Ατομική Διπλωματική Εργασία

Σχεδιασμός ενός Σύγχρονου Αλγόριθμου Αντικατάστασης Γραμμών στην Κρυφή Μνήμη Τελευταίου Επιπέδου

Φίλιππος Παπαφιλίππου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2016

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδιασμός ενός Σύγχρονου Αλγόριθμου Αντικατάστασης Γραμμών στην Κρυφή Μνήμη Τελευταίου Επιπέδου

Φίλιππος Παπαφιλίππου

Επιβλέπων Καθηγητής
Γιαννάκης Σαζεΐδης

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2016

Ευχαριστίες

Θα ήθελα να ευχαριστήσω ιδιαίτερα τον Επιβλέποντα Καθηγητή μου κ. Γιάννο Σαζεΐδη για την πολύτιμη καθοδήγησή του κατά τη συγγραφή της διπλωματικής αυτής εργασίας καθώς και που μου έδωσε την ευκαιρία να ασχοληθώ με ερευνητικά θέματα από τα πρώτα στάδια φοίτησής μου στο Πανεπιστήμιο Κύπρου ως προπτυχιακός φοιτητής.

Από την ενασχόλησή μου στο Xi Computer Architecture Research Group γνώρισα επίσης χαρισματικά άτομα που ασχολούνται με παρόμοια θέματα. Ευχαριστώ τα μέλη της ομάδας αυτής που μου μετέδωσαν τον ενθουσιασμό και κρίση για αυτή τη θεματολογία.

Επιπλέον θα ήθελα να ευχαριστήσω και όλους τους καθηγητές με τους οποίους είχα κάποια συνεργασία διότι μου αποδεικνύεται σχεδόν καθημερινά ότι το Τμήμα Πληροφορικής του Πανεπιστήμιο Κύπρου μου έχει δώσει ισχυρό υπόβαθρο σε πολλές πτυχές της Πληροφορικής.

Περίληψη

Η αύξηση της επίδοση της Κεντρικής Μονάδας Επεξεργασίας (CPU) είναι κάτι πολύ επιθυμητό και ένας τρόπος που μπορούμε να το επιτεύξουμε με σχετική ευκολία είναι με διάφορες "έξυπνες" μεθόδους όπως την βελτιστοποίηση των αλγορίθμων αντικατάστασης γραμμών της κρυφής μνήμης τελευταίου επιπέδου (LLC).

Ένας μοντέρνος μηχανισμός αντικατάστασης γραμμών της κρυφής μνήμης τελευταίου επιπέδου είναι ο DRRIP [1] τον οποίο θα προσπαθήσουμε να βελτιώσουμε εξερευνώντας διάφορες παραλλαγές του. Τις παραλλαγές αυτές ορίσαμε προσθέτοντας χαρακτηριστικά στον αλγόριθμο των policies του DRRIP αλλά και μελετώντας αρκετούς μηχανισμούς εναλλαγής μεταξύ των επιμέρους policies.

Αυτό που θα οδηγήσει στο στόχο είναι να προσθέσουμε επιπλέον προσαρμοστικότητα στον αλγόριθμο αλλά και να βελτιώσουμε τα επιμέρους policies από τα οποία αποτελείται, για να υποβοηθούνται όσο το δυνατό πιο πολλά benchmarks.

Αρχικά κάνουμε χαρακτηρισμό των Benchmarks για να δούμε πώς συμπεριφέρεται το καθένα στην LLC. Ακολούθως σχεδιάζουμε ένα Design Space από υποψήφια replacement policies και το αξιολογούμε χρησιμοποιώντας Funtional Simulations. Με τη χρήση Performance Modeling παίρνουμε προσεγγίσεις της μέσης επίδοσης και τέλος εκτελούμε Performance Simulations για να αξιολογήσουμε καλύτερα τις καλύτερες παραλλαγές του DRRIP που προκύπτουν.

Η πρόταση φέρει βελτίωση στην επίδοση περίπου 1.6% για τα SpecCPU 2006 αν αφαιρέσουμε τα 9 benchmarks που δεν επωφελούνται από την ύπαρξη μιας Last-Level Cache. Αυτό το πετυχαίνουμε με πολύ μικρές αλλαγές στο DRRIP και δεν αξιολογούμε ακόμα τα αποτελέσματα με τους μηχανισμούς για 4 policies που εισάγουμε και φαίνονται υποσχόμενοι από την αντίστοιχη ανάλυση.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

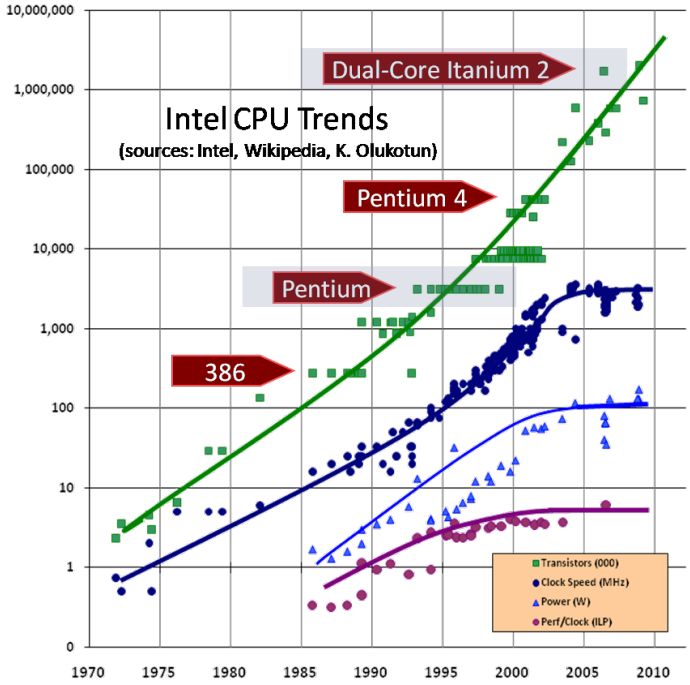
1.1 Ανάγκη για βελτιστοποιήσεις του παρόντος υλικού

Η εύρεση διαφόρων "έξυπνων" τεχνικών για αύξηση στην επίδοση των μικροεπεξεργαστών είναι κάτι το πολύ επιθυμητό στις μέρες μας, αφού πλέον δεν μπορούμε το καταφέρουμε πρακτικά απλώς με την αύξηση της συχνότητας του ρολογιού ή αξιοποίηση της αύξησης του αριθμού των τρανζίστορ για παραλληλοποίηση σε επίπεδο εντολών (ILP – Instruction-level parallelism).

Αυτό είναι αλήθεια διότι αν αυξήσουμε σημαντικά την συχνότητά του θα έχουμε περισσότερη παραγωγή θερμικής ενέργειας σε μικρότερη επιφάνεια καθιστώντας την απομάκρυνση της θερμότητας που παράγεται να είναι αδύνατο να επιτευχθεί με μονοφασική ψύξη. Δηλαδή θα χρειαζόμασταν μηχανισμούς όπως αυτούς που χρησιμοποιούνται σε ψυκτήρες αν συνεχίζαμε με αυτή τη νοοτροπία! Επιπλέον η αύξηση του αριθμού των τρανζίστορ, αντίθετα, δεν είναι ταυτόσημη με την αύξηση στην επίδοση διότι πρέπει το νέο υλικό που προστίθεται να αξιοποιείται αποδοτικά και στοχαστικά.

Αυτές οι δύο πιο πάνω συμβατικές τεχνικές ήταν επίκαιρες μέχρι περίπου το 2004 όταν οι μικροεπεξεργαστές ήταν ακόμα μονοπύρηνοι. Όπως φαίνεται και από το πιο κάτω γράφημα με τις τάσεις σχεδίασης των μικροεπεξεργαστών ο ρυθμός αύξησης της συχνότητας του ρολογιού ήταν σχεδόν σταθερός μέχρι σε ένα σημείο. Ο αριθμός των τρανζίστορ συνεχίζεται να αυξάνεται γραμμικά μέχρι και σήμερα ακολουθώντας το

νόμο του Moore, κατά τον οποίο ο αριθμός των τρανζίστορ θα διπλασιάζεται με την πάροδο δύο χρόνων, παρόλο που προβλέπεται ότι δεν θα ισχύει για πολύ ακόμα.



Σχήμα 1.1: Τάσεις μικροεπεξεργαστών Intel το χρονικό διάστημα 1970-2010 Πηγή: [4]

Ένα κλασσικό παράδειγμα στο οποίο διαφαίνεται η δυσκολία αυτή είναι οι επεξεργαστές Pentium 4 για τους οποίους η Intel εστιάστηκε στην αύξηση της συχνότητας και του βαθμού παραλληλισμού ILP. Με μέγιστο ρολόι τα 3.8 Ghz, TDP – Thermal Power Design 130 Watts και πολύ μεγάλη διασωλήνωση (pipeline) τα συστήματα που τα χρησιμοποιούσαν ήταν θορυβώδη και πολλές φορές είχαν προβληματικό σύστημα ψύξης [5].

Για τα σημερινά δεδομένα, μια η πιο διαδεδομένη τεχνική για την αξιοποίηση του μεγάλου αριθμού τρανζίστορ που μας επιτρέπει η τεχνολογίας είναι ο σχεδιασμός πολυπυρήνων. Οι επεξεργαστές με πολλαπλούς πυρήνες είναι ένα τεράστιο κεφάλαιο από μόνο του. Όμως, πέρα από τις διάφορες επιπλοκές που έχει στον σχεδιασμό και την χρήση τους, είναι αξιοσημείωτο για τους σκοπούς της διπλωματικής εργασίας το ότι οι πυρήνες σε ένα πολυπύρηνο επεξεργαστή μπορούν να θεωρηθούν ως αντίγραφα μιας μόνο υλοποίησης (με εξαίρεση τους ετερογενείς πολυπυρήνες) και υιοθετούν δομικά συστατικά ενός μονοπύρηνου επεξεργαστή.

Ένα από αυτά τα δομικά συστατικά στα οποία και θα εστιαστεί αυτή η μελέτη είναι η κρυφή μνήμη και συγκεκριμένα στη κρυφή μνήμη τελευταίου επιπέδου (Last-Level Cache). Συνοπτικά, η κρυφή μνήμη της Κεντρικής Μονάδας Επεξεργασίας είναι μία μικρή γρήγορη μνήμη που βρίσκεται στο εσωτερικό του επεξεργαστή και σε αυτή υπάρχουν τα δεδομένα που είναι πιθανό να χρησιμοποιηθούν πιο σύντομα παρακάμπτοντας συχνά ανάγκη για επιπλέον προσβάσεις στην κύρια μνήμη. Το όφελος ύπαρξης κρυφών μνήμων είναι τεράστιο και είναι ο λόγος που υπάρχει σε όλους τους μικροεπεξεργαστές γενικής χρήσης σήμερα.

Το ζήτημα είναι να βρεθεί ένας φτηνός σε υλικό και πολυπλοκότητα αλγόριθμος που να προβλέπει ποια είναι τα πιο σημαντικά δεδομένα που πρέπει να παραμείνουν στην κρυφή μνήμη έτσι ώστε να αυξηθεί η επίδοση το μέγιστο δυνατό. Διάφορες προσεγγίσεις το πλησιάζουν με την αντικατάσταση των λιγότερο πιθανό για επαναχρησιμοποίηση μπλοκ (ή γραμμών) δεδομένων γι' αυτό και οι αλγόριθμοι αυτοί ονομάζονται (Block) Replacement Policies. Ο κάθε ένας από αυτούς τους αλγόριθμους έχει δικά του κριτήρια για αντικατάσταση.

Υπάρχει αρκετό ερευνητικό έργο όπως τα [1,6,9,10,16,17,18,20,21] τα οποία προσπαθούν να επιτεύξουν αυτό. Το καθένα από αυτά όμως το πετυχαίνει σε διαφορετικό βαθμό και με διαφορετική επιπρόσθετη δυσκολία στην υλοποίηση. Καθώς δεν υπάρχει τέλεια ισορροπία μεταξύ αυτών, τέτοιου είδους προσπάθειες είναι πολυάριθμες και προσπαθούν γενικά βέλτιστη επίδοση με όσο το δυνατό λιγότερο επιπρόσθετο hardware και λογική.

1.2 Σκοπός της Διπλωματικής Εργασίας

Θα επικεντρωθούμε στην μελέτη, αξιολόγηση και βελτιστοποίηση διαφόρων αλγορίθμων Replacement Policy για την Last-Level Cache μέσω προσομοιώσεων και σχεδιασμού μοντέλων για εξεύρεση των καλύτερων χαρακτηριστικών από ένα ευρύ φάσμα διαφορετικών Replacemnt Policies.

Τα Replacement Policies που θα μελετηθούν βασίζονται σε σύγχρονους αλγόριθμους όπως το DRRIP ("High Performance Cache Replacement Using Re-Reference Interval

Prediction (RRIP)", Aamer J. et al [1]). Το αποτέλεσμα της εργασίας θα είναι η πρόταση ενός νέου αλγόριθμου LLC block replacement policies και σχεδίαση υλοποίησης με χαμηλό hardware overhead.

1.3 Συνεισφορά της Διπλωματικής Εργασίας

Το μεγαλύτερο εμπόδιο στην αξιολόγηση των υποψήφιων Replacement Policies είναι ο τεράστιος χρόνος που θα χρειαζόταν για να τις προσομοιώσουμε όλες σε εκτέλεση ολόκληρων προγραμμάτων. Στη μελέτη αυτή εφαρμόστηκαν διάφορες τεχνικές για την ελαχιστοποίηση του χρόνου αυτού και είναι και αυτή μία από τις κύριες συνεισφορές της προσπάθειας, να παρουσιαστεί δηλαδή μια ολοκληρωμένη μεθοδολογία επιλογής καλύτερων παραμέτρων για σχεδιασμό LLC block replacement policy έχοντας τεράστιο βαθμό ελευθερίας, που θα μπορεί να αξιοποιηθεί και για ευρύτερη χρήση.

Μια άλλη συνεισφορά θα μπορούσε να θεωρηθεί οι επιπρόσθετες παραμέτρους που εισάγουμε στο Design Space των replaement policies. Κάποιες από αυτές είναι υπάρχουσες, όπως το Frequency Promotion που απορρίπτεται από το paper [1] αλλά φαίνεται ότι βοηθά κάποια benchmarks στην ανάλυση του κεφαλαίου 8, και κάποιες άλλες είναι νέες αλλαγές στον αλγόριθμο του DRRIP[1].

Όσον αφορά τους μηχανισμούς με τους οποίους πετυχαίνεται δυναμική εναλλαγή μεταξύ πολλών replacement policies μελετούμε διάφορες παραλλαγές στην εξερεύνησή μας αλλά και εισάγουμε δύο νέους αντίστοιχους μηχανισμούς που πιστεύουμε ότι μπορούν να πετύχουν αποτελεσματικά εναλλαγή μεταξύ περισσότερων από 2 policies.

Η πρόταση προς το παρόν φέρει βελτίωση στην επίδοση περίπου 0.85% για όλα τα benchmarks και 1.6% αν αφαιρέσουμε τα benchmarks που δεν επωφελούνται από την ύπαρξη μιας Last-Level Cache. Αυτό το πετυχαίνουμε με πολύ μικρές αλλαγές στο DRRIP και δεν αξιολογούμε ακόμα τα αποτελέσματα με τους μηχανισμούς για 4 policies που φαίνονται υποσχόμενοι.

Κεφάλαιο 2

Υπόβαθρο

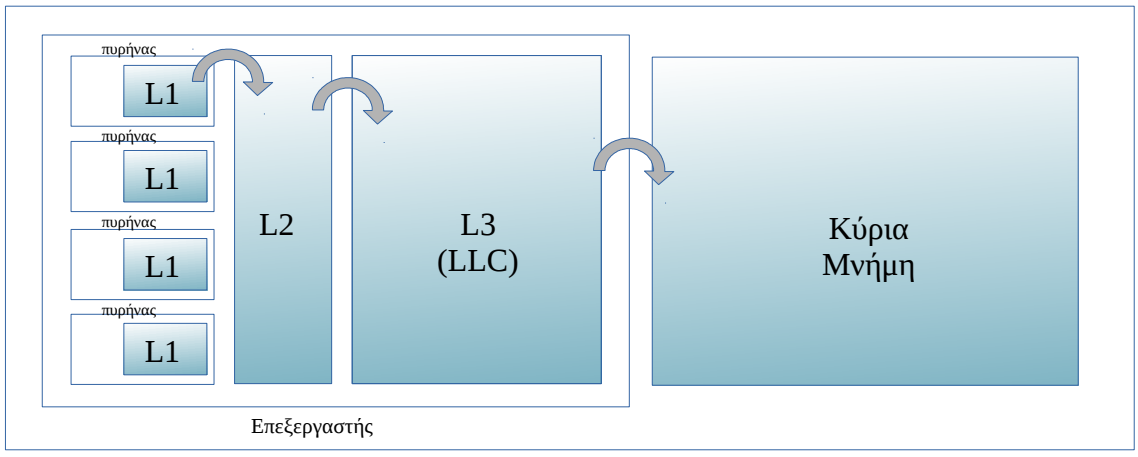
2.1 Ιεραρχία Μνήμης

Τα υπολογιστικά συστήματα διαχωρίζουν τους αποθηκευτικούς τους χώρους σε διάφορα τμήματα με διαφορετικά χαρακτηριστικά και χωρητικότητες. Οι αποθηκευτικοί χώροι παρουσιάζονται εντός μιας ιεραρχίας, της ιεραρχίας μνήμης. Το κάθε επίπεδο είναι πιο γρήγορο και πιο μικρής χωρητικότητας από το επόμενο. Όταν κάτι δεν υπάρχει σε ένα επίπεδο η πληροφορία αναζητείται στο επόμενο. Σε συνδυασμό με την χρήση αλγορίθμων, η ιεραρχία μνήμης είναι αυτό που επιτρέπει στους υπολογιστές να κάνουν πράξεις σε nanoseconds και ταυτόχρονα να αποθηκεύουν πληροφορίες με όγκο Terabytes.

Για τους σκοπούς της διπλωματικής μας ενδιαφέρουν τα ακόλουθα διαδοχικά επίπεδα όπως φαίνονται στο πιο κάτω διάγραμμα. Ξεκινώντας από το τελευταίο έχουμε την κύρια μνήμη στην οποία υπάρχουν τα περισσότερα δεδομένα που χρειάζονται τα προγράμματα τα που τρέχουν συμπεριλαμβανομένων των text, data, stuck και heap της κάθε εφαρμογής. Η πρόσβαση στην μνήμη παίρνει σήμερα περίπου 100 κύκλους, είναι τεχνολογίας DRAM, χωρητικότητας 8 GB ανά DIMM και βρίσκεται εκτός του επεξεργαστή (ή είναι stacked).

Η βασική αρχή στην οποία βασίζεται η ύπαρξη των κρυφών μνημών είναι οι τοπικότητες. Υπάρχουν 2 ειδών τυπικότητες, η χρονική και η τοπική. Έχουμε χρονική τοπικότητα όταν ένα block δεδομένων χρησιμεύει ξανά μετά από μικρό χρονικό διάστημα. Έχουμε χωρική τοπικότητα όταν ένα block δεδομένων χρησιμεύει όταν χρησιμεύει και κάποιο άλλο με κοντινή διεύθυνση.

Ακολούθως έχουμε τα 2 ή 3 επίπεδα κρυφής μνήμης τα οποία είναι κύριος σκοπός τους είναι να συγκρατούν τα δεδομένα που θα χρησιμοποιηθούν το συντομότερο. Όταν χρησιμοποιείται αποδοτικά και η χωρητικότητά της είναι αρκετή, έχουμε ως αποτέλεσμα να δίνεται η ψευδαίσθηση ότι δεν υπάρχει και ότι η κύρια μνήμη έχει τις ταχύτητες της κρυφής αλλά με μεγάλη χωρητικότητα. Η κρυφή μνήμη έχει τεχνολογία SRAM (πιο ακριβή ανά μονάδα δεδομένων), χωρητικότητα από 16 KB μέχρι 12 MB ανάλογα με το επίπεδο και χρόνους πρόσβασης τάξεως μερικών κύκλων. Εκτός από την τεχνολογία της, το ότι βρίσκεται μέσα στον επεξεργαστή είναι ακόμα ένας λόγος που συντείνει στην μικρή καθυστέρηση για πρόσβαση.



Σχήμα 2.1: Ιεραρχία Μνήμης σε ένα τυπικό σύστημα με πολυπύρηνο επεξεργαστή

Η L3 cache είναι εκεί που γίνεται η περισσότερη προσπάθεια όσον αφορά την εύρεση νέων replacement policies. Αυτό αληθεύει διότι λόγω της απόστασής της από τους πυρήνες έχει πιο λίγες και αραιές προσβάσεις σε σχέση με τις L1 και L2. Επομένως η επιπρόσθετη καθυστέρηση από πιο πολύπλοκα replacement policies έχει μικρότερη σημασία όταν είναι στην L3. Έτσι στην L1 και L2 δεν βλέπουμε συχνά αλλαγές των replacement policies τους.

Ακολουθεί σύντομη περιγραφή των τεχνικών λεπτομερειών για την λειτουργία της κρυφής μνήμης τελευταίου επιπέδου.

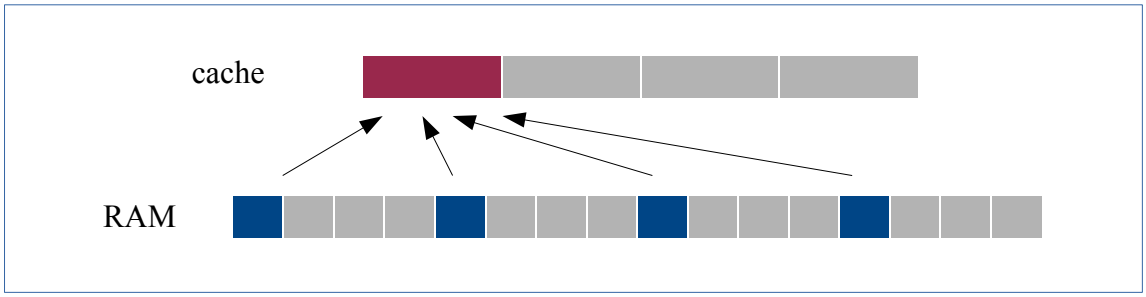
Για ένα αίτημα block δεδομένων στην LLC, όταν υπάρχει τότε λέμε ότι έχουμε hit. Αν όχι, τότε λέμε ότι έχουμε miss.

Το ποσοστό των misses σε σχέση με όλα τα accesses (δηλαδή το άθροισμα hits+misses) ονομάζεται miss rate. Χαμηλότερο miss rate ενώ είναι ένδειξη καλύτερης επίδοσης, δεν πρέπει να θεωρείται μέτρο διότι ο αριθμός των accesses μπορεί να αυξομειώνεται από εκτέλεση σε εκτέλεση σε συστήματα με out-of-order execution (ιδιότητα που προσφέρει ILP).

Ένα καλύτερο μέτρο επίδοσης είναι το MPKI (Misses per K instructions) που είναι ο μέσος αριθμός από misses ανά 1000 εντολές. Παρόλα αυτά το καλύτερο μέτρο είναι το IPC (Instructions per Cycle) διότι εκφράζει το πραγματικό βαθμό του ILP και κάθε πρόγραμμα έχει διαφορετική απόκριση στην επίδοση από την αυξομείωση των misses.

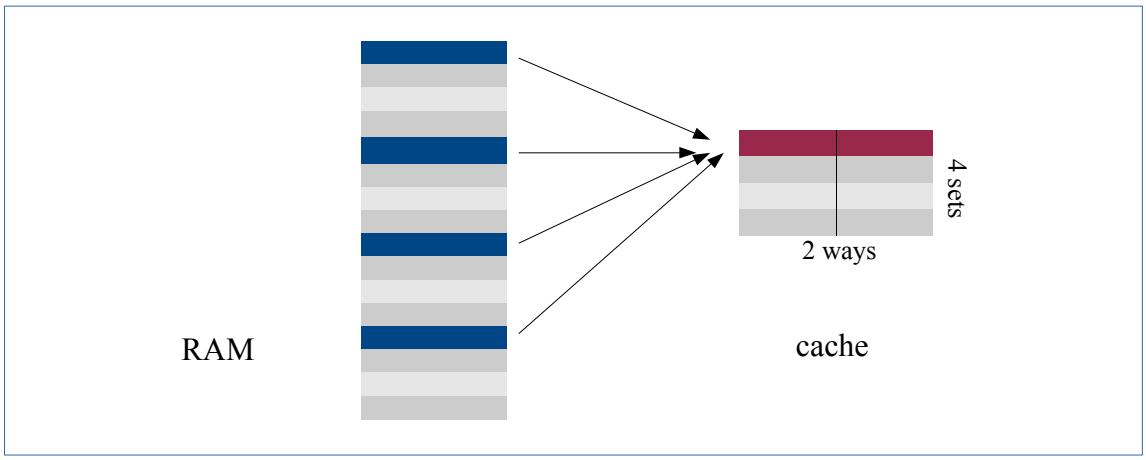
Το αντίστροφο του IPC είναι το CPI (Cycles per Instruction) και είναι χρήσιμο διότι όπως θα δούμε πιο μετά, τείνει πολλές φορές να είναι ανάλογο του MPKI διότι οι καθυστερήσεις από misses μπορούν απλουστευμένα να θεωρηθούν σαν επιπλέον κύκλοι ανά εντολή.

Σε μία Direct-Mapped Cache τα blocks μπαίνουν πάντα σε συγκεκριμένη θέση (διεύθυνση modulo χωρητικότητα) χωρίς να υπάρχει επιλογή, άρα ούτε και replacement policy.



Σχήμα 2.2: Direct-Mapped Cache

Σε μία Set-Associative Cache ένα block μπορεί να μπει σε ν θέσεις (ways). Η Set Associative Cache είναι χωρισμένη σε κ Sets στα οποία κάνουν map τα block με την φόρμουλα διεύθυνση modulo κ.



Σχήμα 2.3: Set-Associative Cache

Επιπλέον αξίζει να αναφέρουμε τις τρεις κατηγορίες misses, που κατηγοριοποιούν τους τρόπους με τους οποίους προκύπτουν τα misses στην LLC για την καλύτερη κατανόηση των ορίων των replacement policies:

- Compulsory misses: Συμβαίνουν όταν τα blocks αναζητούνται για πρώτη φορά, για παράδειγμα κατά την εκκίνηση του υπολογιστή
- Capacity misses: Συμβαίνουν όταν το μέγεθος της cache δεν αρκεί για το μέγεθος που απαιτεί η διεργασία
- Conflict misses: Συμβαίνουν όταν πολλαπλά blocks κάνουν map στο ίδιο set και ο βαθμός του associativity δεν είναι αρκετός

Τα misses που προκύπτουν από ένα μη βέλτιστο policy δεν συμπεριλαμβάνονται στα πιο πάνω.

2.2 Στατικοί αλγόριθμοι block replacement policy

Οι στατικοί αλγόριθμοι block replacement policy είναι αυτοί που χρησιμοποιούνται σε απλούς επεξεργαστές (για παράδειγμα RISC ή χαμηλής ισχύος), στις caches των χαμηλών επιπέδων (L1 και L2) ή στις LLC απαρχαιωμένων CISC επεξεργαστών.

Κύριο χαρακτηριστικό τους είναι ότι δεν έχουν προσαρμοστικότητα για διαφορετικά προγράμματα/ φόρτο εργασίας και ακολουθούν διαρκώς τα ίδια βήματα.

Μερικά παραδείγματα static block replacement policies:

- Random: Για κάθε miss αντικατάστησε ένα τυχαίο way στο set
- LRU (Least Recently Used): Για κάθε miss αντικατάστησε το block που αναφέρθηκε τελευταία φορά πριν από όλα τα άλλα.
  - Συχνά γίνεται αναφορά στο LRU recency stack που είναι η υλοποίηση της απαρίθμησης του κάθε way στο set από το πιο πρόσφατα μέχρι το πιο παλιά αναφερθέν.
- LFU (Least Frequenlt Used): Για κάθε miss αντικατάστησε το block που αναφέρθηκε τις πιο λίγες φορές σε σχέση με τα υπόλοιπα block στο ίδιο set για όρο χρόνο ήταν εκεί.
- FIFO (First in – First out): Για κάθε miss αντικατάστησε το γηραιότερο block
- PLRU (Pseudo-LRU): Διάφορες υλοποιήσεις που προσεγγίζουν το LRU με μικρότερο hardware overhead. Κάποτε δίνουν καλύτερα αποτελέσματα από το LRU.
- NRU (Not Recently Used): PLRU με πληροφορία 1 bit ανά block του κάθε set
  - Αλγόριθμος
    - Σε κάθε hit: Θέσε το PLRU bit του block με το hit να είναι ίσο με 1
    - Σε κάθε miss: Αντικατέστησε το πιο αριστερό block με PLRU bit 0 και θέσε το PLRU bit να είναι ίσο με 1. Αν όλα είχαν PLRU bit ίσο με 1 θέσε τα όλα να είναι 0 πριν την αντικατάσταση.
- LIP (LRU Insertion Policy)[9]: Χρησιμοποίηση της νοοτροπίας του LRU αλλά στην εισαγωγή νέων block θεώρησε ότι αυτό αναφέρθηκε τελευταία φορά πριν από όλα τα άλλα (LRU insertion position στο LRU recency stack), μέχρι να έχει hit.
- BIP (Bimodal Insertion Policy)[9]: Κυρίως χρησιμοποίησε LIP αλλά κάνε και χρήση του LRU με πιθανότητα ε (ή περιοδικά κάθε 1/ε των replacements)

Ο Optimal αλγόριθμος (“OPT”) του Belady είναι μια ειδική περίπτωση που χρησιμοποιείται σαν αξιολόγηση των replacement policies σε functional simulations, αλλά ο ίδιος δεν είναι replacement policy επεξεργαστή. Κάνει τις βέλτιστες επιλογές αντικατάστασης επιλέγοντας πάντα αυτό που να αναφερθεί στο μέλλον το αργότερο. Έτσι απαιτεί να ξέρουμε όλες της διευθύνσεις στις οποίες θα έχουμε πρόσβαση στο μέλλον και με πια σειρά θα αναφερθούν. Σε πραγματικά συστήματα αυτό φυσικά δεν έχει εφαρμογή, αφού είναι αδύνατο να ξέρουμε το μέλλον με τόση ακρίβεια σε αυτό το πλαίσιο.

Στο παράρτημα 2 υπάρχει κώδικας παράδειγμα που υλοποιεί μερικά από τα πιο πάνω.

2.3 Δυναμικοί αλγόριθμοι block replacement policy

Βλέποντας το σύνολο των πιο πάνω αλγορίθμων (εξαιρουμένου του OPT) εύκολα μπορεί να αναρωτηθεί κανείς ποιος έχει την καλύτερη επίδοση. Ενώ μπορεί να δοθεί απάντηση, για παράδειγμα ο LRU ήταν για αρκετό καιρό προτιμώμενος για LLC, στην πραγματικότητα διαφορετικός φόρτος εργασίας (workload) προτιμά διαφορετικό replacement policy [9]. Επιπλέον ρόλο παίζει και το μέγεθος της cache αλλά και η φάση στην οποία βρίσκεται ένα workload/πρόγραμμα.

Ένας από τους λόγους που υπάρχει διχασμός ανά workload στο ποιο replacement policy έχει την καλύτερη επίδοση είναι το ότι το κάθε ένα έχει διαφορετικές απαιτήσεις. Κυρίως διακρίνονται τέσσερεις κατηγορίες benchmark [10] με βάση τη συμπεριφορά τους στην ιεραρχία μνήμης:

- Recency-Friendly: Στην κατηγορία αυτή ανήκουν τα benchmarks τα οποία μπορούν άνετα να υποβοηθηθούν στην επίδοση από την ύπαρξη κρυφής μνήμης διότι ακριβώς κάνουν συχνή επαναχρησιμοποίηση (rereference) blocks. Άλλες ονομασίες που χρησιμοποιούνται ευρέως για αυτή την κατηγορία είναι “LRU Friendly” και “Cache Friendly”
- Thrashing: Στην κατηγορία αυτή ανήκουν τα benchmarks τα οποία δεν μπορούν άνετα να υποβοηθηθούν στην επίδοση από την ύπαρξη κρυφής μνήμης διότι ενώ έχουν rereferences, είναι αραιά λόγω πολλής πρόσβασης στην μνήμη. Αν η

κρυφή μνήμη ήταν ικανοποιητικά μεγάλη έτσι ώστε τα δεδομένα που χρειάζεται ανά πάσα στιγμή (working set) ένα πρόγραμμα χωρούν μέσα στην cache τότε δεν θα υπήρχε πρόβλημα.

- Streaming: Στην κατηγορία αυτή ανήκουν τα benchmarks τα οποία δεν μπορούν καθόλου να υποβοηθηθούν στην επίδοση από την ύπαρξη κρυφής μνήμης διότι δεν κάνουν rereferences με αποτέλεσμα να γεμίζουν τη κρυφή μνήμη με blocks που είναι άχρηστα πλέον (dead blocks). Στόχος είναι αυτά τα δεδομένα να μην παραμένουν για πολύ στην κρυφή μνήμη
- Mixed: Εδώ ανήκουν τα benchmarks τα οποία δεν είναι ξεκάθαρο τι είναι, είτε διότι έχουν φάσεις από διαφορετική προτίμηση, είτε διότι συνδυάζουν συμπεριφορές από τις άλλες τρεις κατηγορίες

Έτσι επινοήθηκαν διάφοροι αλγόριθμοι όπως ο DIP[9] (Dynamic Insertion Policy) με τον οποίο η κρυφή μνήμη αποκτά προσαρμοστικότητα (adaptiveness) στο workload καθώς μπορεί να επιλέγει δυναμικά κατά την διάρκεια της εκτέλεσης ένα από δυο διαθέσιμα replacement policies. Το DIP είναι ένας μηχανισμός με τον οποίο γίνεται προσέγγιση του καλύτερου replacement policy ανά πάσα στιγμή με αποτέλεσμα να έχει 21% μείωση στα MPKI σε σχέση με το απλό LRU (για τα πειράματα του [9]).

Τα 2 replacement policies που χρησιμοποιούνται στο DIP είναι το LRU (Least Recently Used) και το BIP (Bimodal Insertion Policy).

Το LRU είναι ένας αλγόριθμος που λειτουργεί πολύ καλά για Recency-Friendly (εξού και το όνομα LRU friendly) προγράμματα.

Το LIP που προαναφέραμε πιο πάνω χρησιμοποιεί το recency stack του LRU αλλά ένα νεοεισερχόμενο block εισάγεται με προτεραιότητα για αντικατάσταση να είναι ίση με την υψηλότερη. Ένα block παίρνει την χαμηλότερη προτεραιότητα όταν έχει hit, δηλαδή όταν έχει rereference ενώ βρίσκεται ακόμα μέσα στην κρυφή μνήμη. Το πλεονέκτημα του LIP είναι ότι παρέχει αντίσταση σε scanning προγράμματα διότι δίνει πολύ μικρή ευκαιρία σε ένα block να παραμείνει για πολύ μέσα στο set.

Το BIP μπορεί να θεωρηθεί εξέλιξη του LIP διότι προσθέτει σε αυτό ιδιότητες. Το BIP λειτουργεί όπως το LIP με την εξαίρεση ότι ανά τακτά χρονικά διαστήματα (με πιθανότητα ε=1/32) εισάγει το νεοεισερχόμενο block με προτεραιότητα για αντικατάσταση να είναι ίση με την χαμηλότερη. Δηλαδή μέσα μέσα λειτουργεί σαν LRU. Αυτό δίνει και μικρό πλεονέκτημα στα προγράμματα που είναι Recency-Friendly αλλά και παρέχει αντίσταση σε scanning προγράμματα ταυτόχρονα.

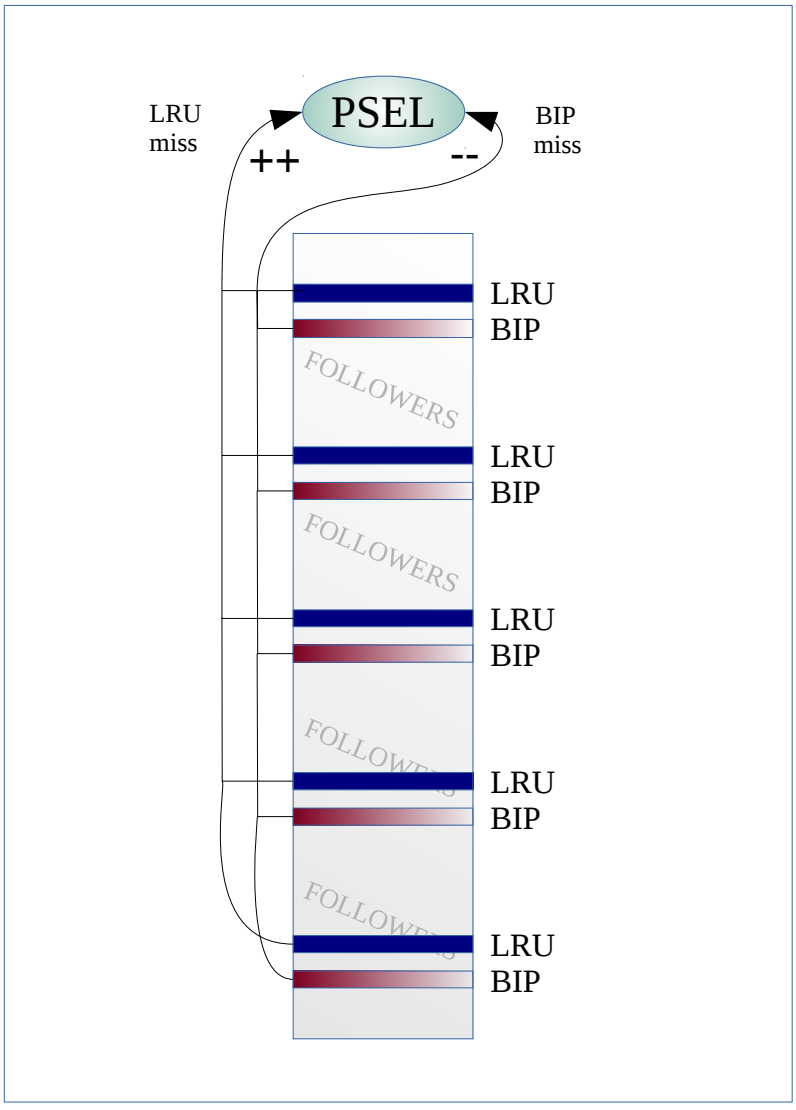
Ένα άλλο πλεονέκτημα του BIP σε σχέση με το LRU είναι ότι το LRU αποτυγχάνει να υποβοηθήσει την επίδοση για thrashing προγράμματα, ενώ αυτό επιτυγχάνει. Είναι αλήθεια διότι χρησιμοποιώντας LRU όταν υπάρχει κυκλική τάση για rereference (για παράδειγμα ένα data structure που καλείται συνέχεια σε ένα while loop) αν το working set (π.χ το μέγεθος του data structure) είναι μικρότερο από το μέγεθος της LLC, τότε επιτυγχάνει, διαφορετικά δεν καταφέρνει να συγκρατήσει τίποτα στην cache. Για παράδειγμα 1 set με 8 ways στο οποίο κάνουν 9 διαδοχικά blocks επαναλαμβανόμενα, το LRU σε κάθε replacement απομακρύνει αυτό που θα αναφερθεί αμέσως μετά, καθιστώντας το επόμενο access να είναι miss. Αντίθετα, το BIP συγκρατεί ένα υποσύνολο του working set στην cache επιτυγχάνοντας καλύτερη επίδοση.

Ο μηχανισμός του DIP που είναι υπεύθυνος για την δυναμική επιλογή μεταξύ LRU και BIP ονομάζεται Set Dueling [9]. Κάνοντας δειγματοληψία χρησιμοποιώντας ένα υποσύνολο των sets παίρνει αποφάσεις για όλα τα υπόλοιπα set της cache. Τα sets που θα χρησιμοποιηθούν για monitors χωρίζονται σε δυο ομάδες, μια για κάθε policy που χρησιμοποιούν στατικά το policy της ομάδας τους. Η κάθε ομάδα ονομάζεται SDM (Set Dueling Monitor). Οι αποφάσεις γίνονται χρησιμοποιόντας ένα μετρητή 10 bits (για LLC με 1024 sets και 32 SDM sets ανά policy) και ονομάζεται PSEL (Policy Selector).

Για κάθε miss στα set με το replacement policy A, η τιμή του PSEL αυξάνεται κατά 1. Για κάθε miss στα set με το replacement policy B, η τιμή του PSEL μειώνεται κατά 1. Σε κάθε εισαγωγή νέου block στα υπόλοιπα sets (followers) το policy που θα χρησιμοποιηθεί αποφασίζεται από το αριστερότερο bit του PSEL. Αν ισούται με 0 τότε χρησιμοποιείται το policy A, διαφορετικά αν ισούται με 1 τότε χρησιμοποιείται το policy B.

Υπάρχουν διάφορες αρχές τοποθέτησης των SDMs μέσα στα set της cache. Ενδεικτικά αναφέρουμε τον αλγόριθμο για το DIP. (Για μια cache 1024 sets) θέσε το set 0 και για κάθε set με απόσταση 33, ξεκινώντας από το 0, να χρησιμοποιούν το policy A (LRU). Θέσε κάθε 31ο set από το 0 να χρησιμοποιεί το policy B (BIP). Άλλες κατανομές, για παράδειγμα η τυχαία, έχουν στόχο την καλύτερη δειγματοληψία. Επιπλέον παίζει ρόλο και ο αριθμός των SDM sets καθώς πολύ λίγα δεν δίνουν αντιπροσωπευτική στατιστική και μεγάλος αριθμός "μολύνει" την cache με επιλογές μη βέλτιστες για μεγάλο αριθμό από sets διότι μειώνεται ο αριθμός των δυναμικών sets (followers) [9].

Πιο κάτω βλέπουμε μια απεικόνιση του πως μοιάζει το Set Dueling για το DIP. Για κάθε miss που συμβαίνει στα SDMs ("leader sets") αναβαθμίζεται ανάλογα το PSEL.



Σχήμα 2.4: Μηχανισμός Set Dueling για το DIP στα LLC sets

2.4 Ο αλγόριθμος αντικατάστασης DRRIP

Ο αλγόριθμος DRRIP [1] (Dynamic Re-Reference Interval Prediction Replacement Policy) μπορεί να θεωρηθεί σαν αναθεωρημένη μορφή του DIP. Αυτό αληθεύει διότι το DRRIP υιοθετεί το μηχανισμό Set Dueling του DIP και βελτιώνει τα επιμέρους replacement policies τα οποία χρησιμοποιεί.

Το Set Dueling μοντέλλο του DRRIP επιλέγει μεταξύ δύο διαφορετικών policies. Του SRRIP (Static Re-Reference Interval Prediction) και του BRRIP (Bimodal Re-Reference Interval Prediction). Το κοινό χαρακτηριστικό αυτό των policies είναι ότι χρησιμοποιούν πληροφορία ενός μετρητή 2 bits ονόματι RRPV (Re-reference Prediction Value) ανά block που καθορίζει την προτεραιότητα για replacement. Τα ονόματα που δόθηκαν για τις 4 δυνατές τιμές των RRPVs είναι near immidiate, immidiate, far, distant από το 0 μέχρι το 3 αντίστοιχα για να δείξουν το συσχετισμό τους με την απόσταση στο μέλλον με την οποία θα γινεί ένα block rereference αν πάρει αυτή τη τιμή.

Το SRRIP είναι το αντίστοιχο του LRU στο DIP όμως παρέχει και scan resistance. Παρατίθεται ο σχετικός αλγόριθμος:

- Σε κάθε miss
  - Εν όσο δεν υπάρχουν RRPVs ίσα με 3 (“distant”) στο set
    - Αύξησε όλα τα RRPVs του set κατά 1
  - Αντικατέστησε το αριστερότερο block με RRPV ίσο με 3 (“distant”)
  - Θέσε το RRPV του καινούριου block να είναι ίσο με 2 (“far”)
- Σε κάθε hit
  - Θέσε το RRPV του αναφερθέντος block να είναι ίσο με 0 (“near immidiate”)

Ο λόγος που έχει scan resistance είναι ότι, όπως βλέπουμε και από τον αλγόριθμο, ένα block δεν τοποθετείται με μικρότερη προτεραιότητα (δηλαδή RRPV 0 /“near immidiate”) όπως θα έκανε το LRU και επομένως δίνεται λιγότερη ευκαιρία σε κάθε block να παραμείνει μέσα στην LLC κρυφή μνήμη.

Μια σύντομη παρατήρηση για τον αλγόριθμου είναι ότι δεν μπορεί να χρησιμοποιηθεί για υλοποίηση LRU. Αν αλλάξει το σημείο που βάζουμε το RRPV του καινούριου block να είναι ίσο με 2 (“far”) με το να είναι 0 (“near immidiate”), προκύπτει το NRU διότι λογικά ανά πάσα στιγμή τα RRPVs έχουν τις τιμές 0 ή 3 μόνο. Έτσι είναι σαν να έχουμε 1-bit υλοποίηση PLRU. Για 2- bit υλοποιήσεις PLRU υπάρχει το Tree-PLRU που δεν θα μελετήσουμε εδώ.

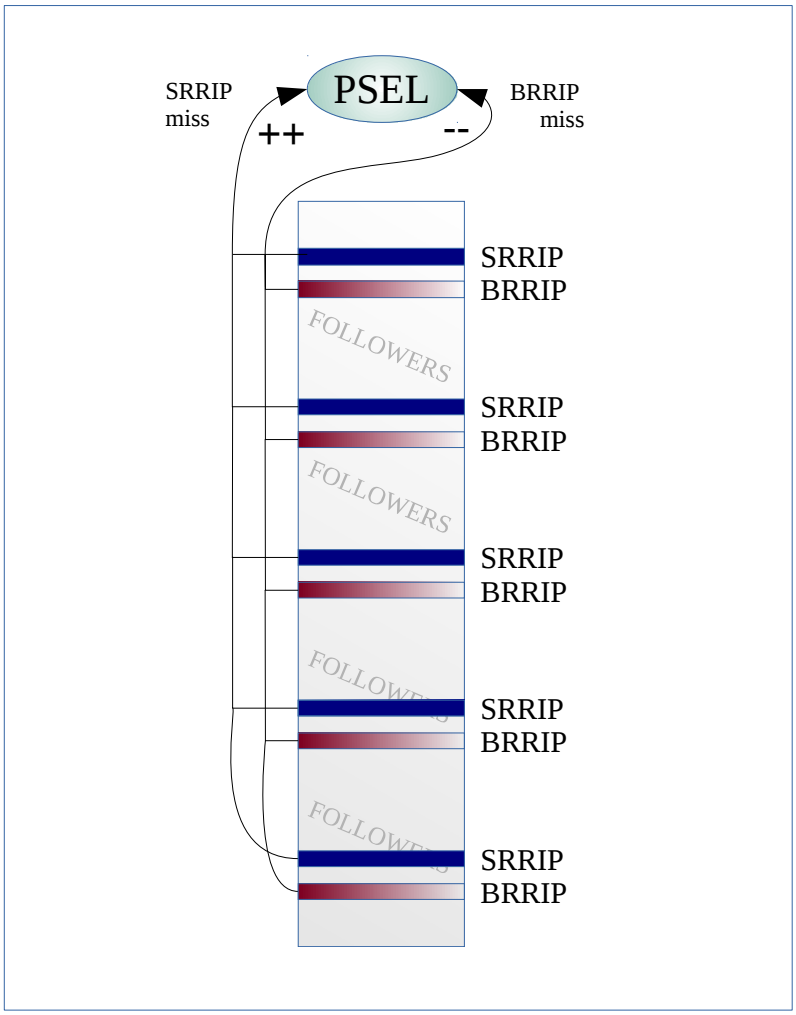
Το BRRIP είναι το αντίστοιχο του BIP στο DIP. Παρατίθεται ο σχετικός αλγόριθμος:

- Σε κάθε miss
  - Εν όσο δεν υπάρχουν RRPVs ίσα με 3 (“distant”) στο set
    - Αύξησε όλα τα RRPVs του set κατά 1
  - Αντικατέστησε το αριστερότερο block με RRPV ίσο με 3 (“distant”)
  - Θέσε το RRPV του καινούριου block να είναι ίσο με 3 (“far”) με πιθανότητα 1-ε και ίσο με 2 με πιθανότητα ε (1/32)
- Σε κάθε hit
  - Θέσε το RRPV του αναφερθέντος block να είναι ίσο με 0 (“near immidiate”)

Ο λόγος που παρέχει scan resistance είναι διότι όπως το SRRIP δεν θέτει νεοεισερχόμενα blocks να έχουν μικρή τιμή RRPV και να εκτοπίζουν τα υπάρχοντα blocks ευκολότερα. Ο λόγος που παρέχει thrashing resistance είναι διότι τίνει να συγκρατεί ένα τμήμα του working set σε thrashing προγράμματα διότι δεν είναι LRU. Όμως δεν είναι κατάλληλο για Recency-Friendly προγράμματα αφού το πιο πιθανό είναι τα blocks να εισέλθουν με RRPV 3 (“far”), δηλαδή αν δεν έχουν ένα διαδοχικό access για το set να αντικατασταθούν από το επόμενο ακόμα και αν ευτό έχει μικρή απόσταση επαναχρησιμοποίησης (reuse distance).

Για λόγους ευκολότερης υλοποίησης στο υλικό η τυχαιότητα στο BRRIP μπορεί να αντικατασταθεί από περιοδικότητα έτσι ώστε μόλις ένας μετρητής έρχεται στην κατάσταση 31 να επανέρχεται στη θέση 0 και να παίρνει τιμή 2 το RRPV με αναλογία 1/32 σε σχέση με τα RRPVs ίσα με 3.

Παρατίθεται το αντίστοιχο σχεδιάγραμμα για τον μηχανισμό Set Dueling για το DRRIP.



Σχήμα 2.5: Μηχανισμός Set Dueling για το DRRIP στα LLC sets

Και το DIP, αλλά και το DRRIP είναι σχετικά πολύ φθηνά replacement policies στο hardware. Αυτό ισχύει λόγο της υλοποίησης του Set Dueling που αποτρέπει την χρήση εξωτερικών sets χωρίς δεδομένα (tag arrays) που έχουν την ίδια λειτουργία με τα SDMs (ATDs [9]). Ένας άλλος λόγος που το DRRIP είναι σχετικά φθηνό στο hardware είναι η χαμηλή πολυπλοκότητα του αλγορίθμου στον χώρο (2 bits ανά block) και τις μεταβάσεις καταστάσεων.

Το DRRIP ή κάποιος παρόμοιος μηχανισμός έχει παρατηρηθεί ότι πρέπει να υπάρχει στους αρκετά σύγχρονους επεξεργαστές της Intel γενεάς “Ivy Bridge” [11].

2.5 Μηχανισμοί δυναμικής εναλλαγής policies

Υπάρχουν διάφοροι μηχανισμοί εναλλαγής μεταξύ πολλαπλών policies. Ένας από αυτούς είναι αυτός που μόλις είδαμε, το Set Dueling του DIP που είναι εξέλιξη άλλων παλαιότερων μηχανισμών. Το βασικό χαρακτηριστικό όλων είναι η παρακολούθηση πώς συμπεριφέρεται τα policies σε συγκεκριμένα sets και η δυναμική επιλογή του βέλτιστου policy για όλα τα sets της LLC (ή μιας μεγάλης πλειοψηφίας).

Αρχικά έχουμε το “TSEL” και “TSEL-Global” [17] τα οποία χρησιμοποιούν 2 εξωτερικά sets για κάθε set τις cache. Στο ένα χρησιμοποιείται το policies A και στο άλλο το policy B. Για κάθε συνδυασμό από hit/miss ο αντίστοιχος counter μειώνεται κατά ένα. Για κάθε συνδυασμό από miss/hit ο αντίστοιχος counter αυξάνεται κατά ένα. Το αριστερότερο bit του counter (MSB) υποδηλώνει πιο policy θα χρησιμοποιείται για τη συγκεκριμένη χρονική στιγμή. Η διαφορά του “TSEL-Global” από το “TSEL” είναι ότι όλα συνδέονται με ένα counter και οι αποφάσεις δεν γίνονται ανεξάρτητα per-set.

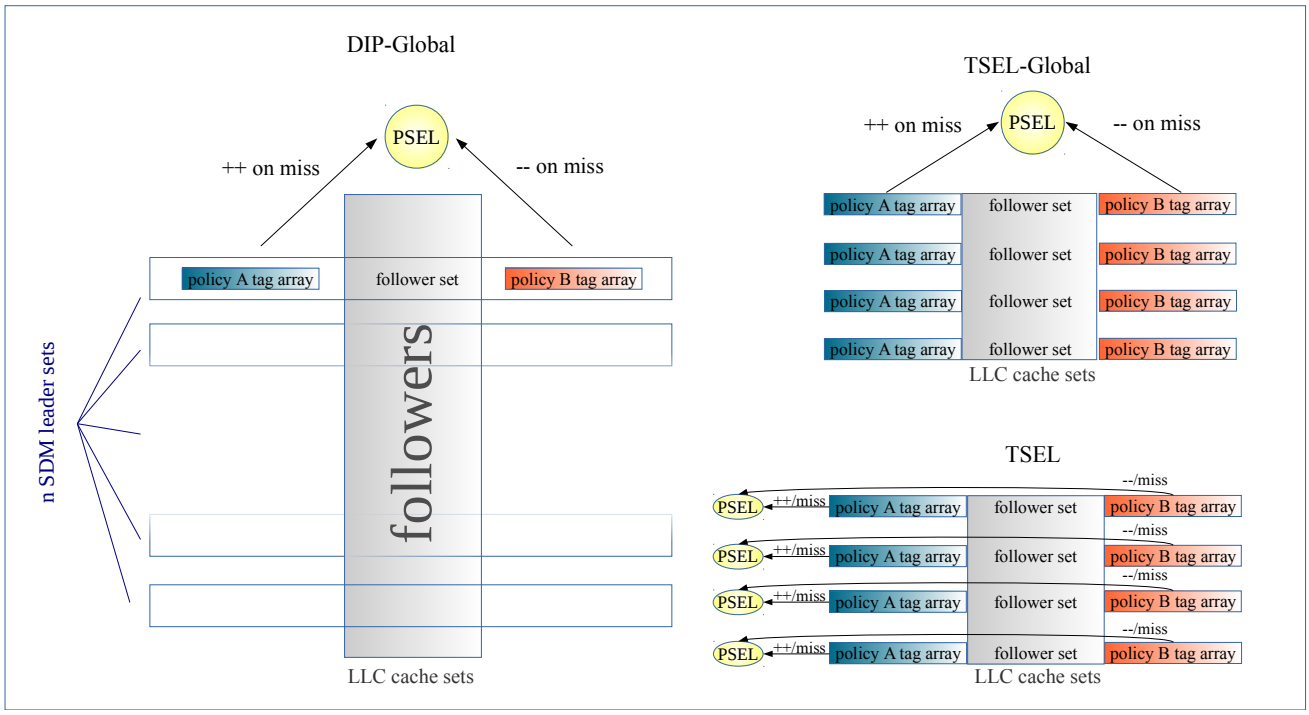
Ακολούθως έχουμε το “DIP-Global” το οποίο μειώνει το κόστος υλοποίησης του “TSEL-Global” κατά πολύ χρησιμοποιώντας δειγματοληψία. Δηλαδή τα εξωτερικά sets είναι πολύ πιο λίγα.

Στο σχήμα 2.6 υπάρχουν απεικονίσεις των “Set Dueling”, “TSEL” και “TSEL-Global”.

Επίσης έχουμε το Set Dueling του DRRIP[1] και DIP[9] (Σχήμα 2.5) που πετυχαίνει σχεδόν την επίδοση του “DIP-Global” αλλά χωρίς να υπάρχουν εξωτερικά set ως monitors (SDMs). Τα SDMs βρίσκονται εντός της cache και είναι ένα υποσύνολο των sets της cache.

Όσον αφορά την εναλλαγή μεταξύ περισσότερων policies από 2, πάλι υπάρχει αρκετό έργο [18]. Για παράδειγμα στο [18] χρησιμοποιείται Decision Tree Analysis (DTA) και turnaments για να παρθούν αποφάσεις σχετικά με το πιο replacement policy θα χρησιμοποιείται και πια policies θα χρησιμοποιούνται στα (εσωτερικά) Leader sets. Αυτός ο μηχανισμός αλλά και άλλοι παρόμοιοι [18] δεν έχουν εφαρμογή στην εργασία αυτή για λόγους που θα δούμε αργότερα. Επιπλέον υπάρχει το κόστος εναλλαγής των

leader sets και η ανάγκη για εύρεση του κατάλληλου χρόνου (accesses) που θα διαρκούν τα turnaments.



Σχήμα 2.6: Απεικονίσεις των “Set Dueling”, “TSEL” και “TSEL-Global” για υποθετικές LLCs

Κεφάλαιο 3

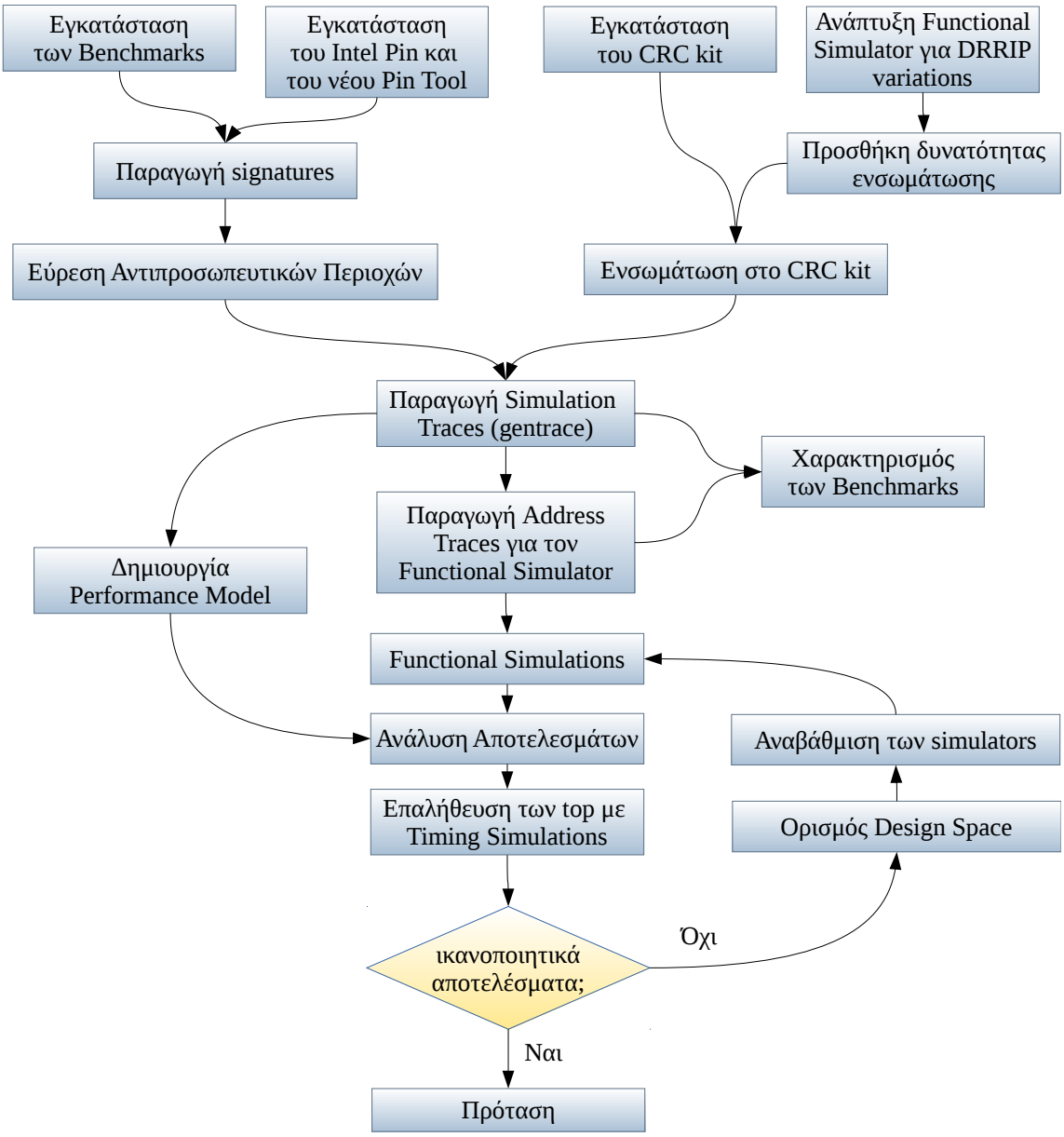
Μεθοδολογία

3.1 Σύνοψη της μεθοδολογίας

Στο πιο κάτω σχεδιάγραμμα (Σχήμα 3.1) περιγράφεται συνοπτικά η διαδικασία που ακολουθήσαμε από την αρχή μέχρι την αποκόμιση συμπερασμάτων. Τα τόξα δηλώνουν τις εξαρτήσεις μεταξύ των βημάτων.

Περιγραφή της μεθοδολογίας σε βήματα:

1 Προετοιμασία της σειράς από προγράμματα που θα χρησιμοποιηθούν για αξιολόγηση των Replacement Policies, το Spec2006 Benchmark Suite [7] από το οποίο πολλά από τα 29 Benchmarks κάνουν πολλές προσβάσεις στην μνήμη (memory intensive), όπως και θα παρατηρήσουμε. (κεφάλαιο 4)
2 Παραγωγή ταυτοτήτων (signatures) για ομάδες εντολών καθ' όλη τη διάρκεια εκτέλεσης των Benhmarks χρησιμοποιώντας instrumentation από εργαλείο του Intel Pin Toolkit [3]. (κεφάλαιο 4)
3 Εύρεση των αντιπροσωπευτικών περιοχών εκτέλεσης (Representative Regions) χρησιμοποιώντας γραφήματα από τις πιο πάνω πληροφορίες. (κεφάλαιο 4)
4 Χρησιμοποίηση του προσομοιωτή CRC kit [8,2] για παραγωγή traces 250M εντολών από τα Representative Regions για κάθε benchmark με προεπιλεγμένο το DRRIP (και υλοποίηση του) ως replacement policy. (κεφάλαιο 4)



Σχήμα 3.1: Εξαρτήσεις μεταξύ των βημάτων της μεθοδολογίας

5 Ανάπτυξη κώδικα για το επιθυμητό εύρος από Replacement Policies ο οποίος μπορεί να τρέχει σαν κομμάτι του CRC kit (για timing siumulations) αλλά και ως αυτόνομος απλός LLC cache simulator (για functional siumulations). (κεφάλαιο 4)

6 Χαρακτηρισμός ιδιοτήτων του κάθε Benchmark από ένα σετ από πειράματα χρησιμοποιώντας το CRC kit από τα παραγόμενα traces (timing siumulations) αλλά και τον functional simulator (κεφάλαιο 5)

7 Επιλογή ενός υποσυνόλου του φάσματος επιλογών (Design Space) των Replacement Policies και εκτέλεσή τους με το CRC kit για κάθε trace έτσι ώστε να παράγουμε ένα μοντέλο (Performance Model) για κάθε Benchmark. (κεφάλαιο 7)

8 Παραγωγή αρχείων address traces που έχουν μόνο της προσβάσεις στην LLC από το παραλλαγμένο CRC kit ώστε να χρησιμοποιεί το DRRIP Replacement Policy. Αυτά είναι σημαντικά διότι θα αποτελέσουν την είσοδο της αυτόνομης εκδοχής του κώδικά μας για πολλά, πιο απλά και γρηγορότερα πειράματα. (κεφάλαιο 4)

9 Εκτέλεση του αυτόνομου simulator (για functional siumulations) για κάθε address traces για κάθε Replacement Policy στο ολόκληρο το φάσμα επιλογών που μελετούμε. (κεφάλαιο 6)

(Εδώ επιλέξαμε ευρύ φάσμα επιλογών και γι' αυτό χρησιμοποιήσαμε τον υπερυπολογιστή Cy-Tera του Cyprus Instutute)

10 Αναλύσεις αποτελεσμάτων για αποφάσεις σχεδίασης και αξιολόγηση των νικητών Replacement Policies με timing siumulations από το CRC Kit (κεφάλαιο 8)

11 Εδώ μπορούμε να επιστρέψουμε στο δεύτερο πιο πάνω βήμα όταν θέλουμε να μελετήσουμε (ή κατευθείαν με timing timulations) πιο εξειδικευμένες επιλογές για Replacement Policies με βάση την παρούσα γνώση (κεφάλαιο 9)

12 Πρόταση υλοποίησης (Κεφάλαιο 8, 9)

3.2 Διαφορές από υπάρχουσες μεθοδολογίες

Η κοντινότερη μεθοδολογία με αυτή που θα χρησιμοποιήσουμε, αν και έχει αρκετές διαφορές είναι αυτή που περιγράφεται στο “Insertion and Promotion for Tree-Based PseudoLRU Last-Level Caches” του D. Jimenez [20].

Η κύρια διαφορά με το [20] είναι ότι βασιζόμαστε στο DRRIP αντί στο PLRU. Έτσι δεν χρειάζεται να υπάρχει κάποια δομή που αντιπροσωπεύει το LRU recency stack αφού θα χρησιμοποιούμε π.χ 2 bits (καθορισμένο μέγεθος) ανά block για να αντιπροσωπεύουν το Rereference Interval [1].

Μια άλλη βασική διαφορά είναι ότι εδώ παρουσιάζονται όλες οι λεπτομέρειες που δεν θα χωρούσαν σε ένα ακαδημαϊκό paper. Τα πιο πάνω βήματα αποτελούν την μεγάλη εικόνα για το τι πρέπει να κάνει κάποιος για να σχεδιάσει ένα replacement policy βασισμένο στο DRRIP όπως κάναμε εμείς, από την αρχή μέχρι το τέλος.

Επιπλέον, δεν χρησιμοποιούμε κάποιο έτοιμο πακέτο γενετικών αλγορίθμων αλλά βασιζόμαστε στον λιγότερο χρόνο των Functional Simulations για να καλύψουμε ένα ευρύτερο φάσμα replacement policies.

Οι μηχανισμοί που χρησιμοποιεί το [20] για εναλλαγή μεταξύ περισσότερων από 2 policies αναγράφονται στο [22] αλλά δεν βρίσκουν εφαρμογή εδώ διότι τα policies που χρησιμοποιούμε δεν περιγράφονται από διανύσματα αλλαγών κατάστασης μέσα στο LRU stack αλλά από πολλές ανεξάρτητες παραμέτρους.

Γι' αυτό στο κεφάλαιο 9 σχεδιάζουμε μηχανισμούς που θα μας βοηθήσουν να τοποθετήσουμε περισσότερα από 2 policies της μορφής που θα ορίσουμε και χωρίς την ανάγκη εναλλαγής policies στα SDMs (turnaments και adaptive leaders [22]).

Τέλος, παρόλο που ο πρώτος στόχος είναι η αύξηση στην επίδοση, αντίθετα με το [20] που μειώνει το hardware overhead, όσον αφορά τα αποτελέσματα για 2 policies, δεν αυξάνεται ο χώρος που χρειάζεται σε κάθε block σε σχέση το DRRIP. Έτσι δεν έχουμε το hardware overhead του να κουβαλούμε μαζί με τα blocks επιπρόσθετες ταυτότητες [10] ή διαφοροποίηση στο software για να το υποστηρίζει όπως εδώ [21].

Κεφάλαιο 4

Προσομοιώσεις

4.1 Functional Simulators και Timing Simulators

Ο τρόπος με τον οποίο αξιολογούνται τα υποψήφια LLC block replacement policies (αλλά και άλλα χαρακτηριστικά του επεξεργαστή) πριν να μπουν σε τελικό προϊόν είναι με τη χρήση προσομοιώσεων.

Το θετικό των προσομοιώσεων είναι ότι μπορούν να γίνουν προβλέψεις για πολλές διαφορετικές παραλλαγές του αντικειμένου που μελετάται και με μεγάλη ελευθερία στον σχεδιασμό. Για παράδειγμα, αν μιλούμε για προσομοιωτή cache στη γλώσσα C, όπως την περίπτωσή μας, δεν χρειάζεται να ορισθεί το υλικό για τις προσθέσεις, την εύρεση τυχαίων αριθμών κ.τ.λ. Ο προγραμματιστής εστιάζεται μόνο στο αντικείμενο που θέλει να μελετήσει. Μπορεί να κάνει πολλές αλλαγές στον κώδικα και να παραλληλίσει την εκτέλεση των πειραμάτων σε διάφορες μηχανές, χωρίς τους περιορισμούς του να προσομοιώνει σε πραγματικά συστήματα όπως την συστοιχία επιτόπια προγραμματιζόμενων πυλών (FPGAs).

Τα αρνητικά χαρακτηριστικά των προσομοιώσεων είναι ο πολύ μεγαλύτερος χρόνος εκτέλεσης από το πραγματικό σύστημα και η ανακρίβεια στα αποτελέσματα. Για το πρώτο η λύση είναι να τρέχουμε περιορισμένες προσομοιώσεις, όπως ένα κομμάτι της εκτέλεσης των προγραμμάτων, ή μικρότερα προγράμματα. Για το δεύτερο η λύση είναι

να τρέχουμε πιο αντιπροσωπευτικό φόρτο εργασίας, ή να γράψουμε ακριβέστερους προσομοιωτές, που και τα δύο πολλές φορές οδηγούν σε περισσότερο χρόνο εκτέλεσης. Έτσι η βέλτιστη ισορροπία μεταξύ χρόνου εκτέλεσης και ακρίβειας θα καθορίζεται με βάση το πόση ακρίβεια θεωρούμε ικανοποιητική, πόσο χρόνο διαθέτουμε και σε πόσους πόρους θα τρέξουμε τις προσομοιώσεις.

4.1.1 Functional Simulators

Οι Functional Simulators είναι οι προσομοιωτές με τους οποίους κάνουμε ανάλυση μόνο των λειτουργιών του αντικειμένου που μελετούμε. Ενώ πολλοί simulators για προσομοίωση ολόκληρου επεξεργαστή μπορούν να τρέξουν και ως Functional Simulators, η δυνατότητα αυτή δεν χρησιμοποιείται με κύριο στόχο την μέτρηση τις επίδοσης. Αυτό αληθεύει διότι το στοιχείο του χρόνου απουσιάζει στα Functional Simulations δηλαδή και οι καθυστερήσεις σε κάθε τμήμα του επεξεργαστή. Αυτός είναι και ο κύριος λόγος που είναι σχετικά γρήγορες προσομοιώσεις.

Στην δική μας περίπτωση ένας Functional Simulator θα είναι χρήσιμος για την μελέτη της συμπεριφοράς των αλγορίθμων replacement policies, που είναι το αντικείμενο που μελετούμε. Πιο συγκεκριμένα θα χρησιμέψει στην αποσφαλμάτωση του νέου κώδικα που θα γράψουμε για τα replacement policies και θα παίρνουμε μια ένδειξη για την επίδοση μέσω μετρικών της ίδιας της LLC, του αριθμού των misses ανά χίλιες εντολές (MPKI). Ακολούθως οι πληροφορίες αυτές θα χρησιμέψουν στην διάκριση ενός υποσυνόλου από replacement policies που έχουν το μικρότερο MPKI, για τα οποία θα τρέξουμε Timing Simulations.

4.1.2 Timing Simulators

Οι Timing Simulators είναι πιο ολοκληρωμένοι προσομοιωτές συστημάτων καθώς λαμβάνουν υπόψη και τις καθυστερήσεις. Χρησιμεύουν στο να δίνουν ακριβέστερες λεπτομέρειες για την επίδοση, όπως τον αριθμό εντολών που εκτελείται ανά κύκλο, που είναι πολύ πιο ρεαλιστικό μετρικό επίδοσης από τα MPKI της κρυφής μνήμης. Έτσι οι

προσομοιώσεις με Timing Simulators παίρνουν μεγαλύτερο χρόνο από τους αντίστοιχούς τους Functional Simulators.

4.2 Ο προσομοιωτής CRC kit

Ο προσομοιωτής CRC kit είναι μία ελεύθερα διαθέσιμη έκδοση του CMP$im[2], ενός ολοκληρωμένου Timing Simulator για πολυπύρηνους επεξεργστές. Το CRC kit παίρνει το όνομά του από ένα διαγωνισμό για σχεδιασμό του καλύτερου replacement policy "Cache Replacement Championship" για ένα workshop του 2010 [12].

Λόγω των αναγκών του διαγωνισμού, ο μόνος κώδικας που είναι ανοικτός είναι τα C++ αρχεία σχετικά το replcement policy και την LLC και κάποια headers. Αυτό έχει ως αποτέλεσμα την ευκολότερη χρήση του simulator αλλά αποκρύπτονται κάποιες χρήσιμες πληροφορίες και δυνατότητες που θα βοηθούσαν την ευρύτερή του χρήση για άλλα πεδία.

Σε γενικές γραμμές, εμείς θα συμπληρώσουμε τον κώδικα για τα replacement policies που μας ενδιαφέρουν και θα μετρήσουμε την επίδοση μέσω του μετρικού CPI (Cycles per Instruction) που παρέχεται για τους επιθυμητούς συνδυασμούς παραμέτρων των replacement policies που θα προκύψουν από πειράματα με Functional Simulator.

Αξίζει να σημειωθεί ότι λόγω του ότι κάποια software blobs του CRC kit είναι pre-compiled και δεν υποστηρίζονται από τις τελευταίες εκδόσεις του kernel του linux. Τα χαρακτηριστικά του λειτουργικού συστήματος στα οποία το έχω τρέξει και λειτουργούν είναι: πυρήνας 2.6.32-431.20.3.el6.x86_64 και διανομή CentOS release 6.5 (Final), έκδοση compiler: gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-4).

Η χρήση του simulator χωρίζεται σε δύο μέρη:

4.2.1 Παραγωγή Trace από εκτέλεση

Το CRC kit περιλαμβάνει το Intel Pin [3] με το οποίο παράγει traces με timing πληροφορίες για όποιο δήποτε εκτελέσιμο του λειτουργικού συστήματος linux. Τα

traces περιέχουν timing πληροφορίες αλλά έχουν proprietary μορφή, δηλαδή για τους σκοπούς του “Cache Replacement Championship” δεν δημοσιοποιήθηκε ο τρόπος με τον οποίο είναι δομημένα.

Αυτά τα traces είναι ανεξάρτητα του replacement policy και θα τα παράξουμε μια φορά χρησιμοποιώντας μια σειρά από Benchmarks. Θα τα χρησιμοποιήσει το ίδιο το CRC kit ως είσοδο για την δεύτερη λειτουργία που είναι οι προσομοιώσεις.

Ακολουθούν τα βήματα για παραγωγή trace από εκτέλεση. Σημειώνεται ότι δεν είναι τα ίδια όπως αναγράφονται στην ιστοσελίδα του CRC kit καθώς τα τροποποίησα για να λειτουργούν στο πιο σύγχρονο προαναφερθέν σύστημα:

- Εγκατάσταση CRC kit
  - Κατεβάζουμε το CRC kit εκτελώντας: wget http://www.jilp.org/jwac-1/CRC_DISTRIB.tgz
  - Αποσυμπιέζουμε εκτελώντας: tar -xzvf CRC_DISTRIB.tgz; cd CRC
  - Θέτουμε το absolute path της τοποθεσίας του CRC kit (δηλαδή το output της εντολής pwd) στη μεταβλητή FULL_PATH_TO_CRC_KIT στο αρχείο ρυθμίσεων pinkit/CONFIG/makefile.gnu.config
  - Αλλάζουμε την τοποθεσία του library libz με την ανάλογη του συστήματος μέσα στο makefile στο σημείο /usr/lib64/libz.a
    - Αυτό μπορούμε να το μάθουμε εκτελώντας: find / 2>/dev/null | grep libz.so.1
  - Κάνουμε compile εκτελώντας: make clean; make CMPsim64
- Παραγωγή Trace
  - Παράγουμε υποφάκελλο για τα traces εκτελώντας: mkdir traces
  - Ξεκινούμε την προσομοίωση εκτελώντας: pinkit/pin-2.7-31933-gcc.3.4.6-ia32_intel64-linux/pin -t ./bin/CMPsim.gentrace.64 -threads 1 -fwd [fast forward instructions in millions] -icount [traced instructions in millions] -o traces/[outputfile] -- [application] [application args] όπου

- [fast forward instructions in millions] είναι ένας ακέραιος που αντιπροσωπεύει τον αριθμό των εντολών σε εκατομύρια που θέλουμε να προσπελάσουμε πριν να αρχίσει να καταγράφεται το Trace
- [traced instructions in millions] είναι ένας ακέραιος που αντιπροσωπεύει τον αριθμό των εντολών σε εκατομμύρια που θέλουμε το Trace να αντιπροσωπεύει. Τυπικές τιμές είναι από 100 μέχρι 500.
- [outputfile] το όνομα του Trace που θα χρησιμοποιηθεί στην χρήση του
- [application] [application args] είναι η εντολή με την οποία θα τρέξει το πρόγραμμα για το οποίο θα παραχθεί το Trace
  - για παράδειγμα μπορούμε να το τρέξουμε για μία εκτέλεση σύντομης εντολής για δοκιμή αντικαθιστώντας το με: /bin/echo “test”

4.2.2 Προσομοίωση για μέτρηση της επίδοσης

Το CRC kit χρησιμοποιεί τα Traces από το προηγούμενο στάδιο για να προσομοιώσει την επίδοση με βάση το καινούριο κομμάτι κώδικα που θα αντιπροσωπεύει τις λειτουργίες της Last-Level Cache.

Τα στάδια που θα περιγραφούν πιο κάτω είναι αυτά που θα επαναλάβουμε πολλές φορές για τα ίδια Traces και αυτό εξαρτάται από το πόσους αλγόριθμους θέλουμε να προσομοιώσουμε και πόσα είναι τα Traces.

- Προσθήκη replacement policies
  - Μπαίνουμε στον κατάλογο του κώδικα εκτελώντας: cd src/LLCsim/
  - Εκεί υπάρχουν 4 header files και 2 αρχεία .cpp με κώδικα:
    - Το “crc_cache.cpp” είναι κομμάτι της υλοποίησης του CRC kit σχετικά με τις λειτουργείς της Last-Level Cache. Εδώ χρησιμεύει μόνο στην καλύτερη κατανόηση του πώς και πότε καλούνται εσωτερικά οι συναρτήσεις στο επόμενο αρχείο.
    - Το “replacement_state.cpp” είναι το σημείο στο οποίο το CRC kit θα παίρνει αποφάσεις σχετικά με το replacement policy της Last-Level Cache. Θα κάνουμε προσθήκη κώδικα σε 4 σημεία:

- Στην συνάρτηση InitReplacementState() τοποθετούμε διάφορες αρχικοποιήσεις που θα εκτελεστούν αρχικά για την προετοιμασία των δομών που θα χρησιμοποιεί το replacement policy μας
- Στην συνάρτηση GetVictimInSet() θα μπει ο κώδικας που είναι υπεύθυνος για το πού θα μπαίνει ένα καινούριο block όταν έχουμε miss. Με άλλα λόγια είναι εδώ που θα γίνει η απόφαση ποιου block θα γίνει evict στο συγκεκριμένο set που κάνει map το νέο block.
  - Σημειώνεται ότι η συνάρτηση αυτή δεν καλείται για τα πρώτα misses κατά την εκκίνηση της προσομοίωσης για τα οποία τα set δεν έχουν γεμίσει πλήρως όλα τους τα ways.
- Στην συνάρτηση UpdateReplacementState() θα παίρνουμε πληροφορίες για την ενημέρωση των δικών μας δομών. Για παράδειγμα καλείται μετά από κάθε hits και
  - Σημειώνεται ότι η συνάρτηση αυτή δεν καλείται για Writeback hits διότι το CRC kit τους δίνει χαμηλή προτεραιότητα για το δικό του αντίγραφο της Last-Level Cache
- Στην συνάρτηση PrintStats() προσθέτουμε τα δικά μας στατιστικά που θέλουμε να εκτυπώνονται στο τέλος της προσομοίωσης και κάθε 100M εντολές, κυρίως για απασφαλμάτωση. Είναι προαιρετικό καθώς το το CRC kit εκτυπώνει τα δικά του στατιστικά όπως τα CPI και MPKI που μας ενδιαφέρουν, αλλά και άλλα και για τα τρία Cacle Levels.

- Για την καλύτερη διαχείριση του κώδικα συνιστάται να δημιουργήσουμε δικά μας αρχεία στον κατάλογο αυτό για την λογική των replacement policies μας. Στο replacement_state.cpp θα μπούν μόνο οι κλείσεις των συναρτήσεων μας. Επιπλέον θα πρέπει να συμπληρώσουμε στα header files και το makefile το όνομα του νέου κώδικα (π.χ #include ”Philippos_drrip.h” στα headers και Philippos_drrip.c στη γραμμή του gcc στο makefile)

- Προσομοιώσεις
  - Αφού κάναμε της απαραίτητες αλλαγές στον κατάλογο του κώδικά πάμε δυο επίπεδα πίσω εκτελώντας: cd ../..

- Κάνουμε compile εκτελώντας: make clean; make CMPsim64
- Παράγουμε υποφάκελλο για τα αποτελέσματα της κάθε προσομοίωσης εκτελώντας: mkdir runs
- Ξεκινούμε την προσομοίωση εκτελώντας: ./bin/CMPsim.usetrace.64 -threads [# threads] (-t [trace file name] | -mix [trace mix file name]) -icount [number of instructions in millions] -o [output file name] -cache [cache configuration] -LLCrepl [Replacement policy number] όπου
  - [# threads] είναι ο αριθμός των threads που θέλουμε να προσομοιώσουμε. Εμείς προς το παρόν θέτουμε 1 διότι μας ενδιαφέρει η single-program επίδοση, αν και είναι αυτονόητο πώς αλλάζει η πιο πάνω εντολή για multi-program πειράματα
  - [trace file name] το όνομα του trace που θέλουμε να τρέξει (π.χ. traces/400.perlbench.trace.gz)
  - [number of instructions in millions] είναι ένας ακέραιος που αντιπροσωπεύει τον αριθμό των εντολών σε εκατομμύρια που θέλουμε να τρέξουμε. Εδώ αυτή την επιλογή την παραβλέπουμε διότι χρησιμεύει σε multi-program πειράματα για να επανεκκινεί όσα traces τερμάτησαν πριν την ολοκλήρωση της προσομοίωσης. (προσθήκη - autorewind 1) και να τα σταματά όλα όταν ολοκληρωθεί ο επιθυμητός αριθμός εντολών
  - [output file name] το όνομα του αρχείου που θα περιλαμβάνει τα στατιστικά της προσομοίωσης (π.χ. runs/400.perlbench.stats)
  - [cache configuration] παίρνει τιμές από 0 μέχρι 2 και καθορίζει ποιος αλγόριθμος replacement policy θα χρησιμοποιηθεί. Το 0 αντιπροσωπεύει το LRU, το 1 το Random και το 2 αυτό που έχουμε γράψει εμείς.
  - [cache configuration] πληροφορίες για τα χαρακτηριστικά της cache. Για παράδειγμα εμείς έχουμε το ακόλουθο cache configuration: UL3:1024:64:16 που είναι μια cache με 1024 set, 16 way associativity ανά set και block size των 64 bytes

Επισημαίνεται ότι για διευκόλυνση της διαδικασίας των προσομοιώσεων και της συλλογής αποτελεσμάτων, μπορούμε να αυτοματοποιήσουμε τα πιο πάνω και την

συλλογή συγκεκριμένων στατιστικών με συγγραφή scripts όπως γλώσσας bash ή python.

Επιπλέον λόγο του σημαντικού χρόνου που καταλαμβάνουν οι προσομοιώσεις μπορούμε να παραλληλίσουμε τις εκτελέσεις αν έχουμε πολλά traces με χρήση condor scripts ή slurm sbatch scripts ανάλογα με το τι job sheduling σύστημα διαθέτει το cluster στο οποίο τρέχουμε. Αν τρέχουμε σε μια μηχανή, πάλι μπορούμε να χρησιμοποιήσουμε condor scripts αλλά μπορούμε και να χρησιμοποιήσουμε threadpool στη python για να τρέχουν μέχρι N προσομοιώσεις ανά χρονική στιγμή.

4.3 Κώδικας για υλοποίηση Cache Replacement Policies

Ο κώδικας υλοποίησης των Cache Replacement Policies που έγραψα για προσομοίωση με τη χρήση του CRC kit τον σχεδίασα έτσι ώστε να μπορεί να λειτουργεί και αυτόνομα.

Με τη χρήση του CRC kit παίρνουμε καλύτερο μέτρο επίδοσης διότι θεωρείται Timing Simulator και με την αυτόνομη εκδοχή του κώδικά μου παίρνουμε MPKI μετρήσεις και θεωρείται Functional Simulator.

Όταν λειτουργεί σαν τμήμα του CRC kit, η main του απενεργοποιείται μέσω δήλωσης flag του pre-compiler και έτσι ο κώδικας του "replacement_state.cpp" αρχείου κάνει κλήσεις στις αντίστοιχες συναρτήσεις του νέου κώδικα. Οι συναρτήσεις αυτές εξυπηρετούν ταυτόχρονα και τις απαιτήσεις του CRC kit για τις επιλογές τοποθέτησης των νέων block αλλά και την λειτουργία ενός απλού address trace-driven Functional Simulator μιας LLC.

Ο Functional Simulator μου είναι address-trace-driven διότι πρέπει να του παραχωρηθεί αρχείο με διευθύνσεις (address trace) που φτάνουν ως αιτήματα στην LLC από Timing Simulation, με τις οποίες θα ελέγχονται οι αποφάσεις των replacement policies που θα σχεδιάσουμε.

Ως πρώτο βήμα υλοποιούμε τις λειτουργίες του DRRIP έτσι όπως περιγράφονται στο

"High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)" [1] και για το Set Dueling [9].

Το DRRIP θα το θεωρούμε σαν baseline των πειραμάτων μας και θα μετρούμε βελτίωση στην επίδοση (speedup) με βάση το DRRIP. Επιπλέον, τα address traces για την Functional Simulator εκδοχή του κώδικά μου θα τα παράξουμε από εκτέλεση του CRC kit χρησιμοποιώντας την υλοποίησή μου για το DRRIP (που είναι και μέρος του Functional Simulator) ως replacement policy.

Διαδικασία απόκτησης address traces από το CRC kit:

- Μπαίνουμε στον κατάλογο του κώδικα εκτελώντας: cd src/LLCsim/
- Προσθέτουμε τον ακόλουθο κώδικα στη γραμμή 298 του αρχείου crc_cache.cpp printf("%llu %d\n",(unsigned long long) paddr, accessType == ACCESS_WRITEBACK);
    - Αυτό θα μας τυπώνει στην έξοδο σε κάθε γραμμή κάθε access στην Last-Level Cache μαζί με ένα bit που θα δηλώνει αν είναι Writeback access.
    - Ενώ μπορούμε να αγνοήσουμε την πληροφορία αν ένα access είναι Writeback access, για καλύτερη προσέγγιση των αποτελεσμάτων του CRC kit από τον Functional Simulator, αν ένα WB access είναι hit τότε το αγνοούμε
- Πηγαίνουμε στον αρχικό φάκελλο του CRC kit και κάνουμε compile εκτελώντας: cd ../..; make clean; make CMPsim64
- Τρέχουμε την προσομοίωση όπως εξηγήθηκε πιο πάνω αλλά κάνουμε redirect το output σε αρχείο
- Φιλτράρουμε το output έτσι ώστε να έχει μόνο τις πληροφορίες που εκτυπώνουμε στο crc_cache.cpp
    - Αυτό το κάνουμε με το να εκτυπώνουμε όλες τς γραμμές που έχουν την μορφή 2 διαδοχικών ακεραίων μέσω ενός python script

Διαδικασία εκτέλεσης προσομοιώσεων με τον Fuctional Simulator:

- Μπαίνουμε στον κατάλογο του κώδικα του Functional Simulator εκτελώντας: cd MODULE7_simulator2
- Αλλάζουμε τη main έτσι ώστε να κλιθεί η συνάρτηση Psimulator() για τα επιθυμητά traces
    - Τα ονόματα των traces δηλώνονται στατικά, για παράδειγμα: char benchmark2006[29,20] = {...};
    - Ο Functional Simulator ψάχνει τα traces στον υποφάκελλο με το όνομα t
        - Μια πρακτική λύση αν έχουμε τα traces σε άλλο φάκελλο είναι να κάνουμε soft link του αρχικού φακέλλου για παράδειγμα εκτελούμε: ln -s ../CRC/address-traces t
    - Η τρίτη παράμετρος της συνάρτησης δηλώνει αν το cache configuration θα δηλωθεί από τις υπόλοιπες παραμέτρους της Psimulator(). Αν είναι 0 τότε τρέχει ένα default configuration που υπάρχει στην PinitializeCaller() συνάρτηση.
- Αν θέλουμε έξτρα αρχεία από στατιστικά τότε παράγουμε υποφάκελλο με το όνομα stats εκτελώντας: mkdir stats και θέτουμε την τιμή της μεταβλητής ExtraInfoFiles = 1; μέσα στην συνάρτηση Psimulator()
- Κάνουμε compile εκτελώντας: make clean; make
- Τρέχουμε εκτελώντας: ./dist/Debug/GNU-Linux-x86/module7_simulator_2

Στο Παράρτημα 1 μπορούμε να δούμε ένα παράδειγμα εκτέλεσης του Functional Simulator.

Παρατηρούμε ότι μας δίνει και πληροφορίες για βελτίωση στην επίδοση (Speedup σχετικά με το DRRIP) και κύκλους για κάθε εντολή (CPI). Όμως δεν του δίνουμε πληροφορίες για καθυστερήσεις σε διάφορα επίπεδα της ιεραρχίας μνήμης. Οι τιμές αυτές είναι προσεγγιστικές και παράγονται από Performance Model το οποίο θα εξηγήσουμε σε επόμενο κεφάλαιο.

Μια άλλη παρατήρηση είναι ότι το άθροισμα των misses και hits δεν ισούται με τον αριθμό των accesses. Αυτό ισχύει διότι στα hits εδώ αγνοούνται τα Writeback hits για να προσεγγίζεται καλύτερα η λειτουργία του CRC kit.

Τα Cache Replacement Policies που υλοποιεί ο κώδικάς μου είναι όλα παραλλαγές του DRRIP. Αυτά ρυθμίζονται μεταβάλλοντας διάφορες παραμέτρους του DRRIP αλλά και μερικών νέων χαρακτηριστικών που έχω προσθέσει τα οποία πιστεύω ότι μπορεί να έχουν αντίκτυπο στην επίδοση. Έτσι μπορούμε ανά πάσα στιγμή να χρησιμοποιούμε το κλασσικό DRRIP επαναφέροντας τις αντίστοιχες τιμές στις παραμέτρους του προσομοιωτή. Το εύρος των παραλλαγών θα σχολιαστεί σε πιο κάτω κεφάλαιο και θα αναφερόμαστε σε αυτό με τον όρο Design Space.

Δεν παραθέτω τον κώδικα του Functional Simulator μου εδώ λόγω της έκτασής του που είναι μερικές χιλιάδες γραμμές. Όμως στο παράρτημα 2 υπάρχει παράδειγμα Functional Simulator για απλά policies.

4.4 Προγράμματα από το Spec2006 Benchmark Suite

Η σειρά από προγράμματα που χρησιμοποιούμε στην αξιολόγηση των διαφόρων replacement policies είναι τα SPEC CPU™ 2006 από το Spec2006 Benchmark Suite [7].

Είναι 29 στο σύνολο και αποτελούνται από 12 Benchmarks που χρησιμοποιούν πράξεις ακεραίων (400.perlbench, 401.bzip2, 403.gcc, 429.mcf, 445.gobmk, 456.hmmer, 458.sjeng, 462.libquantum, 464.h264ref, 471.omnetpp, 473.astar και 483.xalancbmk) και 17 που χρησιμοποιούν πράξεις δεκαδικών (410.bwaves, 416.gamess, 433.milc, 434.zeusmp, 435.gromacs, 436.cactusADM, 437.leslie3d, 444.namd, 447.dealII, 450.soplex, 453.povray, 454.calculix, 459.GemsFDTD, 465.tonto, 470.lbm, 481.wrf και 482.sphinx3).

Η εντολές με τις οποίες εκτελούμε τα benchmarks περιγράφονται στο “SPEC CPU2006 command lines” του Kenneth Hoste[12]. Τρέχουμε πάντα το πρώτο input του κάθε benchmark. Για να παραχθούν τα binaries και τα δεδομένα που χρειάζονται πρέπει να εγκαταστήσουμε το Spec2006 Benchmark Suite και να τα τρέξουμε όλα τουλάχιστον μια φορά με την χρήση της εντολής runspec. Διάφορες περιπτώσεις σφαλμάτων κατά την διάρκεια της εγκατάστασης σε σύγχρονα λειτουργικά συστήματα περιγράφονται

στο “The Unofficial Guide to Running SPEC CPU2000/2006 in 2013 and later” του Hisanobu Tomari[14].

Όπως είδαμε πιο πάνω, θα τα τρέξουμε μια φορά με το pinkit του CRC kit για να παραγάγουμε τα taces. Όμως πριν από αυτό το βήμα, θα χρειαστεί να βρούμε μια αντιπροσωπευτική περιοχή για το καθένα (representative region) μεγέθους 250 εκατομμυρίων εντολών έτσι ώστε να έχουμε αντιπροσωπευτικά αποτελέσματα σε μικρές προσομοιώσεις. Τα αποτελέσματα της εύρεσης των αντιπροσωπευτικών περιοχών θα είναι ο αριθμός των εντολών που κάνουμε προσπέλαση μέχρι να φτάσουμε στην περιοχή των 250M εντολών που μας ενδιαφέρει.

4.4.1 Εύρεση Αντιπροσωπευτικών Περιοχών

Για την εύρεση των αντιπροσωπευτικών περιοχών (representative regions) χρησιμοποιήσαμε μια τροποποιημένη έκδοση ενός pin tool του Intel Pin[3], του “itrace.cpp”, που υπήρχε έτοιμο στο Xi group. Έχουμε παραγάγει γραφήματα που δείχνουν τη συμπεριφορά των πειραμάτων για την εκτέλεσή τους με στόχο να επιλέξουμε τα repsesentative regions (visually) με το μάτι για 250M εντολές. Η τροποποίηση είναι ότι παίρνοντας την τιμή του κάθε PC (Program Counter ή Instrucion Pointer) της εντολής που καλείται παράγονται ταυτότητες για κάθε block μεγέθους N και διάρκειας ενός εκατομμυρίου εντολών. Η ταυτότητα του κάθε block είναι N αριθμοί που αντιπροσωπεύουν πόσες εντολές κάνουν map σε κάθε σημείο (PC modulo N) του block (δηλαδή από 0 μέχρι N-1).

Θεωρητικά τα representative regions μπορούν να βρεθούν οπτικά και με απλό plot των PC που καλούνται, όμως είναι πρακτικά αδύνατο διότι μιλούμε για πολλά δισεκατομμύρια από εντολές και τα PC μπορούν εύκολα να γεμίσουν ένα σκληρό δίσκο.

Η εκτέλεση των προγραμμάτων με το Intel Pin δεν θεωρείται προσομοίωση. Τα Benchmarks θα τρέξουν κανονικά (natively) στην μηχανή, και ταυτόχρονα το Intel Pin θα παίρνει πληροφορίες κατά τη διάρκεια της εκτέλεσης τους. Τερματίζουμε συμβατικά την εκτέλεση στα 50 δισεκατομμύρια εντολές για να κερδίσουμε χρόνο.

Αυτό που είναι σημαντικό στην επιλογή των representative regions με το μάτι είναι το πρόγραμμα να φαίνεται ότι έχει ήδη περάσει από τα αρχικά στάδια της εκτέλεσης του, δηλαδή την initialization φάση του. Παρατηρούμε επίσης ότι πολλές φορές υπάρχει περιοδικότητα στης φάσεις του προγράμματος και σε αυτή την περίπτωση προσπαθούμε να επιλέγουμε representative region που να συμπεριλαμβάνει όσο το δυνατό πιο παρόμοια αναλογία φάσεων με ολόκληρο το πρόγραμμα (αφού έχει περάσει η initialization).

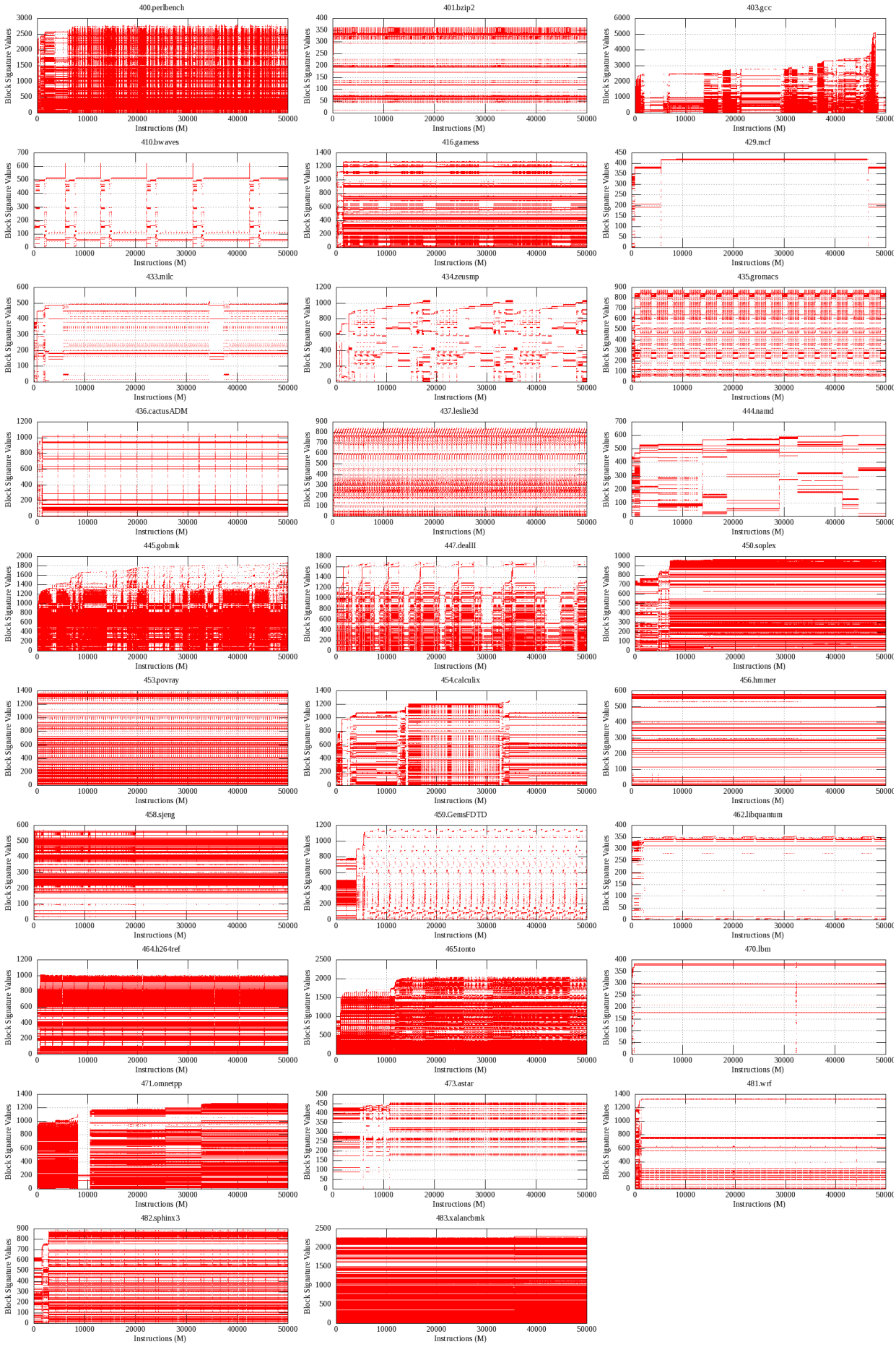
Στον πίνακα 4.1 υπάρχουν τα αποτελέσματα για τα representative regions των εκτελέσεων των 29 benchmarks και όλα τα σχετικά γραφήματα. Υπενθυμίζεται ότι όπου υπήρχαν πολλαπλά διαθέσιμα input έχω επιλέξει το πρώτο. Σημειώνεται επίσης ότι η πραγματική επιλογή των περιοχών έγινε με την zoom δυνατότητα του πακέτου gnuplot για ακριβέστερη προσέγγιση των περιοχών.

Δεν υπάρχει τέλεια επιλογή representative regions και μπορούν να υπάρχουν πολλές διαφορετικές προσεγγίσεις. Εναλλακτική της πιο πάνω μεθόδου είναι να χρησιμοποιήσουμε ένα εργαλείο που κάνει αυτή τη διαδικασία αυτόματα όπως το SimPoint[13].

Στα γραφήματα του σχήματος 4.1 μπορούμε να διακρίνουμε εύκολα ότι σε κάθε benchmark υπάρχει η αρχική φάση και μετά ακολουθούν 1 ή περισσότερες. Υπάρχει αρκετή περιοδικότητα εντός των υπόλοιπων φάσεων γιαυτό και αν και μικρό το εύρος των representative regions που επιλέξαμε είναι αρκετά αντιπροσωπευτικά.

Benchmark	i686 Representative Region (M)
400.perlbench	11900 – 12150
401.bzip2	10000 – 10250
403.gcc	42000 – 42250
429.mcf	10000 – 10250
445.gobmk	39000 – 39250
456.hmmer	36000 – 36250
458.sjeng	35600 – 35850
462.libquantum	7500 – 7750
464.h264ref	26000 – 26250
471.omnetpp	36000 – 36250
473.astar	12200 – 12450
483.xalancbmk	37000 – 37250
410.bwaves	33800 – 34050
416.gamess	35200 – 35450
433.milc	8300 – 8550
434.zeusmp	17300 – 17550
435.gromacs	11800 – 12050
436.cactusADM	2000 -2250
437.leslie3d	1400 – 1650
444.namd	45000 - 45250
447.dealII	6000 – 6250
450.soplex	16000 – 16250
453.povray	600-850
454.calculix	41000 – 41250
459.GemsFDTD	8470 – 8720
465.tonto	25800 – 26050
470.lbm	2000 – 2250
481.wrf	23000 – 23250
482.sphinx3	6600 – 6850

Πίνακας 4.1: Represenative Regions για τα SpecCPU 2006 Benchmarks (32-bit)



Σχήμα 4.1: Γραφήματα εντοπισμού των Represenative Regions των SpecCPU 2006 Benchmarks

Κεφάλαιο 5

Χαρακτηρισμός των Benchmarks

5.1 Εισαγωγή

Στο κεφάλαιο αυτό περιγράφονται διαδικασίες με τις οποίες θα δημιουργήσουμε μια καλύτερη εικόνα του πώς συμπεριφέρεται το κάθε benchmark στην Last-Level Cache. Κατανοώντας πώς επηρεάζεται η επίδοση με την αυξομείωση της Last-Level Cache και πώς προκύπτουν τα Misses είναι αρκετά σημαντικό βήμα διότι γίνεται πιο εύκολο να εντοπισθούν τυχών λάθη στα υπόλοιπα βήματα που θα ακολουθήσουμε.

Εξίσου σημαντικό αποτέλεσμα είναι ότι με τον χαρακτηρισμό μπορούμε να πάρουμε σημαντικές πληροφορίες που ίσως βοηθήσουν στην επινόηση χαρακτηριστικών για σχεδιασμό replacement policies και άλλων χαρακτηριστικών των Last-Level Cache. Για παράδειγμα από την κατηγοριοποίηση των misses μία από τις παρατηρήσεις που θα κάνουμε είναι ότι μπορούμε να διακρίνουμε αν υπάρχει ανάγκη για μεγαλύτερο βαθμό associativity στην Last-Level Cache που προσομοιώνουμε.

Τέλος με την ένταξη των benchmark σε κατηγορίες μπορούμε ακόμα να πάρουμε αποφάσεις σε μελλοντικές εξερευνήσεις βελτίωσης επίδοσης της Last-Level Cache για πολυπύρηνους επεξεργαστές. Σε αυτή την περίπτωση θα παράγαμε αντιπροσωπευτικούς

συνδυασμούς από benchmarks έτσι ώστε να μην χρειάζεται να τρέξουμε μεγάλο αριθμό συνδυασμών για να πάρουμε αντιπροσωπευτικά αποτελέσματα.

5.2 Κατηγοριοποίηση των Misses

Σε αυτό το σημείο στόχος είναι να παράξουμε ένα γράφημα στο οποίο διαφαίνεται εύκολα η αναλογία και η ποσότητα των misses για κάθε κατηγορία ανά benchmark.

Διακρίνουμε τις ακόλουθες κατηγορίες misses, 3 από τις οποίες είναι ήδη γνωστές από το πρώτο κεφάλαιο:

- Compulsory misses: Συμβαίνουν όταν τα blocks αναζητούνται για πρώτη φορά, για παράδειγμα κατά την εκκίνηση του υπολογιστή
- Capacity misses: Συμβαίνουν όταν το μέγεθος της cache δεν αρκεί για το μέγεθος που απαιτεί η διεργασία
- Conflict misses: Συμβαίνουν όταν πολλαπλά blocks κάνουν map στο ίδιο set και ο βαθμός του associativity δεν είναι αρκετός
- Policy misses: Ορίζουμε ως Policy misses τα misses του policy που χρησιμοποιούμε που προκύπτουν διότι δεν λαμβάνει πάντα τις βέλτιστες αποφάσεις

Η διαδικασία που ακολουθεί γίνεται με Functional Simulations διότι θα χρησιμοποιηθεί εκτενώς ο αλγόριθμος Belady's OPT για τον οποίο χρειάζεται γνώση του μέλλοντος. Η είσοδος της κάθε προσομοίωσης είναι το address trace που παράξαμε τρέχοντας το κάθε timing trace 250M εντολών στο CRC kit με DRRIP να είναι το προεπιλεγμένο replacement policy.

Για τα policy misses θα τρέξουμε τον Functional Simulator μου με προεπιλογή ξανά το DRRIP διότι μας ενδιαφέρει η συμπεριφορά του DRRIP.

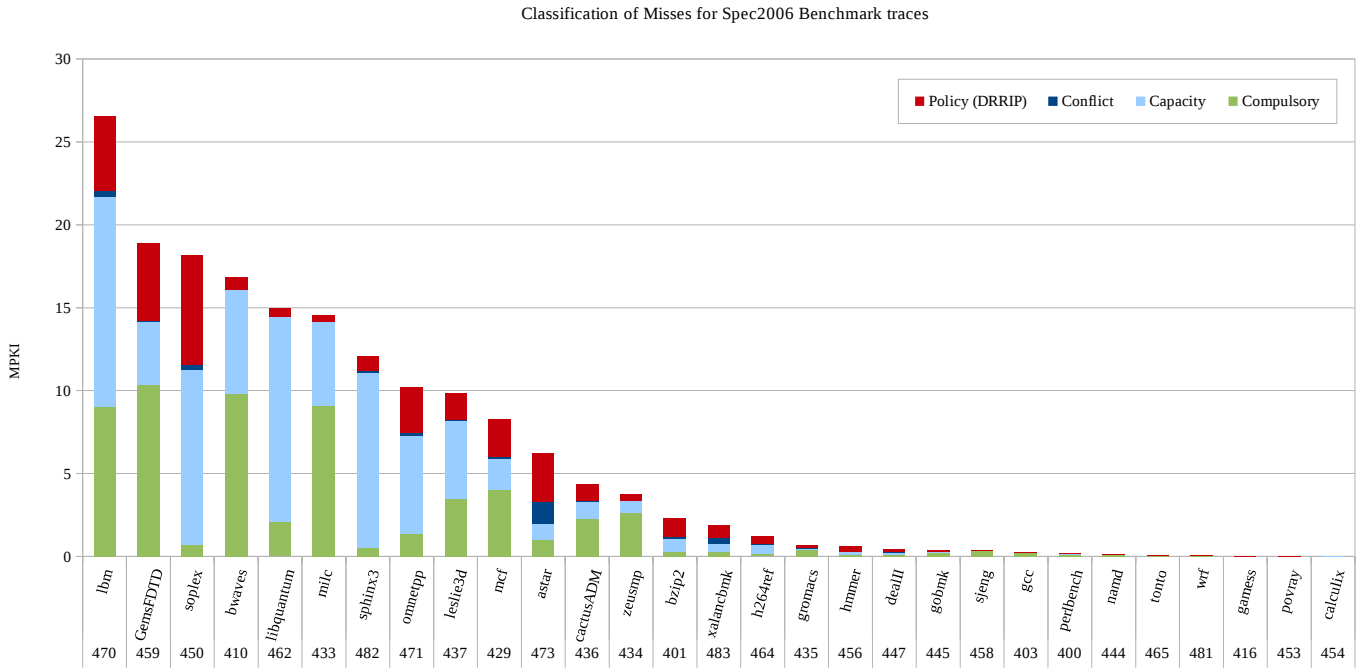
Για τα υπόλοιπα τρία (compalsory, capacity και conflict) χρησιμοποίησα ένα ξεχωριστό δικό μου άλλο Functional Simulator που προσομοιώνει Cache με αλγόριθμο replacement policy τον Belady's OPT σε γλώσσα python.

Για αυτή την διαδικασία δεν θα γίνει διάκριση σε Writebacks. Δηλαδή αντίθετα με το CRC kit, αν έχουμε hit σε Writeback τότε λαμβάνεται υπόψιν ως κανονικό hit.

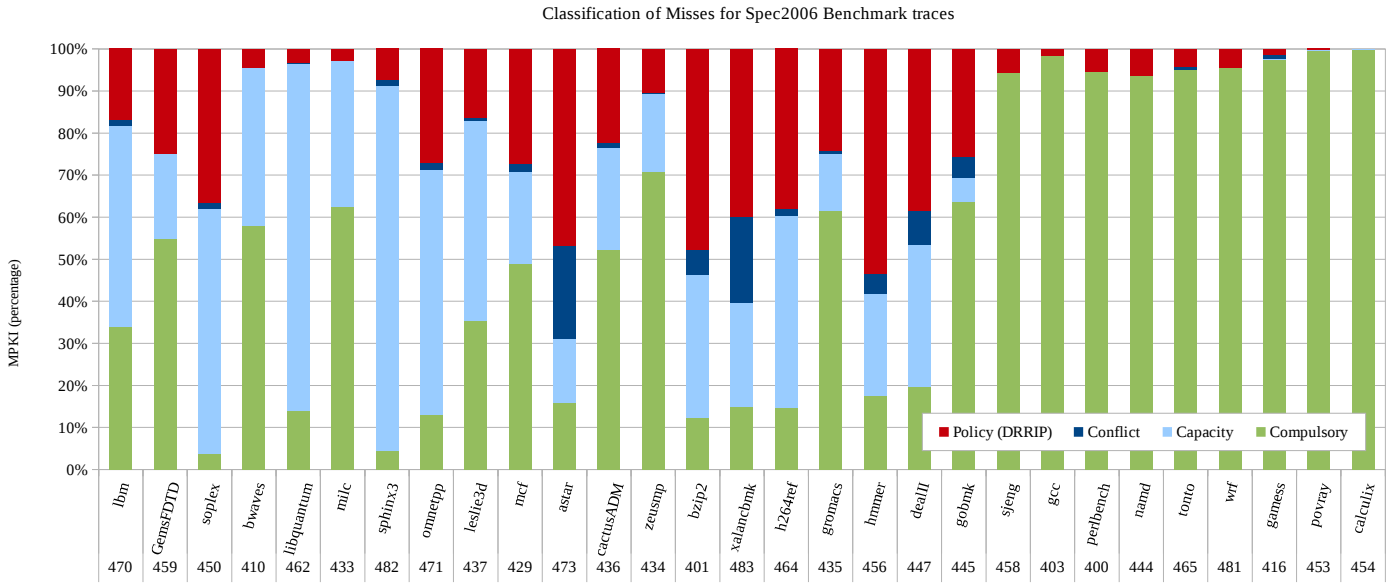
Πιο κάτω περιγράφεται αναλυτικά η μεθοδολογία που σχεδίασα για τον υπολογισμό των misses που ανήκουν σε κάθε κατηγορία για κάθε benchmark σε βήματα:

- Compulsory misses: Συμβαίνουν όταν τα blocks αναζητούνται για πρώτη φορά.
  - Επομένως τρέχουμε τον Belady's OPT functional simulator με άπειρη cache Δηλαδή πολύ μεγάλο αριθμό από ways σε μια Fully-Associative Cache.
  - Το άθροισμα των compulsory misses είναι τα misses που προκύπτουν από αυτή τη προσομοίωση (εναλλακτικά: cat adtrace1.txt | sort | uniq | wc -l)
- Capacity misses: Συμβαίνουν όταν το μέγεθος της cache δεν αρκεί για το μέγεθος που απαιτεί η διεργασία
  - Τρέχουμε τον Belady's OPT functional simulator με μια Fully-Associative Cache μεγέθους 1 Megabyte
  - Το άθροισμα των capacity misses είναι τα misses που προκύπτουν όταν αφαιρέσουμε τα compulsory misses από τα misses αυτής της προσομοίωσης
- Conflict misses: Συμβαίνουν όταν πολλαπλά blocks κάνουν map στο ίδιο set και ο βαθμός του associativity δεν είναι αρκετός
  - Επομένως τρέχουμε τον Belady's OPT functional simulator με μια Set-Associative Cache μεγέθους 1 Megabyte, 1024 sets και 16 ways ανά set
  - Το άθροισμα των conflict misses είναι τα misses που προκύπτουν όταν αφαιρέσουμε τα capacity misses από τα misses αυτής της προσομοίωσης
- Policy misses: Τα misses που προκύπτουν από τις ατέλειες του DRRIP
  - Επομένως τρέχουμε τον αρχικό μου Functional Simulator με μια Set-Associative Cache μεγέθους 1 Megabyte, 1024 sets, 16 ways ανά set και replacement policy το DRRIP
  - Το άθροισμα των policy misses είναι τα misses που προκύπτουν όταν αφαιρέσουμε τα conflict misses από τα misses αυτής της προσομοίωσης

Πιο κάτω βλέπουμε τις δυο εκδοχές γραφημάτων για τα αποτελέσματα. Παρουσιάζονται stacked η κάθε κατηγορία misses για κάθε benchmark. Τα benchmark είναι ταξινομημένα ανά συνολικό αριθμό από MPKI (Misses per K instructions). Στο πρώτο graph τα misses μετρούνται σε MPKI. Στο δεύτερο η κάθε κατηγορία misses αντιπροσωπεύονται ως ποσοστά.



Σχήμα 5.1: Κατηγοριοποίηση των misses για κάθε SpecCPU 2006 benchmark



Σχήμα 5.2: Κατηγοριοποίηση των misses για κάθε SpecCPU 2006 benchmark (σε ποσοστά)

Κάτι που μπορούμε εύκολα να παρατηρήσουμε είναι ότι υπάρχει σχετικά αρκετό περιθώριο βελτίωσης του DRRIP replacement policy και ότι benchmark από benchmark διαφέρει αρκετά στην κατανομή των misses ανά κατηγορία.

Όσον αφορά τα conflict misses φαίνεται ότι δεν θα έχουμε μεγάλη διαφορά αν αυξήσουμε το associativity της Last-Level Cache αφού μόνο σε 2 benchmarks (astar και xalancbmk) είναι πάνω από 10% των συνολικών misses.

Όσον αφορά τα capacity misses είναι γενικά αρκετά και μια μεγαλύτερη Last-Level Cache φαίνεται ότι θα βοηθούσε αρκετά την επίδοση (θα το επαληθεύσουμε στο επόμενο υποκεφάλαιο).

Για τα compulsory misses βλέπουμε ότι είναι και αυτά αρκετά αλλά δεν μπορούμε να κάνουμε κάτι περισσότερο από το να τα κρύψουμε με prefetching. Στη μελέτη αυτή δεν θα ασχοληθούμε με prefetching.

5.3 Μελέτη ευαισθησίας στο μέγεθος της κρυφής μνήμης

Σε αυτό το υποκεφάλαιο θα κάνουμε μια σύντομη μελέτη κατά πόσο μπορούμε να αλλάξουμε την επίδοση με την αύξηση ή μείωση του μεγέθους της Last-Level Cache. Συγκεκριμένα θα τρέξουμε Timing Simulations χρησιμοποιώντας το CRC kit και προεπιλεγμένο replacement policy το DRRIP για τρία διαφορετικα μεγέθη Last-Level Cache. Δεν τρέχουμε Functional Simulations διότι μας ενδιαφέρει η επίδοση (IPC).

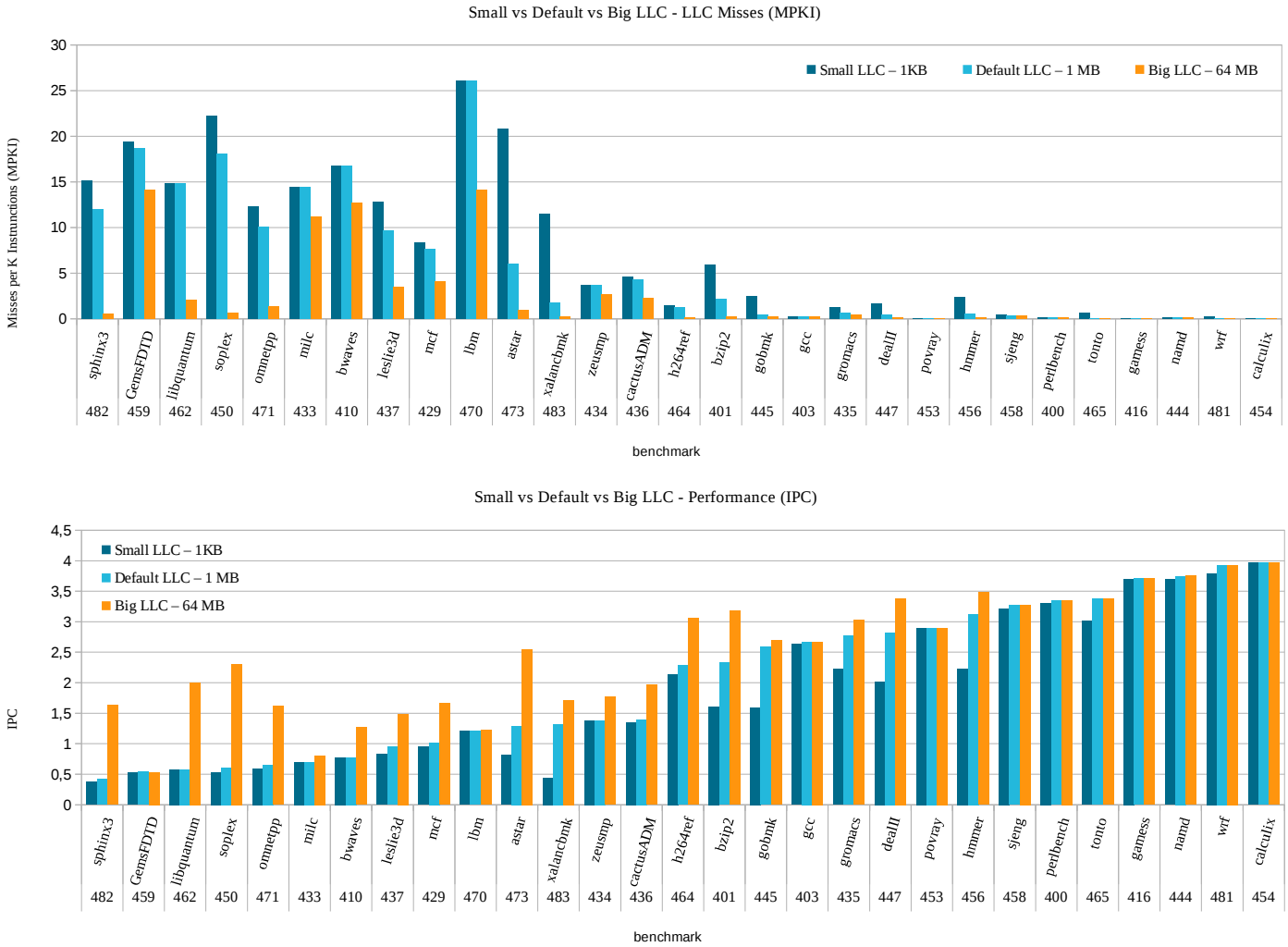
Σε διαφορετικά benchmarks η αυξομείωση στα misses έχει διαφορετικό αντίκτυπο στην επίδοση. Θα αναφερόμαστε στο πόσο επηρεάζεται το IPC από τον αριθμό των misses ως ευαισθησία. Έτσι θα λέμε ότι ένα benchmark ότι έχει μεγάλη ευαισθησία στο μέγεθος της Last-Level Cache όταν παρουσιάζει σημαντική διακύμανση στην επίδοση στα αποτελέσματα για τα 3 διαφορετικά μεγέθη.

Ένας από τους κύριους λόγους που δεν είναι ομοιόμορφα τα benchmarks όσον αφορά την ευαισθησία στο μέγεθος της Last-Level Cache είναι ότι υπάρχουν δύο ακόμα επίπεδα κρυφών μνημών στα οποία η Last-Level Cache είναι “αγνωστικιστική”. Γενικά

υπάρχουν πολλών ειδών καθυστερήσεις που καθορίζουν την ολική επίδοση και accesses στην Last-Level Cache είναι μόνο μια από αυτές.

Τα τρία μεγέθη με τα οποία θα προσομοιώσουμε την εκτέλεση σε διαφορετικές Last-Level Cache είναι 1KB, 1MB και 64MB. Το 1KB θα αντιπροσωπεύει την απουσία της L3 (Last-Level Cache), το 1MB το προεπιλεγμένο μέγεθος Last-Level Cache και το 64MB μια αρκετά μεγάλη Last-Level Cache.

Πιο κάτω βλέπουμε τα αποτελέσματα των 3 προσομοιώσεων για κάθε benchmark υπό την μορφή γραφικών παραστάσεων. Στο πρώτο γράφημα βλέπουμε πως επηρεάζεται το MPKI και στο δεύτερο την επίδοση (IPC). Και τα δύο είναι ταξινομημένα με το IPC των προσομοιώσεων με 1MB Last-Level Cache σε αύξουσα σειρά.



Σχήμα 5.3: Γραφήματα διακύμανσης επίδοσης σε σχέση με το μέγεθος της LLC

Από τα πιο πάνω γραφήματα παρατηρούμε αμέσως ότι γενικά περισσότερα misses φέρουν χαμηλότερη επίδοση.
Για κάποια benchmarks φαίνεται ότι δεν κάνει σχεδόν καθόλου διαφορά η ύπαρξη της Last-Level Cache. Αυτό γίνεται για διαφορετικούς λόγους.

Για το lbm, που είναι ένα scanning benchmark η επίδοση παραμένει περίπου η ίδια και με τα τρία μεγέθη Last-Level Cache.

Επίσης δεν κάνει πολύ διαφορά για την απουσία της 1MB Last-Level Cache τα GemsFDTD, libquantum, milc, bwaves, zeusmp αλλά κάνει μεγάλη διαφορά για την 64MB Last-Level Cache. Αυτό είναι μια ένδειξη ότι είναι thrashing benchmarks αφού για την 1MB Last-Level Cache η επίδοση είναι παρόμοια με το να μην υπάρχει καθόλου Last-Level Cache.

Ακόμα υπάρχουν και τα benchmarks όπως τα gcc, povray, gamess, namd, calculix και sjeng στα οποία η ύπαρξη Last-Level Cache δεν κάνει διαφορά λόγω του μικρού τους working set. Δηλαδή τα άλλα επίπεδα κρυφής μνήμης, τα L1 και L2 είναι αρκετά για τα δεδομένα που κάνουν πρόσβαση κατά την εκτέλεση.

Για τα υπόλοιπα benchmarks, δηλαδή τα sphinx, soplex, omnetpp, leslie3d, mcf, astar, xalancbmk, cactusADM, h264ref, bzip2, gobmk, gromacs, dealII, hmmer, wrf και tonto φαίνεται από αυτό το πείραμα ότι υπάρχει αρκετή επίδραση της Last-Level Cache στην επίδοση, το καθένα σε διαφορετικό βαθμό.

Στο επόμενο υποκεφάλαιο παραθέτω μια διαφορετική προσέγγιση για κατηγοριοποίηση των benchmarks λίγο πιο μεθοδική και με λιγότερες προσομοιώσεις.

5.4 Ένταξη σε κατηγορίες σχετικά με την συμπεριφορά στη κρυφή μνήμη

Σε αυτό το σημείο θα κατηγοριοποιήσουμε τα SpecCPU 2006 benchmarks στις ακόλουθες 4 κατηγορίες, παρόμοιες με αυτές που περιγράφονται στο [10]:

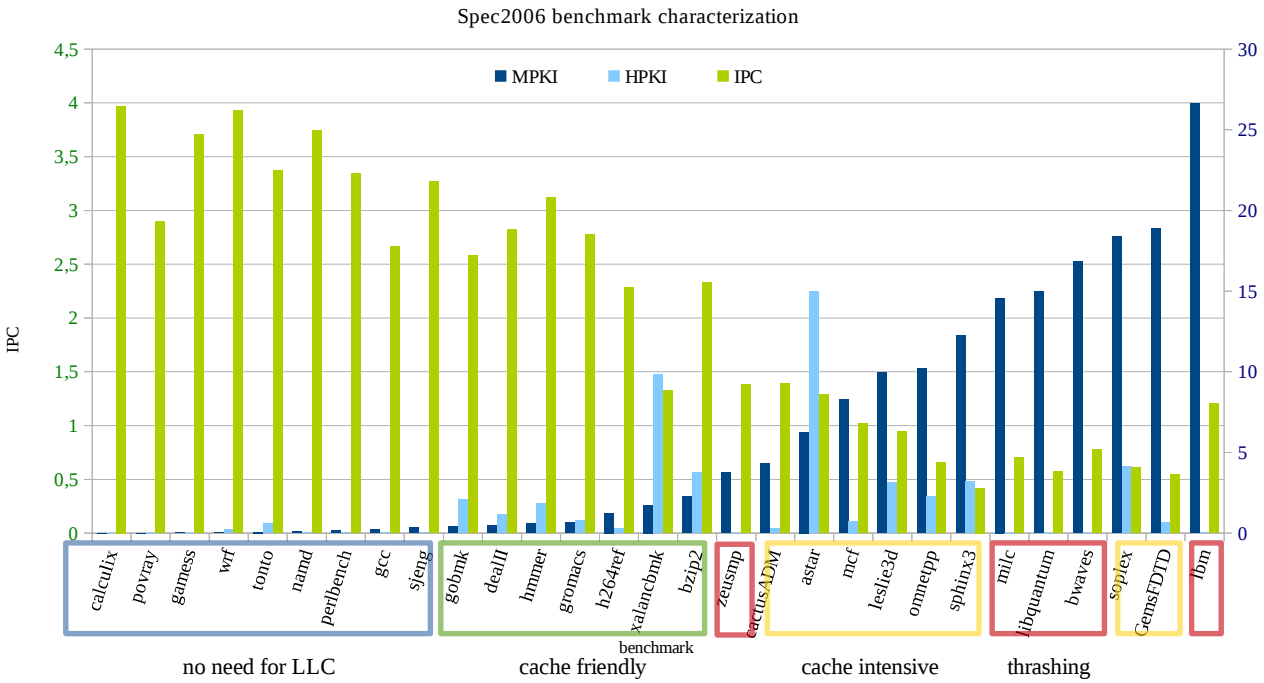
- no need for LLC – Εδώ θα τοποθετήσουμε τα benchmark τα οποία δεν χρειάζονται την L3 διότι έχουν πολύ μικρό αριθμό προσβάσεων στην μνήμη
- cache friendly – Εδώ θα τοποθετήσουμε τα benchmark τα οποία υποβοηθούνται αρκετά από την ύπαρξη της Last-Level Cache
- cache intensive – Εδώ θα τοποθετήσουμε τα benchmark τα οποία υποβοηθούνται αρκετά από την ύπαρξη της Last-Level Cache αλλά έχουν ταυτόχρονα και μεγάλο αριθμό από misses
- thrashing/scanning – Εδώ θα τοποθετήσουμε τα benchmark τα οποία δεν υποβοηθούνται ικανοποιητικά από την Last-Level Cache διότι έχουν η επαναχρησιμοποίηση blocks παρουσιάζεται σε μεγάλη απόσταση

Τα αποτελέσματα είναι από ένα Timing Simulation ανά benchmark χρησιμοποιώντας το CRC kit και προεπιλεγμένο replacement policy το DRRIP 1MB, 16-way Last-Level Cache.

Το κάθε benchmark θα μπει στην κατάλληλη κατηγορία με τους πιο κάτω κανόνες συμβατικά:

- no need for LLC – Εδώ θα τοποθετήσουμε τα benchmarks τα οποία έχουν MPKI (Misses per K instructions) και HPKI (Hits per K instructions) κάτω απο 1
- cache friendly – Εδώ θα τοποθετήσουμε τα benchmarks τα οποία έχουν MPKI κάτω από 4 και HPKI πάνω από 0
- cache intensive – Εδώ θα τοποθετήσουμε τα benchmarks τα οποία έχουν MPKI πάνω από 4 και HPKI πάνω από 0
- thrashing/scanning – Εδώ θα τοποθετήσουμε τα benchmarks τα οποία έχουν HPKI κοντά στο 0

Παρακάτω παρουσιάζονται οι τιμές τις προσομοίωσης για τα IPC (Instructions per cycle), MPKI (Misses per K instructions) και HPKI (Hits per K instructions) για όλα τα benchmarks σε ένα γράφημα. Στον άξονα των benchmarks φαίνεται και η κατηγοριοποίηση των benchmarks στις τέσσερις ομάδες. Τα IPC αναπαριστώνται στον αριστερό ψ άξονα ενώ τα MPKI και HPKI στον δεξιό ψ άξονα.



Σχήμα 5.4: Κατηγοριοποίηση των SpecCPU 2006 σε σχέση με την συμπεριφορά τους στην LLC

Από το πιο πάνω γράφημα παρατηρούμε και εδώ αμέσως ότι γενικά περισσότερα misses φέρουν χαμηλότερη επίδοση.

Μερικά benchmarks θα μπορούσαν να μπουν και σε άλλες κατηγορίες αν δεν είχαμε τόσο αυστηρά μέτρα. Για παράδειγμα το cactusADM θα μπορούσε να έμπαινε στην thrashing κατηγορία.

5.5 Γραφήματα μοναδικών διευθύνσεων

Ένας άλλος τρόπος για να μελετήσουμε την συμπεριφορά του κάθε benchmark όσον αφορά τα αιτήματα που κάνει στην Last-Level Cache είναι να παράξουμε γραφήματα με διευθύνσεις που τις ονομάζουμε με την σειρά που τις βλέπουμε αρχίζοντας από το 0. Με αυτό τον τρόπο μπορούμε να δούμε για την διάρκεια των 250M εντολών των traces μας, αν έχουμε πολλά και κοντινά rereferences ποιοτικά.

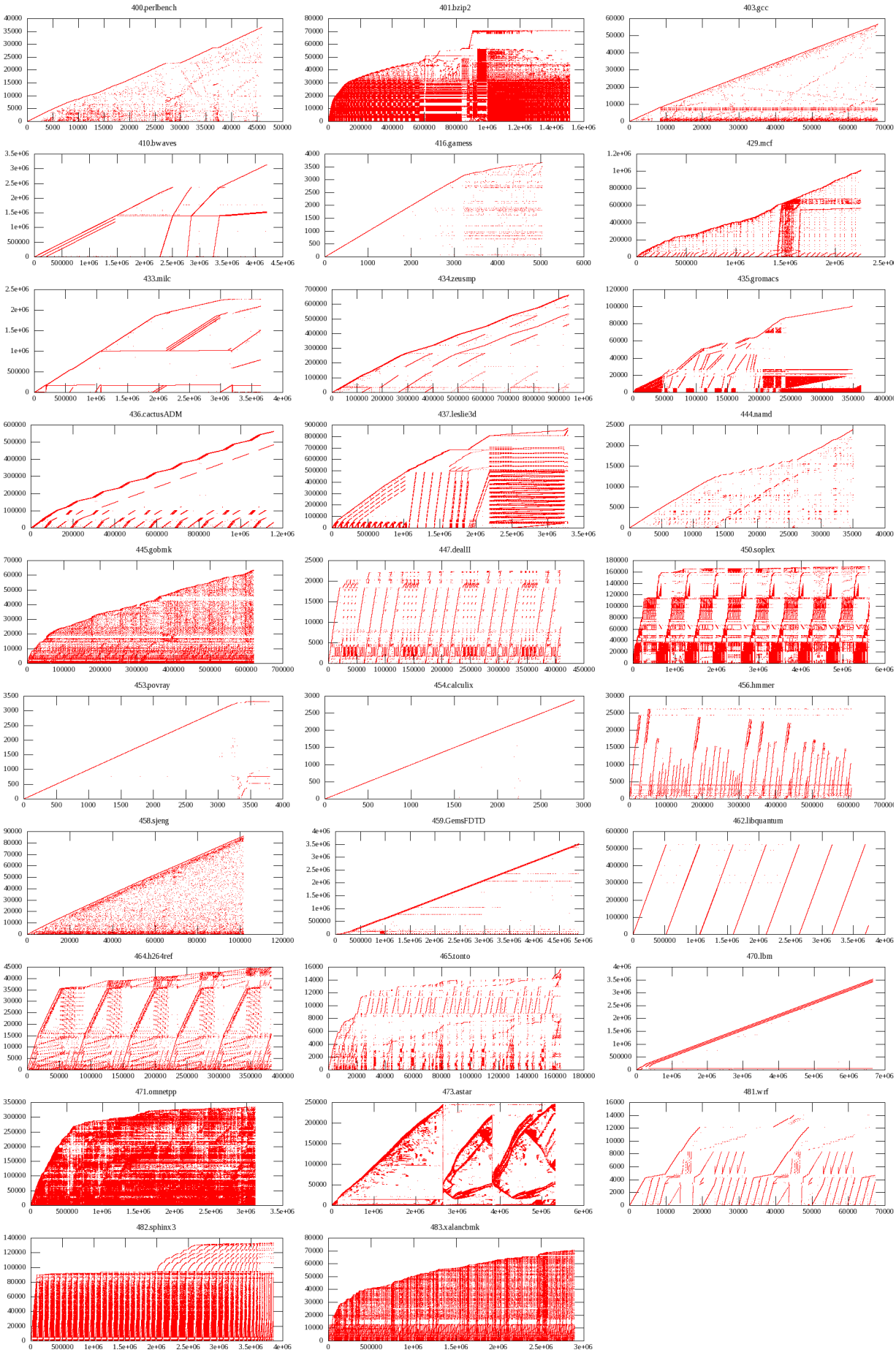
Τα συμπεράσματα από τα γραφήματα αυτά μπορεί να μην μας δίνουν πολύ συγκεκριμένες πληροφορίες αλλά μπορούν να αποτελέσουν επαλήθευση για κάποιες σκέψεις.

Για παράδειγμα για benchmarks όπως το lbm που είναι scanning περιμένουμε να δούμε “κούφιο” γράφημα, με την έννοια ότι επειδή δεν έχουμε πολλά rereferences αλλά μοναδικές διευθύνσεις, θα προκύπτει μια ευθεία από προσβάσεις σε νέες διευθύνσεις μετονομασμένες σε διαδοχικούς αριθμούς ξεκινώντας από το 0.

Άλλο παράδειγμα είναι το xalancbmk το οποίο είναι cache frienly με πολλά hits γι' αυτό πρέπει και να είναι πιο γεμάτο από το lbm αλλά με μικρό αριθμό από μοναδικές διευθύνσεις. Άλλη πιθανότητα είναι να έχει όχι και πολύ γεμάτο γράφημα, να έχει πιο πολλές μοναδικές διευθύνσεις αλλά με κοντινά rerefrences.

Πιο κάτω παρατίθενται τα γραφήματα για όλα τα Benchmarks. Στον ψ άξονα είναι ο αριθμός της μοναδικής διεύθυνσης και στον χ άξονα είναι ο αριθμός της πρόσβασης με χρονολογική σειρά.

Θα μπορούσαμε να πάρουμε περισσότερα συμπεράσματα αλλά δεν θα εστιαστούμε πολύ στη μελέτη των rerefrences.



Σχήμα 5.5: Γραφήματα μοναδικών διευθύνσεων στην LLC για τα SpecCPU 2006 (representative regions)

Κεφάλαιο 6

Design Space των Replacement Policies

6.1 Εισαγωγή

Κύριος στόχος της διπλωματικής αυτής εργασίας είναι ο σχεδιασμός ενός μοντέρνου Last-Level Cache block replacement policy το οποίο να βασίζεται σε σύγχρονες τεχνικές και να πετυχαίνει αύξηση στην επίδοση σε σχέση με το DRRIP και ταυτόχρονα να διατηρεί χαμηλή την πολυπλοκότητα χώρου και λογικής στην υλοποίηση του. Στην διαδικασία σχεδιασμού αυτού του replacement policy εξερευνούμε ένα ευρύ Design Space, δηλαδή ένα μεγάλο σύνολο από διαφορετικά replacement policies που καθορίζονται από συγκεκριμένες παραμέτρους.

Σε αυτό το κεφάλαιο περιγράφονται λεπτομερώς οι παράμετροι οι οποίοι περιγράφουν τα replacement policies που υλοποίησα στον Functional Simulator μου. Όπως προαναφέραμε ο Functional Simulator μου λειτουργεί και ως τμήμα του CRC kit με το οποίο κάνουμε Performance Simulations. Έτσι όσον αφορά τα replacement policies που υλοποίησα είναι ακριβώς τα ίδια και για τους δύο simulators όταν εισάγουμε ίδιους παραμέτρους. Αυτό είναι σημαντικό διότι όπως θα δούμε στη μεθοδολογία έχει μεγάλη αξία το ότι για κάθε replacement policy υπάρχει επιλογή για Functional και Timing προσομοιώσεις που χρησιμοποιούν την ίδια υλοποίηση του replacement policy.

Πολλές από τις παραμέτρους περιγράφονται στο paper του DRRIP [1] γι' αυτό και θεωρώ ότι ο κώδικάς μου προσομοιώνει παραλλαγές του DRRIP αλλά και το ίδιο όταν εισάγουμε τις αντίστοιχες τιμές των παραμέτρων. Παρόλα αυτά διευρύνουμε το Design Space με μεγαλύτερο εύρος τιμών, χαρακτηριστικά από άλλα έργα και κάποιες δικές μου προσθέσεις λειτουργιών που πιστεύω ότι θα βοηθούσαν ένα replacement policy.

Επιπλέον μελετούμε και κάποια ακόμα χαρακτηριστικά τα οποία είναι ακριβά στο υλικό και δεν προτίθενται για υλοποίηση. Ο λόγος είναι διότι η μελέτη τους θα εξακριβώσει αν αξίζουν να μελετηθούν περαιτέρω για να γίνουν γνώμονας στον σχεδιασμό άλλων replacement policies.

Ο κώδικάς μου υλοποιεί τεράστιο αριθμό από συνδυασμούς παραμέτρων για replacement policy. Αρχικά θα περιγράψουμε τι καθορίζει αυτούς τους συνδυασμούς, δηλαδή τα χαρακτηριστικά της κάθε παραμέτρου και που βρίσκουν εφαρμογή στον αλγόριθμο του replacement policy. Ακολούθως θα περιορίσουμε τους συνδυασμούς και θα μελετήσουμε ένα αρκετά μεγάλο υποσύνολο.

Τις παραμέτρους που καθορίζουν τα replacement policies τις κατηγοριοποιούμε σε δύο κατηγορίες. Σε παραμέτρους που είναι σχετικές με τον αλγόριθμο των dueling policies, δηλαδή των εσωτερικών replacement policies, και σε παραμέτρους που είναι σχετικές με τον μηχανισμό εναλλαγής των εσωτερικών replacement policies. Με άλλα λόγια η πρώτη κατηγορία ασχολούνται με τις ενημερώσεις των τιμών των RRPV στο κάθε block και η δεύτερη κατηγορία στο πώς υλοποιούνται τα Set Dueling Monitors.

6.2 Παράμετροι που καθορίζουν τα dueling policies

Οι παράμετροι που καθορίζουν τα dueling policies παρουσιάζονται μια φορά για το κάθε επί μέρους replacement policy. Επομένως αρκεί να δούμε την κάθε παράμετρο μια φορά και θα εννοείται ότι είναι παρόν και για τα (προς το παρόν) 2 επί μέρους replacement policies.

Εδώ θα μελετήσουμε τις παραμέτρους (1) insertion position, (2) bimodal insertion, (3) ε, (4) random insertion position, (5) promotion, (6) others demotion on insertion, (7) others demotion on hit on RRPV0, (8) RRPV bits.

- insertion position – Αυτή η παράμετρος καθορίζει την τιμή του RRPV (δηλαδή της προτεραιότητας για αντικατάσταση) που θα πάρει το κάθε νέο block με την εισαγωγή του στην Last-Level Cache.
  - Όταν το bimodal insertion ισούται με False τότε το πιο πάνω ισχύει για το 100% των insertions
  - Πεδίο τιμών: $[0,3] \cap \mathbb{Z}$ για RRPV bits = 2,
    - $[0,2^n - 1] \cap \mathbb{Z}$ για RRPV bits = n
- bimodal insertion – Αυτή η παράμετρος καθορίζει αν το insertion position θα εναλλάσσεται με το random insertion position. Με άλλα λόγια καθορίζει αν το replacement policy είναι Static (όπως το SRRIP) ή Bimodal (όπως το BRRIP)
  - Πεδίο τιμών: False, True
- ε – Αυτή η παράμετρος καθορίζει την πιθανότητα με την οποία θα χρησιμοποιείται το random insertion position στη θέση του insertion position
  - Πεδίο τιμών: [0,1]
- random insertion position – Αυτή η παράμετρος καθορίζει την τιμή του RRPV (δηλαδή της προτεραιότητας για αντικατάσταση) που θα πάρει το κάθε νέο block με την εισαγωγή του στην Last-Level Cache με πιθανότητα ε.
  - Όταν το bimodal insertion ισούται με True τότε το πιο πάνω ισχύει για το ε των insertions
  - Πεδίο τιμών: $[0,3] \cap \mathbb{Z}$ για RRPV bits = 2,
    - $[0,2^n - 1] \cap \mathbb{Z}$ για RRPV bits = n
- promotion – Αυτή η παράμετρος καθορίζει αν θα χρησιμοποιηθεί Hit ή Frequency Promotion [1]
  - Πεδίο τιμών: (Hit), (Frequency)
  - Hit Promotion είναι το χαρακτηριστικό του DRRIP που θέτει την τιμή του RRPV να είναι ίση με μηδέν σε κάθε block όταν έχει hit
  - Frequency Promotion έχουμε όταν ο αλγόριθμος του replacement policy μειώνει την τιμή του RRPV κατά 1 σε κάθε block όταν έχει hit

        - Σύμφωνα με την πηγή [1] έχει γίνει μια μελέτη του Frequency Promotion αλλά επιλέχθηκε να μην χρησιμοποιηθεί λόγο της καλύτερης επίδοσης του Hit Promotion στα πλαίσια των προσομοιώσεων που έκαναν. Ο λόγος που επέλεξα να το μελετήσω και εγώ είναι διότι εισάγουμε και άλλες παραμέτρους και σε συνδυασμό με αυτές μπορεί να αποδειχθεί κερδοφόρο.
- others demotion on insertion – Η παράμετρος καθορίζει αν θα ενεργοποιηθεί η λειτουργία “others demotion on insertion”
    - others demotion on insertion έχω ονομάσει την επιπρόσθετη διαδικασία αύξησης των RRPVs κατά 1 των υπόλοιπων ways στο set με το νέο block που εισέρχεται
    - Πεδίο τιμών: $[0,2] \cap \mathbb{Z}$
        - 0 σημαίνει ότι δεν έχουμε others demotion on insertion
        - 1 σημαίνει ότι έχουμε others demotion on insertion για RRPVs < $(2^n - 1) - 1$, δηλαδή για RRPVs < 2 όταν RRPV bits = 2
        - 2 σημαίνει ότι έχουμε others demotion on insertion για RRPVs < $(2^n - 1) - 2$, δηλαδή για RRPVs < 3 όταν RRPV bits = 2
- others demotion on hit on RRPV0 – Η παράμετρος καθορίζει αν θα ενεργοποιηθεί η λειτουργία “others demotion on hit on RRPV0”
    - others demotion on hit on RRPV0 έχω ονομάσει την επιπρόσθετη διαδικασία αύξησης των RRPVs κατά 1 των υπόλοιπων ways στο set όπου υπάρχει hit σε 1 block
    - Πεδίο τιμών: False, True
- RRPV bits – Η παράμετρος καθορίζει τον αριθμό των bits σε κάθε RRPV κάθε block στην Last-Level Cache. Έχει αποδειχθεί ότι 2 bits είναι αρκετά [1].
    - Πεδίο τιμών: $[2,30] \cap \mathbb{Z}$

Ο λόγος που σχεδίασα τα “others demotion on insertion” και “others demotion on hit on RRPV0” είναι για να μπορεί να υπάρξει καλύτερη αναπαράσταση του LRU με RRPVs των 2 bits. Όπως εξηγήθηκε σε προηγούμενο κεφάλαιο, αν έχουμε insertion position να είναι 0 στο SRRIP προκύπτει NRU, έτσι το 1 από τα 2 bits του κάθε RRPV είναι άχρηστο. Ένας άλλος λόγος είναι για να προσομοιωθούν περισσότερα bits στα RRPVs.

Σημειώνεται ότι αυτοί οι δύο λόγοι είναι υποθέσεις και δεν έχω αποδείξει ότι πράγματι ισχύουν, αλλά μας ενδιαφέρει αν σχεδιαστεί κάποιο αποδοτικό policy ακόμα και με απλές προσθέσεις σαν αυτές στον αλγόριθμο.

Παρατίθεται ο νέος αλγόριθμος για το BRRIP με τις σωστές τιμές για τις πιο πάνω παραμέτρους

insertion_position = 3
bimodal_insertion = True
ε = 1/32
random_insertion_position = 2
promotion = Hit
others_demotion_on_insertion = 0
others_demotion_on_hit_on_rrpv0 = False
RRPVbits = 3

- Σε κάθε miss
  - Εν όσο δεν υπάρχουν RRPVs ίσα με $2^{RRPVbits}$ - 1 στο set
    - Αύξησε όλα τα RRPVs του set κατά 1
  - Αντικατέστησε το αριστερότερο block με RRPV ίσο με $2^{RRPVbits} – 1$
  - Αν bimodal_insertion
    - Τότε, θέσε το RRPV του καινούριου block να είναι ίσο με insertion_position με πιθανότητα 1-ε και ίσο με random_insertion_position με πιθανότητα ε
    - Διαφορετικά, θέσε το RRPV του καινούριου block να είναι ίσο με insertion_position
  - Αν others_demotion_on_insertion > 0
    - Αύξησε κατά 1 τα RRPVs των υπόλοιπων block που έχουν τιμή $< (2^n - 1) - 2 +$ others_demotion_on_insertion
- Σε κάθε hit
  - Αν promotion == Hit
    - Θέσε το RRPV του αναφερθέντος block να είναι ίσο με 0

- Αν promotion == Frequency
  - Μείωσε κατά 1 το RRPV του αναφερθέντος block
- Αν others_demotion_on_hit_on_rrpv0
  - Αύξησε κατά 1 τα RRPVs των υπόλοιπων block

6.3 Set Dueling Monitor μοντέλα

Εδώ περιγράφουμε τις επιλογές του προσομοιωτή σχετικά με το ποιο μηχανισμό υλοποιεί για επιλογή των επιμέρους policies δυναμικά την ώρα της εκτέλεσης.

6.3.1 Εναλλαγή μεταξύ δύο policies

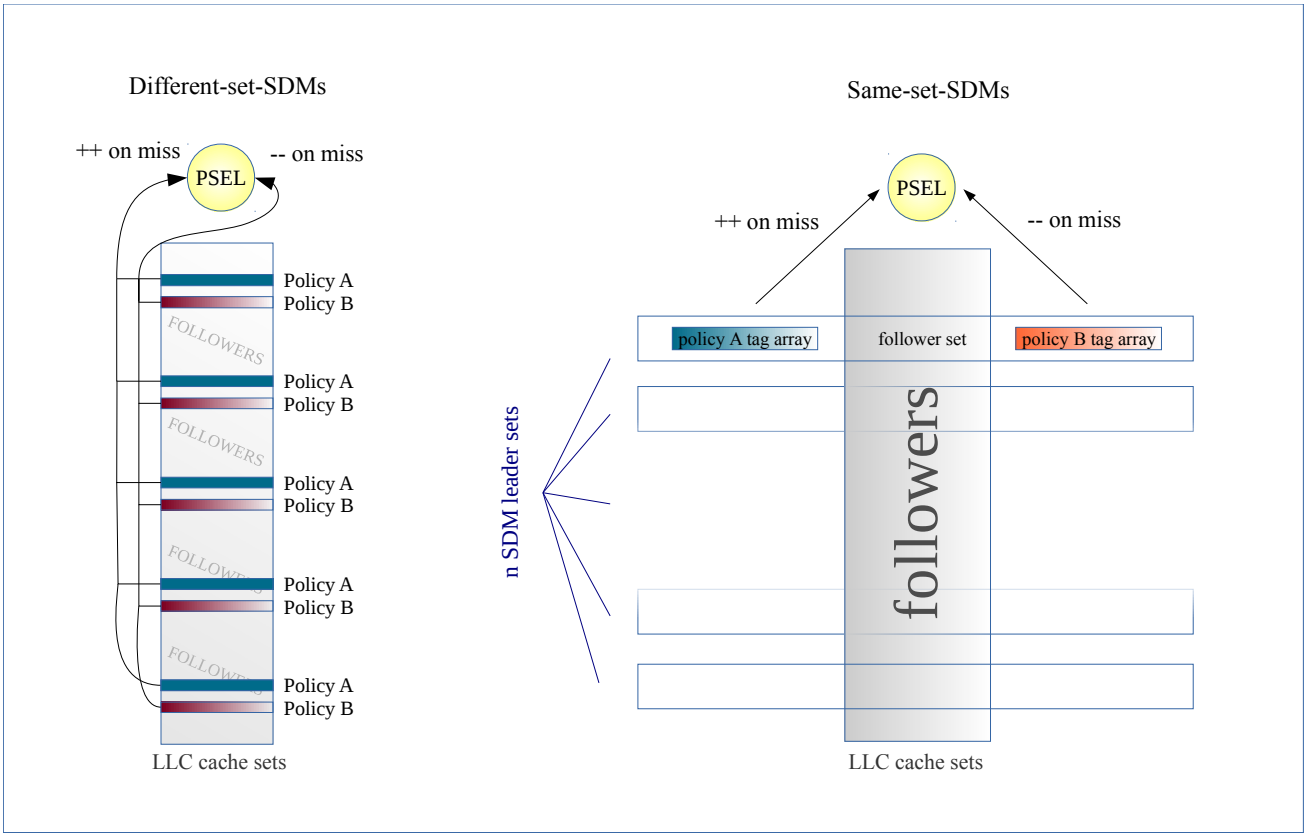
Το DRRIP χρησιμοποιεί ένα συγκεκριμένο μοντέλο [1] αυτό του DIP [9] κατά το οποίο τα SDMs (Set Dueling Monitors) βρίσκονται μέσα στα sets της Last-Level Cache έχοντας μικρό αντίκτυπο στην επίδοση αλλά μικρότερο κόστος παρά όταν ήταν έξω [9]. Εμείς θα αναφερόμαστε σε αυτό το μοντέλο με την ονομασία different-set-SDMs.

Για το μοντέλο different-set-SDMs η κατανομή των SDM sets είναι όπως περιγράφεται στο DIP [9]. Δηλαδή για μια cache 1024 sets, θέσε το set 0 και για κάθε set με απόσταση 33, να χρησιμοποιεί το policy A. Θέσε κάθε 31ο set από το 0 να χρησιμοποιεί το policy B.

Εμείς θα μελετήσουμε και την περίπτωση που τα SDMs είναι έξω από την Last-Level Cache για να δούμε την επίδραση στην επίδοση των δικών μας replacement policies. Ονομάζεται “DIP-Global” [16] αλλά εμείς θα αναφερόμαστε σε αυτό με την ονομασία same-set-SDMs διότι θα το προσαρμόσουμε στα δικά μας replacement policies.

Για το μοντέλο same-set-SDMs θα βάζουμε δύο (ένα για κάθε policy) εξωτερικά SDM sets (μόνο tags ανά block) σε κάθε sets τα οποία θα ήταν για το policy A στο μοντέλο different-set-SDMs. Έτσι θα αντιστοιχεί ίδιος αριθμός από SDM sets ανά policy για κάθε ένα από τα μοντέλα same-set-SDMs και different-set-SDMs.

Πιο κάτω παρουσιάζεται σχηματική απεικόνιση των δύο μοντέλων. Ένα από τα θεωρητικά πλεονεκτήματα του same-set-SDMs είναι ότι στην περίπτωση που υπάρχει ανομοιογένεια στην κατανομή των accesses σε κάθε set, δίνει πιο αντιπροσωπευτική επιλογή για το καλύτερο policy. Αυτό ισχύει διότι για γίνεται σύγκριση για τα ίδια accesses στο κάθε policy στο κάθε set με SDM.



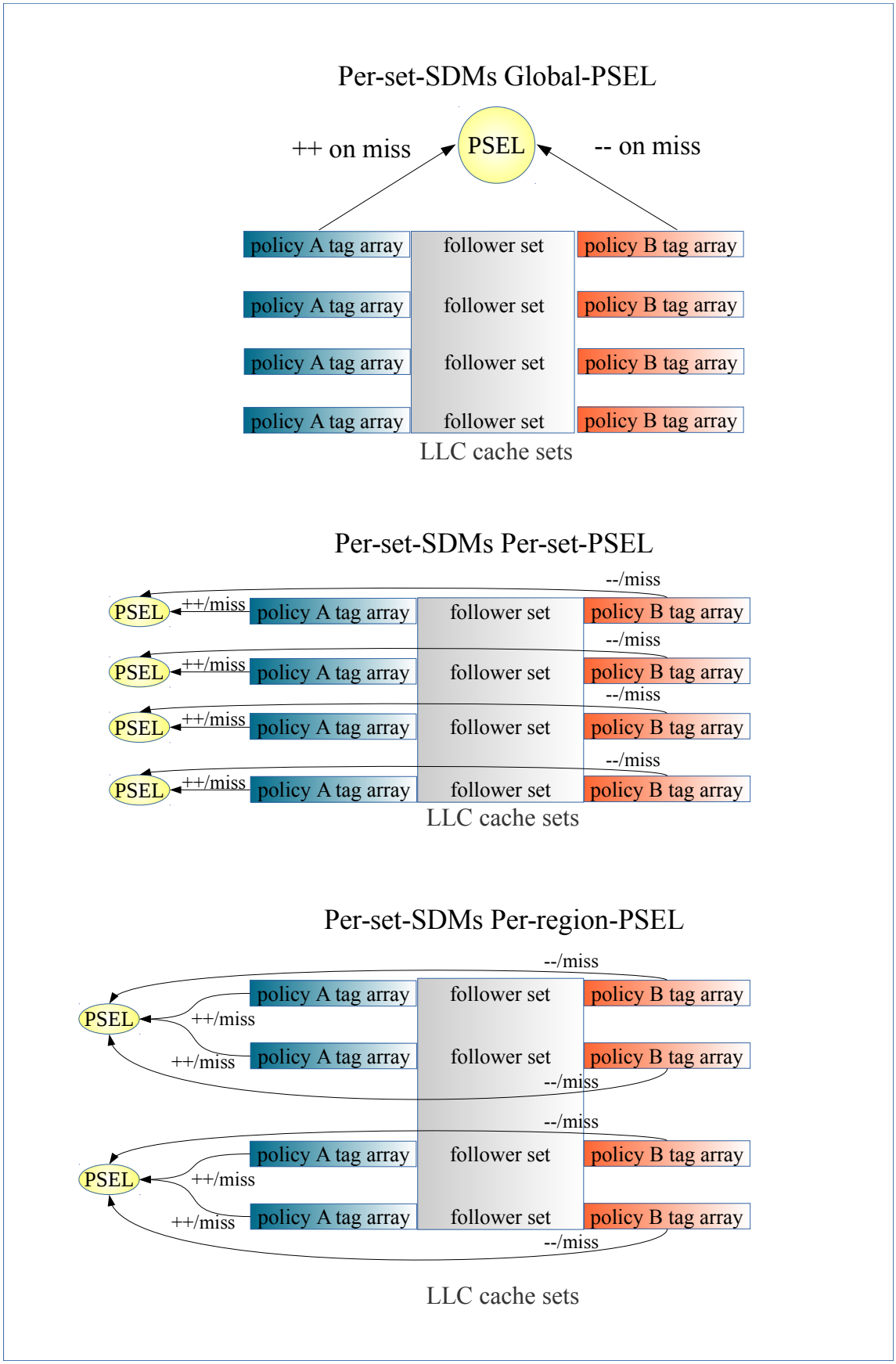
Σχήμα 6.1: Σχηματική Αναπαράσταση των same-set-SDMs και different-set-SDMs μοντέλων

Επιπλέον εξετάζουμε τρεις άλλους μηχανισμούς επιλογής μεταξύ δύο policies στους οποίους έχουμε SDM sets ένα για κάθε policy για κάθε set, εξωτερικά της Last-Level Cache. Η ιδέα βασίζεται στο μηχανισμό "TSEL-Global" [17] του οποίου το "DIP-Global" θεωρείται βελτιστοποίηση διότι καταλαμβάνει σημαντικά λιγότερο υλικό χρησιμοποιώντας δειγματοληψία [16].

Οι τρεις μηχανισμοί πολύ λίγο διαφέρουν μεταξύ τους. Η διαφορά είναι στο πόσα PSEL counters υπάρχουν και συνεπώς ποιο set αντιστοιχεί σε ποιο PSEL counter. Ακολουθούν οι περιγραφές των τριών μηχανισμών:

- Per-set-SDMs Global-PSEL: Έχουμε μόνο ένα PSEL counter το οποίο ενημερώνεται από όλα τα SDMs και από αυτό λαμβάνεται απόφαση ποιο policy θα χρησιμοποιούν όλα τα folloer sets
- Per-set-SDMs Per-set-PSEL: Σε κάθε Last-Level Cache set αντιστοιχούν δύο SDM sets και ένα PSEL counter. Έτσι οι αποφάσεις κάθε LLC set είναι ανεξάρτητη των υπολοίπων LLC sets. ("TSEL" [17])
  - Πιθανό μειονέκτημα: είναι πολύ ευαίσθητος μηχανισμός σε noise και ταυτόχρονα άλλα sets δεν μπορούν να υποβοηθήσουν τις αποφάσεις άλλων
- Per-set-SDMs Per-region-PSEL: Έχουμε n PSEL counters στους οποίους αντιστοιχούν το 1/n όλων των Last-Level Cache sets το καθένα.
  - Πιθανά Μειονεκτήματα:
    - Δεν μπορούμε να αποφασίσουμε σίγουρα ποιο n είναι το βέλτιστο
    - Στο θέμα του ποιό set κάνει map πού, εδώ το έβαλα να είναι συνεχόμενα τα set του ίδιο region, όμως για αξιολόγηση είναι τεράστιος ο αριθμός των mappings

Ο λόγος που το μελετούμε είναι ότι συγκρίνοντας τους στους αλγόριθμους μας μπορεί να πάρουμε απάντηση κατά πόσο αξίζει να δώσουμε έμφαση στην ανομοιογένεια αναμεταξύ των LLC sets όσον αφορά την προτίμηση σε policy, αν υπάρχει. Όσον αφορά την υλοποίηση είναι σχετικά πολύ αρκετά ακριβά, παρόλο που τα SDMs είναι εξωτερικά και δεν έχουν δεδομένα αλλά μόνο block tags.

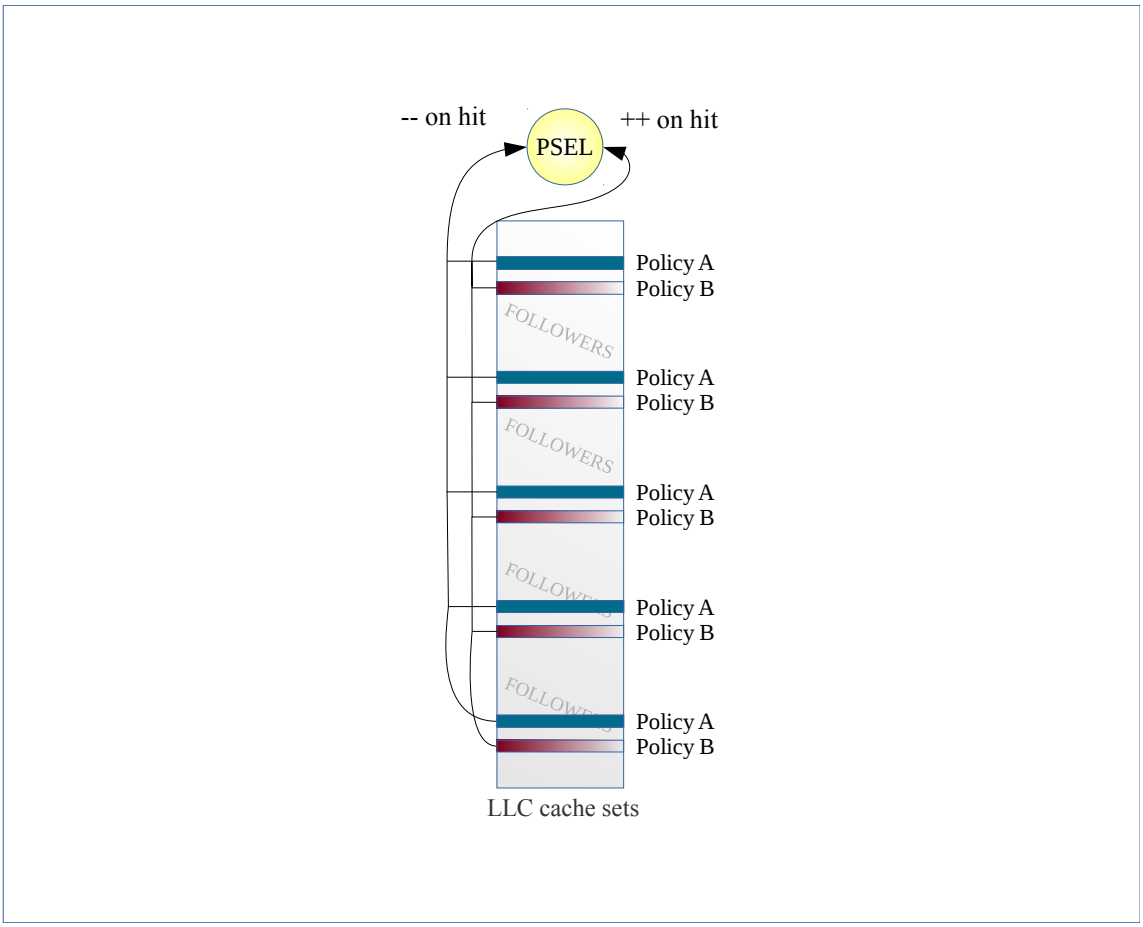


Σχήμα 6.2: Σχηματική Αναπαράσταση των Per-set-SDMs Global-PSEL, Per-set-SDMs Per-set-PSEL και Per-region-PSEL για υποθετική LLC

6.3.2 PSEL trigger

Τέλος μελετούμε την παραλλαγή των πιο πάνω στην περίπτωση όπου οι αναβαθμίσεις των PSELs γίνονται με βάση τα hits αντί τα misses. Στην περίπτωση των "Hit Triggered PSELs" όταν έχουμε hit, τα PSEL αναβαθμίζονται από την αντίθετη κατεύθυνση καθώς το hit προμηνύει προτίμηση από την πλευρά από όπου έχει συμβεί.

Η επιλογή "Hit Triggered PSELs" βρίσκει εφαρμογή μόνο στο "different-set-SDMs" διότι σε όλα τα υπόλοιπα συμπεριφέρεται λογικά το ίδιο με το "Miss Triggered PSELs". Αυτό αληθεύει διότι στα υπόλοιπα έχουμε ταυτόχρονη πρόσβαση στα PSELs από 2 SDM sets και η μόνη περίπτωση που αναβαθμίζονται είναι όταν έχουμε hit/miss ή miss/hit. Η άλλη περίπτωση είναι όταν έχουμε hit/hit ή miss/miss στην οποία και όταν είχαμε "Miss Triggered PSELs" η τιμή του PSEL παραμένει η ίδια.



Σχήμα 6.3: Σχηματική Αναπαράσταση του "different-set-SDMs" με Hit Triggered PSELs για υποθετική LLC

6.4 Παράμετροι σχετικά με τα Set Dueling Monitors

Ο κώδικάς μου έχει τις ακόλουθες παραμέτρους σχετικά με την υλοποίηση των Set Dueling Monitors:

- numberOfPSELs: Η τιμή της μεταβλητής αυτής καθορίζει ποιο από τα πιο πάνω μοντέλα θα υλοποιηθεί και από πόσα policies θα αποτελείται
  - Πεδίο τιμών: $[-3, n] \cap \mathbb{Z}$ για n Last-Level Cache sets
    - -1: same-set-SDMs (2 policies)
    - 0: different-set-SDMs (2 policies)
    - 1: Per-set-SDMs Global-PSEL (2 policies)
    - [2, n-1]: Per-set-SDMs Per-Region-PSELs (2 policies)
    - n: Per-set-SDMs Per-set-PSELs (2 policies)
- PSELbits: [1-30] Μέγεθος των PSEL σε bits για όλα τα μοντέλα
- PSELTrigger: Παράμετρος που καθορίζει κατά πόσο οι αναβαθμίσεις των PSEL γίνονται με τα misses ή με τα hits
  - Πεδίο τιμών: “Miss”, “Hit”
- Initial PSEL value: Παράμετρος που καθορίζει την αρχική τιμή των μετρητών PSEL
  - Πεδίο τιμών: “zero”(0), “middle” ($2^{(\text{PSEL bits}) - 1}$)

6.5 Functional Simulations για ένα μεγάλο υποσύνολο

Δεν θα τρέξουμε προσομοιώσεις για όλους τους συνδυασμούς των τιμών των πιο πάνω παραμέτρους διότι ο αριθμός αυτός είναι τεράστιος. Για κάποιες από τις παραμέτρους θα μειώσουμε συμβατικά το πεδίο τιμών, για κάποιες άλλες θα εξεταστούν μόνο οι τιμές που αντιστοιχούν στο DRRIP ενώ για κάποιες άλλες θα μελετάται ολόκληρο το πεδίο τιμών.

Σημειώνεται επίσης ότι ο αριθμός των μοναδικών συνδυασμών που θα προκύψει δεν είναι πολύ εύκολα υπολογίσιμος διότι κάποιες τιμές για μερικές τιμές αναιρούν άλλες

παραμέτρους. Για παράδειγμα όταν έχουμε bimodal_insertion = True τότε οι τιμές των ε και random insertion position δεν θα επηρεάζουν το αποτέλεσμα των προσομοιώσεων.

Επιπλέον, δεν θα ασχοληθούμε τώρα με τους μηχανισμούς εναλλαγής μεταξύ πολλαπλών policies (multiple policy different-set-SDMs και multiple policy same-set-SDMs) στο σημείο αυτό λόγω του πολύ μεγάλου αριθμού συνδυασμών παραμέτρων. Αντιθέτως, θα προσπαθήσουμε να πάρουμε πληροφορίες για των σχεδιασμό του βέλτιστου μηχανισμού εναλλαγής μεταξύ πολλαπλών policies μέσω πειραμάτων με μηχανισμούς εναλλαγής μεταξύ μόνο 2 policies.

Οι προσομοιώσεις θα είναι προς το παρόν Functional Simulations διότι μας ενδιαφέρει η εξερεύνηση ενός όσο το δυνατό πιο μεγάλου Design Space. Τα Timing Simulations όπως τα διεκπεραιώνουμε με αυτά τα traces είναι περίπου 120 φορές πιο αργά από τα αντίστοιχα Functional Simulations.

Οι τιμές που θα εξερευνηθούν με Functional Simulations βρίσκονται στο πίνακα 6.1.

Το σύνολο συνδυασμών που προκύπτουν από τις πιο πάνω αναθέσεις είναι 333600. Ένα set προσομοιώσεων από 29 benchmarks παίρνει χρόνο ένα λεπτό. Έτσι ο χρόνος που χρειάζεται η εξερεύνηση αυτού του Design Space είναι 5560 core hours. Τις προσομοιώσεις όπως θα δούμε πιο μετά θα τις τρέξουμε παράλληλα σε ένα High Performance Computing facility για να μειώσουμε τον χρόνο αυτό από ~230 μέρες για ένα πυρήνα σε ~1.5 μέρες.

Παράμετρος	Τιμές	Σχόλια
numberOfPSELs, PSELTrigger	-1, N/A 0, “miss” 0, “hit” 1, N/A 1024, N/A	same-set-SDMs different-set-SDMs different-set-SDMs Per-set-SDMs Global-PSEL Per-set-SDMs Per-set-PSELs
PSELbits	1 2 3 4 5 6 7 8 9 10	PSEL counter size
Initial PSEL value	0 middle	
Promotion	0, 0 0, 1 1, 0 1, 1	(policyA, policyB)
Others demotion on insertion	0, 0 0, 1 1, 0 1, 1	(policyA, policyB)
Others demotion on hit on RRPV0	0, 0 0, 1 1, 0 1, 1	(policyA, policyB)
ε	1/32	(μία τιμή μόνο)
RRPV_bits	2	(μία τιμή μόνο)
Συνδιασμοί insertion position και random insertion position	0, 2, 3, 2 3, 0, 3, 0 2, 2, 3, 2 0, 0, 3, 0 3, 1, 3, 1 3, 0, 0, 0 0, 0, 2, 0 0, 1, 3, 1 0, 0, 0, 0 2, 0, 0, 0	(policyA, policyB)

Πίνακας 6.1: Desing Space των replacement policies που εξερευνούμε πειραματικά

Κεφάλαιο 7

Performance Modeling

7.1 Εισαγωγή

Analytical Performance Modeling είναι η μέθοδος με την οποία μελετούνται τα χαρακτηριστικά της συμπεριφοράς ενός συστήματος έτσι ώστε να προσεγγίζεται η επίδοση του συστήματος με χρήση άλλων παραμέτρων. Έτσι αποφεύγεται η χρήση πειραμάτων επίδοσης που είναι πιο χρονοβόρα και πιο ακριβή. [19]

Στη δική μας περίπτωση το σύστημα είναι ο Timing Simulator μας, δηλαδή το CRC kit. Με τη χρήση Performance Modeling θα προσπαθήσουμε να συσχετίσουμε τα LLC MPKI (Last-Level Cache Misses per K Instructions) με το CPI (Clocks per Instruction), που είναι το μετρικό της επίδοσης. Τις τιμές αυτές θα μας τις δώσουν προσομοιώσεις χρησιμοποιώντας το CRC kit και βάσει το Performance Model που θα προκύψει θα μπορούμε να προβλέπουμε το CPI λαμβάνοντας μόνο τη τιμή των MPKI από τον Functional Simulator.

Ο αριθμός των Timing Simulations που τρέξαμε για κάθε πείραμα είναι 300 σε ένα μικρό computing cluster του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου. Κάναμε εξαίρεση για τα benchmarks lbm, libquantum και bzip2 για τα οποία κάναμε 7158 προσομοιώσεις διότι φάνηκε ότι χρειαζόμασταν περισσότερη ακρίβεια.

Τα δείγματα των 300 ή 7158 προσομοιώσεων για κάθε benchmark προέκυψαν από τυχαίους μοναδικούς συνδυασμούς των παραμέτρων του κώδικά μου για τα χαρακτηριστικά της Last-Level Cache. Επιπλέον η επιλογή έγινε ανεξάρτητα για το κάθε benchmark, έτσι τα 300 replacement policies διαφέρουν ως σύνολο από benchmark σε benchmark.

7.2 Συσχετισμός MPKI με CPI

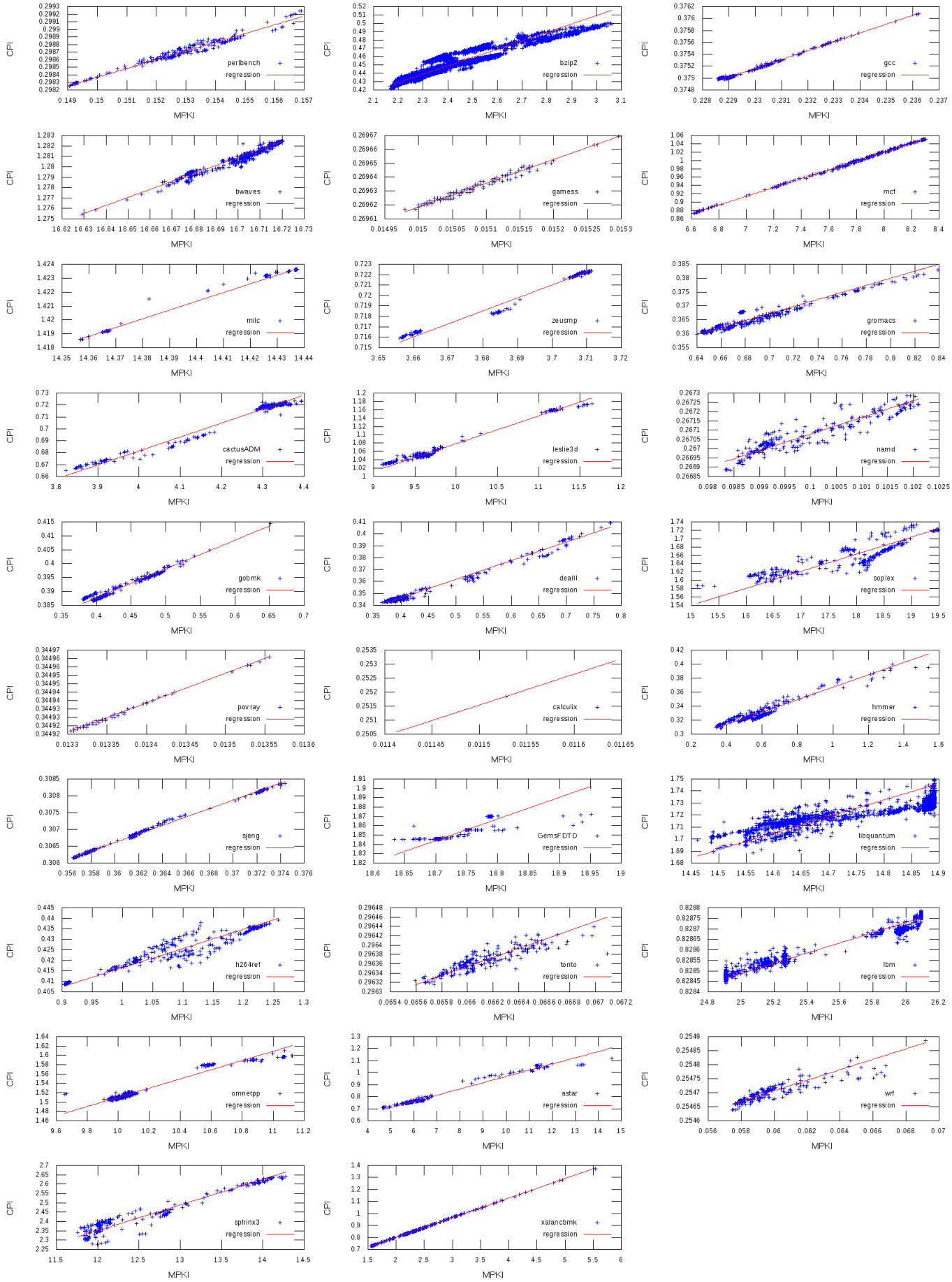
Η μέθοδος με την οποία έγινε ο συσχετισμός είναι με τη χρήση Simple Linear Regression. Με το Simple Linear Regression μπορούμε να παράξουμε πρωτοβάθμιες εξισώσεις από δεδομένα για τιμές χ και ψ, που στην περίπτωσή μας είναι τα MPKI και CPI. Με άλλα λόγια θα κάνει fitting στην πλησιέστερη ευθεία με βάσει τα σημεία σε καρτεσιανό σύστημα αξόνων σε 2 διαστάσεις.

Όπως θα παρατηρήσουμε από τα γραφήματα δεν είναι όλα τα benchmarks που ακολουθούν μια ευθεία γραμμή στη σχέση μεταξύ MPKI και CPI. Θα μπορούσαμε να χρησιμοποιήσουμε πιο πολύπλοκες Regression τεχνικές οι οποίες λαμβάνουν υπόψη τυχών outliers, δηλαδή μια μειοψηφία σημείων απέχουν πολύ από την ευθεία που σχηματίζεται από τα υπόλοιπα.

Άλλες Regression τεχνικές μπορούν να παράξουν εξισώσεις μεγαλύτερου βαθμού από ένα, όμως όπως θα παρατηρήσουμε από τα γραφήματα δεν φαίνεται να ακολουθείται κάπου τέτοια εξίσωση.

Ένας άλλος παράγοντας που δεν θα κάνουμε πιο πολύπλοκη ανάλυση είναι διότι κάποια benchmarks στα οποία φαίνονται ότι υπάρχουν πολλοί outliers, για παράδειγμα το lbm, έχουν πολύ μικρή διακύμανση στο CPI. Επομένως η λάθος προσέγγιση CPI τιμών σε αυτά τα benchmarks θα έχει μικρή επίδραση στην προβλεπόμενη επίδοση του replacement policy.

Πιο κάτω βλέπουμε τις ευθείες που συσχετίζουν τα MPKI και CPI για κάθε benchmark.



Σχήμα 7.1: Performance Modeling χρησιμοποιώντας Regression για συσχετισμό MPKI και CPI για κάθε benchmark

Όλες οι εξισώσεις που προέκυψαν έχουν τη ακόλουθη μορφή:

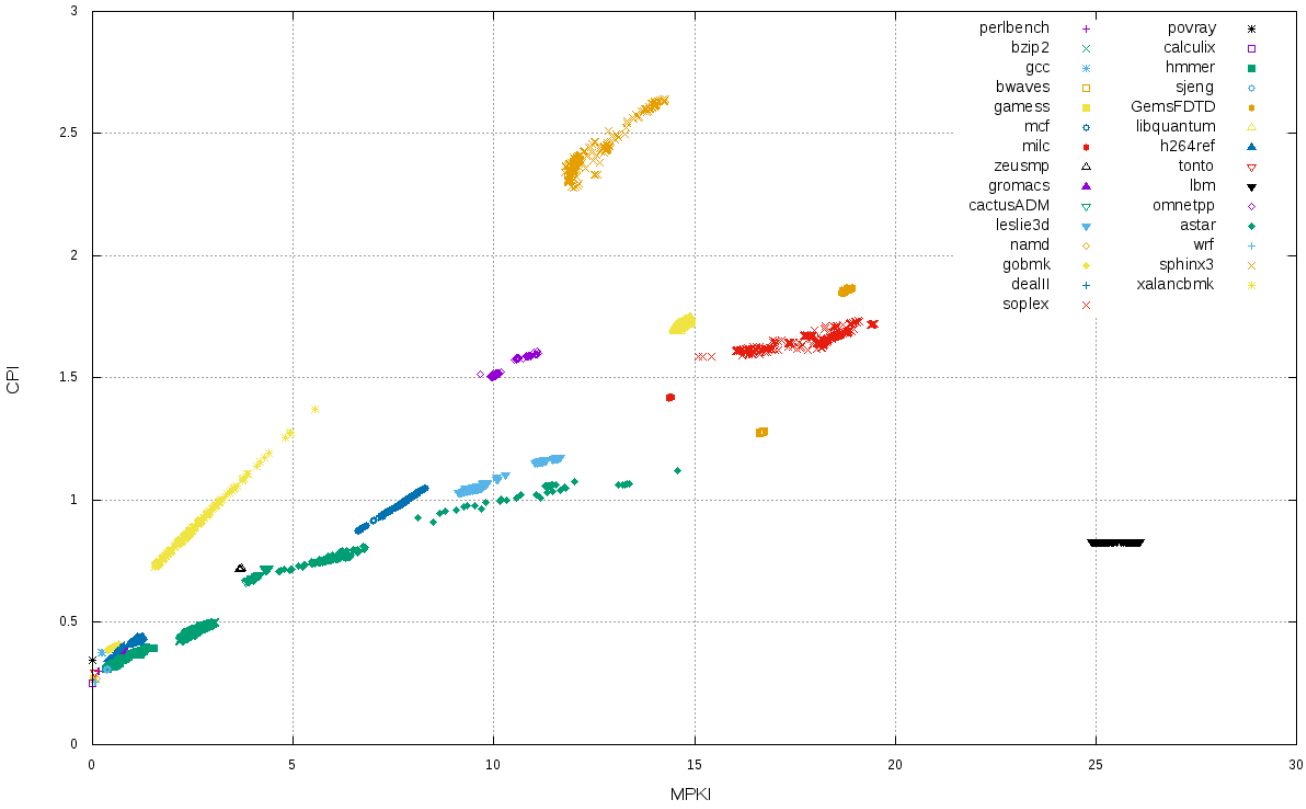
$$CPI_{benchmark}(MPKI_{benchmark}) = a \cdot MPKI_{benchmark} + c$$

Στον ακόλουθο πίνακα βλέπουμε τις τιμές για τα a και c του Performance Model μας.

Benchmark	a	c
perlbench	0,1152115447	0,2810860463
bzip2	0,1002030408	0,2085902433
gcc	0,1485115592	0,3410055887
bwaves	0,0766181352	0,0013606542
gamess	0,1713953611	0,2670475025
mcf	0,1056796625	0,1743114075
milc	0,0627654798	0,5175082893
zeusmp	0,1241955006	0,2614912671
gromacs	0,1268078839	0,2785136351
cactusADM	0,1170651169	0,213148718
leslie3d	0,0664888386	0,4138322214
namd	0,0897079935	0,2581088752
gobmk	0,1032813872	0,3464460068
dealII	0,155797196	0,283965264
soplex	0,0408152217	0,927964028
povray	0,1762515756	0,342576813
calculix	11	0,125039
hmmer	0,0884748955	0,2784431495
sjeng	0,1260972304	0,2612144694
GemsFDTD	0,231692004	-2,4886698281
libquantum	0,1380526347	-0,3106678198
h264ref	0,091207685	0,3252874764
tonto	0,0970235365	0,2899505649
lbm	0,0002438879	0,8223896649
omnetpp	0,0984053781	0,5259819658
astar	0,0517988147	0,4501052655
wrf	0,0187180782	0,2535838491
sphinx3	0,137264149	0,704056962
xalancbmk	0,1632557666	0,4745430943

Πίνακας 7.1: Οι τιμές για τα a και c που προκύπτουν για το Performance Model

Επιπλέον είναι ενδιαφέρον να δούμε και πώς συγκρίνονται τα σημεία του κάθε benchmark με τα σημεία των υπολοίπων benchmarks. Πιο κάτω παρατίθεται γράφημα που συνδιάζει όλα τα σημεία των benchmarks στους ίδιους άξονες. Με αυτό θα μπορούσαμε να κάνουμε περαιτέρω χαρακτηρισμό των benchmarks όσον αφορά την "ευαισθησία" στη αλληλεπίδρασή τους με διαφορετικούς αλγόριθμους replacement policy.



Σχήμα 7.2: Συσχετισμός MPKI και CPI για όλα τα benchmark σε ίδιους άξονες

Μεγαλύτερη κλήση υποδηλώνει περισσότερη επίδραση στην επίδοση με την αυξομείωση των misses και μεγαλύτερο εύρος τιμών για MPKI υποδηλώνει περισσότερη αλλαγή στα misses με τις αλλαγές στο LLC replacement policy. Άλλες πληροφορίες που μπορούμε να αποκομίσουμε είναι το εύρος των τιμών των CPI και MPKI για να βγάλουμε παρόμοια συμπεράσματα όπως στο κεφάλαιο του χαρακτηρισμού των benchmarks. Για παράδειγμα εδώ φαίνεται ότι το lbm μπορεί να είναι scanning διότι έχει το μεγαλύτερο αριθμό από misses από όλα τα άλλα benchmarks αλλά ο αλγόριθμος replacement policy δεν καταφέρνει να υποβοηθήσει την επίδοση, διότι εδώ φαίνεται να έχει ευθεία κλίση.

7.3 Ακρίβεια

Είναι σημαντικό να μάθουμε κατά πόσο τα αποτελέσματα που πήραμε από τη πιο πάνω μελέτη έχουν σφάλμα τόσο μεγάλο που θα επηρεάσει αρνητικά τις επιλογές μας για το βέλτιστο replacement policy. Αυτό θα πρέπει να το καταφέρουμε μετρώντας τα περιθώρια σφάλματος για το κάθε benchmark.

Αρχικά θα δούμε γραφικά την διαφορά μεταξύ των αληθινών CPI τιμών (CPIreal) και των υπολογισμένων μέσω του Performance model (CPIpredicted). Παρατίθεται το γράφημα συσχετισμού των CPIreal και CPIpredicted για κάθε benchmark. Αν το Performance Model μας έχει μικρό σφάλμα, τότε το κάθε σημείο θα πρέπει να είναι όσο το δυνατό πιο κοντά στην ευθεία χ=ψ.

Ακολούθως θα υπολογίσουμε το λάθος και την κανονική απόκλιση σε ποσοστά από τα σημεία (CPIreal, CPIpredicted) για κάθε benchmark, χρησιμοποιώντας τις πιο κάτω φόρμουλες.

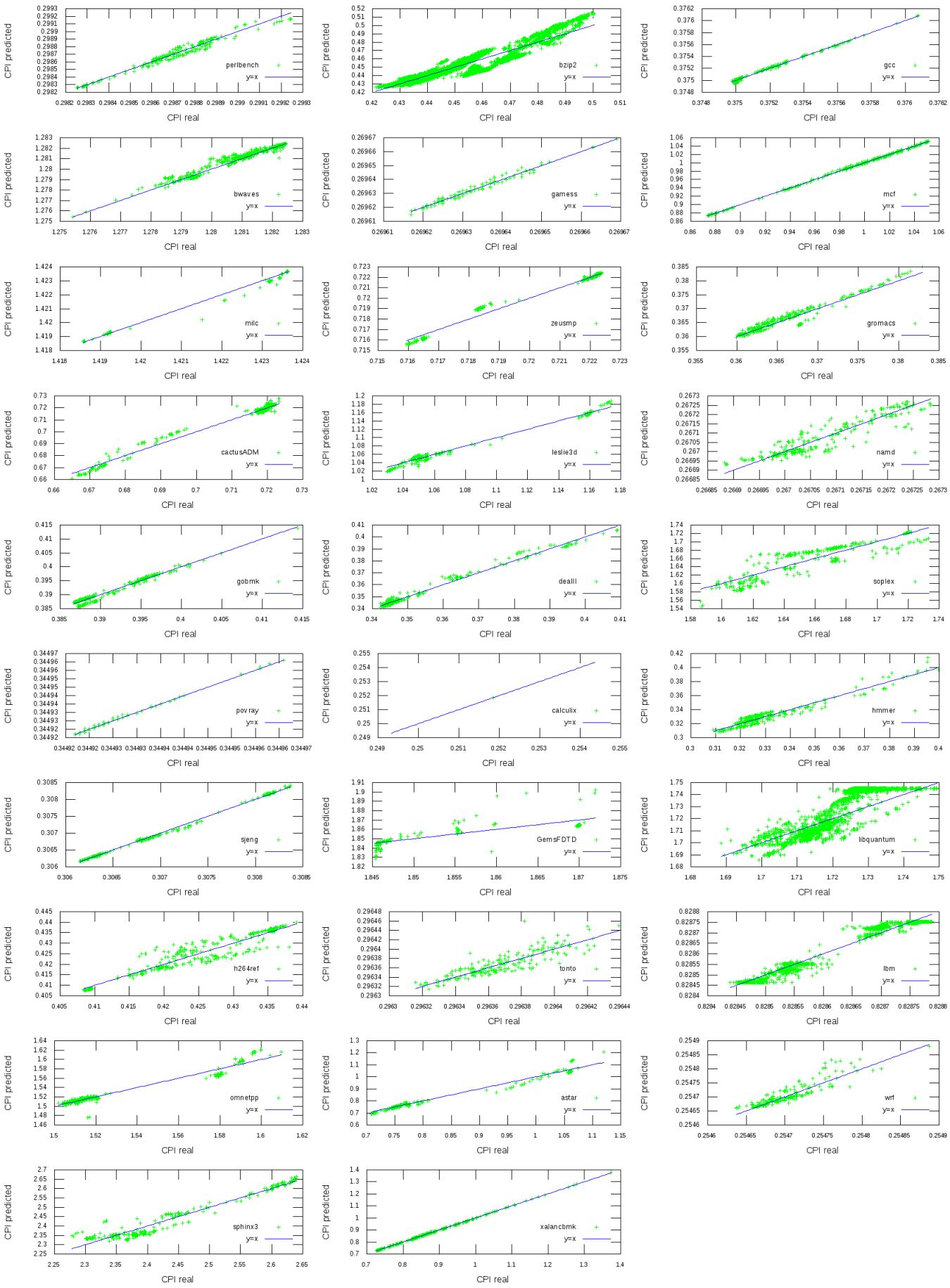
$$ErrorAvg = \left(\sum_{i=1}^{N} \left|\mathrm{CPI\_i}_{predicted}\right| - \frac{\mathrm{CPI\_i}_{real}}{C\mathrm{PI\_i}_{real}}\right) \cdot \frac{100}{N}\,\%$$

$$StdDev = \sqrt{\frac{\sum_{i=1}^{N} \left(\frac{\left|\mathrm{CPI\_i}_{predicted} - \mathrm{CPI\_i}_{real}\right|}{C\mathrm{PI\_i}_{real}} \cdot 100 - ErrorAvg\right)^2}{N-1}}$$

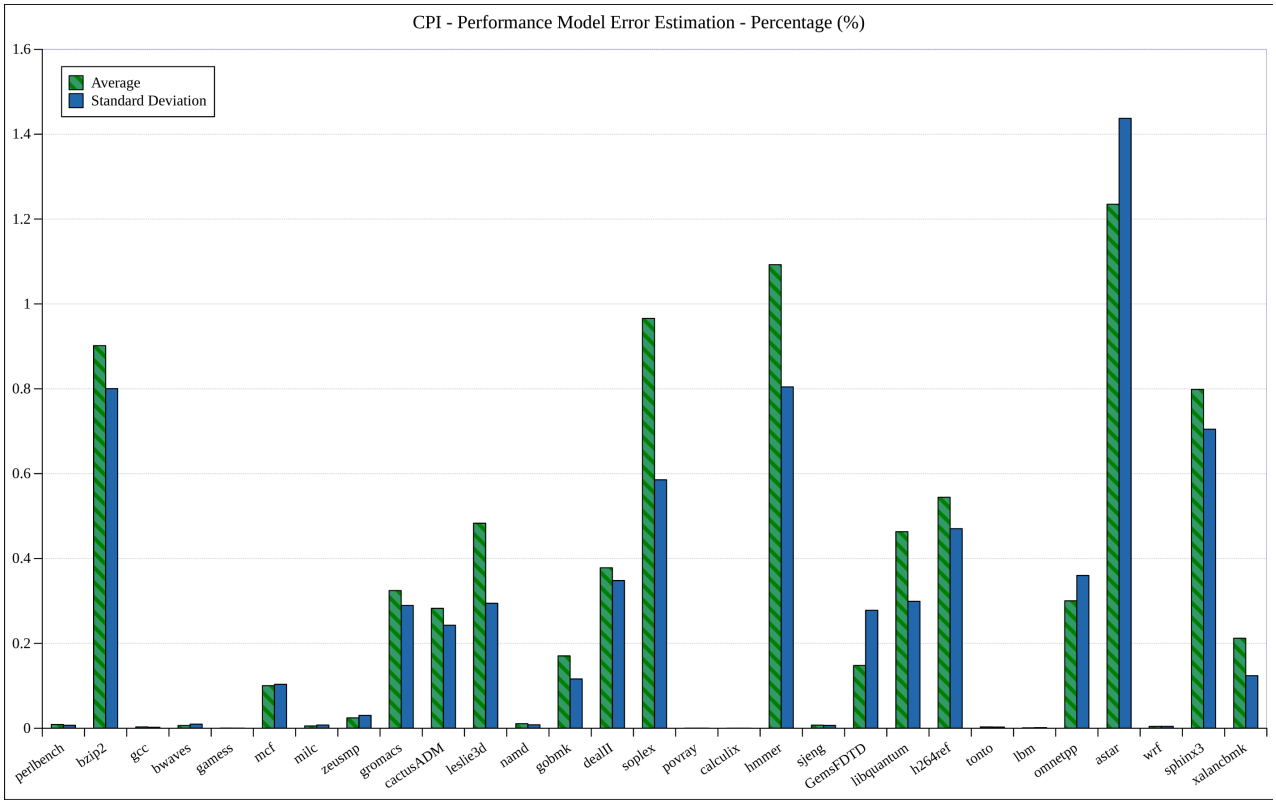
Έχοντας αυτές τις τιμές θα μπορούμε να ξέρουμε σε πιο βαθμό τα αποτελέσματα που παίρνουμε από τα Functional Simultations σε επίδοση χρησιμοποιώντας το Performance Model είναι αξιόπιστα. Επίσης με αυτά τα αποτελέσματα θα είμαστε σε θέση να αποφανθούμε κατά πόσο κάποια αποτελέσματα μπορεί να έχουν λάθη σε κάποιο σημείο της προσομοίωσης στην περίπτωση που θα λαμβάνουμε σφάλμα πολύ μεγαλύτερο από ότι είναι το μέσο και δεν είναι στο εύρος της κανονικής απόκλισης.

Παρατίθεται επίσης και το γράφημα για τα σφάλματα και κανονικές αποκλίσεις για κάθε benchmark. Σύμφωνα με αυτό, όπως βλέπουμε, τα πιο ανησυχητικά benchmarks

που θα μπορούν να διαστρεβλώσουν την εικόνα μας το περισσότερο για βελτίωση της επίδοσης από Functional Simulations είναι τα astar, bzip2, hmmer, sphinx3 και soplex. Όλων των benchmarks το σφάλμα είναι μικρότερο από 1.5% αλλά πρέπει να είμαστε προσεκτικοί διότι πολλά σφάλματα συνδυάζονται και 1.5% μέση βελτίωση από το DRRIP είναι σχετικά αξιόλογο νούμερο από μόνο του. Έτσι για κάθε τελική απόφαση στο σχεδιασμό θα τρέχουμε και Timing Simulations για επαλήθευση.



Σχήμα 7.3: Σύγκριση CPI real και CPI predicted για κάθε benchmark



Σχήμα 7.4: Ποσοστά σφάλματος και κανονική απόκλιση του Performance Model για όλα τα benchmarks

Κεφάλαιο 8

Εξερεύνηση παραλλαγών του DRRIP με 2 επιμέρους policies

8.1 Εισαγωγή

Έχοντας τώρα τα αποτελέσματα του Performance Model μπορούμε να εξάγουμε εκτιμήσεις για βελτίωση στην επίδοση (Speedup) με βάση το DRRIP χρησιμοποιώντας functional simulations. Γνωρίζουμε ότι υπάρχει κάποιο σφάλμα σε κάθε benchmark, αλλά είναι περίπου μέχρι 1% το κάθε ένα και πιθανοτικά μπορούμε να πάρουμε σχετικά καλή απόφαση για τη βέλτιστη παραλλαγή του DRRIP από τις 333600 διαθέσιμες.

Παρόλα αυτά ο στόχος της ανάλυσης αυτής δεν είναι να βρούμε απλώς μια παραλλαγή του DRRIP με δύο επιμέρους policies αλλά να πάρουμε πληροφορίες για σχεδιασμό ενός replacement policy με μεγαλύτερο βαθμό προσαρμοστικότητας στα διάφορα προγράμματα για να πετύχουμε μεγαλύτερη αύξηση στην επίδοση. Έτσι, δεν θα σταθούμε πολύ στα αποτελέσματα των 333600 παραλλαγών από μόνα τους αλλά θα τα αναλύσουμε για να βρούμε παράγοντες που επηρεάζουν την επίδοση σε διαφορετικά benchmarks.

Για αυτό το στάδιο θα πάρουμε προς το παρόν την καλύτερη παραλλαγή από τις 333600 και θα την τρέξουμε με Timing Simulation έτσι ώστε να μετρήσουμε το πραγματικό Speedup που προκύπτει. Αν θέλαμε να εστιαστούμε στην βελτίωση που μπορούμε να

πετύχουμε με 2 μόνο επιμέρους policies θα ήταν σωστό να αξιολογήσουμε περισσότερο από μία παραλλαγή (π.χ top 100) χρησιμοποιώντας timing simulations για κάθε benchmark.

Σχετικά με το τι πετυχαίνουμε με απλές αλλαγές στο DRRIP όπως θα δούμε πιο κάτω έχουμε μέση βελτίωση (speedup) 0.843% και αν αφαιρέσουμε τα 9 benchmarks της κατηγορίας "no need for LLC" (από το benchmark characterisation) προκύπτει 1.6% μέση βελτίωση.

8.2 Πειραματική αξιολόγηση με Functional Simulations

Οι 333600 παραλλαγές του DDRIP αξιολογήθηκαν χρησιμοποιώντας Functional Smulations και address traces 250 εκατομυρίων εντολών για Last-Level Cache 1 MB με 16-way associativity.

Κάθε μία εκτέλεση από τις 333600 (για όλα τα 29 benchmarks) παίρνει περίπου 1 λεπτό και έτρι προκύπτει ότι αν τα τρέχαμε σε ένα μονοπύρηνο επεξεργαστή θα χρειαζόμασταν 5560 core hours, δηλαδή 230 μέρες. Όμως έχουμε χρησιμοποιήσει το CyTera High Performance Computing facility του Cyorus Institute το οποίο διαθέτει περίπου 100 δωδεκαπύρηνα compute nodes και πετύχαμε την συλλογή όλων των αποτελεσμάτων σε περίπου 1.5 μέρες (λαμβάνοντας υπόψιν ότι υπήρχαν και άλλες εργασίες στην ουρά προτεραιότητας εκτέλεσης).

Το κύριο λογισμικό που χρησιμοποιήθηκε σε αυτό το βήμα είναι:

- "MODULE7_simulator2" – Η τελευταία έκδοση του functional simulator μου
- Slurm – Το προεγκατεστημένο σύστημα χρονοπρογραμματισμού HPC εργασιών
- pypy – Η εναλλακτική γρηγορότερη υλοποίηση της python, χρησιμοποιήθηκε για threadpools και ανάλυση αποτελεσμάτων.
- sqlite – Βιβλιοθήκη της python με την οποία αυτοματοποίησα την συλλογή αποτελεσμάτων σε μια βάση δεδομένων sqlite.

Βήματα για να τρέξουμε τα functional simulations:

- Μπαίνουμε στον υποκατάλογο “MODULE7_simulator2” τρέχοντας την εντολή: cd MODULE7_simulator2
- Παράγουμε το εκτελέσιμο τρέχοντας την εντολή: make
- Πηγαίνουμε ένα επίπεδο πιό πάνω και παράγουμε τους υποφακέλλους για τα αποτελέσματα τρέχοντας την εντολή: cd ../; mkdir results scripts scripts/batches traces
- Δημιουργούμε συντόμευση του φακέλλου των traces στον φάκελλο “MODULE7_simulator2” χρησιμοποιώντας την εντολή: ln -s traces MODULE7_simulator2/t
- Με αυτή την υποδομή
    - τοποθετούμε στον φάκελλο “scripts” τα αρχεία batch.py, threadedB.py και DB.py
    - τοποθετούμε στον φάκελλο “traces” ta address traces που έχουμε παράξει χρησιμοποιώντας την παραλλαγή του CRC kit για κάθε benchmark
- Τρέχουμε όλα τα simulations χρησιμοποιώντας την εντολή: cd scripts; pypy batches.py;
- Μόλις παρατηρήσουμε ότι έχει αδειάσει το squeue (χρησιμοποιώντας την εντολή squeue) από τις δικές μας διεργασίες, παράγουμε τη βάση δεδομένων από τα αρχεία εξόδου των simulations χρησιμοποιώντας την εντολή: pypy DB.py

Τώρα έχουμε στη διάθεσή μας το αρχείο policyDB.db το οποίο περιέχει πληροφορίες για τις τιμές των παραμέτρων των replacement policies, τις ταυτότητες των variations, τα MPKI, CPI, Speedup (predicted) για κάθε benchmark και μέσες τιμές.

Ακολούθως θα δούμε τα αποτελέσματα για το πρώτο καλύτερο variation (με top speedup predicted) και από timing simulation και θα κάνουμε την ανάλυση των σημαντικών παραγόντων που επηρεάζουν την επίδοση για όλα τα variations σε διαφορετικά benchmarks

8.3 Βέλτιστη Παραλλαγή από αυτή την μεθοδολογία

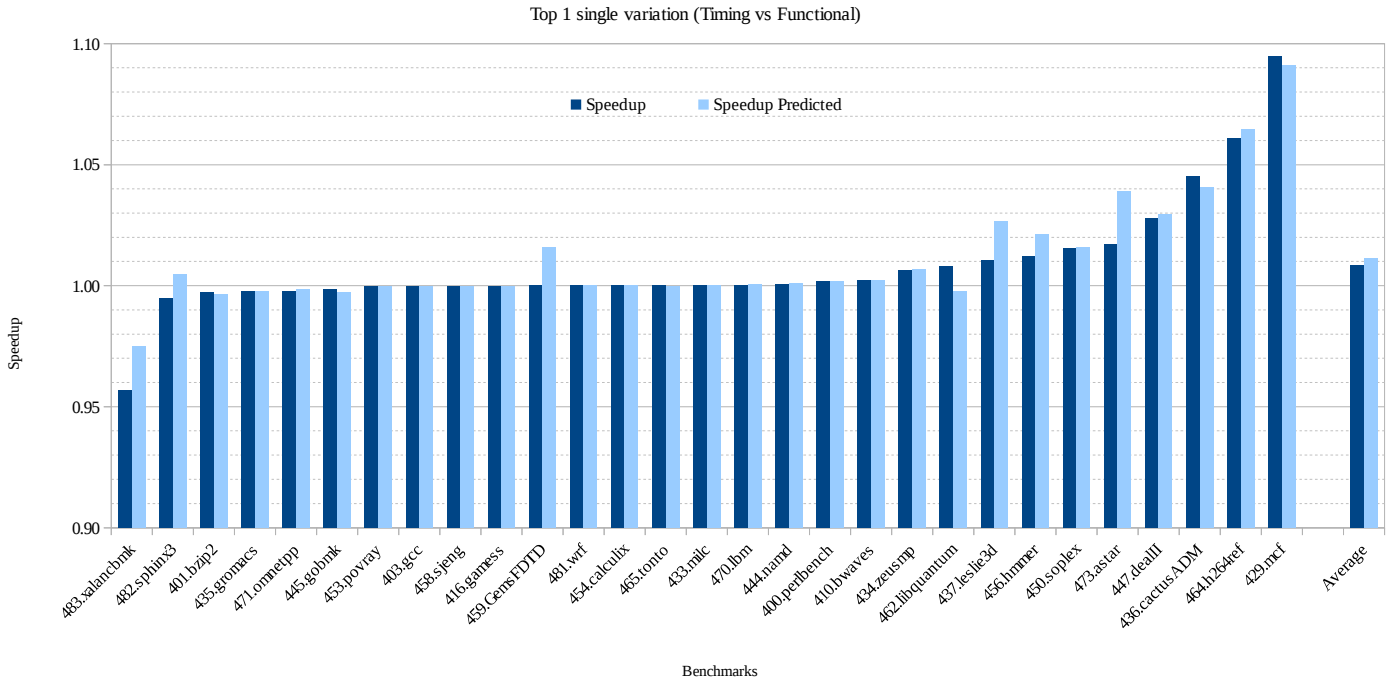
Η βέλτιστη παραλλαγή που προκύπτει από το συγκεκριμένο design space αποτελείται από 2 μόνο επιμέρους policies αφού μέχρι στιγμής δεν πειραματιστεί με SDM Models με περισσότερα από 2 επιμέρους policies.

Για να έχει νόημα ένα εύρημα θα πρέπει φυσικά να το επαληθεύσουμε με timing simulation για κάθε benchmark.

Ακολουθούν τα χαρακτηριστικά του DRRIP variation με το υψηλότερο (functional) Speedup:

- Functional Speedup Overall: 1.1139 %
- Timing Speedup Overall: 0.8426 %
- Τιμές παραμέτρων:
  - SDM Model: Different-set (2 policies)
  - PSEL Trigger: Hit
  - PSEL size: 10 bits
  - Initial PSEL value: middle
  - Promotion: Hit/Hit
  - Others demotion on insertion: Yes/No
  - Others demotion on hit on RRPV0: No/No
  - Bimodal Insertion: Yes/Yes
  - Insertion Position: 3/2
  - Bimodal Insertion Position: 0/0

Πιο κάτω βλέπουμε ένα γράφημα των αποτελεσμάτων για Speedup (timing) και Speedup (functional) για κάθε benchmark ταξινομημένα κατά αύξων Speedup (timing).



Timing και Functional Speedups του Top 1 single variation

8.4 Ανάλυση ευαισθησίας παραμέτρων για μεγαλύτερη προσαρμοστικότητα

Με την ανάλυση συνδυασμών θα προσπαθήσουμε να διακρίνουμε ποιες παραμέτρους είναι σημαντικό να είναι ελεύθερες, δηλαδή να αλλάζουν δυναμικά κατά την εκτέλεση αυξάνοντας την προσαρμοστικότητα του replacement policy ανάλογα με το workload.

Κάνοντας αυτή την ανάλυση θα πάρουμε πληροφορίες για πιθανές παραμέτρους στις οποίες αξίζει να προσπαθήσουμε να μεταβάλλουμε δυναμικά κατά την διάρκεια των εκτελέσεων. Δηλαδή θα μας δώσει κίνητρο για να κατευθυνθούμε σε συγκεκριμένο σχεδιασμό ενός πιο πολύπλοκου replacement policy το οποίο θα περιλαμβάνει επιπλέον adaptiveness με έμφαση στις παραμέτρους που θα υποδειχθούν σε αυτό το στάδιο.

Προς το παρόν η ιδέα είναι ότι διαφορετικά benchmarks προτιμούν διαφορετικά policies, γεγονός που θα διαφανεί και από την ίδια την ανάλυση αυτή. Θα μπορούσαμε φυσικά να επεκταθούμε και σε διαφορετικές φάσεις των ιδίων προγραμμάτων, ιδέα η οποία δεν μπορεί να διαφανεί με αυτή τη μέθοδο.

Διαχωρίζουμε τις παραμέτρους σε 8 ομάδες παραμέτρων έτσι όπως φαίνονται στον πίνακα 8.1.

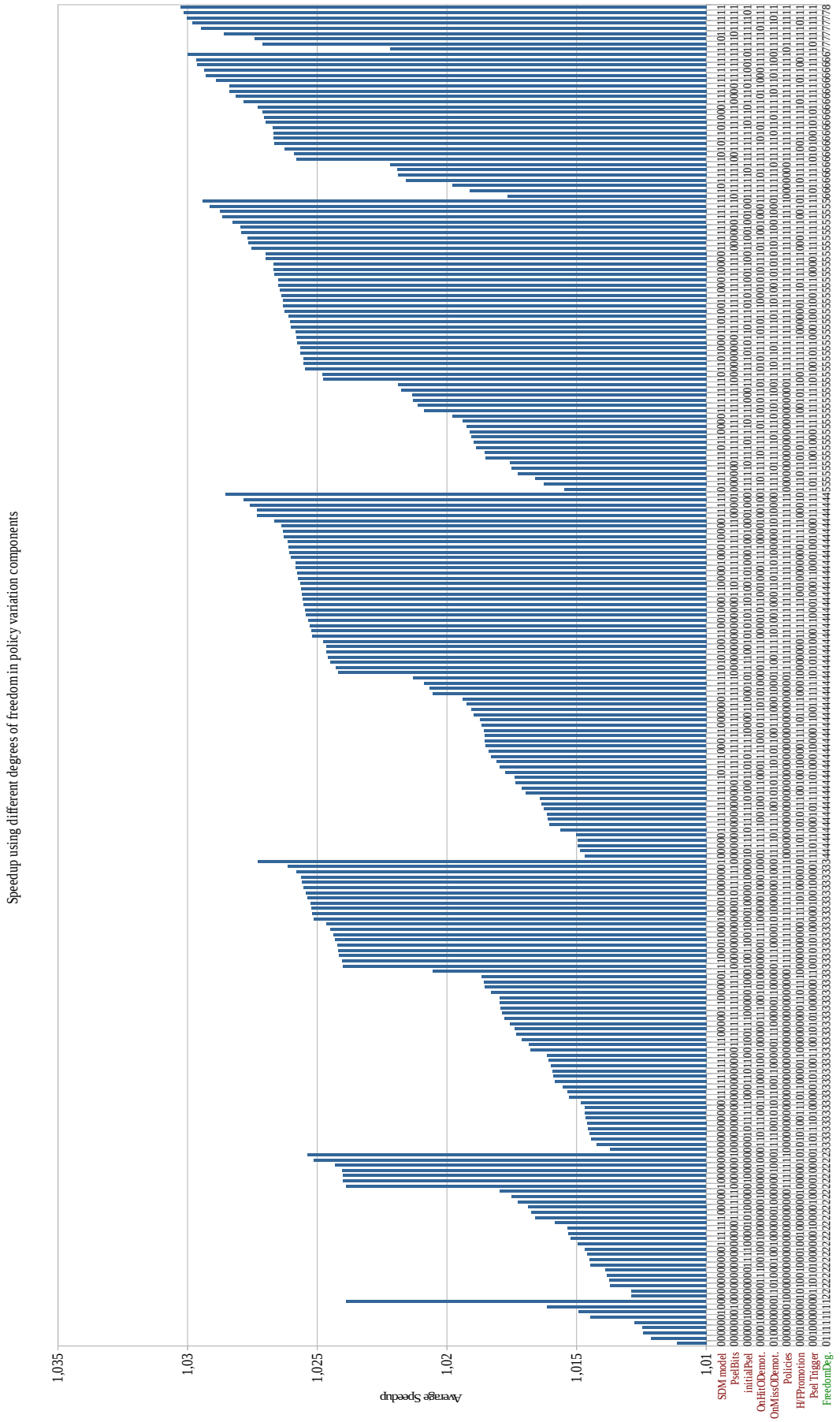
Ομάδα Παραμέτρων	Περιγραφή
SDM Model	Μια παράμετρος που περιλαμβάνει την SDM Model δηλαδή τον μηχανισμό με τον οποίο εναλλάσσονται τα policies (στην περίπτωσή μας μελετούμε 4 τιμές, μία για κάθε ένα από τα “same-set”, “different-set”, “per-set SDM global PSEL”, “per-set SDM per-set PSELs”)
PSEL bits	Μια παράμετρος που περιλαμβάνει το μέγεθος των PSEL counters [1-11]
initialPSEL	Μια παράμετρος που περιλαμβάνει την αρχική τιμή των PSEL counters {0, μέση}
Others demotion on insertion	Others demotion on insertion για το policy A Others demotion on insertion για το policy B
Others demotion on hit on RRPV0	Others demotion on hit on RRPV0 για το policy A Others demotion on hit on RRPV0 για το policy B
Policies	Insertion Position για το policy A Insertion Position για το policy B Bimodal Insertion για το policy A Bimodal Insertion για το policy B Bimodal Insertion Position για το policy A Bimodal Insertion Position για το policy B
Hit/Frequency Promotion	Promotion για το policy A Promotion για το policy B
PSEL Trigger	Μια παράμετρος που περιλαμβάνει το PSEL Trigger δηλαδή το γεγονός (hit ή miss) το οποίο θα αναβαθμίζει το PSEL counter όταν έχουμε different-set SDM Model.

Πίνακας 8.1: Ομάδες παραμέτρων για την ανάλυση ευαισθησίας παραμέτρων

Όταν μια ομάδα από αυτές παίρνει την τιμή 1 τότε θεωρούμε ότι οι συνδυασμοί από variations μπορούν να διαφέρουν μεταξύ τους σε αυτά τα χαρακτηριστικά. Αν η αντίστοιχη ομάδα παίρνει την τιμή 0 τότε οι συνδυασμοί από variations δεν μπορούν να διαφέρουν μεταξύ τους σε αυτά τα χαρακτηριστικά που ορίζει η ομάδα.

Όταν αναφερόμαστε σε συνδυασμούς εννοούμε ότι θα επιλέγουμε υποσύνολα από τις 333600 παραλλαγές του DRRIP των οποίων τα στοιχεία θα έχουν μεταξύ τους κοινές παραμέτρους όπως ορίζονται από τις τιμές που θα δίνουμε στις ομάδες παραμέτρων (0 για κοινή ομάδα και 1 για ελεύθερες μεταβλητές).

Από αυτούς τους συνδυασμούς από variations θα προβλέπουμε ένα speedup overall λαμβάνοντας υπόψιν μόνο το βέλτιστο διαθέσιμο για κάθε benchmark. Έτσι οι



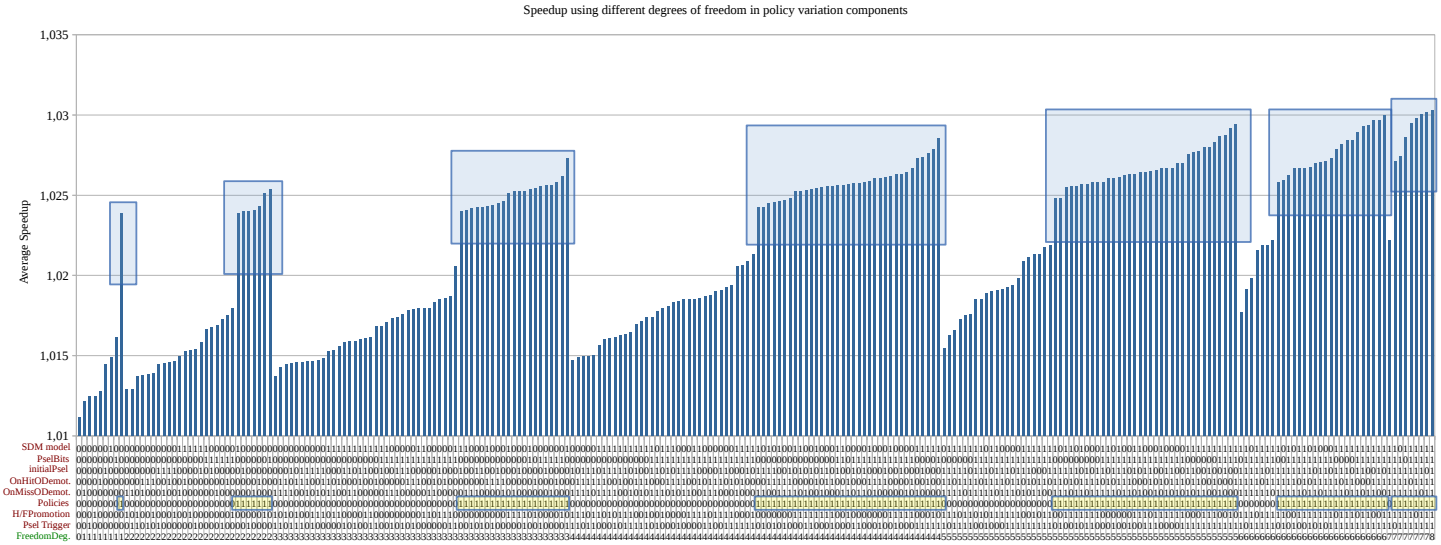
Σχήμα 8.1: Γράφημα εντοπισμού ευαισθησιών σε ελευθερίες ομάδων παραμέτρων

συνδυασμοί έχουν νόημα να λέμε ότι φτάνουν να έχουν μέχρι 29 variations, ένα για κάθε benchmark.

Στο σχήμα 8.1 παρατηρούμε τις υπολογιζόμενες βελτιώσεις επίδοσης του βέλτιστου συνδυασμού variations για κάθε συνδυασμό (μπάρα) από ελευθερίες σε ομάδες παραμέτρων. Η τελευταία γραμμή υποδηλώνει τον βαθμό ελευθερίας, δηλαδή τον αριθμό των ελεύθερων μεταβλητών, στων οποίων οι συνδυασμοί variations της αντίστοιχης μπάρας μπορούν να διαφέρουν.

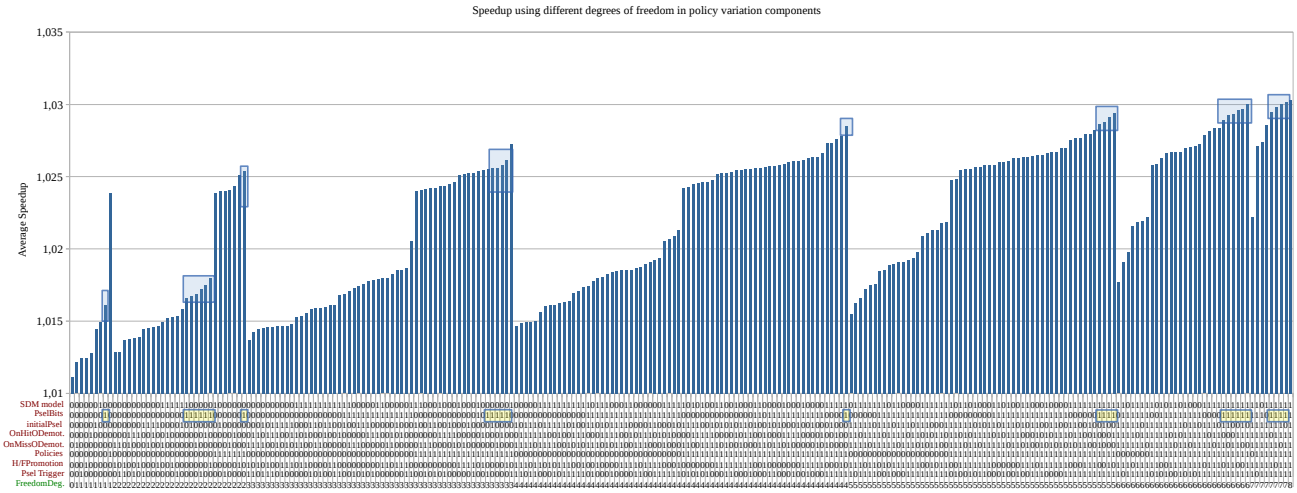
Παρατηρήσεις:

- Όταν δεν έχουμε βαθμό ελευθερίας προκύπτει το top variation από τα 336000
  - Δηλαδή με functional speedup ίσο με 1.11388966 %
- Όταν έχουμε μέγιστο βαθμό ελευθερίας σημαίνει ότι οι συνδυασμοί μπορούν να διαφέρουν σε όλες τις παραμέτρους. Επομένως το βέλτιστο speedup είναι η μέση τιμή του καλύτερο functional speedup που υπήρξε στα 336000 variations για κάθε benchmark.
  - Προκύπτει functional speedup (εκτιμώμενο) ίσο με 3.02885172 %
- Η σημαντικότερη ομάδα παραμέτρων που πρέπει να είναι ελεύθερη είναι το “policies” (Βλέπε το πιο κάτω γράφημα)
  - Αυτό διαφαίνεται από το γεγονός ότι υπάρχει ένα τεράστιο “σκαλί” στο γράφημα
  - Όπου εμφανίζεται το policies ελεύθερο αναμεταξύ των μπαρών τότε το μέγιστο speedup που προκύπτει ξεπερνά το 2.4 %
  - Αυτό ισχύει και όταν έχουμε βαθμό ελευθερίας 1 όπου όλοι οι συνδυασμοί ομάδων εκτός αυτού του policies είναι κάτω από 1.7 %
  - Επιπλέον είναι παρόν στα καλύτερους συνδυασμούς ομάδων παραμέτρων για κάθε διαφορετικό βαθμό ελευθερίας



Σχήμα 8.2: Σημειώσεις στο σχήμα 8.1 για υπόδειξη σημαντικότητας της ομάδας “policies”

- Η δεύτερη σημαντικότερη ομάδα παραμέτρων που πρέπει να είναι ελεύθερη είναι το “SDM Model” (Βλέπε το πιο κάτω γράφημα)
  - Όταν συνδυάζεται με άλλες ομάδες και συγκεκριμένα όταν ο βαθμός ελευθερίας είναι μεγαλύτερος από 2 τότε υπάρχει στους καλύτερους συνδυασμούς
  - Διαισθητικά φαίνεται να είναι δύσκολη η υλοποίηση μιας Last-Level Cache η οποία να μεταβάλλει το SDM Model της
- Η τρίτη σημαντικότερη ομάδα παραμέτρων που πρέπει να είναι ελεύθερη είναι το “PSEL bits”
  - Είναι η δεύτερη σημαντικότερη όταν ο βαθμός ελευθερίας είναι μικρότερος ή ίσος του 2



Σχήμα 8.3: Σημειώσεις στο σχήμα 8.1 για υπόδειξη σημαντικότητας της ομάδας “SDM Model”

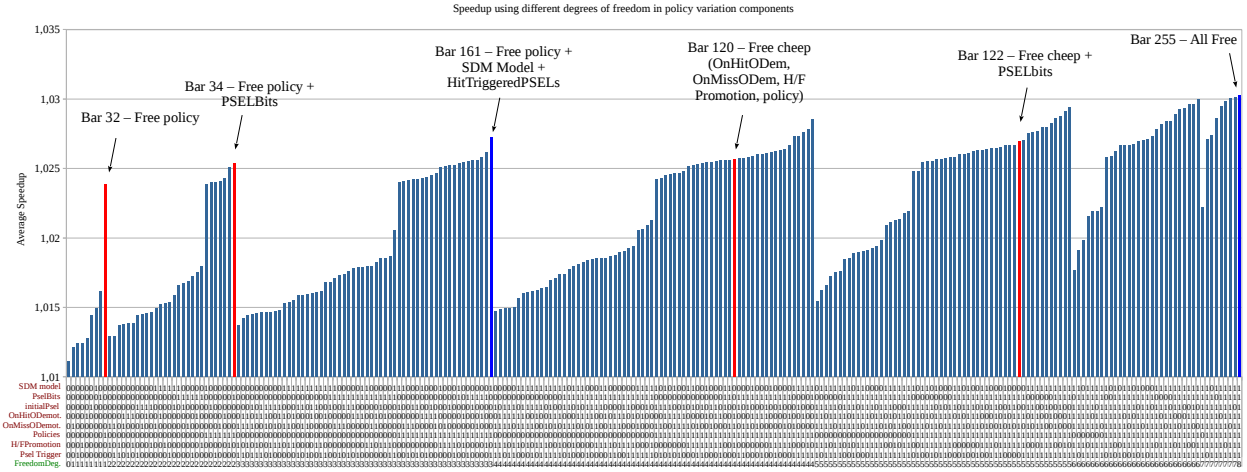
Είναι σημαντικό να σημειώσουμε ότι βαθμός ελευθερίας δεν είναι ταυτόσημος της ευκολίας υλοποίησης. Για παράδειγμα μπορεί να έχουμε ελεύθερη μεταβλητή μόνο την ομάδα “policies” αλλά να προκύψει βέλτιστος συνδυασμός που να αποτελείται από 29 διαφορετικά variations ενώ για κάποιο μεγαλύτερο βαθμό ελευθερίας να προκύπτουν λιγότερα.

8.5 Ανάλυση ευαισθησίας παραμέτρων με περιορισμό των συνδυασμών

Σαν πρώτο μέτρο μείωσης των συνδυασμών γενικά είναι να επιλέξουμε μόνο τις μπάρες που μας ενδιαφέρουν από το γράφημα ευαισθησίας παραμέτρων. Η επιλογή θα γίνει με βάση κάποια κριτήρια όπως την σημαντικότητα των ελεύθερων ομάδων από την πιο πάνω ανάλυση, την διαισθητική ευκολία υλοποίησης, το ενδιαφέρον για ανάλυση και η υπερκάλυψη συνδυασμών από μπάρες μεγαλύτερου βαθμού ελευθερίας.

Τα ονόματα των μπαρών με τα οποία θα αναφερόμαστε είναι οι δυαδικές αναπαραστάσεις των μέτρων ελευθερίας στο δεκαδικό σύστημα. Για παράδειγμα η μπάρα 1 είναι αυτή που έχει μόνο το “PSEL Trigger” ελεύθερη ομάδα παραμέτρων.

Έχουμε επιλέξει τις μπάρες 32, 34, 161, 120, 122 και 255 από τις 256 διαθέσιμες. Η επιλογή αυτή μπορεί να φανεί στο ακόλουθο γράφημα όπου υπάρχουν σημειώσεις στις μπάρες που μας ενδιαφέρουν.



Σχήμα 8.4:Επιλογή ενδιαφέροντων μπαρών στο γράφημα ευαισθησίας παραμέρτων

Επιπλέον παρατίθεται συνοπτική περιγραφή της κάθε μία από αυτές σε πίνακα που περιλαμβάνει τον λόγο επιλογής, τις ελεύθερες παραμέτρους, το βαθμό ελευθερίας, σχετική εκτιμώμενη δυσκολία υλοποίησης με βάση το 10, μέγιστο speedup και μέγιστο speedup με συνδιασμό μόνο 2 variations.

	Bar 32	Bar 34	Bar 120	Bar 122	Bar 161	Bar 255
Reason for selecting	Policy is the single most important parameter	Introduces feedom in PSEL bits in Bar 32	All theoretically cheap-to-implement parameters	Introduces feedom in PSEL bits in Bar 120	To explore freedom in SDMs – top bar with degree 3	All free
Free parameters	-policy	-policy -PSEL bits	-policy -OnHitODem -OnMissODem -H/F Promotion	-policy -OnHitODem -OnMissODem -H/F Promotion -PSEL bits	-policy -SDM Model -PSEL trigger	-policy -OnHitODem -OnMissODem -H/F Promotion -PSEL bits -SDM model -PSEL trigger -Initial PSEL
Freedom degree	1	2	4	5	3	8
Estimated Implementation difficulty	1	2	1.5	2.5	7	10
Maximum Speedup	1,0238708621	1,0253828966	1,025661931	1,0269944138	1,0272819655	1,0302885172
Exhaustive Top Pair Speedup	1,0194202759	1,020195069	1,0203503103	1,0207042069	1,0197481379	1.02105206897

Πίνακας 8.2: Χαρακτηριστικά για κάθε επιλεγμένο συνδυασμό ελεύθερων μεταβλητών

Σαν δεύτερο βήμα θα περιορίζουμε τον αριθμό των variations που θα μπορούν να υπάρχουν σε ένα συνδυασμό μέσα στις μπάρες. Ο λόγος που το κάνουμε αυτό είναι για

να φτάσουμε ένα βήμα πιο κοντά στην υλοποίηση αφού η προσέγγιση που θα χρησιμοποιήσουμε είναι προς το παρόν με εναλλαγές policies χρησιμοποιώντας SDM μοντέλα. Επιπλέον είναι πολύ καλύτερη ένδειξη της απλότητας υλοποίησης διότι ο βαθμός ελευθερίας δεν περιορίζει τον συνολικό αριθμό από variations για ένα συνδυασμό.

Μια παρατήρηση που θα μπορούσαμε να κάνουμε ακόμα από την προηγούμενη ανάλυση είναι ότι οι παράμετροι δεν είναι ανεξάρτητοι μεταξύ τους, δηλαδή δεν μπορούμε να εστιαστούμε στον καθένα ξεχωριστά και να βγάλουμε ένα καλό replacement policy μεταβάλλοντας μόνο μια μεταβλητή τη φορά. Είναι ο λόγος που τα Set-Dueling μοντέλα δεν θα προσπαθήσουμε να μεταβάλλουν ξεχωριστές παραμέτρους αλλά ολόκληρα τα επιμέρους policies.

Παρουσιάζονται τρεις διαφορετικές προσεγγίσεις για εύρεση συνδυασμών. Κάθε μία από αυτές έχει διαφορετική χρονική πολυπλοκότητα, η κάθε μία είναι υποσύνολο της επόμενης και τα αποτελέσματα της επόμενης μπορεί να είναι ίδια ή καλύτερα από την προηγούμενη. Η κάθε μια από αυτές τις μεθόδους θα γίνει για κάθε μπάρα ξεχωριστά φυσικά, αφού αναφερόμαστε σε συνδυασμούς με συγκεκριμένες ελεύθερες παραμέτρους.

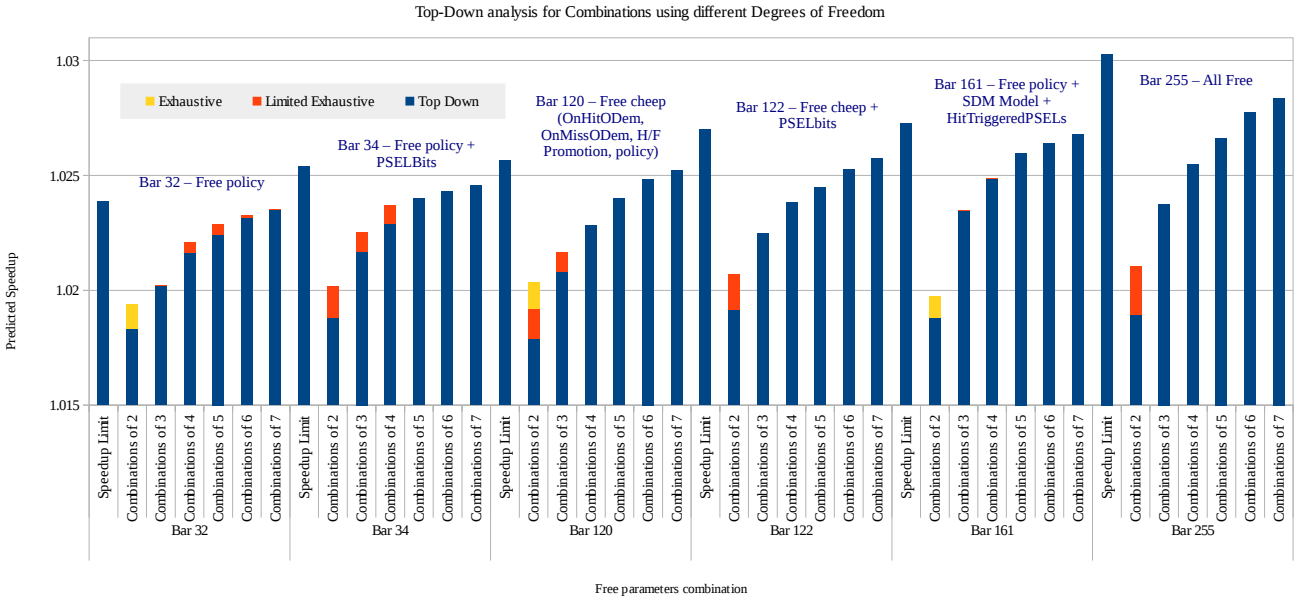
- Top Down Analysis
  - Εύρεση συνδυασμών από μέχρι k variations (συνδυασμούς των k) από μόνο τα variations που προκύπτουν από το βέλτιστο συνδυασμό της επιλεγμένης μπάρας
  - Συνολικά έχουμε το πολύ $\sum_{i=2}^{k}\binom{29}{i}$ συνδυασμούς, όπου 29 είναι ο αριθμός των benchmarks και k είναι ο μέγιστος αριθμός από variations ανά συνδυασμό
  - Χρονική πολυπλοκότητα $k \le 29 \Rightarrow \sum_{i=2}^{k}\binom{29}{i} \in O(1)$
- Limited Exhaustive

- Εύρεση συνδυασμών από μέχρι k variations (συνδυασμούς των k) από όλα τα variations που έχουν τις ίδιες μη-ελεύθερες ομάδες παραμέτρους με αυτές του βέλτιστου συνδυασμού της επιλεγμένης μπάρας
- Συνολικά έχουμε το πολύ $\sum_{i=2}^{k}\binom{n}{i}$ συνδυασμούς όπου n είναι ο αριθμός των πιθανών συνδυασμών τιμών των ελεύθερων παραμέτρων και k είναι ο μέγιστος αριθμός από variations ανά συνδυασμό
- Χρονική πολυπλοκότητα $\sum_{i=2}^{k}\binom{n}{i} \in O(2^n)$

- Exhaustive
  - Εύρεση συνδυασμών variations με κοινές τις μη-ελεύθερες παραμέτρους από μέχρι k variations (συνδυασμούς των k) από ολόκληρο το σύνολο των διαθέσιμων variations
  - Το πολύ $\sum_{j=1}^{g}(\sum_{i=2}^{k}\binom{m_j}{i})$ συνδυασμούς, όπου k είναι ο μέγιστος αριθμός από variations ανά συνδυασμό, g οι συνδυασμοί τιμών των μη-ελεύθερων παραμέτρων και m_j ο αριθμός των συνδυασμών των τιμών των ελεύθερων παραμέτρων για το συγκεκριμένο συνδυασμό μη-ελεύθερων παραμέτρων.
  - Χρονική πολυπλοκότητα $\in O(g * 2^{max(m)})$

Από τις πιο πάνω μεθόδους είναι καλό να γίνεται η επιλογή με βάση το πιο είναι εφικτό να γίνει σε λογικό χρόνο με προτίμηση στο πιο τελευταίο δηλαδή αυτό που μελετά τις περισσότερες περιπτώσεις.

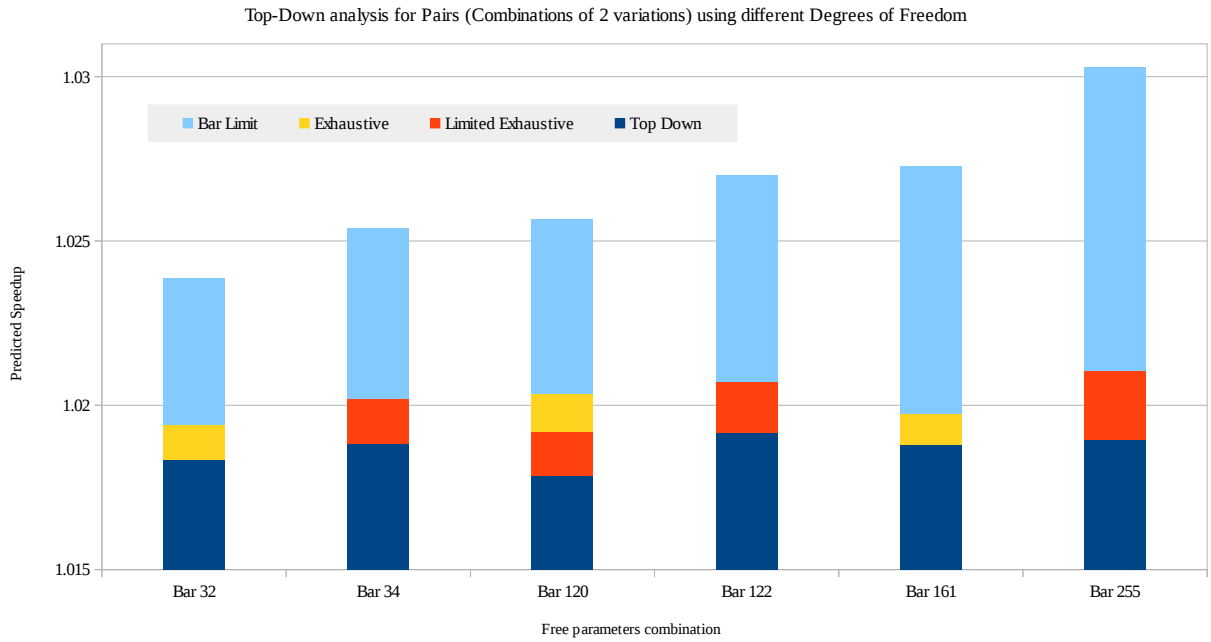
Στο πιο κάτω γράφημα παρουσιάζονται τα αποτελέσματα για τις επιλεγμένες μπάρες με τις πιο πάνω μεθόδους για περιορισμένους αριθμούς συνδυασμών. Δεν ήταν εφικτό να τρέξουν όλοι οι συνδυασμοί σε λογικό χρόνο διότι για κάποια προκύπτει τεράστιος αριθμός επιλογών. Μέσα από την ανάλυση αυτή μπορούμε να πάρουμε πληροφορίες όπως τι μπορεί να μας προσφέρει η κάθε ξεχωριστή μέθοδος επιλογής συνδυασμός όταν ήταν δυνατό να τρέξουν όλες και τι περιθώριο για βελτίωση έχουμε με περιορισμένο αριθμό από variations.



Σχήμα 8.5: Αποτελέσματα περιορισμού συνδυασμών σε 6 μπάρες και μικρό αριθμό από variations

Συμβατικά, για λόγους υλοποίησης, ύπαρξης αποτελεσμάτων με την Exhaustive μέθοδο και φαινομενικά σχετικά ικανοποιητικών αποτελεσμάτων επιλέγουμε μόνο τους συνδυασμούς από 2 variations (δηλαδή συνολικά 4 επιμέρους policies).

Παρατίθενται από τα πιο πάνω αποτελέσματα μόνο αυτά για συνδυασμούς από 2 δηλαδή ζεύγη από DRRIP variations.



Σχήμα 8.6: Μείωση σε συνδυασμούς από 2 DRRIP variations

Παρατηρήσεις:

- Σε μερικές περιπτώσεις οι απλουστευμένες μέθοδοι όπως το Top Down δίνει καλή προσέγγιση για την μέθοδο Limited Exhaustive.
  - Παράδειγμα οι συνδυασμοί των 4, 5 και 6 variations για την μπάρα 32.
- Όπου δεν έχει ανοιχτά χρώματα είτε δεν έχουμε δεδομένα λόγω των πολλών συνδυασμών που παράγονται, είτε προσεγγίζονται πολύ καλά από την απλούστερη μέθοδο.
- Μπορούμε να πούμε ότι οι καλύτεροι συνδυασμοί των 2 variations σε κάθε μπάρα είναι σχετικά παρόμοιας προβλεπόμενης επίδοσης. Έτσι, μπορούμε να αγνοήσουμε προς το παρών τις μπάρες που θεωρούμε ότι απαιτούν σημαντικό hardware overhead σε πιθανή υλοποίηση.
- Το κάθε variarion έχει 2 επιμέρους policies. Επομένως εδώ για συνδυασμό k variations πρέπει να δημιουργηθεί Set-Dueling μοντέλο που να εναλλάσσει 2*k policies.
- Το Exhaustive δίνει εκ φύσεως τα καλύτερα αποτελέσματα αλλά μπορούμε να τροποποιήσουμε την μέθοδο να μας δίνει συνδυασμούς με ορισμένες μη-ελεύθερες παραμέτρους που μας συμφέρουν στην υλοποίηση
  - Για παράδειγμα το Limited Exhaustive μπορεί να μας δώσει top συνδυασμό ο οποίος να είναι “different-set SDMs” ενώ ο Exhaustive μπορεί να επιστρέψει συνδυασμό ο οποίος να είναι per-set “SDMs per-set PSELs”
- Μπορούμε να τροποποιήσουμε περαιτέρω τον Exhaustive έτσι ώστε να μας επιστρέφει top 5 συνδιασμούς με τους συγκεκριμένους περιορισμούς

Στον ακόλουθο πίνακα βλέπουμε τα χαρακτηριστικά των καλύτερων ζευγών από variations όπως προκύπτουν από την Exhaustive μέθοδο όπως ορίσαμε πιο πριν. Με κίτρινο χρώμα είναι οι μη-ελεύθερες μεταβλητές, δηλαδή αυτές που πρέπει να είναι όπως οτιδήποτε κοινές μεταξύ των δύο variations. Με γκρίζο χρώμα είναι οι ελεύθερες παράμετροι.

				(Dueling Policy Parameters)													Speedup	Bar
Bar	PSEL Model	PSELbits	Initial PSEL	FP1	FP2	Hdem1	HDem2	ODem1	ODem2	BR1	BR2	Ins. Pos. 1	Ins. Pos. 2	B. Ins P. 1	B. Ins P. 2	PSEL Trig.	Predicted	Limit
32	Diff-set	5	Zero	No	Yes	No	No	No	No	No	Yes	3	3	0	2	Hit	1.01942028	1.0238709
	Diff-set	5	Zero	No	Yes	No	No	No	No	Yes	Yes	2	3	1	1	Hit		
34	Diff-set	2	Zero	No	Yes	No	No	No	Yes	Yes	Yes	2	3	0	0	Hit	1.02019507	1.0253829
	Diff-set	6	Zero	No	Yes	No	No	No	Yes	No	Yes	3	3	0	2	Hit		
120	Diff-set	6	Zero	No	No	No	Yes	No	No	Yes	Yes	2	3	1	1	Hit	1.02035031	1.0256619
	Diff-set	6	Zero	No	Yes	No	No	No	No	No	Yes	3	3	0	2	Hit		
122	Diff-set	2	Zero	No	No	No	Yes	No	Yes	Yes	Yes	2	3	0	0	Hit	1.02070421	1.0269944
	Diff-set	6	Zero	No	Yes	No	No	No	No	No	Yes	3	3	0	2	Hit		
161	Same-set	5	Zero	No	Yes	No	No	No	No	No	Yes	3	3	0	2	Miss	1.01974814	1.027282
	Diff-set	5	Zero	No	Yes	No	No	No	No	Yes	Yes	2	3	1	1	Hit		
255	Diff-set	6	Zero	No	Yes	No	No	No	No	No	Yes	3	3	0	2	Hit	1.02105207	1.0302885
	Per-set S/P	1	Middle	No	No	No	Yes	No	No	Yes	Yes	2	3	0	0	Miss		

Πίνακας 8.2: Χαρακτηριστικά καλύτερων ζευγών για τις επιλεγμένες μπάρες ελευθεριών

Παρατηρήσεις:

- Όπου το επιμέρους policy είναι “different-set SDMs” φαίνεται ότι προτιμούνται Hit Triggered PSELs
  - Στις άλλες περιπτώσεις που είναι Miss Triggered PSELs όποως προαναφέραμε δεν έχει νόημα να ήταν διαφορετικό από το Hit Trigger.
- Προς το παρόν είναι θετική ένδειξη που προτιμάται το different-set όταν το SDM Model δεν είναι ελεύθερο διότι απαιτεί το λιγότερο hardware για υλοποίηση
  - Όμως δεν αξιολογούμε ακόμα τι γίνεται όταν υπάρχουν περισσότερες από 2 ομάδες SDMs (περισσότερα sets με προκαθορισμένα policies στην LLC ενδεχομένως να επηρεάσουν την επίδοση λόγω περισσότερων μη βέλτιστων επιλογών)
- Φαίνεται ότι υπάρχει ανάγκη να υπάρχουν και τα δύο είδη promotions (δηλαδή Hit Promotion και Frequency Promotion (για FP No και Yes αντίστοιχα)) στα policies

Κεφάλαιο 9

Εξερεύνηση παραλλαγών του DRRIP με 4 επιμέρους policies

9.1 Εισαγωγή

Όπως φάνηκε από την ανάλυση του κεφαλαίου 8 τα benchmarks ως σύνολο χρειάζονται περισσότερο από 2 replacement policies για να βελτιωθεί περισσότερο η μέση επίδοση.

Τα αποτελέσματα που προκύπτουν από την Ανάλυση ευαισθησίας παραμέτρων με περιορισμό των συνδυασμών (κεφάλαιο 8.5) μπορούμε να αποκομίσουμε ζεύγη (ή και συνδυασμούς με περισσότερα variations) τα οποία υποδηλώνουν πιθανή αύξηση της επίδοσης μέχρι και 2 περίπου τοις εκατό όταν συνδυαστούν σε ένα νέο μηχανισμό.

Έτσι χρειαζόμαστε ένα μηχανισμό με τον οποίο να υλοποιούμε περισσότερα από δύο επιμέρους policies και να μπορεί να επιλέγει μεταξύ αυτών την ώρα της εκτέλεσης δυναμικά.

9.2 Προτεινόμενοι μηχανισμοί εναλλαγής πολλαπλών policies

Ο κώδικάς μου υλοποιεί και τις εκδοχές για 4 policies των μηχανισμών "same-set-SDMs" και "different-set-SDMs". Μας ενδιαφέρει διότι θέλουμε να μελετήσουμε αν μπορούμε να αυξήσουμε την επίδοση με την αύξηση των policies που εναλλάσσονται.

Ήδη υπάρχουν αρκετές δουλειές που ασχολούνται με μηχανισμούς εναλλαγής πολλαπλών policies [18].

Οι δύο μηχανισμοί που παρατίθενται εδώ σχεδιάστηκαν έτσι ώστε να είναι όσο πιο κοντά στη νοοτροπία των αντίστοιχων για δύο policies.

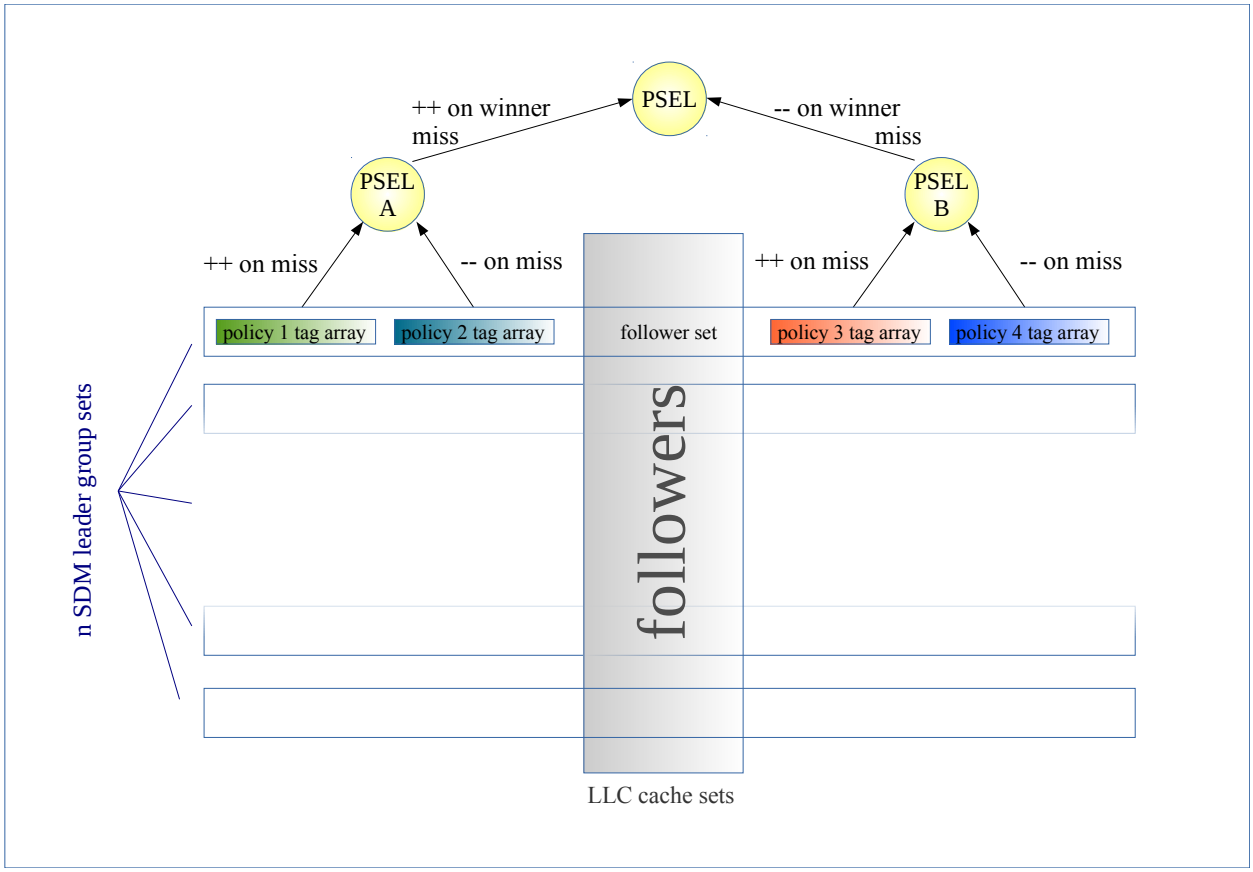
Περιγραφή των "multiple policy same-set-SDMs" και "multiple policy different-set-SDMs":

- Ο αριθμός των policies n είναι δύναμη του 2
- Παράγουμε ένα γράφο από n-1 PSEL counters στη μορφή πλήρους δυαδικού δέντρου
- Στο τελευταίο επίπεδο είναι τα PSEL counters τα οποία ενημερώνουν ζευγάρια των n policies. Δηλαδή σε κάθε φύλλο του δέντρου miss στο policy Left προκαλεί αύξηση κατά 1, ενώ miss στο policy Right προκαλεί μείωση κατά 1.
- Ένα miss διαδίδεται στο πιο πάνω επίπεδο αν και μόνο αν αυτός που το προκάλεσε είναι τοπικός νικητής. Δηλαδή αν σε ένα PSEL counter έχουμε leftmost bit ίσο με 1 τότε misses του policy Left (ή PSEL left) δεν διαδίδονται στο πιο πάνω επίπεδο.
- Η απόφαση μπορεί να βρεθεί σε log2(n) βήματα ξεκινώντας από τη ρίζα του δέντρου και ακολουθώντας τα leftmost bit των PSELs για κατεύθυνση.

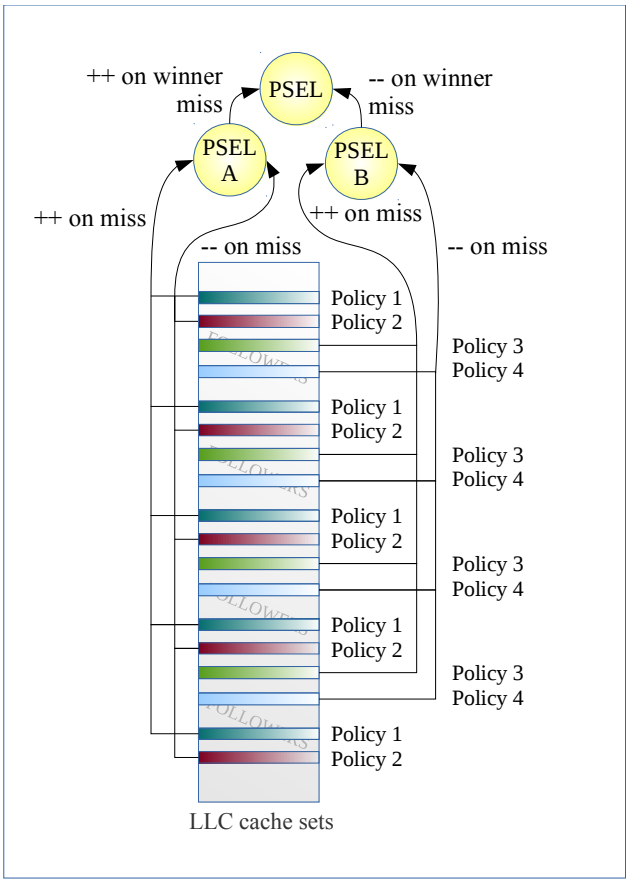
Παρατίθενται τα σχετικές απεικονίσεις (Σχήμα 9.1 και 9.2) των "multiple policy same-set-SDMs" και "multiple policy different-set-SDMs".

Ένα πλεονέκτημα του "multiple policy different-set-SDMs" σε σχέση με το "multiple policy same-set-SDMs" είναι ότι δεν χρειάζεται επιπρόσθετο hardware για τα SDMs διότι βρίσκονται εντός της Last-Level Cache.

Ένα πλεονέκτημα του "multiple policy same-set-SDMs" σε σχέση με το "multiple policy different-set-SDMs" είναι ότι τα SDMs δεν κάνουν "μολύνουν" την cache με μεγάλο αριθμό από policies που δεν είναι βέλτιστα για τα set. Επιπλέον, σε περιπτώσεις ετερογένειας των accesses φαίνεται ότι υπερτερούν παίρνοντας πιο δίκαιες αποφάσεις.



Σχήμα 9.1: Σχηματική Αναπαράσταση του “multiple policy same-set-SDMs” για υποθετική LLC



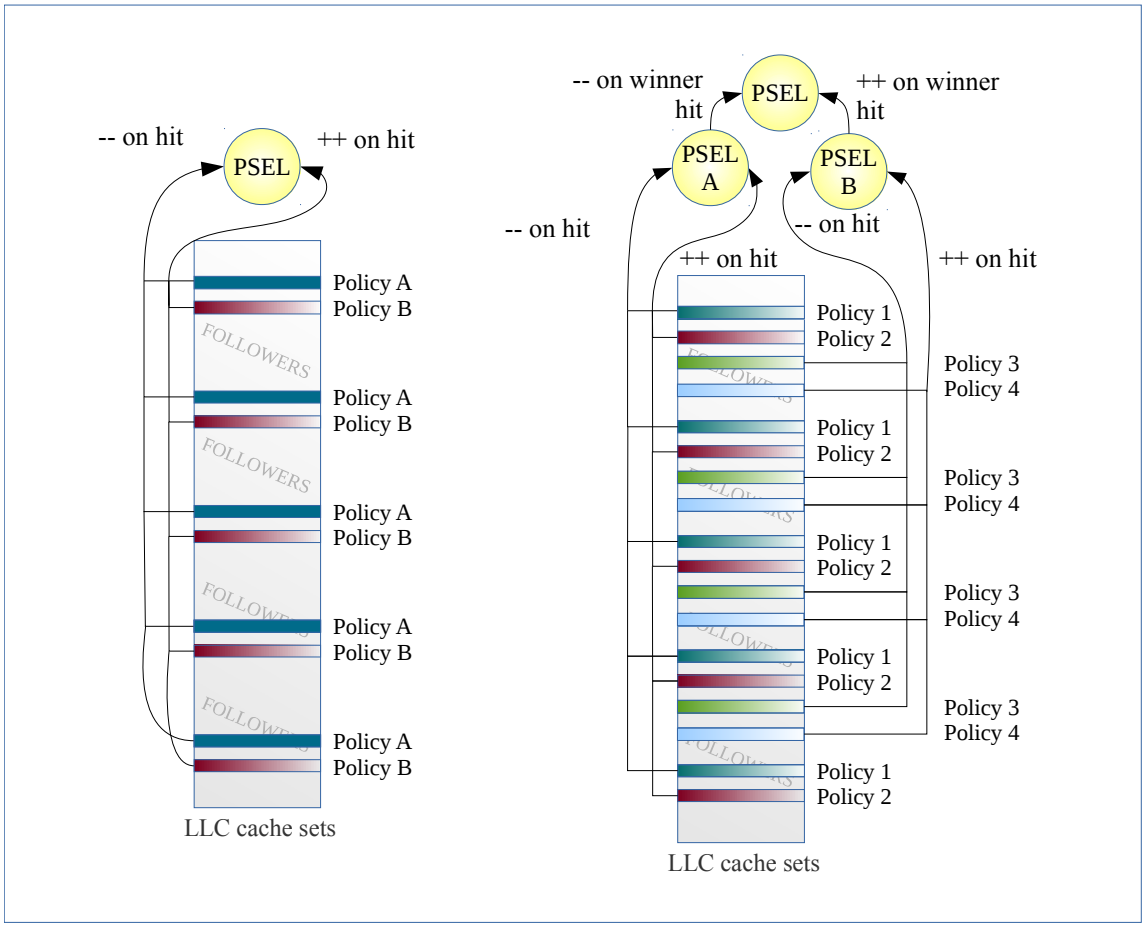
Σχήμα 9.2: Σχηματική Αναπαράσταση του “multiple policy different-set-SDMs” για υποθετική LLC

9.2.1 PSEL trigger

Τέλος μελετούμε και εδώ την παραλλαγή των πιο πάνω στην περίπτωση όπου οι αναβαθμίσεις των PSELs γίνονται με βάση τα hits αντί τα misses. Στην περίπτωση των "Hit Triggered PSELs" όταν έχουμε hit, τα PSEL αναβαθμίζονται από την αντίθετη κατεύθυνση καθώς το hit προμηνύει προτίμηση από την πλευρά από όπου έχει συμβεί.

Η επιλογή "Hit Triggered PSELs" βρίσκει εφαρμογή στο "multiple policy different-set-SDMs". Το σκεπτικό είναι το ίδιο με του "different-set-SDMs" με 2 policies και πάλι δεν βρίσκει εφαρμογή στην same-set παραλλαγή διότι συμπεριφέρεται λογικά το ίδιο με το "Miss Triggered PSELs".

Αυτό αληθεύει διότι στο same-set έχουμε ταυτόχρονη πρόσβαση στα PSELs φύλλα από 2 SDM sets τη φορά και η μόνη περίπτωση που αναβαθμίζονται είναι όταν έχουμε hit/miss, miss/hit. Η άλλη περίπτωση είναι όταν έχουμε hit/hit ή miss/miss στην οποία και όταν είχαμε "Miss Triggered PSELs" η τιμή του PSEL παραμένει η ίδια. Για το Global PSEL (ή όλοι οι εσωτερικοί κόμβοι του δέντρου, όταν έχουμε περισσότερα από 4 policies) προκύπτουν οι ίδιες περιπτώσεις ανάλογα με το ποια SDM groups θεωρούνται οι winners τη χρονική στιγμή της ενημέρωσης.



Σύγκριση των “different-set-SDMs” και “multiple policy different-set-SDMs” με Hit Triggered PSELs για υποθετική LLC

9.3 Παράμετροι σχετικά με τα Set Dueling Monitors

Ο Functional Simulator μου έχει επεκταθεί έτσι ώστε να υλοποιεί τα “multiple policy different-set-SDMs” και “multiple policy same-set-SDMs”. Η αλλαγή αυτή έχει επεκτίνει τις παραμέτρους σχετικά με την υλοποίηση των Set Dueling Monitors και είναι οι ακόλουθες:

- numberOfPSELs: Η τιμή της μεταβλητής αυτής καθορίζει ποιο από τα πιο πάνω μοντέλα θα υλοποιηθεί και από πόσα policies θα αποτελείται
  - Πεδίο τιμών: $[-3, n] \cap \mathbb{Z}$ για n Last-Level Cache sets
    - -3: multiple policy different-set-SDMs (4 policies)
    - -2: multiple policy same-set-SDMs (4 policies)
    - -1: same-set-SDMs (2 policies)

    - 0: different-set-SDMs (2 policies)
    - 1: Per-set-SDMs Global-PSEL (2 policies)
    - [2, n-1]: Per-set-SDMs Per-Region-PSELs (2 policies)
    - n: Per-set-SDMs Per-set-PSELs (2 policies)
- PSELbits: [1-30] Μέγεθος των PSEL σε bits για τα μοντέλα
    - same-set-SDMs
    - different-set-SDMs
    - Per-set-SDMs Global-PSEL
    - Per-set-SDMs Per-Region-PSELs
- PSELAbits [1-30]: Μέγεθος του PSELA σε bits για τα μοντέλα
    - multiple policy different-set-SDMs
    - multiple policy same-set-SDMs
- PSELBbits [1-30]: Μέγεθος του PSELB σε bits για τα μοντέλα
    - multiple policy different-set-SDMs
    - multiple policy same-set-SDMs
- PSELTrigger: Παράμετρος που καθορίζει κατά πόσο οι αναβαθμίσεις των PSEL γίνονται με τα misses ή με τα hits
    - Πεδίο τιμών: “Miss”, “Hit”
- Initial PSEL value: Παράμετρος που καθορίζει την αρχική τιμή των μετρητών PSEL
    - Πεδίο τιμών: “zero”(0), “middle” ($2^{(\text{PSEL bits}) - 1}$)

9.4 Πειραματική Αξιολόγηση

Από την Ανάλυση ευαισθησίας παραμέτρων με περιορισμό των συνδυασμών (κεφάλαιο 8.5) μπορούμε να συλλέξουμε ζεύγη από variations τα οποία υπολογίζεται ότι ο συνδυασμός τους θα επέφερε αύξηση στην επίδοση μέχρι 2 περίπου τοις εκατό.

Η προτεινόμενη μέθοδος είναι να τρέξουμε Timing Simulations για ένα συγκεκριμένο μικρό Design Space το οποίο θα απορρέει από την επιλογή των ζευγών στην ανάλυση.

Έχοντας επιλέξει τις μπάρες 32, 120 και 122 μπορούμε να υλοποιήσουμε ζεύγη αυτών όταν αυτά είναι είτε “Same-set SDMs” είτε “Different-set SDMs”. Το νέο ζήτημα είναι πόσο να είναι το μέγεθος του PSEL counter που θα επιλέγει μεταξύ των 2 variations, δηλαδή τα bits της ρίζας του PSEL δέντρου. Το PSELA και PSELB θα είναι αντίστοιχου μεγέθους με αυτών που υπάρχουν σαν μοναδικά PSELs στο κάθε ένα από τα 2 variations για να γίνεται η επιλογή μεταξύ των 2 επιμέρους policies στο κάθε variation.

Ένα άλλο ζήτημα είναι ότι όπως έχουμε δείξει στο Performance Modeling υπάρχει κάποιο σφάλμα και θα ήταν βοηθητικό να μην αξιολογήσουμε με Timing Simulations απλώς 1 top ζεύγος variations αλλά ένα μεγαλύτερο φάσμα, λογικού βέβαια συνολικού χρόνου εκτέλεσης.

Έτσι το νέο Design Space για Performance (Timing) αξιολόγηση θα μπορούσε να αποτελείται από τα top 10 “Same-set SDMs” και top 10 “Different-set SDMs” για την κάθε μία από τις μπάρες 32, 120 και 122. Στις μπάρες αυτές το PSEL Trigger είναι μια μη-μεταβλητή παράμετρος και επομένως εδώ θα επιλέγουμε το αντίστοιχο στα “Different-set SDMs” ζεύγη.

Λόγω χρόνου δεν προλαβαίνουμε να απασφαλματώσουμε την υλοποίηση των μηχανισμών αυτών και τα αποτελέσματά τους. Έτσι παρατίθεται ως μελλοντικό έργο η ολοκλήρωση του σημείου αυτού και μπορούμε να πούμε ότι βάσει της ανάλυσης ευαισθησίας οι μηχανισμοί αυτοί είναι ίσως αρκετά υποσχόμενοι.

Κεφάλαιο 10

Συμπεράσματα και Μελλοντικό Έργο

10.1 Συμπεράσματα

Το γενικότερο συμπέρασμα είναι ότι με πολύ μικρές αλλαγές μπορούμε να βελτιώσουμε την επίδοση του DRRIP κατά περίπου 0.85% ή 1.6% αν αφαιρέσουμε τα "no-need-for-LLC" Benchmarks που εντοπίσαμε στον χαρακτηρισμό των SpecCPU 2006 (κεφάλαιο 5).

Επιπλέον από την ανάλυση των αποτελεσμάτων στο κεφάλαιο 8 φαίνεται ότι ο συνδιασμός περισσότερων από 2 policies όπως με ένα μηχανισμό σαν αυτό που εισάγουμε στο 9 μπορεί να φέρει σημαντική βελτίωση στην επίδοση. Σύμφωνα με την ανάλυση ευαισθησίας παραμέτρων με περιορισμό των συνδυασμών (υποκεφάλαιο 8.5) εναλλάσσοντας 2 RRIP variations, δηλαδή 4 policies στο σύνολο μπορούμε να φτάσουμε βελτίωση στην επίδοση τάξεως 2%.

Επίσης αν είχαμε κάποιο τρόπο να αλλάζουν όλες οι διαθέσιμες παράμετροι δυναμικά με βέλτιστες επιλογές για κάθε benchmark μπορούμε να φτάσουμε βελτίωση στην επίδοση περίπου 3%.

Άλλα συμπεράσματα από ολόκληρη την μεθοδολογία:

- Μπορούμε εύκολα να εντοπίσουμε το είδος κάθε Miss χρησιμοποιώντας την μεθοδολογία που περιγράφεται στο κεφάλαιο 5

- Μπορούμε εύκολα να εντοπίσουμε κατηγοριοποιήσουμε τα Benchmarks σε σχέση με την συμπεριφορά τους στην Last-Level Cache.
- Μπορούμε εύκολα να κατασκευάσουμε Performance Model για την εκτίμηση της βελτίωσης στην επίδοση με Functional Simulations. Όμως καθώς η σχέση MPKI και CPI δεν είναι πάντα γραμμική και υπάρχουν και άλλα επίπεδα ιεραρχίας μνήμης, προκύπτει σφάλμα περίπου μέχρι 1% ανά benchmark.
- Αξιολογώντας το ευρύ μας Design Space φάνηκε ότι υπάρχουν και άλλες παράμετροι που θα μπορούσαν να χρησιμοποιηθούν από τα replacement policies για αύξηση της προσαρμοστικότητάς τους και της επίδοσής τους.
- Γενικά υπάρχουν πολλοί τρόποι για μείωση του χρόνου των προσομοιώσεων, της αξιολόγησης και της ανάλυσης.
- Όμως υπάρχει και άπειρος αριθμός επιλογών που μπορούν να εμφανιστούν κατά την διεκπεραίωσης της διαδικασίας εύρεσης ενός σύγχρονου replacement policy. Παράδειγμα είναι ο ορισμός των παραμέτρων και το εύρους τιμών κάθε παραμέτρου των αλγορίθμων.
- Μια λύση για το προηγούμενο πρόβλημα είναι συμβάσεις ανάλογα με το πόση υπολογιστική ώρα διαθέτουμε και η καθοδήγηση με υπάρχουσα γνώση και προηγούμενες αναλύσεις αποτελεσμάτων, όπως έγινε στο κεφάλαιο 9.

10.2 Μελλοντικό Έργο

Η πειραματική αξιολόγηση της Εξερεύνηση παραλλαγών του DRRIP με 4 επιμέρους policies είναι ένα βήμα που αφέθηκε ως μελλοντικό έργο λόγω έλλειψης χρόνου. Σύμφωνα με τα αποτελέσματα στο κεφάλαιο 8 ένας τέτοιος μηχανισμός όπως αυτός που περιγράφεται στο κεφάλαιο 9 είναι αρκετά υποσχόμενος.

Το αμέσως επόμενο βήμα ήταν να τρέξουμε ένα Fine-Tuner ο οποίος είναι ένας κώδικας που εξερευνά πώς ένα replacment policy μεταβάλλοντας μία μία τις παραμέτρους του, να πετύχουμε καλύτερο timing speedup. Αρχικά θα του δώσουμε σαν input τις παραμέτρους του replacement policy της επιλογής μας και αυτό θα εξετάζει σε γραμμικό χρόνο ποια είναι η βέλτιστη τιμή για κάθε παράμετρο από το αντίστοιχο καθορισμένο εύρος τιμών. Θα γίνονται επαναλήψεις μέχρι να μην βελτιώνεται άλλο και

έτσι θα προκύψει μια βελτιωμένη έκδοση του replacement policy της επιλογής μας. Η τεχνική αυτή μοιάζει με τα “permutations” στο [20].

Η διπλωματική εργασία αυτή έχει ερευνητικό χαρακτήρα με την έννοια ότι προσπαθήσαμε παράλληλα να εξερευνήσουμε ήδη ένα μεγάλο αριθμό από replacement policy variations. Όμως ήδη περιορίσαμε αρκετά το Design Space Exploration σε σχέση με τους πιθανούς συνδυασμούς αναθέσεων, παρόλο προέκυψαν 333600 παραλλαγές του DRRIP. Έτσι χωρίς αλλαγές στους αλγόριθμους, μπορεί κάποιος να δημιουργήσει νέα εξερεύνηση με διαφορετικά αλλά και πολύ περισσότερα replacement policies.

Κάτι άλλο που θα μπορούσε να γίνει είναι να διασταυρωθούν τα αποτελέσματα με διαφορετική σειρά από Benchmarks όπως γίνεται και εδώ για επαλήθευση ότι δεν προέκυψαν καλά αποτελέσματα λόγω τυχαιότητας [20].

Άλλο πιθανό μελλοντικό έργο:

- Μελέτη ή επιμήκυνση των αντιπροσωπευτικών περιοχών έτσι ώστε να σιγουρευτούμε ότι παίρνουμε πιο αντιπροσωπευτικά αποτελέσματα.
- Αξιολόγηση της επίδοσης για Multi-program περιβάλλοντα.
- Μελέτη Thread-aware παραλλαγών των ίδιων replacement policies.

Ιδέες που πιστεύω ότι θα άξιζε να διερευνηθούν επίσης:

- Προσθήκη βαρύτητας στη μία πλευρά των PSEL counters έτσι ώστε να προκύπτει το ένα από τα επιμέρους policies με περισσότερη ευκολία.
- Στην θέση των PSEL counters να υπάρχει μια δομή που να επιστρέφει την συχνότητα με την οποία το ένα από τα δύο επιμέρους policies υπερτερούσε του άλλου στα τελευταία N accesses
    - Σαν υλοποίηση για παράδειγμα θα μπορούσαμε να υιοθετούσαμε το same-set SDMs και να αποθηκεύαμε τα γεγονότα Hit/Miss και Miss/Hit μόνο ως 0 και 1 αντίστοιχα. Ακολούθως, θα μπορούσαμε να μειώσουμε τον αριθμό των bits της δομής με διάφορες προσεγγίσεις Data Mining.

- Πειραματισμός με την διευθυνσιοδότηση του λειτουργικού συστήματος έτσι ώστε να υποβοηθηθεί η λειτουργία της Last-Level Cache
    - Για παράδειγμα να ομοιογενοποιείται ο αριθμός των accesses σε κάθε set ή να υλοποιηθούν διαφορετικά οι τεχνικές cache partitioning ανά thread και συμπεριφορά όπως για παράδειγμα αυτές που περιγράφονται στο [21]

Βιβλιογραφία

[1] Aamer Jaleel, Kevin Theobald, Simon C. Steely Jr, and Joel Emer "High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)", In International Symposium on Computer Architecture (ISCA), Saint-Malo, France, June 2010

[2] "CMP$im: A Pin-Based On-The-Fly Multi-Core Cache Simulator", Aamer Jaleel, Robert S. Cohn, Chi-Keung Luk, and Bruce Jacob. In the Fourth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA'2008

[3] "Pin - A Dynamic Binary Instrumentation Tool" - https://software.intel.com/en-us/articles/pintool

[4] Herb Sutter, "The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software", - http://www.gotw.ca/publications/concurrency-ddj.htm

[5] "The greatest tech U-Turns of all time: Intel and Netburst" http://www.pcauthority.com.au/News/163122,the-greatest-tech-u-turns-of-all-time-intel-and-netburst.aspx

[6] S. Khan and D. A. Jiménez Computer Design (ICCD), "Insertion policy selection using Decision Tree Analysis", 2010 IEEE International Conference on, Amsterdam, 2010

[7] "SPEC CPU2006 Benchmark Descriptions", ACM SIGARCH Computer Architecture News, Vol. 35, No. 1 - March 2007

[8] "1st JILP Workshop on Computer Architecture Competitions (JWAC-1): Cache Replacement Championship" http://www.jilp.org/jwac-1/

[9] Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. “Adaptive insertion policies for high performance caching”, 2007. In Proceedings of the 34th annual international symposium on Computer architecture (ISCA '07)

[10] Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely, Jr., and Joel Emer. 2011. “SHiP: signature-based hit predictor for high performance caching”. In Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-44)

[11] Henry Wong “Intel Ivy Bridge Cache Replacement Policy” - http://blog.stuffedcow.net/2013/01/ivb-cache-replacement/

[12] The Journal of Instruction-Level Parallelism - 1st JILP Workshop on Computer Architecture Competitions (JWAC-1): “Cache Replacement Championship” - http://www.jilp.org/jwac-1/

[13] Kenneth Hoste, “SPEC CPU2006 command lines” - http://kejo.be/ELIS/spec_cpu2006/spec_cpu2006_command_lines.html

[14] Hisanobu Tomari - “The Unofficial Guide to Running SPEC CPU2000/2006 in 2013 and later” - http://www-hiraki.is.s.u-tokyo.ac.jp/members/tomari/runspec.html

[15] Cristiano Pereira, Jeremy Lau, Brad Calder, and Rajesh Gupta “Dynamic Phase Analysis for Cycle-Close Trace Generation” - International Conference on Hardware/Software Codesign and System Synthesis, September 2005

[16] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt. “A case for MLP-aware cache replacement”, ISCA-33, 2006

[17] Qureshi, Moinuddin Khalil Ahmed, “Adaptive caching for high-performance memory systems”, The University of Texas at Austin, ProQuest Dissertations Publishing, 2007

[18] Khan, Samira, and Daniel A. Jiménez. "Insertion policy selection using decision tree analysis." Computer Design (ICCD), 2010 IEEE International Conference on. IEEE, 2010.

[19] Wescott, Bob (2013). “The Every Computer Performance Book, Chapter 7: Modeling”. CreateSpace. ISBN 1482657759

[20] Daniel A. Jimenez, “Insertion and Promotion for Tree-Based PseudoLRU Last-Level Caches”, MICRO-46 2013, pp. 284-296, 2013

[21] Aamer Jaleel, Hashem H. Najaf-abadi, Samantika Subramaniam, Simon C. Steely, and Joel Emer. 2012. “CRUISE: cache replacement and utility-aware scheduling”. (ASPLOS XVII). ACM, New York, NY, USA, 249-260

[22] Khan, Samira, and Daniel A. Jiménez. "Insertion policy selection using decision tree analysis." Computer Design (ICCD), 2010 IEEE International Conference on. IEEE, 2010.

Παράρτημα Α

Παράδειγμα output εκτέλεσης του Functional Simulator:

```
philippos@phoenix:~/NetBeansProjects/MODULE7_simulator_2$ ./dist/Debug/GNU-Linux-
x86/module7_simulator_2
Welcome to the LLC Cache Simulator for DRRIP variants!
=====================================================
29 traces with addresses of 6-bit offset will be read_
Ph. Variation: -3  6 6 6 6  1024 16 10 1 5 1 1  0 0  0 0 0 1  0 1 0 0  0 0 0 0  1 1 0 1  2 3 3 3  0 0
3 2  1
-> Number of cache sets = 1024
-> Degree of associativity (ways): 16

Benchmark    Miss Rate %    dHits   dMisses  Accesses  #of_pol_chn MPKI  CPIPred  Speedup
BiasTimes(for1b)
-------------------------------------------------------------------------------------------
400.perlbench  82.430422      8030     37674     55497      22   0.150696 0.298448 1.001226        0
   401.bzip2  41.611579    871693    621228   2033259      34   2.484912 0.457586 0.938831        0
     403.gcc  84.718158     10340     57322     88570       1   0.229288 0.375058 0.999833        0
  410.bwaves  99.759411     10059   4170931   5254671       1  16.683724 1.279636 1.002066        0
  416.gamess  75.605880      1228      3806      5458       1   0.015224 0.269657 0.999875        0
     429.mcf  79.984716    420105   1678816   2906756       2   6.715264 0.883978 1.111489        0
    433.milc  99.713857     10329   3599407   4756432       2  14.397628 1.421182 1.001708        0
  434.zeusmp  98.184636     16869    912366   1276522      25   3.649464 0.714738 1.009803        0
 435.gromacs  46.392053    193091    167100    489154       5   0.668400 0.363272 0.992647        0
436.cactusADM  84.997848    172177    975505   1459750      78   3.902020 0.669939 1.070938        0
437.leslie3d  74.676104    810761   2390804   4015031      49   9.563216 1.049679 1.005688        0
    444.namd  70.941368     10165     24816     41295      11   0.099264 0.267014 1.000747        0
   445.gobmk  16.499276    517211    102198    846001       0   0.408792 0.388667 0.995385        0
  447.dealII  25.030341    304541    101678    516160       4   0.406712 0.347330 1.019843        0
  450.soplex  80.238124   1097989   4458108   8557444     222  17.832432 1.655799 0.993122        0
  453.povray  88.414955       440      3358      3907       1   0.013432 0.344944 0.999941        0
454.calculix  99.654219        10      2882      2908       1   0.011528 0.251847 1.000000        0
   456.hmmer  16.467912    500201     98612   1182123      59   0.394448 0.313342 1.021772        0
   458.sjeng  89.997185      9949     89513    174397       1   0.358052 0.306364 0.999413        0
459.GemsFDTD  94.851297    249557   4597431   6572134       3  18.389724 1.772082 1.042147        0
462.libquantum  97.894212     78396   3644486   4247216       0  14.577944 1.701856 1.022243        0
 464.h264ref  60.984443    147915    231203    533733      10   0.924812 0.409637 1.067095        0
   465.tonto  10.227391    146982     16745    275340      16   0.066980 0.296449 0.999665        0
     470.lbm  95.779238    275372   6248852  11328289       5  24.995408 0.828486 1.000324        0
 471.omnetpp  84.822641    468441   2618005   4351673      20  10.472020 1.556485 0.976546        0
   473.astar  42.946868   2996018   2255259   7151470     128   9.021036 0.917384 0.843193        0
     481.wrf  21.555942     52745     14494    120646      16   0.057976 0.254669 1.000070        0
 482.sphinx3  78.915741    799179   2991227   4026967       6  11.964908 2.346410 1.013561        0
483.xalancbmk  15.820570   2430777    456837   2968076      46   1.827348 0.772868 0.978550        0

AVG Miss Rate = 67.555737 %
AVG Misses = 1467953.896552
AVG SPEEDUP (PRED) 1.003715
Compressing statistics... [OK]
```

Παράρτημα Β

Παράδειγμα απλούστερου Functional Simulator που υλοποιεί τα replacement policies Random, FIFO, LRU, LIP και BIP και λειτουργεί με address traces.

(Ο Functional Simulator μου με τον οποίο αξιολογώ παραλλαγές του DRRIP δεν παρατίθεται διότι καταλαμβάνει μερικές χιλιάδες γραμμές).

```
/*
 * File:   cache.c
 * Author: Philippos Papaphilippou
 *
 * Created on July 8, 2013, 11:15 AM
 * This code is licensed under GNU GPL
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef struct Block {
    int Tag;
    struct Block* next;
} BLOCK;

typedef struct Set {
    BLOCK* Front;
    int Size;
    int misses;
    int hits;
} SET;

typedef struct Cache {
    SET **set;
    int SetSize;
    int nofSets;
    int nofSetsB;   // (for log2 of nofSets)
    int BlockSize;
    int BlockSizeB; // (for log2 of BlockSize)
} CACHE;

enum mode {Random, FIFO, LRU, LIP, BIP};
float epsilon; // bimodal throttle parameter for BIP

int exists(SET *s, int tag);
void hit(CACHE *c, int set, int tag);
void replace(CACHE *c, int set, int tag, int mode);
void printC(CACHE *c, FILE* out, int on);
void initC(CACHE *c, int n, int setsize, int m);
int findSet(CACHE *c,int Address);
int power(int a, int b);

int exists(SET *s, int tag) {
    BLOCK *cur = s->Front;
    while (cur != NULL) {
        if (tag == cur->Tag)
            return 1;
        cur = cur->next;
    }
    return 0;
}

void replace(CACHE *c, int set, int tag, int mode) {
    BLOCK* n = malloc(sizeof (BLOCK));
    n->Tag = tag;
```

```
    n->next = c->set[set]->Front;

    if (c->set[set]->Size < c->SetSize){
        (c->set[set]->Size)++;
        c->set[set]->Front = n;

    } else {

        if (mode == BIP){
            if(rand()%1000 < epsilon*1000)
                mode = LRU;
            else
                mode = LIP;
        }

        int pos = 0;
        if (mode == Random) pos = rand()% c->SetSize;
        if (mode == LIP) pos = c->SetSize -1;
        BLOCK *cur = c->set[set]->Front;
        BLOCK *prev = NULL;

        while (cur != NULL) {
            if (mode==Random || mode==LIP) {
                if (pos == 0)
                    break;
                else
                    pos--;
            }
            if((mode==FIFO || mode==LRU) && cur->next == NULL) break;

            prev = cur;
            cur = cur->next;
        }

        if (mode==Random || mode==LIP) {
            n->next = cur->next;
            if (prev)
                prev->next = n;
            else
                c->set[set]->Front = n;
        }

        if(mode==FIFO || mode==LRU){
            c->set[set]->Front = n;
            if(prev) prev->next=NULL;
        }

        free(cur);
        cur = NULL;
    }
}

void hit(CACHE *c, int set, int tag) {
    BLOCK *prev = NULL;
    BLOCK *cur = c->set[set]->Front;
    while (cur != NULL) {
        if(tag == cur->Tag)
            break;
        prev = cur;
        cur = cur->next;
    }

    if (prev == NULL) return;
    if (cur == NULL) {
        printf("ERROR");
        exit(-1);
    }

    prev->next = cur->next;
    cur->next = c->set[set]->Front;
    c->set[set]->Front = cur;
}

void printC(CACHE *c, FILE *out, int on) {
    int set, z;
    for (set = 0; set < c->nofSets; set++) {
```

```
            fprintf(out,"%2d", set);
            BLOCK *n = c->set[set]->Front;
            int i;
            for (i = 0; i < c->SetSize; i++) {
                if (n != NULL) {
                    fprintf(out,"|%8d", n->Tag);
                    n = n->next;
                } else
                    fprintf(out,"|%8c", ' ');
            }
            if (set == on)
                fprintf(out,"|<-\n");
            else
                fprintf(out,"|\n");
        }

        for (z = 0; z < 3 + (8 + 1) * c->SetSize; z++)
            fprintf(out,"-");
        fprintf(out,"\n");
    }

    void initC(CACHE *c, int n, int setsize, int m){
        c->SetSize = setsize;
        c->nofSetsB = n;
        int nofsets = power(2,n);
        c->nofSets = nofsets;
        c->set = malloc(nofsets*sizeof(SET*));
        for(nofsets = 0; nofsets < c->nofSets; nofsets++){
            c->set[nofsets] = malloc(sizeof(SET));
            c->set[nofsets]->Front=NULL;
            c->set[nofsets]->Size=0;
            c->set[nofsets]->misses=0;
            c->set[nofsets]->hits=0;
        }
        c->BlockSizeB = m;
        c->BlockSize  = power(2,m);
    }

    int findSet (CACHE *c,int Address){
        int n = c->nofSetsB;
        int i, set = 0;

        for (i=0; i<n; i++)
            set |= (Address & (1<<i));

        return set;
    }

    // (to avoid linked library Math for pow())
    int power(int a, int b){
        if(b==0)
            return 1;
        int i;
        int base=a;
        for(i=1;i<b;i++){
            a*=base;
        }
        return a;
    }

    int main(int argc, char** argv) {
        FILE *fp, *out;
        int hits = 0, misses = 0, line, mode, n, m, setsize, set, offset, tag;

        printf("Welcome to the Cache Simulator!");
        printf("\n================================");

        if ((fp = fopen("input.dat", "r")) == NULL) {
            printf("\nCouldn't find input.dat\n");
            return EXIT_FAILURE;
        }
        if ((out = fopen("output.dat", "w")) == NULL) {
            printf("Couldn't write to output.dat\n");
            return EXIT_FAILURE;
        }
        printf("\nChoose replacement policy:\n0) Random"
```

```
            "\n1) FIFO (First in - First out)"
            "\n2) LRU  (Least Recently Used)"
            "\n3) LIP  (LRU Insertion Policy)"
            "\n4) BIP  (Bimodal Insertion Policy)\n");

    do {printf(">>> "); scanf("%d",&mode);} while (mode<0 || mode>4);

    fprintf(out,"Replacement policy: ");
    switch (mode) {
        case Random:
            srand(time(NULL));
            fprintf(out,"Random\n");
            break;
        case FIFO:
            fprintf(out,"FIFO (First in - First out)\n");
            break;
        case LRU:
            fprintf(out,"LRU (Least Recently Used)\n");
            break;
        case LIP:
            fprintf(out,"LIP (LRU Insertion Policy)\n");
            break;
        case BIP:
            srand(time(NULL));
            epsilon = (float)1/32;
            fprintf(out,"BIP (Bimodal Insertion Policy)\n");
            break;
    }

    printf("Enter nof index bits for the sets of the Cache (n): ");
    scanf("%d",&n);
    printf(     "-> Number of cache sets = %d\n",power(2,n));
    fprintf(out,"-> Number of cache sets = %d\n",power(2,n));

    printf("Enter nof index bits for the Block Size (nof words in block m):
");
    scanf("%d",&m);
    printf(      "-> Number of words in block = %d (offset = %d)\n",
power(2,m), 2+m);
    fprintf(out, "-> Number of words in block = %d (offset = %d)\n",
power(2,m), 2+m);

    printf("Enter the degree of associativity (set size): ");
    scanf("%d",&setsize);
    fprintf(out,  "-> Degree of associativity (set size): %d\n",setsize);

    CACHE *c = malloc(sizeof(CACHE));
    initC(c, n, setsize, m);
    offset = 2 + c->BlockSizeB;

    while (!feof(fp)) {
        fscanf(fp, "address %d", &line);
        fscanf(fp, " \n\t ");
        set = findSet(c, line >> offset);
        tag = line >> (offset + c->nofSetsB);
        fprintf(out, "Address = %d, Tag = %d, set = %d/%d ", line, tag, set,
                c->nofSets);

        if (exists(c->set[set], tag)) {
            if(mode==LRU || mode==LIP || mode== BIP) hit(c, set, tag);
            hits++;
            (c->set[set]->hits)++;
            fprintf(out, "(hit)\n");

        } else {
            replace(c, set, tag, mode);
            misses++;
            (c->set[set]->misses)++;
            fprintf(out,"(miss)\n");
        }
        printC(c, out, set);
    }
    fclose(fp);

    // Print results and Free allocated memory
    printf("\nGLOBAL: %d hits, %d misses",hits, misses);
```

```
    printf("\nHit rate  = %f %%", (double) hits / (hits + misses)*100);
    printf("\nMiss rate = %f %%\n", (double) misses / (hits + misses)*100);
    printf("\n(Replacements and detailed miss rates are captured in file "
           "output.dat)\n");

    for (set = 0; set < c->nofSets; set++) {
        fprintf(out, "\nSET %d: %d hits, %d misses", set, c->set[set]->hits,
                 c->set[set]->misses);
        fprintf(out, "\nHit rate  = %f %%",   (double) c->set[set]->hits   /
                (c->set[set]->hits + c->set[set]->misses) *100);
        fprintf(out, "\nMiss rate = %f %%\n", (double) c->set[set]->misses /
                (c->set[set]->hits + c->set[set]->misses) *100);
        BLOCK *n = c->set[set]->Front;
        while (n != NULL) {
            c->set[set]->Front = c->set[set]->Front->next;
            free(n);
            n = c->set[set]->Front;
        }
        free(c->set[set]);
    }
    free(c);

    fprintf(out, "\nGLOBAL: %d hits, %d misses",hits, misses);
    fprintf(out, "\nHit rate  = %f %%", (double) hits / (hits + misses)*100);
    fprintf(out, "\nMiss rate = %f %%\n", (double) misses / (hits +
misses)*100);
    fclose(out);
    return (EXIT_SUCCESS);
}
```