών [1, 24].



Εικόνα 3.4: Η διαδικασία χρήσης τριών βημάτων του μηχανισμού CMT

3.2 Η τεχνολογία Cache Allocation Technology (CAT)

Οι τελευταίες γενιές επεξεργαστών Intel Xeon προσφέρουν τη δυνατότητα χρήσης και παραμετροποίησης της τεχνολογίας κατανομής της cache (Cache Allocation Technology, CAT). Η CAT δεν απαιτεί καμία τροποποίηση του λειτουργικού συστήματος ή του kernel για να χρησιμοποιηθεί. Με τον καθορισμό και την ανάθεση μιας κλάσης υπηρεσίας (CLOS) σε κάθε πυρήνα, ο χρήστης (το λειτουργικό σύστημα, ο hypervisor ή ο ελεγκτής VMs) μπορεί να αναθέσει τμήματα της LLC σε συγκεκριμένους πυρήνες περιορίζοντας έτσι το μέρος της LLC στο οποίο κάθε πυρήνας είναι σε θέση να εκχωρήσει δεδομένα. Για τον λόγο αυτό δεν είναι πλέον δυνατό για τον πυρήνα να εκδιώξει γραμμές της κρυφής μνήμης (cache lines) εκτός της περιοχής που του αποδόθηκε.

Η μνήμη cache είναι διαιρεμένη σε προκαθορισμένα ίσα τμήματα-δρόμους, που ονομάζονται ways, τα οποία μπορούν να χωριστούν ή να μοιραστούν μεταξύ των πυρήνων μέσω της CAT. Αν και ένας πυρήνας μπορεί να έχει περιοριστεί σε ένα υποσύνολο της LLC για την εισαγωγή και απομάκρυνση cache lines, η ανάγνωση ή η εγγραφή από έναν πυρήνα μπορεί να έχει ως αποτέλεσμα ευστοχία στην cache (cache hit) εάν η cache line υπάρχει οπουδήποτε στην LLC.

Η CAT είναι ιδιαίτερα χρήσιμη σε σενάρια όπου μία εφαρμογή ζητά ένα μεγάλο όγκο δεδομένων, ασκώντας πίεση σε άλλες εφαρμογές, και δεν επαναχρησιμοποιεί τα δεδομένα που έφερε στην LLC. Μία τέτοια εφαρμογή μπορεί να είναι ένα πρόγραμμα συνεχούς ροής βίντεο (video streaming program). Αυτού του είδους οι εφαρμογές μπορούν να καταναλώσουν

ολόκληρη την LLC, χωρίς όμως να επωφεληθούν από αυτή αφού δεν επαναχρησιμοποιούν τα περισσότερα από τα δεδομένα που ζητούν. Επομένως θα ήταν ωφέλιμο εάν ο πυρήνας στον οποίο εκτελείται μία τέτοια εφαρμογή περιοριστεί σε μία μικρή περιοχή της cache. Ως αποτέλεσμα, άλλες εφαρμογές έχουν καλύτερη ευκαιρία να επωφεληθούν από την LLC, αν και υπάρχει πιθανότητα μείωσης της επίδοσης της παραπάνω εφαρμογής. Στους βασικούς μηχανισμούς της CAT περιλαμβάνονται [26] :

1. Ένας μηχανισμός για την ανίχνευση των διαθέσιμων δυνατοτήτων κατανομής της μνήμης στην πλατφόρμα και των διαθέσιμων τύπων πόρων στους οποίους μπορούν να εφαρμοστούν (μέσω της εντολής CPUID).

2. Ένας μηχανισμός για το λειτουργικό σύστημα ή τον hypervisor για την ανάθεση συγκεκριμένου μέρους ενός πόρου σε μία κλάση υπηρεσίας (class of service, CLOS).

3. Ένας μηχανισμός για το λειτουργικό σύστημα ή τον hypervisor για την αντιστοίχιση μιας εφαρμογής με μία κλάση υπηρεσιών.

4. Hardware μηχανισμοί για την καθοδήγηση της πολιτικής πλήρωσης της LLC ανάλογα με την κλάση υπηρεσίας στην οποία ανήκει κάθε εφαρμογή.

3.2.1 Ο μηχανισμός Class of Service (CLOS)

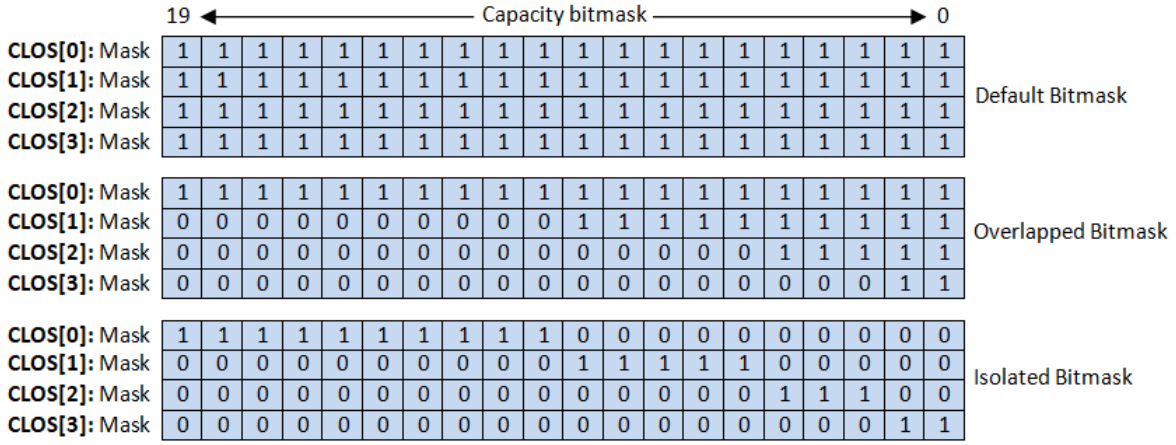
Ο μηχανισμός CAT είναι ξεχωριστός, αλλά παρόμοιος, με τον CMT που αναπτύχθηκε στην Ενότητα 3.1. Ο επεξεργαστής παρέχει ένα σύνολο κλάσεων υπηρεσίας (CLOS), στις οποίες μπορούν να αντιστοιχηθούν οι εφαρμογές (ή threads). Στη συνέχεια η μνήμη cache για τις αντίστοιχες εφαρμογές ή threads περιορίζεται με βάση το CLOS με το οποίο σχετίζονται. Κάθε CLOS μπορεί να διαμορφωθεί με τη χρήση μασκών bit (bitmasks) που αντιπροσωπεύουν τη χωρητικότητα της LLC και υποδηλώνουν το βαθμό επικάλυψης και απομόνωσης μεταξύ των κλάσεων. Για κάθε λογικό επεξεργαστή υπάρχει ένας καταχωρητής, ο IA32_PQR_ASSOC MSR (PQR), που επιτρέπει στο λειτουργικό σύστημα να καθορίσει ένα CLOS όταν δρομολογηθεί μία εφαρμογή, thread ή VM. Ο καταχωρητής αυτός είναι ο ίδιος με αυτόν που χρησιμοποιείται για την απόδοση του RMID, όπως αναλύθηκε στην Ενότητα 3.1.1, και ο πλήρης ορισμός του παρουσιάζεται στην Εικόνα 3.2. Ωστόσο η επιλογή του CLOS δεν σχετίζεται με την επιλογή του RMID, επιτρέποντας έτσι την ανεξάρτητη υλοποίηση των τεχνολογιών παρακολούθησης και κατανομής της μνήμης.

Αφού καθοριστεί το CLOS, η κατανομή της μνήμης cache για την υποδεικνυόμενη εφαρμογή, thread, ή VM ελέγχεται αυτόματα από το hardware σύμφωνα με το CLOS που της έχει αποδοθεί και τη bitmask που σχετίζεται με αυτό. Οι bitmasks διαμορφώνονται μέσω των καταχωρητών IA32_resourceType_MASK_n MSRs, όπου το resourceType αντιστοιχεί στον τύπο πόρων (π.χ. ”L3” για την L3 cache) και το n στον αριθμό του CLOS. Μία μάσκα χωρητικότητας (capacity bitmask, CBM) υποδεικνύει στο hardware το χώρο της μνήμης cache όπου πρέπει να περιοριστεί η εφαρμογή, καθώς και μία ένδειξη επικάλυψης και απομόνωσης της cache από άλλες εφαρμογές που την διεκδικούν. Το μήκος (σε bits) της μάσκας χωρητικότητας εξαρτάται γενικά από τον τρόπο διαμόρφωσης της cache και μπορεί να βρεθεί μέσω της εντολής CPUID (μπορεί να διαφέρει μεταξύ μοντέλων σε μία οικογένεια επεξεργαστών).

Στην Εικόνα 3.5 παρουσιάζεται ένα παράδειγμα μασκών χωρητικότητας της cache με μήκος 20 bits. Γενικά αναμένεται ότι σε υλοποιήσεις βασισμένες σε ways, ένα bit της μάσκας αντιστοιχεί σε έναν ορισμένο αριθμό ways της cache, συνήθως σε ένα way, αλλά η συγκεκριμένη αντιστοίχιση εξαρτάται από την εκάστοτε υλοποίηση. Ωστόσο, σε όλες τις περιπτώσεις,

ένα bit της μάσκας που έχει οριστεί σε ”1” καθορίζει ότι μία συγκεκριμένη κλάση υπηρεσίας (CLOS) μπορεί να τοποθετήσει δεδομένα στο υποσύνολο της cache που αντιστοιχεί σε αυτό το bit. Αντίθετα η τιμή ”0” σε ένα bit της μάσκας καθορίζει ότι η κλάση υπηρεσίας δεν μπορεί να τοποθετήσει δεδομένα στο συγκεκριμένο υποσύνολο της cache. Ωστόσο γίνονται δεκτοί μόνο συνεχείς συνδυασμοί από ”1” (π.χ. FFFFFH, 00FF0H κτλ.).

Συγκεκριμένα, η Εικόνα 3.5 περιλαμβάνει τρία παραδείγματα μασκών χωρητικότητας της cache, που παρουσιάζονται ως πίνακες των 20 bits. Το πρώτο παράδειγμα δείχνει την προεπιλεγμένη περίπτωση, όπου και οι τέσσερις κλάσεις υπηρεσιών (ο συνολικός τους αριθμός ποικίλει και εξαρτάται από την υλοποίηση) έχουν πλήρη πρόσβαση στη μνήμη cache. Το δεύτερο παράδειγμα παρουσιάζει μία επικαλυπτόμενη περίπτωση, η οποία σε ένα υποθετικό σενάριο θα επέτρεπε σε ορισμένα threads με χαμηλότερη προτεραιότητα να μοιράζονται χώρο στη μνήμη cache με threads υψηλότερης προτεραιότητας, θεωρώντας πως στο CLOS0 αντιστοιχίζονται οι εφαρμογές με τη μεγαλύτερη προτεραιότητα, ακολουθούμενο από το CLOS1 κ.ο.κ., αν και δεν υπάρχει περιορισμός στο hardware που να επιβάλει αυτή την πολιτική. Το τρίτο παράδειγμα δείχνει μερικές περιπτώσεις μη-επικαλυπτόμενων συνδυασμών διαμοιρασμού της cache, προσφέροντας πλήρη απομόνωση των τμημάτων της cache. Αξίζει να σημειωθεί πως με την εκκίνηση του συστήματος όλα τα threads αρχικοποιούνται στο CLOS0, που έχει από προεπιλογή πλήρη πρόσβαση στη μνήμη cache [1, 26].



Εικόνα 3.5: Παραδείγματα μασκών - CLOS Bitmasks

3.2.2 Ανίχνευση υποστήριξης CAT

Με την εντολή CPUID είναι δυνατή η ανίχνευση υποστήριξης της τεχνολογίας CAT στην πλατφόρμα, όπως και η εύρεση όλων των παραμέτρων της. Για να ανιχνευτεί γενικά η παρουσία της λειτουργίας κατανομής της μνήμης στην πλατφόρμα, ελέγχεται το bit 15 στο CPUID.0x7.0 (ένα διάνυσμα που περιέχει bits για να υποδείξει την παρουσία πολλαπλών διαφορετικών τύπων χαρακτηριστικών στον επεξεργαστή). Μετά την επιβεβαίωση της παρουσίας του μηχανισμού CAT, μπορούν να βρεθούν λεπτομερείς πληροφορίες για τις παραμέτρους του μέσω του CPUID 0x10. Παραδείγματα αυτών των παραμέτρων είναι οι πόροι στους οποίους υποστηρίζεται η λειτουργία αυτή, ο αριθμός των κλάσεων υπηρεσίας (CLOS), το μήκος των μασκών χωρητικότητας κ.ά. [26].

3.3 Υλοποίηση μηχανισμών CMT-CAT

Η χρήση των δυνατοτήτων παρακολούθησης και κατανομής της κρυφής μνήμης μπορεί να διαφέρει ανά πλατφόρμα ή λειτουργικό σύστημα. Ωστόσο από πλευράς ανάπτυξης κώδικα είναι σχετικά απλή, αφού οι καταχωρητές συγκεκριμένων μοντέλων (model specific registers, MSR) παρέχουν τη διασύνδεση για την αξιοποίηση των μηχανισμών. Όλα τα σύγχρονα λειτουργικά συστήματα παρέχουν διεπαφές προγραμματισμού εφαρμογών (Application Programming Interfaces, APIs) που επιτρέπουν στους χρήστες να διαβάζουν και να γράφουν στους καταχωρητές MSR. Για παράδειγμα, το Linux παρέχει ένα πακέτο εργαλείων που περιέχει τις εντολές rdmsr και wrmsr για την ανάγνωση και την εγγραφή των MSRs αντίστοιχα. Υπάρχουν δύο προσεγγίσεις παρακολούθησης της κρυφής μνήμης, μία αυτόνομη και μία βασισμένη στον scheduler προσέγγιση [1], οι οποίες παρουσιάζονται στις επόμενες ενότητες.

3.3.1 Αυτόνομη παρακολούθηση της cache

Η προσέγγιση της αυτόνομης παρακολούθησης της cache εξετάζει τη χρήση του τελευταίου επιπέδου της κρυφής μνήμης από την πλευρά του πυρήνα ή thread, ανεξάρτητα από τις εφαρμογές που εκτελούνται. Ένα αναγνωριστικό παρακολούθησης πόρων (Resource Monitoring ID, RMID) αποδίδεται στατικά σε έναν πυρήνα/thread και περιοδικά μετριέται η χρήση της LLC. Επομένως εάν η απόδοση των RMIDs γίνει στατικά και οι εφαρμογές εκτελούνται σε συγκεκριμένους πυρήνες, τότε αυτή η μέθοδος αποφέρει αποδεκτά αποτελέσματα. Η συγκεκριμένη μέθοδος χρησιμοποιήθηκε και στην παρούσα διπλωματική εργασία.

3.3.2 Παρακολούθηση της cache βασισμένη στον scheduler

Η προσέγγιση της βασισμένης στον scheduler παρακολούθησης της cache περιλαμβάνει, όπως είναι λογικό, τη συμμετοχή του scheduler του λειτουργικού συστήματος [1]. Στην προηγουμένως αναφερόμενη αυτόνομη μέθοδο, το RMID/CLOS δεν αντιστοιχίζεται με το αναγνωριστικό (id) της διεργασίας και επομένως δεν είναι δυνατή η μέτρηση της χρήσης της LLC ανά εφαρμογή, εκτός και αν ο χρήστης δηλώσει ρητά την εκτέλεσή της σε ένα συγκεκριμένο πυρήνα. Στην περίπτωση που απαιτείται η παρακολούθηση των εφαρμογών καθώς αυτές μετακινούνται μεταξύ των πυρήνων, είναι αναγκαίες ορισμένες αλλαγές στον scheduler. Ο χρήστης, μέσω του κατάλληλου λογισμικού, αναθέτει ένα RMID σε μία εφαρμογή και με τη σειρά του ο scheduler συνδέει τον πυρήνα, στον οποίο έχει προγραμματιστεί να εκτελεστεί η εφαρμογή αυτή, με το RMID της. Όταν η εφαρμογή τερματιστεί ή μετακινηθεί σε ένα διαφορετικό πυρήνα, ο scheduler ενημερώνει την ανάθεση RMID για να σιγουρευτεί ότι η μέτρηση της χρήσης της LLC γίνεται από τα σωστά RMIDs και μόνο όταν η υπό εξέταση εφαρμογή εκτελείται. Το λογισμικό του συστήματος είναι επίσης υπεύθυνο για την ενημέρωση των RMIDs σε περίπτωση μετακίνησης ή εκ νέου ανάθεσης της εφαρμογής σε διαφορετικό socket επεξεργαστών. Δεδομένου ότι τα RMIDs και CLOS είναι ξεχωριστά για κάθε socket, όταν μια εφαρμογή μεταφέρεται σε άλλο socket, το λειτουργικό σύστημα είναι υπεύθυνο για την εύρεση διαθέσιμων RMID και CLOS στο socket προορισμού ώστε να συνεχιστεί η παρακολούθηση της εφαρμογής.

3.3.3 Η βιβλιοθήκη PQoS

Η Intel έχει αναπτύξει μία αυτόνομη βιβλιοθήκη, την PQoS (διαθέσιμη στη διεύθυνση: https://01.org/), η οποία παρέχει υποστήριξη για τους μηχανισμούς CMT-CAT. Με την εγκατάσταση της βιβλιοθήκης ελέγχεται εάν η πλατφόρμα υποστηρίζει τους μηχανισμούς CMT-

CAT. Εφόσον επιβεβαιωθεί η ύπαρξη των μηχανισμών στην πλατφόρμα, η βιβλιοθήκη παρέχει μία διεπαφή υψηλού επιπέδου που δίνει τη δυνατότητα στους προγραμματιστές να παρακολουθήσουν τη χρήση της LLC, των misses της LLC και το bandwidth από τη μνήμη ανά πυρήνα ή thread, αλλά και να κάνουν χρήση της λειτουργίας κατανομής της cache. Με τις δυνατότητες αυτές μπορούν να αναπτυχθούν βρόχοι ανάδρασης για τη δυναμική μεταβολή της κατανομής της LLC στους πυρήνες/threads. Αυτή η βιβλιοθήκη χρησιμοποιήθηκε και επεκτάθηκε για τις ανάγκες της παρούσας διπλωματικής εργασίας.

Κεφάλαιο 4

Πειραματική μεθοδολογία

Στο κεφάλαιο αυτό παρουσιάζεται η μεθοδολογία διεξαγωγής όλων των πειραμάτων που πραγματοποιήθηκαν στην παρούσα διπλωματική εργασία. Τα πειράματα αυτά πραγματοποιήθηκαν σε πραγματικό πολυπύρηνο σύστημα, το οποίο υποστηρίζει τις νέες τεχνολογίες CMT και CAT της Intel για την παρακολούθηση και την κατανομή του τελευταίου επιπέδου των caches (LLC) αντίστοιχα. Στις επόμενες ενότητες παρουσιάζονται με τη σειρά τα ακόλουθα:

- Αφού αναφερθούν τα χαρακτηριστικά του πραγματικού συστήματος, γίνεται παρουσίαση των εφαρμογών που χρησιμοποιήθηκαν στα πειράματα.
- Έπειτα παρουσιάζονται τα 20 πειράματα που πραγματοποιήθηκαν για κάθε εφαρμογή, η οποία εκτελέστηκε αρχικά μόνη της στο σύστημα και σε κάθε πείραμα της αποδόθηκε διαφορετικό ποσοστό της LLC. Με αυτόν τον τρόπο εξάγεται ένα προφίλ για κάθε εφαρμογή, το οποίο συμβάλει στην κατανόηση των ποικίλων συμπεριφορών των εφαρμογών ανάλογα με το τμήμα της cache που έχουν στη διάθεσή τους.
- Σύμφωνα με αυτά τα προφίλ, γίνεται η κατηγοριοποίηση των εφαρμογών σε τρία είδη σε θεωρητικό επίπεδο με βάση τη συμπεριφορά τους καθώς μεταβάλλεται το μέγεθος της cache.
- Στη συνέχεια περιγράφεται η υλοποίηση των κυκλωμάτων παρακολούθησης της χρησιμότητας της cache UMON (utility monitoring circuits) [4] σε επίπεδο προσομοίωσης, καθώς δεν έχουν υλοποιηθεί στο hardware. Από τα κυκλώματα αυτά εξάγονται τα αντίστοιχα προφίλ των εφαρμογών, τα οποία χρησιμοποιήθηκαν τόσο για την υλοποίηση της τεχνικής διαμοιρασμού της cache UCP, όσο και στον αντίστοιχο μηχανισμό που προτείνουμε στο Κεφάλαιο 5.
- Μετά από την παραπάνω ανάλυση των εφαρμογών, παρουσιάζεται μία σειρά 100 πειραμάτων όπου σε κάθε πείραμα εκτελέστηκαν ταυτόχρονα 8 εφαρμογές σε 8 διαφορετικούς πυρήνες. Για την εκτέλεση των εφαρμογών και τη λήψη των κατάλληλων μετρήσεων χρησιμοποιήθηκε και επεκτάθηκε η βιβλιοθήκη PQoS (Ενότητα 3.3.3) η οποία κάνει χρήση των τεχνολογιών CMT-CAT της Intel.
- Έπειτα αναλύονται τα πειράματα συνεκτελέσεων που πραγματοποιήθηκαν στο πραγματικό μηχάνημα χωρίς την επιβολή κάποιου διαχωρισμού στην LLC, έτσι ώστε να αναδειχθεί το πρόβλημα της μείωσης της απόδοσης του συστήματος λόγω της διαμάχης των εφαρμογών για τους κοινόχρηστους πόρους.
- Τέλος, παρουσιάζεται η υλοποίηση της τεχνικής διαμοιρασμού της cache UCP και η αξιολόγησή της πραγματοποιώντας τα πειράματα συνεκτελέσεων στο πραγματικό μηχάνημα και εφαρμόζοντας αυτήν την πολιτική διαχωρισμού στην LLC.

4.1 Σύστημα πειραματικής αξιολόγησης

Όλα τα πειράματα της παρούσας διπλωματικής εργασίας πραγματοποιήθηκαν στον επεξεργαστή Intel Xeon E5-2630 v4 της γενιάς μικροαρχιτεκτονικής Broadwell. Στον συγκεκριμένο επεξεργαστή υποστηρίζονται οι τεχνολογίες CMT-CAT της Intel για την παρακολούθηση και τον διαμοιρασμό της κοινόχρηστης L3 cache (LLC), οι οποίες παρουσιάστηκαν στο Κεφάλαιο 3. Συγκεκριμένα, περιλαμβάνει 80 RMIDs, 16 CLOS και η μάσκα χωρητικότητας κάθε CLOS έχει μήκος 20 bits, δηλαδή ένα bit της μάσκας αντιστοιχεί σε ένα way της LLC. Τα χαρακτηριστικά του συστήματος πειραματικής αξιολόγησης παρουσιάζονται στον Πίνακα 4.1.

Processor	Intel Xeon E5-2630 v4 (Broadwell) 10 cores, 2.2GHz
L1 cache	Icache and Dcache: 32KB, private per core, 8-way set associative
L2 cache	256KB, private per core, 8-way set associative
LLC	25MB, shared, 20-way set associative
Memory	256GB
Memory Bandwidth	64GB/s
OS	Debian 8.7, Linux kernel version 4.7.0

Πίνακας 4.1: Χαρακτηριστικά Συστήματος

4.2 Χαρακτηριστικά εφαρμογών

Για τη διεξαγωγή των πειραμάτων χρησιμοποιήθηκαν 35 εφαρμογές-μετροπρογράμματα (benchmarks), 9 από το πακέτο PARSEC-3.0 [27] και 26 από το πακέτο SPEC CPU2006 [28], τα οποία παρουσιάζονται στον Πίνακα 4.2. Πραγματοποιήθηκε ανάλυση των παραπάνω εφαρμογών προκειμένου να γίνει κατανοητή η συμπεριφορά τους ανάλογα με το ποσοστό της LLC που έχουν στη διάθεσή τους αλλά και να γίνει εξαγωγή ορισμένων χαρακτηριστικών που χρησιμοποιήθηκαν για την επιλογή των εφαρμογών στα πειράματα συνεκτελέσεων.

Suit	**Benchmarks**
PARSEC 3.0	blackscholes, bodytrack, canneal, dedup, ferret, fluidanimate, rtview, streamcluster, swaptions
SPEC CPU2006	astar, bwaves, bzip2, cactusADM, calculix, gcc, GemsFDTD, gobmk, gromacs, h264ref, hmmer, lbm, leslie3d, libquantum, mcf, milc, namd, omnetpp, perlbench, povray, sjeng, soplex, sphinx3, tonto, xalancbmk, zeusmp

Πίνακας 4.2: Διαθέσιμα benchmarks

Συγκεκριμένα, κάθε εφαρμογή εκτελέστηκε μόνη της στο πραγματικό μηχάνημα (*κατάσταση Alone*) ώστε να μελετηθεί η συμπεριφορά της όταν δεν έρχεται σε σύγκρουση με άλλες εφαρμογές για τους κοινούς πόρους. Κάθε εφαρμογή εκτελέστηκε 20 φορές. Στην πρώτη εκτέλεση αποδόθηκε 1 way (1280KB) της LLC στην εκάστοτε εφαρμογή και σε κάθε επόμενη εκτέλεση το τμήμα της LLC που της κατανεμόταν αυξανόταν κατά ένα way. Επομένως στην 20ή εκτέλεση η εφαρμογή είχε πλήρη πρόσβαση στην LLC. Η κατανομή της μνήμης στην εφαρμογή και η λήψη των μετρικών, όπως το ipc, τα misses και το ποσοστό χρήσης

της LLC, έγινε με τη χρήση των τεχνολογιών CAT και CMT της Intel αντίστοιχα και μέσω της βιβλιοθήκης PQoS (Ενότητα 3.3.3), η οποία εκτελείται σε διαφορετικό socket του μηχανήματος προκειμένου να έχει πρόσβαση σε διαφορετική LLC και να μην επηρεάζει την εκτελούμενη εφαρμογή.

Στον Πίνακα 4.3 παρουσιάζονται τα benchmarks με αλφαβητική σειρά μαζί με το IPC και MPKI (misses per kilo instructions, αστοχίες ανά χιλιάδα εντολών) του κάθε benchmark κατά την εκτέλεση που είχε πλήρη πρόσβαση στην LLC. Επιπλέον αναφέρεται το χαρακτηριστικό $w_{95\%}$, δηλαδή ο ελάχιστος αριθμός των ways που απαιτείται ώστε κάθε benchmark να φτάσει στο 95% της μέγιστης απόδοσής του (σύμφωνα με το IPC). Στο Παράρτημα B παρουσιάζονται για όλα τα benchmarks οι γραφικές παραστάσεις του IPC και του MPKI συναρτήσει του μεγέθους της LLC. Παρατηρώντας τις τιμές του MPKI και του $w_{95\%}$ είναι φανερό πως η συμπεριφορά των εφαρμογών ποικίλει, αφού υπάρχουν εφαρμογές με σχεδόν μηδενικό MPKI και την απαίτηση μόνο ενός way της LLC για την επίτευξη του 95% της μέγιστης απόδοσής τους, ενώ άλλες έχουν έως και 18 MPKI και απαιτούν τουλάχιστον 18 ways για την επίτευξη του 95% της μέγιστης απόδοσής τους. Επομένως προκύπτει η ανάγκη για την κατηγοριοποίηση των εφαρμογών ανάλογα με τη συμπεριφορά τους καθώς μεταβάλλεται το ποσοστό της LLC που τους αποδίδεται. Η κατηγοριοποίηση αυτή των εφαρμογών παρουσιάζεται σε θεωρητικό επίπεδο στην Ενότητα 4.3.

Bench	IPC	MPKI	$w_{95\%}$	Bench	IPC	MPKI	$w_{95\%}$
astar	1.09	0.51	12	leslie3d	2.12	10.36	5
blackscholes	1.81	0.55	1	libquantum	2.20	18.28	18
bodytrack	2.32	0.01	2	mcf	0.53	27.71	8
bwaves	2.43	13.05	2	milc	1.22	25.56	2
bzip2	1.49	0.13	4	namd	2.24	0.01	1
cactusADM	1.90	2.24	7	omnetpp	0.95	5.73	18
calculix	3.14	0.08	2	perlbench	2.40	0.01	2
canneal	0.45	16.83	11	povray	2.37	0.00	2
dedup	1.72	0.29	2	rtview	2.28	0.63	2
ferret	1.65	0.57	2	sjeng	1.56	0.32	2
fluidanimate	1.95	1.72	2	soplex	1.17	2.13	16
gcc	1.54	1.82	13	sphinx3	2.23	0.00	5
GemsFDTD	1.77	18.14	3	streamcluster	0.93	15.52	2
gobmk	1.21	0.14	2	swaptions	2.24	0.00	2
gromacs	2.37	0.00	2	tonto	2.31	0.01	2
h264ref	2.86	0.00	2	xalancbmk	1.83	0.90	12
hmmer	2.64	0.00	1	zeusmp	1.95	4.47	2
lbm	1.41	15.35	6				

Πίνακας 4.3: Χαρακτηριστικά των benchmarks

4.3 Κατηγοριοποίηση εφαρμογών

Όπως παρατηρήθηκε στην προηγούμενη ενότητα, η απόδοση κάθε εφαρμογής διαφέρει καθώς αυξάνεται ο αριθμός των ways της LLC που της αποδίδονται. Υπάρχουν εφαρμογές που επωφελούνται από την αύξηση του τμήματος της cache, ενώ σε άλλες δεν έχει ή

έχει ελάχιστη επίδραση στην επίδοσή τους. Σύμφωνα με αυτόν το συλλογισμό προκύπτουν τρία είδη εφαρμογών, οι cache friendly, cache fitting και cache thrashing εφαρμογές, χωρίς να είναι απόλυτος αυτός ο διαχωρισμός. Παρακάτω αναλύονται αυτές οι τρεις κατηγορίες εφαρμογών.

- **Cache friendly εφαρμογές:** Στην κατηγορία αυτή ανήκουν εφαρμογές των οποίων η απόδοση βελτιώνεται συνεχώς καθώς αυξάνεται ο αριθμός των ways που τους αποδίδεται. Συνήθως οι εφαρμογές αυτές επαναχρησιμοποιούν διαρκώς τα δεδομένα τους, γι' αυτό και όσο περισσότερη μνήμη έχουν στη διάθεσή τους τόσο υψηλότερη απόδοση πετυχαίνουν.

- **Cache fitting εφαρμογές:** Στην κατηγορία αυτή ανήκουν εφαρμογές οι οποίες χρειάζονται μόνο ένα υποσύνολο των διαθέσιμων ways ώστε να αποκτήσουν τη μέγιστη απόδοσή τους. Το μέγεθος του working set (ποσοστό μνήμης που χρειάζεται μία εφαρμογή σε ένα δεδομένο χρονικό διάστημα) των εφαρμογών αυτών είναι μικρό και χωράει στην LLC.

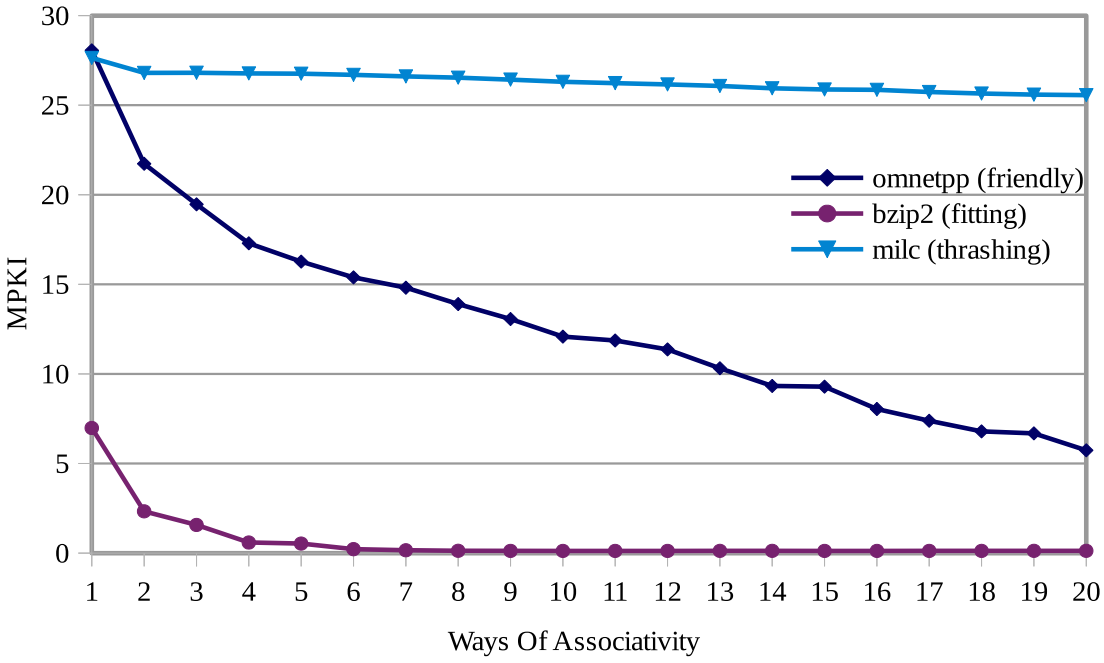
- **Cache thrashing εφαρμογές:** Στην κατηγορία αυτή ανήκουν εφαρμογές οι οποίες δεν επηρεάζονται ή επηρεάζονται ελάχιστα από το τμήμα της LLC που έχουν στη διάθεσή τους, ενώ πραγματοποιούν μεγάλο αριθμό αστοχιών. Οι εφαρμογές αυτές έχουν working set το οποίο είναι αρκετά μεγαλύτερο από το μέγεθος της LLC και πιθανώς να μην επαναχρησιμοποιούν τα δεδομένα τους (streaming εφαρμογές). Γι' αυτόν τον λόγο οι εφαρμογές αυτού του είδους πρέπει να περιορίζονται όταν εκτελούνται ταυτόχρονα με άλλες εφαρμογές.

Σύμφωνα με τους παραπάνω ορισμούς και τα χαρακτηριστικά των benchmarks που παρουσιάστηκαν στην Ενότητα 4.2 και στο Παράρτημα B, χωρίζουμε τα benchmarks στις τρεις κατηγορίες εφαρμογών, όπως φαίνεται στον Πίνακα 4.4, για το υπάρχον σύστημα πειραματικής αξιολόγησης. Ωστόσο ο συγκεκριμένος διαχωρισμός δεν είναι απόλυτος, αφού κάποια benchmarks μπορεί οριακά να ανήκουν σε περισσότερες κατηγορίες ή/και κατά τη διάρκεια εκτέλεσής τους να παρουσιάζουν διαφορετικές συμπεριφορές.

Friendly	astar, cactusADM, canneal, gcc, lbm, mcf, omnetpp, soplex, xalancbmk
Fitting	bodytrack, bzip2, calculix, dedup, ferret, gobmk, gromacs, h264ref, hmmer, namd, perlbench, povray, rtview, sjeng, sphinx3, swaptions, tonto
Thrashing	blackscholes, bwaves, fluidanimate, GemsFDTD, leslie3d, libquantum, milc, streamcluster, zeusmp

Πίνακας 4.4: Κατηγοριοποίηση των benchmarks

Στην Εικόνα 4.1 παρουσιάζεται η γραφική παράσταση του MPKI συναρτήσει του αριθμού των ways της LLC για τρία χαρακτηριστικά benchmarks, καθένα από τα οποία ανήκει σε μία διαφορετική κατηγορία εφαρμογών. Όπως φαίνεται το benchmark omnetpp ανήκει στην κατηγορία των cache friendly εφαρμογών καθώς ο αριθμός των misses ανά χιλιάδα εντολών μειώνεται διαρκώς καθώς αυξάνεται το ποσοστό της LLC που του αποδίδεται. Από την άλλη πλευρά το benchmark bzip2 ανήκει στην κατηγορία των cache fitting εφαρμογών διότι χρειάζεται μόνο 4 από τα 20 ways της LLC ώστε να έχει σχεδόν μηδενικό MPKI. Τέλος το benchmark milc ανήκει στις cache thrashing εφαρμογές, αφού παρουσιάζει σταθερά υψηλό MPKI ανεξάρτητα του αριθμού των ways που έχει στη διάθεσή του.



Εικόνα 4.1: MPKI για τρία ενδεικτικά benchmarks στην κατάσταση Alone

4.4 Εύρεση προφίλ εφαρμογών αντίστοιχων των UMON

Για τη διεξαγωγή πολλών πειραμάτων στην παρούσα διπλωματική εργασία ήταν απαραίτητη η πληροφορία που παρέχεται από τα κυκλώματα παρακολούθησης της χρησιμότητας της cache UMON, τα οποία παρουσιάστηκαν στην Ενότητα 2.2.1. Τα κυκλώματα αυτά παρέχουν τη δυνατότητα, δυναμικά, κατά τη διάρκεια εκτέλεσης πολλών εφαρμογών σε ένα σύστημα να καταμετρούνται τα hits κάθε εφαρμογής ανά way, που θα πραγματοποιούσε αν εκτελούταν μόνη της στο σύστημα και είχε πρόσβαση σε ολόκληρη την κοινόχρηστη cache. Δυστυχώς τα κυκλώματα UMON δεν έχουν υλοποιηθεί στο hardware και για τον λόγο αυτό υλοποιήθηκαν σε επίπεδο προσομοίωσης ώστε να εξαχθούν τα αντίστοιχα προφίλ των εφαρμογών που θα χρησιμοποιηθούν μετέπειτα σε πειράματα στο πραγματικό μηχάνημα.

Το εργαλείο προσομοίωσης που χρησιμοποιήθηκε είναι το Pin 3.2 [29] της Intel. Το Pin δίνει τη δυνατότητα δημιουργίας εργαλείων (pintools) για τη δυναμική ανάλυση εφαρμογών, τα οποία εισάγουν κώδικα ανάμεσα στις εντολές της εφαρμογής κατά τη διάρκεια εκτέλεσής της. Το Pin παρέχει κατάλληλες λειτουργίες για την καταμέτρηση των εντολών της εφαρμογής που αναλύει και αναγνωρίζει πότε πραγματοποιείται πρόσβαση στη μνήμη ενώ συγκρατεί την εικονική διεύθυνση του block της μνήμης στο οποίο γίνεται η πρόσβαση. Κάθε φορά που γίνεται πρόσβαση στη μνήμη στο πραγματικό μηχάνημα, ελέγχεται αρχικά η ιεραρχία των caches για την εύρεση του block, ξεκινώντας από την L1 cache και καταλήγοντας στην LLC, και έπειτα στην κύρια μνήμη, όπως έχει αναφερθεί στην Ενότητα 1.2. Επομένως γνωρίζοντας μέσω του Pin πότε πραγματοποιείται πρόσβαση σε κάποιο block της μνήμης αλλά και ποια είναι η διεύθυνσή του, είναι εφικτό να αναπαρασταθεί η λειτουργία όλων των επιπέδων της cache. Επομένως δημιουργήθηκε ένα pintool για την προσομοίωση της ιεραρχίας των caches, η οποία ήταν αντίστοιχη με αυτή του πραγματικού μηχανήματος.

Η πρώτη πρόκληση που καλούμαστε να αντιμετωπίσουμε είναι η υλοποίηση της LLC και κατ’ επέκταση των κυκλωμάτων UMON. Στο πραγματικό μηχάνημα οι L1 και L2 caches έχουν απλή ομοιόμορφη δομή με το σύνολο των sets τους να είναι δύναμη του 2 έτσι ώστε η εύρεση του αριθμού του set της cache στο οποίο ανήκει κάθε block να ισούται με τον

αριθμό που προκύπτει εφόσον απομονωθούν συγκεκριμένα bits από τη διεύθυνσή του. Από την άλλη πλευρά, η LLC δεν δομείται με αυτόν τον τρόπο (Ενότητα 1.2), αλλά χωρίζεται σε slices. Ο συνολικός αριθμός των sets της LLC δεν είναι δύναμη του 2 αφού ισούται με $\frac{LLC\_size/ways\_of\_associativity}{block\_size} = \frac{25Mbytes/20}{64Bytes} = 20480$ και έτσι δεν προκύπτει άμεσα από τη διεύθυνση του block ο αριθμός του set στο οποίο ανήκει. Η εύρεση του slice και του set στο οποίο θα αντιστοιχηθεί κάθε block προκύπτει από έναν αλγόριθμο κατακερματισμού. Ωστόσο ο αλγόριθμος αυτός δεν έχει δημοσιευτεί, με αποτέλεσμα να μη μπορεί να γίνει ακριβής αναπαράσταση της LLC στην προσομοίωση. Επομένως έγινε προσεγγιστικά η προσομοίωση της LLC μειώνοντας το μέγεθός της από 25MB σε 20MB και διατηρώντας 20 way associativity. Με αυτόν τον τρόπο ο συνολικός αριθμός των sets γίνεται $\frac{LLC\_newSize/ways\_of\_associativity}{block\_size} = \frac{20Mbytes/20}{64Bytes} = 16384 = 2^{14}$, που όπως φαίνεται είναι δύναμη του 2 και έτσι ο αριθμός του set στο οποίο ανήκει ένα block προκύπτει άμεσα από τα 14 αντίστοιχα bits της διεύθυνσής του. Τέλος η ιεραρχία των caches που προσομοιώθηκε ήταν inclusive, όπως και στο πραγματικό μηχάνημα, δηλαδή κάθε επίπεδο στην ιεραρχία αποτελεί υπερσύνολο των δεδομένων των προηγούμενων επιπέδων.

Μία ακόμη προσέγγιση που καλούμαστε να πραγματοποιήσουμε είναι η πολιτική αντικατάστασης (replacement policy) που χρησιμοποιεί η cache, αφού η Intel δεν έχει αποκαλύψει την πολιτική αντικατάστασης που έχει υλοποιήσει στους τελευταίους επεξεργαστές της. Για τον λόγο αυτό, στην προσομοίωση υλοποιήθηκε η ευρέως χρησιμοποιούμενη πολιτική αντικατάστασης LRU σε όλα τα επίπεδα της cache. Στον Πίνακα 4.5 συγκεντρώνονται τα χαρακτηριστικά όλων των επιπέδων της cache που προσομοιώθηκε μέσω του Pin.

L1 cache	32KB, 8-way associative, 64=2^6sets, LRU replacement policy
L2 cache	256KB, 8-way associative, 512=2^9sets, LRU replacement policy
LLC-UMON	20MB, 20-way associative, 16384=2^{14}sets, LRU replacement policy

Πίνακας 4.5: Η ιεραρχία της cache κατά την προσομοίωση των UMON

Το pintool πραγματοποιεί ανάλυση για μία εφαρμογή κάθε φορά, επομένως η LLC ταυτίζεται με το κύκλωμα UMON εφόσον προστεθεί και ένας μετρητής των hits για κάθε way της LLC. Επιπλέον προσθέτουμε και έναν μετρητή των misses που πραγματοποιούνται στην LLC και η εκτέλεση της εκάστοτε εφαρμογής χωρίζεται σε διαστήματα των 500 εκατομμυρίων εντολών για την καταγραφή των μετρητών. Τελικά μετά την ανάλυση κάθε benchmark με το pintool που περιγράφηκε, παρέχεται ένα προφίλ το οποίο περιέχει ανά διάστημα εκτέλεσης της εφαρμογής το σύνολο των hits ανά way και το σύνολο των misses που πραγματοποίησε. Επομένως από το προφίλ αυτό μπορούν να υπολογιστούν τα επιπλέον misses που θα πραγματοποιήσει η εφαρμογή με την ελάττωση της cache που της αποδίδεται ανά way, πληροφορία που είναι χρήσιμη για τη λήψη αποφάσεων σε αλγορίθμους διαμοιρασμού της κοινόχρηστης μνήμης.

4.5 Συνεκτελέσεις εφαρμογών

Μετά την ανάλυση της διαφορετικής συμπεριφοράς των εφαρμογών σε σχέση με το ποσοστό της LLC που έχουν στη διάθεσή τους, πραγματοποιήθηκαν πειράματα με συνεκτελέσεις εφαρμογών στο πραγματικό μηχάνημα. Για τη διεξαγωγή των πειραμάτων αυτών, όπου ένας ορισμένος αριθμός εφαρμογών εκτελείται ταυτόχρονα στο σύστημα, ήταν απαραίτητη η επιλογή διαφορετικών συνόλων εφαρμογών (workloads) αποτελούμενα από k=8

benchmarks το καθένα. Έχοντας n=35 διαφορετικά benchmarks οι συνολικοί συνδυασμοί προκύπτουν από τον διωνυμικό συντελεστή ως εξής:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{35!}{8!(35-8)!} = 23535820$$

Οι συνδυασμοί αυτοί είναι ακόμη περισσότεροι εάν επιτρέψουμε ένα benchmark να μπορεί να εμφανιστεί περισσότερες από μία φορές στο ίδιο workload. Όπως είναι κατανοητό είναι πρακτικά αδύνατον να εκτελεστούν τόσα workloads σε κάθε πείραμα, όπου κάθε workload έχει κατά μέσο όρο χρόνο εκτέλεσης 15 λεπτά. Επομένως επιλέχθηκαν 100 διαφορετικά workloads ως εξής:

(a) Έχοντας βρει τον ελάχιστο αριθμό από ways, $w_{95\%}\_i$, που χρειάζεται το benchmark i για να πετύχει το 95% της μέγιστης απόδοσής του (Πίνακας 4.3), βρίσκουμε όλα τα workloads τα οποία θεωρητικά χρειάζονται αθροιστικά περισσότερη μνήμη από το τετραπλάσιο μέγεθος της υπάρχουσας 20-way LLC, δηλαδή:

$$\sum_{i=1}^{8} w_{95\%}\_i > 4 * LLC\_ways = 80ways$$

Με αυτόν τον τρόπο έχουμε στη διάθεσή μας 1812326 workloads που χρειάζονται από 81 έως 140 ways της υπάρχουσας LLC και αναμένουμε με την εκτέλεσή τους να υπάρχει σημαντική επιβράδυνση των benchmarks. Στη συνέχεια ομαδοποιούμε τα workloads με βάση το θεωρητικό άθροισμα των απαιτούμενων ways σε 6 ομάδες, δηλαδή σε 81-90, 91-100,...,131-140 ways. Από την κάθε ομάδα επιλέγουμε τυχαία 8 workloads, καταλήγοντας με 48 διαφορετικά workloads με μεγαλύτερες θεωρητικά απαιτήσεις μνήμης από το συνολικό αριθμό των διαθέσιμων ways. Τα 48 αυτά workloads περιέχουν με μεγάλη συχνότητα τα benchmarks libquantum και omnetpp καθώς αυτά χρειάζονται την περισσότερη μνήμη (18 ways) για να επιτευχθεί το 95% της μέγιστης απόδοσής τους.

(b) Επιπλέον επιλέχθηκαν τυχαία από το σύνολο των 35 benchmarks άλλα 52 διαφορετικά workloads των 8 benchmarks, τα οποία είναι διαφορετικά από αυτά που επιλέχθηκαν προηγουμένως και δεν είναι απαραίτητο να χρειάζονται μεγαλύτερο μέγεθος LLC από το διαθέσιμο. Τέλος, και τα 100 workloads παρουσιάζονται στο Παράρτημα C.

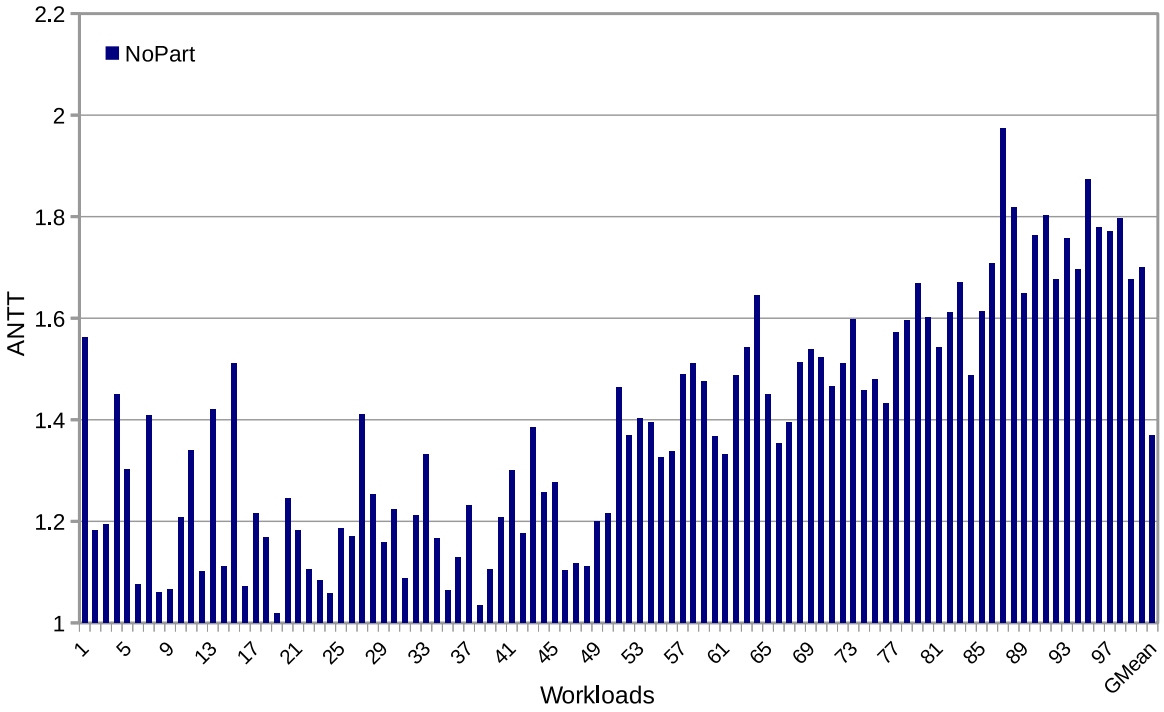
Η εκτέλεση κάθε workload ακολουθεί μία συγκεκριμένη μεθοδολογία. Χρησιμοποιήθηκε η βιβλιοθήκη PQoS (Ενότητα 3.3.3), η οποία κάνει χρήση των τεχνολογιών CMT και CAT της Intel, για τη λήψη των μετρήσεων και την κατανομή της μνήμης αντίστοιχα, και επεκτάθηκε έτσι ώστε η εκκίνηση-επανεκκίνηση των εφαρμογών να γίνεται μέσω αυτής. Τα 8 benchmarks που περιλαμβάνει κάθε workload εκτελούνται ταυτόχρονα στο πραγματικό μηχάνημα και ορίζεται στατικά κάθε ένα από αυτά να εκτελεστεί σε διαφορετικό πυρήνα. Τα benchmarks ξεκινούν την εκτέλεσή τους μαζί και συνεχίζουν τα εκτελούνται έως ότου όλα τα benchmarks πραγματοποιήσουν τουλάχιστον μία ολοκληρωμένη εκτέλεση. Τα benchmarks που τερματίζουν νωρίτερα επανεκκινούνται στον ίδιο πυρήνα που εκτελούνταν αλλά η αξιολόγηση της απόδοσής τους γίνεται μόνο για την πρώτη ολοκληρωμένη εκτέλεσή τους. Το χρονικό διάστημα παρακολούθησης της PQoS, στο τέλος του οποίου λαμβάνονται οι μετρήσεις, ορίζεται στα 100msec.

4.5.1 Ανάδειξη του προβλήματος λόγω διαμάχης για τους κοινόχρηστους πόρους

Σε πρώτο στάδιο εκτελέστηκαν τα 100 workloads στο πραγματικό μηχάνημα χωρίς την επιβολή κάποιου διαχωρισμού στην LLC (*κατάσταση NoPart*), ώστε να αναδειχθεί το πρόβλημα της επιβράδυνσης του συστήματος λόγω της διαμάχης για τους κοινόχρηστους πόρους. Για κάθε benchmark μετριέται το IPC του για την κατάσταση συνεκτελέσεων NoPart και συγκρίνεται με το αντίστοιχο IPC στην κατάσταση Alone που έχει καταγραφεί και παρουσιαστεί στην προηγούμενη ενότητα. Πιο συγκεκριμένα, η μετρική που χρησιμοποιείται για τον υπολογισμό της επιβράδυνσης των benchmarks είναι ο μέσος κανονικοποιημένος χρόνος ολοκλήρωσης (average normalized turnaround time, ANTT) που ορίζεται ως εξής για n=8 εφαρμογές:

$$ANTT = \frac{1}{n}\sum_{i=1}^{n} NTT_i = \frac{1}{n}\sum_{i=1}^{n} \frac{IPC_i^{Alone}}{IPC_i^{NoPart}}$$

Η μετρική NTT_i, η οποία αντιπροσωπεύει τον κανονικοποιημένο χρόνο ολοκλήρωσης, μπορεί να έχει τιμή μεγαλύτερη ή ίση της μονάδας και υποδεικνύει την επιβράδυνση που υπέστη η εφαρμογή i κατά την ταυτόχρονη εκτέλεσή της με τις υπόλοιπες εφαρμογές σε σχέση με την κατάσταση Alone. Έτσι η μετρική ANNT ισούται με το μέσο NTT για n συνεκτελούμενες εφαρμογές και μπορεί να πάρει τιμή μεγαλύτερη ή ίση της μονάδας. Όσο μεγαλύτερη της μονάδας είναι η τιμή του ANNT τόσο μεγαλύτερη επιβράδυνση έχει υποστεί το σύστημα. Στην Εικόνα 4.2 παρουσιάζεται η επιβράδυνση για τα 100 workloads (με τη σειρά που παρουσιάζονται στο Παράρτημα C, τα 52 πρώτα είναι τα τυχαία και τα επόμενα 48 είναι αυτά που θεωρήσαμε πως θα έχουν μεγάλη επιβράδυνση) καθώς και ο γεωμετρικός μέσος τους.



Εικόνα 4.2: Επιβράδυνση του συστήματος από τη συνεκτέλεση 8 benchmarks

Από την Εικόνα 4.2 γίνεται φανερή η μείωση της απόδοσης του συστήματος καθώς τα 8 benchmarks μοιράζονται τους κοινόχρηστους πόρους. Υπάρχουν workloads που μειώνουν κατακόρυφα την απόδοση του μηχανήματος διότι μπορεί τα συνεκτελούμενα benchmarks να

έχουν υψηλές απαιτήσεις μνήμης ή/και να υπάρχουν thrashing benchmarks τα οποία κάνουν κατάχρηση της LLC και περιορίζουν τα υπόλοιπα benchmarks που μπορεί να επωφελούνταν περισσότερο από τη μνήμη αυτή. Από την άλλη πλευρά υπάρχουν κάποια workloads στα οποία δεν παρατηρείται σημαντική μείωση στην απόδοση του συστήματος, πιθανώς γιατί αποτελούνται από benchmarks που δεν επηρεάζονται από το ποσοστό της LLC που έχουν στη διάθεσή τους ή/και τα περισσότερα benchmarks καταλαμβάνουν το μέρος της μνήμης που τους είναι χρήσιμο. Με άλλα λόγια όταν συνεκτελούνται friendly/fitting με friendly/fitting εφαρμογές ή thrashing με thrashing εφαρμογές, η πολιτική αντικατάστασης του συστήματος λειτουργεί επαρκώς καλά, ενώ όταν συνεκτελούνται friendly ή fitting με thrashing εφαρμογές πλήττεται η απόδοση του συστήματος. Επομένως κρίνεται απαραίτητη η χρήση μίας πολιτικής διαχωρισμού της κοινόχρηστης μνήμης στις συνεκτελούμενες εφαρμογές με στόχο τη βελτίωση της συνολικής απόδοσης του συστήματος.

4.5.2 Μελέτη της τεχνικής διαμοιρασμού της κοινόχρηστης cache UCP

Στο στάδιο αυτό υλοποιήθηκε και εξετάστηκε η τεχνική διαμοιρασμού της κοινόχρηστης cache UCP, που αναλύθηκε στην Ενότητα 2.2.1. Για το σκοπό αυτό επεκτάθηκε περαιτέρω η βιβλιοθήκη PQoS της Intel, ώστε μετά το τέλος κάθε διαστήματος παρακολούθησης, όπου λαμβάνονται οι κατάλληλες μετρήσεις από τη βιβλιοθήκη, να εκτελείται ο αλγόριθμος lookahead της UCP και να εφαρμόζεται ο εκάστοτε διαχωρισμός της LLC. Ο αλγόριθμος αυτός, ο ψευδοκώδικας του οποίου παρουσιάζεται στο Παράρτημα A, χρειάζεται την πληροφορία που παρέχουν τα κυκλώματα UMON. Επειδή τα UMON δεν είναι διαθέσιμα στο hardware, παρέχουμε εμείς στην PQoS την πληροφορία που θα έπαιρνε από αυτά και την οποία έχουμε εξάγει με τον τρόπο που περιγράφηκε στην Ενότητα 4.4.

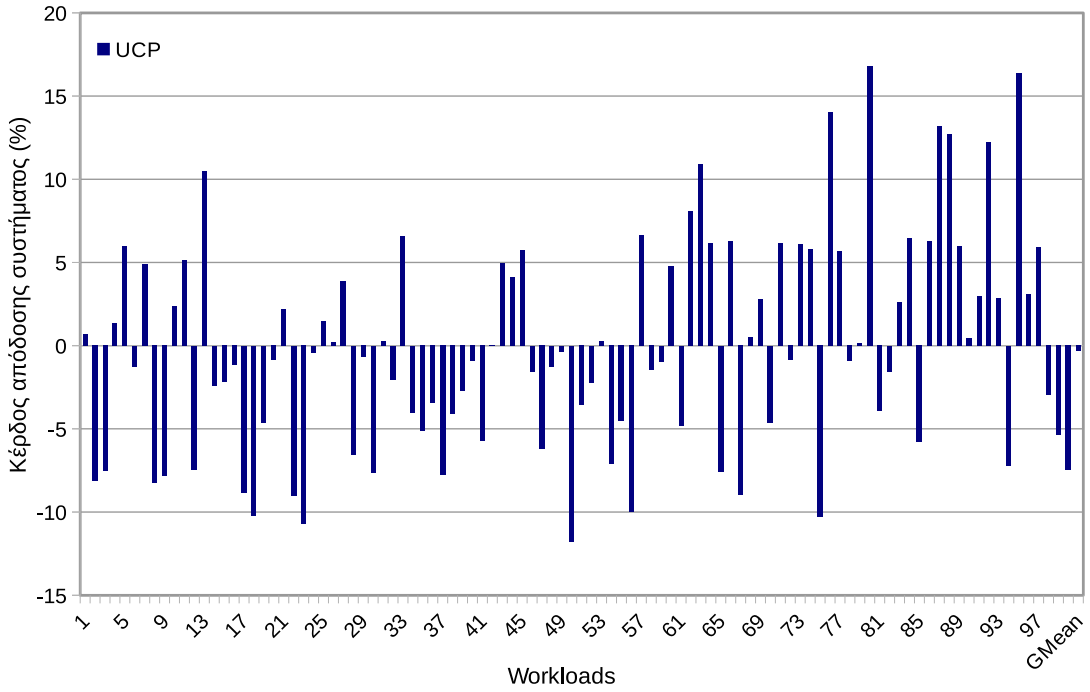
Τα προφίλ των benchmarks που έχουν εξαχθεί και αντιπροσωπεύουν την πληροφορία που θα δινόταν από τα κυκλώματα UMON, δηλαδή το σύνολο των hits ανά way της LLC που θα πραγματοποιούσε το benchmark αν εκτελούταν μόνο του και είχε πρόσβαση σε ολόκληρη την LLC, είναι χωρισμένα σε διαστήματα των 500 εκατομμυρίων εντολών. Επομένως κατά τις συνεκτελέσεις αντιστοιχίζουμε το σύνολο των εντολών που εκτέλεσε κάθε benchmark κατά το διάστημα παρακολούθησης της PQoS (100ms) με τα διαστήματα εντολών του προφίλ του. Με αυτόν τον τρόπο βρίσκουμε τα hits ανά way που θα πραγματοποιούσε κάθε benchmark αν εκτελούταν μόνο του, τα οποία και χρειάζονται για τη λήψη των αποφάσεων για τον διαχωρισμό της LLC.

Οι αποφάσεις που λαμβάνονται μέσω του αλγορίθμου lookahead για τον διαχωρισμό της LLC, δηλαδή ο αριθμός των ways που θα αποδοθεί ιδιωτικά σε κάθε benchmark, εφαρμόζονται πρακτικά στο hardware μέσω της τεχνολογίας CAT της Intel. Συγκεκριμένα κάθε πυρήνας-benchmark αντιστοιχίζεται με ένα από τα 16 CLOS του συστήματος και όσα ways θέλουμε να αποδοθούν στον πυρήνα τόσα bits της μάσκας χωρητικότητας του CLOS του θα πάρουν την τιμή 1, προσέχοντας να μην υπάρχει επικάλυψη στις μάσκες των CLOS των πυρήνων για να είναι ιδιωτικά τα ways.

Με τον παραπάνω τρόπο εκτελέστηκαν τα 100 workloads στο πραγματικό μηχάνημα ώστε να παρατηρήσουμε εάν η τεχνική UCP βελτιώνει την απόδοση του συστήματος όταν εκτελούνται ταυτόχρονα n=8 εφαρμογές σε σχέση με την κατάσταση NoPart. Για τον υπολογισμό της απόδοσης του συστήματος χρησιμοποιήθηκε η μετρική weighted speedup, η οποία ορίζεται ως εξής:

$$weightedSpeedup = \frac{1}{n} \sum_{i=1}^{n} \frac{IPC_i^{UCP}}{IPC_i^{NoPart}}$$

Στην Εικόνα 4.3 παρουσιάζεται το επί τοις εκατό κέρδος στην απόδοση του συστήματος σε σχέση με την κατάσταση NoPart.



Εικόνα 4.3: Κέρδος στην απόδοση του συστήματος με την εφαρμογή της UCP σε σχέση με την κατάσταση NoPart

Παρατηρώντας την Εικόνα 4.3 γίνεται κατανοητό πως ο διαμοιρασμός της κοινόχρηστης cache σύμφωνα με την τεχνική UCP όχι μόνο δε βελτιώνει αλλά οριακά επιδεινώνει (επιβράδυνση κατά 0.32%) την απόδοση του πραγματικού, σύγχρονου συστήματος. Ενώ ο βασικός στόχος της τεχνικής UCP είναι να βελτιώσει την απόδοση του συστήματος, στην πράξη με τη συνεκτέλεση πολλών εφαρμογών είναι μη αποτελεσματική. Το γεγονός αυτό μπορεί να εξηγηθεί από τη μείωση της ευελιξίας των εφαρμογών λόγω του απόλυτου διαχωρισμού της κοινόχρηστης cache σε επίπεδο way που επισκιάζει τα πλεονεκτήματα της UCP. Με άλλα λόγια η τεχνική UCP αποδίδει σε κάθε εφαρμογή τουλάχιστον 1 way από τα 20 σε κάθε μία από τις 8 εφαρμογές, με αποτέλεσμα να μένουν 12 ways για να μοιραστούν στις εφαρμογές που θα ωφεληθούν περισσότερο. Ωστόσο κάποιες εφαρμογές μπορεί να μη χρειάζονται ολόκληρο αυτό το ένα way και επομένως ορισμένο μέρος της cache να μένει αχρησιμοποίητο, ή κάποιες εφαρμογές να χρειάζονται ενάμιση way κάτι που δεν είναι εφικτό με την UCP. Οι Qureshi και Patt [4] είχαν αξιολογήσει την τεχνική UCP σε συνεκτελέσεις μόνο 2 ή 4 εφαρμογών και σε επίπεδο προσομοίωσης όπου δε λάμβαναν υπ’ όψιν τους πολλές παραμέτρους των πραγματικών μηχανημάτων, όπως οι prefetchers και ο ελεγκτής της κύριας μνήμης. Επιπλέον θεωρούσαν πως η πολική αντικατάστασης των caches ήταν η LRU, γεγονός που δεν ισχύει πλέον στους σύγχρονους επεξεργαστές. Τελικά ο διαχωρισμός της LLC σε επίπεδο way ιδιωτικά σε κάθε μία από τις συνεκτελούμενες εφαρμογές έχει γίνει μη ρεαλιστικός στα σύγχρονα μηχανήματα αφού δεν μπορεί να κλιμακωθεί με την αύξηση του αριθμού των εφαρμογών. Επομένως προκύπτει η ανάγκη για τη δημιουργία ενός μηχανισμού που θα επιτρέπει ορισμένη επικάλυψη στα ways που αποδίδονται στις εφαρμογές και θα μπορεί να κλιμακωθεί με την αύξηση του αριθμού των εφαρμογών.

Κεφάλαιο 5

Νέος μηχανισμός διαμοιρασμού της κοινόχρηστης cache, ACUTE

Στο κεφάλαιο αυτό παρουσιάζεται ένας νέος μηχανισμός, ο ACUTE (Adaptive Clustering for UTility-based cachE partitioning), που πραγματοποιεί δυναμική ομαδοποίηση εφαρμογών για τον διαχωρισμό της cache με βάση τη χρησιμότητά της (utility). Σκοπός του μηχανισμού αυτού είναι η βελτίωση της συνολικής απόδοσης του πραγματικού συστήματος. Όπως παρουσιάστηκε στην Ενότητα 4.5 η διαμάχη για τους κοινόχρηστους πόρους και ιδιαίτερα την LLC, προκαλεί επιβράδυνση στην εκτέλεση των εφαρμογών που εκτελούνται ταυτόχρονα στο ίδιο σύστημα. Πολλοί ερευνητές έχουν προτείνει τεχνικές διαμοιρασμού της LLC για την αντιμετώπιση αυτού του προβλήματος (Κεφάλαιο 2), όμως λίγοι από αυτούς είχαν στη διάθεσή τους τον κατάλληλο hardware μηχανισμό για την πρακτική επιβολή του διαχωρισμού στη μνήμη. Επομένως, έχοντας πλέον διαθέσιμους τους κατάλληλους μηχανισμούς CMT-CAT από την Intel (Κεφάλαιο 3) για την παρακολούθηση και την κατανομή της LLC, υλοποιήθηκε ένας μηχανισμός για τη δυναμική κατανομή της στις συνεκτελούμενες εφαρμογές.

Στις επόμενες ενότητες παρουσιάζεται ο τρόπος λειτουργίας του προτεινόμενου μηχανισμού, γίνεται η αξιολόγησή του με την εφαρμογή του στο πραγματικό σύστημα, εξετάζεται η απόδοσή του με την μεταβολή ορισμένων παραμέτρων και τέλος υπολογίζεται το κόστος υλοποίησής του σε ένα σύγχρονο πραγματικό σύστημα.

5.1 Περιγραφή μηχανισμού

Με τον μηχανισμό ACUTE στοχεύουμε στη βελτίωση της συνολικής απόδοσης του συστήματος λαμβάνοντας υπ’ όψιν τη διαφορετική ανάγκη των συνεκτελούμενων εφαρμογών για μνήμη. Η κεντρική ιδέα του μηχανισμού βασίζεται στον διαχωρισμό των εφαρμογών σε δύο ομάδες με βάση το πόσο ωφελούνται από την αύξηση του τμήματος της LLC που τους αποδίδεται. Ο διαχωρισμός αυτός γίνεται δυναμικά, κατά το χρόνο εκτέλεσης των εφαρμογών, ώστε να αναγνωρίζονται οι διαφορετικές φάσεις των εφαρμογών, κατά τις οποίες είναι πιθανόν να παρουσιάζουν διαφορετική συμπεριφορά.

Έχουν προταθεί πολλοί μηχανισμοί για την εύρεση των χαρακτηριστικών συμπεριφοράς των εφαρμογών ανάλογα με το ποσοστό της LLC που έχουν στη διάθεσή τους, όμως ο hardware μηχανισμός που έχει προταθεί από τους Qureshi και Patt [4], δηλαδή τα κυκλώματα παρακολούθησης της χρησιμότητας της cache UMON (Ενότητα 2.2.1), υπερτερεί σε σχέση με τους υπολοίπους, διότι αν και προσθέτει ένα μικρό κόστος κατασκευής δεν προσθέτει πολυπλοκότητα και επιβάρυνση στο σύστημα ενώ παρέχει ακριβή πληροφορία. Επομένως αυτός ο τρόπος προτιμήθηκε για την τροφοδότηση του μηχανισμού ACUTE. Ωστόσο τα κυκλώματα UMON δεν έχουν υλοποιηθεί στο hardware και τα προφίλ των benchmarks που θα δίνονταν δυναμικά κατά τη συνεκτέλεση των εφαρμογών λήφθηκαν μέσω προσομοίωσης με

τον τρόπο που αναλύθηκε στην Ενότητα 4.4. Τελικά, τα προφίλ αυτά παρέχουν τα hits ανά way (επομένως και τον συνολικό αριθμό των hits) και τον συνολικό αριθμό των misses ανά διάστημα εκτέλεσης (500 εκατομμυρίων εντολών) κάθε εφαρμογής. Με άλλα λόγια, παρέχουν τις καμπύλες αστοχιών (miss curves) των εφαρμογών ανά διάστημα εκτέλεσης.

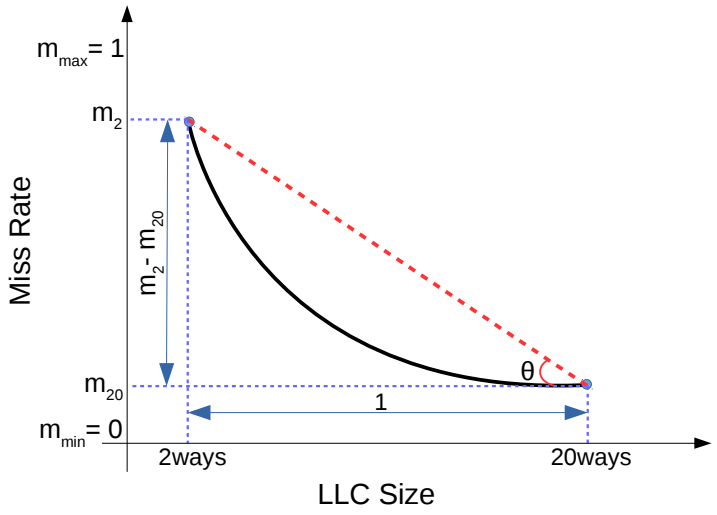
Επομένως κατά τη διάρκεια εκτέλεσης των εφαρμογών, ο μηχανισμός ACUTE διαχωρίζει αρχικά τις εφαρμογές σε δύο ομάδες σύμφωνα με τις miss curves τους από τα κυκλώματα UMON. Έπειτα συνδυάζονται με κατάλληλο τρόπο οι miss curves των εφαρμογών κάθε ομάδας και προκύπτουν οι συνδυασμένες καμπύλες αστοχιών (combined miss curves) για κάθε ομάδα εφαρμογών. Τέλος αυτές οι καμπύλες χρησιμοποιούνται για τον διαχωρισμό της κοινόχρηστης μνήμης σε δύο ιδιωτικά μέρη, ένα για κάθε μία ομάδα εφαρμογών. Τα βήματα αυτά παρουσιάζονται αναλυτικά στις δύο επόμενες ενότητες.

5.1.1 Δυναμική κατηγοριοποίηση εφαρμογών

Στο κεφάλαιο 4.3 αναλύθηκαν σε θεωρητικό επίπεδο τρία είδη εφαρμογών τα οποία προκύπτουν με βάση το πόσο ωφέλιμα είναι για κάθε εφαρμογή διαφορετικά μεγέθη της LLC. Οι εφαρμογές χωρίστηκαν σε cache friendly, όπου η απόδοση των εφαρμογών αυξάνεται σημαντικά με την αύξηση του μεγέθους της LLC, σε cache fitting, όπου οι εφαρμογές αυτές χρειάζονται μόνο μερικά ways της LLC ώστε να αποκτήσουν τη μέγιστη απόδοσή τους και τέλος σε cache thrashing, όπου η απόδοση των εφαρμογών δεν επηρεάζεται από το ποσοστό της LLC που έχουν στη διάθεσή τους και πραγματοποιούν μεγάλο αριθμό misses. Όπως παρατηρήθηκε στην Ενότητα 4.5.1, όταν εκτελούνται ταυτόχρονα στο ίδιο σύστημα friendly/fitting με friendly/fitting εφαρμογές ή thrashing με thrashing εφαρμογές η πολιτική αντικατάστασης του συστήματος λειτουργεί επαρκώς καλά, ενώ όταν εκτελούνται friendly ή fitting με thrashing εφαρμογές προκαλείται σημαντική μείωση στην απόδοση του συστήματος. Ο μηχανισμός ACUTE πραγματοποιεί δυναμικό διαχωρισμό των εφαρμογών σε δύο ομάδες. Η πρώτη ομάδα περιλαμβάνει τις cache friendly και fitting εφαρμογές ενώ η δεύτερη τις cache thrashing εφαρμογές. Στη υπόλοιπη εργασία όταν αναφέρεται η ομάδα των friendly εφαρμογών εννοούμε το σύνολο των friendly/fitting εφαρμογών. Ο διαχωρισμός των εφαρμογών στις δύο παραπάνω ομάδες αποδίδει ευελιξία στις εφαρμογές της κάθε ομάδας, ενώ μπορεί να εφαρμοστεί ανεξάρτητα από το συνολικό αριθμό των συνεκτελούμενων εφαρμογών.

Ο διαχωρισμός των εφαρμογών στις δύο ομάδες γίνεται δυναμικά, δηλαδή κατά το χρόνο εκτέλεσης των εφαρμογών ώστε να λαμβάνονται υπ' όψιν οι διαφορετικές φάσεις εκτέλεσης και επομένως οι διαφορετικές συμπεριφορές των εφαρμογών. Για τον χαρακτηρισμό μίας εφαρμογής ως friendly ή thrashing χρησιμοποιούνται οι miss rate curves που λαμβάνονται από τα κυκλώματα UMON ως εξής:

- Συγκεκριμένα υπολογίζουμε το ημίτονο της γωνίας θ που σχηματίζεται μεταξύ του οριζόντιου άξονα και της ευθείας που σχηματίζεται από το σημείο για ways=2 και το σημείο για ways=20 της miss rate curve, όπως φαίνεται στην Εικόνα 5.1. Η ευθεία αυτή δεν ξεκινά από το σημείο για ways=1 διότι θεωρούμε πως κάθε εφαρμογή θα λάβει τουλάχιστον 1 way. Η γωνία θ μπορεί να πάρει τιμές στο πεδίο [0, 45°], αφού οι τιμές του miss rate είναι μεταξύ του 0 και του 1, και επομένως το ημίτονο της γωνίας θ μπορεί να πάρει τιμές στο πεδίο $[0, \sqrt{2}/2] \simeq [0, 0.7]$. Επομένως ελέγχεται το ημίτονο της γωνίας θ και αν υπερβαίνει ένα κατώφλι (threshold), t_{thr}, θεωρείται friendly διαφορετικά thrashing. Το κατώφλι αυτό ορίζεται $t_{thr} = 0.1$, δηλαδή θεωρούμε αρκετά μικρή τη γωνία θ των thrashing εφαρμογών αφού έχουν σχεδόν σταθερό miss rate, και η τιμή του εξετάζεται στην Ενότητα 5.3.

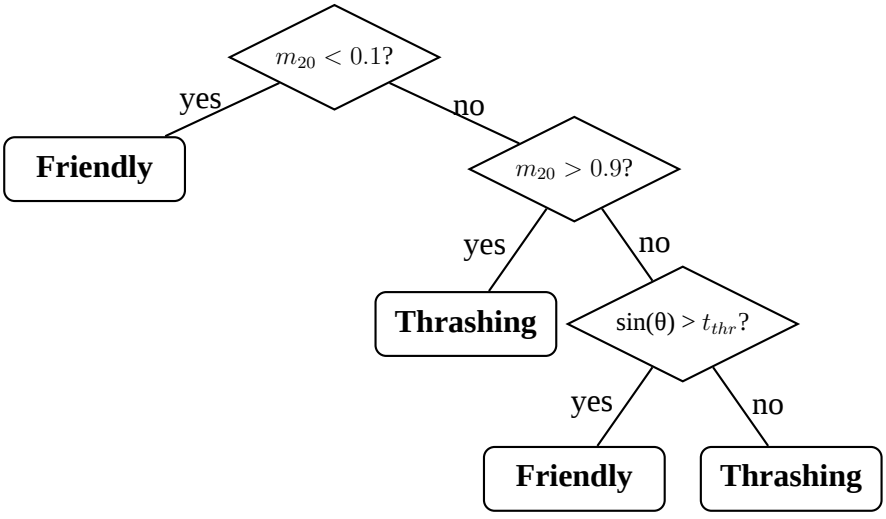


Εικόνα 5.1: Γωνία θ για την κατηγοριοποίηση μιας εφαρμογής

- Επιπλέον ελέγχονται ορισμένες οριακές καταστάσεις. Συγκεκριμένα ελέγχουμε το m_{20} (miss rate στα 20 ways) και αν $m_{20} < 0.1$ τότε η εφαρμογή θεωρείται friendly, αφού θα πραγματοποιούσε σχεδόν μηδενικά misses αν είχε στη διάθεσή της 20 ways. Αυτό σημαίνει είτε πως και με λιγότερα ways θα έχει την ίδια καλή συμπεριφορά είτε πως με την αύξηση του μέρους της LLC που της αποδίδεται μειώνει το miss rate της έως ότου φτάσει σε πολύ χαμηλή τιμή, άρα σε κάθε περίπτωση ανήκει στην ομάδα των cache friendly εφαρμογών.

- Διαφορετικά αν $m_{20} > 0.9$ τότε η εφαρμογή θεωρείται thrashing, αφού αν έχει τόσο υψηλό miss rate στα 20 ways, θα έχει το ίδιο ή μεγαλύτερο αν της αποδοθεί μειωμένο μέρος της cache.

Στο δέντρο αποφάσεων της Εικόνας 5.2 φαίνονται συγκεντρωτικά τα βήματα για την προσθήκη μίας εφαρμογής στην ομάδα των cache friendly ή cache thrashing εφαρμογών.



Εικόνα 5.2: Δέντρο αποφάσεων για την κατηγοριοποίηση μιας εφαρμογής

5.1.2 Συνδυασμός των miss curves ανά ομάδα εφαρμογών και πολιτική διαχωρισμού της cache

Έχοντας χωρίσει τις εφαρμογές δυναμικά σε δύο ομάδες, τις cache friendly και τις cache thrashing, το επόμενο στάδιο του μηχανισμού ACUTE αφορά το χωρισμό της LLC σε δύο μέρη, όπου καθένα από αυτά θα κατανεμηθεί στις ομάδες εφαρμογών. Προκειμένου να ληφθεί η απόφαση για τον διαχωρισμό της LLC απαιτείται για κάθε ομάδα η πληροφορία για τη χρησιμότητα της cache, η οποία προκύπτει από τη συγκεντρωτική miss curve κάθε ομάδας.

Για τον υπολογισμό της συγκεντρωτικής miss curve, δε γίνεται απλά να προστεθούν τα misses ανά way που θα πραγματοποιούσε κάθε εφαρμογή αν εκτελούταν μόνη της στο σύστημα, πληροφορία που λαμβάνεται από τα κυκλώματα UMON. Αυτό συμβαίνει διότι πλέον κάθε εφαρμογή ανήκει σε μία ομάδα εφαρμογών, στην οποία θα αποδοθεί συγκεκριμένο μέρος της LLC και οι εφαρμογές που ανήκουν σε αυτή θα το διεκδικήσουν. Επομένως τα misses που πραγματοποιεί μία εφαρμογή όταν εκτελείται μόνη της και όταν εκτελείται ταυτόχρονα με άλλες εφαρμογές διαφέρουν και συγκεκριμένα αυξάνονται στη δεύτερη περίπτωση. Για τον λόγο αυτό προκύπτει η ανάγκη για το συνδυασμό των miss curves κάθε εφαρμογής από τα κυκλώματα UMON για τη δημιουργία μίας συνδυασμένης miss curve (combined miss curve) για κάθε ομάδα εφαρμογών που θα ανταποκρίνεται περισσότερο στην πραγματικότητα.

Για τον υπολογισμό της combined miss curve κάθε ομάδας εφαρμογών, χρησιμοποιήθηκε ο αλγόριθμος που προτάθηκε από τους Mukkara κ.ά. [30] στο μηχανισμό τους Whirlpool και επεκτάθηκε για την προσαρμογή του στον μηχανισμό ACUTE. Για τη δημιουργία μιας combined miss curve χρησιμοποιείται η ”ροή” των blocks στην cache, θεωρώντας πως η πολιτική αντικατάστασης της cache είναι η LRU. Η πολιτική LRU εισάγει ένα νέο block στη θέση MRU και το μετακινεί προς τη θέση LRU καθώς εισέρχονται άλλα νέα blocks στην cache, έως ότου το απομακρύνει από αυτήν. Επομένως, η ροή ορίζεται ως ο ρυθμός μετακίνησης των blocks προς τη θέση LRU. Ωστόσο τα blocks μπορεί να προβιβαστούν ξανά στη θέση MRU, με αποτέλεσμα τη μείωση της ροής λόγω πραγματοποίησης hits. Τελικά η ροή σε ένα συγκεκριμένο σημείο - μέγεθος της LLC ισούται με το miss rate σε αυτό το σημείο. Όταν δύο ή περισσότερες εφαρμογές συνεκτελούνται, κάθε εφαρμογή μετακινεί blocks όλων των εφαρμογών προς τη θέση LRU και επομένως υπερισχύουν οι εφαρμογές με τη μεγαλύτερη ροή. Συνεπώς για τον υπολογισμό της combined miss rate curve προστίθενται τα miss rates των εφαρμογών, προχωρώντας όμως ανά βήματα τις ατομικές miss rate curves τους ανάλογα με το ύψος των τιμών τους. Για την πλήρη κατανόηση του αλγορίθμου αυτού παρουσιάζεται παρακάτω ο ψευδοκώδικάς του (Αλγόριθμος 1). Στον ψευδοκώδικα θεωρούμε για απλότητα n=3 εφαρμογές με m1, m2 και m3 οι αντίστοιχες ατομικές miss rate curves τους, m η ζητούμενη combined miss rate curve και N ο αριθμός των ways της cache.

Algorithm 1 Ψευδοκώδικας για τον υπολογισμό της combined miss rate curve

```
1: s1 = 0, s2 = 0, s3 = 0
2: for s = 0 to N do:
3:     m[s] = m1[s1] + m2[s2] + m3[s3]
4:     if (m1[s1] / m[s] >= 1/n): s1 += 1
5:     if (m2[s2] / m[s] >= 1/n): s2 += 1
6:     if (m3[s3] / m[s] >= 1/n): s3 += 1
7: return m
```

Με την εφαρμογή του Αλγορίθμου 1 σε κάθε μία από τις δύο ομάδες εφαρμογών, προκύπτουν οι αντίστοιχες combined miss rate curves τους χρησιμοποιώντας τη μετρική MPKI.

Τελικά, γίνεται αναζήτηση του βέλτιστου διαχωρισμού της LLC στις δύο αυτές ομάδες στοχεύοντας στην ελαχιστοποίηση των συνολικών misses. Συγκεκριμένα γίνεται εξαντλητική αναζήτηση όλων των πιθανών διαχωρισμών της LLC, οι οποίοι είναι N-1, αφού έχουμε 2 ομάδες εφαρμογών και N ways της cache. Επομένως για N = 20 ways υπολογίζονται τα συνολικά misses M_{tot} για κάθε συνδυασμό, όπως φαίνεται παρακάτω, όπου m_{fr} και m_{thr} οι combined miss curves για τις ομάδες των cache friendly και thrashing αντίστοιχα.

$$M_{tot} = m_{fr}[i] + m_{thr}[20 - i], i = 1 \text{ μέχρι } (20 - 1)$$

Τελικά επιλέγεται ο συνδυασμός με την ελάχιστη τιμή M_{tot}. Επιπλέον σε κάθε ομάδα εφαρμογών αποδίδεται τουλάχιστον 1 way και τουλάχιστον 2 ways αν σε κάποια ομάδα ανήκουν περισσότερες από μία εφαρμογές. Στην περίπτωση που όλες οι εφαρμογές ανήκουν σε μία ομάδα μόνο, τότε ολόκληρη η cache αποδίδεται σε όλες τις εφαρμογές χωρίς την επιβολή κάποιου διαχωρισμού.

5.2 Αξιολόγηση του μηχανισμού ACUTE

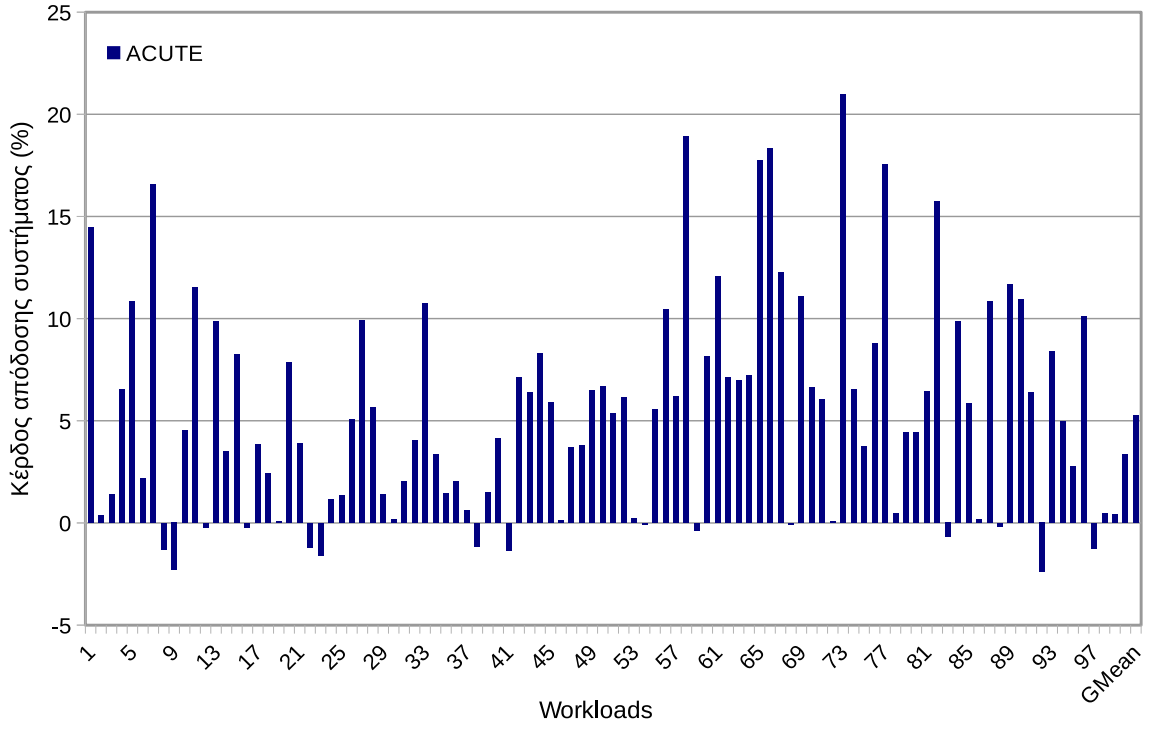
Ο μηχανισμός ACUTE, όπως αυτός παρουσιάστηκε στις προηγούμενες ενότητες, υλοποιήθηκε και ενσωματώθηκε στη βιβλιοθήκη PQoS (Ενότητα 3.3.3), η οποία κάνει χρήση των τεχνολογιών CMT και CAT της Intel, για τη λήψη μετρήσεων και την κατανομή της LLC αντίστοιχα. Στη βιβλιοθήκη αυτή παρέχουμε την πληροφορία που θα έπαιρνε δυναμικά από τα κυκλώματα UMON και η αντιστοίχιση των εντολών που εκτέλεσε κάθε benchmark κατά το διάστημα παρακολούθησης της PQoS (100ms) με τα διαστήματα εντολών του offline προφίλ του γίνεται με τον τρόπο που περιγράφηκε στην Ενότητα 4.5.2. Επιπλέον οι αποφάσεις που λαμβάνονται μέσω του μηχανισμού ACUTE για τον διαχωρισμό της cache στις δύο ομάδες εφαρμογών, εφαρμόζονται μέσω του hardware μηχανισμού CAT της Intel. Συγκεκριμένα κάθε benchmark-πυρήνας αντιστοιχίζεται με ένα CLOS και αποδίδεται η ίδια μάσκα χωρητικότητας στα CLOS των benchmarks που ανήκουν στην ίδια ομάδα εφαρμογών.

Για την αξιολόγηση του μηχανισμού ACUTE πραγματοποιήθηκε ξεχωριστή εκτέλεση των 100 workloads, σε κάθε ένα από τα οποία εκτελούνται ταυτόχρονα n=8 benchmarks στο πραγματικό μηχάνημα, εφαρμόζοντας τη συγκεκριμένη πολιτική διαχωρισμού της LLC. Στη συνέχεια συγκρίνεται η απόδοση του συστήματος με την εφαρμογή του μηχανισμού ACUTE σε σχέση με την κατάσταση NoPart. Για τον υπολογισμό της απόδοσης του συστήματος χρησιμοποιήθηκε η μετρική weighted speedup, η οποία ορίζεται ως εξής:

$$weightedSpeedup = \frac{1}{n} \sum_{i=1}^{n} \frac{IPC_i^{ACUTE}}{IPC_i^{NoPart}}$$

Στην Εικόνα 5.3 παρουσιάζεται το επί τοις εκατό κέρδος στην απόδοση του συστήματος σε σχέση με την κατάσταση NoPart.

Παρατηρώντας την Εικόνα 5.3 γίνεται κατανοητό πως η απόδοση του συστήματος βελτιώνεται με την επιβολή του μηχανισμού ACUTE. Συγκεκριμένα επιτυγχάνεται αύξηση της απόδοσης του πραγματικού μηχανήματος κατά μέσο όρο 5.26% ενώ στην καλύτερη περίπτωση κατά 21%. Επιπλέον συγκρίνοντας το μηχανισμό ACUTE με το μηχανισμό UCP που αξιολογήθηκε στην Ενότητα 4.5.2 και μειώνει την απόδοση του συστήματος κατά μέσο όρο 0.34%, προκύπτει πως ο ACUTE καταφέρνει να πετύχει μία σημαντική βελτίωση του συστήματος. Ακόμη, στην καλύτερη περίπτωση ο ACUTE βελτιώνει το σύστημα κατά 21% ενώ ο UCP κατά 17% και στη χειρότερη περίπτωση ο ACUTE επιβραδύνει το σύστημα κατά 2%



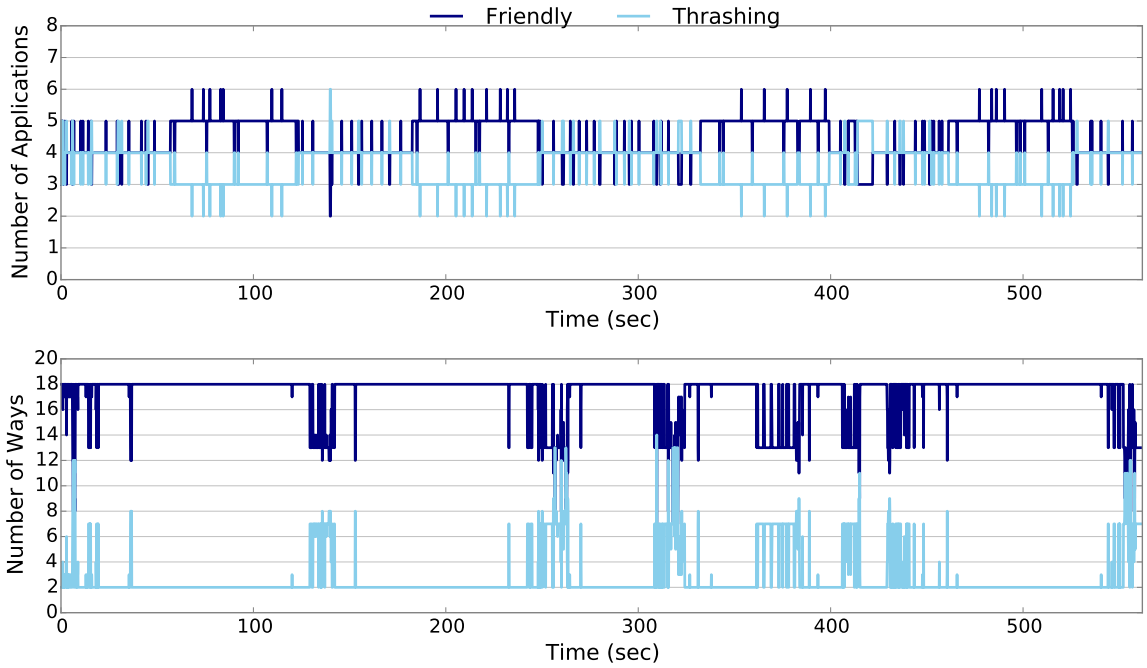
Εικόνα 5.3: Κέρδος στην απόδοση του συστήματος με την εφαρμογή του ACUTE σε σχέση με την κατάσταση NoPart

ενώ ο UCP κατά 12%. Τέλος ο UCP επιβραδύνει το σύστημα στο 54% των workloads ενώ ο ACUTE μόνο στο 15%.

Προκειμένου να γίνει κατανοητός ο δυναμικός διαχωρισμός των εφαρμογών και η δυναμική κατανομή της LLC, αναλύεται η εκτέλεση ενός workload. Στην Εικόνα 5.4 παρουσιάζεται για το workload blackscholes-gcc-bwaves-soplex-milc-mcf-libquantum-omnetpp, το πλήθος των εφαρμογών που ανήκουν στην ομάδα των cache friendly και cache thrashing εφαρμογών, αλλά και το πλήθος των ways της LLC που κατανέμεται σε κάθε ομάδα κατά το χρόνο εκτέλεσης του workload. Όπως έχει αναφερθεί, η εκτέλεση ενός workload ολοκληρώνεται όταν όλα τα benchmarks από τα οποία αποτελείται πραγματοποιήσουν τουλάχιστον μία ολοκληρωμένη εκτέλεση, ενώ όσα τερματίζουν νωρίτερα επανεκκινούνται.

Όπως φαίνεται στην Εικόνα 5.4 ο μηχανισμός ACUTE κατηγοριοποιεί δυναμικά τις εφαρμογές και υπάρχουν διακριτές φάσεις εκτέλεσης όπου διαφοροποιείται το πλήθος των εφαρμογών κάθε ομάδας. Αντίστοιχα, κατανέμεται δυναμικά και η LLC στις δύο ομάδες, όπου φαίνεται οι thrashing εφαρμογές να περιορίζονται σε σχέση με τις friendly. Πιο συγκεκριμένα, στις thrashing εφαρμογές κατανέμονται κυρίως 2-7 ways της cache ενώ στις friendly 13-18. Επομένως ο μηχανισμός ACUTE καταφέρνει να αναγνωρίσει τις φάσεις εκτέλεσης των εφαρμογών και να κατανείμει το μεγαλύτερο μέρος της cache στις εφαρμογές που θα ωφεληθούν περισσότερο τη δεδομένη στιγμή.

Επιπλέον, για τη συγκεκριμένη εκτέλεση του workload blackscholes-gcc-bwaves-soplex-milc-mcf-libquantum-omnetpp παρουσιάζονται στον Πίνακα 5.1 οι εναλλαγές που πραγματοποίησε κάθε benchmark από τη μία ομάδα εφαρμογών στην άλλη. Όπως φαίνεται υπάρχουν benchmarks τα οποία ανήκουν σταθερά σε μία ομάδα εφαρμογών, όπως το blackscholes και το soplex τα οποία ανήκουν στην ομάδα των cache thrashing και cache friendly εφαρμογών αντίστοιχα, ενώ άλλα εναλλάσσονται αρκετές φορές μεταξύ των δύο ομάδων ανάλογα με τη φάση εκτέλεσής τους.

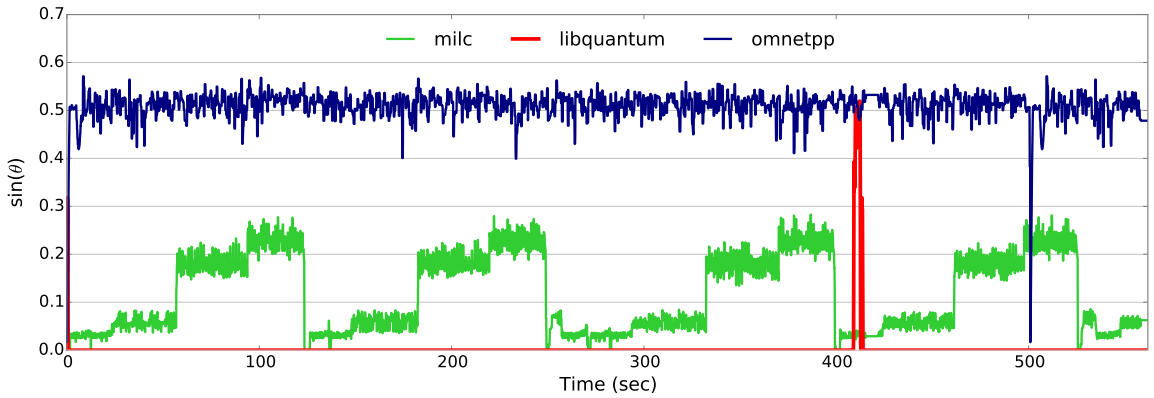


Εικόνα 5.4: Δυναμική εκτέλεση του workload blackscholes-gcc-bwaves-soplex-milc-mcf-libquantum-omnetpp

Bench	*blackscholes*	*gcc*	*bwaves*	*soplex*	*milc*	*mcf*	*libquantum*	*omnetpp*
Εναλλαγές	0	92	164	0	8	14	6	3

Πίνακας 5.1: Εναλλαγές των benchmarks μεταξύ των ομάδων των cache friendly και cache thrashing εφαρμογών κατά την εκτέλεση του workload blackscholes-gcc-bwaves-soplex-milc-mcf-libquantum-omnetpp

Ακόμη, στην Εικόνα 5.5 φαίνεται η τιμή του ημιτόνου της γωνίας θ, η οποία χρησιμοποιείται για την κατηγοριοποίηση των εφαρμογών (Ενότητα 5.1.1), όπως αυτή υπολογίζεται δυναμικά από τον μηχανισμό ACUTE για τρία benchmarks του υπό εξέταση workload.



Εικόνα 5.5: Δυναμικός υπολογισμός του sin(θ) τριών benchmarks από την εκτέλεση του workload blackscholes-gcc-bwaves-soplex-milc-mcf-libquantum-omnetpp

Παρατηρούμε πως το sin(θ) του benchmark omnetpp παίρνει σταθερά τιμές κοντά στο 0.5, δηλαδή έχει αρκετά μεγάλη γωνία $\theta \simeq 30°$, γι' αυτό και ανήκει στις cache friendly εφαρ-

μογές, εκτός από την αρχή της εκτέλεσής του και το σημείο της επανεκκίνησής του (t = 501 sec) όπου περιλαμβάνεται στις thrashing εφαρμογές για μία μικρή φάση εκτέλεσης. Από την άλλη πλευρά το sin(θ) του benchmark libquantum παίρνει σταθερά τιμές κοντά στο 0, δηλαδή έχει πολύ μικρή γωνία θ $\simeq$ 0°, γι' αυτό και ανήκει στις cache thrashing εφαρμογές, εκτός από την αρχή της εκτέλεσής του και το σημείο της επανεκκίνησής του (t = 413 sec) όπου περιλαμβάνεται στις friendly εφαρμογές για μία μικρή φάση εκτέλεσης. Τέλος, φαίνεται πως ο μηχανισμός ACUTE αναγνωρίζει διακριτές φάσεις εκτέλεσης του benchmark milc, το οποίο πραγματοποίησε μία ακριβώς ολοκληρωμένη εκτέλεση και δε χρειάστηκε να επανεκκινηθεί. Συγκεκριμένα, το sin(θ) του milc κυμαίνεται σε κάποιες φάσεις εκτέλεσης από 0 - 0.1 και περιλαμβάνεται στις cache thrashing εφαρμογές, ενώ σε άλλες από 0.15 - 0.25 και περιλαμβάνεται στις cache friendly εφαρμογές, γεγονός που αναδεικνύει την ανάγκη για δυναμική κατηγοριοποίηση των εφαρμογών.

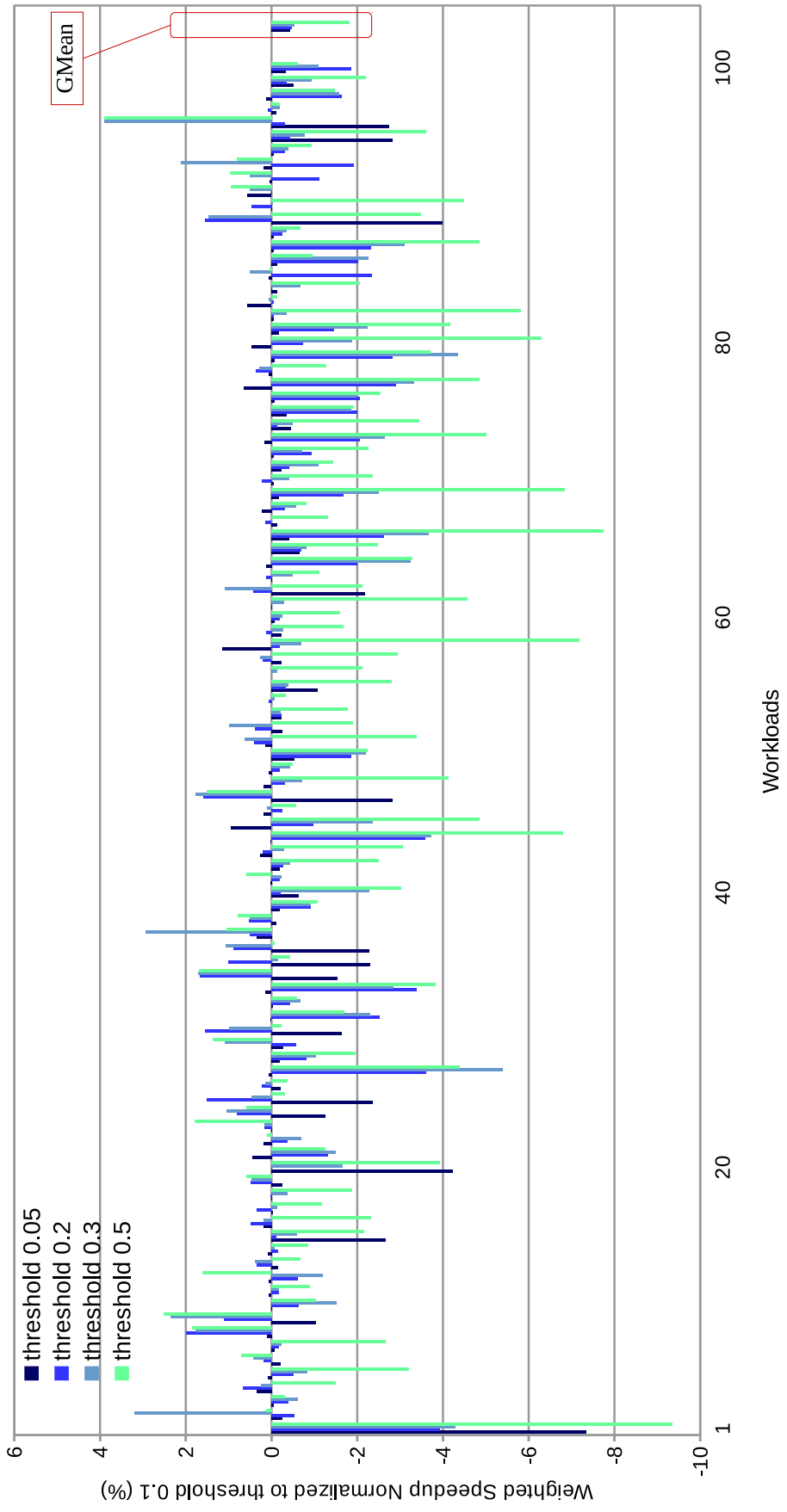
5.3 Ανάλυση παραμέτρων του μηχανισμού ACUTE

Στη συγκεκριμένη ενότητα αναλύεται η επίδραση ορισμένων παραμέτρων στον μηχανισμό ACUTE. Μία σημαντική παράμετρος του μηχανισμού ACUTE αποτελεί το κατώφλι t_{thr} που χρησιμοποιείται για τον διαχωρισμό των εφαρμογών σε cache friendly και cache thrashing, αφού αυτό καθορίζει εάν μία εφαρμογή ωφελείται ή όχι από την LLC. Μία άλλη παράμετρος είναι ο αριθμός των sets των κυκλωμάτων UMON, ο οποίος έχει μεγάλη επίδραση στο συνολικό κόστος του μηχανισμού. Τέλος το χρονικό διάστημα λήψης μετρήσεων και αποφάσεων για τον διαχωρισμό της LLC είναι μία ακόμη παράμετρος που μπορεί να επηρεάσει τη λειτουργία του μηχανισμού. Επομένως, στη συνέχεια εξετάζονται οι τιμές αυτών των τριών παραμέτρων με την εκτέλεση των κατάλληλων πειραμάτων.

5.3.1 Κατώφλι χαρακτηρισμού των thrashing εφαρμογών

Στην Ενότητα 5.1.1 παρουσιάστηκε ο τρόπος με τον οποίο ο μηχανισμός ACUTE κατηγοριοποιεί δυναμικά τις εφαρμογές στις ομάδες των cache friendly ή cache thrashing εφαρμογών. Σημαντικό ρόλο στη λήψη της παραπάνω απόφασης έχει η τιμή του ημιτόνου της γωνίας θ που σχηματίζεται μεταξύ του οριζόντιου άξονα και της ευθείας από το σημείο για ways=2 και το σημείο για ways=20 της miss rate curve κάθε εφαρμογής. Μετά τον έλεγχο κάποιων οριακών συνθηκών, ελέγχεται αν sin(θ) $\leq$t$_{thr}$ οπότε η εφαρμογή κατατάσσεται στις cache thrashing διαφορετικά στις cache friendly. Όπως έχει αναφερθεί, η τιμή του sin(θ) μπορεί να πάρει τιμές από 0 έως 0.7 και η τιμή του κατωφλίου t_{thr} έχει οριστεί στο 0.1. Για την ανάδειξη της τιμής του t_{thr} που επιφέρει τη μεγαλύτερη βελτίωση στο σύστημα, εκτελέστηκαν τα ίδια πειράματα με αυτά της Ενότητας 5.2 για $t_{thr} = 0.05, 0.2, 0.3, 0.5$. Στην Εικόνα 5.6 παρουσιάζεται για τα 100 workloads, η επί τοις εκατό μεταβολή του weighted speedup, για τις παραπάνω τιμές κατωφλίου, ως προς το επιλεγμένο κατώφλι 0.1.

Όπως φαίνεται στην Εικόνα 5.6, αν και υπάρχουν μεμονωμένες περιπτώσεις όπου κάποιες τιμές κατωφλίου παρουσιάζουν καλύτερη απόδοση απ' ότι το επιλεγμένο ($t_{thr} = 0.1$), κατά μέσο όρο κανένα από τα κατώφλια 0.05, 0.2, 0.3, 0.5 δε βελτιώνει την απόδοση του συστήματος περισσότερο από το 0.1. Πιο συγκεκριμένα για $t_{thr} = 0.05, 0.2, 0.3$ η διαφορά είναι της τάξης του 0.5% ενώ για $t_{thr} = 0.5$ είναι μεγαλύτερη, της τάξης του 1.8%. Για τον λόγο αυτό και στη συνέχεια των πειραμάτων διατηρήθηκε $t_{thr} = 0.1$.



Εικόνα 5.6: Σύγκριση τιμών του κατωφλίου t_{thr} του μηχανισμού ACUTE

5.3.2 Μέγεθος των κυκλωμάτων UMON

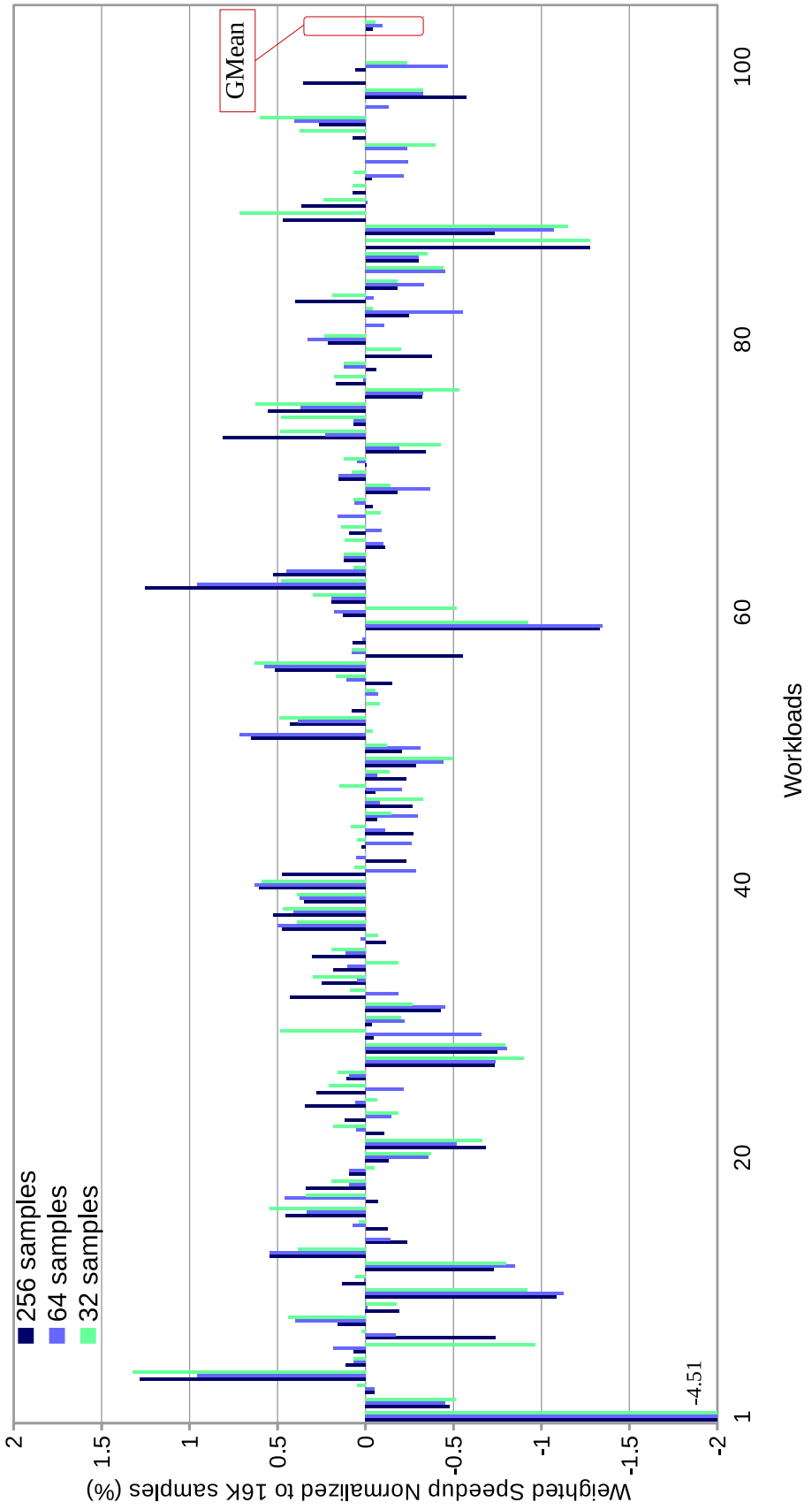
Μία ακόμη παράμετρος που πρέπει να εξεταστεί είναι ο αριθμός των sets της LLC που παρακολουθούνται από τα κυκλώματα UMON, τα οποία παρέχουν τις miss curves των εφαρμογών. Όπως παρουσιάστηκε στην Ενότητα 2.2.1, ένα κύκλωμα UMON, και συγκεκριμένα ο βοηθητικός κατάλογος ετικετών (ATD) του, κρατά πληροφορία για κάθε set της LLC. Μέχρι τώρα ο μηχανισμός ACUTE εξετάστηκε με τη χρήση αυτών των κυκλωμάτων UMON, που παρείχαν την πλήρη πληροφορία. Όμως το κόστος υλοποίησης ενός τέτοιου κυκλώματος για κάθε πυρήνα είναι αρκετά υψηλό. Γι' αυτόν τον λόγο εξετάζεται η χρήση κυκλωμάτων UMON τα οποία προσεγγίζουν τη συμπεριφορά της cache δειγματοληπτώντας μόνο μερικά sets της. Για το σκοπό αυτό εκτελέστηκαν τα ίδια πειράματα με αυτά της Ενότητας 5.2 με τη χρήση όμως κυκλωμάτων UMON που δειγματοληπτούν την LLC ανά 64, 256 και 512 sets και αφού το πλήρες κύκλωμα UMON που χρησιμοποιήθηκε αποτελείται από 16K sets θα αποτελούνται συνολικά από 256, 64 και 32 sets-δείγματα (samples) αντίστοιχα. Στην Εικόνα 5.7 παρουσιάζεται, για τα 100 workloads, η επί τοις εκατό μεταβολή του weighted speedup για 256, 64, και 32 samples στα κυκλώματα UMON, ως προς το weighted speedup των 16K samples (η μέχρι τώρα πλήρης πληροφορία).

Παρατηρώντας την Εικόνα 5.7 φαίνεται πως η πληροφορία που παρέχουν τα κυκλώματα UMON όταν δειγματοληπτούν την cache με συνολικά 256, 64 και 32 samples, είναι αντίστοιχα καλή με την πλήρη πληροφορία, αφού κατά μέσο όρο η μείωση σε σχέση με τα 16K sets είναι αμελητέα (της τάξης του 0.04%, 0.1% και 0.06% αντίστοιχα). Μία τέτοια μείωση στην απόδοση που πετυχαίνει ο μηχανισμός ACUTE δεν είναι σημαντική σε σχέση με τη μείωση της μνήμης που καταλαμβάνει το κύκλωμα UMON και κατ' επέκταση τη μείωση του κόστους υλοποίησής του, καθώς μειώνεται ο αριθμός των δειγμάτων-sets. Επομένως επιλέγεται η υλοποίηση των κυκλωμάτων UMON που αποτελούνται από 32 samples και τελικά ο ACUTE πετυχαίνει αύξηση της συνολικής απόδοσης του συστήματος κατά 5.2%. Το κόστος υλοποίησης των δειγματοληπτικών UMON υπολογίζεται και παρουσιάζεται στην Ενότητα 5.5.

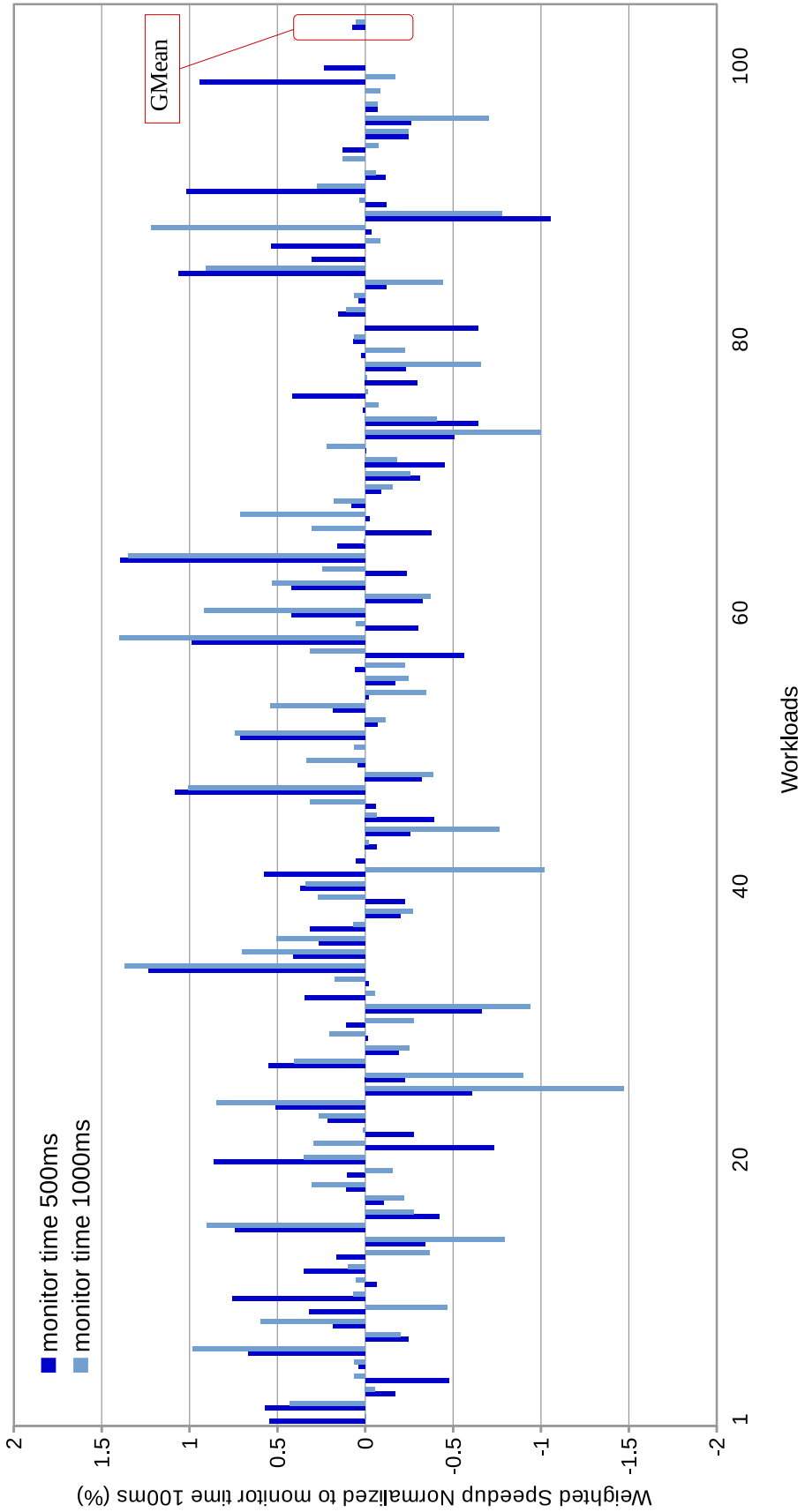
5.3.3 Χρονικό διάστημα παρακολούθησης

Το χρονικό διάστημα κατά το οποίο λαμβάνονται μετρήσεις και αποφάσεις για τον διαχωρισμό της LLC μέσω της βιβλιοθήκης PQoS, είναι μία παράμετρος που πρέπει να εξεταστεί. Στα μέχρι τώρα πειράματα αυτό το χρονικό διάστημα παρακολούθησης (monitor time) έχει καθοριστεί στα 100ms. Ωστόσο ένα πολύ μικρό διάστημα μπορεί να επιβαρύνει το σύστημα, ενώ σε ένα αρκετά μεγάλο μπορεί να χαθούν φάσεις εκτέλεσης των εφαρμογών. Γι' αυτόν τον λόγο εκτελέστηκαν τα ίδια πειράματα με αυτά της Ενότητας 5.2 για monitor time = 500 και 1000 msec και χρησιμοποιώντας τα κυκλώματα UMON που αποτελούνται από 32 samples, τα οποία επιλέχθηκαν στην προηγούμενη ανάλυση. Στην Εικόνα 5.8 παρουσιάζεται, για τα 100 workloads, η επί τοις εκατό μεταβολή του weighted speedup για τις δύο παραπάνω τιμές του monitor time, ως προς το weighted speedup για monitor time = 100ms, που έχει χρησιμοποιηθεί μέχρι στιγμής.

Στην Εικόνα 5.8 φαίνεται πως οριακά για monitor time = 500ms πετυχαίνεται η μεγαλύτερη βελτίωση στην απόδοση του συστήματος. Πιο συγκεκριμένα, για monitor time = 500ms και 1000ms η απόδοση που πετυχαίνει ο μηχανισμός ACUTE αυξάνεται κατά 0.07% και 0.05% αντίστοιχα σε σχέση με αυτή που πετυχαίνει για monitor time = 100ms. Αυτό συμβαίνει διότι ένα μικρό χρονικό διάστημα όπως τα 100ms επιβαρύνει περισσότερο το σύστημα σε σχέση με κάποιο μεγαλύτερο και είναι πιθανόν οι εφαρμογές να μην προλαβαίνουν σε αυτό το διάστημα να αξιοποιήσουν το τμήμα της LLC που τους αποδίδεται κάθε φορά.



Εικόνα 5.7: Σύγκριση τιμών των sets-δειγμάτων των UMON στον μηχανισμό ACUTE

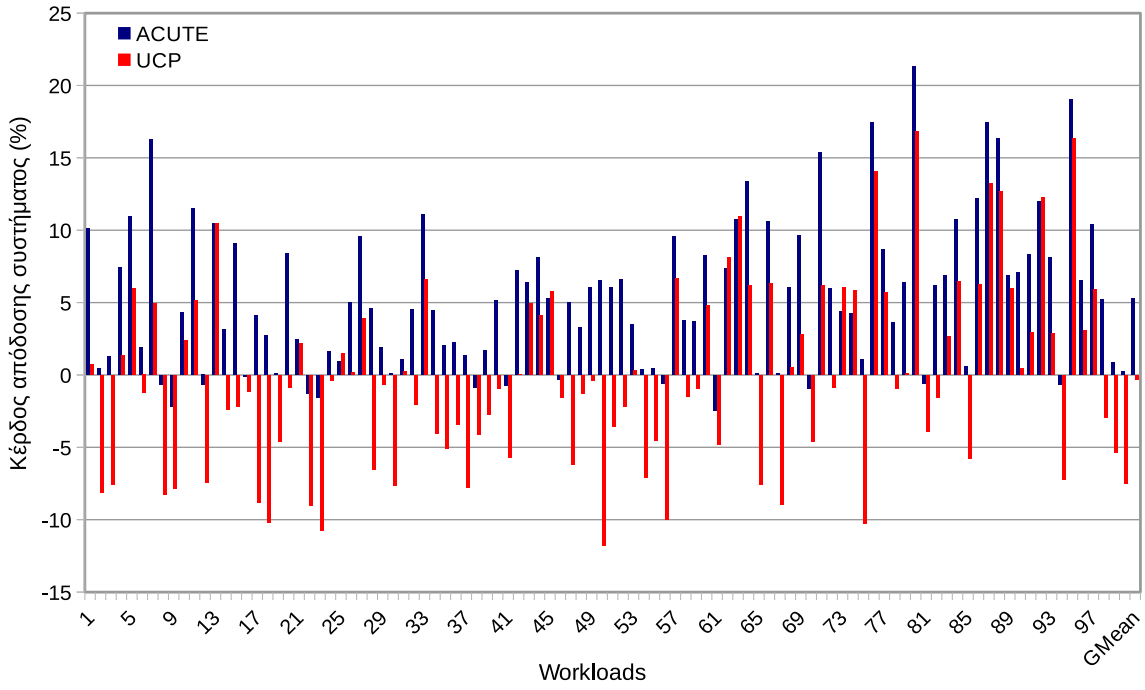


Εικόνα 5.8: Σύγκριση τιμών του monitor time στον μηχανισμό ACUTE

Από την άλλη πλευρά για monitor time = 500ms πετυχαίνεται καλύτερη απόδοση σε σχέση με τα 1000ms, διότι σε ένα μεγάλο χρονικό διάστημα όπως το 1000ms είναι πιθανόν να παραλείπονται φάσεις εκτέλεσης των εφαρμογών.

5.4 Σύγκριση ACUTE με UCP

Τελικά, προτείνουμε την εφαρμογή του μηχανισμού ACUTE με $t_{thr} = 0.1$, τη χρήση κυκλωμάτων UMON των 32 samples και monitor time = 500ms, όπου με αυτόν τον συνδυασμό επιτυγχάνεται αύξηση της συνολικής απόδοσης του συστήματος κατά 5.27%. Στην Εικόνα 5.9 παρουσιάζεται η τελική υλοποίηση του ACUTE σε σύγκριση με την τεχνική UCP. Όπως φαίνεται, ο μηχανισμός ACUTE υπερισχύει του UCP, αφού πετυχαίνει αύξηση της απόδοσης του συστήματος κατά 5.27% έναντι του -0.34% του UCP. Επιπλέον, στην καλύτερη περίπτωση ο ACUTE βελτιώνει το σύστημα κατά 21% ενώ ο UCP κατά 17% και στη χειρότερη περίπτωση ο ACUTE επιβραδύνει το σύστημα κατά 2% ενώ ο UCP κατά 12%. Τέλος, ο UCP επιβραδύνει το σύστημα στο 54% των workloads ενώ ο ACUTE μόνο στο 14%. Επομένως, ο μηχανισμός ACUTE πετυχαίνει σημαντική αύξηση της συνολικής απόδοσης του συστήματος και το κόστος υλοποίησής του παρουσιάζεται στην επόμενη ενότητα.



Εικόνα 5.9: Σύγκριση μηχανισμών ACUTE και UCP

5.5 Υπολογισμός κόστους υλοποίησης του μηχανισμού ACUTE

Το βασικό στοιχείο της διαμόρφωσης του κόστους υλοποίησης του μηχανισμού ACUTE είναι τα κυκλώματα UMON. Όπως αναφέρθηκε στην Ενότητα 5.3, επιλέχθηκε κάθε κύκλωμα UMON να αποτελείται από 32 sets-δείγματα προκειμένου να μειωθεί η μνήμη που καταλαμβάνει. Παρακάτω παρουσιάζεται αναλυτικά η επιπλέον μνήμη που απαιτείται για την υλοποίηση ενός κυκλώματος UMON με 32 sets-δείγματα, θεωρώντας μέγεθος φυσικής διεύθυνσης 64bits:

- Κάθε είσοδος στον κατάλογο ATD αποτελείται από 1bit εγκυρότητας, 44-bit tag και 5-bit LRU, δηλαδή συνολικά 50bits.
- Οι είσοδοι στον ATD για κάθε set-δείγμα είναι 20, αφού η LLC είναι 20-way set associative, οπότε το συνολικό κόστος για κάθε set-δείγμα στον ATD είναι 50bits * 20 = 1000bits = 125bytes.
- Επομένως, το συνολικό κόστος του ATD για 32 sets-δείγματα είναι 125bytes * 32 = 4000bytes.
- Το κόστος των μετρητών των hits (ένας για κάθε way, άρα 20) και του μετρητή των misses, είναι 21 * 4bytes = 84bytes. Επίσης απαιτείται και ένας αθροιστής για τη μέτρηση των hits, misses αλλά η αξία του είναι αμελητέα και παραλείπεται σε αυτή την ανάλυση.
- Επομένως το συνολικό κόστος του UMON είναι το άθροισμα του κόστους του ATD και των μετρητών, δηλαδή 4000bytes + 84bytes = 4084bytes.

Η μνήμη που καταλαμβάνει ένα κύκλωμα UMON στην πραγματικότητα είναι ακόμη λιγότερη, αφού συνήθως χρησιμοποιείται 48-bit φυσική διεύθυνση ενώ στην ανάλυσή μας θεωρήσαμε τη χειρότερη περίπτωση της 64-bit φυσικής διεύθυνσης.

Η μνήμη που καταλαμβάνει η LLC στο σύστημα είναι το άθροισμα των 25MB δεδομένων και των απαραίτητων tags για την αναγνώριση του πυρήνα στον οποίο ανήκει κάθε είσοδος στην cache (sets * ways * tag_size = 20480 * 20 * 4bytes = 1600Kbytes). Επομένως η συνολική μνήμη που καταλαμβάνει η LLC είναι 1600Kbytes + 25MB = 27200Kbytes. Τελικά για την υλοποίηση ενός κυκλώματος UMON απαιτείται αύξηση της μνήμης της LLC κατά (4084bytes / 27200Kbytes) = 0.015%. Και για την υλοποίηση ενός κυκλώματος UMON για κάθε έναν από τους 10 πυρήνες του συστήματος απαιτείται αύξηση της μνήμης της LLC κατά 0.15%. Συνεπώς το κόστος υλοποίησης των κυκλωμάτων UMON, και επομένως του μηχανισμού ACUTE, είναι χαμηλό ακόμη και με την αύξηση του αριθμού των πυρήνων. Στην περίπτωση που δεν είχαν επιλεγεί δειγματοληπτικά κυκλώματα UMON αλλά αυτά που διατηρούσαν τη συνολική πληροφορία από την LLC, δηλαδή αποτελούνταν από 16K sets, τότε το κόστος υλοποίησής τους θα αυξανόταν κατακόρυφα. Συγκεκριμένα το κόστος υλοποίησης ενός UMON θα ήταν 2000Kbytes, δηλαδή αύξηση της μνήμης της LLC κατά 7.35%, και για την υλοποίηση ενός κυκλώματος UMON για κάθε έναν από τους 10 πυρήνες του συστήματος αύξηση της μνήμης της LLC κατά 73.5%, κόστος που είναι απαγορευτικό.

Τελικά με την επιλογή των δειγματοληπτικών UMON, ο ACUTE αποτελεί ένα χαμηλού κόστους μηχανισμό που βελτιώνει τη συνολική απόδοση των σύγχρονων, πραγματικών, πολυπύρηνων συστημάτων.

Κεφάλαιο 6

Συμπεράσματα και Μελλοντικές επεκτάσεις

Πολλές έρευνες έχουν πραγματοποιηθεί με στόχο την εύρεση ενός τρόπου για τη βελτίωση της απόδοσης των πολυεπεξεργαστικών συστημάτων, η οποία πλήττεται από τη διαμάχη των συνεκτελούμενων εφαρμογών για τους κοινόχρηστους πόρους. Ο βασικότερος από αυτούς τους πόρους είναι το τελευταίο επίπεδο της cache (LLC), που είναι κοινό για όλους τους πυρήνες ενός συστήματος, και έχουν προταθεί πολλές τεχνικές για τον διαμοιρασμό του. Ωστόσο οι περισσότερες από αυτές έχουν αξιολογηθεί μέσω προσομοιώσεων, αφού δεν υπήρχε hardware υποστήριξη για την επιβολή του διαχωρισμού στην cache. Μετά την εισαγωγή αυτής της hardware υποστήριξης από την Intel και την ARM, η ερευνητική κοινότητα έχει στραφεί στην υλοποίηση και αξιολόγηση τεχνικών διαμοιρασμού της LLC σε πραγματικά μηχανήματα.

Στην παρούσα διπλωματική εργασία έγινε παρουσίαση πολλών τεχνικών διαμοιρασμού της κοινόχρηστης cache που έχουν προταθεί πριν και μετά την εισαγωγή των hardware μηχανισμών της Intel και της ARM. Επιπλέον παρουσιάστηκε ο hardware μηχανισμός της Intel, CMT-CAT, ο οποίος χρησιμοποιήθηκε για τη διεξαγωγή όλων των πειραμάτων στο πραγματικό μηχάνημα. Στη συνέχεια αναλύθηκε η συμπεριφορά ορισμένων εφαρμογών σε σχέση με το μέγεθος της LLC που έχουν διαθέσιμο και έπειτα αναδείχθηκε το πρόβλημα που δημιουργείται από την συνεκτέλεση πολλών εφαρμογών στο ίδιο σύστημα. Επομένως καταλήξαμε στο συμπέρασμα πως μία τεχνική διαμοιρασμού της cache πρέπει να λαμβάνει υπ' όψιν της τα διαφορετικά χαρακτηριστικά των εφαρμογών ως προς τη χρησιμότητα της cache αλλά και να μπορεί να κλιμακωθεί με τη συνεχή αύξηση του αριθμού των πυρήνων και εφαρμογών.

Τελικά προτάθηκε ο μηχανισμός ACUTE, ένας χαμηλού κόστους, αποδοτικός μηχανισμός για τον διαμοιρασμό της κοινόχρηστης cache, με στόχο τη βελτίωση της συνολικής απόδοσης ενός σύγχρονου πολυεπεξεργαστικού συστήματος. Ο μηχανισμός ACUTE λειτουργεί κατηγοριοποιώντας τις εφαρμογές δυναμικά, κατά το χρόνο εκτέλεσής τους, σε δύο κατηγορίες ανάλογα με το πόσο επωφελούνται ή όχι από τη χρήση της LLC. Έπειτα διαμοιράζει την cache σε επίπεδο way στις δύο ομάδες εφαρμογών κάνοντας χρήση της νέας τεχνολογίας CAT της Intel. Με αυτόν τον τρόπο προσφέρει ευελιξία στις εφαρμογές καθώς διατηρεί τη μεγάλη συσχετιστικότητα της cache, ιδιαίτερα στο τμήμα της cache που αποδίδεται στις εφαρμογές που τη χρειάζονται περισσότερο. Σε σχέση με την τεχνική UCP, ο μηχανισμός ACUTE υπερισχύει, πετυχαίνοντας σημαντική βελτίωση στην απόδοση του πραγματικού συστήματος και έχοντας τη δυνατότητα να κλιμακωθεί με την αύξηση του αριθμού των εφαρμογών.

Η παρούσα διπλωματική εργασία μπορεί μελλοντικά να επεκταθεί προς πολλές κατευθύνσεις. Αρχικά η λήψη της πληροφορίας για τη χρησιμότητα της LLC κάθε εφαρμογής, που στον μηχανισμό ACUTE λαμβάνεται από τα κυκλώματα UMON, μπορεί να γίνει και με την υλοποίηση κάποιου μηχανισμού στο λογισμικό. Για παράδειγμα, μπορεί να υπάρχουν τακτικές φάσεις στην εκτέλεση των εφαρμογών κατά τις οποίες μία εφαρμογή κάθε φορά εξετάζεται για την εξαγωγή του προφίλ της ενώ οι υπόλοιπες περιορίζονται. Επιπλέον ο μηχανισμός

ACUTE πραγματοποιεί διαχωρισμό των εφαρμογών σε δύο ομάδες, τις cache friendly/fitting και τις cache thrashing, επομένως μία επέκτασή του μπορεί να είναι ο διαχωρισμός των εφαρμογών σε περισσότερες των δύο ομάδων. Ακόμη, για τον διαχωρισμό των εφαρμογών μπορεί να χρησιμοποιηθεί επιπλέον πληροφορία από τη χρήση άλλων κοινόχρηστων πόρων, όπως το bandwidth της κύριας μνήμης. Τέλος, αντί να ορίζεται κάθε εφαρμογή να εκτελεστεί στατικά σε έναν πυρήνα και να αντιστοιχίζεται ο πυρήνας σε ένα RMID/CLOS για τη χρήση της τεχνολογίας CMT/CAT, μπορεί να δοκιμαστεί η αντιστοίχιση του RMID/CLOS με το αναγνωριστικό id μιας διεργασίας, ώστε να γίνει παρακολούθηση των εφαρμογών καθώς αυτές μετακινούνται μεταξύ των πυρήνων, πραγματοποιώντας ορισμένες αλλαγές στον scheduler. Σε κάθε περίπτωση οι τεχνολογίες CMT-CAT της Intel παρέχουν πολλές νέες δυνατότητες και έχουν ανοίξει έναν νέο δρόμο για την αξιολόγηση των προτεινόμενων τεχνικών διαμοιρασμού των κοινόχρηστων πόρων στα σύγχρονα συστήματα.

Βιβλιογραφία

[1] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer. Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family. In *HPCA*, 2016.

[2] Intel Corporation. *"Improving Real-Time Performance by Utilizing Cache Allocation Technology"*. April 2015. Chapter 3.6.

[3] C. Maurice, N. Scouarnec, C. Neumann, O. Heen, and A. Francillon. Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters. In *Proceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 9404*, RAID 2015, pages 48–65, New York, USA, 2015.

[4] Moinuddin K. Qureshi and Yale N. Patt. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 39, pages 423–432, Washington, DC, USA, 2006. IEEE Computer Society.

[5] Konstantinos Nikas, Matthew Horsnell, and Jim D. Garside. An Adaptive Bloom Filter Cache Partitioning Scheme for Multicore Architectures. In *2008 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 25–32. IEEE, July 2008.

[6] Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely Jr., and Joel S. Emer. Adaptive Insertion Policies for High Performance Caching. In *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA*, pages 381–391, 2007.

[7] Aamer Jaleel, William Hasenplaugh, Moinuddin K. Qureshi, Julien Sebot, Simon C. Steely Jr., and Joel S. Emer. Adaptive Insertion Policies for Managing Shared Caches. In *17th International Conference on Parallel Architecture and Compilation Techniques, PACT 2008, Toronto, Ontario, Canada, October 25-29, 2008*, pages 208–219, 2008.

[8] Yuejian Xie and Gabriel H. Loh. PIPP: Promotion/Insertion Pseudo-Partitioning of Multi-core Shared Caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 174–183, New York, NY, USA, 2009. ACM.

[9] Daniel Sanchez and Christos Kozyrakis. Vantage: Scalable and Efficient Fine-grain Cache Partitioning. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 57–68, New York, NY, USA, 2011. ACM.

[10] Karthik T. Sundararajan, Vasileios Porpodas, Timothy M. Jones, Nigel P. Topham, and Björn Franke. Cooperative Partitioning: Energy-Efficient Cache Partitioning for High-Performance CMPs. In *HPCA*, pages 311–322. IEEE Computer Society, 2012.

[11] H. Cook, M. Moreto, S. Bird, K. Dao, D. A. Patterson, and K. Asanovic. A Hardware Evaluation of Cache Partitioning to Improve Utilization and Energy-Efficiency while Preserving Responsiveness. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 308–319, New York, NY, USA, 2013. ACM.

[12] Xiaodong Wang and José F. Martínez. Xchange: A market-based approach to scalable dynamic multi-resource allocation in multicore architectures. In *21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, February 7-11, 2015*, pages 113–125, 2015.

[13] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 450–462, New York, NY, USA, 2015. ACM.

[14] J. Brock, C. Ye, C. Ding, Y. Li, X. Wang, and Y. Luo. Optimal Cache Partition-Sharing. In *2015 44th International Conference on Parallel Processing*, pages 749–758, Sept 2015.

[15] Ioannis Papadakis, Konstantinos Nikas, Vasileios Karakostas, Georgios Goumas, and Nectarios Koziris. Improving QoS and Utilisation in modern multi-core servers with Dynamic Cache Partitioning. In *2nd Workshop on Co-Scheduling of HPC Applications (COSH 2017) - held in conjuction with HiPEAC*, COSH 2017, 2017.

[16] X. Wang, S. Chen, J. Setter, and J. F. Martínez. SWAP: Effective Fine-Grain Management of Shared Last-Level Caches with Minimum Hardware Support. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 121–132, Feb 2017.

[17] M. Xu, L. Thi, X. Phan, H. Y. Choi, and I. Lee. vCAT: Dynamic Cache Management Using CAT Virtualization. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 211–222, April 2017.

[18] V. Selfa, J. Sahuquillo, L. Eeckhout, S. Petit, and M. E. Gómez. Application Clustering Policies to Address System Fairness with Intel's Cache Allocation Technology. In *26th International Conference on Parallel Architectures and Compilation Techniques, PACT 2017, Portland, OR, USA, September 9-13, 2017*, pages 194–205, 2017.

[19] Sparsh Mittal. A Survey of Techniques for Cache Partitioning in Multicore Processors. *ACM Computing Surveys, vol.50, no. 2*, pages 27:1–27:39, May 2017.

[20] Moinuddin K. Qureshi, Daniel N. Lynch, Onur Mutlu, and Yale N. Patt. A Case for MLP-Aware Cache Replacement. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ISCA '06, pages 167–178, Washington, DC, USA, 2006. IEEE Computer Society.

[21] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and QoS-aware Cluster Management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 127–144, New York, NY, USA, 2014. ACM.

[22] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society, Applied Statistics, vol.28, no. 1*, pages 100–108, 1979.

[23] J. C. Dunn. A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics, vol.3, no. 3*, pages 32–57, 1973.

[24] Platform shared resource monitoring: Cache monitoring technology. In *Intel® 64 and IA-32 Architectures Developer's Manual: Vol.3B*, pages 147–153, 2015.

[25] Nguyen Khang T (Intel). *"Intel's Cache Monitoring Technology Software-Visible Interfaces"*. December 2014.

[26] Platform shared resource monitoring: Cache allocation technology. In *Intel® 64 and IA-32 Architectures Developer's Manual: Vol.3B*, pages 153–163, 2015.

[27] Princeton Application Repository for Shared-Memory Computers. `http://parsec.cs.princeton.edu/`. [Online].

[28] Standard Performance Evaluation Corporation. `https://www.spec.org/`. [Online].

[29] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.

[30] Anurag Mukkara, Nathan Beckmann, and Daniel Sanchez. Whirlpool: Improving dynamic cache management with static data classification. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 113–127, New York, NY, USA, 2016. ACM.

Παράρτημα A

Ο αλγόριθμος lookahead

Παρακάτω παρουσιάζεται ο αλγόριθμος lookahead όπως αυτός προτάθηκε από τους Qureshi και Patt [4] το 2006:

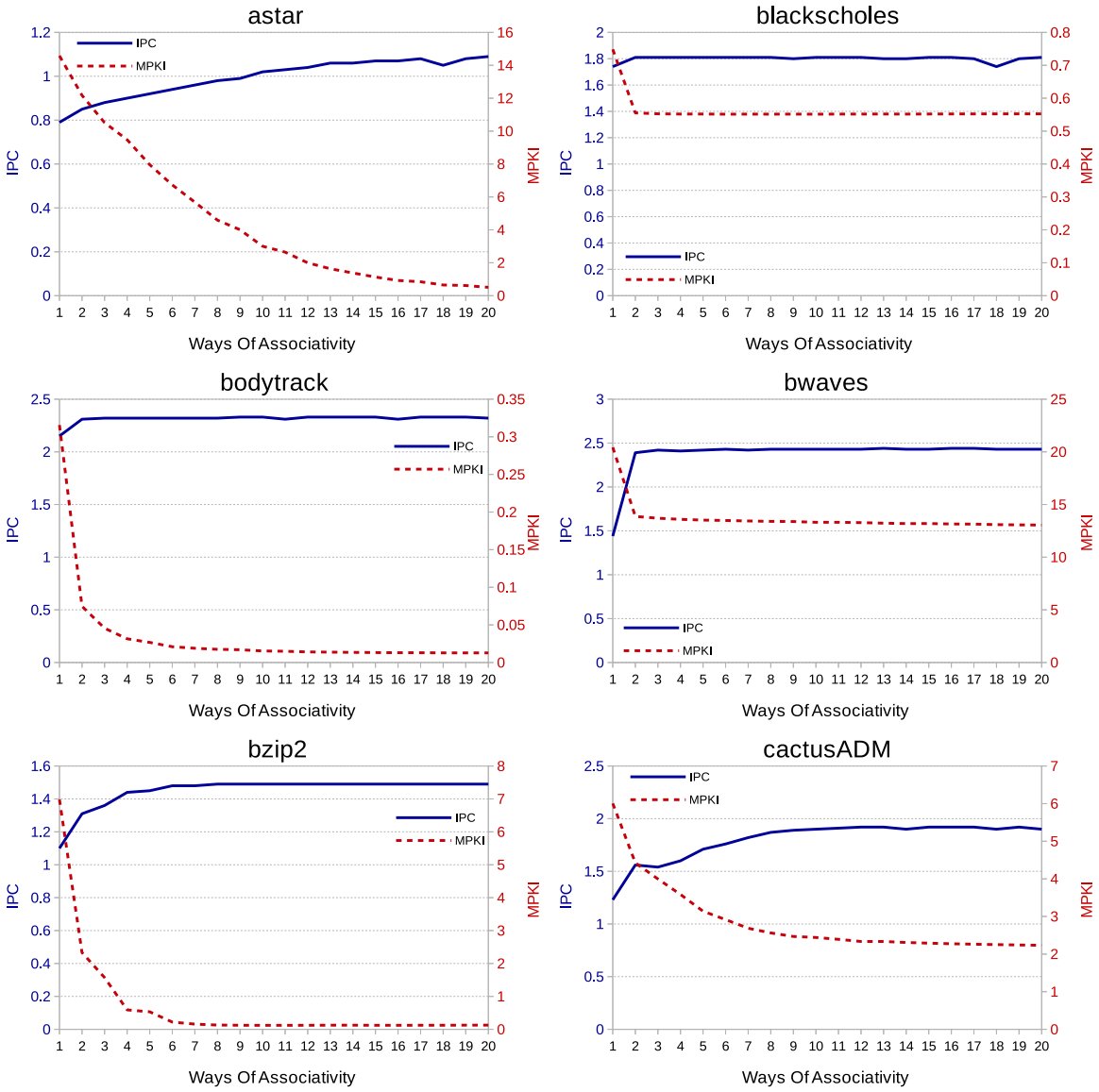
Algorithm 2 Ψευδοκώδικας για τον αλγόριθμο lookahead

```
1:  balance = N      /*N is the num of ways to be allocated */
2:  foreach application i do:
3:      allocations[i] = 0
4:  while(balance) do:
5:      foreach application i do:
6:          alloc = allocations[i]
7:          max_mu[i] = get_max_mu(i, alloc, balance)
8:          ways_req = min ways to get max_mu[i] for i
9:      winner = application with maximum value of max_mu
10:     allocations[winner] += ways_req[winner]
11:     balance -= ways_req[winner]
12: return allocations
13: get_max_mu(p, alloc, balance):
14:     max_mu = 0
15:     for(ii=1; ii<=balance; ii++) do:
16:         mu = get_mu_value(p,alloc,alloc+ii)
17:         if(mu > max_mu): max_mu = mu
18:     return max_mu
19: get_mu_value(p, a, b):
20:     U = change in misses for application p when the
        number of ways assigned to it increases from a to b
21:     return U/(b-a)
```

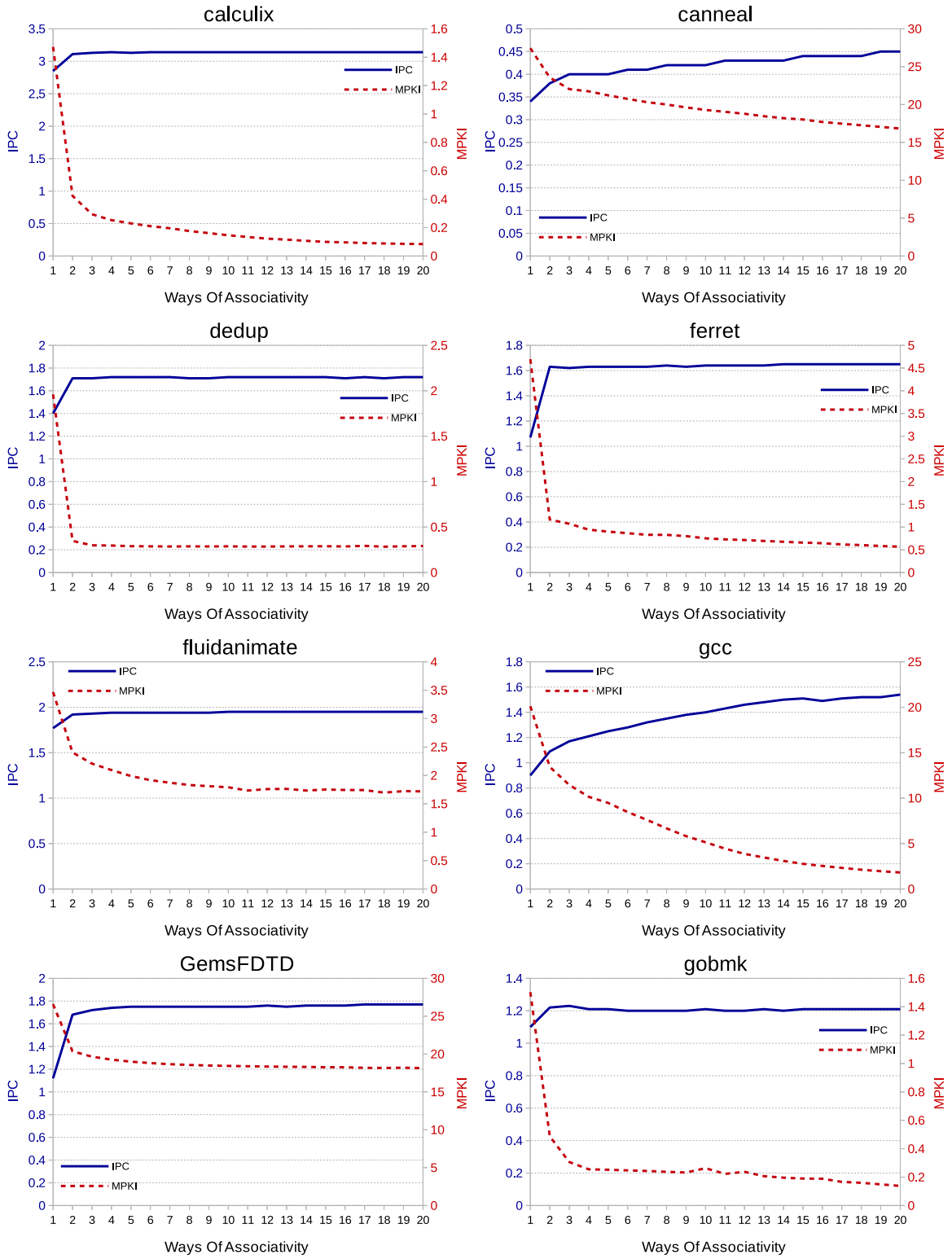
Παράρτημα Β

Χαρακτηριστικά των benchmarks

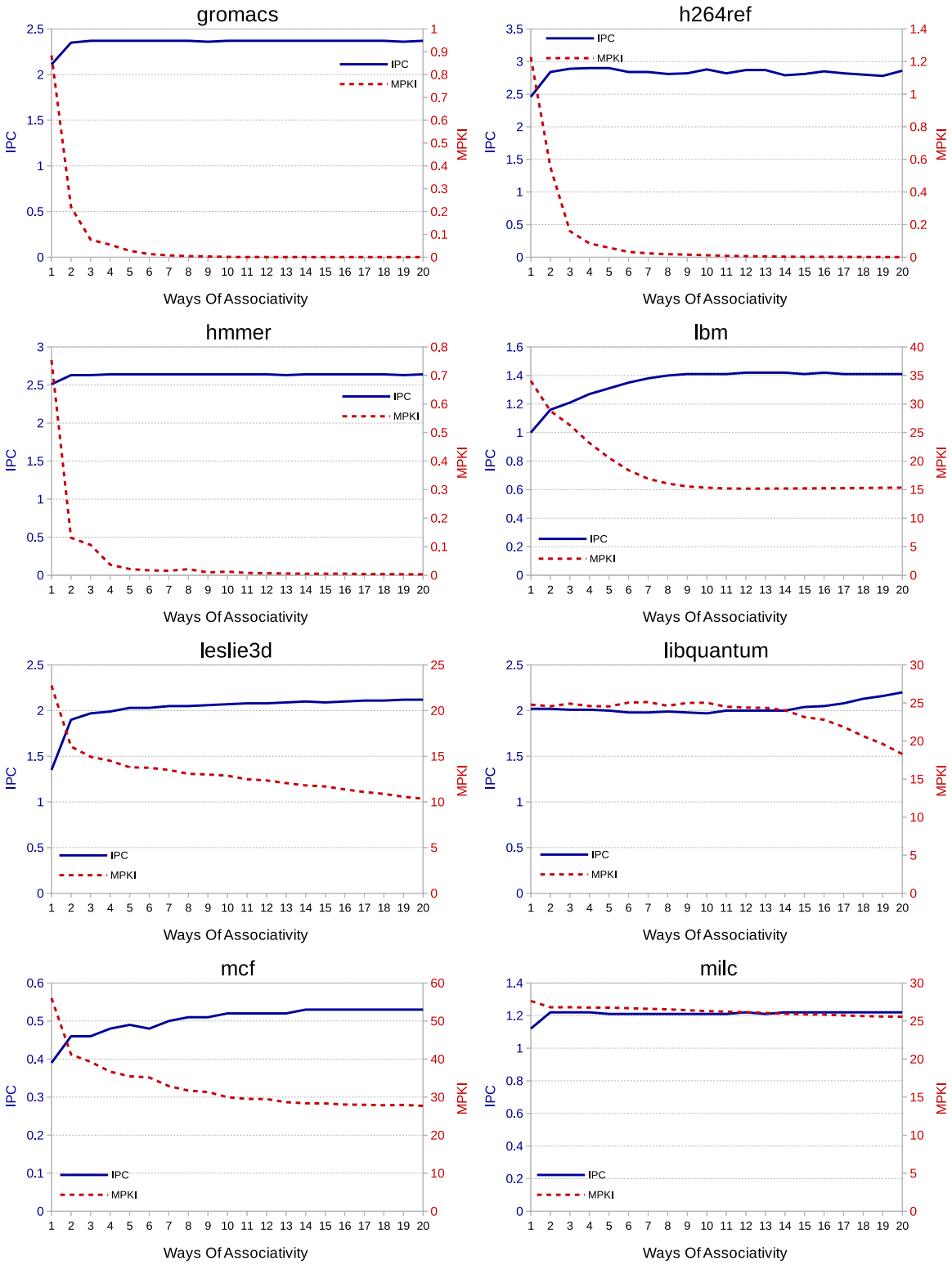
Στο παράρτημα αυτό παρουσιάζονται οι γραφικές παραστάσεις του IPC και του MPKI συναρτήσει του μεγέθους της LLC, σε επίπεδο way, για τα 35 benchmarks που χρησιμοποιήθηκαν στην παρούσα διπλωματική εργασία, όταν αυτά εκτελέστηκαν μόνα τους στο πραγματικό σύστημα.



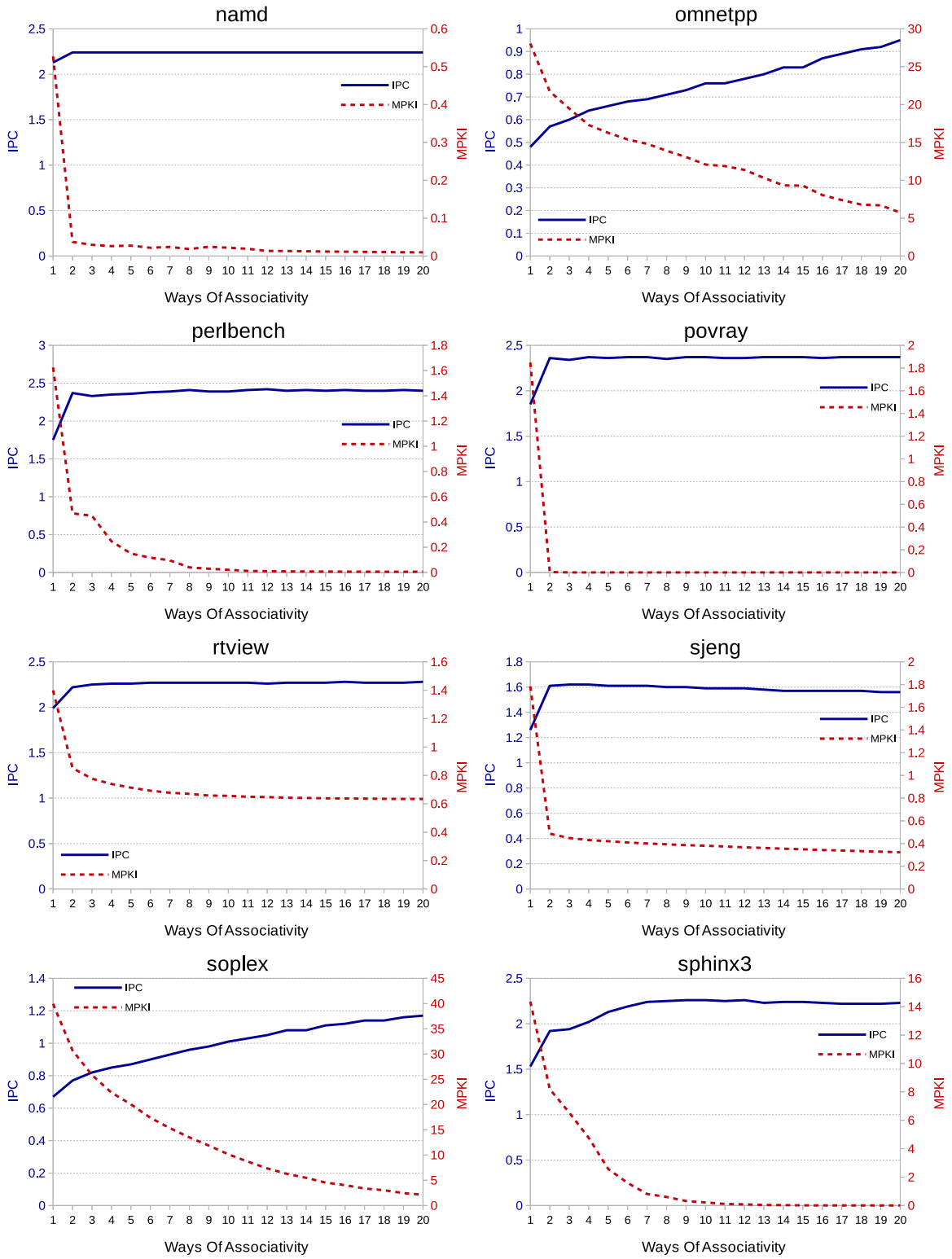
Εικόνα Β.1: IPC, MPKI ανά μέγεθος της LLC για τα benchmarks: astar, blackscholes, bodytrack, bwaves, bzip2, cactusADM



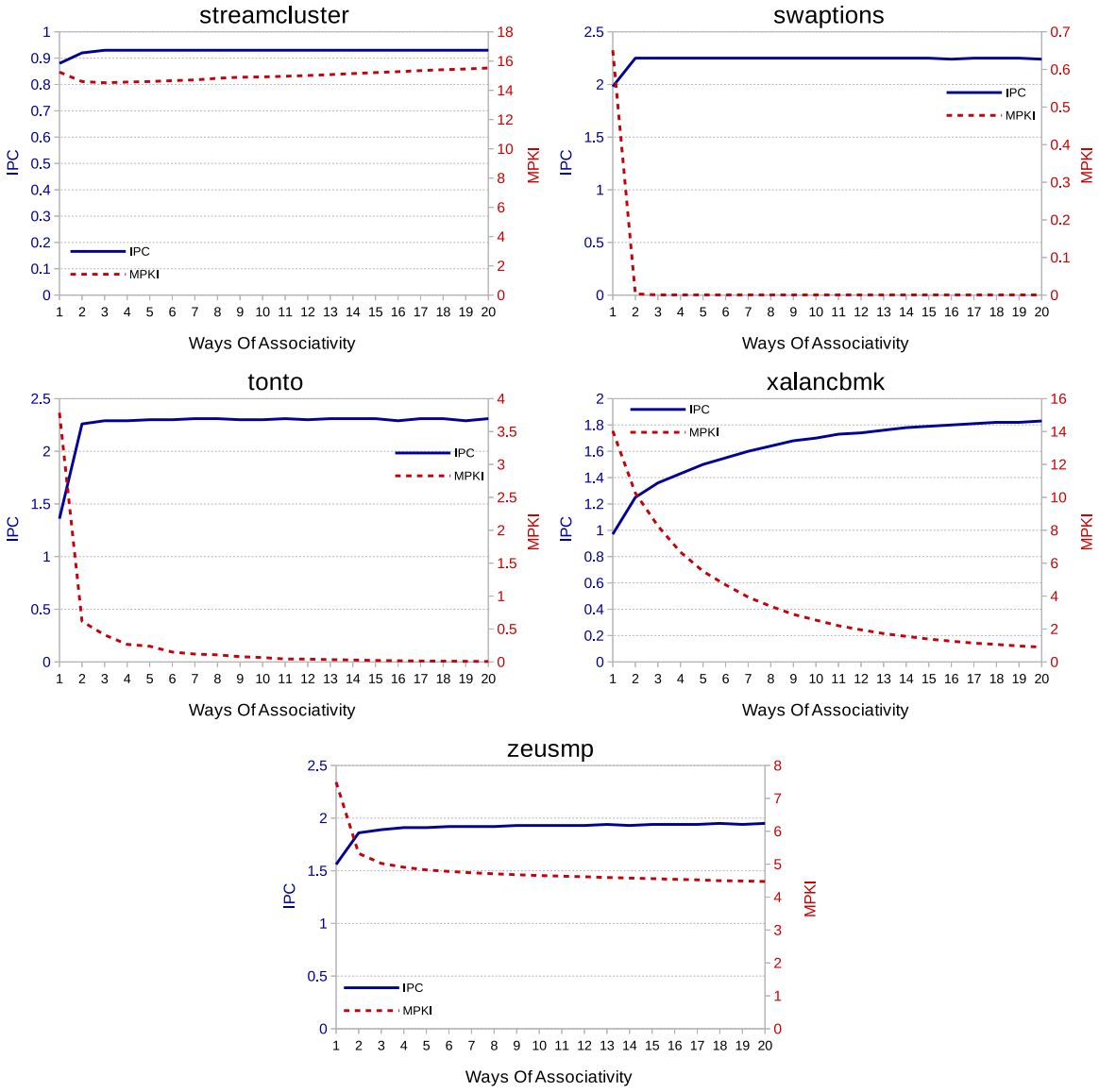
Εικόνα B.2: IPC, MPKI ανά μέγεθος της LLC για τα benchmarks: calculix, canneal, dedup, ferret, fluidanimate, gcc, GemsFDTD, gobmk



Εικόνα Β.3: IPC, MPKI ανά μέγεθος της LLC για τα benchmarks: gromacs, h264ref, hmmer, lbm, leslie3d, libquantum, mcf, milc



Εικόνα Β.4: IPC, MPKI ανά μέγεθος της LLC για τα benchmarks: namd, omnetpp, perlbench, povray, rtview, sjeng, soplex, sphinx3



Εικόνα Β.5: IPC, MPKI ανά μέγεθος της LLC για τα benchmarks: streamcluster, swaptions, tonto, xalancbmk, zeusmp

Παράρτημα C

Workloads

Στο παράρτημα αυτό παρουσιάζονται τα workloads των 8 benchmarks που χρησιμοποιήθηκαν στα πειράματα της παρούσας διπλωματικής εργασίας με τη σειρά που εμφανίζονται σε όλες τις γραφικές παραστάσεις. Τα 52 πρώτα είναι αυτά που επιλέχθηκαν τυχαία ενώ τα 48 επόμενα είναι αυτά που θεωρήθηκε πως θα παρουσιάσουν σημαντική επιβράδυνση. Τα benchmarks που ανήκουν στο πακέτο των SPEC CPU2006 [28] αναγράφονται με τον αναγνωριστικό αριθμό τους, ενώ τα benchmarks που ανήκουν στο PARSEC-3.0 [27] αναγράφονται μαζί με το αναγνωριστικό *parsec*.

1. 462.libquantum-471.omnetpp-429.mcf-450.soplex-410.bwaves-433.milc-482.sphinx3-459.GemsFDTD
2. parsec.dedup-483.xalancbmk-parsec.streamcluster-parsec.ferret-429.mcf-437.leslie3d-410.bwaves-444.namd
3. parsec.ferret-444.namd-470.lbm-410.bwaves-473.astar-471.omnetpp-parsec.streamcluster-401.bzip2
4. 429.mcf-483.xalancbmk-433.milc-429.mcf-462.libquantum-403.gcc-parsec.streamcluster-464.h264ref
5. 462.libquantum-471.omnetpp-473.astar-462.libquantum-parsec.ferret-454.calculix-482.sphinx3-456.hmmer
6. parsec.blackscholes-401.bzip2-464.h264ref-462.libquantum-437.leslie3d-465.tonto-456.hmmer-parsec.bodytrack
7. 465.tonto-482.sphinx3-470.lbm-462.libquantum-462.libquantum-483.xalancbmk-456.hmmer-462.libquantum
8. 464.h264ref-454.calculix-434.zeusmp-436.cactusADM-410.bwaves-435.gromacs-445.gobmk-482.sphinx3
9. parsec.bodytrack-434.zeusmp-437.leslie3d-473.astar-465.tonto-parsec.bodytrack-parsec.bodytrack-410.bwaves
10. 483.xalancbmk-462.libquantum-464.h264ref-483.xalancbmk-parsec.fluidanimate-437.leslie3d-458.sjeng-parsec.fluidanimate
11. parsec.fluidanimate-471.omnetpp-465.tonto-462.libquantum-482.sphinx3-parsec.streamcluster-462.libquantum-482.sphinx3
12. 454.calculix-parsec.swaptions-433.milc-470.lbm-410.bwaves-482.sphinx3-parsec.bodytrack-parsec.streamcluster
13. 462.libquantum-462.libquantum-454.calculix-462.libquantum-437.leslie3d-482.sphinx3-433.milc-433.milc
14. 462.libquantum-parsec.blackscholes-462.libquantum-482.sphinx3-410.bwaves-parsec.ferret-parsec.ferret-parsec.ferret
15. 403.gcc-483.xalancbmk-470.lbm-436.cactusADM-471.omnetpp-482.sphinx3-459.GemsFDTD-462.libquantum
16. parsec.dedup-parsec.ferret-464.h264ref-445.gobmk-462.libquantum-462.libquantum-454.calculix-465.tonto

17. 400.perlbench-450.soplex-parsec.blackscholes-470.lbm-435.gromacs-483.xalancbmk-462.libquantum-parsec.bodytrack
18. parsec.streamcluster-410.bwaves-473.astar-465.tonto-483.xalancbmk-parsec.bodytrack-450.soplex-453.povray
19. parsec.swaptions-parsec.dedup-435.gromacs-401.bzip2-434.zeusmp-parsec.blackscholes-400.perlbench-parsec.fluidanimate
20. 459.GemsFDTD-458.sjeng-429.mcf-parsec.streamcluster-403.gcc-462.libquantum-parsec.rtview-parsec.bodytrack
21. parsec.streamcluster-436.cactusADM-436.cactusADM-parsec.fluidanimate-parsec.fluidanimate-parsec.fluidanimate-462.libquantum-473.astar
22. 435.gromacs-482.sphinx3-parsec.swaptions-456.hmmer-465.tonto-470.lbm-parsec.streamcluster-429.mcf
23. 445.gobmk-parsec.streamcluster-470.lbm-410.bwaves-456.hmmer-444.namd-465.tonto-473.astar
24. 459.GemsFDTD-parsec.swaptions-459.GemsFDTD-454.calculix-434.zeusmp-456.hmmer-401.bzip2-444.namd
25. 462.libquantum-458.sjeng-459.GemsFDTD-parsec.streamcluster-401.bzip2-parsec.streamcluster-parsec.streamcluster-parsec.fluidanimate
26. 450.soplex-462.libquantum-462.libquantum-parsec.swaptions-parsec.swaptions-410.bwaves-parsec.fluidanimate-445.gobmk
27. 470.lbm-parsec.canneal-433.milc-462.libquantum-parsec.swaptions-470.lbm-482.sphinx3-462.libquantum
28. 470.lbm-483.xalancbmk-462.libquantum-482.sphinx3-parsec.rtview-473.astar-453.povray-482.sphinx3
29. 483.xalancbmk-437.leslie3d-parsec.swaptions-434.zeusmp-459.GemsFDTD-464.h264ref-433.milc-458.sjeng
30. 434.zeusmp-436.cactusADM-471.omnetpp-410.bwaves-483.xalancbmk-473.astar-464.h264ref-parsec.ferret
31. parsec.ferret-462.libquantum-433.milc-464.h264ref-parsec.dedup-parsec.ferret-436.cactusADM-444.namd
32. parsec.streamcluster-473.astar-462.libquantum-400.perlbench-401.bzip2-471.omnetpp-444.namd-parsec.swaptions
33. 465.tonto-482.sphinx3-433.milc-parsec.canneal-462.libquantum-462.libquantum-parsec.canneal-401.bzip2
34. 453.povray-parsec.rtview-459.GemsFDTD-464.h264ref-459.GemsFDTD-401.bzip2-parsec.streamcluster-462.libquantum
35. 434.zeusmp-410.bwaves-403.gcc-458.sjeng-458.sjeng-parsec.dedup-459.GemsFDTD-454.calculix
36. 434.zeusmp-483.xalancbmk-459.GemsFDTD-parsec.ferret-473.astar-434.zeusmp-445.gobmk-434.zeusmp
37. parsec.streamcluster-parsec.streamcluster-453.povray-403.gcc-458.sjeng-429.mcf-433.milc-437.leslie3d
38. parsec.canneal-parsec.fluidanimate-465.tonto-parsec.fluidanimate-parsec.rtview-parsec.swaptions-parsec.streamcluster-parsec.swaptions
39. 436.cactusADM-parsec.canneal-456.hmmer-parsec.fluidanimate-433.milc-400.perlbench-401.bzip2-456.hmmer
40. 454.calculix-473.astar-458.sjeng-parsec.streamcluster-parsec.blackscholes-450.soplex-433.milc-462.libquantum
41. 483.xalancbmk-483.xalancbmk-456.hmmer-429.mcf-parsec.swaptions-470.lbm-482.sphinx3-483.xalancbmk

42. parsec.bodytrack-444.namd-462.libquantum-462.libquantum-471.omnetpp-445.gobmk-454.calculix-435.gromacs
43. 462.libquantum-434.zeusmp-450.soplex-462.libquantum-464.h264ref-parsec.swaptions-462.libquantum-429.mcf
44. parsec.fluidanimate-462.libquantum-436.cactusADM-483.xalancbmk-464.h264ref-parsec.dedup-473.astar-462.libquantum
45. parsec.fluidanimate-462.libquantum-462.libquantum-445.gobmk-462.libquantum-459.GemsFDTD-470.lbm-parsec.ferret
46. parsec.streamcluster-462.libquantum-470.lbm-parsec.rtview-parsec.blackscholes-454.calculix-parsec.fluidanimate-410.bwaves
47. 459.GemsFDTD-401.bzip2-456.hmmer-parsec.ferret-433.milc-482.sphinx3-435.gromacs-459.GemsFDTD
48. 462.libquantum-429.mcf-400.perlbench-454.calculix-parsec.dedup-parsec.fluidanimate-parsec.ferret-465.tonto
49. 462.libquantum-465.tonto-450.soplex-parsec.rtview-parsec.swaptions-462.libquantum-parsec.fluidanimate-482.sphinx3
50. 410.bwaves-453.povray-parsec.bodytrack-436.cactusADM-471.omnetpp-462.libquantum-parsec.streamcluster-444.namd
51. parsec.blackscholes-403.gcc-410.bwaves-450.soplex-433.milc-429.mcf-462.libquantum-471.omnetpp
52. 462.libquantum-483.xalancbmk-parsec.streamcluster-471.omnetpp-433.milc-456.hmmer-444.namd-450.soplex
53. 462.libquantum-450.soplex-450.soplex-473.astar-parsec.canneal-437.leslie3d-433.milc-465.tonto
54. 471.omnetpp-471.omnetpp-471.omnetpp-483.xalancbmk-429.mcf-482.sphinx3-464.h264ref-444.namd
55. 471.omnetpp-471.omnetpp-403.gcc-403.gcc-403.gcc-436.cactusADM-465.tonto-parsec.bodytrack
56. 471.omnetpp-471.omnetpp-471.omnetpp-450.soplex-470.lbm-445.gobmk-parsec.rtview-parsec.swaptions
57. 462.libquantum-462.libquantum-450.soplex-403.gcc-437.leslie3d-482.sphinx3-401.bzip2-459.GemsFDTD
58. 471.omnetpp-471.omnetpp-471.omnetpp-parsec.canneal-429.mcf-459.GemsFDTD-459.GemsFDTD-parsec.swaptions
59. 462.libquantum-471.omnetpp-450.soplex-403.gcc-470.lbm-470.lbm-parsec.fluidanimate-parsec.streamcluster
60. 462.libquantum-462.libquantum-403.gcc-483.xalancbmk-parsec.canneal-437.leslie3d-435.gromacs-parsec.fluidanimate
61. 471.omnetpp-471.omnetpp-471.omnetpp-473.astar-473.astar-parsec.canneal-464.h264ref-parsec.dedup
62. 462.libquantum-462.libquantum-462.libquantum-450.soplex-450.soplex-436.cactusADM-433.milc-parsec.swaptions
63. 462.libquantum-462.libquantum-471.omnetpp-483.xalancbmk-483.xalancbmk-483.xalancbmk-436.cactusADM-434.zeusmp
64. 462.libquantum-462.libquantum-471.omnetpp-471.omnetpp-483.xalancbmk-459.GemsFDTD-410.bwaves-433.milc
65. 471.omnetpp-471.omnetpp-403.gcc-403.gcc-403.gcc-429.mcf-482.sphinx3-401.bzip2
66. 462.libquantum-462.libquantum-403.gcc-403.gcc-473.astar-473.astar-436.cactusADM-435.gromacs

67. 471.omnetpp-471.omnetpp-450.soplex-450.soplex-403.gcc-436.cactusADM-465.tonto-456.hmmer
68. 462.libquantum-471.omnetpp-471.omnetpp-403.gcc-483.xalancbmk-parsec.canneal-437.leslie3d-445.gobmk
69. 462.libquantum-462.libquantum-450.soplex-450.soplex-403.gcc-483.xalancbmk-470.lbm-parsec.bodytrack
70. 471.omnetpp-471.omnetpp-471.omnetpp-450.soplex-450.soplex-473.astar-459.GemsFDTD-445.gobmk
71. 462.libquantum-462.libquantum-462.libquantum-462.libquantum-450.soplex-483.xalancbmk-parsec.bodytrack-parsec.blackscholes
72. 462.libquantum-471.omnetpp-471.omnetpp-450.soplex-473.astar-473.astar-483.xalancbmk-parsec.fluidanimate
73. 462.libquantum-462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-429.mcf-436.cactusADM-parsec.dedup
74. 462.libquantum-462.libquantum-462.libquantum-462.libquantum-462.libquantum-parsec.canneal-444.namd-parsec.blackscholes
75. 471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-403.gcc-483.xalancbmk-410.bwaves-parsec.dedup
76. 462.libquantum-462.libquantum-462.libquantum-462.libquantum-403.gcc-473.astar-465.tonto-parsec.rtview
77. 462.libquantum-462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-403.gcc-parsec.canneal-parsec.swaptions
78. 462.libquantum-450.soplex-450.soplex-450.soplex-450.soplex-403.gcc-parsec.canneal-437.leslie3d
79. 462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-470.lbm-464.h264ref
80. 462.libquantum-462.libquantum-462.libquantum-462.libquantum-462.libquantum-403.gcc-473.astar-435.gromacs
81. 471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-parsec.canneal-429.mcf-435.gromacs
82. 462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-450.soplex-403.gcc-429.mcf-454.calculix
83. 462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-483.xalancbmk-483.xalancbmk-parsec.streamcluster
84. 462.libquantum-462.libquantum-403.gcc-403.gcc-403.gcc-473.astar-473.astar-483.xalancbmk
85. 471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-473.astar-482.sphinx3
86. 462.libquantum-462.libquantum-462.libquantum-471.omnetpp-471.omnetpp-450.soplex-450.soplex-465.tonto
87. 462.libquantum-462.libquantum-462.libquantum-462.libquantum-462.libquantum-450.soplex-450.soplex-429.mcf
88. 462.libquantum-462.libquantum-462.libquantum-462.libquantum-462.libquantum-462.libquantum-471.omnetpp-parsec.bodytrack
89. 462.libquantum-462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-parsec.canneal-parsec.dedup
90. 462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-403.gcc-470.lbm
91. 462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-459.GemsFDTD

92. 462.libquantum-462.libquantum-462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-483.xalancbmk-parsec.swaptions
93. 462.libquantum-462.libquantum-471.omnetpp-471.omnetpp-450.soplex-450.soplex-450.soplex-403.gcc
94. 471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-450.soplex-450.soplex-483.xalancbmk
95. 462.libquantum-462.libquantum-462.libquantum-462.libquantum-462.libquantum-471.omnetpp-483.xalancbmk-parsec.canneal
96. 462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-483.xalancbmk
97. 462.libquantum-462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-482.sphinx3
98. 462.libquantum-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-450.soplex-450.soplex
99. 471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-482.sphinx3
100. 471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-471.omnetpp-450.soplex-483.xalancbmk