ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη Αλγορίθμων Συνδρομολόγησης Εφαρμογών σε Πολυπύρηνες Αρχιτεκτονικές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΔΡΙΑΝΗ ΜΑΠΠΟΥΡΑ

Επιβλέπων : Γεώργιος Γκούμας
Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Μελέτη Αλγορίθμων Συνδρομολόγησης Εφαρμογών σε Πολυπύρηνες Αρχιτεκτονικές

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΔΡΙΑΝΗ ΜΑΠΠΟΥΡΑ

Επιβλέπων : Γεώργιος Γκούμας
Λέκτορας Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 8η Ιουλίου 2016.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Γεώργιος Γκούμας	Νεκτάριος Κοζύρης	Παναγιώτης Τσανάκας
Λέκτορας Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2016

(Υπογραφή)

.....

ΜΑΠΠΟΥΡΑ ΑΝΔΡΙΑΝΗ

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

Περίληψη

Σήμερα οι πολυπύρηνες αρχιτεκτονικές αποτελούν σχεδόν την αποκλειστική επιλογή σχεδιασμού για κάθε σύγχρονο υπολογιστικό σύστημα. Έτσι, συνεχώς αναπτύσσονται πολυνηματικές εφαρμογές που θα μπορούν να εκμεταλλευτούν τις αρχιτεκτονικές αυτές. Οι πυρήνες, όμως, που βρίσκονται πάνω στο ίδιο τσιπ μοιράζονται κομμάτια από την ιεραρχία της μνήμης, όπως είναι ο δίαυλος και οι κρυφές μνήμες, επομένως, η συνεκτέλεση διάφορων νημάτων έχει ως συνέπεια τον ανταγωνισμό στην χρήση των κοινών πόρων. Ως αποτέλεσμα, παρατηρούνται σημαντικές καθυστερήσεις στον χρόνο εκτέλεσης των εφαρμογών και περιορίζεται η επίδοσή τους.

Ο σκοπός της διπλωματικής εργασίας ήταν η μελέτη κλασσικών μεθόδων χρονοδρομολόγησης καθώς και η ανάπτυξη ενός προγράμματος, σε επίπεδο χρήστη, για τη δημιουργία, μελέτη και σύγκριση διαφορετικών πολιτικών που θα μπορούσαν να λάβουν υπόψη τους τον ανταγωνισμό για κοινούς πόρους. Για τον σκοπό αυτό, χρησιμοποιήθηκαν εφαρμογές από το Parsec 3.0 και τα πειράματα εκτελέστηκαν σε δύο διαφορετικές πλατφόρμες πολυπύρηνης αρχιτεκτονικής με 8 πυρήνες.

Λέξεις Κλειδιά:

Χρονοδρομολόγηση, πολυπύρηνες αρχιτεκτονικές, επίδοση, νήματα

Abstract

Nowadays, multi-core architectures are part of almost every computer system. Thus, multithreaded applications are being developed so that they take advantage of these architectures. Multiple cores that are situated on the same physical package share resources of the memory hierarchy, such as memory links and cache, and when different threads are running at the same time, they may suffer from competition of shared resources. As a result, significant delay is caused and applications' performance is restricted.

The scope of this thesis was the study of different scheduling methods that do not take into consideration the competition of shared resources, as well as, the development of a user-level program that implements some other scheduling policies that could take into account this competition factor. Experimental evaluation was performed on two different platforms with 8 cores using applications of Parsec 3.0.

Keywords:

Scheduling, parallel architectures, performance, threads

Ευχαριστίες

Αρχικά, ήθελα να ευχαριστήσω όλο το προσωπικό του εργαστηρίου υπολογιστικών συστημάτων που με βοήθησε κατά τη διάρκεια της διπλωματικής εργασίας σε τεχνικά ζητήματα και τον κ. Β. Αγγέλου που μου παρείχε πολύ σημαντική βοήθεια ώστε να ξεκινήσω το πρακτικό κομμάτι.

Ειδικότερα θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα καθηγητή μου Γ. Γκούμα για τη διαρκή καθοδήγησή του, τις χρήσιμες συμβουλές του, τις γνώσεις που μου μετέδωσε καθώς και την υποστήριξή του.

Τέλος θα ήθελα να ευχαριστήσω την οικογένεια και τους φίλους μου, οι οποίοι βρίσκονταν δίπλα μου και με στήριζαν όλο αυτό το χρονικό διάστημα.

Περιεχόμενα

1

Εισαγωγή

1.1 Πολυπύρηνες Αρχιτεκτονικές

Ο σχεδιασμός και η υλοποίηση των σύγχρονων αρχιτεκτονικών έχει εξελιχθεί σε σχέση με παλαιότερα, όπου οι επεξεργαστές περιλάμβαναν έναν μόνο πυρήνα. Στα μονοπύρηνα συστήματα, βελτιστοποιήσεις στην κατασκευή των πυρήνων, ώστε να επιτυγχάνονται μεγαλύτερες συχνότητες ρολογιού και μικρότερη κατανάλωση ενέργειας, καθώς και στο σχεδιασμό της ιεραρχίας της μνήμης είχαν ως αποτέλεσμα την αύξηση της αποδοτικότητάς τους. Ωστόσο, με το πέρασμα του χρόνου ο ρυθμός βελτίωσης των συστημάτων αυτών μειωνόταν, καθώς προέκυψαν κατασκευαστικά προβλήματα που επέβαλαν καινούργια όρια. Έτσι, προέκυψε η ιδέα για σχεδιασμό συστημάτων με πολλούς πυρήνες, οι οποίοι θα μπορούσαν να δουλεύουν παράλληλα και να μοιράζονται τον φόρτο εργασίας.

Η ύπαρξη δύο ή περισσοτέρων πυρήνων στο ίδιο chip (Chip Multi-processors, CMPs) είχε ως αποτέλεσμα τεράστια αύξηση των δυνατοτήτων των συστημάτων. Ο κάθε πυρήνας έχει μία τουλάχιστον ιδιωτική κρυφή μνήμη. Οι πυρήνες ενός chip μπορούν να λειτουργήσουν ταυτόχρονα και να εκτελέσουν εντολές ανεξάρτητα, ενώ μοιράζονται μια κοινή κρυφή μνήμη, memory links και memory controllers. Τα πολυπύρηνα chips αναπτύχθηκαν ιδιαίτερα τα τελευταία χρόνια και αποτελούν πλέον πραγματικότητα σε servers, προσωπικούς υπολογιστές και υπερυπολογιστές.

Παρ' όλα αυτά, η σωστή αξιοποίηση των συστημάτων αυτών έφερε στην επιφάνεια ορισμένα ζητήματα. Καθώς, οι πυρήνες μοιράζονται αρκετούς πόρους του συστήματος, η λειτουργία τους δεν είναι τελικά ανεξάρτητη και είναι αναμενόμενο ότι θα εμφανίζονται φαινόμενα ανταγωνισμού μεταξύ τους. Ένας ιδανικός χρονοδρομολογητής δεν θα αγνοούσε τα φαινόμενα αυτά. Ωστόσο, οι πλέον αναγνωρισμένοι χρονοδρομολογητές που χρησιμοποιούνται από το λειτουργικό σύστημα σήμερα θεωρούν ότι οι πυρήνες είναι εντελώς ανεξάρτητοι μεταξύ τους, καθώς είχαν αρχικά σχεδιαστεί για συστήματα όπου οι πυρήνες μοιράζονταν μόνο την κύρια μνήμη.

1.2 Χρονοδρομολόγηση Διεργασιών σε Πολυπύρηνα Συστήματα

Η λειτουργία του χρονοδρομολογητή είναι αυτή που αποφασίζει για το ποια νήματα (threads) θα εκτελεστούν στους επεξεργαστές, πότε θα πρέπει να διακόψουν την εκτέλεσή τους καθώς και ποια θα είναι τα επόμενα νήματα που θα εκτελεστούν. Για τον σκοπό αυτό, οι χρονοδρομολογητές χρησιμοποιούν δομές δεδομένων στις οποίες κατατάσσουν τα υποψήφια νήματα με βάση κάποια κριτήρια. Οι δομή αυτή μπορεί να είναι μια απλή FIFO ουρά αλλά μπορούν και να χρησιμοποιηθούν αρκετά πιο περίπλοκες δομές. Υπάρχουν τέσσερα κριτήρια που χρησιμοποιούνται συχνά για την αξιολόγηση ενός αλγορίθμου χρονοδρομολόγησης. Το πρώτο είναι η χρήση της CPU, ιδανικά θα θέλαμε οι CPUs να χρησιμοποιούνται 100%. Σε ένα πραγματικό σύστημα η χρήση της CPU κυμαίνεται μεταξύ 40% και 90%. Το δεύτερο κριτήριο είναι το throughput, δηλαδή ο αριθμός των διεργασιών που ολοκληρώνονται στη μονάδα του χρόνου και θα θέλαμε ο παράγοντας αυτός να μεγιστοποιείται. Σημαντικό ρόλο παίζει, επίσης, ο χρόνος ολοκλήρωσης των διεργασιών, καθώς και ο χρόνος αναμονής τους στη δομή έτοιμων διεργασιών μέχρι να επιλεχθούν για εκτέλεση. Τέλος, σημαντικό ρόλο παίζει και ο χρόνος απόκρισης των προγραμμάτων. Οι τρεις τελευταίοι παράγοντες θέλουμε να ελαχιστοποιούνται. Τέλος, ανάλογα με το είδος ενός συστήματος, θα υπάρχουν και διαφορετικές απαιτήσεις. Για παράδειγμα, στους προσωπικούς υπολογιστές θα θέλαμε ο χρονοδρομολογητής να εξασφαλίζει όσο το δυνατό πιο καλή αποκρισιμότητα, ενώ σε ένα server πιο σημαντικό ρόλο παίζει το throughput.

Σήμερα, η πολιτική χρονοδρομολόγησης που επικρατεί είναι ο Completely Fair Scheduler (CFS) που χρησιμοποιείται στο λειτουργικό σύστημα Linux. Η πολιτική αυτή χειρίζεται ξεχωριστά κάθε νήμα, από όποια διεργασία και αν προέρχεται, προσπαθώντας να είναι δίκαια ως προς τον χρόνο αναμονής του νήματος. Πιο συγκεκριμένα, ο αλγόριθμος χρησιμοποιεί ένα red-black tree στους κόμβους του οποίου κατατάσσει όλα τα προς εκτέλεση tasks, με βάση

τον χρόνο που τους επιτράπηκε να εκτελεστούν σε κάποιον από τους επεξεργαστές. Επίσης, χρησιμοποιείται η έννοια του μέγιστου χρόνου εκτέλεσης που δικαιούται κάθε task να τρέξει σε κάποιον επεξεργαστή. Ο χρόνος αυτός ισούται με τον λόγο του χρόνου αναμονής του ως προς το πλήθος των tasks. Κάθε φορά επιλέγεται προς εκτέλεση το task με τον ελάχιστο χρόνο εκτέλεσης, που θα βρίσκεται στο πιο αριστερό φύλο του δέντρου. Έτσι θα εκτελεστεί για χρόνο έως και ίσο με τον μέγιστο χρόνο εκτέλεσης που υπολογίστηκε. Στη περίπτωση που αυτό τερματίσει, αφαιρείται από τη δομή, αλλιώς, όταν η εκτέλεσή του διακοπεί επανατοποθετείται στο δέντρο με βάση τον καινούργιο χρόνο εκτέλεσής του. Έτσι, ο αλγόριθμος αυτός προσπαθεί να κρατήσει μία ισορροπία ανάμεσα στα tasks με βάση τον χρόνο εκτέλεσής τους. Ο CFS αποτελεί τον πλέον αναγνωρισμένο αλγόριθμο χρονοδρομολόγησης και χρησιμοποιείται ευρέως σε λειτουργικά συστήματα γενικής χρήσης. Θεωρείται η βέλτιστη επιλογή για desktops όπου η αλληλεπίδραση με τον χρήστη παίζει σημαντικό ρόλο. Παρ' όλα αυτά, δεν είναι ιδανικός στις περιπτώσεις που μας ενδιαφέρει το throughput. [1]

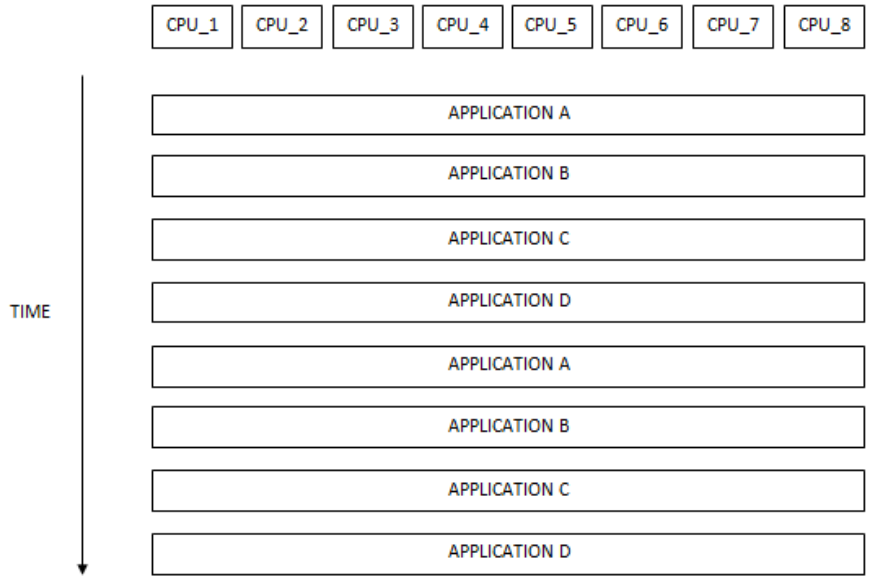
Μία εναλλακτική προσέγγιση είναι αυτή του Gang Scheduler, η οποία έχει ως στόχο να ελαχιστοποιήσει το κόστος συγχρονισμού και επικοινωνίας ανάμεσα στα νήματα μιας διεργασίας. Αυτό επιτυγχάνεται αφήνοντας να εκτελεστούν στους διαθέσιμους πυρήνες νήματα από μία διεργασία ταυτόχρονα για ένα συγκεκριμένο κβάντο χρόνου, ενώ στη συνέχεια θα ακολουθήσουν κυκλικά τα νήματα από τις άλλες διεργασίες, Εικόνα 1.1. Η πολιτική αυτή μπορεί να επιφέρει θετικά αποτελέσματα εφόσον τα νήματα κάθε διεργασίας επικοινωνούν μεταξύ τους, ώστε όταν γίνεται δοσοληψία μηνυμάτων να εξασφαλίζεται ότι τα νήματα που εμπλέκονται δεν κοιμούνται. Εντούτοις, θα θέλαμε τα νήματα που εκτελούνται σε ένα κβάντο χρόνου να μην επικοινωνούν με νήματα που θα εκτελεστούν σε επόμενη χρονική στιγμή ή που εκτελέστηκαν σε προηγούμενη χρονική στιγμή γιατί τότε θα επιβαρύνεται αντί να επωφελείται η επικοινωνία λόγω των πολλαπλών context switches [2]. Επιπλέον, οι διεργασίες τείνουν να δημιουργούν τόσα νήματα όσα και οι διαθέσιμοι πυρήνες της αρχιτεκτονικής. Ως αποτέλεσμα, η Gang πολιτική μπορεί να οδηγήσει σε σπατάλη πόρων, στις περιπτώσεις κατά τις οποίες οι διεργασίες κλιμακώνουν μέχρι ένα αριθμό νημάτων, μικρότερου των διαθέσιμων πυρήνων. Τέλος, στην πολιτική αυτή σημαντικό ρόλο παίζει ο χρόνος που θα επιτραπεί στα νήματα να τρέξουν μέχρι να διακοπεί η εκτέλεσή τους για να συνεχίσουν τα επόμενα. Πολύ μικρά διαστήματα χρόνου μπορεί να προκαλέσουν ανώφελες καθυστερήσεις. Για τον σκοπό αυτό, μετρήθηκε η καθυστέρηση που μπορεί να προκληθεί από συνεχόμενα context switches όταν δύο νήματα τρέχουν εναλλάξ στον ίδιο πυρήνα, σε σχέση με τον χρόνο εναλλαγής τους. Η καθυστέρηση υπολογίζεται με την Εξίσωση 1.1:

$$\text{Delay} = \frac{t1+t2}{\max(t1_{gang}, t2_{gang})} \qquad \text{Εξίσωση 1.1}$$

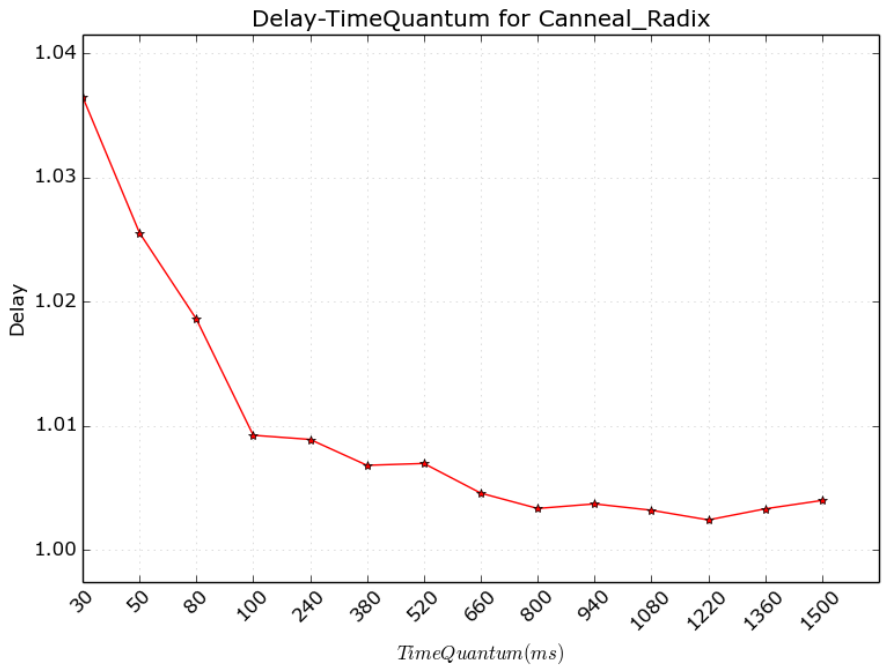
t1, t2: Οι χρόνοι που χρειάζεται κάθε εφαρμογή για να τερματίσει όταν εκτελείται μόνη της

$t1_{gang}$, $t2_{gang}$: Οι χρόνοι που χρειάζεται κάθε εφαρμογή για να τερματίσει όταν οι δύο εφαρμογές εκτελούνται παράλληλα με την Gang πολιτική.

Για την εκτέλεση των εφαρμογών Canneal και Radix από το Parsec 3.0 τα αποτελέσματα παρουσιάζονται στην Εικόνα 1.2.

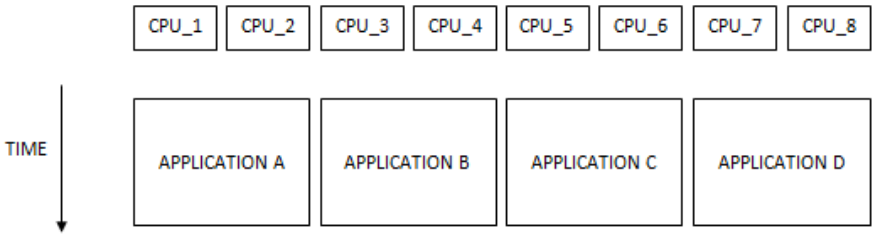


Εικόνα 1.1 – Gang Scheduler



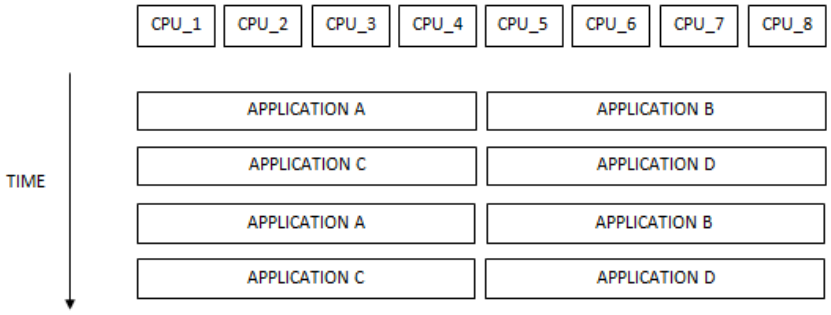
Εικόνα 1.2 – Delay-TimeSlot

Στις περιπτώσεις κατά τις οποίες προκαλείται σπατάλη πόρων με την Gang πολιτική, θα μπορούσε να προτιμηθεί η πολιτική Space-Sharing, η οποία μοιράζει τους διαθέσιμους πυρήνες στις διάφορες διεργασίες, Εικόνα 1.3. Με αυτόν τον τρόπο, εκμεταλλευόμαστε σε κάποιο βαθμό τα πλεονεκτήματα συγχρονισμού της Gang πολιτικής, περιορίζοντας ταυτόχρονα τα προβλήματα που προκύπτουν λόγω περιορισμένης κλιμακωσιμότητας. Το μειονέκτημα αυτής της μεθόδου είναι ότι θα υπάρχουν διεργασίες οι οποίες θα έχουν επιπτώσεις λόγω της παράλληλης συνεκτέλεσής τους με άλλες πιθανόν πιο απαιτητικές σε κοινούς πόρους διεργασίες. Επίσης, όταν υπάρχουν αρκετές διεργασίες σε σχέση με το πλήθος των διαθέσιμων πυρήνων τελικά μπορεί να περιοριστεί η δυνατότητα κλιμακωσιμότητάς τους.

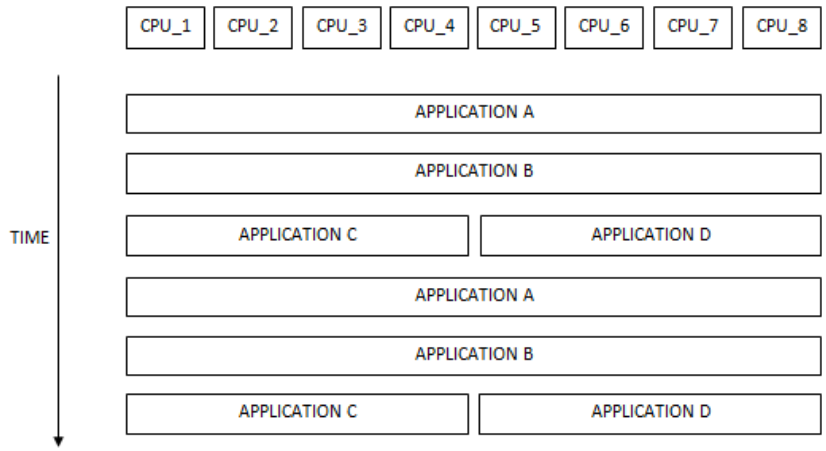


Εικόνα 1.3 – Space-Sharing Scheduler

Ως αποτέλεσμα, για συνεκτελέσεις τεσσάρων διεργασιών σε αρχιτεκτονική οκτώ πυρήνων, εξετάστηκαν ακόμα δύο πολιτικές χρονοδρομολόγησης, που συνδυάζουν τις μεθοδολογίες των Gang και Space-Sharing. Η πρώτη μοιράζει τους μισούς πυρήνες στις πρώτες δύο διεργασίες και τους άλλους μισούς στις άλλες δύο διεργασίες. Έτσι, τρέχουν παράλληλα για συγκεκριμένο κβάντο χρόνου ανά δύο οι διεργασίες, κυκλικά, όπως φαίνεται στην Εικόνα 1.4. Όταν κάποια διεργασία τερματίσει, τότε μία από τις εναπομείναντες τρεις διεργασίες θα εκτελείται και στους οκτώ πυρήνες. Η δεύτερη πολιτική, εναποθέτει στις δύο διεργασίες και τους 8 πυρήνες, ενώ οι άλλες δύο θα τρέχουν παράλληλα σε 4 πυρήνες η κάθε μία. Η χρονοδρομολόγηση τους γίνεται και πάλι κυκλικά όπως πριν, Εικόνα 1.5.



Εικόνα 1.4 – New Method 1



Εικόνα 1.5 – New Method 2

Τέλος, υπάρχουν οι Contention-aware Schedulers. Όπως δηλώνει και το όνομά τους, στόχος τους είναι να εστιάσουν και να περιορίσουν τις κακές επιπτώσεις που μπορεί να προκληθούν στις διεργασίες όταν εκτελούνται παράλληλα. Οι επιπτώσεις αυτές οφείλονται κατά κύριο λόγο στον ανταγωνισμό για κοινούς πόρους. Συνεπώς, οι Contention-aware Schedulers χρειάζονται μια κατηγοριοποίηση των διεργασιών ανάλογα με την χρήση των διαθέσιμων πόρων, όπως είναι το memory link bandwidth, η χρήση των κρυφών μνημών και το LLC miss rate, κάτι που δεν λαμβάνουν υπόψη τους κλασσικοί χρονοδρομολογητές, όπως ο CFS. Ωστόσο, ο υπολογισμός του πόσο "επικίνδυνη" ή "ευαίσθητη" είναι μια εφαρμογή και πώς αυτή μπορεί να επηρεάσει ή να επηρεαστεί από την συνεκτέλεσή της με άλλες διεργασίες είναι το πιο δύσκολο κομμάτι στην υλοποίηση ενός τέτοιου χρονοδρομολογητή.

1.3 Αντικείμενο διπλωματικής

Η παρούσα διπλωματική εργασία εστιάζει στο αντικείμενο της χρονοδρομολόγησης διεργασιών, σε CMPs. Όπως αναφέρθηκε προηγουμένως, οι state-of-the-art χρονοδρομολογητές χειρίζονται τους πυρήνες των συστημάτων ως ανεξάρτητα μονάδες, γεγονός που δεν ισχύει σε CMPs όπου οι πυρήνες μοιράζονται ορισμένους πόρους του συστήματος με διαφορετικούς τρόπους.

Για τον λόγο αυτό, έχουν πολλές φορές προταθεί contention-aware schedulers. Γνωρίζοντας την αρχιτεκτονική ενός συστήματος και επομένως τον τρόπο με τον οποίο οι πυρήνες μοιράζονται τους πόρους του, μπορεί να μελετηθεί η συμπεριφορά των εφαρμογών και να γίνει μία πρόβλεψη των επιπτώσεων που θα έχουν σε πιθανά σενάρια συνεκτελέσεων, ώστε να εφαρμοστούν αυτά που θα τις ελαχιστοποιήσουν.

Έτσι, στη διπλωματική αυτή εργασία μελετούμε διάφορα σενάρια παράλληλων εκτελέσεων σε δύο διαφορετικές πειραματικές πλατφόρμες, πολυπύρηνης αρχιτεκτονικής. Για τους σκοπούς της διπλωματικής χρησιμοποιήθηκε ένα πρόγραμμα χρονοδρομολόγησης σε επίπεδο χρήστη (user level scheduler), που υλοποιεί ορισμένες πολιτικές χρονοδρομολόγησης. Μέσω του προγράμματος αυτού, έγινε σύγκριση και αξιολόγηση πολιτικών χρονοδρομολόγησης σε σενάρια παράλληλης εκτέλεσης διαφορετικών εφαρμογών. Επίσης, προκειμένου να χρησιμοποιηθούν στα πειράματα εφαρμογές με διαφορετικές απαιτήσεις σε πόρους του συστήματος, επομένως και με διαφορετικές συμπεριφορές, μελετήθηκαν παράλληλα προγράμματα και επιλέχθηκαν ορισμένες ενδεικτικές εφαρμογές.

1.3.1 Συνεισφορά

Η συνεισφορά της διπλωματικής συνοψίζεται ως εξής:

1. Μελέτη πολιτικών χρονοδρομολόγησης που εφαρμόζονται στα σύγχρονα συστήματα, καθώς και ορισμένων contention-aware schedulers.
2. Προσαρμογή και επέκταση του προγράμματος με το οποίο εξάχθηκαν οι πειραματικές μετρήσεις.
3. Μελέτη και επιλογή διαφορετικών εφαρμογών.
4. Πειραματικές μετρήσεις σε σενάρια παράλληλης εκτέλεσης δύο εφαρμογών, με τρεις διαφορετικές πολιτικές χρονοδρομολόγησης.
5. Πειραματικές μετρήσεις σε σενάρια παράλληλης εκτέλεσης τεσσάρων εφαρμογών, με πέντε διαφορετικές πολιτικές χρονοδρομολόγησης.
6. Επανάληψη των πιο πάνω πειραμάτων σε δεύτερη πειραματική πλατφόρμα, με διαφορετικό σχεδιασμό και διαφορετική οργάνωση στην ιεραρχία της μνήμης.
7. Ανάλυση και αξιολόγηση των αποτελεσμάτων.
8. Τελικά Συμπεράσματα.

1.4 Οργάνωση κειμένου

Οι τεχνικές λεπτομέρειες, όπως περιγραφή του προγράμματος που χρησιμοποιήθηκε και των πειραματικών πλατφορμών στις οποίες έγιναν οι μετρήσεις, παρουσιάζονται στο δεύτερο κεφάλαιο.

Στο τρίτο κεφάλαιο παρουσιάζονται ορισμένες κατηγοριοποιήσεις εφαρμογών που προτάθηκαν από σχετικές μελέτες για contention-aware χρονοδρομολογητές. Στη συνέχεια, γίνεται μια σύντομη παρουσίαση των εφαρμογών που επιλέχθηκαν για τα πειράματα, καθώς και τα αποτελέσματα και συμπεράσματα που προέκυψαν από τις συνεκτελέσεις των εφαρμογών αυτών, ανά δύο, με διαφορετικές πολιτικές χρονοδρομολόγησης. Τέλος, γίνεται μια σύγκριση και αξιολόγηση των πολιτικών με βάση τα αποτελέσματα αυτά.

Στο τέταρτο κεφάλαιο παρουσιάζονται τα αποτελέσματα από παράλληλες εκτελέσεις τεσσάρων εφαρμογών με τις πέντε διαφορετικές πολιτικές χρονοδρομολόγησης που

υλοποιήθηκαν. Ακολούθως, αξιολογούνται και πάλι οι πολιτικές που εφαρμόστηκαν. Επίσης, παρουσιάζονται κάποιες διαφοροποιήσεις στις καινούργιες πολιτικές με τα σχετικά συμπεράσματα.

Το πέμπτο κεφάλαιο περιέχει μία σύνοψη και τα τελικά συμπεράσματα, από τη διπλωματική εργασία, καθώς και ιδέες για μελλοντικές επεκτάσεις.

2

Τεχνικές λεπτομέρειες

Για την υλοποίηση και αξιολόγηση των πολιτικών χρονοδρομολόγησης που αναλύθηκαν, χρησιμοποιήθηκε ένα προγραμματιστικό εργαλείο για το οποίο ακολουθεί μια περιγραφή. Στη συνέχεια, παρουσιάζονται οι δύο διαφορετικές πειραματικές πλατφόρμες στις οποίες πραγματοποιήθηκαν οι μετρήσεις.

2.1 Λεπτομέρειες υλοποίησης προγραμματιστικού εργαλείου

Για την υλοποίηση των διαφόρων μεθόδων χρονοδρομολόγησης χρησιμοποιήθηκε ένα έτοιμο εργαλείο του εργαστηρίου, που αρχικά ήταν διαμορφωμένο ώστε να υλοποιεί δύο είδη χρονοδρομολόγησης (Gang και Space-Sharing) και επεκτάθηκε σαν μέρος αυτής της διπλωματικής εργασίας.

Το εργαλείο αυτό αποτελείται από δύο κύρια μέρη, την βιβλιοθήκη rt-interposer.so που είχε ως στόχο την διαμόρφωση κάποιων συναρτήσεων και το πρόγραμμα usr-sched.c το οποίο υλοποιεί τον αλγόριθμο χρονοδρομολόγησης. Κάθε εφαρμογή που θέλουμε να συμπεριληφθεί στον χρονοδρομολογητή αυτό, κάνει preload την πιο πάνω βιβλιοθήκη. Το πρόγραμμα δημιουργεί μια καινούρια POSIX message queue και μέσω αυτής μπορεί να επικοινωνεί με τις διεργασίες.

Πιο συγκεκριμένα, η βιβλιοθήκη rt-interposer.so περιέχει ένα constructor, ο οποίος αναλαμβάνει να ανοίξει την ουρά με οποία θα επικοινωνεί η διεργασία με το usr-sched. Ο

constructor μέσω της ουράς ενημερώνει το πρόγραμμα για τη εκκίνηση της διεργασίας, δημιουργεί μια λίστα που θα περιέχει τα threads της με τα αντίστοιχα στοιχεία τους, όπως tid, pid, state, cpuset, και ορίζει τους handlers για τα σήματα τα οποία θα χρησιμοποιεί για να διακόπτει και να συνεχίζει την εκτέλεση της διεργασίας. Επίσης, ορίζει στη διεργασία τους διαθέσιμους πυρήνες με βάση την υπάρχουσα αρχιτεκτονική και αρχικοποιεί το Performance Application Programming Interface (PAPI), με το οποίο θα μπορεί, ανά τακτά χρονικά διαστήματα, να συλλέγει μετρήσεις από performance counters, ώστε να ανακτά πληροφορίες σχετικές με τα cache misses, τις εντολές που ολοκληρώνονται, τους κύκλους ρολογιού κ.ά.. Η συλλογή αυτή θα γίνεται μέσω ενός καινούριου thread, το οποίο ελέγχει ταυτόχρονα την message queue και όταν υπάρχουν μηνύματα από το usr-sched, τα χειρίζεται κατάλληλα. Τα μηνύματα αυτά μπορεί να αφορούν τη διακοπή ή την επανεκκίνηση της διεργασίας, επομένως και όλων των νημάτων που έχει δημιουργήσει καθώς και τον επαναδιορισμό του CPU set στο οποίο μπορεί να εκτελούνται τα νήματα της διεργασίας. Αντίστοιχα με τον constructor, υπάρχει ο destructor ο οποίος ενημερώνει το πρόγραμμα για τον τερματισμό της διεργασίας. Τέλος, η βιβλιοθήκη rt-interposer.so, προσαρμόζει συναρτήσεις όπως είναι οι pthread_join, pthread_exit, pthread_mutex_lock και pthread_create, ώστε να διαμορφώνει κατάλληλα τα στοιχεία των threads και να ενημερώνει, εφόσον χρειάζεται, τον usr-sched για την κλήση αυτών των συναρτήσεων.

Από την πλευρά του usr-sched, ο ρόλος του είναι να προσθέτει τις διεργασίες που εκκινούν σε μια δομή δεδομένων, μία διπλά συνδεδεμένη λίστα. Ανάλογα με την πολιτική χρονοδρομολόγησης, η οποία επιλέγεται κατά την μεταγλώττιση του προγράμματος, υπολογίζει πώς θα διαμοιράσει τους διαθέσιμους πυρήνες στις διεργασίας και στη συνέχεια τους στέλνει μέσω της message queue τα αντίστοιχα μηνύματα ώστε αυτές να επαναπροσδιορίσουν το δικό τους CPU set. Στην περίπτωση που έχει επιλεχθεί χρονοδρομολογητής ο οποίος απαιτεί εκτέλεση διαφορετικών διεργασιών ανά κβάντο χρόνου, όπως είναι ο Gang Scheduler, το πρόγραμμα αναλαμβάνει να ενημερώνει κάθε φορά που ολοκληρώνεται το χρονικό αυτό διάστημα, και πάλι μέσω της ουράς, τις διεργασίες που πρέπει να διακόψουν την εκτέλεσή τους ή να συνεχίσουν την εκτέλεσή τους που είχε προηγουμένως διακοπεί.

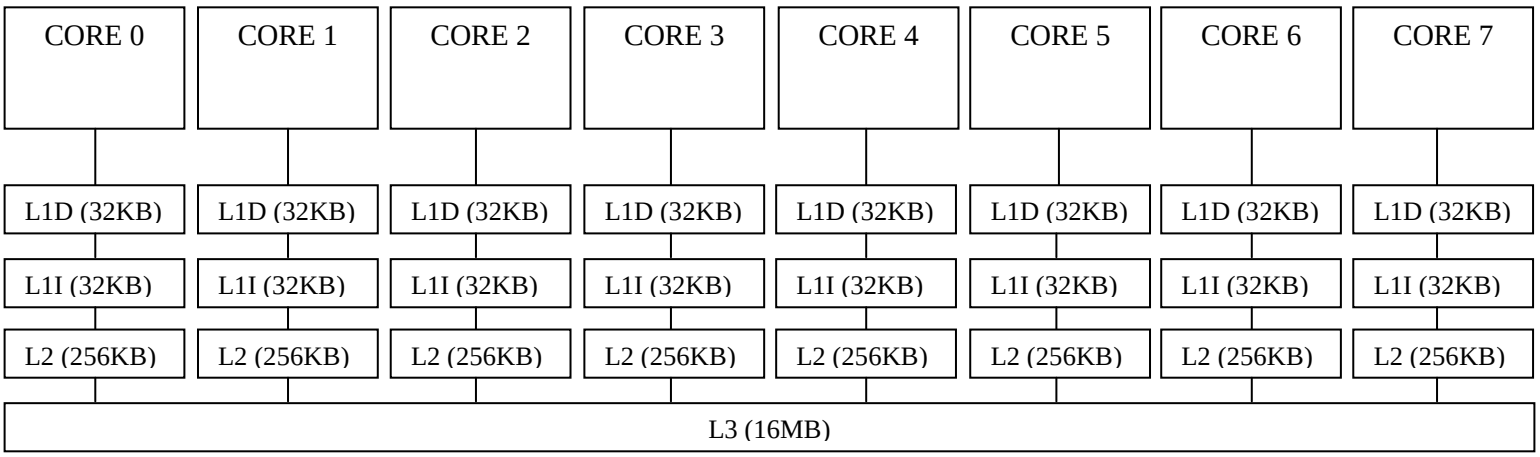
2.2 Πειραματικές Πλατφόρμες

Τα πειράματα εκτελέστηκαν σε δύο πειραματικές πλατφόρμες οκτώ πυρήνων οι οποίες παρουσιάζονται αναλυτικά στη συνέχεια.

2.2.1 Πειραματική Πλατφόρμα Sandy Bridge

Η αρχιτεκτονική της πρώτης πλατφόρμας, φαίνεται στην Εικόνα 2.1 και περιγράφεται από τον Πίνακα 2.1.

Πρόκειται για πολυπύρηνο επεξεργαστή Intel Xeon CPU E5-4620 με 8 πυρήνες.



Εικόνα 2.1

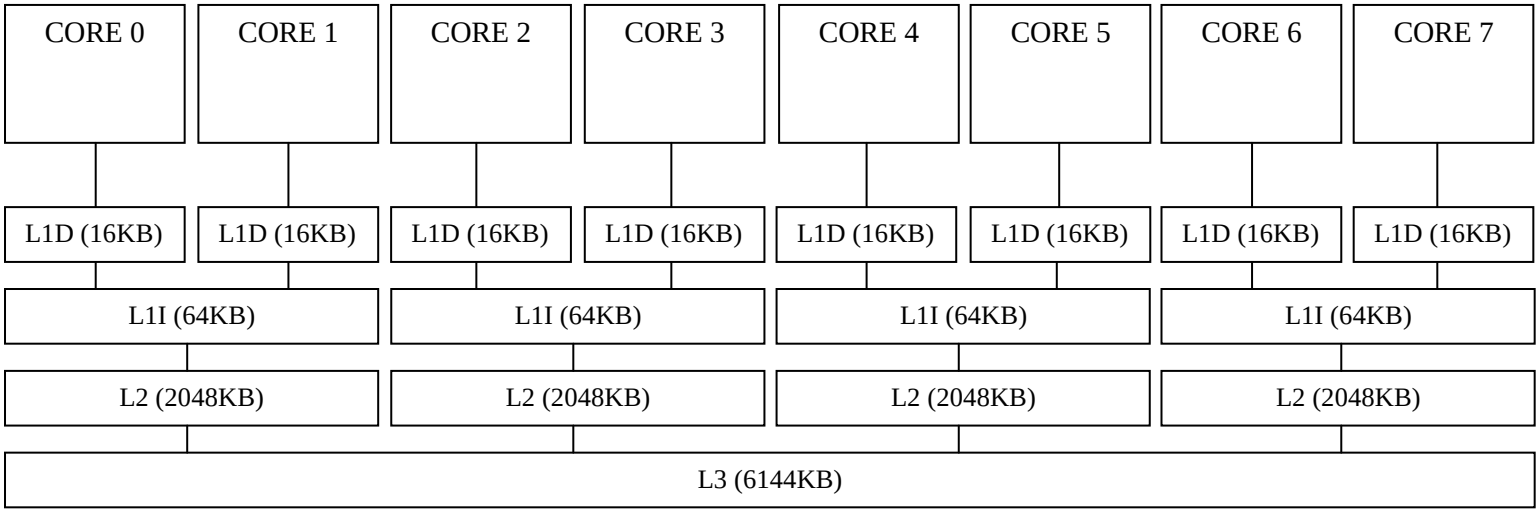
L1	L1D Κρυφή ιδιωτική μνήμη δεδομένων, 32KB, 8-way, μέγεθος block 64 bytes L1I Κρυφή ιδιωτική μνήμη εντολών, 32KB, 8-way, μέγεθος block 64 bytes
L2	Κρυφή ιδιωτική μνήμη, 256KB, 8-way, μέγεθος block 64 bytes
L3	Κρυφή κοινή μνήμη, 16MB, 16-way, μέγεθος block 64 bytes
Memory	64GB

Πίνακας 2.1

2.2.2 Πειραματική Πλατφόρμα Opteron

Η αρχιτεκτονική της δεύτερης πλατφόρμας, φαίνεται στην Εικόνα 2.2 και περιγράφεται από τον Πίνακα 2.2.

Πρόκειται για πολυπύρηνο επεξεργαστή AMD Opteron(tm) Processor 6378 με 8 πυρήνες.



Εικόνα 2.2

L1	L1D Κρυφή ιδιωτική μνήμη δεδομένων, 16KB, 2-way, μέγεθος block 64 bytes L1I Κρυφή ιδιωτική μνήμη εντολών, 64KB, 2-way, μέγεθος block 64 bytes
L2	Κρυφή ιδιωτική μνήμη, 2048KB, 16-way, μέγεθος block 64 bytes
L3	Κρυφή κοινή μνήμη, 6144KB, 64-way, μέγεθος block 64 bytes
Memory	63GB

Πίνακας 2.2

3

Πολυνηματικές Εφαρμογές

3.1 Κατηγοριοποίηση Εφαρμογών από Σχετικές Μελέτες

Όπως αναφέρθηκε πρωτύτερα, οι contention- aware schedulers, βασίζουν τη λειτουργία τους σε κάποιο είδος κατηγοριοποίησης εφαρμογών. Αν και έχουν προταθεί κατηγοριοποιήσεις με βάση τα LLC misses ή το memory link bandwidth, το να λαμβάνουμε υπόψη μας τη χρήση ενός μόνο κοινού πόρου δεν μας δίνει μια ξεκάθαρη εικόνα για το πως θα επηρεαστούν οι εφαρμογές κατά την ταυτόχρονη εκτέλεσή τους.

Αρχικά, υπήρξαν πολλές προσεγγίσεις που αφορούσαν τη χρήση την κοινής κρυφής μνήμης. Μια κατηγοριοποίηση που προτάθηκε το 2008 από τους Xie και Loh [3] κατατάσσει τις εφαρμογές σε μία από τις εξής τέσσερις ομάδες ζώων: χελώνες, πρόβατα, κουνέλια και tasmanian devils. Στην ομάδα των χελωνών ανήκαν οι εφαρμογές που δεν χρησιμοποιούσαν ιδιαίτερα την κοινή cache και γενικότερα παρουσιάζουν μια αργή κίνηση δεδομένων. Στην ομάδα των προβάτων και των κουνελιών ανήκαν οι εφαρμογές που χρησιμοποιούν την κοινή cache, με την διαφορά ότι η δεύτερη ομάδα είναι πιο ευαίσθητη στον αριθμό των ways που της ανατίθενται. Έτσι, εφαρμογές στην ομάδα των προβάτων διατηρούν χαμηλό miss rate στην LLC και συνεπώς έχουν πιο προβλεπόμενη συμπεριφορά χωρίς να επηρεάζονται εύκολα από άλλες εφαρμογές. Αντίθετα, εφαρμογές από την ομάδα των κουνελιών σε περίπτωση που δεν έχουν αρκετό διαθέσιμο χώρο στην κοινή cache, αυξάνουν το miss rate στην LLC και μειώνουν την επίδοσή τους, χαρακτηρίζονται δηλαδή από μια γρήγορη κίνηση που ευνοείται όταν έχουν μεγαλύτερο χώρο στη διάθεσή τους. Τέλος στην ομάδα των tasmanian devils

ανήκουν εφαρμογές οι οποίες έχουν πάντα ψηλό miss rate στην κοινή cache και επιβαρύνουν με την παρουσία τους τις υπόλοιπες διεργασίες.

Μία άλλη κατηγοριοποίηση (Lin et al. 2008 [4]) διέκρινε τις εφαρμογές σε 4 ομάδες χρωμάτων, ανάλογα με την μείωση της επίδοσης που έχουν όταν έχουν στη διάθεσή τους 1MB cache και όταν έχουν 4MB. Οι πιο ευαίσθητες εφαρμογές που θα υποστούν υποβάθμιση μεγαλύτερη του 20% θα ανήκουν στην κόκκινη ομάδα, αυτές που μειώνουν την επίδοση τους κατά 5%-20% μπαίνουν στην κίτρινη ομάδα, ενώ αυτές που το ποσοστό μείωσής τους δεν ξεπερνά το 5% θα ανήκουν είτε στην πράσινη είτε στην μαύρη ομάδα ανάλογα με τον λόγο misses προς κύκλους ρολογιού.

Ένα εναλλακτικό σχήμα ταξινόμησης με βάση και πάλι τη χρήση της LLC (Jaleel et al. 2012 [5]) κατατάσσει τις εφαρμογές στις τέσσερις εξής διαφορετικές ομάδες. Εφαρμογές που χρησιμοποιούν σύνολα δεδομένων που χωράνε στις ιδιωτικές caches και έτσι δεν επωφελούνται από την κοινή μνήμη, ανήκουν στην κατηγορία Core Cache Fitting, ενώ εφαρμογές με σύνολα δεδομένων μεγαλύτερα από την LLC, κατατάσσονται στην κατηγορία των LLC Thrashing. Η πρώτη κατηγορία δεν επηρεάζει εφαρμογές, που συνεκτελούνται και χρησιμοποιούν την LLC, σε αντίθεση με τη δεύτερη κατηγορία. Η τρίτη ομάδα, LLC Fitting, αφορά τις εφαρμογές που χρειάζονται σχεδόν ολόκληρη την LLC, ενώ η τέταρτη κατηγορία, LLC Friendly, συμπεριλαμβάνει εφαρμογές που χρησιμοποιούν την LLC αλλά χωρίς η επίδοσή τους να επηρεάζεται σημαντικά από τον χώρο της κοινής cache που της διατίθεται.

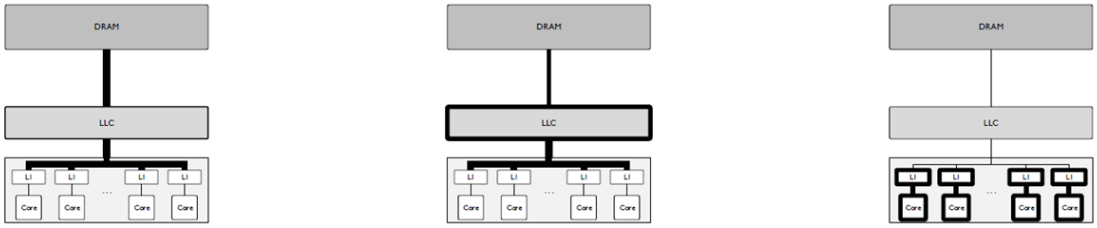
Αξίζει να σημειωθεί ότι οι πιο πάνω ταξινομήσεις έχουν δύο βασικά μειονεκτήματα. Το πρώτο είναι το γεγονός ότι προσπαθούν να ερμηνεύσουν τις πιθανές συνέπειες παράλληλων εκτελέσεων, λαμβάνοντας υπόψη τους μόνο την LLC, ενώ μπορεί κατά την ταυτόχρονη εκτέλεση εφαρμογών, οι επιπτώσεις που προκύπτουν να αφορούν κάποιο άλλο κοινό πόρο του συστήματος. Το δεύτερο μειονέκτημα αφορά το ότι πρόκειται για κατηγοριοποιήσεις που χρειάζονται πληροφορίες οι οποίες προκύπτουν από στατιστική ανάλυση ή απαιτούν επιπλέον hardware. Έτσι, δεν είναι μέθοδοι που μπορούν εύκολα να υλοποιηθούν και να εφαρμοστούν σε κάποιο σύστημα.

Μία τέταρτη προσέγγιση (Blagodurov et al. 2010 [6]), με βάση τις τιμές του LLC miss rate προσπαθεί να προσδιορίσει πόσο επιθετικά χρησιμοποιούν οι εφαρμογές την κοινή cache και έτσι να υπολογίσει πόσο "ευαίσθητη" ή "επικίνδυνη" μπορεί να είναι κάθε εφαρμογή όταν εκτελείται παράλληλα με άλλες. Αντίστοιχα, μία άλλη μελέτη (Merkel et al. 2010 [7]) είχε ως στόχο να ερμηνεύσει την ευαισθησία που παρουσιάζουν οι εφαρμογές, σχετικά με τη μνήμη, με βάση τη χρήση του κοινού memory bus. Ένα διαφορετικό σχήμα (Tang et al. 2011 [8]) προσπαθεί να σκιαγραφήσει καλύτερα τον χαρακτήρα των εφαρμογών, χρησιμοποιώντας πληροφορίες σχετικές με την χρήση της LLC, του link bandwidth και του perfecting traffic, χωρίς δηλαδή να περιορίζεται στη χρήση ενός μόνο κοινού πόρου.

Το 2010, οι Bhadauria και McKee προσπάθησαν να ισορροπήσουν την ζήτηση σε πόρους των εφαρμογών που τρέχουν ταυτόχρονα, λαμβάνοντας υπόψη τους το cache miss rate ή τη χρήση του memory bus, χωρίς όμως η αποφυγή των φαινομένων ανταγωνισμού να είναι ο κεντρικός τους στόχος. Οι χρονοδρομολογητές τους προσπαθούν να αποφασίσουν για τον αριθμό τον νημάτων σε κάθε εφαρμογή καθώς και για την συνύπαρξη εφαρμογών στο ίδιο time slot.

Αργότερα, με το σχήμα χρονοδρομολόγησης ADAPT (Kumar Pusukuri et al. 2013 [9]), ενώ οι εφαρμογές χαρακτηρίζονταν πόσο ευαίσθητες είναι στην χρήση της κοινής μνήμης από το LLC miss rate, λήφθηκαν επίσης υπόψη οι παράγοντες των lock contention και thread latency. Έτσι, ενώ παρακολουθείται η χρήση πόρων από τις εφαρμογές ώστε αυτές να χρονοδρομολογηθούν με τρόπο που θα ελαχιστοποιεί φαινόμενα interference, ταυτόχρονα, αποφασίζεται το σύνολο πόρων που θα έχουν οι εφαρμογές στην διάθεσή τους.

Τέλος, μια άλλη μελέτη, The Link and Cache contention aware scheduler (LCA), που είχε ως στόχο να συνοψίσει κάποιες από τις πιο πάνω ιδέες προτάθηκε το 2017 [11], προτείνοντας μια ταξινόμηση σε τέσσερις ομάδες εφαρμογών σε μία προσπάθεια να αποφύγει την συνδρομολόγηση εφαρμογών που παρουσιάζουν πολύ ανταγωνιστικό χαρακτήρα στην χρήση κοινών πόρων. Στη συνέχεια, προτάθηκε ένα πιο απλοποιημένο σχήμα κατηγοριοποίησης εφαρμογών, με το όνομα SCN, σε τρεις ομάδες, [20]. Η πρώτη ομάδα, Class S (Streaming), αφορά τις εφαρμογές με σταθερή ροή δεδομένων σε όλα τα memory links της ιεραρχίας της μνήμης. Έτσι, στην ομάδα αυτή θα ανήκουν εφαρμογές που δεν επαναχρησιμοποιούν δεδομένα από τις caches. Αυτό μπορεί να συμβαίνει είτε επειδή απλώς δεν προσπελαύνει συχνά τα ίδια δεδομένα, είτε επειδή το μέγεθος της cache δεν είναι αρκετό για να χωρέσει όλα τα δεδομένα που χρειάζεται η εφαρμογή και συνεπώς μέχρι να χρειαστεί ξανά δεδομένα που ήρθαν προηγουμένως στην cache, θα έχουν ήδη φύγει από αυτήν. Αυτό σημαίνει ότι οι εφαρμογές που ανήκουν στη ομάδα S, ουσιαστικά δεν μπορούν να επωφεληθούν από οποιοδήποτε level της cache και απλώς τελικά γεμίζουν άσκοπα τη cache. Στην δεύτερη ομάδα, Class C (Cache Sensitive), ανήκουν οι εφαρμογές που επωφελούνται από τη χρήση της LLC, όπως αυτές που χρησιμοποιούν μικρά σύνολα δεδομένων που επαναχρησιμοποιούνται συχνά, ή εφαρμογές που ο κώδικας τους μπορεί να είναι προσαρμοσμένος ώστε να βελτιστοποιείται η χρήση της LLC. Εφαρμογές με πολλούς υπολογισμούς, μικρά σύνολα δεδομένων ή με βελτιστοποιήσεις στην επαναχρησιμοποίηση δεδομένων ανήκουν στην τρίτη κατηγορία, Class N (No Contention). Οι εφαρμογές αυτές περιορίζουν την κίνηση και επαναχρησιμοποίηση δεδομένων στους ιδιωτικούς πόρους του συστήματος. Η Εικόνα 3.1.1 δείχνει πώς ερμηνεύεται για κάθε κατηγορία η κίνηση και χρήση δεδομένων.



Class S Class C Class N

Εικόνα 3.1.1

Η πιο πάνω κατηγοριοποίηση μπορεί να γίνει κατά τη διάρκεια της εκτέλεσης των εφαρμογών, συλλέγοντας πληροφορίες που δεν απαιτούν επιπλέον υλικό. Πιο συγκεκριμένα, αυτό που χρειάζεται είναι η παρακολούθηση της κίνησης δεδομένων από τη κύρια μνήμη προς τους πυρήνες του συστήματος. Συγκρίνοντας, τις ροές που προκύπτουν από ένα επίπεδο μνήμης ως προς το επόμενο επίπεδο, μπορεί να εκτιμηθεί εάν γίνεται επαναχρησιμοποίηση δεδομένων και αν ναι σε ποιο επίπεδο.

Αυτό που τελικά προκύπτει είναι ότι έχοντας κατηγοριοποιήσει τις εφαρμογές με ένα τέτοιο σχήμα μπορεί να αποφευχθεί η συνύπαρξη εφαρμογών από τις κατηγορίες S και C. Τα φαινόμενα ανταγωνισμού θα μεγιστοποιούνταν αν επιλέγαμε να εκτελεστούν ταυτόχρονα S και C εφαρμογές. Αντίθετα, αυτό που προτείνεται σε σενάρια όπου συνυπάρχουν δύο εφαρμογές για συγκεκριμένο χρονικό διάστημα και μοιράζονται τους κοινούς πόρους, είναι να εξασφαλίσουμε ότι όσο το δυνατό περισσότερες class-N εφαρμογές, οι οποίες δεν προκαλούν και ούτε επηρεάζονται από φαινόμενα ανταγωνισμού, θα συνυπάρχουν με class-S εφαρμογές. Μετά, οι εναπομείναντες class-N εφαρμογές είναι προτιμότερο να συνεκτελεστούν με class-C εφαρμογές. Όταν οι class-N εφαρμογές δεν είναι αρκετές, είναι προτιμότερο εφαρμογές από την ομάδα C να συνεκτελεστούν με άλλες εφαρμογές από την ομάδα C και οι class-S εφαρμογές να συνεκτελεστούν με άλλες, επίσης, class-S εφαρμογές.

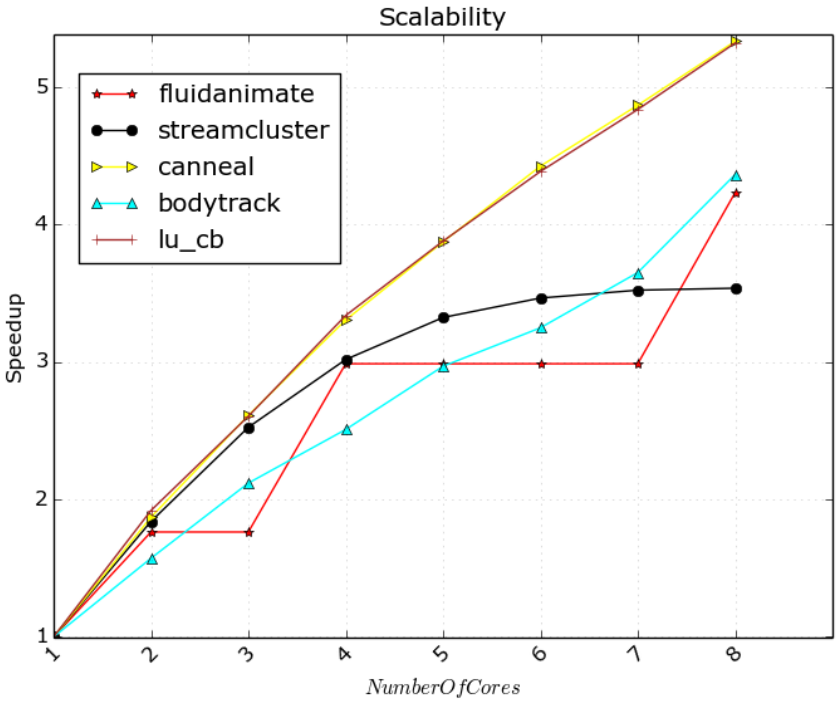
3.2 Επιλογή Πολυνηματικών Εφαρμογών

Για τα πειράματα που υλοποιήθηκαν θέλαμε να συμπεριλάβουμε εφαρμογές με διαφορετικές απαιτήσεις σε πόρους, όπως είναι κοινές και ιδιωτικές caches και τα memory links αλλά και διαφορετικές συμπεριφορές σε παράγοντες όπως είναι το scalability. Ο λόγος είναι για να μπορέσουμε στη συνέχεια να μελετήσουμε κατά πόσο εφαρμογές με διαφορετικές συμπεριφορές μπορούν να επηρεάσουν ή να επηρεαστούν κατά τις εκτελέσεις με διάφορες πολιτικές χρονοδρομολόγησης και με διαφορετικές εφαρμογές να τρέχουν παράλληλα.

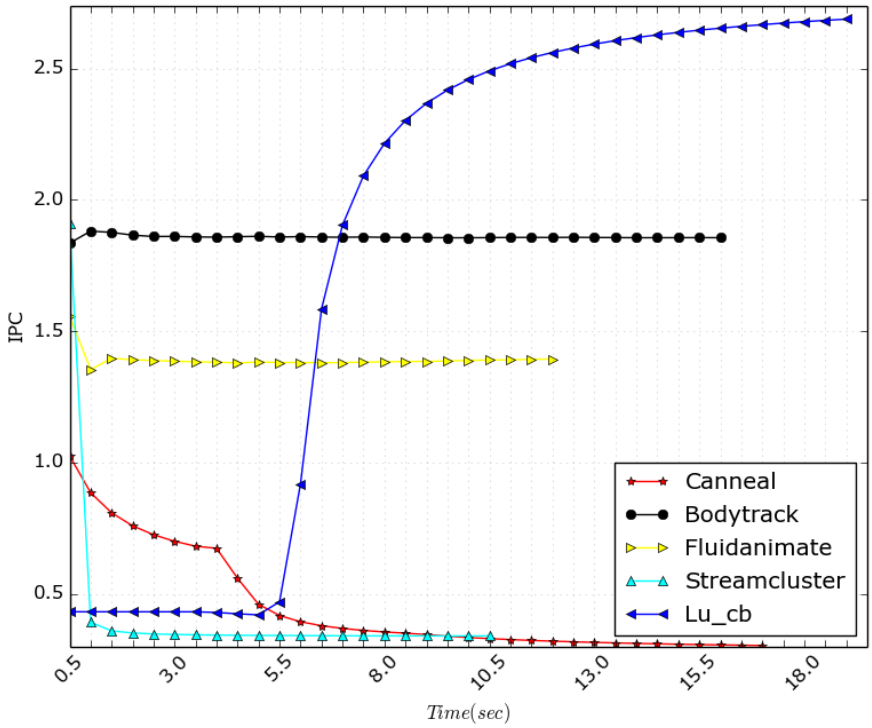
Μετά από κατάλληλες μετρήσεις σε scalability, IPC και cache misses, επιλέχθηκαν τελικά πέντε διαφορετικές εφαρμογές από το Parsec 3.0 [12], [13]. Η κάθε εφαρμογή χαρακτηρίζεται από συγκεκριμένη συμπεριφορά με αποτέλεσμα να παρουσιάζει και διαφορετικά αποτελέσματα ανάλογα με το είδος χρονοδρομολόγησης.

Ακολουθεί μια σύντομη περιγραφή των εφαρμογών που εξετάστηκαν [14]-[19]. Στη συνέχεια, παρουσιάζονται τα αποτελέσματα καθώς και τα σχετικά συμπεράσματα από τις συνεκτελέσεις αυτών των εφαρμογών ανά δύο. Για όλες τις εφαρμογές, ο παραλληλισμός υλοποιήθηκε με Posix Threads.

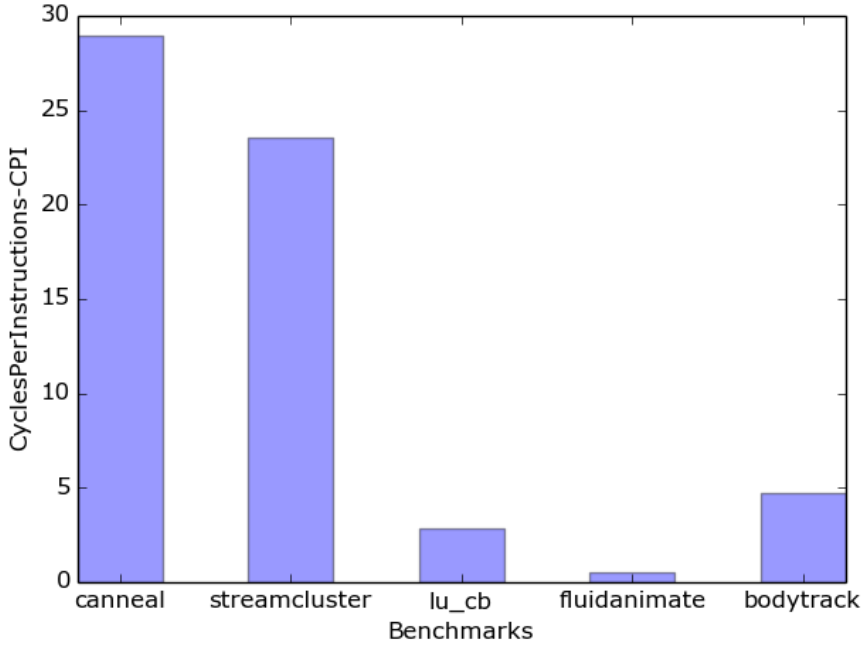
Στην Εικόνα 3.2.1 παρουσιάζεται η κλιμακωσιμότητα των εφαρμογών αυτών ως συνάρτηση του πλήθους νημάτων. Σημειώνεται ότι η εφαρμογή Fluidanimate τρέχει μόνο για πλήθος νημάτων που μπορεί να εκφραστεί ως δύναμη του δύο και έτσι οι μετρήσεις πάρθηκαν μόνο για τιμές 1, 2, 4 και 8. Επίσης, στην Εικόνα 3.2.2 φαίνεται ο λόγος των συνολικών κύκλων ρολογιού ως προς το συνολικό πλήθος εντολών των εφαρμογών (average CPI), ενώ στην Εικόνα 3.2.3 παρουσιάζεται ο λόγος εντολών ως προς το πλήθος κύκλων ρολογιού (IPC) ως συνάρτηση του χρόνου. Τέλος στις Εικόνες 3.2.4 και 3.2.5 παρουσιάζονται οι εφαρμογές με ψηλές και χαμηλές τιμές miss rate, αντίστοιχα, στην Last Level Cache.



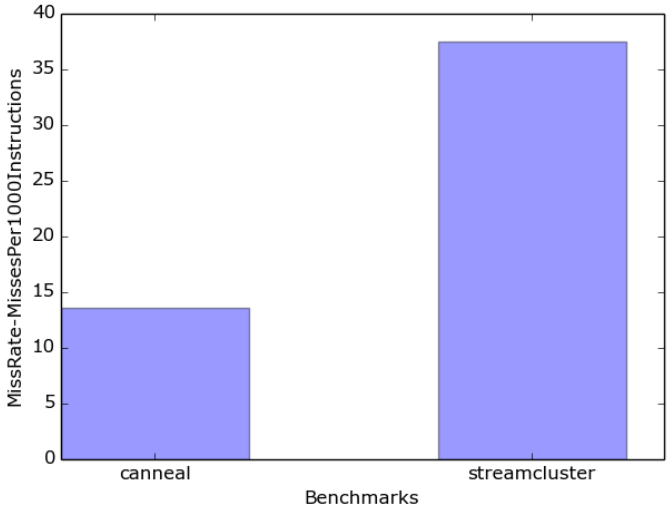
Εικόνα 3.2.1



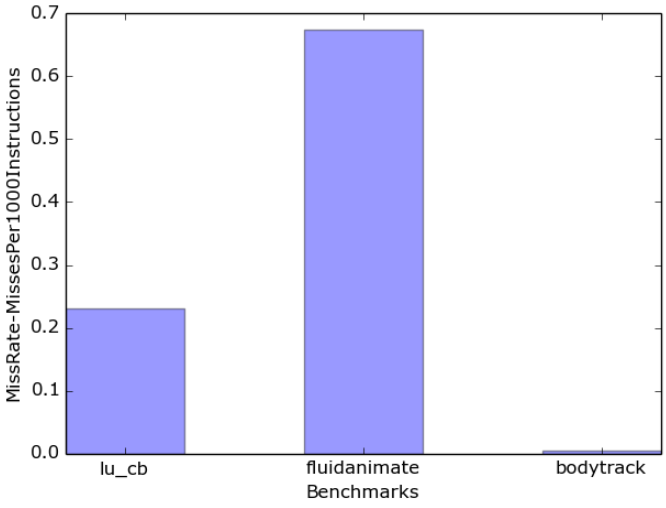
Εικόνα 3.2.2



Εικόνα 3.2.3



Εικόνα 3.2.4



Εικόνα 3.2.5

3.3 Περιγραφή Εφαρμογών

3.3.1 Canneal

Πρόκειται για μια εφαρμογή που ελαχιστοποιεί το κόστος δρομολόγησης της σχεδίασης ενός chip, χρησιμοποιώντας cache-aware simulated annealing. Ο αλγόριθμος που χρησιμοποιεί προσπαθεί να αυξήσει την επαναχρησιμοποίηση των δεδομένων, μειώνοντας τα capacity misses στην cache. Η παραλληλοποίηση της εφαρμογής, είναι fine-grained, δηλαδή, οι υπολογισμοί χωρίζονται σε πολλά μικρά μέρη και υπάρχει σημαντική επικοινωνία ανάμεσα στα νήματα της εφαρμογής. Συγκεκριμένα, όπως αναφέρεται σε ορισμένες από τις πηγές, [14] [16], χρησιμοποιεί "aggressive synchronization στρατηγική". Επίσης, χρησιμοποιεί τεράστια σύνολα εργασίας με lock-free τεχνικές συγχρονισμού και προκαλεί μεγάλες τιμές traffic (bytes/instruction), σημαντικό μέρος του οποίου οφείλεται σε writebacks. Αξίζει, ακόμη, να σημειωθεί ότι ενώ το μεγαλύτερο μέρος του συνόλου εργασίας χρησιμοποιείται από όλα τα νήματα της εφαρμογής, η κρυφή μνήμη μπορεί να χωρέσει ένα πολύ μικρό μέρος αυτού. Έτσι, η πιθανότητα να επαναχρησιμοποιηθεί ένα cache line από ένα δεύτερο νήμα, πριν τα δεδομένα του να αντικατασταθούν από άλλα, είναι τελικά ελάχιστη. Γενικότερα, είναι εφαρμογή που παρουσιάζει ευαισθησία σε παράγοντες όπως το memory bandwidth και το memory latency.

3.3.2 LU

Η εφαρμογή αυτή χρησιμοποιήθηκε από το πακέτο SPLASH-2 (lu_cb) και στόχος της είναι η παραγοντοποίηση ενός πίνακα σε γινόμενο ενός κάτω και ενός άνω τριγωνικού πίνακα. Έχει καλή κλιμακωσιμότητα ενώ η επικοινωνία είναι σημαντική. Συγκεκριμένα, ο μέσος χρόνος συγχρονισμού μπορεί να ξεπερνά το 25% του συνολικού χρόνου εκτέλεσης. Επίσης, η εφαρμογή αυτή κάνει καλή αξιοποίηση της κρυφής μνήμης καθώς μπορεί να χωρέσει το σύνολο εργασίας που της είναι σημαντικό.

3.3.3 Fluidanimate

Η εφαρμογή αυτή έχει ως στόχο την προσομοίωση της κίνησης του υγρού και χρησιμοποιείται ευρέως σε ηλεκτρονικά παιχνίδια. Ο αλγόριθμος που χρησιμοποιείται απαιτεί πολύ μεγάλα σύνολα εργασίας ενώ η επικοινωνία ανάμεσα στα νήματα είναι σημαντική και αυξάνεται χρησιμοποιώντας περισσότερα νήματα. Επίσης, σημειώνεται ότι τα νήματά της μοιράζονται μικρή ποσότητα δεδομένων, ενώ μπορεί να αξιοποιήσει τις ιδιωτικές

caches. Γενικότερα είναι εφαρμογή που παρουσιάζει μέτρια κλιμακωσιμότητα, καθώς περιορίζεται από το αυξανόμενο overhead παραλληλοποίησης.

3.3.4 Bodytrack

Σκοπός της εφαρμογής αυτής είναι η ανίχνευση και παρακολούθηση ενός ανθρώπινου σώματος, χρησιμοποιώντας μικρά σε μέγεθος σύνολα εργασίας και μερική επικοινωνία, ενώ ο παραλληλισμός της χαρακτηρίζεται ως medium-granular. Επιπλέον, παρουσιάζει χαμηλές τιμές miss rate, εφόσον δεν έχει ανάγκη από κρυφή μνήμη μεγάλης χωρητικότητας. Τα σύνολα εργασίας που χρησιμοποιούνται μπορούν να χωρέσουν στις κρυφές μνήμες που διατίθενται από τα τελευταία CMPs, ενώ μπορεί επίσης να αξιοποιήσει σε μεγάλο βαθμό και τις private caches. Η εφαρμογή αυτή δεν έχει καλή κλιμακωσιμότητα, αφού περιλαμβάνει ορισμένα σειριακά κομμάτια στον κώδικα, ενώ ακόμη μπορεί να επηρεαστεί από μειωμένες δυνατότητες σε bandwidth. Συγκεκριμένα, η μεταφορά δεδομένων από τη μνήμη πραγματοποιείται από ένα μικρό μέρος του σειριακού τμήματος, αλλά σε αρχιτεκτονικές με περιορισμένο memory bandwidth η διαδικασία αυτή μπορεί να αποτελέσει σημαντικό παράγοντα περιορισμένης κλιμακωσιμότητας. Επίσης, υπάρχει σημαντική απόκλιση ανάμεσα στο πλήθος των εντολών που εκτελούνται από τα διάφορα νήματα. Αξίζει, τέλος, να σημειωθεί ότι τα νήματα που δημιουργούνται προσπελαύνουν αρκετά κοινά δεδομένα.

3.3.5 Streamcluster

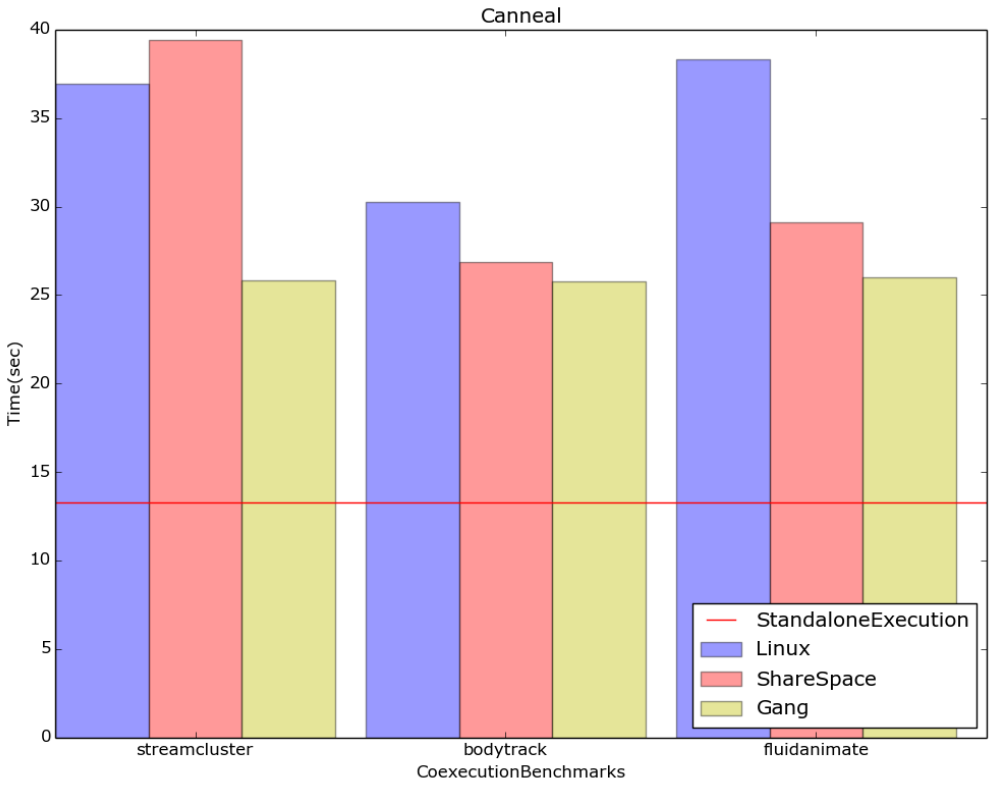
Η Streamcluster χρησιμοποιείται για την κατάταξη ενός συνόλου αντικειμένων σε διαφορετικές ομάδες. Η εφαρμογή αυτή δεν παρουσιάζει καλή κλιμακωσιμότητα αφού στον κώδικα της υπάρχουν σειριακά κομμάτια. Χρησιμοποιεί μεγάλα σύνολα δεδομένων και προκαλεί ψηλό miss rate. Επηρεάζεται από την ταχύτητα της μνήμης και έχει υψηλές απαιτήσεις σε bandwidth, προκαλώντας τις πιο μεγάλες τιμές στη κίνηση δεδομένων (bytes/instructions) από όλες τις εφαρμογές. Επίσης, χρησιμοποιεί μεγάλο αριθμό barriers συγκριτικά με τις άλλες εφαρμογές και υλοποιεί medium-granular παραλληλισμό. Τα νήματα της εφαρμογής αυτής δεν χρησιμοποιούν από κοινού μεγάλο πλήθος δεδομένων.

3.4 Συνεκτελέσεις Εφαρμογών

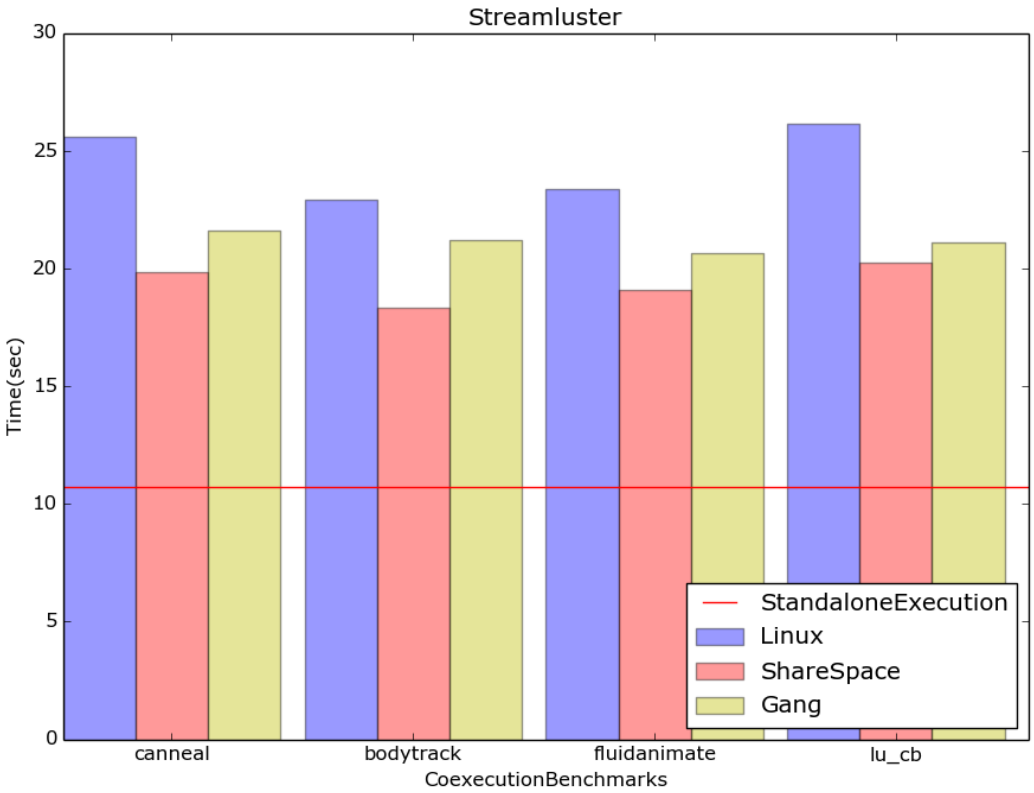
Προκειμένου να μελετηθεί και να επαληθευτεί η συμπεριφορά της κάθε εφαρμογής υλοποιήθηκαν οι πιο κάτω εκτελέσεις. Κάθε εφαρμογή εκτελέστηκε παράλληλα με κάθε μία από τις υπόλοιπες, εξασφαλίζοντας ότι καθ' όλη τη διάρκεια της εκτέλεσής της θα υπάρχουν συνολικά 16 νήματα, 8 από κάθε εφαρμογή, σε πολυπύρηνες πλατφόρμες 8 πυρήνων. Επίσης, κάθε ζεύγος μελετήθηκε με τρεις διαφορετικές πολιτικές χρονοδρομολόγησης υλοποιώντας τους Gang, Space-Sharing και Linux Schedulers. Κάθε γραφική δείχνει τον χρόνο που χρειάστηκε μια εφαρμογή για να εκτελεστεί σε κάθε περίπτωση ξεχωριστά, ενώ φαίνεται επίσης ο χρόνος που χρειάστηκε η συγκεκριμένη εφαρμογή για να εκτελεστεί στην συγκεκριμένη πλατφόρμα μόνη της (standalone execution time).

3.4.1 Αποτελέσματα στην πειραματική πλατφόρμα Sandy Bridge

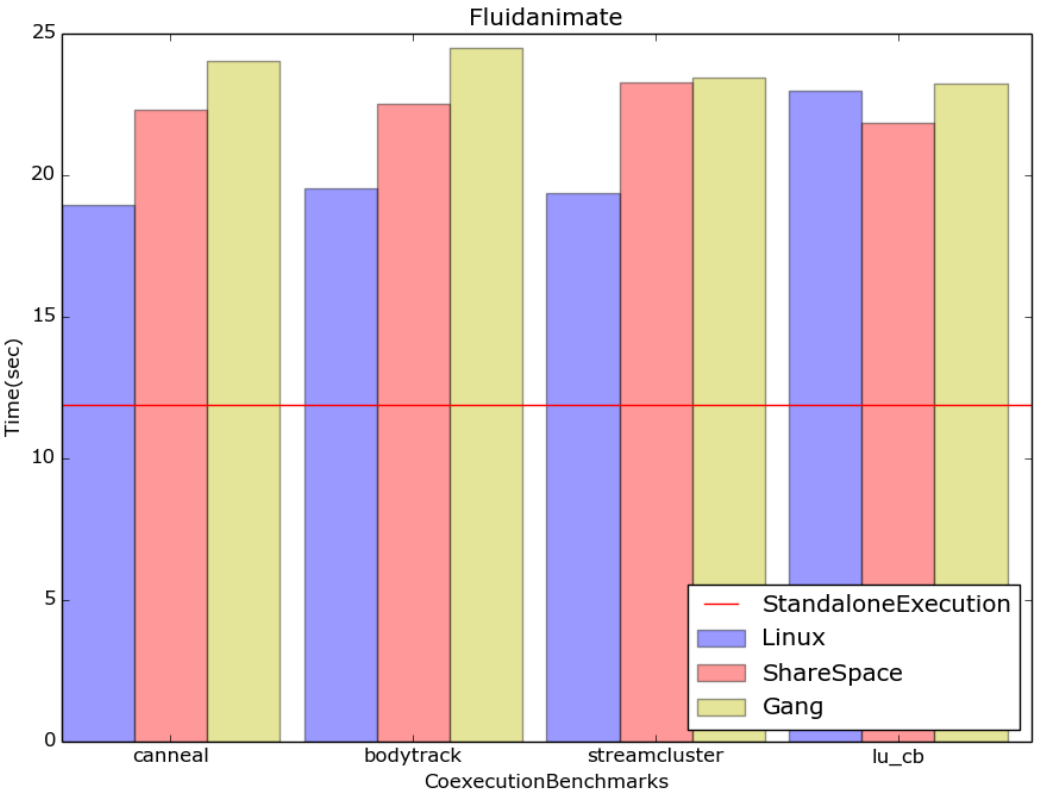
Τα αποτελέσματα για κάθε εφαρμογή ξεχωριστά φαίνονται στις Εικόνες 3.4.1-3.4.5.



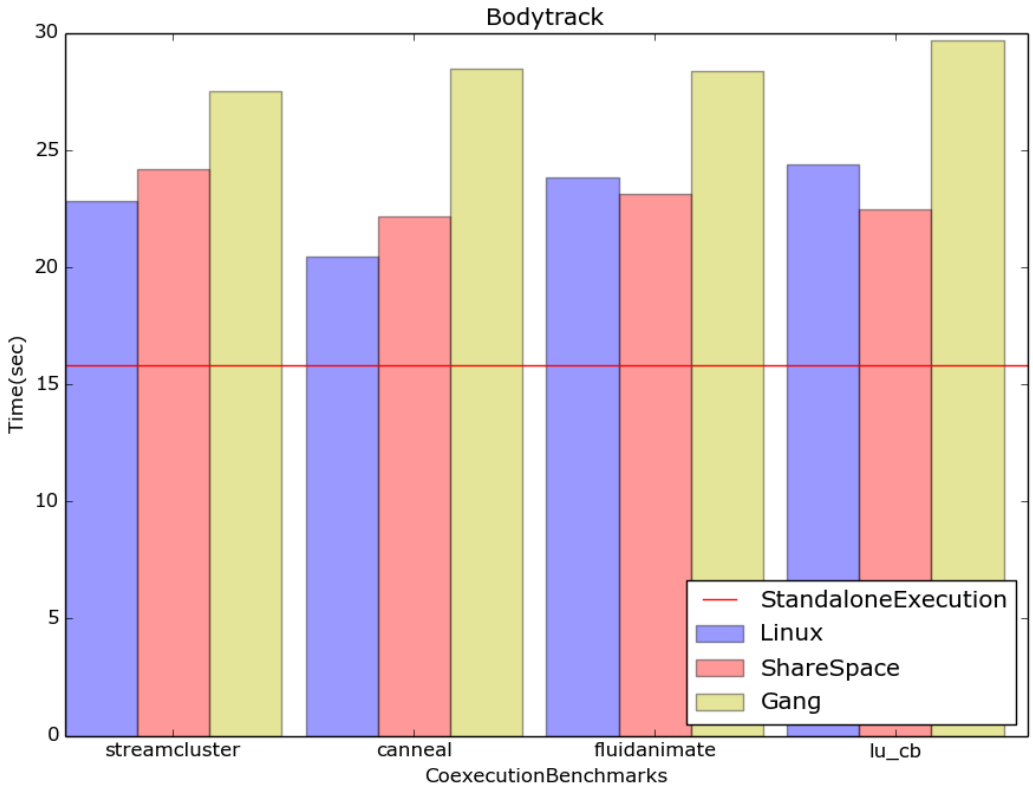
Εικόνα 3.4.1



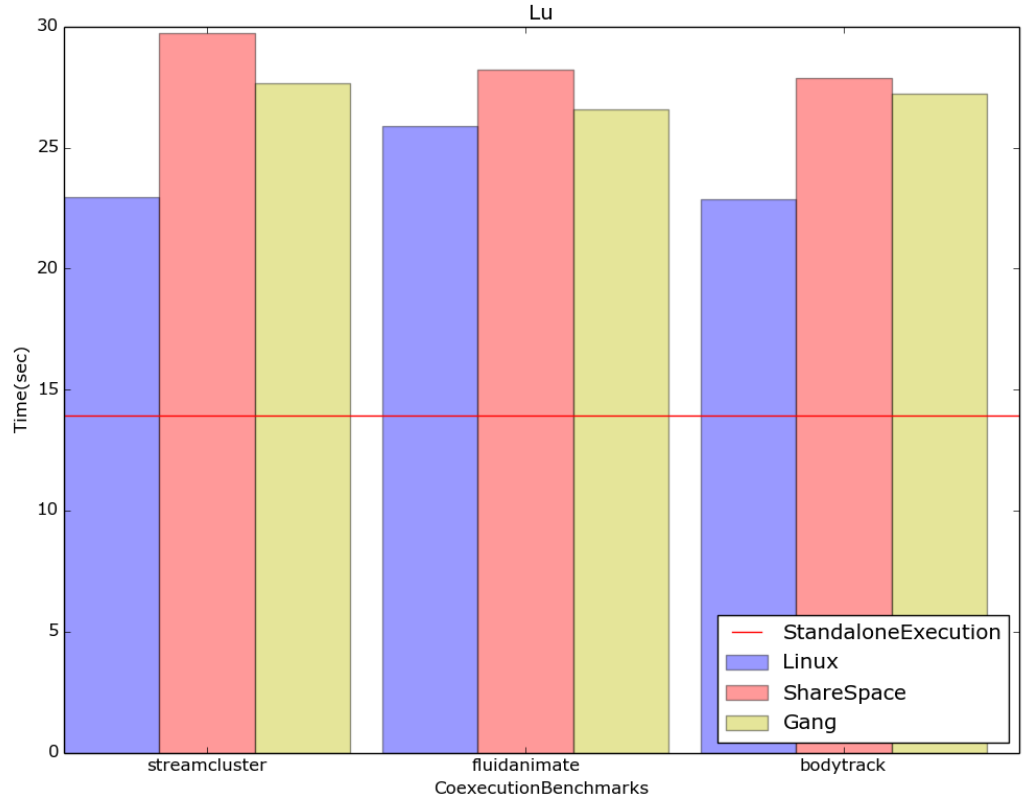
Εικόνα 3.4.2



Εικόνα 3.4.3



Εικόνα 3.4.4



Εικόνα 3.4.5

Από τα πιο πάνω προκύπτουν αρκετές πληροφορίες σχετικά με την κάθε εφαρμογή, καθώς και με τις διαφορετικές πολιτικές χρονοδρομολόγησης.

Το πρώτο πράγμα που θα περίμενε κανείς είναι ο χρόνος που χρειάζεται μια εφαρμογή για να ολοκληρώσει την εκτέλεσή της, όταν εκτελείται παράλληλα με μία δεύτερη εφαρμογή, να είναι τουλάχιστον διπλάσιος από τον χρόνο που χρειάστηκε για να εκτελεστεί μόνη της. Παρ' όλα αυτά σε πολλές περιπτώσεις όπου εφαρμόστηκε ο CFS του Linux, αυτό δεν ισχύει. Αυτό κατά κύριο λόγο οφείλεται στις βελτιστοποιήσεις που κάνει το λειτουργικό ούτως ώστε όταν τα νήματα εκτελούν I/O να μην μένουν ανενεργοί οι πυρήνες. Οι βελτιστοποιήσεις αυτές δεν μπορούν να εφαρμοστούν στην υλοποίηση της Gang πολιτικής.

Από τις γραφικές, αρχικά φαίνεται ξεκάθαρα ότι η Canneal επωφελείται πολύ περισσότερο από την Gang πολιτική. Αυτό πολύ πιθανόν να οφείλεται στο γεγονός ότι είναι μια εφαρμογή με fine-grained παραλληλισμό και υπάρχει αρκετή επικοινωνία ανάμεσα στα νήματά της. Όσο αφορά τον Space-Sharing scheduler στην περίπτωση της Canneal, φαίνεται ότι για τον ίδιο λόγο μπορεί να βελτιώσει τον χρόνο εκτέλεσης σε σχέση με την Linux πολιτική, όμως εδώ θα παίξει ρόλο και η εφαρμογή η οποία εκτελείται παράλληλα. Για παράδειγμα, στην περίπτωση που η Canneal εκτελείται παράλληλα με την Streamcluster υπάρχει επιβάρυνση που οφείλεται στο γεγονός ότι εκτελούνται ταυτόχρονα δύο διεργασίες με μεγάλες απαιτήσεις σε bandwidth και cache.

Από την πλευρά της Streamcluster, φαίνεται ότι η Space-Sharing πολιτική μπορεί να μειώσει τον χρόνο εκτέλεσής της περισσότερο από τις άλλες δύο πολιτικές. Αυτό οφείλεται κυρίως σε δύο λόγους. Ο πρώτος είναι ότι στην εφαρμογή αυτή, όπως και στην Canneal, υπάρχει αρκετή επικοινωνία ανάμεσα στα νήματα και έτσι οι πολιτικές Gang και Space-Sharing υπερτερούν έναντι αυτής του Linux. Ο δεύτερος λόγος είναι το γεγονός ότι πρόκειται για μια εφαρμογή που φτάνει το μέγιστο speed up με λιγότερους από 8 πυρήνες και έτσι επωφελείται περισσότερο όταν τρέχει καθ' όλη τη διάρκεια χωρίς διακοπές, στους μισούς πυρήνες. Έτσι, η εκτέλεσή της διαρκεί λιγότερο χρόνο στην περίπτωση του Space-Sharing από ότι με τον Gang Scheduler. Επίσης, φαίνεται ότι ο χρόνος εκτέλεσης της Streamcluster όταν χρησιμοποιείται η πολιτική Space-Sharing είναι σχεδόν σταθερός, με όποια εφαρμογή και να τρέχει παράλληλα, πράγμα που δεν φαίνεται να ισχύει στον ίδιο βαθμό για τις υπόλοιπες εφαρμογές. Ο λόγος είναι ότι πρόκειται για την εφαρμογή με το πιο ψηλό miss rate στην κοινή μνήμη, που προκαλεί τις πιο ψηλές τιμές traffic. Έτσι, προκύπτει ότι η συνεκτέλεσή της με μια άλλη εφαρμογή δεν μπορεί να επιβαρύνει την ίδια σημαντικά, αλλά μάλλον θα επιβαρυνθεί η δεύτερη.

Όσο αφορά τις Fluidanimate και Bodytrack, υπάρχουν πολλά κοινά σημεία. Για παράδειγμα, είναι εφαρμογές που εκμεταλλεύονται καλύτερα τις ιδιωτικές κρυφές μνήμες. Επομένως, η Gang πολιτική τις επιβαρύνει αφού κάθε φορά που διακόπτουν τα νήματα την εκτέλεσή τους

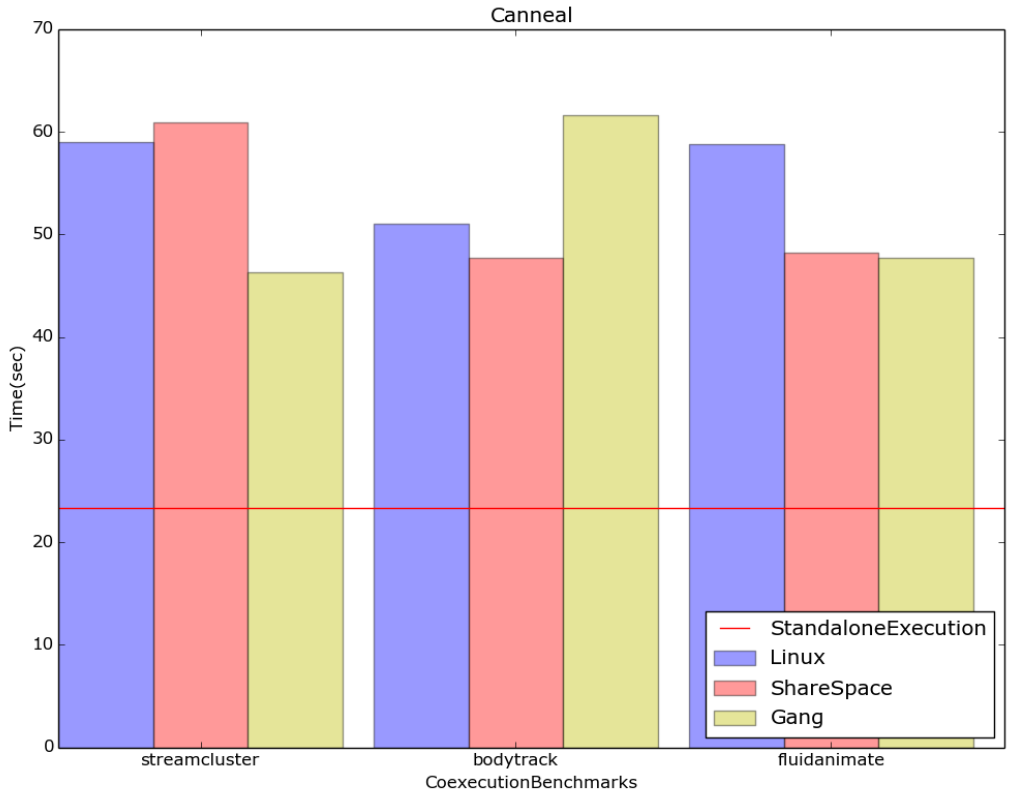
για να εκτελεστούν τα νήματα από την επόμενη διεργασία, οι private caches αλλάζουν τα δεδομένα τους. Η κύρια διαφορά τους είναι ότι η Bodytrack φαίνεται να δίνει καλούς χρόνους εκτέλεσης, στην περίπτωση του Space-Sharing σε αντίθεση με την Fluidanimate, στις εκτελέσεις της οποίας φαίνεται καθαρά ότι ο Linux Scheduler αποτελεί την καλύτερη επιλογή. Τα νήματα της Bodytrack χρησιμοποιούν αρκετά κοινά δεδομένα και συνεπώς η πολιτική Space-Sharing μπορεί να την ευνοεί λόγω του ότι γίνεται καλύτερη αξιοποίηση της cache. Επίσης, καθώς η κίνηση δεδομένων στην Bodytrack είναι πολύ μικρή, αφού χρησιμοποιούνται μικρά σύνολα δεδομένων, στην πολιτική Space-Sharing δεν παρουσιάζονται σημαντικές διαφορές στους χρόνους εκτέλεσης για τις διαφορετικές συνεκτελέσεις της εφαρμογής αυτής.

Τέλος, η εφαρμογή Lu φαίνεται να μην έχει καλά αποτελέσματα στην περίπτωση του Space-Sharing. Λαμβάνοντας υπόψη ότι παρουσιάζει καλή κλιμακωσιμότητα και σημαντική επικοινωνία δικαιολογείται το γεγονός ότι η Gang πολιτική θα δίνει πιο μικρούς χρόνους εκτέλεσης από την Space-Sharing. Ωστόσο, φαίνεται ότι στην Linux πολιτική προκύπτουν ακόμα καλύτεροι χρόνοι εκτέλεσης.

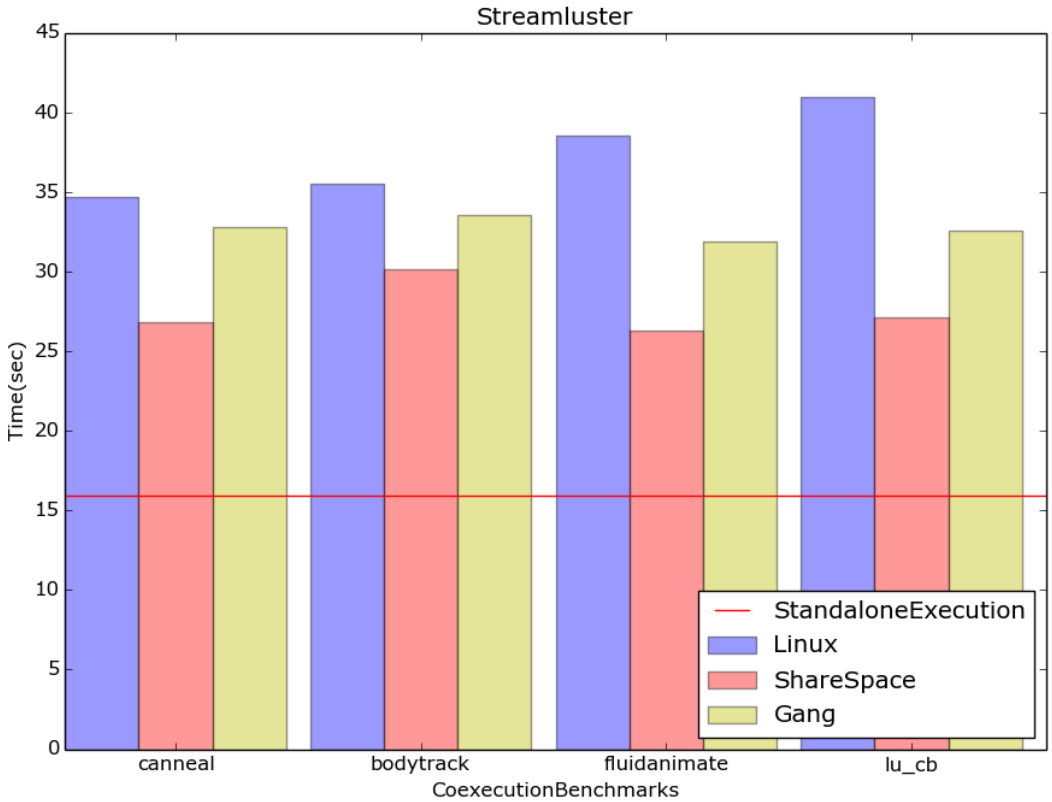
Όσο αφορά τις διάφορες πολιτικές, η πολιτική του Gang Scheduler ελαχιστοποιεί το φαινόμενο του interference, δηλαδή, λόγω του ότι κάθε εφαρμογή εκτελείται μόνη της σε κάθε κβάντο χρόνου, ο χρόνος εκτέλεσής της είναι σχεδόν ανεξάρτητος από το ποια εφαρμογή θα εκτελείται στη συνέχεια. Ο ανταγωνισμός για τους κοινούς πόρους του συστήματος ελαχιστοποιείται, σε αντίθεση με την περίπτωση του Space-Sharing, όπου ουσιαστικά μεγιστοποιείται. Για αυτό τον λόγο, στην πολιτική του Space-Sharing μπορεί να υπάρξουν σημαντικές διαφορές στους χρόνους εκτέλεσης ανάλογα με το ποιες εφαρμογές εκτελούνται παράλληλα. Περισσότερα συμπεράσματα σχετικά με τις τρεις διαφορετικές πολιτικές παρουσιάζονται σε επόμενη υποενότητα.

3.4.2 Αποτελέσματα στην πειραματική πλατφόρμα Opteron

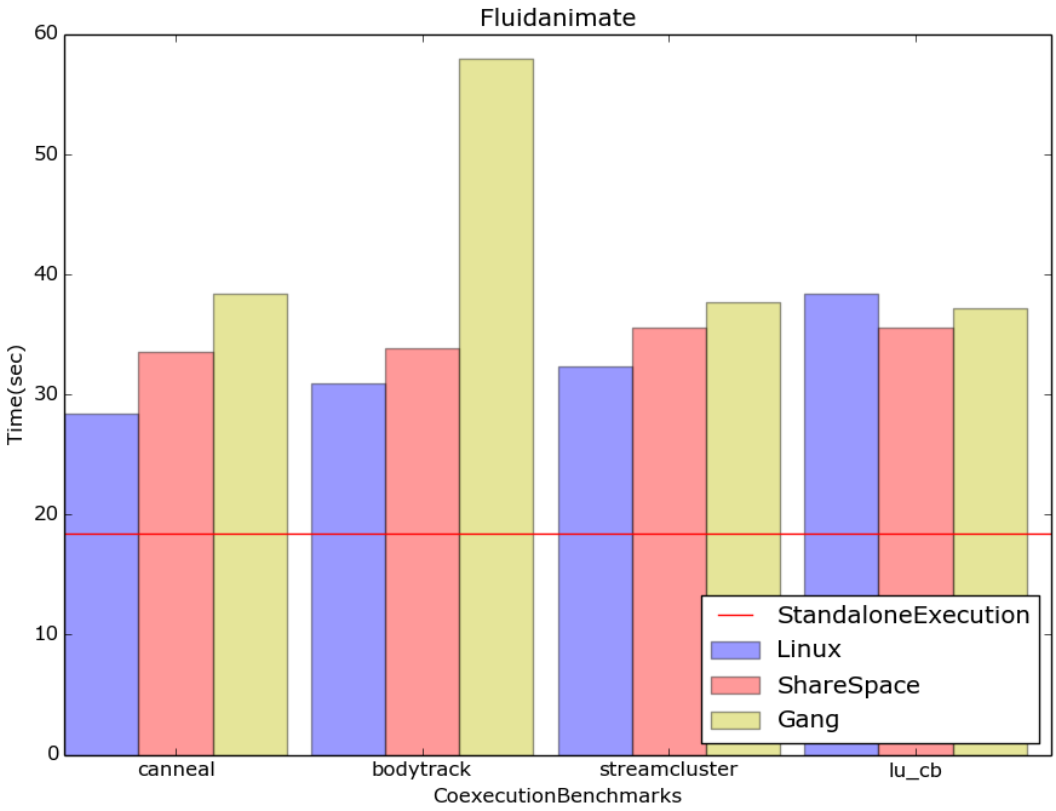
Για επαλήθευση των παραπάνω, έγινε επανάληψη των εκτελέσεων σε δεύτερη πειραματική πλατφόρμα χρησιμοποιώντας και πάλι 8 πυρήνες και 8 νήματα ανά εφαρμογή. Τα αποτελέσματα φαίνονται στις Εικόνες 3.4.6-3.4.10.



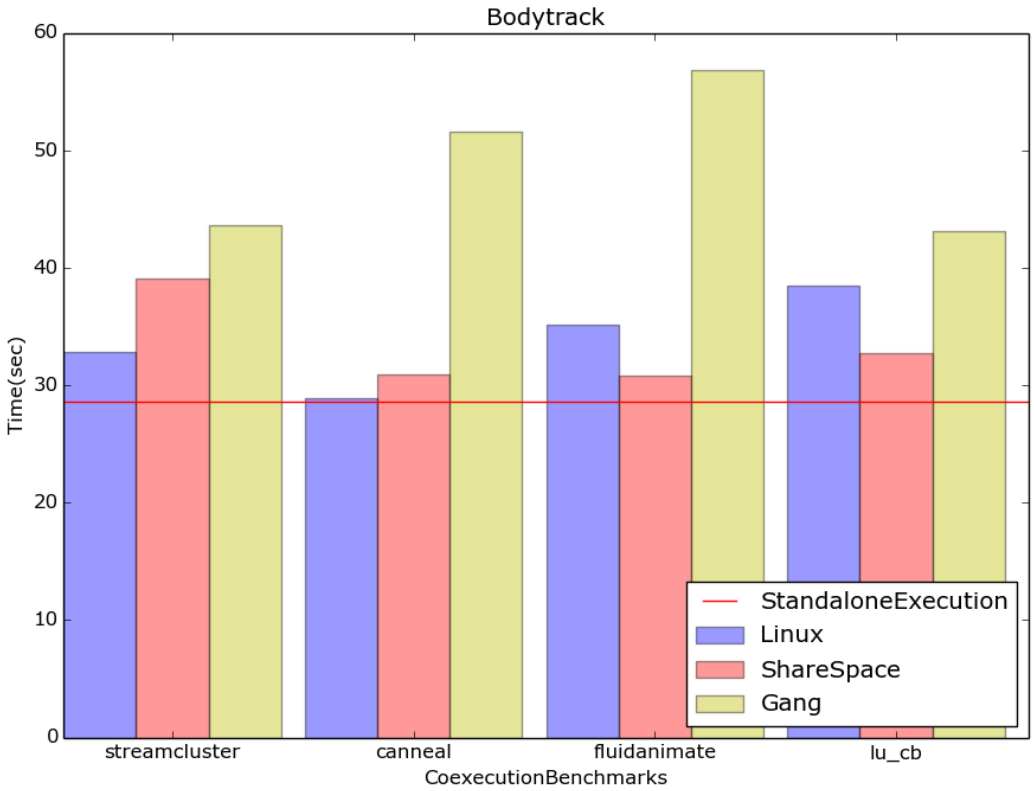
Εικόνα 3.4.6



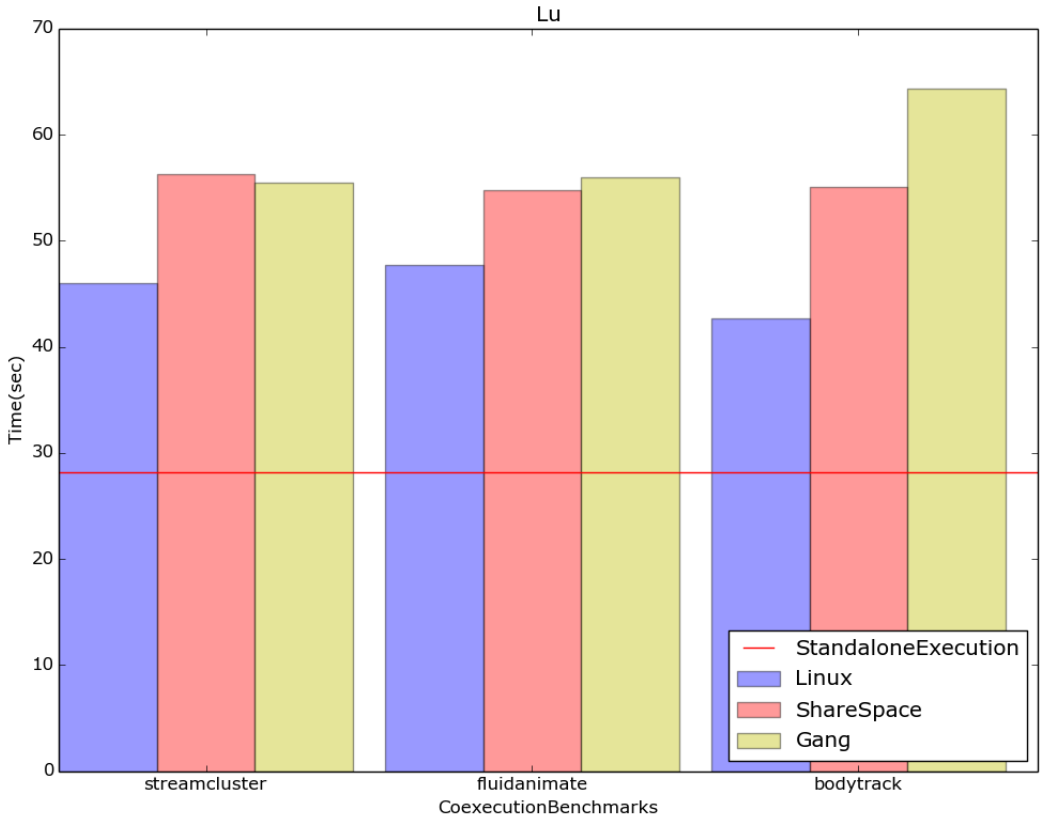
Εικόνα 3.4.7



Εικόνα 3.4.8



Εικόνα 3.4.9



Εικόνα 3.4.10

Στις πιο πάνω γραφικές παραστάσεις, κατά κύριο λόγο, οι εφαρμογές παρουσιάζουν παρόμοιες συμπεριφορές με τα προηγούμενα αποτελέσματα, με κάποιες εξαιρέσεις.

Η πλατφόρμα στην οποία εκτελέστηκαν τα πειράματα αυτά (Εικόνα 2.2), έχει L2 caches, κοινές ανά δύο πυρήνες, με μεγαλύτερη χωρητικότητα από την προηγούμενη πλατφόρμα και ίσως για αυτό τον λόγο, εφαρμογές όπως η Bodytrack, που αξιοποιεί καλύτερα τις private caches ενώ τα νήματά της χρησιμοποιούν κοινά δεδομένα, να παρουσιάζουν σημαντικές βελτιώσεις στις περισσότερες περιπτώσεις που ακολουθούν Space-Sharing ή Gang πολιτική σε σχέση με προηγουμένως. Παρόμοια, στην περίπτωση της Fluidanimate εφαρμογής, φαίνεται σε αρκετές συνεκτελέσεις της να ευνοείται τόσο με την Space-Sharing όσο και με την Gang πολιτική, σε σχέση με την Linux, πράγμα που πριν δεν ίσχυε ίσως λόγω των πιο περιορισμένων ιδιωτικών πόρων.

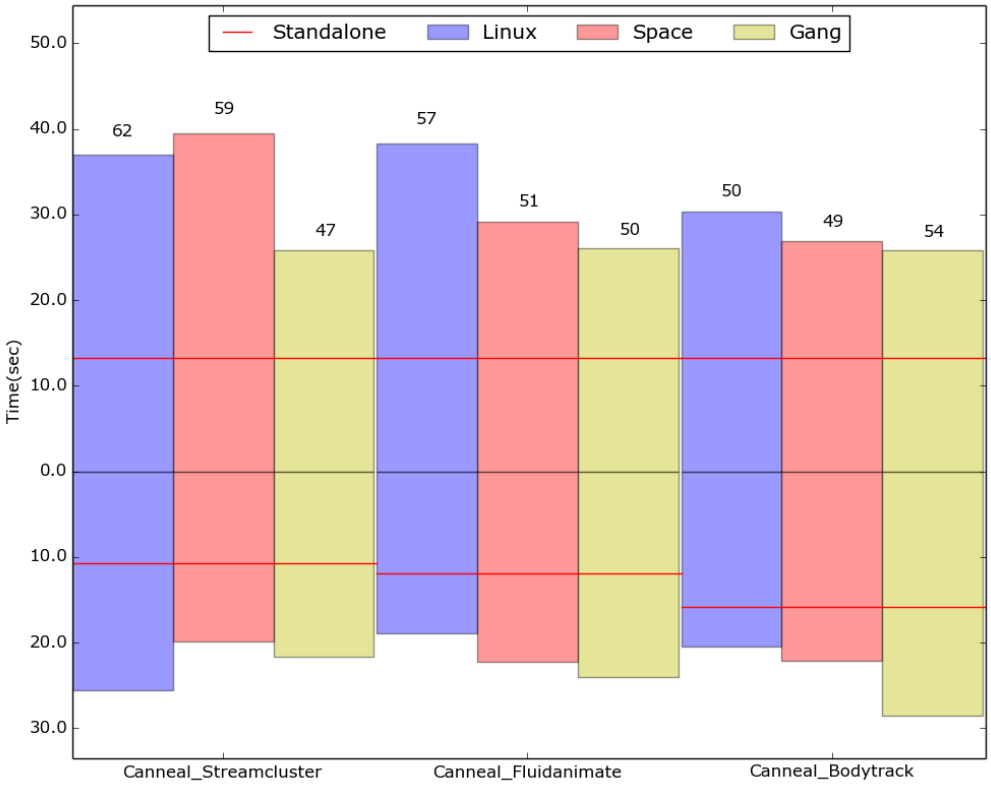
Επίσης, από τα πιο πάνω προκύπτει ότι υπάρχει μια επιβάρυνση των διεργασιών όταν τρέχουν παράλληλα με την Bodytrack σε Gang Scheduler και το φαινόμενο αυτό σε ορισμένες περιπτώσεις είναι πιο έντονο ενώ αλλού πιο περιορισμένο.

Συνοψίζοντας, φαίνεται ότι δεν υπάρχει ιδανικός χρονοδρομολογητής για κάθε εφαρμογή. Ανάλογα με την πολιτική χρονοδρομολόγησης, καθώς και την αρχιτεκτονική του συστήματος

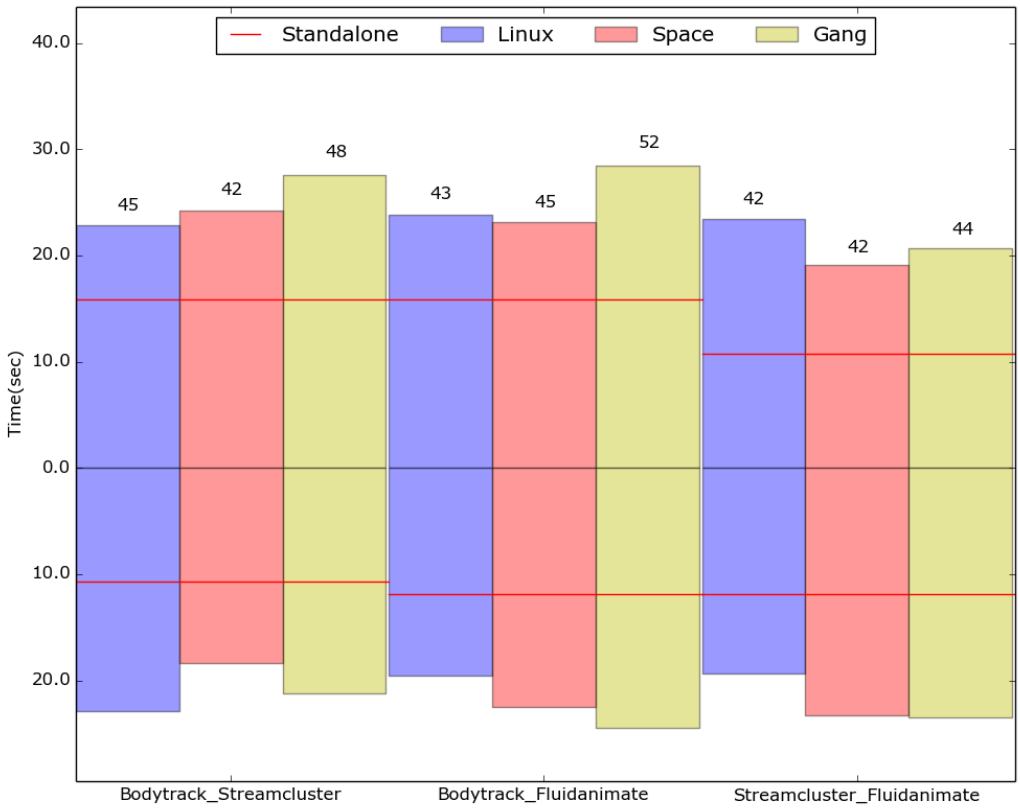
μπορεί να προκύψουν διαφορετικά φαινόμενα ανταγωνισμού ανάμεσα σε κάθε ζεύγος συνεκτέλεσης και αυτά τα φαινόμενα επηρεάζουν σημαντικά τους χρόνους ολοκλήρωσης των εφαρμογών.

3.5 Σύγκριση Πολιτικών Χρονοδρομολόγησης

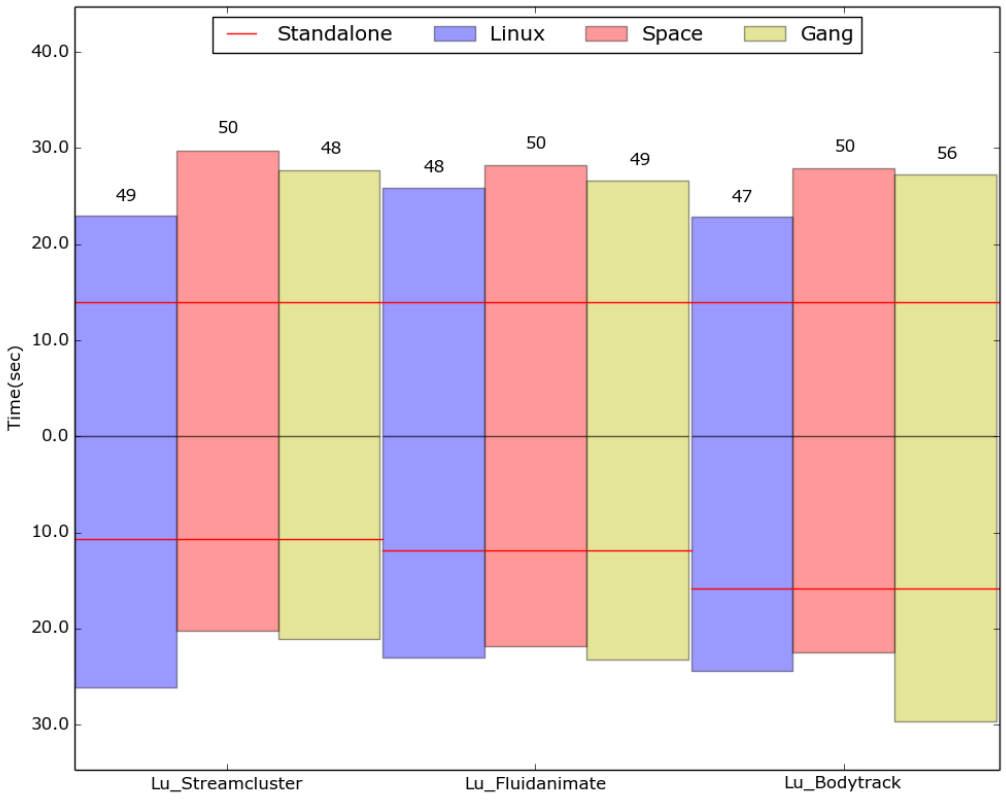
Στις γραφικές στις Εικόνες 2.5.1-2.5.6 παρουσιάζονται τα προηγούμενα αποτελέσματα με διαφορετική μορφή ώστε να γίνει καλύτερη σύγκριση των πολιτικών χρονοδρομολόγησης ανά ζεύγος. Στην κορυφή κάθε μπάρας υπάρχει σημειωμένο το άθροισμα των χρόνων ολοκλήρωσης των εφαρμογών για κάθε ζεύγος. Από τις γραφικές αυτές φαίνεται κατά πόσο κάθε εκτέλεση επιβαρύνει τις εφαρμογές συγκρίνοντας με τον χρόνο του standalone execution. Έτσι, φαίνεται επίσης πιο καθαρά εάν η εκτέλεση ενός ζεύγους εφαρμογών με συγκεκριμένη πολιτική χρονοδρομολόγησης, καθίσταται επωφελής για μια εφαρμογή εις βάρος της άλλης, ή στην καλύτερη περίπτωση, εάν προκύπτουν δίκαια αποτελέσματα και για τις δύο εφαρμογές. Επομένως, εκτός από μικρό χρόνο ολοκλήρωσης θα θέλαμε οι μπάρες να είναι όσο το δυνατό πιο συμμετρικές ως προς τον άξονα.



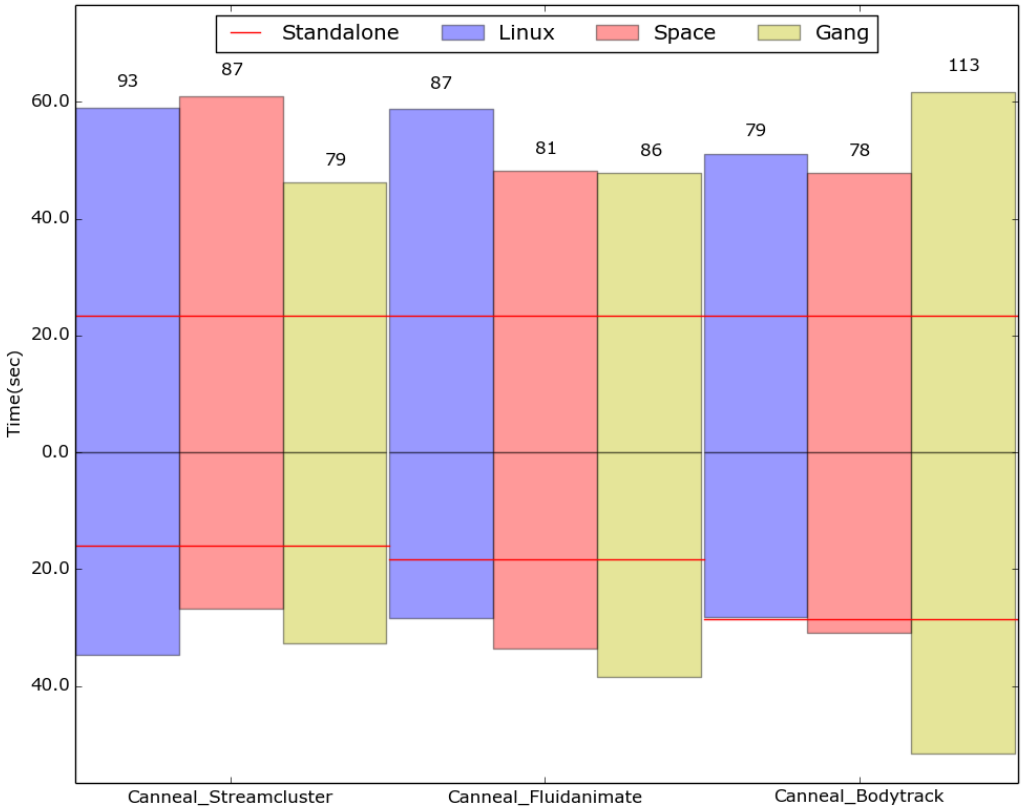
Εικόνα 3.5.1 – Πειραματική πλατφόρμα Sandy Bridge



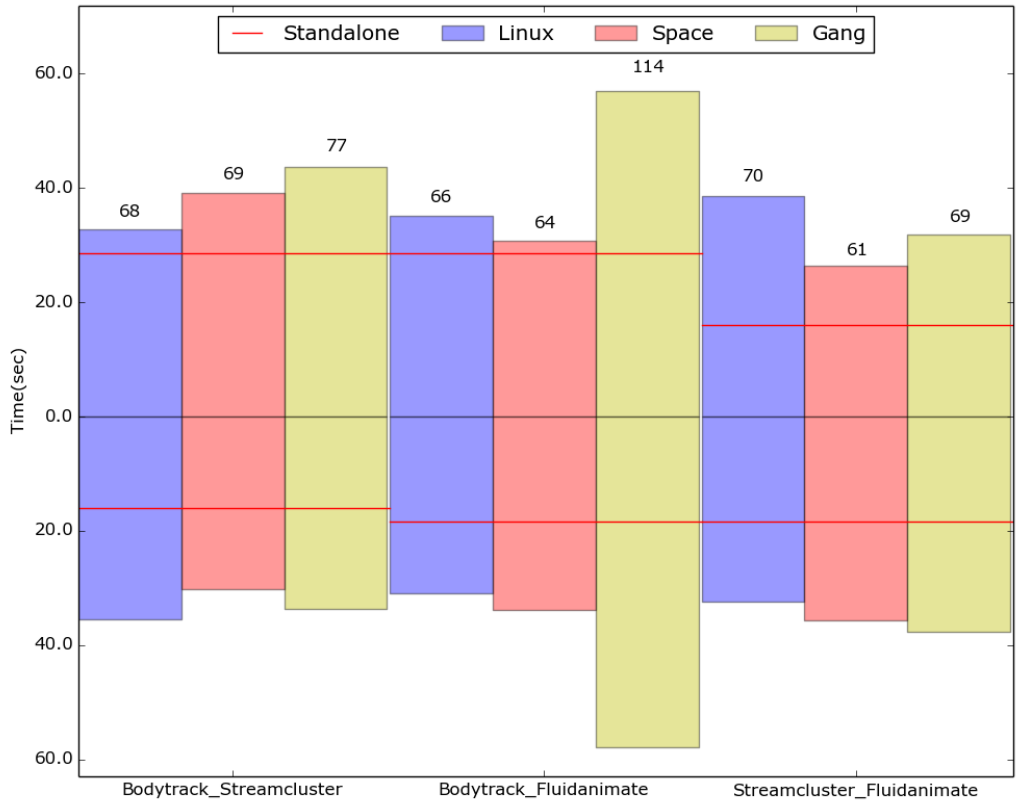
Εικόνα 3.5.2 – Πειραματική πλατφόρμα Sandy Bridge



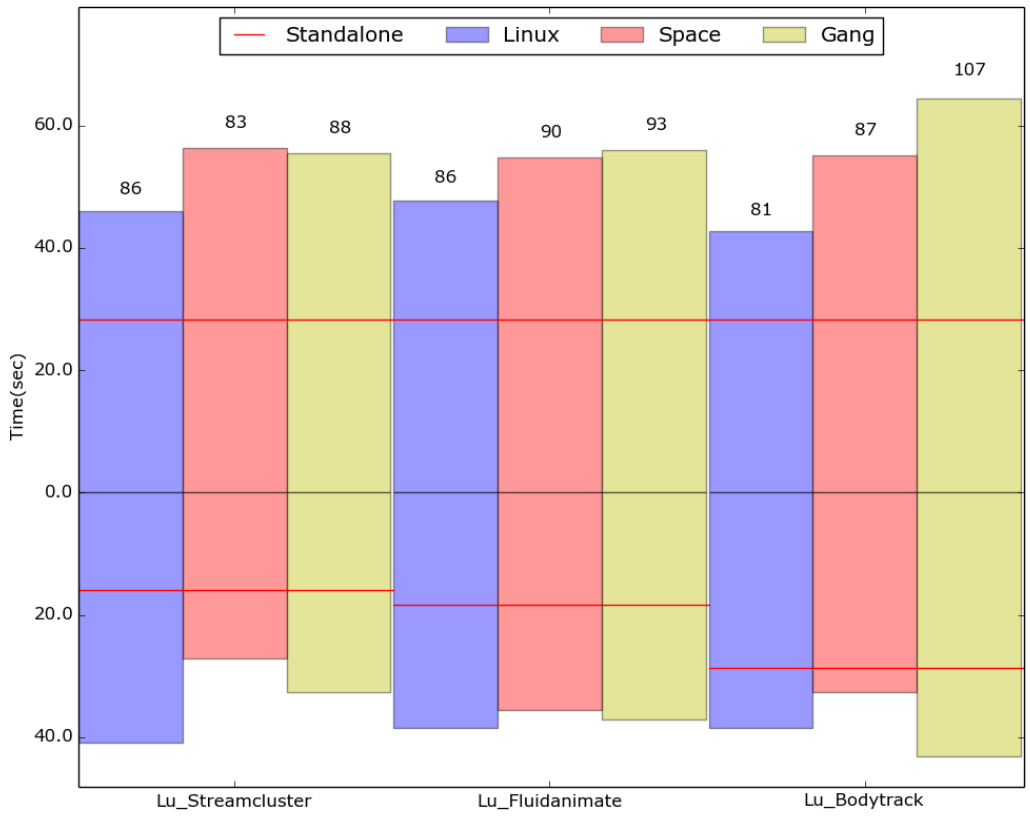
Εικόνα 3.5.3 – Πειραματική πλατφόρμα Sandy Bridge



Εικόνα 3.5.4 – Πειραματική πλατφόρμα Opteron



Εικόνα 3.5.5 – Πειραματική πλατφόρμα Opteron



Εικόνα 3.5.6 – Πειραματική πλατφόρμα Opteron

Από τα πιο πάνω μπορεί να προκύψει ποιος χρονοδρομολογητής είναι καλύτερος για κάθε ζεύγος εφαρμογών, όπως παρουσιάζεται και στον Πίνακα 3.5.

	Πειραματική Πλατφόρμα Sandy Bridge	*Πειραματική Πλατφόρμα Opteron*
Canneal_Streamcluster:	Gang	Gang
Canneal_ Fluidanimate:	Space-Sharing / Gang	Space-Sharing
Canneal_ Bodytrack:	Space-Sharing	Space-Sharing
Bodytrack_Streamcluster:	Space-Sharing	Space-Sharing
Bodytrack_Fluidanimate:	Linux	Linux
Streamcluster_Fluidanimate:	Space-Sharing	Space-Sharing
Lu_Streamcluster:	Gang	Space-Sharing
Lu_ Fluidanimate:	Linux / Gang	Linux
Lu_ Bodytrack:	Linux	Linux

Πίνακας 3.5

Σε αρκετές περιπτώσεις, οι διαφορές ανάμεσα στις πολιτικές δεν ήταν ιδιαίτερα σημαντικές. Εμφανής διαφορά ωστόσο υπάρχει και στις δύο πειραματικές πλατφόρμες στην περίπτωση που συνυπάρχουν οι Canneal και Streamcluster, καθώς πρόκειται για δύο πολύ απαιτητικές σε πόρους εφαρμογές. Εδώ, η Gang πολιτική δίνει πολύ καλύτερα αποτελέσματα από τις δύο άλλες πολιτικές, ενώ η Space-Sharing είναι η πιο άδικη εφόσον τα φαινόμενα ανταγωνισμού για κοινούς πόρους επιβαρύνουν σημαντικά την εφαρμογή που είναι πιο "ευαίσθητη" από τις δύο. Εντούτοις, η Space-Sharing πολιτική αποτελεί μια καλή επιλογή, που δεν απέχει πολύ από την πολιτική του Linux, για τις πολλές από τις υπόλοιπες περιπτώσεις, όπου τουλάχιστον η μία από τις δύο εφαρμογές δεν παρουσιάζει "επιθετική" συμπεριφορά.

Τόσο η Gang πολιτική όσο και η Space-Sharing φαίνεται να επηρεάζονται από την αρχιτεκτονική του συστήματος. Συγκεκριμένα, στη δεύτερη πειραματική πλατφόρμα (Opteron), η Space-Sharing πολιτική μπορεί να δώσει καλύτερους χρόνους, ενώ η Gang πολιτική μπορεί να χαρακτηριστεί ως recourse-wasting για ορισμένες εκτελέσεις.

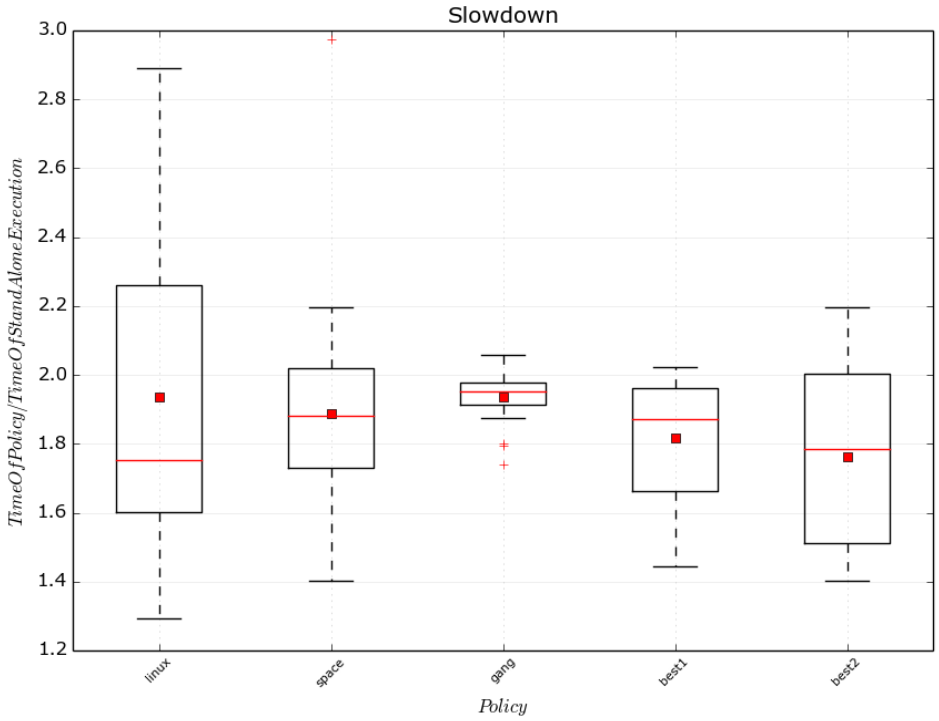
Για καλύτερη αξιολόγηση των πολιτικών χρονοδρομολόγησης, ως κριτήριο μπορούμε να χρησιμοποιήσουμε τις τιμές επιβραδύνσεων (slowdown) των διάφορων συνεκτελέσεων. Η επιβράδυνση ορίζεται ως ο λόγος του χρόνου εκτέλεσης της κάθε εφαρμογής κατά τη συνεκτέλεσή της με κάποια άλλη εφαρμογή ως προς τον χρόνο του standalone execution.

Η παρουσίαση των αποτελεσμάτων γίνεται με Boxplots, με τα οποία έχουμε τη δυνατότητα να παραστήσουμε γραφικά τις στατιστικές κατανομές, χωρίς να γίνονται οποιεσδήποτε υποθέσεις όσο αφορά τις κατανομές αυτές. Σε κάθε κουτί το πάνω και το κάτω μέρος αντιπροσωπεύουν το πρώτο και το τρίτο τεταρτημόριο της κατανομής (first quartile Q1, third quartile Q3), αντίστοιχα, ενώ φαίνεται επίσης η διάμεση τιμή με μια γραμμή μέσα στο κουτί. Πάνω και κάτω, έξω από το κουτί, σημειώνονται με τα whiskers η ελάχιστη και η μέγιστη τιμή των δεδομένων, που βρίσκονται στο 1,5 του εύρους του πρώτου και τρίτου τεταρτημορίου. Δεδομένα με μεγαλύτερες ή μικρότερες τιμές, αναπαριστούνται ξεχωριστά. Τέλος, σημειώνεται με μια κουκίδα η μέση τιμή της κατανομής.

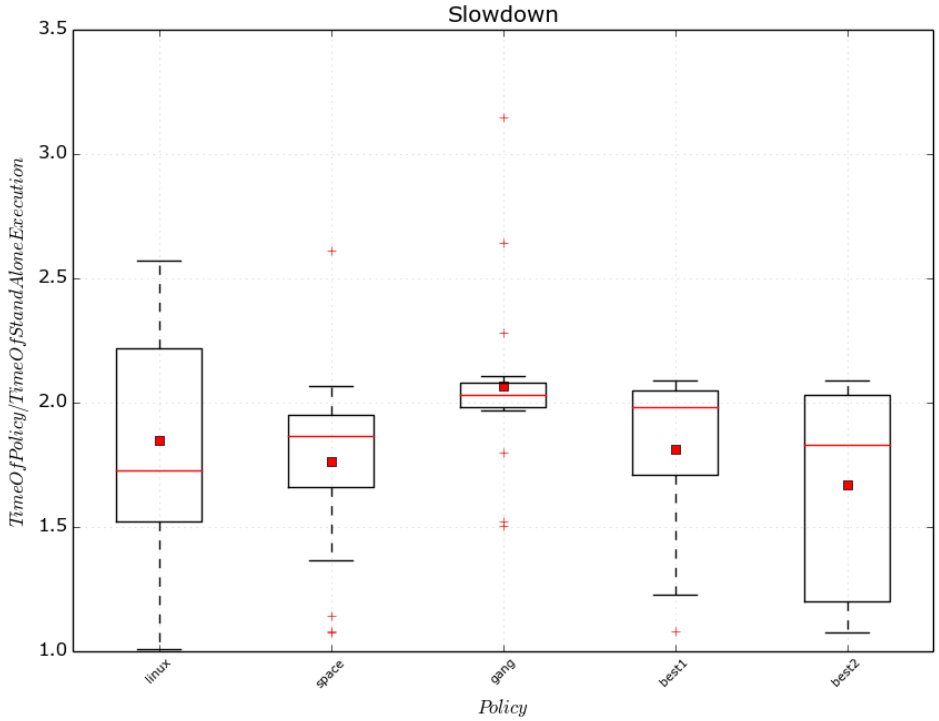
Όσο πιο χαμηλή είναι η διάμεση τιμή, αλλά και η μέση τιμή, τόσο καλύτερα σημαίνει ότι είναι τα αποτελέσματα. Επίσης, ένας δίκαιος χρόνοδρομολογητής, θα έδινε boxplot με όσο το δυνατό πιο περιορισμένο εύρος καθώς θα υπήρχαν μικρές αποκλίσεις ανάμεσα στις τιμές της κατανομής.

Στις Εικόνες 3.5.7 και 3.5.8 φαίνονται τα Boxplots από τις εκτελέσεις στις δύο διαφορετικές πλατφόρμες. Εκτός από τις κατανομές των χρονοδρομολογητών Linux, Gang και Space-Sharing έχουν προστεθεί και οι κατανομές Best1 και Best2. Οι κατανομές αυτές δημιουργήθηκαν προσπαθώντας με δύο διαφορετικούς τρόπους να γίνει επιλογή του καλύτερου χρονοδρομολογητή για κάθε ζεύγος συνεκτέλεσης, προσθέτοντας έτσι σε αυτές τα αντίστοιχα αποτελέσματα από έναν από τους τρεις πιο πάνω χρονοδρομολογητές για κάθε

ζεύγος. Το κριτήριο επιλογής έχει να κάνει με τον μέσο όρο των δύο slowdowns από το ζεύγος συνεκτέλεσης. Επίσης, λαμβάνουμε υπόψη μας για την επιλογή αυτή, την απόκλιση που έχουν μεταξύ τους οι δύο τιμές slowdown από κάθε εφαρμογή της συνεκτέλεσης ώστε να εξασφαλίσουμε ότι με τον συγκεκριμένο χρονοδρομολογητή δεν επωφελείται η μια διεργασία εις βάρος της άλλης. Τέλος, στην επιλογή του καλύτερου χρονοδρομολογητή παίζουν ρόλο και οι αποκλίσεις που έχουν μεταξύ τους στις τιμές του slowdown οι τρεις διαφορετικές πολιτικές, για κάθε εφαρμογή ξεχωριστά.



Εικόνα 3.5.7 – Πειραματική πλατφόρμα Sandy Bridge



Εικόνα 3.5.8 – Πειραματική πλατφόρμα Opteron

Από τα πιο πάνω, είναι προφανές ότι και στις δύο περιπτώσεις, ο χρονοδρομολογητής του Linux δίνει τη μικρότερη ενδιάμεση τιμή, όμως παρουσιάζει τις πιο μεγάλες αποκλίσεις ανάμεσα στις διάφορες τιμές slowdown σε σχέση με τις άλλες πολιτικές. Αυτό φαίνεται από το μεγάλο μέγεθος του box, αντίθετα με τις άλλες πολιτικές που φαίνεται να έχουν πιο δίκαια συμπεριφορά στις διάφορες συνεκτελέσεις. Από την πλευρά του ο Gang Scheduler έχει τις πιο ψηλές τιμές slowdown αλλά είναι πιο δίκαιος από όλους τους άλλους χρονοδρομολογητές. Αυτό εξάλλου είναι λογικό λαμβάνοντας υπόψη μας το γεγονός ότι διανέμει τους πυρήνες στις διεργασίες για ίσο χρονικό διάστημα, ενώ κατά την εκτέλεση των διεργασιών δεν υπάρχουν φαινόμενα interference.

Επίσης, από τις best κατανομές φαίνεται ότι μπορούμε να κρατήσουμε χαμηλή διάμεση τιμή, αντίστοιχη με αυτή του Linux, ενώ ταυτόχρονα, περιορίζεται το εύρος της κατανομής και επιτυγχάνεται μια πιο δίκαια χρονοδρομολόγηση των διεργασιών. Επομένως, μια πολιτική που θα μπορούσε ανάλογα με τις ανάγκες κάθε εφαρμογής να αποφασίσει με πιο τρόπο θα χρονοδρομολογηθούν οι διεργασίες, θα μπορούσε να βελτιώσει την επίδοση του συστήματος.

Συνοψίζοντας τα πιο πάνω επαληθεύονται ορισμένα χαρακτηριστικά για κάθε πολιτική χρονοδρομολόγησης που εξετάστηκε και παρουσιάζονται στα εξής σημεία:

1. Αρχικά, ο Gang χρονοδρομολογητής μπορεί να αποτελέσει βέλτιστη επιλογή σε συγκεκριμένες περιπτώσεις. Η πρώτη είναι όταν υπάρχει μεγάλη επικοινωνία ανάμεσα στα νήματα της εφαρμογής, όπως αποδεικνύεται από τα αποτελέσματα της Canneal. Η δεύτερη περίπτωση είναι όταν θέλουμε να εξασφαλίσουμε απομονωμένο περιβάλλον εκτέλεσης για εφαρμογές που επηρεάζονται σημαντικά από φαινόμενα ανταγωνισμού κοινών πόρων. Η τρίτη περίπτωση είναι όταν οι σκοποί του συστήματος μας επιβάλλουν την επιλογή της πιο δίκαιας πολιτικής. Από την άλλη πλευρά, υπάρχουν και καταστάσεις όπου η επιλογή της Gang πολιτικής μπορεί να μειώσει σημαντικά την επίδοση των εφαρμογών. Αυτό συμβαίνει στις περιπτώσεις κατά τις οποίες οι εφαρμογές δεν παρουσιάζουν καλή κλιμακωσιμότητα και θα ήταν σπατάλη πόρων να δίνονται για ένα χρονικό διάστημα όλοι οι διαθέσιμοι πόροι στην εφαρμογή αυτή. Επιπλέον, με την πολιτική μεγιστοποιείται ο χρόνος αναμονής των εφαρμογών που περιμένουν να εκτελεστούν.
2. Η Space-sharing πολιτική δεν ευνοεί τις περιπτώσεις όπου τρέχουν παράλληλα εφαρμογές με μεγάλες απαιτήσεις σε πόρους και οι απαιτήσεις αυτές διαμορφώνονται από την αρχιτεκτονική του συστήματος. Όταν δεν υπάρχουν φαινόμενα interference, σε αρκετές περιπτώσεις η Space-Sharing δεν διαφέρει σε μεγάλο βαθμό όπου η πολιτική του Linux, λόγω του ότι δεν επιβαρύνει τον χρόνο αναμονής των εφαρμογών, όπως κάνει η Gang πολιτική.
3. Ο CFS του Linux ακολουθώντας μια λογική που στοχεύει στην ελαχιστοποίηση του χρόνου αναμονής κάθε νήματος ξεχωριστά, μπορεί να ευνοήσει εφαρμογές όπως την LU. Παρά το γεγονός ότι σε ορισμένες περιπτώσεις υπάρχουν σημαντικά περιθώρια βελτίωσής της, συγκρίνοντας με τους χρόνους που δίνουν οι άλλες δύο πολιτικές, είναι μια ασφαλής πολιτική, που δίνει κατά κανόνα ικανοποιητικά αποτελέσματα.

4

Πειραματική Αξιολόγηση Πολιτικών Χρονοδρομολόγησης

Στο προηγούμενο κεφάλαιο, εξετάστηκαν ορισμένες εφαρμογές, αναλύθηκαν τα αποτελέσματα τριών χρονοδρομολογητών για παράλληλες εκτελέσεις δύο εφαρμογών και έτσι προέκυψαν κάποια πορίσματα τόσο για τις εφαρμογές όσο και για τις πολιτικές χρονοδρομολόγησης. Ωστόσο, όταν ένας χρονοδρομολογητής αναλαμβάνει περισσότερες εφαρμογές, τα πράγματα είναι αρκετά πιο περίπλοκα. Οι απαιτήσεις συγχρονισμού και τα φαινόμενα ανταγωνισμού για κοινούς πόρους μπορούν να γίνουν πιο έντονα και οι συνέπειες να είναι διαφορετικές.

Επομένως, στο κεφάλαιο αυτό εξετάζονται οι πέντε πολιτικές χρονοδρομολόγησης που αναφέρθηκαν στην Εισαγωγή σε δύο εκτελέσεις. Κάθε εκτέλεση περιλαμβάνει τέσσερεις εφαρμογές οι οποίες εκκινούν ταυτόχρονα και εκτελούνται από μία φορά. Ο Πίνακας 4.1 περιγράφει την αντιστοίχηση των δύο εκτελέσεων για τις γραφικές που ακολουθούν. Οι μετρήσεις έγιναν στις δύο πειραματικές πλατφόρμες, όπως και προηγουμένως.

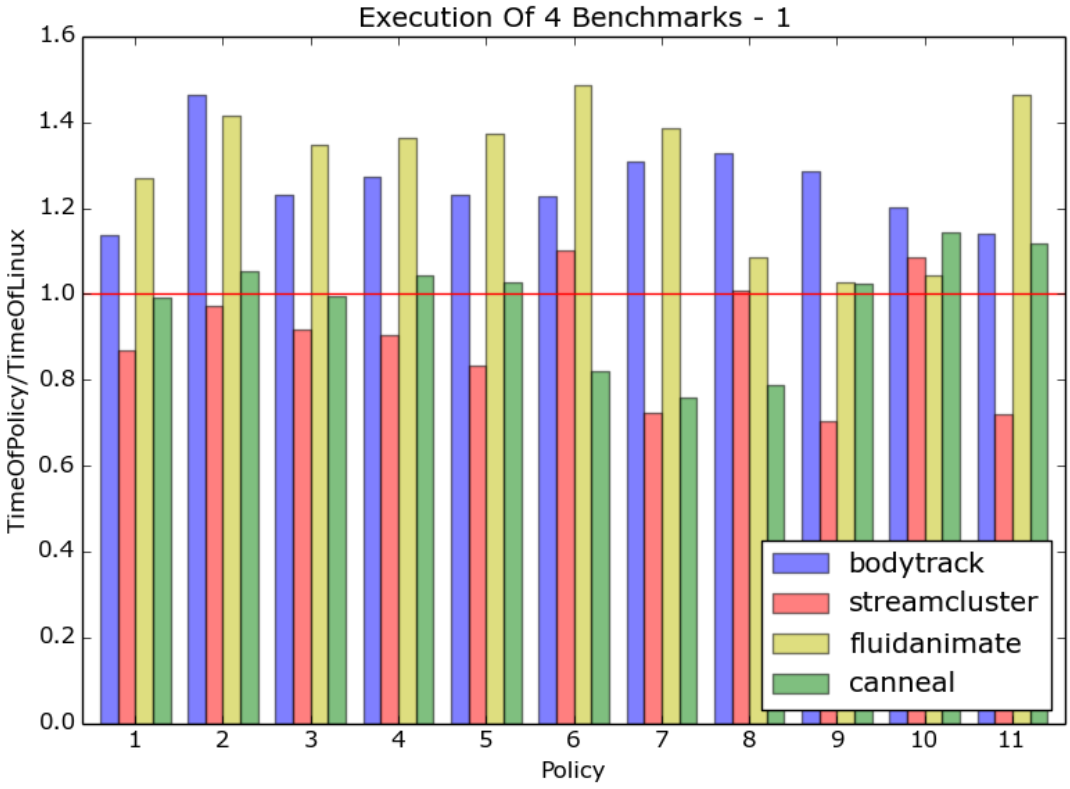
Στην υλοποίηση των δύο νέων μεθόδων χρονοδρομολόγησης (Εικόνες 1.4 & 1.5) υλοποιήθηκαν όλοι οι δυνατοί συνδυασμοί ανάμεσα στις 4 εφαρμογές, για λόγους σύγκρισης. Στις γραφικές παραστάσεις που ακολουθούν κάθε συνδυασμός παρουσιάζεται σαν ξεχωριστή πολιτική. Στον Πίνακα 4.2 παρουσιάζεται η αντιστοίχηση και επεξήγηση των πολιτικών που φαίνονται στις γραφικές.

Execution 1	Execution 2
Benchmark 1: Canneal Benchmark 2: Bodytrack Benchmark 3: Streamcluster Benchmark 4: Fluidanimate	Benchmark 1: Lu Benchmark 2: Bodytrack Benchmark 3: Streamcluster Benchmark 4: Fluidanimate

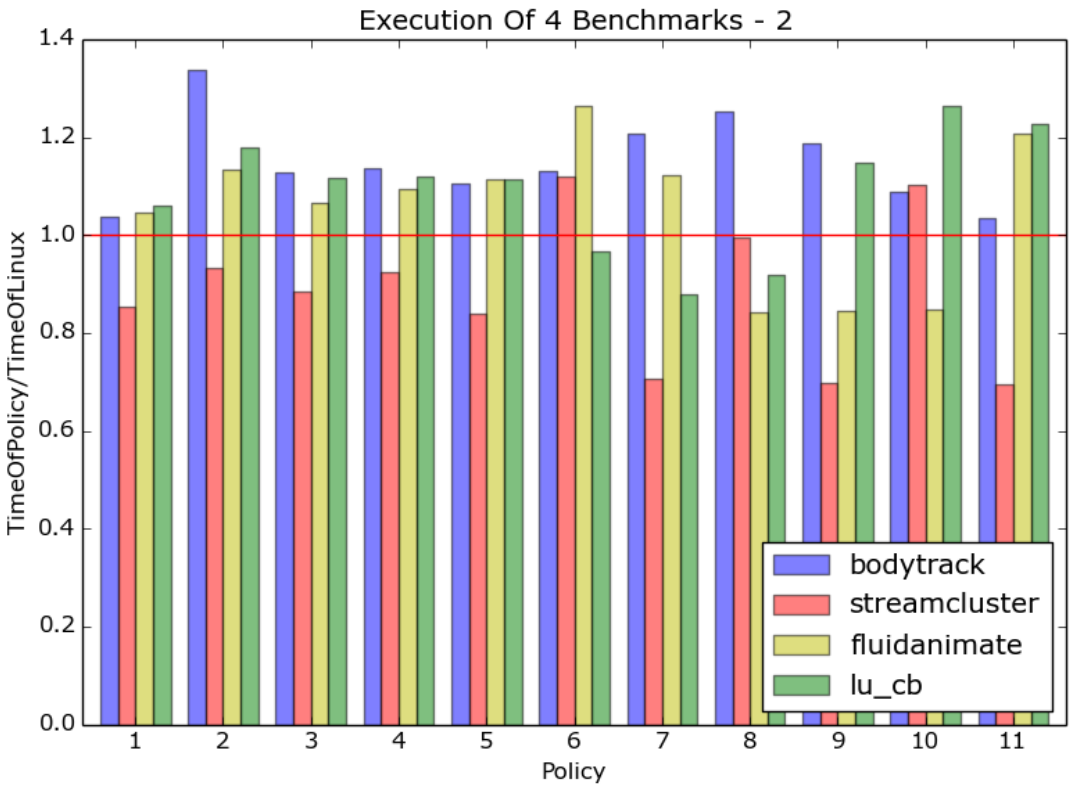
Πίνακας 4.1

4.1 Αποτελέσματα Εφαρμογών

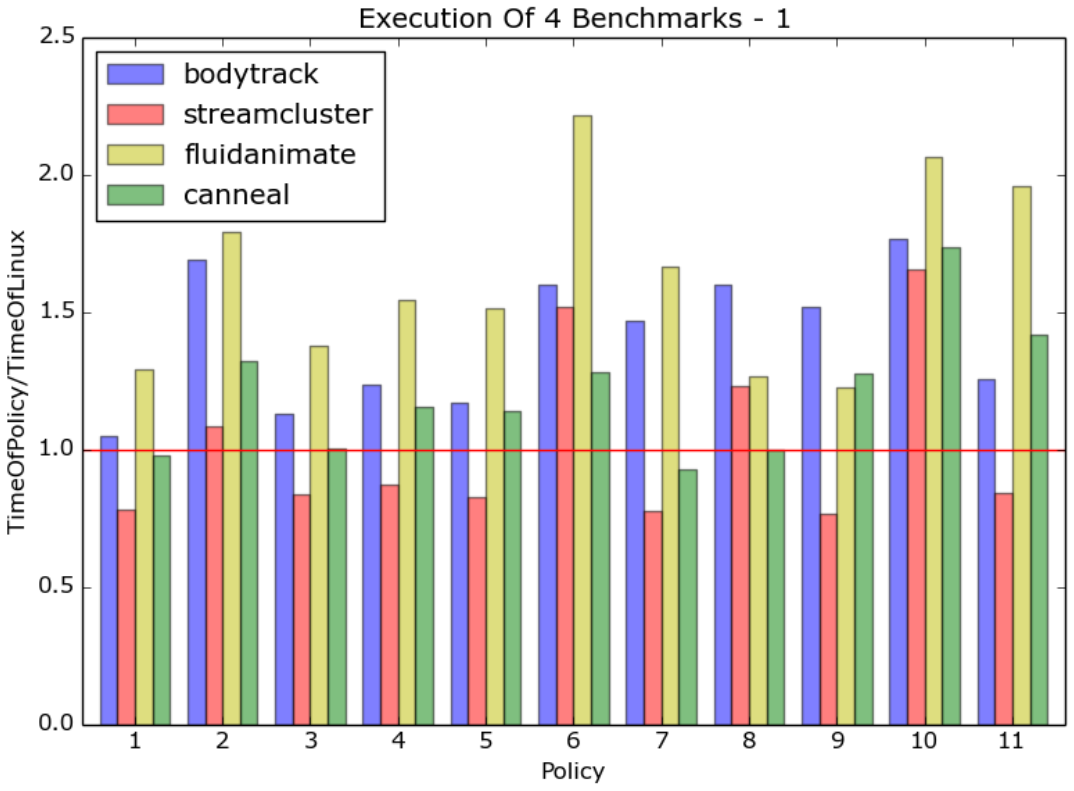
Πιο κάτω, στις Εικόνες 4.1.1-4.1.4 φαίνονται τα αποτελέσματα των πολιτικών χρονοδρομολόγησης για τα δύο set εκτελέσεων στις δύο διαφορετικές πειραματικές πλατφόρμες. Συγκεκριμένα, παρουσιάζονται οι χρόνοι ολοκλήρωσης των εφαρμογών για κάθε εκτέλεση και για κάθε πολιτική ως προς τον αντίστοιχο χρόνο εκτέλεσης του Linux Scheduler.



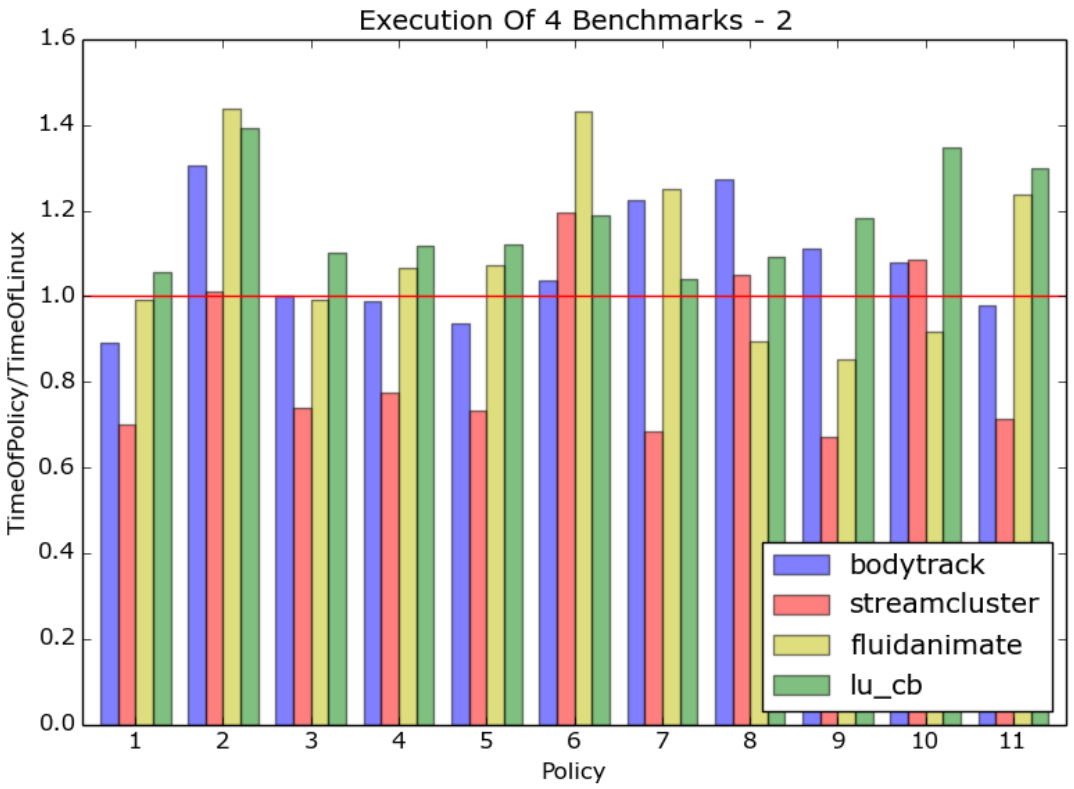
Εικόνα 4.1.1 – Execution 1 – Πειραματική Πλατφόρμα Sandy Bridge



Εικόνα 4.1.2 – Execution 2 – Πειραματική Πλατφόρμα Sandy Bridge



Εικόνα 4.1.3 – Execution 1 – Πειραματική Πλατφόρμα Opteron



Εικόνα 4.1.4 – Execution 2 – Πειραματική Πλατφόρμα Opteron

Από τα πιο πάνω φαίνεται ότι υπάρχουν πολιτικές με τις οποίες οι χρόνοι εκτέλεσης των εφαρμογών μπορούν να βελτιωθούν σημαντικά σε σχέση με τον αντίστοιχο χρόνο που επιτυγχάνει ο scheduler του Linux. Συγκεκριμένα, η εφαρμογή Streamcluster στις περισσότερες πολιτικές που εφαρμόστηκαν σημείωνε καλύτερους χρόνους, με μείωση έως και 30%. Αυτό βέβαια ήταν αναμενόμενο λαμβάνοντας υπόψη τα προηγούμενα συμπεράσματα καθώς πρόκειται για μια εφαρμογή που δεν επηρεάζεται σημαντικά όταν μοιράζεται τους πόρους του συστήματος με άλλες διεργασίες που τρέχουν παράλληλα, ενώ τα νήματά της επικοινωνούν μεταξύ τους και επομένως υπάρχει όφελος όταν αυτά εκτελούνται σε στο ίδιο κβάντο χρόνου. Αντίστοιχα, η Canneal με κατάλληλους συνδυασμούς μπορεί να βελτιώσει τον χρόνο εκτέλεσής της. Από την άλλη πλευρά, οι υπόλοιπες εφαρμογές φαίνεται ότι μπορούν να παρουσιάσουν βελτιώσεις ανάλογα με τις εφαρμογές που εκτελούνται παράλληλα καθώς και με την αρχιτεκτονική του συστήματος.

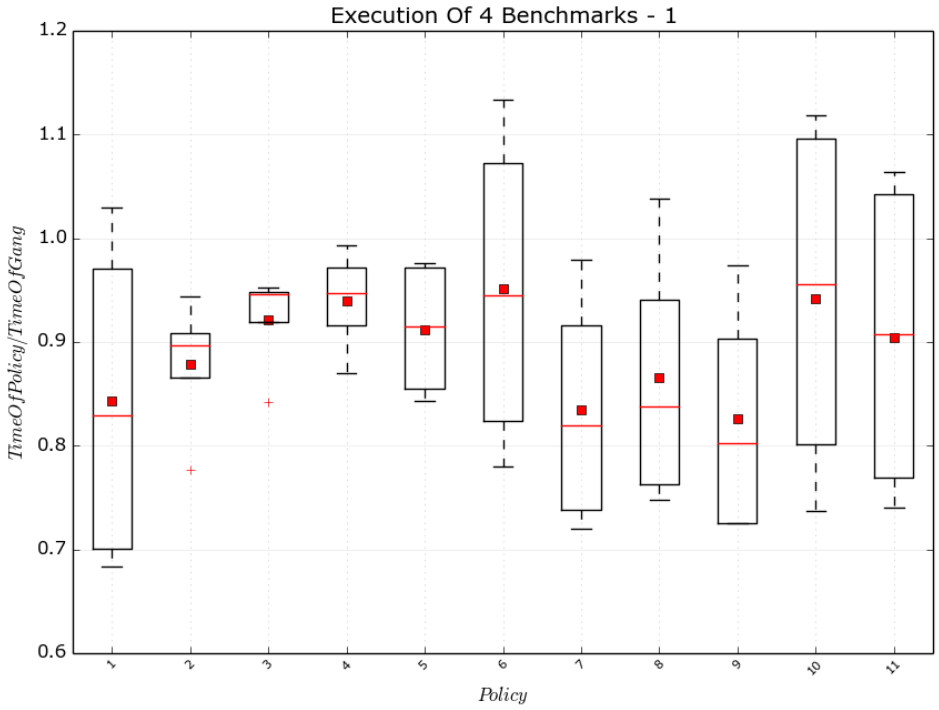
Policy 1	Space-Sharing
Policy 2	*Gang*
Policy 3	*New Method 1* First Time Slot Co-running Benchmarks 1 & 2 Second Time Slot Co-running Benchmarks 3 & 4
Policy 4	*New Method 1* First Time Slot Co-running Benchmarks 1 & 3 Second Time Slot Co-running Benchmarks 2 & 4
Policy 5	*New Method 1* First Time Slot Co-running Benchmarks 1 & 4 Second Time Slot Co-running Benchmarks 2 & 3
Policy 6	*New Method 2* First Time Slot Running Benchmark 1 Second Time Slot Running Benchmark 2 Third Time Slot Co-running Benchmarks 3 & 4
Policy 7	*New Method 2* First Time Slot Running Benchmark 1 Second Time Slot Running Benchmark 3 Third Time Slot Co-running Benchmarks 2 & 4
Policy 8	*New Method 2* First Time Slot Running Benchmark 1 Second Time Slot Running Benchmark 4 Third Time Slot Co-running Benchmarks 2 & 3
Policy 9	*New Method 2* First Time Slot Running Benchmark 3 Second Time Slot Running Benchmark 4 Third Time Slot Co-running Benchmarks 1 & 2
Policy 10	*New Method 2* First Time Slot Running Benchmark 2 Second Time Slot Running Benchmark 4 Third Time Slot Co-running Benchmarks 1 & 3
Policy 11	*New Method 2* First Time Slot Running Benchmark 2 Second Time Slot Running Benchmark 3 Third Time Slot Co-running Benchmarks 1 & 4

Πίνακας 4.2

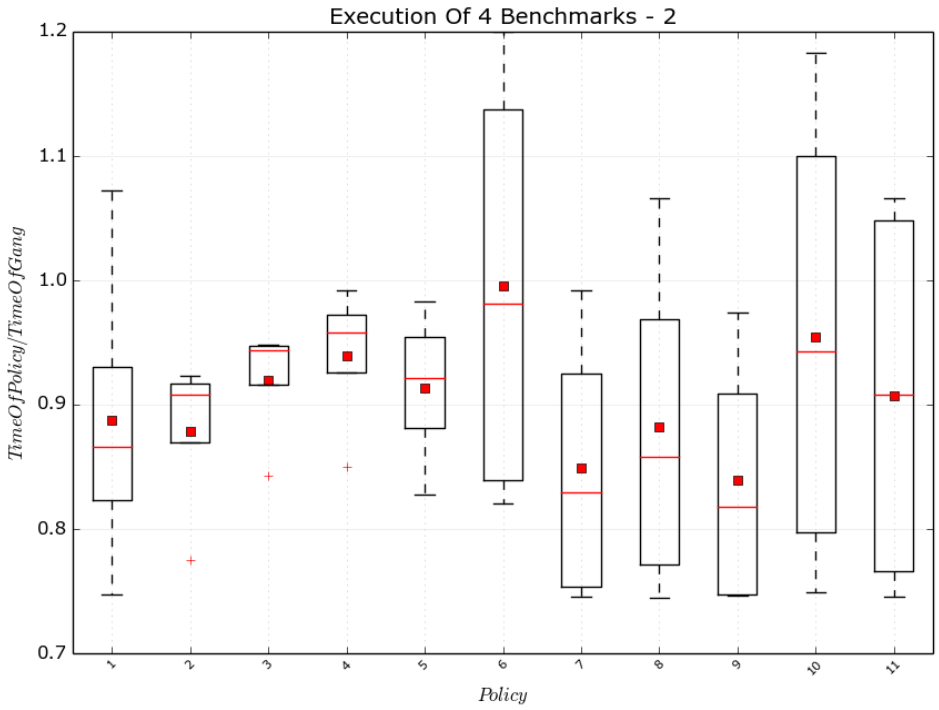
4.2 Σύγκριση Αλγορίθμων

Για την αξιολόγηση των αλγορίθμων χρησιμοποιήθηκαν όπως και πριν Boxplots που παρουσιάζονται στις Εικόνες 4.2.1-4.2.4. Αυτή τη φορά ως παράμετρος αποδοτικότητας χρησιμοποιήθηκε ο λόγος του χρόνου εκτέλεσης των εφαρμογών εφαρμόζοντας κάθε πολιτική ως προς τον αντίστοιχο χρόνο εκτέλεσης της Gang πολιτικής. Ο λόγος είναι γιατί σύμφωνα με τα πιο πάνω, ο Gang Scheduler παρουσιάζει πιο δίκαια συμπεριφορά. Επίσης, με αυτόν τον τρόπο γίνεται ευδιάκριτη η σύγκριση των υπόλοιπων πολιτικών ως προς την πολιτική του Linux.

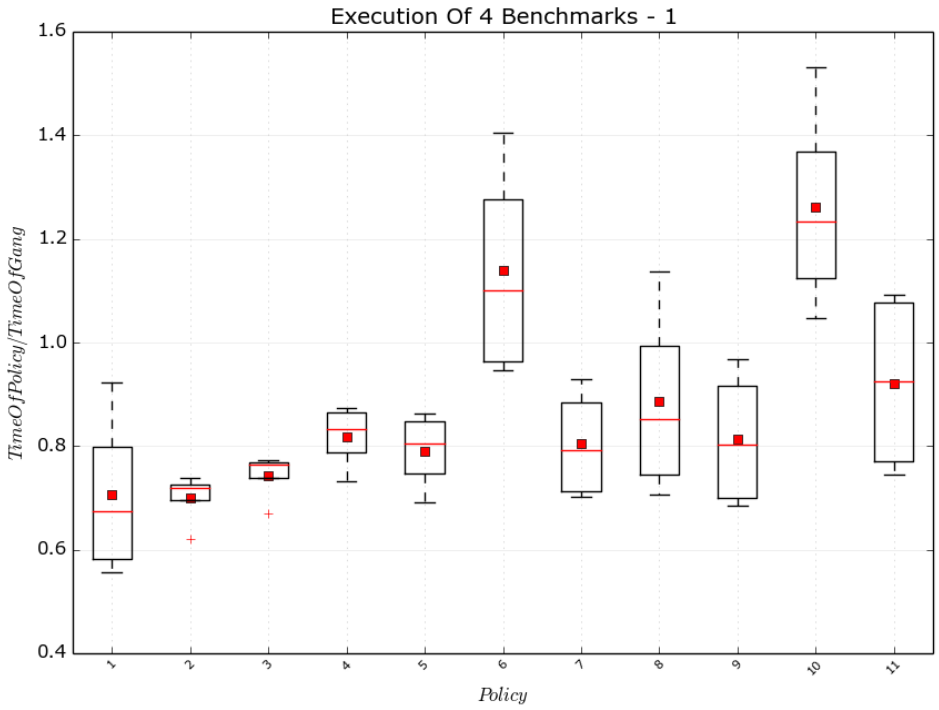
Η αντιστοίχηση των πολιτικών χρονοδρομολόγησης που παρουσιάζονται είναι η ίδια με αυτή του Πίνακα 4.2 με την διαφορά ότι η Policy 1 τώρα είναι αυτή του Linux και η Policy 2 είναι η Space-Sharing.



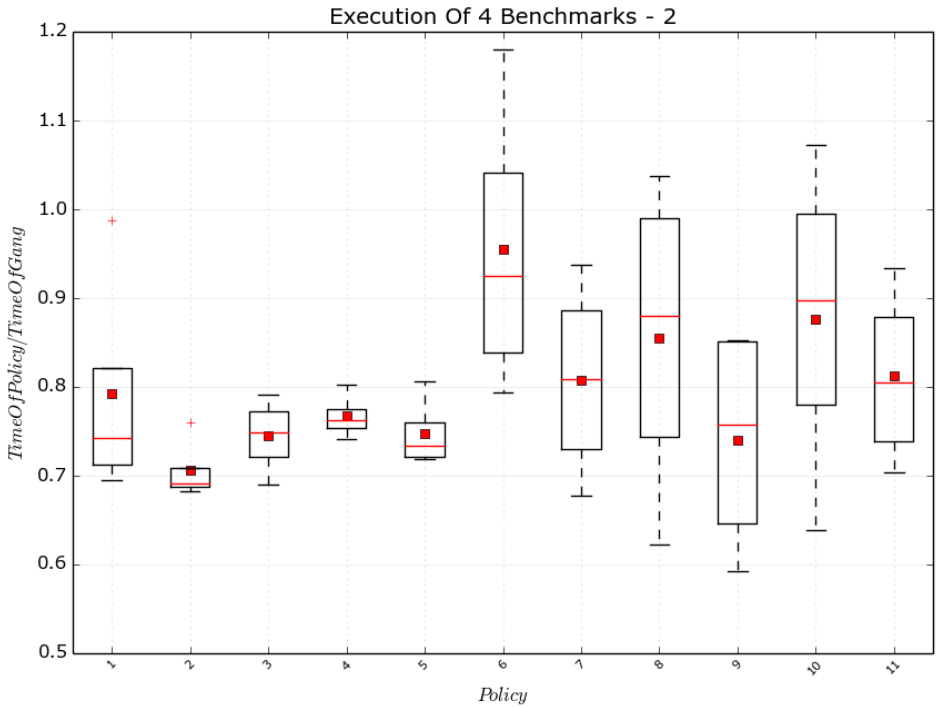
Εικόνα 4.2.1 – Execution 1 – Πειραματική Πλατφόρμα Sandy Bridge



Εικόνα 4.2.2 – Execution 2 – Πειραματική Πλατφόρμα Sandy Bridge



Εικόνα 4.2.3 – Execution 1 – Πειραματική Πλατφόρμα Opteron



Εικόνα 4.2.4 – Execution 2 – Πειραματική Πλατφόρμα Opteron

Από τα αποτελέσματα που πήραμε στο Κεφάλαιο 2, μπορούν εύκολα να δικαιολογηθούν οι συνδυασμοί των δύο νέων μεθόδων που δίνουν καλύτερους χρόνους εκτέλεσης. Για παράδειγμα, δεν θα θέλαμε οι εφαρμογές Canneal και Streamcluster, να εκτελούνται στο ίδιο κβάντο χρόνου. Έτσι, για το πρώτο set εφαρμογών, Execution 1, από τις πολιτικές 3-5 θα αποφεύγαμε την 4η. Αντίστοιχα, για τις πολιτικές 6-11, θα επιλέγαμε τους συνδυασμούς που υλοποιούνται από τις πολιτικές 7 ή 9, στις οποίες η Streamcluster τρέχει απομονωμένη χωρίς να επηρεάζει κάποια άλλη εφαρμογή. Επιπλέον στις πολιτικές 7 και 9 φροντίζουμε ότι η Bodytrack θα μοιράζεται τους πυρήνες με κάποια άλλη εφαρμογή κατά την εκτέλεσή της. Ο λόγος είναι γιατί πρόκειται για μια εφαρμογή χωρίς μεγάλες απαιτήσεις σε κοινούς πόρους, η οποία όπως αποδείχτηκε δεν αξιοποιεί κατάλληλα και τους 8 πυρήνες όταν τους έχει στη διάθεσή της και ούτε αντιμετωπίζει σημαντικά ζητήματα επικοινωνίας ανάμεσα στα νήματά της.

Επίσης, από τα αποτελέσματα του Κεφαλαίου 2, είδαμε ότι η εναλλαγή των εφαρμογών στους πυρήνες της πλατφόρμας Opteron με την Gang πολιτική σε αρκετές περιπτώσεις προκαλούσε σημαντικές καθυστερήσεις. Έτσι, αναμέναμε ότι και εδώ όσο καθώς αυξάνονται οι εναλλαγές των εφαρμογών, αυτές θα επιβαρύνονται αντίστοιχα. Αυτό επαληθεύεται και από τα πιο πάνω, όπου ορισμένες από τις πολιτικές 6-11 στην πλατφόρμα Opteron, έχουν τα

χειρότερα αποτελέσματα. Συγκεκριμένα αυτό συμβαίνει όταν η μια από τις δύο εφαρμογές που καταλαμβάνουν και τους 8 πυρήνες είναι η Bodytrack (Policies 6 & 10).

Αξίζει επίσης να σημειωθεί ότι στις πολιτικές 3-5 ουσιαστικά κατανέμονται πιο δίκαια οι πόροι του συστήματος από ότι στις πολιτικές 6-11. Ο λόγος είναι ότι στις τελευταίες 6 υλοποιήσεις οι δύο από τις 4 διεργασίες εκτελούνται μόνες τους και στους 8 πυρήνες του συστήματος για κάποιο χρονικό διάστημα, ενώ οι άλλες δύο διεργασίες θα εκτελεστούν παράλληλα στους μισούς πυρήνες. Ο χρόνος, όμως, που δίνεται στις διεργασίες είναι ο ίδιος για όλες. Επομένως, το γεγονός ότι οι πολιτικές 3-5 αναπαριστούνται με boxes πιο περιορισμένου εύρους ήταν αναμενόμενο.

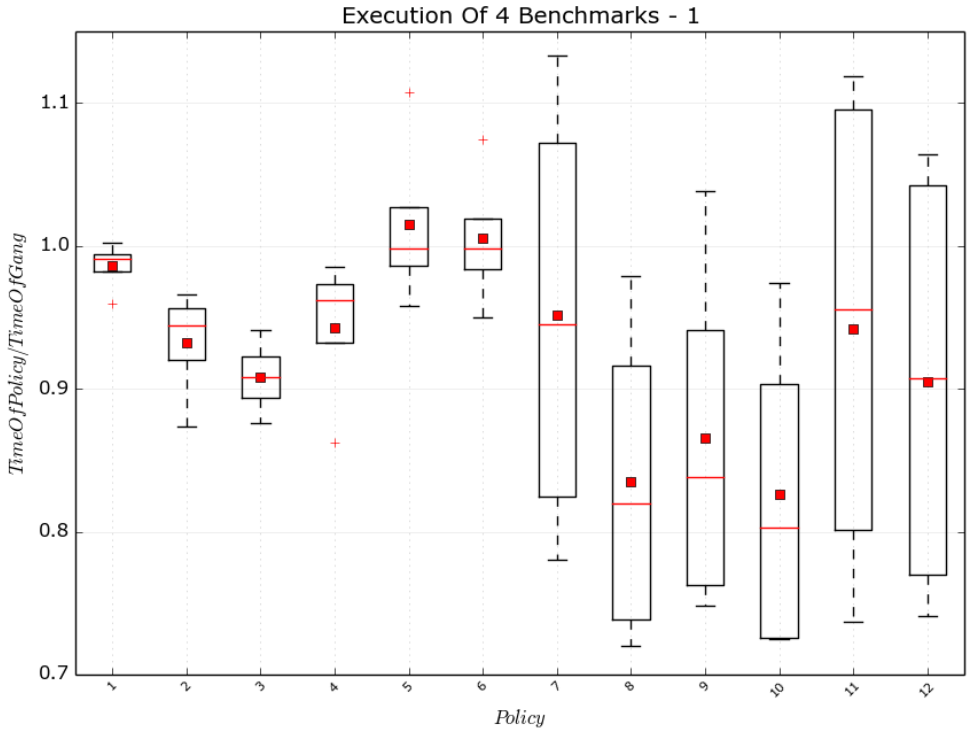
Ωστόσο, οι πολιτικές 3-5 δεν μπορούν πάντα να βελτιώνουν και τις τιμές των χρόνων εκτέλεσης. Όσον αφορά το δεύτερο set εφαρμογών (Execution 2) στη πειραματική πλατφόρμα Opteron, μπορούν να επιτευχθούν καλύτερα αποτελέσματα με τις πολιτικές αυτές, καθώς η πρώτη εφαρμογή του (Lu) είναι λιγότερο επιθετική από την αντίστοιχη εφαρμογή του πρώτου set (Canneal).

Επίσης, είναι προφανές ότι καθοριστικό ρόλο παίζει και η αρχιτεκτονική του συστήματος η οποία μπορεί να ευνοήσει ή να επιβαρύνει τις εκτελέσεις με συγκεκριμένες πολιτικές. Χαρακτηριστικό παράδειγμα, είναι το γεγονός ότι για το ίδιο set εφαρμογών η Space-Sharing πολιτική στην μία πλατφόρμα εκτέλεσης δίνει πολύ καλύτερα αποτελέσματα από το Linux Scheduler ενώ στην άλλη τα αποτελέσματα είναι χειρότερα από αυτά του Linux.

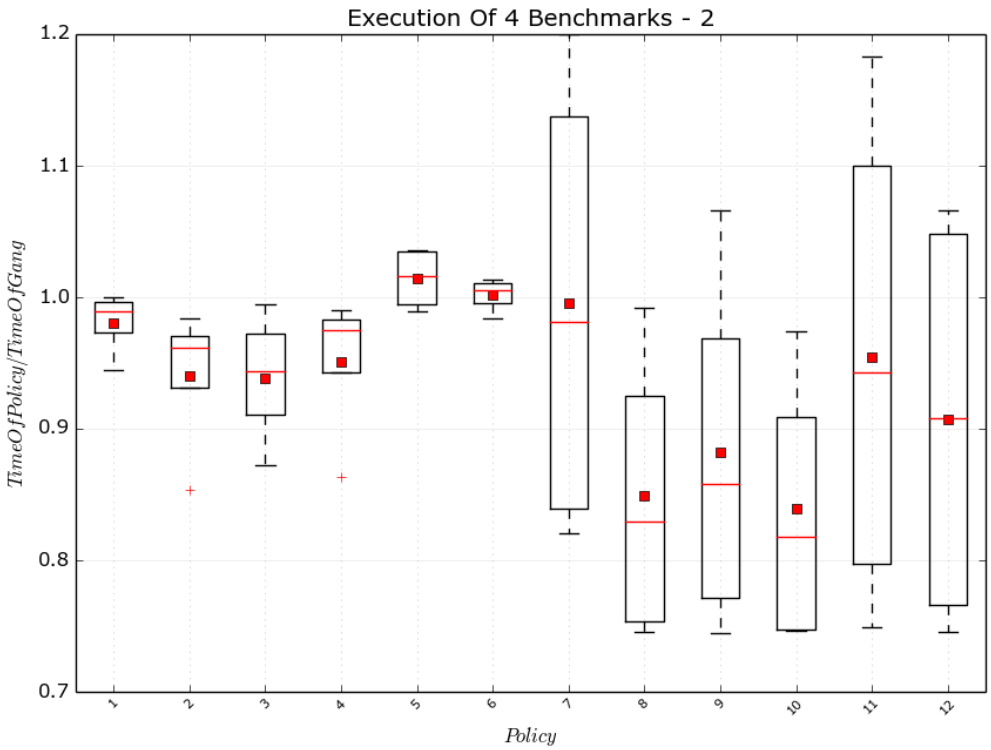
Το γενικότερο συμπέρασμα που προκύπτει από τα πιο πάνω, είναι ότι μπορούμε να έχουμε όφελος εφαρμόζοντας μία από τις καινούργιες μεθόδους, εάν πρώτα γίνει σωστή επιλογή των εφαρμογών που θα συνυπάρχουν και θα μοιράζονται τους πόρους του συστήματος μέσα στο ίδιο χρονικό διάστημα, με βάση τα χαρακτηριστικά που παρουσιάζουν στην εκάστοτε αρχιτεκτονική. Λανθασμένοι συνδυασμοί μπορούν να επιβαρύνουν τόσο τις εφαρμογές ξεχωριστά, όσο και τη συνολική εικόνα, σε μεγάλο βαθμό.

4.3 *Πειραματισμός στο time slot*

Σε μια προσπάθεια οι πολιτικές 6-11, που παρουσιάστηκαν προηγουμένως, να παρουσιάσουν πιο δίκαιη συμπεριφορά εκτελέστηκαν ξανά τα προηγούμενα πειράματα (Execution 1 & Execution 2) όμως αυτή τη φορά, στις εφαρμογές που προηγουμένως είχαν διαθέσιμους μόνο τέσσερις πυρήνες τώρα διπλασιάστηκε το time slot. Στις Εικόνες 4.3.1-4.3.2 φαίνεται η σύγκριση των προηγούμενων 6 πολιτικών που κρατούσαν σταθερό το κβάντο χρόνου (Policies 7-12) σε σχέση με την καινούργια αυτή υλοποίηση στους 6 δυνατούς συνδυασμούς (Policies 1-6). Η σύγκριση όπως και πριν γίνεται με βάση τους χρόνους εκτέλεσης των εφαρμογών σε κάθε πολιτική ως προς τους αντίστοιχους χρόνους εκτέλεσης με την Gang πολιτική.



Εικόνα 4.3.1 – Execution 1



Εικόνα 4.3.2 – Execution 2

Όπως φαίνεται από τα πιο πάνω αποτελέσματα, μπορεί οι νέες μέθοδοι να είναι πιο δίκαιοι αλλά επιβαρύνουν σημαντικά τη συνολική εικόνα της επίδοσης. Οι χρόνοι εκτέλεσης αυξάνονται σημαντικά στις περισσότερες περιπτώσεις. Ο τρόπος, λοιπόν, με τον οποίο θα διανέμουμε τον χρόνο δεν είναι κάτι δεδομένο. Αυτό που ιδανικά θα θέλαμε είναι ο χρόνος που θα δίνεται στις εφαρμογές να είναι τέτοιος ώστε να μην γίνεται άδικος ο χρονοδρομολογητής ως προς τη διαθεσιμότητα των πόρων αλλά και χωρίς να μειώνεται η επίδοση των εφαρμογών. Μια προσέγγιση αυτού του προβλήματος είναι η υλοποίηση αλγορίθμου που θα υπολογίζει δυναμικά τον κατάλληλο χρόνο που θα αφήσει κάθε εφαρμογή να εκτελεστεί, με βάση την επιβάρυνση που υπέστηκαν μέχρι στιγμής οι εφαρμογές. Όπως αναφέρεται και στη βιβλιογραφία [17], η επιβάρυνση μιας εφαρμογής μπορεί να υπολογιστεί λαμβάνοντας υπόψη παράγοντες όπως τον χρόνο που περιμένει κάθε εφαρμογή μέχρι να εκτελεστεί καθώς και το IPC των εφαρμογών όταν εκτελούνται μόνες τους σε σχέση με το IPC που έχουν κατά τη συνεκτέλεση.

5

Επίλογος

5.1 Σύνοψη και συμπεράσματα

Σε αυτήν τη διπλωματική εργασία είχαμε την ευκαιρία αρχικά να μελετήσουμε πολυνηματικές εφαρμογές από το Parsec. Οι εφαρμογές αυτές είναι ιδιαίτερα σημαντικές καθώς σχεδιάστηκαν με σκοπό να χρησιμοποιηθούν για μελέτη στη χρήση και στον σχεδιασμό των CMPs. Επίσης, εξετάστηκαν state-of-the-art χρονοδρομολογητές, όπως ο CFS του Linux και άλλοι, ενώ υλοποιήθηκαν δύο ακόμα μέθοδοι. Από τα πειράματα που εκτελέστηκαν είδαμε τα πλεονεκτήματα και τα μειονεκτήματα κάθε πολιτικής χρονοδρομολόγησης καθώς και το πώς τα χαρακτηριστικά των εφαρμογών μπορούν να καθορίσουν την επιλογή του πιο δίκαιου και αποδοτικού αλγορίθμου χρονοδρομολόγησης.

5.2 Μελλοντικές επεκτάσεις

Κλείνοντας, με βάση τα αποτελέσματα της παρούσας διπλωματικής, θα θέλαμε να εισηγηθούμε ορισμένες επεκτάσεις που θα μπορούσαν να υλοποιηθούν μελλοντικά.

Όσο αφορά τις εφαρμογές που χρησιμοποιήθηκαν, θα μπορούσαν να κατηγοριοποιηθούν με βάση κάποιο σχήμα που θα λαμβάνει υπόψη του την χρήση των κοινών πόρων, memory bandwidth και LLC.

Τέλος, για την ανάπτυξη ενός πιο αποτελεσματικού αλγορίθμου χρονοδρομολόγησης θα μπορούσαν να επεκταθούν οι νέες μέθοδοι χρονοδρομολόγησης που υλοποιήθηκαν με βάση μια κατηγοριοποίηση, προκειμένου να επιλέγονται με συγκεκριμένα κριτήρια οι εφαρμογές που θα εκτελεστούν ταυτόχρονα, μοιραζόμενες τους πυρήνες του συστήματος. Εκτός αυτού, όπως εξηγήθηκε και στο Κεφάλαιο 3, σε αυτές τις περιπτώσεις οι εφαρμογές θα επωφελούνταν εάν ρυθμιζόταν κατάλληλα το time slot το οποίο τους δίνεται, με βάση κάποιο παράγοντα που θα αντιπροσωπεύει τις συνέπειες που είχε η συνεκτέλεση στην επίδοσή τους.

Στην συνέχεια, θα θέλαμε να προτείνουμε έναν συγκεκριμένο αλγόριθμο που θα μπορούσε να υλοποιηθεί ως συνέχεια της διπλωματικής αυτής εργασίας.

5.2.1 Παραδοχές και Απαιτήσεις

Όπως προκύπτει από τα πιο πάνω, συνδυάζοντας κατάλληλα τις εφαρμογές που θα τρέχουν στο ίδιο χρονικό διάστημα μπορούμε να μειώσουμε τα φαινόμενα ανταγωνισμού και να βελτιώσουμε την επίδοσή τους. Κάνοντας την παραδοχή ότι θέλουμε να χρονοδρομολογήσουμε πολυνηματικές εφαρμογές, που χρησιμοποιούν τόσα νήματα όσοι και οι διαθέσιμοι πυρήνες του συστήματος και ότι τα νήματα των εφαρμογών αυτών πολύ πιθανόν να μοιράζονται δεδομένα ή να επικοινωνούν μεταξύ τους, θα θέλαμε να εφαρμόσουμε μία από τις νέες μεθόδους που υλοποιήθηκαν. Στις μεθόδους αυτές εξασφαλίζουμε ότι τα νήματα των εφαρμογών θα εκτελούνται μέσα στο ίδιο χρονικό διάστημα, είτε σε απομονωμένο περιβάλλον, είτε μαζί με τα νήματα ακόμα μίας εφαρμογής.

Από τις μετρήσεις που πήραμε, τις αντίστοιχες συγκρίσεις καθώς και από τις πληροφορίες που είχαμε για κάθε εφαρμογή, καταλήξαμε σε δύο χαρακτηριστικά των εφαρμογών τα οποία αν συνδυάσουμε κατάλληλα μπορούμε σε μεγάλο βαθμό να προβλέψουμε με ποια ζεύγη συνεκτελέσεων θα επιτυγχάναμε καλύτερα αποτελέσματα σε ένα πιθανό σενάριο εκτέλεσης. Το πρώτο χαρακτηριστικό είναι η δυνατότητα κλιμάκωσης των εφαρμογών στην εκάστοτε αρχιτεκτονική. Το δεύτερο είναι το ποσοστό χρησιμοποίησης των πόρων του συστήματος, συμπεριλαμβανομένου του memory bandwidth (off-chip traffic) και των διάφορων level της cache. Γνωρίζοντας αυτές τις πληροφορίες μπορούμε να αποφασίσουμε κατά πόσο θα έχουμε όφελος με το να επιτρέψουμε σε μια εφαρμογή να εκτελεστεί απομονωμένη στους διαθέσιμους πυρήνες. Στην αντίθετη περίπτωση που θα προτιμήσουμε να της δώσουμε τους μισούς πυρήνες, θα μπορούμε να εκτιμήσουμε με ποια εφαρμογή θα ήταν καλύτερα να εκτελεστεί παράλληλα.

Αρχικά μπορούμε να θεωρήσουμε ότι τα χαρακτηριστικά αυτά είναι γνωστά για κάθε εφαρμογή. Η επιλογή που θα κάνει ο αλγόριθμος θέλουμε να υπακούει στους εξής κανόνες: Πρώτον, θα αποφεύγει όσο γίνεται την ταυτόχρονη εκτέλεση εφαρμογών που παρουσιάζουν

μεγάλες απαιτήσεις στον ίδιο κοινό πόρο του συστήματος, δηλαδή την LLC ή το memory bandwidth. Δεύτερον, θα αφήνει μια εφαρμογή να εκτελεστεί μόνη της στους διαθέσιμους πυρήνες εφόσον παρουσιάζει καλή κλιμακωσιμότητα και προκαλεί έντονα φαινόμενα ανταγωνισμού και επομένως θα θέλαμε να αποφύγουμε την ταυτόχρονη εκτέλεσή της με κάποια άλλη εφαρμογή.

5.2.2 Κατηγοριοποίηση Εφαρμογών

Για το σκοπό αυτό, οι εφαρμογές μπορούν να ταξινομηθούν σε 4 κατηγορίες. Η πρώτη και δεύτερη κατηγορία θα περιέχουν εφαρμογές χωρίς μεγάλες απαιτήσεις σε bandwidth, με τη διαφορά ότι οι εφαρμογές στην πρώτη κατηγορία θα αξιοποιούν καλύτερα τις ιδιωτικές caches, ενώ στη δεύτερη θα χρησιμοποιούν περισσότερο την κοινή cache. Αντίστοιχα, η τρίτη και τέταρτη κατηγορία θα περιλαμβάνουν τις εφαρμογές που έχουν μεγάλες απαιτήσεις σε memory bandwidth. Στη τρίτη κατηγορία οι εφαρμογές θα αξιοποιούν καλύτερα τις ιδιωτικές caches, ενώ η τέταρτη κατηγορία θα περιέχει εφαρμογές για τις οποίες η χρήση της κοινής cache θα έχει πιο σημαντική θέση.

Η χρήση των πόρων του συστήματος μπορεί να εκτιμηθεί χρησιμοποιώντας performance counters κατά τη διάρκεια της εκτέλεσης των εφαρμογών. Όσο για το scalability, είναι ένα χαρακτηριστικό που θα μπορούσε να εκτιμηθεί συγκρίνοντας πως μεταβάλλεται η επίδοση μίας εφαρμογής όταν εκτελείται σε όλους τους διαθέσιμους πυρήνες και όταν εκτελείται στους μισούς πυρήνες.

5.2.3 Άπληστος Αλγόριθμος

Για την τήρηση του πρώτου κανόνα θα θέλαμε εφαρμογές από την πρώτη κατηγορία να εκτελεστούν παράλληλα με αυτές της τέταρτης κατηγορίας. Επίσης, θα θέλαμε να εκτελεστούν ταυτόχρονα εφαρμογές από την δεύτερη και τρίτη κατηγορία. Επομένως, θα μπορούσε να υλοποιηθεί ένας αλγόριθμος που θα προκαθορίζει τα ζεύγη των εφαρμογών που θα εκτελεστούν ταυτόχρονα. Οι εφαρμογές θα επιλέγονται από τέσσερις λίστες, όπου η κάθε λίστα θα αντιπροσωπεύει μία από τις τέσσερις κατηγορίες. Επιπλέον, κατά την επιλογή των εφαρμογών από μία κατηγορία, θέλουμε να δίνεται προτεραιότητα στις εφαρμογές με μέτριο ή κακό scalability. Επομένως, εφαρμογές με καλό scalability θα εισάγονται στο τέλος της λίστας ενώ οι υπόλοιπες θα εισάγονται στην αρχή της λίστας.

Προτείνουμε έτσι τον εξής αλγόριθμο που χωρίζεται σε τρία κύρια βήματα:

1. Το πρώτο πράγμα που θα θέλαμε να εξασφαλίσουμε είναι ότι θα εκτελεστούν παράλληλα όσες περισσότερες εφαρμογές γίνεται από την πρώτη ομάδα με

εφαρμογές από την τέταρτη. Στη συνέχεια, θα θέλαμε να δημιουργήσουμε όσα παραπάνω ζεύγη γίνεται με εφαρμογές από την δεύτερη και τρίτη ομάδα.

2. Μετά το πρώτο βήματα θα έχουν περισσέψει εφαρμογές από 2 κατηγορίες.
    a. Αν περίσσεψαν εφαρμογές από την πρώτη ομάδα, θα τις βάλουμε να εκτελεστούν παράλληλα με εφαρμογές από την άλλη κατηγορία (δεύτερη ή τρίτη). Εάν ακόμα περισσεύουν εφαρμογές από την πρώτη ομάδα θα εκτελεστούν σε ζεύγη μεταξύ τους.
    b. Αν μετά το πρώτο βήμα περίσσεψαν εφαρμογές από την δεύτερη και τέταρτη κατηγορία ή την τρίτη και τέταρτη κατηγορία, θα βάλουμε να εκτελεστούν παράλληλα εφαρμογές χωρίς καλό scalability, από όποια κατηγορία και αν προέρχονται. Τελικά θα περισσέψουν οι εφαρμογές με καλό scalability που θα εκτελεστούν απομονωμένες.
3. Μετά από το βήμα 2.α. μπορεί να έχουν περισσέψει εφαρμογές από την δεύτερη ή τρίτη κατηγορία. Τότε, θα κάνουμε ότι και στο βήμα 2.β. Δηλαδή, θα εκτελεστούν ταυτόχρονα εφαρμογές χωρίς καλό scalability και απομονωμένες αυτές με καλό scalability.

Ο αλγόριθμος αυτός με την κατάλληλη υλοποίηση μπορεί να βρει σε χρόνο O(n), όπου n είναι το πλήθος των εφαρμογών, πως θα εκτελεστούν οι εφαρμογές

6

Βιβλιογραφία

[1] The Linux kernel Documentation, CFS Scheduler. Available from: https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt

[2] Wikipedia, “Gang Scheduling”. Available from: https://en.wikipedia.org/wiki/Gang_scheduling

[3] Xie, Yuejian, and Gabriel Loh. "Dynamic classification of program memory behaviors in CMPs." In the 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects. 2008.

[4] Lin, Jiang, Qingda Lu, Xiaoning Ding, Zhao Zhang, Xiaodong Zhang, and P. Sadayappan. "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems." In High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on, pp. 367-378. IEEE, 2008.

[5] Jaleel, Aamer, Hashem H. Najaf-Abadi, Samantika Subramaniam, Simon C. Steely, and Joel Emer. "Cruise: cache replacement and utility-aware scheduling." In ACM SIGARCH Computer Architecture News, vol. 40, no. 1, pp. 249-260. ACM, 2012.

[6] Blagodurov, Sergey, Sergey Zhuravlev, and Alexandra Fedorova. "Contention-aware scheduling on multicore systems." ACM Transactions on Computer

Systems (TOCS) 28, no. 4 (2010): 8.

[7] Merkel, Andreas, Jan Stoess, and Frank Bellosa. "Resource-conscious scheduling for energy efficiency on multicore processors." In Proceedings of the 5th European conference on Computer systems, pp. 153-166. ACM, 2010.

[8] Tang, Lingjia, Jason Mars, and Mary Lou Soffa. "Contentiousness vs. sensitivity: improving contention aware runtime systems on multicore architectures." In Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, pp. 12-21. ACM, 2011.

[9] Bhadauria, Major, and Sally A. McKee. "An approach to resource-aware co-scheduling for cmps." In Proceedings of the 24th ACM International Conference on Supercomputing, pp. 189-199. ACM, 2010.

[10] K. Kumar Pusukuri, R. Gupta, and L. N. Bhuyan, "ADAPT: A framework for coscheduling multithreaded programs," ACM Trans. Archit. Code Optim., vol. 9, pp. 45:1–45:24, Jan. 2013.

[11] Haritatos, Alexandros-Herodotos, Georgios Goumas, Nikos Anastopoulos, Konstantinos Nikas, Kornilios Kourtis, and Nectarios Koziris. "LCA: a memory link and cache-aware co-scheduling approach for CMPs." In Proceedings of the 23rd international conference on Parallel architectures and compilation, pp. 469-470. ACM, 2014.

[12] PARSEC, "Download PARSEC 3.0", 2014. Available from: http://parsec.cs.princeton.edu/download.htm#parsec

[13] PARSEC, "Overview PARSEC", 2014. Available from: http://parsec.cs.princeton.edu/overview.htm

[14] Christian Bienia. Benchmarking Modern Multiprocessors. Ph.D. Thesis. Princeton University, January 2011. Available from: http://parsec.cs.princeton.edu/publications/bienia11benchmarking.pdf

[15] Christian Bienia and Kai Li. Fidelity and Scaling of the PARSEC Benchmark Inputs. In Proceedings of the IEEE International Symposium on Workload Characterization, December 2010. Available from: http://parsec.cs.princeton.edu/publications/bienia10inputs.pdf

[16] Christian Bienia, Sanjeev Kumar, Jaswidner Pal Singh and Kai Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. Technical Report TR-811-08, Princeton University, January 2008. Available from: http://parsec.cs.princeton.edu/doc/parsec-report.pdf

[17] Major Bhadauria, Vince Weaver and Sally A. McKee. A Characterization of

the PARSEC Benchmark Suite for CMP Design. Technical Report CSL-TR-2008-1052, Cornell University, September 2008. Available from: http://parsec.cs.princeton.edu/doc/cornell-report.pdf

[18] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd International Symposium on Computer Architecture, June 1995. Available from: http://www.csd.uoc.gr/~hy527/papers/splash2_isca.pdf

[19] Christian Bienia, Sanjeev Kumar and Kai Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. In Proceedings of the IEEE International Symposium on Workload Characterization, September 2008. Available from: http://parsec.cs.princeton.edu/publications/bienia08comparison.pdf

[20] Alexandros-Herodotos Haritatos, Georgios Goumas, Konstantinos Nikas and Nectarios Koziris. Contention-aware scheduling policies for fairness and throughput.

[21] Ναταλία Χέριγκ. Performance analysis and optimization of modern applications on Chip Multiprocessor Architectures. Diploma thesis, School of Electrical and Computer Engineering, N.T.U.A., 2014.