ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Βελτίωση Ποιότητας Υπηρεσίας με Τεχνικές Διαμοιρασμού Κρυφής Μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Α. Παπαδάκης

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

Αθήνα, Οκτώβριος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Βελτίωση Ποιότητας Υπηρεσίας με Τεχνικές Διαμοιρασμού Κρυφής Μνήμης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Α. Παπαδάκης

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

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 6η Οκτωβρίου 2016

.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Δημήτριος Τσουμάκος
Επ. Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Επ. Καθηγητής Ιόνιο Πανεπιστήμιο

Αθήνα, Οκτώβριος 2016

.....
Ιωάννης Α. Παπαδάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός & Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Η παρούσα διπλωματική εργασία είναι το αποτέλεσμα της συνεργασίας μου με το Εργαστήριο Υπολογιστικών Συστημάτων από το Νοέμβριο του 2015 έως τον Οκτώβριο του 2016.

Για τη συνεργασία αυτή, καθώς και για την αδιάκοπη υποστήριξη, την αμέριστη εμπιστοσύνη, την υπομονή και την πολύτιμη βοήθεια αισθάνομαι την ανάγκη να εκφράσω τις θερμότερες ευχαριστίες μου στον επίκουρο καθηγητή κ. Γεώργιο Γκούμα, και το μεταδιδακτορικό ερευνητή κ. Κωνσταντίνο Νίκα, οι οποίοι ήταν ενεργά δίπλα μου καθ' όλη τη διάρκεια των 11 αυτών μηνών. Οι γνώσεις, οι ιδέες και οι προτάσεις τους αποτέλεσαν ανεκτίμητο οδηγό για την εκπόνηση αυτής της εργασίας. Επίσης θα ήθελα να ευχαριστήσω τον υποψήφιο διδάκτορα κ. Δημήτριο Σιακαβάρα για τη βοήθεια, την υπομονή και την υποστήριξη του καθ' όλη τη διάρκεια της συνεργασίας μας.

Χρωστώ ένα θερμό ευχαριστώ στους αγαπημένους μου φίλους και ιδιαίτερα στον Αλέξανδρο, το Λεωνίδα, τον Πέτρο και το Σπύρο για τα 6 υπέροχα και αξέχαστα χρόνια που περάσαμε μαζί, καθώς και για τις ευχάριστες αλλά και τις δύσκολες στιγμές που αντιμετωπίσαμε.

Τέλος, το μεγαλύτερο ευχαριστώ αλλά και η ευγνωμοσύνη μου δεν μπορεί παρά να ανήκει στους γονείς μου, χάρη στην απεριόριστη αγάπη και στήριξη των οποίων η φοίτηση μου στο Εθνικό Μετσόβιο Πολυτεχνείο ήρθε εις πέρας.

Γιάννης Α. Παπαδάκης

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Γραφημάτων

Ευρετήριο Πινάκων

Περίληψη

Η εξασφάλιση της ανεπηρέαστης λειτουργίας και η βελτίωση της ποιότητας υπηρεσίας εφαρμογών υψηλής προτεραιότητας είναι ένα αξιοπρόσεκτο ζήτημα, καθώς η συνεκτέλεση εφαρμογών στα σύγχρονα πολυπύρηνα συστήματα δημιουργεί ανταγωνισμό για τη χρήση των κοινών επεξεργαστικών πόρων, ο οποίος μπορεί να επηρεάσει αρνητικά την επίδοση των εφαρμογών και του συστήματος. Το ζήτημα αυτό αντιμετωπίζεται με τη δυνατότητα διαχείρισης των κοινόχρηστων πόρων, η οποία παρέχεται μέσω της ανάπτυξης των κατάλληλων μηχανισμών και τεχνολογιών.

Η διαχείριση του κοινόχρηστου τελευταίου επιπέδου της κρυφής μνήμης είναι εφικτή με τη χρήση των τεχνολογιών της Intel, Cache Monitoring Technology και Cache Allocation Technology, που προσφέρουν αντίστοιχα δυνατότητες παρακολούθησης χρήσης και καταμερισμού του τελευταίου επιπέδου κρυφής μνήμης μεταξύ των εφαρμογών ή των πυρήνων του συστήματος. Στην παρούσα διπλωματική εργασία μελετώνται οι δύο αυτές τεχνολογίες και παρουσιάζεται ένας δυναμικός μηχανισμός προστασίας, ο οποίος στοχεύει στη βελτίωση της ποιότητας υπηρεσίας, καθώς εξασφαλίζει την επίτευξη της μέγιστης δυνατής επίδοσης και τη μείωση της επιβράδυνσης για μία εφαρμογή υψηλής προτεραιότητας.

Η αξιολόγηση του μηχανισμού πραγματοποιείται στον επεξεργαστή Intel Xeon E5-2699 v4, ο οποίος περιέχει 22 πυρήνες και 55MB cache. Αποδεικνύεται ότι με τη χρήση του μηχανισμού εξασφαλίζεται η επίδοση της εφαρμογής που προστατεύεται, ενώ οι υπόλοιπες εφαρμογές που εκτελούνται στο σύστημα επωφελούνται την παροχή του χώρου στην cache που μένει αναξιοποίητος από την εφαρμογή υψηλής προτεραιότητας.

Λέξεις κλειδιά: Διαμοιρασμός κρυφής μνήμης, Καταμερισμός κρυφής μνήμης, Ποιότητα Υπηρεσίας, Εφαρμογή υψηλής προτεραιότητας, Διαχείριση κοινών πόρων, Intel CMT – CAT, Δυναμικός μηχανισμός προστασίας επίδοσης, Επιβράδυνση, Συνεκτέλεση εφαρμογών.

Abstract

Ensuring the unaffected execution and a high quality of service of high priority tasks is a remarkable issue, because co-running applications in modern multicore systems compete for the utilization of common resources, which can negatively influence the tasks' or the system's performance. Tackling this issue is possible through mechanisms and technologies that provide the ability of managing the system's common resources.

Managing the last level of the cache memory is feasible through Intel's Cache Monitoring Technology and Cache Allocation Technology, which respectively provide the ability to monitor the usage of the Last Level Cache and the ability to enforce allocation schemes throughout the tasks or the cores of the system. This Diploma Thesis introduces the aforementioned technologies and suggests a dynamic mechanism, whose goal is to enhance the quality of service by ensuring that one high priority task achieves the maximum possible performance, or the least deceleration.

The mechanism is evaluated through the use of Intel Xeon E5-2699 v4 processor, which consists of 22 processing cores and 55MB cache memory. The evaluation proves that the optimal performance is achieved, while the rest of the executed tasks benefit from the provided cache space that is unexploited by the high priority task.

Key words: Cache sharing, Cache allocation, Quality of service, High priority task, Common resource management, Intel CMT – CAT, Dynamic performance protecting mechanism, Slowdown, Co-running applications

1. Εισαγωγή

1.1 Πολυεπεξεργαστικά Συστήματα

Ενώ η ραγδαία πρόοδος της τεχνολογίας και της πληροφορικής συμβάλλει στην αύξηση της επίδοσης των παραγόμενων υπολογιστικών συστημάτων, οι ολοένα αυξανόμενες απαιτήσεις των χρηστών επιβάλλουν την περαιτέρω ανάπτυξη των συστημάτων, και τη συνεχή βελτίωση της παρεχόμενης ποιότητας υπηρεσίας. Αυτή η ανάγκη καλυπτόταν μέχρι τα τέλη της δεκαετίας του 1990 από υπολογιστές, οι οποίοι περιείχαν μία επεξεργαστική μονάδα. Καθώς όμως εμφανίστηκαν εμπόδια στην επιβεβαίωση του Νόμου του Moore [1], λόγω προσέγγισης κβαντικών αποστάσεων, κατανάλωσης ενέργειας και παραγωγής θερμότητας, η έννοια της παραλληλίας, που προϋπήρχε σε επίπεδα εντολής (Instruction Level Parallelism) και νημάτων (Simultaneous Multithreading), επεκτάθηκε και στο επίπεδο των πυρήνων, για να παραχθούν τα πολυεπεξεργαστικά συστήματα (Chip Multiprocessors, CMPs). Έτσι, φτάνουμε να μιλάμε σήμερα για εμπορικούς προσωπικούς υπολογιστές, αλλά και φορητές συσκευές με 2 και περισσότερους πυρήνες επεξεργασίας, αλλά, κυριότερα, για εξυπηρετητές (Servers), συγκροτήματα υπολογιστών (Computer Clusters) και κέντρα δεδομένων (Data Centers) με δεκάδες ή εκατοντάδες πυρήνες.

Καθώς τα πολυεπεξεργαστικά συστήματα εδραιώθηκαν ως τα προτιμότερα συστήματα για παροχή αυξημένης επίδοσης και διεκπεραιωτικής ικανότητας (throughput), μερικοί ερευνητές [2], [3], [4] εξέφρασαν την άποψη ότι τα CMPs θα αντιμετωπίσουν προκλήσεις στην ποιότητα παροχής υπηρεσιών (Quality of Service, QoS) λόγω του ανταγωνισμού για μοιραζόμενους πόρους [5]. Αυτοί είναι, μεταξύ άλλων, η κρυφή μνήμη (cache memory), η χρήση του διαύλου δεδομένων (bus), ο οποίος έχει περιορισμένο εύρος ζώνης (bandwidth) καθώς και άλλοι πόροι που δεν είναι ορατοί στο λογισμικό του συστήματος (λειτουργικά συστήματα, εικονικές μηχανές κ.α.). Ο ανταγωνισμός αυτός επηρεάζει την ικανότητα παροχής ντετερμινιστικής επίδοσης, καθώς και επιβολής προτεραιότητας μεταξύ των διαφορετικών εφαρμογών που συνεκτελούνται σε ένα μηχάνημα. Ως αποτέλεσμα, εφαρμογές που εκτελούνται μαζί με άλλες εργασίες σε ένα μηχάνημα, μπορεί να εμφανίσουν σημαντικές μεταπτώσεις στην παρατηρούμενη επίδοση. Για την καταπολέμηση αυτού του φαινομένου, ερευνητές έχουν αναλύσει το πρόβλημα και έχουν προτείνει λύσεις ([6], [7], [8], [9], [10], [11], [12]) για να επιτευχθεί η αποδοτική κατανομή των πόρων μεταξύ των εφαρμογών. Πολλές από αυτές τις εργασίες επικεντρώνονται στον ανταγωνισμό για τη μοιραζόμενη cache ως ένα καίριο πρόβλημα, καθώς έχει καθοριστική επίπτωση στον ντετερμινισμό της επίδοσης και στην ποιότητα της παρεχόμενης υπηρεσίας.

1.2 Προκλήσεις στα Πολυεπεξεργαστικά Συστήματα

Η βελτίωση της ποιότητας υπηρεσίας και η εξασφάλιση ντετερμινιστικής επίδοσης είναι προκλήσεις που αντιμετωπίζονται στην παροχή υπηρεσιών σε όλα τα εμπορικά πολυεπεξεργαστικά συστήματα. Από τις κινητές συσκευές, τους προσωπικούς σταθερούς και φορητούς υπολογιστές, μέχρι και servers, ή data centers, όλοι οι χρήστες ή διαχειριστές απαιτούν να εκτελέσουν παράλληλα ένα μεγάλο αριθμό εφαρμογών στα μηχανήματα τους, χωρίς όμως

να επηρεάζεται η επίδοση αυτών των εφαρμογών. Συγκεκριμένα, στις συσκευές προσωπικής χρήσης εκτελούνται παράλληλα πολλές εφαρμογές, με παραδείγματα εφαρμογές επεξεργασίας κειμένου, φυλλομετρητές ή τηλεφωνία με χρήση δικτυακών πρωτοκόλλων (Voice Over IP, VoIP). Από τις εφαρμογές αυτές αναμένεται να εκτελούνται χωρίς καθυστερήσεις, με όσο το δυνατό καλύτερη επίδοση, για να εξασφαλιστεί η βέλτιστη εμπειρία χρήσης. Επίσης στα συστήματα εξυπηρέτησης, δηλαδή σε servers ή data centers, στα οποία είναι πιθανό να εκτελούνται εικονικές μηχανές (Virtual Machines, VMs), επιστημονικές εφαρμογές, και διεργασίες με μεγάλο επεξεργαστικό φόρτο, οι πελάτες που εμπιστεύονται την εκτέλεση των εργασιών τους σε αυτά τα περιβάλλοντα απαιτούν την επίτευξη της μέγιστης επίδοσης. Από την άλλη πλευρά, οι διαχειριστές των συστημάτων αυτών στοχεύουν στη μεγιστοποίηση της χρησιμοποίησης των διαθέσιμων πόρων τους, αλλά και στη βέλτιστη επίδοση των εφαρμογών που εκτελούνται, ώστε, μέσω της παροχής υψηλής ποιότητας υπηρεσιών, να αυξηθεί το οικονομικό τους κέρδος.

Όλες οι προηγούμενες απαιτήσεις των χρηστών πολυεπεξεργαστικών συστημάτων αντιμετωπίζουν το πρόβλημα της υποβάθμισης της επίδοσης λόγω του ανταγωνισμού μεταξύ των εφαρμογών για τη χρήση των μοιραζόμενων υπολογιστικών πόρων. Η έλλειψη της δυνατότητας διαχείρισης των μοιραζόμενων πόρων οδηγεί στην απαγόρευση της συνεκτέλεσης εφαρμογών με υψηλές απαιτήσεις για ελαχιστοποίηση της καθυστέρησης (Latency Critical Tasks, LC), ή εφαρμογών που λογίζονται ως υψηλής προτεραιότητας (high priority tasks) με άλλες εφαρμογές. Αυτό το φαινόμενο οδηγεί, με τη σειρά του, στην υποχρησιμοποίηση των συστημάτων που συνεπάγεται αυξημένο λειτουργικό κόστος. Η αντιμετώπιση του ζητήματος αυτού είναι απαραίτητη για τους χρήστες και τους διαχειριστές των πολυεπεξεργαστικών συστημάτων. Η διαχείριση των μοιραζόμενων πόρων μπορεί να πραγματοποιηθεί μέσω αλλαγών στο υλικό (hardware), παροχής υποστήριξης στο Λειτουργικό Σύστημα (Operating System, O.S.), ανάπτυξης μηχανισμών λογισμικού (software), ή συνδυασμού των προηγουμένων.

1.3 Εισαγωγή στην Intel CMT – CAT

Η κρυφή μνήμη (cache memory), και συγκεκριμένα το τελευταίο της επίπεδο (Last Level Cache, LLC) είναι ένας από τους πόρους για τους οποίους ανταγωνίζονται οι εκτελούμενες εφαρμογές σε ένα σύστημα, καθώς είναι μοιραζόμενη από τους πυρήνες ενός επεξεργαστή. Η παρακολούθηση της χρήσης καθώς και η διαχείριση της κατανομής της LLC απασχόλησε την Intel, η οποία ανέπτυξε τις αντίστοιχες τεχνολογίες Intel Cache Monitoring Technology (CMT) και Intel Cache Allocation Technology (CAT). [5]

Η παρούσα διπλωματική εργασία μελετά τις τεχνολογίες Intel CMT – CAT και τις πιθανές χρήσεις τους για τη βελτίωση της επίδοσης των εφαρμογών που εκτελούνται σε ένα σύστημα και αξιολογεί στην πράξη τις δύο τεχνολογίες με τη χρήση του εργαλείου PQoS (που χρησιμοποιεί τις τεχνολογίες). Επίσης αναλύει και υλοποιεί ένα μηχανισμό, ο οποίος συνδυάζοντας τις δύο τεχνολογίες, εξασφαλίζει την επίδοση μιας εφαρμογής που χαρακτηρίζεται ως υψηλής προτεραιότητας όσο το δυνατόν εγγύτερα στη βέλτιστη κατάσταση. Η λειτουργία του μηχανισμού γίνεται με δυναμικό τρόπο, χωρίς την απαίτηση για οποιαδήποτε πληροφορία για τις εφαρμογές που εκτελούνται στο σύστημα.

2. Μέθοδοι Διαχείρισης της Κρυφής Μνήμης

2.1 Επίδραση του Ανταγωνισμού στην Επίδοση

Στα σύγχρονα πολυεπεξεργαστικά συστήματα, οι πόροι όπως η κύρια μνήμη, ο δίαυλος δεδομένων, τα ανώτερα επίπεδα της cache, η πρόσβαση στον επεξεργαστή για χρόνο εκτέλεσης και η πρόσβαση στο δίκτυο είναι μοιραζόμενοι μεταξύ των πυρήνων και ως εκ τούτου μεταξύ των εφαρμογών που εκτελούνται. Το γεγονός αυτό καθιστά αναπόφευκτο τον ανταγωνισμό μεταξύ των εφαρμογών για τη χρήση τους, και είναι φυσικό ότι αυτό το φαινόμενο οδηγεί κάποιες εφαρμογές στην έλλειψη των απαραίτητων πόρων για να επιτύχουν τη βέλτιστη δυνατή επίδοση. Αυτή η μείωση της επίδοσης των εφαρμογών αλλά και συνολικά του συστήματος προκαλείται από την απουσία της δυνατότητας για διαχείριση των πόρων, η οποία είναι επίσης η αιτία για την άδικη κατανομή των πόρων, ακόμα και «λιμοκτονία» (starvation) των εφαρμογών για απαραίτητους πόρους. Παρόλα αυτά, η διαχείριση όλων των πιθανών κοινόχρηστων πόρων κατά το χρόνο εκτέλεσης είναι μια ακριβή διαδικασία για το σύστημα, και για αυτό το λόγο επιλέγεται η διαχείριση των πόρων στους οποίους ο ανταγωνισμός έχει τις σοβαρότερες επιπτώσεις στην επίδοση των εφαρμογών.

Σε αυτούς τους πόρους συγκαταλέγονται η cache και η χρήση του διαύλου δεδομένων. Ο ανταγωνισμός για τη χρήση του κοινόχρηστου μέρους της cache, που είναι η LLC, μπορεί να οδηγήσει σε σημαντική μείωση της επίδοσης των εφαρμογών, καθώς στην cache αποθηκεύονται δεδομένα των εκτελούμενων εφαρμογών ώστε να παρέχεται η γρήγορη πρόσβαση σε αυτά. Επειδή όμως το μέγεθος της cache είναι μικρό, και οι εκτελούμενες εφαρμογές είναι πολυάριθμες, είναι πολύ πιθανό να εξάγονται και να αντικαθίστανται διαρκώς δεδομένα, με αποτέλεσμα να αναζητούνται πλέον στην κύρια μνήμη, μια διαδικασία που είναι πολύ πιο χρονοβόρα. Η καθυστέρηση αυτή έχει σημαντική επίπτωση στην επίδοση των εκτελούμενων εφαρμογών.

Η προσθήκη της δυνατότητας διαχείρισης του καταμερισμού της cache δεν είναι μια πρόσφατη ιδέα, αλλά ξεκινά από τις αρχές της προηγούμενης δεκαετίας με προτάσεις που υλοποιούνται με τη χρήση μηχανισμών στο υλικό, και την παροχή της δυνατότητας αποφάσεων και προβλέψεων στο scheduler του O.S. Οι προτάσεις που αφορούν την ενεργοποίηση του hardware βασίζονται στον καταμερισμό της cache σύμφωνα με διάφορες πολιτικές και προτάσεις εναλλακτικών πολιτικών εισαγωγής και αντικατάστασης. Επίσης, υπάρχουν προτάσεις για την αντιμετώπιση του προβλήματος με μηχανισμούς μεταβολής του χρονισμού του ρολογιού του συστήματος. Οι προτάσεις που αφορούν τη δυνατότητα αντιμετώπισης της υποβάθμισης της επίδοσης με τη χρήση του χρονοδρομολογητή του O.S. βασίζονται στη δημιουργία ζευγαριών συνεκτέλεσης, στα οποία οι επιπτώσεις από τον ανταγωνισμό δεν είναι σημαντικές. Είναι εμφανές ότι οι δύο αυτές διαφορετικές προσεγγίσεις δεν είναι αλληλοαποκλειόμενες, επομένως είναι δυνατός ο συνδυασμός τους, για την εκμετάλλευση των πλεονεκτημάτων των δύο προσεγγίσεων.

2.2 Αντιμετώπιση με Χρήση του Scheduler

Η δυνατότητα πρόβλεψης της επίδοσης με χρήση μετρικών όπως οι εντολές ανά κύκλο μηχανής (Instructions per Cycle, IPC) ή το ποσοστό αστοχίας (miss rate) προσδίδει στον scheduler τη δυνατότητα να δρα προληπτικά στο ζήτημα του χωρισμού της cache μεταξύ των εφαρμογών, ώστε να αποφευχθεί η πτώση της επίδοσης του συστήματος.

Ο διαμοιρασμός της cache επηρεάζει τα νήματα εκτέλεσης ποικιλοτρόπως, καθώς κάποια νήματα είναι πιθανό να επιβραδυνθούν σημαντικά, ενώ άλλα όχι. Το O.S. οφείλει να γνωρίζει και να αποφεύγει προβλήματα όπως υποβέλτιστο throughput, υπερκατανάλωση της cache, και λιμοκτονία νημάτων για χώρο στην cache. Με βάση αυτά, οι D. Chandra, F. Guo, S. Kim, Y. Solihin [2] πρότειναν ένα μοντέλο επίδοσης που προβλέπει τις επιπτώσεις του διαμοιρασμού της cache σε νήματα που συνεκτελούνται. Το μοντέλο προβλέπει τον αριθμό των επιπλέον αστοχιών στη LLC για κάθε νήμα που τη μοιράζεται. Η αξιολόγηση του μοντέλου σε προσομοίωση διπύρηνης αρχιτεκτονικής οδήγησε σε μέση λανθασμένη πρόβλεψη 3.8%.

Οι χωρητικότητες των μνημών που περιέχονται στα σύγχρονα συστήματα εξυπηρετητών κυμαίνονται σε εκατοντάδες GB. Αυτό καθιστά δυνατή την ανάπτυξη εφαρμογών οι οποίες αποθηκεύουν όλα τα δεδομένα τους στην κύρια μνήμη (παράδειγμα οι in-memory databases). Επίσης, είναι λογική η πρόβλεψη ότι οι εφαρμογές, που αναπτύσσονται και αξιολογούνται σε μηχανήματα με 16 ή 24 πυρήνες, θα εκτελούνται στο μέλλον σε μηχανήματα με πολλαπλάσιους πυρήνες. Επομένως τίθεται το ερώτημα της μεταβολής της συμπεριφοράς μιας εφαρμογής, λόγω της μεταφοράς της σε ένα τέτοιο σύστημα.

Το ζήτημα της επεκτασιμότητας μιας εφαρμογής σε ένα μηχάνημα με πολλές επεξεργαστικές μονάδες, δεδομένης της επίδοσης της σε ένα μικρό μηχάνημα με λίγους πυρήνες, αντιμετωπίζεται από το εργαλείο ESTIMA (Extrapolating ScalabiliTy of In-Memory Applications) [13]. Με ελάχιστη είσοδο από το χρήστη, το εργαλείο αυτό είναι σε θέση να προβλέπει την επεκτασιμότητα μιας παράλληλης in-memory εφαρμογής, χωρίς να είναι απαραίτητες οι λεπτομέρειες της εφαρμογής ή του μηχανήματος στο οποίο θα εκτελεστεί. Το εργαλείο μετρά τους stalled (στάσιμους) κύκλους σε λίγους πυρήνες και προεκτείνει το αποτέλεσμα σε περισσότερους πυρήνες. Η εφαρμογή του εργαλείου είναι εύκολη σε κάθε in-memory application, αφού, αντίθετα με άλλες προσεγγίσεις για τη λύση του ίδιου προβλήματος, το ESTIMA δεν απαιτεί πληροφορίες σχετικές με την εκάστοτε εφαρμογή, οι οποίες πιθανώς να μην είναι διαθέσιμες.

Ο υπολογισμός του ποσοστού επιβράδυνσης κάθε εφαρμογής συναρτήσει των εφαρμογών με τις οποίες συνεκτελείται μπορεί να συμβάλει στην επίλυση του ζητήματος αυτού, μέσω επανατοποθετήσεων των εφαρμογών, έτσι ώστε να επιτυγχάνονται ζεύγη με μικρή αλληλεπίδραση στην απόδοσή τους. Η διαδικασία αυτή μπορεί να πραγματοποιηθεί εκτελώντας όλους τους πιθανούς συνδυασμούς, με πολυπλοκότητα $O(N^2)$. Με εκατοντάδες ή χιλιάδες εφαρμογές να εκτελούνται σε ένα data center, και δεδομένων των συχνών αναβαθμίσεων των εφαρμογών αυτών, η εξαντλητική προσέγγιση καθίσταται μη πρακτική.

Η μεθοδολογία Bubble – Up [10] αποτελεί μια προτεινόμενη λύση σε αυτό το πρόβλημα. Με τη χρήση μιας «φυσαλίδας», για την εφαρμογή μιας μεταβλητής «πίεσης» στη μνήμη, η μεθοδολογία αυτή είναι σε θέση να προβλέψει την παρεμβολή στην επίδοση με ακρίβεια σχεδόν 99% της πραγματικής τιμής. Έτσι προκύπτει ένα προφίλ για κάθε εφαρμογή που περιέχει αφενός μια καμπύλη ευαισθησίας (πόσο επηρεάζεται η εφαρμογή) και αφετέρου ένα βαθμό πίεσης (πόσο επηρεάζει). Δεδομένων δύο εφαρμογών Α και Β, η μεθοδολογία Bubble – Up προβλέπει την επίπτωση στην επίδοση της Α όταν τοποθετείται μαζί με τη Β, απλώς βρίσκοντας το βαθμό πίεσης της Β στην καμπύλη της Α. Η δημιουργία αυτών των προφίλ έχει, σε αυτήν την περίπτωση, πολυπλοκότητα $O(N)$. Χρησιμοποιώντας τα προφίλ των εφαρμογών, είναι δυνατό να δημιουργηθούν έξυπνες συνεγκαταστάσεις με σκοπό να ελαχιστοποιηθούν οι παρεμβολές μεταξύ των συνεκτελούμενων διεργασιών, το οποίο δρα ευεργετικά στην επίδοση του συστήματος.

Η μελέτη των επιπτώσεων του διαμοιρασμού της cache σε εφαρμογές data center της Google οδήγησε τους L. Tang, J. Mars, N. Vachharajani, R. Hundt και M. L. Soffa [14] στη διαπίστωση της σπουδαιότητας των αντιστοιχίσεων νημάτων με πυρήνες, καθώς, μέσω αυτών, είναι δυνατό να ελεγχθεί το ποια νήματα μοιράζονται την cache και το δίαυλο δεδομένων (εάν για παράδειγμα τοποθετηθούν σε διαφορετικό socket, το οποίο διαθέτει τη δική του cache).

Εκτός αυτού, τονίζουν ότι η βέλτιστη αντιστοίχιση για κάθε εφαρμογή διαφέρει συναρτήσει των εφαρμογών που έχουν επιλεχθεί για συνεκτέλεση. Τα χαρακτηριστικά που επηρεάζουν την επίδοση στα διάφορα σενάρια αντιστοιχίσεων νημάτων – πυρήνων είναι, μεταξύ άλλων, το μέγεθος των δεδομένων που μοιράζονται τα νήματα, η χρήση του διαύλου δεδομένων, και το αποτύπωμα (footprint) της εφαρμογής στην cache. Αυτά τα χαρακτηριστικά χρησιμοποιούνται από έναν αλγόριθμο που υπολογίζει αντιστοιχίσεις νημάτων – πυρήνων, με δύο προσεγγίσεις, μια ευριστική, και μια προσαρμοζόμενη, η οποία «μαθαίνει» σε χρόνο εκτέλεσης τις κατάλληλες αντιστοιχίσεις.

Παρόμοιες ανησυχίες ενέπνευσαν και τη δημιουργία του CAMP [15], ενός μοντέλου επίδοσης, το οποίο κάνει χρήση ιστογραμμάτων συχνοτήτων πρόσβασης στην cache και τη σχέση που διέπει το miss rate με το throughput κάθε διεργασίας με σκοπό να προβλέψει το χώρο που χρειάζεται η διεργασία στην cache όταν τη μοιράζεται με τις συνεκτελούμενες διεργασίες. Με αυτήν την πρόβλεψη, μπορεί να εξαχθεί μια εκτίμηση για την υποβάθμιση της επίδοσης λόγω της διαμάχης για χώρο στην cache μεταξύ των διεργασιών που εκτελούνται σε ένα CMP. Επίσης το CAMP προσφέρει έναν αυτοματοποιημένο τρόπο να εξαχθούν τα απαραίτητα ιστογράμματα, χωρίς εκτέλεση της εκάστοτε εφαρμογής offline, μετατροπές στο λειτουργικό σύστημα ή επιπλέον υλικό. Η αξιολόγηση του CAMP έγινε με χρήση benchmarks της σουίτας SPEC2000 σε διπύρηνο μηχάνημα, και το μέσο σφάλμα πρόβλεψης ήταν 1,57%.

Η ελλιπής απομόνωση της επίδοσης μιας εφαρμογής που εκτελείται σε ένα CMP αφορά την εξάρτηση της επίδοσης της συγκεκριμένης εφαρμογής από τη συμπεριφορά των συνεκτελούμενων εφαρμογών. Η εξάρτηση αυτή δημιουργείται από τον άδικο καταμερισμό της μοιραζόμενης cache, και είναι η ρίζα αρκετών προβλημάτων. Πρώτον, προκαλείται μη ντετερμινιστική

συμπεριφορά του scheduler, η οποία οδηγεί σε ελαττωμένο έλεγχο της επιθυμητής προτεραιότητας των εφαρμογών. Επίσης, η κοστολόγηση με βάση το χρόνο στη CPU (per-CPU-hour billing) περιπλέκεται, καθώς, εάν μια εφαρμογή επιβραδύνεται από μια επιθετική συνεκτελούμενη της, είναι άδικη η χρέωση ολόκληρου του χρόνου για αυτήν την εφαρμογή. Λόγω αυτού του φαινομένου, η εκπλήρωση του QoS, που επιτυγχάνεται μέσω παροχής πόρων, όπως ένα μέρος CPU cycles, γίνεται δυσκολότερη.

Μια προτεινόμενη λύση για τα προηγούμενα ζητήματα είναι η σχεδίαση ενός νέου «cache-fair» αλγορίθμου χρονοδρομολόγησης [18]. Αυτός ο αλγόριθμος εξασφαλίζει ότι μια εφαρμογή θα εκτελείται το ίδιο γρήγορα, όπως υπό συνθήκες δίκαιου καταμερισμού της cache, χωρίς όμως να παίζει ρόλο ο πραγματικός της καταμερισμός. Αυτό επιτυγχάνεται μέσω του ελέγχου των κβάντων χρόνου που μοιράζει ο αλγόριθμος, ρυθμίζοντας μέσω αυτών τη μεταβλητότητα που παρουσιάζεται στο IPC. Συγκεκριμένα, όταν το πραγματικό IPC ενός νήματος είναι μικρότερο από το IPC υπό δίκαιο καταμερισμό της cache («δίκαιο» IPC) τότε αυξάνεται το κβάντο χρόνου του νήματος. Αντίθετα, όταν το πραγματικό IPC είναι μεγαλύτερο από το δίκαιο IPC τότε το κβάντο χρόνου μειώνεται. Η αξιολόγηση αυτού του αλγορίθμου με χρήση benchmarks έδειξε ότι η μεταβλητότητα των εφαρμογών μειώθηκε από 28% σε 4%, με αμελητέο overhead στον scheduler.

Η δικαιοσύνη στον καταμερισμό της cache είναι ένα χαρακτηριστικό, το οποίο είναι κρίσιμο να βελτιστοποιηθεί, καθώς ο scheduler του O.S αφήνει στο hardware τη δικαιοσύνη στην cache μεταξύ των συνεκτελούμενων νημάτων. Ενώ το O.S επιβάλλει την προτεραιότητα μεταξύ των εργασιών μέσω των κβάντων χρόνου που διαθέτει, βασίζεται στην υπόθεση ότι σε ένα κβάντο χρόνου ο ρυθμός επίδοσης όλων των νημάτων προς εκτέλεση επηρεάζεται ομοιόμορφα, κάτι που είναι πιθανό να μην ισχύει, διότι η δυνατότητα ενός νήματος να διεκδικήσει χώρο στην cache καθορίζεται από τη χρονική συμπεριφορά επαναχρησιμοποίησης των δεδομένων του.

Όταν αυτή η υπόθεση δεν ισχύει, ο scheduler αντιμετωπίζει το πρόβλημα της εξάρτησης του ρυθμού προόδου ενός νήματος από το σύνολο των συνεκτελούμενων νημάτων, από το οποίο προκύπτουν τα εξής δύο προβλήματα. Το πρώτο πρόβλημα είναι η λιμοκτονία ενός νήματος, που συμβαίνει όταν ένα νήμα αποτυγχάνει να καταλάβει τον απαραίτητο χώρο στην cache για να έχει πρόοδο. Το δεύτερο είναι η αντιστροφή προτεραιότητας, κατά την οποία ένα νήμα υψηλής προτεραιότητας έχει πιο αργό ρυθμό προόδου από ένα νήμα χαμηλής προτεραιότητας παρά το γεγονός ότι ο scheduler του παρέχει περισσότερα κβάντα χρόνου για να εκτελεστεί. Αυτό συμβαίνει διότι το χαμηλής προτεραιότητας νήμα κερδίζει το υψηλής προτεραιότητας νήμα στον ανταγωνισμό για περισσότερο χώρο στην cache. Το φαινόμενο επιδεινώνεται καθώς το O.S. δεν έχει τρόπο να ενημερωθεί και επομένως να επιλύσει αυτήν την κατάσταση.

Για την αντιμετώπιση αυτών των προβλημάτων, το hardware πρέπει να παρέχει δίκαιη κατανομή της cache, δηλαδή ένα σχήμα το οποίο εξασφαλίζει το ότι οι επιπτώσεις του διαμοιρασμού της cache θα είναι ομοιόμορφες για όλα τα συνεκτελούμενα νήματα. Αυτό μπορεί να επιτευχθεί από ένα στατικό και ένα δυναμικό αλγόριθμο καταμερισμού της LLC, οι οποίοι είχαν προταθεί από τους S. Kim, D. Chandra και Y. Solihin [4] το 2004 και εξασφαλίζουν τη δίκαιη

κατανομή. Ο δυναμικός αλγόριθμος είναι εύκολος στην υλοποίηση, απαιτεί ελάχιστη πρότερη γνώση για το προφίλ των εφαρμογών, έχει μικρό κόστος και δεν περιορίζει την πολιτική αντικατάστασης στην LRU. Ο στατικός αλγόριθμος, παρόλο που απαιτεί την διατήρηση πληροφορίας σε στοίβα από την LRU, μπορεί να βοηθήσει το O.S να αποφύγει την υπερχρησιμοποίηση (thrashing) της cache. Είναι σημαντικό να τονιστεί η σχέση μεταξύ throughput και δικαιοσύνης στην cache, καθώς η εξασφάλιση δικαιοσύνης αυξάνει το throughput, ενώ το αντίθετο δεν ισχύει απαραίτητα, καθώς το throughput μπορεί να αυξηθεί σε βάρος της δικαιοσύνης στην cache.

Μια ερμηνεία της βελτίωσης της παρεχόμενης υπηρεσίας είναι η εξασφάλιση του υψηλού QoS για εφαρμογές με υψηλή προτεραιότητα, που πιθανόν να είναι εφαρμογές όπως διαδικτυακή αναζήτηση, χάρτες, λογισμικό-ως-υπηρεσία, κοινωνικά δίκτυα, ή latency critical εφαρμογές και έχουν υψηλό στόχο επιπέδου υπηρεσίας (Service Level Objective, SLO).

Έτσι, οι πολιτικές που σχεδιάζονται με βάση την προηγούμενη λογική έχουν ως κύριο στόχο να βελτιώσουν την επίδοση των εφαρμογών με κριτήριο την προτεραιότητα που τους έχει ανατεθεί [8]. Αυτές οι πολιτικές διαφέρουν από τις προηγούμενες από την άποψη της ιεράρχησης του σκοπού (στόχοι για τις εφαρμογές υψηλής προτεραιότητας και περιορισμοί για τις εφαρμογές χαμηλής προτεραιότητας) τις μετρικές που χρησιμοποιούνται και τη φύση της εκχώρησης των πόρων (στατική αντί δυναμική). Από την αξιολόγηση αυτών των πολιτικών προκύπτει ότι μια αρχιτεκτονική μνήμης που λαμβάνει υπόψη της το QoS είναι σε θέση να βελτιώσει σημαντικά την επίδοση της εφαρμογής υψηλής προτεραιότητας ακόμα και με την παρουσία άλλων εφαρμογών στο σύστημα. Επίσης, αποδεικνύεται ότι οι δυναμικές πολιτικές έχουν ιδιαίτερη σημασία σε αυτές τις αρχιτεκτονικές, καθώς επιτρέπουν στο hardware να χειριστεί την εκχώρηση πόρων βασιζόμενο σε πραγματικές αλλαγές και περιορισμούς στην επίδοση.

Παράδειγμα δομής, η οποία λαμβάνει αποφάσεις για την προτεραιότητα των εφαρμογών, αποτελεί το CQoS [3]. Η συγκεκριμένη δομή περιέχει μηχανισμούς για εκχώρηση και επιβολή προτεραιότητας. Πρώτο βήμα, όμως, αποτελεί η ταξινόμηση των εφαρμογών, δηλαδή η αναγνώριση της ετερογένειας στην πρόσβαση στη μνήμη μεταξύ των εφαρμογών και η τοποθέτηση τους στα υποστηριζόμενα επίπεδα προτεραιότητας. Οι μηχανισμοί για την επιβολή της προτεραιότητας είναι, πρώτον, η επιλεκτική εκχώρηση της cache, σύμφωνα με την οποία η πολιτική εκχώρησης ή αφαίρεσης μέρους της cache εξαρτάται από την προτεραιότητα της αιτούμενης εφαρμογής, δεύτερον, ο στατικός ή δυναμικός καταμερισμός των sets στα οποία χωρίζεται η cache, κατά τον οποίο οι εφαρμογές υψηλής προτεραιότητας καταλαμβάνουν περισσότερα ways στην cache, και τρίτον, οι ετερογενείς περιοχές της cache, που αφορά την πρόσβαση σε victim caches ή stream buffers.

2.3 Μηχανισμοί Αντιμετώπισης σε Επίπεδο Υλικού

Η συμβατική πολιτική αντικατάστασης LRU μπορεί να υποβιβάσει σημαντικά την επίδοση της cache σε συνθήκες ανταγωνισμού μεταξύ εφαρμογών με διαφορετικές απαιτήσεις για μνήμη. Η πολιτική δυναμικής εισαγωγής (Dynamic Insertion Policy, DIP) [17], που είχε προταθεί για μονοεπεξεργαστικά συστήματα, αγνοεί τα χαρακτηριστικά των εφαρμογών,

και, λόγω αυτού, δεν προσφέρει μια αποδεκτή λύση στο ζήτημα. Η επέκτασή της όμως σε Thread-Aware Dynamic Insertion Policy [9], ώστε να λαμβάνονται υπόψη οι απαιτήσεις για μνήμη κάθε μίας από τις συνεκτελούμενες εφαρμογές προσφέρει σημαντική βελτίωση στην επίδοση, καθώς, μετά από αξιολόγηση, προκύπτει ότι το συνολικό throughput ενός συστήματος με cache που ακολουθεί την εν λόγω πολιτική αυξάνεται σε σχέση με την περίπτωση χρήσης της συμβατικής LRU πολιτικής. Τα οφέλη της χρήσης της TADIP ομοιάζουν με τα οφέλη του διπλασιασμού του μεγέθους μιας cache που ακολουθεί LRU, ενώ το overhead είναι σχεδόν αμελητέο, με χωρικό κόστος λιγότερο από 2 bytes ανά πυρήνα.

Το ζήτημα των μετρικών που χρησιμοποιούνται ώστε να αξιολογηθεί η επίδοση του συστήματος απασχόλησε τους R. Iyer, L. Hsu, S. Reinhardt και S. Makineni [7], οι οποίοι πρότειναν τρεις πολιτικές καταμερισμού της cache, δανειζόμενοι όρους της οικονομικής θεωρίας. Έτσι προκύπτουν η «κομμουνιστική» πολιτική, που στοχεύει στη μεγιστοποίηση της δικαιοσύνης μεταξύ των νημάτων, την «ωφελιμιστική» πολιτική, στόχος της οποίας είναι να μεγιστοποιήσει το συνολικό κέρδος για το σύστημα, για παράδειγμα την αύξηση του throughput, και, τέλος, η «καπιταλιστική» πολιτική, η οποία αφήνει την cache ελεύθερη για όλα τα νήματα, και είναι η πολιτική που εφαρμόζεται από τα περισσότερα συστήματα. Για να αξιολογηθούν αυτές οι πολιτικές αλλά και γενικότερα η επίδοση του συστήματος, πρέπει να αποφασιστεί πρώτα ποιες μετρικές θα ληφθούν υπόψη. Αυτές οι μετρικές μπορεί να είναι το miss rate, το IPC ή η χρήση bandwidth, συμπεριλαμβανομένων τόσο απόλυτων όσο και σχετικών τιμών για κάθε μία. Μετά από χρήση αναλυτικών μοντέλων, αλλά και προσομοίωσης, προέκυψε ότι ο βέλτιστος καταμερισμός της cache ποικίλει ανάλογα με τις διαφορετικές ερμηνείες που μπορούν να αποδοθούν στη βελτιστότητα. Επίσης προκύπτει ότι, ενώ οι στόχοι της «κομμουνιστικής» και της «ωφελιμιστικής» πολιτικής είναι συμβατοί, υπάρχουν περιπτώσεις για κάθε πολιτική, στις οποίες αποτυγχάνει να προσφέρει βέλτιστη επίδοση ή δικαιοσύνη, αντίστοιχα.

Στα προηγούμενα καταδείχθηκε η σημασία της δικαιοσύνης στον καταμερισμό της cache, που μεταφράζεται σε ισοκατανομή των ωφελειών και των επιπτώσεων σε όλες τις εργασίες που χρησιμοποιούν την cache. Έχει, όμως, αξία να ερευνηθεί και μια διαφορετική προσέγγιση στην έννοια της δικαιοσύνης, σύμφωνα με την οποία τα οφέλη που λαμβάνει μια εφαρμογή από τους πόρους της cache που χρησιμοποιεί είναι πιθανό να μη συνδέονται άμεσα με τις απαιτήσεις αυτής της εφαρμογής. Για παράδειγμα, μια εφαρμογή ροϊκής πρόσβασης δεδομένων (streaming) απαιτεί μεγάλο μέρος της cache, με μικρή όμως πιθανότητα να επαναχρησιμοποιηθούν αρκετά από τα δεδομένα που αποθηκεύονται. Επομένως, η παραχώρηση μεγάλου μέρους της cache δε θα έχει θετική επίπτωση στην επίδοση μιας εφαρμογής τέτοιου τύπου. Επομένως έχει νόημα να χωρίζεται η cache με κριτήριο τα πιθανά οφέλη της εκάστοτε εφαρμογής από αυτήν (το οποίο μπορεί να μεταφραστεί σε μείωση των cache misses), αντί για την απαίτηση που εμφανίζει. Έτσι προκύπτει ένας μηχανισμός καταμερισμού της cache, που ονομάζεται Utility-based Cache Partitioning (UCP) [11], και υλοποιεί την προαναφερθείσα πολιτική. Η παρακολούθηση των εφαρμογών από το μηχανισμό αυτό γίνεται κατά το χρόνο εκτέλεσης από ένα κύκλωμα hardware το οποίο απαιτεί λιγότερο από 2kB μνήμης. Η αξιολόγηση του μηχανισμού αυτού έδειξε πως η επίδοση ενός διπύρηνου μηχανήματος

βελτιώνεται μέχρι και 23%, και κατά μέσο όρο 11% σε σχέση με μηχανισμούς καταμερισμού που βασίζονται στην πολιτική LRU.

Παράλληλα με τη διαχείριση της επίδοσης των διαφορετικών διεργασιών που εκτελούνται σε ένα CMP και ανταγωνίζονται για χώρο στην cache καθώς και στην κύρια μνήμη, μία ακόμα πρόκληση της εποχής των CMP είναι αναμφισβήτητα η διαχείριση της ενέργειας που καταναλώνεται σε ένα σύστημα. Για την αντιμετώπιση του τελευταίου ζητήματος έχουν προταθεί τεχνικές περιορισμού της κατανάλωσης ενέργειας, οι οποίες μεταβάλλουν το χρονισμό του ρολογιού του συστήματος ώστε να εξοικονομήσουν ενέργεια.

Οι κυριότερες τεχνικές που χρησιμοποιούνται για αποδοτική διαχείριση ενέργειας είναι η Dynamic Voltage and Frequency Scaling (DVFS) και η διαμόρφωση ρολογιού [6]. Η τεχνική κλιμάκωσης συχνότητας DVFS επιτρέπει στο O.S. να βελτιώσει την επίδοση του ή να εξοικονομήσει ενέργεια μέσω της μεταβολής της συχνότητας ή της τάσης λειτουργίας. Στη διαμόρφωση ρολογιού, το ρολόι λειτουργίας πυροδοτείται από ένα ρολόι πύλης για να δημιουργήσει φάσεις εκτέλεσης και αναμονής σε κάθε πυρήνα.

Η λειτουργία αυτών των τεχνικών είναι δυνατό να επεκταθεί και στην αντιμετώπιση του πρώτου ζητήματος, δηλαδή τη διαχείριση των θεμάτων που έχουν να κάνουν με το QoS στην cache ή στην κύρια μνήμη. Η κεντρική ιδέα πίσω από αυτό είναι η επιβράδυνση του ρυθμού επεξεργασίας ενός πυρήνα στην περίπτωση που εκτελεί μια εργασία χαμηλής προτεραιότητας που παρεμβάλλεται με την εκτέλεση μιας εργασίας υψηλής προτεραιότητας λόγω του ανταγωνισμού για πόρους. Από τις τεχνικές αυτές, η διαμόρφωση του ρολογιού έχει μεγαλύτερη εφαρμογή στη διαχείριση του QoS της cache από την τεχνική της DVFS. Επίσης, η εφαρμογή αυτών των τεχνικών στη διαχείριση των πόρων προσφέρει αποτελεσματική ενεργειακή διαχείριση σε συστήματα CMP.

Αυτές οι τεχνικές χρησιμοποιούνται και στην αρχιτεκτονική PI-RATE [16] (Proportional-Integral Rate control mechanism), στην οποία χρησιμοποιείται ένας ελεγκτής που παρακολουθεί τις μετρικές επίδοσης και προσαρμόζει δυναμικά το ρυθμό εκτέλεσης, ώστε να εξισορροπηθεί η χρησιμοποίηση των πόρων και να αρθούν οι άδικες ανισότητες. Στόχος της αρχιτεκτονικής αυτής είναι να καταστεί δυνατός ο έλεγχος του QoS στην πηγή των αιτημάτων για πόρους, αντί του ελέγχου στο επίπεδο των πόρων (hardware), όπου και ικανοποιούνται αυτά τα αιτήματα. Η αξιολόγηση αυτής της αρχιτεκτονικής πραγματοποιήθηκε σε συστήματα που χρησιμοποιούν τις προαναφερθείσες τεχνικές, και προέκυψε ότι εργασίες υψηλής προτεραιότητας εμφάνισαν αξιοσημείωτη βελτίωση στην επίδοση τους, ενώ ταυτόχρονα, ικανοποιείται και το πάγιο αίτημα της εξοικονόμησης ενέργειας σε μεγάλα data centers.

2.4 Συνδυασμός των Δύο Προσεγγίσεων

Ο ρόλος του O.S στη διαχείριση πόρων, όπως ο χρόνος εκτέλεσης (CPU time), η μνήμη, ακόμα και η διαχείριση ενέργειας είναι δεδομένος. Παρόλα αυτά, το λειτουργικό σύστημα στερείται της δυνατότητας να διαχειριστεί την cache, πράγμα που δεν είναι απλή υπόθεση, καθώς, παρόλο που κάτι τέτοιο θα προσέφερε ευελιξία στην επιλογή πολιτικής που θα εφαρμοστεί, η διαχείριση γεγονότων όπως ο καταμερισμός της cache μπορεί να αποδειχθεί ακριβή χρονικά διαδικασία για το O.S.

Οι N. Rafique, W. Lim και M. Thottethodi [12] πρότειναν μία αρχιτεκτονική υποστήριξη για το O.S, η οποία συνδυάζει τις δύο προηγούμενες προσεγγίσεις. Συγκεκριμένα, το προτεινόμενο σχήμα περιέχει ένα μηχανισμό διαχείρισης της cache στο hardware, ο οποίος εξασφαλίζει ότι οι αποφάσεις που λαμβάνονται από το O.S θα εφαρμόζονται στην cache. Έτσι αποφεύγεται η εμπλοκή του O.S σε αυτήν τη φάση, επομένως και το αντίστοιχο κόστος στην επίδοση. Υλοποιείται ένα σύνολο πολιτικών στο επίπεδο του O.S., το οποίο διατηρεί την ευελιξία του, αφού οι αποφάσεις μπορούν να εναλλάσσονται και να ρυθμίζονται κατά τη διάρκεια των παρεμβάσεων του O.S., καθώς και μια διεπαφή στο O.S. Το σχήμα αυτό είναι σε θέση να υποστηρίξει πληθώρα πολιτικών. Ο μηχανισμός αυτός έχει μικρό κόστος (overhead) σε hardware, ενώ το κόστος των πολιτικών εξαρτάται από την πολυπλοκότητά τους. Το κύριο πλεονέκτημα βρίσκεται στο γεγονός ότι επεμβαίνει όσο το δυνατό σπανιότερα σε συνθήκες contention, ενώ σε απουσία τέτοιων συνθηκών, δεν εμποδίζει την ελεύθερη χρήση της cache.

Οι latency critical εφαρμογές, που έχουν ήδη αναφερθεί, αποτελούν την αιτία για το χαμηλό ποσοστό χρησιμοποίησης των σύγχρονων servers. Η αντιμετώπιση αυτού του φαινομένου μπορεί να επιτευχθεί εκτελώντας, μαζί με τις LC εφαρμογές, πακέτα εργασιών «best effort», οι οποίες θα καταναλώσουν τους πόρους που μένουν αχρησιμοποίητοι από τις LC εφαρμογές. Η κύρια πρόκληση αυτής της προσέγγισης είναι η παρεμβολή μεταξύ των συνεκτελούμενων εργασιών στους μοιραζόμενους πόρους, τη στιγμή που οι LC εφαρμογές έχουν αυστηρά SLO, και ακόμα και μικρές παρεμβολές μπορούν να προκαλέσουν παραβάσεις. Για την προστασία των εφαρμογών αυτών, αρκετά συστήματα αποφεύγουν ή τερματίζουν τη συνεγκατάσταση των εφαρμογών, μειώνοντας, όμως, ταυτόχρονα τις ευκαιρίες για μεγαλύτερη χρησιμοποίηση των μηχανημάτων.

Ο Heracles [19] είναι ένας δυναμικός ελεγκτής, ο οποίος δε χρησιμοποιεί αυτήν τη μέθοδο προστασίας των εφαρμογών, αλλά, αντίθετα, με διαχείριση τεσσάρων hardware και software μηχανισμών απομόνωσης, επιτρέπουν σε LC εργασίες να συνεκτελεστούν με οποιαδήποτε best-effort εργασία. Οι δύο μηχανισμοί απομόνωσης στο hardware είναι ο καταμερισμός της cache και οι ρυθμίσεις ισχύος και συχνότητας και οι αντίστοιχοι software μηχανισμοί είναι η χρονοδρομολόγηση πυρήνων/νημάτων και ο έλεγχος κίνησης του δικτύου. Ο στόχος είναι να εξαλειφθούν όλες οι παραβάσεις του SLO σε κάθε επίπεδο φόρτου για την LC εργασία, και να μεγιστοποιηθεί παράλληλα το throughput για τις best-effort εργασίες. Αυτός ο στόχος επιτυγχάνεται παρέχοντας στην LC εργασία ακριβώς τους πόρους που είναι απαραίτητοι, ενώ η χρησιμοποίηση των υπόλοιπων πόρων μεγιστοποιείται χάρη στις best-effort εργασίες. Ο Heracles αναγνωρίζει τις περιπτώσεις που οι μοιραζόμενοι πόροι οδεύουν προς τον κορεσμό, και προκαλούν κίνδυνο παραβίασης του SLO, και τις προλαμβάνει χρησιμοποιώντας τον κατάλληλο μηχανισμό απομόνωσης. Η αξιολόγηση του ελεγκτή δείχνει ότι επιτυγχάνεται χρησιμοποίηση του συστήματος της τάξης του 90%, ενώ βελτιώνει και το throughput/TCO (Total Cost of Ownership) από 15% έως 300%, ανάλογα με την αρχική μέση χρησιμοποίηση του συστήματος.

3. Οι Τεχνολογίες Intel CMT – CAT

Οι επεξεργαστές της Intel που προορίζονται για χρήση σε συστήματα εξυπηρετητών έχουν σχεδιαστεί για μεγάλα υπολογιστικά φορτία και περιέχουν μεγάλη ποικιλία δυνατοτήτων και τεχνολογιών για τη βελτίωση της επίδοσης και την επιτάχυνση συγκεκριμένων εργασιών. Παραδείγματα αυτών των τεχνολογιών αποτελούν οι caches για δεδομένα και εντολές, ο μεγάλος αριθμός πυρήνων, και η τεχνολογία Hyper-Threading της Intel. Καθώς γίνεται δυνατή η εκτέλεση πολλών εργασιών παράλληλα, γίνεται όλο και πιο ωφέλιμη η παροχή της δυνατότητας προσαρμογής της χρήσης των μοιραζόμενων πόρων. Μια τέτοια δυνατότητα θα ήταν ιδιαίτερα χρήσιμη σε περιπτώσεις που συγκεκριμένες εργασίες χρησιμοποιούν πόρους όπως η cache αλλά δεν εκμεταλλεύονται στο μέγιστο τα οφέλη που προσφέρουν αυτοί οι πόροι. Πιο συγκεκριμένα, όταν σε ένα πολυεπεξεργαστικό σύστημα εκτελούνται ταυτόχρονα πολλές εφαρμογές, η επίδραση του ανταγωνισμού τους για πόρους εξαρτάται σημαντικά από την κατανάλωση πόρων κάθε εφαρμογής. Το γεγονός αυτό εγείρει τα εξής δύο ερωτήματα:

- Ορατότητα στη χρήση πόρων: *Μπορεί το σύστημα να παρακολουθήσει τη χρήση πόρων για κάθε εφαρμογή που εκτελείται σε αυτό;*
- Επιβολή της χρήσης: *Μπορεί το σύστημα να επιτρέψει την ανακατανομή της χρήσης των πόρων για κάθε εφαρμογή που εκτελείται σε αυτό;*

Στόχος των τεχνολογιών που θα παρουσιαστούν είναι να δώσουν απάντηση στα δύο παραπάνω ερωτήματα και να παρέχουν στο λογισμικό του συστήματος τις αντίστοιχες δυνατότητες. Στην ανάπτυξη των τεχνολογιών αυτών έχουν ληφθεί υπόψη οι ακόλουθοι παράγοντες:

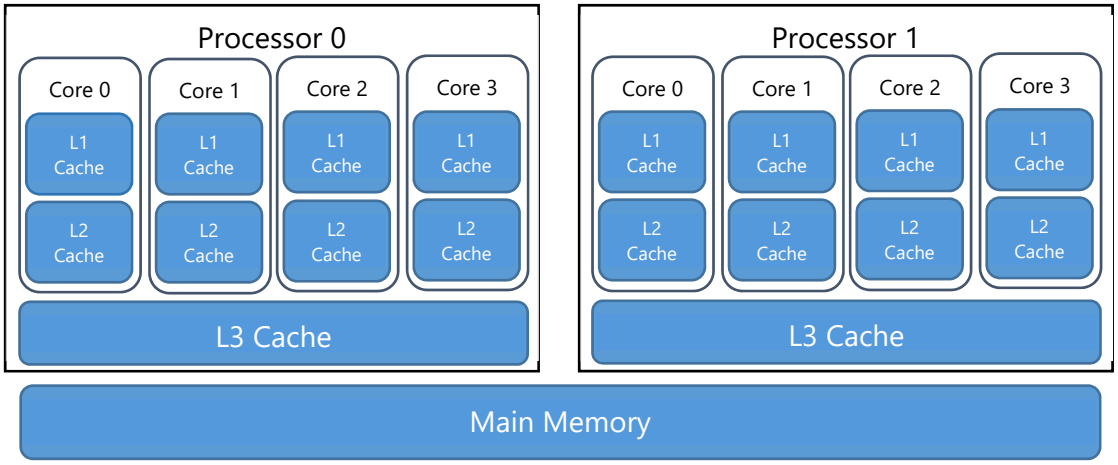
- Ευελιξία χρήσης: Οι πιθανές χρήσεις των δυνατοτήτων αυτών είναι πολλές, μεταξύ των οποίων η διευθέτηση αναγκών απομόνωσης, επιβολής προτεραιότητας, καθώς και αποδοτικότητας των πόρων. Το λογισμικό είναι ελεύθερο να λάβει αποφάσεις ανάλογα με την επιθυμητή χρήση.
- Ανεξαρτησία αρχιτεκτονικής: Η διεπαφή και η αρχιτεκτονική δεν επηρεάζονται από αλλαγές στη διαμόρφωση του συστήματος (αριθμός επεξεργαστών, πυρήνων, μέγεθος caches), είναι εύκολες στη χρήση, και μπορούν να προσαρμοστούν σε μελλοντικές αλλαγές στην αρχιτεκτονική ή στην ιεραρχία της μνήμης. Επίσης έχουν χαμηλό κόστος σε χώρο και ενέργεια.

Στην επόμενη ενότητα παρουσιάζεται η αρχιτεκτονική των cache στα σύγχρονα επεξεργαστικά συστήματα. Στις ενότητες που ακολουθούν, παρουσιάζονται οι τεχνολογίες Cache Monitoring Technology (CMT), Memory Bandwidth Monitoring (MBM) και Cache Allocation Technology (CAT). Οι επεξεργαστές Haswell υποστηρίζουν μόνο την πρώτη τεχνολογία, ενώ οι επεξεργαστές επόμενης γενιάς Broadwell υποστηρίζουν και τις τρεις.

3.1 Επισκόπηση της Αρχιτεκτονικής των Caches

Οι επεξεργαστές Intel Xeon, πάνω στους οποίους σχεδιάστηκε η τεχνολογία που παρουσιάζεται στην παρούσα διπλωματική εργασία, περιέχουν τρία επίπεδα cache memory: τις L1, L2 και L3 caches. Η L1 cache είναι η

μικρότερη και ταχύτερη cache και είναι τοποθετημένη κοντά σε κάθε πυρήνα. Η L2 cache, ή cache μεσαίου επιπέδου (Mid-Level Cache, MLC) είναι πολλές φορές μεγαλύτερη αλλά ταυτόχρονα και αρκετά πιο αργή από την L1. Τέλος, η L3 cache ή cache τελευταίου επιπέδου (LLC) είναι η μεγαλύτερη και πιο αργή cache στους επεξεργαστές Intel Xeon. Η LLC είναι κοινόχρηστη μεταξύ των πυρήνων μίας επεξεργαστικής μονάδας. Τέλος, η κύρια μνήμη είναι κοινόχρηστη μεταξύ όλων των επεξεργαστικών μονάδων του συστήματος.



Εικόνα 1: Ιεραρχία Μνημών στα Πολυεπεξεργαστικά συστήματα

Κάθε φυσικός πυρήνας έχει ιδιωτικές L1 και L2 caches. Αντίθετα η LLC είναι μοιραζόμενη και μπορεί να χρησιμοποιηθεί από όλους τους πυρήνες ενός επεξεργαστή. Επίσης η LLC είναι inclusive, το οποίο σημαίνει ότι περιέχει τα δεδομένα που υπάρχουν στα κατώτερα επίπεδα των cache. Επομένως, αν μια αίτηση για κάποιο δεδομένο αστοχήσει στα δύο πρώτα επίπεδα της cache, είναι πιθανό να εξυπηρετηθεί από την LLC. Αν η αίτηση ικανοποιηθεί από την LLC, τότε μπορεί να χρειαστεί έλεγχος για να διατηρηθεί η συνάφεια της μνήμης με κάποιον άλλο πυρήνα που μπορεί να έχει το ζητούμενο δεδομένο. Σε αντίθετη περίπτωση, όταν δηλαδή υπάρχει αστοχία και στη LLC, η αίτηση εξυπηρετείται από τη μνήμη.

Στην πλειοψηφία των περιπτώσεων οι αιτήσεις των πυρήνων για δεδομένα δημιουργούν αντίγραφο σε κάθε επίπεδο της cache. Για αυτό και όταν μια αίτηση αστοχεί σε ένα επίπεδο, μπορεί να εξυπηρετηθεί στο επόμενο. Εάν όμως μια γραμμή της LLC φύγει από την cache, θα πρέπει να ακυρωθεί και στα επίπεδα 1 και 2, εφόσον υπάρχει. Αυτό μπορεί να συμβεί, όταν για παράδειγμα ένας πυρήνας έχει δεδομένα που δεν έχει χρησιμοποιήσει για ένα χρονικό διάστημα. Καθώς οι άλλοι πυρήνες χρησιμοποιούν την cache, ο αλγόριθμος LRU είναι πιθανό να αντικαταστήσει τα δεδομένα αυτά, ακυρώνοντας και τις εγγραφές στα δύο κατώτερα επίπεδα.

3.2 Η Τεχνολογία Cache Monitoring Technology (CMT)

3.2.1 Ορισμοί, Χρήσεις και Παρεχόμενοι Μηχανισμοί

Η τεχνολογία Cache Monitoring Technology (CMT) επιτρέπει σε ένα O.S, ή ένα μέσο διαχείρισης συστήματος να προσδιορίσει τη χρήση της cache από εφαρμογές που εκτελούνται στο σύστημα. Προς το παρόν, η υλοποίηση είναι στοχευμένη στην παρακολούθηση της L3 cache (το τελευταίο επίπεδο cache στα περισσότερα συστήματα εξυπηρετητών).

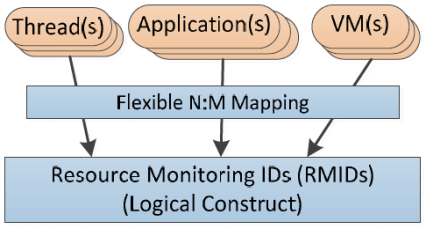
Η τεχνολογία Memory Bandwidth Monitoring (MBM) επιτρέπει την παρακολούθηση του χρήσης του διαύλου δεδομένων από ένα επίπεδο της cache στο επόμενο. Στη συγκεκριμένη περίπτωση, η τεχνολογία εστιάζεται στην L3 cache, που ακολουθείται από την κύρια μνήμη στην ιεραρχία. Επομένως είναι δυνατή η παρακολούθηση του bandwidth από τη μνήμη. [20]

Η τεχνολογία CMT εξυπηρετεί μεγάλο εύρος χρήσεων. Σε αυτές περιλαμβάνονται η δημιουργία προφίλ εφαρμογών, δηλαδή η καταγραφή της επίδοσης και του χώρου που καταλαμβάνει μια εφαρμογή όταν εκτελείται μαζί με άλλες εργασίες σε ένα σύστημα, η δρομολόγηση εργασιών από το O.S, δηλαδή η εύρεση του κατάλληλου πυρήνα και χώρου στην cache ώστε μια εφαρμογή να εκτελεστεί βέλτιστα, η λήψη μέτρησης για χρέωση ή επιβολή ορίου χρήσης σε πελάτη που χρησιμοποιεί το σύστημα, και, τέλος, η βελτίωση του QoS και της αποδοτικότητας, όταν συνδυάζεται με την Cache Allocation Technology, η οποία παρουσιάζεται και αναλύεται στο επόμενο μέρος. Στους παρεχόμενους μηχανισμούς περιλαμβάνονται οι ακόλουθοι:

- Ένας μηχανισμός για ανίχνευση της παρουσίας δυνατότητας παρακολούθησης στο σύστημα καθώς και για ενημέρωση για παραμέτρους που μπορεί να μεταβληθούν, ο οποίος χρησιμοποιεί την εντολή CPUID με κατάλληλη είσοδο.
- Ένας μηχανισμός για την παροχή της δυνατότητας να υποδειχθεί ένα ID για κάθε software thread που εκτελείται σε ένα λογικό επεξεργαστή, το οποίο ονομάζεται Resource Monitoring ID (RMID).
- Ένας μηχανισμός στο υλικό για παρακολούθηση του κατειλημμένου χώρου στην cache (cache occupancy) και του bandwidth.
- Ένας software μηχανισμός για την ανάγνωση των προηγούμενων τιμών.

3.2.2 Resource Monitoring IDs

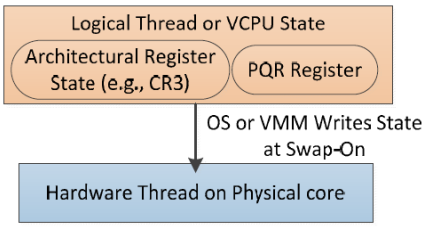
Για την ανάγνωση των δεδομένων, εισάγεται ένα επίπεδο αφαίρεσης με τη χρήση των Resource Monitoring IDs (RMIDs). Κάθε λογικός πυρήνας (hardware thread) μπορεί να αντιστοιχηθεί με ένα RMID, ενώ υπάρχει η δυνατότητα ομαδοποίησης αντιστοιχίζοντας το ίδιο RMID σε πολλούς πυρήνες. Ένα RMID μπορεί επίσης να αντιστοιχηθεί με νήματα, εφαρμογές ή εικονικές μηχανές ώστε να υπάρχει ευελιξία στην παρακολούθηση των πόρων ανάλογα με τις ανάγκες.



Εικόνα 2: Αντιστοίχιση RMID με νήματα, εφαρμογές ή VMs

Σε μια δεδομένη στιγμή μόνο ένα RMID μπορεί να είναι ενεργό σε ένα πυρήνα. Η αντιστοίχιση γίνεται με τη χρήση του MSR (MSR: Model Specific Register) IA32_PQR_ASSOC ή αλλιώς PQR, ο οποίος υποδεικνύει το τρέχον RMID του κάθε πυρήνα. Όταν μία εφαρμογή τοποθετείται για εκτέλεση σε ένα πυρήνα, το O.S ή ο ελεγκτής μιας εικονικής μηχανής θέτει στον PQR το RMID που έχει αντιστοιχηθεί στην εν λόγω εφαρμογή, επιτρέποντας στο υλικό να

ανιχνεύσει τη χρήση πόρων από την εφαρμογή μέσω του RMID της. Η διαδικασία της αντιστοίχισης RMID με ένα νήμα είναι ανεξάρτητη από τον πόρο που θα παρακολουθηθεί (δηλαδή από το αν χρησιμοποιείται η CMT ή η MBM τεχνολογία). Η ακριβής διάταξη του PQR φαίνεται στην εικόνα 4.



Εικόνα 3: Εγγραφή του RMID στον PQR

Αξίζει να σημειωθεί ότι, επειδή το RMID είναι ένας πεπερασμένος πόρος στο hardware, υπάρχει η πιθανότητα να κριθεί απαραίτητη η ανακύκλωση των RMID ώστε να γίνει εφικτή η παρακολούθηση περισσότερων νημάτων από το διαθέσιμο πλήθος των RMIDs. Τα bits που είναι δεσμευμένα για την αναπαράσταση του RMID στον PQR είναι 10, αλλά το πραγματικό πλήθος των RMIDs που υποστηρίζει ένας επεξεργαστής μπορεί να διαφέρει, για αυτό και πρέπει να ελέγχεται με τη χρήση της εντολής CPUID.

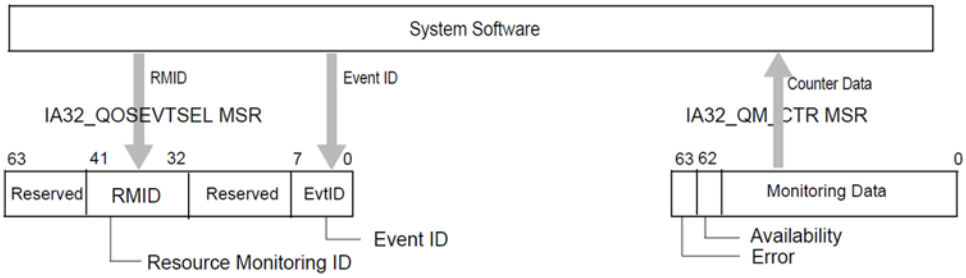
Στην αρχική υλοποίηση της CMT διατίθενται 2 RMIDs για κάθε hardware thread, δηλαδή σε ένα σύστημα με 44 πυρήνες (88 hardware threads) υποστηρίζονται 176 RMIDs για παρακολούθηση των νημάτων. Στη χρήση των RMIDs υπάρχει ένα tradeoff μεταξύ του αριθμού των νημάτων που παρακολουθούνται και της στιγμιαίας ακρίβειας των μετρήσεων. Συγκεκριμένα, προτείνονται οι τρεις επόμενοι τρόποι αντιμετώπισης ανάλογα με τον αριθμό των νημάτων ($\#Threads$) σε σύγκριση με το πλήθος των RMIDs ($\#RMID$):

- $\#Threads < \#RMID$: Σε κάθε thread αντιστοιχίζεται ένα RMID, και επιτυγχάνεται μέγιστη ακρίβεια.
- $\#Threads > \#RMID$: Υλοποίηση ενός LRU αλγορίθμου για την επιλογή του RMID που θα αντιστοιχηθεί με το νήμα που εισέρχεται προς εκτέλεση. Η χρήση του LRU RMID γίνεται μόνο εάν το cache occupancy του είναι κοντά στο 0 (αλλιώς αναζητείται άλλο RMID). Αυτό εξασφαλίζει υψηλή ακρίβεια, με ένα χαμηλό κόστος ελέγχου.
- $\#Threads \gg \#RMID$: Συνεχής ανακύκλωση των RMIDs, με χρήση αλγορίθμου LRU (χωρίς τη συνθήκη για μηδενικό cache occupancy)

3.2.3 Επιλογή Πόρων και Αναφορά Μετρήσεων

Ο μηχανισμός αναφοράς των μετρήσεων εκτίθεται ως ένα ζεύγος MSR, από το οποίο μπορούν να αντληθούν οι μετρήσεις για την cache occupancy και το bandwidth ανά RMID. Ο πρώτος καταχωρητής είναι ο IA32_QM_EVTSEL, ο οποίος μπορεί να τεθεί από το software με ένα ζεύγος RMID και αριθμού συμβάντος (Event ID) για να αναγνωστεί ο κατάλληλος counter για τη ζητούμενη μέτρηση. Αφού τεθεί το ζεύγος {RMID, Event ID}, το υλικό επιστρέφει την απάντηση στον καταχωρητή IA32_QM_CTR, ο οποίος περιέχει πεδία για αναφορά σφάλματος ή μη διαθεσιμότητας της μέτρησης. Αφού ληφθεί η μέτρηση, πολλαπλασιάζεται με έναν όρο για να μετατραπεί στην

κατάλληλη μονάδα μέτρησης (kB για cache occupancy, Mbps για το bandwidth). [5]



Εικόνα 4: Η διαδικασία της λήψης μέτρησης

3.3 Η Τεχνολογία Cache Allocation Technology (CAT)

3.3.1 Ορισμός και Παραδείγματα Χρήσης

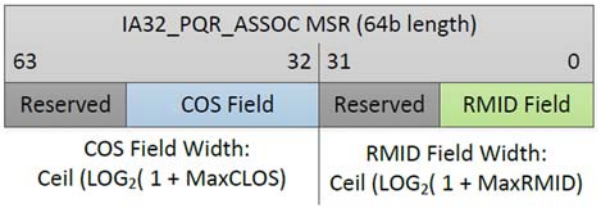
Η Cache Allocation Technology (CAT) [21] επιτρέπει σε ένα O.S, διαχειριστή εικονικών μηχανών ή κάποιο παρόμοιο σύστημα διαχείρισης να προσδιορίσουν το χώρο της cache, στον οποίο μπορεί να χρησιμοποιήσει ένας πυρήνας ή μια εφαρμογή. Η αρχική υλοποίηση της CAT εστιάζει στον καταμερισμό της L3 cache, αλλά η τεχνολογία μπορεί να επεκταθεί και για τον καταμερισμό άλλων επιπέδων σε διαφορετικές γενιές αρχιτεκτονικών.

Περιορίζοντας το χώρο, τον οποίο επιτρέπεται να χρησιμοποιήσει μία εφαρμογή ή ένας πυρήνας, καθίσταται αδύνατο να εξαχθεί από την cache μια γραμμή που δεν ανήκει σε αυτόν το χώρο. Παρόλα αυτά, η ανάγνωση ή η εγγραφή από έναν πυρήνα μπορεί να καταλήξει σε ευστοχία αν η γραμμή υπάρχει σε οποιαδήποτε cache line, συμπεριλαμβανομένων, δηλαδή, και των γραμμών στις οποίες δεν επιτρέπεται η καταχώρηση.

Η CAT είναι ιδιαιτέρως χρήσιμη σε περιπτώσεις που μια εφαρμογή έχει επιθετική συμπεριφορά, δηλαδή αποθηκεύει μεγάλη ποσότητα δεδομένων, ασκώντας πίεση στις υπόλοιπες εφαρμογές, χωρίς να επαναχρησιμοποιούν τα δεδομένα. Παραδείγματα αποτελούν οι εφαρμογές file hosting ή streaming. Ιδανικά, αυτές οι εφαρμογές μπορούν να περιοριστούν σε ένα μικρό τμήμα της cache, ώστε οι υπόλοιπες εφαρμογές να είναι σε θέση να χρησιμοποιήσουν το χώρο που χρειάζονται.

3.3.2 Δομή και Λειτουργία της CAT

Στην αρχιτεκτονική της CAT εισάγεται ένα επίπεδο αφαίρεσης που ονομάζεται Class Of Service (COS), στο οποίο μπορούν να ομαδοποιηθούν πυρήνες, εφαρμογές ή εικονικές μηχανές, και ως αποτέλεσμα, η κατανομή της cache περιορίζεται με βάση το COS στο οποίο ανήκουν. Οι COS μοιράζονται τον ίδιο καταχωρητή με τα RMID, δηλαδή τον PQR που αναφέρθηκε και στο προηγούμενο μέρος.



Εικόνα 5: PQR ανά λογικό πυρήνα

Η παραπάνω εικόνα δείχνει την εσωτερική δομή του καταχωρητή PQR, όπου φαίνονται τα bits που διατίθενται για την αποθήκευση του RMID και του COS.

Αφού γίνει η αντιστοίχιση ενός λογικού πυρήνα με ένα COS, η διαχείριση της κατανομής της cache πραγματοποιείται αυτόματα από το hardware βασιζόμενη στο COS και τη μάσκα που σχετίζεται με αυτό. Οι μάσκες ρυθμίζονται από τον MSR IA32_<resourceType>_MASK_n, όπου resourceType είναι ο τύπος του πόρου (π.χ. για την L3 cache το resourceType είναι «L3»), και το n δείχνει τον αριθμό του COS.

Τα βασικά συστατικά της CAT είναι τα ακόλουθα:

- Ένας μηχανισμός που χρησιμοποιεί την CPUID για να ανιχνεύσει αν η CAT υποστηρίζεται από το σύστημα, καθώς και ποιοι τύποι πόρων μπορούν να ελεγχθούν.
- Για κάθε διαθέσιμο resourceType, το CPUID ενημερώνει για το διαθέσιμο αριθμό των COS, καθώς και για το μήκος της μάσκας που χρησιμοποιείται για την επιβολή του καταμερισμού της cache.
- Ένας μηχανισμός που επιτρέπει στο περιβάλλον εκτέλεσης (O.S ή άλλος ελεγκτής) τη ρύθμιση της συμπεριφοράς των COS αλλάζοντας κατάλληλα τις σχετιζόμενες μάσκες.
- Ένας μηχανισμός με τον οποίο γίνεται δυνατή η αντιστοίχιση ενός COS σε ένα νήμα software.

Η μάσκα (capacity bitmask, CBM) παρέχει στο hardware την ένδειξη για το χώρο της cache στον οποίο πρέπει να περιοριστεί μια εφαρμογή, καθώς και ένδειξη για επικάλυψη ή απομόνωση από τις άλλες εφαρμογές που ανήκουν σε διαφορετικό COS.

	W7	W6	W5	W4	W3	W2	W1	W0
COS0	1	1	1	1	1	1	1	1
COS1	1	1	1	1	1	1	1	1
COS2	1	1	1	1	1	1	1	1
COS3	1	1	1	1	1	1	1	1

Default Bitmask

	W7	W6	W5	W4	W3	W2	W1	W0
COS0	1	1	1	1	1	1	1	1
COS1	0	0	0	0	1	1	1	1
COS2	0	0	0	0	0	0	1	1
COS3	0	0	0	0	0	0	0	1

Overlapped Bitmask

	W7	W6	W5	W4	W3	W2	W1	W0
COS0	1	1	1	1	0	0	0	0
COS1	0	0	0	0	1	1	0	0
COS2	0	0	0	0	0	0	1	0
COS3	0	0	0	0	0	0	0	1

Isolated Bitmask

Εικόνα 6: Παραδείγματα Μασκών

Στην εικόνα 6 φαίνονται παραδείγματα μασκών που μπορούν να συσχετιστούν με τα COS. Συγκεκριμένα, οι περιοχές (τα ways) στις οποίες επιτρέπεται η πρόσβαση για κάθε COS φαίνονται με την τιμή 1 στη μάσκα, και με πράσινο χρώμα. Σημειώνεται ότι μόνο συνεχόμενοι συνδυασμοί «1» είναι επιτρεπτοί (δηλαδή `0x3F=0011 1111`, `0x7c=0111 1110`). Σε υλοποιήσεις

που βασίζονται στην associativity της cache, κάθε bit μιας μάσκας αντιστοιχεί σε έναν αριθμό ways στην cache. Σε κάθε περίπτωση, εφόσον το bit στη μάσκα είναι 0, η χρήση του αντίστοιχου αριθμού των ways απαγορεύεται, ενώ αντίστοιχα αν το bit είναι 1, επιτρέπεται η χρήση των ways. Στη γενική περίπτωση, η χρήση περισσότερης cache από μια εφαρμογή συνεπάγεται βελτιωμένη επίδοση. Κατά την ενεργοποίηση του συστήματος, όλες οι μάσκες είναι αρχικοποιημένες στο 1, το οποίο σημαίνει ότι η CAT είναι αρχικά απενεργοποιημένη.

Η εικόνα 6 δείχνει τρία παραδείγματα μασκών, οι οποίες για λόγους απλότητας έχουν μήκος 8 bits. Στην πρώτη περίπτωση, φαίνεται η γενική περίπτωση, στην οποία τα τέσσερα COS έχουν πρόσβαση σε όλη την cache. Η δεύτερη περίπτωση είναι η περίπτωση επικάλυψης, στην οποία κάποια νήματα χαμηλής προτεραιότητας επιτρέπεται να μοιράζονται ένα κομμάτι της cache με τα νήματα υψηλής προτεραιότητας. Στην τρίτη περίπτωση φαίνονται σχήματα απομόνωσης. Η πολιτική που αναμένεται να ακολουθήσει το software είναι να θεωρήσει το COS[0] ως το COS για τα νήματα με την υψηλότερη προτεραιότητα, ακολουθούμενο από το COS[1], κ.ο.κ.. Δεν υπάρχει, όμως, κάποιος περιορισμός που να επιβάλλει αυτήν την αντιστοίχιση. Γενικά, οι πολιτικές επικάλυψης προσφέρουν καλύτερο throughput, ενώ οι πολιτικές απομονωμένων μασκών προσφέρουν καλύτερη απομόνωση της επίδοσης και ντετερμινισμό. [5], [21]

3.3.3 Code and Data Prioritization

Η τεχνολογία Code and Data Prioritization (CDP) είναι μια επέκταση της CAT, η οποία ενεργοποιεί την απομόνωση και την ξεχωριστή ιεράρχηση των ανακτήσεων δεδομένων και εντολών στη L3 cache. Η CDP επεκτείνει την CAT παρέχοντας ξεχωριστές μάσκες για δεδομένα και εντολές ανά COS.

Στη γενική περίπτωση, το CDP είναι απενεργοποιημένο. Όταν το CDP ενεργοποιηθεί, οι MSR αντιστοιχίζονται εκ νέου σε ζεύγη MSR για ανάκτηση δεδομένων και εντολών, και οι COS ανακατατάσσονται, με το πρώτο μισό των COS να χρησιμοποιείται για το CDP. Με τη χρήση του CDP, μπορεί να επιτευχθεί απομόνωση μεταξύ δεδομένων και εντολών, παρόμοια με την απομόνωση που παρέχεται από τις ξεχωριστές instruction και data L1 caches.

Example of CAT-Only Usage - 16-bit Capacity Masks

COS0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
COS1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
COS2	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
COS3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Traditional CAT

Example of Code-Data Prioritization Usage - 16-bit Capacity Masks

COS0.Data	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
COS0.Code	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
COS1.Data	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0
COS1.Code	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
Other COS.Data	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
Other COS.Code	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

CAT with CDP

Εικόνα 7: Capacity Bitmasks για την τεχνολογία CDP

Στην εικόνα 7 φαίνεται ένα παράδειγμα της αντιστοίχισης των συμβατικών COS σε διακριτές Data και Code COS. Στο άνω μέρος φαίνεται το

μοντέλο χρήσης των COS από την CAT, όπου η αναλογία των μασκών με τα COS είναι 1: 1, ενώ στο κάτω μέρος φαίνεται η περίπτωση, στην οποία το CDP είναι ενεργοποιημένο, και κάθε COS έχει αναλογία 1: 2 με 2 μάσκες, μία για εντολές και μία για δεδομένα. Με αυτόν τον τρόπο είναι δυνατή η εφαρμογή της επιθυμητής πολιτικής για το διαχωρισμό, την απομόνωση ή την επικάλυψη των δεδομένων και των εντολών, ανάλογα με τις ανάγκες του συστήματος. Η αναλογία των συμβατικών COS με την περίπτωση λειτουργίας του CDP, καθώς και ο ανανεωμένος ρόλος των MSR, είναι εμφανή και στην επόμενη εικόνα.

Mask MSR	CAT-Only Operation	CDP Operation
IA32_L3_QOS_Mask_0	COS0	COS0.Data
IA32_L3_QOS_Mask_1	COS1	COS0.Code
IA32_L3_QOS_Mask_2	COS2	COS1.Data
IA32_L3_QOS_Mask_3	COS3	COS1.Code
IA32_L3_QOS_Mask_4	COS4	COS2.Data
IA32_L3_QOS_Mask_5	COS5	COS2.Code
…	…	…
IA32_L3_QOS_Mask_'2n'	COS'2n'	COS'n'.Data
IA32_L3_QOS_Mask_'2n+1'	COS'2n+1'	COS'n'.Code

Εικόνα 8: Αντιστοίχιση MSR και COS με ενεργό CDP

Πριν την ενεργοποίηση ή την απενεργοποίηση του CDP, το software οφείλει να επαναφέρει όλες τις CAT/CDP μάσκες στο 1, ώστε να εξασφαλιστεί η ομαλή λειτουργία. Επίσης, λόγω της διαφορετικής σημασίας που αποκτούν οι COS όταν το CDP είναι ενεργοποιημένο, το οποίο σημαίνει ότι επί παραδείγματι το COS[1] με CDP είναι πλέον το κομμάτι της COS[0] για εντολές, όπως φαίνεται και στην εικόνα 8, εκτός από την επαναφορά των bit στις μάσκες, προτείνεται η επαναφορά όλων των συσχετίσεων μεταξύ πυρήνων και COS στο COS[0], ώστε να αποφευχθεί αθέμιτη αντιστοίχιση. [21]

3.4 Υλοποιήσεις της Intel CMT – CAT

Η χρήση των δυνατοτήτων των τεχνολογιών που αναλύθηκαν είναι απλή από την άποψη της υλοποίησης σε λογισμικό, καθώς τα σύγχρονα O.S παρέχουν στον προγραμματιστή τις κατάλληλες διεπαφές, έτσι ώστε να πραγματοποιηθούν εγγραφές και αναγνώσεις στους MSR. Για παράδειγμα, το Linux παρέχει τα msr-tools, ένα πακέτο που περιέχει τις εντολές `rdmsr` και `wrmsr`, για ανάγνωση και εγγραφή αντίστοιχα. Υπάρχουν δύο προσεγγίσεις για την παρακολούθηση με τη CMT, καθώς το σύστημα μπορεί να παρακολουθηθεί χρησιμοποιώντας μια αυτόνομη προσέγγιση που δε βασίζεται στον scheduler, ή χρησιμοποιώντας μια προσέγγιση που βασίζεται στον scheduler.

3.4.1 Προσεγγίσεις και Ρόλος του Χρονοδρομολογητή

Η αυτόνομη προσέγγιση ελέγχει την cache occupancy από την προοπτική του πυρήνα ή νήματος, ανεξάρτητα με την εφαρμογή που εκτελείται. Γίνεται στατική αντιστοίχιση ενός RMID στον πυρήνα και περιοδικά διαβάζεται η χρήση της cache. Αυτή η προσέγγιση ενδείκνυται εάν το σύστημα είναι ρυθμισμένο στατικά και κάθε εφαρμογή εκτελείται σε συγκεκριμένο πυρήνα, ή στην περίπτωση που το ενδιαφέρον των διαχειριστών έγκειται στο αν το σύστημα είναι σε ισορροπία και δεν υπάρχουν εφαρμογές που έχουν επιθετική συμπεριφορά.

Η προσέγγιση που βασίζεται στον scheduler εμπλέκει προφανώς την ενεργοποίηση του scheduler. Στην προηγούμενη προσέγγιση δε γίνεται παρακολούθηση του process ID μιας εφαρμογής, επομένως τα αποτελέσματα δεν είναι εφικτό να αφορούν εφαρμογές, αν αυτές δεν εκτελούνται σε συγκεκριμένους πυρήνες. Για να γίνει δυνατή η παρακολούθηση μιας εφαρμογής χωρίς να είναι αναγκαία αυτή η συνθήκη, πρέπει να αντιστοιχηθεί ένα RMID στην εφαρμογή (από το λογισμικό παρακολούθησης), και ο scheduler να συσχετίζει τον πυρήνα στον οποίο εκτελείται κάθε φορά η εφαρμογή με το RMID της. Όταν η εφαρμογή πάψει να εκτελείται (π.χ. γιατί έληξε το κβάντο χρόνου της) ο scheduler ανανεώνει τη συσχέτιση για να εξασφαλιστεί ότι δε θα χρεωθούν στη συγκεκριμένη εφαρμογή μετρήσεις που δεν της ανήκουν, και η παρακολούθηση να γίνεται μόνο όταν αυτή εκτελείται. Το λογισμικό είναι υπεύθυνο και για τη διαχείριση των ανακατατάξεων που πιθανώς να χρειαστούν. Τα RMIDs και COS είναι τοπικά σε κάθε socket, επομένως, εάν η εφαρμογή μετακινηθεί σε άλλο socket, είναι αναγκαία η εύρεση των κατάλληλων RMIDs και COS, για να συνεχιστεί απρόσκοπτα η επιθυμητή λειτουργία των CMT – CAT. [5]

3.4.2 Διαθέσιμα Πακέτα Λογισμικού

Για την πρακτική εφαρμογή των προσεγγίσεων που αναλύθηκαν, η Intel έχει αναπτύξει την *Cache Monitoring/Allocation Library* η οποία είναι διαθέσιμη από το Intel Open Source Technology Center (www.01.org). Η συγκεκριμένη αυτόνομη βιβλιοθήκη επιτρέπει την παρακολούθηση της L3 cache, ενώ υπάρχει ήδη η επέκταση για την L2 cache. Η παρακολούθηση είναι δυνατό να γίνει ανά πυρήνα, αλλά και ανά PID. Επίσης, υπάρχει η δυνατότητα χρήσης της CAT, αφού παρέχεται μια εύχρηστη διεπαφή για τον ορισμό COS και το συσχετισμό μεταξύ COS και πυρήνων. Κατά την αρχικοποίηση, ελέγχεται η παρουσία υποστήριξης των CMT και CAT. Κατόπιν, ακολουθεί η παρακολούθηση των επιλεγμένων πόρων με την έξοδο να εμφανίζεται σε μορφή ανάλογη της «`top`», είτε σε αρχείο κειμένου, xml, ή csv. Για τη χρήση των λειτουργιών της βιβλιοθήκης παρέχεται η εφαρμογή PQoS. Σε αυτήν την υλοποίηση βασίζεται η παρούσα διπλωματική εργασία. Η βιβλιοθήκη αυτή υποστηρίζεται και από το υποσύστημα cgroups (control groups), με το οποίο το Linux υποστηρίζει τη δυνατότητα ομαδοποίησης εργασιών. Η χρήση αυτού του υποσυστήματος, μαζί με αλλαγές στον scheduler του Linux καθιστούν δυνατή την ανίχνευση της αντιστοίχισης του RMID καθώς οι εφαρμογές ή τα νήματα δρομολογούνται στους πυρήνες. Η cache occupancy των εφαρμογών, που ανήκουν σε ένα σύνολο, συσσωρεύεται και προβάλλεται στο χρήστη.

3.5 Σύστημα Πειραματικής Αξιολόγησης

Η μελέτη των δύο τεχνολογιών πραγματοποιήθηκε στον επεξεργαστή γενιάς Broadwell, Intel® Xeon® Processor E5-2699 v4, ο οποίος περιέχει 22 πυρήνες, ενώ η L3 cache του έχει μέγεθος 55MB ή 56.320kB και είναι 20-way associative. Όσον αφορά τις τεχνολογίες Intel CMT – CAT, ο επεξεργαστής υποστηρίζει 176 RMIDs και 16 COS, ενώ οι μάσκες των COS έχουν μήκος 20 bits. Επομένως υπάρχει 1:1 αντιστοιχία bits σε μία μάσκα με ways of associativity. Για την άρση της εξάρτησης των μετρήσεων από τη συχνότητα του ρολογιού, η συχνότητα τέθηκε σταθερά ίση με 2.2GHz για όλους τους πυρήνες. Επίσης απενεργοποιήθηκε η τεχνολογία Intel Hyper-Threading.

4. Μελέτη και Χαρακτηριστικά Εφαρμογών

Στο παρόν κεφάλαιο παρουσιάζονται τα μετροπρογράμματα (benchmarks) που χρησιμοποιήθηκαν για την κατανόηση της λειτουργίας των τεχνολογιών Intel CMT – CAT, την πρακτική επιβεβαίωση της ύπαρξης του προβλήματος που δημιουργεί ο ανταγωνισμός μεταξύ των συνεκτελούμενων εφαρμογών και, τέλος, για την ανάπτυξη και αξιολόγηση του μηχανισμού προστασίας που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Συγκεκριμένα, χρησιμοποιήθηκαν 28 benchmarks της σουίτας SPEC2006 και 11 benchmarks της σουίτας Parsec 3.0. 9 εκ των 28 benchmarks της SPEC2006 διαθέτουν περισσότερες από μία εισόδους, για κάθε μία από τις οποίες παρουσιάζουν διαφορετική συμπεριφορά και, για αυτό το λόγο κάθε benchmark μελετάται ξεχωριστά για κάθε του είσοδο.

Με τη χρήση των τεχνολογιών Intel CMT – CAT πραγματοποιήθηκαν 4 φάσεις μετρήσεων για την εξαγωγή των χαρακτηριστικών καθενός από τα 65 διαθέσιμα benchmarks.

- Στην πρώτη φάση οι μετρήσεις στοχεύουν στη δημιουργία ενός προφίλ για κάθε benchmark, το οποίο αφορά την επίδοση (μετρούμενη μέσω του IPC) σε σχέση με τον παρεχόμενο χώρο στην cache. Πιο συγκεκριμένα, λαμβάνονται 20 μετρήσεις για κάθε benchmark, εκκινώντας από την παραχώρηση ενός way, που είναι και το μικρότερο μέρος της cache που μπορεί να παραχωρηθεί, και καταλήγοντας στην παραχώρηση ολόκληρης της cache. Στη φάση αυτή, κάθε benchmark εκτελείται μόνο του στο μηχάνημα.
- Στη δεύτερη φάση εξάγεται για κάθε benchmark η συνάρτηση του IPC με το χρόνο, σε συνθήκες που παρέχεται στο benchmark ολόκληρη η cache και, όπως προηγουμένως, εκτελείται μόνο του στο μηχάνημα. Παράλληλα, εξάγεται ένα συνολικό προφίλ για κάθε benchmark, σε συνδυασμό με την πρώτη φάση των μετρήσεων. Τα benchmarks ομαδοποιούνται σε 5 κατηγορίες, ανάλογα με τη διακύμανση της επίδοσης τους κατά τη διάρκεια της εκτέλεσης. Η κατάσταση εκτέλεσης υπό αυτές τις συνθήκες ονομάζεται κατάσταση «Alone».
- Στην τρίτη φάση σχηματίζονται ζευγάρια συνεκτέλεσης που έχουν ένα benchmark που λογίζεται ως εφαρμογή υψηλής προτεραιότητας και 21 εκτελέσεις του ίδιου benchmark που λογίζονται ως χαμηλής προτεραιότητας. Κάθε benchmark τοποθετείται στατικά σε ένα πυρήνα για εκτέλεση. Και οι 22 εφαρμογές έχουν πλήρη πρόσβαση στην cache, ώστε να παρατηρηθεί η επίδραση του ανταγωνισμού για την cache στην επίδοση της εφαρμογής υψηλής προτεραιότητας. Αυτή η κατάσταση εκτέλεσης ονομάζεται κατάσταση Full Cache.

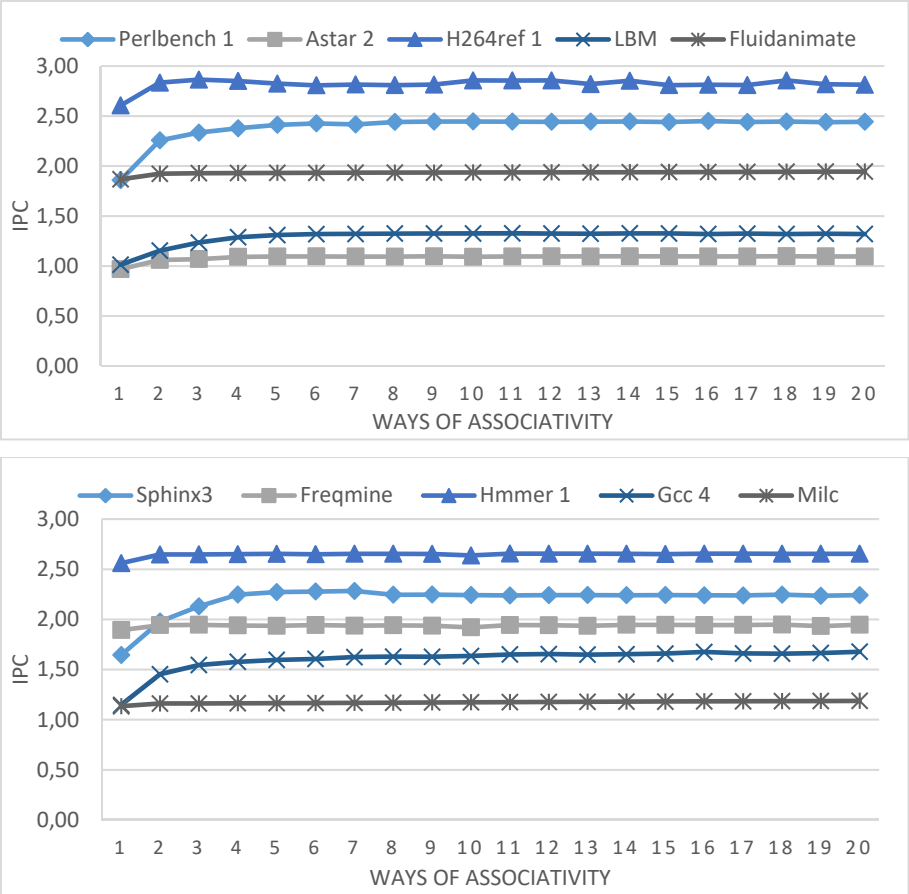
Στις επόμενες ενότητες παρουσιάζονται και αναλύονται γραφικές παραστάσεις και πίνακες από τις 3 φάσεις των μετρήσεων. Σημειώνεται ότι ο αριθμός δίπλα στο όνομα του benchmark χρησιμοποιείται για να καταδειχθεί η είσοδος που δόθηκε στο benchmark, και υπάρχει στα benchmarks που έχουν περισσότερες από μία εισόδους. Στην τελευταία ενότητα παρουσιάζεται και αξιολογείται μία στατική λύση για την αντιμετώπιση της υποβάθμισης της επίδοσης των high priority εφαρμογών κατά τη συνεκτέλεση τους.

4.1 Επίδοση Συναρτήσει του Χώρου στην Cache

Σε αυτό το μέρος των μετρήσεων, στόχος είναι να εξαχθεί για κάθε benchmark μια εκτίμηση της επίδοσης του σε σχέση με το μέγεθος της cache που του παραχωρείται. Για αυτό το σκοπό, εκτελούνται 20 μετρήσεις για κάθε benchmark, αυξάνοντας σε κάθε μία το διαθέσιμο ποσοστό της cache για το benchmark. Σε κάθε μέτρηση, εξάγεται το μέσο IPC που επιτυγχάνει το benchmark, και τελικά προκύπτει η σχέση μεταξύ επίδοσης και χώρου στην cache.

Τα benchmarks μπορούν να ομαδοποιηθούν σε τρεις κατηγορίες με βάση τον αριθμό των ways που είναι αναγκαίος για να επιτευχθεί το 95% του IPC που επιτυγχάνεται στην περίπτωση που το benchmark καταλαμβάνει όλη την cache.

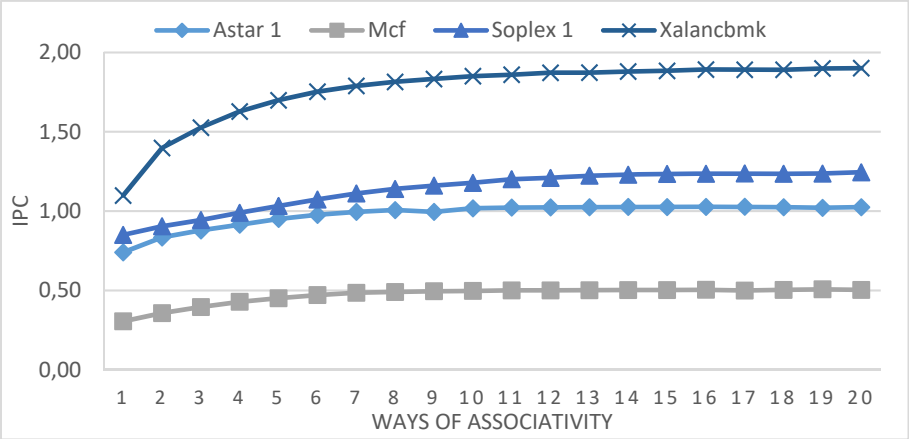
- Στην πρώτη και πολυπληθέστερη κατηγορία ανήκουν τα benchmarks που απαιτούν το πολύ 5 ways για να παρουσιάσουν επίδοση πολύ κοντά στην επίδοση υπό συνθήκες κατοχής όλης της cache. Συγκεκριμένα, στην κατηγορία αυτή ανήκουν 52 από τα 65 benchmarks. Τα διαγράμματα που ακολουθούν είναι ενδεικτικά αυτής της κατηγορίας.



Γράφημα 1: Benchmarks με Μικρή Απαίτηση Cache

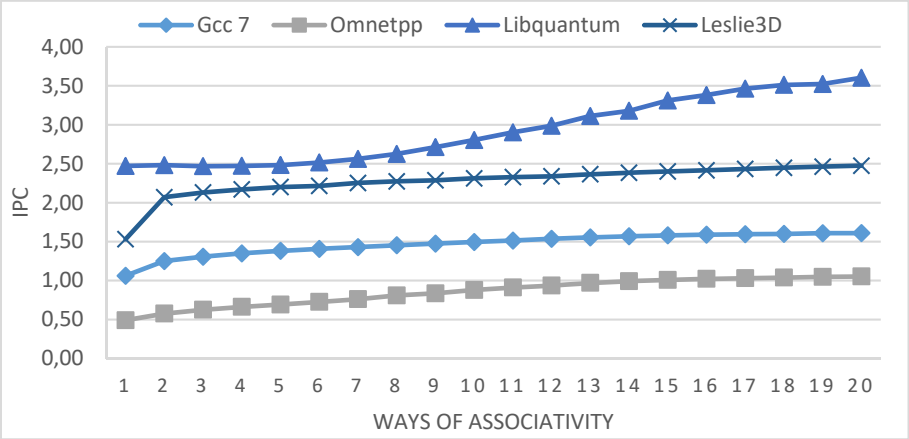
- Στη δεύτερη κατηγορία ανήκουν 8 benchmarks, τα οποία χρειάζονται από 6 έως 10 ways (δηλαδή μέχρι και τη μισή cache) ώστε να φτάσει η επίδοση τους κοντά στη μέγιστη δυνατή, σε συνθήκες δηλαδή που κατέχουν ολόκληρη την cache. Οι επόμενες γραφικές παραστάσεις είναι ενδεικτικές

αυτής της κατηγορίας, ενώ σε κάποιες από αυτές μπορούν να παρατηρηθούν και περιοχές γραμμικότητας.



Γράφημα 2: Benchmarks με Μέση Απαίτηση Cache

- Στην τρίτη και τελευταία κατηγορία ανήκουν 5 benchmarks, τα οποία χρειάζονται μεγάλο μέρος της cache έτσι ώστε να βελτιώσουν την επίδοση τους σε επίπεδα κοντινά σε συνθήκες κατοχής όλης της cache. Στα επόμενα γραφήματα παρατηρείται γραμμικότητα μεταξύ της επίδοσης και της παρεχόμενης cache.



Γράφημα 3: Benchmarks με Υψηλή Απαίτηση Cache

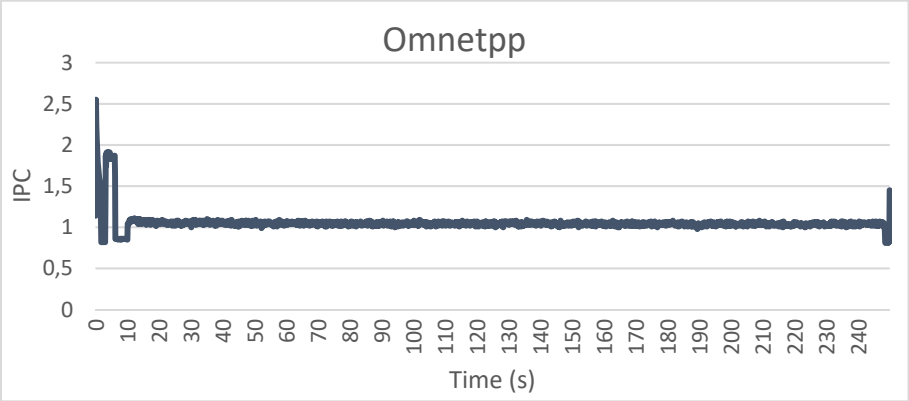
4.2 Επίδοση Συναρτήσει του Χρόνου

Στη δεύτερη φάση των μετρήσεων σχεδιάστηκε για κάθε ένα από τα 65 benchmarks η επίδοση του σε συνάρτηση με το χρόνο. Οι μετρήσεις προκύπτουν από εκτέλεση του κάθε benchmark χωρίς άλλες συνεκτελούμενες εφαρμογές, ώστε να μην υπάρχει καμία παρεμβολή, ενώ παραχωρείται στο benchmark ολόκληρη η cache. Με αυτόν τον τρόπο είμαστε σε θέση να γνωρίζουμε τη συμπεριφορά κάθε benchmark, καθώς και τις διακυμάνσεις της επίδοσης του κατά την εκτέλεση του, στις βέλτιστες συνθήκες. Αργότερα, στόχος θα είναι να οδηγήσουμε το benchmark με υψηλή προτεραιότητα όσο εγγύτερα γίνεται σε αυτήν ακριβώς την κατάσταση.

Εκτελώντας τα benchmarks, και σχεδιάζοντας τις γραφικές παραστάσεις IPC συναρτήσει του χρόνου, διαπιστώνουμε ότι, όπως και προηγουμένως, τα

benchmarks είναι δυνατό να διαχωριστούν σε 5 κατηγορίες, ανάλογα με τις διακυμάνσεις που παρατηρούνται στο IPC. Στην ανάλυση της κάθε κατηγορίας, που ακολουθεί, παρουσιάζεται μία γραφική παράσταση επίδοσης (IPC) κατά τη διάρκεια της εκτέλεσης από ένα benchmark που ανήκει στην κατηγορία, και δίνει μία αίσθηση για τη συμπεριφορά των υπολοίπων benchmarks. Στον πίνακα που παρατίθεται φαίνονται τα χαρακτηριστικά όλων των benchmarks που ανήκουν σε αυτήν την κατηγορία, όταν εκτελούνται μόνα τους στο σύστημα και έχουν πρόσβαση σε όλη την cache. Συγκεκριμένα, οι στήλες περιέχουν το μέσο IPC που επιτεύχθηκε (στήλη IPC), το χρόνο εκτέλεσης σε δευτερόλεπτα (στήλη Time), ο αριθμός αστοχιών ανά χιλιάδα εντολών (Misses per Kilo Instructions, MPKI), το ποσοστό του χώρου της cache που χρησιμοποιήθηκε (στήλη Util.), το μέσο bandwidth μεταξύ cache και κύριας μνήμης (στήλη MB), και τον αριθμό των ways που είναι απαραίτητος για την επίτευξη του 95% της μέγιστης επίδοσης.

- Στην πρώτη κατηγορία (Class A) τοποθετούνται τα benchmarks, των οποίων το IPC είναι σταθερό, ή παρουσιάζει μια μικρή διακύμανση. Σε αυτήν την κατηγορία ανήκουν 14 benchmarks.

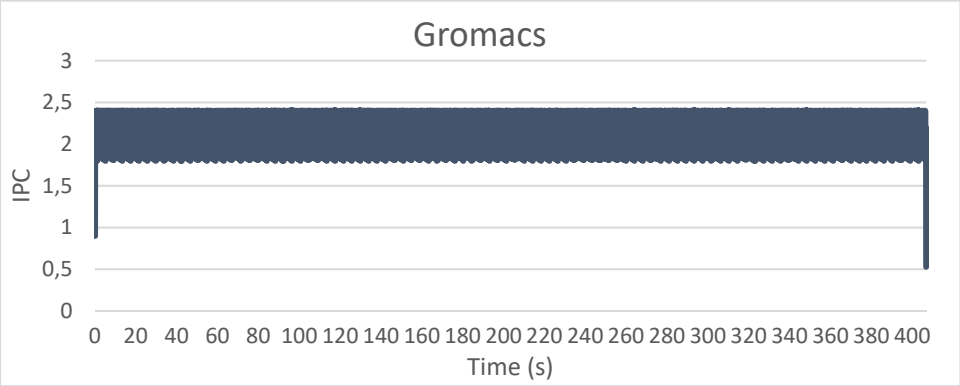


Γράφημα 4: Παράδειγμα Benchmark της Class A: Omnetpp

Benchmark	IPC	Time (s)	MPKI	Util.	MB	W_95
Bodytrack	2.30	166	0.01	96.3%	4.87	2
Gamess 2	3.01	129	0.00	36.5%	0.00	2
Gamess 3	3.05	534	0.00	40.4%	0.00	2
H264ref 1	2.81	78	0.00	41.7%	0.00	2
Hmmer 1	2.65	152	0.00	79.9%	0.14	1
Hmmer 2	2.59	331	0.00	11.7%	0.00	2
Lbm	1.32	428	15.73	99.6%	2900.82	4
Leslie3d	2.48	283	4.01	96.9%	376.01	13
Omnetpp	1.05	250	0.33	97.3%	4.43	15
Povray	2.41	183	0.00	25.1%	0.00	2
Sjeng	1.64	617	0.21	98.4%	39.94	2
Streamcluster	0.90	541	13.63	99.8%	753.15	3
Swaptions	2.25	297	0.00	19.1%	0.00	1
x264	1.89	102	0.63	94.4%	89.96	2

Πίνακας 1: Χαρακτηριστικά των Benchmarks της Class A

- Η δεύτερη κατηγορία (Class B) περιέχει τα benchmarks που εμφανίζουν μεγάλη διακύμανση στο IPC, το οποίο όμως ταλαντεύεται γύρω από μια σχεδόν σταθερή μέση τιμή. Στην Class B ανήκουν 11 benchmarks. Το benchmark που παρουσιάζεται είναι το Gromacs.

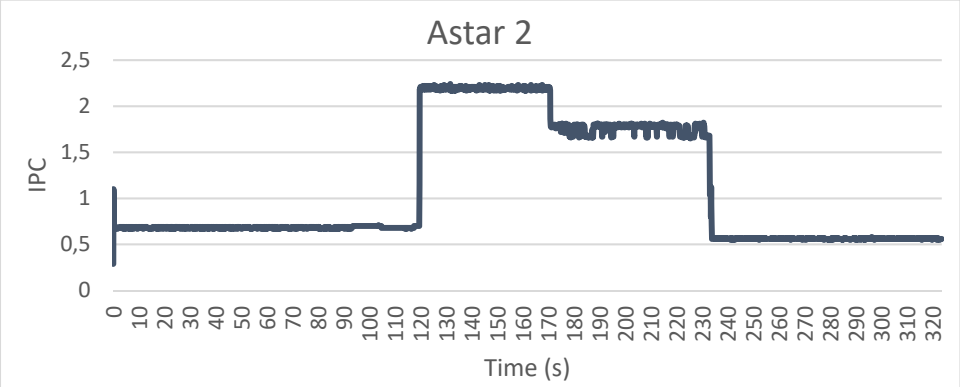


Γράφημα 5: Παράδειγμα Benchmark της Class B: Gromacs

Benchmark	IPC	Time (s)	MPKI	Util.	MB	W_95
CactusADM	1.72	657	2.45	99.8%	595.03	4
Calculix	3.12	900	0.04	98.7%	19.11	1
Ferret	1.69	468	0.10	96.3%	1.31	1
Fluidanimate	1.94	392	1.38	99.7%	266.32	1
GemsFDTD	1.89	374	15.76	99.8%	4782.93	3
Gromacs	2.19	406	0.00	28.9%	0.00	2
Libquantum	3.60	288	0.15	91.1%	1.12	17
Namd	2.15	478	0.00	71.2%	0.01	1
Perlbench 1	2.44	195	0.00	95.1%	2.39	3
Wrf	2.50	551	1.77	99.5%	483.80	2
Zeusmp	1.96	416	4.05	99.6%	1005.00	2

Πίνακας 2: Χαρακτηριστικά των Benchmarks της Class B

- Στη τρίτη κατηγορία (Class C) ανήκουν τα benchmarks, στις γραφικές παραστάσεις των οποίων παρατηρούνται 2 με 3 ευδιάκριτες φάσεις εκτέλεσης με διαφορετικό επίπεδο IPC η καθεμία. Στην Class C ανήκουν 11 benchmarks. Χαρακτηριστικό αυτής της κατηγορίας είναι το Astar 2, το οποίο παρουσιάζεται.

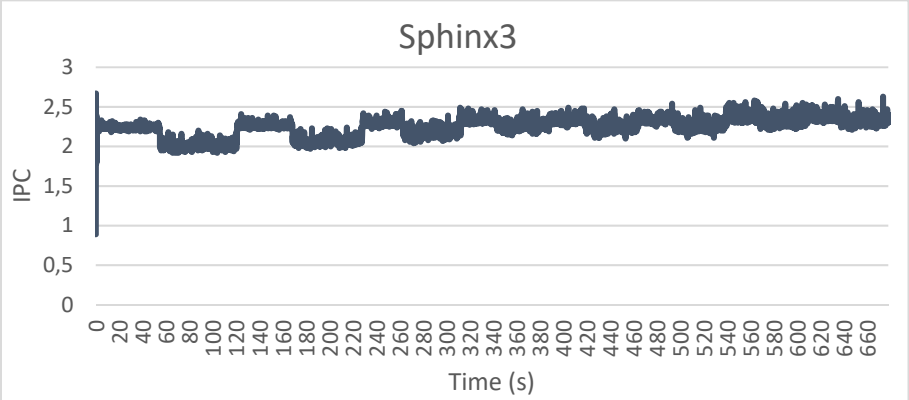


Γράφημα 6: Παράδειγμα Benchmark της Class C: Astar 2

Benchmark	IPC	Time (s)	MPKI	Util.	MB	W_95
Astar 1	1.03	170	0.13	96.4%	15.17	6
Astar 2	1.10	323	0.00	89.3%	0.68	2
Blackscholes	1.80	211	0.54	99.2%	108.50	1
Bwaves	2.23	425	12.98	99.8%	3620.46	2
Bzip2 2	1.93	40	0.03	93.5%	4.62	3
Bzip2 3	2.06	63	0.01	94.3%	2.44	4
Canneal	0.45	124	12.41	99.5%	474.93	13
Raytrace	2.25	227	0.61	99.1%	142.44	2
Soplex 1	1.24	132	0.07	94.1%	15.19	10
Soplex 2	1.56	109	19.50	99.7%	3142.60	3
Xalancbmk	1.90	239	0.33	97.9%	52.98	8

Πίνακας 3: Χαρακτηριστικά των Benchmarks της Class C

- Τα benchmarks της τέταρτης κατηγορίας (Class D) παρουσιάζουν περιοδικότητα, καθώς επαναλαμβάνουν ανά συγκεκριμένα χρονικά διαστήματα ορισμένες φάσεις εκτέλεσης. Στην κατηγορία αυτή ανήκουν 12 benchmarks. Παρουσιάζεται το γράφημα του Sphinx3.

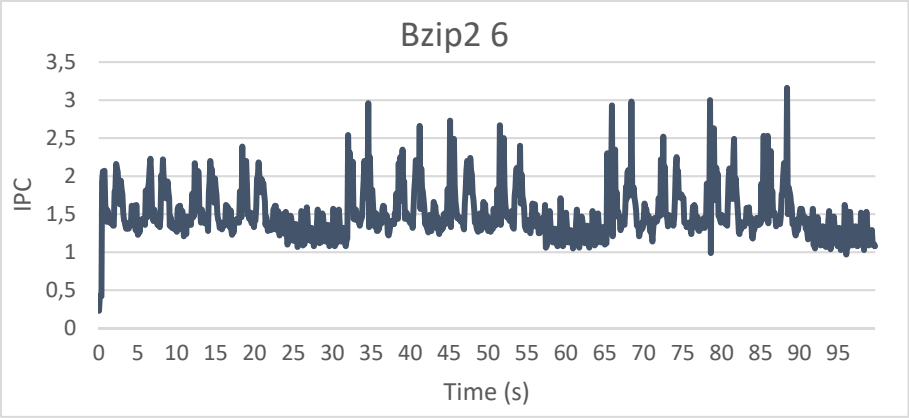


Γράφημα 7: Παράδειγμα Benchmark της Class D: Sphinx3

Benchmark	IPC	Time (s)	MPKI	Util.	MB	W_95
Bzip2 4	1.79	136	0.10	97.6%	40.95	2
Bzip2 5	1.82	144	0.07	97.8%	32.96	3
Dedup	1.47	44	0.26	88.7%	105.08	1
Gamess 1	2.76	181	0.00	20.1%	0.00	2
H264ref 2	2.37	62	0.00	31.9%	0.00	2
H264ref 3	2.40	556	0.00	84.5%	0.17	2
Mcf	0.5	284	26.30	99.4%	1377.85	7
Milc	1.19	466	23.39	99.5%	3220.14	2
Perlbench 2	2.61	64	0.06	96.1%	21.13	2
Perlbench 3	2.96	102	0.08	98.3%	41.47	2
Sphinx3	2.24	677	0.00	84.4%	0.10	3
Tonto	2.34	626	0.00	53.5%	0.00	2

Πίνακας 4: Χαρακτηριστικά των Benchmarks της Class D

- Στην πέμπτη κατηγορία (Class E) ανήκουν τα benchmarks, τα οποία δεν παρουσιάζουν κάποια κανονικότητα, όπως τα προηγούμενα benchmarks, και δεν εμπίπτουν σε καμία από τις παραπάνω κατηγορίες. Στην Class C ανήκουν 17 benchmarks. Το benchmark που παρουσιάζεται είναι το Freqmine από τη σουίτα Parsec 3.0.



Γράφημα 8: Παράδειγμα Benchmark της Class E: Freqmine

Benchmark	IPC	Time (s)	MPKI	Util.	MB	W_95
Bzip2 1	1.46	128	0.11	91.9%	39.20	2
Bzip2 6	1.5	99	0.11	97.8%	36.21	3
Freqmine	1.95	574	0.03	96.2%	5.17	1
Gcc 1	1.45	23	0.44	98.4%	84.98	6
Gcc 2	1.48	44	0.59	96.7%	68.01	4
Gcc 3	1.51	40	0.52	97.3%	96.74	4
Gcc 4	1.68	27	0.47	98.3%	93.50	5
Gcc 5	1.62	31	0.93	98.2%	143.21	6
Gcc 6	1.58	43	1.33	98.5%	181.67	8
Gcc 7	1.61	50	1.57	98.0%	170.44	12
Gcc 8	1.55	48	1.34	98.4%	196.20	9
Gcc 9	1.46	17	0.17	91.6%	8.24	2
Gobmk 1	1.22	85	0.00	39.7%	0.00	2
Gobmk 2	1.30	213	0.00	40.0%	0.00	2
Gobmk 3	1.35	106	0.00	40.2%	0.00	2
Gobmk 4	1.24	84	0.00	37.5%	0.00	1
Gobmk 5	1.30	114	0.00	37.8%	0.00	2

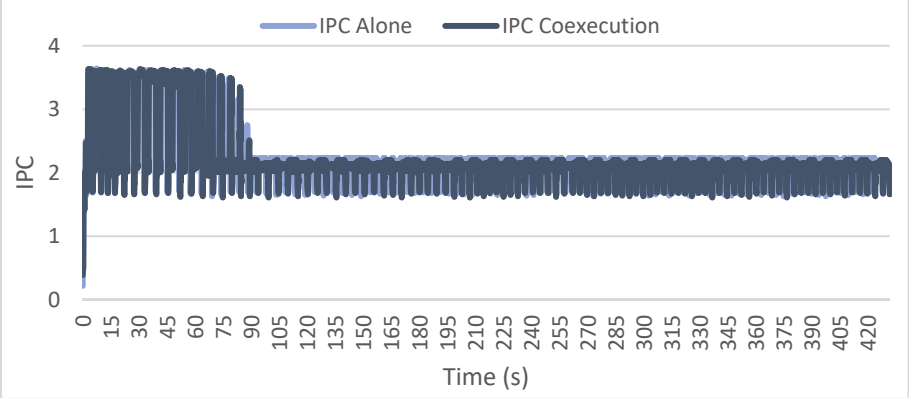
Πίνακας 5: Χαρακτηριστικά των Benchmarks της Class E

4.3 Συνεκτελέσεις των Benchmarks

Έχοντας ολοκληρώσει τη δημιουργία ενός προφίλ με τη χρήση της Intel CMT – CAT για κάθε ένα από τα benchmarks που χρησιμοποιούνται, το επόμενο στάδιο είναι να διαπιστωθεί η ύπαρξη και το μέγεθος του προαναφερθέντος προβλήματος που προκαλείται από τη συνεκτέλεση εφαρμογών στο ίδιο μηχάνημα και τον ανταγωνισμό για τους μοιραζόμενους πόρους. Για το σκοπό αυτό, δημιουργούνται ζευγάρια συνεκτελέσεων ως εξής. Στον Core 0 του μηχανήματος τοποθετείται προς εκτέλεση ένα benchmark που

θεωρούμε ότι έχει υψηλή προτεραιότητα, και η ομαλή εκτέλεση του αποτελεί πρωταρχικό στόχο. Στους υπόλοιπους 21 πυρήνες τοποθετούνται ισάριθμες εφαρμογές (instances) του ίδιου benchmark. Έτσι, προκύπτουν συνολικά $65 \times 65 = 4225$ ζευγάρια συνεκτέλεσης. Για κάθε ένα ζευγάρι μετράμε για το benchmark υψηλής προτεραιότητας, με την εφαρμογή PQoS, το μέσο IPC, το MPKI, το μέσο bandwidth, το χώρο στην cache και το χρόνο εκτέλεσης. Από τις μετρήσεις αυτές εξάγονται η επιβράδυνση του IPC (IPC Slowdown) και η χρονική επιβράδυνση (Slowdown) που προκύπτουν ως οι λόγοι του μετρούμενου IPC και χρόνου προς το IPC και το χρόνο (αντιστοίχως) που μετρήθηκαν για το benchmark υψηλής προτεραιότητας σε συνθήκες που εκτελείται μόνο του στο σύστημα (και είναι διαθέσιμες στην προηγούμενη ενότητα).

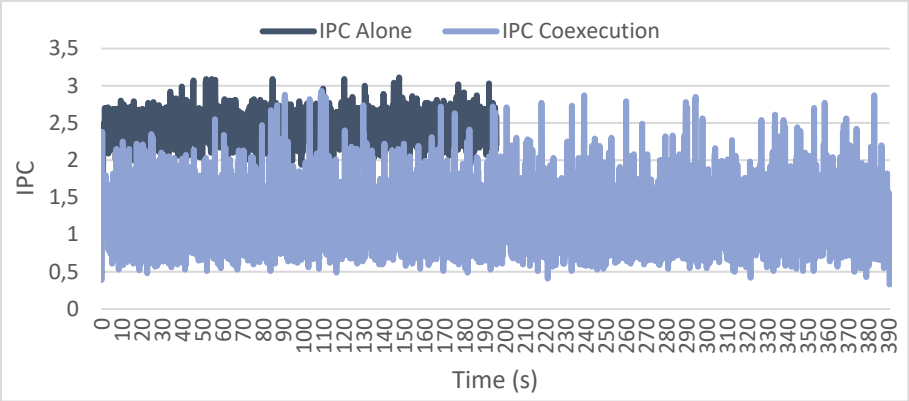
Όπως προαναφέρθηκε, η παρεμβολή μεταξύ των συνεκτελούμενων εφαρμογών και η επίδραση που έχει στην επίδοση τους εξαρτάται σε μεγάλο βαθμό από το είδος και τη συμπεριφορά των εφαρμογών που εκτελούνται στο μηχάνημα. Με βάση αυτό, είναι λογικό να παρατηρηθούν ζευγάρια benchmarks, στα οποία το benchmark υψηλής προτεραιότητας δεν επηρεάζεται από την εκτέλεση των benchmarks χαμηλής προτεραιότητας, συνεπώς ολοκληρώνει την εκτέλεση του σε χρόνο σχεδόν ίσο με το χρόνο ολοκλήρωσης σε συνθήκες που εκτελείται μόνο του στο μηχάνημα. Ένα παράδειγμα τέτοιου ζευγαριού είναι το Bwaves - Calculix, κατά την εκτέλεση του οποίου το bwaves (high priority) ολοκληρώνει την εκτέλεση του σε 431 sec, μόλις 1.4% πιο αργά από την περίπτωση που εκτελείται μόνο του στο σύστημα. Επομένως, το bwaves επηρεάζεται ελάχιστα από την εκτέλεση των 21 instances του calculix στους υπόλοιπους πυρήνες. Στο επόμενο γράφημα είναι εμφανής η πλήρης επικάλυψη των IPC της συνεκτέλεσης και της εκτέλεσης στην κατάσταση «Alone».



Γράφημα 9: Σύγκριση IPC στο ζευγάρι Bwaves - Calculix

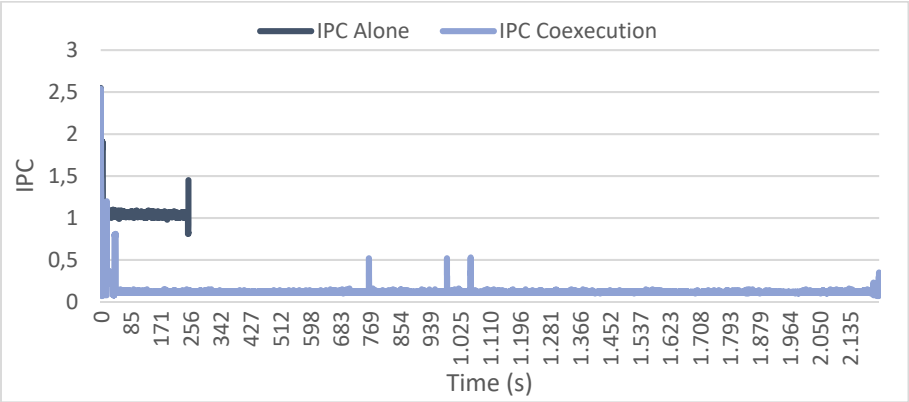
Από την άλλη, υπάρχουν ζευγάρια στα οποία το benchmark υψηλής προτεραιότητας επιβραδύνεται σημαντικά λόγω της εκτέλεσης των benchmarks χαμηλής προτεραιότητας. Ένα τέτοιο παράδειγμα αποτελεί το ζευγάρι Perlbench 1 - Lbm, στο οποίο το Perlbench 1 ολοκληρώνεται σε χρόνο 200% μεγαλύτερο από το βέλτιστο χρόνο εκτέλεσης του, αφού από 3′15″, ολοκληρώνεται σε 6′30″. Αντίστοιχα, το IPC κατά τη συνεκτέλεση είναι 1,23 το οποίο είναι το 50,2% του IPC που επιτυγχάνεται όταν το Perlbench 1 εκτελείται μόνο του στο σύστημα. Στο επόμενο γράφημα φαίνεται η μείωση της επίδοσης

του Perlbench 1, καθώς και η χρονική επιβράδυνση του σε σχέση με την κατάσταση «Alone».



Γράφημα 10: Σύγκριση Επιδόσεων στο ζευγάρι Perlbench 1 - Lbm

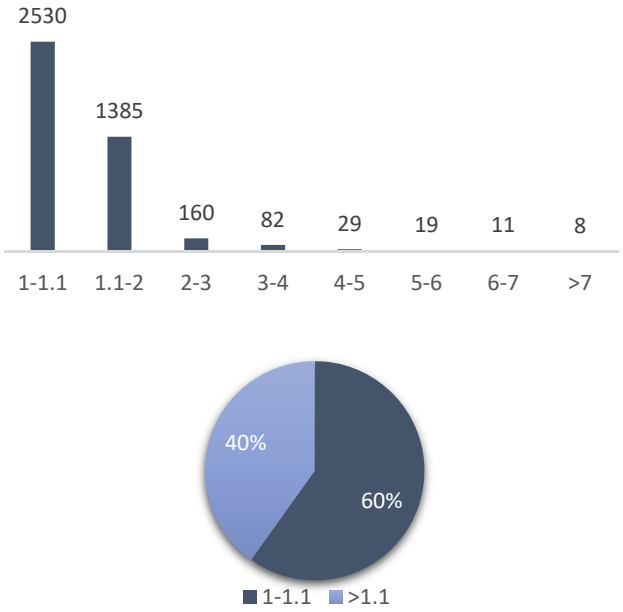
Η πιο ακραία περίπτωση που παρατηρήθηκε είναι το ζευγάρι Omnetpp – Lbm, στο οποίο το omnetpp ολοκληρώνεται σε 36′58′′, ενώ όταν εκτελείται μόνο του στο σύστημα, ολοκληρώνεται σε 4′10′′. Αυτό σημαίνει ότι επιβραδύνεται κατά 8.87 φορές. Αντίστοιχα το μέσο IPC κατά τη συγκεκριμένη συνεκτέλεση είναι 0,12, το οποίο είναι σχεδόν δέκα φορές μικρότερο από το IPC που αναμένεται όταν εκτελείται μόνο του (1,05). Από τη σύγκριση των IPC στο επόμενο γράφημα, γίνεται αντιληπτή η επίδραση που έχει η συνεκτέλεση των εφαρμογών στην επίδοση του omnetpp.



Γράφημα 11: Σύγκριση IPC στο ζευγάρι Omnetpp - Lbm

Για να γίνει ευκολότερη η επισκόπιση της επίδρασης των συνεκτελέσεων στην επίδοση των high priority benchmarks, τα ζευγάρια συνεκτέλεσης ομαδοποιήθηκαν με βάση τη χρονική επιβράδυνση (slowdown) του benchmark υψηλής προτεραιότητας σε 8 κατηγορίες. Στην πρώτη κατηγορία ανήκουν τα ζευγάρια στα οποία το slowdown είναι μεταξύ 1 και 1.1, και θεωρείται ότι η επίδραση της συνεκτέλεσης είναι αμελητέα. Αυτή η κατηγορία είναι η πολυπληθέστερη, καθώς σε αυτή ανήκουν 2.530 από τα 4.225 benchmarks. Αυτό σημαίνει ότι η επίδραση του ανταγωνισμού για πόρους και της συνεκτέλεσης στο 60% των ζευγαριών δε γίνεται ιδιαιτέρως αισθητή από τη high priority εφαρμογή. Οι επόμενες κατηγορίες περιέχουν τα ζευγάρια των οποίων το slowdown είναι από 1.1 εως 2, από 2 έως 3, και ούτω καθ' εξής μέχρι την τελευταία κατηγορία που έχει τα ζευγάρια με slowdown άνω του 7. Σε αυτές

τις κατηγορίες, στις οποίες το slowdown δεν είναι ευκαταφρόνητο, και μπορεί να προκαλέσει αισθητή μείωση της ποιότητας υπηρεσιών (QoS), ανήκει το υπόλοιπο 40%.



Γράφημα 12: Πλήθος Benchmarks ανά κατηγορία

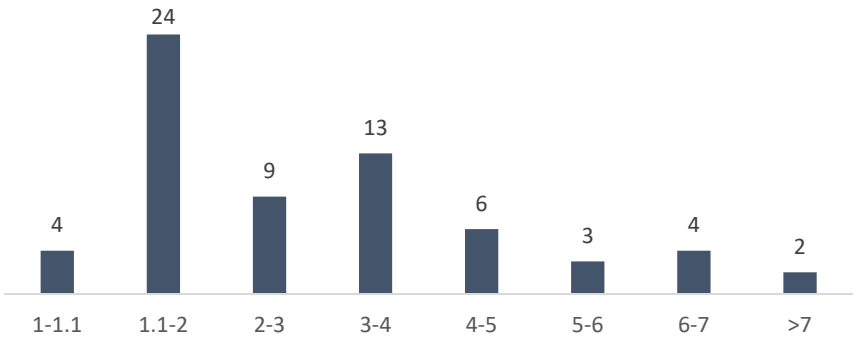
Στον πίνακα που ακολουθεί παρουσιάζεται ένα υποσύνολο των μετρήσεων που λήφθηκαν για τα 4.225 ζευγάρια στην κατάσταση Full Cache. Συγκεκριμένα, σε κάθε benchmark αντιστοιχούν 65 ζευγάρια, στα οποία το benchmark έχει υψηλή προτεραιότητα και κάθε ένα από τα 65 benchmarks έχει χαμηλή προτεραιότητα. Στο υποσύνολο που θα μελετηθεί τοποθετήθηκε για κάθε benchmark το ζευγάρι αυτό, στο οποίο παρατηρήθηκε η μεγαλύτερη χρονική επιβράδυνση (slowdown) για το benchmark αυτό (ως high priority benchmark), κατά τη μέτρηση της προηγούμενης ενότητας. Έτσι προκύπτει το υποσύνολο των 65 ζευγαριών, τα οποία είναι οι χειρότερες περιπτώσεις που παρουσιάστηκαν σε κάθε benchmark υψηλής προτεραιότητας. Ο πίνακας περιέχει τις ακόλουθες στήλες:

- *High Priority*: Το όνομα του high priority benchmark του ζευγαριού.
- *Low Priority*: Το όνομα του low priority benchmark του ζευγαριού.
- *IPC Slowdown*: Ο λόγος του IPC του high priority benchmark στην κατάσταση Full Cache προς το αντίστοιχο IPC στην κατάσταση «Alone».
- *Time Slowdown*: Η χρονική επιβράδυνση του high priority benchmark, δηλαδή ο λόγος του χρόνου εκτέλεσής του στην κατάσταση Full Cache προς το χρόνο εκτέλεσης στην κατάσταση «Alone».
- *MPKI*: Ο αριθμός αστοχιών ανά χιλιάδα εντολών του high priority benchmark κατά την εκτέλεση του ζευγαριού.
- *Memory Bandwidth*: Η μέτρηση του bandwidth του high priority benchmark.
- *Cache Degradation*: Ο λόγος του χώρου της cache που καταλαμβάνει το high priority benchmark σε σχέση με το χώρο στην κατάσταση Alone δηλαδή το χώρο που χρειάζεται στην ιδανική περίπτωση.

High Priority	Low Priority	IPC Sl.	T. Sl.	MPKI	MB	Degrad.
Omnetpp	Lbm	11.3%	8.87	88.0	327.29	1.5%
Libquantum	Lbm	12.0%	8.39	122.5	1319.78	4.6%
Bzip2 3	Leslie3d	15.0%	6.75	1098.3	370.61	1.3%
Mcf	Lbm	15.5%	6.27	2.6	541.45	2.9%
Soplex 1	Leslie3d	16.1%	6.23	436.5	192.14	0.6%
Canneal	Lbm	16.9%	6.06	2.4	205.64	1.0%
Xalancbmk	Lbm	18.1%	5.55	31.4	344.62	1.6%
Bzip2 2	Leslie3d	18.8%	5.46	270.8	294.60	0.9%
Streamcluster	Lbm	18.3%	5.44	1.3	261.34	1.1%
Milc	Lbm	20.2%	4.91	1.1	828.15	2.8%
Gcc 1	Lbm	21.6%	4.75	34.0	497.10	2.2%
Lbm	Lbm	22.5%	4.45	1.8	1437.10	4.5%
Gcc 7	Leslie3d	22.9%	4.38	11.8	741.40	1.7%
Soplex 2	Lbm	24.3%	4.15	1.1	1169.50	4.5%
Sphinx3	Leslie3d	25.0%	4.01	28602.3	841.01	3.5%
GemsFDTD	Lbm	25.5%	3.94	0.9	1322.98	5.3%
Gcc 8	Leslie3d	25.8%	3.93	8.2	680.76	1.7%
Leslie3d	Lbm	25.8%	3.89	3.2	1529.70	6.4%
CactusADM	Leslie3d	26.8%	3.73	2.0	305.40	0.9%
Astar 1	Leslie3d	26.9%	3.73	117.5	653.68	2.1%
Bzip2 5	Leslie3d	27.3%	3.67	65.0	258.03	0.9%
Gcc 6	Leslie3d	28.9%	3.50	7.2	434.70	1.4%
Gcc 2	Leslie3d	28.9%	3.48	13.3	336.40	1.1%
Gcc 4	Lbm	29.6%	3.41	12.7	360.06	1.7%
Gcc 5	Leslie3d	29.8%	3.36	9.1	530.25	1.4%
Bzip2 6	Leslie3d	30.6%	3.30	48.6	177.78	0.7%
Gcc 3	Leslie3d	33.1%	3.08	15.0	528.82	1.9%
Bzip2 1	Leslie3d	32.7%	3.07	41.9	139.43	0.5%
Zeusmp	Lbm	34.2%	2.88	1.2	490.20	2.0%
Gcc 9	Leslie3d	36.3%	2.82	33.0	263.22	1.1%
Bzip2 4	Leslie3d	40.3%	2.48	28.9	113.21	0.4%
x264	Lbm	57.7%	2.40	3.4	263.86	1.6%
Bwaves	Bwaves	41.8%	2.40	0.7	1695.40	4.5%
Rtview	Astar 1	89.9%	2.23	1.6	187.54	1.7%
Calculix	Streamcluster	97.0%	2.07	9.8	133.33	0.9%
Astar 2	Bwaves	49.1%	2.04	1920.6	281.59	1.1%
Perlbench 1	Lbm	50.2%	2.00	351.8	139.55	0.9%
Wrf	Bwaves	54.0%	1.85	2.4	1044.97	3.7%
Sjeng	Leslie3d	54.2%	1.85	5.3	79.25	0.2%
H264ref 1	Leslie3d	60.1%	1.70	12842.7	1236.47	7.5%
Ferret	Lbm	61.2%	1.63	18.5	191.42	1.0%
Gobmk 1	Leslie3d	64.3%	1.62	3226.3	142.27	1.2%
Gobmk 4	Lbm	67.8%	1.58	2052.4	163.25	2.7%
Perlbench 3	Lbm	64.5%	1.56	9.4	255.41	1.0%
Gobmk 3	Leslie3d	65.3%	1.55	2184.7	451.39	2.7%
Gobmk 2	Leslie3d	66.9%	1.53	1348.8	143.09	3.1%
Tonto	Bwaves	67.1%	1.49	10564.1	416.38	3.7%
Fluidanimate	Lbm	67.9%	1.48	1.7	354.44	1.4%

High Priority	Low Priority	IPC Sl.	T. Sl.	MPKI	MB	Degrad.
H264ref 2	Leslie3d	69.8%	1.46	9110.7	779.60	9.0%
Dedup	Gcc 9	77.2%	1.39	1.2	86.54	1.7%
H264ref 3	Leslie3d	72.4%	1.38	550.3	321.05	1.3%
Gobmk 5	Leslie3d	75.5%	1.34	2324.9	302.63	2.5%
Freqmine	Hmmer 1	97.9%	1.30	19.5	73.60	4.8%
Gamess 3	Leslie3d	81.2%	1.24	1684.3	85.20	1.2%
Bodytrack	Leslie3d	80.6%	1.22	35.6	150.58	1.2%
Gromacs	Bwaves	84.0%	1.20	11271.8	102.86	4.3%
Gamess 2	CactusADM	88.6%	1.20	269.5	15.82	1.0%
Povray	Leslie3d	85.5%	1.17	2596.4	25.22	0.5%
Gamess 1	Leslie3d	86.9%	1.16	2434.6	71.24	2.3%
Hmmer 1	Leslie3d	88.9%	1.13	1854.5	618.40	3.3%
Perlbench 2	Leslie3d	89.3%	1.13	4.5	74.75	0.5%
Blackscholes	Astar 1	90.7%	1.10	1.1	104.71	0.8%
Swaptions	Libquantum	92.3%	1.09	613.1	39.83	1.0%
Namd	GemsFDTD	92.4%	1.08	233.7	71.00	0.5%
Hmmer 2	Bwaves	95.5%	1.05	1724.2	125.83	13.5%

Πίνακας 6: Χαρακτηριστικά των High Priority Benchmarks (Full Cache)



Γράφημα 13: Ταξινόμηση των Ζευγαριών σε Κατηγορίες Slowdown

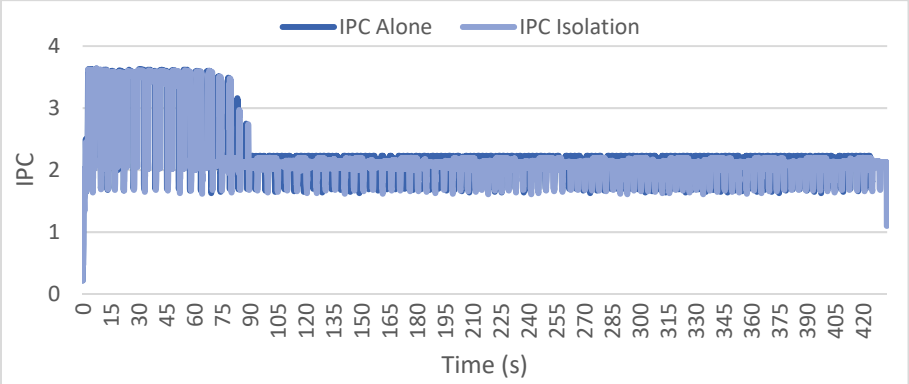
Στο προηγούμενο γράφημα παρουσιάζεται η κατανομή των 65 ζευγαριών στις κατηγορίες που ορίστηκαν με βάση το slowdown. Όπως φαίνεται, η επιβράδυνση μπορεί να θεωρηθεί μικρή μόνο σε 4 περιπτώσεις, ενώ σε 37 περιπτώσεις είναι μεγαλύτερη από 100%.

4.4 Κατάσταση Isolation – Take Over

Εφόσον η επιβράδυνση που παρατηρείται στα high priority benchmarks οφείλεται στον ανταγωνισμό των κοινόχρηστων πόρων, μεταξύ των οποίων και η cache, μία λύση που μπορεί να δοθεί από την Intel CAT για την αντιμετώπιση της επιβράδυνσης είναι η παραχώρηση του μεγαλύτερου δυνατού μέρους της cache στο high priority benchmark. Συγκεκριμένα, στο σύστημα που εξετάζουμε, το οποίο υπενθυμίζεται πως έχει 20 ways of associativity, παρέχονται 19 ways στο high priority benchmark και 1 way σε όλα τα low priority benchmarks. Με αυτόν τον καταμερισμό, εξαλείφεται εντελώς ο ανταγωνισμός για χώρο στην cache για το high priority benchmark, αφού έχει πλήρη και αποκλειστική πρόσβαση σε 19 ways. Ο χώρος αυτός είναι ο μεγαλύτερος που είναι δυνατό να παραχωρηθεί, ώστε να ικανοποιηθούν πλήρως οι ανάγκες του high priority benchmark για χώρο στην cache, με σκοπό

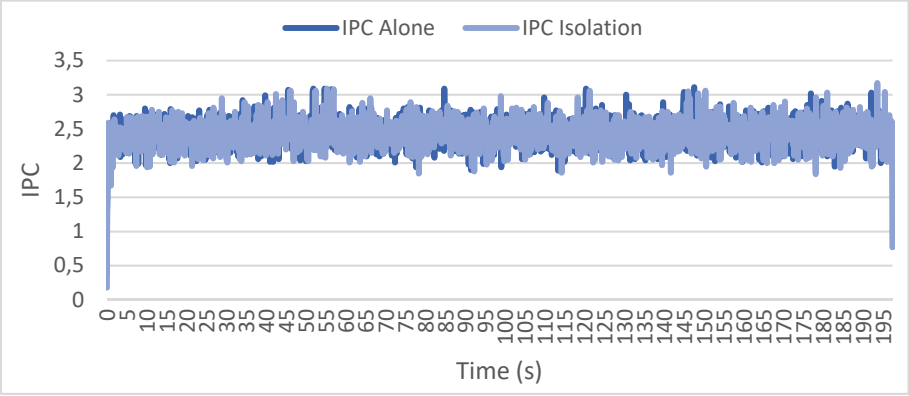
την αύξηση της επίδοσης του. Ονομάζουμε αυτήν την κατάσταση εκτέλεσης, κατάσταση Isolation – Take Over.

Η εκτέλεση των τριών ζευγαριών, που παρουσιάστηκαν στην προηγούμενη ενότητα, στην κατάσταση Isolation – Take Over βελτιώνει σημαντικά την επίδοση των high priority benchmarks. Πιο συγκεκριμένα, στην περίπτωση του ζευγαριού Bwaves – Calculix, οι καταστάσεις Full Cache και Isolation – Take Over δεν έχουν διαφορά, καθώς η επίδοση του Bwaves ήταν ήδη πολύ κοντά στη βέλτιστη. Έτσι, στην κατάσταση Isolation – Take Over, η επίδοση του Bwaves είναι στο ίδιο ακριβώς επίπεδο τόσο με την επίδοση της κατάστασης «Alone» όσο και με αυτή της κατάστασης Full Cache. Το προηγούμενο είναι εμφανές στο παρακάτω γράφημα.



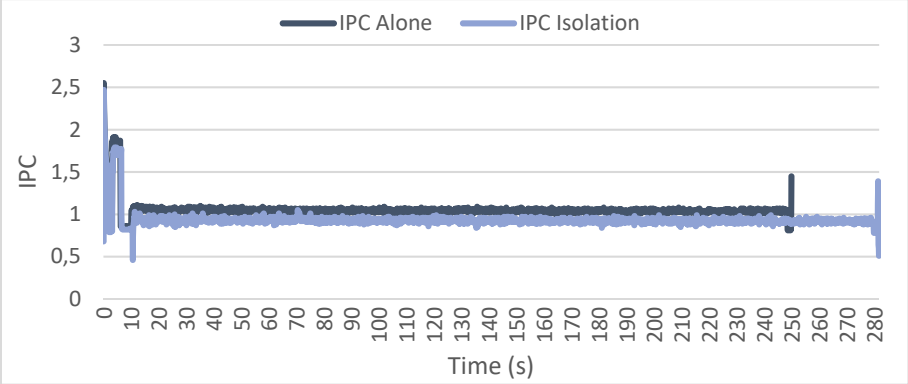
Γράφημα 14: Σύγκριση Επιδόσεων στο Ζευγάρι Bwaves – Calculix

Η εκτέλεση του ζευγαριού Perlbench 1 – Lbm προκαλούσε στο Perlbench διπλασιασμό του χρόνου εκτέλεσης του. Η μετάβαση στην κατάσταση Isolation – Take Over δίνει λύση σε αυτό το πρόβλημα, καθώς το Perlbench ολοκληρώνεται σε χρόνο μόλις 0,8% μεγαλύτερο από το βέλτιστο χρόνο της κατάστασης Alone. Στο επόμενο γράφημα φαίνεται ότι η επίδοση είναι ακριβώς στο επίπεδο της επίδοσης στην κατάσταση Alone.



Γράφημα 15: Σύγκριση Επιδόσεων στο Ζευγάρι Perlbench 1 – Lbm

Η επιβράδυνση του Omnetpp κατά την εκτέλεση του ζευγαριού Omnetpp – Lbm είναι η μεγαλύτερη που παρατηρήθηκε, καθώς ο χρόνος εκτέλεσης ήταν κατά 787% μεγαλύτερος του βέλτιστου χρόνου. Σε αυτήν την περίπτωση, η εκτέλεση στην κατάσταση Isolation – Take Over βελτιώνει αισθητά το χρόνο εκτέλεσης του Omnetpp, ο οποίος είναι 13% μεγαλύτερος του βέλτιστου.



Γράφημα 16: Σύγκριση Επιδόσεων στο Ζευγάρι Omnetpp - Lbm

Στον πίνακα που ακολουθεί φαίνονται οι μετρήσεις για κάθε ζευγάρι του υποσυνόλου που παρουσιάστηκε στην προηγούμενη ενότητα. Οι στήλες του πίνακα έχουν διατηρηθεί ίδιες με τον πίνακα της προηγούμενης ενότητας. Η μόνη διαφορά παρατηρείται στην τελευταία στήλη Utilization, η οποία δείχνει το ποσοστό χρήσης του χώρου στην cache που παρέχεται στο high priority benchmark, ο οποίος σε όλες τις περιπτώσεις είναι 53.504kB.

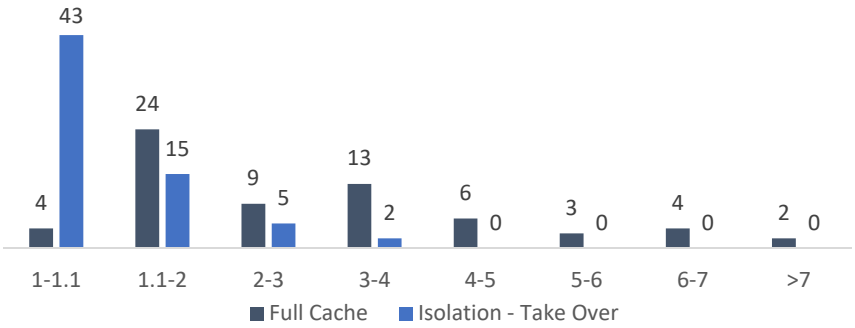
High Priority	Low Priority	IPC Sl.	Time Sl.	MPKI	MBL	Util.
Omnetpp	Lbm	88.9%	1.13	1.46	11.69	99.6%
Libquantum	Lbm	71.4%	1.40	2.54	0.93	80.4%
Bzip2 3	Leslie3d	98.3%	1.03	1.37	4.32	97.1%
Mcf	Lbm	50.1%	2.01	0.98	688.97	99.5%
Soplex 1	Leslie3d	97.3%	1.03	1.15	20.98	93.3%
canneal1	Lbm	41.6%	2.43	1.06	209.89	100.0%
xalancbmk1	Lbm	87.8%	1.16	1.14	51.23	99.3%
Bzip2 2	Leslie3d	99.5%	1.04	1.17	5.96	96.0%
Streamcluster	Lbm	30.8%	3.26	1.04	306.71	99.9%
Milc	Lbm	29.8%	3.37	0.98	1003.62	100.0%
Gcc 1	Lbm	94.3%	1.09	1.05	92.97	99.6%
Lbm	Lbm	34.9%	2.87	0.96	992.97	100.0%
Gcc 7	Leslie3d	83.7%	1.20	1.09	167.99	99.6%
Soplex 2	Lbm	40.8%	2.47	0.85	1315.02	99.9%
Sphinx3	Leslie3d	100.2%	1.00	11.53	0.48	96.0%
GemsFDTD	Lbm	42.0%	2.38	0.85	2027.57	100.0%
Gcc 8	Leslie3d	88.8%	1.14	1.05	180.13	99.7%
Leslie3d	Lbm	62.2%	1.61	1.02	185.93	99.5%
CactusADM	Leslie3d	72.7%	1.38	0.89	428.94	100.0%
Astar 1	Leslie3d	96.8%	1.04	1.14	15.76	99.5%
Bzip2 5	Leslie3d	98.2%	1.02	1.00	33.09	99.9%
Gcc 6	Leslie3d	83.9%	1.20	1.04	165.69	99.6%
Gcc 2	Leslie3d	87.1%	1.15	1.07	79.49	98.8%
Gcc 4	Lbm	89.9%	1.13	1.08	93.98	99.6%
Gcc 5	Leslie3d	88.9%	1.13	1.07	136.12	98.7%
Bzip2 6	Leslie3d	98.3%	1.03	0.98	36.39	99.9%
Gcc 3	Leslie3d	92.6%	1.10	1.03	91.31	99.5%
Bzip2 1	Leslie3d	98.3%	1.02	1.01	38.89	99.8%
Zeusmp	Lbm	53.4%	1.88	0.86	548.37	100.0%

High Priority	Low Priority	IPC Sl.	Time Sl.	MPKI	MBL	Util.
Gcc 9	Leslie3d	94.5%	1.09	1.35	22.96	96.3%
Bzip2 4	Leslie3d	98.5%	1.02	0.99	39.67	99.9%
x2641	Lbm	83.8%	1.20	1.12	81.25	95.7%
Bwaves	Bwaves	92.4%	1.08	0.85	3220.38	100.0%
rtview1	Astar 1	95.9%	1.04	1.00	141.48	99.9%
Calculix	Streamcluster	99.7%	1.01	1.14	19.34	99.8%
Astar 2	Bwaves	98.3%	1.02	3.33	0.93	96.6%
Perlbench 1	Lbm	99.4%	1.01	2.16	2.70	98.6%
Wrf	Bwaves	98.4%	1.02	1.01	505.63	100.0%
Sjeng	Leslie3d	91.4%	1.10	1.05	38.62	100.0%
H264ref 1	Leslie3d	99.7%	1.02	21.31	0.37	79.7%
ferret1	Lbm	96.1%	1.04	1.29	3.98	99.1%
Gobmk 1	Leslie3d	100.9%	1.00	13.84	0.48	95.3%
Gobmk 4	Lbm	100.3%	1.00	11.68	0.12	65.4%
Perlbench 3	Lbm	97.6%	1.03	1.01	43.77	99.8%
Gobmk 3	Leslie3d	101.5%	1.00	7.95	0.05	74.1%
Gobmk 2	Leslie3d	99.7%	1.01	6.69	0.11	92.2%
Tonto	Bwaves	99.2%	1.01	23.09	0.10	78.8%
fluidanimate1	Lbm	83.3%	1.20	0.99	226.99	100.0%
H264ref 2	Leslie3d	101.0%	1.01	26.44	0.27	76.2%
dedup1	Gcc 9	101.7%	1.10	0.93	102.93	98.4%
H264ref 3	Leslie3d	99.3%	1.01	4.60	1.28	86.4%
Gobmk 5	Leslie3d	101.2%	1.00	12.33	0.21	57.5%
freqmine1	Hmmer 1	99.6%	1.01	1.28	4.71	99.7%
Gamess 3	Leslie3d	99.6%	1.01	9.41	0.06	28.0%
bodytrack1	Leslie3d	99.6%	1.02	1.16	5.56	99.1%
Gromacs	Bwaves	99.8%	1.01	62.65	0.04	92.6%
Gamess 2	CactusADM	100.5%	1.00	8.21	0.07	11.8%
Povray	Leslie3d	99.7%	1.01	32.35	0.05	91.1%
Gamess 1	Leslie3d	99.2%	1.01	21.16	0.04	9.4%
Hmmer 1	Leslie3d	99.5%	1.01	4.56	0.61	95.8%
Perlbench 2	Leslie3d	98.1%	1.03	1.05	24.33	99.0%
blackscholes1	Astar 1	99.3%	1.00	1.01	108.35	99.8%
swaptions1	Libquantum	99.5%	1.01	29.19	0.01	92.9%
Namd	GemsFDTD	100.0%	1.00	3.99	0.12	96.3%
Hmmer 2	Bwaves	99.8%	1.00	8.23	0.01	89.1%

Πίνακας 7: Χαρακτηριστικά των High Priority Benchmarks (Isolation)

Όπως φαίνεται στο επόμενο γράφημα, στο οποίο ομαδοποιούνται τα αποτελέσματα του προηγούμενου πίνακα, η κατάσταση Isolation – Take Over δεν αντιμετωπίζει επιτυχώς όλες τις περιπτώσεις, καθώς υπάρχουν ζευγάρια στα οποία το high priority benchmark συνεχίζει να έχει αξιοσημείωτη καθυστέρηση. Αυτό συμβαίνει διότι υπάρχουν αρκετοί ακόμα μοιραζόμενοι πόροι, και ο ανταγωνισμός για τη χρήση τους συνεχίζει να επηρεάζει το high priority benchmark. Επίσης είναι σημαντική η παρατήρηση ότι η μέση χρησιμοποίηση της cache στις περιπτώσεις που η επιβράδυνση αντιμετωπίζεται (οι περιπτώσεις της κατηγορίας 1-1.1) είναι ίση με 87,9%.

Επομένως, τα high priority benchmarks δε χρησιμοποιούν όλη την cache που τους παρέχεται.



Γράφημα 17: Ταξινόμηση των Ζευγαριών σε Isolation – Take Over

Όπως φαίνεται στο προηγούμενο γράφημα, η εκτέλεση των ζευγαριών στην κατάσταση Isolation – Take Over δεν προσφέρει λύσεις σε όλες τις περιπτώσεις, καθώς σε 22 περιπτώσεις η επιβράδυνση είναι ακόμα αισθητή, ενώ σε 7 από αυτές συνεχίζει να είναι άνω του 100%. Αυτό οφείλεται στο γεγονός ότι συνεχίζει να υπάρχει ανταγωνισμός για άλλους κρίσιμους κοινόχρηστους πόρους, τους οποίους δε διαχειρίζεται το σύστημα, όπως η πρόσβαση στο δίαυλο δεδομένων. Όσον αφορά την αντιμετώπιση του ζητήματος με διαχείριση του καταμερισμού της cache, και εφόσον δεν πραγματοποιείται καμία αλλαγή στα ζευγάρια συνεκτέλεσης, αυτή η κατάσταση είναι και η βέλτιστη εφικτή.

Η επιβολή αυτής της πολιτικής διαμοιρασμού αποτελεί μια λύση του προβλήματος, καθώς όλα ανεξαιρέτως τα benchmarks του επιλεγμένου υποπροβλήματος παρουσίασαν μικρότερη χρονική επιβράδυνση σε σχέση με την περίπτωση που τα benchmarks έχουν πρόσβαση σε όλη την cache χωρίς την επιβολή πολιτικής καταμερισμού. Τα μειονεκτήματα αυτής της λύσης όμως είναι η μεγάλη επιβράδυνση που προκαλείται στα benchmarks χαμηλής προτεραιότητας, καθώς περιορίζονται στη χρήση μόνο 2.816kB στην cache (1 way), σε συνδυασμό με το γεγονός ότι αρκετά από τα benchmarks υψηλής προτεραιότητας θα μπορούσαν να έχουν αντίστοιχο επίπεδο επίδοσης με την κατάληψη λιγότερου χώρου στην cache. Τα μειονεκτήματα αυτά αποτελούν κίνητρο για την ανάπτυξη ενός δυναμικού μηχανισμού, ο οποίος θα είναι σε θέση να μεταβάλλει την κατανομή της cache μεταξύ high και low priority εφαρμογών ώστε να εξασφαλίζει την επίδοση της high priority εφαρμογής, ενώ ταυτόχρονα θα παρέχει στις low priority εφαρμογές το χώρο που δεν είναι απαραίτητος στην high priority εφαρμογή ώστε να βελτιωθεί η επίδοση τους, και ως αποτέλεσμα να βελτιωθεί η επίδοση και το throughput του συστήματος.

5. Δυναμική Προστασία Επίδοσης

5.1 Ο Μηχανισμός Δυναμικής Προστασίας Επίδοσης

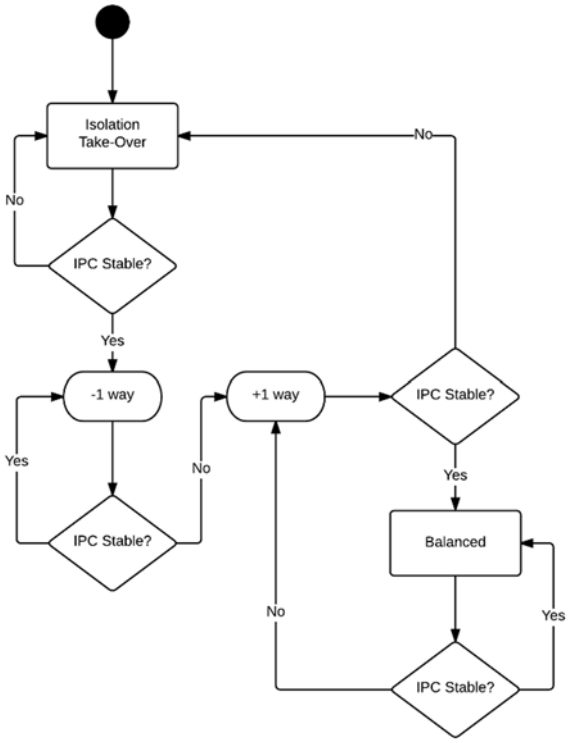
Ο στόχος αρκετών από τις προτάσεις που παρουσιάστηκαν και αναλύθηκαν στο 2ο κεφάλαιο είναι η παροχή του απαιτούμενου QoS και ο περιορισμός της παρεμβολής στην επίδοση, που παρατηρείται μεταξύ συνεκτελούμενων εφαρμογών σε ένα CMP. Η δημιουργία ενός ελεγκτή που θα εξασφαλίζει την επίτευξη της επιθυμητής επίδοσης για μία εφαρμογή, η οποία λογίζεται ως υψηλής προτεραιότητας, αποτέλεσε αντικείμενο μελέτης των [3], [8] και [19]. Οι έρευνες αυτές, όμως, δεν είχαν στη διάθεση τους την υποστήριξη που παρέχουν οι τεχνολογίες Intel CMT – CAT τόσο σε hardware όσο και σε software επίπεδο.

Συνδυάζοντας τη δυνατότητα παρακολούθησης της χρήσης πόρων που παρέχεται από την Intel CMT με τη δυνατότητα επιβολής της κατανομής που παρέχεται από την Intel CAT, μπορεί να ελεγχθεί δυναμικά και σε χρόνο εκτέλεσης η επίδοση και η πρόσβαση σε πόρους μιας εφαρμογής με υψηλή προτεραιότητα (high priority εφαρμογή). Στις ενότητες που ακολουθούν προτείνουμε, αναλύουμε, υλοποιούμε και αξιολογούμε το Μηχανισμό Δυναμικής Προστασίας Επίδοσης, που βασίζεται στο συνδυασμό των δύο τεχνολογιών.

5.1.1 Δομή και Λειτουργία

Σκοπός του μηχανισμού προστασίας είναι να επιτρέψει σε μια εφαρμογή, που έχει οριστεί να εκτελείται σε έναν πυρήνα και έχει υψηλή προτεραιότητα, να μείνει όσο το δυνατό ανεπηρέαστη από τις εφαρμογές που εκτελούνται στους υπόλοιπους πυρήνες του συστήματος. Ο μηχανισμός που προτείνεται, δρα δυναμικά και αλλάζει κατά το χρόνο εκτέλεσης την κατανομή της cache μεταξύ των εφαρμογών. Με αυτήν την πολιτική εξασφαλίζει ότι παρέχει τον απαραίτητο χώρο στην high priority εφαρμογή, αποδίδοντας παράλληλα τον περιττό χώρο στις υπόλοιπες εφαρμογές, ώστε να εκμεταλλευτούν την cache κατά την εκτέλεση τους.

Στη γενική περίπτωση η παροχή περισσότερου χώρου στην cache ευνοεί την επίδοση μιας εφαρμογής. Επομένως, εάν επιλεγόταν η στατική κατανομή για την προστασία μιας εφαρμογής, θα παρεχόταν σε αυτή το μεγαλύτερο δυνατό μέρος της cache, και θα επιτρεπόταν στις υπόλοιπες εφαρμογές να χρησιμοποιήσουν ένα μικρό κομμάτι της. Από το σημείο αυτό εκκινεί και ο μηχανισμός προστασίας. Αν η διαθέσιμη cache έχει N ways of associativity, ο μηχανισμός την χωρίζει κατανέμοντας $N - 1$ ways στη high priority εφαρμογή και 1 way στις low priority εφαρμογές. Κατόπιν, λαμβάνει μετρήσεις του IPC ανά ένα προκαθορισμένο χρονικό διάστημα Δt. Όταν συγκεντρωθούν n μετρήσεις του IPC, δηλαδή ανά χρονικό διάστημα $n * \Delta t$, ο μηχανισμός καλείται να λάβει απόφαση σχετικά με τη σταθερότητα στην επίδοση της high priority εφαρμογής. Όσο η επίδοση κρίνεται σταθερή, μειώνεται ο χώρος που παρέχεται στη high priority εφαρμογή. Αντίθετα, ο εντοπισμός αστάθειας οδηγεί στην αύξηση του χώρου που παρέχεται στη high priority εφαρμογή. Η λειτουργία του μηχανισμού συνοψίζεται στο επόμενο διάγραμμα ροής.



Εικόνα 9: Διάγραμμα Ροής του Μηχανισμού Προστασίας

Στο διάγραμμα φαίνονται η αρχική κατάσταση (Isolation – Take Over) στην οποία η high priority εφαρμογή έχει πρόσβαση σε $N-1$ ways, και οι low priority εφαρμογές έχουν πρόσβαση στο 1 way που απομένει. Τα blocks αποφάσεων «IPC Stable?» αντιπροσωπεύουν τις αποφάσεις που λαμβάνει ο μηχανισμός σχετικά με τη σταθερότητα της επίδοσης της high priority εφαρμογής. Σκοπός του μηχανισμού είναι να φτάσει από την αρχική κατάσταση στην κατάσταση Balanced, στην οποία η high priority εφαρμογή χρησιμοποιεί τον ελάχιστο χώρο της cache, ο οποίος της είναι απαραίτητος για να εκτελεστεί χωρίς να επηρεάζεται αρνητικά η επίδοση της, ενώ ο υπόλοιπος χώρος παρέχεται στις low priority εφαρμογές.

5.1.2 Ανάλυση του Αλγορίθμου Λήψης Αποφάσεων

Για τη λήψη των αποφάσεων, ο μηχανισμός χρησιμοποιεί το IPC ως μετρική της επίδοσης. Επειδή όμως το IPC είναι μια μετρική η οποία είναι πολύ πιθανό να εμφανίσει διακυμάνσεις, ορίζεται ένα ποσοστό επίτρεψης α, το οποίο είναι το επιτρεπτό ποσοστό διακύμανσης μεταξύ δύο τιμών IPC, ώστε αυτές να θεωρούνται ίσες. Πιο συγκεκριμένα, δύο τιμές IPC_A και IPC_B θεωρούνται ίσες αν ισχύει:

$$(1-\alpha) * IPC_A \leq IPC_B \leq (1+a) * IPC_A$$

Για τη λήψη των αποφάσεων χρησιμοποιούνται δύο λίστες. Η μία έχει μήκος n και χρησιμοποιείται για την αποθήκευση των μετρήσεων του IPC στο αντίστοιχο διάστημα $n * \Delta t$. Η δεύτερη λίστα, η οποία έχει σταθερό μήκος 3, περιέχει τις 3 μέσες τιμές των τριών τελευταίων n −άδων μετρήσεων. Οι τρεις αυτές τιμές ονομάζονται, από τη νεότερη στην παλαιότερη, IPC_0, IPC_1 και IPC_2. Σε κάθε μία από τις μέσες τιμές που αποθηκεύονται αντιστοιχεί μία απόφαση σχετικά με τη σταθερότητα της επίδοσης της εφαρμογής. Οι αποφάσεις

ονομάζονται s_0, s_1 και s_2, σε αντιστοιχία με τις μέσες τιμές, δηλαδή από την νεότερη στην παλαιότερη. Μία απόφαση έχει τιμή 1 αν η επίδοση θεωρείται σταθερή, και τιμή 0 αν η επίδοση θεωρείται ασταθής.

Σε κάθε χρονικό διάστημα $n * \Delta t$ ολοκληρώνεται μία n-άδα μετρήσεων. Ο μηχανισμός πρώτα υπολογίζει την IPC_0, δηλαδή τη μέση τιμή της n-άδας των μετρήσεων. Ακολούθως, λαμβάνεται η απόφαση s_0 που αντιστοιχεί στη μέση τιμή. Ο μηχανισμός συγκρίνει αρχικά την IPC_0 με την IPC_1.

- Εάν οι δύο τιμές θεωρούνται ίσες, η επίδοση θεωρείται σταθερή. Εξαίρεση αποτελεί η περίπτωση στην οποία η IPC_0 δε θεωρείται ίση με την IPC_2 και οι προηγούμενες αποφάσεις είναι $s_2 = 1$ και $s_1 = 0$. Συγκεκριμένα, η επίδοση θεωρήθηκε σταθερή κατά τον υπολογισμό της IPC_2, αλλά ασταθής κατά τον υπολογισμό της IPC_1, και η προσθήκη χώρου στην cache (που επιβάλλεται από την αστάθεια που παρατηρήθηκε) δεν επανέφερε την επίδοση στο επίπεδο της IPC_2. Επομένως, η επίδοση θεωρείται ασταθής.
- Εάν οι δύο τιμές δε θεωρούνται ίσες, τότε η επίδοση δε θεωρείται σταθερή. Η εξαίρεση σε αυτήν την περίπτωση εντοπίζεται όταν η IPC_0 είναι ίση με την IPC_2, και οι προηγούμενες αποφάσεις είναι $s_2 = 1$ και $s_1 = 0$. Συγκεκριμένα, ενώ η επίδοση ήταν σταθερή, η αφαίρεση χώρου τη διατάραξε, και ο μηχανισμός πρόσθεσε το χώρο που είχε προηγουμένως αφαιρέσει. Αυτή η ενέργεια επανέφερε την επίδοση στο προηγούμενο επίπεδο, άρα η επίδοση θεωρείται και πάλι σταθερή. Ταυτόχρονα έχει βρεθεί το σημείο ισορροπίας που αναζητείται, και ο μηχανισμός μεταβαίνει στην κατάσταση Balanced.

Ακολουθεί ο πίνακας αληθείας του προηγούμενου συλλογισμού. Όπου IPC_{01}, εννοείται η ισότητα μεταξύ IPC_0 και IPC_1 (σημειώνεται 0 σε περίπτωση ανισότητας και 1 σε περίπτωση ισότητας). Το αντίστοιχο ισχύει και για το IPC_{02}. Από τον προηγούμενο πίνακα αληθείας, μπορεί να σχεδιαστεί ο αντίστοιχος πίνακας Karnaugh, από όπου εξάγεται η λογική εξίσωση που διέπει το συλλογισμό που παρουσιάστηκε.

IPC_{01}	IPC_{02}	s_1	s_2	s_0
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Πίνακας 8: Πίνακας Αληθείας της Διαδικασίας Λήψης Απόφασης

$IPC_{01}IPC_{02}$ \ s_0s_1	00	01	11	10
00	0	0	0	0
01	0	1	0	0
11	1	1	1	1
10	1	0	1	1

Πίνακας 9: Πίνακας Karnaugh της Συνάρτησης Αποφάσεων

Από το παραπάνω, προκύπτει ότι η συνάρτηση της απόφασης, η οποία εξαρτάται από 4 λογικές μεταβλητές, τις $IPC_{01}, IPC_{02}, s_1, s_2$, είναι η ακόλουθη:

$$s_0(IPC_{01}, IPC_{02}, s_1, s_2) = IPC_{01}(s_1 + s_2' + IPC_{02}) + IPC_{02}s_1's_2$$

Κατά την παραμονή στην κατάσταση Balanced, ο μηχανισμός εξακολουθεί να λαμβάνει αποφάσεις σύμφωνα με τον προηγούμενο συλλογισμό. Αυτό που αλλάζει είναι η ερμηνεία των αποφάσεων, καθώς όσο η επίδοση παραμένει σταθερή, ο μηχανισμός δεν εκτελεί κάποια αλλαγή στον καταμερισμό της cache, ενώ σε περίπτωση αστάθειας, ο μηχανισμός εξέρχεται από την κατάσταση Balanced, και η ερμηνεία των αποφάσεων γίνεται και πάλι σύμφωνα με τον προηγούμενο συλλογισμό.

Ο μηχανισμός επιδιώκει να κρατήσει την επίδοση της εφαρμογής όσο το δυνατό σταθερότερη. Αυτό, όμως δεν είναι πάντοτε εφικτό, καθώς πολλές εφαρμογές έχουν περισσότερες από μία φάσεις εκτέλεσης, στις οποίες το IPC μεταβάλλεται. Η αύξηση, λοιπόν, του IPC μπορεί να ερμηνευτεί ως βελτίωση της επίδοσης της εφαρμογής αλλά και ως αλλαγή της φάσης εκτέλεσης. Εάν ο μηχανισμός αγνοούσε την αύξηση του IPC, ερμηνεύοντας την ως βελτίωση της επίδοσης, είναι πιθανό το IPC να μην έφτανε στη μέγιστη δυνατή τιμή του, λόγω του περιορισμένου χώρου που διατίθεται στην εφαρμογή. Για αυτό το λόγο, όταν το IPC αυξάνεται, ο μηχανισμός θα οδηγηθεί στην αρχική κατάσταση, για να επιδιώξει να βρει εκ νέου το σημείο ισορροπίας.

5.2 Πειραματική Αξιολόγηση του Μηχανισμού

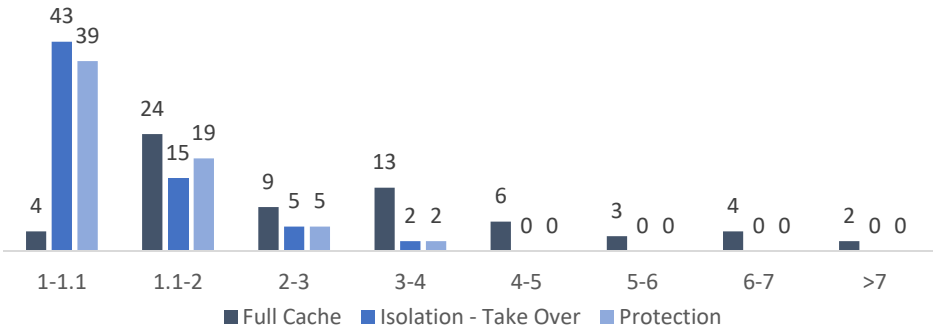
Σε αυτήν την ενότητα παρουσιάζονται τα αποτελέσματα της αξιολόγησης λειτουργίας του μηχανισμού προστασίας. Συγκεκριμένα, στον πίνακα που ακολουθεί παρουσιάζονται οι μετρήσεις που αφορούν τα 65 ζευγάρια του υποπροβλήματος που παρουσιάστηκε στο προηγούμενο κεφάλαιο, υπό το καθεστώς της λειτουργίας του μηχανισμού. Οι στήλες που προστέθηκαν στον πίνακα είναι οι εξής:

- *IPC Isolation Slowdown (3η στήλη)*: Ο λόγος του IPC του high priority benchmark κατά τη λειτουργία του μηχανισμού προς το αντίστοιχο IPC στην κατάσταση Isolation – Take Over.
- *Time Isolation Slowdown (5η στήλη)*: Ο λόγος του χρόνου εκτέλεσης του high priority benchmark κατά τη λειτουργία του μηχανισμού προς τον αντίστοιχο χρόνο στην κατάσταση Isolation – Take Over.
- *$Occupancy_{avg}$ (8η στήλη)*: Ο μέσος όρος του χώρου στην cache που παραχωρήθηκε στο high priority benchmark από το μηχανισμό. Η χρησιμοποίηση της cache (τελευταία στήλη) υπολογίζεται ως το πηλίκο του μέσου cache occupancy προς την $Occupancy_{avg}$.

High Priority	IPC Sl.	IPC Is.	T. Sl.	T. Is.	MPKI	MBL	Occ_{avg}	Util.
Omnetpp	89.0%	100.0%	1.13	1.00	1.3	3.85	53419	94.1%
Libquantum	72.2%	101.1%	1.39	0.99	2.0	0.68	53413	76.2%
Bzip2 3	98.0%	99.7%	1.03	1.00	2.9	9.51	35782	60.2%
Mcf	44.3%	88.4%	2.27	1.13	1.0	607.14	48667	97.8%
Soplex 1	93.6%	96.2%	1.07	1.04	2.8	21.29	46820	72.8%
Canneal	39.8%	95.8%	2.53	1.04	1.1	200.82	53408	99.9%
Xalancbmk	81.0%	92.3%	1.25	1.08	1.6	62.60	42713	93.9%
Bzip2 2	99.0%	99.5%	1.04	1.00	1.8	13.28	41164	85.5%
Streamcluster	30.4%	98.7%	3.30	1.01	1.0	277.80	53473	98.5%
Milc	27.6%	92.9%	3.63	1.08	1.0	908.94	53241	99.5%
Gcc 1	93.7%	99.3%	1.09	1.00	1.0	85.78	53291	99.5%
Lbm	35.7%	102.2%	2.81	0.98	1.0	1045.83	53497	100.0%
Gcc 7	82.9%	99.1%	1.21	1.01	1.1	155.79	53109	96.9%
Soplex 2	39.5%	97.0%	2.56	1.03	0.8	1238.54	53385	99.9%
Sphinx3	97.8%	97.6%	1.02	1.02	283.9	3.14	22440	64.3%
GemsFDTD	38.2%	90.9%	2.62	1.10	0.8	1815.36	53470	100.0%
Gcc 8	80.9%	91.1%	1.25	1.10	1.1	169.27	52489	97.0%
Leslie3d	62.6%	100.6%	1.60	0.99	1.0	161.17	53485	95.7%
CactusADM	70.6%	97.0%	1.42	1.03	0.9	405.02	53389	99.5%
Astar 1	94.4%	97.5%	1.06	1.03	2.3	18.26	39513	75.3%
Bzip2 5	96.4%	98.2%	1.04	1.02	1.3	34.94	25421	67.3%
Gcc 6	80.7%	96.1%	1.25	1.04	1.1	165.48	52600	97.2%
Gcc 2	86.1%	98.9%	1.17	1.01	1.1	95.71	53098	96.1%
Gcc 4	87.7%	97.6%	1.15	1.02	1.1	96.36	52888	97.3%
Gcc 5	83.3%	93.7%	1.20	1.07	1.2	125.79	52440	96.6%
Bzip2 6	98.6%	100.3%	1.02	1.00	1.0	40.19	51230	86.3%
Gcc 3	89.8%	97.0%	1.14	1.03	1.1	101.46	52536	94.8%
Bzip2 1	98.1%	99.9%	1.02	1.00	1.0	39.58	52635	94.5%
Zeusmp	51.4%	96.3%	1.95	1.04	0.9	525.07	53349	100.0%
Gcc 9	88.9%	94.0%	1.15	1.06	2.0	34.87	48964	79.9%
Bzip2 4	95.9%	97.4%	1.04	1.03	1.3	40.97	26919	64.4%
x264	82.1%	98.0%	1.27	1.06	1.1	81.02	52172	94.6%
Bwaves	85.4%	92.4%	1.17	1.08	0.8	3091.36	52987	99.5%
Rtview	94.9%	98.9%	1.06	1.01	1.1	149.14	21944	80.1%
Calculix	100.1%	100.4%	1.00	1.00	1.2	22.38	51923	92.9%
Astar 2	96.4%	98.0%	1.04	1.02	32.0	6.53	20079	30.1%
Perlbench 1	99.1%	99.8%	1.01	1.00	2.6	3.18	42749	69.6%
Wrf	94.7%	96.2%	1.06	1.04	1.0	485.13	53107	99.9%
Sjeng	86.7%	94.9%	1.16	1.05	1.3	49.00	44407	87.6%
H264ref 1	96.0%	96.4%	1.06	1.04	845.3	6.19	14515	68.3%
Ferret	94.0%	97.8%	1.06	1.02	1.6	7.43	51697	87.8%
Gobmk 1	99.8%	98.9%	1.01	1.01	39.0	0.70	48013	41.6%
Gobmk 4	99.8%	99.5%	1.00	1.00	27.5	0.19	48306	29.4%
Perlbench 3	96.2%	98.6%	1.05	1.01	1.2	46.13	45832	95.8%
Gobmk 3	97.8%	96.3%	1.04	1.04	149.6	13.28	45398	44.1%
Gobmk 2	99.5%	99.8%	1.01	1.00	18.2	0.16	46978	41.0%
Tonto	97.9%	98.6%	1.02	1.01	30.5	0.33	49355	54.8%
Fluidanimate	77.3%	92.8%	1.30	1.08	1.1	243.05	36951	99.5%

High Priority	IPC Sl.	IPC Is.	T. Sl.	T. Is.	MPKI	MBL	$Occ._{avg}$	Util.
H264ref 2	100.0%	99.0%	1.02	1.01	24.0	0.24	53125	26.9%
Dedup	90.4%	89.0%	1.15	1.05	1.0	96.86	48859	95.6%
H264ref 3	99.0%	99.6%	1.01	1.00	12.8	7.37	33198	44.4%
Gobmk 5	100.9%	99.7%	1.00	1.00	17.9	0.05	50007	33.6%
Freqmine	99.6%	100.0%	1.01	1.00	3.2	16.22	41293	61.7%
Gamess 3	99.3%	99.7%	1.01	1.00	48.6	2.83	9996	63.6%
Bodytrack	94.1%	94.5%	1.07	1.05	8.3	21.87	8813	87.1%
Gromacs	98.1%	98.3%	1.03	1.02	2344.7	26.94	10207	75.6%
Gamess 2	101.3%	100.8%	0.99	0.99	23.7	1.23	10279	39.9%
Povray	96.5%	96.8%	1.04	1.03	239.4	0.17	10130	72.1%
Gamess 1	99.4%	100.1%	1.01	1.00	23.4	0.11	48766	11.6%
Hmmer 1	95.8%	96.3%	1.05	1.04	564.6	75.12	9170	71.0%
Perlbench 2	97.5%	99.4%	1.03	1.01	1.4	32.34	44506	87.4%
Blackscholes	99.6%	100.2%	1.00	1.00	1.0	109.69	12111	85.3%
Swaptions	99.5%	100.0%	1.01	1.00	27.0	0.09	7516	47.6%
Namd	99.9%	99.9%	1.00	1.00	6.5	1.45	49306	59.5%
Hmmer 2	98.5%	98.8%	1.02	1.01	354.3	0.05	7308	35.5%

Πίνακας 10: Χαρακτηριστικά των High Priority Benchmarks (Protected)

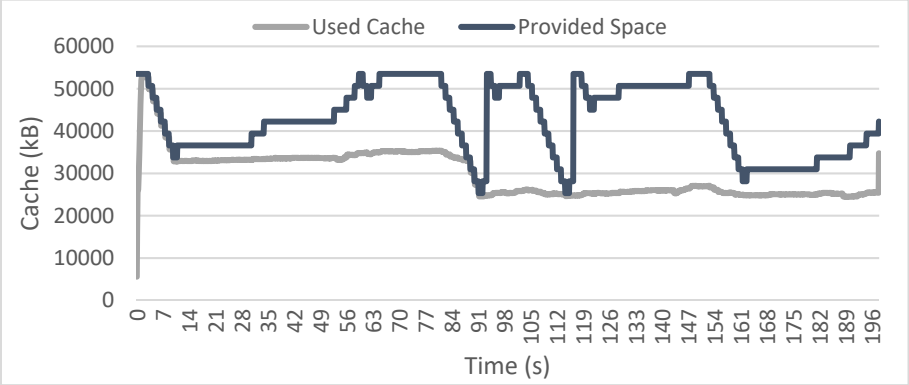


Γράφημα 18: Ταξινόμηση των Ζευγαριών με το Μηχανισμό Προστασίας

Το προηγούμενο γράφημα περιέχει το πλήθος των ζευγαριών benchmarks ανά κατηγορία Slowdown, όπου είναι εμφανές ότι η χρήση του μηχανισμού πλησιάζει σε μεγάλο βαθμό την κατάσταση Isolation – Take Over. Η ειδοποιός διαφορά μεταξύ των δύο είναι το μέγεθος της παρεχόμενης cache στο high priority benchmark. Ενώ στην κατάσταση Isolation – Take Over ο μέσος όρος της παρεχόμενης cache είναι 53.504kB, στην κατάσταση χρήσης του μηχανισμού μειώθηκε σε 41.126kB. Ένα μειονέκτημα του μηχανισμού παρατηρείται στο ποσοστό χρησιμοποίησης της παρεχόμενης cache από το high priority benchmark, το οποίο ανέρχεται σε 76.9%. Το ποσοστό χρησιμοποίησης όλης της cache είναι 83.13%.

Η επίδοση της high priority εφαρμογής είναι πιθανό να μην αντικατοπτρίζει πάντοτε την ανάγκη της για χώρο στην cache. Αυτό οδηγεί, σε κάποιες περιπτώσεις, σε άσκοπες μεταβάσεις στην αρχική κατάσταση, καθώς η εφαρμογή δε χρησιμοποιεί το χώρο που της παρέχεται, μειώνοντας με αυτόν τον τρόπο τη χρησιμοποίηση της cache. Στο γράφημα που ακολουθεί, παρουσιάζεται ο χώρος που παρέχεται στο Perlbench 1 από το μηχανισμό σε σύγκριση με το χώρο που τελικά χρησιμοποιεί το Perlbench 1 κατά τη διάρκεια της εκτέλεσης του με 21 instances του Lbm. Όπως φαίνεται, οι μεταβάσεις που

γίνονται στην αρχική κατάσταση είναι άσκοπες, καθώς το Perlbench 1 δε χρησιμοποιεί το χώρο που του παρέχεται. Αυτό οφείλεται σε διακυμάνσεις που παρουσιάζει η επίδοση του Perlbench 1, και λογίζονται από το μηχανισμό ως αστάθειες στην επίδοση.



Γράφημα 19: Υποχρησιμοποίηση της cache στο Perlbench 1 – Lbm

5.3 Ελεγκτής Χρησιμοποίησης Cache

Για την αντιμετώπιση των μειονεκτημάτων που παρατηρήθηκαν στη λειτουργία του Μηχανισμού Προστασίας, δηλαδή της υποχρησιμοποίησης της cache, αλλά και για τη μείωση του αριθμού των άσκοπων μεταβάσεων στην αρχική κατάσταση, αναπτύχθηκε ο Ελεγκτής Χρησιμοποίησης Cache. Ο ελεγκτής αυτός είναι μια επέκταση του μηχανισμού προστασίας. Ο ελεγκτής χρησιμοποιεί τις μετρήσεις της χρήσης της cache occupancy και του bandwidth, ώστε να επιβλέπει τη χρήση του παρεχόμενου χώρου από τη high priority εφαρμογή. Ο ελεγκτής λειτουργεί παράλληλα με το μηχανισμό προστασίας, επομένως η αρχική κατάσταση και ο αλγόριθμος αποφάσεων δεν μεταβάλλονται. Σε κάθε διάστημα Δt ο μηχανισμός αποθηκεύει τη μετρούμενη τιμή του IPC. Στο ίδιο διάστημα, ο ελεγκτής αποθηκεύει τις μετρήσεις του cache occupancy και του bandwidth. Μετά από n μετρήσεις, όταν ο μηχανισμός εισέρχεται στη διαδικασία λήψης απόφασης, ο ελεγκτής συγκρίνει τη μέγιστη τιμή των n μετρήσεων του bandwidth με ένα προκαθορισμένο όριο bandwidth. Εάν το bandwidth είναι μικρότερο από το όριο, τότε μπορεί να θεωρηθεί αμελητέο. Τότε ο ελεγκτής συγκρίνει τη μέγιστη τιμή των n μετρήσεων του cache occupancy με το χώρο που έχει παραχωρηθεί στη high priority εφαρμογή, και αν εντοπίσει ότι υπάρχει χώρος που δε χρησιμοποιείται, τον αφαιρεί από τη high priority εφαρμογή. Εάν η εφαρμογή χρησιμοποιεί όλο το χώρο που της δίνεται, τότε δεν εκτελείται κάποια ενέργεια.

Επίσης, η μετάβαση εκτός κατάστασης ισορροπίας πλέον, εκτός από την αρνητική απόφαση για τη σταθερότητα του IPC, απαιτεί για την τιμή του bandwidth να βρίσκεται πάνω από το καθορισμένο όριο, να μη θεωρείται δηλαδή αμελητέο. Αυτό σημαίνει ότι υπάρχει κίνηση μεταξύ της cache και της μνήμης, δηλαδή η εφαρμογή πιθανώς να έχει ανάγκη από επιπλέον χώρο στην cache. Με αυτόν τον τρόπο, αποφεύγονται άσκοπες μεταβάσεις στην κατάσταση Isolation Take-Over, κατά τις οποίες παρέχεται σχεδόν όλη η cache στη high priority εφαρμογή, αλλά δε χρησιμοποιείται.

5.4 Πειραματική Αξιολόγηση του Ελεγκτή

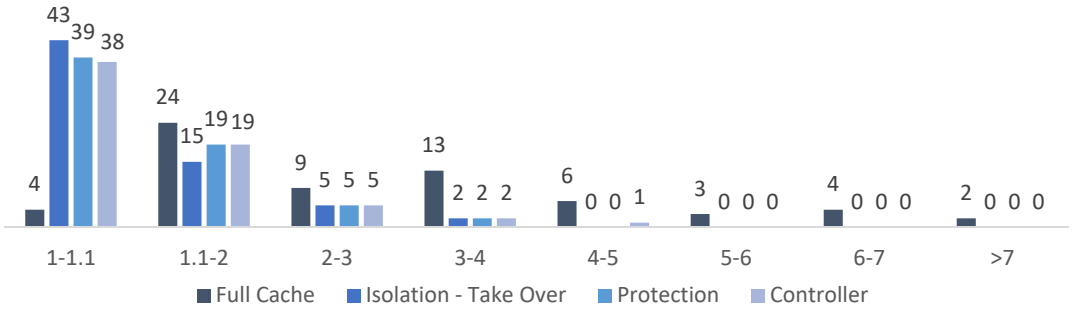
Οι μετρήσεις των 65 benchmarks του υποπροβλήματος με τη χρήση του Ελεγκτή Χρησιμοποίησης παρουσιάζονται στον επόμενο πίνακα.

High Priority	IPC Sl.	IPC Is.	T. Sl.	T. Is.	MPKI	MBL	$Occ._{avg}$	Util.
Omnetpp	86.5%	97.3%	1.16	1.03	1.71	12.12	52308	94.5%
Libquantum	22.9%	32.0%	4.39	3.12	36.44	56.51	41634	64.0%
Bzip2 3	96.2%	97.8%	1.05	1.02	5.68	20.69	29628	60.7%
Mcf	44.0%	87.9%	2.29	1.14	0.99	605.17	48479	99.1%
Soplex 1	93.3%	95.9%	1.07	1.04	3.02	17.37	46017	80.0%
Canneal	40.7%	97.7%	2.48	1.02	1.07	205.48	53416	99.9%
Xalancbmk	81.5%	92.8%	1.25	1.08	1.55	54.65	42457	94.5%
Bzip2 2	97.6%	98.1%	1.06	1.02	1.93	11.94	41446	85.5%
Streamcluster	29.5%	95.7%	3.40	1.04	1.04	272.89	53469	99.3%
Milc	27.7%	93.1%	3.63	1.07	0.98	890.12	53220	99.5%
Gcc 1	94.4%	100.1%	1.09	1.00	1.04	88.20	53290	99.5%
Lbm	33.7%	96.3%	2.98	1.04	0.96	961.45	23966	100.0%
Gcc 7	83.2%	99.4%	1.21	1.01	1.11	146.90	53313	98.3%
Soplex 2	39.2%	96.2%	2.57	1.04	0.85	1281.01	53375	99.8%
Sphinx3	98.9%	98.6%	1.01	1.01	111.46	0.33	22796	82.4%
GemsFDTD	38.9%	92.4%	2.58	1.08	0.84	1849.32	53431	100.0%
Gcc 8	84.8%	95.5%	1.20	1.05	1.06	170.93	52925	98.5%
Leslie3d	62.6%	100.6%	1.60	0.99	1.01	437.24	53485	95.4%
CactusADM	71.2%	97.9%	1.41	1.02	0.88	406.37	53364	99.3%
Astar 1	87.0%	89.9%	1.15	1.11	5.01	23.90	32988	78.4%
Bzip2 5	97.9%	99.7%	1.03	1.00	1.02	32.81	40065	95.1%
Gcc 6	80.6%	96.1%	1.25	1.04	1.08	146.08	52715	97.6%
Gcc 2	84.8%	97.3%	1.19	1.03	1.11	75.33	52155	93.5%
Gcc 4	88.4%	98.4%	1.14	1.02	1.13	89.80	53003	96.6%
Gcc 5	86.5%	97.3%	1.16	1.03	1.11	129.52	53096	98.4%
Bzip2 6	98.2%	99.9%	1.03	1.00	1.00	38.01	47426	92.9%
Gcc 3	88.6%	95.7%	1.15	1.04	1.14	92.91	52117	95.2%
Bzip2 1	98.2%	99.9%	1.02	1.00	1.02	39.86	53321	96.2%
Zeusmp	51.3%	96.1%	1.96	1.04	0.85	532.77	53330	99.6%
Gcc 9	91.1%	96.4%	1.13	1.04	1.94	31.46	48385	74.2%
Bzip2 4	95.4%	96.8%	1.05	1.03	1.45	44.23	27957	79.2%
x264	82.2%	98.2%	1.52	1.26	1.06	115.96	45592	87.7%
Bwaves	88.3%	95.6%	1.14	1.05	0.81	3082.51	53257	99.8%
Rtview	94.7%	98.8%	1.06	1.01	1.13	148.00	22058	80.3%
Calculix	100.0%	100.3%	1.00	1.00	1.16	20.51	52833	94.9%
Astar 2	96.5%	98.1%	1.04	1.02	20.09	2.15	15253	67.3%
Perlbench 1	99.3%	99.9%	1.01	1.00	2.36	2.88	31384	91.9%
Wrf	97.8%	99.4%	1.02	1.01	0.99	510.71	53489	100.0%
Sjeng	87.5%	95.7%	1.15	1.04	1.23	47.94	45721	88.5%
H264ref 1	96.0%	96.3%	1.06	1.04	937.94	12.47	10591	75.8%
Ferret	93.2%	97.0%	1.07	1.03	1.83	4.38	48790	95.6%
Gobmk 1	95.6%	94.8%	1.05	1.05	459.49	17.57	32467	51.6%
Gobmk 4	95.2%	94.8%	1.05	1.05	391.73	18.11	33566	54.5%
Perlbench 3	95.9%	98.3%	1.05	1.02	1.19	47.89	46727	94.3%

High Priority	IPC Sl.	IPC Is.	T. Sl.	T. Is.	MPKI	MBL	$Occ._{avg}$	Util.
Gobmk 3	88.2%	86.8%	1.15	1.15	660.75	12.20	27511	65.1%
Gobmk 2	96.1%	96.5%	1.04	1.04	126.46	4.45	26984	67.6%
Tonto	93.1%	93.8%	1.08	1.07	905.88	20.35	33138	32.7%
Fluidanimate	74.3%	89.1%	1.35	1.12	1.23	278.76	20148	94.8%
H264ref 2	100.0%	99.0%	1.02	1.01	116.60	2.25	13937	64.5%
Dedup	100.1%	98.4%	1.01	0.92	1.04	110.05	52895	96.7%
H264ref 3	97.4%	98.1%	1.03	1.02	42.91	10.89	12751	62.5%
Gobmk 5	98.6%	97.5%	1.03	1.03	216.42	12.90	41877	41.3%
Freqmine	99.4%	99.8%	1.01	1.00	2.20	7.92	42730	93.2%
Gamess 3	98.4%	98.7%	1.02	1.01	78.11	2.22	6023	89.9%
Bodytrack	91.0%	91.3%	1.09	1.07	11.08	27.50	6526	88.6%
Gromacs	98.1%	98.2%	1.03	1.02	1098.18	1.56	11554	63.5%
Gamess 2	101.5%	101.0%	0.99	0.99	51.95	0.57	6583	59.4%
Povray	94.6%	94.8%	1.06	1.05	380.44	0.07	6656	65.2%
Gamess 1	98.8%	99.6%	1.02	1.00	77.18	0.19	6157	59.4%
Hmmer 1	94.9%	95.4%	1.06	1.05	656.38	126.26	6578	86.6%
Perlbench 2	97.4%	99.3%	1.03	1.01	1.15	25.01	52017	91.2%
Blackscholes	99.7%	100.3%	1.00	1.00	1.01	109.49	11923	86.2%
Swaptions	99.1%	99.7%	1.01	1.00	34.06	0.10	6108	44.3%
Namd	99.5%	99.6%	1.01	1.00	17.28	4.34	19056	81.7%
Hmmer 2	98.1%	98.3%	1.02	1.02	502.30	559.70	6201	68.3%

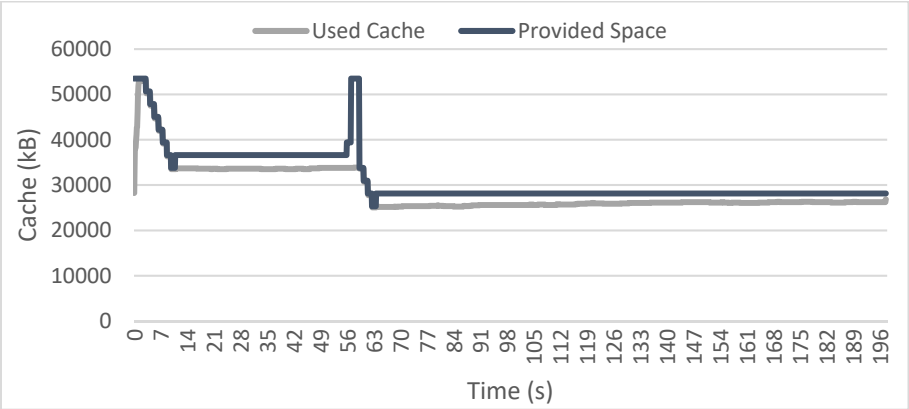
Πίνακας 11: Χαρακτηριστικά των High Priority Benchmarks (Ελεγκτής)

Η παρατήρηση της μειωμένης χρησιμοποίησης της cache από τα high priority benchmarks κατά τη λειτουργία του Μηχανισμού Προστασίας οδήγησε στη δημιουργία του Ελεγκτή Χρησιμοποίησης Cache, σκοπός του οποίου είναι να οδηγήσει την επίδοση σε αντίστοιχα επίπεδα με αυτά του μηχανισμού, αφαιρώντας παράλληλα από το high priority benchmark το χώρο στην cache που δε χρησιμοποιεί. Στο γράφημα που ακολουθεί, φαίνεται η ταξινόμηση των ζευγαριών στις κατηγορίες Slowdown για κάθε μία από τις 4 περιπτώσεις λειτουργίας. Τα αξιοπρόσεκτα σημεία της λειτουργίας του ελεγκτή είναι, πρώτον, το επίπεδο επίδοσης, το οποίο στην πλειοψηφία των περιπτώσεων είναι ίδιο με την περίπτωση του Μηχανισμού Προστασίας, δεύτερον, ο μέσος όρος της παρεχόμενης cache, ο οποίος από 41.126kB μειώθηκε σε 36.204kB, και τρίτον, το ποσοστό χρησιμοποίησης της παρεχόμενης cache από το high priority benchmark, το οποίο αυξήθηκε σε 83.5%. Η αναγωγή σε όλο το χώρο της cache οδηγεί σε ποσοστό χρησιμοποίησης της cache της τάξης του 89.4%.



Γράφημα 20: Ταξινόμηση των Ζευγαριών με τον Ελεγκτή

Το δεύτερο μειονέκτημα του μηχανισμού προστασίας που καλείται να αντιμετωπίσει ο ελεγκτής χρησιμοποίησης είναι οι άσκοπες μεταβάσεις στην αρχική κατάσταση. Στο επόμενο γράφημα παρουσιάζεται ο παρεχόμενος χώρος στην cache και η χρήση του από το Perlbench 1 κατά τη διάρκεια εκτέλεσης του ζευγαριού Perlbench 1 – Lbm. Σε αντίθεση με την εικόνα που παρουσιάστηκε στην προηγούμενη ενότητα, φαίνεται πως η χρήση του ελεγκτή χρησιμοποίησης έχει μειώσει στο ελάχιστο τις άσκοπες μεταβάσεις στην αρχική κατάσταση, ενώ παράλληλα ελέγχει το χώρο που χρησιμοποιείται από το Perlbench 1, αφαιρώντας τον περιττό χώρο όταν τον εντοπίσει.



Γράφημα 21: Μείωση των Άσκοπων Μεταβάσεων στο Perlbench 1 - Lbm

5.5 Περιπτώσεις Χρήσης

Στην παρούσα ενότητα παρουσιάζεται η λειτουργία του μηχανισμού σε 7 περιπτώσεις χρήσης. Συγκεκριμένα παρουσιάζεται μία περίπτωση από κάθε κατηγορία (class A, B, C, D, E, ενότητα 4.2) και άλλες 2 ειδικές περιπτώσεις. Για κάθε μία από τις 7 περιπτώσεις παρουσιάζονται τα αποτελέσματα των 4 ακόλουθων μετρήσεων.

Αρχικά, συγκρίνονται οι επιδόσεις της κατάσταση Isolation – Take Over, του Μηχανισμού Προστασίας και του Ελεγκτή Χρησιμοποίησης με την επίδοση στην κατάσταση «Alone». Στις δύο τελευταίες περιπτώσεις παρατίθενται τα γραφήματα του παρεχόμενου χώρου στην cache και του χώρου που χρησιμοποιείται.

Οι δύο επόμενες μετρήσεις που παρουσιάζονται αφορούν την επίδοση των 21 low priority benchmarks στις 4 καταστάσεις λειτουργίας. Στην πρώτη, παρουσιάζεται ο λόγος επιβράδυνσης των low priority benchmarks ανά κατάσταση λειτουργίας που προκύπτει ως το πηλίκο του μέσου χρόνου εκτέλεσης προς το χρόνο εκτέλεσης των benchmarks στην κατάσταση «Alone», μαζί με το λόγο επιβράδυνσης του high priority benchmark σε κάθε κατάσταση.

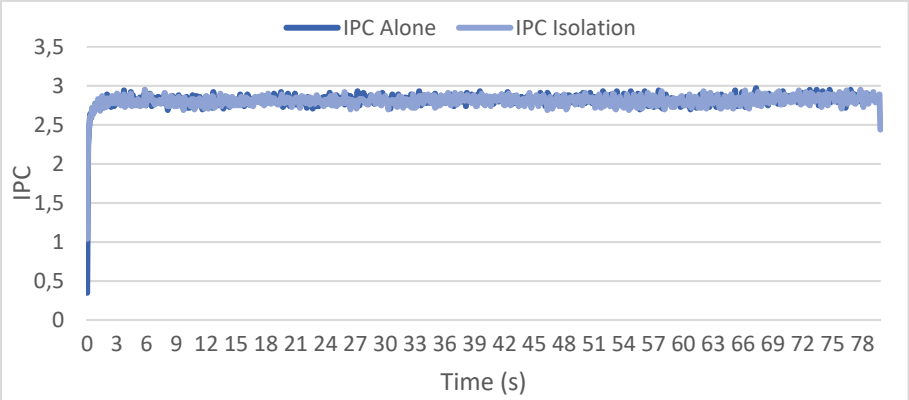
Στη δεύτερη, ορίζεται ένα χρονικό παράθυρο λειτουργίας, διαφορετικό σε κάθε περίπτωση, ενώ αλλάζει το ζευγάρι συνεκτέλεσης. Συγκεκριμένα, το high priority benchmark παραμένει ίδιο, ενώ επιλέγονται 21 τυχαία και διαφορετικά low priority benchmarks. Τα 22 benchmarks εκτελούνται μέσα στο παράθυρο λειτουργίας, και, εάν ένα benchmark ολοκληρωθεί, επανεκκινείται. Η μέτρηση που λαμβάνεται είναι το πλήθος των benchmarks που ολοκληρώθηκαν εντός του παραθύρου λειτουργίας.

5.5.1 Περίπτωση 1η: H264ref 1 – Leslie3d

Στην πρώτη περίπτωση χρήσης αναλύεται η συμπεριφορά του ζευγαριού H264ref 1 – Leslie3d. Το h264ref 1 ανήκει στην class A. Το Leslie3d είναι το benchmark που προκαλεί τη μεγαλύτερη χρονική καθυστέρηση στο h264ref 1, με το χρόνο εκτέλεσης του να είναι, κατά την κατάσταση Full Cache, 132.4 sec., από 78 sec., στην κατάσταση «Alone», δηλαδή ο λόγος επιβράδυνσης είναι 1.7.

- **Κατάσταση Isolation – Take Over**

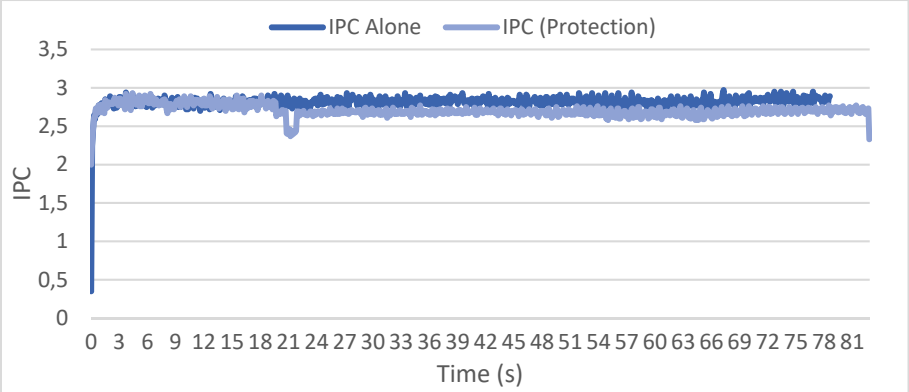
Στο επόμενο γράφημα συγκρίνονται το IPC στην εκτέλεση του H264ref 1 στην κατάσταση Alone με το IPC στην κατάσταση Isolation – Take Over. Όπως φαίνεται, τα δύο IPC συμπίπτουν. Συγκεκριμένα, ο λόγος των δύο IPC είναι 99.7%, ενώ ο λόγος των χρόνων εκτέλεσης είναι 1.02, άρα το h264ref ολοκληρώνεται σχεδόν στον ίδιο χρόνο στις δύο καταστάσεις.



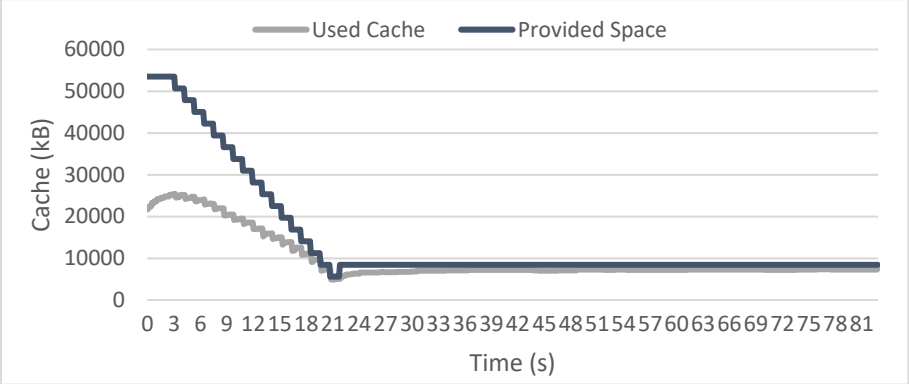
Γράφημα 22: H264ref 1 - Leslie3d (Isolation - Take Over)

- **Λειτουργία του Μηχανισμού Προστασίας**

Στα επόμενα γραφήματα συγκρίνεται το IPC της κατάστασης «Alone» με το IPC της λειτουργίας του μηχανισμού προστασίας και παρουσιάζεται ο παρεχόμενος χώρος και ο χώρος που χρησιμοποιήθηκε από το h264ref 1. Ο λόγος των δύο IPC είναι 96.1%, ενώ ο λόγος των χρόνων εκτέλεσης είναι 1,05. Επομένως, ο μηχανισμός οδηγεί το H264ref 1 στο βέλτιστο επίπεδο επίδοσης, ενώ ταυτόχρονα του παραχωρεί το χώρο που κρίθηκε απαραίτητος, αφήνοντας στα low priority benchmarks σημαντικό χώρο στην cache. Το ποσοστό χρήσης της παρεχόμενης cache είναι 68.3%, ενώ το ποσοστό χρήσης του συνόλου της cache είναι 91.8%.



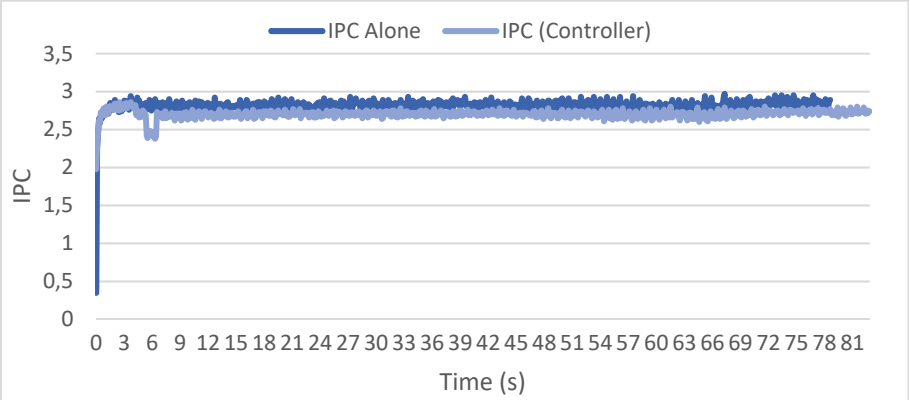
Γράφημα 23: Επίδοση του H264ref 1 (Μηχανισμός Προστασίας)



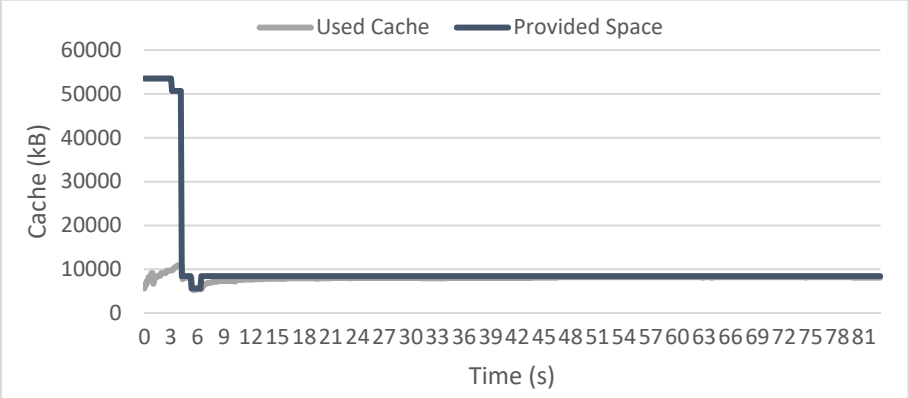
Γράφημα 24: Χρήση της Cache (H264ref 1, Μηχανισμός Προστασίας)

- **Λειτουργία του Ελεγκτή Χρησιμοποίησης**

Η ενεργοποίηση του ελεγκτή χρησιμοποίησης θα βοηθήσει στη μείωση του ποσοστού της αχρησιμοποίητης cache. Τα δύο επόμενα γραφήματα περιέχουν τη σύγκριση του IPC στην κατάσταση «Alone» με το IPC υπό τη λειτουργία του ελεγκτή, και τον παρεχόμενο χώρο με το χώρο που τελικά χρησιμοποιήθηκε από το h264ref 1. Στην περίπτωση αυτή ο λόγος των IPC είναι 96%, και ο λόγος των χρόνων εκτέλεσης είναι 1,05. Επομένως το benchmark είχε το απαιτούμενο επίπεδο επίδοσης, ενώ η εύρεση του απαραίτητου χώρου στην cache ολοκληρώθηκε πιο γρήγορα, με αποτέλεσμα το ποσοστό χρήσης της παρεχόμενης cache να είναι 75.8%, και το ποσοστό χρήσης όλης της cache να είναι 95.4%.



Γράφημα 25: Επίδοση του H264ref 1 (Ελεγκτής Χρησιμοποίησης)



Γράφημα 26: Χρήση της Cache (H264ref 1, Ελεγκτής Χρησιμοποίησης)

Στον πίνακα που ακολουθεί παρουσιάζονται αναλυτικά οι μετρήσεις που αφορούν το h264ref σε κάθε μία από τις 5 καταστάσεις.

High Priority	IPC	IPC Sl.	Time	T. Sl.	MPKI	MBL	$Occ._{avg}$	Util.
Alone	2.81	-	78	-	0.00	0.00	56320	41.7%
Full Cache	1.69	60.1%	132.4	1.70	1.97	1236.47	56320	3.1%
Isolation	2.80	99.7%	80	1.02	0.00	0.37	53504	79.7%
Protection	2.70	96.0%	82.9	1.06	0.13	6.19	14515	68.3%
Controller	2.70	96.0%	82.9	1.06	0.14	12.47	10591	75.8%

Πίνακας 12: Χαρακτηριστικά του H264ref 1 ανά Κατάσταση Εκτέλεσης

- **Επίδοση των Low Priority Benchmarks**

Το κέρδος της ενεργοποίησης του μηχανισμού σε σχέση με την κατάσταση Isolation είναι ότι ο χώρος που δε χρησιμοποιείται, παραχωρείται στα low priority benchmarks, γεγονός που είναι ευεργετικό για την επίδοση τους. Η επίδοση των low priority benchmarks μειώνεται αισθητά στην κατάσταση Isolation – Take Over, αφού και οι 21 εφαρμογές μοιράζονται μόλις 2.816kB. Η λειτουργία του μηχανισμού (είτε ο ελεγκτής είναι ενεργός είτε ανενεργός) επαναφέρει την επίδοση στο επίπεδο της κατάστασης Full Cache. Προφανώς, λόγω του ανταγωνισμού μεταξύ των 21 συνεκτελούμενων low priority benchmarks δεν αναμένεται να επιτευχθεί το βέλτιστο επίπεδο επίδοσης για αυτές τις εφαρμογές. Στον πίνακα που ακολουθεί, συγκρίνονται οι επιβραδύνσεις για τα 21 leslie3d (low priority benchmarks) και το h264ref (high priority benchmark).

	Alone	Full Cache	Isolation	Protection	Controller
Leslie3d (L.P.)	283 sec.	3.68	7.28	3.81	3.75
H264ref 1 (H.P.)	78 sec.	1.70	1.02	1.06	1.06

Πίνακας 13: Επιβράδυνση των L.P. Leslie3d ανά Κατάσταση (H264ref 1)

Στο δεύτερο μέρος της μέτρησης της επίδοσης των low priority benchmarks, το παράθυρο λειτουργίας τέθηκε ίσο με 250 sec. και, μαζί με το H264ref 1 εκτελέστηκαν 21 διαφορετικά, τυχαία low priority benchmarks. Σε αυτόν το χρόνο, το h264ref 1 αναμένεται να ολοκληρωθεί 3 φορές, αν η επίδοση του πλησιάσει τη βέλτιστη. Όπως φαίνεται στον πίνακα που ακολουθεί, αυτό συνέβη και στις 4 καταστάσεις εκτέλεσης (ο μέσος χρόνος ολοκλήρωσης ήταν 80 sec.), το οποίο σημαίνει ότι τα 21 τυχαία benchmarks δεν επηρέασαν σε μεγάλο βαθμό την εκτέλεση του H264ref 1. Το αξιοπρόσεκτο γεγονός είναι ότι κατά τη λειτουργία του μηχανισμού προστασίας αλλά και του ελεγκτή χρησιμοποίησης, τα 21 τυχαία low priority benchmarks ολοκληρώθηκαν ισάριθμες φορές με την κατάσταση «Full Cache», κάτι που δε συνέβη στην κατάσταση Isolation – Take Over, στην οποία το throughput του συστήματος μειώθηκε αισθητά.

	Full Cache	Isolation	Protection	Controller
Completed (L.P.)	23	2	23	22
Completed (H.P.)	3	3	3	3
Provided Space	-	53.504kB	11.644kB	13.671kB

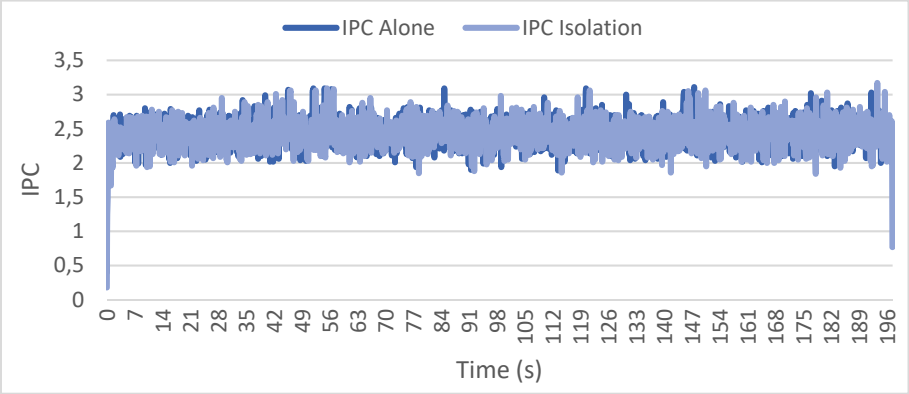
Πίνακας 14: Ολοκληρωμένα Benchmarks ανά Κατάσταση (H264ref 1)

5.5.2 Περίπτωση 2η: Perlbench 1 – Lbm

Στη δεύτερη περίπτωση χρήσης εξετάζεται το ζευγάρι Perlbench 1 – Lbm. Το Perlbench 1 ανήκει στην Class B. Κατά τη συνεκτέλεση των δύο benchmarks με ελεύθερη πρόσβαση στην cache, το Perlbench 1 εκτελείται σε διπλάσιο χρόνο σε σχέση με την κατάσταση «Alone» (συγκεκριμένα, 390,7 sec. από 195 sec.), ενώ το IPC μειώνεται στο 50%.

- **Κατάσταση Isolation – Take Over**

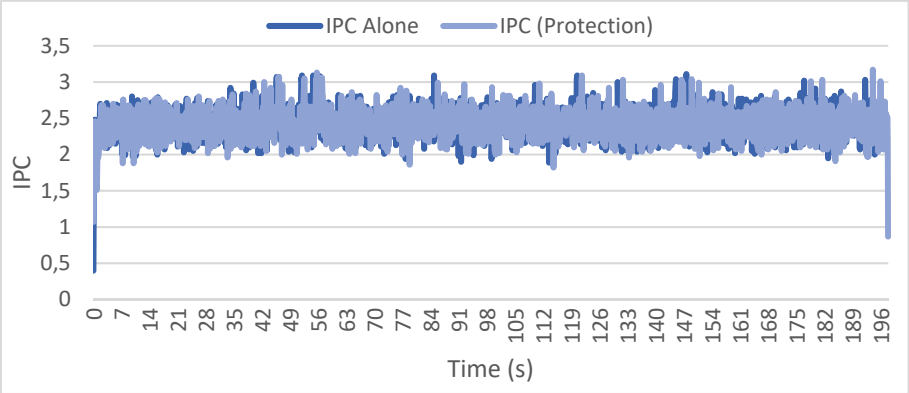
Στην κατάσταση Isolation – Take Over η επιβράδυνση αντιμετωπίζεται επιτυχώς, και η επίδοση είναι πολύ κοντά στην ιδανική, γεγονός το οποίο είναι εμφανές και στο επόμενο γράφημα. Συγκεκριμένα, ο λόγος των IPC είναι 99,4%, ενώ ο λόγος των χρόνων εκτέλεσης είναι 1,01.



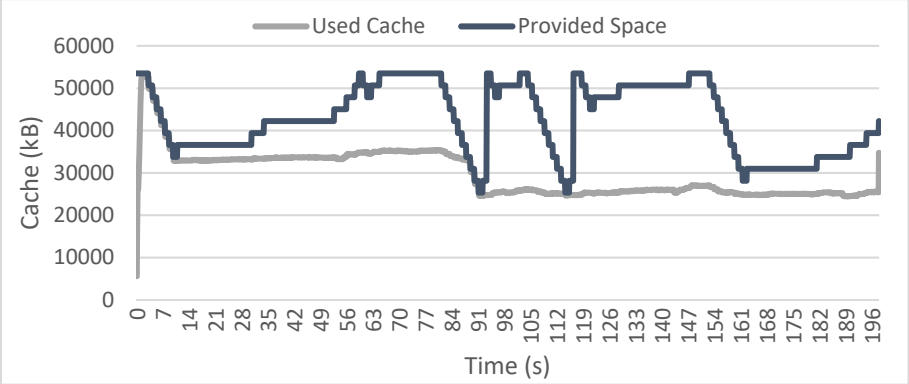
Γράφημα 27: Perlbench 1 - Lbm (Isolation - Take Over)

- **Λειτουργία του Μηχανισμού Προστασίας**

Στα επόμενα γραφήματα συγκρίνεται η επίδοση στην περίπτωση χρήσης του μηχανισμού προστασίας με τη βέλτιστη επίδοση, και παρουσιάζεται ο χώρος στην cache που παραχωρήθηκε στο Perlbench 1 μαζί με το χώρο που τελικά χρησιμοποιήθηκε. Ενώ το επίπεδο επίδοσης επιτυγχάνεται με το λόγο των IPC να είναι 99,1%, και τη χρονική επιβράδυνση να εξαλείφεται καθώς ο λόγος των χρόνων εκτέλεσης είναι και πάλι 1,01, παρατηρείται ένα σημαντικό ποσοστό αχρησιμοποίητης cache. Συγκεκριμένα, από την παρεχόμενη cache που είναι κατά μέσο όρο 42.749kB, χρησιμοποιείται το 69,6% αυτής, ενώ το συνολικό ποσοστό χρησιμοποίησης της cache είναι 76.9%.



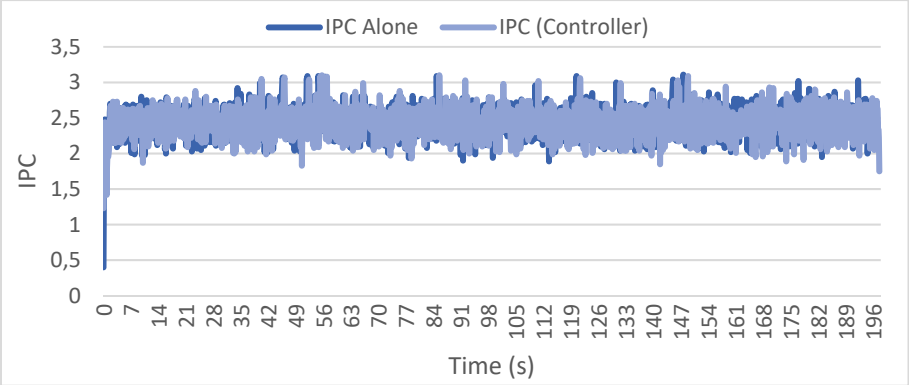
Γράφημα 28: Επίδοση του Perlbench 1 (Μηχανισμός Προστασίας)



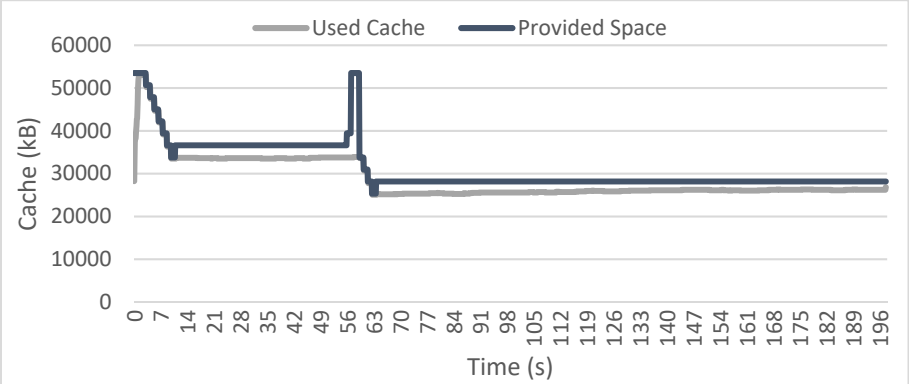
Γράφημα 29: Χρήση της Cache (Perlbench 1, Μηχανισμός Προστασίας)

- **Λειτουργία του Ελεγκτή Χρησιμοποίησης**

Για την αντιμετώπιση του μεγάλου ποσοστού αχρησιμοποίητης cache, ενεργοποιείται ο ελεγκτής χρησιμοποίησης. Ο ελεγκτής καταφέρνει να οδηγήσει την επίδοση πολύ κοντά στη βέλτιστη. Παράλληλα η cache που παρέχεται στο Perlbench 1 είναι λιγότερη σε σχέση με την προηγούμενη περίπτωση (31.384kB), και χρησιμοποιείται σε πολύ μεγαλύτερο βαθμό. Συγκεκριμένα, το IPC φτάνει στο 99.3% του μέγιστου, με τη χρονική επιβράδυνση να έχει αντιμετωπιστεί και σε αυτήν την περίπτωση, ενώ το ποσοστό χρήσης της cache από το Perlbench 1 είναι 91.9%, οπότε συνολικά το ποσοστό χρήσης ολόκληρης της cache είναι 95,5%. Επίσης, οι αποφάσεις που λαμβάνονται και οι μεταβολές που πραγματοποιούνται στην κατανομή της cache είναι λιγότερες.



Γράφημα 30: Επίδοση του Perlbench 1 (Ελεγκτής Χρησιμοποίησης)



Γράφημα 31: Χρήση της Cache (Perlbench 1, Ελεγκτής Χρησιμοποίησης)

Οι αναλυτικές μετρήσεις που αφορούν το perlbench 1 για κάθε μία από τις 5 καταστάσεις εκτέλεσης παρουσιάζονται στον πίνακα που ακολουθεί.

High Priority	IPC	IPC Sl.	Time	T. Sl.	MPKI	MBL	$Occ._{avg}$	Util.
Alone	2.44	-	195	-	0.00	2.39	56320	95.1%
Full Cache	1.23	50.2%	390.7	2.00	1.37	139.55	56320	0.9%
Isolation	2.43	99.4%	198	1.01	0.01	2.70	53504	98.6%
Protection	2.42	99.1%	197.9	1.01	0.01	3.18	42749	69.6%
Controller	2.42	99.3%	197.6	1.01	0.01	2.88	31384	91.9%

Πίνακας 15: Χαρακτηριστικά του Perlbench 1 ανά Κατάσταση Εκτέλεσης

- **Επίδοση των Low Priority Benchmarks**

Το όφελος της χρήσης του μηχανισμού σε σχέση με την κατάσταση Isolation – Take Over εντοπίζεται στη βελτίωση της επίδοσης των low priority benchmarks. Στον επόμενο πίνακα παρουσιάζεται ο λόγος του μέσου χρόνου ολοκλήρωσης προς το βέλτιστο χρόνο (κατάσταση «Alone») των 21 Lbm (low priority benchmarks) σε κάθε κατάσταση εκτέλεσης σε σύγκριση με το λόγο επιβράδυνσης του Perlbench 1. Όπως φαίνεται, η επιβράδυνση στην περίπτωση που ο ελεγκτής είναι ανενεργός είναι μεγαλύτερη, καθώς η cache που παρέχεται στα low priority benchmarks είναι μικρότερη κατά 11.365kB σε σχέση με την εκτέλεση με ενεργό ελεγκτή. Η βελτίωση, όμως, της επίδοσης σε σχέση με την κατάσταση Isolation, είναι εμφανής.

	Alone	Full Cache	Isolation	Protection	Controller
Lbm (L.P.)	428 sec.	4.22	8.60	5.46	4.74
Perlbench 1 (H.P.)	195 sec.	2.00	1.01	1.01	1.01

Πίνακας 16: Επιβράδυνση των L.P. Lbm ανά Κατάσταση (Perlbench 1)

Το παράθυρο εκτέλεσης για τη δεύτερη φάση της μέτρησης επιλέχθηκε ίσο με 400 sec. Σε αυτό το διάστημα και με τη μέγιστη επίδοση, το perlbench 1 αναμένεται να ολοκληρωθεί 2 φορές. Όπως φαίνεται στον επόμενο πίνακα, αυτό συμβαίνει στην κατάσταση Isolation – Take Over, καθώς και κατά τη λειτουργία του μηχανισμού. Συγκεκριμένα, σε αυτές τις καταστάσεις το Perlbench 1 ολοκληρώθηκε σε 197 sec. κατά μέσο όρο, ενώ στην κατάσταση Full Cache ολοκληρώθηκε σε 233 sec. Στην πρώτη περίπτωση, όμως, τα low priority benchmarks ολοκληρώνονται μόλις 12 φορές, σε αντίθεση με το καθεστώς λειτουργίας του μηχανισμού, στο οποίο το throughput του συστήματος αυξάνεται αισθητά. Παρόλο που είναι μικρότερο από το throughput της κατάστασης Full Cache, ο μηχανισμός έχει πετύχει τον κύριο στόχο του, που είναι η βελτιστοποίηση της επίδοσης του Perlbench 1.

	Full Cache	Isolation	Protection	Controller
Completed (L.P.)	65	12	48	52
Completed (H.P.)	1	2	2	2
Provided Space	-	53.504kB	44.469kB	38.100kB

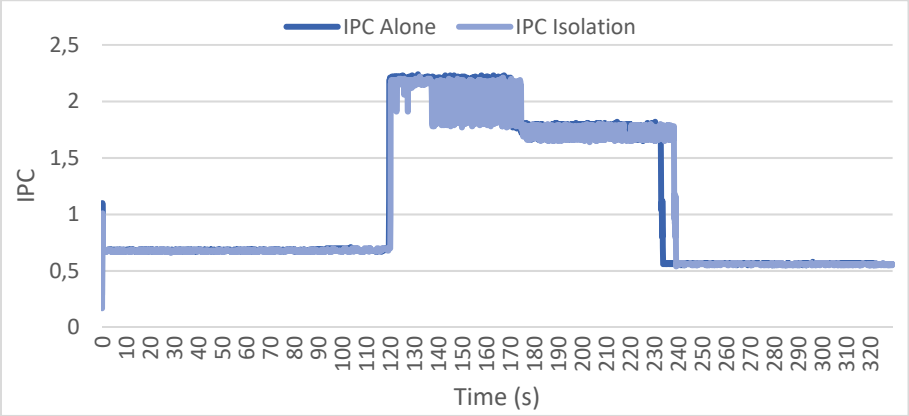
Πίνακας 17: Ολοκληρωμένα Benchmarks ανά Κατάσταση (Perlbench 1)

5.5.3 Περίπτωση 3η: Astar 2 - Bwaves

Στην τρίτη περίπτωση χρήσης παρουσιάζεται το ζευγάρι Astar 2-Bwaves. Το Astar 2 ανήκει στην Class C. Κατά τη συνεκτέλεση τους, το Astar 2 επιβραδύνεται με λόγο 2,04, ενώ το IPC είναι στο 49% του ιδανικού.

- **Κατάσταση Isolation – Take Over**

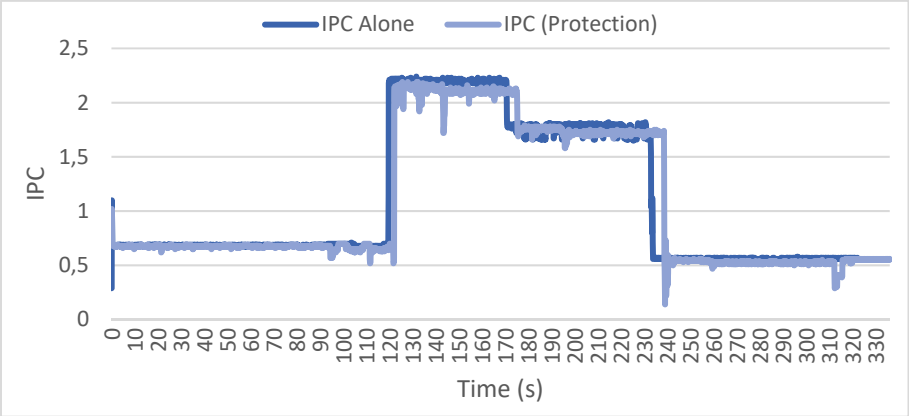
Στην κατάσταση Isolation – Take Over η επίδοση βελτιώνεται αισθητά, καθώς φτάνει το 98.3% της επίδοσης στην κατάσταση «Alone»), και αντιμετωπίζεται η χρονική επιβράδυνση, καθώς ο χρόνος εκτέλεσης του Astar 2 είναι κατά 2% μεγαλύτερος από το βέλτιστο. Το επίπεδο επίδοσης που επιτυγχάνεται φαίνεται στο επόμενο γράφημα.



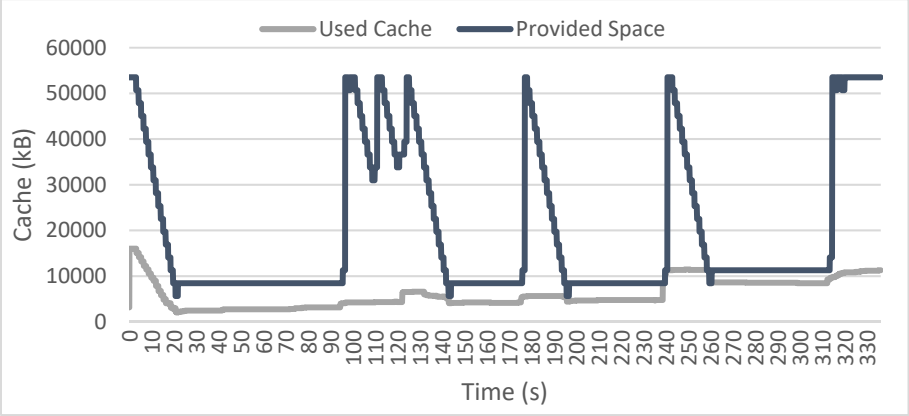
Γράφημα 32: Astar 2 - Bwaves (Isolation - Take Over)

- **Λειτουργία Μηχανισμού Προστασίας**

Ο μηχανισμός προστασίας επιτρέπει στο Astar 2 να φτάσει στο 98% της μέγιστης επίδοσης και να ολοκληρωθεί σε χρόνο κατά 4% μεγαλύτερο από το βέλτιστο. Το μειονέκτημα της λειτουργίας είναι το σημαντικό ποσοστό αχρησιμοποίητης cache, καθώς το Astar 2 χρησιμοποιεί μόλις το 30,1% της cache που του παρέχεται. Το ποσοστό χρησιμοποίησης ολόκληρης της cache είναι 75%. Στα επόμενα γραφήματα φαίνεται η επίδοση του Astar 2 κατά τη λειτουργία του μηχανισμού, και η μεγάλη διαφορά μεταξύ παρεχόμενου και χρησιμοποιούμενου χώρου στην cache. Επίσης, παρατηρούνται μεταβολές στην κατάσταση Isolation – Take Over, οι οποίες είναι άσκοπες, καθώς το Astar 2 δε χρησιμοποιεί το χώρο που του παρέχεται.



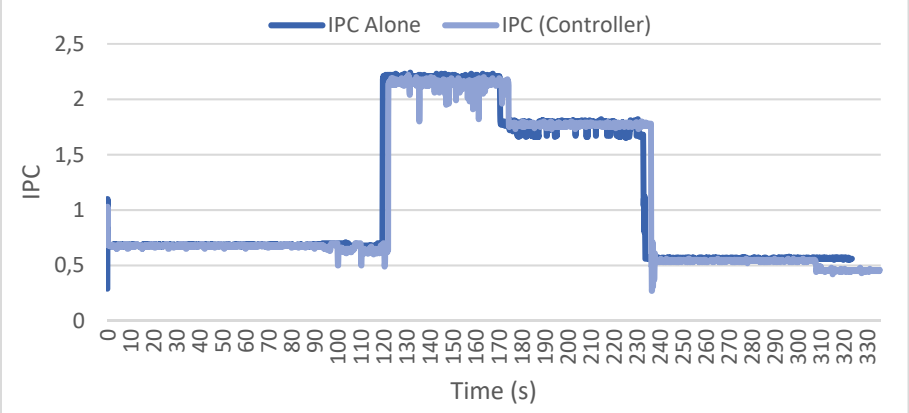
Γράφημα 33: Επίδοση του Astar 2 (Μηχανισμός Προστασίας)



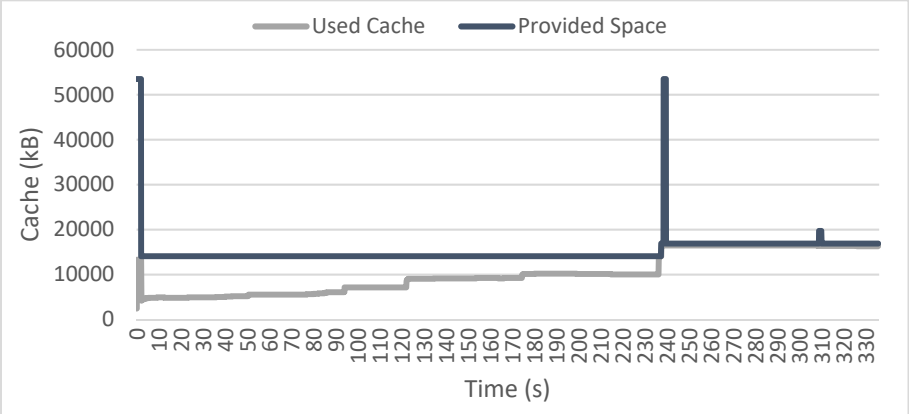
Γράφημα 34: Χρήση της Cache (Astar 2, Μηχανισμός Προστασίας)

- **Λειτουργία Ελεγκτή Χρησιμοποίησης**

Το πρόβλημα της υποχρησιμοποίησης της cache ελαττώνεται με τη χρήση του ελεγκτή χρησιμοποίησης. Συγκεκριμένα, το Astar 2 χρησιμοποιεί, σε αυτήν την περίπτωση, το 67,3% της cache που του παραχωρείται, επομένως η συνολική χρησιμοποίηση της cache είναι πλέον 91,1%. Επίσης ελαττώνεται και ο μέσος παρεχόμενος χώρος στην cache από 20.079kB σε 15.253kB. Παράλληλα, το μέσο IPC του Astar 2 δεν επηρεάζεται, καθώς παραμένει στο 98% της μέγιστης δυνατής τιμής, ενώ αντίστοιχα, μένει αναλλοίωτος ο χρόνος εκτέλεσης, που είναι 2% μεγαλύτερος του βέλτιστου.



Γράφημα 35: Επίδοση του Astar 2 (Ελεγκτής Χρησιμοποίησης)



Γράφημα 36: Χρήση της Cache (Astar 2, Ελεγκτής Χρησιμοποίησης)

High Priority	IPC	IPC Sl.	Time	T. Sl.	MPKI	MBL	$Occ._{avg}$	Util.
Alone	1.10	-	323	-	0.00	0.68	56320	89.3%
Full Cache	0.54	49.1%	658.8	2.04	5.58	281.59	56320	1.0%
Isolation	1.08	98.3%	330	1.02	0.01	0.93	53504	96.6%
Protection	1.06	96.4%	336.1	1.04	0.09	6.53	20079	30.1%
Controller	1.06	96.5%	335.7	1.04	0.06	2.15	15253	67.3%

Πίνακας 18: Χαρακτηριστικά του Astar 2 ανά Κατάσταση Εκτέλεσης

- **Επίδοση των Low Priority Benchmarks**

Το πλεονέκτημα της χρήσης του μηχανισμού προστασίας αντί της κατάστασης Isolation – Take Over είναι η βελτίωση της επίδοσης των low priority benchmarks. Στον επόμενο πίνακα παρουσιάζεται ο λόγος επιβράδυνσης των 21 Bwaves μαζί με τον αντίστοιχο λόγο επιβράδυνσης για το Astar 2 ανά κατάσταση εκτέλεσης. Όπως φαίνεται στον παρακάτω πίνακα, η λειτουργία του μηχανισμού οδηγεί την επίδοση των low priority benchmarks κοντά στην επίδοση που έχουν στην κατάσταση Full Cache, ενώ παράλληλα επιτυγχάνεται η επιθυμητή επίδοση για το Astar 2.

	Alone	Full Cache	Isolation	Protection	Controller
Bwaves (L.P.)	425 sec.	2.34	5.61	2.39	2.31
Astar 2 (H.P.)	323 sec.	2.04	1.02	1.04	1.04

Πίνακας 19: Επιβράδυνση των L.P. Bwaves ανά Κατάσταση (Astar 2)

Για τη δεύτερη φάση, το χρονικό παράθυρο εκτέλεσης είναι 660 sec., διάστημα στο οποίο το Astar 2 μπορεί να ολοκληρωθεί 2 φορές. Αυτό επιτυγχάνεται στην κατάσταση Isolation – Take Over (χρόνος ολοκλήρωσης: 326 sec.), με κόστος όμως τη ραγδαία μείωση του πλήθους των ολοκληρωμένων low priority benchmarks. Ο μηχανισμός προστασίας και ο ελεγκτής έχει το ίδιο αποτέλεσμα (χρόνος ολοκλήρωσης: 328 sec.), χωρίς όμως να επηρεάζει την εκτέλεση των low priority benchmarks, που ολοκληρώνονται με τον ίδιο ρυθμό όπως στην κατάσταση Full Cache.

	Full Cache	Isolation	Protection	Controller
Completed (L.P.)	65	11	60	66
Completed (H.P.)	1	2	2	2
Provided Space	-	53.504kB	28.360kB	13.322kB

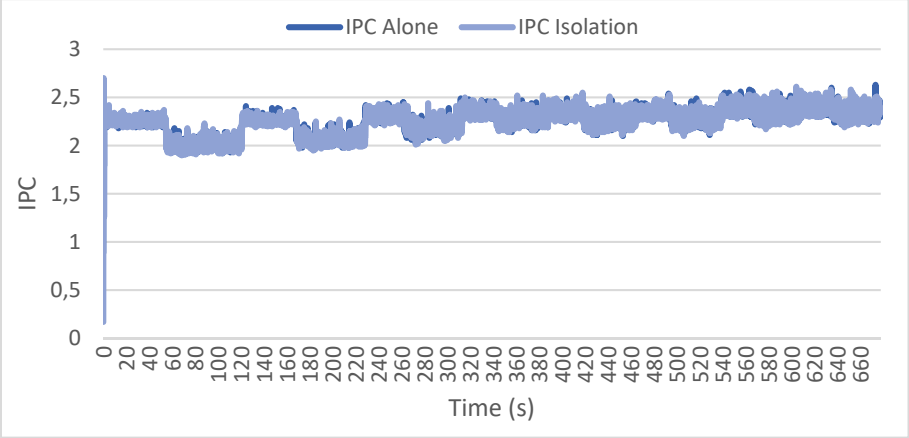
Πίνακας 20: Ολοκληρωμένα Benchmarks ανά Κατάσταση (Astar 2)

5.5.4 Περίπτωση 4η: Sphinx3 – Leslie3d

Στην 4η περίπτωση χρήσης αναλύεται το ζευγάρι Sphinx3 – Leslie3d. Το sphinx3 ανήκει στην Class D, καθώς έχει φάσεις που επαναλαμβάνονται περιοδικά. Κατά την εκτέλεση του ζευγαριού, ο λόγος επιβράδυνσης του sphinx3 είναι 4,01, και το IPC του είναι ίσο με το 25% του μέγιστου.

- **Κατάσταση Isolation – Take Over**

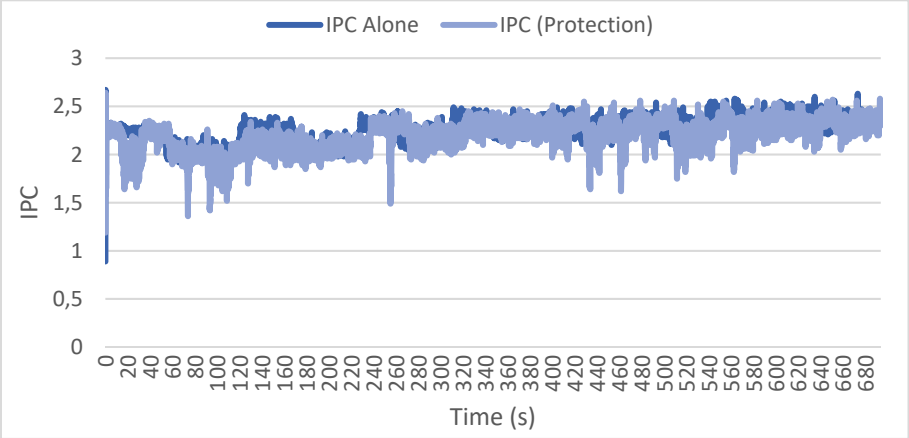
Στην κατάσταση Isolation – Take Over αντιμετωπίζεται πλήρως η επιβράδυνση που παρατηρείται στην εκτέλεση του sphinx3, καθώς ολοκληρώνεται σε 677 sec., ο οποίος είναι ίσος με το χρόνο ολοκλήρωσης στην κατάσταση «Alone».



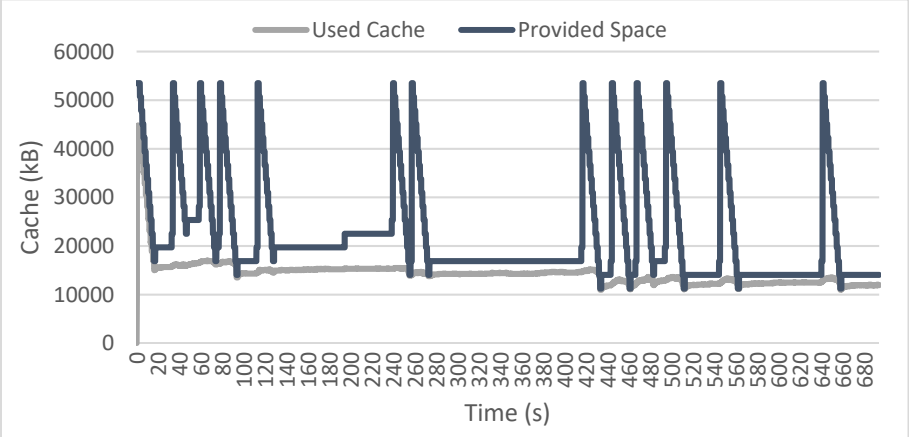
Γράφημα 37: Sphinx3 - Leslie3d (Isolation - Take Over)

- **Λειτουργία Μηχανισμού Προστασίας**

Το sphinx3 έχει αντίστοιχη επίδοση (της τάξης του 97,8% του IPC, χρονική επιβράδυνση: 1,02) και στην περίπτωση της λειτουργίας του μηχανισμού προστασίας. Συγχρόνως, παραχωρείται σημαντικό κομμάτι της cache στα low priority benchmarks, αφού η μέση παρεχόμενη cache είναι 22.440kB. Από αυτήν την cache, το sphinx3 χρησιμοποιεί το 64,3%, οπότε το συνολικό ποσοστό χρησιμοποίησης είναι 85,8%.



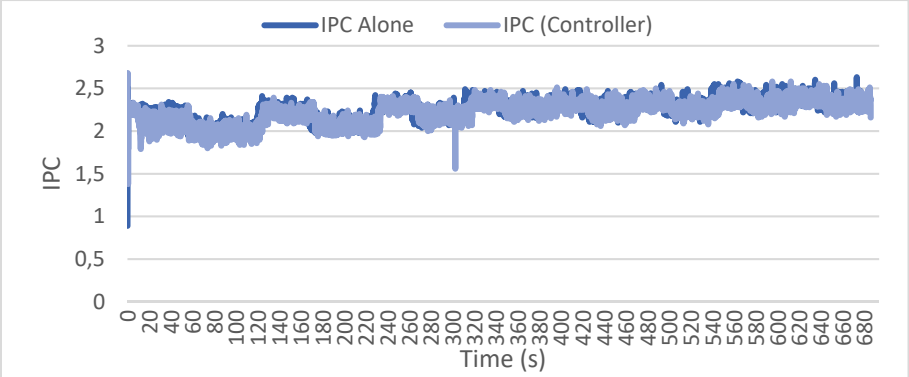
Γράφημα 38: Επίδοση του Sphinx3 (Μηχανισμός Προστασίας)



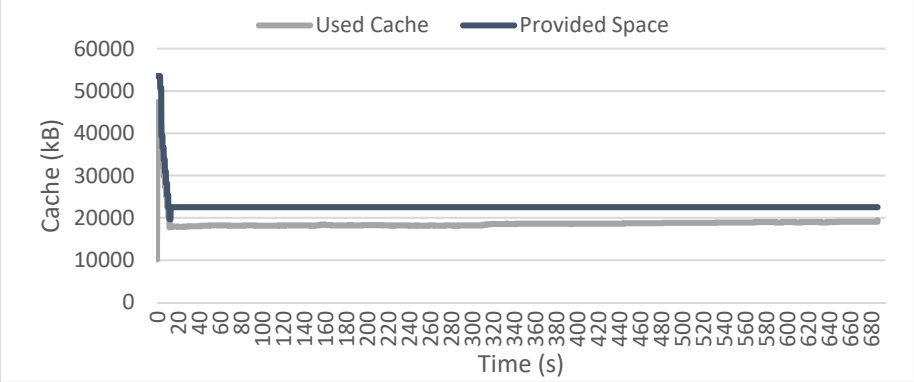
Γράφημα 39: Χρήση της Cache (Sphinx3, Μηχανισμός Προστασίας)

- **Λειτουργία του Ελεγκτή Χρησιμοποίησης**

Η χρήση του ελεγκτή χρησιμοποίησης της cache έχει τριπλό όφελος στο συγκεκριμένο ζευγάρι. Πρώτον, η επίδοση βελτιώνεται ελαφρώς από την προηγούμενη περίπτωση (στο 98,9% του μέγιστου IPC), δεύτερον, μειώνεται αισθητά το πλήθος των αποφάσεων του μηχανισμού, και τρίτον, βελτιώνεται το ποσοστό χρησιμοποίησης της cache, το οποίο είναι τώρα 82.4%, που οδηγεί σε ποσοστό χρήσης όλης της cache ίσο με 92,9%.



Γράφημα 40: Επίδοση του Sphinx3 (Ελεγκτής Χρησιμοποίησης)



Γράφημα 41: Χρήση της Cache (Sphinx3, Ελεγκτής Χρησιμοποίησης)

High Priority	IPC	IPC Sl.	Time	T. Sl.	MPKI	MBL	$Occ._{avg}$	Util.
Alone	2.24	-	677	-	0.00	0.10	56320	84.4%
Full Cache	0.56	25.0%	2714.6	4.01	11.38	841.01	56320	3.0%
Isolation	2.25	100.2%	677	1.00	0.00	0.48	53504	96.0%
Protection	2.19	97.8%	693.9	1.02	0.11	3.14	22440	64.3%
Controller	2.22	98.9%	686.3	1.01	0.04	0.33	22796	82.4%

Πίνακας 21: Χαρακτηριστικά του Sphinx3 ανά Κατάσταση Εκτέλεσης

- **Επίδοση των Low Priority Benchmarks**

Η χρήση του μηχανισμού προστασίας και του ελεγκτή χρησιμοποίησης ευνοεί την επίδοση των low priority benchmarks σε σχέση με την επιβολή της κατάστασης Isolation – Take Over. Όπως φαίνεται στον επόμενο πίνακα, η επιβράδυνση των 21 Leslie3d κατά τη λειτουργία του μηχανισμού βρίσκεται αρκετά κοντά με την επιβράδυνση στην κατάσταση Full Cache, και είναι κατά πολύ βελτιωμένη σε σχέση με την επιβράδυνση στην κατάσταση Isolation – Take Over.

	Alone	Full Cache	Isolation	Protection	Controller
Leslie3d (L.P.)	283 sec.	3.69	7.31	3.73	3.76
Sphinx3 (H.P.)	677 sec.	4.01	1.00	1.02	1.01

Πίνακας 22: Επιβράδυνση των L.P. Leslie3d ανά Κατάσταση (Sphinx3)

Το παράθυρο εκτέλεσης για τη δεύτερη φάση μετρήσεων επιλέχθηκε να είναι 1400 sec., χρόνος στον οποίο το sphinx3 μπορεί να ολοκληρωθεί 2 φορές. Αυτό συμβαίνει κατά τη λειτουργία του μηχανισμού, με κάθε sphinx3 να ολοκληρώνεται σε 680 sec. κατά μέσο όρο, παράλληλα με τη βελτίωση του throughput του συστήματος, σε αντίθεση με την κατάσταση Isolation – Take Over, στην οποία, ενώ κάθε sphinx3 ολοκληρώνεται σε μέσο χρόνο 677 sec., τα low priority benchmarks ολοκληρώνονται με πιο αργό ρυθμό, αλλά και την κατάσταση Full Cache, στην οποία το sphinx3 ολοκληρώνεται μόνο μία φορά, σε χρόνο 728 sec.

	Full Cache	Isolation	Protection	Controller
Completed (L.P.)	258	65	225	235
Completed (H.P.)	1	2	2	2
Provided Space	-	53.504kB	26.443kB	20.566kB

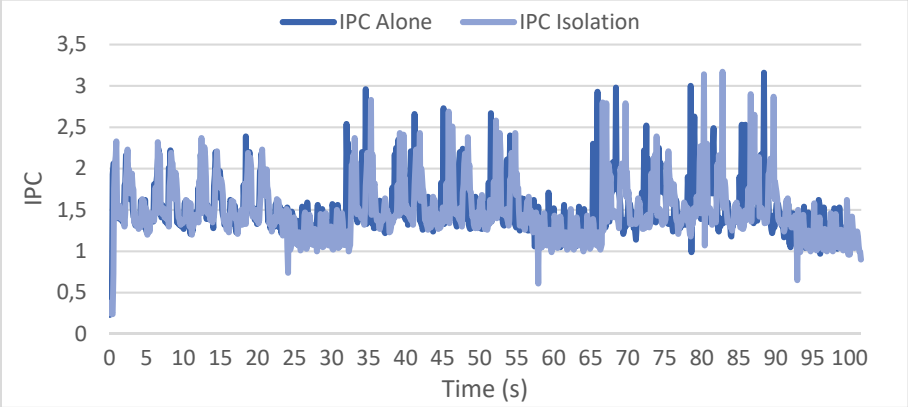
Πίνακας 23: Ολοκληρωμένα Benchmarks ανά Κατάσταση (Sphinx3)

5.5.5 Περίπτωση 5η: Bzip2 6 – Leslie3d

Στην πέμπτη περίπτωση χρήσης εξετάζεται το ζευγάρι Bzip2 6 – Leslie3d. Το bzip2 6 ανήκει στην Class E, καθώς, παρόλο που μπορεί να παρατηρηθεί μια περιοδικότητα στην επίδοση του, οι επιμέρους φάσεις δεν παρουσιάζουν καμία κανονικότητα. Η εκτέλεση του ζευγαριού αυτού προκαλεί σημαντική επιβράδυνση στο bzip2 6, καθώς ο λόγος επιβράδυνσης είναι 3,21.

- **Κατάσταση Isolation – Take Over**

Στην κατάσταση Isolation – Take Over μειώνεται αισθητά η επιβράδυνση, καθώς ο χρόνος εκτέλεσης είναι μόλις 3% μεγαλύτερος του βέλτιστου.

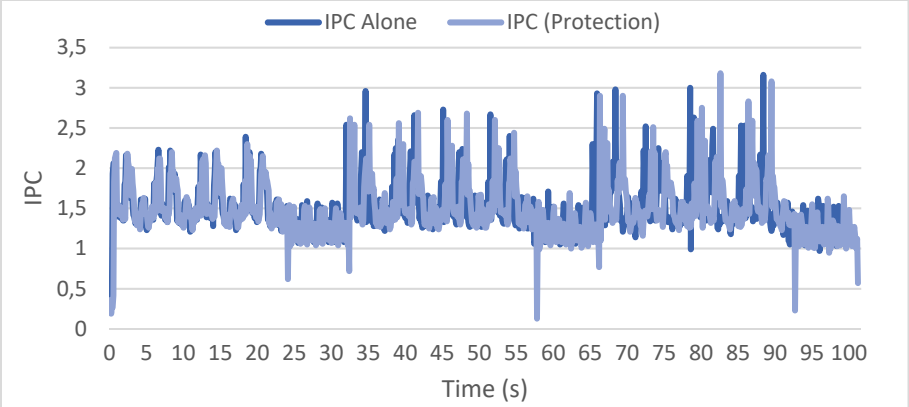


Γράφημα 42: Bzip2 6 - Leslie3d (Isolation - Take Over)

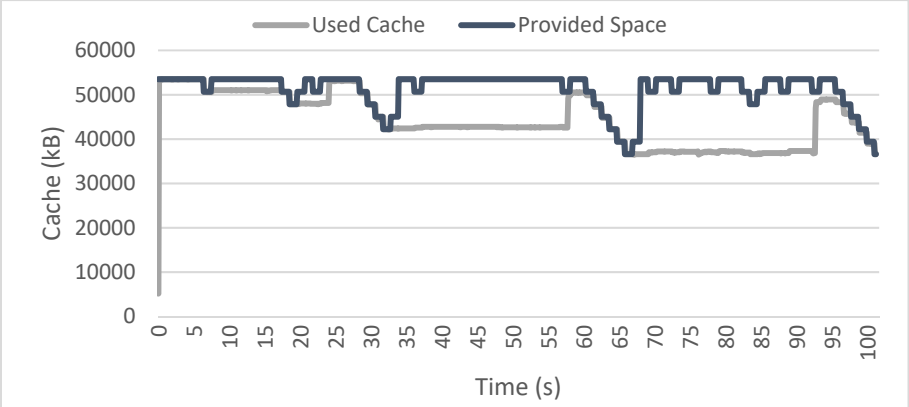
- **Λειτουργία του Μηχανισμού Προστασίας**

Ο μηχανισμός προστασίας, όπως προαναφέρθηκε, βασίζεται στη σταθερότητα της επίδοσης. Επομένως, η έλλειψη κανονικότητας στην επίδοση του Bzip2 6 είναι λογικό να τον οδηγήσει σε προσεκτικές αποφάσεις, καθώς η εύρεση σταθερότητας είναι, στην περίπτωση αυτή, αρκετά δύσκολη. Παρόλα

αυτά, το bzip2 6 έχει επίδοση πολύ κοντά στη βέλτιστη, αφού ο λόγος των IPC είναι 98,6%. Το μειονέκτημα που μπορεί να παρατηρηθεί είναι το μεγάλο μέγεθος της παρεχόμενης cache, το οποίο είναι αποτέλεσμα της λειτουργίας του μηχανισμού, και όχι των αναγκών του benchmark. Αυτό σημαίνει ότι είναι πιθανό το bzip2 6 να λειτουργούσε σε αντίστοιχα επίπεδα επίδοσης με μικρότερο μέρος της cache. Το bzip2 6, βέβαια, χρησιμοποιεί την cache που του δίνεται σε ποσοστό 86,3%, οπότε ολόκληρη η cache χρησιμοποιείται σε ποσοστό 87,5%. Στα παρακάτω γραφήματα φαίνεται η σύγκριση της επίδοσης με τη βέλτιστη επίδοση καθώς και η παρεχόμενη cache.



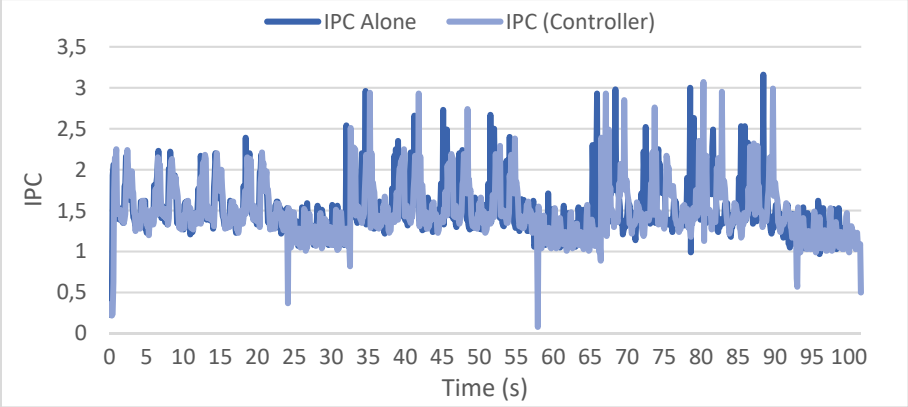
Γράφημα 43: Επίδοση του Bzip2 6 (Μηχανισμός Προστασίας)



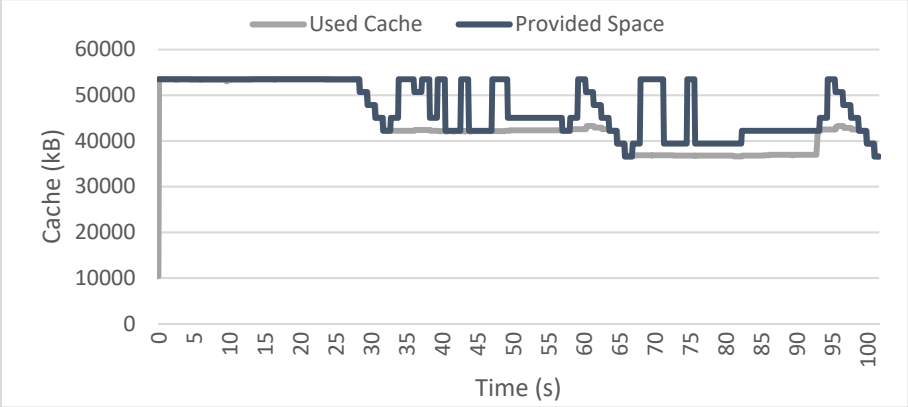
Γράφημα 44: Χρήση της Cache (Bzip2 6, Μηχανισμός Προστασίας)

- **Λειτουργία του Ελεγκτή Χρησιμοποίησης**

Η λειτουργία του ελεγκτή χρησιμοποίησης δεν έχει διαφορετική επίδραση στην επίδοση, καθώς επιτυγχάνεται αντίστοιχο επίπεδο με τις δύο προηγούμενες καταστάσεις, ούτε μειώνει την παρεχόμενη cache σε σημαντικό βαθμό (υπάρχει μια μείωση 3.800kB), βελτιώνει όμως τη χρησιμοποίηση της cache, καθώς το bzip2 6 χρησιμοποιεί το 92,9% της cache που του παρέχεται, άρα το συνολικό ποσοστό χρήσης της cache είναι 94%. Ακολουθούν οι αντίστοιχες γραφικές παραστάσεις.



Γράφημα 45: Επίδοση του Bzip2 6 (Ελεγκτής Χρησιμοποίησης)



Γράφημα 46: Χρήση της Cache (Bzip2 6, Ελεγκτής Χρησιμοποίησης)

High Priority	IPC	IPC Sl.	Time	T. Sl.	MPKI	MBL	$Occ._{avg}$	Util.
Alone	1.5	-	99	-	0.11	36.21	56320	97.8%
Full Cache	0.46	30.6%	326.5	3.30	5.11	177.78	56320	0.7%
Isolation	1.47	98.3%	102	1.03	0.10	36.39	53504	99.9%
Protection	1.48	98.6%	101.3	1.02	0.11	40.19	51230	86.3%
Controller	1.47	98.2%	101.7	1.03	0.11	38.01	47426	92.9%

Πίνακας 24: Χαρακτηριστικά του Bzip2 6 ανά Κατάσταση Εκτέλεσης

- **Επίδοση των Low Priority Benchmarks**

Χρησιμοποιώντας το μηχανισμό προστασίας αντί της κατάστασης Isolation – Take Over για την εξασφάλιση του επιθυμητού επιπέδου επίδοσης για το bzip2 6, αυξάνεται η επίδοση των 21 low priority benchmarks. Σε αυτήν την περίπτωση, η επίδοση δε φτάνει κοντά στο επίπεδο της κατάστασης Full Cache, όμως η βελτίωση είναι αισθητή. Επίσης, υπάρχει σημαντική βελτίωση στην επίδοση όταν χρησιμοποιείται ο ελεγκτής, καθώς ο μέσος χρόνος εκτέλεσης μειώνεται δραστικά. Στον επόμενο πίνακα παρουσιάζεται ο λόγος επιβράδυνσης των Leslie3d (μέσος χρόνος εκτέλεσης προς το χρόνο εκτέλεσης στην κατάσταση «Alone») ανά κατάσταση εκτέλεσης του ζευγαριού.

	Alone	Full Cache	Isolation	Protection	Controller
Leslie3d (L.P.)	283 sec.	3.67	7.36	5.31	4.40
Bzip2 6 (H.P.)	99 sec.	3.30	1.03	1.02	1.03

Πίνακας 25: Επιβράδυνση των L.P. Leslie3d ανά Κατάσταση (Bzip2 6)

Επιλέγοντας το παράθυρο εκτέλεσης ίσο με 200 sec., το bzip2 6 αναμένεται να ολοκληρωθεί 2 φορές, εφόσον η επίδοση του είναι βέλτιστη. Αυτό συμβαίνει στην κατάσταση Isolation – Take Over, καθώς και με τη λειτουργία του μηχανισμού, στις οποίες κάθε bzip2 6 ολοκληρώνεται σε 99 sec. αντί για 124 sec. στην κατάσταση Full Cache, ωστόσο η βελτίωση του throughput του συστήματος δεν είναι τόσο μεγάλη όσο στις προηγούμενες περιπτώσεις. Αυτό οφείλεται στο μεγάλο χώρο της cache που παρέχει ο μηχανισμός στο bzip2 6.

	Full Cache	Isolation	Protection	Controller
Completed (L.P.)	28	5	10	16
Completed (H.P.)	1	2	2	2
Provided Space	-	53.504kB	51.407kB	47.131kB

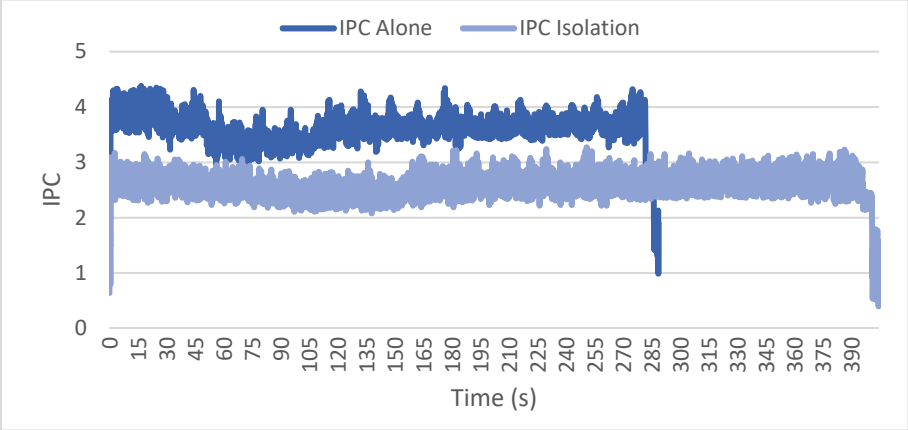
Πίνακας 26: Ολοκληρωμένα Benchmarks ανά Κατάσταση (Bzip2 6)

5.5.6 Περίπτωση 6η: Libquantum – Lbm

Όπως αναφέρθηκε, η επίδοση της κατάστασης «Alone» δεν αποτελεί εφικτό στόχο για όλες τις περιπτώσεις. Στις περιπτώσεις που ισχύει το προηγούμενο, ο εφικτός στόχος του μηχανισμού προστασίας είναι να φτάσει στο επίπεδο επίδοσης της κατάστασης Isolation – Take Over. Οι δύο περιπτώσεις χρήσης που ακολουθούν εμπίπτουν σε αυτήν την κατηγορία. Η πρώτη από τις δύο περιπτώσεις χρήσης αφορά το ζευγάρι Libquantum – Lbm. Το libquantum ανήκει στην Class B. Το lbm προκαλεί, κατά την εκτέλεση του ζευγαριού με ελεύθερη πρόσβαση στην cache, σημαντική επιβράδυνση στο libquantum, καθώς ο λόγος επιβράδυνσης είναι 8,39. Δηλαδή, το libquantum στην κατάσταση «Alone» ολοκληρώνεται σε 4′48″, ενώ όταν εκτελείται μαζί με το lbm ολοκληρώνεται σε 40′15″. Αντίστοιχα, το IPC του είναι το 12% του βέλτιστου.

- **Κατάσταση Isolation – Take Over**

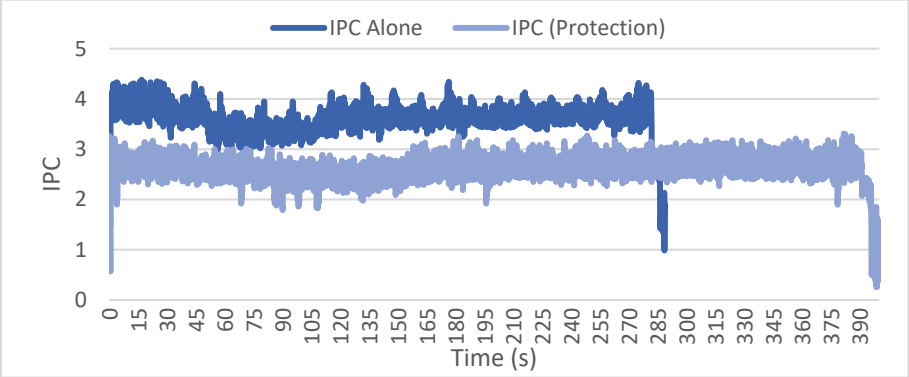
Η κατάσταση Isolation – Take Over δεν πλησιάζει την επίδοση της κατάστασης «Alone», αλλά εξασφαλίζει σημαντική βελτίωση, καθώς ο χρόνος ολοκλήρωσης είναι, σε αυτήν την περίπτωση, 40% μεγαλύτερος από το βέλτιστο χρόνο, και το IPC φτάνει στο 71% του βέλτιστου. Στο επόμενο γράφημα είναι εμφανής η διαφορά μεταξύ των καταστάσεων Isolation – Take Over και Alone.



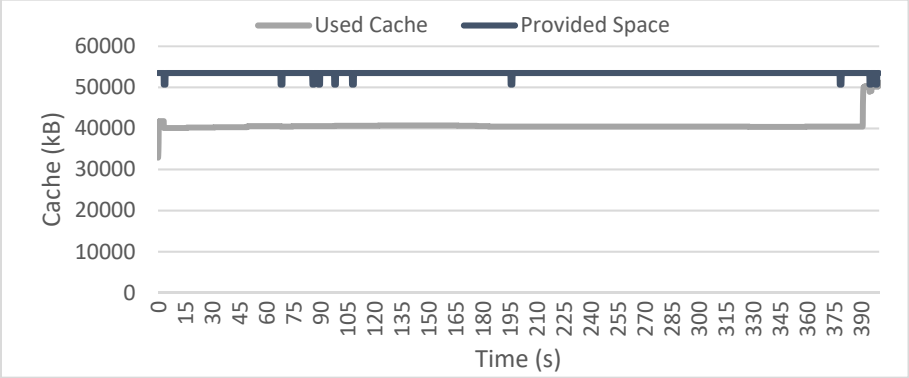
Γράφημα 47: Libquantum - Lbm (Isolation - Take Over)

- **Λειτουργία Μηχανισμού Προστασίας**

Η χρήση του μηχανισμού προστασίας επιβεβαιώνει την ανάγκη του libquantum για την παροχή μεγάλου μέρους της cache, η οποία όμως δε χρησιμοποιείται στο σύνολο της (το ίδιο συμβαίνει και στην κατάσταση Isolation – Take Over). Συγκεκριμένα, επιτυγχάνεται το ίδιο επίπεδο επίδοσης, το οποίο είναι θετικό για το μηχανισμό, αλλά το ποσοστό χρήσης της παρεχόμενης cache είναι 76,2%. Επειδή η cache παρέχεται σχεδόν εξ' ολοκλήρου στο libquantum, το ποσοστό χρήσης ολόκληρης της cache δε διαφέρει πολύ, καθώς είναι 77,4%.



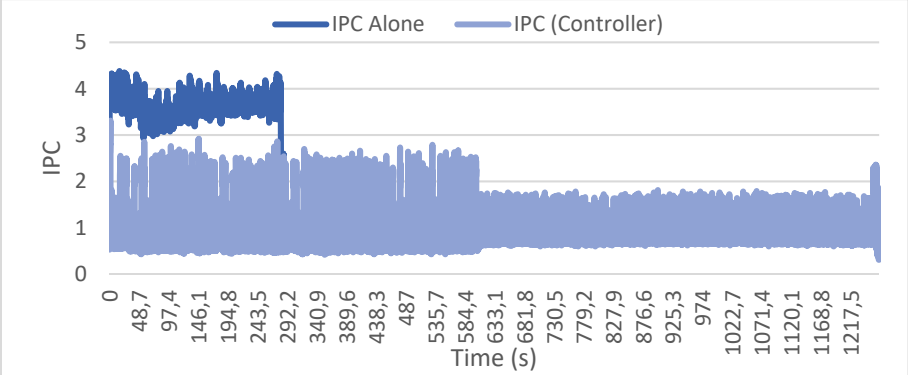
Γράφημα 48: Επίδοση του Libquantum (Μηχανισμός Προστασίας)



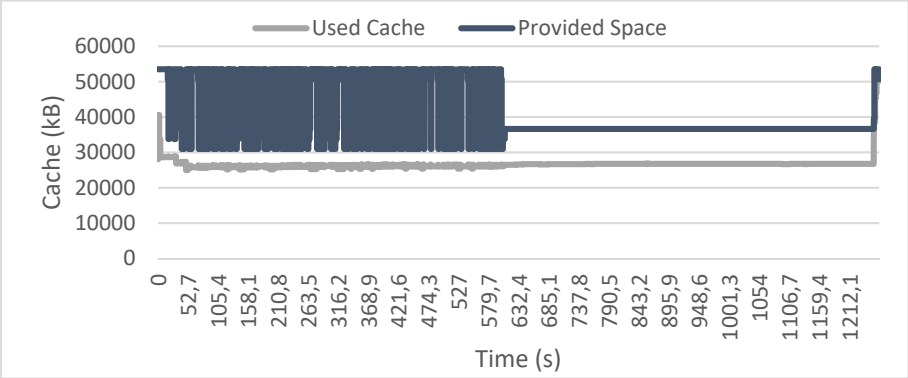
Γράφημα 49: Χρήση της Cache (Libquantum, Μηχανισμός Προστασίας)

- **Λειτουργία Ελεγκτή Χρησιμοποίησης**

Η χρήση του ελεγκτή χρησιμοποίησης, ενώ θεωρητικά θα έπρεπε να αφαιρέσει από το libquantum την cache που δε χρησιμοποιείται, χωρίς να υπάρχει πρόβλημα, αποτυγχάνει να βρει το σημείο λειτουργίας στην cache, και να εξασφαλίσει ένα αποδεκτό επίπεδο επίδοσης. Συγκεκριμένα, αυξάνει σημαντικά την επιβράδυνση σε σχέση με τις δύο προηγούμενες καταστάσεις, αφού σε αυτήν την περίπτωση, το libquantum έχει χρόνο ολοκλήρωσης 339% μεγαλύτερο από τον ιδανικό. Πρέπει ωστόσο να σημειωθεί ότι, ακόμα και υπό αυτές τις συνθήκες, η αρχική επιβράδυνση βελτιώνεται σημαντικά. Ο χώρος που παρέχεται στην cache είναι λιγότερος σε σχέση με την προηγούμενη κατάσταση (41.634kB από 53.413kB), αλλά μειώνεται και το ποσοστό χρήσης της παρεχόμενης cache που είναι 64%. Το ποσοστό χρήσης ολόκληρης της cache είναι 73,4%. Στα επόμενα γραφήματα φαίνεται η αδυναμία του μηχανισμού να αποφασίσει το σημείο λειτουργίας για μεγάλο χρονικό διάστημα, και η επίδραση της στην επίδοση.



Γράφημα 50: Επίδοση του Libquantum (Ελεγκτής Χρησιμοποίησης)



Γράφημα 51: Χρήση της Cache (Libquantum, Ελεγκτής Χρησιμοποίησης)

High Priority	IPC	IPC Sl.	Time	T. Sl.	MPKI	MBL	$Occ._{avg}$	Util.
Alone	3.60	-	288	-	0.15	1.12	56320	91.1%
Full Cache	0.43	12.0%	2415.4	8.39	17.90	1319.78	56320	4.2%
Isolation	2.57	71.4%	405	1.40	0.37	0.93	53504	80.4%
Protection	2.60	72.2%	400.1	1.39	0.29	0.68	53413	76.2%
Controller	0.82	22.9%	1264.1	4.39	5.32	56.51	41634	64.0%

Πίνακας 27: Χαρακτηριστικά του Libquantum ανά Κατάσταση Εκτέλεσης

- **Επίδοση Low Priority Benchmarks**

Όπως φάνηκε και στα προηγούμενα γραφήματα, το libquantum απαιτεί μεγάλο μέρος της cache, ώστε να έχει ένα αποδεκτό επίπεδο επίδοσης. Επομένως, στην περίπτωση του μηχανισμού προστασίας, δεν αναμένεται μεγάλη διαφορά με την κατάσταση Isolation – Take Over. Αντίθετα, στην περίπτωση που ο ελεγκτής είναι ενεργός, τα low priority benchmarks επωφελούνται από την cache που τους παρέχει ο μηχανισμός, ο οποίος, όμως, δεν πέτυχε το στόχο του, καθώς η επίδοση του libquantum δεν πλησίασε το επίπεδο της επίδοσης στην κατάσταση Isolation – Take Over.

	Alone	Full Cache	Isolation	Protection	Controller
Lbm (L.P.)	428 sec.	4.32	8.72	8.19	5.10
Libquantum (H.P.)	288 sec.	8.39	1.40	1.39	4.39

Πίνακας 28: Επιβράδυνση των L.P. Lbm ανά Κατάσταση (Libquantum)

Το παράθυρο εκτέλεσης για τη δεύτερη φάση μετρήσεων επιλέχθηκε ίσο με 900 sec., χρόνος στον οποίο το libquantum μπορεί να ολοκληρωθεί 3 φορές έχοντας τη μέγιστη επίδοση. Ωστόσο, αυτό δεν επιτυγχάνεται σε καμία από τις

καταστάσεις, καθώς επηρεάζεται από τις συνεκτελούμενες εφαρμογές. Στην κατάσταση Isolation – Take Over, καθώς και στην κατάσταση λειτουργίας του μηχανισμού, το libquantum ολοκληρώνεται 2 φορές, σε μέσο χρόνο 310 sec (επομένως η τρίτη εκτέλεση έφτασε κοντά στην ολοκλήρωση της), ενώ στην κατάσταση Full Cache, ολοκληρώνεται μία φορά σε 638 sec. Το throughput του συστήματος δεν παρουσιάζει μεγάλη βελτίωση στην περίπτωση του μηχανισμού προστασίας. Αντίθετα, το πλήθος των ολοκληρωμένων εφαρμογών είναι μεγαλύτερο όταν ο ελεγκτής είναι ενεργός, σε βάρος όμως της επίδοσης του libquantum, καθώς ένα εκ των δύο ολοκληρώνεται σε χρόνο 440 sec.

	Full Cache	Isolation	Protection	Controller
Completed (L.P.)	111	35	48	74
Completed (H.P.)	1	2	2	2
Provided Space	-	53.504kB	53.057kB	47.389kB

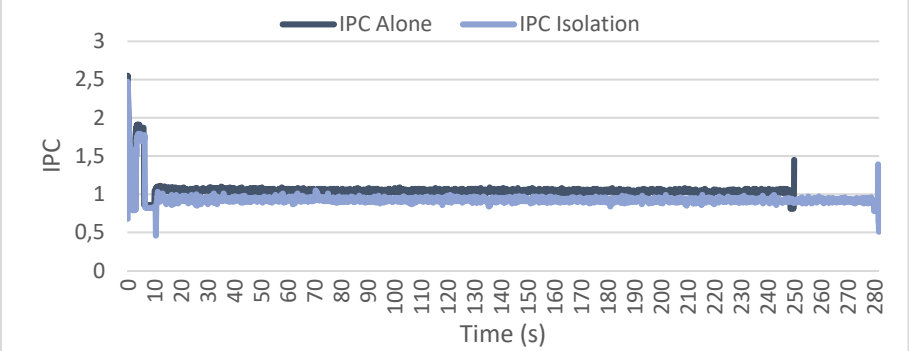
Πίνακας 29: Ολοκληρωμένα Benchmarks ανά Κατάσταση (Libquantum)

5.5.7 Περίπτωση 7η: Omnetpp – Lbm

Στην έβδομη περίπτωση παρουσιάζεται το ζευγάρι Omnetpp – Lbm. Το Omnetpp ανήκει στην Class A, καθώς το IPC του είναι σχετικά σταθερό. Η επιβράδυνση που προκαλεί το Lbm στο Omnetpp κατά τη συνεκτέλεση τους είναι η μεγαλύτερη που παρατηρήθηκε στο σύνολο των 4.225 ζευγαριών. Ο χρόνος ολοκλήρωσης του Omnetpp είναι 8,87 φορές μεγαλύτερος από το βέλτιστο, δηλαδή από 4′10″ που είναι ο χρόνος ολοκλήρωσης όταν το omnetpp εκτελείται μόνο του, ολοκληρώνεται σε 36′58″ όταν συνεκτελείται με το lbm. Στο Γράφημα 10 (σελ. 44) συγκρίνονται το IPC της συνεκτέλεσης με το βέλτιστο IPC της κατάστασης Alone. Ο λόγος των δύο IPC είναι 11%.

- **Κατάσταση Isolation – Take Over**

Η επίδοση στην κατάσταση Isolation – Take Over αυξάνεται αισθητά, καθώς το IPC φτάνει το 88,9% του βέλτιστου, και η επιβράδυνση μειώνεται σε 1,13. Στο επόμενο γράφημα φαίνεται η σύγκριση των αποδόσεων στις δύο καταστάσεις.

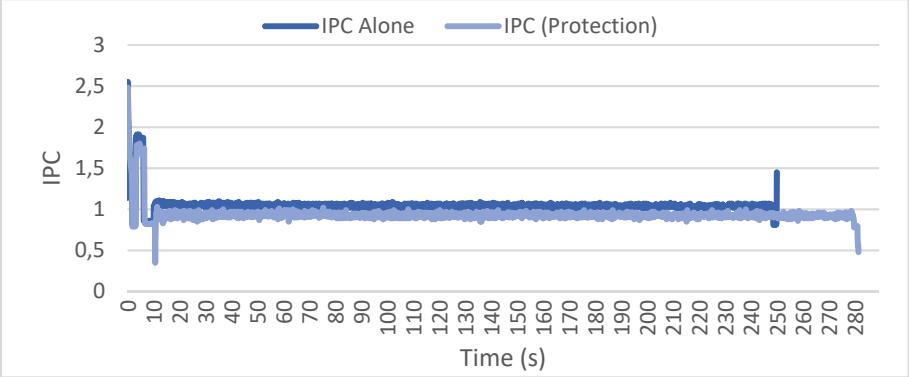


Γράφημα 52: Omnetpp - Lbm (Isolation - Take Over)

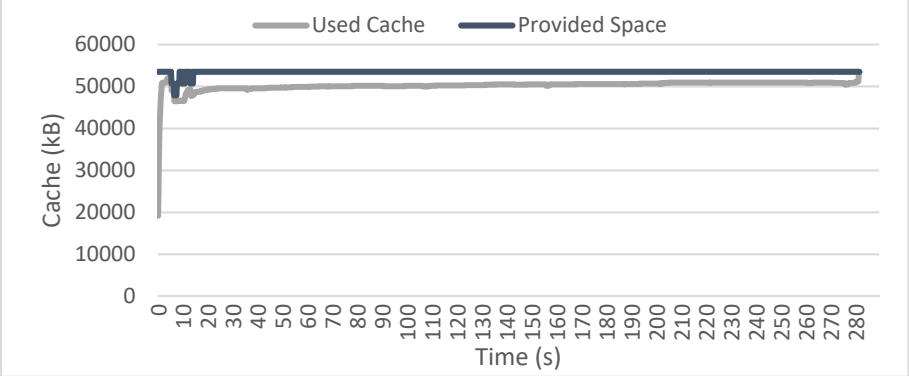
- **Λειτουργία Μηχανισμού Προστασίας**

Ο μηχανισμός προστασίας εντοπίζει την ανάγκη του Omnetpp για παροχή μεγάλου μέρους της cache, και του παρέχει τον ανάλογο χώρο. Έτσι επιτυγχάνεται αντίστοιχο επίπεδο επίδοσης με την κατάσταση Isolation – Take Over, της τάξης του 89%. Σε αυτήν την περίπτωση δεν υπάρχουν ιδιαίτερα

οφέλη για τα low priority benchmarks, αλλά εξασφαλίζεται ότι η εφαρμογή υψηλής προτεραιότητας θα έχει όσο το δυνατό καλύτερη επίδοση. Η cache που παρέχεται, καταναλώνεται από το omnetpp σε ποσοστό 94,1%. Το ποσοστό χρήσης όλης της cache είναι 94,4%. Τα προηγούμενα φαίνονται στα παρακάτω γραφήματα.



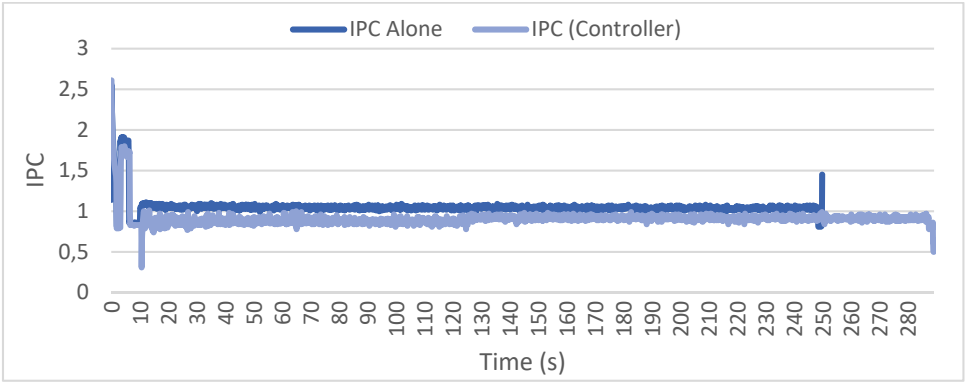
Γράφημα 53: Επίδοση του Omnetpp (Μηχανισμός Προστασίας)



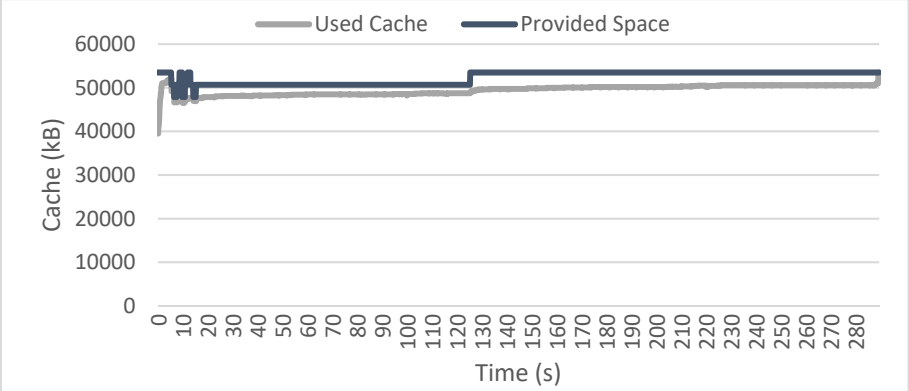
Γράφημα 54: Χρήση της Cache (Omnetpp, Μηχανισμός Προστασίας)

- **Λειτουργία του Ελεγκτή Χρησιμοποίησης**

Η χρήση του ελεγκτή οδηγεί την επίδοση σε ελαφρώς χαμηλότερο επίπεδο (λόγος των IPC 86.5%). Η χρήση της cache παραμένει στο προηγούμενο επίπεδο, καθώς το ποσοστό είναι 94,5%.



Γράφημα 55: Επίδοση του Omnetpp (Ελεγκτής Χρησιμοποίησης)



Γράφημα 56: Χρήση της Cache (Omnetpp, Ελεγκτής Χρησιμοποίησης)

High Priority	IPC	IPC Sl.	Time	T. Sl.	MPKI	MBL	$Occ._{avg}$	Util.
Alone	1.05	-	250	-	0.33	4.43	56320	97.3%
Full Cache	0.12	11.3%	2218.1	8.87	28.74	327.29	56320	1.5%
Isolation	0.94	88.9%	282	1.13	0.48	11.69	53504	99.6%
Protection	0.94	89.0%	281.4	1.13	0.43	3.85	53419	94.1%
Controller	0.91	86.5%	289.3	1.16	0.56	12.12	52308	94.5%

Πίνακας 30: Χαρακτηριστικά του Omnetpp ανά Κατάσταση Εκτέλεσης

- **Επίδοση των Low Priority Benchmarks**

Η διαφορά μεταξύ της χρήσης του μηχανισμού και της κατάστασης Isolation – Take Over, όσον αφορά την επίδοση των low priority benchmarks είναι μικρή, καθώς το σημείο λειτουργίας που επιλέγει ο μηχανισμός είναι (στην περίπτωση του ανενεργού ελεγκτή) τα 19 ways.

	Alone	Full Cache	Isolation	Protection	Controller
Lbm (L.P.)	428 sec.	4.25	8.73	7.78	6.18
Omnetpp (H.P.)	250 sec.	8.87	1.13	1.13	1.16

Πίνακας 31: Επιβράδυνση των L.P. Lbm ανά Κατάσταση (Omnetpp)

Για το δεύτερο μέρος της μέτρησης της επίδοσης των low priority benchmarks, το παράθυρο εκτέλεσης ορίστηκε στα 600 sec. Σε αυτό το χρόνο το omnetpp αναμένεται να ολοκληρωθεί 2 φορές, αν έχει τη βέλτιστη επίδοση, πράγμα το οποίο επιτυγχάνεται τόσο στην κατάσταση Isolation – Take Over (σε μέσο χρόνο 260 sec.), όσο και στη λειτουργία του μηχανισμού (σε μέσο χρόνο 270 sec.), κατά την οποία παρατηρείται και σημαντική αύξηση του throughput του συστήματος, καθώς ολοκληρώνεται μεγαλύτερο πλήθος εφαρμογών. Στην κατάσταση Full Cache παρατηρείται το βέλτιστο throughput, όμως η επίδοση του omnetpp είναι πολύ χαμηλότερη της βέλτιστης, καθώς ο χρόνος ολοκλήρωσης είναι 583 sec.

	Full Cache	Isolation	Protection	Controller
Completed (L.P.)	110	24	72	64
Completed (H.P.)	1	2	2	2
Provided Space	-	53.504kB	44.749kB	46.753kB

Πίνακας 32: Ολοκληρωμένα Benchmarks ανά Κατάσταση (Omnetpp)

5.6 Παράμετροι του Μηχανισμού

Συνοπτικά, οι παράμετροι του μηχανισμού που μπορούν να οριστούν από το χρήστη είναι οι ακόλουθοι:

- Μέγεθος της cache που είναι αρχικά διαθέσιμο στην προστατευόμενη εφαρμογή. Η προεπιλογή, αν η cache έχει N ways of associativity, είναι $N-1$. Στην περίπτωση του επεξεργαστή που χρησιμοποιήθηκε στην αξιολόγηση του μηχανισμού και του ελεγκτή, η τιμή αυτή είναι 19 ways, ωστόσο μπορεί να επιλεχθεί διαφορετική αρχική ρύθμιση.
- Μέγεθος n του παραθύρου των μετρήσεων που αποθηκεύονται. Η προεπιλεγμένη τιμή είναι 10. Όσο μικρότερο μέγεθος επιλεχθεί, τόσο περισσότερες αποφάσεις θα κληθεί να λάβει ο μηχανισμός.
- Χρονικό διάστημα Δt μεταξύ της λήψης των μετρήσεων. Η προεπιλεγμένη τιμή είναι 100ms.
- Ποσοστό επίτρεψης ισότητας a για την ισότητα δτιμών IPC. Η προεπιλεγμένη τιμή είναι 5%. Όσο μικρότερο είναι το ποσοστό, τόσο πιο ευαίσθητος σε αλλαγές του IPC γίνεται ο μηχανισμός.
- Όριο αμελητέου bandwidth. Η προεπιλεγμένη τιμή είναι 27.5 Mbps. Η τιμή αυτή επιλέχθηκε διότι, με αυτήν την ταχύτητα, μπορούν να καταληφθούν 2816 kB σε 100 ms, όπου 2816 kB είναι το μέγεθος ενός way της cache του επεξεργαστή που μελετούμε, και 100ms είναι το ελάχιστο διάστημα μεταξύ δύο μετρήσεων της εφαρμογής PQoS. Η τιμή αυτή, επομένως, διαφέρει ανάλογα με το μέγεθος της cache, το εργαλείο που χρησιμοποιείται για την εφαρμογή των CMT και CAT, αλλά και από τις ανάγκες της εφαρμογής που προστατεύεται.
- Ενεργοποίηση του ελεγκτή χρησιμοποίησης της cache. Η χρήση ή όχι του ελεγκτή εξαρτάται από τις ανάγκες της χρήσης του μηχανισμού, καθώς και από τη συμπεριφορά της high priority εφαρμογής.
- Πυρήνας εκτέλεσης της high priority εφαρμογής. Η εκτέλεση της εφαρμογής που θα προστατευτεί μπορεί να τοποθετηθεί σε οποιονδήποτε πυρήνα, με τον περιορισμό ότι δε μετακινείται σε άλλον πυρήνα κατά την εκτέλεση της.
- Πολιτική καταμερισμού, μεταξύ επικάλυψης και απομόνωσης, η οποία είναι και η προεπιλογή. Στην πολιτική επικάλυψης, η high priority εφαρμογή έχει πρόσβαση στο σύνολο της cache, ενώ ο χώρος που κατανέμεται στις low priority εφαρμογές είναι ίδιος όπως και στην περίπτωση της απομόνωσης. Στην επόμενη εικόνα φαίνονται οι αρχικές κατανομές για κάθε μία από τις πολιτικές που περιεγράφηκαν.

	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18	W19	W20
COS1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
COS2																					1

Πολιτική Απομόνωσης

	W0	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17	W18	W19	W20
COS1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
COS2																					1

Πολιτική Επικάλυψης

Εικόνα 10: Αρχική Κατανομή της Cache ανά Πολιτική

6. Επίλογος

6.1 Συμπεράσματα

Στην παρούσα διπλωματική εργασία παρουσιάστηκαν οι τεχνολογίες που ανέπτυξε η Intel για την παρακολούθηση της χρήσης της cache, και για την επιβολή καταμερισμού της cache μεταξύ των εφαρμογών και των πυρήνων. Εξετάστηκαν οι δυνατότητες, τα επίπεδα αφαίρεσης και τα εργαλεία που παρέχονται στον χρήστη για τη διαχείριση της cache που παρέχονται μέσω των CMT και CAT.

Ο μηχανισμός που προτάθηκε, βασίζεται και συνδυάζει τις δύο τεχνολογίες, με σκοπό να βελτιώσει την ποιότητα υπηρεσίας, δηλαδή να προστατεύσει μια εφαρμογή υψηλής προτεραιότητας από την επίδραση του ανταγωνισμού για την cache μεταξύ των συνεκτελούμενων στο σύστημα εφαρμογών. Ο μηχανισμός βασίζεται στη μετρική του IPC, ενώ η επέκταση που παρουσιάστηκε χρησιμοποιεί τη μέτρηση για τη χρήση της cache (cache occupancy) και τη χρήση του διαύλου δεδομένων από την εφαρμογή υψηλής προτεραιότητας. Αποθηκεύοντας ένα μικρό αριθμό μετρήσεων, ο μηχανισμός μειώνει ή αυξάνει το χώρο που παρέχεται στην εφαρμογή υψηλής προτεραιότητας ανάλογα με τις ανάγκες της. Οι μεταβλητές του μηχανισμού προσφέρουν τη δυνατότητα της προσαρμογής του σε εφαρμογές με διαφορετικά χαρακτηριστικά, αλλά και σε διαφορετικές απαιτήσεις για επίδοση.

Μέσω της χρήσης της υλοποίησης της Intel CMT – CAT και των σουιτών SPEC2006 και Parsec 3.0 αναγνωρίσαμε αρχικά το πρόβλημα, καθώς προέκυψε ότι η επιβράδυνση που προκαλείται σε μια εφαρμογή λόγω της συνεκτέλεσης της με άλλες εφαρμογές δεν είναι αμελητέα στο 40% των περιπτώσεων. Ακολούθως, επιχειρήθηκε να δοθεί μια λύση στο πρόβλημα αυτό με στατική κατανομή της cache, η οποία όμως έχει ως αποτέλεσμα τη μείωση του throughput του συστήματος αλλά και την άδικη κατανομή της cache, καθώς σε αρκετές περιπτώσεις η προστατευόμενη εφαρμογή έχει αποκλειστική πρόσβαση σε μεγαλύτερο μέρος της cache από αυτό που της είναι αναγκαίο. Για αυτόν το λόγο, χρησιμοποιείται ο δυναμικός μηχανισμός προστασίας, ο οποίος παράγει ανάλογα αποτελέσματα όσον αφορά την επίδοση της εφαρμογής υψηλής προτεραιότητας, καθώς η μέση διαφορά των χρόνων εκτέλεσης μεταξύ των δύο λειτουργιών του μηχανισμού και της κατάστασης Isolation – Take Over στο υποσύνολο του προβλήματος που παρουσιάστηκε είναι μόλις 2%. Ταυτόχρονα αυξάνεται το throughput του συστήματος, μέσω της βελτίωσης της επίδοσης των εφαρμογών χαμηλής προτεραιότητας. Όλα τα προηγούμενα παρουσιάζονται μέσω της ανάλυσης 7 περιπτώσεων χρήσης.

6.2 Μελλοντικές Κατευθύνσεις

Η παρούσα εργασία θα μπορούσε να επεκταθεί προς πολλές κατευθύνσεις, καθώς οι δυνατότητες που προσφέρουν οι νέες τεχνολογίες της Intel είναι πολυάριθμες.

Πρώτον, είναι δυνατή η επέκταση του παρόντος μηχανισμού για την προστασία περισσότερων από μίας εφαρμογών. Για παράδειγμα, εάν δύο εφαρμογές έχουν ίδια (υψηλή) προτεραιότητα, μπορεί να επεκταθεί η

λειτουργία του μηχανισμού, ώστε να ελέγχει και τις δύο εφαρμογές και να κατανείμει τον χώρο στην cache κατάλληλα, ώστε να προστατεύσει την επίδοση τους. Επίσης, στην περίπτωση ύπαρξης μιας ιεραρχίας προτεραιότητας, μπορεί ο μηχανισμός να έχει ως πρωταρχικό στόχο την προστασία της εφαρμογής με την υψηλότερη προτεραιότητα, και ακολούθως, εφόσον κριθεί δυνατό, να παρέχει τον απαραίτητο χώρο στην cache για τη βελτιστοποίηση της επίδοσης της επόμενης εφαρμογής στην ιεραρχία.

Μια δεύτερη κατεύθυνση είναι η απεξάρτηση της εφαρμογής από την εκτέλεση σε ένα συγκεκριμένο πυρήνα, αλλά και η προστασία εφαρμογών που έχουν περισσότερες από μία διεργασίες. Ο παρών μηχανισμός επιτρέπει την προστασία μιας διεργασίας εφόσον εκτελείται μόνη της σε έναν μόνο πυρήνα. Αυτός ο περιορισμός μπορεί να αρθεί εφόσον γίνουν οι κατάλληλες τροποποιήσεις στο μηχανισμό και στην υλοποίηση της Intel CMT – CAT ώστε να γίνει πιο φιλική προς το χρήστη η διαχείριση των RMID. Έτσι, θα είναι δυνατή η προστασία εφαρμογών με περισσότερες από μία διεργασίες μέσω της ομαδοποίησης τους με το ίδιο RMID, ενώ η επιβολή της κατανομής θα γίνεται αυτόματα από την CAT καθώς η διεργασία θα μετακινείται από πυρήνα σε πυρήνα. Το προηγούμενο μπορεί να γίνει πράξη με τη μεταφορά του μηχανισμού σε μία υλοποίηση όπως το cgroups, που χρησιμοποιεί τον κατάλληλα τροποποιημένο χρονοδρομολογητή του O.S, μέσω του οποίου μπορεί να γίνει αυτόματα η αντιστοίχιση των RMID και COS στους πυρήνες που εκτελείται η εφαρμογή υψηλής προτεραιότητας.

Σε κάποιες περιπτώσεις παρατηρήθηκε πως η απομόνωση της χρήσης της cache μεταξύ των εφαρμογών δε βελτιώνει την απόδοση της high priority εφαρμογής. Αυτό οφείλεται στον ανταγωνισμό μεταξύ των εφαρμογών για άλλους κοινόχρηστους πόρους όπως η χρήση του διαύλου δεδομένων. Επομένως, μία μελλοντική έρευνα που θα έδινε λύση στα προβλήματα τα οποία απέτυχε να λύσει ο καταμερισμός της cache, είναι η ανάπτυξη διαχειριστικών εργαλείων για άλλους κοινόχρηστους πόρους. Για παράδειγμα, η διαχείριση της πρόσβασης του διαύλου δεδομένων μπορεί να επιβάλλει όριο στη χρήση του εύρους ζώνης σε κάθε εφαρμογή. Με αυτόν τον τρόπο θα αποφευχθεί η υπερχρησιμοποίηση του διαύλου από συγκεκριμένες επιθετικές εφαρμογές.

Τέλος, είναι άξια μελέτης η επίδραση της χρήσης της τεχνολογίας Code – Data Prioritization που παρουσιάστηκε στην ενότητα 3.2.3, καθώς είναι πιθανό η απομόνωση εντολών και δεδομένων να αποδειχθεί ευεργετική για κάποιες εφαρμογές, και με αυτόν τον τρόπο να αυξηθεί η επίδοση τους.

Βιβλιογραφία

[1] G. Moore, «Cramming more components onto integrated circuits,» *Electronics Magazine,* 19 Απριλίου 1965.

[2] D. Chandra, F. Guo, S. Kim και Y. Solihin, «Predicting interthread cache contention on a chip multi-processor architecture,» σε *HPCA '05*, Washington DC, 2005.

[3] R. Iyer, «CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms,» σε *Int'l Conference on Supercomputing (ICS)*, 2004.

[4] S. Kim, D. Chandra και Y. Solihin, «Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture,» σε *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, PACT 2004*, 2004.

[5] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal και R. Iyer, «Cache QoS: From Concept to Reality in the Intel® Xeon® Processor E5-2600 v3 Product Family,» σε *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Barcelona, Spain, 2016.

[6] A. Herdrich, R. Illikkal, R. Iyer και D. Newell, «Rate-based QoS techniques for cache/memory in CMP platforms,» σε *ICS '09*, 2009.

[7] L. Hsu, S. Reinhardt, R. Iyer και S. Makineni, «Communist, utilitarian, and capitalist cache policies on CMPs: caches as a shared resource,» σε *15th International Conference on Parallel Architectures and Compilation Techniques, PACT 2006*, Seattle, Washington, 2006.

[8] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu και S. Reinhardt, «QoS policies and architecture for cache/memory in cmp platforms,» σε *SIGMETRICS '07*, 2007.

[9] A. Jaleel, W. Hasenplaugh, M. Qureshi, J. Sebot, S. Steely και J. Emer, «Adaptive insertion policies for managing shared caches,» σε *PACT '08*, 2008.

[10] J. Mars, L. Tang, R. Hundt, K. Skadron και M. L. Soffa, «Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via sensible co-locations,» σε *44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011.

[11] M. Qureshi και Y. Patt, «Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches,» σε *MICRO 39*, 2006.

[12] N. Rafique, W. T. Lim και M. Thottethodi, «Effective management of DRAM bandwidth in multicore processors,» σε *PACT 2007*, 2007.

[13] G. Chatzopoulos, A. Dragojevic και R. Guerraoui, «ESTIMA: Extrapolating ScalabiliTy of In-Memory Applications,» Barcelona, Spain, 2016.

[14] L. Tang, J. Mars, N. Vachharajani, R. Hundt και M. L. Soffa, «The impact of memory subsystem resource sharing on datacenter applications,» σε *ISCA '11*, New York, 2011.

[15] C. Xu, X. Chen, R. Dick και Z. Mao, «Cache contention and application performance prediction for multi-core systems,» σε *ISPASS 2010*, 2010.

[16] R. Illikkal, V. Chadha, A. Herdrich, R. Iyer και D. Newell, «PI-RATE: QoS and Performance Management in CMP Architectures,» Hillsboro, Oregon.

[17] M. Qureshi, A. Jaleel, Y. Patt, S. S. Jr και J. Emer, «Adaptive Insertion Policies for High Performance Caching,» σε *ISCA '07*, San Diego, California, 2007.

[18] A. Fedorova, M. Seltzer και M. Smith, «Improving performance isolation on chip multiprocessors via an operating system scheduler,» σε *PACT '07*, 2007.

[19] C. Kozyrakis, D. Lo, L. Cheng, R. Govindaraju και P. Ranganathan, «Heracles: Improving Resource Efficiency at Scale,» σε *ISCA '15*, Portland, 2015.

[20] Intel Corporation, «Platform Shared Resource Monitoring: Cache Monitoring Technology,» σε *Intel® 64 and IA-32 Architectures Software Developer's Manual*, τόμ. 3B, Intel Corporation, 2015, pp. 147 - 153.

[21] Intel Corporation, «Platform Shared Resource Control: Cache Allocation Technology,» σε *Intel® 64 and IA-32 Architectures Software Developer's Manual*, τόμ. 3B, Intel Corporation, 2015, pp. 153 - 164.

[22] Intel Corporation, «Improving Real-Time Performance by Utilizing Cache Allocation Technology,» Intel Corporation, 2015.