



Διδακτορική Διατριβή

Διαχείριση Κοινόχρηστων Πόρων Σε Πολυεπεξεργαστικά Συστήματα Ενός Ολοκληρωμένου

ΠΑΥΛΟΣ ΑΘΑΝ. ΠΕΤΟΥΜΕΝΟΣ

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και
Τεχνολογίας Υπολογιστών

Τμήμα Ηλεκτρολόγων Μηχανικών
και Τεχνολογίας Υπολογιστών

Πολυτεχνική Σχολή

Πανεπιστήμιο Πατρών

Αριθμός Διατριβής: 275

Πάτρα, Ιούνιος 2011

Πιστοποίηση

Πιστοποιείται ότι η παρούσα διδακτορική διατριβή με θέμα:

“Διαχείριση Κοινόχρηστων Πόρων σε Πολυεπεξεργαστικά Συστήματα Ενός Ολοκληρωμένου”

του Πετούμενου Παύλου του Αθανασίου, Διπλωματούχου Ηλεκτρολόγου Μηχανικού & Τεχνολογίας Υπολογιστών παρουσιάστηκε δημοσίως στο Τμήμα Ηλεκτρολόγων Μηχανικών & Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών την 30^η Ιουνίου 2011 και εξετάστηκε και εγκρίθηκε από την ακόλουθη Εξεταστική Επιτροπή:

Στέφανος Καζίρας, Επίκουρος Καθηγητής Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών, Επιβλέπων Συμβουλευτικής Επιτροπής

Σταύρος Κουμπιάς, Καθηγητής Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών, Μέλος Συμβουλευτικής Επιτροπής

Δημήτριος Σερπάνος, Καθηγητής Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών, Μέλος Συμβουλευτικής Επιτροπής

Οδυσσέας Κουφοπαύλου, Καθηγητής Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών

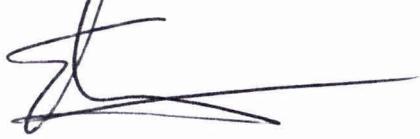
Δημήτριος Νικολός, Καθηγητής Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών

Διονύσιος Πνευματικάτος, Καθηγητής Πολυτεχνείου Κρήτης

Ευθύμιος Χούσος, Καθηγητής Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών

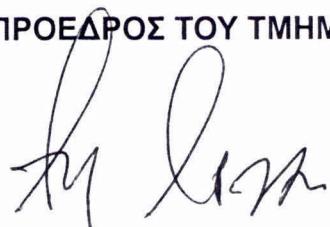
Πάτρα, 30η Ιουνίου 2011

Ο ΕΠΙΒΛΕΠΩΝ



Στέφανος Καζίρας, Επίκ. Καθηγητής

Ο ΠΡΟΕΔΡΟΣ ΤΟΥ ΤΜΗΜΑΤΟΣ



Αντώνιος Τζης, Καθηγητής

Πρόλογος

Η παρούσα διδακτορική διατριβή εκπονήθηκε στο Εργαστήριο Ηλεκτρονικών Εφαρμογών (APEL) του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών του Πανεπιστημίου Πατρών κατά την περίοδο Οκτώβριος 2005 - Ιούνιος 2011. Το αντικείμενο που πραγματεύεται είναι η διαχείριση των κοινόχρηστων πόρων των υπολογιστικών συστημάτων με πολλαπλούς επεξεργαστές σε ένα ολοκληρωμένο. Η τριμελής Συμβουλευτική Επιτροπή αποτελείται από τον κ. Στέφανο Καζίρα, Επίκουρο Καθηγητή (Επιβλέπων), τον κ. Σταύρο Κουμπιά, Καθηγητή (Μέλος) και τον κ. Δημήτριο Σερπάνο, Καθηγητή (Μέλος).

Αισθάνομαι την ανάγκη να ευχαριστήσω όλους όσους συνετέλεσαν στην ολοκλήρωση της διατριβής. Καταρχήν, θα ήθελα να εκφράσω τις ευχαριστίες μου στον επιβλέποντα καθηγητή μου κ. Στέφανο Καζίρα για την εμπιστοσύνη που μου έδειξε και την πολύτιμη καθοδήγησή που μου παρείχε στην διάρκεια αυτών των πεντέμισι χρόνων. Οι συζητήσεις μας και οι ιδέες του καθόρισαν σε εξαιρετικό βαθμό τις εργασίες που παρουσιάζονται σε αυτήν την διατριβή και γενικότερα την εξέλιξή μου σαν ερευνητή. Επιπλέον, ιδιαίτερα σημαντική ήταν και η δυνατότητα που μου έδωσε να απασχοληθώ στα ευρωπαϊκά ερευνητικά προγράμματα SARC (Scalable Computer ARChitecture), HEAP (a Highly Efficient Adaptive multi-Processor), ERA (Embedded Reconfigurable Architecture) και HiPEAC (High Performance and Embedded Architecture and Compilation), καθώς και στο πρόγραμμα βασική έρευνας του Πανεπιστημίου Πατρών “Κ. Καραθεοδωρή”.

Ιδιαίτερες ευχαριστίες οφείλω επίσης και στον Καθηγητή κ. Σταύρο Κουμπιά, Διευθυντή του Εργαστηρίου, και τον Καθηγητή κ. Δημήτριο Σερπάνο. Χωρίς την βοήθεια και την υποστήριξη τους θα ήταν δύσκολη η επιτυχής ολοκλήρωση της διατριβής μου, και σίγουρα αδύνατη η έγκαιρη ολοκλήρωσή της. Ειδικά στον κ. Κουμπιά χρωστάω την βαθιά ευγνωμοσύνη μου για το ενεργό ενδιαφέρον που έδειξε στο να ξεπεραστούν όλοι οι σκόπελοι των τελευταίων μηνών.

Μέσα σε αυτά τα πεντέμισι χρόνια είχα την τύχη να συνεργαστώ με πολλούς καλούς ερευνητές. Από το Πανεπιστήμιο της Uppsala, με τον καθηγητή Erik Hagersten και τον διδάκτωρα Håkan Zeffler εργαστήκαμε μαζί στις πρώτες μου ερευνητικές εργασίες. Πέρα από το ερευνητικό σκέλος της συνεργασίας μας, ο κ. Hagersten μου έδωσε την ευκαιρία να εργαστώ για ένα εξάμηνο στην εταιρεία Acumem, μία από τις πιο πολύτιμες εμπειρίες που είχα τα τελευταία χρόνια, και για αυτό τον ευχαριστώ ξεχωριστά. Από το Πανεπιστήμιο της

Murcia, συνεργάστηκα κατά το δεύτερο μισό του διδακτορικού μου με τον καθηγητή Juan Luis Aragon και τον υποψήφιο διδάκτωρα Juan Manuel Cebrián González. Από το Πανεπιστήμιο Πατρών, παρα το ότι δεν καταφέραμε να δημοσιεύσουμε κάποια εργασία μαζί, χαίρομαι για κάθε λεπτό της συνεργασίας με τον Βασιλή Σπηλιώπουλο και τον Γιάννη Οικονόμου. Επίσης, είχα την χαρά να συμβουλεύσω αρκετούς προπτυχιακούς φοιτητές, κατά την διάρκεια των διπλωματικών τους, με τους περισσότερους από τους οποίους καταφέραμε να παράξουμε εργασίες για τις οποίες είμαι πραγματικά περήφανος.

Τέλος, πρέπει να αναφέρω ξεχωριστά τρεις ανθρώπους που πέρα από στενοί συνεργάτες, υπήρξαν και από τους πιο πολύτιμους φίλους μου: τον διδάκτορα του Πανεπιστημίου Πατρών Γιώργο Κεραμίδα, τον διδάκτορα του Πανεπιστημίου του Edinburgh Χρόνη Ξεκαλάκη και την υποψήφια διδάκτορα του Πανεπιστημίου του Aachen Γεωργία Ψύχου. Με τον Γιώργο δουλέψαμε μαζί σε σχεδόν όλες τις ερευνητικές εργασίες μου, βοήθησε ενεργά στις υπόλοιπες και πρότεινε αρκέτες από τις ιδέες που παρουσιάζονται σε αυτήν την διατριβή. Χωρίς τον Γιώργο, αυτή η διατριβή θα ήταν πολύ διαφορετική και, σχεδόν σίγουρα, πιο φτωχή. Με τον Χρόνη, η συνεργασία μας αυτή καθ'αυτή ήταν σύντομη, αλλά σε όλη την διάρκεια αυτών των πεντέμισι χρόνων είχαμε μία διαρκή ανταλλαγή ιδεών, η οποία μου προσέφερε μία πολύ πιο βαθιά κατανόηση της Αρχιτεκτονικής Υπολογιστών. Με την Γεωργία, εργαστήκαμε μαζί για την ανάπτυξη της τελευταίας μεθοδολογίας που με απασχόλησε ερευνητικά. Η βοήθεια και η επιμονή της ήταν καθοριστικές για την επιτυχή ανάπτυξη της μεθοδολογίας. Και τους τρεις τους ευχαριστώ θερμά, και για την συνεργασία μας αλλά και για την φιλία μας.

Ιδιαίτερες ευχαριστίες χρειάζονται στους επιστήθιους φίλους που με στήριξαν όλα αυτά τα χρόνια. Γιώργο Ρ., Κώστα Μ., Μαρία, Κώστα Σ. και Ναταλία, σας ευχαριστώ για την βοήθεια και την ενθάρρυνση που μου προσφέρατε.

Τέλος, η διατριβή μου, όπως και κάθε τι άλλο στην ζωή μου, αφιερώνεται στην οικογένειά μου: στους γονείς μου, Θανάση και Γεωργία, στον αδερφό μου, Νίκο, στην σύζυγό του, Ιωάννα, και στην μικρή μου ανιψιά, την Δανάη.

Παύλος Α. Πετούμενος

Πάτρα, Ιούνιος 2011

Περιεχόμενα

Περίληψη	xvii
Abstract	xxi
Κεφάλαιο 1: Εισαγωγή, Συνεισφορές και Οργάνωση	1
1.1 Πολυεπεξεργαστικά συστήματα ενός ολοκληρωμένου	1
1.2 Συνεισφορές	3
1.2.1 Μελέτη & διαχείριση του διαμοιρασμού κοινόχρηστων κρυφών μνημών	3
1.2.2 Μηχανισμός αντικατάστασης κρυφών μνημών	4
1.2.3 Διαχείριση της ουράς εντολών	5
1.3 Οργάνωση της Διατριβής	7
Κεφάλαιο 2: Βασικές Έννοιες Αρχιτεκτονικής Υπολογιστών	11
2.1 Πολυεπεξεργαστές ενός Ολοκληρωμένου	11
2.2 Η Ιεραρχία Μνήμης στους Πολυεπεξεργαστές ενός Ολοκληρωμένου	13
2.2.1 Το Χάσμα Μνήμης - Επεξεργαστή	13
2.2.2 Κρυφές Μνήμες και Πολυεπεξεργαστές ενός Ολοκληρωμένου	14
2.3 Η μεθοδολογία StatCache	15
2.4 Memory Level Parallelism	18
2.5 Η Τεχνική Cache Decay	21
Κεφάλαιο 3: Μοντελοποίηση και Διαχείριση του Διαμοιρασμού Κοινόχρηστων Κρυφών Μνημών	25
3.1 Εισαγωγή	25
3.2 Σχετικές Ερευνητικές Εργασίες	27
3.2.1 Τεχνικές Διαμοιρασμού Κοινόχρηστων Κρυφών Μνημών	27
3.2.2 Η Μεθοδολογία StatCache	28
3.3 Το στατιστικό μοντέλο StatShare	29
3.3.1 Χρόνος σε CAT	29
3.3.2 Ιστογράμματα των CAT reuse distances	30
3.3.3 Βασικές εξισώσεις	31

3.4	Spacetime	33
3.4.1	Spacetime μίας εφαρμογής	33
3.4.2	Spacetime των ευστοχών	34
3.4.3	Spacetime των αστοχιών	34
3.4.4	Επιπτώσεις του Decay στο Spacetime	35
3.5	Ενσωμάτωση του Decay στο Θεωρητικό Μοντέλο	36
3.5.1	Decayed εξισώσεις f	37
3.5.2	Decayed εξισώσεις του spacetime	40
3.6	Υλοποίηση	41
3.6.1	Συλλογή Ιστογραμμάτων	41
3.6.2	Υλοποίηση του Decay και της Πολιτικής Αντικατάστασης	44
3.6.3	Αποφάσεις διαχείρισης στο επίπεδο του Λειτουργικού Συστήματος	44
3.7	Πειραματική Μελέτη	45
3.7.1	Επιβεβαίωση της ακρίβειας του μοντέλου (2 επεξεργ.)	45
3.7.2	Επιβεβαίωση της ακρίβειας του μοντέλου (4 επεξεργ.)	47
3.7.3	Μελέτη των επιπτώσεων της διαχείρισης με CAT Decay (2 επεξεργ.)	49
3.7.4	Μελέτη των επιπτώσεων της διαχείρισης με CAT Decay (4 επεξεργ.)	51
3.7.5	Παροχή εγγυήσεων Ποιότητας Εξυπηρέτησης (QoS)	53
3.8	Ανακεφαλαίωση	55

Κεφάλαιο 4: Αποδοτική Πολιτική Αντικατάστασης σε Κρυφές Μνήμες 57

4.1	Εισαγωγή	57
4.2	Πρόβλεψη των Αποστάσεων Επαναχρησιμοποίησης	60
4.2.1	Ορισμός των αποστάσεων επαναχρησιμοποίησης	61
4.2.2	Προβλεψιμότητα των αποστάσεων επαναχρησιμοποίησης	63
4.2.3	Δομές πρόβλεψης	65
4.2.4	Εξερεύνηση των χαρακτηριστικών των μετρητών αξιοπιστίας	69
4.2.5	Εξερεύνηση του μεγέθους του Predictor	70
4.3	Συλλογή Αποστάσεων Επαναχρησιμοποίησης	71
4.3.1	Μηχανισμός συλλογής αποστάσεων επαναχρησιμοποίησης με στόχο την υψηλή απόδοση	72
4.3.2	Μηχανισμός συλλογής αποστάσεων επαναχρησιμοποίησης με στόχο το χαμηλό κόστος	77
4.4	Πολιτική Αντικατάστασης της Κρυφής Μνήμης Τελευταίου Επιπέδου	80
4.4.1	Βέλτιστος αλγόριθμος αντικατάστασης και LRU	80
4.4.2	Αλγόριθμος της πολιτικής αντικατάστασης	82
4.5	Επιλογές Υλοποίησης	84
4.5.1	Κβάντιση των αποστάσεων επαναχρησιμοποίησης	84
4.5.2	Υλοποίηση υψηλής απόδοσης	86
4.5.3	Υλοποίηση χαμηλού κόστους	87

4.6 Πειραματικά Αποτελέσματα	90
4.6.1 Πειραματική μεθοδολογία	90
4.6.2 Μετρήσεις	94
4.7 Πειραματικά Αποτελέσματα Πρωταθλήματος	100
4.7.1 Πειραματική μεθοδολογία	100
4.7.2 Ανεπίσημα Αποτελέσματα	102
4.7.3 Επίσημα Αποτελέσματα	103
4.8 Προηγούμενες Ερευνητικές Εργασίες	107
4.8.1 Πολιτικές Αντικατάστασης	107
4.8.2 Πρόβλεψη των αποστάσεων επαναχρησιμοποίησης	110
4.9 Ανακεφαλαίωση	111

Κεφάλαιο 5: Διαχείριση της Ουράς Εντολών διατηρώντας το MLP 113

5.1 Εισαγωγή	113
5.2 Προηγούμενες Ερευνητικές Εργασίες	115
5.2.1 Μείωση της κατανάλωσης ενέργειας του παράθυρου εντολών	115
5.2.2 MLP-aware τεχνικές	118
5.3 MLP και μεταβολή του μεγέθους της Ουράς Εντολών	120
5.4 Διαχείριση της Ουράς Εντολών με πληροφορία για το MLP	121
5.4.1 Ποσοτικοποιώντας το MLP: MLP-distance	121
5.4.2 Συσχέτιση του MLP-distance με περιοχές του προγράμματος	124
5.4.3 Πολιτική διαχείρισης	124
5.5 Επιλογές Υλοποίησης	125
5.5.1 Κατάτμηση της ουράς εντολών	125
5.5.2 Superpaths	126
5.6 Πειραματικά Αποτελέσματα	127
5.6.1 Πειραματική μεθοδολογία	127
5.6.2 Συνέπειες της MLP-distance πληροφορίας	128
5.6.3 Σύγκριση της μεθοδολογίας ILP/MLP-aware με την ILP-feedback	130
5.7 Ανακεφαλαίωση	134

Κεφάλαιο 6: Συμπεράσματα και Προεκτάσεις 137

6.1 Εισαγωγή	137
6.2 Συνεισφορές της Διδακτορικής Διατριβής	137
6.3 Μελλοντικές Κατευθύνσεις και Προεκτάσεις	138
6.4 Αποτελέσματα Ερευνητικής Δραστηριότητας	140
6.5 Αναφορές των Δημοσιευμένων Εργασιών από άλλες Ερευνητικές Ομάδες .	142

Βιβλιογραφία 147

Παράρτημα Α: Γλωσσάριο	157
A.1 Αντιστοίχηση Αγγλικών - Ελληνικών όρων	157
A.2 Επεξηγήσεις Συντομεύσεων-Ακρωνυμιών	159
Παράρτημα Β: Εργαλεία και Εξομοιωτές	161
B.1 Ο εξομοιωτής Simplescalar	161
B.2 Ο εξομοιωτής Wattch	162
B.3 Ο εξομοιωτής HotLeakage	163
B.4 Ο εξομοιωτής CMP-SMT Simplescalar	163
B.5 Το εργαλείο CACTI	164
Παράρτημα C: Σουίτες Μετροπρογραμμάτων	165
C.1 Η σουίτα SPEC CPU2000	165
C.2 Η σουίτα SPEC CPU2006	167

Κατάλογος Σχημάτων

Σχήμα 1: Εξέλιξη του αριθμού τρανζίστορς σε ένα ολοκληρωμένο	12
Σχήμα 2: Καμπύλες StatCache για μερικά από τα μετροπρογράμματα της σουίτας SPEC2000	15
Σχήμα 3: Reuse Distances - Κάθε κουτί αντιπροσωπεύει μία προσπέλαση κρυφής μνήμης στην διεύθυνση που περιέχει το κουτί. Τα βέλη δείχνουν την επαναχρησιμοποίηση κάθε γραμμής και ο αριθμός δίπλα τους το reuse distance	16
Σχήμα 4: IPC του επεξεργαστή κατά την διάρκεια εξυπηρέτησης μεμονωμένης αστοχίας στο τελευταίο επίπεδο κρυφής μνήμης	18
Σχήμα 5: IPC του επεξεργαστή κατά την διάρκεια εξυπηρέτησης δύο παράλληλων αστοχιών στο τελευταίο επίπεδο κρυφής μνήμης	20
Σχήμα 6: Ζωή ενός δεδομένου στην κρυφή μνήμη	22
Σχήμα 7: Καμπύλες StatCache	28
Σχήμα 8: Ιστογράμματα $h(i)$, για το art και το equake (λογαριθμικοί άξονες)	30
Σχήμα 9: f και f εξισώσεις για Random πολιτική αντικατάστασης και πλήρως συσχετιστική κρυφή μνήμη	31
Σχήμα 10: Ιστογράμματα αστοχιών πολλαπλασιάζοντας το $h(i)$ με την $f(i)$	32
Σχήμα 11: Ιστογράμματα ευστοχιών πολλαπλασιάζοντας το $h(i)$ με την $f(i)$	32
Σχήμα 12: Spacetime των ευστοχιών και των αστοχιών για το equake (ο χ άξονας είναι γραμμικός)	35
Σχήμα 13: Spacetime των ευστοχιών και των αστοχιών για το art (ο χ άξονας είναι γραμμικός)	35
Σχήμα 14: Πειραματικές και θεωρητικές καμπύλες f . για το art και το equake σε 256KB κρυφή μνήμη, με μειούμενα Decay Times να εφαρμόζονται στο art	39
Σχήμα 15: f καμπύλη και μπάρες που προσεγγίζουν την f για εκθετικά κβαντισμένα reuse distances	42
Σχήμα 16: Ιστογράμματα του art και του equake όταν μοιράζονται μία 256KB κρυφή μνήμη, το πρώτο ιστόγραμμα προκύπτει από συλλογή όλων των reuse distances, το δεύτερο από δειγματοληψία 1:1024, το τρίτο από κβάντιση των reuse distances.	43
Σχήμα 17: mcf-parser: ποσοστά αστοχιών, murphy αντικαταστάσεις και κατειλημμένος χώρος από εξομοιώσεις και θεωρητικό μοντέλο	46

Σχήμα 18: mcf-parser-equake-vpr: πειραματικά αποτελέσματα για την κατανομή χώρου και τα ποσοστά αστοχιών	48
Σχήμα 19: mcf-parser-equake-vpr: αποκλίσεις μεταξύ εξομοιώσεων και θεωρητικού μοντέλου για την κατανομή χώρου και τα ποσοστά αστοχιών	48
Σχήμα 20: art-equake και art-gzip, κατειλημμένος χώρος, decayed χώρος & χρήσιμος χώρος	50
Σχήμα 21: art-equake και art-gzip, ποσοστά αστοχιών και Murphу αντικαταστάσεις	50
Σχήμα 22: Ιστογράμματα του art και του gzip, όταν μοιράζονται μία 256KB κρυφή μνήμη	51
Σχήμα 23: art-gzip-mcf-equake: πειραματικά αποτελέσματα για την κατανομή χώρου και τα ποσοστά αστοχιών, εφαρμόζοντας διάφορα Decay Time σε καμία, μία ή όλες τις εφαρμογές (x-άξονας)	52
Σχήμα 24: Χώρος που καταλαμβάνουν το art (αριστερά) και το mcf (δεξιά) όταν εκτελούνται μόνα τους σε κρυφή μνήμη των 256KB, για διάφορα Decay Time	53
Σχήμα 25: Χώρος που καταλαμβάνουν το tramp και το gzip και κανονικοποιημένα ποσοστά αστοχιών για διάφορα Decay Times στο tramp	54
Σχήμα 26: Φωτογραφίες του ολοκληρωμένου από 4 διαφορετικές γενιές επεξεργαστών της Intel. Το έγχρωμο/έντονο κομμάτι κάθε φωτογραφίας, δείχνει το τελευταίο επίπεδο κρυφής μνήμης	58
Σχήμα 27: Παράδειγμα απλού προγράμματος	60
Σχήμα 28: Οι πιο σημαντικές εντολές προσπέλασης του art, του mcf και του vpr, και η κατανομή των αποστάσεων επαναχρησιμοποίησης της κάθε εντολής	64
Σχήμα 29: Οργάνωση και δομή του Instruction-based Reuse Distance Predictor.	66
Σχήμα 30: Ακρίβεια και κάλυψη των προβλέψεων του Predictor, για διαφορετικούς μετρητές αξιοπιστίας	68
Σχήμα 31: Ακρίβεια και κάλυψη των προβλέψεων του Predictor, για διαφορετικά μεγέθη του Predictor	70
Σχήμα 32: Single-Prediction Sampler	73
Σχήμα 33: Εντολή με πολύ μικρές και πολύ μεγάλες αποστάσεις επαναχρησιμοποίησης. Το πάνω σενάριο δείχνει δειγματοληψία μεγάλης απόστ. επαναχρ. η οποία μπλοκάρει 6 μικρές και 1 μεγάλη απόστ. επαναχρ. Το κάτω σενάριο δείχνει δειγματοληψία μικρής απόστ. επαναχρ., η οποία μπλοκάρει μόνο μία μικρή απόστ. επαναχρ.	74
Σχήμα 34: Εντολή με πολύ μικρές και πολύ μεγάλες αποστάσεις επαναχρησιμοποίησης. Ο SRDS μπορεί να συλλάβει παράλληλα πολλαπλές μικρές απόστ. επαναχρ. ενώ ο SPS συλλαμβάνει μόνο μία μεγάλη απόστ. επαναχρ.	75

Σχήμα 35: Ακρίβεια και κάλυψη του Predictor για τροφοδότηση από τον Full Sampler, τον SPS ή τον συνδυασμό SPS και SRDS	76
Σχήμα 36: RDSSampler-Αναζήτηση	78
Σχήμα 37: RDSSampler-Δειγματοληψία	79
Σχήμα 38: Κανονικοποιημένες, ως προς την βέλτιστη πολιτική αντικατάστασης, αστοχίες των προγραμμάτων για διαχείριση με LRU	81
Σχήμα 39: Δομή και οργάνωση του μηχανισμού συλλογής και πρόβλεψης, στην υλοποίηση χαμηλού κόστους	89
Σχήμα 40: Κανονικοποιημένος, ως προς LRU, γεωμετρικός μέσος του χρόνου εκτέλεσης και των αστοχιών για όλους τους μηχανισμούς διαχείρισης	94
Σχήμα 41: Πειραματικά αποτελέσματα για τα προγράμματα χαμηλού MPKI	95
Σχήμα 42: Πειραματικά αποτελέσματα για τα προγράμματα χαμηλού MPKI	96
Σχήμα 43: Πειραματικά αποτελέσματα για τα προγράμματα υψηλού MPKI και γεωμετρικοί μέσοι των αποτελεσμάτων ανά κρυφή μνήμη	97
Σχήμα 44: Κατανομή του MLP-cost του galgel όταν χρησιμοποιεί μία κρυφή μνήμη των 256KB, την οποία διαχειρίζομαστε είτε με LRU είτε με IbRDP πολιτικές αντικατάστασης	100
Σχήμα 45: IPC και αστοχίες, κανονικοποιημένα ως προς LRU, για διαχείριση με DIP, IbRDP και IbRDP με Selective Caching	102
Σχήμα 46: Κατάταξη όλων των συμμετοχών στο μονονηματικό σκέλος του πρωταθλήματος [42]	103
Σχήμα 47: Συγκεντρωτικά αποτελέσματα του μονονηματικού σκέλους του πρωταθλήματος για τους μηχανισμούς IbRDP (petoumenos) και DSB (gao)	105
Σχήμα 48: Αναλυτικά αποτελέσματα του μονονηματικού σκέλους του πρωταθλήματος για τους μηχανισμούς IbRDP (petoumenos) και DSB (gao)	105
Σχήμα 49: Κατάταξη όλων των συμμετοχών στο πολυνηματικό σκέλος του πρωταθλήματος [42]	106
Σχήμα 50: Συγκεντρωτικά αποτελέσματα του πολυνηματικού σκέλους του πρωταθλήματος για τους μηχανισμούς IbRDP (petoumenos) και DSB (gao)	106
Σχήμα 51: Αναλυτικά αποτελέσματα του πολυνηματικού σκέλους του πρωταθλήματος για τους μηχανισμούς IbRDP (petoumenos) και DSB (gao)	108
Σχήμα 52: Δομή μίας κατατετμημένης Ουράς Εντολών. Το κάθε τμήμα χωρίζεται σε CAM και SRAM κομμάτι. Το CAM κομμάτι υλοποιεί την λογική αφύπνισης, ενώ το SRAM κομμάτι αποθηκεύει την πληροφορία που χρειάζεται για την εκτέλεση της εντολής [49]	117

Σχήμα 53: Κατανομή των αποστάσεων (σε εντολές) μεταξύ παράλληλων αστοχιών στο τελευταίο επίπεδο κρυφής μνήμης, και αύξηση του χρόνου εκτέλεσης όταν το μέγεθος της Ουράς Εντολών μειώνεται στο μέγεθος που δείχνει ο x-άξονας, για το art και το gcc	119
Σχήμα 54: Μέγιστες αποστάσεις μεταξύ των αστοχιών του twolf στην διάρκεια ενός παραθύρου 20K εντολών	121
Σχήμα 55: MLP Distance Buffer για παράδειγμα ουράς εντολών με 4 εγγραφές ανά τμήμα	122
Σχήμα 56: Αναγνώριση Superpath	126
Σχήμα 57: Predictor για πρόβλεψη των MLP-distances	127
Σχήμα 58: Γεωμετρικοί μέσοι του EDP, του χρόνου εκτέλεσης και της κατανάλωσης ενέργειας για τον ILP-feedback και τον ILP/MLP-feedback μηχανισμό, για διάφορες τιμές κατωφλίου	129
Σχήμα 59: Αναλυτικά αποτελέσματα για το EDP όλων των προγραμμάτων, για διαχείριση με ILP-feedback ή ILP/MLP-feedback, με τους δύο μηχανισμούς ρυθμισμένους με την βέλτιστη τιμή κατωφλίου τους	131
Σχήμα 60: Αναλυτικά αποτελέσματα για τον χρόνο εκτέλεσης όλων των προγραμμάτων, για διαχείριση με ILP-feedback ή ILP/MLP-feedback, με τους δύο μηχανισμούς ρυθμισμένους με την βέλτιστη τιμή κατωφλίου τους	132
Σχήμα 61: Αναλυτικά αποτελέσματα για την κατανάλωση ενέργειας όλων των προγραμμάτων, για διαχείριση με ILP-feedback ή ILP/MLP-feedback, με τους δύο μηχανισμούς ρυθμισμένους με την βέλτιστη τιμή κατωφλίου τους	132
Σχήμα 62: Σωστές Προβλέψεις και λάθος, αλλά Συντηρητικές, Προβλέψεις του Predictor	133

Κατάλογος Πινάκων

Πίνακας 1:	Παράμετροι των υλοποιήσεων των 5 πολιτικών αντικατάστασης	93
Πίνακας 2:	Αστοχίες ανά 1K εντολές (MPKI) στην 512KB κρυφή μνήμη για τα προγράμματα της σουίτας SPEC2000	95
Πίνακας 3:	Παράμετροι του εξομοιωμένου υπολογιστικού συστήματος	128
Πίνακας 4:	Μετροπρογράμματα Κινητής Υποδιαστολής της σουίτας SPEC CPU2000	164
Πίνακας 5:	Μετροπρογράμματα Ακεραίων Αριθμών της σουίτας SPEC CPU2000	164
Πίνακας 6:	Μετροπρογράμματα Κινητής Υποδιαστολής της σουίτας SPEC CPU2006	165
Πίνακας 7:	Μετροπρογράμματα Ακεραίων Αριθμών της σουίτας SPEC CPU2006	166

Περίληψη

Στην παρούσα διατριβή προτείνονται μέθοδοι διαχείρισης των κοινόχρηστων πόρων σε υπολογιστικά συστήματα όπου πολλαπλοί επεξεργαστές μοιράζονται το ίδιο ολοκληρωμένο. Λόγω της συνεχούς συρρίκνωσης των τρανζίστορς του ολοκληρωμένου, και της αυξανόμενης μη-αποδοτικότητας των συμβατικών επεξεργαστών με εκτέλεση εκτός σειράς, και η βιομηχανία επεξεργαστών, αλλά και η ερευνητική κοινότητα, στράφηκαν προς συστήματα με πολλαπλούς επεξεργαστές σε ένα ολοκληρωμένο (CMP). Χρησιμοποιώντας πολλαπλούς απλούς πυρήνες αντί ενός σύνθετου, τα CMP συστήματα επιτυγχάνουν αυξημένα επίπεδα απόδοσης, πιο αποδοτική χρήση της επιφάνειας πυριτίου και πιο αποδοτική εκμετάλλευση της καταναλισκόμενης ισχύος. Αποτέλεσμα αυτών των πλεονεκτημάτων είναι ότι τα CMP συστήματα έχουν καθιερωθεί πια σε ολόκληρο το εύρος των υπολογιστικών συστημάτων.

Κεντρικό πρόβλημα στις CMP αρχιτεκτονικές είναι η διαχείριση των κοινόχρηστων πόρων. Ενώ μέχρι πρόσφατα ο σχεδιασμός ενός υπολογιστικού συστήματος στόχευε στην ικανοποίηση των απαιτήσεων μόνο μίας εφαρμογής ανά χρονική περίοδο, τώρα πια απαιτείται και η εξισορρόπηση των απαιτήσεων διαφορετικών εφαρμογών που ανταγωνίζονται για την κατοχή των ίδιων πόρων. Η επιλογή του κατάλληλου μηχανισμού διαχείρισης που θα αντιμετωπίσει τις αρνητικές συνέπειες του διαμοιρασμού είναι ένα δύσκολο πρόβλημα, που απαιτεί ισχυρά πειραματικά και θεωρητικά εργαλεία για να επιλυθεί. Σε πολλές περιπτώσεις, όμως, αυτό δεν αρκεί από μόνο του. Ο πολλαπλασιασμός των επεξεργαστών που μοιράζονται τον ίδιο πόρο, πολλαπλασιάζει και το φορτίο που πρέπει να εξυπηρετήσουν οι κοινόχρηστες δομές του ολοκληρωμένου. Ακόμη και αν επιτευχθεί κάποιος ιδανικός διαμοιρασμός του πόρου, αν δεν βελτιστοποιηθεί ο τρόπος με τον οποίο χρησιμοποιούν οι επεξεργαστές τον κοινόχρηστο πόρο, δεν θα καταφέρει να εξυπηρετήσει ικανοποιητικά το αυξημένο φορτίο.

Για να αντιμετωπιστούν τα προβλήματα που πηγάζουν από τον διαμοιρασμό των κοινόχρηστων πόρων, στην παρούσα εργασία προτείνονται τρεις εναλλακτικοί μηχανισμοί διαχείρισης. Η κύρια έμφαση δίνεται στο υποσύστημα μνήμης και στην αλληλεπίδρασή του με τον επεξεργαστή, αφενός γιατί σημαντικά κομμάτια του είναι κοινόχρηστα στις CMP

αρχιτεκτονικές και αφετέρου γιατί αποτελεί ένα από τα κρισιμότερα υποσυστήματα όλων των σύγχρονων επεξεργαστών. Λόγω του διαρκώς αυξανόμενου χάσματος μεταξύ της ταχύτητας του επεξεργαστή και της ταχύτητας της κύριας μνήμης, ο επεξεργαστής μένει την περισσότερη ώρα αδρανοποιημένος, περιμένοντας να ικανοποιηθεί κάποια αίτηση δεδομένων από την κύρια μνήμη. Οι κρυφές μνήμες αποτελούν το σημαντικότερο μέσο με το οποίο αντιμετωπίζεται αυτό το χάσμα, παρέχοντας ένα μικρό υποσύνολο των δεδομένων του εκτελούμενου προγράμματος με πολύ μικρότερους χρόνους απόκρισης από ότι η κύρια μνήμη. Στα περισσότερα CMP συστήματα, όμως, αυτός ο κρίσιμος πόρος είναι κοινόχρηστος, με αποτέλεσμα ο ανταγωνισμός μεταξύ των επεξεργαστών και το αυξημένο φορτίο να θέτουν σε κίνδυνο την ικανότητα των κρυφών μνημών να κρύβουν τον μεγάλο χρόνο απόκρισης της κύριας μνήμης.

Για αυτόν τον λόγο, στην παρούσα διατριβή προτείνονται δύο μεθοδολογίες διαχείρισης των κρυφών μνημών. Η πρώτη εισάγει μία νέα θεωρητική μοντελοποίηση του διαμοιρασμού της κρυφής μνήμης, η οποία μπορεί να χρησιμοποιηθεί παράλληλα με την εκτέλεση των προγραμμάτων που διαμοιράζονται την κρυφή μνήμη. Η μεθοδολογία αξιοποιεί στην συνέχεια αυτήν την μοντελοποίηση, για να ελέγχει τον διαμοιρασμό της κρυφής μνήμης και να επιτύχει δικαιοσύνη στο πως κατανέμεται ο χώρος της κρυφής μνήμης μεταξύ των επεξεργαστών. Η δεύτερη μεθοδολογία παρουσιάζει μία νέα τεχνική για την πρόβλεψη της τοπικότητας των προσπελάσεων της κρυφής μνήμης. Καθώς η τοπικότητα είναι η βασική παράμετρος που καθορίζει την χρησιμότητα των δεδομένων της κρυφής μνήμης, χρησιμοποιώντας αυτήν την τεχνική πρόβλεψης μπορούν να οδηγηθούν μηχανισμοί διαχείρισης που βελτιώνουν την αξιοποίηση του χώρου της κρυφής μνήμης. Στα πλαίσια της μεθοδολογίας παρουσιάζουμε έναν τέτοιο μηχανισμό, ο οποίος στοχεύει στην ελαχιστοποίηση των αστοχιών της κρυφής μνήμης μέσω μίας νέας πολιτικής αντικατάστασης.

Η τελευταία μεθοδολογία που παρουσιάζεται είναι μία μεθοδολογία για την μείωση της κατανάλωσης ενέργειας του επεξεργαστή. Παρά την αυξημένη ενεργειακή αποδοτικότητα των CMP συστημάτων, η κατανάλωση ενέργειας του ολοκληρωμένου παραμένει υψηλή και επομένως απαιτούνται τεχνικές που να την περιορίζουν. Η μεθοδολογία μείωσης της κατανάλωσης ενέργειας επικεντρώνεται στην ουρά εντολών, που είναι μία από τις πιο ενεργειακά απαιτητικές δομές του επεξεργαστή. Στα πλαίσια της μεθοδολογίας, δείχνεται ότι το κλειδί για την μείωση της κατανάλωσης ενέργειας της ουράς εντολών, χωρίς δυσανάλογη μείωση της απόδοσης του επεξεργαστή, βρίσκεται στην αλληλεπίδραση της με

το υποσύστημα μνήμης: όσο η διαχείριση της ουράς εντολών δεν μειώνει την αξιοποίηση του υποσυστήματος μνήμης, οι συνέπειες της διαχείρισης στην ταχύτητα εκτέλεσης του επεξεργαστή είναι αμελητέες. Με βάση αυτό το συμπέρασμα, παρουσιάζουμε έναν νέο μηχανισμό δυναμικής διαχείρισης του μεγέθους της ουράς εντολών, ο οποίος καταφέρνει και συνδυάζει επιθετική μείωση της κατανάλωσης ενέργειας του επεξεργαστή με διατήρηση της υψηλής απόδοσής του.

Abstract

This dissertation proposes methodologies for the management of shared resources in chip multi-processors (CMP). Due to the constant shrinking of the chip's transistors and the increasing inefficiency of conventional processors with out-of-order execution, both the industry and the research community moved their attention towards CMPs. Using multiple simple processing cores instead of one complex core, CMPs provide increased overall performance, more efficient use of the silicon area and increased power-efficiency. Because of all these advantages, CMPs are now the established paradigm for virtually every category of computing systems.

A central problem of CMP architectures is how to manage the shared resources. Until recently, the design of a computing system had to satisfy the computational and storage needs of a single program during each time period. Now instead, the designer has to balance the, perhaps conflicting, needs of multiple programs competing for the same resources. Choosing the appropriate management mechanism, which will deal with the negative effects of resource sharing, is a difficult problem to solve and requires powerful experimental and theoretical tools. But, in many cases, even this is not enough. Multiplying the number of processors sharing the same resource, multiplies also the load that this resource has to service. Even if we could invent a perfect way to manage sharing, without optimizing the way that each processor uses the shared resource, the resource could not deal efficiently with the increased load.

In order to handle the negative effects of resource sharing, this dissertation proposes three management mechanisms. The main focus of these mechanisms is the memory subsystem and its interaction with the processor, because, first, the memory subsystem is one of the most critical subsystems of modern processors and, second, significant parts of it are shared in CMP architectures. Because of the increasing gap between processor and main memory speeds, the processor spends most of the time waiting for data from the main memory. Caching is the main way that we use to hide this gap, by offering a small subset of the program's data at much shorter access times than the main memory. Unfortunately, in most CMP architectures at least one cache level is shared, so the competition among the proces-

sors and the increased load put at risk the ability of the chip's caches to hide the long access latency of the main memory.

For this reason, we propose two methodologies for managing cache memories. The first one introduces a novel theoretical model of the sharing of the shared cache, which can be used at run-time. Furthermore, our methodology uses the model to control sharing and to achieve a sense of justice in the way the cache is shared among the processors. Our second methodology presents a new technique for predicting the locality of cache accesses. Since locality determines, almost entirely, the usefulness of cache data, our technique can be used to drive any management mechanism which strives to improve the efficiency of the cache. As part of our methodology, we present such a mechanism, a new cache replacement policy which tries to minimize cache misses by near-optimal replacement decisions.

The last methodology presented in this dissertation, targets the energy consumption of the processor. Despite the increased power-efficiency of CMPs, energy consumption and the resulting heat are still high and have to be reduced. Additionally, since all the processors of a CMP use the same power supply and heat sinks, energy consumption can be treated as a shared resource. To that end, our methodology shows that the key to reducing the power consumption of the Issue Queue, without disproportional performance degradation, lies at the interaction of the Issue Queue with the memory subsystem: as long as the management of the Issue Queue doesn't reduce the utilization of the memory subsystem, the effects of the management on the processor's performance will be minimal. Based on this conclusion, we introduce a new mechanism for dynamically resizing the Issue Queue, which achieves aggressive downsizing and energy savings with almost no performance degradation.

Κεφάλαιο 1

Εισαγωγή, Συνεισφορές και Οργάνωση

1.1 Πολυεπεξεργαστικά συστήματα ενός ολοκληρωμένου

Η συνεχής συρρίκνωση των τρανζίστορς του ολοκληρωμένου, ακολουθώντας σταθερά τον νόμο του Moore, και η αυξανόμενη μη-αποδοτικότητα των συμβατικών επεξεργαστών με εκτέλεση εκτός σειράς, οδήγησαν τα τελευταία χρόνια τους ερευνητές και την βιομηχανία σε συστήματα με πολλαπλούς επεξεργαστές σε ένα ολοκληρωμένο (Chip Multi-Processors - CMP). Χρησιμοποιώντας πολλαπλούς απλούς πυρήνες αντί ενός σύνθετου, τα CMP συστήματα επιτυγχάνουν αυξημένα επίπεδα απόδοσης, καλύτερη αξιοποίηση της επιφάνειας πυριτίου και πιο αποδοτική εκμετάλλευση της καταναλισκόμενης ισχύος. Αποτέλεσμα αυτής της μεταστροφής του σχεδιαστικού παραδείγματος είναι ότι σήμερα τα CMP συστήματα έχουν καθιερωθεί σε ολόκληρο το εύρος των υπολογιστικών εφαρμογών, από εξυπηρετητές υψηλής απόδοσης [53][54] μέχρι ενσωματωμένες συσκευές [5][34].

Η είσοδος στην εποχή των πολυεπεξεργαστικών συστημάτων ενός ολοκληρωμένου, όμως, συνοδεύεται από αλλάγες πολύ βαθύτερες από την απλή αύξηση της αποδοτικότητας. Για παράδειγμα, η αξιοποίηση των εσωτερικών διαύλων και αποθηκευτικών δομών του ολοκληρωμένου για την επικοινωνία μεταξύ των επεξεργαστών, μεταφράζεται σε κατά τάξεις μεγέθους μείωση των καθυστερήσεων που σχετίζονται με την επικοινωνία των νημάτων των πολυνηματικών εφαρμογών, αυξάνοντας δραματικά τις

κατηγορίες εφαρμογών που μπορούν να παραλληλοποιηθούν αποτελεσματικά. Το αντίτιμο, όμως, που συνοδεύει αυτές τις νέες δυνατότητες, είναι η εμφάνιση νέων προβλημάτων.

Ένα από αυτά τα προβλήματα είναι και η διαχείριση των κοινόχρηστων πόρων σε αρχιτεκτονικές πολλαπλών επεξεργαστών. Ενώ μέχρι πρόσφατα ο σχεδιασμός ενός υπολογιστικού συστήματος στόχευε στην ικανοποίηση των απαιτήσεων μόνο μίας εφαρμογής ανά χρονική περίοδο, τώρα πια απαιτείται αφενός η διαχείριση του αυξημένου φορτίου που εξυπηρετούν οι κοινόχρηστες δομές και αφετέρου η εξισορρόπηση των απαιτήσεων διαφορετικών εφαρμογών που διαγωνίζονται για την κατοχή των ίδιων πόρων. Για αρκετούς από τους πόρους, όπως για παράδειγμα οι κρυφές μνήμες πρώτου επιπέδου, η πιο απλή και αποδοτική λύση είναι ο πολλαπλασιασμός των διαθέσιμων πόρων, ώστε κάθε επεξεργαστής να κατέχει ένα αντίγραφο της ίδιας δομής. Για άλλες περιπτώσεις όμως, όπως οι κρυφές μνήμες των κατωτέρων επιπέδων της ιεραρχίας μνήμης, οι εξωτερικοί δίαυλοι ή το υποσύστημα ισχύος, μία τέτοια λύση θα ήταν αδύνατη ή μη αποδοτική.

Σε αυτές τις περιπτώσεις, ο αρχιτέκτονας του επεξεργαστικού συστήματος πρέπει να υλοποιήσει μηχανισμούς και τεχνικές που να στοχεύουν στον δίκαιο διαμοιρασμό των πόρων και στην κατά το εφικτό απομόνωση των εφαρμογών που διαμοιράζονται τους πόρους, χωρίς όμως σημαντική υποβάθμιση της απόδοσης των επιμέρους επεξεργαστών. Όμως ακόμη και σε ότι αφορά αυτούς τους στόχους ο σχεδιαστής θα πρέπει να πάρει κρίσιμες αποφάσεις: ο μηχανισμός θα προσπαθεί να ελαχιστοποιήσει την συνολική υποβάθμιση της απόδοσης του κοινόχρηστου πόρου ακόμη και σε βάρος κάποιων εκ των πυρήνων, θα προσπαθεί να μοιράζει ισόποσα το κοινό πόρο ακόμη και αν κάποιοι πυρήνες δεν χρειάζονται το μερίδιό τους ή θα προσπαθήσει να βρει κάποιο σημείο ισορροπίας ανάμεσα σε αυτούς τους δύο αντικρουόμενους στόχους; Η επιλογή του κατάλληλου μηχανισμού είναι πρόβλημα χωρίς απόλυτη λύση. Ο σχεδιαστής θα πρέπει να έχει βαθιά κατανόηση των διαθέσιμων επιλογών και των συνεπειών τους, καθώς και τα πειραματικά και θεωρητικά εργαλεία που θα τον οδηγήσουν σε μία ικανοποιητική αντιμετώπιση του προβλήματος του διαμοιρασμού.

Όμως, σε πολλές περιπτώσεις αυτό δεν αρκεί από μόνο του. Ο πολλαπλασιασμός των επεξεργαστών που μοιράζονται τον ίδιο πόρο, πολλαπλασιάζει και το φορτίο που πρέπει να εξυπηρετήσουν οι κοινόχρηστες δομές του ολοκληρωμένου. Ακόμη και αν επιτευχθεί κάποιος ιδανικός διαμοιρασμός του πόρου, είναι πολύ πιθανό ότι το μερίδιο του

κοινόχρηστου πόρου του κάθε επεξεργαστή δεν θα είναι αρκετό για να λειτουργήσει ο επεξεργαστής στο μέγιστο των δυνατοτήτων του. Για να αντιμετωπιστεί αυτό το πρόβλημα, νέες αποδοτικές τεχνικές διαχείρισης είναι απαραίτητες, για όλες τις δομές που άμεσα ή έμμεσα συμμετέχουν στον διαμοιρασμό

1.2 Συνεισφορές

1.2.1 Μελέτη και διαχείριση του διαμοιρασμού κοινόχρηστων κρυφών μνημών

Η κύρια έμφαση της διατριβής δίνεται στο υποσύστημα μνήμης μιας και αυτό αποτελεί ένα από τα πιο κρίσιμα υποσυστήματα των σύγχρονων επεξεργαστών. Εξαιτίας της δυσανάλογης αύξησης της ταχύτητας λειτουργίας των επεξεργαστών σε σχέση με την κύρια μνήμη, κάθε προσπέλαση μνήμης στοιχίζει σήμερα 100 έως 500 κύκλους ρολογιού. Χωρίς αντιμετώπιση αυτού του προβλήματος, ο επεξεργαστής θα περίμενε αδρανοποιημένος για δεδομένα από την μνήμη στο μεγαλύτερο κομμάτι της λειτουργίας του. Οι κρυφές μνήμες που μεσολαβούν ανάμεσα στον επεξεργαστή και την μνήμη κρύβουν σε πολύ σημαντικό βαθμό το πρόβλημα, επιτρέποντας στον επεξεργαστή να λειτουργεί αρκετά κοντά στον μέγιστο βαθμό απόδοσης του. Ακριβώς επειδή το πρόβλημα που καλύπτουν οι κρυφές μνήμες μπορεί να έχει τόσο καταστροφική επίδραση στον επεξεργαστή, η σωστή και αποδοτική διαχείρισή τους είναι απαραίτητη.

Το πρώτο θέμα με το οποίο ασχολείται η διατριβή είναι ο διαμοιρασμός των κοινόχρηστων κρυφών μνημών. Ενώ είναι συνήθης η διαχείριση τους με αλγορίθμους και μηχανισμούς πανομοιότυπους με αυτούς που χρησιμοποιούνται σε ιδιωτικές κρυφές μνήμες, αρκετές εργασίες έχουν δείξει ότι μηχανισμοί που δεν ενσωματώνουν την έννοια του διαμοιρασμού στην λογική διαχείρισης αποτυγχάνουν να μεγιστοποιήσουν το ποσοστό ευστοχιών της κρυφής μνήμης ή οδηγούνται σε διαμοιρασμό της κρυφής μνήμης που βλάπτει υπερβολικά πολύ την απόδοση κάποιων εκ των εφαρμογών που μοιράζονται την κρυφή μνήμη. Η αντιμετώπιση αυτών των φαινομένων απαιτεί μηχανισμούς που να συλλέγουν πληροφορία για το πως διαμοιράζεται η κρυφή μνήμη, να προβλέπουν πως αλλάζει η απόδοση των παράλληλα εκτελούμενων εφαρμογών όταν αλλάζει ο διαμοιρασμός, να διαλέγουν στόχους για τον χώρο που θα αποδίδεται σε κάθε επεξεργαστή και να επιβάλλουν αυτήν την απόφαση στην κρυφή μνήμη.

Η συνεισφορά της διατριβής σε αυτόν τον τομέα είναι διπλή. Πρώτα παρουσιάζεται ένα νέο μοντέλο που περιγράφει θεωρητικά τον διαμοιρασμό μίας κοινόχρηστης κρυφής μνήμης, σε ότι αφορά τον χώρο που αποδίδεται σε κάθε επεξεργαστή και τις συνέπειες του διαμοιρασμού στον βαθμό αστοχίας κάθε εφαρμογής. Το μοντέλο είναι σχεδιασμένο με τέτοιον τρόπο ώστε και η συλλογή της πληροφορίας εισόδου αλλά και οι εξισώσεις που περιέχει να επιτρέπουν την χρήση του ταυτόχρονα με την εκτέλεση των εφαρμογών που μοιράζονται την κρυφή μνήμη. Στην συνέχεια παρουσιάζεται ένας καινοτόμος μηχανισμός διαχείρισης του διαμοιρασμού, ο οποίος είναι στενά συνδεδεμένος με το προαναφερθέν θεωρητικό μοντέλο: τα χαρακτηριστικά του του επιτρέπουν και να περιγραφεί αλλά και να οδηγηθεί από το μοντέλο. Με την χρήση της θεωρητικής μοντελοποίησης και του μηχανισμού διαχείρισης, η διατριβή δείχνει ότι είναι εφικτή η επίτευξη δικαιοσύνης και υψηλής απόδοσης, αλλά και η παροχή εγγυήσεων ποιότητας υπηρεσίας σε ότι αφορά την χρήση της κοινόχρηστης κρυφής μνήμης από τους επεξεργαστές του ολοκληρωμένου.

1.2.2 Μηχανισμός αντικατάστασης κρυφών μνημών

Πάρα την δυνατότητα του προηγούμενου μηχανισμού να διαμοιράσει πιο δίκαια και πιο αποδοτικά την κοινόχρηστη κρυφή μνήμη, αυτό από μόνο του δεν αρκεί. Για επιτευχθούν οφέλη, θα πρέπει κάποιες από τις εφαρμογές που μοιράζονται την κρυφή μνήμη να μπορούν να αποδεσμεύσουν τμήμα του χώρου που καταλαμβάνουν χωρίς σημαντική μείωση του ποσοστού ευστοχίας τους και κάποιες άλλες να μπορούν να αξιοποιήσουν τον αποδεσμευμένο χώρο ώστε να βελτιώσουν την απόδοσή τους. Όταν αυτές οι δύο συνθήκες δεν ισχύουν, τότε ο μόνος τρόπος να βελτιωθεί η αξιοποίηση της κρυφής μνήμης είναι η αλλαγή του αλγορίθμου αντικατάστασης που χρησιμοποιεί ο μηχανισμός διαχείρισης.

Η ανεπάρκεια των random και LRU αλγορίθμων αντικατάστασης στην διαχείριση κρυφών μνημών στα κατώτερα επίπεδα της ιεραρχίας μνήμης έχει αποδειχθεί σε αρκετές ερευνητικές εργασίες. Βασικός λόγος για αυτό είναι ότι ενώ οι κλασικοί μηχανισμοί αντικατάστασης υποθέτουν (άμεσα ή έμμεσα) ότι οι πιο πρόσφατα χρησιμοποιημένες γραμμές είναι και αυτές που θα χρησιμοποιηθούν πιο σύντομα, στα κατώτερα επίπεδα της ιεραρχίας μνήμης αυτό συνήθως δεν ισχύει. Η τοπικότητα των προσπελάσεων του προγράμματος συλλαμβάνεται και φιλτράρεται από το πρώτο επίπεδο κρυφών μνημών — οι προσπελάσεις που φτάνουν στα κατώτερα επίπεδα είναι ακριβώς αυτές που δεν παρουσιάζουν σημαντική τοπικότητα και επαναχρησιμοποίηση.

Βασισμένη σε αυτές τις παρατηρήσεις, η διατριβή παρουσιάζει μία νέα πολιτική αντικατάστασης, την IbRDP. Στόχος της είναι η ελαχιστοποίηση του βαθμού αστοχίας των κρυφών μνημών, ορθογώνια προς τους στόχους που έχει, ίσως, θέσει κάποιος μηχανισμός ελέγχου του διαμοιρασμού. Θεμελιώδης υπόθεση της πολιτικής αντικατάστασης είναι ότι υπάρχει υψηλή συσχέτιση ανάμεσα στην εντολή που προκαλεί μία προσπέλαση προς μία γραμμή της κρυφής μνήμης και την απόσταση στο μέλλον μέχρι την επαναχρησιμοποίηση αυτής της γραμμής. Με βάση αυτήν την υπόθεση σχεδιάστηκε ένα σύστημα παρακολούθησης και πρόβλεψης των αποστάσεων επαναχρησιμοποίησης των δεδομένων της κρυφής μνήμης, το οποίο παρέχει έγκυρες και ακριβείς προβλέψεις για τις περισσότερες προσπελάσεις. Η πολιτική αντικατάστασης χρησιμοποιεί αυτήν την πληροφορία για να προσπαθήσει να προβλέψει την γραμμή της κρυφής μνήμης που θα χρησιμοποιηθεί πιο μακριά στο μέλλον και να την αντικαταστήσει, προσομοιώνοντας έτσι τον θεωρητικά βέλτιστο αλγόριθμο αντικατάστασης του Bellady. Τα χαρακτηριστικά και ο τρόπος λειτουργίας του μηχανισμού αυτού παρουσιάζονται και αναλύονται μέσα από μία σειρά από πειράματα και μέσω σύγκρισής του με άλλους σημαντικούς μηχανισμούς αντικατάστασης που προτάθηκαν τα τελευταία χρόνια. Από αυτήν την πειραματική διαδικασία προκύπτει ότι η IbRDP πολιτική αντικατάστασης διαχειρίζεται πολύ πιο αποτελεσματικά την τοπικότητα των προσπελάσεων από ότι οι υπόλοιποι αλγόριθμοι αντικατάστασης.

1.2.3 Διαχείριση της ουράς εντολών

Στα πλαίσια της υλοποίησης της IbRDP πολιτικής, παρατηρήθηκε ότι η σχέση μεταξύ των αστοχιών της κρυφής μνήμης και της υποβάθμισης της απόδοσης του επεξεργαστή που αυτές προκαλούν είναι πιο σύνθετη από ότι είχε αρχικά θεωρηθεί. Ο βασικός υπεύθυνος για αυτό είναι ο παραλληλισμός μεταξύ των προσβάσεων κύριας μνήμης (Memory Level Parallelism — MLP).

Πιο συγκεκριμένα, στους σύγχρονους επεξεργαστές με εκτέλεση εκτός σειράς, στην διάρκεια της αναμονής για την μεταφορά δεδομένων από την ιεραρχία μνήμης, αντί ο επεξεργαστής να αδρανεί, χρησιμοποιεί τις διαθέσιμες αριθμητικές και λογικές μονάδες για να εκτελέσει όποια εντολή είναι έτοιμη προς εκτέλεση, ακόμη και αν αυτό σημαίνει ότι η σειρά με την οποία θα εκτελεστούν οι εντολές διαφέρει από την σειρά τους στον κώδικα. Ενώ αυτή η τεχνική καταφέρνει συνήθως να κρύψει το κόστος της προσπέλασης μίας κρυφής μνήμης, δεν ισχύει το ίδιο αν χρειαστεί να διαβαστεί το δεδομένο από την κύρια

μνήμη. Σε αυτήν την περίπτωση, σχετικά σύντομα μετά την έναρξη της προσπέλασης θα εκτελεστούν όλες οι έτοιμες εντολές και έπειτα ο επεξεργαστής θα αδρανήσει.

Ο τρόπος με τον οποίο περιπλέκει την ανάλυση το MLP είναι ο εξής: αν μετά την προσπέλαση που αστοχεί στο τελευταίο επίπεδο κρυφής μνήμης, αλλά πριν εξαντληθούν οι έτοιμες εντολές, εκτελεστεί μία δεύτερη εντολή μεταφοράς δεδομένων που και αυτή αστοχεί στις κρυφές μνήμες, τότε ο συνολικός χρόνος που θα μείνει αδρανής ο επεξεργαστής είναι σχεδόν ίσος με τον χρόνο που θα έμενε αδρανής αν υπήρχε μόνο μία προσπέλαση κύριας μνήμης. Διαφορετικά διατυπωμένο, το κόστος μίας αστοχίας στο τελευταίο επίπεδο κρυφής μνήμης μοιράζεται μεταξύ των αστοχιών που διαβάζουν δεδομένα ταυτόχρονα από την κύρια μνήμη.

Με δεδομένη την εξαιρετικά αρνητική επίδραση των προσπελάσεων κύριας μνήμης στην απόδοση του επεξεργαστή, η δυνατότητα να εξαλείφεται το κόστος μίας αστοχίας παραλληλίζοντας την με μία άλλη, κάνει το MLP έναν πολύ κρίσιμο στόχο της σχεδίασης του επεξεργαστή. Παρόλα αυτά, μέχρι πρόσφατα λίγες εργασίες είχαν εξετάσει την σύνδεση μεταξύ του MLP και της διαχείρισης των δομών που συμμετέχουν στο παραλληλισμό. Μία τέτοια δομή είναι και η ουρά εντολών (Instruction Queue — IQ), η οποία περιέχει τις εντολές που περιμένουν να εκτελεστούν λειτουργικές μονάδες του επεξεργαστή. Η σύνδεση της IQ με το MLP είναι ότι για να μπορέσουν δύο προσπελάσεις να εκτελεστούν παράλληλα, θα πρέπει να χωράνε ταυτόχρονα στην IQ. Πρακτικά, το μέγεθος της IQ είναι, μαζί με τον αριθμό των miss handling registers, η βασική παράμετρος που περιορίζει το MLP που μπορούμε να εξαγάγουμε από ένα πρόγραμμα.

Παράλληλα όμως, η δυναμική μείωση του μέγεθος της IQ είναι ανάμεσα στους στόχους πολλών μεθοδολογιών που προσπαθούν να μειώσουν την δυναμική κατανάλωση ενέργειας του επεξεργαστή. Ο λόγος για αυτό είναι ότι, αφενός η ουρά εντολών λόγω του μεγέθους και της δομής της είναι από τα πιο ενεργοβόρα υποσυστήματα του επεξεργαστικού πυρήνα, και αφετέρου στο μεγαλύτερο μέρος της εκτέλεσης ενός προγράμματος η ουρά εντολών υποχρησιμοποιείται, καθώς είναι σχεδιασμένη ώστε να εξυπηρετεί τις σποραδικές φάσεις του προγράμματος, όπου υπάρχει υψηλός παραλληλισμός ανάμεσα στις εντολές. Με δεδομένο το ότι ο περιορισμός της κατανάλωσης ενέργειας και ισχύος (και ο συνεπαγόμενος περιορισμός της θερμότητας που εκλύει το ολοκληρωμένο) είναι ανάμεσα στους βασικούς στόχους της σχεδίασης ολοκληρωμένων, γίνεται προφανές ότι η δυναμική

μείωση του μεγέθους της IQ είναι απαραίτητη στα σύγχρονα πολυεπεξεργαστικά συστήματα.

Οι περισσότερες από τις ερευνητικές εργασίες στην περιοχή αυτή χρησιμοποιούν την συνεισφορά σε IPC του νεότερου κομματιού της IQ σαν ένδειξη για το αν πρέπει να απενεργοποιηθεί ή όχι: αν λίγες εντολές εκτελούνται από αυτό το κομμάτι, τότε πιθανή απενεργοποίησή του θα έχει ελάχιστες συνέπειες στην ταχύτητα εκτέλεσης του προγράμματος. Το κοινό τους μειονέκτημα είναι η μη κατανόηση του ότι ακόμη και αν ελάχιστες εντολές εκτελούνται από το νεότερο κομμάτι, αν μία από αυτές προκαλούσε αρχικά παράλληλη προσπέλαση κύριας μνήμης, τότε πιθανή μείωση του μεγέθους της IQ θα μειώσει το MLP, με δυσανάλογα μεγάλες συνέπειες στην ταχύτητα εκτέλεσης του επεξεργαστή.

Στο πρόβλημα του σχεδιασμού μίας ουράς εντολών, που θα επιτυγχάνει ταυτόχρονα και χαμηλή κατανάλωση ενέργειας και υψηλό MLP, επικεντρώνεται η τρίτη μεθοδολογία αυτής της διατριβής, το MLP-aware IQ Resizing. Η μεθοδολογία αυτή αναδεικνύει την διατήρηση του MLP σαν τον βασικό περιοριστικό παράγοντα στις προσπάθειες για μείωση του μεγέθους της IQ και προτείνει έναν μηχανισμό που συνδυάζει πληροφορία και για το ILP και το MLP του προγράμματος για να πάρει αποφάσεις για την διαχείριση της IQ. Η πειραματική μελέτη του μηχανισμού και η σύγκρισή του με παλιότερες τεχνικές που αγνοούν το MLP, δείχνουν ότι το MLP-aware IQ Resizing επιτυγχάνει επιθετική μείωση του μεγέθους της ουράς εντολών όταν δεν υπάρχει MLP και συντηρητική μείωση όταν υπάρχει. Με αυτόν τον τρόπο ο MLP-aware μηχανισμός παράγει μεγαλύτερα οφέλη σε εξοικονόμηση ενέργειας σε σχέση με εναλλακτικούς μηχανισμούς, και ταυτόχρονα μικρότερη υποβάθμιση της απόδοσης του επεξεργαστή.

1.3 Οργάνωση της Διατριβής

Το κύριο μέρος της διατριβής οργανώνεται ως ακολούθως. Στο Κεφάλαιο 2 δίνονται οι βασικές έννοιες της αρχιτεκτονικής υπολογιστών, της ιεραρχίας μνήμης και των κρυφών μνημών στα σύγχρονα υπολογιστικά συστήματα. Επίσης, γίνεται μια σύντομη περιγραφή προηγούμενων μεθοδολογιών πάνω στις οποίες στηρίχθηκαν τα αποτελέσματα που παρήχθησαν στην συγκεκριμένη διδακτορική διατριβή.

Στο Κεφάλαιο 3 παρουσιάζεται η μεθοδολογία StatShare που μοντελοποιεί τον διαμοιρασμό κοινόχρηστων κρυφών μνημών. Συγκεκριμένα, εξηγείται το κίνητρο για την

μοντελοποίηση των κοινόχρηστων κρυφών μνημών, αναλύεται το μοντέλο και οι εξισώσεις του, περιγράφεται ο τρόπος με τον οποίο μπορεί να χρησιμοποιηθεί κατά την διάρκεια της εκτέλεσης και εξαγάγονται θεωρητικά συμπεράσματα για τον διαμοιρασμό της κρυφής μνήμης. Με βάση την πληροφορία του στατιστικού μοντέλου, αναλύεται στην συνέχεια ένας νέος μηχανισμός διαχείρισης για κοινόχρηστες κρυφές μνήμες. Δείχνεται πως ο μηχανισμός ενσωματώνεται στο μοντέλο StatShare και πως επιτυγχάνει δίκαιο και αποδοτικό διαμοιρασμό. Επίσης περιγράφεται η δυνατότητα του μηχανισμού για την παροχή εγγυήσεων υπηρεσίας σε ότι αφορά τον χώρο που αποδίδεται σε κάθε επεξεργαστή και την αντιμετώπιση κακόβουλων διεργασιών που προσπαθούν να αρνηθούν την χρήση της κοινόχρηστης κρυφής μνήμης στις υπόλοιπες εκτελούμενες διεργασίες.

Το Κεφάλαιο 4 προτείνει μία νέα πολιτική αντικατάστασης για ιδιωτικές ή κοινόχρηστες κρυφές μνήμες που επιτυγχάνει να αυξήσει τον βαθμό ευστοχίας της κρυφής μνήμης και την αποτελεσματική εκμετάλλευση του χώρου που καταλαμβάνει κάθε εφαρμογή. Παρουσιάζεται η κεντρική ιδέα της τεχνικής, η πρόβλεψη των αποστάσεων επαναχρησιμοποίησης των γραμμών της κρυφής μνήμης και δείχνεται το πως μπορεί βάσει αυτής της πληροφορίας να υλοποιηθεί ένας μηχανισμός αντικατάστασης που να μιμείται όσο το δυνατόν καλύτερα τις αποφάσεις του θεωρητικά βέλτιστου μηχανισμού αντικατάστασης. Στην συνέχεια αναλύεται η επιτυχία της τεχνικής σε σύγκριση με τους πιο σημαντικούς μηχανισμούς αντικατάστασης των τελευταίων χρόνων και φωτίζονται τα δυνατά και τα αδύνατα σημεία του κάθε μηχανισμού.

Στο Κεφάλαιο 5 παρουσιάζεται η τελευταία μεθοδολογία της παρούσας διατριβής, μία τεχνική για την δυναμική ρύθμιση του μεγέθους της ουράς εντολών, η οποία λαμβάνει υπόψιν της την κρισιμότητα της διατήρησης του MLP. Αρχικά παρουσιάζονται οι σημαντικότεροι από τους παρόμοιους υπάρχοντες μηχανισμούς και δείχνεται ότι η χρησιμότητά τους περιορίζεται λόγω του ότι αγνοούν τον παραλληλισμό επιπέδου εντολών. Στην συνέχεια παρουσιάζεται ο μηχανισμός που ανιχνεύει και ποσοτικοποιεί την ύπαρξη παράλληλων προσπελάσεων μνήμης και εξηγείται το πως μπορεί να χρησιμοποιηθεί αυτή η πληροφορία για την διαχείριση του μεγέθους της ουράς εντολών.

Τέλος, στο Κεφάλαιο 6 ολοκληρώνεται η διατριβή με την παράθεση γενικών συμπερασμάτων και κατευθύνσεων για μελλοντική έρευνα. Επίσης παρατίθενται οι δημοσιευμένες εργασίες πάνω στα ερευνητικά αποτελέσματα της διατριβής, καθώς και μια

λίστα δημοσιεύσεων άλλων ερευνητικών ομάδων, στις οποίες χρησιμοποιούνται ως αναφορές οι εργασίες αυτές.

Μετά το τέλος του κυρίως μέρους της διατριβής ακολουθεί η βιβλιογραφία, ένα γλωσσάρι τεχνικών όρων και πληροφορίες σχετικά με τις λεπτομέρειες της πειραματικής μεθοδολογίας που ακολουθήθηκε στα πλαίσια της διατριβής.

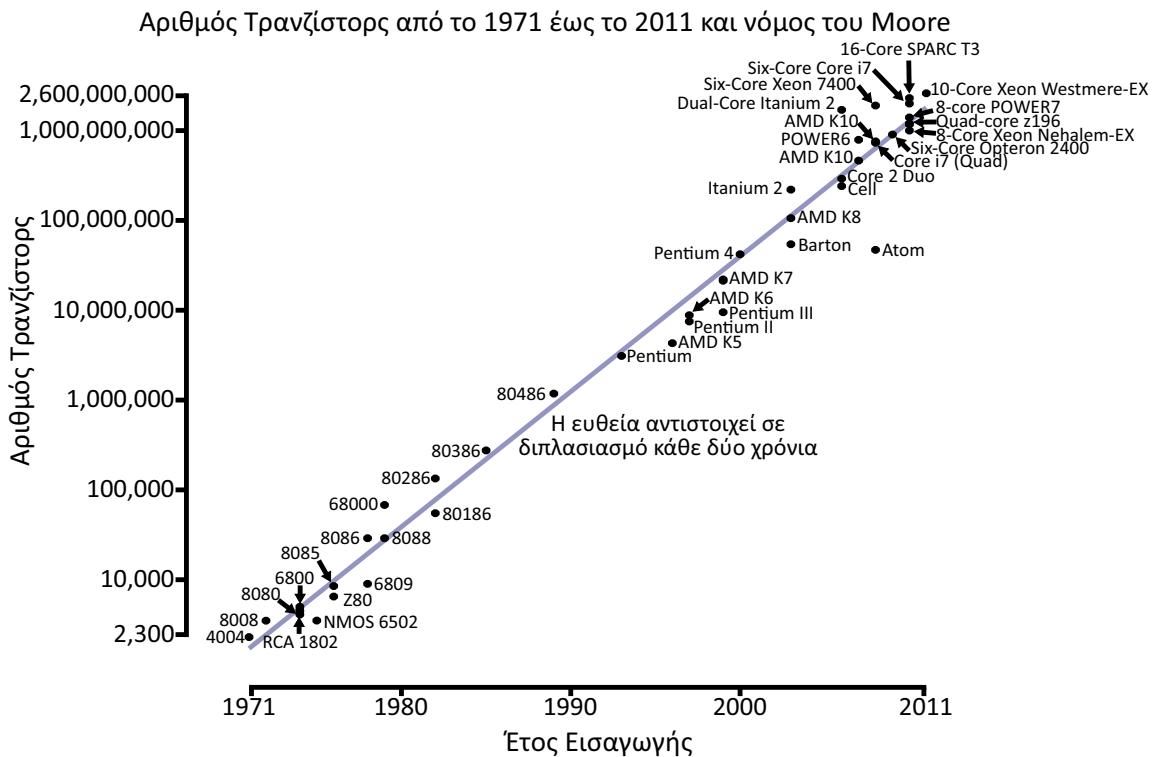
Κεφάλαιο 2

Βασικές Έννοιες Αρχιτεκτονικής Υπολογιστών

2.1 Πολυεπεξεργαστές ενός Ολοκληρωμένου

Ο νόμος του Moore[63][64], ένας συνδυασμός πραγματικής πρόβλεψης και αυτο-εκπληρούμενης προφητείας, έχει καταφέρει να περιγράψει με ακρίβεια την εξέλιξη της τεχνολογίας ολοκλήρωσης τα τελευταία 46 χρόνια (Σχήμα 1). Ο σταθερός ρυθμός αύξησης της τεχνολογίας ολοκλήρωσης που προβλέπει ο νόμος του Moore επέτρεψε στους αρχιτέκτονες υπολογιστών να σχεδιάζουν επεξεργαστικά συστήματα με ολοένα μεγαλύτερη πολυπλοκότητα: περισσότερες εντολές, ενσωμάτωση ελεγκτών και βασικών περιφερειακών μέσα στο ολοκληρωμένο, μεγάλες κρυφές μνήμες, βαθύτερα pipelines, μεγαλύτερος παραλληλισμός στην εκτέλεση των εντολών. Η αυξημένη πολυπλοκότητα του ολοκληρωμένου, σε συνδυασμό με την αύξηση της συχνότητας του ρολογιού του επεξεργαστή, επέτρεψε στα επεξεργαστικά συστήματα να διπλασιάζουν την απόδοσή τους κάθε 18 μήνες για ένα μεγάλο κομμάτι της ιστορίας των μικροεπεξεργαστών. Την τελευταία δεκαετία όμως, αυτή η σταθερή εξέλιξη άρχισε να φτάνει σε τέλμα:

- Όσο μικραίνει η τεχνολογία ολοκλήρωσης και αυξάνει η συχνότητα λειτουργίας του επεξεργαστή, τα ρεύματα διαρροής ενισχύονται, αυξάνοντας έτσι την κατανάλωση ενέργειας[11].



Σχήμα 1: Εξέλιξη του αριθμού τρανζίστορς σε ένα ολοκληρωμένο

- Το χάσμα ανάμεσα στην ταχύτητα του επεξεργαστή και της μνήμης μεγαλώνει, με αποτέλεσμα τα οφέλη ενός πιο γρήγορου ρολογιού να είναι μειωμένα [11][6].
- Οι RC καθυστερήσεις των καλωδίων του ολοκληρωμένου αυξάνονται με κάθε μείωση της τεχνολογίας ολοκλήρωσης, φράζοντας έτσι τα περιθώρια για μείωση της περιόδου του ρολογιού [11].
- Τα πιο βαθιά pipelines και η δυναμική εξαγωγή περισσότερου παραλληλισμού από τα προγράμματα παράγοντα διαρκώς μειούμενα οφέλη για διαρκώς μεγαλύτερο κόστος σε επιφάνεια ολοκληρωμένου και κατανάλωση ενέργειας [6].

Τα αποτελέσματα αυτών των προβλημάτων συνοψίστηκαν ιδανικά στην έκφραση “Power Wall + Memory Wall + ILP Wall = Brick Wall” [6]. Για να αντιμετωπίσει η βιομηχανία ημιαγωγών τα πολλαπλά αυτά “τείχη”, αναγκάστηκε από το 2005 και μετά να εγκαταλείψει το κυρίαρχο παράδειγμα εξέλιξης των μικροεπεξεργαστών μέσω αυξημένης πολυπλοκότητας και να στραφεί προς μία άλλη κατεύθυνση: τα ολοκληρωμένα με πολλαπλούς επεξεργαστές. Η προσθήκη πολλών επεξεργαστών σε ένα ολοκληρωμένο επιτυγχάνει να ξεπεράσει το τέλμα των μονοεπεξεργαστικών συστημάτων, όχι μόνο γιατί

αυξάνει την απόδοση μέσω της παράλληλης εκτέλεσης πολλών νημάτων/εφαρμογών, αλλά και γιατί το επιτυγχάνει αυτό χρησιμοποιώντας *απλούς, αργούς και ενεργειακά πιο αποδοτικούς επεξεργαστές*.

2.2 Η Ιεραρχία Μνήμης στους Πολυεπεξεργαστές ενός Ολοκληρωμένου

2.2.1 Το Χάσμα Μνήμης - Επεξεργαστή

Ενώ η περίοδος του κύκλου ρολογιού και του επεξεργαστή και της κύριας μνήμης μειώνονται εκθετικά σε ολόκληρη διάρκεια της εξέλιξης των μικροεπεξεργαστών, υπάρχει σημαντική διαφορά στον ρυθμό αυτής της εκθετικής μείωσης: ο ρυθμός μείωσης βρίσκεται ανάμεσα στο 5% [32] και στο 7% [91] για τις DRAM μνήμες, ενώ για τους επεξεργαστές η μείωση φτάνει το 33-38% ανά χρόνο [4]. Το συνολικό αποτέλεσμα είναι ότι η ίδια η απόκλιση ανάμεσα στην ταχύτητα του επεξεργαστή και της κύριας μνήμης αυξάνεται εκθετικά. Ήδη από το 1994 οι Wulf και McKee [93] είχαν προβλέψει ότι αυτή η συμπεριφορά θα έχει σαν αναπόφευκτο αποτέλεσμα την αντιμετώπιση ενός “τείχους μνήμης” (Memory Wall), δηλαδή το γεγονός ότι η πραγματική επεξεργαστική ισχύς περιορίζεται άμεσα από την ταχύτητα της κύριας μνήμης και όχι από τα χαρακτηριστικά του ρολογιού ή της αρχιτεκτονικής του επεξεργαστή. Παρότι υπάρχουν ακόμη κάποια περιθώρια αύξησης της απόδοσης του επεξεργαστή μέσω αύξησης της συχνότητας του ρολογιού, ήδη έχουμε αγγίξει το Memory Wall.

Αρκετές από τις αρχιτεκτονικές που έχουν ενσωματωθεί στους σύγχρονους επεξεργαστές έχουν ακριβώς σαν στόχο, να καθυστερήσουν και να αντιμετωπίσουν το Memory Wall. Οι κρυφές μνήμες προσπαθούν να γεφυρώσουν το χάσμα επεξεργαστή-μνήμης παρέχοντας στον επεξεργαστή ένα υποσύνολο των δεδομένων της εφαρμογής με πολύ χαμηλότερη καθυστέρηση απόκρισης από ότι η κύρια μνήμη. Η προφόρτωση (prefetching) δεδομένων προσπαθεί να προβλέψει τα δεδομένα που θα χρειαστεί στο μέλλον η εκτελούμενη εφαρμογή και να ξεκινήσει την φόρτωσή τους σε κάποια κρυφή μνήμη πριν τα ζητήσει ο επεξεργαστής. Η εκτέλεση εκτός σειράς προσπαθεί να καλύψει (περισσότερο ή λιγότερο επιτυχημένα) τους χρόνους απόκρισης των κρυφών μνημών και της κύριας μνήμης εκτελώντας διαθέσιμες εντολές όση ώρα διαρκεί η προσπέλαση. Οι SMT (Simultaneous MultiThreading) τεχνικές προσπαθούν να χρησιμοποιήσουν τις λειτουργικές μονάδες του επεξεργαστή που παραμένουν ανενεργές λόγω της αναμονής για δεδομένα

από την μνήμη εκτελώντας άλλα διαθέσιμα νήματα. Γενικότερα, η αντιμετώπιση του χάσματος επεξεργαστή-μνήμης είναι ανάμεσα στους κεντρικούς στόχους της σχεδίασης ενός επεξεργαστή.

2.2.2 Κρυφές Μνήμες και Πολυεπεξεργαστές ενός Ολοκληρωμένου

Όπως αναφέρθηκε, οι κρυφές μνήμες είναι από τις κυριότερες δομές που συμμετέχουν στην αντιμετώπιση του χάσματος μνήμης-επεξεργαστή και για αυτό ο σωστός σχεδιασμός και η αποδοτική διαχείρισή τους παίζει κεντρικό ρόλο στην επίτευξη υψηλών επιπέδων απόδοσης. Με την είσοδο στην εποχή των πολυεπεξεργαστικών συστημάτων ενός ολοκληρωμένου (CMPS), όμως, αυτός ο ρόλος γίνεται ακόμη πιο σημαντικός:

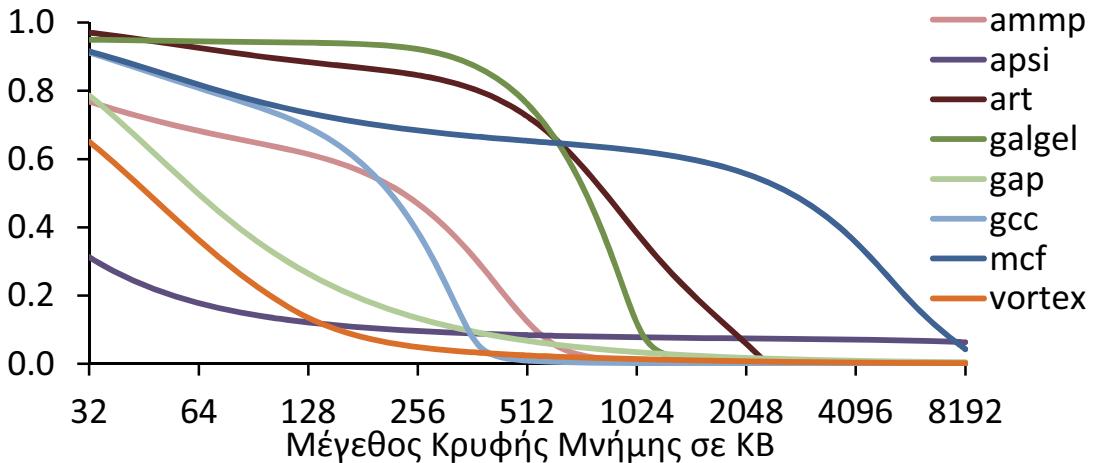
- Στις περισσότερες CMP αρχιτεκτονικές, το τελευταίο επίπεδο κρυφής μνήμης διαμοιράζεται ανάμεσα στους επεξεργαστές και μπορεί να λειτουργεί σαν κανάλι επικοινωνίας μεταξύ των νημάτων μίας εφαρμογής, επιτρέποντας έτσι μηνύματα και συγχρονισμό με εξαιρετικά μικρή καθυστέρηση και διαμοιρασμό μεγάλου όγκου πληροφορίας [17]. Η αποδοτική διαχείριση του τελευταίου επιπέδου της κρυφής είναι απαραίτητη ώστε να μπορέσουν οι πολυνηματικές εφαρμογές να εκμεταλλευτούν πλήρως το καινούργιο κανάλι επικοινωνίας.
- Για μη-πολυνηματικές εφαρμογές που εκτελούνται σε CMPS, η κοινόχρηστη φύση του τελευταίου επίπεδου της κρυφής μνήμης επιτρέπει τον ανταγωνισμό μεταξύ διαφορετικών εφαρμογών για την κατοχή ενός τόσο σημαντικού πόρου. Αν αυτός ο ανταγωνισμός δεν ελεγχθεί, τότε είναι αρκετά πιθανό κάποιες από τις εφαρμογές να μονοπωλήσουν τον έλεγχο της κρυφής μνήμης σε βάρος των υπόλοιπων [52].
- Η τοποθέτηση πολλαπλών επεξεργαστών σε ένα ολοκληρωμένο, χωρίς αντίστοιχη αύξηση της χωρητικότητας του εξωτερικού διαύλου δεδομένων του ολοκληρωμένου, αυξάνει την πιθανότητα να μετατραπεί αυτός ο δίαυλος σε bottleneck του συστήματος. Για να αποφευχθεί αυτό το ενδεχόμενο, θα πρέπει οι κρυφές μνήμες του ολοκληρωμένου να καταφέρουν να εξυπηρετήσουν ακόμη μεγαλύτερο κομμάτι των προσπελάσεων δεδομένων σε σχέση με τα μονο-επεξεργαστικά συστήματα.

Όλα αυτά τα προβλήματα θα πρέπει αντιμετωπιστούν από τους αρχιτέκτονες υπολογιστικών συστημάτων. Παρότι υπάρχει πάντα η δυνατότητα αύξησης της χωρητικότητας των κρυφών μνημάτων για να αντιμετωπιστούν πολλά από αυτά τα προβλήματα, μόνη της είναι μη αποδοτική λύση: οι μνήμες είναι ή μεγάλες σε χωρητικότητα

ή γρήγορες (vast or fast). Η μεγάλη αύξηση του μεγέθους των κρυφών μνημών, θα προκαλούσε ανάλογα μεγάλη αύξηση του χρόνου προσπέλασης τους, με αρνητικές συνέπειες στην ταχύτητα επεξεργασίας του επεξεργαστή.

2.3 Η μεθοδολογία StatCache

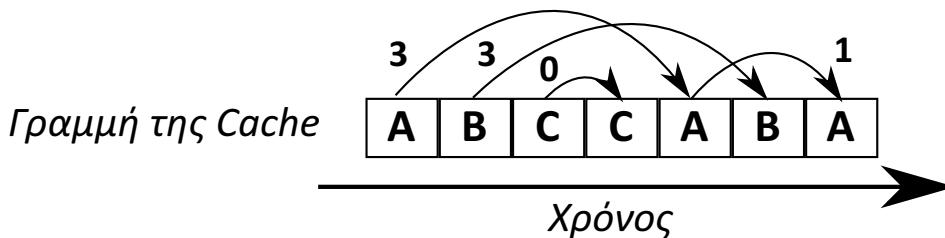
Σε όλα τα προβλήματα που σχετίζονται με την διαχείριση κρυφών μνημών, είτε ιδιωτικών είτε κοινόχρηστων, είναι σημαντική η κατανόηση του πως χρησιμοποιούν οι εφαρμογές την κρυφή μνήμη και ποιος είναι πραγματικά ο χώρος που χρειάζονται. Συνήθως, μία τέτοια ανάλυση απαιτεί πολλαπλές εξομοιώσεις της κρυφής μνήμης, ώστε να υπολογιστεί ο βαθμός αστοχίας για κάθε πιθανή χωρητικότητα κρυφής μνήμης και να εξαχθούν συμπεράσματα για τις ανάγκες της εφαρμογής σε κρυφή μνήμη. Άλλες μεθοδολογίες βασίζονται στην συλλογή στατιστικών για το βάθος στην LRU στοίβα όπου βρίσκεται κάθε γραμμή την στιγμή της προσπέλασής της (stack distance). Τέτοιου είδους στατιστικά επιτρέπουν την εύκολη εύρεση του βαθμού αστοχίας για κάθε πιθανό μέγεθος κρυφής μνήμης βάσει μίας μέτρησης μόνο, αλλά έχουν το μειονέκτημα της δύσκολης συλλογής: η κατάσταση της LRU αλυσίδας είναι συνάρτηση όλων των προσπελάσεων που έχουν προηγηθεί και επομένως απαιτείται η πλήρης εξομοίωσή της ώστε να μπορέσουν να υπολογιστούν τα stack distances.



Σχήμα 2: Καμπύλες StatCache για μερικά από τα μετροπρογράμματα της σουίτας SPEC2000

Η μεθοδολογία StatCache των Berg και Hagersten [8][9] ακολουθεί μία διαφορετική προσέγγιση στο πρόβλημα της εκτίμησης του βαθμού αστοχίας: την πιθανοτική μοντελοποίηση της κρυφής μνήμης. Πιο συγκεκριμένα, κατά την εκτέλεση ενός προγράμματος η StatCache δειγματοληπτεί στατιστική πληροφορία για την τοπικότητα

των προσπελάσεων, τις αποστάσεις επαναχρησιμοποίησης (reuse distances). Βάσει αυτής της πληροφορίας παράγεται ένα ιστόγραμμα των reuse distances, το οποίο αντιπροσωπεύει την κατανομή της τοπικότητας του προγράμματος. Αφού ολοκληρωθεί η εκτέλεση του προγράμματος, το ιστόγραμμα τροφοδοτείται σε ένα πιθανοτικό μοντέλο, το οποίο συσχετίζει κάθε reuse distance με την πιθανότητα να παραχθεί αστοχία μνήμης και υπολογίζει για το ιστόγραμμα εισόδου το ποσοστό των αστοχιών του προγράμματος. Ένα παράδειγμα της εξόδου της μεθοδολογίας StatCache φαίνεται στο Σχήμα 2.



Σχήμα 3: Reuse Distances - Κάθε κουτί αντιπροσωπεύει μία προσπέλαση κρυφής μνήμης στην διεύθυνση που περιέχει το κουτί. Τα βέλη δείχνουν την επαναχρησιμοποίηση κάθε γραμμής και ο αριθμός δίπλα τους το reuse distance

Τα πλεονεκτήματα της μεθοδολογίας StatCache πηγάζουν από τον ορισμό των reuse distances. Όπως φαίνεται και στο Σχήμα 3, reuse distance είναι ο αριθμός προσπελάσεων ανάμεσα σε δύο προσπελάσεις προς την ίδια γραμμή της κρυφής μνήμης. Παρότι, μοιάζουν με τα stack distances, διαφέρουν σε ένα κρίσιμο σημείο: για τον καθορισμό του reuse distance ανάμεσα σε δύο προσπελάσεις προς την ίδια γραμμή χρειάζεται μόνο ο αριθμός των ενδιάμεσων προσπελάσεων, ενώ για το καθορισμό του stack distance χρειάζεται και πληροφορία για τις διευθύνσεις που προσπελάστηκαν. Αυτή η διαφορά έχει δύο συνέπειες:

- Για την μέτρηση των reuse distances αρκεί η ύπαρξη ενός μετρητή προσπελάσεων και η αποθήκευση, για κάθε γραμμή της κρυφής μνήμης, της τιμής του μετρητή την στιγμή της τελευταίας προσπέλασης. Με αυτόν τον τρόπο, το reuse distance της προσπέλασης μίας γραμμής είναι απλά η διαφορά ανάμεσα στην τρέχουσα και την αποθηκευμένη τιμή του μετρητή.
- Εφόσον δεν χρειάζεται η παρακολούθηση όλων των γραμμών της κρυφής μνήμης για να εξαχθεί ένα reuse distance, η μεθοδολογία μπορεί να δειγματοληπτεί τις προσπελάσεις της κρυφής μνήμης, δηλαδή να κρατάει χρόνους τελευταίας προσπέλασης μόνο για ένα μικρό υποσύνολο των προσπελάσεων και μόνο για αυτές να παράγει reuse distances.

Συνολικότερα, η συλλογή της πληροφορίας επαναχρησιμοποίησης απαιτεί μόνο τρεις μηχανισμούς: επιλογή και αποθήκευση δειγμάτων, μέτρηση προσπελάσεων και παρακολούθηση των προσπελάσεων για να βρεθούν προσπελάσεις προς γραμμές που έχουν δειγματοληφθεί (watchpoint mechanism). Και οι τρεις μηχανισμοί χαρακτηρίζονται από χαμηλή υπολογιστική πολυπλοκότητα και χαμηλές απαιτήσεις σε αποθηκευτικό χώρο, οπότε η παρεμβολή τους στην εκτέλεση του προγράμματος έχει σχετικά χαμηλές συνέπειες. Επιπλέον, σε πολλές αρχιτεκτονικές επεξεργαστών η λειτουργικότητα των μηχανισμών συλλογής στατιστικών της StatCache είναι ήδη υλοποιημένη σε υλικό (πχ performance counters για μέτρηση προσπελάσεων, debug/watchpoint registers για παρακολούθηση των προσπελάσεων), και επομένως η πληροφορία επαναχρησιμοποίησης μπορεί να συλλεχθεί με ασήμαντο κόστος σε ότι αφορά την επιβράδυνση του παρακολουθούμενου προγράμματος.

Μετά την ολοκλήρωση της εκτέλεσης του προγράμματος, η πληροφορία επαναχρησιμοποίησης που συλλέχθηκε τροφοδοτείται στο πιθανοτικό μοντέλο της StatCache. Οι κεντρικές συναρτήσεις του μοντέλου, για κρυφές μνήμες με Random πολιτική αντικατάστασης, είναι:

$$N \times R \approx h(1) \times f(R) + h(2) \times f(2R) + h(3) \times f(3R) + \dots$$

$$f(n) = \left[1 - \left(\frac{1}{L} \right)^n \right]$$

όπου N ο αριθμός των δειγμάτων που συλλέχθηκαν, R το ποσοστό αστοχιών, $h(i)$ ο αριθμός δειγμάτων με reuse distance i , $f(n)$ η πιθανότητα μία γραμμή να μην βρίσκεται στην κρυφή μνήμη μετά από n αντικαταστάσεις και L το μέγεθος της κρυφής μνήμης μετρημένο σε cache blocks.

Αντίστοιχα, για κρυφές μνήμες με LRU αλγόριθμο αντικατάστασης, η κεντρική συνάρτηση του στατιστικού μοντέλου, έτσι όπως εξελίχθηκε από τους Eklov και Hagersten [22], είναι:

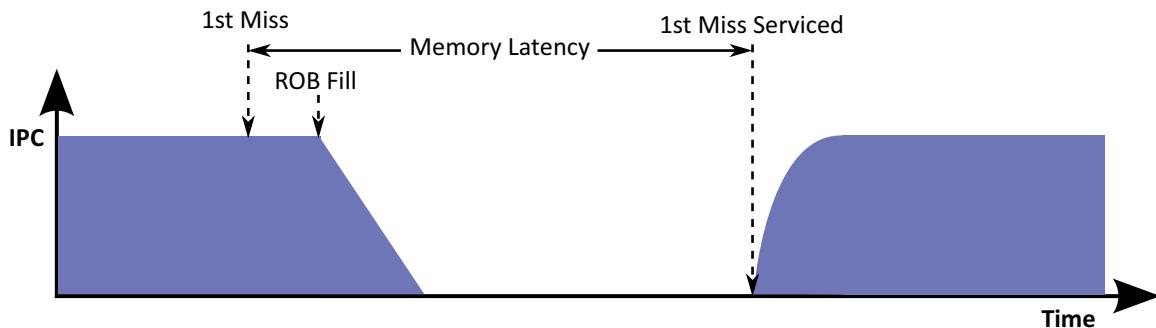
$$ES(r) = \sum_{j=1}^r \frac{\hat{n}(j)}{\hat{n}}$$

όπου $ES(r)$ το αναμενόμενο stack distance για reuse distance r , $n(j)$ ο αριθμός δειγμάτων με reuse distance μεγαλύτερο του j και \hat{n} ο αριθμός των δειγμάτων. Με βάσει το $ES(r)$ η κατανομή των reuse distances μεταφράζεται σε κατανομή των stack distances και από αυτήν την κατανομή υπολογίζεται το ποσοστό αστοχιών.

Και οι δύο ομάδες εξισώσεων χαρακτηρίζονται από υψηλή πολυπλοκότητα, γεγονός που περιορίζει την χρησιμότητα της μεθοδολογίας σε offline μελέτη των χαρακτηριστικών των εφαρμογών. Όμως, όπως θα γίνει εμφανές και στην συνέχεια, η μεθοδολογία StatCache και ειδικά η ιδέα της περιγραφής της τοπικότητας μέσω των reuse distances μπορούν να αποτελέσουν αφετηρία για αρκετές online τεχνικές διαχείρισης και μελέτης των κρυφών μνημών.

2.4 Memory Level Parallelism

Μια από τις έννοιες που σχετίζονται άμεσα με το χάσμα επεξεργαστή-μνήμης και την επίδρασή του στην απόδοση του επεξεργαστή είναι το Memory Level Parallelism (MLP). Το MLP είναι η δυνατότητα των επεξεργαστών εκτός σειράς να εκτελούν παράλληλα προσπελάσεις της κύριας μνήμης, με αποτέλεσμα να επικαλύπτονται μεταξύ τους τα χρονικά διαστήματα που χρειάζονται για την φόρτωση των δεδομένων από την μνήμη. Με αυτόν τον τρόπο το κόστος μίας προσπέλασης της κύριας μνήμης μοιράζεται ανάμεσα στις παράλληλες προσπελάσεις και επομένως για τον επεξεργαστή το φαινομενικό κόστος κάθε τέτοιας προσπέλασης είναι ένα κλάσμα του κόστους μίας μεμονωμένης προσπέλασης κύριας μνήμης.



Σχήμα 4: IPC του επεξεργαστή κατά την διάρκεια εξυπηρέτησης μεμονωμένης αστοχίας στο τελευταίο επίπεδο κρυφής μνήμης

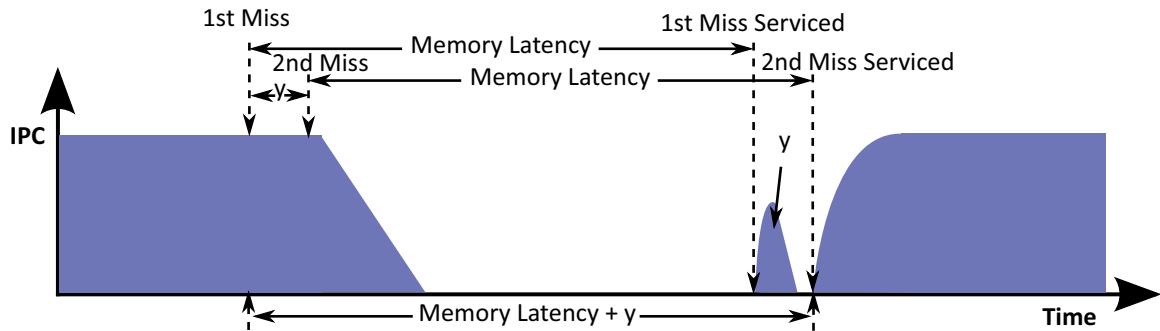
Το θεωρητικό μοντέλο των Karkhanis και Smith [44], που αναπτύχθηκε περαιτέρω από τους Eyerman et al. [24], παρέχει μία αρκετά καλή κατανόηση του τρόπου με τον οποίο το MLP επηρεάζει την λειτουργία του επεξεργαστή. Στο Σχήμα 4 παρουσιάζεται το IPC του επεξεργαστή συναρτήσει του χρόνου, κατά την εξυπηρέτηση μίας αστοχίας στο τελευταίο επίπεδο κρυφής μνήμης, σύμφωνα με το μοντέλο. Πριν και μετά την εξυπηρέτηση της αστοχίας, το IPC θεωρείται σταθερό και εξαρτάται από την IW χαρακτηριστική του προγράμματος που εκτελείται και από το πλάτος του σταδίου issue του επεξεργαστή. Όταν

συμβεί μία αστοχία, για ένα μικρό χρονικό διάστημα ο επεξεργαστής συνεχίζει να εκτελεί εντολές χωρίς ιδιαίτερη μεταβολή στον ρυθμό εκτέλεσης, μέχρι η αστοχία να φτάσει στην κορυφή του παραθύρου εντολών, να σταματήσει η απόσυρση εντολών και να γεμίσουν όλες οι εγγραφές του ROB (ROB Fill). Στο αμέσως επόμενο χρονικό διάστημα (Ramp-Down), ο επεξεργαστής συνεχίζει να εκτελεί εντολές, αλλά επειδή δεν μπαίνουν νέες εντολές στον ROB, ο scheduler βρίσκει ολοένα και λιγότερες διαθέσιμες για εκτέλεση εντολές, οπότε ο ρυθμός εκτέλεσης πέφτει. Μετά από μερικούς κύκλους, όσες εντολές μπορούσαν να εκτελεστούν εκτελέστηκαν, οπότε το IPC πέφτει στο μηδέν και ο επεξεργαστής αδρανεί πλήρως μέχρι να τελειώσει η εξυπηρέτηση της προσπέλασης κρυφής μνήμης. Αφού ολοκληρωθεί η εξυπηρέτηση, οι εντολές που περίμεναν τα δεδομένα της αστοχίας γίνονται διαθέσιμες για εκτέλεση, οπότε ανεβαίνει άμεσα ο ρυθμός εκτέλεσης (Ramp-Up). Ταυτόχρονα, αρχίζουν να αποσύρονται εντολές και πάλι, οπότε μετά από μερικούς κύκλους ρολογιού όλες οι εντολές που είχαν “κολλήσει” στο ROB έχουν αποσυρθεί και αντικατασταθεί με εντολές που περιμένουν εκτέλεση, και επομένως το σύστημα έχει επιστρέψει πια στη κατάσταση που βρισκόταν πριν την αστοχία κρυφής μνήμης. Σύμφωνα με την ανάλυση των Karkhanis και Smith, το κόστος μίας τέτοιας προσπέλασης της κύριας μνήμης είναι:

$$\text{Cost} = \text{memoryLatency} + \text{rampUp} - \text{robFill} - \text{rampDown}$$

Στην περίπτωση όμως που υπάρχει ταυτόχρονα στο παράθυρο εντολών και μία δεύτερη προσπέλαση η οποία προκαλεί αστοχία στο τελευταίο επίπεδο κρυφής μνήμης, η συμπεριφορά του επεξεργαστή περιγράφεται από το Σχήμα 5. Αφού συμβεί η πρώτη αστοχία, συνεχίζουν και εκτελούνται κανονικά οι εντολές του παραθύρου εντολών και έτσι κάποιες γενικές αστοχίες αργότερα εκτελείται και μία εντολή μνήμης που προκαλεί μία δεύτερη αστοχία κρυφής μνήμης. Κατά τις φάσεις του ROB Fill, Ramp Up και αδράνειας, ο επεξεργαστής συμπεριφέρεται ακριβώς όπως και στην περίπτωση της μεμονωμένης αστοχίας.

Αφού όμως εξυπηρετηθεί η πρώτη αστοχία, οι γενικές αστοχίες αποσύρονται ή/και εκτελούνται, οπότε απελευθερώνεται χώρος στο ROB για γενικές αστοχίες, αρκετές από τις οποίες εκτελούνται, πριν αδρανήσει και πάλι ο επεξεργαστής. Αθροιστικά, αναμένεται ότι θα σε αυτό το διάστημα θα εκτελεστούν γενικές αστοχίες. Η διαδικασία ολοκληρώνεται με την εξυπηρέτηση της δεύτερης αστοχίας και την φάση του



Σχήμα 5: IPC του επεξεργαστή κατά την διάρκεια εξυπηρέτησης δύο παράλληλων αστοχιών στο τελευταίο επίπεδο κρυφής μνήμης

Ramp Up. Με βάσει αυτήν την περιγραφή της εξυπηρέτησης δύο παράλληλων αστοχιών, οι Karkhanis και Smith καταλήγουν στο ότι το κόστος και των δύο αστοχιών είναι:

$$\begin{aligned} \text{Cost} &= \text{memoryLatency} + y + \text{rampUp} - \text{robFill} - \text{rampDown} - y \\ \text{Cost} &= \text{CostOfIsolatedMiss} \end{aligned}$$

που σημαίνει ότι το αθροιστικό κόστος των παράλληλων αστοχιών είναι ακριβώς το ίδιο με το κόστος μία μεμονωμένης αστοχίας, ανεξάρτητα από την απόσταση μεταξύ των αστοχιών και τα χαρακτηριστικά του επεξεργαστή, της μνήμης και του προγράμματος.

Ήδη από το 1998, ο Andrew Glew [30] επεσήμανε την κεντρικότητα που πρέπει να έχει το MLP στον σχεδιασμό επεξεργαστών και πρότεινε τις βασικές κατευθύνσεις μέσω των οποίων θα μπορέσουν οι μελλοντικοί επεξεργαστές να εκμεταλλευτούν το MLP. Τα επόμενα χρόνια αρκετές εργασίες κινήθηκαν πάνω στις ιδέες του Andrew Glew, είτε σχεδιάζοντας αρχιτεκτονικές που αυξάνουν τον βαθμό παραλληλισμού των προσπελάσεων κύριας μνήμης είτε αναπτύσσοντας τεχνικές διαχείρισης του επεξεργαστή και της ιεραρχίας μνήμης που έχουν σαν στόχο την προστασία του MLP. Οι δύο πιο σημαντικές, σε ότι αφορά την παρούσα διατριβή, είναι η εργασία “A Case for MLP-Aware Cache Replacement” των Qureshi et al [72] και η εργασία “A Memory-Level Parallelism Aware Fetch Policy for SMT Processors” των Eyerman και Eeckhout [23].

Στην [72], οι συγγραφείς έδειξαν ότι οι τεχνικές που στοχεύουν στην μείωση του βαθμού αστοχίας των κρυφών μνημών είναι αρκετά πιθανό να μην οδηγήσουν σε αντίστοιχη αύξηση της ταχύτητας επεξεργασίας, επειδή πολλές από τις προσπελάσεις που προκαλούν αστοχίες κρυφής μνήμης είναι εξαιρετικά παράλληλες. Αν τέτοιες προσπελάσεις μετατραπούν σε ευστοχίες κρυφής μνήμης, δεν θα έχουν κανένα ή σχεδόν κανένα θετικό αποτέλεσμα στην απόδοση του επεξεργαστή. Για αυτόν τον λόγο, οι συγγραφείς πρότειναν μία τροποποιημένη LRU πολιτική αντικατάστασης, η οποία στοχεύει στην ελαχιστοποίηση

του κόστους των αστοχιών. Η πολιτική επιλέγει πιο εύκολα για αντικατάσταση γραμμές της κρυφής μνήμης που αναμένεται να προσπελασθούν με υψηλό MLP και κρατάει για περισσότερη ώρα γραμμές που αναμένεται να μην έχουν παραλληλισμό αν φορτωθούν από την κύρια μνήμη. Με αυτόν τον τρόπο, η πολιτική καταφέρνει να μειώνει το συνολικό κόστος των αστοχιών κρυφής μνήμης και να αυξάνει το IPC του επεξεργαστή, ακόμη και όταν αυξάνεται ο αριθμός των αστοχιών.

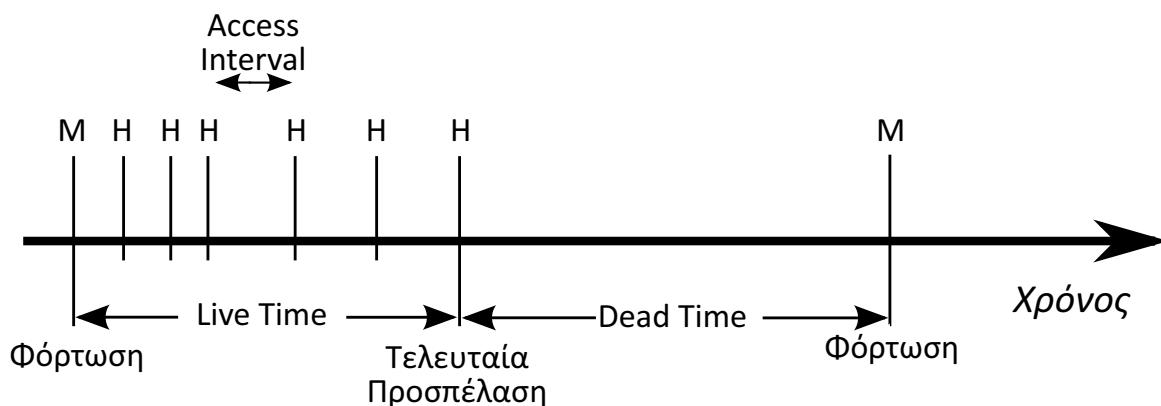
Στην [23], οι συγγραφείς μελετούν πολιτικές διαχείρισης του σταδίου fetch για SMT επεξεργαστές. Μέσα από την εργασία τους δείχνουν ότι η πιο σημαντική πληροφορία που χρειάζεται για την αποτελεσματική διαχείριση του σταδίου fetch είναι αν μία αστοχία μνήμης είναι μεμονωμένη ή παράλληλη. Αν είναι μεμονωμένη, τότε από τις εντολές που την ακολουθούν στο πρόγραμμα σχετικά λίγες θα καταφέρουν να εκτελεστούν πριν την εξυπηρέτηση της αστοχίας, οπότε είναι πιο αποδοτικό να εισαχθούν στον επεξεργαστή εντολές από τα υπόλοιπα νήματα που μοιράζονται τον επεξεργαστή. Αν όμως η αστοχία αναμένεται να είναι παράλληλη, τότε πρέπει να εισαχθούν στον επεξεργαστή αρκετές εντολές του ίδιου νήματος, ώστε να εκτελεστούν και οι υπόλοιπες προσπελάσεις που θα προκαλέσουν παράλληλη προσπέλαση της κύριας μνήμης.

Συνολικότερα, όλες οι προαναφερθείσες εργασίες είχαν σημαντικό ρόλο στις κατευθύνσεις που ακολουθήθηκαν στα πλαίσια της διατριβής. Οι μεν [44] και [30] μας βοήθησαν να κατανοήσουμε την σημαντικότητα του MLP και τον τρόπο με τον οποίο επιδρά στον επεξεργαστή, ενώ οι [72] και [23] μας έδειξαν ότι οι μηχανισμοί διαχείρισης του επεξεργαστή είναι απαραίτητο να χειρίζονται και να προστατεύουν το MLP.

2.5 Η Τεχνική Cache Decay

Ο Cache Decay μηχανισμός αποτελεί έναν τρόπο ελέγχου των κρυφών μνημών και αναπτύχθηκε από τους Kaxiras, Hu και Martonosi [33][46]. Ο στόχος του αρχικού μηχανισμού [46] ήταν η μείωση των ρευμάτων διαρροής στις κρυφές μνήμες του επεξεργαστή, κλείνοντας την τροφοδοσία σε μεγάλα τμήματα τους με συγκριτικά χαμηλή υποβάθμιση της απόδοσης του επεξεργαστή. Στην δεύτερη εργασία [33], ο πυρήνας της τεχνικής Cache Decay εξελίχθηκε και προσαρμόστηκε ώστε να οδηγήσει μία τεχνική προφόρτωσης δεδομένων στην κρυφή μνήμη (prefetching). Η τεχνική ενέπνευσε κομμάτια των μεθοδολογιών που παρουσιάζονται στα Κεφάλαια 3 και 4 και γι' αυτό κρίνεται σκόπιμο να γίνει μία παρουσίαση του Cache Decay.

Η κεντρική ιδέα του Decay μηχανισμού προκύπτει από την παρατήρηση της συμπεριφοράς των δεδομένων στην κρυφή μνήμη. Στο Σχήμα 6 φαίνεται η τυπική εξέλιξη της ζωής των δεδομένων μέσασε μία κρυφή μνήμη πρώτου επιπέδου, από την στιγμή που κάποιο δεδομένο εισάγεται στην κρυφή μνήμη μέχρι αυτό να αντικατασταθεί από κάποιο άλλο δεδομένο. Όπως φαίνεται στο Σχήμα 6, μετά την εισαγωγή του δεδομένου στην κρυφή μνήμη ακολουθεί μία περίοδος έντονης χρήσης, η οποία ακολουθείται από ολοένα και πιο αραιές προσπελάσεις έως ότου το δεδομένο παύσει να χρησιμοποιείται οπότε και αντικαθίσταται από ένα νέο.



Σχήμα 6: Ζωή ενός δεδομένου στην κρυφή μνήμη

Ιδανικά, ένας μηχανισμός διαχείρισης θα έπρεπε να αναγνωρίζει την τελευταία χρήση ενός δεδομένου και να το αντικαθιστά άμεσα, ώστε να ελαχιστοποιηθεί ο χρόνος που το δεδομένο καταλαμβάνει χώρο της κρυφής μνήμης χωρίς να προσφέρει στο ποσοστό ευστοχίας της (dead time). Προφανώς, αυτός ο ιδανικός μηχανισμός δεν είναι δυνατό να υλοποιηθεί, καθώς απαιτεί γνώση της μελλοντικής συμπεριφοράς κάθε δεδομένου. Το Cache Decay όμως επιτυγχάνει να τον προσεγγίσει σε σημαντικό βαθμό, υποθέτοντας ότι υπάρχει ένα πάνω όριο στο χρονικό διάστημα ανάμεσα σε δύο επαναχρησιμοποιήσεις ενός δεδομένου κατά την διάρκεια ζωής του δεδομένου στην κρυφή μνήμη. Αν ξεπεραστεί αυτό το όριο, το οποίο ονομάζεται Decay Time, τότε κατά πάσα πιθανότητα το δεδομένο έχει σταματήσει να χρησιμοποιείται και η τεχνική Cache Decay μπορεί να το διώξει από την κρυφή μνήμη χωρίς κόστος.

Η επιλογή του κατάλληλου Decay Time είναι κρίσιμη για την επιτυχή λειτουργία του Cache Decay. Εφόσον, δεν παρουσιάζουν ούτε όλα τα προγράμματα ούτε όλα τα δεδομένα την ίδια συμπεριφορά σε ότι αφορά τα πρότυπα επαναχρησιμοποίησής τους, το Decay Time επιλέγεται ώστε να ισορροπήσει δύο αντικρουνόμενους στόχους: την όσο το δυνατόν πιο

γρήγορη αναγνώριση του τέλους του live time του δεδομένου και την όσο το δυνατόν μικρότερη αύξηση του ποσοστού αστοχίας της κρυφής μνήμης. Πιο συγκεκριμένα, όσο περισσότερο περιμένει το Cache Decay για να χαρακτηρίσει ένα δεδομένο ως άχρηστο τόσο μικραίνει η πιθανότητα είναι λάθος αυτή η απόφαση, αλλά η συνέπεια της μεγάλης αναμονής είναι ότι τα άχρηστα δεδομένα μένουν περισσότερο χρόνο στην κρυφή μνήμη. Μέσα από την αναλυτική μελέτη των χαρακτηριστικών των προγραμμάτων, η χρυσή τομή ανάμεσα στους δύο αυτούς στόχους βρέθηκε κοντά στους 8000 κύκλους ρολογιού.

Η υλοποίηση της Cache Decay τεχνικής είναι εξαιρετικά απλή. Σε κάθε block της κρυφής μνήμης προσθέτουμε έναν μετρητή των 2 bits. Ο μετρητής αυτός μηδενίζεται κάθε φορά που προσπελάζεται το block και αυξάνεται από τον παλμό εξόδου ενός μετρητή, κοινού για όλα τα blocks, ο οποίος μετράει έναν προκαθορισμένο αριθμό από κύκλους. Όταν ο τοπικός μετρητής ενός block έχει φτάσει την μέγιστη τιμή του και έρθει ένας ακόμη παλμός από τον κεντρικό μετρητή, τότε το δεδομένο έχει μείνει αχρησιμοποίητο για σχεδόν 4 x (περίοδος του κεντρικού μετρητή) κύκλους, το οποίο είναι και το Decay Time του μηχανισμού, οπότε ή κλείνει η τροφοδοσία της γραμμής (στην περίπτωση του [46]) ή ενημερώνεται ο prefetcher ότι υπάρχει διαθέσιμος χώρος στην κρυφή μνήμη (στην περίπτωση του [33]). Μία επέκταση της τεχνικής Cache Decay, η οποία περιγράφεται στην εργασία των Kaxiras et al [48] απλοποιεί ακόμη περισσότερο την υλοποίηση του Cache Decay χρησιμοποιώντας, αντί για κεντρικούς και τοπικούς μετρητές, ένα 4T decaying cell [47] για κάθε block της κρυφής μνήμης. Τα 4T decaying cells μεταβαίνουν αυτόμata σε χαμηλή κατάσταση αν δεν προσπελαστούν για ένα χρονικό διάστημα, οπότε αν αυτό το χρονικό διάστημα τεθεί ίσο με το Decay Time και το κελί προσπελάζεται μαζί με την προσπέλαση του block, το κελί μπορεί να οδηγήσει τις αποφάσεις του Cache Decay. Σαν συνέπεια, Cache Decay τεχνικές μπορούν να υλοποιηθούν με κόστος μόνο 4 τρανζίστορς ανά block.

Αρκετές μεθοδολογίες [3][90][97][50] έχουν αξιοποιήσει το Decay με πολλαπλούς τρόπους, η καθεμία για τους δικούς της στόχους, αλλά όλες βασίζονται, πρώτον, στην ικανότητα του Cache Decay να αναγνωρίζει, με απλούς μηχανισμούς, το τέλος της ζωής ενός δεδομένου και, δεύτερον, να διαχειρίζεται τον χώρο της κρυφής μνήμης με ανάλυση του ενός block. Αυτά ακριβώς τα χαρακτηριστικά οδήγησαν στην χρήση Cache Decay τεχνικών και σε τμήματα της παρούσας διατριβής.

Κεφάλαιο 3

Μοντελοποίηση και Διαχείριση του Διαμοιρασμού Κοινόχρηστων Κρυφών Μνημών

3.1 Εισαγωγή

Όπως αναλύθηκε και στα προηγούμενα κεφάλαια, τα τελευταία χρόνια τα ολοκληρωμένα με πολλαπλούς επεξεργαστές έχουν γίνει το κεντρικό παράδειγμα σχεδιασμού επεξεργαστών. Σε τέτοια συστήματα, κάθε επεξεργαστής περιέχει το δικό του αντίγραφο του επεξεργαστικού πυρήνα, των κρυφών μνημών των ανωτέρων επιπέδων, των διεπαφών προς τους εσωτερικούς διαύλους του επεξεργαστή, ενώ οι υπόλοιπες δομές του ολοκληρωμένου μοιράζονται μεταξύ των επεξεργαστών. Μία από αυτές τις δομές είναι συνήθως και το τελευταίο επίπεδο κρυφής μνήμης. Όπως με όλους τους κοινόχρηστους πόρους, χωρίς διαχείριση της κοινόχρηστης κρυφής μνήμης είναι εφικτό για έναν επεξεργαστή να μονοπωλήσει τον έλεγχο της. Λόγω της κρισιμότητας του τελευταίου επιπέδου μνήμης στην απόδοση του επεξεργαστή (Ενότητα 2.2.1), ένα τέτοιο σενάριο έχει καταστροφικές συνέπειες για τους υπόλοιπους επεξεργαστές. Επομένως, είναι απαραίτητη η κατανόηση και η έξυπνη διαχείριση του ανταγωνισμού μεταξύ των επεξεργαστών για τον έλεγχο του χώρου της κρυφής μνήμης.

Στα πλαίσια αυτού του κεφαλαίου παρουσιάζεται μία νέα θεωρητική μεθοδολογία, η μεθοδολογία StatShare, που περιγράφει με ακρίβεια το πως αλληλεπιδρούν οι επεξεργαστές

στην κοινόχρηστη κρυφή μνήμη. Το μοντέλο της μεθοδολογίας StatShare εμπνεύστηκε από το αντίστοιχο μοντέλο StatCache [8] που παρουσιάστηκε στο προηγούμενο κεφάλαιο. Οι διαφορές ανάμεσα στα δύο μοντέλα πηγάζουν από την διαφορετική περιγραφή των αποστάσεων επαναχρησιμοποίησης (reuse distance): ενώ για το StatCache reuse distance είναι ο αριθμός των προσπελάσεων ανάμεσα σε δύο προσπελάσεις του ίδιου δεδομένου, το StatShare χρησιμοποιεί σαν μονάδα μέτρησης των reuse distances τον αριθμό αντικαταστάσεων της κρυφής μνήμης που μελετάμε (Cache Allocation Ticks, CAT [45]). Αυτή η φαινομενικά μικρή διαφορά έχει σαν αποτέλεσμα ένα πολύ απλό μοντέλο το οποίο μπορεί να αξιοποιηθεί ακόμη και κατά την διάρκεια εκτέλεσης των μελετώμενων προγραμμάτων.

Στα πλαίσια της μεθοδολογίας StatShare επιδεικνύεται επίσης το πως ακριβώς μπορεί να χρησιμοποιηθεί η πληροφορία του στατιστικού μοντέλου για την διαχείριση του διαμοιρασμού της κοινόχρηστης κρυφής μνήμης. Η τεχνική διαχείρισης που αναπτύσσεται βασίζεται στην τεχνική Cache Decay (Ενότητα 2.5). Λόγω της ικανότητας του Cache Decay να απενεργοποιεί ή να απομακρύνει γραμμές της κρυφής μνήμης που είναι απίθανο να ξαναχρησιμοποιηθούν στο κοντινό μέλλον, μπορούμε να το χρησιμοποιήσουμε για να περιορίσουμε τον χώρο της κοινόχρηστης κρυφής μνήμης που καταλαμβάνει κάθε επεξεργαστής χωρίς σημαντική υποβάθμιση της απόδοσής του. Αξιοποιώντας αυτήν την δυνατότητα περιορισμού του χώρου που καταλαμβάνει κάθε εφαρμογή μπορούμε έπειτα να υλοποιήσουμε υψηλού επιπέδου πολιτικές, όπως QoS πολιτικές [95], δικαιοσύνη στον διαμοιρασμό της κρυφής μνήμης [52][18][58] ή απλά μεγιστοποίηση της συνολικής απόδοσης του πολυεπεξεργαστικού συστήματος [84][78]. Για να προσαρμοστεί το Cache Decay στον ρόλο του ως μηχανισμός διαχείρισης της κοινόχρηστης κρυφής μνήμης, η τεχνική τροποποιείται σε δύο κρίσιμα σημεία. Πρώτον, οι γραμμές που έχουν μείνει αχρησιμοποίητες για χρόνο πάνω από το Decay Time (decayed γραμμές) δεν απενεργοποιούνται, απλά σημειώνονται ως διαθέσιμες για αντικατάσταση. Δεύτερον, το Decay Time δεν μετριέται σε κύκλους αλλά σε CAT, ώστε να βασίζεται στην ίδια έννοια του χρόνου με το μοντέλο StatShare και να είναι δυνατή η ενσωμάτωση του στο μοντέλο.

Η τεχνική που προκύπτει από αυτές τις τροποποιήσεις, το *CAT Decay*, παρουσιάζει σημαντικά πλεονεκτήματα σε σχέση με άλλες τεχνικές διαχείρισης του διαμοιρασμού της κρυφής μνήμης. Η μονάδα διαχείρισης του CAT Decay είναι το ένα μεμονωμένο block της κρυφής μνήμης, δηλαδή δεν περιορίζει τις εφαρμογές σε συγκεκριμένα sets ή ways, όπως άλλες τεχνικές. Κάθε εφαρμογή μπορεί να τοποθετεί τα δεδομένα της σε οποιοδήποτε block

της κρυφής μνήμης —ο μόνος στόχος του CAT Decay είναι ο έλεγχος του συνολικού χώρου που καταλαμβάνει η εφαρμογή. Επιπλέον, το Decay επιβάλλει όρια μόνο στον μέσο χώρο της κάθε εφαρμογής, επιτρέποντας προσωρινές αυξομειώσεις ώστε ο χώρος που καταλαμβάνει η εφαρμογή να ακολουθεί τις στιγμιαίες μεταβολές στις ανάγκες της.

3.2 Σχετικές Ερευνητικές Εργασίες

3.2.1 Τεχνικές Διαμοιρασμού Κοινόχρηστων Κρυφών Μνημάτων

Το θέμα του δίκαιου διαμοιρασμού μία κοινόχρηστης κρυφής μνήμης μελετήθηκε πρώτη φορά στην εργασία “Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture” των Kim et al. [52]. Στην εργασία αυτή εισάγεται μία ομάδα από κριτήρια μέτρησης της δικαιοσύνης στον διαμοιρασμό και υλοποιούνται ένας στατικός και ένας δυναμικός αλγόριθμος διαμοιρασμού στο επίπεδο του λειτουργικού συστήματος. Οι αποφάσεις του αλγορίθμου βασίζονται σε μετρητές των stack distances των προσπελάσεων [60]. Αν με βάση την stack distance πληροφορία, ο αλγόριθμος αποφασίσει ότι μία εφαρμογή δικαιούται περισσότερο χώρο της κρυφής μνήμης, τότε επιλέγει για αντικατάσταση γραμμές άλλων εφαρμογών. Αντίστροφα, αν η εφαρμογή κατέχει περισσότερο χώρο από ότι της αναλογεί, τότε διαλέγει για αντικατάσταση μόνο δικές της γραμμές.

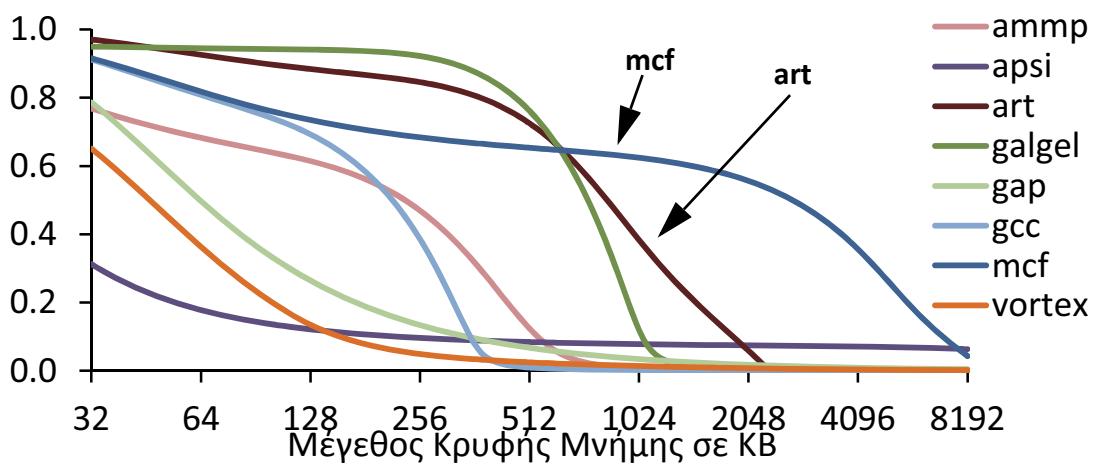
Η ίδια συγγραφική ομάδα επέκτεινε την προηγούμενη εργασία προς την κατεύθυνση της μεγιστοποίησης της συνολικής απόδοσης του συστήματος [18]. Με χρήση και πάλι μετρητών stack distances, βρίσκεται η συμπεριφορά που θα είχαν οι εφαρμογές αν είχαν υπό τον έλεγχο τους ολόκληρη την κρυφή μνήμη και εξάγεται, για κάθε πιθανό διαμοιρασμό της κρυφής μνήμης, η αύξηση στο ποσοστό αστοχίας που οφείλεται στον διαμοιρασμό. Με βάση αυτήν την πληροφορία διαμοιράζεται η κρυφή μνήμη ώστε να ελαχιστοποιηθεί η συνολική αύξηση στο ποσοστό αστοχιών. Μία άλλη αλλά παρόμοια τεχνική αναπτύχθηκε και από τους Suh et al. [84]. Τέλος, οι Yeh και Reiman εξέτασαν το πρόβλημα του δίκαιου διαμοιρασμού σε φυσικά κατανεμημένες NUCA κρυφές μνήμες, οπού η έννοια της δικαιοσύνης δεν καλύπτει μόνο τις αστοχίες που προκαλεί η κάθε εφαρμογή αλλά και την διάκριση ανάμεσα σε αργές και γρήγορες ευστοχίες [95].

Γενικότερα, οι περισσότερες μεθοδολογίες που έχουν προταθεί για διαχείριση του διαμοιρασμού κοινόχρηστων κρυφών μνημάτων εξετάζουν την συμπεριφορά της κάθε εφαρμογής απομονωμένα από την αλληλεπίδρασή της με τις υπόλοιπες εφαρμογές που

μοιράζονται την κρυφή μνήμη και προσπαθούν να προβλέψουν έπειτα τις συνέπειες του διαμοιρασμού. Αντίθετα, η μεθοδολογία StatShare συλλαμβάνει και την συμπεριφορά των εφαρμογών και την αλληλεπίδρασή τους και χάρις στην απλή μοντελοποίηση του διαμοιρασμού μπορεί να οδηγήσει τις αποφάσεις ενός ευέλικτου μηχανισμού διαχείρισης.

3.2.2 Η Μεθοδολογία StatCache

Το πως ακριβώς λειτουργεί η μεθοδολογία StatCache παρουσιάστηκε και μελετήθηκε στην Ενότητα 2.3. Συνοπτικά, μηχανισμοί συλλογής reuse distances παρακολουθούν την ροή των προσπελάσεων προς την κρυφή μνήμη, δειγματοληπτούν ένα μικρό κομμάτι αυτών των προσπελάσεων ($\sim 10^{-4}$), υπολογίζουν τα reuse distances των δειγμάτων και ομαδοποιούν αυτήν την πληροφορία σε ένα ιστόγραμμα που δείχνει την συχνότητα εμφάνισης κάθε reuse distance. Μετά την εκτέλεση του προγράμματος, το ιστόγραμμα τροφοδοτείται σε ένα στατιστικό μοντέλο που συσχετίζει κάθε reuse distance είτε με την πιθανότητα παραγωγής αστοχίας (για random πολιτικές αντικατάστασης), είτε με το αναμενόμενο stack distance της προσέλασης (για LRU πολιτικές αντικατάστασης). Τέλος, βάσει αυτής της συσχέτισης υπολογίζεται το ποσοστό αστοχίας του προγράμματος για κάθε πιθανό μέγεθος κρυφής μνήμης.



Σχήμα 7: Καμπύλες StatCache

Στο Σχήμα 7 βλέπουμε τις καμπύλες που παράγει η StatCache μεθοδολογία για μία σειρά από προγράμματα της σουίτας SPEC2000. Αυτό το σχήμα αποτελεί το κίνητρο της μεθοδολογίας που παρουσιάζεται στην συνέχεια. Όπως φαίνεται, ορισμένα προγράμματα παρουσιάζουν σχεδόν μηδενική κλίση στις καμπύλες τους για μία περιοχή μεγεθών κρυφής μνήμης, όπως για παράδειγμα το mcf στην περιοχή 256KB-1MB. Σε αυτές τις περιπτώσεις,

αυτό σημαίνει ότι η εφαρμογή μπορεί να περιοριστεί σε μικρό κομμάτι της κρυφής μνήμης, με ελάχιστη αλλαγή στο ποσοστό αστοχίας της. Αντίστροφα, αν το πρόγραμμα βρίσκεται σε περιοχή όπου η καμπύλη παρουσιάζει μεγάλη κλίση, όπως το art στην περιοχή 512KB-2MB, τότε το πρόγραμμα, ακόμη και με μικρές αυξήσεις του μεγέθους της κρυφής μνήμης, θα μείωνε σημαντικά το ποσοστό αστοχίας του. Σε κοινόχρηστες κρυφές μνήμες, τέτοια πληροφορία θα μπορούσε να διευκολύνει πολύ την διαχείριση: θα αρκούσε να περιοριστούν όσες εφαρμογές βρίσκονται στην επίπεδη περιοχή των καμπυλών StatCache και ο χώρος που απελευθερώνεται να αποδοθεί σε εφαρμογές που βρίσκονται σε περιοχή με κλίση.

3.3 Το στατιστικό μοντέλο StatShare

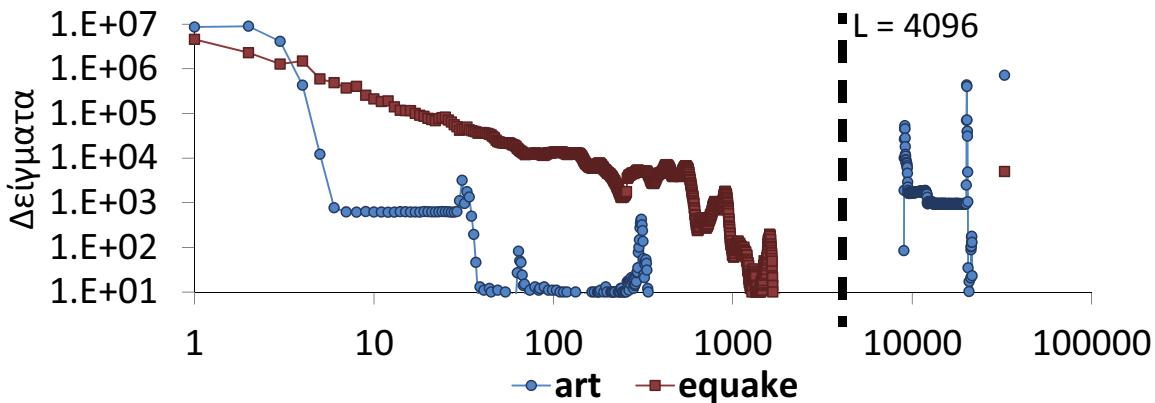
Όπως αναφέρθηκε και στην εισαγωγή, το μοντέλο StatShare εμπνεύστηκε από το αντίστοιχο μοντέλο StatCache, και χαρακτηρίζεται από τους ίδιους περιορισμούς και απλοποιήσεις όπως και το StatCache, με κυριότερη απλοποίηση το ότι η κρυφή μνήμη που περιγράφεται είναι πλήρως συσχετιστική με random πολιτική αντικατάστασης. Η παραδοχή μίας πλήρως συσχετιστικής κρυφής μνήμης είναι αρκετά καλή προσέγγιση για σχετικά μεγάλες κρυφές μνήμες με μέτριο ή μεγάλο associativity, όπως είναι και οι κρυφές μνήμες που μελετάμε [32]. Αυτό επιβεβαιώνεται και από την μελέτη των αποτελεσμάτων της μεθοδολογίας StatShare, τα οποία δείχνουν ότι μπορούμε να περιγράψουμε με μεγάλη επιτυχία κρυφές μνήμες των 8 ways.

3.3.1 Χρόνος σε CAT

Γενικότερα, το reuse distance μίας προσπέλασης μετριέται σε μονάδες γεγονότων που μεσολαβούν ανάμεσα στις δύο προσπελάσεις που ορίζουν το reuse distance. Ενώ στην StatCache τα γεγονότα αυτά είναι οι προσπελάσεις κρυφής μνήμης, στην μεθοδολογία StatShare χρησιμοποιούμε μία διαφορετική έννοια “χρόνου”, τα Cache Allocation Ticks (CAT). Τα Cache Allocation Ticks δεν είναι τίποτε άλλο, παρά οι αντικαταστάσεις γραμμών στην κρυφή μνήμη και επιλέχθηκαν σαν μονάδα χρόνου στην τρέχουσα μεθοδολογία λόγω της άμεσης σύνδεσης που παρέχουν ανάμεσα στον χρόνο και στον χώρο της κρυφής μνήμης: μία μονάδα χρόνου (ένα CAT) ισοδυναμεί με την αντικατάσταση μίας γραμμής από την κρυφή μνήμη και την δέσμευση του χώρου για μία καινούργια γραμμή.

3.3.2 Ιστογράμματα των CAT reuse distances

Όπως εξηγήθηκε ήδη, το ιστόγραμμα δείχνει την κατανομή των reuse distances μίας εφαρμογής. Στην μεθοδολογία StatShare, το ιστόγραμμα συμβολίζεται ως $h(i)$ για $i = (0, \infty)$. Το Σχήμα 8 δείχνει τα ιστογράμματα για δύο προγράμματα της συνίτας SPEC2000, το art και το equake, όταν μοιράζονται μία κρυφή μνήμη των 256KB. Τα ιστογράμματα αντιστοιχούν σε παράθυρο 200M εντολών.

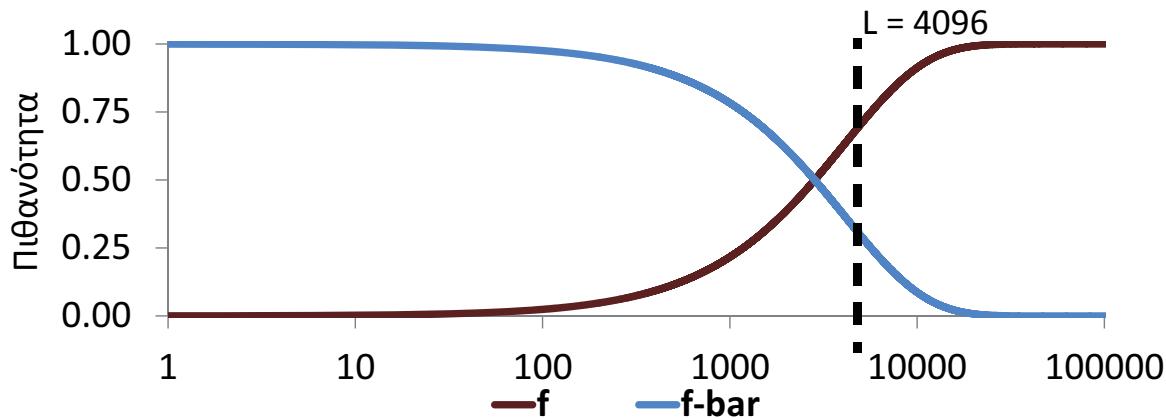


Σχήμα 8: Ιστογράμματα $h(i)$, για το art και το equake (λογαριθμικοί άξονες)

Το art παρουσιάζει μία δυαδική κατανομή: πάρα πολλά δείγματα έχουν χαμηλό reuse distance (<30), αλλά υπάρχει απομονωμένη και μία μεγάλη ομάδα δειγμάτων με μεγάλα reuse distances (ανάμεσα στα 9K και στα 22K). Αυτή η ομάδα δειγμάτων βρίσκεται αρκετά πάνω από το μέγεθος της κρυφής μνήμης (4K μετρημένο σε γραμμές), οπότε αναμένεται ότι πολλές προσπελάσεις του art δεν θα χωράνε στην κρυφή μνήμη και θα προκαλούν αστοχίες. Όπως θα φανεί και στην συνέχεια του κεφαλαίου, αυτές οι προσπελάσεις με υψηλά reuse distances είναι υπεύθυνες για την συμπεριφορά του art που έχει την τάση να μονοπωλεί τον έλεγχο της κρυφής μνήμης. Αντίθετα το equake έχει μία πιο ομαλή κατανομή reuse distances που μειώνεται με σχετικά σταθερό ρυθμό μέχρι το 2K, οπότε και μηδενίζεται. Αυτή η κατανομή σημαίνει ότι το equake ήδη είναι περιορισμένο στην κρυφή μνήμη, παράγει ευστοχίες από τις περισσότερες γραμμές που έχει υπό τον έλεγχο του και δεν μπορούμε να το περιορίσουμε περαιτέρω χωρίς σημαντική αύξηση του ποσοστού αστοχιών του. Γενικότερα, τα περισσότερα προγράμματα συμπεριφέρονται όπως το art ή το equake. Προγράμματα παρόμοια με το art είναι υποψήφια για διαχείριση και περιορισμό, ενώ προγράμματα όπως το equake είναι αυτά στα οποία αποδίδεται ο απελευθερωμένος χώρος.

3.3.3 Βασικές εξισώσεις

Κεντρικό ρόλο στο μοντέλο StatShare παίζουν οι εξισώσεις f και \bar{f} (Σχήμα 9). Οι δύο αυτές συναρτήσεις δίνουν την πιθανότητα μία προσπέλαση κρυφής μνήμης να καταλήξει σε ευστοχία (\bar{f}) ή αστοχία (f) για ένα δεδομένο reuse distance. Οι f -εξισώσεις είναι στενά δεμένες με την πολιτική αντικατάστασης της κρυφής μνήμης, καθώς αντικατοπτρίζουν τον τρόπο με τον οποίο η πολιτική αντικατάστασης εκμεταλλεύεται την τοπικότητα των προσπελάσεων. Σε αυτό το κομμάτι της μεθοδολογίας StatShare, η έμφαση είναι στην random πολιτική αντικατάστασης, επειδή η στατιστική φύση και η έλλειψη μνήμης της random πολιτικής επιτρέπουν απλές αναπαραστάσεις των εξισώσεων f .



Σχήμα 9: f και \bar{f} εξισώσεις για Random πολιτική αντικατάστασης και πλήρως συσχετιστική κρυφή μνήμη

Η λογική με την οποία εξάγονται οι εξισώσεις f είναι η εξής:

- Σε μία πλήρως συσχετιστική κρυφή μνήμη, με random πολιτική αντικατάστασης και μέγεθος L (μετρημένο σε γραμμές), μία γραμμή έχει πιθανότητα να αντικατασταθεί σε περίπτωση αστοχίας.
- Ισοδύναμα, η πιθανότητα να παραμείνει στην κρυφή μνήμη είναι $(1-1/L)$.
- Μετά από i αντικαταστάσεις, η πιθανότητα η γραμμή να παραμένει στην κρυφή μνήμη είναι $(1-1/L)^i$.
- Ισοδύναμα, η πιθανότητα να έχει ήδη αντικατασταθεί είναι $1-(1-1/L)^i$.

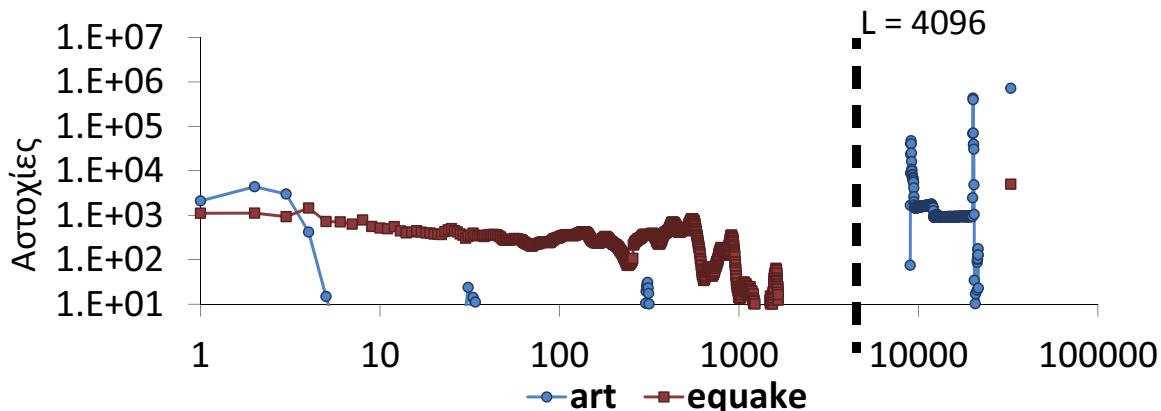
Ορίζουμε λοιπόν την πιθανότητα παραγωγής αστοχίας (f) και την πιθανότητα παραγωγής ευστοχίας (\bar{f}) για μία προσπέλαση με CAT reuse distance i ως:

$$f(i) = 1 - \left(1 - \frac{1}{L}\right)^i \quad \overline{f(i)} = 1 - f(i) = \left(1 - \frac{1}{L}\right)^i$$

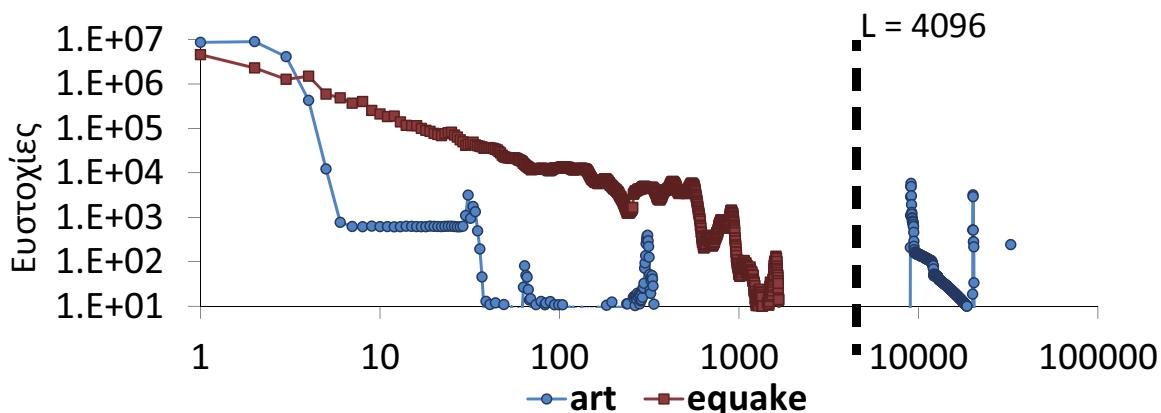
Στο επόμενο βήμα, για να υπολογιστούν οι ευστοχίες και οι αστοχίες ενός προγράμματος αρκεί να πολλαπλασιαστεί το ιστόγραμμα του με την \overline{f} και την f αντίστοιχα (Σχήμα 10 και Σχήμα 11):

$$hits = \sum_{i=0}^{\infty} h(i) \times \overline{f(i)} \quad misses = \sum_{i=0}^{\infty} h(i) \times f(i)$$

Όπως θα αποδειχθεί και στην συνέχεια του κεφαλαίου, τα αποτελέσματα των εξισώσεων συμφωνούν σε σημαντικό βαθμό με τα αποτελέσματα των εξομοιώσεων. Για να αυξήσουμε όμως ακόμη περισσότερο την ακρίβεια των προβλέψεων του μοντέλου, πρέπει να ενσωματώσουμε και τις αστοχίες που προκαλούνται κατά την πρώτη προσπέλαση κάθε γραμμής και οι οποίες προφανώς δεν σχετίζονται με κάποιο reuse distance. Ακολουθώντας την ίδια λογική με την εργασία [10], εκτιμούμε τον αριθμό αυτών των αστοχιών βάσει του αριθμού των δειγμάτων για τα οποία δεν έχουμε reuse distance όταν τελειώσει η μέτρηση.



Σχήμα 10: Ιστογράμματα αστοχιών πολλαπλασιάζοντας το $h(i)$ με την $f(i)$



Σχήμα 11: Ιστογράμματα ευστοχιών πολλαπλασιάζοντας το $h(i)$ με την $\overline{f}(i)$

3.4 Spacetime

Το spacetime δίνει μία έννοια του μέσου χώρου που καταλαμβάνει μία εφαρμογή στην διάρκεια μία μέτρησης και ορίζεται ως το άθροισμα του χώρου που καταλαμβάνει κάθε γραμμή επί την διάρκεια ζωής της γραμμής (με τον χρόνο μετρημένο σε CAT). Με βάση αυτόν τον ορισμό, το μέσο ποσοστό του χώρου που καταλαμβάνει μία εφαρμογή θα είναι το spacetime της διαιρεμένο με το συνολικό spacetime ($L \times$ διάρκεια μέτρησης σε CAT).

3.4.1 Spacetime μίας εφαρμογής

Η αναμενόμενη διάρκεια ζωής μίας γραμμής ανάμεσα σε δύο προσπελάσεις της με reuse distance i , είναι ίση με το άθροισμα των πιθανοτήτων του να βρίσκεται ακόμη στην κρυφή μνήμη για κάθε ένα από i CAT που θα συμβούν μέχρι να προσπελαστεί ξανά, δηλαδή:

$$l(i) = \sum_{k=1}^i P(\text{in cache during tick } k)$$

Εφόσον η πιθανότητα η γραμμή να βρίσκεται ακόμη στην κρυφή μνήμη κατά το k tick είναι η πιθανότητα να μην έχει αντικατασταθεί κατά τα προηγούμενα $k-1$ tick, δηλαδή $\overline{f(k-1)}$, η αναμενόμενη διάρκεια ζωής της γραμμής είναι:

$$l(i) = \sum_{k=1}^i \overline{f(k-1)} = \sum_{k=1}^i \left(1 - \frac{1}{L}\right)^i = L \times f(i)$$

Επομένως, το συνολικό spacetime της εφαρμογής θα είναι το άθροισμα των spacetimes όλων των δειγμάτων και θα είναι:

$$S = \sum_{i=0}^{\infty} l(i) \times h(i) = \sum_{i=0}^{\infty} L \times f(i) \times h(i) = L \times misses$$

και το ποσοστό του χώρου που καταλαμβάνει η εφαρμογή θα είναι:

$$MeanSpace = \frac{S_{application}}{S_{total}} = \frac{L \times misses_{application}}{L \times misses_{total}} = \frac{misses_{application}}{misses_{total}}$$

Δηλαδή, το spacetime που καταλαμβάνει μία εφαρμογή είναι ίσο με τον αριθμό των αστοχιών που παράγει επί το μέγεθος της κρυφής μνήμης και το ποσοστό του χώρου που καταλαμβάνει είναι το ποσοστό των αστοχιών που παράγει επί των συνολικών αστοχιών. Η

σημασία αυτής της εξίσωσης είναι διπλή. Όχι μόνο συνδέει άμεσα τον χώρο που καταλαμβάνει κάθε εφαρμογή με τον αριθμό των αστοχιών της, αλλά επιπλέον μας δείχνει το πρόβλημα που δημιουργεί η έλλειψη διαχείρισης του διαμοιρασμού: όσο λιγότερο μπορεί να αξιοποιήσει τον χώρο της κρυφής μνήμης μία εφαρμογή, τόσο περισσότερο χώρο καταλαμβάνει.

3.4.2 Spacetime των ευστοχιών

Σε περίπτωση ευστοχίας, η γραμμή καταλαμβάνει τον χώρο μίας γραμμής για ολόκληρη την διάρκεια του reuse distance, επομένως ο χώρος που καταλαμβάνουν οι ευστοχίες είναι:

$$S_{hits} = \sum_{i=0}^{\infty} i \times h(i) \times f(i)$$

Το spacetime των ευστοχιών είναι *χρήσιμο*: η γραμμή καταλαμβάνει χώρο, αλλά μας προσφέρει το όφελος μίας ευστοχίας. Για πολύ μεγάλα reuse distances όμως είναι πιθανό ο χώρος που καταλαμβάνει μία τέτοια γραμμή να γίνει υπερβολικά μεγάλος και να είναι πιο προσδοφόρο να την αποσύρουμε από την κρυφή μνήμη και να χρησιμοποιήσουμε τον απελευθερωμένο χώρο πιο αποδοτικά.

3.4.3 Spacetime των αστοχιών

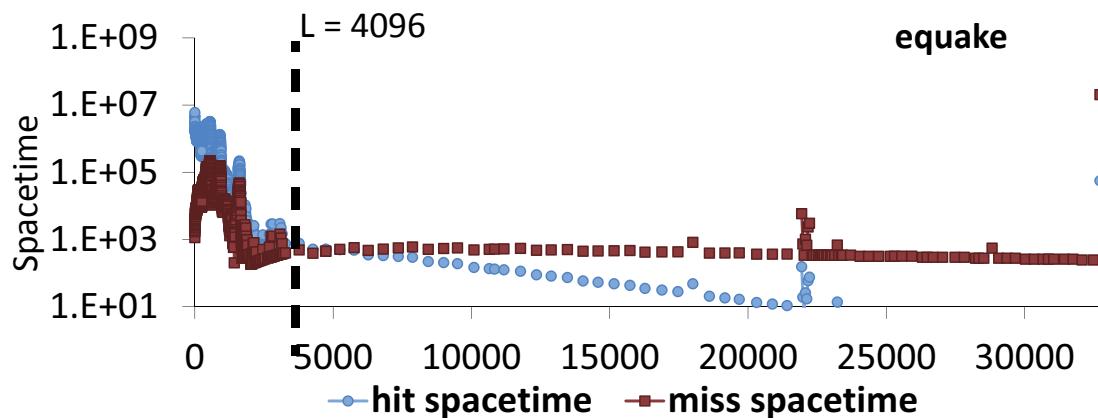
Με βάση τις εξισώσεις για το συνολικό spacetime και το spacetime των ευστοχιών, το spacetime των αστοχιών είναι:

$$S_{misses} = \sum_{i=0}^{\infty} L \times f(i) \times h(i) - \sum_{i=0}^{\infty} i \times f(i) \times h(i) = \sum_{i=0}^{\infty} [(L + i) \times f(i) - i] \times h(i)$$

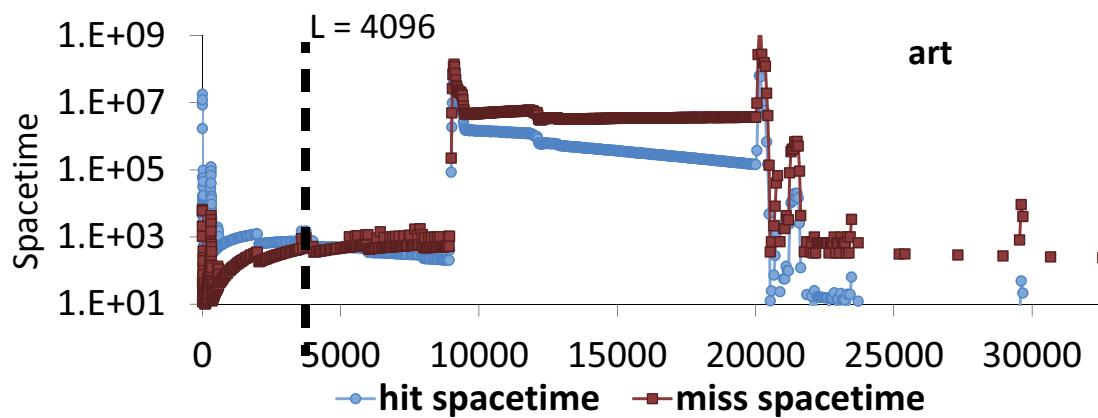
Το spacetime των αστοχιών είναι *άχρηστο* spacetime: η γραμμή καταλαμβάνει τον χώρο της χωρίς να προσφέρει στην αύξηση του ποσοστού ευστοχίας. Όμως, παρότι είναι άχρηστο, δεν μπορούμε να το αποφύγουμε τελείως —δεν ξέρουμε εκ των προτέρων ποιες γραμμές θα προσπελαστούν ξανά και ποιες θα αντικατασταθούν.

Στο Σχήμα 12 και στο Σχήμα 13 παρουσιάζεται το spacetime των αστοχιών και των ευστοχιών για το equake και το art αντίστοιχα. Και στα δύο γραφήματα ο χάξονας είναι σε γραμμική κλίμακα ώστε να φαίνεται καθαρά η συμμετοχή όλων των reuse distances στο συνολικό spacetime, το οποίο ισούται με την περιοχή κάτω από τις καμπύλες. Φαίνεται εύκολα από τα γραφήματα ότι το art καταλαμβάνει πολύ περισσότερο άχρηστο χώρο από

ότι χρήσιμο. Το equake καταλαμβάνει και αυτό αρκετό άχρηστο χώρο, αλλά είναι συγκριτικά λιγότερος από τον χρήσιμο χώρο του. Η πιο ενδιαφέρουσα πληροφορία που προκύπτει από τα δύο αυτά γραφήματα είναι ότι η ομάδα δειγμάτων του art με υψηλά reuse distances (9K-22K) καταλαμβάνει το μεγαλύτερο κομμάτι της κοινόχρηστης κρυφής μνήμης, ενώ όπως είδαμε πριν συνεισφέρει πολύ λίγο στο συνολικό ποσοστό ευστοχιών. Η τεχνική διαχείρισης που παρουσιάζεται στις επόμενες ενότητες έχει σαν στόχο ακριβώς αυτές τις περιπτώσεις, να απελευθερώσει δηλαδή τον υπερβολικά μεγάλο χώρο που καταλαμβάνουν γραμμές με χαμηλή πιθανότητα ευστοχίας.



Σχήμα 12: Spacetime των ευστοχιών και των αστοχιών για το equake (ο χάρτης είναι γραμμικός)



Σχήμα 13: Spacetime των ευστοχιών και των αστοχιών για το art (ο χάρτης είναι γραμμικός)

3.4.4 Επιπτώσεις του Decay στο Spacetime

Για να γίνουν πιο κατανοητοί οι στόχοι της τεχνικής διαχείρισης που εμπεριέχεται στην μεθοδολογία StatShare, ας εξετάσουμε το σενάριο όπου εφαρμόζουμε Decay με Decay Time ίσο με D CAT σε μία από τις εφαρμογές που μοιράζονται την κρυφή μνήμη. Σε αυτή την περίπτωση, με κάθε γραμμή που σημειώνεται decayed, το Decay απελευθερώνει spacetime

από την εφαρμογή. Ο χώρος αυτός γίνεται μετά διαθέσιμος για χρήση από τις υπόλοιπες εφαρμογές. Στην χειρότερη περίπτωση, όλες οι decayed γραμμές θα περάσουν στον έλεγχο των άλλων εφαρμογών, οπότε τα hits που θα γίνονταν κανονικά σε αυτές τις γραμμές μετατρέπονται σε decay-induced misses (DIM):

$$DIM = \sum_{i=D}^{\infty} h(i) \times \overline{f(i)}$$

Με βάση τις εξισώσεις που παρουσιάσαμε παραπάνω, αν εφαρμόσουμε Decay σε μία κρυφή μνήμη 256KB που μοιράζεται ανάμεσα στο art και στο equake, μπορούμε να συμπεράνουμε ότι:

- Το art μπορεί να περιοριστεί με την χρήση Decay εύκολα και με ελάχιστες συνέπειες, καθώς ο αριθμός των DIM παραμένει χαμηλός ακόμη και για πολύ μικρά D . Αν όμως εξετάζαμε το ίδιο σενάριο για μεγαλύτερη κρυφή μνήμη, τότε το art θα άρχιζε να χωράει, οπότε η εφαρμογή Decay θα αύξανε σημαντικά τις αστοχίες του.
- Το equake δεν μπορεί να περιοριστεί αποδοτικά με κανένα Decay Time D , καθώς παράγει πολλές ευστοχίες κρυφής μνήμης για ένα μεγάλο εύρος reuse distances, ευστοχίες που θα μετατραπούν σε DIM αν χρησιμοποιηθεί Decay.

Σε αυτά τα σενάρια εφαρμογής Decay, απελευθερώνεται spacetime που σχετίζεται και με ευστοχίες και με αστοχίες κρυφής μνήμης. Το spacetime των αστοχιών είναι spacetime που δεν μπορούσε να το αξιοποιήσει η decayed εφαρμογή, οπότε παίρνοντάς το και μοιράζοντάς το στις υπόλοιπες εφαρμογές που ίσως να μπορούν να το αξιοποιήσουν, σίγουρα βελτιώνουμε την συνολική αξιοποίηση της κρυφής μνήμης. Το spacetime των ευστοχιών όμως ήταν χρήσιμος χώρος και η απελευθέρωσή του μπορεί να έχει αρνητικές συνέπειες για ολόκληρο το σύστημα, ειδικά αν οι υπόλοιπες εφαρμογές δεν μπορούν να το αξιοποιήσουν.

3.5 Ενσωμάτωση του Decay στο Θεωρητικό Μοντέλο

Εφόσον ο στόχος της μεθοδολογίας που παρουσιάζεται σε αυτό το κεφάλαιο είναι η παραγωγή ενός θεωρητικού μοντέλου που να μπορεί να οδηγήσει τις αποφάσεις ενός μηχανισμού διαχείρισης που βασίζεται στο decay, θα πρέπει να ενσωματωθεί αυτός ο μηχανισμός στο θεωρητικό μοντέλο. Η σημαντικότερη αλλαγή που προκαλεί το CAT Decay στο μοντέλο μας είναι ότι η πολιτική αντικατάστασης δεν είναι πια πραγματικά Ran-

dom: οι decayed γραμμές αντικαθιστώνται πρώτα και μόνο αν δεν βρεθεί τέτοια γραμμή, διαλέγουμε μία τυχαία γραμμή. Στην συνέχεια της ενότητας παρουσιάζουμε τις επιπτώσεις του CAT Decay στις εξισώσεις του μοντέλου.

3.5.1 Decayed εξισώσεις f

Ένας εύκολος τρόπος να ποσοτικοποιηθεί η επίδραση του CAT Decay στην κρυφή μνήμη είναι η διάκριση των αντικαταστάσεων σε “Murphy” και ελεγχόμενες αντικαταστάσεις. Οι Murphy αντικαταστάσεις είναι αυτές που ξεφεύγουν από την προσπάθειά μας να τις διαχειριστούμε και πιο συγκεκριμένα είναι αυτές όπου δεν βρίσκουμε καμία decayed γραμμή να αντικαταστήσουμε και καταφεύγουμε σε Random πολιτική αντικατάστασης. Τα πειράματα μας έδειξαν ότι είναι πρακτικά αδύνατο να εξαλειφθούν οι Murphy αντικαταστάσεις: ακόμη και αν εφαρμόσουμε πολύ χαμηλό Decay Time, ο αριθμός decayed γραμμών θα είναι υψηλός κατά μέσο όρο, αλλά περιστασιακά θα μηδενίζεται.

Τα παραπάνω σημαίνουν, ότι η πιθανότητα μ μία αντικατάσταση να καταλήξει σε Murphy αντικατάσταση μπορεί να κυμαίνεται σημαντικά στην διάρκεια του χρόνου. Για λόγους απλότητας όμως θα θεωρήσουμε στην υπόλοιπη ανάλυση μία μέση τιμή για την μ σε κάθε παράθυρο μέτρησης:

$$\mu = \frac{Replacements_{Murphy}}{Replacements_{total}}$$

Για τις εφαρμογές που δεν περιορίζονται από το CAT Decay, η πιθανότητα μία γραμμή της εφαρμογής να αντικατασταθεί μετά από μία αστοχία είναι η πιθανότητα να μην βρεθεί decayed γραμμή (δηλαδή μ) επί την πιθανότητα να επιλεχθεί από την Random πολιτική αντικατάστασης αυτή η γραμμή (δηλαδή $1/L$). Οπότε η πιθανότητα επιλογής για αντικατάσταση είναι μ/L και η πιθανότητα μη αντικατάστασης είναι $(1 - \mu/L)$ και επομένως οι εξισώσεις f για εφαρμογές που δεν περιορίζονται από το decay θα είναι:

$$f_{nd}(i) = 1 - \left(1 - \frac{\mu}{L}\right)^i \quad \overline{f_{nd}(i)} = \left(1 - \frac{\mu}{L}\right)^i$$

Η σημασία αυτών των εξισώσεων είναι σημαντική: για τις εφαρμογές που εκμεταλλεύονται τον χώρο που απελευθερώνει το Decay, η συμπεριφορά της κρυφής μνήμης είναι πανομοιότυπη με αυτήν μίας μη-διαχειριζόμενης κρυφής μνήμης μεγέθους L/μ .

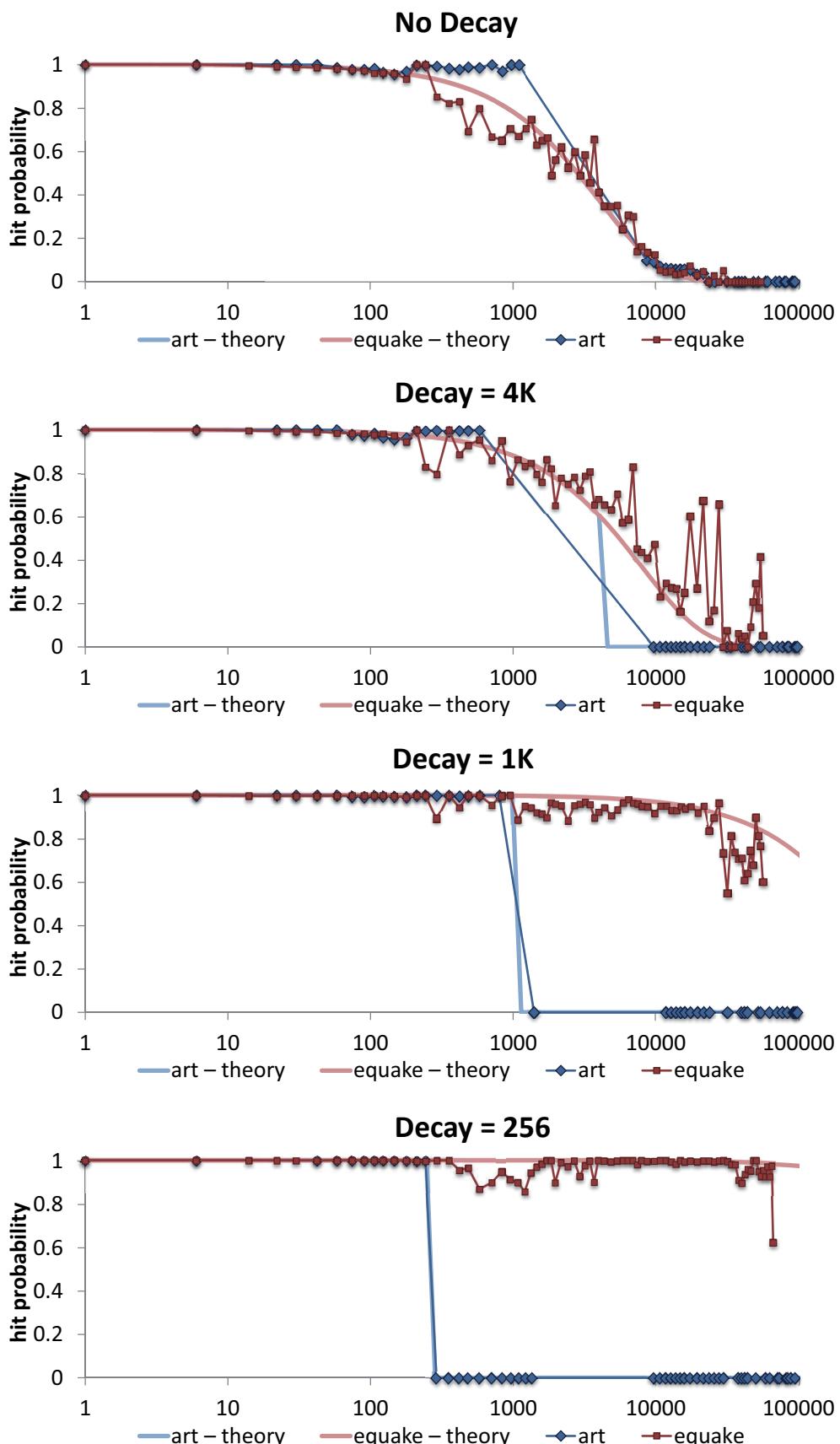
Για τις εφαρμογές που τις περιορίζουμε με CAT Decay, οι καινούργιες εξισώσεις f είναι παρόμοιες. Για reuse distances μικρότερα ή ίσα του Decay Time D, η συμπεριφορά είναι ακριβώς ή ίδια με αυτήν των reuse distances των εφαρμογών χωρίς decay. Για reuse distances όμως μεγαλύτερα του D, η γραμμή είτε θα έχει αντικατασταθεί πριν το D ή θα γίνει decayed, οπότε θεωρούμε ότι η πιθανότητα να παραχθεί αστοχία από αυτό το reuse distance είναι 100%. Συνολικότερα ισχύει:

$$f_d(i) = \begin{cases} 1 - \left(1 - \frac{\mu}{L}\right)^i, & (i \leq D) \\ 1, & (i > D) \end{cases} \quad \overline{f_d(i)} = \begin{cases} \left(1 - \frac{\mu}{L}\right)^i, & (i \leq D) \\ 0, & (i > D) \end{cases}$$

με το D για τις εφαρμογές που δεν περιορίζονται από Decay να θεωρείται άπειρο.

Αυτή η προσέγγιση των εξισώσεων f αποδεικνύεται από την πειραματική μελέτη μας ότι περιγράφει ικανοποιητικά την συμπεριφορά των εφαρμογών, παρά τις υποθέσεις που αναγκαστήκαμε να κάνουμε και τις διαφορές ανάμεσα στην κρυφή μνήμη που περιγράφει το μοντέλο και την κρυφή μνήμη που εξομοιώσαμε. Όπως θα φανεί στην συνέχεια, σημαντικά σφάλματα υπάρχουν μόνο όταν υπάρχει υπερπληθώρα decayed γραμμών. Όταν συμβαίνει αυτό, οι decayed γραμμές θα περάσουν αρκετό χρόνο στην κρυφή μνήμη πριν αντικατασταθούν και επομένως υπάρχει μη αμελητέα πιθανότητα να προσπελαστούν πριν την αντικατάστασή τους, σε αντίθεση με την υπόθεσή μας ότι δεν γίνονται εύστοχες προσπελάσεις σε decayed γραμμές.

Για να μελετήσουμε την ακρίβεια των εξισώσεων f στην περιγραφή της κρυφής μνήμης, συλλέξαμε δεδομένα από εξομοιώσεις κρυφών μνημάτων, υπολογίσαμε την πραγματική \bar{f} καμπύλη (διαιρώντας το πειραματικά εξακριβωμένο ιστόγραμμα των ευστοχιών με το ιστόγραμμα όλων των προσπελάσεων) και την συγκρίναμε με την θεωρητικά προβλεπόμενη \bar{f} . Στο Σχήμα 14 παρουσιάζεται μία τέτοια σύγκριση για την περίπτωση που το art και το equake μοιράζονται μία κρυφή μνήμη των 256KB και εφαρμόζουμε σταδιακά ολοένα και χαμηλότερο Decay Time στο art. Το παράδειγμα αυτό δείχνει την επιτυχία του μοντέλου: παρά τις περιστασιακές αποκλίσεις ανάμεσα στις θεωρητικές και πραγματικές \bar{f} , το θεωρητικό μοντέλο είναι αρκετά κοντά στην πραγματικότητα και για τα περισσότερα reuse distances αλλά και για όλες τις τιμές του Decay Time.



Σχήμα 14: Πειραματικές και θεωρητικές καμπύλες \bar{f} . για το art και το earthquake σε 256KB κρυφή μνήμη, με μειούμενα Decay Times να εφαρμόζονται στο art

3.5.2 Decayed εξισώσεις του spacetime

Για τις εφαρμογές που δεν περιορίζονται από το decay, οι εξισώσεις του spacetime παραμένουν οι ίδιες, με μόνη αλλαγή την αντικατάσταση του L από το L/μ και των εξισώσεων f με τις αντίστοιχες για decay:

$$S = \sum_{i=0}^{\infty} \frac{L}{\mu} \times f_d(i) \times h(i)$$

$$S_{hits} = \sum_{i=0}^{\infty} i \times \bar{f}_d(i) \times h(i)$$

$$S_{misses} = \sum_{i=0}^{\infty} \left[\left(\frac{L}{\mu} + i \right) \times f_d(i) - i \right] \times h(i)$$

Για την εφαρμογή που περιορίζεται όμως, αυτές οι εξισώσεις δεν ισχύουν. Σε αυτή την περίπτωση, όσες γραμμές έχουν μείνει στην κρυφή μνήμη αχρησιμοποίητες για D CAT γίνονται decay και όσον αφορά το μοντέλο δεν πρόκειται να δώσουν ευστοχίες. Επομένως, η αναμενόμενη διάρκεια ζωής των γραμμών της decayed εφαρμογής είναι:

$$l(i) = \sum_{k=1}^i \left(1 - \frac{\mu}{L} \right)^{k-1} = \frac{L}{\mu} \times f_d(i) \quad , i \leq D$$

$$l(i) = \sum_{k=1}^D \left(1 - \frac{\mu}{L} \right)^{k-1} = \frac{L}{\mu} \times f_d(D) \quad , i > D$$

και άρα οι εξισώσεις των spacetimes θα είναι:

$$S = \sum_{i=0}^D \frac{L}{\mu} \times f_d(i) \times h(i) + \sum_{i=D+1}^{\infty} \frac{L}{\mu} \times f_d(D) \times h(i)$$

$$S_{hits} = \sum_{i=0}^D i \times \bar{f}_d(i) \times h(i)$$

$$S_{misses} = \sum_{i=0}^D \left[\left(\frac{L}{\mu} + i \right) \times f_d(i) - i \right] \times h(i) + \sum_{i=D+1}^{\infty} \frac{L}{\mu} \times f_d(D) \times h(i)$$

3.6 Υλοποίηση

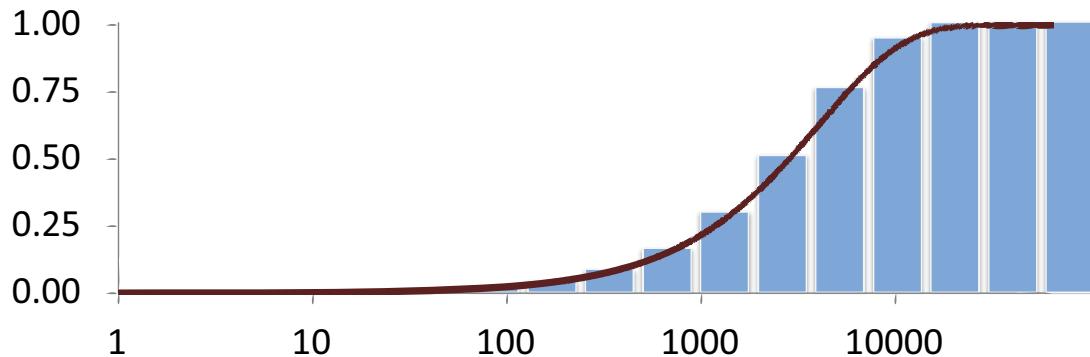
3.6.1 Συλλογή Ιστογραμμάτων

Τα ιστογράμματα των reuse distances έτσι όπως έχουν περιγραφεί μέχρι τώρα, με εύρος τιμών από το 0 μέχρι το άπειρο, είναι προφανώς ακατάλληλα για συλλογή και επεξεργασία στην διάρκεια εκτέλεσης. Για να διευκολύνουμε την χρήση των ιστογραμμάτων και να απλοποιήσουμε τους μηχανισμούς συλλογής χρησιμοποιούμε δύο τεχνικές: δειγματοληψία και κβάντιση.

Με την δειγματοληψία, αντί να συλλέγουμε reuse distances για όλες τις προσπελάσεις της κρυφής μνήμης, επιλέγουμε ένα μικρό μόνο ποσοστό των προσπελάσεων και παρακολουθούμε τις γραμμές που προσπέλασαν μέχρις ότου αυτές επαναχρησιμοποιηθούν, οπότε και υπολογίζουμε το reuse distance. Εφόσον η δειγματοληψία γίνεται με στατιστικά τυχαίο τρόπο, το ιστόγραμμα που θα προκύψει θα έχει τις ίδιες στατιστικές ιδιότητες με το πλήρες (μη-δειγματοληφθέν) ιστόγραμμα και θα διαφέρει μόνο στην κλίμακα του γάζονα. Στόχος μας με την δειγματοληψία είναι η απλοποίηση του μηχανισμού συλλογής των ιστογραμμάτων: κάθε χρονική στιγμή ο μηχανισμός θα παρακολουθεί μόνο μία μικρή ομάδα γραμμών, οπότε ένας αντίστοιχα μικρός αριθμός από watchpoint καταχωρητές αρκεί. Επιπλέον, λόγω της φύσης των reuse distances, η πληροφορία που πρέπει να κρατάμε στους watchpoint καταχωρητές είναι μόνο η διεύθυνση της γραμμής και η τιμή του μετρητή CAT της στιγμή της προσπέλασης. Στα αποτελέσματα που παρουσιάζουμε στην συνέχεια, όλες οι μετρήσεις έγιναν δειγματοληπτώντας 1 στις 1024 προσπελάσεις.

Η δεύτερη τεχνική απλοποίησης είναι η κβάντιση των ιστογραμμάτων. Στα παραδείγματα που παρουσιάσαμε στις προηγούμενες ενότητες, τα ιστογράμματα περιέχουν μία τιμή για κάθε reuse distance ανάμεσα στο 0 και στο 512K-1, με τα reuse distances που ξεπερνούν το 512K-1 να συγκεντρώνονται στην τελευταία θέση του ιστογράμματος. Υποθέτοντας 32 bit μετρητή για κάθε θέση του ιστογράμματος, κάθε τέτοιο ιστόγραμμα απαιτεί 2MB αποθηκευτικού χώρου, μέγεθος απαγορευτικό για έναν μηχανισμό που λειτουργεί στην διάρκεια της εκτέλεσης. Η ιδιότητα που εκμεταλλευόμαστε για να μειώσουμε τον χώρο που καταλαμβάνουν τα ιστογράμματα είναι η εκθετική μορφή της εξίσωσης f . Λόγω της εκθετικής συμπεριφοράς της, όσο μεγαλύτερο είναι ένα reuse distance, τόσο λιγότερο στατιστικά σημαντικές είναι οι διαφορές ανάμεσα σε γειτονικά reuse distances. Με βάσει αυτήν την ιδιότητα, ομαδοποιούμε τα reuse distances σε ομάδες με

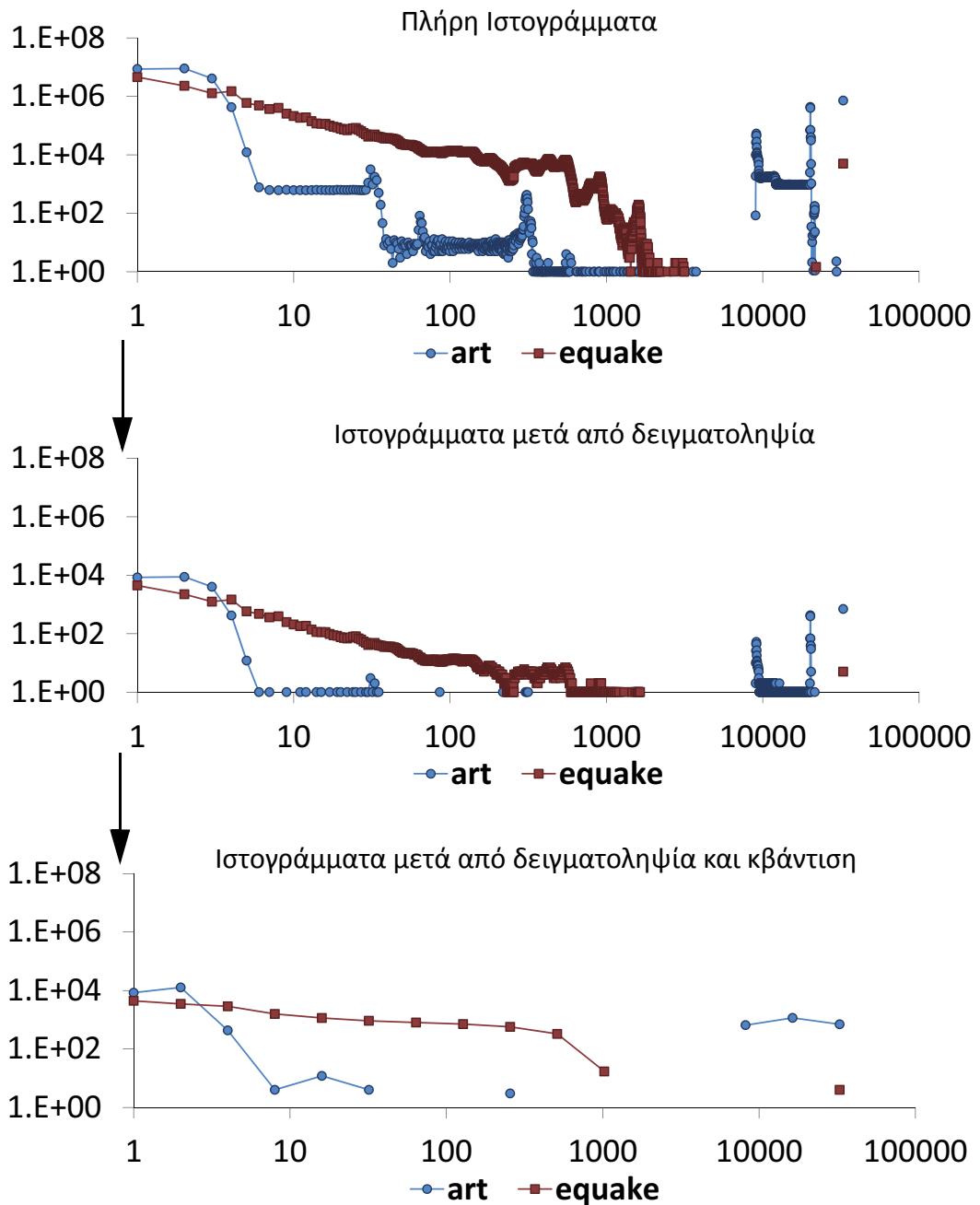
εκθετικά αυξανόμενο εύρος (0, 1-2, 3-6, 7-14, 15-30, 31-62 κοκ). Για να γίνει πιο κατανοητό το γιατί είναι εφικτή η εκθετική κβάντιση των reuse distances, στο Σχήμα 15 φαίνεται η εξίσωση f , μαζί με μπάρες που αντιπροσωπεύουν μία προσέγγιση της f για τα κβαντισμένα εύρη reuse distances.



Σχήμα 15: f καμπύλη και μπάρες που προσεγγίζουν την f για εκθετικά κβαντισμένα reuse distances

Όπως βλέπουμε, αξιόλογες αποκλίσεις ανάμεσα στην κβαντισμένη και την κανονική f υπάρχουν μόνο σε λίγες περιοχές reuse distances και ακόμη και εκεί δεν υπερβαίνουν το 5-10%. Αξίζει να παρατηρήσουμε ότι, αν χρησιμοποιούσαμε γραμμική κβάντιση για τα reuse distances και αντίστοιχο πλήθος ομάδων reuse distances με το προηγούμενο παράδειγμα, το εύρος κάθε ομάδας θα έπρεπε να είναι 3-4K CAT, θα ομαδοποιούσε μαζί όλα τα μικρά reuse distances (που εμφανίζουν πιθανότητα αστοχίας από 0% έως 25%) και οι περισσότερες ομάδες reuse distances θα κάλυπταν τα μεγάλα reuse distances ($> 20K$) όπου υπάρχει ελάχιστη μεταβολή της πιθανότητας αστοχίας ($<1\%$). Τα εκθετικά κβαντισμένα reuse distances παρουσιάζουν και ένα δεύτερο πλεονέκτημα: η περιοχή όπου έχουμε την μεγαλύτερη ακρίβεια στην τιμή του reuse distance και τα μικρότερα σφάλματα στην τιμή της f , είναι και η περιοχή όπου συγκεντρώνεται η πλειοψηφία των δειγμάτων των περισσότερων εφαρμογών, δηλαδή τα πολύ μικρά reuse distances.

Στην υλοποίησή μας χρησιμοποιούμε 20 εκθετικά κβαντισμένες ομάδες reuse distances, που καλύπτουν το εύρος τιμών από 0 έως 512K. Με αυτόν τον τρόπο, το ιστόγραμμα κάθε εφαρμογής μπορεί να συλλεχθεί σε μόλις 20 καταχωρητές των 32 bit (80 bytes). Ένα παράδειγμα του πως μετασχηματίζονται τα ιστογράμματα από την δειγματοληψία και την κβάντιση, βλέπουμε στο Σχήμα 16, όπου παρουσιάζεται το art και το equake όταν μοιράζονται μία 256KB κρυφή μνήμη. Χρησιμοποιώντας τις εξισώσεις του StatShare σε καθένα από τα τρία ιστογράμματα, βρίσκουμε ότι το σφάλμα που προσθέτει η



Σχήμα 16: Ιστογράμματα του art και του equake όταν μοιράζονται μία 256KB κρυφή μνήμη, το πρώτο ιστόγραμμα προκύπτει από συλλογή όλων των reuse distances, το δεύτερο από δειγματοληψία 1:1024, το τρίτο από κβάντιση των reuse distances.

δειγματοληψία στην πρόβλεψη του ποσοστού αστοχιών είναι κάτω από 0.002% ενώ το σφάλμα που προσθέτει η κβάντιση είναι κάτω από 0.06%. Γενικότερα, ελέγχαμε τα αποτελέσματα αυτών των δύο απλοποιήσεων για όλα τα προγράμματα που εξετάζουμε και βρήκαμε ότι οι έξοδοι του StatShare μένουν πρακτικά οι ίδιες είτε το τροφοδοτήσουμε με το πλήρες ιστόγραμμα όλων των προσπελάσεων είτε με το δειγματοληφθέν και κβαντισμένο ιστόγραμμα.

3.6.2 Υλοποίηση του Decay και της Πολιτικής Αντικατάστασης

Για να προσαρμόσουμε την πολιτική αντικατάστασης στους στόχους της διαχείρισης του διαμοιρασμού κάνουμε μία πολύ απλή τροποποίηση: πριν διαλέξουμε τυχαία μία γραμμή για αντικατάσταση, αναζητούμε και αντικαθιστούμε οποιαδήποτε διαθέσιμη decayed γραμμή. Το CAT Decay οδηγείται από μία ομάδα καταχωρήτων, προσβάσιμων από το λειτουργικό σύστημα, στους οποίους αποθηκεύεται το Decay Time κάθε εφαρμογής που συμμετέχει στον διαμοιρασμό της κρυφής μνήμης. Εφαρμογές που δεν θέλουμε να περιοριστούν από το Decay έχουν άπειρο Decay Time, ή αντίστοιχα έχουν την μέγιστη δυνατή τιμή στον καταχωρητή που τους αντιστοιχεί.

Για να υλοποιηθεί η λογική του CAT Decay, σε κάθε προσπέλαση μίας γραμμής της κρυφής μνήμης αποθηκεύεται η τιμή του μετρητή των CAT εκείνη την στιγμή. Σε κάθε αντικατάσταση, ο μηχανισμός αντικατάστασης:

- Σαρώνει όλες τις γραμμές του set.
- Για κάθε μία διαβάζει την αποθηκευμένη τιμή του CAT μετρητή και την συγκρίνει με την τρέχουσα.
- Αν η διαφορά τους έχει ξεπεράσει το Decay Time, τότε η γραμμή θεωρείται decayed και μπορεί να επιλεχθεί για αντικατάσταση.

3.6.3 Αποφάσεις διαχείρισης στο επίπεδο του Λειτουργικού Συστήματος

Στην μεθοδολογία μας, οι αποφάσεις που ελέγχουν την διαχείριση της κοινόχρηστης κρυφής μνήμης, δηλαδή η επιλογή των κατάλληλων Decay Time, παίρνονται στο επίπεδο του Λειτουργικού Συστήματος και πιο συγκεκριμένα στον scheduler των νημάτων. Ο scheduler είναι το κατάλληλο σημείο για τις αποφάσεις της μεθοδολογίας μας για πολλαπλούς λόγους. Πρώτον, η συλλογή των ιστογραμμάτων, η ανάλυσή τους και η λήψη αποφάσεων πρέπει να είναι περιοδική, αφού η συμπεριφορά ενός προγράμματος αλλάζει σημαντικά κατά τις διάφορες φάσεις της εκτέλεσής τους [77]. Δεύτερον, ο scheduler δημιουργεί, καταστρέφει ή παύει προσωρινά τα νήματα που εκτελούνται και κάθε φορά που παίρνει μία τέτοια απόφαση απαιτείται και μία νέα απόφαση διαχείρισης. Τρίτον, οι εγγυήσεις ποιότητας υπηρεσιών (QoS), που πρέπει να ληφθούν υπόψιν κατά την διαχείριση της κοινόχρηστης κρυφής μνήμης, παρέχονται από το λειτουργικό σύστημα. Ακόμη και αν καταφέρουμε να ικανοποιήσουμε σε υλικό τις δύο πρώτες απαιτήσεις (περιοδικότητα,

αναγνώριση των αλλαγών στο σετ εφαρμογών που εκτελούνται), μόνο στο επίπεδο του λειτουργικού συστήματος μπορούμε να συμπεριλάβουμε στην διαδικασία λήψης αποφάσεων τις QoS εγγυήσεις.

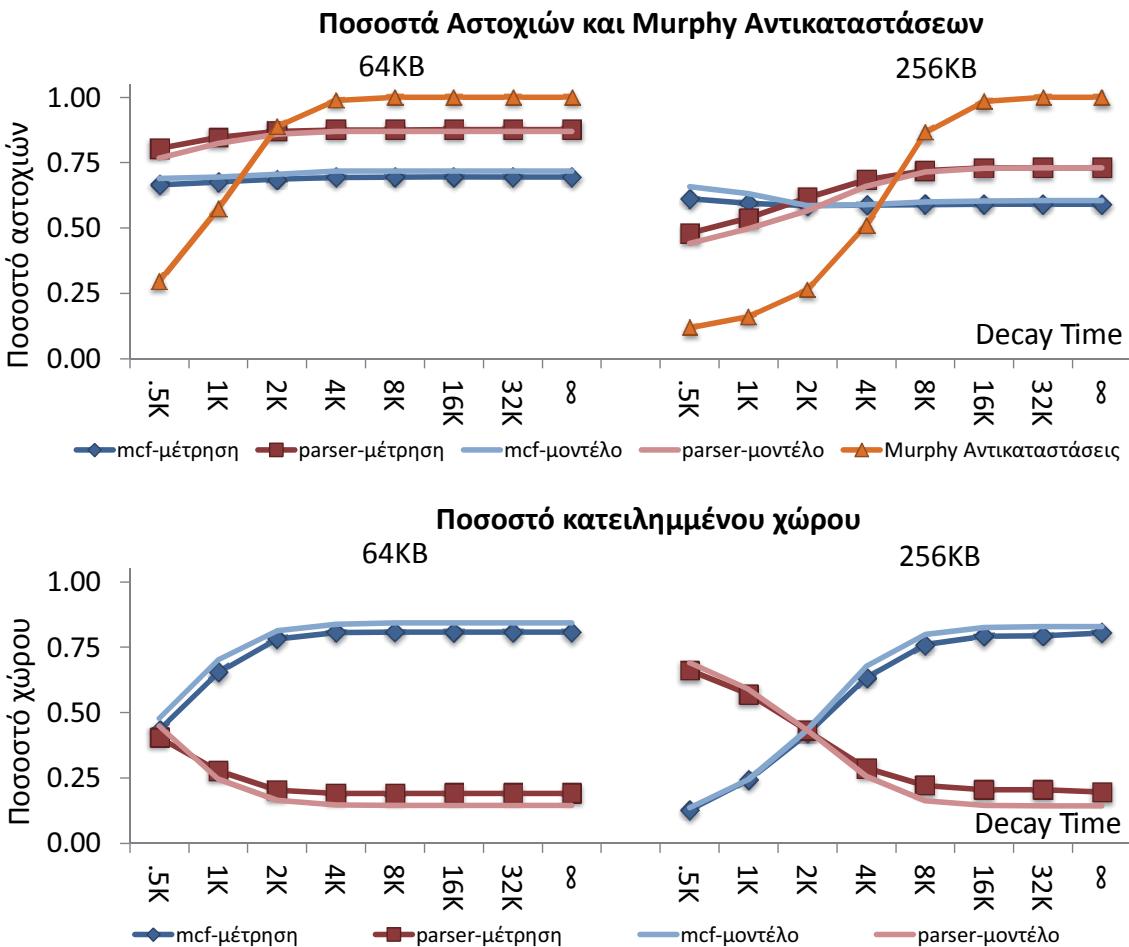
3.7 Πειραματική Μελέτη

Για την εξαγωγή των πειραματικών αποτελεσμάτων χρησιμοποιήσαμε έναν πλήρη εξομοιωτή μιας CMP αρχιτεκτονικής με 2 και 4 επεξεργαστές [31]. Κάθε επεξεργαστής είναι 4-way superscalar που μπορεί να εκτελεί εντολές εκτός σειράς. Η ιεραρχία μνήμης αποτελείται από κρυφές μνήμες πρώτου επιπέδου ξεχωριστές για εντολές και δεδομένα και μια κοινή κρυφή μνήμη δευτέρου επιπέδου, η οποία είναι συσχετιστικότητας 8 δρόμων με 64Bytes σε κάθε γραμμή. Το μέγεθος της κοινόχρηστης κρυφής μνήμης είναι μία από τις παραμέτρους των πειραμάτων μας και κυμαίνεται από 64KB έως 1MB. Τέλος, η καθυστέρηση της κύριας μνήμης είναι 300 κύκλοι.

Για εφαρμογές χρησιμοποιήθηκε ένα υποσύνολο των πιο απαιτητικών σε μνήμη προγραμμάτων της σουίτας SPEC2000: art, gzip, equake, vpr, mcf και parser. Προγράμματα με λιγότερο σημαντικές απαιτήσεις σε μνήμη εξετάστηκαν επίσης, αλλά λόγω του ότι καταλαμβάνουν ένα πολύ μικρό κομμάτι της κρυφής μνήμης, δεν παρουσιάζει ιδιαίτερο ενδιαφέρον η διαχείρισή τους, όπως επιβεβαιώνουν και σχετικές εργασίες [95][52][18][84]. Για όλα τα προγράμματα, αγνοούμε την φάση της αρχικοποίησης, εκτελούμε 50M εντολές για να φέρουμε τις κρυφές μνήμες σε σταθερή κατάσταση και έπειτα μελετάμε και διαχειρίζομαστε την κοινόχρηστη κρυφή μνήμη για 200M εντολές. Οι αποφάσεις διαχείρισης του StatShare παίρνονται κάθε 45M εντολές.

3.7.1 Επιβεβαίωση της ακρίβειας του μοντέλου (2 επεξεργαστές)

Στο πρώτο κομμάτι της πειραματικής μελέτης εξετάστηκε η ακρίβεια της μοντελοποίησης του διαμοιρασμού, μελετώντας το σενάριο όπου το mcf και το parser μοιράζονται την κρυφή μνήμη. Το mcf είναι ένα από τα πιο απαιτητικά σε μνήμη προγράμματα της σουίτας SPEC2000 όπως φαίνεται και από τις καμπύλες του StatCache στο Σχήμα 7.Το Σχήμα 17 παρουσιάζει τις προβλέψεις του StatShare συναρτήσει του Decay Time και τις συγκρίνει με τα πειραματικά αποτελέσματα. Το πρώτο γράφημα του σχήματος δείχνει το ποσοστό αστοχιών, ενώ το δεύτερο δείχνει το ποσοστό του χώρου της κρυφής μνήμης που καταλαμβάνει η κάθε εφαρμογή. Τα μεγέθη κρυφών μνημών που μελετάμε σε



Σχήμα 17: mcf-parser: ποσοστά αστοχιών, murphy αντικαταστάσεις και κατειλημμένος χώρος από εξομοιώσεις και θεωρητικό μοντέλο

αυτό το παράδειγμα είναι 64KB και 256B. Decay εφαρμόζεται μόνο στο mcf, αφού είναι το πιο απαιτητικό σε μνήμη από τα δύο προγράμματα, ενώ τα Decay Time που εφαρμόζουμε περιορίζονται σε 8 τιμές (512-32K και άπειρο).

Στο γράφημα που δείχνει τον κατειλημμένο χώρο, μπορούμε να δούμε την αποτελεσματικότητα του Decay στο να διαχειρίζεται την κρυφή μνήμη. Ενώ και για τις δύο κρυφές μνήμες το mcf καταλαμβάνει πάνω από το 80% των γραμμών, όταν εφαρμόζουμε Decay το mcf αρχίζει να περιορίζεται και να δίνει τις γραμμές του στο parser. Με το πιο επιθετικό Decay Time καταφέρνουμε να περιορίσουμε το mcf στο 48% της 64KB κρυφής μνήμης και στο 13% της 256KB κρυφής μνήμης. Επιπλέον, βλέπουμε ότι το parser καταλαμβάνει τον απελευθερωμένο χώρο και έτσι αυξάνει τον αριθμό των γραμμών που χρησιμοποιεί μέχρι 4.1 φορές.

Το γράφημα του ποσοστού αστοχιών μας δείχνει πως επηρεάζει αυτή η αναδιανομή του χώρου τις αστοχίες των δύο εφαρμογών. Και στις δύο κρυφές μνήμες το parser καταφέρνει

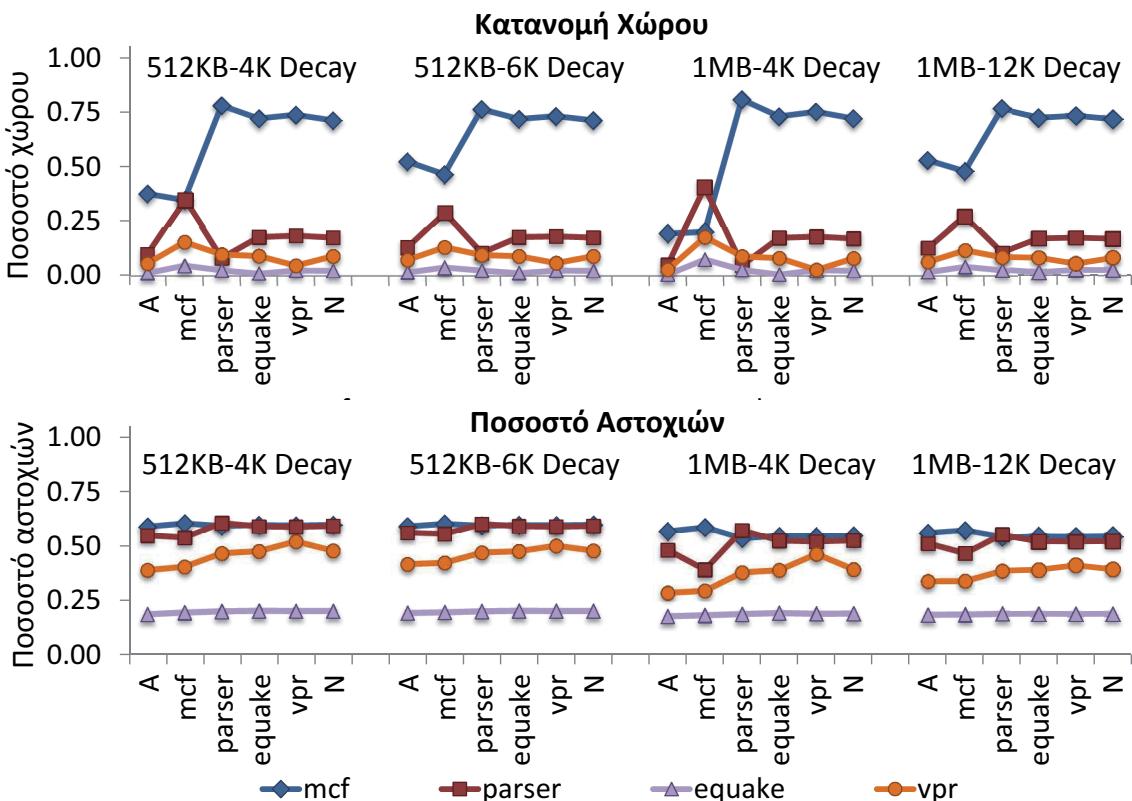
να μειώσει τον αριθμό των αστοχιών του: κατά 42% στην 256KB κρυφή μνήμη, κατά 8% στην 64KB κρυφή μνήμη. Το mcf, από την άλλη, επιβαρύνεται με μερικές επιπλέον αστοχίες για επιθετικά Decay Times στην 256KB κρυφή μνήμη, ενώ στην 64KB κρυφή μνήμη το ποσοστό αστοχιών του μένει πρακτικά σταθερό. Ο λόγος που το Decay δεν έχει σημαντική επίδραση στο ποσοστό αστοχιών του mcf, παρά την δραματική μείωση του χώρου που καταλαμβάνει, είναι ότι το mcf βρίσκεται στην σχετικά επίπεδη περιοχή της StatCache καμπύλης του (Σχήμα 7), οπότε η μείωση του χώρου του έχει συγκριτικά μικρή επίδραση.

Σε ότι αφορά την ακρίβεια του μοντέλου StatShare, βλέπουμε ότι οι θεωρητικές καμπύλες είναι πάρα πολύ κοντά στις πειραματικές. Η μέγιστη απόκλιση ανάμεσα στην πρόβλεψη του StatShare για την κατανομή του χώρου της κρυφής μνήμης και τις τιμές που προέκυψαν από τις εξομοιώσεις είναι μόλις 6%, ενώ η μέση απόκλιση είναι 3.1% για το mcf και 4% για το parser. Στα αποτελέσματα του ποσοστού αστοχιών, η απόκλιση είναι ακόμη χαμηλότερη: μέγιστη απόκλιση στο 4.9%, μέση απόκλιση 2% για το mcf 1.6% για το parser.

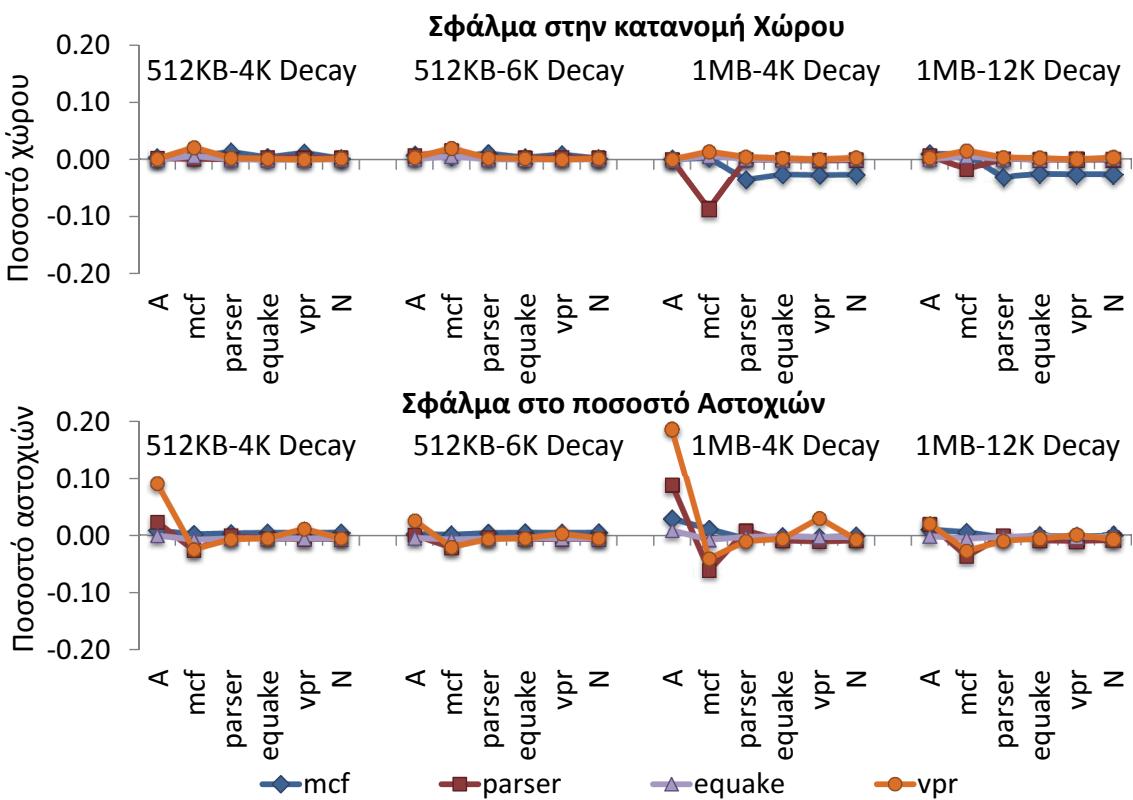
3.7.2 Επιβεβαίωση της ακρίβειας του μοντέλου (4 επεξεργαστές)

Σε ένα δεύτερο πείραμα για την μελέτη της θεωρητικής μοντελοποίησης εξετάσαμε το σενάριο όπου η κρυφή μνήμη διαμοιράζεται ανάμεσα σε 4 επεξεργαστές, οι οποίοι εκτελούν τις εφαρμογές mcf, parser, equake και vpr. Στο σενάριο αυτό χρησιμοποιούμε δύο διαφορετικά μεγέθη κρυφών μνημάτων (512KB και 1MB) και δύο διαφορετικά Decay Times για κάθε κρυφή μνήμη (4K/6K για την 512KB, 4K/12K για την 1MB κρυφή μνήμη). Στο Σχήμα 18 και στο Σχήμα 19 βλέπουμε τα αποτελέσματα των εξομοιώσεων και του μοντέλου, για Decay σε όλες τις εφαρμογές (γράμμα A στους x-άξονες), σε μία μόνο εφαρμογή (το αντίστοιχο όνομα στους x-άξονες) ή για την αρχική κατάσταση χωρίς διαχείριση (γράμμα N στους x-άξονες). Το Σχήμα 18 δείχνει το ποσοστό αστοχιών και την κατανομή του χώρου όπως προέκυψαν από τις εξομοιώσεις, ενώ το Σχήμα 19 δείχνει τις αποκλίσεις ανάμεσα στις προβλέψεις του μοντέλου και τα πειραματικά αποτελέσματα.

Βλέπουμε ότι και σε αυτό το πείραμα, το StatShare επιτυγχάνει να προβλέψει τα αποτελέσματα του διαμοιρασμού και της διαχείρισης της κρυφής μνήμης με μεγάλη ακρίβεια. Στα αποτελέσματα για την κατανομή του χώρου της κρυφής μνήμης, το σφάλμα στις περισσότερες περιπτώσεις είναι κάτω από το 2%. Αποκλίσεις πάνω από το 2% βλέπουμε μόνο στην κρυφή μνήμης του 1MB και μόνο για το mcf και το parser, αλλά ακόμη και για αυτές τις περιπτώσεις το σφάλμα δεν ξεπερνά το 8%. Στα αποτελέσματα για τις



Σχήμα 18: mcf-parser-equake-vpr: πειραματικά αποτελέσματα για την κατανομή χώρου και τα ποσοστά αστοχιών



Σχήμα 19: mcf-parser-equake-vpr: αποκλίσεις μεταξύ εξομοιώσεων και θεωρητικού μοντέλου για την κατανομή χώρου και τα ποσοστά αστοχιών

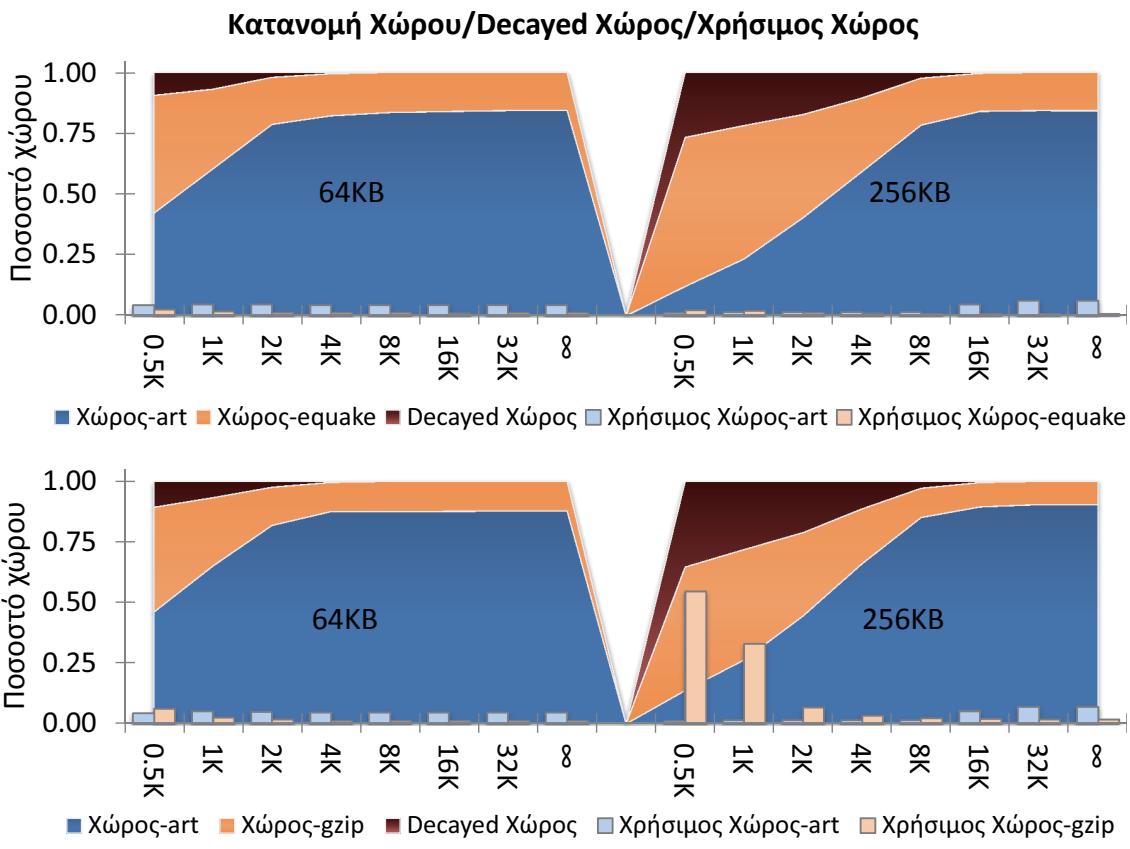
αστοχίες κρυφής μνήμης βλέπουμε αντίστοιχη ακρίβεια στις προβλέψεις του μοντέλου, με εξαίρεση τις περιπτώσεις όπου εφαρμόζεται decay σε όλες τις εφαρμογές. Οι μεγάλες αποκλίσεις (μέχρι 18%) σε αυτές τις περιπτώσεις, οφείλονται στον χειρισμό των εύστοχων προσπελάσεων decayed γραμμών ως αστοχίες. Όταν εφαρμόζεται decay σε όλες τις εφαρμογές, το ποσοστό της κρυφής μνήμης που καταλαμβάνεται από decayed γραμμές φτάνει μέχρι και το 50% στις εξομοιώσεις μας, οπότε αναμένεται να μεσολαβεί μεγάλο χρονικό διάστημα ανάμεσα στην στιγμή που γίνεται decay μία γραμμή μέχρι να αντικατασταθεί και επομένως αυξάνεται σημαντικά η πιθανότητα να προσπελαστεί πάλι μία γραμμή όσο είναι σε decayed κατάσταση.

3.7.3 Μελέτη των επιπτώσεων της διαχείρισης με CAT Decay (2 επεξεργαστές)

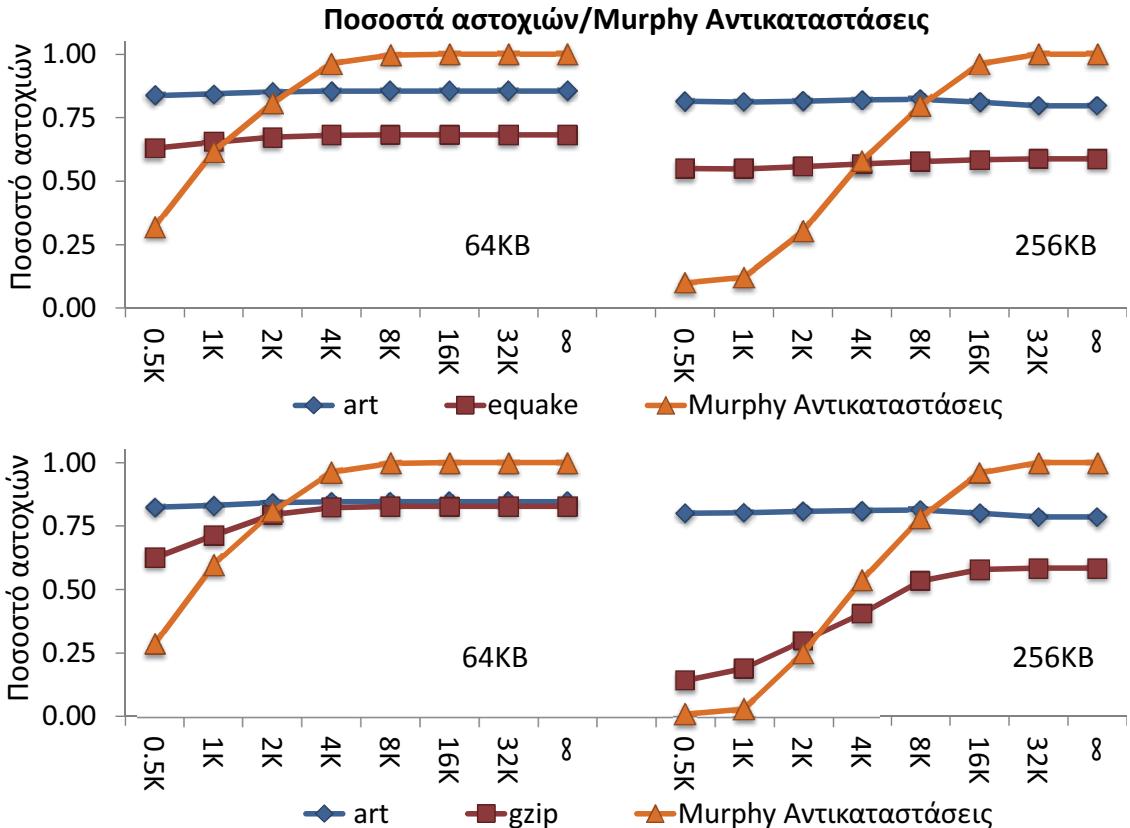
Στα προηγούμενα παραδείγματα δείξαμε ότι το StatShare επιτυγχάνει να προβλέψει με υψηλή ακρίβεια την συμπεριφορά των εφαρμογών που μοιράζονται την κρυφή μνήμη, είτε υπάρχει είτε δεν υπάρχει διαχείριση του διαμοιρασμού. Στα επόμενα παραδείγματα θα δείξουμε ότι με το StatShare μπορούμε να οδηγήσουμε τις αποφάσεις πολιτικών υψηλού επιπέδου, όπως πολιτικές που προσπαθούν να ελαχιστοποιήσουν τις συνολικές αστοχίες της κρυφής μνήμης ή πολιτικές που προσπαθούν να επιβάλουν εγγυήσεις QoS στο πως διαμοιράζεται η κρυφή μνήμη.

Το πρώτο παράδειγμα εξετάζει τον διαμοιρασμό της κρυφής μνήμης είτε ανάμεσα στο art και το equake, είτε ανάμεσα στο art και το gzip. Στο Σχήμα 20 βλέπουμε τα αποτελέσματα των εξομοιώσεων για το πως κατανέμεται ο χώρος της κρυφής μνήμης, για δύο διαφορετικά μεγέθη κρυφής μνήμης και 8 διαφορετικά Decay Times. Χωρίς διαχείριση, το art καταλαμβάνει το 90% της κρυφής μνήμης και για τις δύο ομάδες μετρήσεων και για τις δύο κρυφές μνήμες. Εφαρμόζοντας Decay στο art μπορούμε να απελευθερώσουμε σημαντικό τμήμα αυτού του χώρου και να το αποδώσουμε στο equake ή στο gzip.

Όμως, βλέπουμε στο Σχήμα 21 ότι το equake δεν μπορεί να εκμεταλλευτεί τον επιπλέον χώρο. Όπως φαίνεται και στο Σχήμα 8, το equake δεν έχει σχεδόν καμία προσπέλαση με reuse distance μεγαλύτερο των 2K CAT και η συντριπτική πλειοψηφία των προσπελάσεών του έχουν reuse distance κάτω από 512 CAT. Αυτό σημαίνει ότι η αύξηση του “φαινόμενου” L που προκαλεί το Decay δεν μετατρέπει αρκετές αστοχίες σε ευστοχίες. Με άλλα λόγια, το equake δεν έχει τον σωστό τύπο ιστογράμματος για να εκμεταλλευτεί το χώρο που απελευθερώνει το Decay από το art. Στο Σχήμα 20 βλέπουμε επίσης το ποσοστό του χώρου

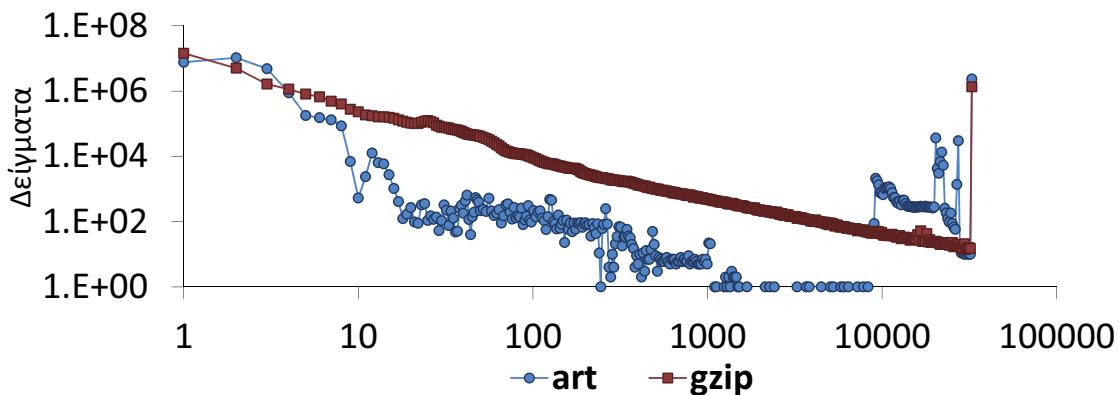


Σχήμα 20: art-equake και art-gzip, κατειλημμένος χώρος, decayed χώρος & χρήσιμος χώρος



Σχήμα 21: art-equake και art-gzip, ποσοστά αστοχιών και Murphy αντικαταστάσεις

της κρυφής μνήμης που καταλαμβάνουν οι χρήσιμες γραμμές του art και του equake (χρήσιμος χώρος). Αυτός ο χώρος βλέπουμε ότι μένει σχεδόν σταθερός και για τις δύο εφαρμογές, ανεξαρτήτως του Decay Time, το οποίο σημαίνει ότι αφενός το decay επηρεάζει σχεδόν αποκλειστικά τις γραμμές του art που δεν θα ξαναπροσπελαστούν, και αφετέρου ότι παρά την σημαντική αύξηση του χώρου που καταλαμβάνει το equake (3.1 φορές αύξηση στην 64KB κρυφή μνήμη, 3.87 φορές στην 256KB), δεν αξιοποιείται αυτός ο επιπλέον χώρος.

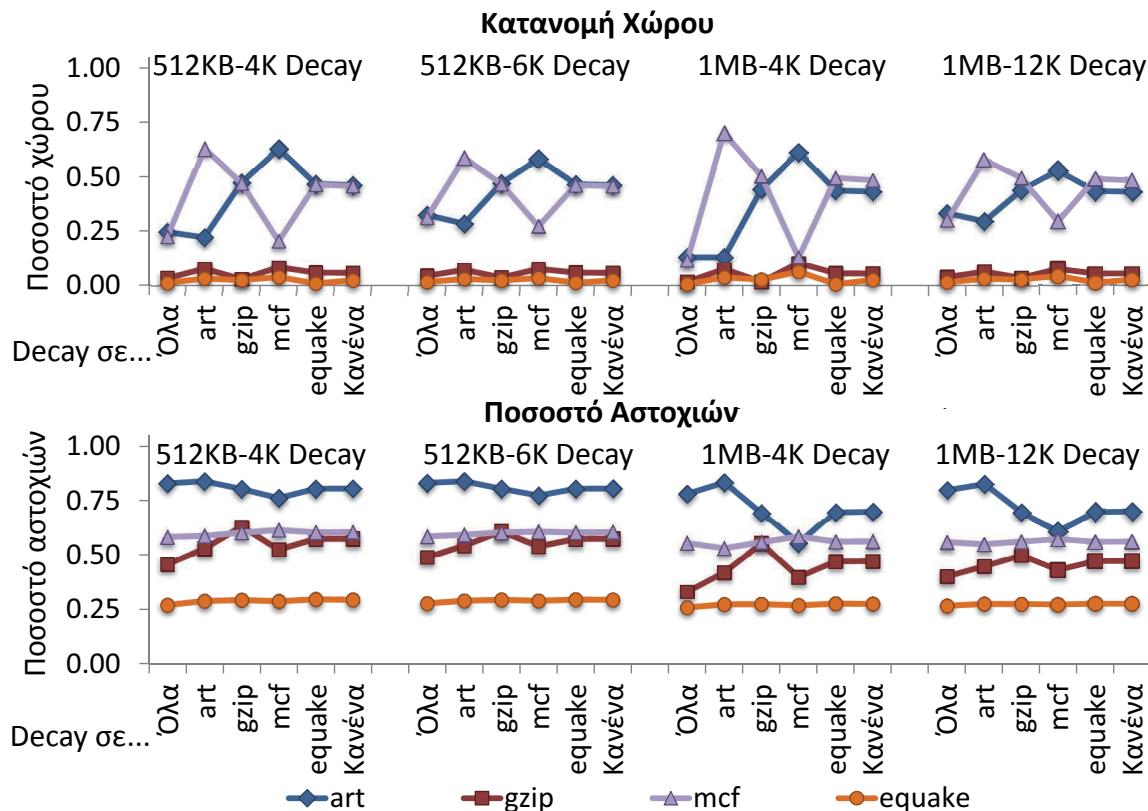


Σχήμα 22: Ιστογράμματα του art και του gzip, όταν μοιράζονται μία 256KB κρυφή μνήμη

Αντίθετα, το gzip βλέπουμε ότι έχει το σωστό τύπο ιστογράμματος για να εκμεταλλευτεί τον απελευθερωμένο χώρο. Στο Σχήμα 22 φαίνονται τα συλλεγμένα ιστογράμματα του art και του gzip. Επειδή το gzip έχει πάρα πολλά δείγματα σε μεγάλα reuse distances, όσο περισσότερο χώρο του παρέχουμε, τόσο καλύτερα τον αξιοποιεί. Το κάτω γράφημα στο Σχήμα 21 δείχνει πως αλλάζουν τα ποσοστά αστοχιών και οι Murphy αντικαταστάσεις καθώς μειώνουμε το Decay Time στο art: το gzip μειώνει τις αστοχίες του μέχρι και 25% για 64KB κρυφή μνήμη, ενώ πετυχαίνει ακόμη καλύτερα αποτελέσματα στην 256KB κρυφή μνήμη, όπου μειώνει τις αστοχίες του μέχρι και 76%. Παρόμοια εικόνα μας δίνει και το κάτω μισό του Σχήματος 20. Το gzip κερδίζει αντίστοιχο χώρο στην κρυφή μνήμη με αυτόν που κέρδιζε το equake, αλλά επειδή καταφέρνει να μεταφράσει αυτό το κέρδος σε μείωση των αστοχιών, ο χρήσιμος χώρος του αυξάνεται ραγδαία.

3.7.4 Μελέτη των επιπτώσεων της διαχείρισης με CAT Decay (4 επεξεργαστές)

Στο επόμενο πείραμα εξετάζουμε την διαχείριση του διαμοιρασμού μίας κρυφής μνήμης ανάμεσα στο art, το gzip, το mcf και το equake. Στο ΣΧΗΜΑΑ παρουσιάζονται η κατανομή του χώρου της κρυφής μνήμης και τα ποσοστά αστοχιών για δύο κρυφές μνήμες, 512KB και



Σχήμα 23: art-gzip-mcf-equake: πειραματικά αποτελέσματα για την κατανομή χώρου και τα ποσοστά αστοχιών, εφαρμόζοντας διάφορα Decay Time σε καμία, μία ή όλες τις εφαρμογές (χάρονας)

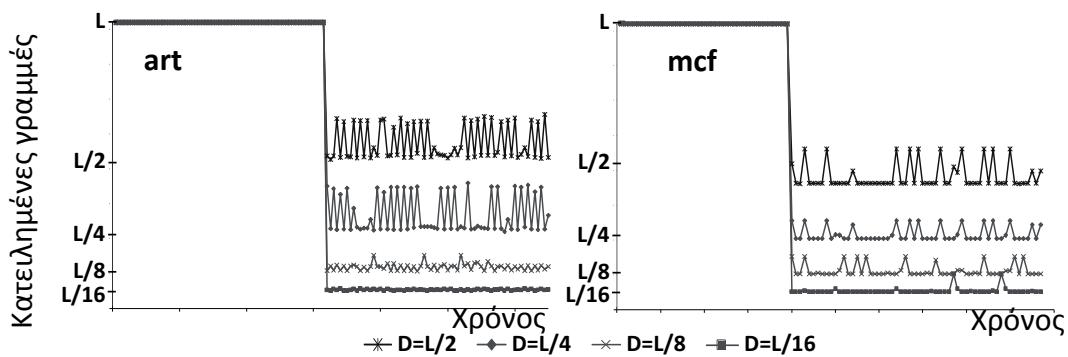
1MB, με σενάρια διαχείρισης αντίστοιχα με του προηγούμενου παραδείγματος για 4 επεξεργαστές: δύο διαφορετικά Decay Times για κάθε κρυφή μνήμη (4K/6K για την μικρότερη κρυφή μνήμη, 4K/12K για την μεγαλύτερη), και εφαρμογή του Decay Time σε καμία, μία ή όλες τις εφαρμογές.

Όπως και στην περίπτωση του art-equake προηγουμένως, βλέπουμε ότι το equake, λόγω της κατανομής των reuse distances του, δεν μπορεί να εκμεταλλευτεί τον χώρο που απελευθερώνει το Decay, ανεξαρτήτως των παραμέτρων της διαχείρισης ή της κρυφής μνήμης. Αντίστοιχα με τα προηγούμενα παραδείγματα, βλέπουμε πάλι ότι η διαχείριση του art σε σχετικά μικρές κρυφές μνήμες δεν επηρεάζει το ποσοστό αστοχιών του. αλλά δεν ισχύει το ίδιο για την κρυφή μνήμη του 1MB. Η ομάδα των υψηλών reuse distances του art αρχίζει να χωράει σε μεγάλες κρυφές μνήμες, και επομένως ο περιορισμός του art μέσω του Decay ξαναμετατρέπει αυτές τις προσπελάσεις σε αστοχίες, με αποτέλεσμα την αύξηση του ποσοστού αστοχιών του μέχρι και 13%. Αντίθετα, σε μεγάλες κρυφές μνήμες το art αφελείται από τον επιπλέον χώρο, όπως στις περιπτώσεις που περιορίζουμε το mcf, και καταφέρνει να μειώσει το ποσοστό αστοχιών του μέχρι 14.5%. Το gzip, τέλος, σε όλες τις

περιπτώσεις όπου εφαρμόζεται Decay στα άλλα προγράμματα εκμεταλλεύεται αποδοτικά τον επιπλέον χώρο και μειώνει σημαντικά τις αστοχίες του (μέχρι 21% στην κρυφή μνήμη των 512KB και μέχρι 31% στην 1MB κρυφή μνήμη).

3.7.5 Παροχή εγγυήσεων Ποιότητας Εξυπηρέτησης (QoS)

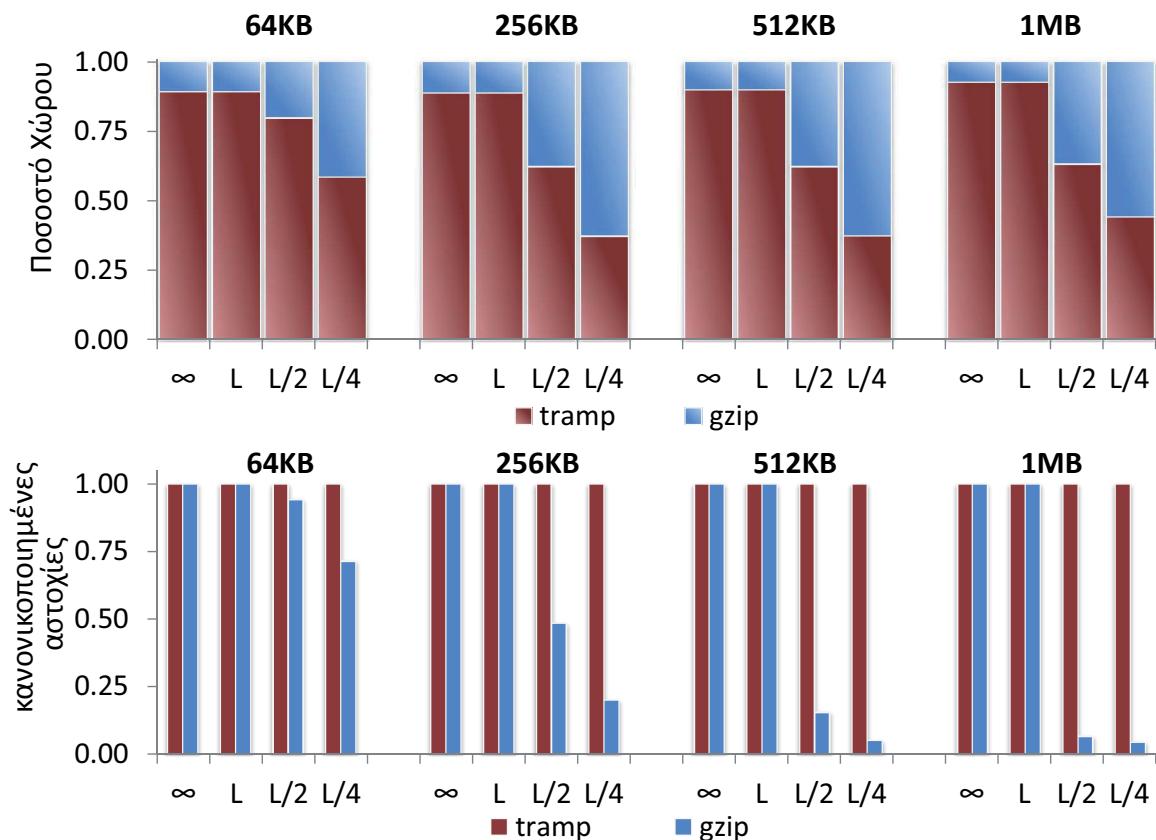
Η τελευταία σειρά πειραμάτων επικεντρώνεται στο πως μπορούμε να χρησιμοποιήσουμε την πληροφορία που παρέχει η μεθοδολογία StatShare για να αντιμετωπίσουμε σενάρια, όπου μία από τις εφαρμογές προσπαθεί να μονοπωλήσει την κατοχή της κρυφής μνήμης και γενικότερα να προσφέρουμε εγγυήσεις ποιότητας εξυπηρέτησης στην κρυφή μνήμη. Πιο συγκεκριμένα, μέσω του θεωρητικού μοντέλου υπολογίζουμε τον χώρο που προσπαθεί να καταλάβει η κάθε εφαρμογή, τις ταξινομούμε με βάση την επικινδυνότητα του να μονοπωλήσουν τον έλεγχο της κρυφής μνήμης και εφαρμόζουμε Decay στα πιο επικίνδυνα προγράμματα.



Σχήμα 24: Χώρος που καταλαμβάνουν το art (αριστερά) και το mcf (δεξιά) όταν εκτελούνται μόνα τους σε κρυφή μνήμη των 256KB, για διάφορα Decay Time

Εφόσον σε αυτήν την σειρά πειραμάτων στόχος μας είναι κυρίως ο έλεγχος του χώρου που καταλαμβάνει η κάθε εφαρμογή και όχι η αλληλεπίδραση των εφαρμογών, επιλέγουμε σαν μονάδα μέτρησης του χρόνου για το CAT Decay τις αντικαταστάσεις της κάθε μίας εφαρμογής και όχι τις συνολικές αντικαταστάσεις της κρυφής μνήμης. Το γιατί φαίνεται στο Σχήμα 24. Σε αυτό το σχήμα παρουσιάζεται ο χώρος που καταλαμβάνουν το art και το mcf όταν εκτελούνται μόνα τους και εφαρμόζουμε πάνω τους Decay με Decay Time L/2, L/4, L/8 ή L/16, όπου L ο αριθμός των γραμμών της κρυφής μνήμης. Βλέπουμε, ότι για όλα τα Decay Times, ο χώρος που κρατάει η κάθε εφαρμογή είναι λίγο μεγαλύτερος κατά μέσο όρο από το Decay Time που εφαρμόστηκε. Γενικότερα, τα πειράματα μας δείχνουν ότι ο χώρος που καταλαμβάνει η κάθε εφαρμογή είναι λίγο μεγαλύτερος από τον αριθμό αντικαταστάσεων της εφαρμογής που περιμένουμε πριν κάνουμε decay μία γραμμή.

Σε αυτήν την σειρά πειραμάτων χρησιμοποιήσαμε ένα επιπλέον πρόγραμμα, το οποίο εξομοιώνει την συμπεριφορά ενός κακόβουλου προγράμματος. Αυτό το πρόγραμμα, το tramp, είναι ένα συνθετικό πρόγραμμα το οποίο διατρέχει συνεχώς μία πολύ μεγάλη περιοχή μνήμης (πολύ μεγαλύτερη του μεγέθους της κρυφής μνήμης), με στόχο τον μέγιστο δυνατό ρυθμό παραγωγής αστοχιών. Σύμφωνα με τις εξισώσεις του StatShare, ένα τέτοιο πρόγραμμα θα πρέπει να καταλαμβάνει ένα πάρα πολύ μεγάλο κομμάτι της κρυφής μνήμης, συμπεριφορά δηλαδή παρόμοια με αυτήν ενός προγράμματος που προσπαθεί να μονοπωλήσει τον έλεγχο της κρυφής μνήμης.



Σχήμα 25: Χώρος που καταλαμβάνουν το tramp και το gzip και κανονικοποιημένα ποσοστά αστοχιών για διάφορα Decay Times στο tramp

Στο Σχήμα 25 παρουσιάζονται τα αποτελέσματα του διαμοιρασμού της κρυφής μνήμης ανάμεσα στο tramp και στο gzip για 4 διαφορετικά μεγέθη κρυφής μνήμης και 4 διαφορετικά Decay Times στο tramp (άπειρο, L, L/2 και L/4). Στο πάνω μισό του σχήματος βλέπουμε το κομμάτι του χώρου της κρυφής μνήμης που καταλαμβάνει κάθε εφαρμογή. Κάθε ομάδα αντιστοιχεί σε διαφορετική κρυφή μνήμη και κάθε μπάρα της ομάδας σε διαφορετικό decay time. Το γράφημα δείχνει ότι με την μεθοδολογία StatShare μπορούμε να πετύχουμε τον άμεσο έλεγχο της κοινόχρηστης κρυφής μνήμης: αρχικά το gzip είναι

περιορισμένο στο 7% έως 11% της κρυφής μνήμης, αλλά όταν εφαρμόζουμε L/4 decay time το gzip επιτυγχάνει να πάρει πάνω από την μισή κρυφή μνήμη. Το tramp φαίνεται να καταλαμβάνει περισσότερο χώρο από αυτόν που θα περιμέναμε, δηλαδή 50% για L/2 decay time και 25% για L/4 decay time, επειδή στο Σχήμα 25 προσμετρείται στο tramp και ο decayed χώρος του.

Το κάτω μισό του Σχήματος 25 δείχνει τις συνέπειες της αναδιανομής του χώρου της κρυφής μνήμης στις αστοχίες των δύο εφαρμογών (κανονικοποιημένες ως προς τις αστοχίες όταν δεν υπάρχει διαχείριση). Το μοντέλο της μεθοδολογίας StatShare αναγνωρίζει επιτυχημένα ότι το tramp όχι μόνο καταλαμβάνει δυσανάλογα μεγάλο χώρο, αλλά και ότι αυτός ο χώρος δεν αξιοποιείται, οπότε μπορεί να το συμπιέσει με ασήμαντα αποτελέσματα στο ποσοστό αστοχών του, όπως φαίνεται και στο γράφημα. Από την άλλη, το gzip επωφελείται από τον επιπλέον χώρο, όπως και σε όλα τα μέχρι τώρα πειράματα, οπότε καταφέρνει να μειώσει τις αστοχίες του μέχρι και 97%.

3.8 Ανακεφαλαίωση

Στο παρόν κεφάλαιο παρουσιάστηκε μία νέα μεθοδολογία για την διαχείριση κοινόχρηστων κρυφών μνημάτων σε πολυεπεξεργαστικά συστήματα ενός ολοκληρωμένου. Η μεθοδολογία αποτελείται από το θεωρητικό μοντέλο StatShare, που περιγράφει και προβλέπει τις συνέπειες του διαμοιρασμού της κρυφής μνήμης, και από έναν μηχανισμό διαχείρισης του διαμοιρασμού. Το μοντέλο μας τροφοδοτείται από αραιά δειγματοληφθέντα δεδομένα που περιγράφουν την τοπικότητα των προσπελάσεων (CAT reuse distances), τα οποία συλλέγονται μέσω υλικού κατά την εκτέλεση των εφαρμογών, με ελάχιστο κόστος. Τα δεδομένα αυτά εισάγονται στο μοντέλο για να εκτιμηθούν οι αστοχίες κρυφής μνήμης της κάθε εφαρμογής και ο χώρος που καταλαμβάνουν οι εφαρμογές, και για την περίπτωση που η κρυφή μνήμη λειτουργεί με μία συνηθισμένη πολιτική αντικατάστασης αλλά και για την περίπτωση που την διαχειρίζόμαστε. Όλοι οι υπολογισμοί του μοντέλου μπορούν να γίνουν σε πραγματικό χρόνο, οπότε οι εκτιμήσεις του μοντέλου μπορούν να αξιοποιηθούν από το λειτουργικό σύστημα ή έναν μηχανισμό διαχείρισης σε υλικό, ώστε να επιτευχθούν διάφορες υψηλού επιπέδου πολιτικές.

Για να αποτιμήσουμε την ακρίβεια της μεθοδολογίας και να εξετάσουμε τους τρόπους με τους οποίους μπορούμε να χρησιμοποιήσουμε την πληροφορία εξόδου της StatShare για την διαχείριση της κρυφής μνήμης, υλοποιήσαμε την μεθοδολογία σε έναν CMP εξομοιωτή.

Χρησιμοποιώντας τον εξομοιωτή μελετήσαμε τα αποτελέσματα του διαμοιρασμού της κρυφής μνήμης ανάμεσα στα πιο απαιτητικά σε μνήμη προγράμματα της σουίτας SPEC2000. Τα πειραματικά αποτελέσματα απέδειξαν, αφενός ότι το μοντέλο StatShare μπορεί να περιγράψει με ακρίβεια την συμπεριφορά των εφαρμογών στην κοινόχρηστη κρυφή μνήμη, και αφετέρου ότι ο μηχανισμός διαχείρισης που υλοποιήσαμε χρησιμοποιώντας την πληροφορία της StatShare μπορεί να αυξήσει την αξιοποίηση του χώρου της κρυφής μνήμης και να μειώσει τις αστοχίες, με ελάχιστες αρνητικές συνέπειες στα προγράμματα όπου εφαρμόζουμε την διαχείριση.

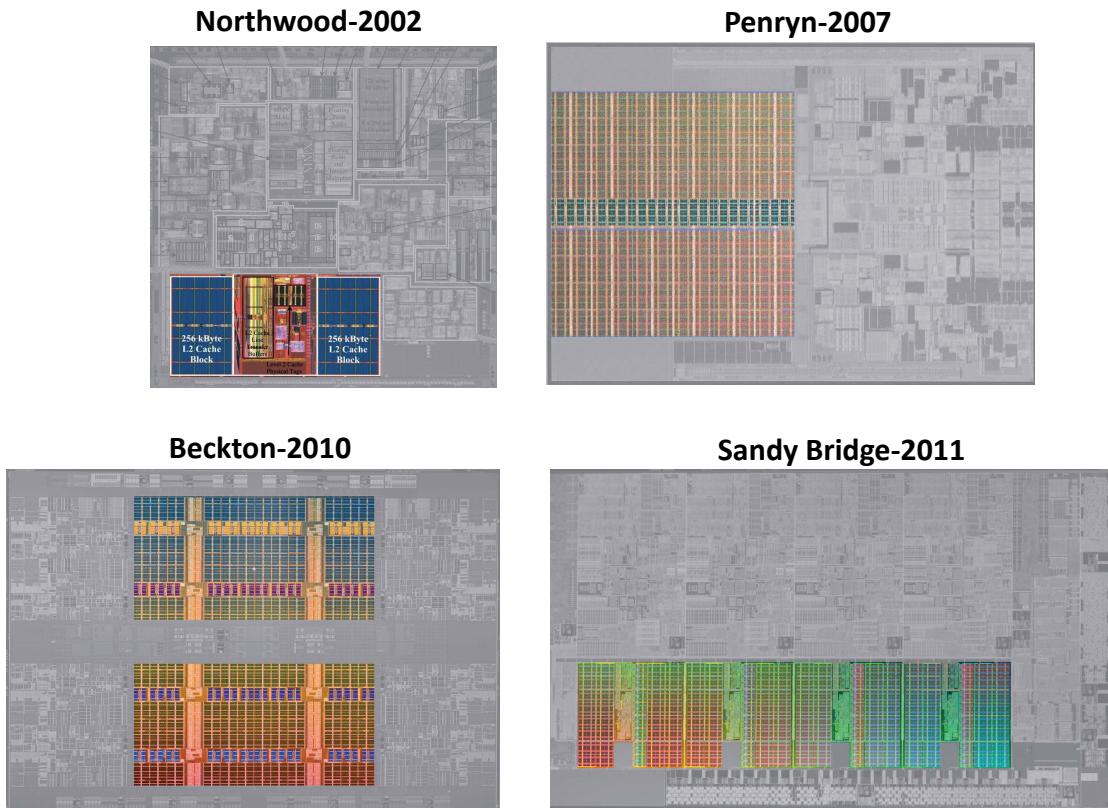
Κεφάλαιο 4

Αποδοτική Πολιτική Αντικατάστασης σε Κρυφές Μνήμες

4.1 Εισαγωγή

Όπως είδαμε και στα εισαγωγικά κεφάλαια, λόγω του χάσματος μεταξύ της ταχύτητας του επεξεργαστή και της ταχύτητας της μνήμης, η απόδοση των κρυφών μνημών είναι πια στενά συνδεδεμένη με την απόδοση των υπολογιστικών συστημάτων. Για αυτό τον λόγο, οι αρχιτέκτονες υπολογιστών σχεδιάζουν κρυφές μνήμες που καταλαμβάνουν ολοένα και μεγαλύτερο κομμάτι της επιφάνειας του ολοκληρωμένου (40% έως 60% στην τρέχουσα γενιά επεξεργαστών υψηλής απόδοσης) και που παρέχουν σύνθετες τεχνικές προφόρτωσης και πολιτικών αντικατάστασης. Παρόλα αυτά, το κόστος των προσπελάσεων της κύριας μνήμης συνεχίζει και συνεισφέρει σημαντικά στην υποβάθμιση της απόδοσης του επεξεργαστή, για πολλές εφαρμογές.

Επιπλέον, είδαμε στο Κεφάλαιο 3 ότι η μετάβαση σε πολυεπεξεργαστικά συστήματα ενός ολοκληρωμένου, κάνει τις κρυφές μνήμες, και ειδικά αυτές του τελευταίου επιπέδου, εξαιρετικά κρίσιμες: οι κρυφές μνήμες πρέπει πια να εξυπηρετούν πολλαπλές εφαρμογές, να λειτουργούν σαν δίαυλος επικοινωνίας για πολυνηματικές εφαρμογές και να κρύβουν την περιορισμένη χωρητικότητα του εξωτερικού διαύλου. Τεχνικές διαχείρισης του διαμοιρασμού, όπως η μεθοδολογία StatShare, μπορούν να βελτιώσουν την συμπεριφορά των κοινόχρηστων κρυφών μνημών, αλλά η επιτυχία τους στην μείωση των αστοχιών



Σχήμα 26: Φωτογραφίες του ολοκληρωμένου από 4 διαφορετικές γενιές επεξεργαστών της Intel. Το έγχρωμο/έντονο κομμάτι κάθε φωτογραφίας, δείχνει το τελευταίο επίπεδο κρυφής μνήμης

περιορίζεται στις περιπτώσεις όπου μία εφαρμογή μπορεί να απελευθερώσει, χωρίς σημαντικό κόστος, μεγάλο μέρος του χώρου που κατέχει. Σε σενάρια όπου όλες οι εφαρμογές αξιοποιούν τον χώρο τους και σε σενάρια όπου όλες οι εφαρμογές χρειάζονται επιπλέον χώρο, η διαχείριση του διαμοιρασμού μπορεί μόνο να ανακατανείμει “δίκαια” τις αστοχίες, δεν μπορεί να τις μειώσει σημαντικά.

Μία αποδοτική διαχείριση της κρυφής μνήμης απαιτεί, όπως μας έδειξαν και τα προηγούμενα κεφάλαια, την κατανόηση και εκμετάλλευση της τοπικότητας των προσπελάσεων. Στο παρελθόν, αρκετές μεθοδολογίες έχουν προσπαθήσει να αναλύσουν τα χαρακτηριστικά των εφαρμογών σε ότι αφορά την τοπικότητα. Η πρώτη κατηγορία μεθοδολογιών στηρίζεται στην ανάλυση του πηγαίου κώδικα κατά την φάση της μετάφρασης [96][21][25][26], αλλά εμφανίζει το μειονέκτημα του ότι δεν μπορεί να συλλάβει την δυναμική συμπεριφορά του κώδικα. Η δεύτερη κατηγορία αναλύει το πρόγραμμα κατά την διάρκεια της εκτέλεσής του, ώστε να συλλάβει σημαντικό κομμάτι της δυναμικής συμπεριφοράς [9][60][16], αλλά η επιτυχία της εξαρτάται από την αντιπροσωπευτικότητα των εισόδων που τροφοδοτούνται στο πρόγραμμα.

Και οι δύο κατηγορίες μεθοδολογιών, είτε στατικής είτε δυναμικής ανάλυσης, μπορούν να προσφέρουν “στατική” πληροφορία. Σε αυτό το κεφάλαιο, ο στόχος μας είναι να καταφέρουμε να περιγράψουμε την τοπικότητα των προσπελάσεων της κρυφής μνήμης δυναμικά και κατά την διάρκεια της εκτέλεσης της εφαρμογής. Πιο συγκεκριμένα δείχνουμε ότι είναι εφικτό να προβλεφθούν οι αποστάσεις επαναχρησιμοποίησης των γραμμών της κρυφής μνήμης, χρησιμοποιώντας πρόβλεψη βασισμένη στην εντολή που προκαλεί την προσπέλαση, κατά την διάρκεια της εκτέλεσης.

Για να το πετύχουμε αυτό, σχεδιάσαμε νέες δομές πρόβλεψης των αποστάσεων επαναχρησιμοποίησης (reuse distances) και δείξαμε ότι αυτές οι δομές μας παρέχουν πολύ υψηλή ακρίβεια στις προβλέψεις, έχουν σύντομους χρόνους απόκρισης και ότι μπορούν να λειτουργήσουν αποδοτικά ακόμη και με ελάχιστο αποθηκευτικό χώρο. Η πληροφορία που προσφέρουν οι μηχανισμοί πρόβλεψης μπορούν να χρησιμοποιηθεί με πολλαπλούς τρόπους και σε πολλαπλά επίπεδα της ιεραρχίας των κρυφών μνημών, αλλά σε αυτό το κεφάλαιο επικεντρωνόμαστε στην πολιτική αντικατάστασης του τελευταίου επιπέδου κρυφής μνήμης για δύο λόγους:

- Στα ανώτερα επίπεδα της ιεραρχίας, ο στόχος των κρυφών μνημών είναι να παρέχουν γρήγορα τα δεδομένα που χρειάζεται ο επεξεργαστής, οπότε οι κρυφές μνήμες πρέπει να είναι απλές, ακόμη και αν αυτό συνεπάγεται περισσότερες αστοχίες. Αντίθετα, το τελευταίο επίπεδο κρυφής μνήμης στοχεύει στην ελαχιστοποίηση των πολύ ακριβών προσπελάσεων κρυφής μνήμης και για αυτό είναι επιθυμητό να επιτυγχάνει την καλύτερη δυνατή διαχείριση του χώρου του, έστω και αν αυτό προκαλεί ελαφρά αύξηση του χρόνου απόκρισης.
- Οι κλασικές πολιτικές αντικατάστασης κρυφών μνημών (LRU, NRU, Random) απέχουν σημαντικά από το να είναι βέλτιστες όταν διαχειρίζονται τα κατώτερα επίπεδα κρυφών μνημών [69][94][51][73][81]. Όλοι οι κλασικοί μηχανισμοί υποθέτουν άμεσα ή έμμεσα ότι όσο πιο πρόσφατα έχει χρησιμοποιηθεί μία γραμμή, τόσο πιο σύντομα θα επαναχρησιμοποιηθεί, δηλαδή υποθέτουν πολύ υψηλή τοπικότητα στις προσπελάσεις, το οποίο συνήθως ισχύει για το πρώτο επίπεδο κρυφής μνήμης. Στα κατώτερα επίπεδα όμως, ακριβώς επειδή οι προσπελάσεις με υψηλή τοπικότητα έχουν φιλτραριστεί από το πρώτο επίπεδο κρυφής μνήμης, παρατηρείται χαμηλή τοπικότητα, με αποτέλεσμα οι κλασικές πολιτικές αντικατάστασης να παράγουν πολύ περισσότερες αστοχίες.

Για αυτούς τους λόγους, τα τελευταία χρόνια έχει σημειωθεί σημαντική ερευνητική προσπάθεια στον τομέα των πολιτικών αντικατάστασης και ιδιαίτερα σε μηχανισμούς που προβλέπουν τα χαρακτηριστικά τοπικότητας της κάθε εφαρμογής [69][94][51][73][40][57]. Όμως, όλες αυτές οι τεχνικές προσπαθούν να κάνουν απλές προβλέψεις όπως αν μία γραμμή θα επαναχρησιμοποιηθεί ή μετά από πόσες προσπελάσεις θα σταματήσει μία γραμμή να επαναχρησιμοποιείται. Αντίθετα, εμείς στοχεύουμε στην πλήρη ποσοτικοποίηση της τοπικότητας των προσπελάσεων, χρησιμοποιώντας την έννοια των reuse distances. Με αυτόν τον τρόπο μπορούμε να συγκρίνουμε ποσοτικά την προβλεπόμενη τοπικότητα των γραμμών της κρυφής μνήμης και να ελέγξουμε με μεγαλύτερη ακρίβεια τις αποφάσεις του μηχανισμού αντικατάστασης.

```

for (i = 0; i < 1024; i++)
{
    scanf("%d\n", &temp);
    array[i] = temp;
    ....
}
.....
for (i = 0; i < 1024; i++)
{
    var1 += array[i];
    ....
}
.....
for (i = 0; i < 1024; i++)
{
    array[i] *= 1.41;
    ....
}
}

```

Σχήμα 27: Παράδειγμα απλού προγράμματος

4.2 Πρόβλεψη των Αποστάσεων Επαναχρησιμοποίησης

Στην μεθοδολογία αυτού του κεφαλαίου, έχουμε ήδη αναφέρει, ότι ο στόχος μας είναι να προβλέπουμε τις αποστάσεις επαναχρησιμοποίησης των γραμμών της κρυφής μνήμης βάσει του ποια εντολή (PC) τις προσπέλασε τελευταία φορά. Ο λόγος που θεωρούμε ότι πρέπει να υπάρχει συσχέτιση ανάμεσα στις εντολές και τα reuse distances είναι ότι η δομή των περισσότερων προγραμμάτων παρουσιάζει κάποια κανονικότητα. Για παράδειγμα, στο Σχήμα 27 βλέπουμε το κεντρικό κομμάτι ενός υποθετικού προγράμματος, με έμφαση στις προσπελάσεις ενός πίνακα. Αυτό το υποθετικό πρόγραμμα αποτελείται από τρεις βρόχους:

ο πρώτος αρχικοποιεί τα δεδομένα ενός πίνακα, ο δεύτερος αθροίζει ολόκληρο τον πίνακα, ενώ ο τρίτος επιβάλλει έναν μετασχηματισμό στον πίνακα. Είναι προφανές ότι σε αυτό το παράδειγμα, η επαναχρησιμοποίηση των δεδομένων εξαρτάται πλήρως από την εντολή που τα προσπέλασε τελευταία φορά. Για παράδειγμα, ξέρουμε ότι τα δεδομένα που προσπέλασε η αρχικοποίηση θα ξαναδιαβαστούν στο σώμα του δεύτερου βρόχου και ότι για όλα τα δεδομένα του πίνακα η απόσταση επαναχρησιμοποίησης θα πρέπει να είναι παρόμοια. Επίσης, ξέρουμε ότι τα δεδομένα του πίνακα, αφού διαβαστούν από τον τρίτο βρόχο, δεν θα ξαναχρησιμοποιηθούν.

Τα πραγματικά προγράμματα είναι αρκετά πιο σύνθετα από αυτό το παράδειγμα: δυναμικές δομές δεδομένων, διαφορετική ροή του κώδικα για διαφορετικά δεδομένα, αλληλεπίδραση με το λειτουργικό σύστημα ή με τον χρήστη κοκ. Σε κάθε περίπτωση όμως, πάρα πολλά προγράμματα εμφανίζουν παρόμοια δομή με αυτήν του παραδείγματος σε κομμάτια της εκτέλεσής τους, οπότε για κάποιες από τις εντολές του προγράμματος θα μπορέσουμε να βρούμε ισχυρή συσχέτιση με μία απόσταση επαναχρησιμοποίησης.

4.2.1 Ορισμός των αποστάσεων επαναχρησιμοποίησης

Οι αποστάσεις επαναχρησιμοποίησης παρουσιάστηκαν ήδη στα Κεφάλαια 2 και 3. Συνοπτικά, και με τον ευρύτερο δυνατό ορισμό, απόσταση επαναχρησιμοποίησης είναι το πλήθος από έναν τύπο γεγονότων που μεσολαβούν ανάμεσα σε δύο προσπελάσεις του ίδιου δεδομένου. Με βάση το ποια ακριβώς γεγονότα θα επιλέξουμε να μετρήσουμε, το ποιες προσπελάσεις παρακολουθούμε και τι αντιμετωπίζουμε σαν ένα δεδομένο, μπορούμε να ορίσουμε πολλούς διαφορετικούς τύπους reuse distances. Για παράδειγμα, στην μεθοδολογία StatCache παρακολουθούμε όλες τις προσπελάσεις τις κρυφής μνήμης που θέλουμε να περιγράψουμε, τα γεγονότα που μετράμε είναι οι προσπελάσεις της κρυφής μνήμης και κάθε γραμμή θεωρείται ένα δεδομένο. Στην μεθοδολογία StatShare ο ορισμός του reuse distance είναι παρόμοιος, με σημαντικότερη διαφοροποίηση ότι τα γεγονότα που μετράμε είναι οι αντικαταστάσεις της κρυφής μνήμης.

Όπως και με την μεθοδολογία StatShare, έτσι και εδώ ο ορισμός του reuse distance θα πρέπει να μπορεί να υποστηρίξει τον μηχανισμό διαχείρισης. Εφόσον, ο τελικός μας στόχος είναι μία πολιτική αντικατάστασης γραμμών στο τελευταίο επίπεδο κρυφής μνήμης, θα πρέπει το reuse distance να περιγράφει την τοπικότητα των προσπελάσεων της κρυφής μνήμης και επομένως θα πρέπει να μετράει την απόσταση ανάμεσα σε δύο προσπελάσεις

της ίδιας γραμμής της κρυφής μνήμης. Σαν μονάδα του χρόνου, θα μπορούσαμε να χρησιμοποιήσουμε είτε τον αριθμό των εντολών που μεσολαβούν, τον αριθμό των προσπελάσεων της κρυφής μνήμης ή τον αριθμό των αντικαταστάσεων της κρυφής μνήμης. Κατά την εξέλιξη της μεθοδολογίας μελετήσαμε και τις τρεις επιλογές, αλλά καταλήξαμε στην μέτρηση του πλήθους των προσπελάσεων της κρυφής μνήμης που διαχειριζόμαστε.

Ο αριθμός των εντολών που μεσολαβούν, παρότι έχει την πιο άμεση σχέση με την συμπεριφορά του προγράμματος, δεν είναι πολύ χρήσιμος για την διαχείριση της κρυφής μνήμης, γιατί εμπεριέχει τις μικροδιαφορές στην ροή του κώδικα που παρουσιάζονται από εκτέλεση σε εκτέλεση ενός βρόχου, διαφορές που συνήθως δεν είναι ορατές στα κατώτερα επίπεδα των κρυφών μνημών και δεν επηρεάζουν την διαχείρισή τους. Ο αριθμός των αντικαταστάσεων της κρυφής μνήμης από την άλλη, έχει την πιο άμεση σχέση με την συμπεριφορά της κρυφής μνήμης αλλά παρουσιάζει χαμηλή συσχέτιση με την συμπεριφορά των εντολών του προγράμματος. Όπως αναφέραμε και στο Κεφάλαιο 3, μετρώντας αντικαταστάσεις τα reuse distances που προκύπτουν, περιγράφουν και την συμπεριφορά του προγράμματος αλλά και την αλληλεπίδραση της εφαρμογής με τις υπόλοιπες εφαρμογές που μοιράζονται την κρυφή μνήμη και την αλληλεπίδραση με τους μηχανισμούς διαχείρισης της κρυφής μνήμης. Επομένως, χρησιμοποιώντας ένα τέτοιο reuse distance, η συσχέτιση ανάμεσα στην εντολή που προκαλεί την προσπέλαση μίας γραμμής και το reuse distances της επόμενης προσπέλασης, θα είναι αρκετά χαμηλή. Το πλήθος των προσπελάσεων της κρυφής μνήμης δίνουν την καλύτερη ισορροπία ανάμεσα στους δύο στόχους μας, δηλαδή την περιγραφή της συμπεριφοράς της κρυφής μνήμης και της συμπεριφοράς των εντολών του προγράμματος.

Μία τελευταία λεπτομέρεια του ορισμού που χρησιμοποιούμε για τις αποστάσεις επαναχρησιμοποίησης είναι ότι στην παρούσα μεθοδολογία αγνοούμε τις προσπελάσεις εγγραφής προς την κρυφή μνήμη. Ενώ στην κρυφή μνήμη πρώτου επιπέδου μία προσπέλαση εγγραφής προκαλείται άμεσα από μία εντολή, στα υπόλοιπα επίπεδα της ιεραρχίας μνήμης εγγραφή προκαλείται μόνο όταν αντικαθιστάται μία τροποποιημένη γραμμή από το αμέσως υψηλότερο επίπεδο. Αυτό σημαίνει ότι η προσπέλαση αυτή δεν σχετίζεται σε προγραμματιστικό επίπεδο με την εντολή που την προκάλεσε. Γενικότερα, στην μεθοδολογία μας θεωρούμε ότι οι προσπελάσεις εγγραφής προς την κρυφή μνήμη που διαχειριζόμαστε δεν σχετίζονται με καμία εντολή και ότι δεν αποτελούν κομμάτι της συμπεριφοράς του προγράμματος, οπότε τις αγνοούμε και σε ότι αφορά το ποιες προσπελάσεις ορίζουν τα áκρα μίας απόστασης επαναχρησιμοποίησης και σε ότι αφορά την

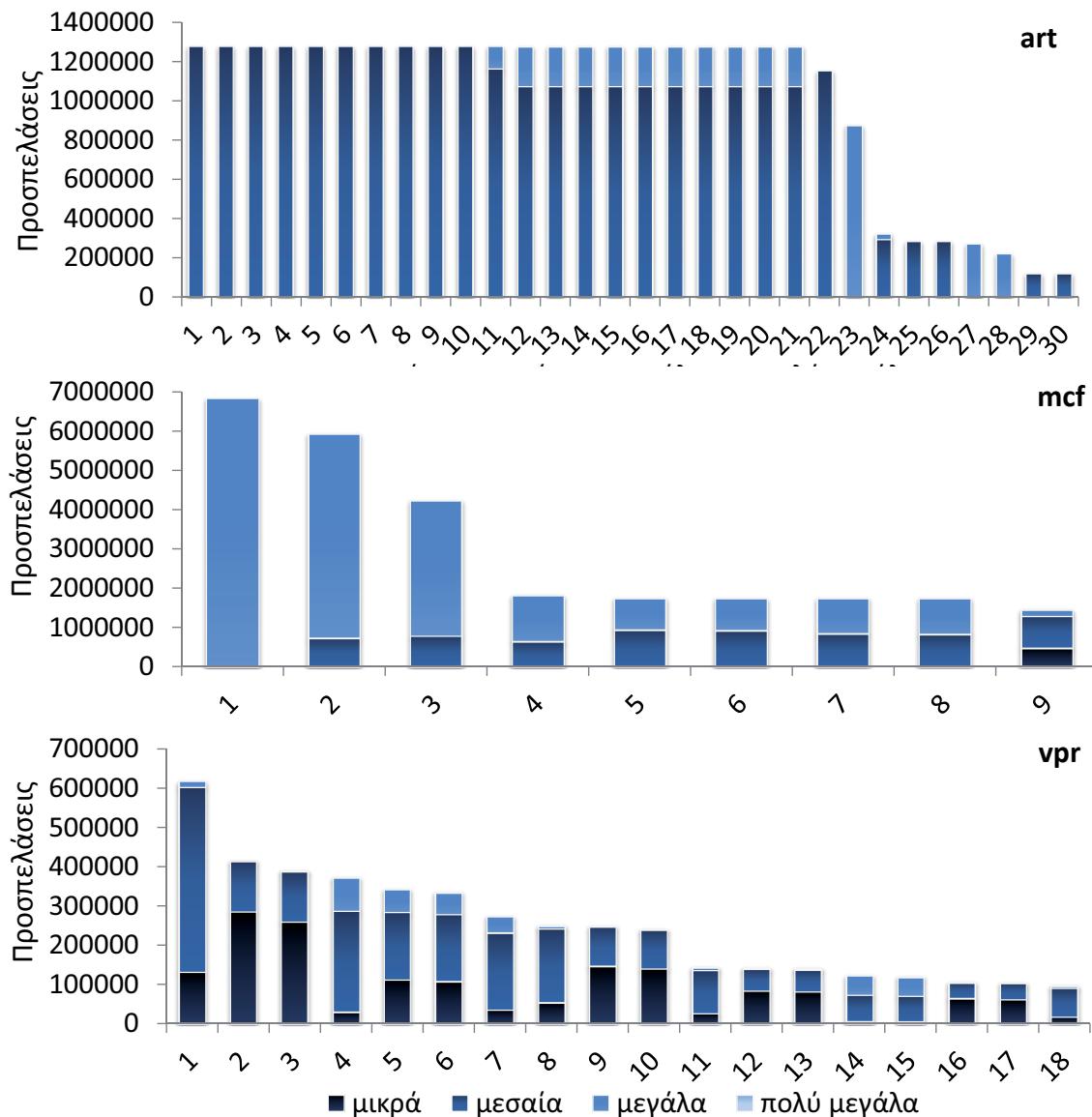
μονάδα χρόνου των αποστάσεων επαναχρησιμοποίησης. Έτσι, καταλήγουμε στον τελικό ορισμό που χρησιμοποιούμε στην παρούσα μεθοδολογία: *απόστασης επαναχρησιμοποίησης είναι το πλήθος των προσπελάσεων ανάγνωσης της κρυφής μνήμης ανάμεσα σε δύο προσπελάσεις ανάγνωσης της ίδιας γραμμής.*

4.2.2 Προβλεψιμότητα των αποστάσεων επαναχρησιμοποίησης

Για να μελετήσουμε κατά πόσο συσχετίζονται πραγματικά οι εντολές του προγράμματος με τις αποστάσεις επαναχρησιμοποίησης των γραμμών που προσπελαύνουν, εξετάσαμε όλα τα προγράμματα της σουίτας SPEC2000 και βρήκαμε τις αποστάσεις επαναχρησιμοποίησης που παράγει κάθε εντολή του προγράμματος. Στο Σχήμα 28 παρουσιάζουμε τα αποτελέσματα για το art, το mcf και το vpr. Για κάθε πρόγραμμα, δείχνουμε στον x-άξονα τις πιο σημαντικές εντολές του προγράμματος που παράγουν τουλάχιστον 98.5% των συνολικών προσπελάσεων της κρυφής μνήμης. Για κάθε εντολή η μπάρα δείχνει τον αριθμό των προσπελάσεων που προκάλεσε η εντολή και είναι χωρισμένη σε 4 κομμάτια ανάλογα με την απόσταση επαναχρησιμοποίησης των προσπελάσεων. Οι μικρές αποστάσεις επαναχρησιμοποίησης καλύπτουν το διάστημα 0-4K, οι μεσαίες το διάστημα 4K-64K, οι μεγάλες το διάστημα 64-512K, ενώ οι πολύ μεγάλες είναι αυτές που ξεπερνούν τις 512K ενδιάμεσες προσπελάσεις.

Όπως βλέπουμε και για τα τρία προγράμματα, για να πάρουμε μία αντιπροσωπευτική εικόνα της συμπεριφοράς τους στην κρυφή μνήμη αρκεί να μαζέψουμε τις αποστάσεις επαναχρησιμοποίησης λίγων μόνο εντολών. Για να περιγραφεί το 98.5% των προσπελάσεων, στο art χρειαζόμαστε 30 εντολές, στο vpr 18, ενώ στο mcf μόλις 9. Παρόμοια αποτελέσματα είδαμε και για τα υπόλοιπα προγράμματα της σουίτας SPEC2000. Γενικότερα, ελάχιστα προγράμματα έχουν πάνω από 32 σημαντικές εντολές και ακόμη και σε αυτά οι 10-20 σημαντικότερες εντολές παράγουν 9 στις 10 προσπελάσεις. Σε κάθε περίπτωση, ο περιορισμένος αριθμός εντολών που προσπελαύνουν την κρυφή μνήμη σημαίνει ότι δεν χρειάζεται μεγάλος αποθηκευτικός χώρος για να αποθηκεύσουμε αργότερα τις προβλέψεις μας.

Η συμπεριφορά που βλέπουμε να εμφανίζει το art είναι αρκετά προβλέψιμη. Οι περισσότερες εντολές του παράγουν σταθερά μεσαίου μεγέθους αποστάσεις επαναχρησιμοποίησης και οι υπόλοιπες παράγουν κυρίως μεσαίες αλλά σποραδικά παράγουν και μεγάλες αποστάσεις επαναχρησιμοποίησης. Η συμπεριφορά αυτή του art



Σχήμα 28: Οι πιο σημαντικές εντολές προσπέλασης του art, του mcf και του vpr, και η κατανομή των αποστάσεων επαναχρησιμοποίησης της κάθε εντολής

πηγάζει από την κανονικότητα του αλγορίθμου που εκτελεί: ο κεντρικός βρόχος του art αποτελείται από αρκετούς μικρότερους βρόχους που σαρώνουν σχεδόν γραμμικά μία ομάδα από πίνακες. Σε ένα τέτοιο πρόγραμμα το πότε θα επαναπροσπελαστεί ένα δεδομένο και από ποια εντολή είναι σχεδόν προκαθορισμένο.

Το mcf παρουσιάζει και αυτό αρκετά σταθερή συμπεριφορά. Η πιο σημαντική εντολή του παράγει μόνο μεγάλες αποστάσεις επαναχρησιμοποίησης, ενώ οι δύο επόμενες παράγουν κυρίως μεγάλες αποστάσεις. Οι υπόλοιπες σημαντικές εντολές παράγουν με παρόμοια συχνότητα και μεσαίες και μεγάλες αποστάσεις. Η σταθερότητα του mcf είναι σημαντική, γιατί η λογική που ακολουθούν οι εντολές προσπέλασης του mcf δεν είναι ιδιαίτερα προκαθορισμένη. Το κεντρικό κομμάτι του αλγορίθμου διατρέχει απλά

διασυνδεδεμένες λίστες, η δομή των οποίων μεταβάλλεται κατά την διάρκεια της εκτέλεσης του προγράμματος. Παρόλα αυτά όμως, οι προσπελάσεις που εξυπηρετούνται από το τελευταίο επίπεδο της κρυφής μνήμης έχουν αρκετά προβλέψιμη και σταθερή συμπεριφορά.

Τέλος, το vpr βλέπουμε ότι δεν έχει τόσο σταθερές αποστάσεις επαναχρησιμοποίησης. Κάθε εντολή συσχετίζεται με πολλές και αρκετά διαφορετικές αποστάσεις επαναχρησιμοποίηση, οπότε θα είναι δύσκολο να προβλέψουμε με ακρίβεια για αυτές τις εντολές την απόσταση επαναχρησιμοποίησης των δεδομένων που προσπελαύνουν. Ακόμη και έτσι όμως, η πρόβλεψη μπορεί να είναι αρκετά καλή ώστε να οδηγήσει την πολιτική αντικατάστασής μας. Για παράδειγμα, για την δεύτερη και την τρίτη πιο σημαντική εντολή είναι αρκετά ασφαλές να προβλεφθεί μικρή απόσταση επαναχρησιμοποίησης, γιατί ακόμη και αν κάνουμε λάθος, η πολιτική αντικατάστασης απλά θα προσπαθήσει λίγο περισσότερο να κρατήσει στην κρυφή μνήμη γραμμές με μεσαία απόσταση επαναχρησιμοποίησης. Γενικότερα, ακόμη και όταν οι εντολές δεν είναι στενά συσχετισμένες με μία μόνο απόσταση επαναχρησιμοποίησης, υπάρχουν περιθώρια αξιοποίησης της όποιας συσχέτισης υπάρχει.

4.2.3 Δομές πρόβλεψης

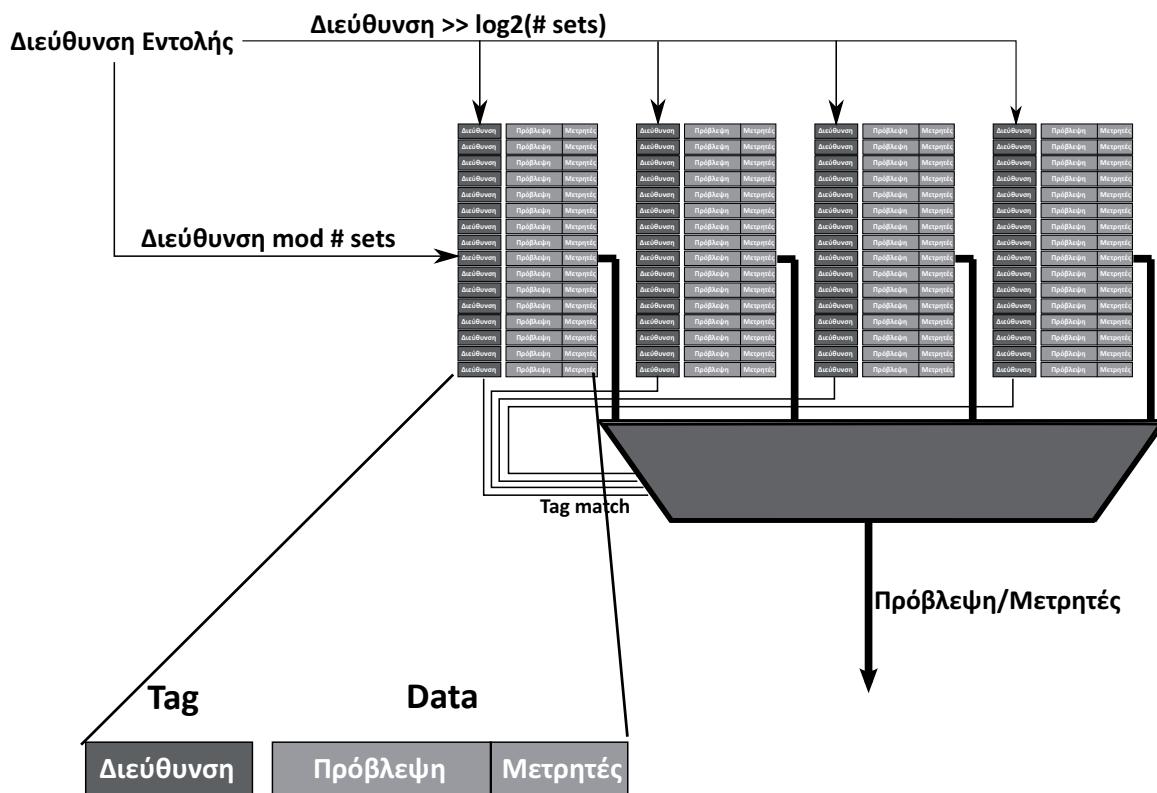
Με βάση, τα πειράματα που περιγράψαμε προηγουμένως, σχεδιάσαμε την δομή που κρατάει τις πρόβλεψεις μας (Σχήμα 29). Αυτή η δομή ονομάζεται Instruction-based Reuse Distance Predictor (IbRDP/Predictor) και είναι σχεδιασμένη όπως μία μικρή κρυφή μνήμη. Ο Predictor διευθυνσιοδοτείται από την διεύθυνση μίας εντολής: βάσει των χαμηλότερων bits της διεύθυνσης επιλέγεται το σετ του Predictor, ενώ τα υπόλοιπα bits συγκρίνονται με τα αποθηκευμένα tags για κάθε δρόμο του σετ, ώστε να βρεθεί η εγγραφή που αντιστοιχεί στην εντολή. Κάθε εγγραφή του Predictor αποθηκεύει την τρέχουσα πρόβλεψη απόστασης επαναχρησιμοποίησης για την εντολή και ένα πεδίο μετρητών, το οποίο δείχνει τον βαθμό εμπιστοσύνης στην πρόβλεψη. Το μέγεθος και η πολιτική ανανέωσης των μετρητών αποτελούν σημαντικές παραμέτρους του σχεδιασμού του Predictor.

Ο Predictor πρέπει να επιτελεί δύο λειτουργίες για να μπορέσει να οδηγήσει την συνολικότερη μεθοδολογία μας, την αναζήτηση και την ανανέωση. Πιο αναλυτικά:

- Κάθε φορά που μία εντολή προκαλεί προσπέλαση της κρυφής μνήμης, η διεύθυνση της εντολής χρησιμοποιείται ως είσοδος στον Predictor. Αν βρεθεί κάποια εγγραφή, τότε διαβάζονται τα δεδομένα της, τα οποία περιέχουν την πρόβλεψη απόστασης

επαναχρησιμοποίησης και τους μετρητές που αναφέραμε προηγουμένως. Αν η τιμή των μετρητών είναι πάνω από κάποια τιμή κατωφλίου, τότε θεωρούμε την πρόβλεψη έγκυρη, οπότε την εξάγουμε προς τους μηχανισμούς διαχείρισης που την χρησιμοποιούν.

- Κάθε φορά που προσδιορίζεται η απόσταση επαναχρησιμοποίησης μίας γραμμής, αυτή η απόσταση στέλνεται στον Predictor, μαζί με την διεύθυνση της εντολής που προσπέλασε τελευταία φορά την γραμμή. Η διεύθυνση χρησιμοποιείται όπως πριν για να βρεθεί η εγγραφή που αντιστοιχεί στην εντολή. Αν βρεθεί η εγγραφή, τότε συγκρίνουμε την καινούργια απόσταση επαναχρησιμοποίησης με την αποθηκευμένη πρόβλεψη. Αν οι δύο τιμές είναι ίσες, αυξάνουμε τους μετρητές εμπιστοσύνης, αν είναι διαφορετικές τους μειώνουμε, και αν οι τιμές διαφέρουν και οι μετρητές περιέχουν μηδενική τιμή, τότε θεωρούμε την αποθηκευμένη πρόβλεψη λάθος, οπότε την αντικαθιστούμε με την καινούργια απόσταση επαναχρησιμοποίησης. Τέλος, αν δεν βρεθεί εγγραφή για την εντολή, τότε δεσμεύουμε μία, θέτουμε το tag κομμάτι της, και την αρχικοποιούμε με πρόβλεψη την καινούργια απόσταση επαναχρησιμοποίησης.



Σχήμα 29: Οργάνωση και δομή του Instruction-based Reuse Distance Predictor.

Επειδή είναι εξαιρετικά απίθανο να παραχθούν ακριβώς οι ίδιες αποστάσεις επαναχρησιμοποίησης, ακόμη και αν το πρόγραμμα έχει τελείως προβλέψιμη και σταθερή συμπεριφορά, στην παρούσα μεθοδολογία αποθηκεύουμε και συγκρίνουμε κβαντισμένες αποστάσεις επαναχρησιμοποίησης. Με άλλα λόγια, θεωρούμε μία πρόβλεψη σωστή όχι όταν επιβεβαιώνεται ακριβώς, αλλά όταν είναι της ίδιας τάξη μεγέθους με την πραγματική απόσταση. Υπάρχουν πολλαπλοί τρόποι για να παράξουμε κβαντισμένες αποστάσεις επαναχρησιμοποίησης. Σε αυτήν την εργασία χρησιμοποιούμε δύο διαφορετικούς τρόπους: με εκθετικά μεγαλύτερες ομάδες αποστάσεων επαναχρησιμοποίησης (όπως στο Κεφάλαιο 3) ή με σταθερού μεγέθους ομάδες. Η πρώτη μέθοδος πρακτικά αποθηκεύει το δυαδικό λογάριθμο της απόστασης επαναχρησιμοποίησης, ενώ η δεύτερη αποθηκεύει ένα υποσύνολο των bits της πραγματικής απόστασης. Η δύο μέθοδοι χρησιμοποιούνται σε δύο διαφορετικές υλοποιήσεις του μηχανισμού πρόβλεψης, οπότε η ανάλυσή τους θα γίνει στις επόμενες ενότητες, όπου παρουσιάζονται οι λεπτομέρειες των υλοποιήσεων. Μέχρι τότε, το γενικό κομμάτι της ανάλυσής μας υποθέτει κβάντιση παρόμοια με τον StatShare.

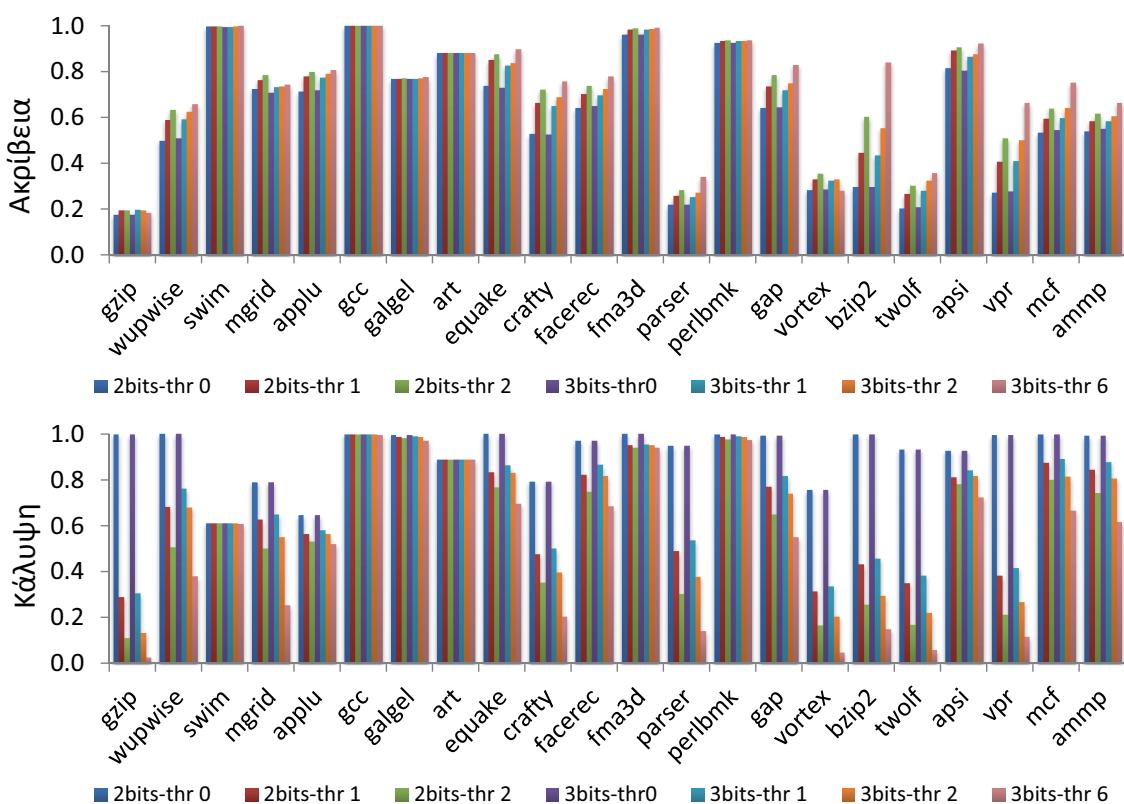
Με την λογική που παρουσιάσαμε για την ανανέωση και την αναζήτηση, ο Predictor πρακτικά προβλέπει την πιο συχνή απόσταση επαναχρησιμοποίησης που έχει παρατηρήσει να συσχετίζεται με μία εντολή στο πρόσφατο χρονικό διάστημα. Στα πλαίσια της ανάπτυξης της μεθοδολογίας εξετάσαμε και άλλες εναλλακτικές λύσεις:

- Διαφορετική πρόβλεψη για κάθε συνδυασμό των δύο τελευταίων εντολών που προσπέλασαν την γραμμή, αντί για την τελευταία μόνη της.
- Χρήση κάποιων από τα bits της διεύθυνσης του δεδομένου που προσπελάζεται για την διευθυνσιοδότηση του Predictor, ώστε η πρόβλεψη να αναφέρεται στον συνδυασμό εντολής-περιοχής διευθύνσεων.

Οι δύο εναλλακτικές αύξησαν αισθητά την ακρίβεια των προβλέψεων μας. Η πρώτη λύση στοχεύει στις περιπτώσεις όπου η συμπεριφορά του δεδομένου είναι συνάρτηση της γενικότερης ακολουθίας προσπελάσεων που έχουν προηγηθεί. Για παράδειγμα, μία εντολή προσπέλασης σε μία μικρή συνάρτηση που καλείται από διαφορετικά σημεία του προγράμματος αναμένεται να συσχετίζεται με διαφορετικές αποστάσεις επαναχρησιμοποίησης ανάλογα με το από που καλέστηκε. Σε αυτές τις περιπτώσεις για να κάνουμε επιτυχημένες προβλέψεις, χρειαζόμαστε γνώση και των προηγούμενων εντολών που προσπέλασαν την γραμμή.

Η δεύτερη λύση στοχεύει στις περιπτώσεις όπου διαφορετικές ομάδες των δεδομένων που διαβάζει μία εντολή, έχουν διαφορετική συμπεριφορά σε ότι αφορά τις αποστάσεις επαναχρησιμοποίησης. Το πιο απλό τέτοιο παράδειγμα είναι όταν ένας βρόχος σαρώνει έναν πίνακα από χαμηλότερες προς υψηλότερες διευθύνσεις, και ο επόμενος βρόχος που διαβάζει τον πίνακα τον σαρώνει από υψηλότερες προς χαμηλότερες διευθύνσεις. Παρότι η συμπεριφορά του κώδικα είναι προβλέψιμη και απλή, σε μία τέτοια περίπτωση οι αποστάσεις επαναχρησιμοποίησης των χαμηλών διευθύνσεων θα είναι αρκετά μεγαλύτερες από αυτές των υψηλών διευθύνσεων. Χρησιμοποιώντας λοιπόν κομμάτι της διεύθυνσης που προσπελάζεται για να διευθυνσιοδοτήσουμε τον Predictor, κρατάμε διαφορετικές προβλέψεις για διαφορετικές περιοχές δεδομένων.

Σε κάθε περίπτωση, η αύξηση της ακρίβειας των προβλέψεων συνδυάστηκε με σημαντική αύξηση της πολυπλοκότητας και του αποθηκευτικού χώρου του Predictor. Για αυτόν τον λόγο, σε συνδυασμό με την ικανοποιητική ακρίβεια του βασικού Predictor, αποφασίσαμε στην συνέχεια της μεθοδολογίας μας να μην χρησιμοποιήσουμε τους σύνθετους αυτούς Predictors.



Σχήμα 30: Ακρίβεια και κάλυψη των προβλέψεων του Predictor, για διαφορετικούς μετρητές αξιοπιστίας

4.2.4 Εξερεύνηση των χαρακτηριστικών των μετρητών αξιοπιστίας

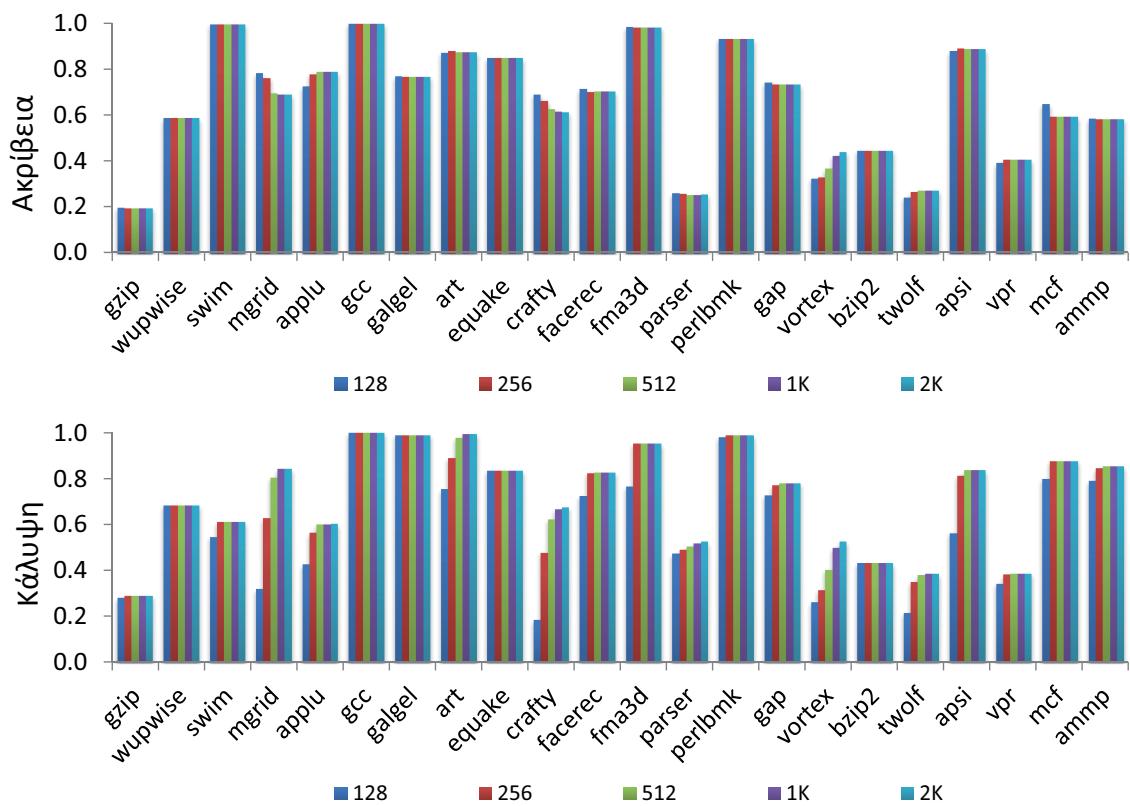
Στην προηγούμενη ενότητα αναφέραμε ότι το μέγεθος και η τιμή κατωφλίου των μετρητών αξιοπιστίας είναι κρίσιμες παράμετροι του σχεδιασμού του Predictor. Το μεν μέγεθος επηρεάζει την “μνήμη” του μετρητή: όσο μεγαλύτερος είναι ο μετρητής, τόσο περισσότερες ανανεώσεις με διαφορετική απόσταση επαναχρησιμοποίησης πρέπει να γίνουν μέχρι να αλλάξει μία κατά τα άλλα σταθερή πρόβλεψη. Διαφορετικά διατυπωμένο, όσο μεγαλύτερος είναι ο μετρητής τόσο μικραίνει η πιθανότητα να αλλάξει η πρόβλεψη εξαιτίας μίας παροδικής αλλαγής της συμπεριφοράς. Η δε τιμή κατωφλίου ρυθμίζει το τι θεωρούμε αξιόπιστη πρόβλεψη: όσο μεγαλύτερη είναι η τιμή κατωφλίου, τόσο περισσότερο σταθερή συμπεριφορά πρέπει να έχουν οι αποστάσεις επαναχρησιμοποίησης της εντολής προκειμένου να κάνουμε πρόβλεψη.

Για να μελετήσουμε τους μετρητές αξιοπιστίας, εκτελέσαμε μία σειρά πειραμάτων όπου για διάφορα μεγέθη και τιμές κατωφλίου των μετρητών μετρήσαμε το ποσοστό επιτυχίας των προβλέψεων (ακρίβεια) και το ποσοστό των προσπελάσεων για τις οποίες κάναμε πρόβλεψη (κάλυψη). Χρησιμοποιήσαμε 2 διαφορετικά μεγέθη (2 bits και 3 bits), 3 διαφορετικές τιμές κατωφλίου για τον μετρητή των 2 bits και 4 διαφορετικές τιμές κατωφλίου για τον μετρητή των 3 bits.

Στο Σχήμα 30 βλέπουμε τα αποτελέσματα των πειραμάτων. Για 11 από τα 22 προγράμματα της σουίτας SPEC2000 που μελετήσαμε, η ακρίβεια δεν εξαρτάται σχεδόν καθόλου από τα χαρακτηριστικά του μετρητή αξιοπιστίας. Για τα υπόλοιπα βλέπουμε ότι η αύξηση του κατωφλίου πρόβλεψης αυξάνει το ποσοστό των σωστών προβλέψεων, λίγο στις περισσότερες περιπτώσεις, αλλά σε προγράμματα όπως το crafty, το gap, το bzip2, το vpr και το mcf η αύξηση είναι σημαντική (μέχρι 53% αύξηση στο bzip2). Από την άλλη, η αύξηση του μεγέθους του μετρητή από 2 σε 3 bits επηρεάζει σημαντικά μόνο λίγα προγράμματα, όπως το bzip2, το vpr και το mcf, και μόνο για υψηλή τιμή κατωφλίου.

Στο κάτω μισό του σχήματος βλέπουμε την κάλυψη του Predictor. Αναμενόμενα, για μηδενική τιμή κατωφλίου η κάλυψη σε όλα τα προγράμματα είναι πολύ υψηλή και στα περισσότερα αγγίζει το 100%. 7 προγράμματα δεν επηρεάζονται από την άνοδο της τιμής κατωφλίου σε ότι αφορά την κάλυψη, αλλά για τα υπόλοιπα, η πτώση της κάλυψης είναι σημαντική. Επίσης, βλέπουμε ότι το μέγεθος του μετρητή έχει σχεδόν μηδενική επίδραση στο ποσοστό κάλυψης.

Γενικότερα, φαίνεται από τα πειράματα ότι το μέγεθος του μετρητή μπορεί να αυξήσει λίγο την ορθότητα των προβλέψεων μας, αλλά δεν είναι αρκετό ώστε να δικαιολογήσει την αύξηση του απαιτούμενου αποθηκευτικού χώρου. Η τιμή κατωφλίου έχει πολύ μεγαλύτερη επίδραση στην ορθότητα των προβλέψεων, αλλά συνδυάζεται με δυσανάλογη μείωση της κάλυψης του Predictor. Λόγω αυτής της συμπεριφοράς δεν υπάρχει κάποια βέλτιστη τιμή κατωφλίου. Η κατάλληλη τιμή εξαρτάται από την ευαισθησία των μηχανισμών που χρησιμοποιούν την πρόβλεψη. Αν ο μηχανισμός μπορεί να χειριστεί τις λάθος προβλέψεις και να μειώσει το κόστος τους, τότε χαμηλές τιμές κατωφλίου είναι προτιμητέες ώστε να μεγιστοποιηθεί ο αριθμός των προσπελάσεων για τις οποίες έχουμε σωστή ή σχεδόν σωστή πρόβλεψη. Αν όμως ο μηχανισμός έχει λίγη ανοχή σε λάθος προβλέψεις, τότε χρειάζεται υψηλή τιμή κατωφλίου ώστε να ελαχιστοποιήσουμε το ποσοστό των λαθών.



Σχήμα 31: Ακρίβεια και κάλυψη των προβλέψεων του Predictor, για διαφορετικά μεγέθη του Predictor

4.2.5 Εξερεύνηση του μεγέθους του Predictor

Στα πλαίσια της ίδιας σειράς πειραμάτων εξετάσαμε και την επίδραση του μεγέθους του Predictor στην ποιότητα των προβλέψεων. Στο Σχήμα 31 βλέπουμε την ακρίβεια και την κάλυψη του Predictor, χρησιμοποιώντας μετρητή αξιοπιστίας των 2 bit, με τιμή κατωφλίου

1, για μεγέθη που κυμαίνονται από τις 128 εγγραφές μέχρι τις 2048 εγγραφές. Το μέγεθος του Predictor έχει ελάχιστη επίδραση στην ακρίβεια. Στο vortex μόνο ανεβαίνει αισθητά η ακρίβεια, ενώ σε 3 προγράμματα υπάρχει μικρή πτώση της ακρίβειας. Η σημαντική επίπτωση της αύξησης του μεγέθους είναι στην κάλυψη: 7 προγράμματα ανεβάζουν πάνω από 20% την κάλυψή τους όταν το μέγεθος αυξάνεται από 128 εγγραφές σε 2K εγγραφές, ενώ για άλλα 7 υπάρχει μεταβολή από 5% έως 20%.

Γενικότερα, φαίνεται ότι η πλειοψηφία των προγραμμάτων μπορεί να πετύχει ακρίβεια και κάλυψη κοντά στο μέγιστο για Predictor με 256 εγγραφές, ενώ με 512 εγγραφές μόνο το vortex απέχει αισθητά από την μέγιστη δυνατή κάλυψή του. Αυτό το συμπέρασμα βρίσκεται σε συμφωνία με την Ενότητα 4.2.2: οι εντολές που προκαλούν σημαντικό αριθμό προσπελάσεων στο τελευταίο επίπεδο κρυφής μνήμης είναι ένα μικρό μόνο υποσύνολο των εντολών προσπέλασης μνήμης του προγράμματος.

4.3 Συλλογή Αποστάσεων Επαναχρησιμοποίησης

Στην προηγούμενη ενότητα περιγράφηκαν δομές που παράγουν προβλέψεις για τις μελλοντικές αποστάσεις επαναχρησιμοποίησης, βάσει των αποστάσεων επαναχρησιμοποίησης που παράχθηκαν στο παρελθόν από κάθε εντολή. Αυτό που παραλείπει όμως από την περιγραφή των δομών πρόβλεψης, είναι το πως συλλέγονται οι αποστάσεις επαναχρησιμοποίησης που τροφοδοτούνται στον Predictor.

Ιδανικά, θα θέλαμε να έχουμε έναν μηχανισμό που παρακολουθεί όλες τις γραμμές της μνήμης του προγράμματος και κάθε φορά που προσπελάζεται μία γραμμή να στέλνεται στον Predictor η απόσταση επαναχρησιμοποίησης και η διεύθυνση της εντολής που την προσέλασε την προηγούμενη φορά. Στην συνέχεια της μεθοδολογίας μας, αποκαλούμε αυτόν τον ιδανικό μηχανισμό Full Sampler (FS). Προφανώς, ο Full Sampler δεν είναι υλοποιήσιμος (τουλάχιστον όχι πρακτικά): εφόσον πρέπει να παρακολουθεί όλες τις γραμμές της μνήμης του προγράμματος, θα πρέπει να αποθηκεύει για κάθε γραμμή τον χρόνο τελευταίας προσπέλασης και την εντολή που προκάλεσε την προσπέλαση, και επομένως ο αποθηκευτικός χώρος που απαιτείται είναι συγκρίσιμος με τον χώρο που καταλαμβάνει στην μνήμη το πρόγραμμα. Επιπλέον, αυτός ο αποθηκευτικός χώρος θα πρέπει να σαρώνεται για κάθε προσπέλαση της κρυφής μνήμης ώστε να βρίσκεται η εγγραφή που αντιστοιχεί στην γραμμή που προσπελαύνουμε. Για να επιτευχθεί αυτό, θα πρέπει ο Full Sampler να είναι οργανωμένος ως μία πολύ μεγάλη CAM, ή έστω ως ένα πολύ

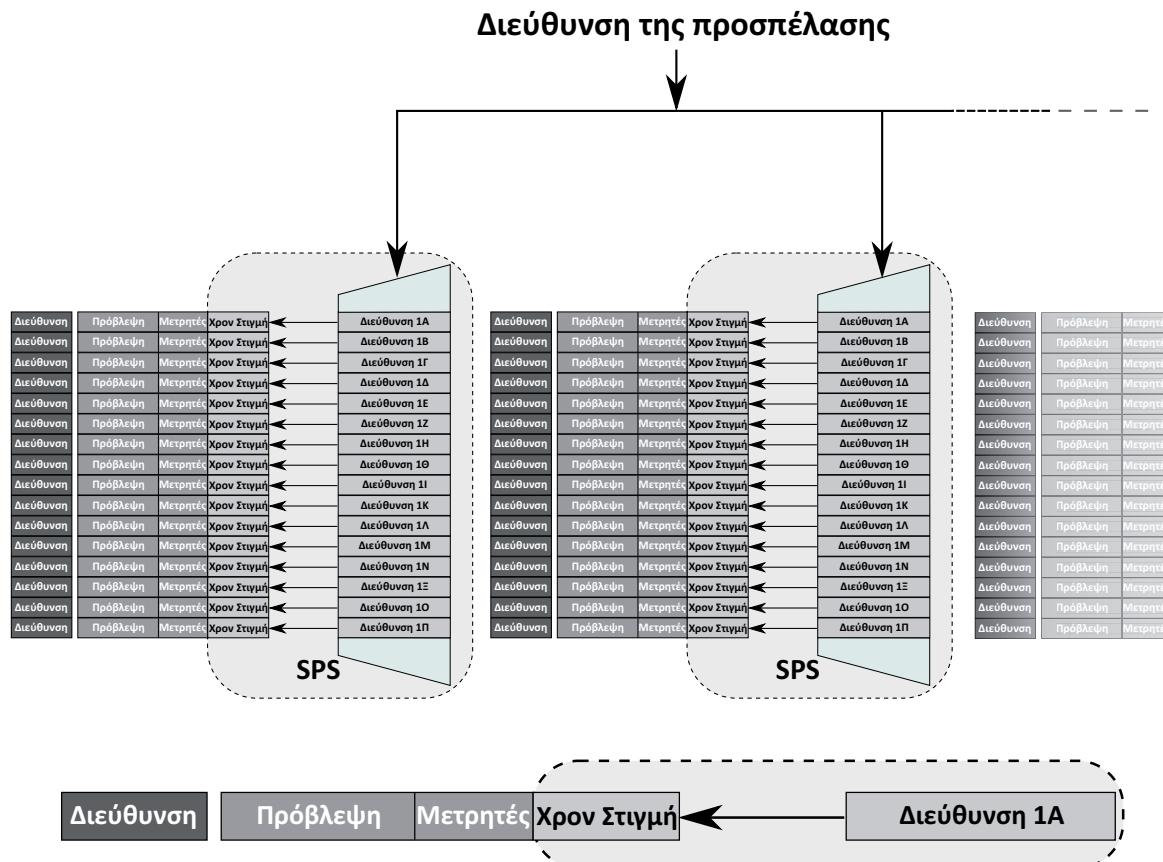
μεγάλο hash table. Και με τις δύο εναλλακτικές, ο χώρος που απαιτείται είναι απαγορευτικός, ενώ με την χρήση CAM επίσης απαγορευτική είναι και η ενέργεια που θα καταναλώνεται από την δομή. Είναι εμφανές ότι χρειαζόμαστε μία πιο πρακτική οργάνωση για την δομή που συλλέγει τις αποστάσεις επαναχρησιμοποίησης.

4.3.1 Μηχανισμός συλλογής αποστάσεων επαναχρησιμοποίησης με στόχο την υψηλή απόδοση

Η πρώτη λύση που παρουσιάζεται για την υλοποίηση του μηχανισμού συλλογής αποστάσεων επαναχρησιμοποίησης έχει σαν στόχο την παραγωγή πληροφορίας επαναχρησιμοποίησης που να είναι όσο το δυνατόν πιο κοντά στην πληροφορία που θα παρήγαγε ο Full Sampler.

Η βασική ιδέα αυτού του μηχανισμού συλλογής είναι η σύνδεση της διαδικασίας συλλογής με τον Predictor. Πιο συγκεκριμένα, υλοποιούμε μαζί με τον Predictor μία δομή που αποθηκεύει τα χαρακτηριστικά μίας μόνο προσπέλασης για κάθε εγγραφή του Predictor. Η δομή αυτή αποκαλείται Single-Prediction Sampler (SPS). Αν μία εντολή προκαλέσει προσπέλαση κρυφής μνήμης και η εγγραφή του SPS που αντιστοιχεί στην εντολή είναι κενή, τότε αποθηκεύουμε την στιγμή της προσπέλασης και την διεύθυνση που προσπελάστηκε στην εγγραφή. Μέχρι να εξακριβωθεί η απόσταση επαναχρησιμοποίησης της αποθηκευμένης διεύθυνσης, ο Predictor συνεχίζει κανονικά να κάνει προβλέψεις αλλά η πρόβλεψή του δεν ανανεώνεται. Όταν εξακριβωθεί η απόσταση επαναχρησιμοποίησης, ο Predictor ανανεώνεται και η εγγραφή του SPS αδειάζει. Σαν βελτιστοποίηση, ο χρόνος που επιτρέπεται να μείνει ένα δείγμα στον SPS είναι περιορισμένος: αν το δείγμα παραμείνει στον SPS για πάνω από 1M προσπελάσεις, το οποίο είναι και η μέγιστη απόσταση επαναχρησιμοποίησης που χρησιμοποιούμε στην παρούσα μεθοδολογία, τότε ανανεώνουμε τον Predictor με τιμή την μέγιστη απόσταση επαναχρησιμοποίησης. Με αυτόν τον τρόπο, αποτρέπουμε αφενός το ενδεχόμενο να γεμίσουν οι εγγραφές του SPS από δείγματα για γραμμές που δεν επαναχρησιμοποιείται και αφετέρου επιταχύνουμε τις ανανεώσεις για δείγματα με εξαιρετικά μεγάλες αποστάσεις επαναχρησιμοποίησης. Η δομή του SPS σε σχέση με τον Predictor φαίνεται στο Σχήμα 32.

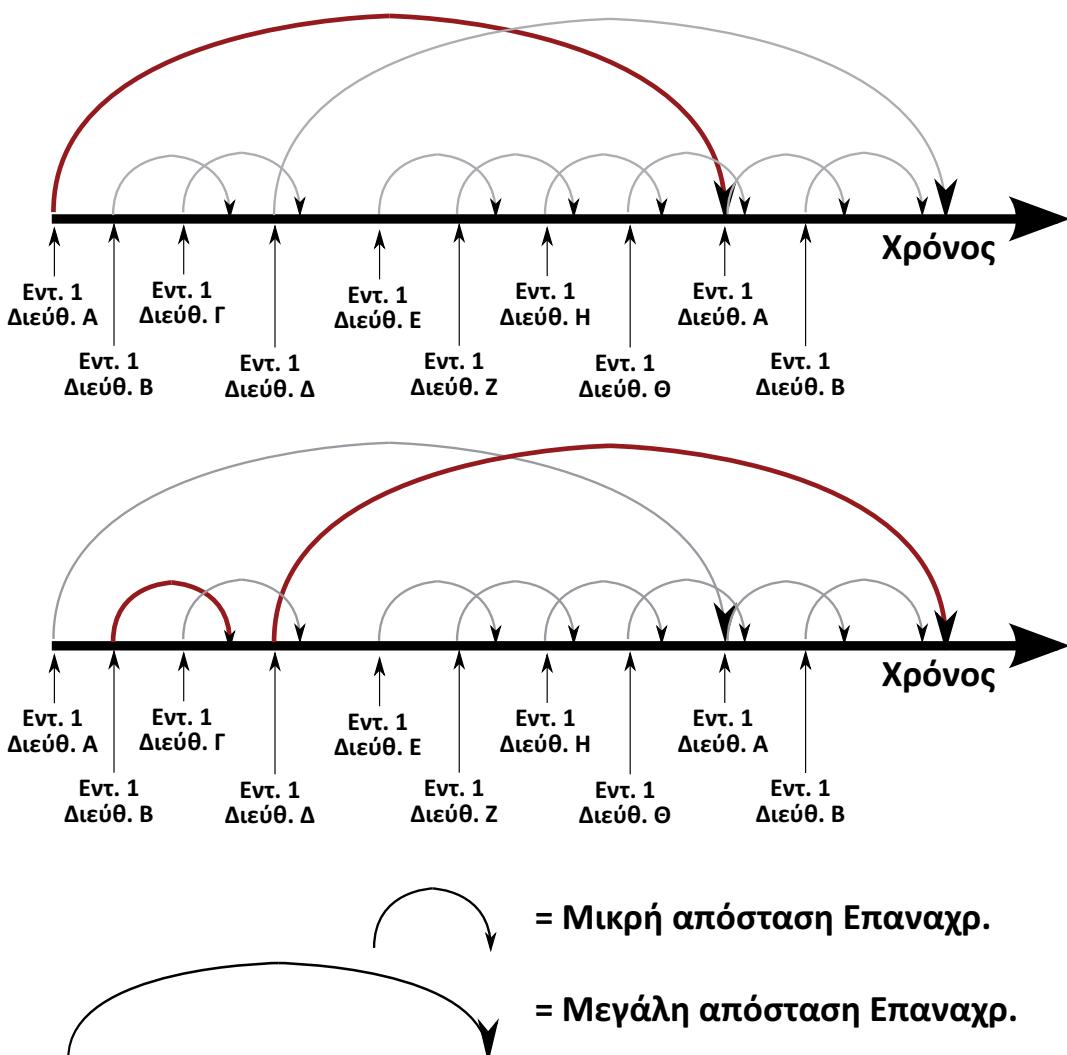
Ο SPS παρότι δειγματοληπτεί πάρα πολύ λίγες από τις προσπελάσεις της κρυφής μνήμης, η απόδοση του είναι αρκετά κοντά στον Full Sampler, όπως βλέπουμε και στο Σχήμα 35. Αλλά, δυστυχώς, υπάρχουν κάποιες περιπτώσεις που η συμπεριφορά του είναι πολύ κατώτερη του επιθυμητού. Για παράδειγμα, όταν μία εντολή συσχετίζεται και με πολύ



Σχήμα 32: Single-Prediction Sampler

μεγάλες και με πολύ μικρές αποστάσεις επαναχρησιμοποίησης, οι μεγάλες αποστάσεις επαναχρησιμοποίησης υπερ-εκπροσωπούνται στα δείγματα του SPS. Ο λόγος για αυτήν την συμπεριφορά φαίνεται στο Σχήμα 33: ένα δείγμα που θα δώσει μεγάλη απόσταση επαναχρησιμοποίησης “μπλοκάρει” τον SPS για ένα πολύ μεγάλο χρονικό διάστημα, στο οποίο θα χαθούν δείγματα για πολλαπλές μικρές αποστάσεις επαναχρησιμοποίησης. Αν αντίθετα, δειγματοληπτηθεί μία μικρή απόσταση επαναχρησιμοποίησης, θα χαθούν ελάχιστα ή και κανένα δείγματα με μεγάλη απόσταση.

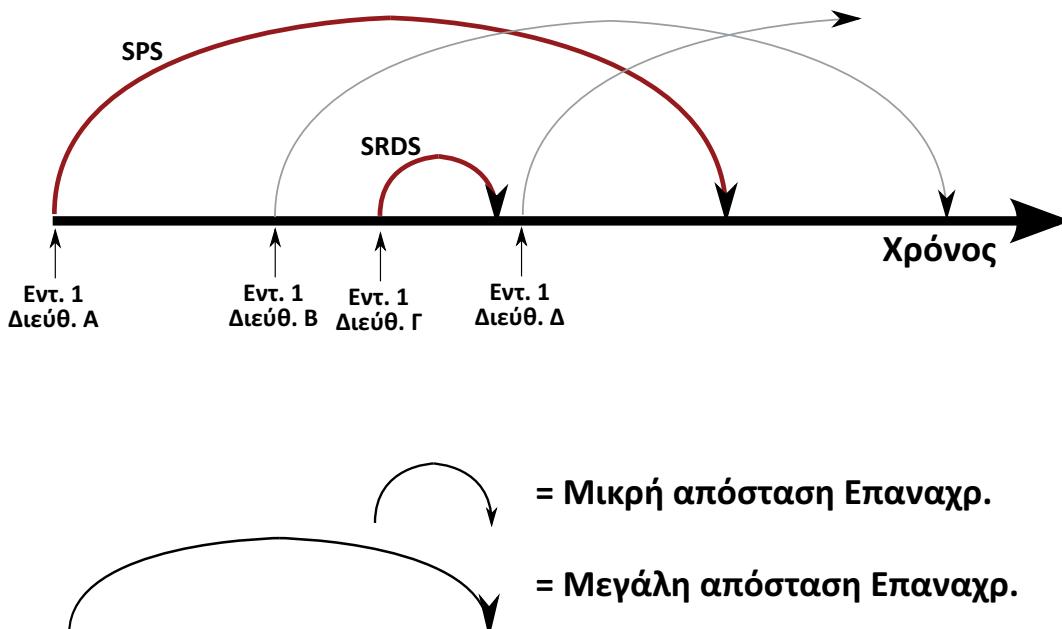
Πιο αναλυτικά, στο παράδειγμα του Σχήματος 33 βλέπουμε ότι αν η δειγματοληψία ξεκινήσει από την πολύ μεγάλη απόσταση επαναχρησιμοποίησης, τότε θα ανανεώσουμε τον Predictor με 1 μεγάλη και 2 μικρές αποστάσεις επαναχρησιμοποίησης, ενώ αν ξεκινήσουμε από την πρώτη μικρή απόσταση επαναχρησιμοποίησης, θα ανανεώσουμε τον Predictor με 1 μεγάλη και 1 μικρή απόσταση επαναχρησιμοποίησης. Με άλλα λόγια η αναλογία μικρών/μεγάλων αποστάσεων που δειγματοληπτούμε είναι από 1:1 έως 2:1, ενώ ο Full Sampler θα έδινε αναλογία 4:1. Αυτή η συμπεριφορά είναι ιδιαίτερα επιβλαβής: ο predictor δεν κινδυνεύει απλά να κάνει περισσότερες λάθος προβλέψεις, αλλά κινδυνεύει να



Σχήμα 33: Εντολή με πολύ μικρές και πολύ μεγάλες αποστάσεις επαναχρησιμοποίησης. Το πάνω σενάριο δείχνει δειγματοληψία μεγάλης απόστ. επαναχρ. η οποία μπλοκάρει 6 μικρές και 1 μεγάλη απόστ. επαναχρ. Το κάτω σενάριο δείχνει δειγματοληψία μικρής απόστ. επαναχρ., η οποία μπλοκάρει μόνο μία μικρή απόστ. επαναχρ.

προβλέπει πολύ μεγαλύτερες αποστάσεις επαναχρησιμοποίησης από τις πραγματικές. Όπως θα δούμε και παρακάτω, οι πολύ μεγάλες αποστάσεις επαναχρησιμοποίησης είναι αυτές που κυρίως διαχειρίζομαστε, καθώς μία τέτοια προσπέλαση είναι εξαιρετικά απίθανό να προκαλέσει ευστοχία κρυφής μνήμης, οπότε διαλέγουμε να αντικαθιστούμε τις γραμμές με τέτοια πρόβλεψη όσο το δυνατόν νωρίτερα. Αν ο predictor έχει την τάση να υπερεκτιμά σημαντικά την απόσταση επαναχρησιμοποίησης κάποιων εντολών, αυτό θα σημαίνει για την διαχείριση μας ότι θα αντικαθιστούμε πολλές γραμμές που κανονικά χωράνε στην κρυφή μνήμη. Με άλλα λόγια, ο Predictor μας δεν κάνει απλά λάθος προβλέψεις, αλλά κάνει λάθος προβλέψεις με τον πιο επιβλαβή τρόπο που θα μπορούσε.

Παρότι λοιπόν ο SPS λειτουργεί αποδοτικά στις περισσότερες περιπτώσεις, πρέπει να προφυλαχθούμε απέναντι σε σενάρια σαν το προηγούμενο. Για να το πετύχουμε αυτό, προσθέτουμε έναν επιπλέον μικρό sampler, ο οποίος στοχεύει στην σύλληψη κυρίως των μικρών αποστάσεων επαναχρησιμοποίησης. Αυτός ο sampler, που ονομάζουμε Short Reuse-Distance Sampler (SRDS), είναι βασισμένος στον sampler των StatCache και StatShare τεχνικών και είναι μία απλή FIFO ουρά, η οποία δειγματοληπτεί τυχαία τις προσπελάσεις της κρυφής μνήμης με κάποιον μέσο ρυθμό δειγματοληψίας $1/N$. Εφόσον ο SRDS είναι οργανωμένος σαν FIFO και δειγματοληπτεί μία στις N προσπελάσεις, η μέγιστη απόσταση επαναχρησιμοποίησης που μπορεί να συλλάβει είναι ίση με μέγεθος_SRDS $\times N$. Για παράδειγμα, αν ο SRDS έχει μέγεθος 8 εγγραφές και δειγματοληπτεί μία στις 256 προσπελάσεις, η μέγιστη απόσταση επαναχρησιμοποίησης που μπορεί να συλλάβει είναι 2K προσπελάσεις.

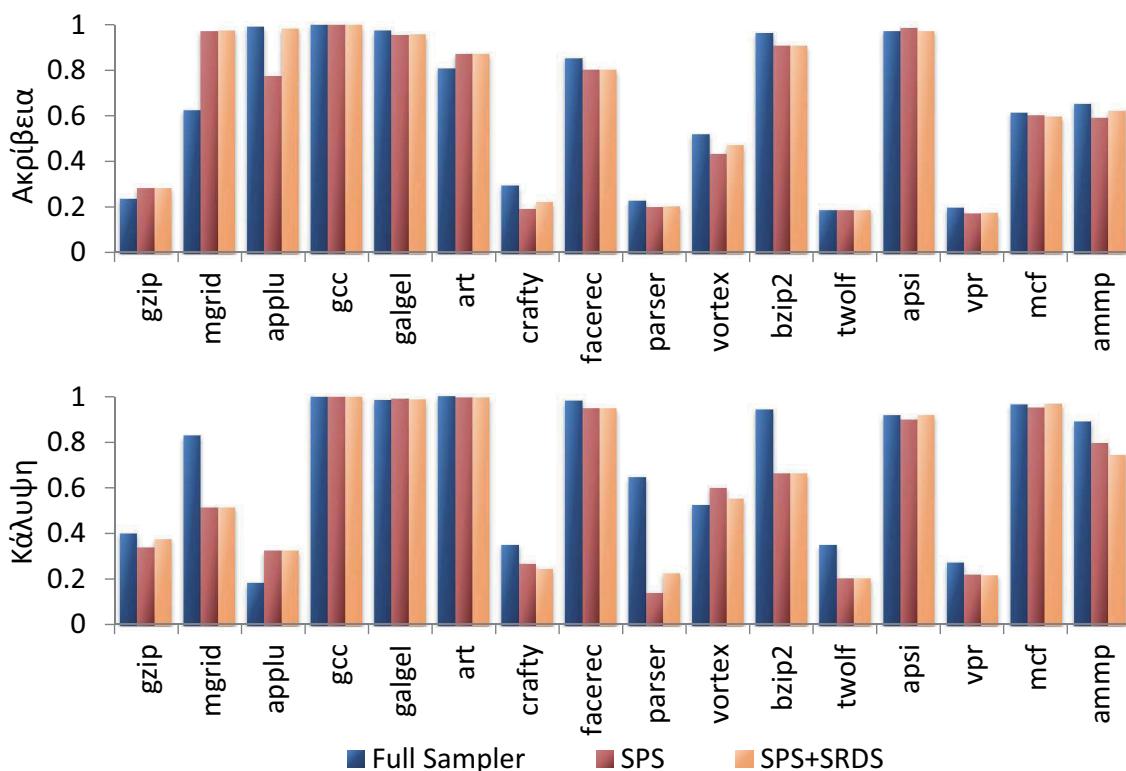


Σχήμα 34: Εντολή με πολύ μικρές και πολύ μεγάλες αποστάσεις επαναχρησιμοποίησης. Ο SRDS μπορεί να συλλάβει παράλληλα πολλαπλές μικρές απόστ. επαναχρ. ενώ ο SPS συλλαμβάνει μόνο μία μεγάλη απόστ. επαναχρ.

Ο SRDS λειτουργεί παράλληλα προς τον SPS και μπορεί να ανανεώνει τον predictor ανεξάρτητα κάθε φορά που αναγνωρίζει μία μικρή απόσταση επαναχρησιμοποίησης. Όμως αυτή η συμπεριφορά οδηγεί σε υπερδειγματοληψία των μικρών αποστάσεων επαναχρησιμοποίησης: ενώ ο SRDS μπορεί για κάθε εντολή να παρακολουθήσει παράλληλα πολλαπλές γραμμές που θα επαναπροσπελαστούν σύντομα, ο SPS μπορεί να παρακολουθήσει μόνο μία γραμμή κάθε φορά, οπότε δεν μπορεί να συλλάβει μεγάλες

αποστάσεις επαναχρησιμοποιήσεις που συμβαίνουν παράλληλα. Στο παράδειγμα του Σχήματος 34, αυτό σημαίνει ότι μόνο μία από τις τρεις μεγάλες αποστάσεις επαναχρησιμοποίησης μπορεί να συλληφθεί από τον SPS, ενώ ο SRDS μπορεί να συλλάβει πολλαπλές μικρές αποστάσεις επαναχρησιμοποίησης.

Για να χειριστούμε αυτό το σενάριο, τροποποιούμε ελαφρώς τον μηχανισμό συλλογής αποστάσεων επαναχρησιμοποίησης. Αν μία εγγραφή του SRDS αντικατασταθεί χωρίς να έχει υπολογιστεί απόσταση επαναχρησιμοποίησης, τότε η απόσταση επαναχρησιμοποίησης που της αντιστοιχεί είναι μεγαλύτερη από $\text{μέγεθος_SRDS} \times N$, οπότε θεωρούμε ότι χάσαμε μία ανανέωση με μεγάλη απόσταση επαναχρησιμοποίησης, γεγονός για το οποίο ενημερώνουμε τον Predictor. Ο Predictor, με την σειρά του, αποθηκεύει τον αριθμό ανανέωσεων με μεγάλη απόσταση επαναχρησιμοποίησης που έχουν χαθεί από τον SRDS και, για κάθε τέτοια χαμένη ανανέωση, αποτρέπει μία ανανέωση με μικρή απόσταση επαναχρησιμοποίησης. Πρακτικά, ο Predictor προσπαθεί να κρατήσει την αναλογία μικρών και μεγάλων αποστάσεων επαναχρησιμοποίησης που ανανεώνουν τις προβλέψεις του κοντά στην πραγματική αναλογία τους.



Σχήμα 35: Ακρίβεια και κάλυψη του Predictor για τροφοδότηση από τον Full Sampler, τον SPS ή τον συνδυασμό SPS και SRDS

Στο Σχήμα 35 βλέπουμε την ακρίβεια και την κάλυψη του Instruction-based Reuse Distance Predictor για τροφοδότηση από τους μηχανισμούς συλλογής αποστάσεων επαναχρησιμοποίησης που αναλύθηκαν στις προηγούμενες παραγράφους. Η πρώτη μπάρα κάθε ομάδας αντιστοιχεί στον ιδανικό Full Sampler, η δεύτερη στον SPS μόνον του και η τρίτη στον συνδυασμό SPS και SRDS. Βλέπουμε ότι ο SPS από μόνος του συμπεριφέρεται αρκετά καλά: στα περισσότερα προγράμματα οι διαφορές μεταξύ του Full Sampler και του SPS είναι ελάχιστες, ενώ υπάρχουν και 3 προγράμματα (gzip, mgrid, apsi) όπου ο SPS πετυχαίνει μεγαλύτερη ακρίβεια από ότι ο Full Sampler, κυρίως γιατί ο SPS είναι λιγότερο εναίσθητος σε παροδικές αλλαγές της συμπεριφοράς του προγράμματος. Στις υπόλοιπες περιπτώσεις όπου υπάρχει αισθητή πτώση της ακρίβειας λόγω του SPS, προσθέτοντας και τον SRDS στον συνολικό μηχανισμό, μπορούμε να αναπληρώσουμε αρκετή από την χαμένη ακρίβεια, όπως βλέπουμε για παράδειγμα στο applu.

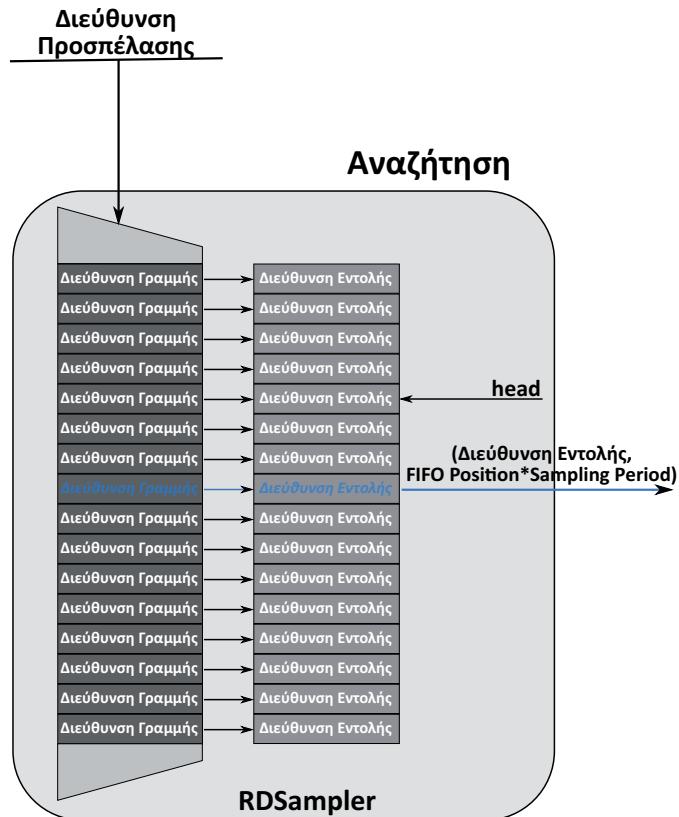
4.3.2 Μηχανισμός συλλογής αποστάσεων επαναχρησιμοποίησης με στόχο το χαμηλό κόστος

Ο μηχανισμός που παρουσιάστηκε στην προηγούμενη υποενότητα επιτυγχάνει να τροφοδοτήσει τον Predictor με πληροφορία παρόμοια με αυτήν που παράγει ο Full Sampler, αλλά εμφανίζει αρκετή πολυπλοκότητα και σχετικά υψηλό κόστος σε αποθηκευτικό χώρο¹. Για αρχιτεκτονικές υψηλής απόδοσης, αυτό το κόστος είναι γενικά ανεκτό, αλλά θεωρήθηκε σκόπιμο να αναπτυχθεί και ένας δεύτερος, αρκετά πιο απλός, μηχανισμός, ο οποίος μπορεί να δώσει αντίστοιχα καλά αποτελέσματα.

Ο μηχανισμός αυτός, με το όνομα RDSSampler, είναι παρόμοιος με τον Small Reuse-Distance Sampler: είναι και αυτός οργανωμένος σαν μία μικρή FIFO ουρά, η οποία δειγματοληπτεί τις προσπελάσεις της κρυφής μνήμης. Όμως διαφέρει σε τρία σημαντικά σημεία:

- Η συχνότητα δειγματοληψίας ($1/N$) του RDSSampler είναι πολύ μικρότερη από αυτή του SRDS και το μέγεθος του μερικές φορές μεγαλύτερο, ώστε η μέγιστη απόσταση επαναχρησιμοποίησης που μπορεί να συλλάβει ο RDSSampler ($FIFO_size \times sampling_period$) να καλύπτει ολόκληρο το εύρος αποστάσεων επαναχρησιμοποίησης που μας ενδιαφέρουν, και όχι μόνο τις μικρές αποστάσεις επαναχρησιμοποίησης.

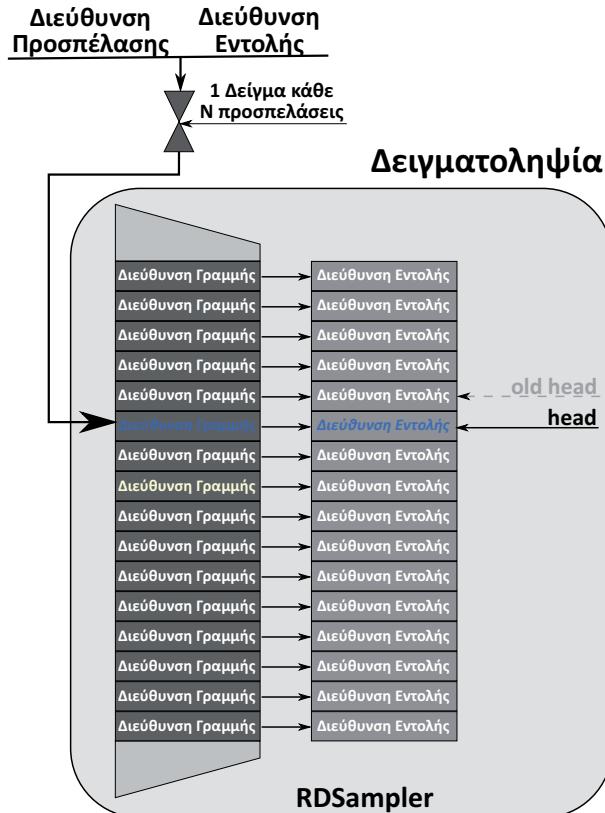
1. Το ακριβές κόστος όλων των μηχανισμών αναλύεται στις επόμενες ενότητες, οι οποίες αφορούν τις παραμέτρους και τις επιλογές των τελικών υλοποιήσεων



Σχήμα 36: RDSampler-Αναζήτηση

- Η δειγματοληψία γίνεται ακριβώς, και όχι κατά μέσο όρο, κάθε N προσπελάσεις, ώστε να μπορούμε να εξάγουμε την απόσταση επαναχρησιμοποίησης από την θέση της εγγραφής στην FIFO ουρά ($reuse_distance = fifo_position \times sampling_period$). Με αυτόν τον τρόπο αποφεύγουμε την ανάγκη να αποθηκεύουμε μαζί με το δείγμα την χρονική στιγμή όπου συνέβη η δειγματοληπτημένη προσπέλαση, μειώνοντας έτσι σημαντικά τις απαιτήσεις του μηχανισμού σε χώρο.
- Δείγματα τα οποία απομακρύνονται από την ουρά χωρίς να έχει προσδιοριστεί η απόσταση επαναχρησιμοποίησής τους, δεν θεωρούμε ότι αντιστοιχούν απλά σε μεγάλη απόσταση επαναχρησιμοποίησης, αλλά στη μέγιστη απόσταση επαναχρησιμοποίησης που παράγει ο RDSampler.

Με αυτήν την απλή οργάνωση αποφεύγεται η πολυπλοκότητα του μηχανισμού SPS+SRDS. Δεν χρειάζεται αποθήκευση δειγμάτων στον Predictor, πλήρως συσχετιστικές αναζητήσεις στον μεγάλο SPS ή συμψηφισμός μικρών και μεγάλων αποστάσεων επαναχρησιμοποίησης —όλη η συλλογή των αποστάσεων επαναχρησιμοποίησης γίνεται από μία σχετικά μικρή ουρά. Η δομή του RDSampler και οι δύο λειτουργίες που υποστηρίζει (δειγματοληψία και σύγκριση) παρουσιάζονται στο Σχήμα 36 και στο Σχήμα 37.



Σχήμα 37: RDSampler-Δειγματοληψία

Το μόνο σημαντικό μειονέκτημα του RDSampler σε σχέση με τον συνδυασμό SPS+SRDS είναι η σύνδεση της ποιότητας της πληροφορίας που τροφοδοτεί στον Predictor με τον ρυθμό δειγματοληψίας. Όσο μικρότερη είναι η περίοδος δειγματοληψίας, τόσο περισσότερα δείγματα παίρνονται και τόσο περισσότερες ανανεώσεις στέλνονται στον Predictor, αλλά πρέπει να μεγαλώσει αντίστοιχα ο αριθμός εγγραφών του RDSampler για να μπορούμε να συλλάβουμε το ίδιο εύρος από αποστάσεις επαναχρησιμοποίησης ($max_reuse_distance = FIFO_size \times sampling_period$). Και αντίστροφα, όσο μεγαλώνει η περίοδος δειγματοληψίας, τόσες λιγότερες εγγραφές χρειαζόμαστε για να καλύψουμε το ίδιο εύρος αποστάσεων επαναχρησιμοποίησης, αλλά στέλνουμε ανανεώσεις προς τον Predictor πιο σπάνια. Επιπλέον, η περίοδος δειγματοληψίας τοποθετείται κάτω όριο στην ανάλυση των αποστάσεων επαναχρησιμοποίησης που μετράει ο RDSampler και επομένως όσο πιο αραιά δειγματοληπτούμε τις προσπελάσεις της κρυφής μνήμης, τόσο μειώνεται η ακρίβεια των μετρήσεών μας.

Για παράδειγμα, αν χρειαζόμαστε να καλύψουμε το εύρος αποστάσεων επαναχρησιμοποίησης από 0 έως 1M (όπως ο σύνθετος sampler), μπορούμε να το επιτύχουμε με 256 εγγραφές και 4K περίοδο, 128 εγγραφές και 8K περίοδο, 64 εγγραφές και

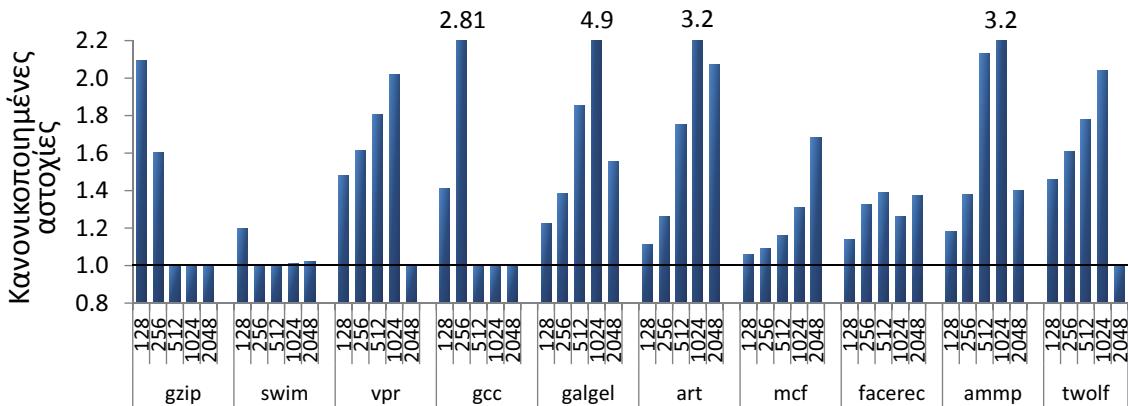
16K περίοδο ή 32 εγγραφές και 32K περίοδο. Με δεδομένο, ότι η μέγιστη περίοδος δειγματοληψίας, που δεν αλλάζει σημαντικά τα στατιστικά χαρακτηριστικά των δειγμάτων σε σχέση τα χαρακτηριστικά του συνόλου των προσπελάσεων, είναι της τάξης του 1K (σύμφωνα με την StatCache μεθοδολογία), γίνεται εμφανές ότι δεν μπορεί ο RDSSampler να πετύχει ταυτόχρονα: i) και ικανοποιητικό ρυθμό δειγματοληψίας, ii) και μεγάλο εύρος αποστάσεων επαναχρησιμοποίησης, iii) και μικρότερο μέγεθος από τον SPS. Αναγκαστικά, η υλοποίηση του RDSSampler θα πρέπει να θυσιάσει τουλάχιστον έναν από τους τρεις αυτούς στόχους.

4.4 Πολιτική Αντικατάστασης της Κρυφής Μνήμης Τελευταίου Επιπέδου

Οι μηχανισμοί συλλογής και πρόβλεψης των αποστάσεων επαναχρησιμοποίησης που περιγράφηκαν στις προηγούμενες ενότητες μπορούν να χρησιμοποιηθούν για διάφορες βελτιστοποιήσεις. Για παράδειγμα, σε διπλωματικές εργασίες έχουν υλοποιηθεί μηχανισμοί συμπίεσης της κρυφής μνήμης, που συμπιέζουν τα δεδομένα με μεσαίου μεγέθους προβλεπόμενες αποστάσεις επαναχρησιμοποίησης, μηχανισμοί για διαχείριση NUCA κρυφών μνημών, όπου οι γραμμές που προβλέπεται να χρησιμοποιηθούν σύντομα τοποθετούνται στα πιο γρήγορα κομμάτια της κρυφής μνήμης, και μηχανισμοί Cache Decay, όπου οι γραμμές που προβλέπεται ότι δεν χωράνε στην κρυφή μνήμη απενεργοποιούνται άμεσα, αντί για να περιμένουμε Decay Time κύκλους. Σε αυτό το κεφάλαιο, όμως, θα επικεντρωθούμε στην αποτελεσματική διαχείριση των αντικαταστάσεων της κρυφής μνήμης τελευταίου επιπέδου.

4.4.1 Βέλτιστος αλγόριθμος αντικατάστασης και LRU

Το βασικό κίνητρό μας για την υλοποίηση μίας νέας πολιτικής αντικατάστασης είναι η μη ικανοποιητική συμπεριφορά των κλασικών πολιτικών αντικατάστασης και ειδικότερα του LRU στις κρυφές μνήμες των κατώτερων επιπέδων. Όπως αναφέρθηκε στην αρχή του κεφαλαίου, ο βασικός λόγος για την αδυναμία του LRU να διαχειριστεί αποδοτικά τις κρυφές των κατωτέρων επιπέδων είναι ότι το LRU υποθέτει υψηλή τοπικότητα στις προσπελάσεις, τοπικότητα που συνήθως φιλτράρεται από τις κρυφές μνήμης του πρώτου επιπέδου και δεν φτάνει στα κατώτερα επίπεδα.



Σχήμα 38: Κανονικοποιημένες, ως προς την βέλτιστη πολιτική αντικατάστασης, αστοχίες των προγραμμάτων για διαχείριση με LRU

Για να δείξουμε αυτήν την έλλειψη αποδοτικότητας του LRU, δείχνουμε στο Σχήμα 38 την σύγκριση του ποσοστού αστοχιών μεταξύ του LRU και του θεωρητικά βέλτιστου αλγόριθμου αντικατάστασης του Belady[7].Ο βέλτιστος αλγόριθμος αντικατάστασης αναζητεί την γραμμή που θα χρησιμοποιηθεί πιο μακριά στο μέλλον και την αντικαθιστά. Προφανώς, εφόσον απαιτεί τέλεια γνώση της μελλοντικής συμπεριφοράς των γραμμών, δεν μπορεί να χρησιμοποιηθεί για την διαχείριση μίας πραγματικής κρυφής μνήμης, αλλά είναι χρήσιμος γιατί μπορεί να μας δώσει ένα κάτω όριο για την εφικτή μείωση των αστοχιών ενός προγράμματος. Στο Σχήμα βλέπουμε ότι στις περιπτώσεις όπου το πρόγραμμα χωράει στην κρυφή μνήμη ή απέχει πολύ από το να χωρέσει (επίπεδες περιοχές των καμπύλων Stat-Cache), το LRU καταφέρνει και φτάνει κοντά στην βέλτιστη πολιτική αντικατάστασης. Άλλα όταν το πρόγραμμα αρχίζει και χωράει στην κρυφή μνήμη, τότε το LRU παράγει πολύ περισσότερες αστοχίες (μέχρι και 5 φορές περισσότερες αστοχίες).

Αυτή η απόσταση ανάμεσα στο LRU και στην βέλτιστη πολιτική αντικατάστασης δίνει το έναυσμα για την πολιτική αντικατάστασης που υλοποιήσαμε. Η πρόβλεψη των αποστάσεων επαναχρησιμοποίησης μας επιτρέπει να προβλέψουμε ποια γραμμή θα χρησιμοποιηθεί πιο μακριά στο μέλλον, οπότε μπορούμε να προσεγγίσουμε την βέλτιστη πολιτική αντικατάστασης και έτσι να μειώσουμε σημαντικά τις αστοχίες της κρυφής μνήμης σε σχέση με την LRU πολιτική. Αυτή είναι και η σημαντικότερη ποιοτική διαφορά ανάμεσα στην μεθοδολογία μας και άλλες μεθοδολογίες πρόβλεψης που στοχεύουν στην διαχείριση της κρυφής μνήμης [69,94,51]: μόνο η μεθοδολογία μας ποσοτικοποιεί την τοπικότητα των προσπελάσεων με τέτοιο τρόπο ώστε να είναι εφικτή η προσομοίωση της βέλτιστης πολιτικής αντικατάστασης.

4.4.2 Αλγόριθμος της πολιτικής αντικατάστασης

Ας υποθέσουμε αρχικά ότι ο Predictor επιτυγχάνει ακρίβεια και κάλυψη της τάξης του 100%. Σε αυτήν την περίπτωση, για να μπορέσουμε να εφαρμόσουμε μία σχεδόν βέλτιστη πολιτική αντικατάστασης, χρειαζόμαστε να αποθηκεύουμε σε κάθε γραμμή της κρυφής μνήμης μόνο δύο πληροφορίες: την χρονική στιγμή (μετρημένη σε προσπελάσεις κρυφής μνήμης) οπότε συνέβη η τελευταία προσπέλαση της γραμμής και την πρόβλεψη της απόστασης επαναχρησιμοποίησης. Με αυτές τις δύο τιμές μπορούμε να υπολογίσουμε πότε αναμένεται η επόμενη προσπέλαση κάθε γραμμής, και έτσι να διαλέξουμε για αντικατάσταση την γραμμή που θα προσπελαστεί πιο μακριά στο μέλλον, ακριβώς όπως και η βέλτιστη πολιτική αντικατάστασης.

Το κόστος μίας τέτοιας λογικής αντικατάστασης είναι αρκετά χαμηλότερο από ότι φαίνεται αρχικά. Αντί για αριθμητικές πράξεις για τον υπολογισμό της στιγμής που θα συμβεί η επόμενη προσπέλαση, μπορούμε απλά να αποθηκεύουμε την πρόβλεψη και, με λογική αντίστοιχη με του Cache Decay, να μειώνουμε την αποθηκευμένη τιμή μετά από κάθε προσπέλαση. Με αυτόν τον τρόπο, η τιμή που βρίσκεται αποθηκευμένη στην γραμμή δείχνει ανά πάσα χρονική στιγμή το πόσες προσπελάσεις μένουν μέχρι την επόμενη χρήση της γραμμής, οπότε η πολιτική αντικατάστασης μπορεί απλά να διαλέξει την γραμμή με την μεγαλύτερη αποθηκευμένη τιμή. Επιπλέον, οι προβλέψεις που αποθηκεύουμε, όπως αναφέρθηκε και στην ενότητα 4.2, είναι κβαντισμένες όποτε ο αριθμός bits που χρειαζόμαστε για να αποθηκεύσουμε την πληροφορία της πολιτικής αντικατάστασης είναι μικρός.

Τέλος, είναι σημαντική η συνειδητοποίηση του ότι αποστάσεις επαναχρησιμοποίησης με ανάλυση μικρότερη των 256 προσπελάσεων παρέχουν ασήμαντα οφέλη στην ποιότητα της διαχείρισης που μπορούμε να πετύχουμε: η μικρότερη κρυφή μνήμη που μελετάμε έχει 256 sets, οπότε υποθέτοντας σχετικά ομοιόμορφη κατανομή των προσπελάσεων κρυφής μνήμης, η κατάσταση του κάθε set θα αλλάζει μόνο μία φορά κάθε 256 προσπελάσεις. Επιπροσθέτως, λόγω των διακυμάνσεων στις τιμές των αποστάσεων επαναχρησιμοποίησης και στην συμπεριφορά του προγράμματος και του επεξεργαστή γενικότερα, ο “θόρυβος” του συστήματος είναι ούτως ή άλλως της τάξης των 16-64 προσπελάσεων. Επομένως, η ακρίβεια που παρέχουν οι υπερβολικά λεπτομερείς προβλέψεις, δεν μπορεί να αξιοποιηθεί για την διαχείριση της κρυφής μνήμης. Με βάση αυτήν την κατανόηση, μπορούμε αφενός να μειώσουμε την ακρίβεια των προβλέψεων και αφετέρου να μειώνουμε την αποθηκευμένη

απόσταση μέχρι την επόμενη χρήση της κάθε γραμμής κάθε 256 προσπελάσεις αντί μετά από κάθε προσπέλαση. Το χρονικό εύρος των 256 προσπελάσεων αντιστοιχεί, κατά μέσο όρο, σε μερικές χιλιάδες έως μερικές δεκάδες χιλιάδες κύκλους ρολογιού, εύρος αντίστοιχο με αυτό που μεσολαβεί ανάμεσα στις μειώσεις των τοπικών μετρητών του Cache Decay, και επομένως το ενεργειακό κόστος της πολιτικής αντικατάστασης θα είναι αντίστοιχο με της τεχνικής Cache Decay, δηλαδή ελάχιστο.

Η πολιτική αντικατάστασης που περιγράφηκε, απαιτεί τέλεια πρόβλεψη για όλες τις γραμμές της κρυφής μνήμης. Όπως είδαμε, όμως, στις προηγούμενες ενότητες ο μηχανισμός πρόβλεψης που υλοποιήσαμε δεν έχει ποτέ 100% κάλυψη και μόνο σπάνια η ακρίβειά του αγγίζει το 100%. Αν εφαρμόζαμε την πολιτική αντικατάστασης ως έχει και την οδηγούσαμε με βάση την ατελή πληροφορία που παράγει ο Predictor, τότε:

- i. Δεν θα είχαμε κάποιον μηχανισμό για την αντικατάσταση των γραμμών χωρίς πρόβλεψη
- ii. Γραμμές με πολύ μεγάλες αποστάσεις επαναχρησιμοποίησης για τις οποίες λανθασμένα προβλέψαμε σύντομη επαναχρησιμοποίηση, θα έμεναν αδρανείς στην κρυφή μνήμη για πολύ μεγάλα χρονικά διαστήματα.

Για να χειριστούμε περιπτώσεις σαν αυτές, παίρνουμε υπόψιν και την LRU πληροφορία, διαλέγοντας την LRU γραμμή σαν μία δεύτερη υποψήφια για αντικατάσταση γραμμής. Η χρήση της LRU πληροφορίας εξασφαλίζει ότι οι γραμμές με λάθος ή καθόλου πρόβλεψη θα αντικατασταθούν κάποτε από την κρυφή μνήμη. Για να διαλέξουμε ανάμεσα στις δύο υποψήφιες προς αντικατάσταση γραμμές, αυτήν που έχει μείνει περισσότερο χρόνο αδρανής στο παρελθόν και αυτήν που θα μείνει αχρησιμοποίητη στο μέλλον για τον περισσότερο χρόνο, συγκρίνουμε τους χρόνους παρελθοντικής και μελλοντικής αδράνειάς τους. Αν η γραμμή που θα χρησιμοποιηθεί πιο μακριά στο μέλλον, προβλέπεται να μείνει αδρανής για χρόνο μεγαλύτερο από αυτόν της LRU γραμμής, τότε είναι λογικό να υποθέσουμε ότι η γραμμή αυτή, αν δεν αντικατασταθεί τώρα, θα φτάσει κάποτε να γίνει LRU γραμμή και να αντικατασταθεί ούτως ή άλλως, οπότε μας συμφέρει να την αντικαταστήσουμε τώρα. Αντίθετα, αν η γραμμή που θα αντικατασταθεί πιο μακριά στο μέλλον, προβλέπεται να μείνει αδρανής για χρόνο μικρότερο από αυτόν της LRU γραμμής, αυτό σημαίνει ότι κατά πάσα περίπτωση η επαναχρησιμοποίηση της γραμμής θα γίνει πριν αυτή φτάσει στο τέλος της LRU αλυσίδας και άρα η προσπέλασή της θα προκαλέσει ευστοχία, οπότε μας συμφέρει να αντικαταστήσουμε την LRU γραμμή.

Για την υλοποίηση της LRU λογικής ιδανικά χρειαζόμαστε να διατηρούμε την πλήρη LRU σειρά των γραμμών κάθε set και να αποθηκεύουμε σε κάθε γραμμή την χρονική στιγμή της τελευταίας προσπέλασης. Μπορούμε, όμως, να προσεγγίσουμε πάρα πολύ καλά αυτήν την πληροφορία χρησιμοποιώντας μόνο τις χρονικές στιγμές τελευταίας προσπέλασης κβαντισμένες με τον ίδιο τρόπο όπως και οι προβλέψεις ή ακόμη καλύτερα προσθέτοντας μετρητές παρόμοιους με αυτούς όπου αποθηκεύουμε την πρόβλεψη, τους οποίους αρχικοποιούμε σε μηδέν και αυξάνουμε κάθε 256 προσπελάσεις. Με μία τέτοια υλοποίηση, η μία ομάδα μετρητών μετράει τον χρόνο που απομένει για την επόμενη προσπέλαση και η άλλη ομάδα μετράει τον χρόνο που έχει περάσει από την τελευταία προσπέλαση, οπότε η πολιτική αντικατάστασης αρκεί να αναζητήσει την μεγαλύτερη τιμή μετρητή μεταξύ και των δύο ομάδων μετρητών και να αντικαταστήσει την αντίστοιχη γραμμή.

Μία τελική επέκταση της πολιτικής αντικατάστασης είναι να παίρνουμε υπόψιν κατά την αναζήτηση γραμμής προς αντικατάσταση και την γραμμή της οποίας η προσπέλαση προκάλεσε την τρέχουσα εξυπηρετούμενη αστοχία. Αν αυτή γραμμή προβλέπεται να μείνει αδρανής στην κρυφή μνήμη για χρόνο μεγαλύτερο από τον χρόνο που έχουν μείνει ή που θα μείνουν αδρανείς όλες οι γραμμές του set, τότε επιλέγουμε να μην την τοποθετήσουμε στην κρυφή μνήμη και απλά την προωθούμε στα ανώτερα επίπεδα της ιεραρχίας μνήμης. Στις επόμενες ενότητες αποκαλούμε αυτήν την επέκταση Selective Caching (SC).

4.5 Επιλογές Υλοποίησης

4.5.1 Κβάντιση των αποστάσεων επαναχρησιμοποίησης

Στις προηγούμενες ενότητες αναφέραμε την χρήση κβαντισμένων αποστάσεων επαναχρησιμοποίησης ώστε να μειώσουμε τον χώρο που απαιτείται για την αποθήκευσή τους στον Predictor και στις γραμμές της κρυφής μνήμης. Στην παρούσα μεθοδολογία χρησιμοποιούμε δύο διαφορετικές τεχνικές για την κβάντιση των αποστάσεων επαναχρησιμοποίησης, λογαριθμική και γραμμική κβάντιση.

Η λογαριθμική κβάντιση ομαδοποιεί τις αποστάσεις επαναχρησιμοποίησης σε ομάδες με εκθετικά αυξανόμενα εύρη: [0,2), [2,4), [4,8), [8,16) και ούτω καθεξής¹. Με αυτήν την αντιστοιχία η απόσταση επαναχρησιμοποίησης αντικαθιστάται από τον αύξων αριθμό της

1. Κανονικά για εκθετικά αυξανόμενα εύρη η ακολουθία των ομάδων είναι [0,1), [1,3), [3,7), [7,15) κοκ, αλλά την τροποποιήσαμε ώστε να απλοποιηθεί η κβάντιση.

ομάδας στην οποία ανήκει η αρχική απόσταση επαναχρησιμοποίησης, ή ισοδύναμα η απόσταση επαναχρησιμοποίησης αντικαθιστάται από τον δυαδικό λογάριθμό της. Το πλεονέκτημα της λογαριθμικής κβάντιση είναι ότι καταφέρνει να συμπιέσει πολύ μεγάλα εύρη αποστάσεων επαναχρησιμοποίησης σε λίγα bits. Για παράδειγμα, το εύρος από 0 έως $1M-1$, αντιστοιχεί σε 19 ομάδες αποστάσεων επαναχρησιμοποίησης και επομένως απαιτούνται 5 bits. Επιπλέον, αν αξιοποιηθεί το ότι ο μηχανισμός αντικατάστασης δεν απαιτεί ακρίβεια μικρότερη των 256 προσπελάσεων, οι απαιτούμενες ομάδες μειώνονται στις 12 και τα bits σε 4.

Η λογαριθμική κβάντιση όμως παρουσιάζει ένα μειονεκτήματα. Εφόσον οι μετρητές διαχείρισης των γραμμών δεν μετράνε γραμμικά, απαιτείται ο κεντρικός μετρητής να παράγει πολλαπλά σήματα προς τις γραμμές (και όχι μόνο ένα όπως στο Cache Decay και στην StatShare) και αυτά τα σήματα να φιλτράρονται από κατάλληλη συνδυαστική λογική προκειμένου να αποφασιστεί αν θα μειωθεί/αυξηθεί η τιμή του τοπικού μετρητή. Για παράδειγμα, αν ένας μετρητής του χρόνου αδράνειας έχει τιμή 5 (δηλαδή 32 έως 63 προσπελάσεις), τότε θα πρέπει να αυξηθεί όταν περάσουν 32 ακόμη προσπελάσεις, ώστε να πάρει την τιμή 6 που αντιστοιχεί σε 64 έως 127 προσπελάσεις. Επομένως, ο κεντρικός μετρητής θα πρέπει να παράγει ένα σήμα αύξησης κάθε 32 προσπελάσεις. Όμως για να κάνει την επόμενη μετάβαση από την τιμή 6 στην τιμή 7 θα πρέπει να περιμένει 64 προσπελάσεις, οπότε απαιτείται και σήμα για κάθε 64 προσπελάσεις. Γενικότερα, ο κεντρικός μετρητής θα πρέπει να εξάγει μία γραμμή για κάθε πιθανή ομάδα αποστάσεων επαναχρησιμοποίησης και το ποια γραμμή θα προκαλέσει μεταβολή της κατάστασης του μετρητή θα πρέπει να επιλέγεται συνδυαστικά βάσει της τρέχουσας τιμής του μετρητή. Αυτή η συμπεριφορά αυξάνει κάπως την πολυπλοκότητα και το ενεργειακό κόστος της μεθοδολογίας μας, αλλά θεωρούμε ότι παραμένουν σε αποδεκτά επίπεδα.

Η γραμμική μέθοδος κβάντισης σχεδιάστηκε με στόχο εκείνες τις υλοποιήσεις της μεθοδολογίας μας όπου το κόστος της λογαριθμικής κβάντισης δεν είναι ανεκτό. Η γραμμική κβάντιση ομαδοποιεί τις αποστάσεις επαναχρησιμοποίησης σε ομάδες με σταθερά εύρη που καλύπτουν διαδοχικές περιοχές αποστάσεων επαναχρησιμοποίησης, το οποίο ισοδυναμεί αριθμητικά με απομόνωση κάποιων από τα bits της απόστασης επαναχρησιμοποίησης. Εφόσον τα μεγέθη όλων των ομάδων είναι τα ίδια, όλοι οι μετρητές που αποθηκεύουν κβαντισμένες αποστάσεις επαναχρησιμοποίησης θα αλλάζουν κατάσταση ταυτόχρονα και με το ίδιο σήμα του κεντρικού μετρητή, και επομένως αποφεύγεται η πολυπλοκότητα της λογαριθμικής κβάντισης.

Το μεγάλο μειονέκτημα της γραμμικής κβάντισης είναι ότι για να καλύψουμε το εύρος αποστάσεων επαναχρησιμοποίησης που χρειαζόμαστε, απαιτείται ή μεγάλος αριθμός γραμμικών ομάδων με ικανοποιητική ανάλυση ή λίγες ομάδες με χαμηλή ανάλυση. Για παράδειγμα, αν πρέπει να καλύψουμε το εύρος 0 έως 1M-1 χρησιμοποιώντας 4 bits όπως η λογαριθμική κβάντιση, το πλάτος της κάθε ομάδας πρέπει να είναι 64K, το οποίο είναι υπερβολικά μεγάλο στις περισσότερες περιπτώσεις, ενώ αν θέλουμε να πετύχουμε καλύτερη ανάλυση, για παράδειγμα 4K, απαιτούνται 256 ομάδες (8 bits).

Ένας τρόπος να αντιμετωπιστεί μερικώς αυτό το πρόβλημα, είναι να περιοριστεί το εύρος των αποστάσεων επαναχρησιμοποίησης που καλύπτουμε στις περιοχές που έχουν μεγάλη πιθανότητα να είναι κρίσιμες για την σωστή διαχείριση της κρυφής μνήμης. Από την πειραματική μελέτη του μηχανισμού αντικατάστασης που προτείνουμε, προέκυψε ότι, για τα περισσότερα προγράμματα, αποστάσεις επαναχρησιμοποίησης μικρότερες από $L/4-L$ προκαλούν πάντα ευστοχίες και αποστάσεις επαναχρησιμοποίησης μεγαλύτερες από $4*L-16*L$ προκαλούν πάντα αστοχίες, με L το μέγεθος της κρυφής μνήμης μετρημένο σε γραμμές. Αυτό εναλλακτικά σημαίνει ότι η περιοχή που πραγματικά μας ενδιαφέρει να περιγράψουμε με ικανοποιητική ακρίβεια βρίσκεται στο εύρος $L/4-16*L$. Στην υλοποίηση χαμηλού κόστος που προτείνουμε αργότερα σε αυτήν την ενότητα, χρησιμοποιήσαμε 16 ομάδες πλάτους $L/2$ και επομένως οι γραμμικά κβαντισμένες αποστάσεις επαναχρησιμοποίησης απαιτούν 4 bit και καλύπτουν το εύρος $L/2$ έως $8*L$, το οποίο είναι μία αρκετά καλή εξισορρόπηση της ανάγκης μας για υψηλή ανάλυση και του στόχου μας για χαμηλό κόστος σε αποθηκευτικό χώρο.

4.5.2 Υλοποίηση υψηλής απόδοσης

Στα πλαίσια της μεθοδολογίας μας αναπτύχθηκαν δύο ξεχωριστές υλοποιήσεις του συνολικού μηχανισμού διαχείρισης της κρυφής μνήμης. Η πρώτη και βασική μας υλοποίηση αναπτύχθηκε με στόχο την μέγιστη εφικτή μείωση των αστοχιών κρυφής μνήμης υποθέτοντας ικανοποιητικό, αλλά όχι μεγάλο, διαθέσιμο χώρο για την αποθήκευση των προβλέψεων και των μετρητών μας. Αντίθετα, η δεύτερη υλοποίηση είχε σαν στόχο την εξαγωγή των καλύτερων δυνατών αποτελεσμάτων κάτω από προαποφασισμένους και εξαιρετικά στενούς περιορισμούς στον διαθέσιμο χώρο και στην αύξηση της πολυπλοκότητας.

Η υλοποίηση υψηλής απόδοσης αποτελείται από τον συνδυασμό:

- ενός Instruction-based Reuse Distance Predictor 256 εγγραφών, οργανωμένων σε 8 δρόμους, με τις προβλέψεις κβαντισμένες λογαριθμικά, μετρητές αξιοπιστίας των 2 bits και κατώφλι πρόβλεψης την τιμή 1
- ενός Single Prediction Sampler, ενσωματωμένο στον Predictor
- ενός Small Reuse Distance Sampler, 8 εγγραφών που δειγματοληπτεί 1 στις 256 προσπελάσεις κρυφής μνήμης
- και ενός ζεύγους μετρητών που κρατάνε τον λογαριθμικά κβαντισμένο χρόνο μέχρι την επόμενη προσπέλαση και τον λογαριθμικά κβαντισμένο χρόνο που έχει μεσολαβήσει από την τελευταία προσπέλαση, για κάθε γραμμή της κρυφής μνήμης.

Ο Predictor μαζί με τον SPS χρειάζονται 67 bits ανά εγγραφή (τα 25 σημαντικότερα bits της διεύθυνσης της εντολής¹, τα 26 σημαντικότερα bits της διεύθυνσης που έχουμε δειγματοληπτήσει, 5 bits η κβαντισμένη χρονική στιγμή κατά την οποία πήραμε το δείγμα, 5 bits η κβαντισμένη πρόβλεψη, 2 bits ο μετρητής αξιοπιστίας, 3 bits για την lru αντικατάσταση των εγγραφών του Predictor και 1 valid bit), οπότε συνολικά απαιτούν $256 \times 67 = 16.75$ Kbit (2.1 KB). Ο SRDS απαιτεί 62 bits ανά εγγραφή (26 bits για την δειγματοληπτημένη διεύθυνση, 30 bits για την διεύθυνση της εντολής, 5 bits για την χρονική στιγμή της προσπέλασης και ένα valid bit), οπότε απαιτεί $8 \times 62 = 496$ bits (62 bytes). Τέλος, για την υποστήριξη της πολιτικής αντικατάστασης χρειαζόμαστε 2 μετρητές των 5 bits σε κάθε γραμμή της κρυφής μνήμης, το οποίο μεταφράζεται σε κόστος από 40 Kbit (για την κρυφή μνήμη των 256KB) έως 320 Kbit (για την κρυφή μνήμη των 2MB). Αθροιστικά, το κόστος του μηχανισμού κυμαίνεται από 7.1KB (2.7% του χώρου της 256KB κρυφής μνήμης) έως 42.1KB (2.1% του χώρου της κρυφής μνήμης των 2MB).

4.5.3 Υλοποίηση χαμηλού κόστους

Η δεύτερη υλοποίηση αναπτύχθηκε με στόχο την συμμετοχή στο πρωτάθλημα πολιτικών αντικατάστασης για κρυφές μνήμες, που διεξήχθη στα πλαίσια του ISCA-37 [41]. Βασικός όρος του πρωταθλήματος ήταν ότι ο συνολικός χώρος που απαιτεί ο μηχανισμός δεν πρέπει να ξεπερνάει τα 129 Kbit (8 bit για κάθε μία από τις 16K γραμμές και 1 Kbit επιπλέον), δηλαδή λιγότερο από το 1.5% της 1MB κρυφής μνήμης πάνω στην οποία

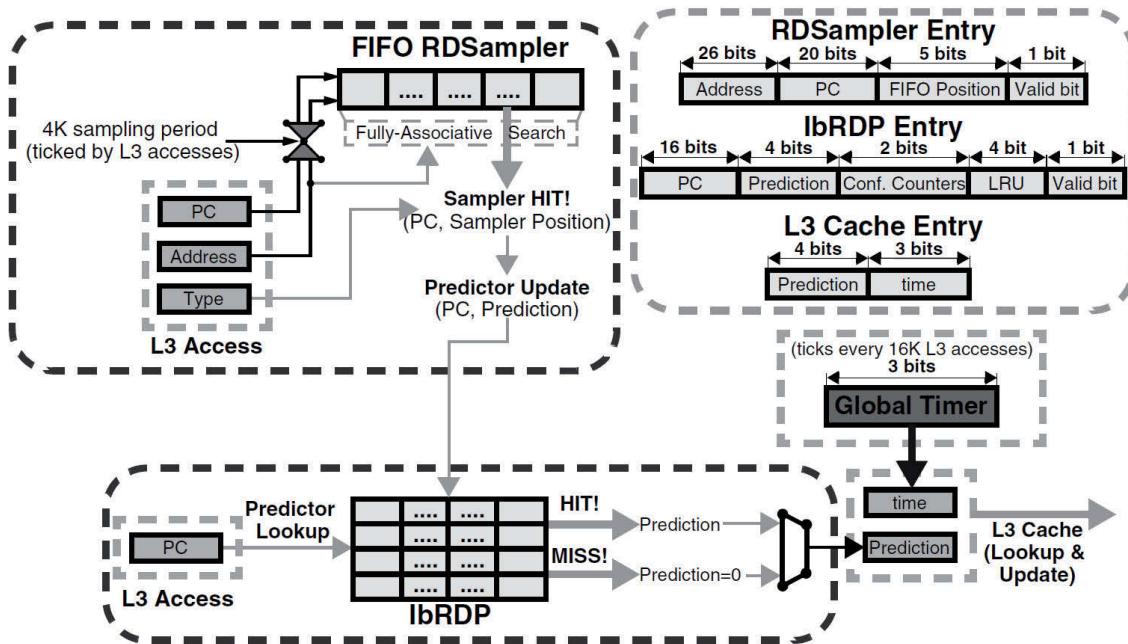
1. Από τα 32 bits της διεύθυνσης, τα 2 χαμηλότερα είναι πάντα μηδέν οπότε τα αγνοούμε και τα 5 αμέσως σημαντικότερα χρησιμοποιούνται σαν index του predictor.

συγκρίθηκαν οι μηχανισμοί. Εφόσον, η αρχική μας υλοποίηση απαιτούσε 33% περισσότερο χώρο από αυτόν που παρείχε η επιτροπή του πρωταθλήματος, υποχρεωθήκαμε να επέμβουμε δραστικά στην λειτουργία και την δομή του μηχανισμού.

Προκειμένου να μειωθούν οι απαιτήσεις του μηχανισμού μας, υλοποιήσαμε τον μηχανισμό συλλογής αποστάσεων επαναχρησιμοποίησης χαμηλού κόστους (RDSampler) που παρουσιάστηκε στην Ενότητα 4.3.2 και τις γραμμικά κβαντισμένες αποστάσεις επαναχρησιμοποίησης που παρουσιάστηκαν στην Ενότητα 4.5.1. Ο RDSampler υλοποιήθηκε με 32 εγγραφές και περίοδο δειγματοληψίας 4K προσπελάσεις (L/4 για L=16K), η οποία είναι 4 φορές μεγαλύτερη από την περίοδο του 1K που χρησιμοποιούμε στον αντίστοιχο Sampler της StatShare. Αυτή η επιλογή είχε σαν συνέπεια σχετικά αραιές ανανεώσεις προς τον Predictor και επηρέασε την ακρίβεια των προβλέψεων μας, αλλά δεν ήταν εφικτή η περαιτέρω μείωση της περιόδου δειγματοληψίας. Οι προβλέψεις αποστάσεων επαναχρησιμοποίησης κβαντίστηκαν σε 16 ομάδες σταθερού πλάτους 8K (L/2), ενώ ο μετρητής σε κάθε γραμμή που κρατάει την χρονική στιγμή της τελευταίας προσπέλασης κβαντίστηκε με ακόμη χαμηλότερη ανάλυση (16K), ώστε να εξοικονομηθεί ένα bit από κάθε γραμμή χωρίς να αλλάξει ο μέγιστος χρόνος που μπορεί να μετρήσει. Για την αποθήκευση των προβλέψεων χρησιμοποιήσαμε τον Instruction-based Reuse Distance Predictor, με 256 εγγραφές και σε αυτήν την υλοποίηση, αλλά με 16 δρόμους πια αντί για 8. Η τιμή κατωφλίου των μετρητών αξιοπιστίας του Predictor μειώθηκε σε 0, ώστε να αντιμετωπίσουμε την πολύ αργή αύξηση των μετρητών των νέων εγγραφών λόγω των αραιών ανανεώσεων. Η πολιτική αντικατάστασης παρέμεινε στην πράξη η ίδια, με μόνη αλλαγή ότι εφαρμόζουμε Selective Caching και όταν η πρόβλεψη της απόστασης επαναχρησιμοποίησης της γραμμή που φέρνουμε από την κύρια μνήμη έχει την μέγιστη κβαντισμένη τιμή. Η δομή και η οργάνωση του συνολικού μηχανισμού φαίνεται στο Σχήμα 39.

Το κόστος σε χώρο αυτής της υλοποίησης αναλύεται ως εξής:

- **IbRDP:** Κάθε εγγραφή απαιτεί 16 bits για την διεύθυνση της εντολής (τα δύο χαμηλότερα είναι μηδέν, τα επόμενα 4 χρησιμοποιούνται σαν index και αγνοούμε τα bits 22 έως 63), 4 bits για την πρόβλεψη, 2 bits για τον μετρητή αξιοπιστίας, 4 bits για την lru πληροφορία των εγγραφών του set του Predictor και ένα valid bit, οπότε συνολικά κάθε εγγραφή απαιτεί 27 bits. Για ολόκληρο τον Predictor απαιτούνται 256 x 27 = 6192 bits (774 bytes).



Σχήμα 39: Δομή και οργάνωση του μηχανισμού συλλογής και πρόβλεψης, στην υλοποίηση χαμηλού κόστους

- RDSampler:** Κάθε εγγραφή περιέχει 26 bits για την διεύθυνση του δείγματος (τα κατώτερα 6 bits είναι offset μέσα στην γραμμή, αγνοούμε τα bits 32 έως 63), 20 bits για την διεύθυνση της εντολής (αγνοούμε όπως και πριν τα χαμηλότερα δύο bits και τα bits 22 έως 63), 5 bits για την θέση της εγγραφής στον fifo buffer και ένα valid bit, τα οποία μας δίνουν σύνολο 52 bits. Επομένως, για τον RDSampler χρειαζόμαστε $32 \times 52 = 1664$ bits (208 bytes).
- Κρυφή μνήμη:** Για την υποστήριξη της πολιτικής αντικατάστασης χρειαζόμαστε σε κάθε γραμμή 4 bits για την πρόβλεψη και 3 bits για την χρονική στιγμή της τελευταίας προσπέλασης. Οπότε για 16K γραμμές, απαιτούνται 112Kbits (14 KB).

Αθροιστικά, οι μηχανισμοί πρόβλεψης και συλλογής απαιτούν 7.67Kbit (982 bytes), δηλαδή 55% λιγότερο χώρο από ότι στην υλοποίηση υψηλής απόδοσης, ενώ για ολόκληρο τον μηχανισμό (μαζί δηλαδή με την πληροφορία διαχείρισης στην κρυφή μνήμη) χρειαζόμαστε 120.4 Kbits (15KB), πετυχαίνοντας συνολικά περίπου 32% μείωση του αποθηκευτικού χώρου.

4.6 Πειραματικά Αποτελέσματα

4.6.1 Πειραματική μεθοδολογία

Για την αποτίμηση της απόδοσης της προτεινόμενης μεθοδολογίας χρησιμοποιήσαμε τον εξομοιωτή Simplescalar. Ο επεξεργαστής που εξομοιώθηκε στα πειράματα είναι ένας 4-way superscalar επεξεργαστής με εκτέλεση εκτός σειράς και παράθυρο 80 εντολών. Το υποσύστημα μνήμης περιλαμβάνει μία κρυφή μνήμη δεδομένων πρώτου επιπέδου, δύο πορτών, μεγέθους 32KB με 64 bytes ανά γραμμή, οργανωμένη σε 4 δρόμους και με χρόνο πρόσβασης δύο κύκλους. Η κρυφή μνήμη που μελετάμε είναι η ενοποιημένη κρυφή μνήμη δεύτερου επιπέδου, η οποία είναι οργανωμένη σε 16 δρόμους και παρέχει χρόνο πρόσβασης 13 κύκλους. Το μέγεθος της L2 στα πειράματά μας μεταβάλλεται από 256KB έως και 2MB. Η κύρια μνήμη, τέλος, έχει χρόνο πρόσβασης 500 κύκλους και παρέχει 16 bytes κάθε 8 κύκλους.

Σαν είσοδο στον εξομοιωτή χρησιμοποιήσαμε όλες τις εφαρμογές της σουίτας SPEC2000, αλλά σε αυτή την ενότητα επικεντρωνόμαστε στις 18 εφαρμογές που παράγουν περισσότερες από 1 αστοχίες κρυφής μνήμης ανά 1K εντολές, για την μικρότερη κρυφή μνήμη που μελετάμε. Προγράμματα με λιγότερες από 1 αστοχίες ανά 1K εντολές έχουν εξαιρετικά χαμηλές απαιτήσεις σε χώρο κρυφής μνήμης και δεν επωφελούνται ιδιαίτερα από την βελτίωση της πολιτικής αντικατάστασης. Όλα τα στατιστικά αφορούν 200M εξομοιωμένων εντολών, μετά από 100M εξομοίωσης για να φτάσουν οι κρυφές μνήμες σε σταθερή κατάσταση. Τέλος, για όλα τα προγράμματα αγνοήσαμε τις πρώτες 1B έως 3B εντολές, τα οποία αντιστοιχούν στην φάση αρχικοποίησης των προγραμμάτων.

Στα πλαίσια της πειραματικής μελέτης του μηχανισμού μας, υλοποιήσαμε στην ίδια πειραματική πλατφόρμα άλλες πέντε πολιτικές αντικατάστασης, τις οποίες μελετήσαμε και συγκρίναμε με την πολιτική αντικατάστασης που αναπτύξαμε.

Η τεχνική **Dynamic Insertion Policy (DIP)** [73], παρουσιάστηκε το 2007 από τους Qureshi et al. Η τεχνική αναγνωρίζει δυναμικά τις περιοχές του προγράμματος όπου η LRU πολιτική αντικατάστασης δεν λειτουργεί αποδοτικά και σε αυτές τις περιπτώσεις τροποποιεί την θέση εισαγωγής στην LRU στοίβα των γραμμών που εισάγονται στην κρυφή μνήμη. Πιο συγκεκριμένα η κρυφή μνήμη υλοποιεί δύο διαφορετικούς μηχανισμούς διαχείρισης της κρυφής μνήμης: LRU πολιτική αντικατάστασης και Bimodal Insertion Policy (BIP). Η BIP με στατιστικά τυχαίο τρόπο και πιθανότητα $\epsilon=1/32$ εισάγει τις νέες

γραμμές είτε στην MRU, είτε στην LRU θέση της LRU στοίβας (μία στις 32 γραμμές εισάγεται στην MRU θέση, οι υπόλοιπες στην LRU θέση). Ένα μικρό υποσύνολο των sets της κρυφής μνήμης λειτουργεί πάντα με LRU πολιτική και ένα άλλο ισομεγέθες υποσύνολο των sets λειτουργεί πάντα με BIP διαχείριση (leader sets). Η διαχείριση των υπόλοιπων sets της κρυφής μνήμης γίνεται με όποια από τις δύο πολιτικές επιτυγχάνει καλύτερα αποτελέσματα στα leader sets της. Η DIP πολιτική επιτυγχάνει τα καλύτερά της αποτελέσματα όταν το πρόγραμμα είναι αρκετά κοντά στο να χωρέσει στην κρυφή μνήμη. Σε τέτοια σενάρια, η DIP επιλέγει ένα υποσύνολο των δεδομένων του προγράμματος για εισαγωγή στην MRU θέση, ενώ τα υπόλοιπα δεδομένα μπαίνουν στην LRU θέση και γρήγορα αντικαθιστώνται, χωρίς να επηρεάζουν τις υπόλοιπες γραμμές της κρυφής μνήμης. Με αυτόν τον τρόπο τα δεδομένα που μπαίνουν στην MRU θέση μένουν περισσότερο χρόνο στην κρυφή μνήμη, με αποτέλεσμα να προκαλούν ευστοχίες. Πρακτικά, δηλαδή, η DIP προσπαθεί με στατιστικό τρόπο να επιλέξει το κομμάτι των γραμμών που παρουσιάζει την μεγαλύτερη τοπικότητα και να τα απομονώσει από τις συχνές αντικαταστάσεις που προκαλούν οι υπόλοιπες γραμμές.

Η τεχνική **Shepherd Cache** (SC) [74], είναι μία ενδιαφέρουσα προσέγγιση της βέλτιστης πολιτικής αντικατάστασης. Οι συγγραφείς προτείνουν τον χωρισμό της κρυφής μνήμης σε δύο κομμάτια: ένα κομμάτι λειτουργεί σαν κανονική κρυφή μνήμη, αλλά με λίγο μικρότερη συσχετιστικότητα από ότι η αρχική κρυφή μνήμη (Main Cache - MC), και μία μικρή βιοθητική Shepherd Cache η οποία μαζεύει την απαραίτητη πληροφορία για την προσέγγιση της βέλτιστης πολιτικής. Κάθε φορά που μία γραμμή εισάγεται στην κρυφή μνήμη, τοποθετείται στο Shepherd Cache κομμάτι, που χρησιμοποιεί FIFO λογική αντικατάστασης. Όσο η γραμμή παραμένει εκεί, καταγράφονται οι εύστοχιες του set και υπολογίζεται η σχετική σειρά τους σε σχέση με την προσπέλαση που έφερε την γραμμή στην κρυφή μνήμη. Όταν, τέλος, η γραμμή απομακρυνθεί από την Shepherd Cache, τοποθετείται στην Main Cache και αντικαθιστά την γραμμή για την οποία καταγράφηκε ότι χρησιμοποιείται πιο μακριά στο μέλλον σε σχέση με την γραμμή που μεταφέρουμε στην Main Cache. Η τεχνική πετυχαίνει τα καλύτερα αποτελέσματά της, όταν οι ακολουθίες των προσπελάσεων κρυφής μνήμης επαναλαμβάνονται με σχετικά σταθερό τρόπο (οπότε η καταγεγραμμένη ακολουθία προσπελάσεων προσεγγίζει καλά την μελλοντική ακολουθία προσπελάσεων) και αυτή η επαναλαμβανόμενη ακολουθία περιλαμβάνει λιγότερες αντικαταστάσεις ανά set από την συσχετιστικότητα της Shepherd Cache. Αν δεν ισχύει αυτό, τότε οι γραμμές που εισάγονται στην Shepherd Cache δεν παραμένουν εκεί για

αρκετό χρόνο ώστε να βρεθεί η πλήρης σχετική σειρά των προσπελάσεων του set, οπότε η προσέγγιση της βέλτιστης πολιτικής γίνεται μόνο για κάποιες από τις αντικαταστάσεις.

Η τεχνική **MLP-aware Replacement Policy** [72], που αναφέρθηκε ήδη στην Ενότητα 2.4, διαφέρει από τις τυπικές πολιτικές αντικατάστασης στο ότι στόχος της δεν είναι η εκμετάλλευση της τοπικότητας των προσπελάσεων, αλλά η εκμετάλλευση του παραλληλισμού μεταξύ των προσπελάσεων της κύριας μνήμης. Για κάθε γραμμή που εισάγεται στην κρυφή μνήμη, υπολογίζεται το MLP-cost της προσπέλασης που έφερε την γραμμή στην κρυφή μνήμη (περίπου ο χρόνος πρόσβασης της κύριας μνήμης δια τον αριθμό παράλληλων προσπελάσεων) και αποθηκεύεται μαζί με την γραμμή. Οι συγγραφείς δείχνουν ότι για τα περισσότερα προγράμματα, το MLP-cost της τελευταίας αστοχίας της γραμμής είναι αρκετά καλή προσέγγιση των MLP-costs των μελλοντικών αστοχιών της ίδιας γραμμής. Αυτή η πρόβλεψη εισάγεται στην συνάρτηση κόστους του MLP-aware Replacement Policy, η οποία προσπαθεί να “ζυγίσει” το MLP-cost και την θέση στην LRU στοίβα για κάθε γραμμή. Γραμμές στον πάτο της LRU στοίβας με χαμηλό MLP-cost είναι ιδανικοί υποψήφιοι για αντικατάσταση, καθώς παρουσιάζουν χαμηλή τοπικότητα και δεν θα μας στοιχίσει πολύ να ξαναφέρουμε την γραμμή στην κρυφή μνήμη όταν την χρειαστούμε. Αντίθετα, αν οι γραμμές στον πάτο της LRU στοίβας παρουσιάζουν πολύ υψηλό MLP-cost είναι αρκετά πιθανό να διαλέξουμε για αντικατάσταση γραμμές πιο ψηλά στην LRU στοίβα με χαμηλό κόστος MLP. Η βασική αυτή πολιτική επεκτείνεται με μία τεχνική, παρόμοια με αυτήν που χρησιμοποιεί το DIP, ώστε να επιλέγεται δυναμικά η καλύτερη πολιτική ανάμεσα στην LRU και την MLP-aware.

Η τεχνική **Variable Way Set-Associative Cache (V-Way)** [71], έχει σαν στόχο την μείωση των αστοχιών που οφείλονται στη μη-πλήρη συσχετιστικότητα των κρυφών μνημών, αυξάνοντας την συσχετιστικότητα των sets όπου βρίσκονται δυσανάλογα πολλές από τις συχνά χρησιμοποιούμενες γραμμές του προγράμματος. Οι συγγραφείς αρχικά σπάνε την στατική σύνδεση tags και data εγγραφών, ώστε να αποσυνδέσουν την γεωμετρία του tag κομματιού της κρυφής μνήμης από την γεωμετρία του data κομματιού, στην συνέχεια διπλασιάζουν τον αριθμό tag εγγραφών, ώστε η φαινομενική μέγιστη συσχετιστικότητα κάθε set να διπλασιαστεί, και τέλος αναζητούν την υποψήφια για αντικατάσταση γραμμή ανάμεσα σε όλες τις γραμμές της κρυφής μνήμης, ανεξαρτήτως του set με το οποίο είναι συνδεδεμένες. Αν η γραμμή άνηκε σε άλλο set από αυτό που προκάλεσε την αστοχία, τότε η γραμμή επανασυνδέεται στο set που προκάλεσε την αστοχία και ενεργοποιείται μία ακόμη tag εγγραφή σε αυτό το set. Με αυτόν τον τρόπο,

αφενός η πολιτική αντικατάστασης πετυχαίνει καλύτερα αποτελέσματα, γιατί συνήθως διαλέγει την ίδια γραμμή που θα διάλεγε σε μία πλήρως συσχετιστική κρυφή μνήμη, και αφετέρου η φαινομενική συσχετιστικότητα κάθε set μεταβάλλεται δυναμικά κατά την διάρκεια της εκτέλεσης ώστε να προσαρμόζεται στις απαιτήσεις του προγράμματος.

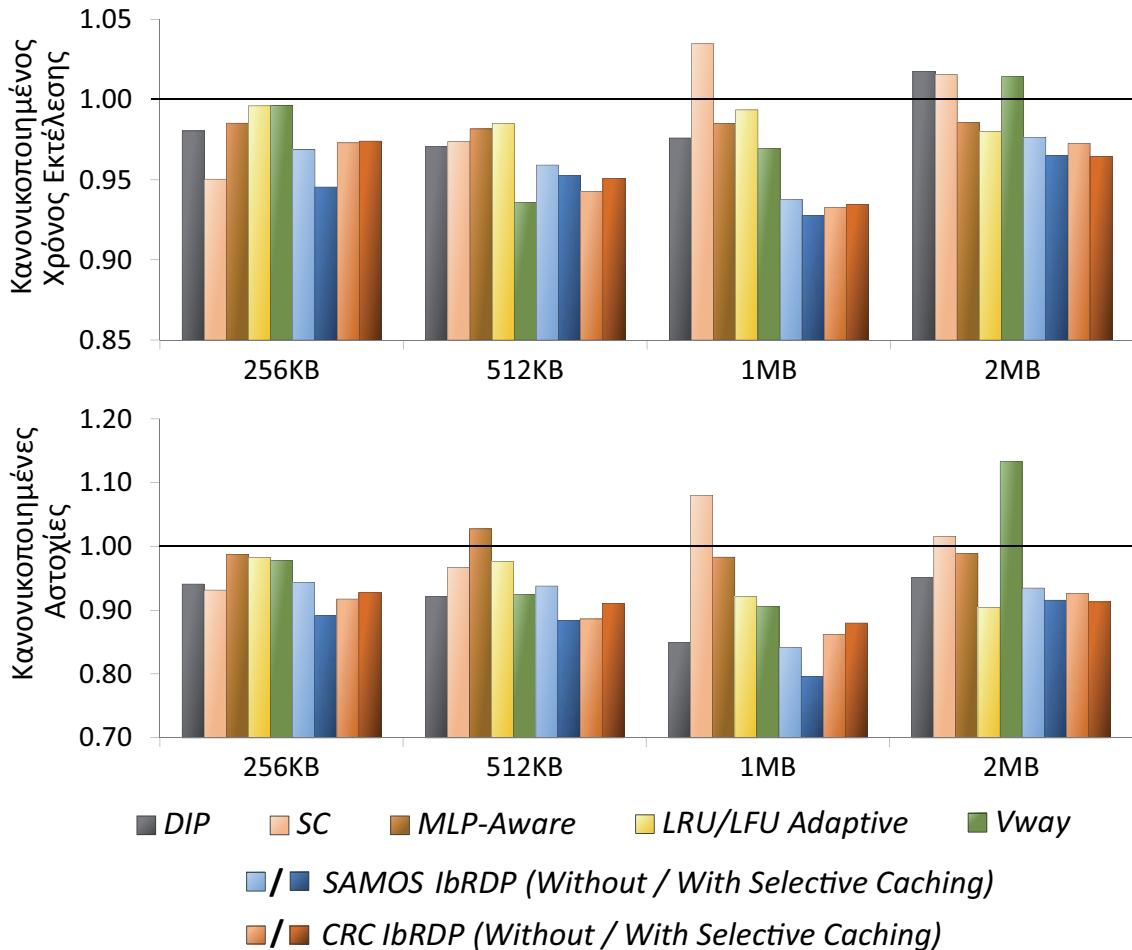
Η **Adaptive LRU/LFU Cache** τεχνική που προτείνεται από τους Subramanian et al [82], αποτελεί κομμάτι μίας ευρύτερης μεθοδολογίας που διαλέγει δυναμικά την καλύτερη από δύο οποιεσδήποτε πολιτικές διαχείρισης. Για κάθε μία από τις πολιτικές διατηρείται μία επιπλέον tag δομή, με την ίδια γεωμετρία αλλά κατά τα άλλα ανεξάρτητη της πραγματικής tag δομής. Η κάθε επιπλέον tag δομή προσπελάζεται παράλληλα με τα πραγματικά tags, αλλά όταν εμφανιστεί αστοχία σε μία tag δομή, η γραμμή που θα αντικατασταθεί επιλέγεται από την πολιτική αντικατάστασης που αντιστοιχεί στην δομή, ανεξάρτητα από τις επιλογές διαχείρισης των άλλων tags. Με αυτόν τον τρόπο, κάθε σύνολο από tags προσομοιώνει την πλήρη κατάσταση μίας κρυφής μνήμης που διαχειρίζεται από μία συγκεκριμένη πολιτική αντικατάστασης. Επιπλέον, για κάθε set, ένας μικρός buffer κρατάει τις τελευταίες ν προσπελάσεις του set, για τις οποίες η μία μόνο από τις δύο πολιτικές εμφάνισαν αστοχία. Με αυτόν τον τρόπο, αν εμφανιστεί αστοχία στην (πραγματική) κρυφή μνήμη, επιλέγεται να χρησιμοποιηθεί για την αντικατάσταση η πολιτική που εμφάνισε τις λιγότερες αστοχίες για αυτό το set κατά το πρόσφατο παρελθόν.

Πίνακας 1: Παράμετροι των υλοποιήσεων των 5 πολιτικών αντικατάστασης

Πολιτική	Παράμετροι
DIP	32+32 Leader Sets, $\epsilon=1/32$
Shepherd Cache	SC=4 ways, MC=12 ways
MLP-aware	SBAR Policy, 1:32 Leader Sets, $\lambda=4$
V-Way	TDR=2, Reuse Replacement
Adaptive LRU/LFU	Miss History Buffer=16 entries, Full Tags

Για όλες τις προαναφερθείσες τεχνικές, προσπαθήσαμε να τις υλοποιήσουμε όσο το δυνατόν πιο πιστά στην περιγραφή τους στις αντίστοιχες δημοσιεύσεις. Για όσες παραμέτρους των υλοποιήσεων υπήρχαν διαθέσιμες τιμές, τις χρησιμοποιήσαμε, ενώ για τις υπόλοιπες διαλέξαμε τις καλύτερες δυνατές τιμές μέσα σε κάποια λογικά όρια. Οι επιλογές για κάθε μηχανισμό φαίνονται στον Πίνακα 1.

Για την μεθοδολογία μας, τέλος, εξετάστηκαν και παρουσιάζονται 4 εναλλακτικές, 2 για κάθε υλοποίηση (με ή χωρίς Selective Caching). Τα αποτελέσματα για την υλοποίηση



Σχήμα 40: Κανονικοποιημένος, ως προς LRU, γεωμετρικός μέσος του χρόνου εκτέλεσης και των αστοχιών για όλους τους μηχανισμούς διαχείρισης

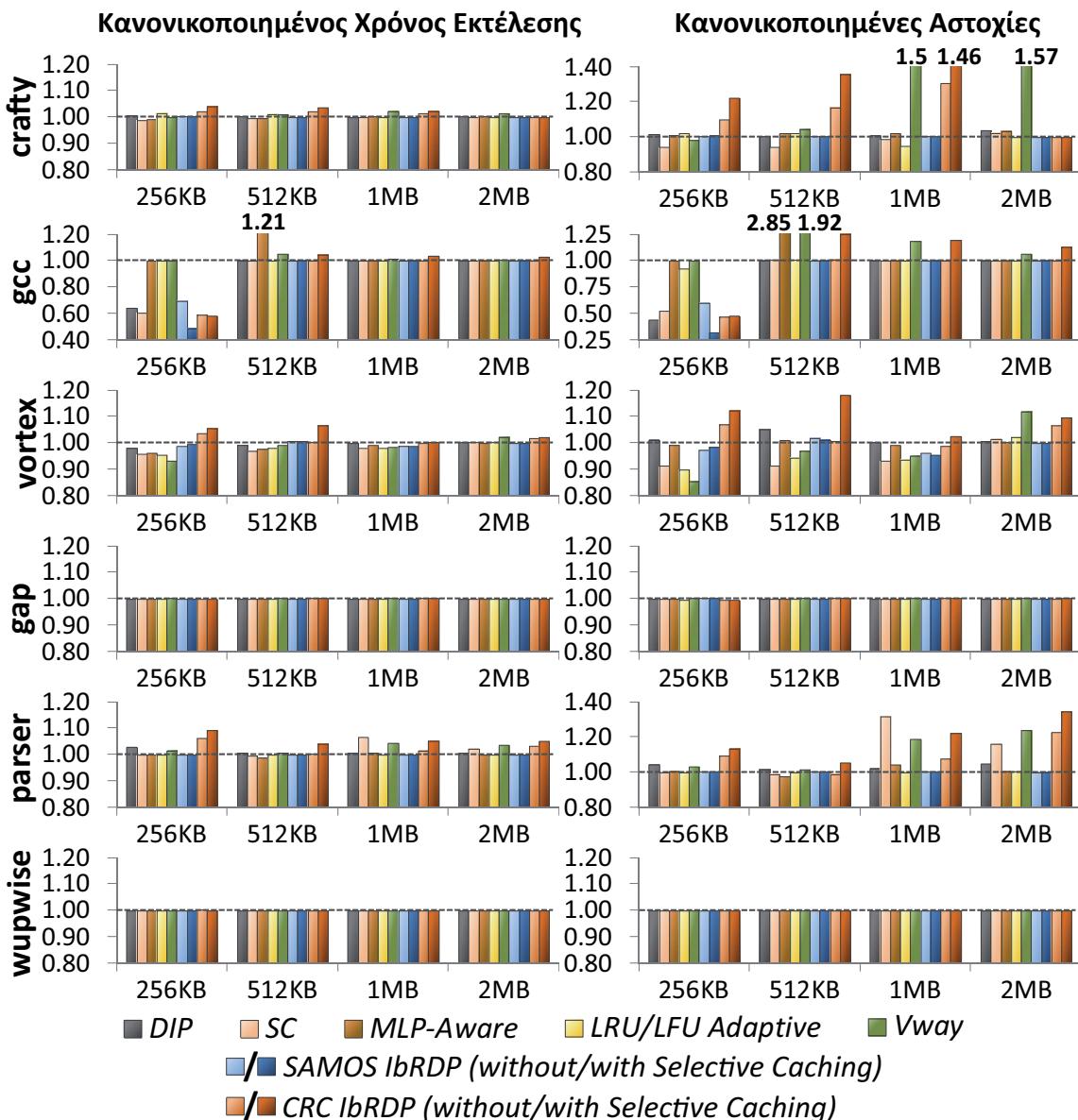
υψηλής απόδοσης σημειώνονται ως SAMOS IbRDP (Instruction-based Reuse Distance Prediction όπως παρουσιάστηκε στο συνέδριο SAMOS-2009), ενώ αυτά της υλοποίησης χαμηλού κόστους ως CRC IbRDP (Instruction-based Reuse Distance Prediction όπως παρουσιάστηκε στο Cache Replacement Championship).

4.6.2 Μετρήσεις

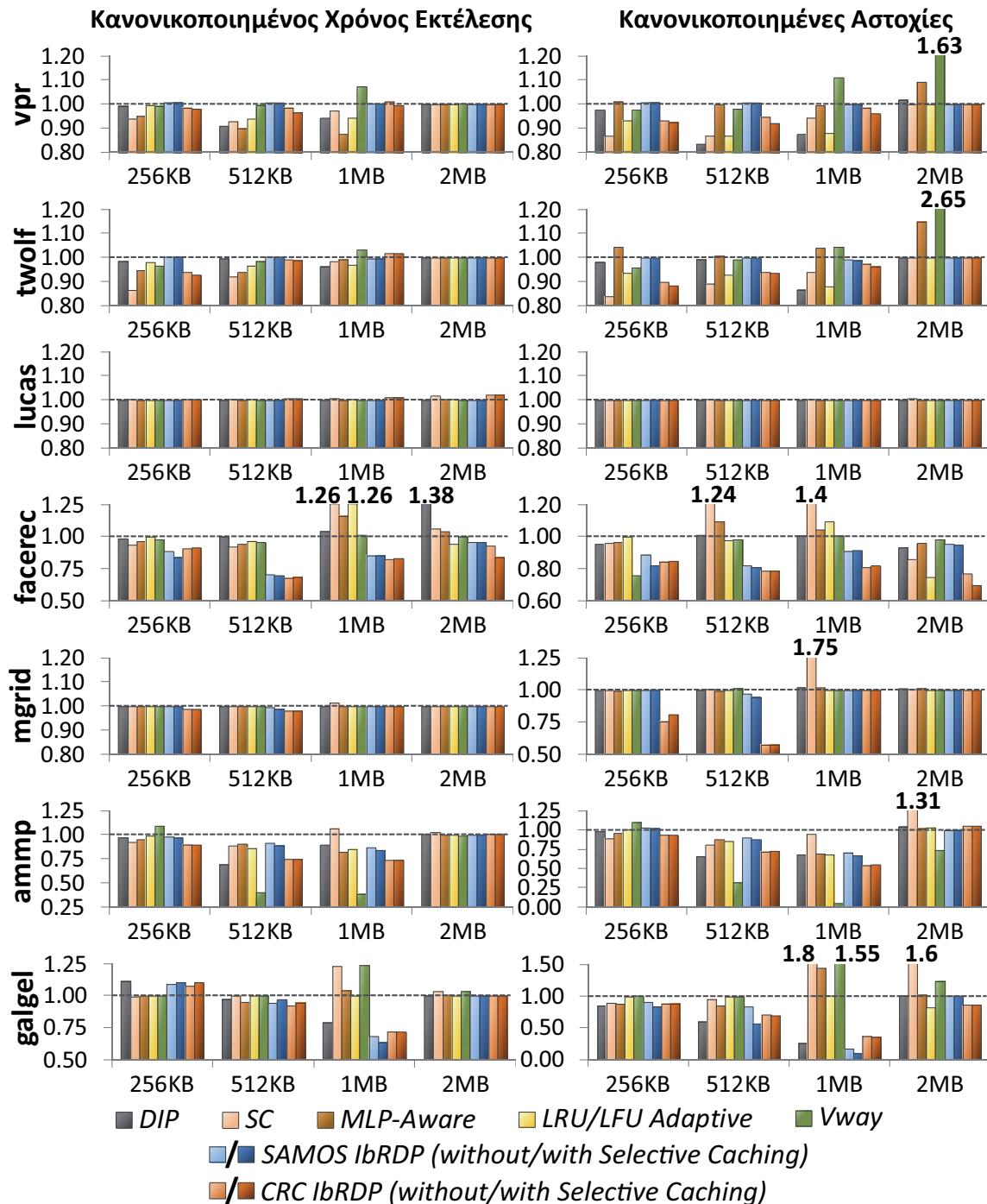
Στο Σχήμα 40 βλέπουμε τα συγκεντρωτικά αποτελέσματα των εξομοιώσεών μας, ενώ στα Σχήματα 41, 42 και 43 βλέπουμε τα αναλυτικά αποτελέσματα. Τα συγκεντρωτικά δίνουν τον γεωμετρικό μέσο του χρόνου εκτέλεσης και των αστοχιών ανά κρυφή μνήμη και ανά μηχανισμό διαχείρισης, κανονικοποιημένα ως προς LRU. Τα αναλυτικά αποτελέσματα είναι χωρισμένα ανάλογα με την κατηγοριοποίηση των προγραμμάτων σε χαμηλών, μεσαίων και υψηλών αστοχιών ανά 1K εντολές (Πίνακας 2) και δίνουν την ίδια πληροφορία με το συγκεντρωτικό σχήμα ξεχωριστά για κάθε πρόγραμμα.

Πίνακας 2: Αστοχίες ανά 1K εντολές (MPKI) στην 512KB κρυφή μνήμη για τα προγράμματα της σουίτας SPEC2000

Χαμηλό MPKI		Μεσαίο MPKI		Υψηλό MPKI	
Πρόγραμμα	MPKI	Πρόγραμμα	MPKI	Πρόγραμμα	MPKI
crafty	0.20	vpr	6.57	applu	21.27
gcc	0.25	twolf	8.86	apsi	22.67
vortex	0.47	lucas	9.26	swim	23.84
gap	0.94	facerec	9.82	mcf	98.62
parser	1.08	mgrid	10.88	art	127.07
wupwise	2.62	ammp	11.60		
		galgel	14.94		

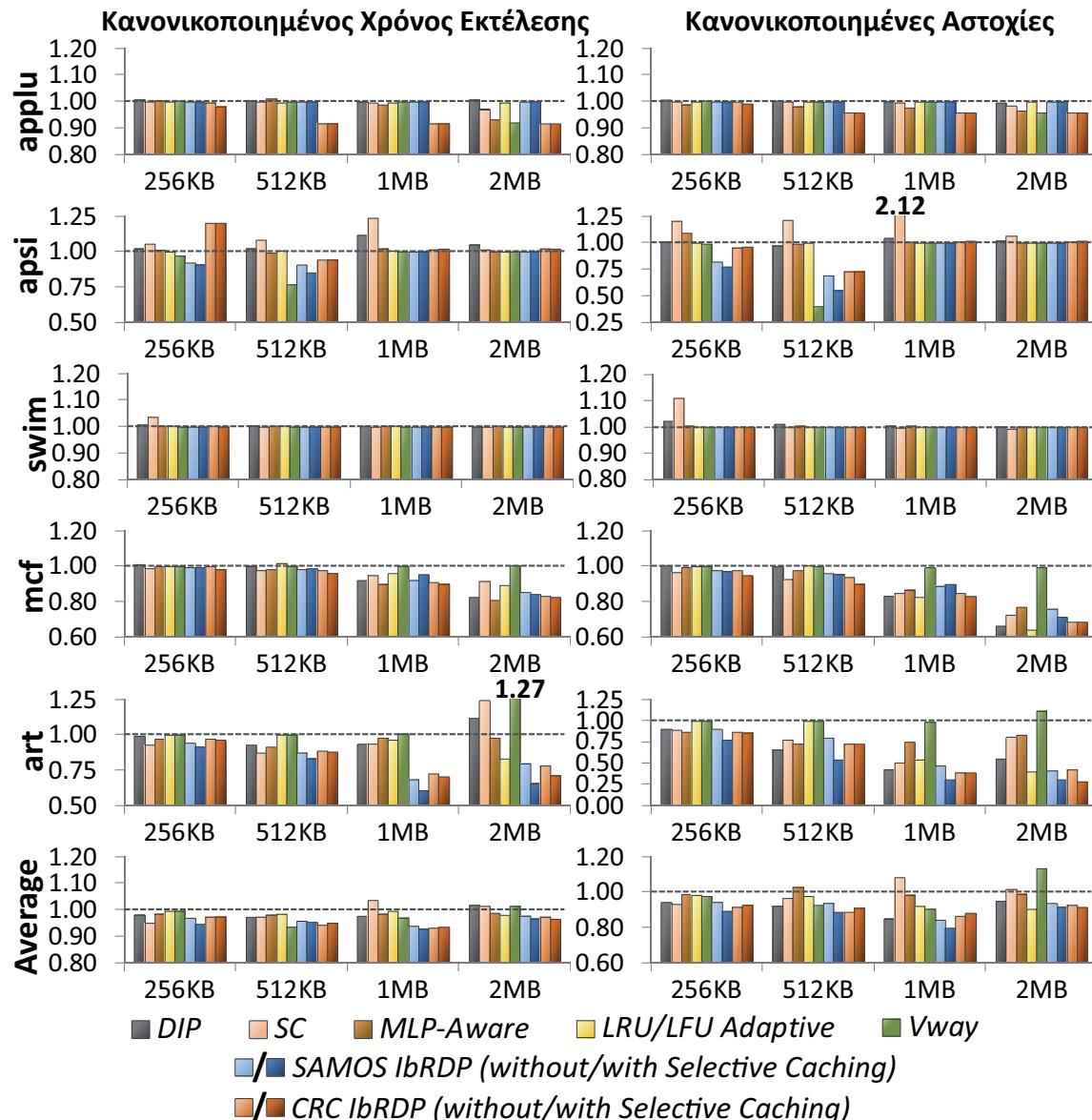


Σχήμα 41: Πειραματικά αποτελέσματα για τα προγράμματα χαμηλού MPKI



Σχήμα 42: Πειραματικά αποτελέσματα για τα προγράμματα χαμηλού MPKI

Όπως βλέπουμε, όλες οι υλοποιήσεις που πηγάζουν από την μεθοδολογία μας επιτυγχάνουν να μειώσουν σημαντικά τις αστοχίες και τον χρόνο εκτέλεσης. Σε ότι αφορά τους γεωμετρικούς μέσους των χρόνων εκτέλεσης, ο SAMOS IbRDP πετυχαίνει μείωση από 2.3% (στην 2MB κρυφή μνήμη) έως 6.2% (στην 1MB κρυφή μνήμη), ενώ με Selective Caching ο χρόνος εκτέλεσης μειώνεται ακόμη περισσότερο: από 3.5% (2MB) έως 7.2% (1MB). Για τον χρόνο εκτέλεσης με τον CRC IbRDP, οι μέσοι όροι είναι εξίσου καλοί: χωρίς Selective



Σχήμα 43: Πειραματικά αποτελέσματα για τα προγράμματα υψηλού MPKI και γεωμετρικοί μέσοι των αποτελεσμάτων ανά κρυφή μνήμη

Caching μείωση από 2.7%(256KB) έως 6.7%(1MB), ενώ με Selective Caching η μέση μείωση κυμαίνεται από 2.7%(256KB) έως 6.5%(1MB). Αξίζει να σημειωθεί, ότι οι γεωμετρικοί μέσοι συμπεριλαμβάνουν όλα τα προγράμματα, ακόμη και αυτά που χωράνε στην κρυφή μνήμη ή που η διαχείριση με LRU είναι ήδη κοντά στην βέλτιστη.

Για όλα τα μεγέθη κρυφών μνημών, οι μηχανισμοί μας πετυχαίνουν τις μεγαλύτερες, κατά μέσο όρο, μειώσεις του χρόνου εκτέλεσης, με εξαίρεση την 256KB κρυφή μνήμη όπου ο μηχανισμός του Shepherd Cache Replacement είναι ο δεύτερος καλύτερος, πίσω από τον SAMOS IbRDP+SC, και την 512KB κρυφή μνήμη όπου ο μηχανισμός της vway cache είναι ελαφρώς καλύτερος από τους μηχανισμούς που προέκυψαν από την μεθοδολογία μας. Με

άλλα λόγια, στις περισσότερες περιπτώσεις ακόμη και η πιο απλή υλοποίηση της μεθοδολογίας μας, παράγει καλύτερα αποτελέσματα από την αμέσως καλύτερη μεθοδολογία.

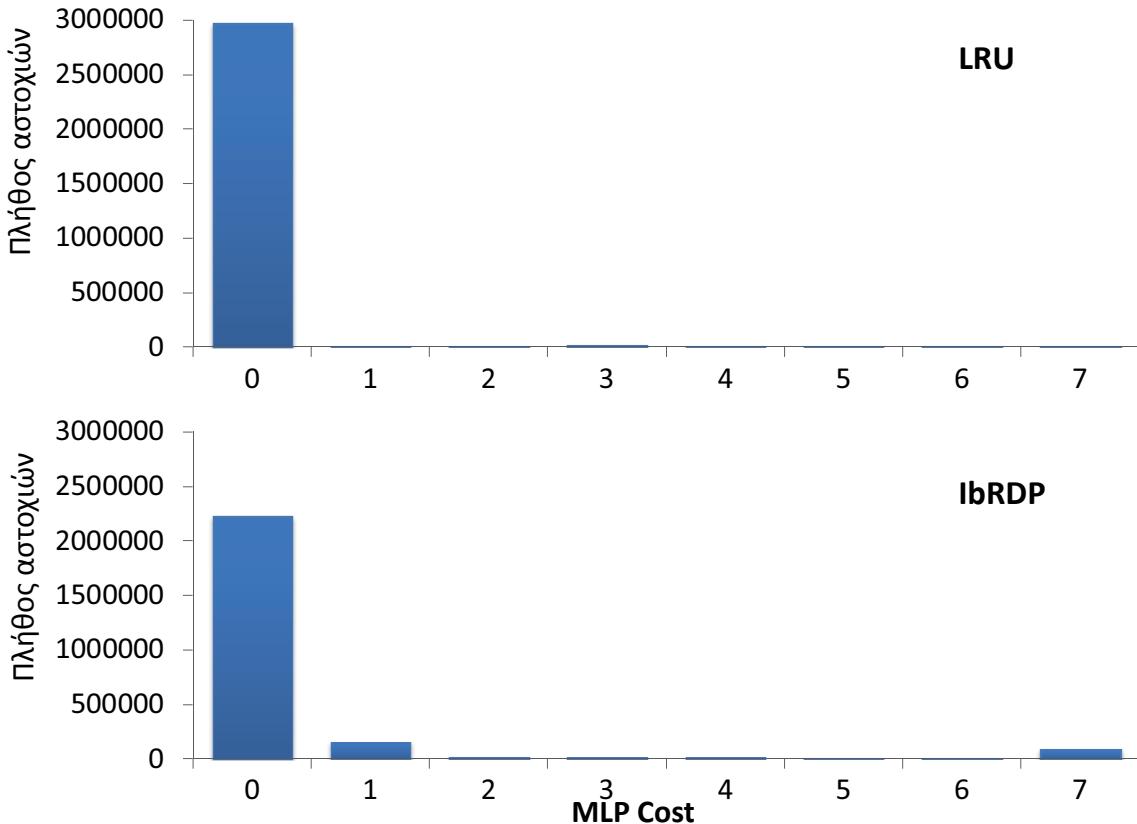
Σε ότι αφορά τους γεωμετρικούς μέσους των αστοχιών κρυφής μνήμης, τα αποτελέσματα είναι παρόμοια: για τον SAMOS IbRDP μειώσεις 5.6%(256KB) έως 15.9%(1MB), για τον SAMOS IbRDP με Selective Caching μειώσεις 8.5%(2MB) έως 20.4%(1MB), για τον CRC IbRDP μειώσεις από 7.3%(2MB) έως 13.8%(1MB), και για τον CRC IbRDP με Selective Caching μειώσεις από 7.3%(256KB) έως 12%(1MB).

Μεταξύ των υλοποιήσεων της μεθοδολογίας μας βλέπουμε ότι ο SAMOS IbRDP πετυχαίνει ελαφρώς καλύτερα αποτελέσματα σε ότι αφορά τον χρόνο εκτέλεσης και αρκετά καλύτερα σε ότι αφορά τις αστοχίες, σε σχέση με τον CRC IbRDP. Ο λόγος για αυτή την διαφορά δεν είναι τόσο η καλύτερη συμπεριφορά του SAMOS IbRDP στις “καλές” περιπτώσεις, όσο η σταθερότητα του στις “κακές” περιπτώσεις. Στα αναλυτικά αποτελέσματα βλέπουμε ότι όταν ο SAMOS IbRDP πετυχαίνει σημαντικές μειώσεις του χρόνου εκτέλεσης, παρόμοια καλά αποτελέσματα παράγει και ο CRC IbRDP, ενώ υπάρχουν και περιπτώσεις όπου ο CRC IbRDP πετυχαίνει σημαντικές μειώσεις εκεί όπου ο SAMOS IbRDP δεν μπορεί (twolf-256KB, ammp-256/512/1024KB, applu-512/1024/2048KB). Όμως ενώ ο SAMOS IbRDP μόνο σε μία περίπτωση αυξάνει τον χρόνο εκτέλεσης κατά περισσότερο από 1% (galgel-256KB), ο CRC μηχανισμός τον αυξάνει για 11 συνδυασμούς προγράμματος-κρυφής μνήμης, με χειρότερο αποτέλεσμα το αριστερό στην 256KB κρυφή μνήμη όπου η αύξηση φτάνει το 20%. Ο λόγος για αυτήν την συμπεριφορά είναι η μειωμένη ακρίβεια του CRC IbRDP: με 3 bits για την αποθήκευση του χρόνου τελευταίας πρόσβασης, και με ανάλυση ίση με L (4K στα 256KB, 32K στα 2MB), η “LRU” γραμμή που διαλέγει ο μηχανισμός αντικατάστασης είναι αρκετά πιθανό να μην είναι στον πάτο ή κοντά στον πάτο της LRU στοίβας. Έτσι, στα προγράμματα και στις κρυφές μνήμης όπου η LRU πολιτική αντικατάστασης λειτουργεί πολύ κοντά στην βέλτιστη πολιτική, ο CRC IbRDP τείνει να χειροτερεύει την απόδοση του συστήματος. Για τους ίδιους λόγους, ενώ η χρήση Selective Caching βελτιώνει σημαντικά τον SAMOS IbRDP (έως 2.3% στους γεωμετρικούς μέσους και έως 22% στα μεμονωμένα προγράμματα), δεν ισχύει το ίδιο και για τον CRC IbRDP: μόνο στα 2MB βελτιώνεται ο γεωμετρικός μέσος και μόνο για 0.6%, ενώ στα μεμονωμένα προγράμματα η βελτίωση φτάνει έως το 8.6%.

Επικεντρώνοντας την υπόλοιπη ανάλυση στον καλύτερο μηχανισμό μας, δηλαδή τον SAMOS IbRDP με Selective Caching, εξετάζουμε τα αποτελέσματα για κάθε πρόγραμμα και

κρυφή μνήμη. Από τις 25 περιπτώσεις, όπου έστω και ένας μηχανισμός κατάφερε να μειώσει σημαντικά (πάνω από 5%) τον χρόνο εκτέλεσης, ο μηχανισμός μας πετυχαίνει σημαντική μείωση στις 16 και στις 10 από αυτές πετυχαίνει την μεγαλύτερη αύξηση της απόδοσης του επεξεργαστή. Το καλύτερο αποτέλεσμα του μηχανισμού παράγεται για την εξομοίωση του gcc σε κρυφή μνήμη των 256KB, όπου ο χρόνος εκτέλεσης μειώνεται κατά 51.6%, το οποίο ισοδυναμεί διπλασιασμό του μεγέθους της κρυφής μνήμης ή με υπερδιπλασιασμό της ταχύτητας επεξεργασίας. Σταθερά θετικά αποτελέσματα παράγονται ακόμη για το art, το mcf και το facerec. Κοινό χαρακτηριστικό αυτών των προγραμμάτων είναι ότι έχουν μεγάλες απαιτήσεις σε μνήμη (όπως δείχνουν και οι καμπύλες της StatCache μεθοδολογίας στο Σχήμα 7) και αρκετά προβλέψιμη συσχέτιση μεταξύ των εντολών και των αποστάσεων επαναχρησιμοποίησης που παράγουν οι προσπελάσεις τους (όπως επιβεβαιώνει και το Σχήμα 35). Σε προγράμματα με χαμηλότερη ακρίβεια πρόβλεψης, τα αποτελέσματα δεν είναι τόσο θετικά. Για παράδειγμα, στο twolf, το vpr και το vortex, ενώ οι υπόλοιποι μηχανισμοί που εκμεταλλεύονται την τοπικότητα (DIP, Shepherd Cache, Adaptive LRU/LRU) πετυχαίνουν σε αρκετές περιπτώσεις σημαντικές μειώσεις του χρόνου εκτέλεσης, ο IbRDP μηχανισμός δεν προκαλεί καμία βελτίωση σε σχέση με το LRU, κάτι που οφείλεται στην πολύ χαμηλή ακρίβεια και κάλυψη του Predictor (20% για vpr και twolf, 50% για το vortex).

Το χειρότερο (και μοναδικό αρνητικό) αποτέλεσμα του μηχανισμού είναι για κρυφή μνήμη των 256KB όταν εκτελείται το galgel, οπότε ο χρόνος εκτέλεσης αυξάνεται κατά 10%. Όμως ακόμη και σε αυτή την περίπτωση, ο IbRDP λειτουργεί αποδοτικά σε ότι αφορά τον χειρισμό της τοπικότητας, αφού επιτυγχάνει να μειώσει τις αστοχίες κατά 16%. Η ασυνέπεια μεταξύ της μείωσης των αστοχιών και της αύξησης του χρόνου εκτέλεσης οφείλεται στο MLP. Στο Σχήμα 44 βλέπουμε την κατανομή του MLP-cost των αντικαταστάσεων του galgel στα 256KB με LRU και IbRDP πολιτική αντικατάστασης. Παρότι, το galgel έχει υψηλό αριθμό αστοχιών υπό LRU διαχείριση, πρακτικά όλες τους έχουν το χαμηλότερο δυνατό κόστος, δηλαδή είναι πλήρως παράλληλες με πολλαπλές άλλες αντικαταστάσεις. Έτσι, όταν ο IbRDP μηχανισμός μετατρέπει το 16% αυτών των αστοχιών σε ευστοχίες, η επίδραση στην απόδοση του επεξεργαστή είναι ασήμαντη. Ταυτόχρονα, προστίθενται κάποιες λίγες επιπλέον αστοχίες, οι οποίες όμως είναι απομονωμένες και παράγουν το μέγιστο δυνατό MLP-cost, με συνέπεια την αύξηση του χρόνου εκτέλεσης.



Σχήμα 44: Κατανομή του MLP-cost του galgel όταν χρησιμοποιεί μία κρυφή μνήμη των 256KB, την οποία διαχειρίζομαστε είτε με LRU είτε με IbRDP πολιτικές αντικατάστασης

Όμως, με εξαίρεση την περίπτωση που εξετάστηκε στην προηγούμενη παράγραφο, οι IbRDP μηχανισμοί χαρακτηρίζονται από αρκετά καλό χειρισμό του MLP. Στα περισσότερα προγράμματα υπάρχει ικανοποιητική αντιστοιχία μεταξύ μείωσης των αστοχιών και μείωσης του χρόνου εκτέλεσης. Για παράδειγμα, για πολλαπλές κρυφές μνήμες του art και του facerec βλέπουμε ότι ενώ οι IbRDP μηχανισμοί πετυχαίνουν μείωση των αστοχιών αντίστοιχη με των υπόλοιπων μηχανισμών (πχ art σε 1MB), μόνο οι μηχανισμοί της μεθοδολογίας μας καταφέρνουν να την μεταφράσουν σε αύξηση της απόδοσης του επεξεργαστή.

4.7 Πειραματικά Αποτελέσματα Πρωταθλήματος

4.7.1 Πειραματική μεθοδολογία

Για τις εξομοιώσεις μας, βασιστήκαμε στην πλατφόρμα εξομοίωσης του πρωταθλήματος [41]. Ο επεξεργαστή που εξομοιώνεται είναι ένα απλός 4-way superscalar επεξεργαστής με εκτέλεση εκτός σειράς, παράθυρο 128 εντολών, pipeline 8 σταδίων και τέλεια πρόβλεψη διακλαδώσεων. Όλες οι εντολές εκτελούνται σε έναν κύκλο ρολογιού, εκτός από τις

αστοχίες κρυφής μνήμης. Οι αστοχίες στο πρώτο επίπεδο στοιχίζουν 10 κύκλους, οι αστοχίες στο δεύτερο επίπεδο στοιχίζουν 40 κύκλους, ενώ η προσπέλαση της κύριας μνήμης στοιχίζει επιπλέον 200 κύκλους, για συνολικό χρόνο εξυπηρέτησης μίας εντολής που φέρνει δεδομένα από την κύρια μνήμη 240 κύκλους.

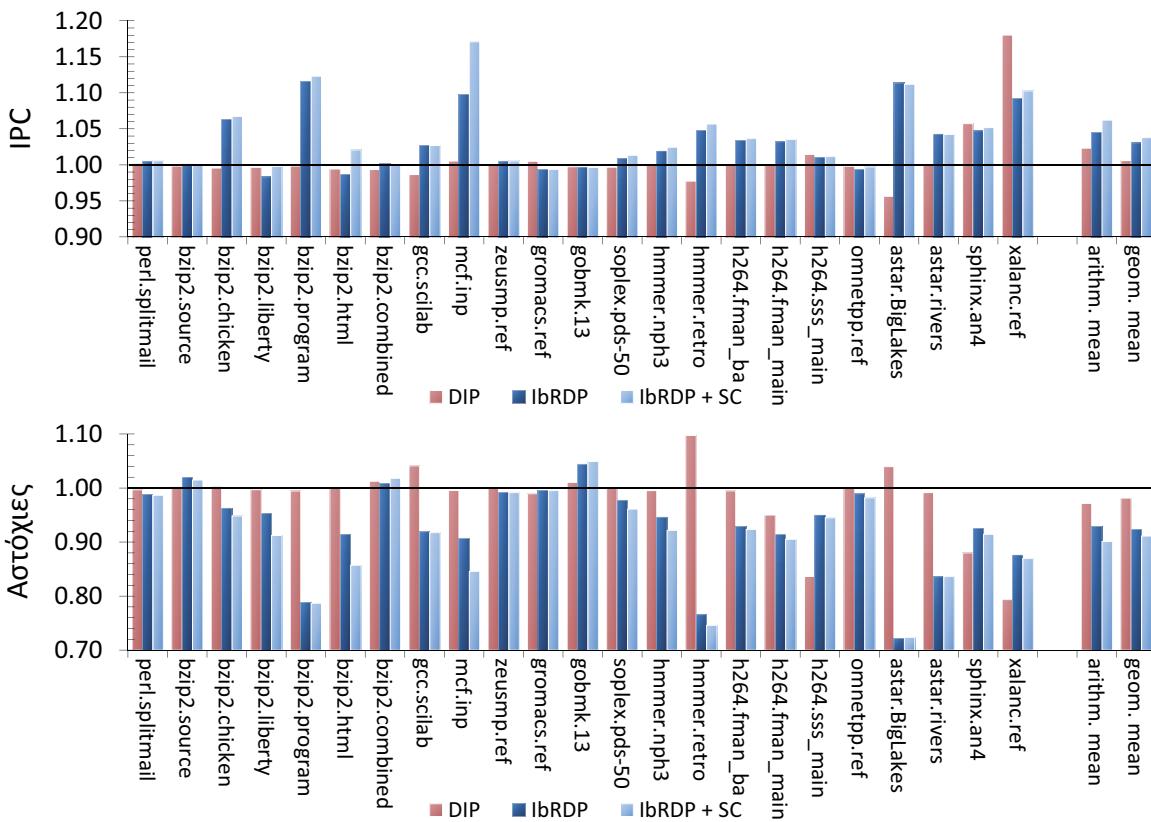
Η ιεραρχία κρυφών μνημάτων αποτελείται από τρία επίπεδα. Όλες οι κρυφές μνήμες αποθηκεύονται γραμμές των 64 bytes και υλοποιούν LRU πολιτική αντικατάστασης. Η L1 εντολών έχει μέγεθος 32KB και είναι οργανωμένη σε 4 ways, η L1 δεδομένων είναι 32KB με 8 ways, η L2 αποθηκεύει 256KB και έχει 8 ways, ενώ η L3, που είναι και η κρυφή μνήμη που διαχειρίζομαστε, είναι οργανωμένη σε 16 ways και έχει μέγεθος 1MB (για το μονονηματικό σκέλος του διαγωνισμού) ή 4MB (για το πολυνηματικό σκέλος).

Τα προγράμματα που θα εξομοιώνονταν στα πλαίσια του διαγωνισμού ήταν άγνωστα κατά την φάση της ανάπτυξης του μηχανισμού, και η μόνη πληροφορία από την επιτροπή του πρωταθλήματος ήταν ότι κάποια από αυτά θα προέρχονταν από την σούντα SPEC CPU2006 [80]. Επομένως, οι εξομοιώσεις που εκτελέσαμε χρησιμοποίησαν το υποσύνολο των προγραμμάτων από τα SPEC CPU2006, για τα οποία ο διπλασιασμός της κρυφής μνήμης από 1MB σε 2MB έχει επίπτωση μεγαλύτερη του 1% στον χρόνο εκτέλεσης. Τα υπόλοιπα προγράμματα έχουν ή μικρότερες ή πολύ μεγαλύτερες απαιτήσεις σε μνήμη, και επομένως δεν αναμένεται να βελτιωθεί η συμπεριφορά τους αν χρησιμοποιηθεί κάποια πιο αποδοτική πολιτική αντικατάστασης. Από κάθε πρόγραμμα εξομοιώθηκαν 100M εντολές, αφού αγνοήθηκαν 40B εντολές που καλύπτουν την αρχικοποίηση των προγραμμάτων. Σαν μέτρο σύγκρισης υλοποιήσαμε στην πλατφόρμα εξομοιώσης και την DIP πολιτική αντικατάστασης.

Η δεύτερη ομάδα αποτελεσμάτων που θα παρουσιάσουμε προέρχεται από τις εξομοιώσεις που εκτέλεσε η οργανωτική επιτροπή του πρωταθλήματος, βάσει των οποίων αξιολογήθηκαν στο τέλος οι συμμετοχές. Τα προγράμματα που εξομοιώθηκαν τελικά καλύπτουν τρεις κατηγορίες (SPEC CPU2006, Enterprise Server, Multimedia), συν άλλη μία για το πολυνηματικό σκέλος του πρωταθλήματος (Mixed). Σύμφωνα με την οργανωτική επιτροπή, για τα μεν SPEC παρήχθησαν traces μέσω του εργαλείου δυναμικής ενορχήστρωσης κώδικα Pin [59] από περιοχές του προγράμματος που επιλέχθησαν βάσει της μεθοδολογίας SimPoints [76], ενώ για τα υπόλοιπα προγράμματα τα traces παρήχθησαν μέσω υλικού από πραγματικά συστήματα. Για όλες τις εφαρμογές εκτός από τα SPEC, ο πηγαίος κώδικας και τα δεδομένα εισόδου είναι άγνωστα.

4.7.2 Ανεπίσημα Αποτελέσματα

Στο Σχήμα 45 παρουσιάζονται τα αποτελέσματα των εξομοιώσεών μας. Το πάνω γράφημα δείχνει το IPC, ενώ το κάτω γράφημα δείχνει τις αστοχίες στο τελευταίο επίπεδο κρυφής μνήμης για τα επιλεγμένα προγράμματα της σουίτας SPEC CPU2006. Και τα δύο γραφήματα είναι κανονικοποιημένα ως προς το LRU.

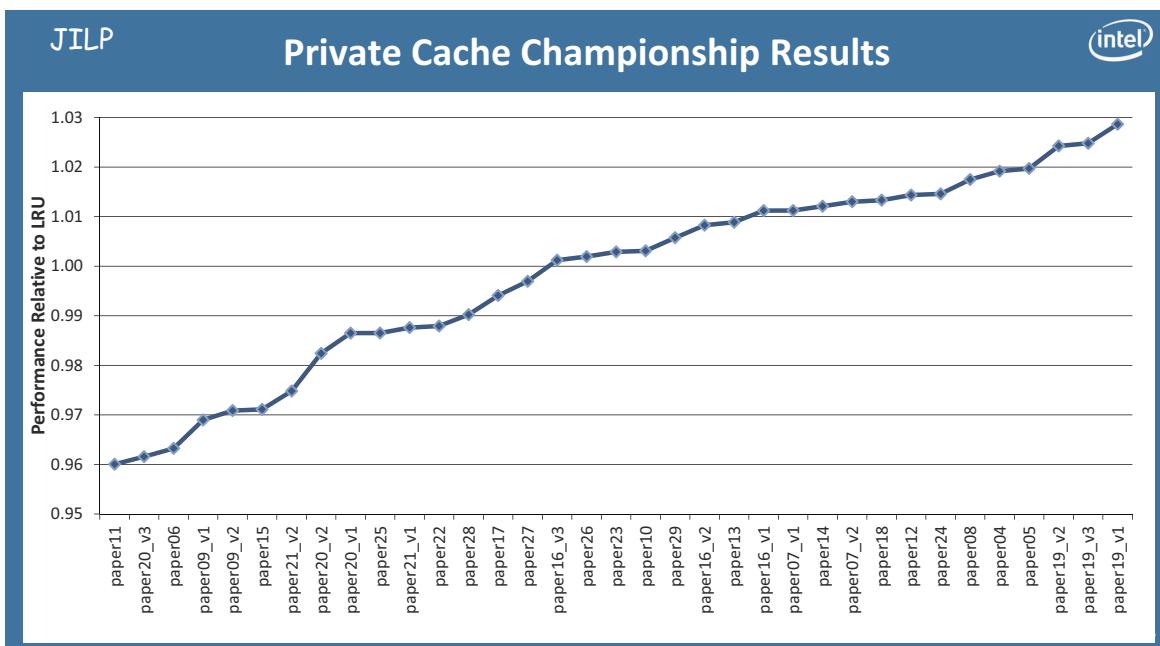


Σχήμα 45: IPC και αστοχίες, κανονικοποιημένα ως προς LRU, για διαχείριση με DIP, IbRDP και IbRDP με Selective Caching

Τα αποτελέσματα είναι συνεπή με αυτά που παρουσιάστηκαν στην προηγούμενη ενότητα: ο IbRDP μηχανισμός επιτυγχάνει να αυξήσει σημαντικά το IPC αρκετών από τα προγράμματα που μελετήθηκαν, ενώ οι περιπτώσεις όπου υποβαθμίζει την απόδοση του επεξεργαστή είναι λίγες και όχι ιδιαίτερα επιβλαβείς. Ο βασικός μηχανισμός πετυχαίνει αύξηση του IPC μέχρι 11.52% (για το bzip2.program), παράγει το χειρότερο αποτέλεσμά του για το bzip2.liberty όπου μειώνει το IPC κατά 1.7% σε σχέση με το LRU, και κατά γεωμετρικό μέσο αυξάνει το IPC 3.05%. Η επέκταση με Selective Caching σε αυτή την ομάδα εξομοιώσεων παράγει καλύτερα αποτελέσματα. Σε καμία περίπτωση δεν μειώνει το IPC σημαντικά σε σχέση με τον βασικό IbRDP μηχανισμό και σε 5 περιπτώσεις το αυξάνει, με καλύτερο αποτέλεσμα την αύξηση του IPC του mcf.inp κατά 7.3% σε σχέση με τον

βασικό μηχανισμό και κατά 17.1% σε σχέση με το LRU. Κατά γεωμετρικό μέσο, το Selective Caching προσφέρει βελτίωση 0.73% σε σχέση με τον βασικό μηχανισμό, οδηγώντας έτσι σε συνολική αύξηση του IPC κατά 3.78%.

Σε ότι αφορά το DIP, και αυτή η σειρά πειραμάτων επιβεβαιώνει την ανωτερότητα της προσέγγισής μας. Για τα περισσότερα προγράμματα, ο IbRDP αυξάνει την απόδοση του επεξεργαστή σχετικά ώς προς το DIP, και μόνο για το xalanc.ref επιτυγχάνει το DIP σημαντικά καλύτερο αποτέλεσμα. Κατά γεωμετρικό μέσο, τέλος, ο IbRDP μηχανισμός είναι 2.9% καλύτερος από το DIP.



Σχήμα 46: Κατάταξη όλων των συμμετοχών στο μονονηματικό σκέλος του πρωταθλήματος [42]

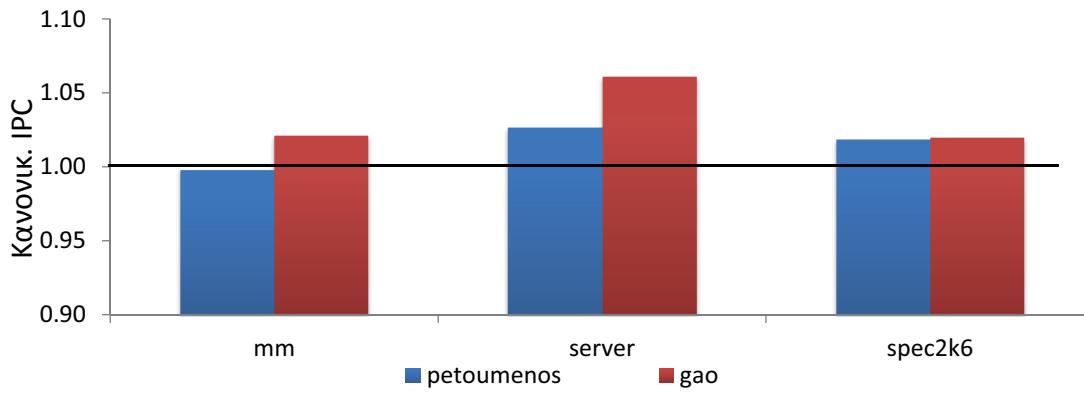
4.7.3 Επίσημα Αποτελέσματα

Το Σχήμα 46 παρουσιάζει την τελική κατάταξη των συμμετοχών στο μονονηματικό σκέλος του πρωταθλήματος. 26 διαφορετικές μεθοδολογίες, με 35 συνολικά διαφορετικές υλοποιήσεις, συμμετείχαν στον διαγωνισμό. Όπως φαίνεται και από το σχήμα, οι μισές περίπου συμμετοχές δεν κατόρθωσαν να αυξήσουν την απόδοση του επεξεργαστή σε σχέση με την LRU πολιτική αντικατάστασης, ενώ ακόμη και η καλύτερη υλοποίηση δεν αύξησε την μέση απόδοση πάνω από 2.8%. Σημαντικό ρόλο σε αυτό το αποτέλεσμα έπαιξε το γεγονός, ότι τα περισσότερα προγράμματα που χρησιμοποιήθηκαν για την αξιολόγηση των μηχανισμών ήταν άγνωστα κατά την φάση της ανάπτυξης των υλοποιήσεων και κάλυπταν κατηγορίες εφαρμογών που συνήθως δεν μελετώνται στον χώρο της αρχιτεκτονικής

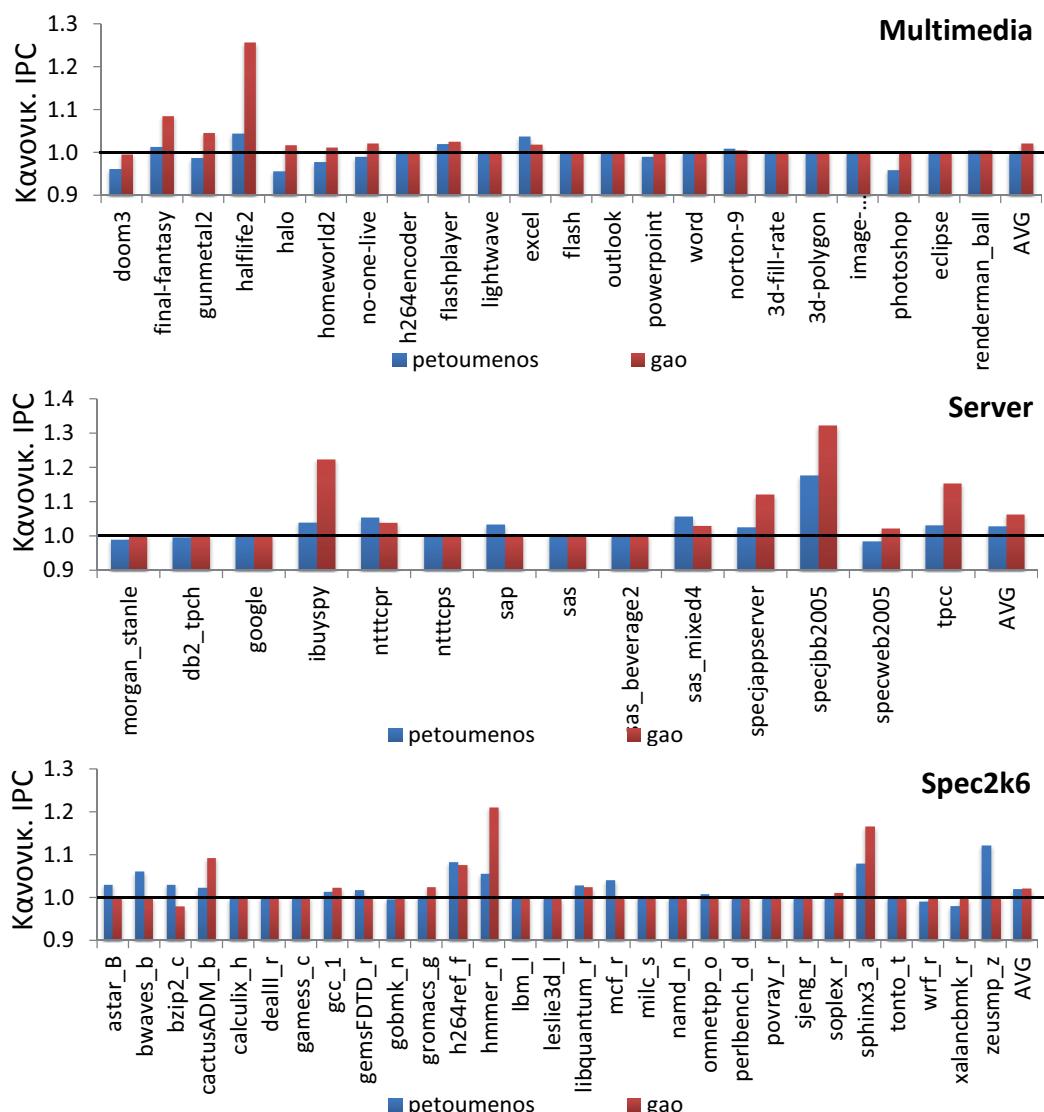
επεξεργαστών. Η μεθοδολογία IbRDP ήταν η 7η συμμετοχή που κατατέθηκε και επομένως αντιπροσωπεύεται στο σχήμα από τα σημεία του x-άξονα paper07_v1 και paper07_v2, για τον βασικό μηχανισμό και τον μηχανισμό με Selective Caching αντίστοιχα. Σύμφωνα με το σχήμα, η μεθοδολογία IbRDP ήταν η 8η καλύτερη που συμμετείχε στο μονονηματικό σκέλος του πρωταθλήματος.

Η νικήτρια μεθοδολογία ήταν η *Dueling Segmented LRU Replacement Algorithm with Adaptive Bypassing (DSB)* των Gao και Wilkerson [28]. Η βάση της μεθοδολογίας είναι η Segmented LRU λογική: πέρα από την πλήρη LRU σειρά των γραμμών του set, κάθε γραμμή περιέχει ένα επιπλέον bit (reference bit), το οποίο δείχνει αν η γραμμή έχει επαναχρησιμοποιηθεί από την στιγμή που τοποθετήθηκε στην κρυφή μνήμη. Με βάση αυτό το bit, ο αλγόριθμος αντικατάστασης επιλέγει για αντικατάσταση την LRU γραμμή ανάμεσα σε αυτές που δεν έχουν επαναχρησιμοποιηθεί (non-referenced list), και μόνο αν όλες οι γραμμές έχουν σεταρισμένο το reference bit επιλέγεται η καθολικά LRU γραμμή. Οι Gao και Wilkerson επεκτείνουν αυτήν την λογική με τρεις τροποποιήσεις. Η πρώτη αναβαθμίζει κάποιες γραμμές με τυχαίο τρόπο από non-referenced σε referenced. Η δεύτερη είναι ένας μηχανισμός γήρανσης, που μηδενίζει το reference bit της LRU γραμμής κάθε φορά που τοποθετείται στο set μία καινούργια γραμμή, ώστε να εξασφαλιστεί ότι θα αντικατασταθούν δεδομένα που δεν χρησιμοποιούνται πια. Η τρίτη είναι ένας μηχανισμός παράκαμψης της κρυφής μνήμης, που για κάθε παράκαμψη παρακολουθεί ποια γραμμή θα επαναχρησιμοποιηθεί πιο σύντομα, η γραμμή που παρακάμφθηκε ή η γραμμή που θα είχε αντικατασταθεί αν δεν παρακάμπταμε την κρυφή μνήμη. Με βάση την ορθότητα των αποφάσεων παράκαμψης, η πιθανότητα παράκαμψης μεταβάλλεται δυναμικά. Ο καλύτερος μηχανισμός των Gao και Wilkerson χρησιμοποιεί δύο πολιτικές αντικατάστασης: η πρώτη υλοποιεί την Segmented LRU λογική με τον μηχανισμό γήρανσης και η δεύτερη υλοποιεί την Segmented LRU λογική μαζί με τον μηχανισμό παράκαμψης. Το ποια από τις δύο πολιτικές θα εφαρμοστεί τελικά επιτυγχάνεται με έναν dueling μηχανισμό παρόμοιο με αυτούς των DIP [73] και MLP-aware Replacement Policy [72].

Στο Σχήμα 47 βλέπουμε πως συγκρίνεται ο IbRDP μηχανισμός με τον καλύτερο μηχανισμό του πρωταθλήματος. Ο IbRDP αύξησε το μέσο IPC του επεξεργαστή για τις server εφαρμογές και τα προγράμματα της σουίτας SPEC CPU2006, αλλά προκάλεσε μία μέση μείωση 0.25% για τις εφαρμογές πολυμέσων. Η διαφορά ανάμεσα στους δύο μηχανισμούς μεγιστοποιείται για την κατηγορία των server εφαρμογών, όπου φτάνει το 3.4%, αλλά για την κατηγορία των SPEC προγραμμάτων, που ήταν και η μόνη κατηγορία

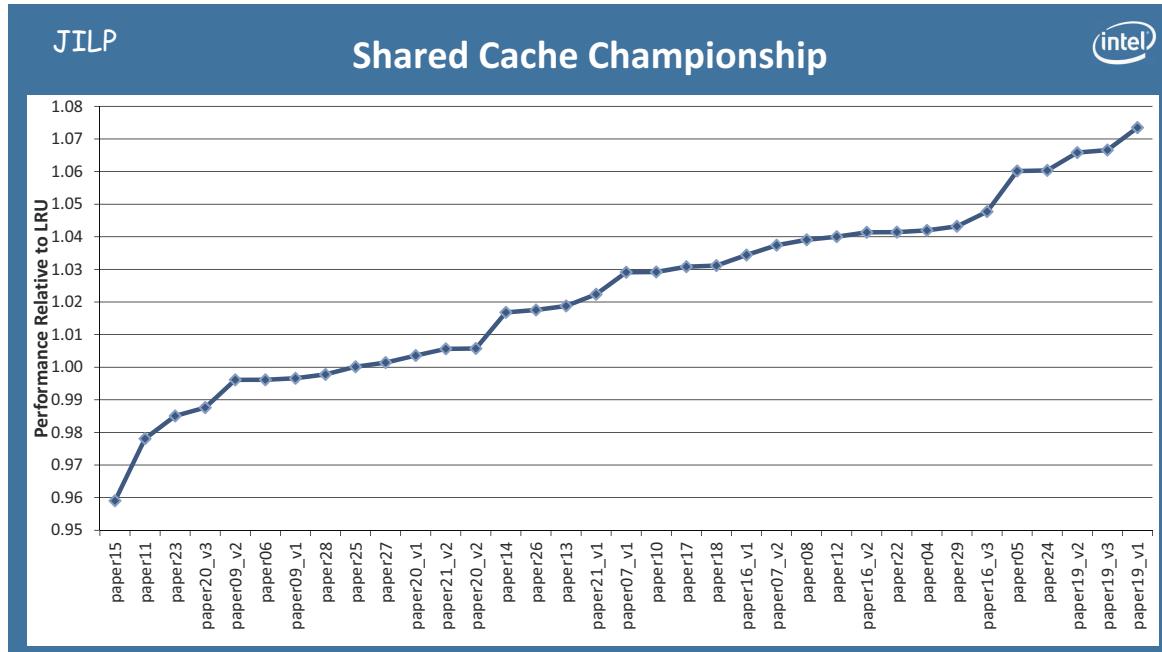


Σχήμα 47: Συγκεντρωτικά αποτελέσματα του μονονηματικού σκέλους του πρωταθλήματος για τους μηχανισμούς IbRDP (petoumenos) και DSB (gao)

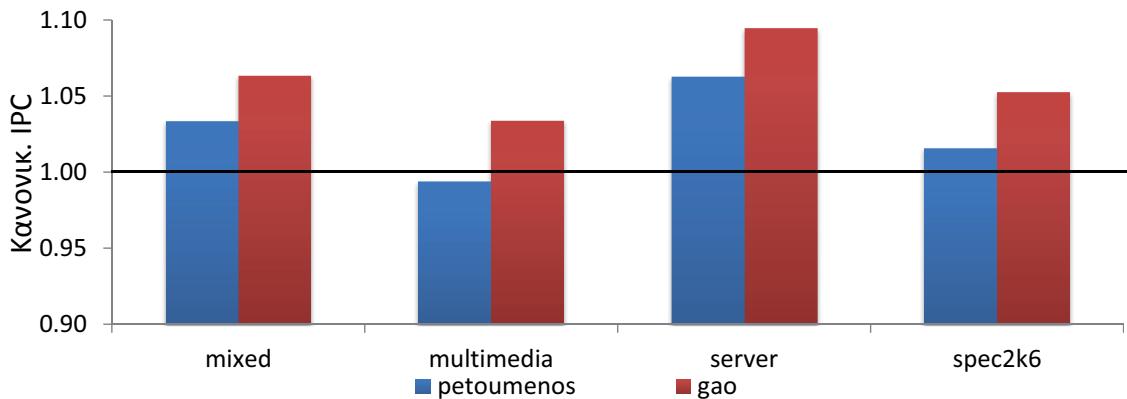


Σχήμα 48: Αναλυτικά αποτελέσματα του μονονηματικού σκέλους του πρωταθλήματος για τους μηχανισμούς IbRDP (petoumenos) και DSB (gao)

που είχαμε την ευκαιρία να μελετήσουμε κατά την ανάπτυξη του μηχανισμού μας, η διαφορά είναι μόλις 0.11%. Τα αναλυτικά αποτελέσματα παρουσιάζονται στο Σχήμα 48.



Σχήμα 49: Κατάταξη όλων των συμμετοχών στο πολυνηματικό σκέλος του πρωταθλήματος [42]



Σχήμα 50: Συγκεντρωτικά αποτελέσματα του πολυνηματικού σκέλους του πρωταθλήματος για τους μηχανισμούς IbRDP (petoumenos) και DSB (gao)

Στο Σχήμα 49 βλέπουμε την κατάταξη για το πολυνηματικό σκέλος. Γενικότερα, λόγω του ανταγωνισμού μεταξύ των εφαρμογών για τον έλεγχο της κρυφής μνήμης, τα περιθώρια βελτίωσης σε σχέση με την διαχείριση με LRU είναι αυξημένα: η βελτίωση που πετυχαίνει ο καλύτερος μηχανισμός φτάνει το 7.3% (2.9% στην ιδιωτική κρυφή μνήμη) και ενώ στην ιδιωτική κρυφή μνήμη οι μισοί μηχανισμοί δεν αυξάνουν το IPC σε σχέση με το LRU, εδώ μόνο 8 δεν πετυχαίνουν αύξηση. Η μεθοδολογία μας βελτίωσε την αύξηση του IPC σε 3.7% (1.3% στην ιδιωτική) και κατέλαβε την 10η θέση μεταξύ των 26 μεθοδολογιών. Η πτώση κατά δύο θέσεις στην κατάταξη οφείλεται στην μη-εκμετάλλευση του αυξημένου διαθέσιμου αποθηκευτικού χώρου: ενώ ο διαθέσιμος χώρος για την υλοποίηση της πολιτικής αντικατάστασης αυξήθηκε από 129 Kbit σε 513Kbit, κρατήσαμε σταθερές τις

παραμέτρους και τα μεγέθη του Predictor και του RD Sampler, αξιοποιώντας έτσι μόνο τα 455.67 Kbit.

Το Σχήμα 50 δείχνει τα συγκεντρωτικά αποτελέσματα της σύγκρισης του μηχανισμού μας με τον DSB. Οι κατηγορίες multimedia, server και spec2k6 περιέχουν πειράματα όπου 4 εφαρμογές αυτής της κατηγορίας μοιράζονται την κρυφή μνήμη, ενώ η κατηγορία mixed περιέχει τετράδες με εφαρμογές όλων των κατηγοριών. Όπως και στα αποτελέσματα της ιδιωτικής κρυφής μνήμης, τα χειρότερα αποτελέσματα της IbRDP πολιτικής παράγονται για την κατηγορία των εφαρμογών πολυμέσων, όπου το IPC του επεξεργαστή μειώνεται κατά 0.63% σε σχέση με το LRU. Στις υπόλοιπες κατηγορίες πετυχαίνουμε βελτίωση του IPC: 6.25% στις εφαρμογές server, 1.58% στα SPEC και 3.35% στην κατηγορία mixed. Σε όλες τις κατηγορίες ο μηχανισμός DSB συμπεριφέρεται αισθητά καλύτερα από τον IbRDP, με σχετική διαφορά ανάμεσα στους δύο μηχανισμούς από 3% έως 4%. Τα αναλυτικά αποτελέσματα, τέλος, παρουσιάζονται στο Σχήμα 51.

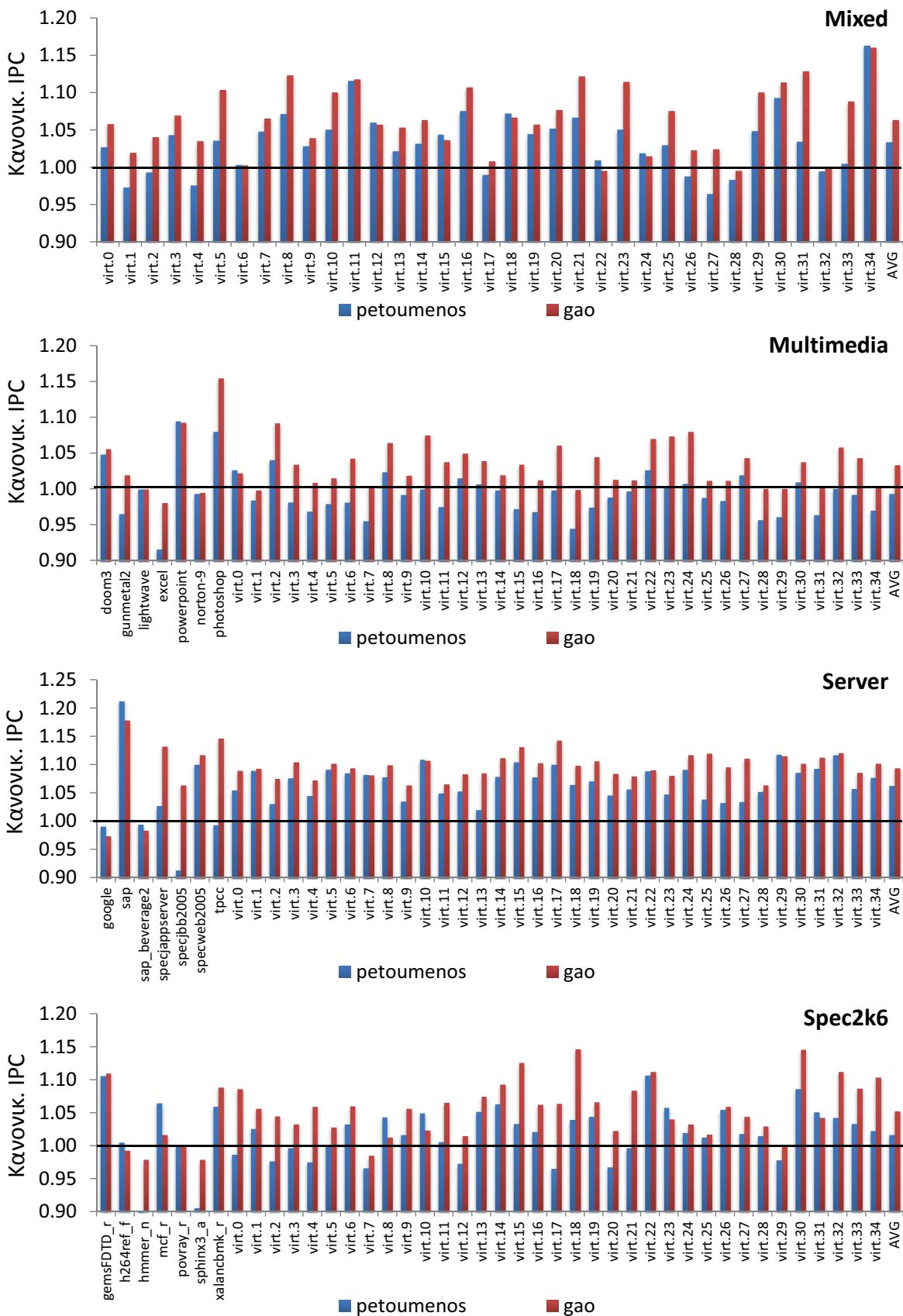
4.8 Προηγούμενες Ερευνητικές Εργασίες

Η μεθοδολογία που περιγράφθηκε σε αυτό το κεφάλαιο καλύπτει δύο ερευνητικές περιοχές: την μελέτη και βελτίωση των πολιτικών αντικατάστασης σε κρυφές μνήμες και τις τεχνικές πρόβλεψης της συμπεριφοράς των εφαρμογών σε ότι αφορά την ιεραρχία κρυφών μνημών. Στην συνέχεια της ενότητας αναφέρουμε τις κυριότερες ερευνητικές εργασίες και στις δύο αυτές περιοχές.

4.8.1 Πολιτικές Αντικατάστασης

Όπως είδαμε σε πολλαπλά σημεία της διατριβής, η βελτιστοποίηση της λειτουργίας της κρυφής μνήμης είναι εξαιρετικά κρίσιμη για την αποδοτική λειτουργία του επεξεργαστή. Για αυτόν το λόγο, η μελέτη και βελτίωση των πολιτικών αντικατάστασης αποτελούσε και αποτελεί μία πολύ ενεργή ερευνητική περιοχή.

Η πρώτη σημαντική ερευνητική προσπάθεια για την βελτίωση των πολιτικών αντικατάστασης ήταν αυτή του Laszlo Belady [7], ο οποίος το 1966 εισήγαγε την θεωρητικά βέλτιστη πολιτική αντικατάστασης, απέδειξε το ότι είναι βέλτιστη και σύγκρινε τα αποτελέσματα την διαχείρισης με LRU ή Random πολιτικές αντικατάστασης με αυτά της βέλτιστης διαχείρισης. Οι Suguman και Abraham [83] και ο Olivier Temam [87] προσάρμοσαν την ιδέα της βέλτιστης πολιτικής αντικατάστασης ώστε να λαμβάνεται



Σχήμα 51: Αναλυτικά αποτελέσματα του πολυνηματικού σκέλους του πρωταθλήματος για τους μηχανισμούς IbRDP (petoumenos) και DSB (gao)

υπόψιν τόσο η χρονική, όσο και η χωρική τοπικότητα. Αυτές οι εργασίες, αν και πολύ σημαντικές, βοήθησαν περισσότερο στο να τεθούν τα θεμέλια για περαιτέρω έρευνα στο αντικείμενο αυτό και για να γίνει κατανοητός ο τρόπος λειτουργίας των αλγορίθμων αντικατάστασης, παρά στο να προτείνουν μία ρεαλιστική λύση, αφού ο αλγόριθμος του Belady προϋποθέτει τέλεια γνώση των μελλοντικών προσπελάσεων της μνήμης.

Στηριζόμενοι σε αυτές τις θεωρητικές εργασίες, πολλοί ερευνητές μελέτησαν μεθοδολογίες για την βελτίωση των πολιτικών αντικατάστασης, είτε δυναμικές, είτε στατικές. Χαρακτηριστικό παράδειγμα δυναμικής μεθοδολογίας είναι ο LRU-K αλγόριθμος [66][65]. Η κεντρική ιδέα του συγκεκριμένου αλγορίθμου είναι η αποθήκευση της σχετικής σειράς, όχι μόνο της τελευταίας προσπέλασης κάθε γραμμής (όπως το LRU), αλλά των K τελευταίων προσπελάσεων. Με αυτήν την πληροφορία, όταν ο LRU-K αλγόριθμος αναζητεί γραμμή προς αντικατάσταση, διαλέγει την γραμμή που η K-οστή προηγούμενη προσπέλασή της βρίσκεται πιο μακριά στο παρελθόν. Ο πιο αποτελεσματικός μηχανισμός της LRU-K οικογένειας παράγεται όταν το K παίρνει την τιμή 2 [92].

Ένα άλλο παράδειγμα δυναμικής πολιτικής αντικατάστασης, είναι αυτό που προτάθηκε από τους Goncalves et al. [31] και μετέπειτα από τους Milutinovic et al. [62]. Σε αυτές τις προσεγγίσεις, η κρυφή μνήμη χωριζόταν στατικά σε δύο μέρη. Στο πρώτο μέρος αποθηκεύονταν τα δεδομένα που χαρακτηρίζονταν από χωρική τοπικότητα, ενώ στο δεύτερο μέρος τα δεδομένα που χαρακτηρίζονταν από χρονική τοπικότητα. Σε επόμενες ερευνητικές εργασίες, προτάθηκαν λύσεις για τον δυναμικό διαμοιρασμό της κρυφής μνήμης ανάμεσα στο χωρικό και το χρονικό κομμάτι, με βασικότερο παράδειγμα την εργασία των Kampe και Dahlgren [43]. Μία διαφορετική μεθοδολογία προτάθηκε από τους Karlsson και Hagersten [45]. Σύμφωνα με την μεθοδολογία τους, όλα τα δεδομένα που προορίζονται για τοποθέτηση στην κρυφή μνήμη πρώτου επιπέδου, αρχικά αποθηκεύονται σε μία μικρή κρυφή μνήμη, με το όνομα K1. Η συμπεριφορά του δεδομένου κατά την παραμονή του στην K1 καθορίζει και το αν τελικά θα τοποθετηθεί στην L1 ή όχι.

Οι Jeong και Dubois [37][36] προσπάθησαν να βελτιώσουν την LRU πολιτική αντικατάστασης εισάγοντας διαφορετικά κόστη ανάμεσα σε αστοχίες εντολών ανάγνωσης και εγγραφής. Στην συνέχεια, οι συγγραφείς επέκτειναν την μεθοδολογία τους με έναν μηχανισμό πρόβλεψης του είδους της επόμενης προσπέλασης κάθε γραμμής της κρυφής μνήμης, ώστε να δίνουν προτεραιότητα για αντικατάσταση στα δεδομένα που θα προσπελαστούν με εντολές εγγραφής [38].

Μια διαφορετική προσέγγιση του προβλήματος παρουσιάστηκε στην εργασία των Hu, Kaxiras και Martonosi [33]. Σύμφωνα με τους συγγραφείς, είναι δυνατόν να προβλεφθεί η διάρκεια ζωής ενός δεδομένου στην κρυφή μνήμη. Αυτή η προσέγγιση αρχικά χρησιμοποιήθηκε για την μείωση της στατικής ισχύος στις κρυφές μνήμες πρώτου επιπέδου [46], καθώς και για την προφόρτωση δεδομένων στην κρυφή μνήμη [33]. Η αξιοποίηση της timekeeping μεθοδολογίας για τον σχεδιασμό αλγορίθμων αντικατάστασης ερευνήθηκε από τους Takagi και Hiraki [85]. Οι συγγραφείς κατέληξαν σε έναν αλγόριθμο αντικατάστασης που συλλέγει και αποθηκεύει τους χρόνους μεταξύ διαδοχικών προσβάσεων στα δεδομένα ενός set της κρυφής μνήμης. Με αυτήν την πληροφορία, ο αλγόριθμος αντικατάστασης προσπαθεί να βρει ποιο δεδομένο πρέπει να αντικατασταθεί κάθε φορά μετά από μία αστοχία κρυφής μνήμης.

Όλες οι προηγούμενες μεθοδολογίες, αν και πολύ επιτυχημένες για τις κρυφές μνήμες πρώτου επιπέδου, δεν μπορούν να πετύχουν υψηλή απόδοση στις κρυφές μνήμες των κατωτέρων επιπέδων, όπως αποδείχτηκε στην εργασία των Abella et al. [1]. Για την ανίχνευση των “νεκρών” δεδομένων στις κρυφές μνήμες των κατώτερων επιπέδων προτάθηκαν διάφορα άλλα σχήματα. Σε αυτά, η εύρεση των νεκρών δεδομένων γίνεται είτε με βάση την διεύθυνση της τελευταίας εντολής που προσπέλασε το κάθε δεδομένο [51][57][89][94], είτε με βάση την διεύθυνση του ίδιου του δεδομένου [19][40][39]. Οι τεχνικές της δεύτερης κατηγορίας χαρακτηρίζονται από μεγάλη ακρίβεια πρόβλεψης, αλλά απαιτούν μεγάλους χρόνους εκπαίδευσης και μεγάλους αποθηκευτικούς χώρους. Οι μηχανισμοί της πρώτης κατηγορίας στηρίζονται στην εργασία των Lai και Falsafí [56] για την βελτιστοποίηση των cache coherency πρωτόκολλων σε πολυ-επεξεργαστικά περιβάλλοντα.

4.8.2 Πρόβλεψη των αποστάσεων επαναχρησιμοποίησης

Οι αποστάσεις επαναχρησιμοποίησης (Ενότητα 4.2.1) είναι ένας βολικός τρόπος περιγραφής της τοπικότητας των προσπελάσεων ενός προγράμματος και για αυτόν τον λόγο αρκετές ερευνητικές εργασίες έχουν επικεντρωθεί στην ανάλυση και στην πρόβλεψη τους [9][21][67][96]. Προηγούμενες εργασίες απέδειξαν ότι οι αποστάσεις επαναχρησιμοποίησης τόσο του προγράμματος [21][96], όσο και των εντολών [25][26], μπορούν α προβλεφθούν χρησιμοποιώντας ειδικές εισόδους στο πρόγραμμα. Στις εργασίες [25][26] επιπλέον παρουσιάστηκε μία μεθοδολογία, η οποία συσχετίζει τις αποστάσεις επαναχρησιμοποίησης που εμφανίζονται σε ένα πρόγραμμα με το μέγεθος του data set του

προγράμματος, με στόχο την πρόβλεψη κατά την μεταγλώττιση του προγράμματος του ποσοστού ευστοχιών στις κρυφές μνήμες πρώτου και δευτέρου επιπέδου.

4.9 Ανακεφαλαίωση

Σε αυτό το κεφάλαιο παρουσιάστηκε μια μεθοδολογία για την πρόβλεψη της απόστασης επαναχρησιμοποίησης των δεδομένων της κρυφής μνήμης, που βασίζεται στην συσχέτιση μεταξύ των εντολών προσπέλασης μνήμης και των αποστάσεων επαναχρησιμοποίησης των γραμμών. Στα πλαίσια της ανάπτυξης της μεθοδολογίας μας, υλοποιήθηκαν δύο ομάδες δομών συλλογής και πρόβλεψης αποστάσεων επαναχρησιμοποίησης, οι οποίες επιτυγχάνουν, με σχετικά μικρό μέγεθος και απλή λειτουργικότητα, υψηλή ακρίβεια στις προβλέψεις τους.

Στην συνέχεια αξιοποιήθηκε η πληροφορία που παράγουν οι δομές πρόβλεψης, ώστε να οδηγηθεί μία νέα πολιτική αντικατάστασης για κρυφές μνήμες τελευταίου επιπέδου που προσομοιώνει τις αποφάσεις της θεωρητικά βέλτιστης πολιτικής αντικατάστασης. Αποτιμήσαμε την συνολική μεθοδολογία σε δύο διαφορετικές σειρές πειραμάτων. Η πρώτη χρησιμοποίησε τον εξομοιωτή Simplescalar και τις πιο απαιτητικές σε μνήμη εφαρμογές της σουίτας SPEC2000, ενώ η δεύτερη χρησιμοποίησε τον εξομοιωτή CMPSim και ως είσοδο προγράμματα από την σουίτα SPEC CPU2006 και traces από πραγματικά υπολογιστικά συστήματα που εκτελούν εμπορικές εφαρμογές. Και οι δύο ομάδες πειραμάτων απέδειξαν την ανωτερότητα της μεθοδολογίας μας σε σχέση με τις πιο πετυχημένες πολιτικές αντικατάστασης που προτάθηκαν τα τελευταία χρόνια, όσον αφορά την μείωση των αστοχιών κρυφής μνήμης και την αύξηση της απόδοσης του επεξεργαστή.

Κεφάλαιο 5

Διαχείριση της Ουράς Εντολών διατηρώντας το MLP

5.1 Εισαγωγή

Τα τελευταία χρόνια, η ενεργειακή αποδοτικότητα των πυρήνων υψηλής απόδοσης έγινε ένα ιδιαιτέρως σημαντικό θέμα στην Αρχιτεκτονική Υπολογιστών. Αρκετές ερευνητικές εργασίες επικεντρώθηκαν στην μείωση της κατανάλωσης ενέργειας των δομών του επεξεργαστή, προσπαθώντας, όμως, ταυτόχρονα να διατηρήσουν την υψηλή απόδοση του επεξεργαστή, στον βαθμό που αυτό είναι εφικτό. Μία από τις βασικές κατηγορίες ερευνητικών εργασιών στο θέμα της μείωσης της κατανάλωσης ενέργειας αφορά την διαχείριση της Ουράς Εντολών (Instruction Queue - IQ). Η Ουρά εντολών είναι από τις πιο ενεργειακά απαιτητικές δομές του επεξεργαστή, εξαιτίας του μεγέθους της, των πλήρως συσχετιστικών αναζητήσεων που υποστηρίζει, και της υψηλής συχνότητας των προσβάσεων της ουράς.

Τρεις προτάσεις, από τους Buyuktosunoglu et al.[13], τους Folegnani και González[27], και τους Kucuk et al.[55], συνοψίζουν ιδανικά τα χαρακτηριστικά των τεχνικών διαχείρισης της Ουράς Εντολών: η κεντρική ιδέα είναι η μείωση της κατανάλωσης ενέργειας μειώνοντας δυναμικά το μέγεθος της Ουράς Εντολών, ώστε αυτή να προσαρμοστεί στις ανάγκες το προγράμματος, ενώ ταυτόχρονα ένας βρόχος ανατροφοδότησης παρακολουθεί τα αποτελέσματα της διαχείρισης και αυξάνει το μέγεθος της Ουράς, αν παρατηρηθεί

σημαντική επιβράδυνση του επεξεργαστή. Η διαχειριζόμενη Ουρά Εντολών μπορεί να είναι είτε φυσικά κατατετμημένη σε τμήματα που μπορούν να απενεργοποιηθούν πλήρως [13][55] είτε λογικά κατατετμημένη [27]. Ενώ η φυσική κατάτμηση είναι πιο επιτυχής στο να μειώνει την κατανάλωση ενέργειας, η πιο έξυπνη διαχείριση που επιτρέπει η λογική κατάτμηση μειώνει σημαντικά την επιβράδυνση του επεξεργαστή. Στην δική μας εργασία χρησιμοποιούμε έναν συνδυασμό της φυσικής κατάτμησης της Ουράς εντολών και της πολιτικής αποφάσεων που παρουσιάζεται στην εργασία των Folegnani και González [27].

Κατά την μελέτη αυτών των εργασιών, είδαμε σε αρκετές περιπτώσεις ότι ακόμη και μικρές αλλαγές στο μέγεθος της Ουράς Εντολών είχαν δυσανάλογα επιβλαβή επίδραση στην απόδοση του επεξεργαστή. Χάρη στην σε βάθος μελέτη της κατάστασης του επεξεργαστή, όταν εμφανίζεται αυτή η συμπεριφορά, και στην προηγούμενη ενασχόλησή μας με τις εργασίες των Chou et al. [20], των Karkhanis και Smith [44], των Eyerman και Eeckhout [23] και των Qureshi et al. [72] μπορέσαμε να αναγνωρίζουμε το MLP σαν τον βασικό υπεύθυνο για την επιβράδυνση του επεξεργαστή. Πιο συγκεκριμένα, η μελέτη μας έδειξε ότι αν υπάρχουν παράλληλες αστοχίες στο τελευταίο επίπεδο της κρυφής μνήμης, τότε ο πιο σημαντικός παράγοντας, που καθορίζει τις επιπτώσεις της διαχείρισης της Ουράς Εντολών στην απόδοση του επεξεργαστή, είναι το αν θα διατηρηθεί ή όχι το MLP.

Ενώ, εκ των υστερών, αυτό το συμπέρασμα μοιάζει προφανές και εύκολα ενσωματώσιμο σε παλιότερες μεθοδολογίες διαχείρισης της Ουράς Εντολών, στην πραγματικότητα απαιτεί μία καινούργια προσέγγιση στο πρόβλημα της διαχείρισης της Ουράς Εντολών. Οι υπάρχοντες μηχανισμοί διαχείρισης βασίζουν τις αποφάσεις τους στην μελέτη της συμπεριφοράς του επεξεργαστή για μεγάλα παράθυρα χρόνου, συνήθως της τάξης των χιλιάδων κύκλων. Αντίθετα, η διαχείριση του MLP απαιτεί μία πολύ πιο λεπτομερή προσέγγιση που να παίρνει ανεξάρτητες αποφάσεις σε διαστήματα αντίστοιχα με τον χρόνο που χρειάζεται για την εκτέλεση των εντολών που χωράνε στην Ουρά Εντολών. Ο λόγος για αυτό είναι ότι παραλληλισμός ανάμεσα σε αστοχίες εμφανίζεται μόνο όταν οι αστοχίες προκλήθηκαν από εντολές που βρίσκονταν ταυτόχρονα στο παράθυρο εντολών. Επομένως το MLP είναι ένα χαρακτηριστικό που συσχετίζεται ανεξάρτητα με κάθε ομάδα εντολών που βρίσκονται ταυτόχρονα στο παράθυρο εντολών. Στην μεθοδολογία που παρουσιάζουμε σε αυτό το κεφάλαιο, χρησιμοποιούμε συνδυασμένα την Ουρά Εντολών και τον Re-order Buffer, στην μορφή του Register Update Unit [79], οπότε το μέγεθος του παραθύρου εντολών και της Ουράς Εντολών συμπίπτουν.

Οι συνεισφορές της μεθοδολογίας μας είναι:

- Δείχνουμε ότι το MLP, όταν υπάρχει, είναι ο σημαντικότερος παράγοντας που επηρεάζει την απόδοση του επεξεργαστή κατά την διαχείριση της Ουράς Εντολών.
- Προτείνουμε έναν τρόπο για να ενσωματωθεί η έννοια του MLP στις υπάρχουσες τεχνικές διαχείρισης, μετρώντας και προβλέποντας το MLP που εμφανίζεται σε κάθε μικρό κομμάτι της εκτέλεσης του προγράμματος.
- Προτείνουμε έναν πρακτικό μηχανισμό που αξιοποιεί την προαναφερθείσα πρόβλεψη και δείχνουμε ότι η προσέγγισή μας επιτυγχάνει να βελτιώσει το συνολικό EDP του επεξεργαστή, όχι μόνο σε σχέση με μία μη-διαχειριζόμενη Ουρά Εντολών, αλλά και σε σχέση με τις υπάρχουσες τεχνικές διαχείρισης.

5.2 Προηγούμενες Ερευνητικές Εργασίες

5.2.1 Μείωση της κατανάλωσης ενέργειας του παράθυρου εντολών

Το παράθυρο εντολών, δηλαδή ο Re-order Buffer, η Ουρά Εντολών και η Load/Store Queue, είναι τυπικός στόχος τεχνικών που στοχεύουν στην μείωση της κατανάλωσης ενέργειας, για δύο λόγους: πρώτον, οι δομές του παράθυρου εντολών καταναλώνουν σημαντικό μέρος της ενέργειας του επεξεργαστή, και δεύτερον, οι δομές αυτές είναι σχεδιασμένες ώστε να υποστηρίζουν την μέγιστη δυνατή απόδοση του επεξεργαστή, απόδοση που επιτυγχάνεται μόνο σε μικρό μέρος της λειτουργίας του.

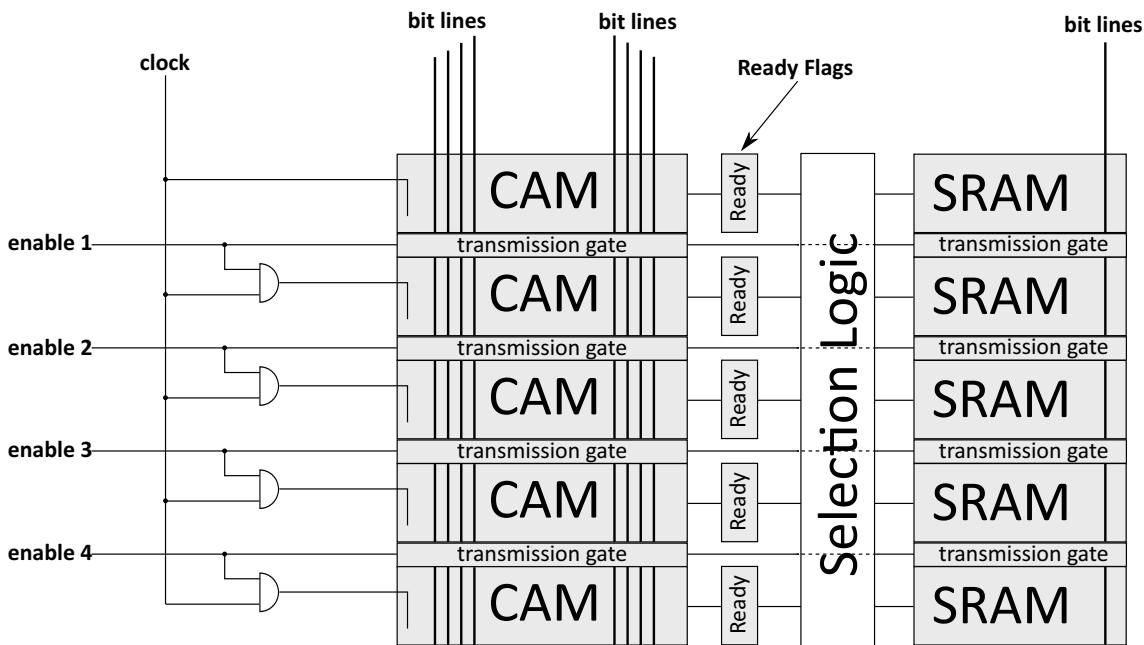
Η βάση των περισσότερων τεχνικών διαχείρισης που μεταβάλλουν το μέγεθος της Ουράς Εντολών είναι η εργασία του David Albonesi “Dynamic IPC/clock rate optimization” [2]. Η τεχνική διαχείρισης που προτείνεται σε αυτή την εργασία δεν έχει σαν στόχο την μείωση της κατανάλωσης ενέργειας, αλλά την μείωση του χρόνου απόκρισης των δομών του επεξεργαστή. Πιο συγκεκριμένα, ο Albonesi βασίστηκε στην παρατήρηση ότι όσο μικραίνει η τεχνολογία ολοκλήρωσης, οι καθυστερήσεις των καλωδίων θα αυξάνονται σε σχέση με τις καθυστερήσεις των πυλών, οπότε για να μείνει ο χρόνος απόκρισης των μεγάλων δομών του επεξεργαστή σε ανεκτά επίπεδα, χρειάζεται η κατάτμηση των μεγάλων καλωδίων παρεμβάλλοντας απομονωτές ανάμεσα στα τμήματα. Ο Albonesi συνειδητοποίησε ότι, σε μία τέτοια αρχιτεκτονική, η φυσική απομόνωση και απενεργοποίηση των τμημάτων των καλωδίων δεν απαιτεί σχεδόν κανένα επιπλέον

κόστος. Βασιζόμενος σε αυτές τις παρατηρήσεις, πρότεινε στην συνέχεια έναν μηχανισμό που δυναμικά ενεργοποιεί ή απενεργοποιεί τμήματα των καλωδίων και τα αντίστοιχα τμήματα των δομών που τροφοδοτούνται από τα καλώδια, ώστε να μειώσει τον χρόνο που απαιτείται για την διάδοση των σημάτων στην δομή. Με αυτόν τον τρόπο, όταν το πρόγραμμα δεν αξιοποιεί πλήρως μία μεγάλη δομή, το μέγεθός της μπορεί να μειώνεται με αντίστοιχη μείωση του χρόνου απόκρισής της.

Επεκτείνοντας αυτήν την ιδέα, αρκετοί ερευνητές πρότειναν την κατάτμηση της Ουράς Εντολών και την δυναμική απενεργοποίηση των τμημάτων της, ώστε να επιτευχθεί μείωση της κατανάλωσης ενέργειας. Από τις μεθοδολογίες που προτάθηκαν, τρεις είναι οι κυριότερες και πιο σχετικές με την παρούσα μεθοδολογία. Η πρώτη προτάθηκε από τους Buyuktosunoglu et al.[13] για την διαχείριση μίας Ουράς Εντολών, φυσικά κατατετμημένης με τρόπο αντίστοιχο με αυτόν της [2] (Σχήμα 52). Ο αλγόριθμος που διαχειρίζεται την ουρά οδηγείται από τον μέσο αριθμό εγγραφών της ουράς που είναι έτοιμες να εκτελεστούν (τα ορίσματά τους έχουν υπολογιστεί) στην διάρκεια του παραθύρου μέτρησης. Αν αυτός ο αριθμός είναι κάτω από κάποιο όριο, τότε ο αλγόριθμος συμπεραίνει ότι οι περισσότερες εγγραφές της Ουράς Εντολών δεν συνεισφέρουν στο IPC του επεξεργαστή, οπότε απενεργοποιεί ένα τμήμα της ουράς. Αν αντίθετα, ο μέσος αριθμός έτοιμων εντολών είναι πάνω από κάποιο όριο, τότε ενεργοποιείται ένα ακόμη τμήμα. Επιπλέον, για να αποφευχθούν περιπτώσεις όπου ο αριθμός έτοιμων εντολών δεν συσχετίζεται πλήρως με το IPC, αν το μέσο IPC του προγράμματος στο παράθυρο μέτρησης μειωθεί υπερβολικά σε σχέση με το προηγούμενο παράθυρο, τότε ενεργοποιείται ένα τμήμα της ουράς.

Η δεύτερη προσέγγιση είναι αυτή των Kucuk et al.[55]. Οι συγγραφείς χρησιμοποιούν και πάλι μία φυσικά κατατμημένη ουρά εντολών, αλλά ο αλγόριθμος διαχείρισής τους οδηγείται από τον μέσο αριθμό των εγγραφών που είναι κατειλημένες, δηλαδή παρατηρείται ο αριθμός κατειλημένων εγγραφών στην διάρκεια του παραθύρου μέτρησης και, στο τέλος του παραθύρου, το μέγεθος της ουράς μειώνεται ώστε να γίνει ίσο με τον μέσο όρο του αριθμού αυτού. Παρόμοια με την [13], υπάρχει και εδώ ένας μηχανισμός που αναιρεί τις αποφάσεις διαχείρισης, αλλά σε αυτή την εργασία ενεργοποιείται από τον άθροισμα των κύκλων που μπλοκάρεται η είσοδος νέων εντολών στο παράθυρο εντολών, λόγω της πληρότητας της ουράς.

Η τρίτη σημαντική μεθοδολογία για την διαχείριση του μεγέθους της ουράς εντολών, προτάθηκε από τους Folegnani και González [27]. Η μεθοδολογία τους διαφοροποιείται



Σχήμα 52: Δομή μίας κατατετμημένης Ουράς Εντολών. Το κάθε τμήμα χωρίζεται σε CAM και SRAM κομμάτι. Το CAM κομμάτι υλοποιεί την λογική αφύπνισης, ενώ το SRAM κομμάτι αποθηκεύει την πληροφορία που χρειάζεται για την εκτέλεση της εντολής [49]

από τις υπόλοιπες στο ότι η κατάτμηση της Ουράς Εντολών είναι λογική αντί για φυσική. Αυτή η επιλογή επιτρέπει μεν πιο ευέλικτη διαχείριση, αλλά παρουσιάζει το μειονέκτημα ότι δεν μπορεί να αποφύγει την κατανάλωση των μεγάλων καλωδίων της ουράς, οπότε τα οφέλη του μηχανισμού περιορίζονται στην αποτροπή των tag matches στις απενεργοποιημένες εγγραφές. Ο αλγόριθμος διαχείρισης των Folegnani και González βασίζεται τον αριθμό των εντολών που εκτελούνται στην διάρκεια του παραθύρου μέτρησης, οι οποίες βρίσκονται κατά την στιγμή της εκτέλεσής τους στο νεότερο (λογικό) τμήμα της Ουράς Εντολών. Αν ο αριθμός αυτός βρίσκεται κάτω από κάποιο όριο, τότε απενεργοποιείται ένα ακόμη λογικό τμήμα. Τέλος, ο μοναδικός μηχανισμός αύξησης του μεγέθους σε αυτή την τεχνική είναι η περιοδική ενεργοποίηση ενός επιπλέον λογικού τμήματος. Παρόλα τα μειονεκτήματα αυτής της τεχνικής, τα πειράματα δείχνουν ότι ο αλγόριθμος διαχείρισης καταφέρνει πολύ καλά να προσαρμόζει το μέγεθος της Ουράς Εντολών στις ανάγκες του προγράμματος, και για αυτόν τον λόγο αποτελεί την βάση της μεθοδολογίας που παρουσιάζουμε στην συνέχεια του κεφαλαίου.

Κοινό χαρακτηριστικό και των τριών μεθοδολογιών είναι ότι παίρνουν τις αποφάσεις διαχείρισης με βάση κάποιο μέγεθος που σχετίζεται με τον βαθμό παραλληλισμού μεταξύ των εντολών που εμφανίζει το πρόγραμμα (Instruction Level Parallelism - ILP): ο μέσος

αριθμός έτοιμων εντολών στην πρώτη, ο μέσος αριθμός κατειλημμένων εγγραφών στην δεύτερη, και ο αριθμός εντολών που εκτελέστηκαν από το νεότερο κομμάτι της Ουράς Εντολών στην τρίτη μεθοδολογία. Όπως θα δείξουμε στην συνέχεια, η προστασία του ILP είναι μεν σημαντική, αλλά δεν είναι το μόνο κριτήριο που απαιτείται για την επιτυχημένη διαχείριση της ουράς εντολών.

Πέρα από μεθοδολογίες που στοχεύουν στην μείωση του ενεργού μεγέθους της Ουράς Εντολών, αρκετές ερευνητικές εργασίες πρότειναν αρχιτεκτονικές, που είτε αντιμετωπίζουν με διαφορετικούς τρόπους την κατανάλωση ενέργειας της ουράς, ή που αυξάνουν την ενεργειακή αποδοτικότητα της ουράς.

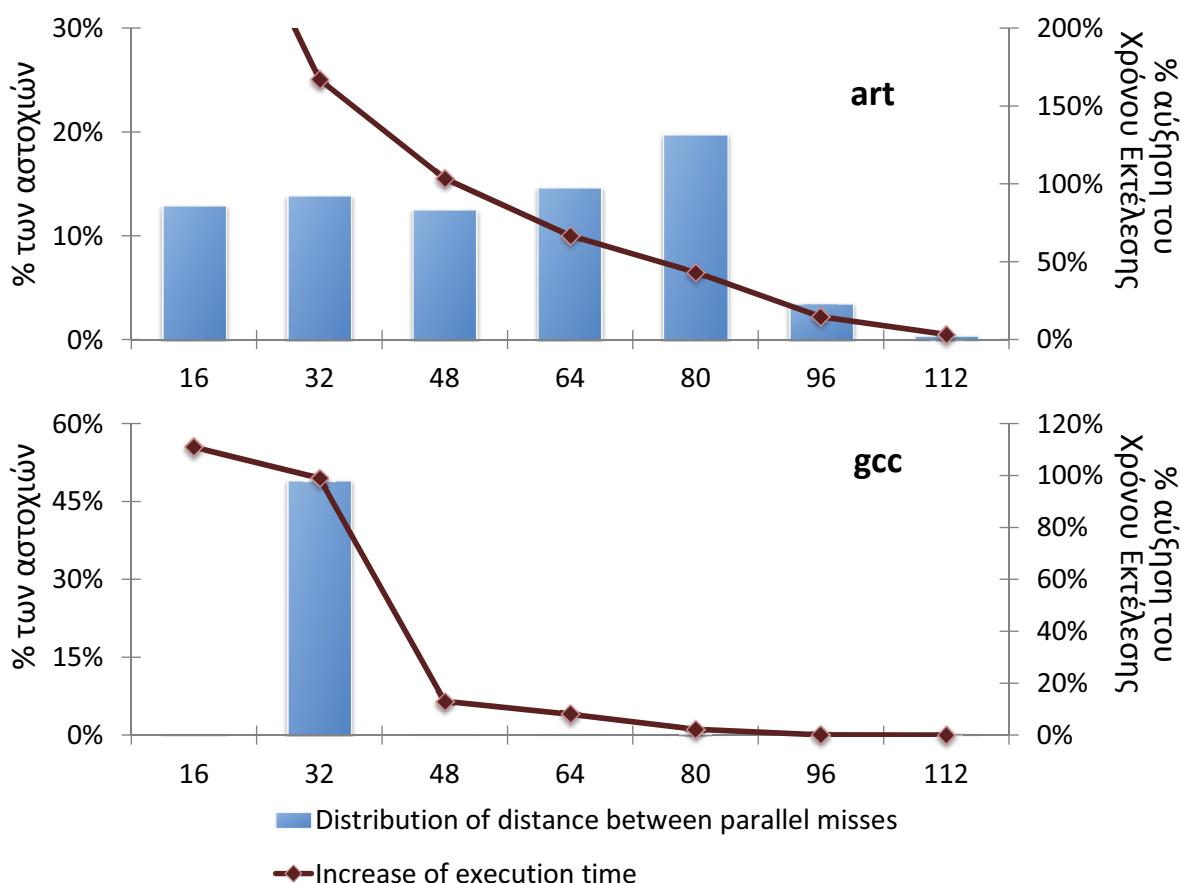
Ενδεικτικά, οι Palacharla, Jouppi, και Smith πρότειναν την μείωση της πολυπλοκότητας των βασισμένων σε CAM ουρών εντολών, αντικαθιστώντας τις με πολλαπλές FIFO ουρές [68]. Αντί για πλήρως συσχετιστικές αναζητήσεις σε ολόκληρο το παράθυρο εντολών, η αναζήτηση περιορίζεται στις κεφαλές των FIFO ουρών. Έπειτα, οι Canal και González [14][15], και ανεξάρτητα οι Michaud και Seznec [61], πρότειναν την χρήση dataflow πινάκων χρονοπρογραμματισμού, με επιπλέον πλήρως συσχετιστικούς buffers για τις λίγες εντολές με απρόβλεπτες διάρκειες εκτέλεσης. Στην εργασία [88], οι Tseng και Patt με μία παρόμοια λογική, ενσωματώνουν στο εκτελέσιμο πληροφορία για τις εντολές που σχηματίζουν κάθε αλυσίδα εξαρτήσεων και την χρησιμοποιούν στον χρόνο εκτέλεσης για να αναθέσουν σε κάθε τέτοια αλυσίδα το δικό της ξεχωριστό υποσύνολο των πόρων του πυρήνα. Με αυτόν τον τρόπο κάθε αλυσίδα εξαρτήσεων εκτελείται παράλληλα και ανεξάρτητα των άλλων, επιτυγχάνοντας απόδοση παρόμοια με των εκτός σειράς επεξεργαστών, αλλά εφόσον η εκτέλεση μέσα σε κάθε αλυσίδα γίνεται εντός σειράς, αποφεύγεται η πολυπλοκότητα και η κατανάλωση ενέργειας που σχετίζεται με την εκτός σειράς εκτέλεση.

5.2.2 MLP-aware τεχνικές

Η κρισιμότητα του MLP για την επίτευξη υψηλής επεξεργαστικής απόδοσης, καθώς και οι σημαντικότερες τεχνικές που εκμεταλλεύονται το MLP, έχουν ήδη αναφερθεί στην Ενότητα 2.4. Η τεχνική που έχει την περισσότερη συνάφεια με την μεθοδολογία μας, σε ότι αφορά το MLP, είναι αυτή που προτείνεται από τους Eyerman και Eeckhout [23].

Σε αυτή την εργασία, οι συγγραφείς μελετούν πολιτικές διαχείρισης του σταδίου fetch για SMT επεξεργαστές και δείχνουν ότι η πιο σημαντική πληροφορία, που χρειάζεται για

την αποτελεσματική διαχείριση του σταδίου fetch, είναι αν μία αστοχία μνήμης είναι μεμονωμένη ή παράλληλη. Αν είναι μεμονωμένη, τότε είναι πιο αποδοτικό να σταματήσει το fetch για το νήμα που προκάλεσε την αστοχία και να εισαχθούν εντολές στον επεξεργαστή για το άλλο νήμα που μοιράζεται τον επεξεργαστή. Αν όμως η αστοχία αναμένεται να είναι παράλληλη, τότε πρέπει να εισαχθούν στον επεξεργαστή αρκετές εντολές του ίδιου νήματος, ώστε να εκτελεστούν και οι υπόλοιπες προσπελάσεις που θα προκαλέσουν παράλληλη προσπέλαση της κύριας μνήμης. Η μεθοδολογία που προτείνουμε στην συνέχεια μοιράζεται αρκετά κοινά με την μεθοδολογία των Eyerman και Eeckhout, αλλά, στην περίπτωσή μας, συσχετίζουμε το MLP με περιοχές του κώδικα και όχι με μεμονωμένες εντολές πρόσβασης της μνήμης.



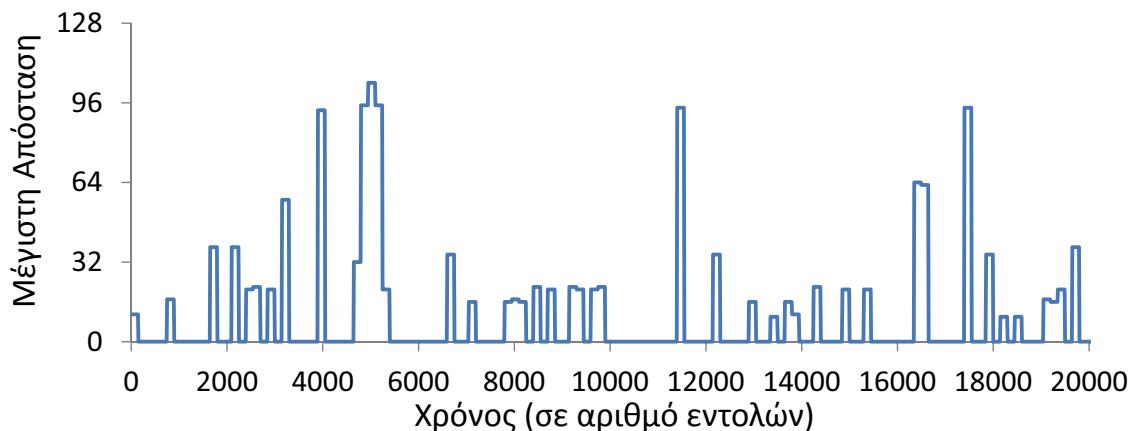
Σχήμα 53: Κατανομή των αποστάσεων (σε εντολές) μεταξύ παράλληλων αστοχιών στο τελευταίο επίπεδο κρυφής μνήμης, και αύξηση του χρόνου εκτέλεσης όταν το μέγεθος της Ουράς Εντολών μειώνεται στο μέγεθος που δείχνει ο x-άξονας, για το art και το gcc

5.3 MLP και μεταβολή του μεγέθους της Ουράς Εντολών

Στο Σχήμα 53 παρουσιάζεται η κατανομή του πλήθους των εντολών, που μεσολαβούν στο παράθυρο εντολών, ανάμεσα στις παράλληλες αστοχίες του art και του gcc (γαλάζιες μπάρες). Η κατανομή είναι ομαδοποιημένη σε διαστήματα των 16 εντολών, με τον x-άξονα να δείχνει το κάτω όριο της ομάδας (δηλαδή η μπάρα στο 16, αντιστοιχεί στις παράλληλες αστοχίες με αποστάσεις από 16 έως και 31 εντολές). Βλέπουμε ότι οι περισσότερες παράλληλες αστοχίες του art απέχουν μεταξύ τους από 1 έως 95 εντολές, με κάποιες λίγες ακόμη αστοχίες να απέχουν μεταξύ τους 96 έως 111 εντολές. Αντίθετα, στο gcc βλέπουμε ότι οι αποστάσεις μεταξύ των αστοχιών είναι πολύ περισσότερο συγκεντρωμένες: το 49% των συνολικών αστοχιών (99.4% των παράλληλων αστοχιών) απέχουν 32 έως 47 εντολές από άλλες παράλληλες αστοχίες. Επίσης, η κόκκινη γραμμή στο σχήμα μας δείχνει την ποσοστιαία αύξηση του χρόνου εκτέλεσης του προγράμματος, όταν το μέγεθος της ουράς εντολών μειώνεται στην αντίστοιχη τιμή του x-άξονα.

Κάθε φορά που μειώνεται το μέγεθος της ουράς εντολών, σύμφωνα με όσα αναφέραμε στα εισαγωγικά κεφάλαια, καταστρέφεται ο παραλληλισμός μεταξύ των αστοχιών που απέχουν μεταξύ τους περισσότερο από το νέο μέγεθος της ουράς εντολών. Στο art, όπου οι αποστάσεις αυτές είναι κατανεμημένες σε μεγάλο εύρος τιμών, ο επεξεργαστής επιβραδύνεται σημαντικά για σχεδόν κάθε μείωση του μεγέθους της ουράς και κυρίως για μεγέθη από 96 και χαμηλότερα, ακολουθώντας αρκετά στενά το MLP που χάνεται. Στο gcc, η κατάσταση είναι ακόμη πιο ξεκάθαρη: για μείωση του μεγέθους της ουράς εντολών μέχρι και 48 εγγραφές, η αύξηση του χρόνου εκτέλεσης οφείλεται αποκλειστικά στο χαμένο ILP και είναι ανεκτή, αλλά αν μειωθεί και άλλο το μέγεθος της ουράς, χάνεται σχεδόν όλο το MLP του προγράμματος, με τον χρόνο εκτέλεσης να αυξάνεται κατά 99%. Αυτά τα δύο παραδείγματα δείχνουν πόσο στενά συνδεδεμένες είναι η απόδοση του επεξεργαστή και το MLP και μας υποδεικνύουν ότι μία πολιτική μείωσης του μεγέθους της ουράς εντολών πρέπει να επικεντρώνεται στο MLP και όχι στο ILP.

Ένα άλλο χαρακτηριστικό του MLP, που κάνει δύσκολη την διαχείριση του από τους υπάρχοντες μηχανισμούς διαχείρισης, είναι ότι οι αποστάσεις μεταξύ των παράλληλων αστοχιών αλλάζουν πολύ γρήγορα στα περισσότερα προγράμματα. Στο Σχήμα 54 βλέπουμε ένα παράθυρο 20K εντολών από την εκτέλεση του twolf. Σε κάθε σημείο σημειώνεται η μέγιστη παρατηρούμενη απόσταση μεταξύ των παράλληλων αστοχιών για το αντίστοιχο χρονικό σημείο. Ιδανικά, ο μηχανισμός διαχείρισης της ουράς εντολών θα



Σχήμα 54: Μέγιστες αποστάσεις μεταξύ των αστοχιών του twolf στην διάρκεια ενός παραθύρου 20K εντολών

έπρεπε να ρυθμίζει το μέγεθος της ουράς, έτσι ώστε σε κάθε χρονική στιγμή να χωράει στην ουρά η μέγιστη απόσταση, χωρίς όμως να μεγαλώνει η ουρά πολύ περισσότερο από όσο χρειάζεται. Όπως βλέπουμε, δεν είναι δυνατόν να πετύχουμε αυτούς τους στόχους με ένα μέγεθος ουράς εντολών για όλο το παράθυρο: οποιοδήποτε μέγεθος μικρότερο από τις 96 εγγραφές θα καταστρέψει μέρος του MLP, αλλά αν δεν μειώσουμε το μέγεθος κάτω από τις 96 εγγραφές, θα χάσουμε τις ευκαιρίες για διαχείριση στα πολλά κομμάτια της εκτέλεσης όπου οι αστοχίες απέχουν ελάχιστα μεταξύ τους. Είναι προφανές λοιπόν, ότι ένας αποδοτικός μηχανισμός διαχείρισης της ουράς εντολών πρέπει να παίρνει αποφάσεις διαχείρισης πολύ πιο συχνά από ότι οι υπάρχοντες μηχανισμοί.

5.4 Διαχείριση της Ουράς Εντολών με πληροφορία για το MLP

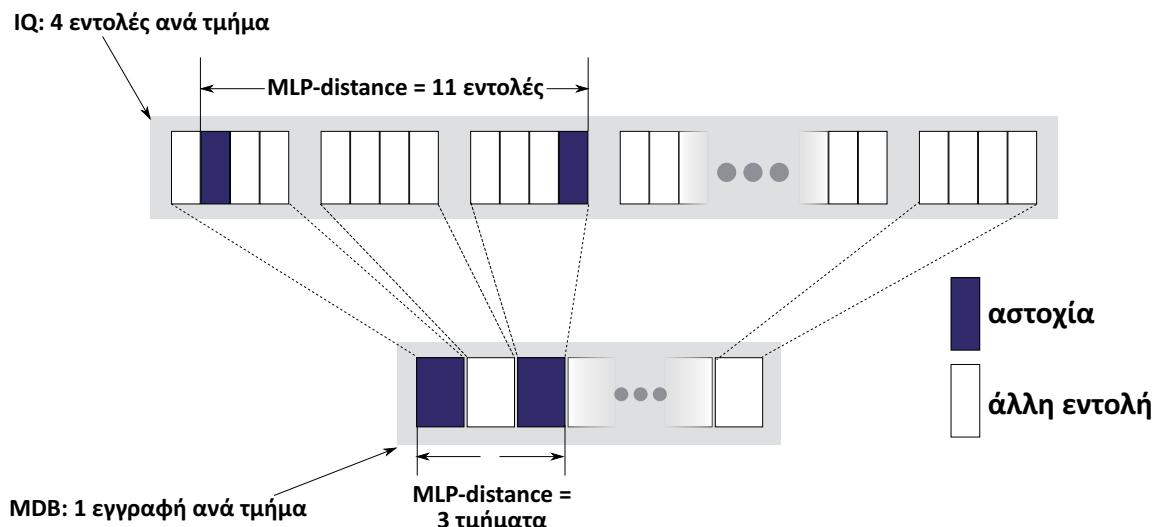
Η προσέγγισή μας στο πρόβλημα της διαχείρισης της ουράς εντολών είναι να ποσοτικοποιήσουμε το MLP που παρουσιάζεται κατά την εκτέλεση του προγράμματος και να συσχετίσουμε αυτήν την πληροφορία με τις αντίστοιχες περιοχές του προγράμματος μέσω μία δομής πρόβλεψης. Έτσι, όταν εκτελεστεί πάλι η ίδια περιοχή, μπορούμε να χρησιμοποιήσουμε την αποθηκευμένη πληροφορία για μεταβάλλουμε το μέγεθος της ουράς εντολών χωρίς να χάσουμε MLP.

5.4.1 Ποσοτικοποιώντας το MLP: MLP-distance

Ο πρώτος στόχος μας είναι να ποσοτικοποιήσουμε τα χαρακτηριστικά του MLP με τέτοιο τρόπο ώστε να μπορούμε να το χρησιμοποιήσουμε για την διαχείριση της ουράς εντολών. Όπως είδαμε, δύο εντολές που αστοχούν στο τελευταίο επίπεδο κρυφής μνήμης

παραλληλίζουν τις αστοχίες τους, αν δεν υπάρχουν εξαρτήσεις μεταξύ τους και ο αριθμός των εντολών που μεσολαβούν μεταξύ τους στο παράθυρο εντολών είναι μικρότερος από το μέγεθος της ουράς εντολών. Αυτός ο αριθμός, που αποκαλείται MLP-distance στην εργασία των Eyerman και Eeckhout [23], είναι η βασική πληροφορία που συλλέγουμε και χρησιμοποιούμε για την διαχείριση της ουράς εντολών.

Ο πιο απλός τρόπος για να μετρηθεί το MLP-distance ανάμεσα σε δύο εντολές είναι να ελέγχουμε την ουρά load/store κάθε φορά που εξυπηρετείται μία αστοχία κρυφής μνήμης και να βρίσκουμε την νεότερη εντολή που περιμένει ακόμη για δεδομένα από την κύρια μνήμη. Αυτή η μέθοδος δεν ταιριάζει στους στόχους μας, αφού μπορεί να αναγνωρίσει παράλληλες αστοχίες μόνο για το τρέχον μέγεθος της ουράς εντολών. Αντίθετα, εμείς χρειαζόμαστε να ελέγχουμε για αστοχίες, όλες τις εντολές που θα μπορούσαν να παραλληλιστούν στην ουρά εντολών αν δεν υπήρχε διαχείριση. Ένας τρόπος λοιπόν να μαζέψουμε πληροφορία για τα MLP-distances που εμφανίζονται στο πρόγραμμα, είναι να κρατάμε πληροφορία για τις τελευταίες N εντολές, όπου N το μέγιστο μέγεθος της ουράς. Αυτός ο τρόπος, δεν είναι ιδιαίτερα αποδοτικός: για να κρατάμε αυτή την πληροφορία θέλουμε μία δομή τόσο μεγάλη όσο και η ουρά εντολών. Έτσι, η λύση στην οποία καταλήγουμε είναι να κρατάμε πληροφορία αθροιστικά για κάθε ομάδα εντολών, με την ομάδα να αποτελείται από διαδοχικές εντολές του προγράμματος που τοποθετούνται στην ίδια κατάτμηση της ουράς εντολών.



Σχήμα 55: MLP Distance Buffer για παράδειγμα ουράς εντολών με 4 εγγραφές ανά τμήμα

Η πληροφορία για το MLP συλλέγεται από έναν μικρό κυκλικό buffer —τον οποίο αποκαλούμε MLP Distance Buffer ή MDB— με τόσες εγγραφές όσο ο μέγιστος αριθμός κατατμήσεων της ουράς εντολών (ΣΧΗΜΑΑ). Κάθε φορά που εισέρχεται μία εντολή στο παράθυρο εντολών και τοποθετείται σε διαφορετικό τμήμα από την προηγούμενη εντολή, δεσμεύουμε και μία καινούργια εγγραφή στον MDB, αλλά σε αντίθεση με τις εγγραφές της ουράς εντολών, η πιο παλιά εγγραφή του MDB αποσύρεται όταν πρέπει να δεσμεύσουμε μία καινούργια εγγραφή. Αυτό σημαίνει ότι οι εγγραφές του MDB αποσύρονται μόνο όταν απέχουν από τις πιο καινούργιες εντολές της ουράς εντολής απόσταση μεγαλύτερη από το φυσικό μέγεθος της ουράς.

Η λογική με την οποία υπολογίζουμε τα MLP-distances είναι η εξής:

- Κάθε φορά που μία εντολή προκαλεί αστοχία στο τελευταίο επίπεδο της κρυφής μνήμης, η αντίστοιχη εγγραφή του MDB σημειώνεται ως εγγραφή που προκάλεσε αστοχία.
- Όταν μία εγγραφή του MDB αποσύρεται, ο MDB σαρώνεται για να βρεθούν οι υπόλοιπες εγγραφές που έχουν συσχετιστεί με αστοχία.
- Αν βρεθούν τέτοιες εγγραφές, αυτό σημαίνει ότι υπάρχει πιθανό MLP μεταξύ τους, οπότε υπολογίζουμε την απόσταση σε τμήματα ανάμεσα στην εγγραφή που αποσύρεται και την νεότερη εγγραφή με αστοχία, και ανανεώνουμε σε όλες τις εγγραφές το πεδίο που κρατάει το MLP-distance, αν η νέα τιμή είναι μεγαλύτερη από την αποθηκευμένη.

Το MLP-distance που μετράμε με αυτόν τον τρόπο δεν ανταποκρίνεται πλήρως στο πραγματικό MLP-distance. Για να παραλληλίσουν δύο εντολές τις αστοχίες τους, δεν αρκεί να βρίσκονται ταυτόχρονα στην ουρά εντολών, θα πρέπει και να μην υπάρχουν εξαρτήσεις ανάμεσα τους. Σε κάθε περίπτωση, το πραγματικό MLP-distance θα είναι μικρότερο ή ίσο από την τιμή που υπολογίζουμε, όποτε τα όποια λάθη στις μετρήσεις μας απλά θα μας στερήσουν κάποιες ευκαιρίες για μεγαλύτερη μείωση του μεγέθους της ουράς εντολών, δεν θα προκαλέσουν απώλεια MLP. Από τα πειράματά μας φαίνεται ότι οι υπερεκτιμήσεις του MLP-distance είναι σχετικά λίγες. Με δεδομένη την απλότητα του μηχανισμού μας, λίγα τέτοια σφάλματα είναι αποδεκτά.

5.4.2 Συσχέτιση του MLP-distance με περιοχές του προγράμματος

Η μέτρηση του MLP-distance μας επιτρέπει να ελέγξουμε το μέγεθος της ουράς εντολών, ώστε όταν εκτελείται το συγκεκριμένο κομμάτι του προγράμματος, να μην χάνουμε MLP. Για να ολοκληρωθεί όμως ο βρόχος, χρειαζόμαστε να αποθηκεύουμε αυτή την πληροφορία και να την συσχετίζουμε με περιοχές του προγράμματος, ώστε όταν ξαναεκτελεστεί η ίδια περιοχή, να μπορέσουμε να προβλέψουμε την συμπεριφορά της και να θέσουμε σωστά το μέγεθος της ουράς εντολών. Αυτές οι περιοχές πρέπει να είναι σε μέγεθος συγκρίσιμες με το φυσικό μέγεθος της ουράς εντολών. Πολύ μικρότερες περιοχές θα ήταν μη αποδοτικές για την διαχείριση ολόκληρης της ουράς, η οποία θα περιείχε πολλαπλές περιοχές κώδικα, με αντικρουόμενες ίσως απαιτήσεις, κάθε χρονική στιγμή. Πολύ μεγαλύτερες περιοχές κώδικα (για παράδειγμα φάσεις του προγράμματος), δεν θα μας παρείχαν την λεπτομερή πληροφορία που χρειαζόμαστε για την διαχείριση του ευμετάβλητου MLP-distance.

Για να πετύχουμε την επιθυμούμενη ισορροπία, ορίζουμε την έννοια των superpaths. Ένα superpath είναι ο συνδυασμός πολλαπλών διαδοχικά εκτελούμενων basic blocks και αντιστοιχεί περίπου στην έννοια του trace της trace cache [75] ή στην έννοια του hotspot [35]. Τα superpaths αναγνωρίζονται στο dispatch και κάθε καινούργια εγγραφή του MDB που δεσμεύεται αντιστοιχίας στο superpath που γίνεται dispatch εκείνη την στιγμή. Στην συνέχεια, κάθε φορά που αποσύρεται μία εγγραφή του MDB, ανανεώνεται η πρόβλεψη του MLP-distance για το superpath στο οποίο ανήκει η εγγραφή, με το αποθηκευμένο MLP-distance της. Έτσι, την επόμενη φορά που το ίδιο superpath αναγνωριστεί στο dispatch, διαβάζουμε την πρόβλεψη του MLP-distance και την αξιοποιούμε για την διαχείριση της ουράς εντολών.

5.4.3 Πολιτική διαχείρισης

Με την πρόβλεψη του MLP-distance έχουμε μία ένδειξη για το ελάχιστο μέγεθος της ουράς εντολών που δεν μειώνει το MLP. Σε πολλές περιπτώσεις, όμως, είτε δεν υπάρχει MLP στην περιοχή κώδικα που εκτελείται ή το MLP-distance είναι πολύ μικρό για να περιορίσει την μείωση του μεγέθους της ουράς εντολών. Εφόσον, ο στόχος μας είναι να μειώσουμε την ουρά εντολών όσο το δυνατόν περισσότερο, χωρίς να επιβραδύνουμε σημαντικά την εκτέλεση του προγράμματος, σε αυτές τις περιπτώσεις θα πρέπει να διατηρήσουμε, εκτός από το MLP, και το ILP.

Για να το πετύχουμε, μπορούμε να χρησιμοποιήσουμε οποιαδήποτε από τις υπάρχουσες τεχνικές διαχείρισης, αλλά σε αυτό το κεφάλαιο περιοριζόμαστε στον μηχανισμό των Folegnani και González (Ενότητα 5.2.1). Ο τρόπος με τον οποίο συνδυάζουμε τις αποφάσεις των δύο μηχανισμών είναι ο εξής: η ILP-feedback προσέγγιση μας δίνει το μέγεθος της ουράς εντολών που δεν μειώνει το ILP, ενώ με την MLP-aware προσέγγισή μας κρίνουμε αν αυτό το μέγεθος μειώνει το MLP, και σε αυτή την περίπτωση αναιρούμε την απόφαση του ILP-feedback μηχανισμού.

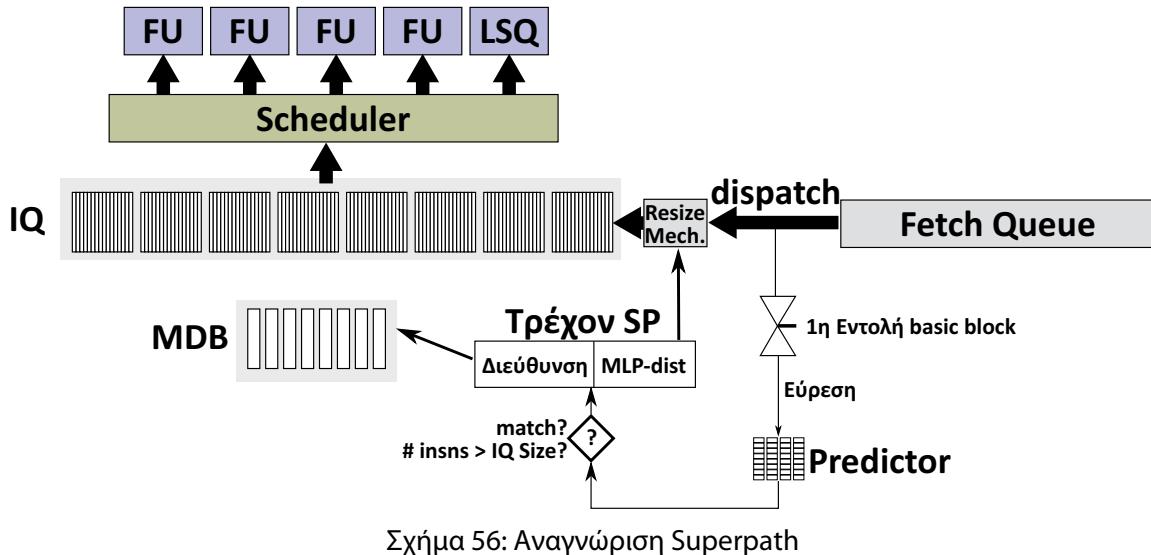
5.5 Επιλογές Υλοποίησης

5.5.1 Κατάτμηση της ουράς εντολών

Για να επιτραπεί η δυναμική μεταβολή του μεγέθους της ουράς εντολών, μοιράζουμε την ουρά σε πολλαπλά ανεξάρτητα τμήματα. Πιο συγκεκριμένα, στην υλοποίησή μας, χρησιμοποιούμε μία ουρά 128 εγγραφών χωρισμένη σε 8 τμήματα των 16 εγγραφών. Η αλλαγή του μεγέθους της ουράς επιτυγχάνεται με κατάτμηση των bitlines [29], παρόμοια με αυτήν που παρουσιάζεται στο Σχήμα 52. Η λειτουργικότητα της ουράς εντολών ακολουθεί αυτή που χρησιμοποιείται στην εργασία των Folegnani και González: η ουρά εντολών είναι δομημένη σαν FIFO, με τις καινούργιες εντολές να εισάγονται στο τέλος της ουράς και τις παλιές να αποσύρονται από την κεφαλή της ουράς. Το σημείο όπου διαφέρουν οι δύο υλοποιήσεις είναι ότι, στην περίπτωσή μας απενεργοποιούνται ολόκληρα φυσικά τμήματα της ουράς.

Στην μεθοδολογία μας, ο ILP-feedback μηχανισμός αποφάσεων απενεργοποιεί ένα segment, όταν οι εντολές που εκτελούνται από το νεότερο (λογικό) τμήμα της ουράς, στην διάρκεια ενός παραθύρου μέτρησης, είναι λιγότερες από μία τιμή κατωφλίου. Κάθε πέντε παράθυρα μέτρησης ένα ακόμη φυσικό τμήμα ενεργοποιείται. Το παράθυρο της μέτρησης είναι 1000 κύκλοι. Η αυξομείωση του μεγέθους της ουράς εντολών ενεργοποιώντας ή απενεργοποιώντας φυσικά τμήματα, οδηγεί σε περιορισμούς αντίστοιχους με αυτούς που αντιμετώπισαν οι Ponomarev et al. [70]: η μείωση του μεγέθους της ουράς επιτρέπεται μόνο όταν δεν έχουν μείνει έγκυρες εγγραφές στο τμήμα που απενεργοποιούμε, ενώ η αύξηση του μεγέθους γίνεται όταν το λογικό τέλος της ουράς φτάσει στο τέλος του τελευταίου ενεργού τμήματος. Αυτοί οι δύο περιορισμοί, προσθέτουν μία χρονική καθυστέρηση ανάμεσα στην λήψη μίας απόφασης διαχείρισης και στην εφαρμογή της, με την καθυστέρηση στο χειρότερο σενάριο να είναι ίση με τον χρόνο που χρειάζεται για να

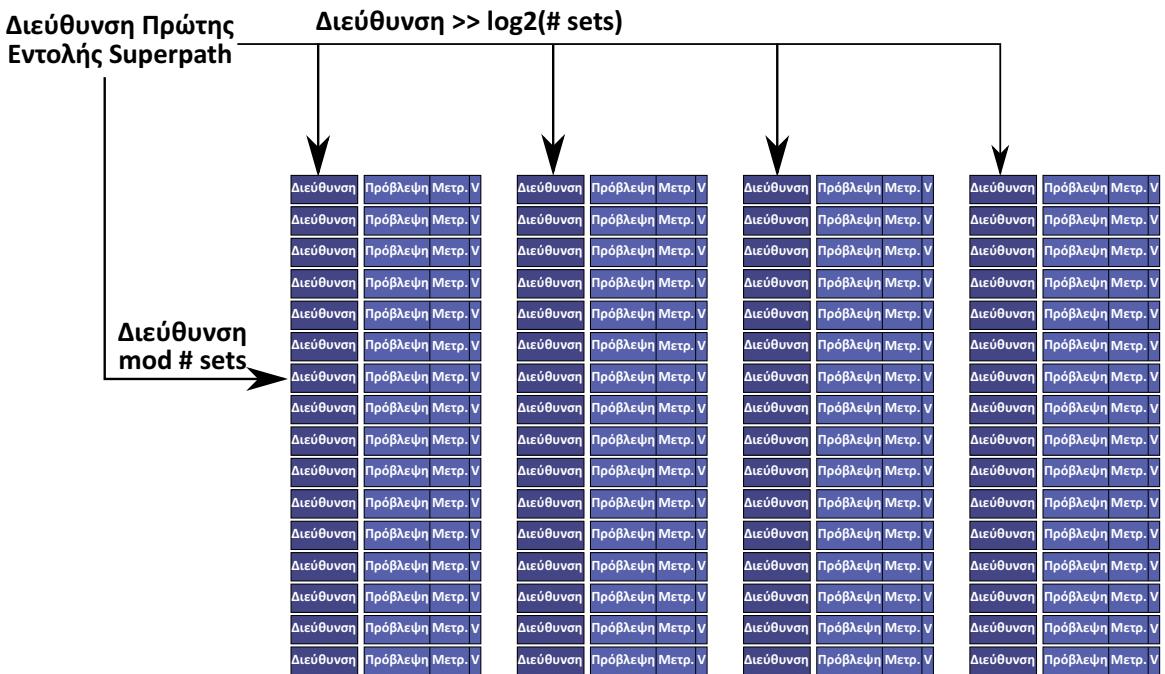
αποσυρθούν όλες οι εντολές που βρίσκονται στην ουρά. Η ύπαρξη αυτής της καθυστέρησης επιβεβαιώνει την ορθότητα της απόφασής μας να διαχειριστούμε το MLP σε ανάλυση συγκρίσιμη με το φυσικό μέγεθος της ουράς.



5.5.2 Superpaths

Η βασική μονάδα διαχείρισης της ουράς εντολών, το superpath, χαρακτηρίζεται από την διεύθυνση της πρώτης της εντολής και το μέγεθος της. Διαδοχικά basic blocks ομαδοποιούνται στο ίδιο superpath στο στάδιο του dispatch, τουλάχιστον μέχρι το superpath να ξεπεράσει σε μέγεθος την ουρά εντολών. Μετά από αυτό το σημείο, η παραγωγή του superpath διακόπτεται και ξεκινάει ένα καινούργιο, όταν φτάσει στο στάδιο dispatch η πρώτη εντολή ενός υπάρχοντος superpath ή όταν το μέγεθος του superpath φτάσει σε υπερδιπλάσιο μέγεθος από την ουρά εντολών. Προτιμώντας να τελειώνουμε ένα superpath εκεί ακριβώς που ξεκινά ένα υπάρχον, μειώνουμε και τον αριθμό και την επικάλυψη των superpaths. Η πλήρης διαδικασία με την οποία αναγνωρίζουμε ένα superpath παρουσιάζεται στο Σχήμα 56.

Για κάθε καινούργιο superpath δεσμεύουμε μία εγγραφή σε μία μικρή δομή που αποθηκεύει τα χαρακτηριστικά του superpath και το προβλεπόμενο MLP-distance. Η δομή αυτή παρουσιάζεται στο Σχήμα 57 και είναι παρόμοια με τον Instruction-based Reuse Distance Predictor που παρουσιάστηκε στην Ενότητα 4.2.3. Ο Predictor σε αυτή την μεθοδολογία έχει όμως αρκετά μικρότερο μέγεθος: 64 εγγραφές οργανωμένες σε 4 ways. Σε κάθε εγγραφή αποθηκεύουμε 27 bits: 20 bits για την αρχή διεύθυνση του superpath (αγνοούμε τα 2 χαμηλότερα bits, τα indexing bits και τα bits 26 έως 63), 3 bits για την



Σχήμα 57: Predictor για πρόβλεψη των MLP-distances

πρόβλεψη του MLP-distance (μετρημένη σε πλήθος τμημάτων), 3 bits για τον μετρητή αξιοπιστίας και ένα valid bit. Συνολικά, ο αποθηκευτικός χώρος που απαιτούμε για τον predictor είναι 1728 bits (216 bytes). Σύμφωνα με το CACTI [86], ο Predictor συνεισφέρει 9.5 mW στην συνολική κατανάλωση ισχύος του επεξεργαστή, το οποίο είναι είναι ένα αποδεκτό κόστος συγκρινόμενο με την κατανάλωση της ουράς εντολών (8.9W για την ουρά που εξομοιώθηκε).

5.6 Πειραματικά Αποτελέσματα

5.6.1 Πειραματική μεθοδολογία

Για την αποτίμηση της απόδοσης της προτεινόμενης μεθοδολογίας χρησιμοποιήσαμε τον εξομοιωτή HotLeakage, ο οποίος συνδυάζει τον εξομοιωτή SimpleScalar με την μοντελοποίηση της κατανάλωσης ισχύος του WATTCH [12]. Τα χαρακτηριστικά του εξομοιωμένου υπολογιστικού συστήματος περιγράφονται στον Πίνακα 3. Για τις εξομοιώσεις μας χρησιμοποιήσαμε ένα υποσύνολο των προγραμμάτων της σουίτας SPEC2000, επιλέγοντας τα προγράμματα με τουλάχιστον 1 εντολή ανάγνωσης με μεγάλο χρόνο απόκρισης ανά 1K εντολές για το μικρότερο μέγεθος L2 που εξομοιώσαμε. Προγράμματα με λιγότερες αστοχίες δεν παρουσιάζουν ενδιαφέρον για την παρούσα μεθοδολογία, καθώς χωρίς αστοχίες όλες οι αποφάσεις λαμβάνονται αποκλειστικά από τον

ILP-feedback μηχανισμό. Τα στατιστικά που παρουσιάζουμε λήφθηκαν από εξομοίωση 300M εντολών, αγνοώντας την φάση της αρχικοποίησης (3B εντολές για το ammp, 2B εντολές για το vpr, το twolf και το mcf, 1B για τα υπόλοιπα προγράμματα).

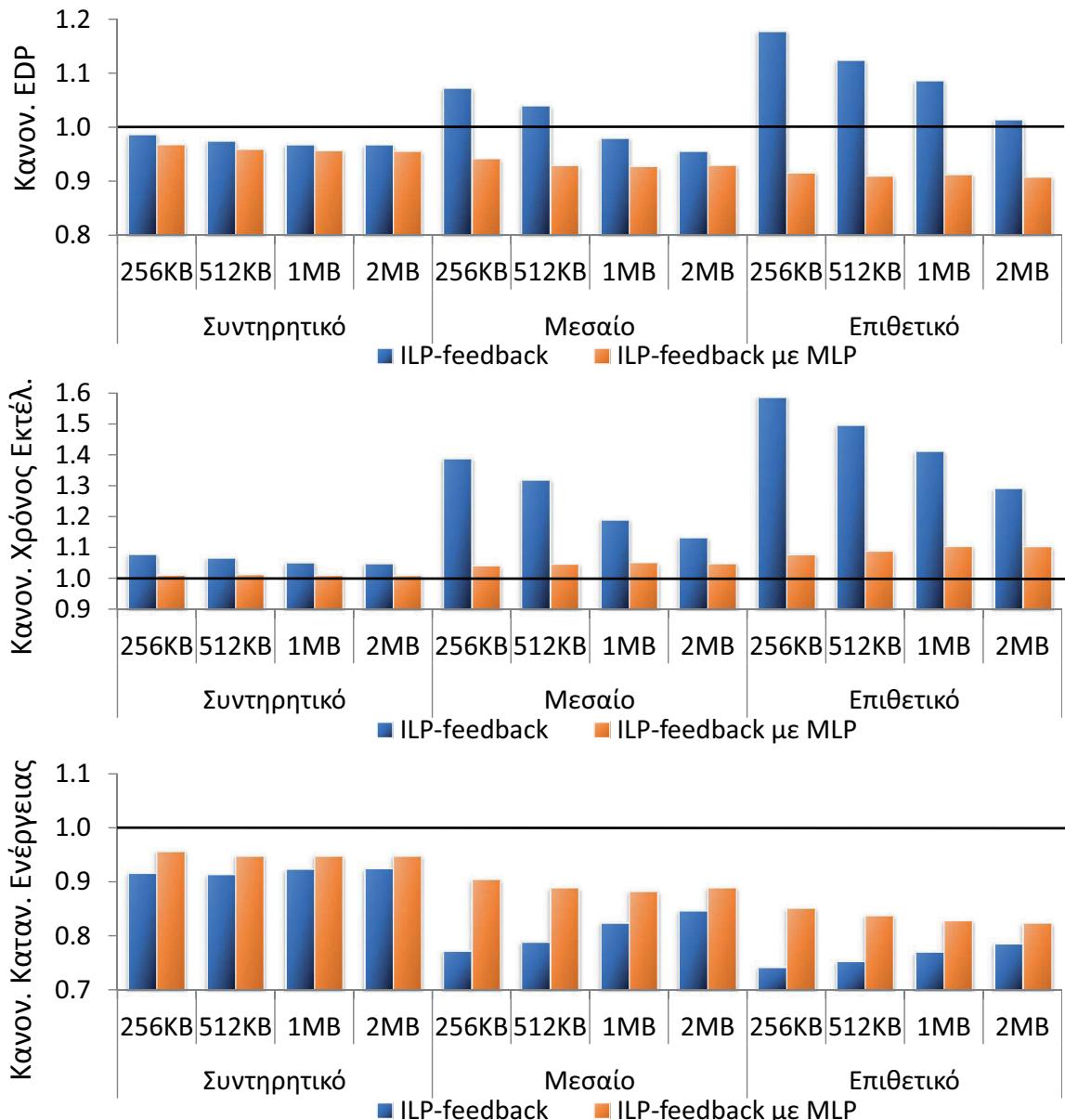
Πίνακας 3: Παράμετροι του εξομοιωμένου υπολογιστικού συστήματος

Παράμετρος	Χαρακτηριστικά
Fetch/Issue/Commit width	4 εντολές ανά κύκλο
Branch Predictor	Υβριδικός με bimodal και 2-Level
BTB	1024 εγγραφές, 4-way
Ουρά Εντολών	128 εγγραφές, φυσικά κατατετμημένη σε τμήματα των 16 εγγραφών, ενοποιημένη με το ROB
Load/Store Queue	64 εγγραφές
Κρυφή Μνήμη L1 εντολών	16KB, 4-way, 64 bytes/block
Κρυφή Μνήμη L1 δεδομένων	8KB, 4-way, 32 bytes/block
Κρυφή Μνήμη L2	Διάφορα Μεγέθη (256KB-2MB), 8-way, 64 bytes/block
TLBs	4K για εντολές, 4K για δεδομένα
Κύρια Μνήμη	120 κύκλοι χρόνος απόκρισης, πλάτος 8 bytes
Λειτουργικές Μονάδες	4 integer ALUs, 2 integer multipliers, 2 floating point ALUs, 2 floating point multipliers

Κάθε μηχανισμός και κάθε πρόγραμμα εξομοιώθηκαν για τέσσερα διαφορετικά μεγέθη κρυφής μνήμης. Τα στατιστικά που μετρήσαμε και παρουσιάζουμε είναι η συνολική κατανάλωση ενέργειας του επεξεργαστή, ο χρόνος εκτέλεσης του προγράμματος και το γινόμενο της καταναλωμένης ενέργειας επί τον χρόνο εκτέλεσης (Energy-Delay Product - EDP). Στην πρώτη ομάδα μετρήσεων παρουσιάζουμε τα αποτελέσματα της ενσωμάτωσης του MLP-distance στον βρόχο λήψεως αποφάσεων, ενώ στην δεύτερη συγκρίνουμε τον βασικό ILP-feedback μηχανισμό με τον συνδυασμό ILP/MLP-feedback που προτείνουμε στην μεθοδολογία μας. Όλα τα αποτελέσματα είναι κανονικοποιημένα ως προς τις αντίστοιχες μετρήσεις για ουρά εντολών χωρίς διαχείριση.

5.6.2 Συνέπειες της MLP-distance πληροφορίας

Στην πρώτη ομάδα πειραμάτων μαζέψαμε στατιστικά για τους δύο μηχανισμούς με ακριβώς την ίδια τιμή κατωφλίου της ILP-feedback πολιτικής αποφάσεων, ώστε η μόνη διαφορά ανάμεσα στους δύο μηχανισμούς να είναι η αξιοποίηση της MLP-distance πληροφορίας από τον μηχανισμό μας. Στο Σχήμα 58 παρουσιάζονται ο γεωμετρικός μέσος του συνολικού EDP του επεξεργαστή και ο γεωμετρικός μέσος του χρόνου εκτέλεσης για



Σχήμα 58: Γεωμετρικοί μέσοι του EDP, του χρόνου εκτέλεσης και της κατανάλωσης ενέργειας για τον ILP-feedback και τον ILP/MLP-feedback μηχανισμό, για διάφορες τιμές κατωφλίου

τρεις διαφορετικές τιμές κατωφλίου: μία “επιθετική” τιμή (768 εντολές), μία μεσαία τιμή (512 εντολές) και μία “συντηρητική” τιμή (256 εντολές). Είναι εύκολα ορατό το πόσο δύσκολο είναι να βελτιωθεί ο γεωμετρικός μέσος του συνολικού EDP του επεξεργαστή. Ο βασικός λόγος για αυτό είναι ότι στην πειραματική μεθοδολογία μας, η ουρά εντολών αντιπροσωπεύει το σχετικά χαμηλό 13.7% της μέγιστης κατανάλωσης ισχύος του επεξεργαστή, οπότε η μεγαλύτερη βελτίωση του συνολικού EDP που μπορούμε να πετύχουμε δεν θα ξεπερνάει αυτό το ποσοστό.

Όπως βλέπουμε στο σχήμα, ο ILP-feedback μηχανισμός μόνος του πετυχαίνει οριακά θετικά αποτελέσματα μόνο για την συντηρητική τιμή κατωφλίου. Για τις υπόλοιπες τιμές

κατωφλίου, το κανονικοποιημένο EDP ή κινείται γύρω από την μονάδα (για τιμή 512) ή είναι ξεκάθαρα μεγαλύτερο (για τιμή κατωφλίου 768), γιατί παρά την σημαντική μείωση που πετυχαίνει στην κατανάλωση ενέργειας, αυξάνει δυσανάλογα τον χρόνο εκτέλεσης. Για παράδειγμα, με την μεταβολή της τιμής κατωφλίου από συντηρητική σε επιθετική, για κρυφή μνήμη 256KB, η κατανάλωση ενέργειας μειώνεται επιπλέον κατά 17.3%, όμως ο χρόνος εκτέλεσης αυξάνεται κατά 50.7%. Επίσης βλέπουμε ότι η αύξηση του χρόνου εκτέλεσης που προκαλεί ο ILP-feedback μηχανισμός, συσχετίζεται αντίστροφα με την αύξηση του μεγέθους της κρυφής μνήμης, δηλαδή όσο μεγαλώνει η κρυφή μνήμη και μειώνονται οι αστοχίες, τόσο πιο αποδοτική γίνεται η διαχείριση της ουράς εντολών. Αυτή η συμπεριφορά είναι μία ακόμη ένδειξη ότι η προβληματική συμπεριφορά των ILP-aware μηχανισμών διαχείρισης οφείλεται στο MLP.

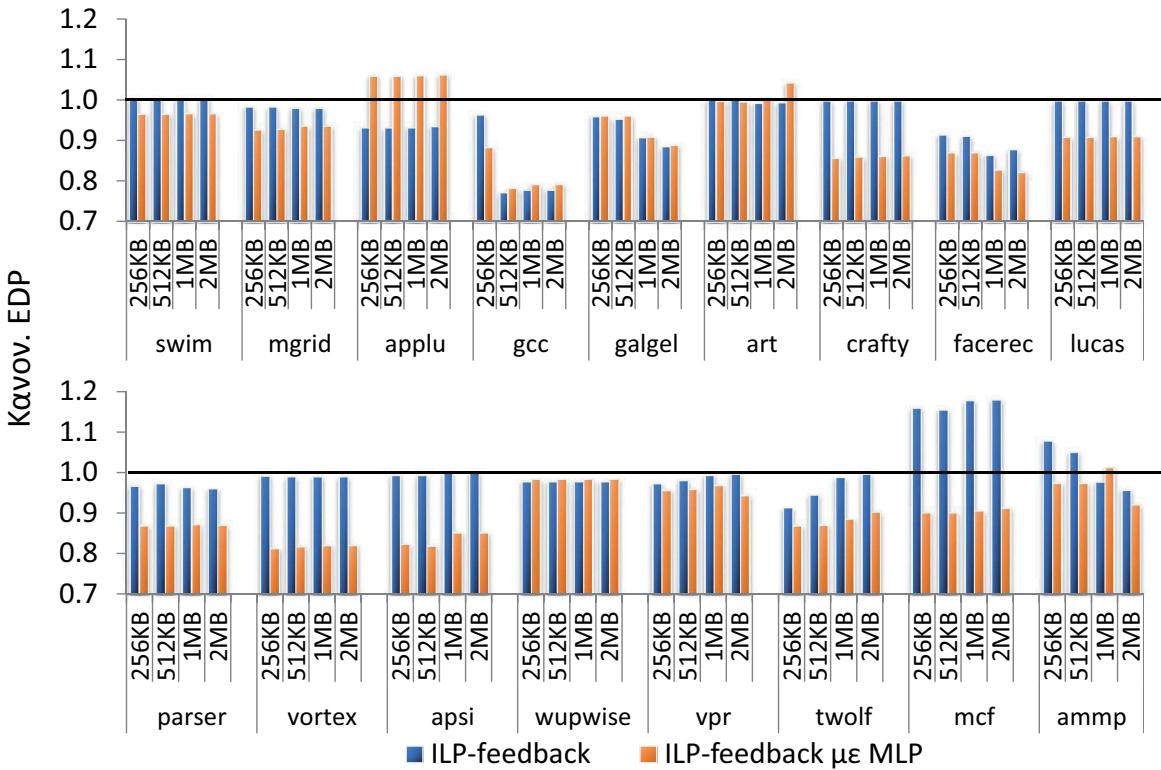
Αυτό το συμπέρασμα αποδεικνύεται και από τα αποτελέσματα του ILP/MLP-aware μηχανισμού. Ενώ ο μηχανισμός μας πετυχαίνει αρκετά μικρότερες μειώσεις της κατανάλωσης ενέργειας σε σχέση με τον βασικό ILP μηχανισμό, για την ίδια τιμή κατωφλίου, βλέπουμε ότι αυτό οφείλεται στο ότι προσπαθεί να διατηρήσει το MLP και την απόδοση του επεξεργαστή σε υψηλά επίπεδα. Σαν αποτέλεσμα, ο μηχανισμός μας πετυχαίνει καλύτερο EDP για όλες τις ομάδες μετρήσεων και, το κυριότερο, συνεχίζει να βελτιώνει το EDP και για πιο επιθετικές τιμές κατωφλίου αγγίζοντας έτσι το 10% στην συνολική βελτίωση του EDP.

Συνολικότερα, αυτό που προκύπτει από τις μετρήσεις είναι ότι, ο ILP-feedback μηχανισμός επιτυγχάνει να μειώσει μεν το EDP, αλλά το πετυχαίνει μόνο όταν η τιμή κατωφλίου είναι αρκετά χαμηλή ώστε να μην μειώνει το MLP. Αν μεγαλώσει η τιμή κατωφλίου περισσότερο, η επιβράδυνση του επεξεργαστή γίνεται μη-αποδεκτή. Αντίθετα, ο μηχανισμός μας αποσυνδέει την διαφύλαξη του ILP από το MLP, και έτσι μπορεί να μειώσει ακόμη πιο επιθετικά το μέγεθος της ουράς εντολών σε βάρος του ILP, αλλά προστατεύοντας ανεξάρτητα την παράμετρο που έχει σημασία: το MLP.

5.6.3 Σύγκριση της μεθοδολογίας ILP/MLP-aware με την ILP-feedback

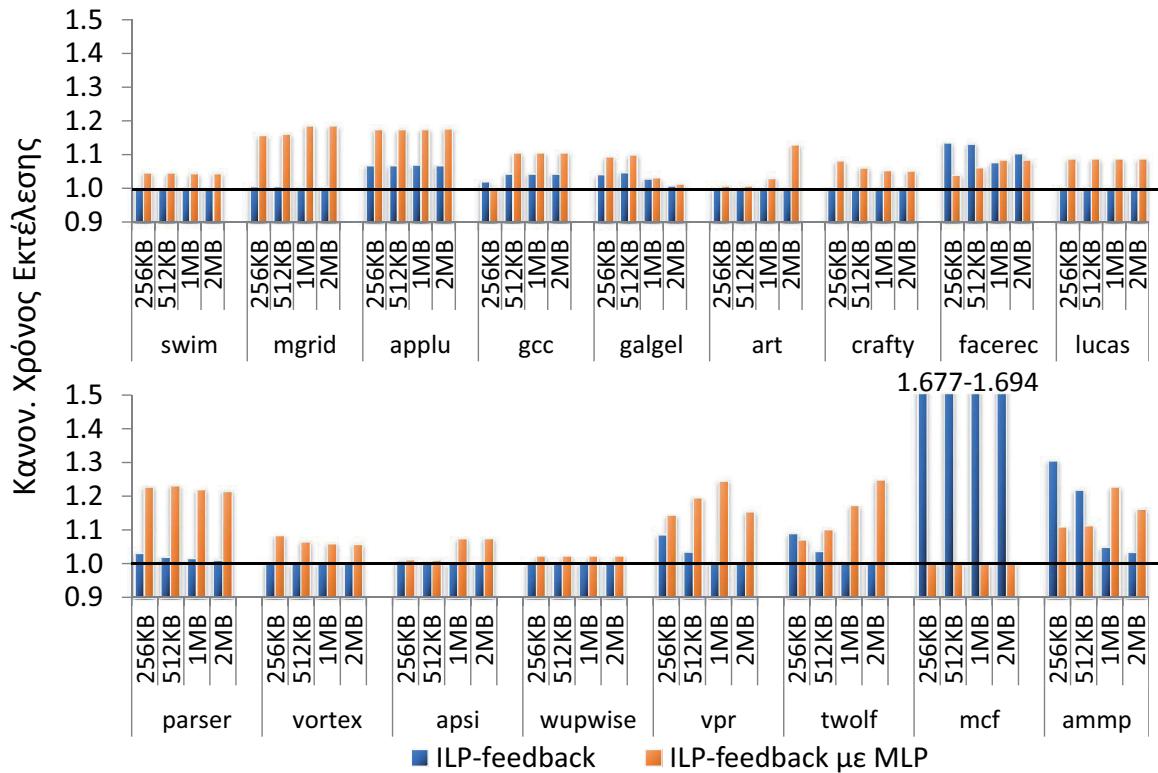
Σε αυτή την ενότητα, κάνουμε την τελική σύγκριση των δύο μηχανισμών για τις τιμές κατωφλίου που ελαχιστοποιούν το συνολικό EDP του κάθε μηχανισμού, χωρίς να αυξάνουν σημαντικά το EDP πολλών μεμονωμένων εφαρμογών. Για τον ILP-feedback μηχανισμό, όμως, αυτό στάθηκε αδύνατο, γιατί παρουσιάζει περιπτώσεις με αυξημένο EDP για όλες τις

τιμές κατωφλίου που παράγουν μετρήσιμη μειώση του συνολικού EDP, οπότε διαλέξαμε την τιμή που απλά ελαχιστοποιεί το συνολικό EDP, δηλαδή 256 εντολές. Ο μηχανισμός που προτείνουμε, όπως είδαμε και από τις προηγούμενες μετρήσεις, μπορεί να πιέσει ακόμη περισσότερο την ουρά εντολών, και έτσι διαλέξαμε τιμή κατωφλίου 768 εντολές.

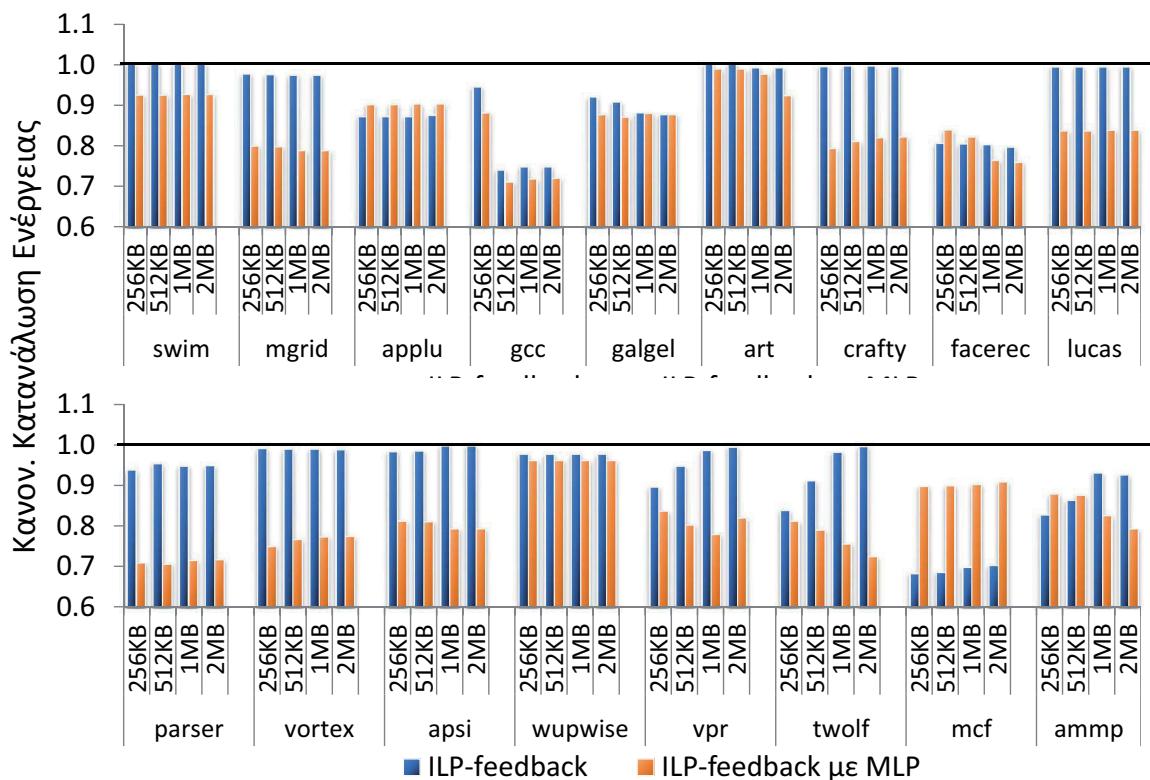


Σχήμα 59: Αναλυτικά αποτελέσματα για το EDP όλων των προγραμμάτων, για διαχείριση με ILP-feedback ή ILP/MLP-feedback, με τους δύο μηχανισμούς ρυθμισμένους με την βέλτιστη τιμή κατωφλίου τους

Στο Σχήμα 59, το Σχήμα 60 και το Σχήμα 61 βλέπουμε το EDP, τον χρόνο εκτέλεσης και την κατανάλωση ενέργειας για τους δύο μηχανισμούς, κανονικοποιημένα ώς προς τα αντίστοιχα μεγέθη χωρίς διαχείριση. Από αυτά τα αποτελέσματα φαίνεται ότι ο μηχανισμός μας, παρά την πολύ πιο επιθετική μείωση του μεγέθους της ουράς εντολών, προκαλεί για τα περισσότερα προγράμματα συγκρίσιμη μείωση της απόδοσης του επεξεργαστή με τον συντηρητικό ILP-feedback μηχανισμό, ενώ ταυτόχρονα επιτυγχάνει να μειώσει την κατανάλωση ενέργειας του επεξεργαστή πολύ περισσότερο. Στις περισσότερες από τις περιπτώσεις όπου αυξάνουμε αρκετά περισσότερο τον χρόνο εκτέλεσης από τον ILP-feedback μηχανισμό, πετυχαίνουμε και αντίστοιχα πολύ μεγαλύτερα οφέλη σε μείωση κατανάλωσης ενέργειας, οπότε μειώνουμε το EDP. Από όλα τα πειράματα που εκτελέσαμε, μόνο σε 6 ο μηχανισμός μας αυξάνει το EDP πάνω από 1% σε σχέση με τον ILP-feedback

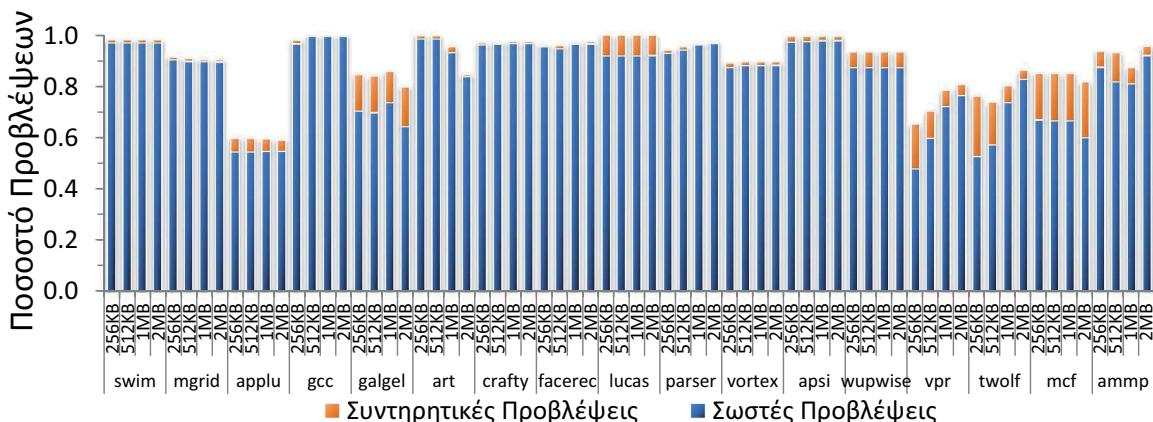


Σχήμα 60: Αναλυτικά αποτελέσματα για τον χρόνο εκτέλεσης όλων των προγραμμάτων, για διαχείριση με ILP-feedback ή ILP/MLP-feedback, με τους δύο μηχανισμούς ρυθμισμένους με την βέλτιστη τιμή κατωφλίου τους



Σχήμα 61: Αναλυτικά αποτελέσματα για την κατανάλωση ενέργειας όλων των προγραμμάτων, για διαχείριση με ILP-feedback ή ILP/MLP-feedback, με τους δύο μηχανισμούς ρυθμισμένους με την βέλτιστη τιμή κατωφλίου τους

μηχανισμό: για το applu σε όλες τα μεγέθη κρυφής μνήμης, για το art στα 2MB και για το ammp στο 1MB.



Σχήμα 62: Σωστές Προβλέψεις και λάθος, αλλά Συντηρητικές, Προβλέψεις του Predictor

Για να καταλάβουμε τους λόγους για αυτή την συμπεριφορά, στο Σχήμα 62 φαίνεται η ακρίβεια των προβλέψεων και το ποσοστό λάθος μεν, αλλά συντηρητικών, προβλέψεων. Βλέπουμε ότι για τα περισσότερα προγράμματα και μεγέθη κρυφών μνημών, η ακρίβεια του Predictor ξεπερνά το 80%, και για τα υπόλοιπα προγράμματα μεγάλο μέρος των λάθος προβλέψεων είναι συντηρητικές, δηλαδή υποθέτουμε παραλληλισμό μεταξύ προσπελάσεων που εκτελούνται σειριακά. Σαν αποτέλεσμα μόνο για τρία προγράμματα ξεπερνούν οι επιθετικά λάθος προβλέψεις το 20%: για το applu, το vpr και το twolf. Αυτά τα τρία προγράμματα είναι, αναμενόμενα, μεταξύ αυτών με την μεγαλύτερη αύξηση του χρόνου εκτέλεσης, με το χειρότερο από τα τρία, το applu, να αυξάνει και το EDP.

Τα άλλα δύο προγράμματα που αυξάνουν το EDP, το art στα 2MB και το ammp στο 1MB, έχουν όμως σχετικά υψηλά επίπεδα σωστών προβλέψεων. Πιο συγκεκριμένα, το art έχει 84% σωστές προβλέψεις στην συγκεκριμένη κρυφή μνήμη, ενώ το ammp 81% σωστές προβλέψεις και 6.3% συντηρητικές. Το πρόβλημα σε αυτές τις δύο περιπτώσεις είναι ότι στα συγκεκριμένα μεγέθη κρυφών μνημών, το art και το ammp αρχίζουν να χωράνε στην κρυφή μνήμη με αποτέλεσμα τα superpaths να εμφανίζουν ριζικά διαφορετικές συμπεριφορές από εκτέλεση σε εκτέλεση σε ότι αφορά το MLP-distance: στις περισσότερες εκτελέσεις, δεν εμφανίζονται αστοχίες ή εμφανίζεται μόνο μία μεμονωμένη αστοχία, οπότε το MLP-distance προβλέπεται μηδέν, αλλά σποραδικά εμφανίζονται πολλαπλές αστοχίες με μεγάλο MLP-distance. Έτσι, ο μηχανισμός μας μειώνει το μέγεθος πολύ πιο επιθετικά από ότι θα έπρεπε με αποτέλεσμα την δυσανάλογα μεγάλη αύξηση του χρόνου εκτέλεσης.

Παρόλα αυτά, είδαμε ότι στις περισσότερες περιπτώσεις, ο μηχανισμός μας καταφέρνει και μειώνει σημαντικά το EDP και την κατανάλωση ενέργειας του επεξεργαστή. Συνολικά, η μείωση του μέσου EDP κυμαίνεται ανάλογα με την κρυφή μνήμη από 8.5% έως 9.3%, ενώ για τον ILP-feedback μηχανισμό η μείωση του EDP κυμαίνεται από 1.4% έως 3.4%. Αντίστοιχα, η μέση μείωση της κατανάλωσης ενέργειας που πετυχαίνουμε κυμαίνεται από 15% έως 17.8%, ενώ για τον ILP μηχανισμό είναι από 7.7% έως 8.7%. Τέλος, αξίζει να σημειώσουμε ότι ακόμη και με πολύ πιο επιθετικό μηχανισμό διαχείρισης, η μεγαλύτερη αύξηση του χρόνου εκτέλεσης που παρουσιάστηκε στις εξομοιώσεις μας είναι 24.5%, η οποία είναι πολύ μικρότερη από ότι την 69.4% αύξηση που προκαλεί ο κατά τα άλλα συντηρητικός μηχανισμός στο mcf.

5.7 Ανακεφαλαίωση

Σε αυτό το κεφάλαιο παρουσιάστηκε μια μεθοδολογία για την μείωση της κατανάλωσης ενέργειας των σύγχρονων επεξεργαστών υψηλής απόδοσης με εκτέλεση εκτός σειράς, μέσω της δυναμικής ρύθμισης του μεγέθους της ουράς εντολών. Οι υπάρχουσες μεθοδολογίες διαχείρισης της ουράς εντολών αποφασίζουν αν θα μεταβάλλουν το μέγεθος της ουράς με βάση χαρακτηριστικά του προγράμματος που σχετίζονται άμεσα ή έμμεσα με το ILP. Στα πλαίσια της μεθοδολογίας μας, δείξαμε ότι η λήψη αποφάσεων με βάση μόνο το ILP, σε πολλές περιπτώσεις οδηγεί σε σημαντική υποβάθμιση της απόδοσης του επεξεργαστή, ενώ ταυτόχρονα τα οφέλη από την μείωση της κατανάλωσης ενέργειας είναι περιορισμένα. Το συνολικό αποτέλεσμα είναι ότι, σε αυτές τις περιπτώσεις, το συνολικό EDP του επεξεργαστή αυξάνεται σε μη αποδεκτά επίπεδα.

Επίσης, δείξαμε ότι για αυτήν την συμπεριφορά ευθύνεται το MLP. Ακόμη και μικρές μειώσεις του μεγέθους της ουράς εντολών μπορούν να μειώσουν σημαντικά το MLP, με δυσανάλογα αρνητικές συνέπειες στην ταχύτητα επεξεργασίας. Βασισμένοι σε αυτή την κατανόηση, υλοποιήσαμε μία τεχνική, η οποία ανιχνεύει την ύπαρξη παραλληλισμού μεταξύ των αστοχιών, ποσοτικοποιεί τα χαρακτηριστικά του MLP όταν αυτό υπάρχει, και χρησιμοποιεί αυτή τη πληροφορία για να οδηγήσει έναν μηχανισμό πρόβλεψης του ελάχιστου μεγέθους της ουράς εντολών που δεν μειώνει το MLP.

Τέλος, η μεθοδολογία μας προτείνει έναν μηχανισμό αποφάσεων για την μεταβολή του μεγέθους της ουράς εντολών, ο οποίος χρησιμοποιεί πληροφορία και για το μέγεθος της ουράς που δεν μειώνει το ILP και για το μέγεθος που δεν μειώνει το MLP, ώστε η τελική

απόφασή του να διατηρεί και τις δύο μορφές παραλληλισμού. Ο μηχανισμός διαχείρισης αποτιμήθηκε μέσω του εξομοιωτή HotLeakage και των πιο απαιτητικών σε μνήμη προγραμμάτων της σουίτας SPEC2000. Τα αποτελέσματα απέδειξαν ότι, η ενσωμάτωση πληροφορίας για το MLP στο βρόχο αποφάσεων διαχείρισης της ουράς εντολών, προστατεύει πιο αποτελεσματικά την απόδοση του επεξεργαστή και επιτρέπει την πολύ πιο επιθετική μείωση του μεγέθους της ουράς, όταν δεν υπάρχει MLP, οδηγώντας έτσι σε σημαντικές μειώσεις και στην κατανάλωση ενέργειας και στο συνολικό EDP του επεξεργαστή.

Κεφάλαιο 6

Συμπεράσματα και Προεκτάσεις

6.1 Εισαγωγή

Στην παρούσα διατριβή μελετήθηκαν μέθοδοι διαχείρισης κοινόχρηστων πόρων σε πολυεπεξεργαστικά συστήματα ενός ολοκληρωμένου. Αρχικά, αναλύθηκαν τα χαρακτηριστικά των υπολογιστικών συστημάτων με πολλαπλούς επεξεργαστές σε ένα ολοκληρωμένο και μελετήθηκαν τα προβλήματα που σχετίζονται με τους κοινόχρηστους πόρους και την διαχείρισή τους. Με βάση αυτή την κατανόηση, προτάθηκαν νέοι μηχανισμοί και μεθοδολογίες διαχείρισης. Η έμφαση δόθηκε στο υποσύστημα μνήμης και στην κατανάλωση ισχύος, λόγω της κεντρικότητάς τους στις προσπάθειες των αρχιτεκτόνων υπολογιστών για σχεδιασμό πιο αποδοτικών υπολογιστικών συστημάτων. Για κάθε μεθοδολογία αναλύθηκε ο τρόπος με τον οποίο παρεμβαίνει στην λειτουργία του επεξεργαστή και των δομών του, συζητήθηκαν οι εναλλακτικοί τρόποι υλοποίησης και τα πλεονεκτήματα που συσχετίζονται με καθέναν από αυτούς. Τέλος, όλες οι μεθοδολογίες αποτιμήθηκαν πειραματικά, για να εξακριβωθεί ο βαθμός στον οποίο επιτυγχάνουν να διαχειριστούν αποδοτικά τις δομές του επεξεργαστή.

6.2 Συνεισφορές της Διδακτορικής Διατριβής

Τα κυριότερα επιστημονικά αποτελέσματα της παρούσας διδακτορικής διατριβής, συνίστανται στα ακόλουθα:

- Μοντελοποίηση του διαμοιρασμού της κοινόχρηστης κρυφής μνήμης σε πολυεπεξεργαστικά συστήματα. Η προτεινόμενη μοντελοποίηση επιτυγχάνει να περιγράψει με ακρίβεια τον τρόπο με τον οποίο διαμοιράζονται την κρυφή μνήμη τα προγράμματα που εκτελούνται στο κάθε επεξεργαστή του ολοκληρωμένου και να δώσει βαθύτερη κατανόηση των αποτελεσμάτων του διαμοιρασμού στην κάθε εφαρμογή. Επιπλέον, επιτυγχάνει αυτούς τους στόχους με πληροφορία που μπορεί να συλλεχθεί εύκολα κατά την διάρκεια εκτέλεσης των εφαρμογών και με απλές εξισώσεις που μπορούν να επιλυθούν χωρίς ιδιαίτερο κόστος από τον χρονοπρογραμματιστή του λειτουργικού συστήματος.
- Ανάπτυξη μεθοδολογίας διαχείρισης του διαμοιρασμού της κοινόχρηστης μνήμης, με στόχο την αύξηση της δικαιοσύνης και της απόδοσης ή με στόχο την παροχή εγγυήσεων ποιότητας υπηρεσίας. Η μεθοδολογία στοχεύει να διαμοιράσει με δίκαιο και αποδοτικό τρόπο την κοινόχρηστη κρυφή μνήμη, χρησιμοποιώντας την πληροφορία που προέκυψε από την μοντελοποίηση της κρυφής μνήμης.
- Ανάπτυξη μίας νέας πολιτικής αντικατάστασης για κρυφές μνήμες στο τελευταίο επίπεδο της ιεραρχίας κρυφών μνημών. Η προτεινόμενη πολιτική προσομοιώνει στις αποφάσεις της την θεωρητικά βέλτιστη πολιτική αντικατάστασης, αξιοποιώντας την πληροφορία που παράγει ένας καινοτόμος μηχανισμός πρόβλεψης της τοπικότητας των δεδομένων της κρυφής μνήμης
- Σύνδεση του παραλληλισμού μεταξύ των προσπελάσεων της κύριας μνήμης, με την αποδοτικότητα της διαχείρισης της ουράς εντολών. Η συγκεκριμένη σύνδεση αξιοποιήθηκε για την ανάπτυξη ενός μηχανισμού που μειώνει σημαντικά την κατανάλωση ενέργειας του επεξεργαστή, απενεργοποιώντας δυναμικά τμήματα της ουράς εντολών, με τέτοιο τρόπο ώστε να μην μειώνεται ο παραλληλισμός των προσπελάσεων της κύριας μνήμης και να διατηρείται η απόδοση του επεξεργαστή σε υψηλά επίπεδα.

6.3 Μελλοντικές Κατευθύνσεις και Προεκτάσεις

Όλες οι μεθοδολογίες που προτάθηκαν στα πλαίσια της παρούσας διατριβής έχουν σαν κοινό χαρακτηριστικό, το ότι προτείνουν αποδοτικές τεχνικές συλλογής βασικών πληροφοριών σχετικά με την συμπεριφορά των εκτελούμενων προγραμμάτων και ότι δείχνουν πως συνδέεται αυτή η πληροφορία με την διαχείριση του επεξεργαστή και των

κρυφών μνημών. Αυτή η πληροφορία μπορεί εύκολα να αξιοποιηθεί για πολλαπλούς άλλους στόχους διαχείρισης, εκτός από όσους περιγράφηκαν σε αυτή την διατριβή. Ενδεικτικά:

- *Αξιοποίηση της πρόβλεψης των αποστάσεων επαναχρησιμοποίησης και της StatShare μοντελοποίησης για σχεδόν βέλτιστο διαμοιρασμό της κρυφής μνήμης.* Η StatShare επιτυγχάνει να περιγράψει με ακρίβεια το πως διαμοιράζεται η κρυφή μνήμη και το πως αξιοποιεί η κάθε εφαρμογή τον χώρο της. Ενσωματώνοντας στην StatShare την πρόβλεψη των αποστάσεων επαναχρησιμοποίησης, μπορούμε να προβλέψουμε ακριβώς ποια δεδομένα δεν χωράνε στο μερίδιο της κρυφής μνήμης κάθε εφαρμογής και να τα προσφέρουμε για αντικατάσταση άμεσα, αντί να περιμένουμε να τα διαλέξει το CAT Decay.
- *Επέκταση του μηχανισμού πρόβλεψης αποστάσεων επαναχρησιμοποίησης, με πρόβλεψη και του MLP-cost.* Αν μπορούμε να προβλέψουμε και τις αποστάσεις επαναχρησιμοποίησης και το MLP-cost των δεδομένων της κρυφής μνήμης, μπορούμε να υλοποιήσουμε μία πολιτική αντικατάστασης που να τείνει στο βέλτιστο, και σε ότι αφορά την αξιοποίηση της τοπικότητας και σε ότι αφορά την αξιοποίηση του παραλληλισμού των προσπελάσεων κύριας μνήμης. Μία τέτοια πολιτική θα μπορούσε να πετύχει καλύτερα αποτελέσματα ακόμη και από την θεωρητικά βέλτιστη πολιτική.
- *Χρήση της πρόβλεψης επαναχρησιμοποίησης για άλλες τεχνικές διαχείρισης που εκμεταλλεύονται την τοπικότητα των προσπελάσεων.* Με την ίδια πληροφορία που χρησιμοποιούμε για να διαλέξουμε την βέλτιστη γραμμή προς αντικατάσταση, μπορούμε να διαλέξουμε την βέλτιστη γραμμή για συμπίεση ή για απενεργοποίηση με decay.
- *Ενοποίηση της διαχείρισης της κρυφής μνήμης, της διαχείρισης της ουράς εντολών και της δυναμικής μεταβολής της συχνότητας λειτουργίας του επεξεργαστή.* Δείξαμε ήδη σε αυτή την διατριβή ότι η διατήρηση του MLP είναι από τους πιο κρίσιμους στόχους και της διαχείρισης της κρυφής μνήμης και της ουράς εντολών. Επιπλέον, άλλες εργασίες έχουν δείξει ότι το ποσοστό των αστοχιών της κρυφής μνήμης και ο παραλληλισμός αυτών των αστοχιών, είναι από τις πιο κρίσιμες παραμέτρους που καθορίζουν αν συμφέρει ή όχι η μείωση της συχνότητας λειτουργίας του επεξεργαστή. Ένας ενοποιημένος μηχανισμός διαχείρισης θα μπορούσε να εκμεταλλευτεί αυτή την

κάπως απρόσμενη αλληλεξάρτηση, ώστε να βελτιώσει όλους τους επιμέρους μηχανισμούς διαχείρισης. Για παράδειγμα, αν υπάρχει σημαντικός παραλληλισμός στις αστοχίες, ο υποθετικός αυτός μηχανισμός θα μπορούσε να αυξήσει το μέγεθος της ουράς εντολών, αυξάνοντας έτσι ακόμη περισσότερο τον παραλληλισμό των αστοχιών. Ταυτόχρονα, εφόσον το πρόγραμμα βρίσκεται σε φάση με αυξημένες αστοχίες, ο μηχανισμός θα μείωνε, με ελάχιστο κόστος, την συχνότητα λειτουργίας, και παράλληλα θα μπορούσε να απενεργοποιήσει τμήματα της κρυφής μνήμης, αφού οι αυξημένες αστοχίες θα παραλληλιστούν και θα έχουν μικρή επίδραση στην απόδοση του επεξεργαστή. Έτσι το κέρδος σε ενέργεια από την μείωση της συχνότητας και την απενεργοποίηση μέρους της κρυφής μνήμης, θα υπερκάλυπτε την αύξηση της κατανάλωσης της ουράς, δίνοντάς μας αθροιστικά μείωση της κατανάλωσης ενέργειας, ενώ η απόδοση του επεξεργαστή θα έμενε σχεδόν σταθερή. Γενικότερα, λόγω του MLP, οι τρεις αυτοί μηχανισμοί είναι στενά αλληλεξαρτημένοι και η διαχείρισή τους πρέπει να είναι συνολική.

6.4 Αποτελέσματα Ερευνητικής Δραστηριότητας

Κατά τη διάρκεια της εκπόνησης της παρούσας διδακτορικής διατριβής, οι μεθοδολογίες που προτάθηκαν, τα πειραματικά αποτελέσματα καθώς και οτιδήποτε νέο και καινοτόμο αναπτύχθηκε, υποβλήθηκε, εγκρίθηκε, παρουσιάστηκε και δημοσιεύθηκε σε διεθνή συνέδρια κύρους. Έτσι, κατοχυρώθηκαν οι προτάσεις που έγιναν, ενώ ταυτόχρονα διασφαλίστηκε η επιστημονική ορθότητα των προτεινόμενων μεθοδολογιών. Οι συνολικές δημοσιεύσεις που έγιναν στα πλαίσια της διδακτορικής διατριβής παρουσιάζονται παρακάτω:

1. Pavlos Petoumenos, Georgios Keramidas, Håkan Zeffer, Stefanos Kaxiras, and Erik Hagersten. Statshare: A statistical model for managing cache sharing via decay. Proceedings of the Second Annual Workshop on Modeling, Benchmarking and Simulation (MoBS 2006), 2006.
2. Georgios Keramidas, Pavlos Petoumenos, Stefanos Kaxiras, Alexandros Antonopoulos, and Dimitrios Serpanos. Preventing Denial-of-Service Attacks in Shared CMP Caches. Proceedings of the 6th International Workshop on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS 2006), 2006

3. Pavlos Petoumenos, Georgios Keramidas, Håkan Zeffler, Stefanos Kaxiras, and Erik Hagersten. StatShare: Modeling and Managing Shared Caches. Proceedings of the Poster Session of the 2nd International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES-2006), 2006.
4. Georgios Keramidas, Pavlos Petoumenos, Stefanos Kaxiras, Alexandros Antonopoulos, and Dimitrios Serpanos. Using Value Locality to Reduce Memory Encryption Overhead in Embedded Processors. Presented in the ACM First Workshop in Embedded Computing (WESS-1), 2006.
5. Pavlos Petoumenos, Georgios Keramidas, Håkan Zeffler, Stefanos Kaxiras, and Erik Hagersten. Modeling Cache Sharing on Chip Multiprocessor Architectures. Proceedings of the 2006 IEEE International Symposium on Workload Characterization (IISWC-2006), 2006.
6. Georgios Keramidas, Pavlos Petoumenos, and Stefanos Kaxiras. Cache replacement based on reuse-distance prediction. Proceedings of the XXV IEEE International Conference on Computer Design (ICCD-2007), 2007.
7. Georgios Keramidas, Pavlos Petoumenos, Stefanos Kaxiras, Alexandros Antonopoulos, and Dimitrios Serpanos. Using Value Locality to Reduce Memory Encryption Overhead in Embedded Processors. Proceedings of the IEEE/ACM Conference on Emerging Technologies and Factory Automation (ETFA 2007), 2007.
8. Juan Manuel Cebrian Gonzalez, Juan Luis Aragon, Jose M. Garcia, Pavlos Petoumenos, and Stefanos Kaxiras. Efficient microarchitecture policies for accurately adapting to power constraints. Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, (IPDPS-2009), 2009.
9. Georgia Psychou, Pavlos Petoumenos, Stefanos Kaxiras, Juan Manuel Cebrian Gonzalez, and Juan Luis Aragon. MLP-Aware Instruction Queue Resizing: The Key to Power-Efficient Performance. Proceedings of the Poster Session of the 5th International Summer School on Advanced Computer Architecture and Compilation for Embedded Systems (ACACES-2009), 2009.
10. Pavlos Petoumenos, Georgios Keramidas, and Stefanos Kaxiras. Instruction-based reuse-distance prediction for effective cache management. Proceedings of the 9th

International conference on Systems, architectures, modeling and simulation (SAMOS'09), 2009.

11. Pavlos Petoumenos, Georgia Psychou, Stefanos Kaxiras, Juan Manuel Cebrian Gonzalez, and Juan Luis Aragon. MLP-Aware Instruction Queue Resizing: The Key to Power-Efficient Performance. 2010 International Conference on Architecture of Computing Systems (ARCS-2010), 2010.
12. Georgios Keramidas, Pavlos Petoumenos, and Stefanos Kaxiras. Where replacement algorithms fail: a thorough analysis. Proceedings of the 2010 ACM International Conference on Computing Frontiers (CF'10), 2010.
13. Pavlos Petoumenos, Georgios Keramidas, and Stefanos Kaxiras. Instruction-based Reuse Distance Prediction Replacement Policy. Proceedings of the 1st JILP Workshop on Computer Architecture Competitions (JWAC-1), 2010.

6.5 Αναφορές των Δημοσιευμένων Εργασιών από άλλες Ερευνητικές Ομάδες

Ορισμένες από τις παραπάνω ερευνητικές δημοσιεύσεις έχουν προταθεί ως αναφορές σε εργασίες άλλων ερευνητών που έχουν δημοσιευθεί σε περιοδικά ή συνέδρια και αναφέρονται παρακάτω:

Δημοσίευση 1

1. Jichuan Chang and Gurindar S. Sohi. Cooperative Caching for Chip Multiprocessors. In Proceedings of the 33rd annual international symposium on Computer Architecture (ISCA '06), 2006.
2. Jichuan Chang and Gurindar S. Sohi. Cooperative cache partitioning for chip multiprocessors. In Proceedings of the 21st annual international conference on Supercomputing (ICS '07), 2007.
3. Feiqi Su, Xudong Shi, Gang Liu, Ye Xia, and Jih-Kwon Peir. Comparative evaluation of multi-core cache occupancy strategies. In Proceedings of the 2007 International Conference on Parallel and Distributed Systems (ICPADS-2007), 2007.

4. Nian-Feng Tzeng, Hongyi Wu, and Gui-Liang Feng. ADENS: Efficient address determination for mobile grids. In Proceedings of the 2007 International Conference on Parallel and Distributed Systems (ICPADS-2007), 2007.
5. Carole-Jean Wu and Margaret Martonosi. A Comparison of Capacity Management Schemes for Shared CMP Caches. In Proceedings of the 7th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD) in conjunction with ISCA'08, 2008.
6. Shuo Li, Gao Chao Xu, Yu Shuang Dong, and Feng Wu. The Review of Cache Partitioning in Multi-Core Processor. pp 1223-1229, Key Engineering Materials, vol. 439-440, 2010.
7. Yan Pei-Xiang, Jiang Jiang, Yang Xian-Ju, and Zhang Min-Xuan. Partitioning mechanism based on dynamic Allocation of Data entries for chip multiprocessors. In Proceedings of the 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), 2010.

Δημοσίευση 5

8. Jie Tao, Kim D. Hoang, and Wolfgang Karl. CMP Cache Architecture and the OpenMP Performance. In Proceedings of the Third International Workshop on OpenMP (IWOMP 2007), 2007.
9. Shekhar Srikantaiah, Mahmut Kandemir, and Mary Jane Irwin. Adaptive set pinning: managing shared caches in chip multiprocessors. In Proceedings of the 13th international conference on Architectural support for programming languages and operating systems (ASPLOS XIII), 2008.
10. Livio Soares, David Tam, and Michael Stumm. Reducing the harmful effects of last-level cache polluters with an OS-level, software-only pollute buffer. In Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture (MICRO 41), 2008.
11. Carole-Jean Wu and Margaret Martonosi. A Comparison of Capacity Management Schemes for Shared CMP Caches. In Proceedings of the 7th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD) in conjunction with ISCA'08, 2008.

12. John M. Calandrino and James H. Anderson. On the Design and Implementation of a Cache-Aware Multicore Real-Time Scheduler. In Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems (ECRTS '09), 2009.
13. Miquel Moreto, Francisco J. Cazorla, Alex Ramirez, and Mateo Valero. Dynamic Cache Partitioning Based on the MLP of Cache Misses. Transactions On High-Performance Embedded Architectures and Compilers III, 2011.
14. Carole-Jean Wu and Margaret Martonosi. Adaptive timekeeping replacement: Fine-grained capacity management for shared CMP caches. ACM Transactions on Architecture and Code Optimization (TACO), Volume 8, Issue 1, February 2011.

Δημοσίευση 6

15. Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. High performance cache replacement using re-reference interval prediction (RRIP). In Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10), 2010.
16. Samira M. Khan, Daniel A. Jiménez, Doug Burger, and Babak Falsafi. Using dead blocks as a virtual victim cache. In Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT '10), 2010.
17. Qiushi Li, Qiuyue Wang, and Shan Wang. Query-Aware Complex Object Buffer Management in XML Information Retrieval. In Proceedings of the 12th International Asia-Pacific Web Conference (APWEB'10), 2010.
18. Samira M. Khan and Daniel A. Jimenez. Insertion policy selection using Decision Tree Analysis. In Proceedings of the IEEE International Conference on Computer Design (ICCD-2010), 2010.
19. R. Manikantan, K. Rajan, and R. Govindarajan. NUcache: An efficient multicore cache organization based on Next-Use distance. In Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture (HPCA'11), 2011.

Δημοσίευση 10

20. Andreas Sandberg, David Eklöv, and Erik Hagersten. Reducing Cache Pollution Through Detection and Elimination of Non-Temporal Memory Accesses,” In Proceed-

- ings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '10), 2010.
- 21.R. Manikantan, K. Rajan, and R. Govindarajan. NUcache: An efficient multicore cache organization based on Next-Use distance. In Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture (HPCA'11), 2011.
- 22.Silvius Rus, Raksit Ashok, and David Xinliang Li. Automated locality optimization based on the reuse distance of string operations. In Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO), 2011.

Βιβλιογραφία

- [1] Jaume Abella, Antonio González, Xavier Vera, and Michael F. P. O'Boyle. IATAC: a smart predictor to turn-off L2 cache lines. *ACM Transactions on Architecture and Code Optimization*, 2, 1 (March 2005), 55-77, 2005.
- [2] David H. Albonesi. Dynamic IPC/clock rate optimization. In *Proceedings of the 25th annual international symposium on Computer architecture (ISCA '98)*, 1998.
- [3] Hari Ananthan, Chris H. Kim, and Kaushik Roy. Larger-than-vdd forward body bias in sub-0.5V nanoscale CMOS. In *Proceedings of the 2004 international symposium on Low power electronics and design (ISLPED '04)*, 2004.
- [4] Vikas Agarwal, M.S. Hrishikesh, Stephen W. Keckler, and Doug Burger. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA '00)*, 2000.
- [5] ARM Ltd. Cortex-A9 Processor <http://www.arm.com/products/processors/cortex-a/cortex-a9.php>
- [6] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report Technical Report No. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, 2006.
- [7] Laszlo A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal* 5, 2 (June 1966), 78-101, 1966.
- [8] Erik Berg and Erik Hagersten. StatCache: a probabilistic approach to efficient and accurate data locality analysis. In *Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '04)*, 2004.
- [9] Erik Berg and Erik Hagersten. Fast Data-Locality Profiling of Native Execution. In *Proceedings of ACM SIGMETRICS 2005*, 2005.

-
- [10] Erik Berg, Håkan Zeffler, and Erik Hagersten. A Statistical Multiprocessor Cache Model. In Proceedings of the 2006 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2006), 2006.
 - [11] Shekhar Y. Borkar, Pradeep Dubey, Kevin C. Kahn, David J. Kuck, Hans Mulder, Stephen S. Pawlowski, and Justin R. Rattner. Platform 2015: Intel Processor and Platform Evolution for the Next Decade. Intel White Paper, 2005.
 - [12] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In Proceedings of the 27th annual international symposium on Computer architecture (ISCA '00), 2000.
 - [13] Alper Buyuktosunoglu, David Albonesi, Stanley Schuster, David Brooks, Pradip Bose, and Peter Cook. A circuit level implementation of an adaptive issue queue for power-aware microprocessors. In Proceedings of the 11th Great Lakes symposium on VLSI (GLSVLSI '01), 2001.
 - [14] Ramon Canal and Antonio González. A low-complexity issue logic. In Proceedings of the 14th international conference on Supercomputing (ICS '00), 2000.
 - [15] Ramon Canal and Antonio González. Reducing the complexity of the issue logic. In Proceedings of the 15th international conference on Supercomputing (ICS '01), 2001.
 - [16] Calin Cașcaval and David A. Padua. Estimating cache misses and locality using stack distances. In Proceedings of the 17th annual international conference on Supercomputing (ICS '03), 2003.
 - [17] Lei Chai, Qi Gao, and Dhabaleswar K. Panda. Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System. In Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGRID '07). 2007.
 - [18] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture. In Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA'05), 2005.
 - [19] Chi-Hung Chi and Henry Dietz. Improving cache performance by selective cache bypass. In Proceedings of the 22th Annual Hawaii International Conference on System Sciences, 1989.

- [20] Yuan Chou, Brian Fahs, and Santosh Abraham. Microarchitecture Optimizations for Exploiting Memory-Level Parallelism. In Proceedings of the 31st annual international symposium on Computer architecture (ISCA '04), 2004.
- [21] Chen Ding and Yutao Zhong. Predicting whole-program locality through reuse distance analysis. In Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation (PLDI '03), 2003.
- [22] David Eklov and Erik Hagersten. StatStack: Efficient Modeling of LRU Caches. In Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '10), 2010.
- [23] Stijn Eyerman and Lieven Eeckhout. A Memory-Level Parallelism Aware Fetch Policy for SMT Processors. In Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture (HPCA '07), 2007.
- [24] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. A mechanistic performance model for superscalar out-of-order processors. ACM Transactions on Computer Systems, 27, 2, Article 3 (May 2009), 2009.
- [25] Changpeng Fang, Steve Carr, Soner Soner Önder, and Zhenlin Wang. Reuse-distance-based miss-rate prediction on a per instruction basis. In Proceedings of the 2004 workshop on Memory system performance (MSP '04), 2004.
- [26] Changpeng Fang, Steve Carr, Soner Önder, and Zhenlin Wang. Instruction Based Memory Distance Analysis and its Application. In Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT '05), 2005.
- [27] Daniele Folegnani and Antonio González. Energy-effective issue logic. In Proceedings of the 28th annual international symposium on Computer architecture (ISCA '01), 2001.
- [28] Hongliang Gao and Chris Wilkerson. A Dueling Segmented LRU Replacement Algorithm with Adaptive Bypassing. 1st JILP Worshop on Computer Architecture Competitions: Cache Replacement Championship (JWAC-2010), 2010.

-
- [29] Kanad Ghose and Milind B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In Proceedings of the 1999 international symposium on Low power electronics and design (ISLPED '99), 1999 .
 - [30] Andrew Glew. MLP yes! ILP no!. In ASPLOS Wild and Crazy Ideas Session '98, 1998.
 - [31] Ronaldo Goncalves, Eduard Ayguade, Mateo Valero, and Philippe Navaux. A Simulator for SMT Architectures: Evaluating Instruction Cache Topologies. In Proceedings of the 12th Symposium on Computer Architecture and High Performance (SBAC-PAD), 2000.
 - [32] John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach. 3rd Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
 - [33] Zhigang Hu, Stefanos Kaxiras, and Margaret Martonosi. Timekeeping in the memory system: predicting and optimizing memory behavior. In Proceedings of the 29th International Sympocium on Computer Architecture (ISCA '02), 2002.
 - [34] Intel Corp. Intel Atom Processor. <http://www.intel.com/products/processor/atom/index.htm#specifications>
 - [35] Anoop Iyer and Diana Marculescu. Power aware microarchitecture resource scaling. In Proceedings of the conference on Design, automation and test in Europe (DATE '01), 2001.
 - [36] Jaeheon Jeong and Michel Dubois. Optimal replacements in caches with two miss costs. In Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures (SPAA '99), 1999.
 - [37] Jaeheon Jeong and Michel Dubois. Cost-sensitive cache replacement algorithms. In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA '03), 2003.
 - [38] Jaeheon Jeong, Per Stenström, and Michel Dubois. Simple penalty-sensitive replacement policies for caches. In Proceedings of the 3rd conference on Computing frontiers (CF '06), 2006.

- [39] Teresa L. Johnson and Wen-mei W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In Proceedings of the 24th annual international symposium on Computer architecture (ISCA '97), 1997.
- [40] Teresa L. Johnson, Daniel A. Connors, Matthew C. Merten, and Wen-mei W. Hwu. Run-Time Cache Bypassing. IEEE Transactions on Computers. 48, 12 (December 1999), 1338-1354, 1999.
- [41] The Journal of Instruction-Level Parallelism, 2010. Cache Replacement Championship. <http://www.jilp.org/jwac-1/>
- [42] The Journal of Instruction-Level Parallelism, 2010. Cache Replacement Championship. http://www.jilp.org/jwac-1/online/slides/000_intro.ppt
- [43] Martin Kampe and Fredrik Dahlgren. Exploration of the Spatial Locality on Emerging Applications and the Consequences for Cache Performance. In Proceedings of the International Parallel and Distributed Computing Symposium (IPDPS-2000), 2000.
- [44] Tejas S. Karkhanis and James E. Smith. A First-Order Superscalar Processor Model. In Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA '04), 2004.
- [45] Martin Karlsson and Erik Hagersten. Timestamp-Based Selective Cache Allocation. In High Performance Memory Systems, Springer-Verlag, 2003.
- [46] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power. In Proceedings of the 28th International Sympocium on Computer Architecture (ISCA '01), 2001.
- [47] Stefanos Kaxiras and Polychronis Xekalakis. 4T-decay sensors: a new class of small, fast, robust, and low-power, temperature/leakage sensors. In Proceedings of the 2004 international symposium on Low power electronics and design (ISLPED '04), 2004.
- [48] Stefanos Kaxiras, Polychronis Xekalakis, and Georgios Keramidas. A simple mechanism to adapt leakage-control policies to temperature. In Proceedings of the 2005 international symposium on Low power electronics and design (ISLPED '05), 2005.
- [49] Stefanos Kaxiras and Margaret Martonosi. Computer Architecture Techniques for Power-Efficiency (1st ed.). Morgan and Claypool Publishers, 2008.

-
- [50] Georgios Keramidas, Polychronis Xekalakis, and Stefanos Kaxiras. Recruiting Decay for Dynamic Power Reduction in Set-Associative Caches. In *Transactions on High-Performance Embedded Architectures and Compilers II*, Lecture Notes In Computer Science, Vol. 5470. Springer-Verlag, 2009.
 - [51] Mazen Kharbutli and Yan Solihin. Counter-Based Cache Replacement Algorithms. In *Proceedings of the 2005 International Conference on Computer Design (ICCD '05)*, 2005.
 - [52] Seongbeom Kim, Dhruba Chandra, and Yan Solihin. Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques (PACT '04)*, 2004.
 - [53] P. Kongetira, K. Aingaran, and K. Olukotun, Niagara: A 32-Way Multithreaded Sparc Processor. *IEEE Micro*, vol. 25, no. 2, Mar/Apr, 2005.
 - [54] K. Krewell. Power5 Tops on Bandwidth. In *Microprocessor Report*, 12/22/03-02, 2003.
 - [55] Gurhan Kucuk, Kanad Ghose, Dmitry V. Ponomarev, and Peter M. Kogge. Energy: efficient instruction dispatch buffer design for superscalar processors. In *Proceedings of the 2001 international symposium on Low power electronics and design (ISLPED '01)*, 2001.
 - [56] An-Chow Lai and Babak Falsafi. Selective, accurate, and timely self-validation using last-touch prediction. In *Proceedings of the 27th annual international symposium on Computer architecture (ISCA '00)*, 2000.
 - [57] Wei-Fen Lin and Steven K. Reinhardt. Predicting last-touch references under optimal replacement. In *Technical Report CSE-TR-447-02*, University of Michigan, 2002.
 - [58] Chun Liu, Anand Sivasubramaniam, and Mahmut Kandemir. Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs. In *Proceedings of the 10th International Symposium on High Performance Computer Architecture (HPCA'04)*, 2004.

- [59] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation (PLDI '05), 2005.
- [60] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger. Evaluation techniques for storage hierarchies. IBM Systems Journal 9, 2 (June 1970), 78-117, 1970.
- [61] Pierre Michaud and André Seznec. Data-Flow Prescheduling for Large Instruction Windows in Out-of-Order Processors. In Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA '01), 2001.
- [62] V. Milutinovic, B. Markovic, M. Tomasevic, and M. Tremblay. The Split Temporal/Spatial Cache: Initial Performance analysis. Journal of Systems Architecture: the EURO-MICRO Journal, 1996.
- [63] G.E. Moore. Cramming more components onto integrated circuits. Electronics: 114-117, April 1965.
- [64] G.E. Moore. Progress in digital integrated electronics. 1975 International Electron Devices Meeting, 1975.
- [65] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. LRU-K page replacement algorithm for database disk buffering. In Proceedings of the Conference on Management of Data, 1993.
- [66] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. An Optimality Proof of the LRU-K Page Replacement Algorithm. Journal of the ACM, 1999.
- [67] Soner Önder. Cost Effective Memory Dependence Prediction using Speculation Levels and Color Sets. In Proceedings of the 2002 International Conference on Parallel Architectures and Compilation Techniques (PACT '02), 2002.
- [68] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In Proceedings of the 24th annual international symposium on Computer architecture (ISCA '97), 1997.

-
- [69] Thomas Piquet, Olivier Roche, and André Seznec. Exploiting Single-Usage for Effective Memory Management. In Proceedings of the 12th Asia-Pacific conference on Advances in Computer Systems Architecture (ACSAC '07), 2007.
 - [70] Dmitry Ponomarev, Gurhan Kucuk, and Kanad Ghose. Dynamic Resizing of Super-scalar Datapath Components for Energy Efficiency. *IEEE Transactions on Computers* 55, 2 (February 2006), 199-213, 2006.
 - [71] Moinuddin K. Qureshi, David Thompson, and Yale N. Patt. The V-Way Cache: Demand Based Associativity via Global Replacement. In Proceedings of the 32nd annual international symposium on Computer Architecture (ISCA '07), 2005.
 - [72] Moinuddin K. Qureshi, Daniel N. Lynch, Onur Mutlu, and Yale N. Patt. A Case for MLP-Aware Cache Replacement. In Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA '06), 2006.
 - [73] Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. Adaptive insertion policies for high performance caching. In Proceedings of the 34th annual international symposium on Computer architecture (ISCA '07), 2007.
 - [74] Kaushik Rajan and Govindarajan Ramaswamy. Emulating Optimal Replacement with a Shepherd Cache. In Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40), 2007.
 - [75] Eric Rotenberg, Steve Bennett, and James E. Smith. Trace cache: a low latency approach to high bandwidth instruction fetching. In Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture (MICRO 29), 1996.
 - [76] Timothy Sherwood, Erez Perelman, and Brad Calder. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. Technical Report. University of California at San Diego, La Jolla, CA, USA. 2001.
 - [77] Timothy Sherwood, S. Sair, and Brad Calder. Phase Tracking and Prediction. In Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA '03), 2003.
 - [78] Allan Snavely, Dean M. Tullsen, and Geoff Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems (SIGMETRICS '02), 2002.

- [79] Gurindar S. Sohi and Sriram Vajapeyam. Instruction issue logic for high-performance, interruptable pipelined processors. In Proceedings of the 14th annual international symposium on Computer architecture (ISCA '87), 1987.
- [80] Standard Performance Evaluation Corporation, 2006. <http://www.spec.org/cpu2006/>
- [81] Ranjith Subramanian, Yannis Smaragdakis, and Gabriel H. Loh. Adaptive Caches: Effective Shaping of Cache Behavior to Workloads. In Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 39), 2006.
- [82] Ranjith Subramanian, Yannis Smaragdakis, and Gabriel H. Loh. Adaptive Caches: Effective Shaping of Cache Behavior to Workloads. In Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 39), 2006.
- [83] Rabin A. Sugumar and Santosh G. Abraham. Efficient simulation of caches under optimal replacement with applications to miss characterization. In Proceedings of the Conference on Measurement and Modeling Computer Systems, 1993.
- [84] G. Edward Suh, Srinivas Devadas, and Larry Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA '02), 2002.
- [85] Masamichi Takagi and Kei Hiraki. Inter-reference gap distribution replacement: an improved replacement algorithm for set-associative caches. In Proceedings of the 18th annual international conference on Supercomputing (ICS '04), 2004.
- [86] David Tarjan, Shyamkumar Thoziyoor, and Norman P.. Jouppi. CACTI 4.0. Hewlett-Packard Laboratories Technical Report #HPL-2006-86, 2006
- [87] Olivier Temam. An algorithm for optimally exploiting spatial and temporal locality in upper memory levels. In IEEE Transactions on Computers, 1999.
- [88] Francis Tseng and Yale N. Patt. Achieving Out-of-Order Performance with Almost In-Order Complexity. In Proceedings of the 35th Annual International Symposium on Computer Architecture (ISCA '08), 2008.
- [89] Gary Tyson, Matthew Farrens, John Matthews, and Andrew R. Pleszkun. A modified approach to data cache management. In Proceedings of the 28th annual international symposium on Microarchitecture (MICRO 28), 1995.

-
- [90] Sivakumar Velusamy, Karthik Sankaranarayanan, Dharmesh Parikh, Tarek Abdelzaher, and Kevin Skadron. Adaptive cache decay using formal feedback control. In Proceedings of the 2nd Annual Workshop On Memory Performance Issues (WMPI-2002), 2002.
 - [91] David Tawei Wang. Modern DRAM memory systems: Performance analysis and a high performance, power-constrained DRAM scheduling algorithm. PhD Thesis, University of Maryland, 2005.
 - [92] Gerhard Weikum, Christof Hasse, Alex Moenkeberg, and Peter Zabback. The CONFORT automatic tuning project. Information Systems Journal, Volume 19 Issue 5, July 1994.
 - [93] Wm A. Wulf and Sally A. McKee. Hitting the Memory Wall: Implications of the Obvious. ACM SIGARCH Computer Architecture News, v.23 n.4, p.20-24, 1995.
 - [94] Wayne A. Wong and Jean-Loup Baer. Modified LRU Policies for Improving Second-Level Cache Behavior. In Proceedings of the Sixth International Symposium on High-Performance Computer Architecture (HPCA '00), 2000.
 - [95] Thomas Y. Yeh and Glenn Reinman. Fast and fair: data-stream quality of service. In Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems (CASES '05), 2005.
 - [96] Yutao Zhong, Steven G. Dropsho, and Chen Ding. Miss Rate Prediction across All Program Inputs. In Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques (PACT '03), 2003.
 - [97] Huiyang Zhou, Mark C. Toburen, Eric Rotenberg, and Thomas M. Conte. Adaptive mode control: A static-power-efficient cache design. ACM Transactions on Embedded Computing Systems 2, 3 (August 2003), 2003.

Παράρτημα Α

Γλωσσάριο

A.1 Αντιστοίχηση Αγγλικών - Ελληνικών όρων

Αγγλικός Όρος	Ελληνικός Όρος
access time	χρόνος πρόσβασης
benchmarks	μετροπρογράμματα
cache lines/blocks	γραμμές της κρυφής μνήμης
cache set	το σύνολο των συσχετιστικών γραμμών της κρυφής μνήμης
cache memory	κρυφή μνήμη
direct mapped cache memory	κρυφή μνήμη απλής συσχετικότητας
fully associative cache memory	κρυφή μνήμη πλήρους συσχετικότητας
hit	ευστοχία
hit ratio	ποσοστό/βαθμός ευστοχίας
instruction queue	ουρά εντολών
load/store instruction	εντολές ανάγνωσης/εγγραφής
main memory	κύρια μνήμη
memory access	αναφορά/πρόσβαση στην μνήμη
memory level parallelism	παραλληλισμός μεταξύ των προσβάσεων της κύριας μνήμης
miss	αστοχία
miss ratio	ρυθμός αστοχίας
out-of-order execution	τεχνική εκτέλεσης των εντολών εκτός σειράς
program phase	φάση ενός προγράμματος κατά την οποία δεν αλλάζει η συμπεριφορά του π.χ. σταθερός ρυθμός ευστοχίας
prediction accuracy	ποσοστό ακρίβειας των προβλέψεων
prediction coverage	συνολικός αριθμός των προβλέψεων προς τον συνολικό αριθμό των προσβάσεων στη μνήμη
prefetching	προφόρτωση/εκ των προτέρων τοποθέτηση δεδομένων στην κρυφή μνήμη

Αγγλικός Όρος	Ελληνικός Όρος
replacement algorithm	αλγόριθμος αντικατάστασης
reuse distance	απόσταση επαναχρησιμοποίησης
reuse distance predictors	δομές πρόβλεψης της απόστασης επαναχρησιμοποίησης
run-time	χρόνος εκτέλεσης
set-associative cache memory	μερικώς συσχετιστική κρυφή μνήμη
shared cache	κοινόχρονη κρυφή μνήμη
spatial locality	χωρική τοπικότητα
temporal locality	χρονική τοπικότητα

A.2 Επεξηγήσεις Συντομεύσεων-Ακρωνυμιών

Όρος	Επεξήγηση
CAM	Context Addressable Memory
CMP	Chip Multiprocessors
DL1	κρυφή μνήμη δεδομένων πρώτου επιπέδου
DL2	κρυφή μνήμη δεδομένων δεύτερου επιπέδου
DoS	Denial of Service
EDP	Energy-Delay Product
IPC	Instructions Per Cycle
IQ	Instruction Queue
L1/2	κρυφή μνήμη πρώτου/δεύτερου επιπέδου
LLC	Last Level Cache
LRU	Least Recently Used
MLP	Memory Level Parallelism
QoS	Quality of Service
ROB	Reorder Buffer
RUU	Register Update Unit

Παράρτημα Β

Εργαλεία και Εξομοιωτές

B.1 Ο εξομοιωτής SimpleScalar

Το βασικό εργαλείο που χρησιμοποιήθηκε κατά την διάρκεια της συγκεκριμένης διδακτορικής διατριβής είναι η σουίτα των εξομοιωτών του εργαλείου SimpleScalar. Το SimpleScalar εργαλείο είναι ένας γρήγορος, ευέλικτος και μεγάλης ακρίβειας εξομοιωτής μοντέρνων επεξεργαστών και πιο συγκεκριμένα σχεδιάστηκε για την MIPS (Million Instructions Per Second) αρχιτεκτονική. Το εργαλείο αυτό δέχεται σαν είσοδο δυαδικά αρχεία, τα οποία είναι μεταγλωττισμένα πάνω στην αρχιτεκτονική του SimpleScalar και εξομοιώνει την εκτέλεση τους πάνω σε διάφορες αρχιτεκτονικές επεξεργαστών. Η επιλογή του συγκεκριμένου εργαλείου έγινε μετά από εκτενή αναζήτηση στον χώρο των εξομοιωτών και εκτιμήθηκε ότι διαθέτει όλα εκείνα τα απαραίτητα χαρακτηριστικά για την διεξαγωγή της έρευνας που διεξάχθηκε στην παρούσα διατριβή.

Πιο συγκεκριμένα, ο SimpleScalar παρέχει διάφορα μοντέλα ικανά να εξομοιώσουν με μικρότερη ή μεγαλύτερη ακρίβεια ένα υπολογιστικό σύστημα. Χαρακτηριστικά αναφέρουμε τον εξομοιωτή sim-cache, ο οποίος παρέχει τα απαραίτητα στατιστικά για τις κρυφές μνήμες, όπως είναι ο ρυθμός ευστοχίας και ο ρυθμός αντικατάστασης. Ο εξομοιωτής sim-fast ο οποίος εξομοιώνει τα λειτουργικά χαρακτηριστικά ενός πλήρους επεξεργαστή.

Για την εκτίμηση των μεθοδολογιών που προτάθηκαν στην παρούσα διατριβή χρησιμοποιήθηκε ο εξομοιωτής sim-outorder. Ο sim-outorder είναι ένας αναλυτικός εξομοιωτής βασισμένος σε κύκλο ρολογιού (cycle based) που διαθέτει όλα τα απαραίτητα μοντέλα για την εξομοίωση ενός επεξεργαστή (pipeline, αρχείο καταχωρητών, μονάδα αριθμητικών και λογικών δεδομένων κτλ.) καθώς και του υποσυστήματος της μνήμης (κρυφές μνήμες πρώτου και δευτέρου επιπέδου, κύρια μνήμη κτλ.). Ο sim-outorder μπορεί να εξομοιώσει επεξεργαστές, στους οποίους εφαρμόζεται η τεχνική εκτέλεσης των εντολών εκτός σειράς και μάλιστα για διάφορα ρεπερτόρια εντολών (PISA, Alpha και ARM).

Φυσικά, για την εκάστοτε μεθοδολογία που προτάθηκε στην παρούσα διατριβή έπρεπε να γίνουν οι κατάλληλες τροποποιήσεις στον πηγαίο κώδικα που παρέχεται από το εργαλείο.

Το pipeline του simplescalar λειτουργεί ως εξής: αρχικά, γίνεται η ανάκληση των εντολών από την κρυφή μνήμη εντολών του πρώτου επιπέδου (σε περίπτωση που οι ζητούμενες εντολές δεν υπάρχουν στην μνήμη του πρώτου επιπέδου, τότε η ανάκληση γίνεται από την μνήμη του δευτέρου επιπέδου ή τελικά από την κύρια μνήμη). Αν πρόκειται για εντολές άλματος, τότε γίνεται και πρόσβαση στον branch predictor για να ληφθεί η απόφαση για το τι είναι πιο πιθανό να γίνει μετά την εκτέλεση της συγκεκριμένης εντολής άλματος (ώστε το pipeline να παραμένει όσο το δυνατόν γεμάτο). Στην συνέχεια, οι εντολές αποκωδικοποιούνται και αποστέλλονται στον reorder buffer και στην issue queue. Οι εντολές ανάγνωσης/εγγραφής επιπλέον αποθηκεύονται στην Load/Store Queue. Οι εντολές μένουν στην issue queue μέχρι τελικά να αποσταλούν στα functional units και στην Load/Store queue μέχρι να ικανοποιηθούν οι αναφορές στην μνήμη (οι αναφορές ανάγνωσης ικανοποιούνται άμεσα, ενώ οι αναφορές εγγραφής ικανοποιούνται μόνο κατά το τελευταίο στάδιο του pipeline). Όταν οι εντολές εκτελεστούν (μετά την έξοδο από τα functional units), τότε αυτές ξυπνούν (wake-up) τις επόμενες εντολές που εξαρτώνται από αυτές. Το επόμενο βήμα είναι τα αποτελέσματα των εντολών να περάσουν στο επόμενο στάδιο (writeback) του pipeline, ώστε να αποθηκευτούν. Σε αυτό το σημείο, γίνεται και έλεγχος, αν ήταν σωστή η απόφαση που λήφθηκε από τον branch predictor. Σε περίπτωση λανθασμένης απόφασης, όλο το pipeline του επεξεργαστή καθαρίζεται (flashed). Τελικά, οι εντολές βγαίνουν από το pipeline (στάδιο commit). Σε αυτό το σημείο γίνεται και οι προσβάσεις στην μνήμη των εντολών εγγραφής.

B.2 Ο εξομοιωτής Wattch

Για την μελέτη των βελτιστοποιήσεων που προτάθηκαν για την μείωση της δυναμικής κατανάλωσης ισχύος χρησιμοποιήθηκε το εργαλείο Wattch. Το εργαλείο Wattch αποτελεί μια επέκταση του εξομοιωτή SimpleScalar και διαθέτει τα κατάλληλα μαθηματικά μοντέλα για τον υπολογισμό της δυναμικής κατανάλωσης ισχύος. Πιο συγκεκριμένα, το εργαλείο Wattch προσφέρει μοντέλα για τις ακόλουθες δομές ενός επεξεργαστή:

- Γενικές δομές μνήμης: κρυφές μνήμες εντολών και δεδομένων, αρχείο καταχωρητών, branch predictor, re-order buffer, issue queue και load/store queue.

- Μνήμες τύπου Content-Addressable Memories (πλήρους συσχετικότητας): στην συγκεκριμένη κατηγορία ανήκουν η issue queue wakeup logic, load/store queue address checks και τα Translation Lookaside Buffers (μόνο αν έχουν ρυθμιστεί ως μνήμες πλήρους συσχετικότητας).
- Ακολουθιακή λογική και χωρητικότητες καλωδίων: functional units, δίαυλοι δεδομένων και εντολών, selection και dependency logic.
- Ρολόι: χωρητικότητες καλωδιώσεων του ρολογιού, ενδιάμεσοι buffers κτλ.

Και στην περίπτωση του Wattch έγιναν οι κατάλληλες επεμβάσεις στον πηγαίο κώδικα του εργαλείου, ώστε να εκτιμηθούν κατάλληλα οι μεθοδολογίες που προτάθηκαν στην παρούσα διδακτορική διατριβή.

B.3 Ο εξομοιωτής HotLeakage

Το εργαλείο Hotleakage αποτελεί μια ακόμα επέκταση του εργαλείου Simplescalar για τον χαρακτηρισμό της στατικής κατανάλωσης ισχύος που καταναλώνεται από τις δομές της κρυφής μνήμης ενός επεξεργαστή. Αν και στην παρούσα διατριβή δεν προτάθηκαν μέθοδοι μείωσης της στατικής ισχύος, ο λόγος που χρησιμοποιήθηκε ο συγκεκριμένος εξομοιωτής είναι γιατί διαθέτει ενσωματωμένο τον αλγόριθμο υλοποίησης του Cache Decay, πάνω στον οποίο στηρίχτηκε μερικώς η μεθοδολογία που προτείνεται στο κεφάλαιο 3.

B.4 Ο εξομοιωτής CMP-SMT Simplescalar

Το συγκεκριμένο εργαλείο αποτελεί μια ακόμη επέκταση του Simplescalar ικανή όμως να εξομοιώσει επεξεργαστικές πλατφόρμες που αποτελούνται από περισσότερους του ενός επεξεργαστές. Το εργαλείο αυτό προέκυψε από την αρχική έκδοση του Simplescalar δημιουργώντας τόσες δομές για κάθε αρχιτεκτονική μονάδα, όσες είναι και ο αριθμός των επεξεργαστών που επιθυμεί ο χρήστης να εξομοιώσει. Ο CMP-SMT Simplescalar χρησιμοποιήθηκε για την μεθοδολογία που προτάθηκε για την διαχείριση του διαμοιρασμού της κοινόχρηστης κρυφής μνήμης σε πολυ-επεξεργαστικές πλατφόρμες.

B.5 Το εργαλείο CACTI

Τέλος, για την αποτίμηση των βελτιστοποιήσεων στις αρχιτεκτονικές μνήμης χρησιμοποιήθηκε το εργαλείο Cacti. Το συγκεκριμένο εργαλείο διαθέτει μοντέλα για την εκτίμηση του χρόνου πρόσβασης, του μεγέθους (area), καθώς και της καταναλισκόμενης δυναμικής και στατικής ισχύος διαφόρων τύπων μνημών. Επίσης, το Cacti (κάνοντας τις κατάλληλες τροποποιήσεις στον πηγαίο κώδικα) έχει χρησιμοποιηθεί για τον προσδιορισμό του χρόνου πρόσβασης διαφόρων άλλων τύπων μνημών, όπως είναι τα αρχεία καταχωρητών, οι CAMs (Context Addressable Memories) και οι TCAMs (Ternary Context Addressable Memories).

- Το μοντέλο που παρέχεται από το CACTI δέχεται τις ακόλουθες παραμέτρους:
- Το μέγεθος της κρυφής μνήμης.
- Η συσχετικότητα της κρυφής μνήμης (απλής, μερικής ή πλήρους συσχετικότητας).
- Το μέγεθος σε bytes που αφιερώνεται σε κάθε γραμμή της κρυφής μνήμης.
- Ο αριθμός αλλά και το είδος των πορτών της μνήμης.
- Το μέγεθος της τεχνολογίας.
- Ο αριθμός των ξεχωριστών τμημάτων της κρυφής μνήμης.

Ως αποτέλεσμα το CACTI παρέχει τους χρονισμούς πρόσβασης, το μέγεθος και την κατανάλωση ισχύος για όλα τα δομικά κομμάτια μιας κρυφής μνήμης, όπως είναι οι αποκωδικοποιητές, τα bitlines, τα wordlines, οι συγκριτές, οι ενισχυτές στάθμης, οι οδηγοί εξόδου κτλ.

Παράρτημα C

Σουίτες Μετροπρογραμμάτων

C.1 Η σουίτα SPEC CPU2000

Τα SPEC CPU2000 είναι μια σουίτα μετρικών προγραμμάτων αποτελούμενη από πραγματικές εφαρμογές. Είναι κοινά αποδεκτή από την επιστημονική κοινότητα και κατάλληλη για να παρέχει μετρικές απόδοσης για τις δυνατότητες του ίδιου του επεξεργαστή καθώς και του υποσυστήματος της μνήμης. Αυτή η συλλογή μετροπρογραμμάτων αποτελείται από δύο διαφορετικά σύνολα. Το ένα σύνολο περιέχει 12 μετροπρογράμματα ακεραίων αριθμών, δηλαδή εφαρμογές, οι οποίες συγκρίνουν την υπολογιστική απόδοση εκτελώντας πράξεις ακεραίων αριθμών. Το άλλο σύνολο αποτελείται από 14 μετροπρογράμματα αριθμών κινητής υποδιαστολής, τα οποία είναι κατάλληλα για να συγκρίνουν την υπολογιστική απόδοση συστημάτων, σε πράξεις δεκαδικών αριθμών. Τα χαρακτηριστικά των μετροπρογραμμάτων δίνονται στον ακόλουθο πίνακα.

Επίσης, στην τρίτη στήλη των πινάκων φαίνεται ο αριθμός των εντολών που πρέπει να προσπεραστεί (fast forwarding) σε κάθε μετροπρόγραμμα (ώστε τελικά να έχουμε μια αντικειμενική εικόνα της συμπεριφοράς του), ενώ στην τελευταία στήλη φαίνεται η γλώσσα προγραμματισμού, με βάση την οποία είναι υλοποιημένο το κάθε μετροπρόγραμμα. Ο αριθμός των εντολών που πρέπει να προσπεραστούν χωρίς εξομοίωση μετρήθηκε προσεκτικά, ώστε τα τελικά αποτελέσματα να αποτελούν μια πραγματική εικόνα της συμπεριφοράς του αντίστοιχου μετροπρογράμματος. Για παράδειγμα, η φάση των αρχικοποιήσεων των πινάκων ενός προγράμματος είναι προφανές ότι δεν μπορεί να δώσει αντιπροσωπευτικά μετρικά αποτελέσματα.

Πίνακας 4: Μετροπρογράμματα Κινητής Υποδιαστολής της σουίτας SPEC CPU2000

Πρόγραμμα	Περιγραφή	Fast Forwarding (M)	Γλώσσα
AMMP	Computational Chemistry	1.400	C
APPLU	Parabolic / Elliptic Partial Differential Equations	200	Fortran 77
APSI	Meteorology: Pollutant Distribution	200	Fortran 77
ART	Image Recognition / Neural Networks	1000	C
EQUAKE	Seismic Wave Propagation Simulation	3.400	C
FACEREC	Image Processing: Face Recognition	200	Fortran 90
FMA3D	Finite-element Crash Simulation	200	Fortran 90
GALGEL	Computational Fluid Dynamics	200	Fortran 90
LUCAS	Number Theory / Primality Testing	3.400	Fortran 90
MESA	3-D Graphics Library	500	C
MGRID	Multi-grid Solver: 3D Potential Field	200	Fortran 77
SIXTRACK	High Energy Nuclear Physics Accelerator Design	500	Fortran 77
SWIM	Shallow Water Modeling	500	Fortran 77
WUPWISE	Physics / Quantum Chromodynamics	3.400	Fortran 77

Πίνακας 5: Μετροπρογράμματα Ακεραίων Αριθμών της σουίτας SPEC CPU2000

Πρόγραμμα	Περιγραφή	Fast Forwarding (M)	Γλώσσα
BZIP2	Compression	200	C
CRAFTY	Game Playing: Chess	200	C
EON	Computer Visualization	200	C++
GAP	Group Theory, Interpreter	200	C
GCC	C Programming Language Compiler	200	C
GZIP	Compression	200	C
MCF	Combinatorial Optimization	3.400	C
PARSER	Word Processing	200	C
PERLBMK	PERL Programming Language	200	C
TWOLF	Place and Route Simulator	500	C
VORTEX	Object-oriented Database	200	C
VPR	FPGA Circuit Placement and Routing	2000	C

C.2 Η σουίτα SPEC CPU2006

Τα SPEC CPU2006 είναι η εξέλιξη της σουίτας SPEC CPU2000 και σταδιακά αντικαθιστούν τα CPU2000 σαν το κοινά αποδεκτό πρότυπο μετροπρογραμμάτων. Όπως και στην προηγούμενη έκδοση της σουίτας, τα μετροπρογράμματα βασίζονται σε πραγματικές εφαρμογές και καλύπτουν τις ίδιες κατηγορίες προγραμμάτων. Τα χαρακτηριστικά τους δίνονται στους ακόλουθους πίνακες:

Πίνακας 6: Μετροπρογράμματα Κινητής Υποδιαστολής της σουίτας SPEC CPU2006

Πρόγραμμα	Περιγραφή	Γλώσσα
bwaves	Fluid Dynamics	Fortran
gamess	Quantum Chemistry	Fortran
milc	Physics / Quantum Chromodynamics	C
zeusmp	Physics / CFD	Fortran
gromacs	Biochemistry / Molecular Dynamics	C, Fortran
cactusADM	Physics / General Relativity	C, Fortran
leslie3d	Fluid Dynamics	Fortran
namd	Biology / Molecular Dynamics	C++
dealII	Finite Element Analysis	C++
soplex	Linear Programming, Optimization	C++
povray	Image Ray-tracing	C++
calculix	Structural Mechanics	C, Fortran
GemsFDTD	Computational Electromagnetics	Fortran
tonto	Quantum Chemistry	Fortran
lbm	Fluid Dynamics	C
wrf	Weather	C, Fortran
sphinx3	Speech recognition	C

Πίνακας 7: Μετροπρογράμματα Ακεραίων Αριθμών της σουίτας SPEC CPU2006

Πρόγραμμα	Περιγραφή	Γλώσσα
perlbench	Programming Language	C
bzip2	Compression	C
gcc	C Compiler	C
mcf	Combinatorial Optimization	C
gobmk	Artificial Intelligence: Go	C
hmmer	Search Gene Sequence	C
sjeng	Artificial Intelligence: chess	C
libquantum	Physics / Quantum Computing	C
h264ref	Video Compression	C
omnetpp	Discrete Event Simulation	C++
astar	Path-finding Algorithms	C++
xalancbmk	XML Processing	C++

