Ατομική Διπλωματική Εργασία

Έρευνα στους πολυπύρηνους επεξεργαστές και εύρεση της αποδοτικότερης διάταξης για την μνήμη L2 cache.

Ζωγραφάκης Ιωάννης

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2010

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Έρευνα στους πολυπύρηνους επεξεργαστές και εύρεση της αποδοτικότερης διάταξης για την μνήμη L2 cache.

Ζωγραφάκης Ιωάννης

Επιβλέπων Καθηγητής

Δρ. Pedro Trancoso

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

Μάιος 2010

Ευχαριστίες

Θερμές ευχαριστίες προς τον επιβλέποντα καθηγητή Δρ. Pedro Trancoso, ο οποίος με την υποστήριξη και την καθοδήγηση του συνέβαλε καθοριστικά για την εκπόνηση αυτής της εργασίας. Θα ήθελα επίσης να ευχαριστήσω τον διδακτορικό φοιτητή Παναγιώτη Πετρίδη για την βοήθεια και τις πολύτιμες συμβουλές του.

Ευχαριστώ τέλος την οικογένεια μου και τους φίλους μου που μου στάθηκαν και σε αυτή την φάση της ζωής μου.

Περίληψη

Η δημιουργία επεξεργαστών σύμφωνα με την αρχιτεκτονική SMP (Symmetric Multiprocessing) στις μέρες μας είναι αρκετά δημοφιλής. Γνωρίζουμε ότι όπως και στους μονοπύρηνους επεξεργαστές έτσι και στους πολυπύρηνους, η μνήμη cache παίζει σημαντικό ρόλο για την καλή λειτουργία τους συστήματος. Καθώς όμως μεγαλώνει ο αριθμός των επεξεργαστών οι οποίοι χρησιμοποιούνται αναρωτιόμαστε αν η ξεχωριστή L2 cache για κάθε CPU είναι η βέλτιστη λύση και αν υπάρχει κάποιος άλλος τρόπος ώστε να πετύχουμε καλύτερη επίδοση για το σύστημα μας.

Στα πλαίσια της διπλωματική μας εργασίας εκτελούμε διάφορα benchmarks με σκοπό να ανακαλύψουμε το καλύτερη δυνατή διάταξη για την μνήμη L2 cache. Για την αποπεράτωση της εργασίας μας χρησιμοποιήσαμε τον SESC simulator για την εκτέλεση των δοκιμών μας, όπως επίσης και αρκετά configuration τα οποία δημιουργήσαμε.

Αξιολογώντας τα αποτελέσματα μας φτάσαμε στο συμπέρασμα ότι δεν υπάρχει κάποια διάταξη η οποία πάντοτε να μας δίνει τον καλύτερο χρόνο εκτέλεσης. Με βάση το παραπάνω πιστεύουμε ότι κάθε εφαρμογή ανάλογα με της ανάγκες της και τις ιδιαιτερότητες της αποδίδει καλύτερα σε διαφορετικό configuration.

Περιεχόμενα

Κεφάλαιο **1** **Εισαγωγή**..... **7**

1.1 Θέμα Διπλωματικής Εργασίας 7

Κεφάλαιο **2** **Τίτλος Δεύτερου Κεφαλαίου**..... **8**

2.1 Πολυπύρηνοι Επεξεργαστές 8

2.1.1 Εισαγωγή στους Πολυπύρηνους Επεξεργαστές 8

2.1.2 Xeon Intel 9

2.1.3 IBM Power 5 11

2.1.4 Intel 80-Core Terascale Processor 13

2.1.5 Larrabee 15

2.1.6 Piranha 17

2.1.7 AMD Opteron 19

2.1.8 IBM PowerXCell 8i 19

2.2 Σχετική Δουλειά 20

Κεφάλαιο **3** **Προσομοιωτής-Benchmarks-Configurations** **22**

3.1 Προσομοιωτής SESC 22

3.2 Benchmarks SPLASH 2 24

3.3 Configurations 24

3.3.1 Γενικές Πληροφορίες για τα Configurations 24

3.3.2 Configuration 1 27

3.3.3 Configuration 2 27

3.3.4 Configuration 3 28

3.3.5 Configuration 4 29

3.3.6 Configuration 5 29

3.3.7 Configuration 6 30

3.3.8 Configuration 7 31

3.3.9 Configuration 8 31

Κεφάλαιο **4** **Πειραματικά Αποτελέσματα** ……..…….. **33**

4.1 Πληροφορίες για τα πειραματικά αποτελέσματα 33

4.1 Benchmark LU 34

4.2 Benchmark Radix 42

4.3 Benchmark FFT 49

4.4 Benchmark Cholesky 58

ΚΕΦΑΛΑΙΟ **5** **Συμπεράσματα – Μελλοντική Εργασία**…….. **73**

5.1 Συμπεράσματα 73

5.2 Μελλοντική Εργασία 75

Β ι β λ ι ο γ ρ α φ ί α ……. **77**

Π α ρ ά ρ τ η μ α Α……..……..…….…… **Α-1**

Κεφάλαιο 1

Εισαγωγή

1.1 Θέμα Διπλωματικής Εργασίας 7

1.1 Θέμα Διπλωματικής Εργασίας

Η δημιουργία επεξεργαστών σύμφωνα με την αρχιτεκτονική SMP (Symmetric Multiprocessing) στις μέρες μας είναι αρκετά δημοφιλής. Όλο και περισσότερο είναι εμφανές ότι το μέλλον των επεξεργαστών βρίσκεται στην αύξηση των πυρήνων με σκοπό την καλύτερη δυνατή επίδοση. Γνωρίζουμε ότι όπως και στους μονοπύρηνους επεξεργαστές έτσι και στους πολυπύρηνους, η μνήμη cache παίζει σημαντικό ρόλο στην καλή λειτουργία τους συστήματος. Καθώς όμως μεγαλώνει ο αριθμός των επεξεργαστών οι οποίοι χρησιμοποιούνται αναρωτιόμαστε αν η ξεχωριστή L2 cache για κάθε CPU είναι η βέλτιστη λύση και αν υπάρχει κάποιος άλλος τρόπος να πετύχουμε καλύτερη επίδοση για το σύστημα μας.

Στα πλαίσια της διπλωματική μας εργασίας εκτελούμε διάφορα benchmarks με σκοπό να ανακαλύψουμε το καλύτερο δυνατό configuration για την μνήμη L2 cache, που θα μας δώσει τον μικρότερο δυνατό χρόνο εκτέλεσης. Για την αποπεράτωση της εργασίας μας χρησιμοποιήσαμε τον SESC simulator για την εκτέλεση των δοκιμών μας, όπως επίσης και αρκετά configuration τα οποία δημιουργήσαμε.

Κεφάλαιο 2

Σχετική Δουλειά

2.1 Πολυπύρηνοι Επεξεργαστές 8
2.1.1 Εισαγωγή στους Πολυπύρηνους Επεξεργαστές 8
2.1.2 Xeon Intel 9
2.1.3 IBM Power 5 11
2.1.4 Intel 80-Core Terascale Processor 13
2.1.5 Larrabee 15
2.1.6 Piranha 17
2.1.7 AMD Opteron 19
2.1.8 IBM PowerXCell 8i 19
2.2 Σχετική Δουλειά 20

2.1.1 Εισαγωγή στους Πολυπύρηνους Επεξεργαστές

Οι πολυπύρηνοι επεξεργαστές προήλθαν από τις ανάγκη για δυνατότερους υπολογιστές χωρίς το μειονέκτημα τις υψηλής κατανάλωσης και τις υψηλής θερμοκρασίας.

Λίγο μετά το 2000 τα ερευνητικά εργαστήρια ανακάλυψαν ότι η αύξηση της συχνότητας των επεξεργαστών , που αποτελούσε μέχρι τότε το κύριο μέσο τις αύξησης τις επίδοσης, είχε περιορισμούς και δεν θα μπορούσε να συνεχίσει επ’ άπειρο. Ο λόγος για αυτό το μεγάλο εμπόδιο ήταν η αύξηση της θερμοκρασίας όσο αυξανόταν η συχνότητα και η μεγάλη ανάγκη για κατανάλωση για την ψύξη αυτών των επεξεργαστών. Η λύση σε αυτό το πρόβλημα ήταν η δημιουργία επεξεργαστών που αντί για ένα δυνατό επεξεργαστή ενσωμάτωναν δύο ή και παραπάνω μικρότερης ισχύς. Αυτοί οι επεξεργαστές μπορούσαν να δουλεύουν παράλληλα βελτιώνοντας την συνολική απόδοση του συστήματος.

Στη συνέχεια βλέπουμε μερικούς επεξεργαστές που έχουν ιδιαίτερο ενδιαφέρον να δούμε με κάποια παραπάνω λεπτομέρεια. Σε αυτούς περιλαμβάνονται σχέδια που υπάρχουν διαθέσιμα στην αγορά όπως ο XEON της Intel και ο Power5 της IBM.Επίσης θα δούμε κάποια ερευνητικά προγράμματα όπως το Piranha ,ο Terascale processor της Intel και τον larrabee ένα επεξεργαστή ο οποίος προορίζεται απο την Intel να υπάρχει τοσο σαν κάρτα γραφικών όσο και σαν επεξεργαστής για εφαρμογές γενικής χρήσης. Τέλος θα περιγράψουμε σύντομα τους επεξεργαστές που χρησιμοποιεί το γρηγορότερο supercomputer σήμερα ,το Roadrunner, τους AMD Opteron και IBM PowerXCell 8i.

2.1.2 Xeon Intel

Οι επεξεργαστές XEON [1] είναι η σειρά της Intel που ενσωματώνει σε servers.Οι πρώτοι XEON βγήκαν σε κυκλοφορία το 1998 σαν αντικαταστάτες του Pentium II pro και είχαν ένα μόνο πυρήνα. Σήμερα βγαίνουν με δύο και τέσσερις πυρήνες και λέγεται ότι ο διάδοχος τους θα περιέχει οχτώ πυρήνες. Στα πλαίσια τις μελέτης μας, είδαμε τον Multi threaded dual core 65nm Xeon [2], ένα διπύρηνο επεξεργαστή που βγήκε στην αγορά το 2006 και υποστηρίζει την τεχνολογία multithreading, με μικρές ανάγκες σε χώρο. Ο επεξεργαστής αυτός έχει δύο πυρήνες των 64-bit ο καθένας και μια ενοποιημένη κρυφή μνήμη τρίτου επιπέδου χωρητικότητας 16 MB (unified L3 cache). Ο κάθε πυρήνας υποστηρίζει δύο threads και 1 MB κρυφή μνήμη δευτέρου επιπέδου. Το ιδιαίτερο χαρακτηριστικό αυτού του επεξεργαστή είναι ο τρόπος που υλοποιείται. Χρησιμοποιεί ένα caching bridge controller (CBC) για να στείλει τα δεδομένα από και προς την cache, τους πυρήνες και το front side bus (FSB). Για να καταφέρει να ενώσει τους δύο πυρήνες με το CBC χρησιμοποιεί ,υψηλής απόδοσης simple direct interface (SID). (figure 1)
Στο figure 2 βλέπουμε μια γενική απεικόνιση του die.

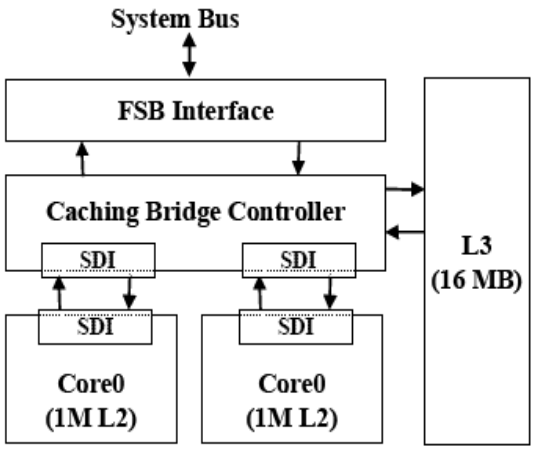


Figure 1: Processor Block Diagram

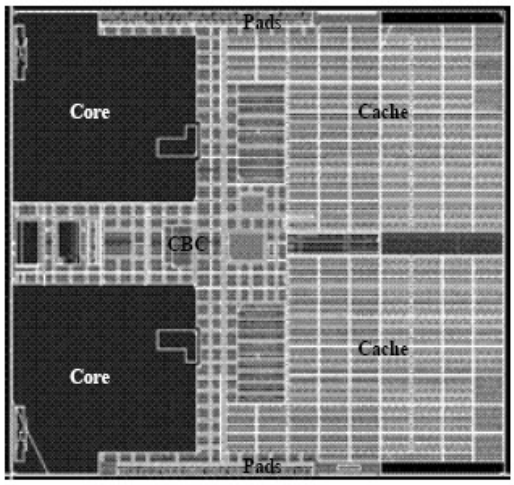


Figure 2: Processor Die Plot

Με την χρήση της αρχιτεκτονικής που περιγράψαμε και που φαίνεται από τις εικόνες, έγινε εφικτό να μειωθούν τα latencies για την προσέγγιση δεδομένων στην cache και στο external bus.

Για την δημιουργία του λογικού μέρους του CBC χρησιμοποιήθηκε η τεχνική βασισμένη στα cells.Η τεχνική αυτή είναι ιδιαίτερα αποδοτική σε σχέδια όπου ο χρόνος είναι το κύριο ζητούμενο ,για να επιτευχθεί καλύτερη επίδοση σε χρόνο μεγαλώνει το μέγεθος και δημιουργούνται μεγαλύτερες ανάγκες σε ισχύ. Η ανάγκη για ισχύ τελικά μειώθηκε με την χρήση ενός άλλου είδους transistors , τα Low Leakage transistors (LL-cells).

Ο επεξεργαστής αυτός κατάφερε τους στόχους του ,να είναι ισχυρός ,μικρός και χωρίς υψηλές ανάγκες σε ισχύ. Αναλυτικά σε εμβαδόν die 435 mm2 περιέχει 1.33B transistors, κάθε πυρήνας με 1,25V μπορεί να φτάσει στα 3.0GHz με χείριστη ανάγκη για ισχύ τα 150W, υπό φυσιολογικές για server συνθήκες χρειάζεται περίπου 100 W.

2.1.3 IBM Power 5

Ο IBM POWER5 [2] είναι ένας dual-core επεξεργαστής, όπου ο κάθε core τρέχει 2 threads. Τα threads διαμοιράζονται πολλούς πόρους, όπου το Global Competition Table (GCT or reorder buffer), το Branch History Table(BHT) και το Translation Lookaside Buffer (TLB). Ο IBM POWER5 εξισοροπεί κατάλληλα τη χρήση των πόρων διαμέσου των threads με διάφορους μηχανισμούς στο hardware, κάτι που σημαίνει μεγαλύτερη απόδοση.

Model	IBM POWER5+
Total number of cores	640
No. of cores per socket	2
Processor used	Dual-core IBM POWER5+
Core clock frequency (GHz)	1.9
Floating point/clock/core	4
Peak perf./core (Gflops)	7.6
Technology (nm)	90
L1 cache size (KB)	64 (I) & 32 (D)
L2 cache size (KB)	1.92 MB (I+D) shared
L3 cache size (MB)	36 (off-chip)
Local memory per node (GB)	32
Cores per node	16
Local memory/core (GB)	2
Total memory (GB)	1,280
Frequency of FSB (MHz)	533
Transfer rate of FSB (GB/s)	8.5
Interconnect	HPS (Federation)
Network topology	Multi-Stage
Operating system	AIX 5.3
Fortran compiler	xlf 10.1
C Compiler	xlc 8.0
MPI	POE 4.3
Page sizes	4 KB, 64 KB, 16 MB
File system	GPFS

Το POWER5+ chip είναι μια ανακατασκευασμένη έκδοση του POWER5 χρησιμοποιώντας 90-nanometer (nm) processor technology αντί των 130 nm και δεν έχει ουσιαστικά κάποια νέα features εκτός του ότι το clock frequence μπορεί να γίνει ελαφρώς υψηλότερο κατά τη διάρκεια της ζωής του. Αυτή η τεχνολογία σμίκρυνσης, κατέστησε δυνατό στην IBM να τοποθετήσει δύο processor cores σε ένα chip. Το chip frequency του POWER5+ είναι στο διάστημα 1.5-1.9 GHz.

Ο POWER5+ processor core περιλαμβάνει private L1 instruction και data caches. Κάθε dual chip module αποτελείται (DMC) από ένα POWER5+ chip (dual-core με on-chip L2) και ένα L3 cache chip. Και η L2 και η L3 cache είναι κοινή και για τους 2 πυρήνες. Το L1 instruction cache έχει χωρητικότητα 64 KB και είναι 2-way set associative, ενώ το L1 data cache έχει 32 KB χωρητικότητα και είναι 4-way set associative. Το POWER5+ chip έχει 1,92 MB L2 cache που διαιρείται εξίσου σε 3 modules και 10-way set associative με cache line 128 bytes. Η off-chip L3 cache έχει χωρητικότητα 36 MB και είναι 12-way set associative με cache line 256 bytes. Η L3 cache είναι επίσης χωρισμένη σε 3 κομμάτια, που

κάθε ένα από αυτά χρησιμεύει σαν spill cache, δηλαδή σαν cache υπερχείλισης του αντίστοιχου L2 κομματιού. Τα L2 cache modules είναι συνδεδεμένα με τους cores μέσω του Core Interface Unit, ένα 2(cores) x 3(L2 modules) crossbar με μέγιστο bandwidth της τάξης 40 bytes/cycle, ανά port. Αυτό επιτρέπει τη μεταφορά 32 bytes είτε στο L1 instruction cache είτε στο L1 data cache του κάθε core, όπως επίσης και την αποθήκευση 8 bytes στη μνήμη την ίδια χρονική στιγμή. Η L3 cache έχει μια λανθάνουσα κατάσταση 80 κύκλων, σε αντιδιαστολή με την κύρια μνήμη που έχει μια λανθάνουσα κατάσταση 220 κύκλων.

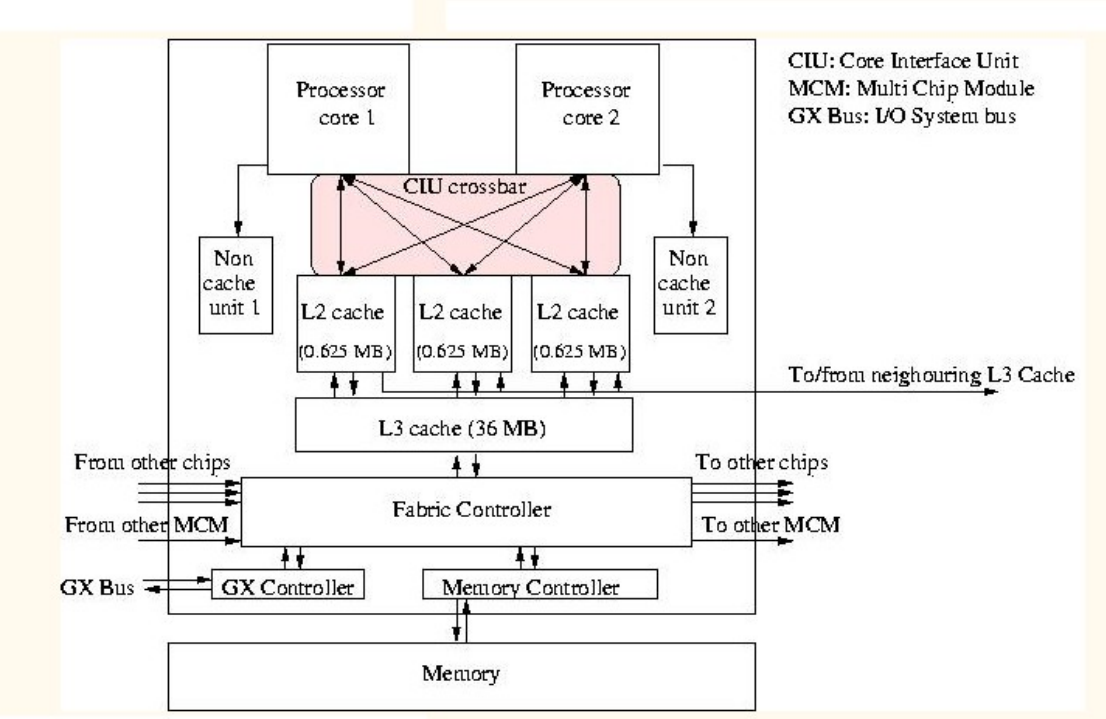


Figure 3: Διάγραμμα του IBM POWER5+ chip layout.

Υπάρχει ένα άλλο χαρακτηριστικό γνώρισμα του POWER5+ που δεν βοηθά για συχνά data access, αλλά που μπορεί να ωφελήσει για προγράμματα όπου το data access δεν είναι τόσο συχνό: το Simultaneus Multithreading (SMT). Τα POWER5 CPUs είναι σε θέση να κρατάνε δύο process threads εν ενεργεία, την ίδια χρονική στιγμή. Τα functional units παίρνουν instructions από οποιοδήποτε από τα δύο threads που είναι σε θέση να γεμίσουν ένα slot σε ένα instruction word, που θα γίνει issue στα functional units. Για τους πολύ συχνούς υπολογισμούς το single thread (ST) mode μπορεί να είναι καλύτερο και αυτό γιατί στο SMT τα 2 threads ανταγωνίζονται για τα entries στις caches που αυτό μπορεί να οδηγήσει στην περίπτωση των συχνών data access. Εδώ πρέπει να σημειώσουμε ότι το SMT είναι κάπως

διαφορετικό από το “normal” multithreading. Στην περίπτωση του “normal” multithreading, ένα thread που γίνεται stall, σταματά και αντικαθίσταται από κάποιο άλλο proccess thread που είναι ενεργό εκείνη τη στιγμή, κάτι που φυσικά κοστίζει σε χρόνο. Αυτό σημαίνει ότι το δεύτερο thread πρέπει να είναι ενεργό για ένα δίκαιο αριθμό κύκλων (κατά προτίμηση μερικές εκατοντάδες κύκλοι τουλάχιστον). Το SMT δεν έχει αυτό το μειονέκτημα, αλλά ο σχεδιασμός των instructions και των δύο threads είναι αρκετά περίπλοκος.

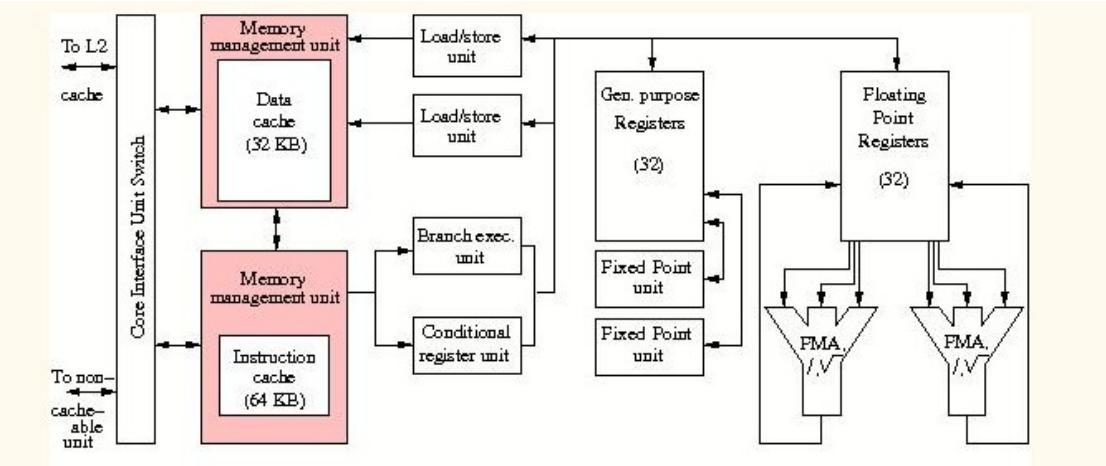


Figure 4: IBM POWER5+ processor core.

2.1.4 Intel 80-Core Terascale Processor

Ο επεξεργαστής αυτός [3] αποτελεί ένα ερευνητικό πρόγραμμα της Intel και είναι ο πρώτος generally programmable microprocessor που έσπασε το φράγμα του ενός Teraflop.Ο επεξεργαστής αυτός, όπως μαρτυράει και το όνομα του περιέχει 80 πυρήνες και κατασκευάστηκε με κύριο στόχο να μελετηθούν οι τρόποι για διαχείριση ενέργειας και να διερευνηθούν διαφορετικές προσεγγίσεις για την διασύνδεση των πυρήνων μέσα στο die. Δευτερεύων στόχος για την Intel, ήταν να αποδείξει ότι μπορεί να επιτευχθεί υψηλή επίδοση με τόσο μεγάλο αριθμό πυρήνων. Ο επεξεργαστής αυτός, ήταν ο πρώτος που κατάφερε να ξεπεράσει τρέχοντας εφαρμογή kernel τις ένα τρισεκατομμύριο floating point πράξεις ανά δευτερόλεπτο.

Ο επεξεργαστής, αποτελείται από ένα «δυσδιάστατο πίνακα» 8 x10 όπου κάθε κελί είναι ένας πυρήνας (figure 3).Συνολικά περιέχονται 100 εκατομμύρια transistors σε ένα die με

εμβαδόν 275mm2. Κάθε ένας από τους 80 πυρήνες περιέχει ένα δρομολογητή με 5 Ports , δύο μονάδες floating point, 3KB instruction memory , 2KB data memory , και ένα 10 port 32 entry register file. Κάθε Floating Point Unit (FPU) μπορεί να έχει μέγιστη επίδοση 4 FLOP/cycle/core.

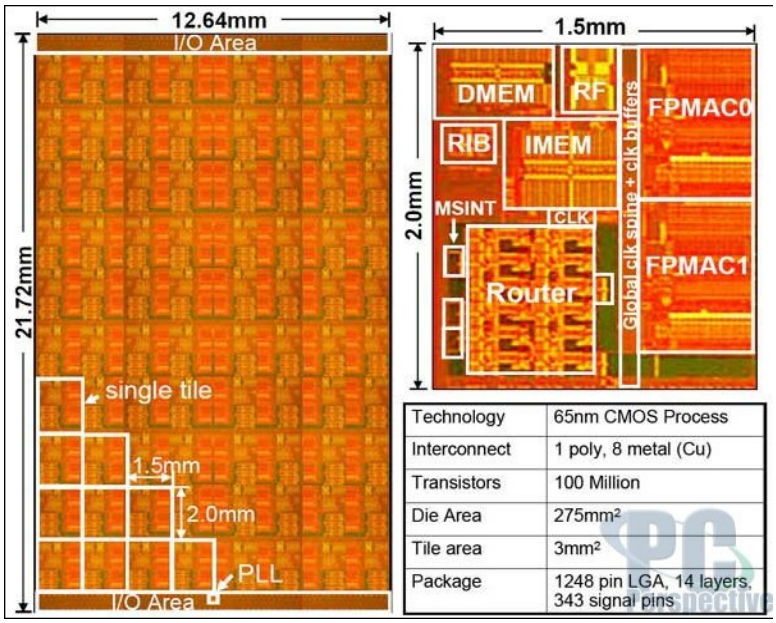


Figure 4: Η δομή του Intel 80 core terascale processor

Καθένας από τους 80 πυρήνες αποδίδει 4,27GHz με ζητούμενη ισχύ 1,07V. Συνολικά ο 80-core terascale processor μπορεί να αποδώσει μέγιστα 1,37 TFLOPS ανεβάζοντας θερμοκρασία 80 βαθμούς Κελσίου και χρειάζεται ισχύ 97 Watts.

Ο επεξεργαστής αυτός, καταφέρνει να επιτύχει καλύτερες επιδόσεις και από τον Tilera ,έναν επεξεργαστή με 64 πυρήνες, αλλά και από τον Intel Core 2 Duo Quad. Αναλυτικότερα, ο Tilera πετυχαίνει επίδοση 0.192 TFLOPS/s θέλοντας 35W ισχύ, ενώ ο 80-core terascale processor καταφέρνει να φτάσει τα 1,37 TFLOPS/s στα 97W πετυχαίνοντας επίδοση 2,5 φορές καλύτερη ανά μονάδα ισχύος (Watt). Αντίστοιχα ο Intel Core 2 Duo Quad πετυχαίνει επίδοση 0,9 GFLOPS/Watt την στιγμή που ο 80-core terascale processor φτάνει τα 19,4 GFLOPS/Watt επίδοση πάνω από 20 φορές μεγαλύτερη από τον κοινό πολυπύρηνο επεξεργαστή που κυκλοφορεί στην αγορά.

Παρόλο που ο 80-core terascale processor είναι ιδιαίτερα ισχυρός έχει ένα μεγάλο μειονέκτημα. Σαν ερευνητικό πείραμα που είναι, έχουν γίνει οι κατάλληλες τροποποιήσεις

έτσι ώστε να αυξηθεί η επίδοση. Στην περίπτωση αυτού του επεξεργαστή οι τροποποιήσεις είναι ιδιαίτερα σημαντικές αφού είναι κατάλληλος για μελέτη kernel. Βασικό χαρακτηριστικό που λείπει από τον επεξεργαστή αυτό, είναι το ελλιπές σύνολο εντολών assembly που μπορεί να εκτελέσει και δεν του επιτρέπει να τρέξει ολοκληρωμένες εφαρμογές.

Ο επεξεργαστής, χρησιμοποιεί επίσης λέξεις μήκους 96 bit, Very Long Instruction Word (VLIW). Με την υλοποίηση του VLIW καταφέρνει να εκτελεί μέχρι 4 πράξεις ανά κύκλο. Σημαντικός είναι και ο τρόπος, πάνω στον οποίο βασίστηκε το προγραμματιστικό μοντέλο του 80-core terascale processor που είναι το message passing για επικοινωνία και συγχρονισμό ανάμεσα στους πυρήνες. Σε αυτό το σημείο θα πρέπει να πούμε ότι το κάθε πρόγραμμα που γράφεται για τον 80-core terascale processor είναι γραμμένο σε assembly στο χέρι ειδικά για αυτόν ,με τον περιορισμό ενός μόνο loop και με συγκεκριμένη έκταση του offset στην μνήμη. Γενικότερα, θεωρείται ερευνητικό πρόγραμμα με μέλλον που περιμένουμε να δούμε επιδόσεις με ολοκληρωμένο instruction set.

2.1.5 Larrabee

Ο Larrabee [4], αποτελεί ένα καινούριο σχέδιο της Intel που προσδοκά να καλύψει τόσο την αγορά των καρτών γραφικών, βραχυπρόθεσμα, όσο και την αγορά των επεξεργαστών, μακροπρόθεσμα . Είναι ένας επεξεργαστής που χρησιμοποιεί πολλούς in-order πυρήνες που επεξεργάζονται το x86 instruction set (figure4). Οι πυρήνες που περιέχει ο Larrabee βασίζονται στους είδη υπάρχοντες in-order Pentium με την προσθήκη εντολών μήκους 64 bits, multithreading και ενός wide vector processor unit (VPU). Οι πυρήνες που ενσωματώνει ο επεξεργαστής έχουν 32KB instruction cache όπως επίσης και 32 KB data cache σε αντίθεση με τα 8KB Icache και 8KB Dcache που είχαν οι απλοί single-threaded Pentiums προκάτοχοι τους.

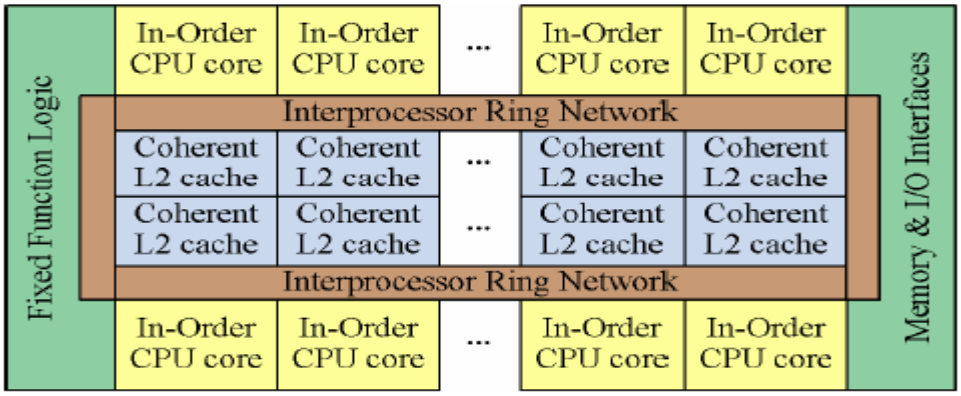


Figure 5: Η δομή του Larrabee

Ο κάθε πυρήνας επικοινωνεί με την L2 που του αντιστοιχεί (256 KB μέρος της συνολικής L2 cache), μέσω του δακτυλίου δύο κατευθύνσεων που περιβάλει την μνήμη. Επίσης ο δακτύλιος με την βοήθεια των fixed function logic αλλά και των memory & I/O interfaces δίνουν την δυνατότητα να επικοινωνούν μεταξύ τους τα cores. Κύριο ρόλο στην επίδοση παίζουν οι vectors που σε συνδυασμό με την in-order εκτέλεση των εντολών μας δίνουν υψηλή επίδοση ανά watt αλλά και σε σχέση με το μέγεθος που καταλαμβάνει ο πυρήνας. Παρακάτω βλέπουμε την δομή του κάθε πυρήνα (figure 5) αλλά και τον πίνακα (figure 6),όπου βλέπουμε τις διαφορές μεταξύ out-of order και in-order εκτέλεση εντολών.

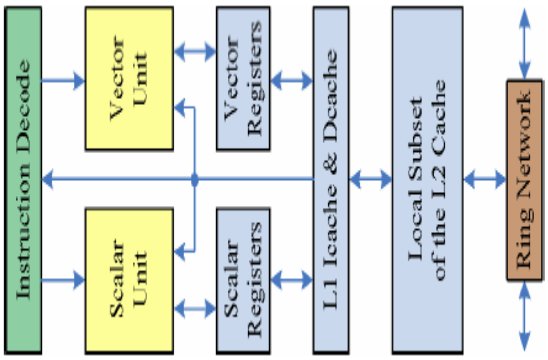


Figure 6 :Δομή του κάθε core στο Larrabee

# CPU cores:	2 out-of-order	10 in-order
Instruction issue:	4 per clock	2 per clock
VPU per core:	4-wide SSE	16-wide
L2 cache size:	4 MB	4 MB
Single-stream:	**4 per clock**	**2 per clock**
Vector throughput:	**8 per clock**	**160 per clock**

Figure 7: Πίνακας σύγκρισης in/out order

Όπως παρουσιάζει και ο πίνακας, όταν έχουμε in-order εκτέλεση το throughput των vectors αυξάνεται κατά πολύ, αν και μειώνονται τα issues, αποδίδοντας πολύ καλά για τον επεξεργαστή Larrabee. Πρέπει επίσης να σημειώσουμε ότι οι Pentium που χρησιμοποιούνται για το Larrabee διαφέρουν όχι μόνο στο cache αλλά και σε άλλα μέρη με τους πρόγονούς τους. Οι κυριότερες διαφορές που έχουν είναι ότι πλέον υποστηρίζουν 64 bit εντολές αλλά και multithreading. Ιδιαίτερα εξελιγμένο είναι και το prefetching που επιτρέπει στον

επεξεργαστή να έχει περιορισμένα misses. Παρόλο τις αλλαγές που έχουν γίνει , οι πυρήνες αυτοί υποστηρίζουν ακόμα το x86 instruction set διατηρώντας backward compatibility.

Ο Larrabee χρησιμοποιεί dual–issue decoder με υψηλό multi issue rate όπως είδαμε και από την ερευνά μας. Το κύριο βάρος της επεξεργασίας των εντολών γίνεται μέσο ενός primary pipeline ενώ το δεύτερο είναι βοηθητικό και εκτελεί απλές πράξεις. Επιπροσθέτως, ο επεξεργαστής που εξετάζουμε διαθέτει τέσσερα threads εκτέλεσης. Η εναλλαγή τους μας βοηθάει να γλιτώσουμε καθυστερήσεις που μπορεί να οφείλονται είτε στην αδυναμία του μεταγλωττιστή να κάνει σωστή χρονοδρομολόγηση ώστε να μην έχουμε stall, είτε στην αδυναμία να γίνουν τα σωστά δεδομένα prefetched στην ώρα τους.

Τέλος, το κύριο πλεονέκτημα του larrabee είναι ότι μπορεί να αποδώσει ιδιαίτερα καλά τόσο σε εφαρμογές γραφικών, όσο και σε εφαρμογές γενικής χρήσης.

2.1.6 Piranha

Το Piranha [5] είναι ένα ερευνητικό πρόγραμμα που ανέπτυξε η Compaq το 2000 με σκοπό να δημιουργήσουν σε λίγο χρόνο και με μικρό κόστος ένα σύστημα που μπορεί να επεξεργάζεται on-line transactions αποδοτικά.

Το κύριο κομμάτι του Piranha είναι ένας processing node με 8 απλούς πυρήνες Alpha με ξεχωριστή L1 cache για instructions και data για κάθε πυρήνα , shared L2 cache ,οχτώ ελεγκτές μνήμης,2 protocol engines και ένα network router όλα σε ένα die (figure 7).

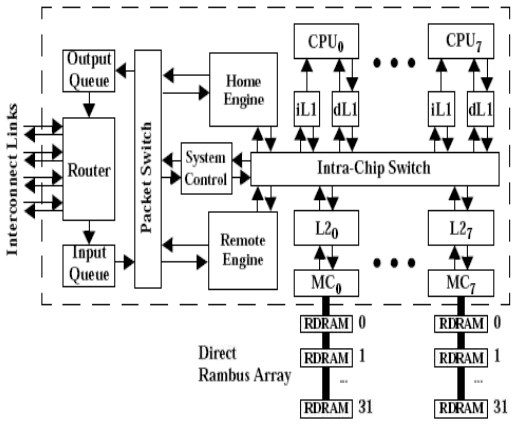


Figure 8: Η δομή του processing chip

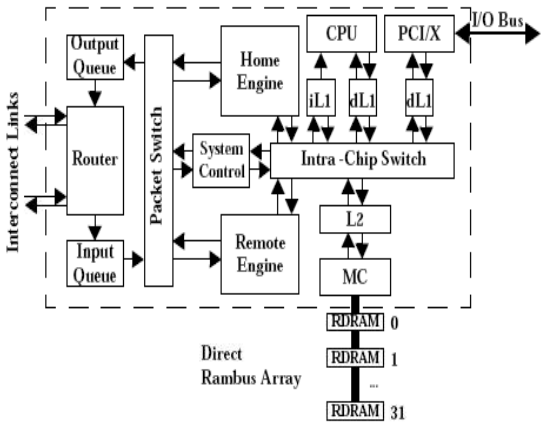


Figure 9: Η δομή του I/O chip

Το chip που περιγράψαμε, δεν έχει την δυνατότητα για I/O ενέργειες. Οι ενέργειες αυτές γίνονται στο I/O chip (figure 8) το οποίο είναι συγκριτικά μικρό σε σχέση με αυτό που χρησιμοποιείται για τις κυρίως πράξεις. Η κύρια διαφορά του με το processing chip είναι ότι περιέχει μόνο ένα πυρήνα και συνεπώς μόνο μια L2 cache και ένα memory controller.Τα δύο αυτά chips ενωμένα μπορούν να μας δώσουν συστήματα με μεγάλη απόδοση.

Τα modules αυτά όπως αναφέραμε και πριν αποτελούνται από επεξεργαστές Alpha. Τα cores αυτά έχουν συχνότητα 500MHz και pipeline 8 σταδίων. Επίσης κάνουν issue μία εντολή ανά κύκλο και τα issues γίνονται in order. Έχουν ξεχωριστή L1 cache για τα instructions και τα data που η καθεμία είναι 64KB με δομή two-way set associative. Η L2 cache που χρησιμοποιείται ενιαία και από τους οχτώ πυρήνες είναι 1MB χωρισμένη σε οχτώ banks, ένα για κάθε core. Κάθε bank είναι 8-way set associative, χρησιμοποιεί round robin replacement policy έχει την δυνατότητα να επικοινωνήσει μέσω του κεντρικού interface με τα άλλα modules στο chip.

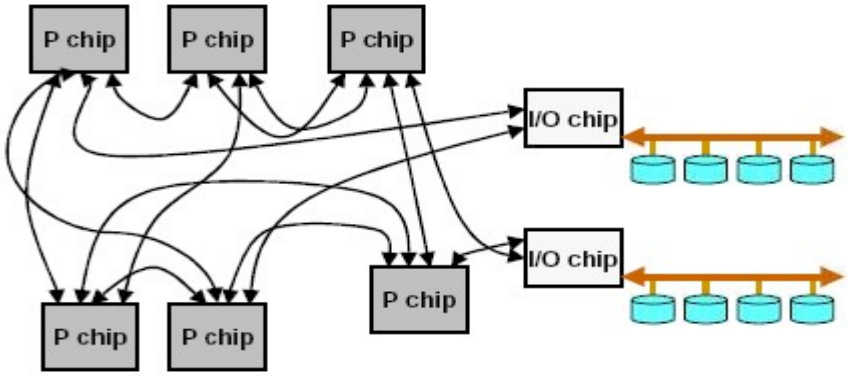


Figure 10: Piranha σύστημα με έξι processing και δύο I/O chips

Μεγάλο πλεονέκτημα του σχεδίου αυτού της Compaq είναι ότι κάθε σύστημα μπορεί να έχει διαφορετικό αριθμό processing nodes δίνοντας κάθε φορά στο σχέδιο την ισχύ και την επίδοση που χρειάζεται. Στην figure 9 μπορούμε να δούμε ένα σύστημα piranha με οχτώ processing nodes και δύο I/O nodes.Με τον τρόπο αυτό μπορούν να επεξεργαστούν στον ίδιο χρόνο πολλά παραπάνω δεδομένα επιτυγχάνοντας καλύτερη επίδοση σε συστήματα που μπορούν να μας δώσουν τον απαιτούμενο παραλληλισμό όπως οι on-line transactions με διαφορετικές βάσεις δεδομένων.

2.1.7 AMD Opteron

Στη συνέχεια θα κάνουμε μία σύντομη παρουσίαση για τους επεξεργαστές που αποτελούν το γρηγορότερο supercomputer αυτή τη στιγμή , τον Roadrunner. Οι παρακάτω επεξεργαστές συνδυασμένοι στον Roadrunner φτάνουν το 1,38 Pflop/s peak performance.

Η βασική υπολογιστική μονάδα του Roadrunner είναι ο επεξεργαστής της AMD, opteron [6]. Ο επεξεργαστής αυτός περιέχει δύο πυρήνες και πετυχαίνει επίδοση 1,8GHz. Επίσης κάθε πυρήνας έχει 64 KB data ,64 KB instruction L1 cache και 2 MB L2 unified cache.Ο καθένας προσφέρει επίδοση 7,2Gflops/s χρησιμοποιώντας double precision (DP) και 14,4Gflops/s χρησιμοποιώντας single precision (SP).

2.1.8 IBM PowerXCell 8i

Ο IBM PowerXCell8i [6] είναι μια παραλλαγή του Cell BE που χρησιμοποιήθηκε στον Roadrunner λόγω των χαμηλών επιδόσεων του αρχικού μοντέλου στις double precision πράξεις. Ο επεξεργαστής αυτός χρονίζεται στα 3,2 GHz και περιέχει συνολικά εννιά πυρίνες. Οι οχτώ απο αυτούς έχουν βοηθητική λειτουργία και για αυτό λέγονται Synergistic Processing Elements (SPE).Κάθε SPE περιέχει μια SIMD (single instruction multiple data) unit που μπορεί να κάνει issue συνολικά 4 double precision floating point ή 8 single point floating point πράξεις τον κύκλο. Ο κύριος πυρήνας λέγεται Power Processing Element (PPE) και μπορεί να κάνει issue 2 double precision floating point πράξεις τον κύκλο. Επίσης ο κύριος πυρήνας δομείται με ιεραρχία cache. Αναλυτικά περιέχει 32 KB L1 data cache και 32 KB L1 instruction cache όπως επίσης και 512 KB L2 cache. Επιπρόσθετα ο επεξεργαστής αυτός υποστηρίζει DDR2 μνήμη μέχρι και 32 GB. Τέλος είναι σημαντικό να πούμε είναι σημαντικό να πούμε ότι οι επεξεργαστές PowerXcell 8i παίζουν ρόλο accelerator στο σύστημα και προσφέρουν το 95 % της συνολικής επίδοσης του supercomputer.

2.2 Σχετική Δουλειά

Καθώς μεγαλώνει η πυκνότητα των επεξεργαστών εκθετικά σε σχέση με τον νόμο του Moore οι επιστήμονες θέτουν σαν στόχο την δημιουργία συστημάτων με αρκετά cores σε ένα chip (CMP – Chip Multi-Processor) [7] . Δυστυχώς όμως το capacity του memory bandwidth εκτός του chip αυξάνεται δυσανάλογα σε σχέση με την αύξηση του αριθμού των cores δημιουργώντας ένα performance και throughput bottleneck γνωστό και ως bandwidth wall problem. Προσπαθώντας να ξεπεράσουν το πρόβλημα δημιουργήθηκε ένα αναλυτικό μοντέλο το οποίο μας επιτρέπει να μελετήσουμε το πρόβλημα του bandwidth wall για τα CMP συστήματα. Μέσα από το μοντέλο αυτό παρατηρήθηκε ότι το bandwidth wall μπορεί να βλάψει σοβαρά και να περιορίσει το core scaling.

Παρατηρήθηκε ότι χωρίς τις τεχνικές για την διατήρηση του bandwidth, το πρόβλημα του bandwidth wall πολλαπλασιάζεται σε μεγάλο βαθμό. Χαρακτηριστικό είναι το παράδειγμα όπου για αρχικό configuration το οποίο έχει οχτώ πυρήνες το οποίο μοιράζει το 50 % του χώρου του die στα cores και το άλλο 50% του συνολικού χώρου στις caches, σε τέσσερις τεχνολογικές γενιές ο χώρος για τις caches θα πρέπει να μεγαλώσει στο 90 % έναντι 10 % για τα cores έτσι ώστε να έχουμε αρκετά μεγάλη cache για να μπορέσουμε να διατηρήσουμε το συνολικό memory traffic ανά πυρήνα στα ίδια επίπεδα. Σαν συνέπεια, ο αριθμός των πυρήνων θα αυξηθεί από 8 σε 24 (3 x αύξηση),αντί για 128 (16 x αύξηση) που περιμέναμε αν είχαμε αναλογική αύξηση.

Διαπιστώθηκε επίσης ότι οι τεχνικές bandwidth conservation διαφέρουν σημαντικά μεταξύ τους όσο αφορά την ικανότητα που έχουν να μειώσουν την ανάγκη για memory traffic. Για παράδειγμα χρησιμοποιώντας DRAM caches μπορούσαμε να αυξήσουμε τον αριθμό των cores μέχρι και σε 47 σε τέσσερις τεχνολογικές γενεές ενώ αυξάνοντας την επίδοση για μικρότερα σε μέγεθος cores για να κερδίσουμε χώρο στο die , έχει μικρότερο όφελος στην προσπάθεια μας να αυξήσουμε τα συνολικά cores.

Μια κατηγοριοποίηση που έγινε σχετικά με τις τεχνικές bandwidth conservation έχει να κάνει με τον τρόπο που επηρεάζουν το bandwidth άμεσα (directly) ή έμμεσα (indirectly). Οι direct techniques αυξάνουν το bandwidth με το να αυξήσουν την συχνότητα των memory interfaces εκτός του chip, και αυξάνοντας τον αριθμό των καναλιών προς την off-chip

μνήμη. Τέτοιου είδους τεχνικές είναι η Link Compression και η Sectored Caches. Αντίθετα οι indirect techniques μειώνουν τα memory traffic requirements αυξάνοντας την effective cache capacity ανά πυρήνα. Τέτοιου είδους τεχνικές είναι η Cache Compression, η DRAM Cache, η 3D-Stacked Cache, η Unused Data Filtering και η τεχνική Smaller Cores. Υπάρχουν τέλος και οι τεχνικές που συνδυάζουν τα δυο παραπάνω είδη, παραδείγματα τέτοιων είναι η Small Cache Lines, η Cache + Link Compression και η Data Sharing.

Συνδυάζοντας παραπάνω από μια τεχνικές για bandwidth conservation καταφέραμε να πάρουμε ιδιαίτερα ευνοϊκά αποτελέσματα. Για παράδειγμα συνδυάζοντας 3D-Stacked Cache, DRAM Cache, Cache + Link Compression και Small Cache Lines καταφέραμε σε 4 τεχνολογικές γενεές να αυξήσουμε τον αριθμό των cores σε 183 (λαμβάνοντας το 71% του die area) κρατώντας το bandwidth σε ιδιαίτερα καλά επίπεδα. Λαμβάνοντας υπόψη το παραπάνω αποτέλεσμα πιστεύουμε ότι μπορούμε να καθυστερήσουμε το bandwidth wall πρόβλημα για μερικές τεχνολογικές γενεές ακόμα με την προϋπόθεση να χρησιμοποιούμε πολλαπλές bandwidth conservation techniques συνδυασμένες αποδοτικά.

Κεφάλαιο 3

Προσομοιωτής – Benchmarks - Configurations

3.1 Προσομοιωτής SESC 22
3.2 Benchmarks SPLASH 2 24
3.3 Configurations 24
3.3.1 Γενικές πληροφορίες για τα Configurations 24
3.3.2 Configuration 1 27
3.3.3 Configuration 2 27
3.3.4 Configuration 3 28
3.3.5 Configuration 4 29
3.3.6 Configuration 5 29
3.3.7 Configuration 6 30
3.3.8 Configuration 7 31
3.3.9 Configuration 8 31

3.1 Προσομοιωτής SESC

Στην έρευνα για μικρό-αρχιτεκτονική υπολογιστών , συχνά οι ερευνητές έχουν κάποιο είδος πρότασης για έναν νέο μικροεπεξεργαστή ο οποίος θα είναι καλύτερος από τους είδη υπάρχοντες. Αυτός ο μικροεπεξεργαστής ίσως να είναι γρηγορότερος, ίσως να είναι περισσότερο energy-efficient και ίσως να είναι πιο αξιόπιστος από τους είδη υπάρχοντες. Σε κάθε περίπτωση, είναι πολυδάπανο να σχεδιάσεις και να κατασκευάσεις ένα επεξεργαστή από την αρχή. Για αυτό τον λόγω οι ερευνητές δημιουργούν τους προσομοιωτές επεξεργαστών. Υπάρχουν αρκετές διαφορετικές προσεγγίσεις ως προς τον τρόπο που θα δουλεύει ο προσομοιωτής με πιο διαδεδομένη τους execution-driven προσομοιωτές, όπου ο simulator όντως εκτελεί το πρόγραμμα που προσομοιώνει. Συνηθισμένο είναι επίσης να χωρίζεται ο προσομοιωτής στον emulator,ο οποίος εκτελεί και την εφαρμογή, και στον

timing simulator , ο οποίος μοντελοποιεί τον χρόνο και την ενέργεια της εφαρμογής που εκτελείται.

Ο SESC λοιπόν είναι ένας simulator ο οποίος αναπτύχθηκε κυρίως από την ομάδα i-acoma στο πανεπιστήμιο του Ιλλινόις και από εκεί και πέρα επεκτείνεται από την διεθνή υπολογιστική κοινότητα, μιας και αποτελεί open source πρόγραμμα. Ο προσομοιωτής αυτός μοντελοποιεί διάφορες αρχιτεκτονικές επεξεργαστών όπως μονοπύρηνους και πολυπύρηνους. Υλοποιεί ένα full out-of-order pipeline με branch prediction, caches, buses, και οτιδήποτε άλλο συστατικό είναι αναγκαίο για μια σύγχρονη προσομοίωση.

Ο SESC είναι ένας event-driven simulator.Έχει ένα emulator που έχει δημιουργηθεί από την MINT, ένα παλαιό εγχείρημα το οποίο κάνει emulate ένα επεξεργαστή τεχνολογίας MIPS. Πολλές λειτουργίες στον πυρήνα του επεξεργαστή καλούνται κάθε κύκλο ενώ άλλες καλούνται μόνο όταν χρειάζονται μέσω events.

Ο SESC είναι γραμμένος σε C++ και είναι open source δίνοντας τη δυνατότητα σε όλους μας να τον επεκτείνουμε. Αν και είναι open source κάθε νέο κομμάτι εξετάζεται ενδελεχώς από την ομάδα του SESC και ειδικά από τον επικεφαλή της Jose Renau για να επιβεβαιωθεί ότι ο νέος κώδικας είναι εξίσου γρήγορος και σωστά γραμμένος όπως και τα παλαιότερα κομμάτια.

Όπως τονίσαμε ιδιαίτερη μέριμνα έχει δοθεί ώστε ό εξομοιωτής αυτός να είναι ιδιαίτερα γρήγορος, χαρακτηριστικό είναι ότι μπορεί να τρέξει πάνω από 1,5 εκατομμύριο εντολές για την αρχιτεκτονική MIPS σε ένα επεξεργαστή Pentium 4 συχνότητας 3 GHz.Επίσης ο SESC έχει την δυνατότητα να τρέξει σε πολλά UNIX λειτουργικά συστήματα όπως Linux, Darwin/MacOS X και επίσης τρέχει σε big-endian και little-endian επεξεργαστές.

Όσο αφορά τις μνήμες Cache ο SESC μπορεί να προσομοιώσει

- Sizes
- Hit & Miss latencies
- Replacement policies
- Cache-line sizes
- Associativities

Η υλοποίηση της Cache μνήμης για τον SESC είναι ιδιαίτερα πολύπλοκη και χρησιμοποιεί πολλά event-driven callbacks, τα οποία είναι αναγκαία για την μοντελοποίηση όλων των latencies και τον transactions που γίνονται στις caches.

3.2 Benchmarks SPLASH 2

Τα SPLASH 2 benchmarks είναι μια σουίτα από παράλληλες εφαρμογές που δημιουργήθηκε από το πανεπιστήμιο του Stanford και προέρχεται από τα αρχικά. **S**tanford **P**arallel **A**pplications for **SH**ared Memory. Έχοντας είδη χρησιμοποιήσει παλαιότερη έκδοση της εν λόγω σουίτας σε προηγούμενη ενασχόληση μας με προσομοιωτές επιλέξαμε τη συγκεκριμένη σουίτα με benchmarks για να εκτελέσουμε τα πειράματα μας μετά από προτροπή του επιβλέποντα καθηγητή.

Η σουίτα αυτή περιέχει 4 kernels και 8 ολοκληρωμένες εφαρμογές που κατά πολύ βασίζονται στα kernels αυτά. Για την εκτέλεση των πειραμάτων μας και την εξαγωγή των αποτελεσμάτων μας χρησιμοποιήσαμε τα 4 kernels και πιο συγκεκριμένα της εφαρμογές, FFT, Radix, LU και Cholesky. Είναι σημαντικό να αναφέρουμε ότι για την εφαρμογή Cholesky χρησιμοποιήσαμε δυο διαφορετικά inputs για να δούμε τις διαφορές στην συμπεριφορά του benchmark αυτού, καθώς δέχεται inputs διαφορετικού όγκου δεδομένων.

3.3.1 Γενικές πληροφορίες για τα Configurations

Αρχικά, πρέπει να πούμε ότι όλες μας τις προσομοιώσεις τις εκτελέσαμε στον υπολογιστή του Πανεπιστημίου Κύπρου Θαλή (Thales) , ό οποίος περιέχει τέσσερις διπύρηνους επεξεργαστές AMD Opteron, χρονισμένους στα 1,8GHz, καθώς επίσης και 4GB κύριας μνήμης.

Για την εκπόνηση των πειραμάτων μας , χρησιμοποιήσαμε 8 διαφορετικά configurations για να έχουμε με συλλογή αποτελεσμάτων τα οποία θα μπορούσαν να μας βοηθήσουν στην εξαγωγή των σωστών συμπερασμάτων. . Σε όλα τα configurations ο κάθε επεξεργαστής έχει

συχνότητα 5 GHz ενώ καταλαμβάνει χώρο ίσο με 32 nm. Επιπλέον ο κάθε επεξεργαστής χρησιμοποιεί out of order , fetch policy , κατά την οποία σε περίπτωση που υπάρχει stall λόγω μια εντολής ο επεξεργαστής εκτελεί την επόμενη εφόσον δεν υπάρχουν εξαρτήσεις, ενώ κάνει issue 4 εντολές κάθε φορά. Κάθε επεξεργαστής έχει ξεχωριστή μνήμη cache επιπέδου ένα που χωρίζεται σε δυο μέρη, την L1 cache για instruction και την L1 cache για Data. Το κάθε ένα από τα δυο αυτά κομμάτια της L1 έχει χωρητικότητα 32 KB και associativity 2 ενώ ακολουθεί πολιτική Write Through (WT), κατά την οποία η πληροφορία γράφεται και στο block στην μνήμη cache αλλά και στην μνήμη χαμηλότερου επιπέδου. Επίσης η κρυφή μνήμη επιπέδου ένα ακολουθεί πολιτική αντικατάστασης Least Recently Used (LRU) κατά την οποία το block μνήμης που αντικαθίσταται είναι αυτό που έχει χρησιμοποιηθεί παλαιότερα. Επιπλέον θέλαμε τα configurations που θα χρησιμοποιήσουμε να μας δίνουν τις περισσότερες δυνατές επιλογές. Για να επιτύχουμε αυτό τον στόχο μας δημιουργήσαμε configurations όπου η μνήμη L2 cache διαμοιραζόταν ανάμεσα στους επεξεργαστές. Τα configurations που δημιουργήσαμε έχουν αλλαγές στον αριθμό των επεξεργαστών που διαμοιράζονται την μνήμη L2 cache. Οι επεξεργαστές που διαμοιράζονται κάθε φορά την μνήμη είναι πάντα δυνάμεις του αριθμού 2. Τον περιορισμό αυτό μας τον επέβαλαν τα benchmarks που χρησιμοποιήσαμε για τις δοκιμές μας. Τον αριθμό των επεξεργαστών που διαμοιράζονται την μνήμη μπορούμε να τον αλλάξουμε από την εντολή που ακολουθεί, η οποία βρίσκεται στο κομμάτι *[L1L2Bus]* του configuration file (παράρτημα Α).

lowerLevel = “L2Cache L2 shareBy X”

Όπου το Χ είναι ο αριθμός των επεξεργαστών που διαμοιράζονται την μνήμη Όπως είναι φυσιολογικό για να έχουμε συγκρίσιμα αποτελέσματα κάθε φορά που αλλάζαμε τον αριθμό των επεξεργαστών που μοιράζονταν την μνήμη L2 cache αλλάζαμε ανάλογα και το μέγεθος της. Τα μεγέθη αυτά μπορούμε να τα δούμε στον πίνακα που ακολουθεί, όπου στην δεξιά στήλη έχουμε τον αριθμό των επεξεργαστών που συνδέονται με την κρυφή μνήμη επιπέδου δυο και στα δεξιά βλέπουμε το μέγεθος της σε bytes. Όπως γνωρίζουμε και από την θεωρεία η αύξηση της L2 έχει σαν άμεσο αποτέλεσμα την μείωση των misses που θεωρητικά θα επηρεάσει θετικά και τον χρόνο εκτέλεσης για την εκτελούμενη εφαρμογή.

Number of CPUs	Cache Size (B)
1	2097152
2	4194304
4	8388608
8	16777216
16	33554432
32	67108864
64	134217728
128	268435456

Σε όλα τα configuration μας η κρυφή μνήμη επιπέδου δυο έχει associativity 8 και ακολουθεί πολιτική

αντικατάστασης LRU. Επίσης ακολουθεί write policy, Write Back (WB), κατά την οποία η πληροφορία αποθηκεύεται μόνο στο block στην cache και μόνο εφόσον έχει αλλαχτεί μεταφέρεται στην main memory όταν έρθει η ώρα να αντικατασταθεί. Σημαντικό είναι να τονίσουμε ότι όσο αυξάνεται η χωρητικότητα της L2 cache τόσο

αυξάνεται ο χρόνος πρόσβασης σε αυτή. Η αλλαγή

Πίνακας 1: Σχέση μεταξύ CPUs και Cache Size

αυτή φαίνεται στα configurations μας από το *HitDelay* και το *MissDelay* που βρίσκονται στο πεδίο *[L2Cache]* του configuration file μας Για να υπολογίσουμε το χρόνο αυτό χρησιμοποιήσαμε το εργαλείο CACTI (http://quid.hpl.hp.com:9081/cacti/). Σαν input για το Line size δώσαμε το 32 όπως και για το Nr.of Banks και το Technology Node. Τα αποτελέσματα που πήραμε από το εργαλείο CACTI φαίνονται στον πίνακα που ακολουθεί, όπου στην πρώτη γραμμή βλέπουμε την πιθανή χωρητικότητα της L2 cache σε bytes και στη δεύτερη το Access Time που μας δίνει σαν έξοδο το εργαλείο CACTI. Στη συνέχεια βλέπουμε το πηλίκο που παίρνουμε διαιρώντας οποιοδήποτε access time με το access time για χωρητικότητα μνήμης 2097152 bytes. Στη συνέχεια με βάση αυτό το πηλίκο βρίσκουμε τα νέα HitDelay και missDelay, γραμμή τέσσερα και πέντε, για τις υπόλοιπες χωρητικότητες. Στο σημείο αυτό πρέπει να πούμε ότι ο προσομοιωτής SESC δεν δέχεται δεκαδικές τιμές στα hitDelay και missDelay και αναγκαστήκαμε να κάνουμε στρογγυλοποίηση σε ακέραιο.

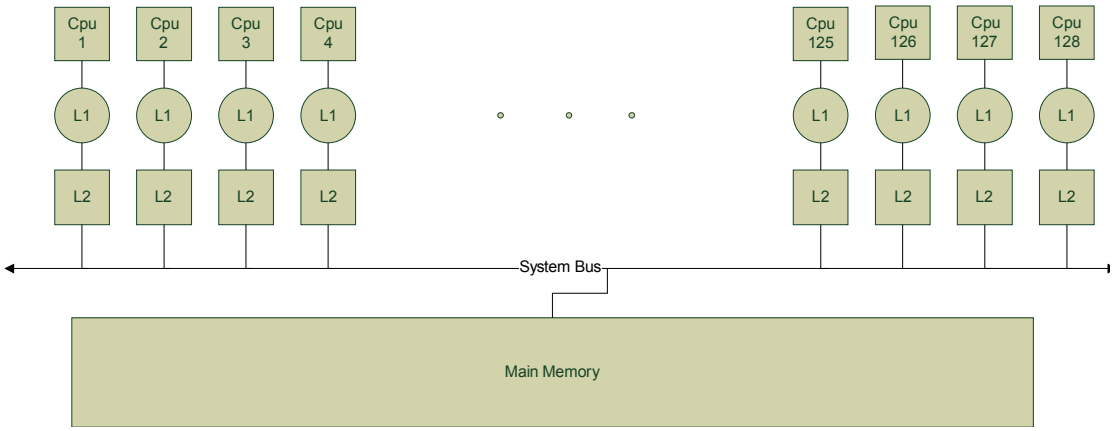
Cache size (B)	2097152	4194304	8388608	16777216	33554432	67108864	134217728	268435456
latency/ access time (ns)	3	3,2575	3,7205	4,6989	6,2538	8,8526	12,007	17,2977
νέο/ αρχικό		1,1095	1,2672	1,6004	2,13	3,0151	4,0895	5,8915
hitDelay	9	9,9855	11,4048	14,4036	19,17	27,1359	36,8055	53,0235
missDelay	11	12,2045	13,9392	17,6044	23,43	33,1661	44,9845	64,8065

Πίνακας 2: Συσχέτιση Cache size με access time και τέλος με hit και miss Delay

Στη συνέχεια παρουσιάζουμε κάθε ένα από τα configurations μας ξεχωριστά για την περίπτωση που έχουμε 128 επεξεργαστές.

3.3.2 Configuration 1

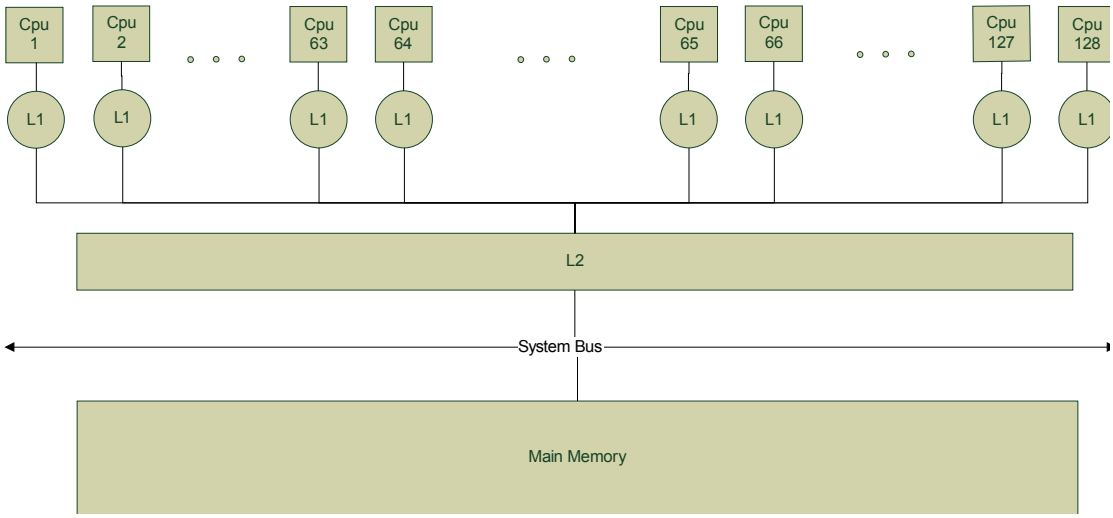
Στο configuration αυτό (*εικόνα 11*), κάθε επεξεργαστής έχει ξεχωριστή μνήμη cache επιπέδου 1 και 2. Αναλυτικότερα έχει μνήμη Cache L2 χωρητικότητας 2MB ενώ το associativity έχει οριστεί στο 8. Είναι το βασικό μας Configuration και στην ουσία με βάση αυτό κάνουμε τις συγκρίσεις μας. Το hitDelay και το missDelay έχει οριστεί στο 9 και 11 αντίστοιχα, ενώ το System Bus διαμοιράζεται από 128 συνδέσεις όσοι είναι και οι επεξεργαστές.



Εικόνα 11:Configuration 1

3.3.3 Configuration 2

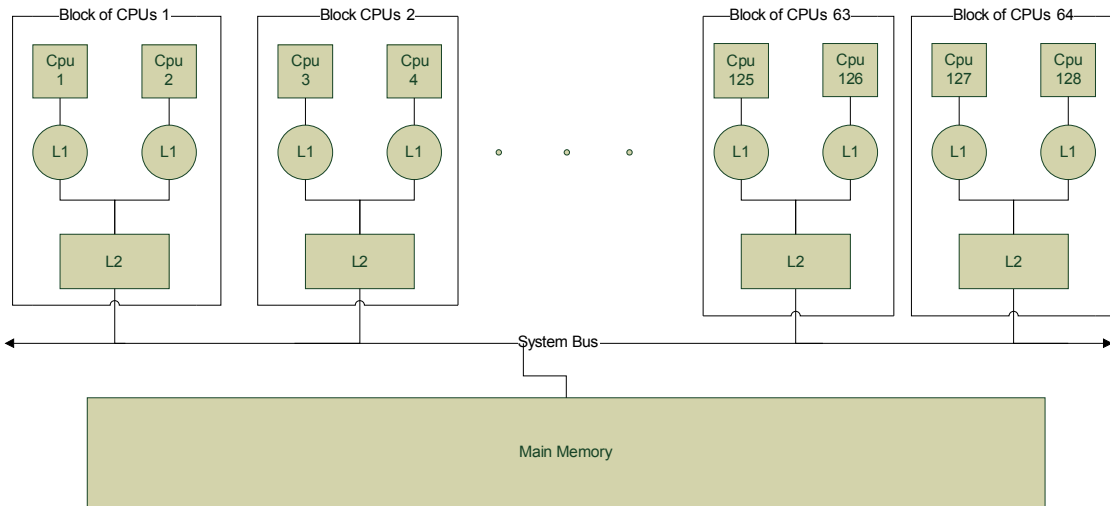
Στο configuration αυτό (*εικόνα 12*) όλοι οι επεξεργαστές έχουν κοινή διαμοιραζόμενη L2 cache. Προς το System Bus πάει μόνο μια σύνδεση αυτή της κοινής για όλους τους επεξεργαστές L2. Η χωρητικότητα της cache μεταβάλλεται ανάλογα τον αριθμό επεξεργαστών που έχουμε. Στην περίπτωση των 128 επεξεργαστών η μνήμη cache L2 έχει χωρητικότητα 256 MB. Όπως είναι φυσιολογικό το Latency για μια τόσο μεγάλη μνήμη cache έχει ανέβει σημαντικά και το hitDelay είναι ορισμένο στα 53 ενώ το missDelay στα 65.



Εικόνα 12:Configuration 2

3.3.4 Configuration 3

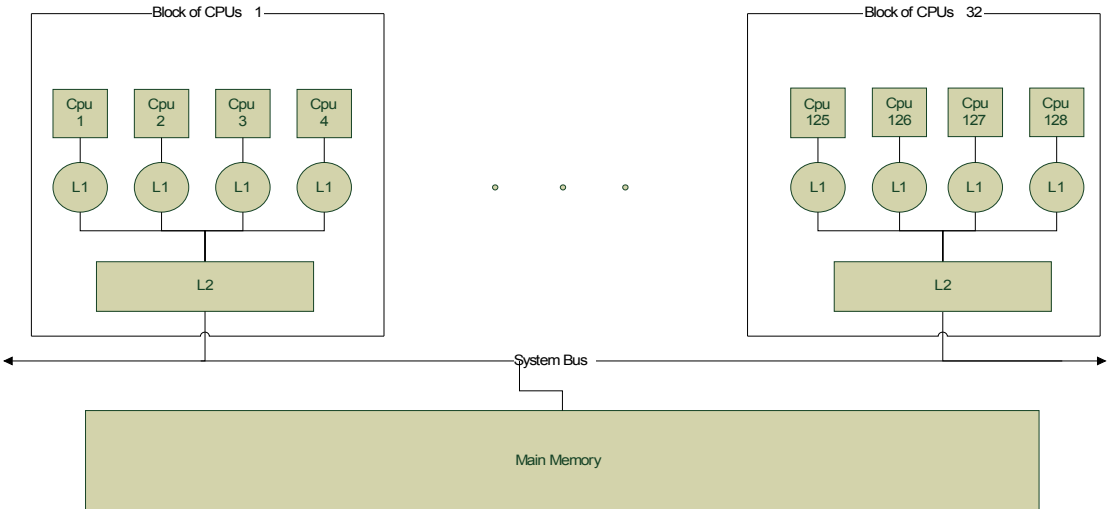
Σε αυτό το Configuration (*εικόνα 13*) οι επεξεργαστές ανά δυο μοιράζονται την ίδια μνήμη cache επιπέδου 2. Η χωρητικότητα της κάθε μνήμης L2 cache είναι στα 4MB ενώ τα Latencies για το HitDelay και το MissDelay έχουν οριστεί στα 10 και 12 αντίστοιχα. Το System Bus διαμοιράζεται από 64 blocks που αποτελούνται από μία L2 cache και 2 επεξεργαστές και L1 cache η κάθε μια.



Εικόνα 13:Configuration 3

3.3.5 Configuration 4

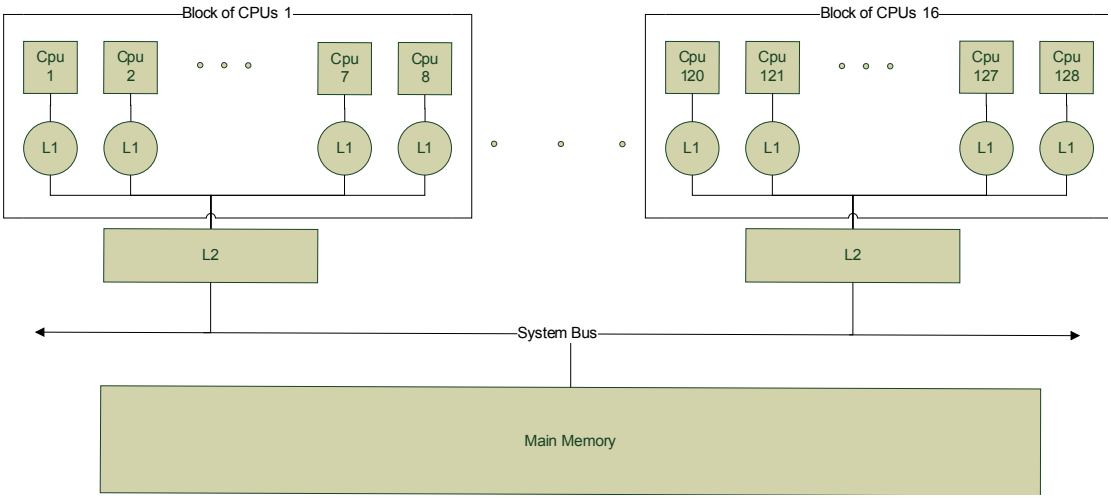
Σε αυτό το configuration (*εικόνα 14*) έχουμε κοινή μνήμη L2 cache για κάθε block τεσσάρων επεξεργαστών. Η κοινή αυτή μνήμη έχει χωρητικότητα 8 MB ενώ το hitDelay και το missDelay έχουν οριστεί στο 11 και 14 αντίστοιχα. Στο σημείο αυτό πρέπει να πούμε ότι το System Bus διαμοιράζεται από 32 blocks των 4 επεξεργαστών όταν έχουμε στη δοκιμή μας 128 CPUs.



Εικόνα 14:Configuration 4

3.3.6 Configuration 5

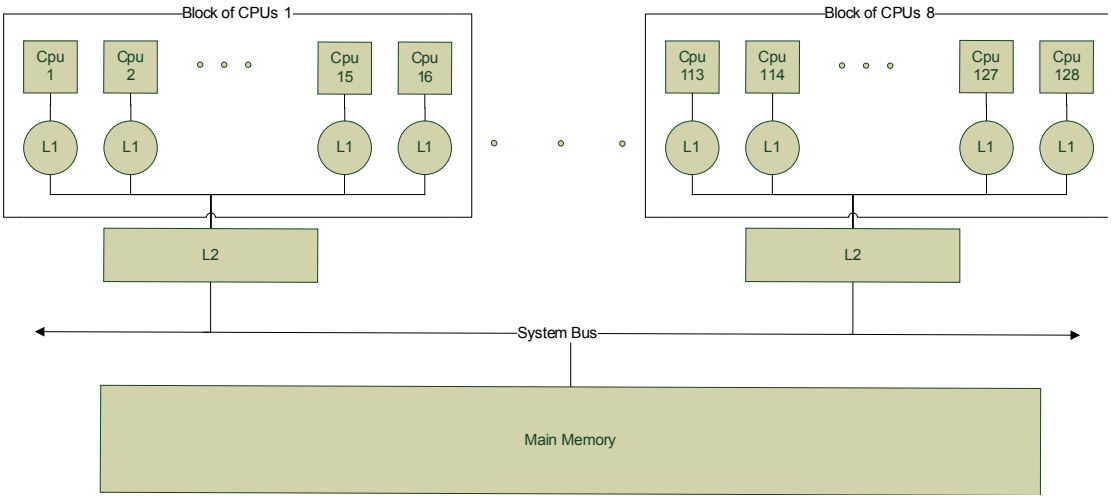
Σε αυτό το Configuration (*εικόνα 15*) οι επεξεργαστές ανά οχτώ μοιράζονται την ίδια μνήμη cache επιπέδου 2. Η χωρητικότητα της κάθε μνήμης L2 cache είναι στα 16MB ενώ τα Latencies για το HitDelay και το MissDelay έχουν οριστεί στα 14 και 17 αντίστοιχα. Το System Bus διαμοιράζεται από 16 blocks που αποτελούνται από μία L2 cache και 8 επεξεργαστές και L1 cache η κάθε μια.



Εικόνα 15:Configuration 5

3.3.7 Configuration 6

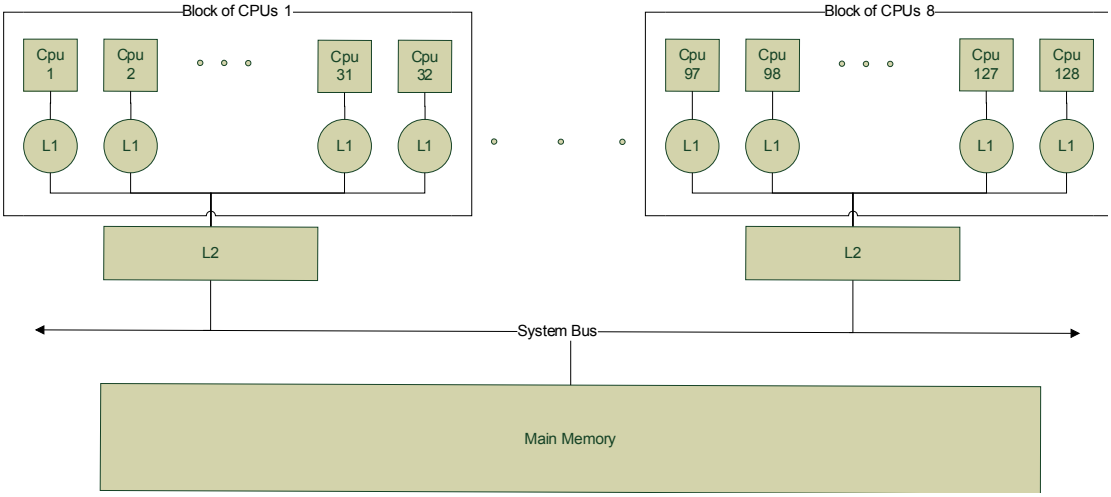
Σε αυτό το configuration (*εικόνα 16*) η μνήμη L2 cache διαμοιράζεται από blocks των 16 επεξεργαστών. Για την δοκιμή με τους 128 επεξεργαστές έχουμε 8 blocks, ενώ η χωρητικότητα της κάθε κρυφής μνήμης επιπέδου δυο είναι 32 MB. Λόγω της αλλαγής της χωρητικότητας της μνήμης έχουν αλλάξει και τα missDelay και hitDelay και έχουν γίνει 19 και 23 αντίστοιχα.



Εικόνα 16:Configuration 6

3.3.8 Configuration 7

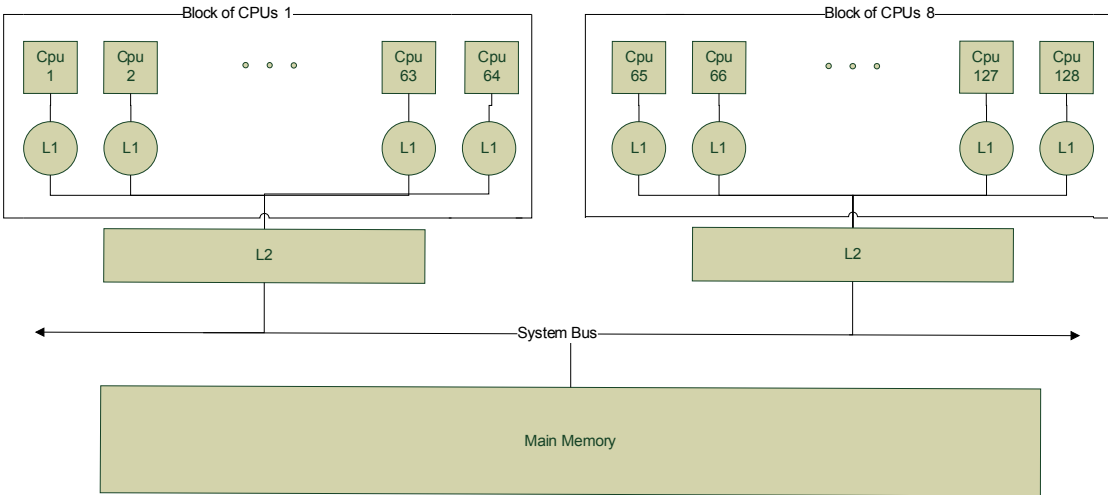
Σε αυτό το configuration (*εικόνα 17*) η μνήμη L2 cache διαμοιράζεται από blocks των 32 επεξεργαστών. Για την δοκιμή με τους 128 επεξεργαστές έχουμε 4 blocks, ενώ η χωρητικότητα της κάθε κρυφής μνήμης επιπέδου δυο είναι 64 MB. Λόγω της αλλαγής της χωρητικότητας της μνήμης έχουν αλλάξει και τα missDelay και hitDelay και έχουν γίνει 33 και 27 αντίστοιχα.



Εικόνα 17:Configuration 7

3.3.9 Configuration 8

Σε αυτό το configuration (*εικόνα 18*) η μνήμη L2 cache διαμοιράζεται από blocks των 64 επεξεργαστών. Για την δοκιμή με τους 128 επεξεργαστές έχουμε 2 blocks, ενώ η χωρητικότητα της κάθε κρυφής μνήμης επιπέδου δυο είναι 128 MB. Λόγω της αλλαγής της χωρητικότητας της μνήμης έχουν αλλάξει και τα missDelay και hitDelay και έχουν γίνει 45 και 37 αντίστοιχα.



Εικόνα 18:Configuration 8

Κεφάλαιο 4

Πειραματικά Αποτελέσματα

4.1 Πληροφορίες για τα πειραματικά αποτελέσματα 33
4.2 Benchmark LU 34
4.3 Benchmark Radix 42
4.4 Benchmark FFT 49
4.5 Benchmark Cholesky 58

4.1 Πληροφορίες για τα πειραματικά αποτελέσματα.

Στη συνέχεια θα σας παρουσιάσουμε τα πειραματικά αποτελέσματα για τα benchmarks της σουίτας εφαρμογών SPLASH 2, LU, Radix, FFT και Cholesky. Για κάθε μια από αυτές τις εφαρμογές θα σας παρουσιάσουμε γραφικές παραστάσεις που θα δείχνουν τον χρόνο εκτέλεσης για κάθε εφαρμογή το miss rate για την μνήμη L1 και L2 cache, καθώς και το speedup για κάθε εφαρμογή. Ο τύπος με τον οποίο υπολογίσαμε το miss rate [8] για την μνήμη L1 είναι ο εξής **:**

L1_miss_rate= (L1_read_miss + L1_write_miss) / (L1_read_miss + L1_write_miss + L1_write_hit + L1_read_hit).

Αντίστοιχα ο τύπος παίρνει σαν εισόδους τα hits/misses για το κομμάτι των εντολών ή των δεδομένων για την L1 cache. Για την μνήμη L2 cache χρησιμοποιήσαμε τον τύπο που ακολουθεί.

L2_miss_rate= (L2_read_miss + L2_write_miss) / (L2_read_miss + L2_write_miss + L2_write_hit + L2_read_hit).

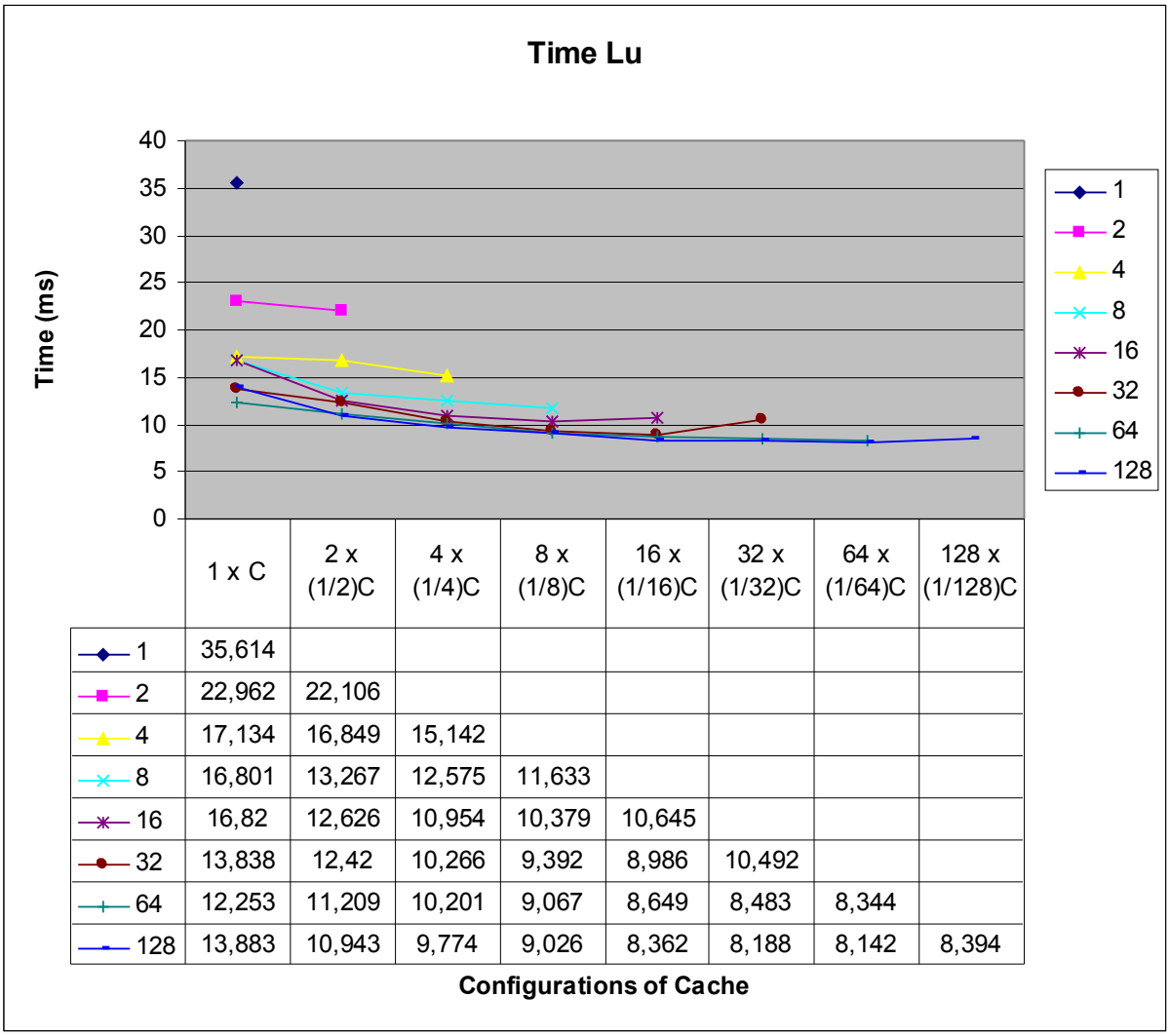
Για τον υπολογισμό του Speedup [8] τον τύπο που ακολουθεί.

$$S_p = \frac{T_1}{T_p}$$

Όπου p είναι ο αριθμός των επεξεργαστών, T_1 ο χρόνος εκτέλεσης του σειριακού προγράμματος και T_P χρόνος εκτέλεσης του παράλληλου προγράμματος για p CPUs.

4.2 Benchmark LU

Το LU benchmark παραγοντοποιεί ένα πυκνό πίνακα στο γινόμενο ενός άνω τριγωνικού και ενός κάτω τριγωνικού πίνακα. Το kernel λειτουργεί σύμφωνα με τον αλγόριθμο των S.C. Woo, J.P Singh, και J.L. Hennessy που περιγράφεται στο άρθρο *The Performance Advantages of Integrating Block Data Transfer in Cache-Coherent Multiprocessors*. Στο πείραμα μας ο πίνακας είναι 512 x 512 ενώ χρησιμοποιούμε block μεγέθους B=16.



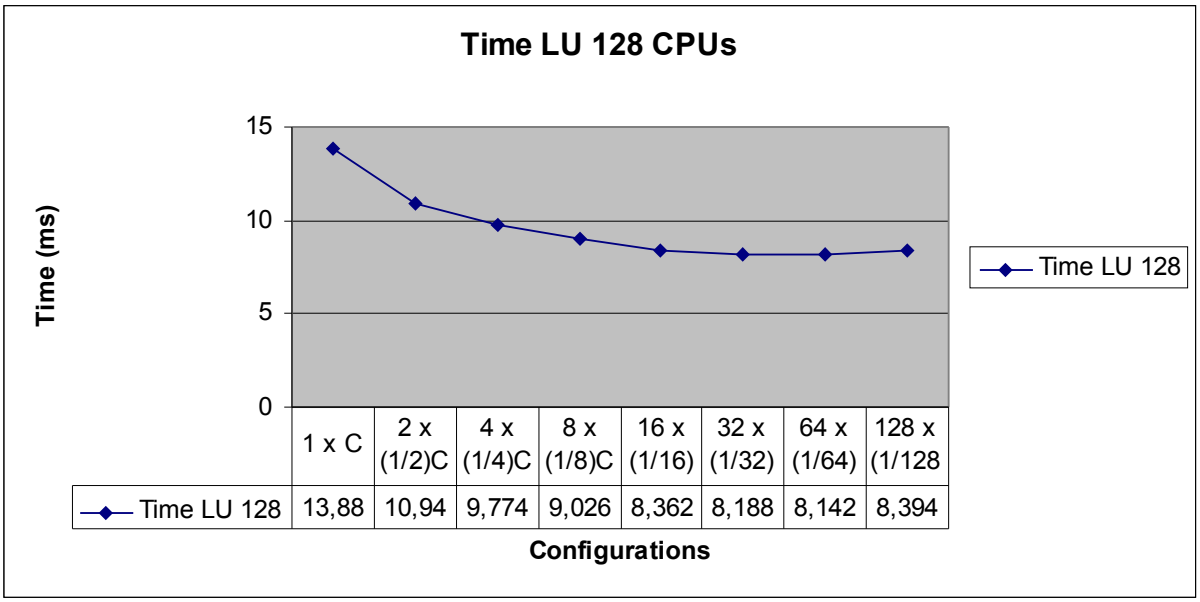
	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	35,614							
2	22,962	22,106						
4	17,134	16,849	15,142					
8	16,801	13,267	12,575	11,633				
16	16,82	12,626	10,954	10,379	10,645			
32	13,838	12,42	10,266	9,392	8,986	10,492		
64	12,253	11,209	10,201	9,067	8,649	8,483	8,344	
128	13,883	10,943	9,774	9,026	8,362	8,188	8,142	8,394

Εικόνα 19: Χρόνος εκτέλεσης για το benchmark LU

Σε αυτή την γραφική παράσταση (*εικόνα 19*) βλέπουμε τον χρόνο που χρειάστηκε για να εκτελεστεί η εφαρμογή LU για τα διάφορα Configurations, αλλά και για τους διαφορετικούς αριθμούς επεξεργαστών. Όπως περιμέναμε, λόγω του παραλληλισμού που προσφέρει η

συγκεκριμένη εφαρμογή, αλλά και τα υπόλοιπα benchmarks του SPLASH2, όσο μεγάλωνε ο αριθμός των επεξεργαστών που επιλέγαμε για να εκτελέσουμε το πρόγραμμα τόσο καλύτερο χρόνο εκτέλεσης είχαμε. Παρατηρούμε ότι τον καλύτερο χρόνο για την εφαρμογή (8,142ms) την έχουμε όταν χρησιμοποιήσουμε 128 επεξεργαστές και το configuration μας διαμοιράζει την μνήμη L2 cache ανά 2 CPUs. Αντίστοιχα για όταν τρέχουμε την ίδια εφαρμογή με 64 επεξεργαστές έχουμε τον καλύτερο χρόνο (8,344) όταν η μνήμη L2 cache είναι ξεχωριστή για κάθε επεξεργαστή. Επίσης βλέπουμε ότι τρέχοντας την εφαρμογή για 16 και 32 επεξεργαστές καλύτερο χρόνο είχαμε όταν η κρυφή μνήμη επιπέδου δυο ήταν κοινή ανά 2 και 4 επεξεργαστές αντίστοιχα. Τρέχοντας το benchmark LU για 64 CPUs (128 MB συνολικής μνήμης L2 cache) είχαμε μικρούς χρόνους εκτέλεσης όταν η μνήμη L2 διαμοιραζόταν μεταξύ 2 (8,483 ms) και 4 (8,694 ms) επεξεργαστών ενώ τον καλύτερο χρόνο για 64 CPUs τον παρατηρήσαμε όταν η μνήμη L2 cache ήταν ξεχωριστή για κάθε CPU (8,344 ms).Όταν χρησιμοποιήσαμε 32 CPUs (64 MB συνολικής μνήμης L2 cache) για να εκτελέσουμε την εφαρμογή μας τον καλύτερο χρόνο τον είχαμε για το configuration όπου η κρυφή μνήμη επιπέδου δυο, ήταν κοινή ανά δυο επεξεργαστές (8,986 ms), ενώ επίσης αρκετά καλό χρόνο είχαμε όταν διαμοιράσαμε την μνήμη ανά τέσσερις επεξεργαστές (9,392 ms). Στα υπόλοιπα configuration οι χρόνοι εκτέλεσης απείχαν τουλάχιστον κατά 0,7ms. Εκτελώντας το πρόγραμμα μας για 16 επεξεργαστές είχαμε την καλύτερη επίδοση όταν η κρυφή μνήμη L2 ήταν κοινή ανά δυο επεξεργαστές (10,379 ms) και ενώ η συνολική μνήμη για την L2 ήταν 32 MB. Τέλος για τα μπορούμε να δούμε και στην γραφική μας ότι εκτελώντας το benchmark LU για 8, 4, και 2 επεξεργαστές είχαμε πάντα τον μικρότερο χρόνο εκτέλεσης για το configuration όπου η μνήμη L2 ήταν ξεχωριστή για κάθε ένα από τα CPUs. Σε αυτές τις δοκιμές η συνολική χωρητικότητα της μνήμης L2 ήταν 16 MB , 8 MB και 4 MB για τους 8 , 4 και 2 επεξεργαστές αντίστοιχα. Σημαντικό θα ήταν να τονίσουμε ότι σε όλες τις δοκιμές μας το configuration όπου η μνήμη L2 cache είναι κοινή για όλα τα CPUs μας έδωσε τον χειρότερο χρόνο εκτέλεσης, κάτι που οφείλεται στο ότι μεγαλώνοντας την χωρητικότητα της μνήμης μεγαλώνει και το access time προς αυτή.

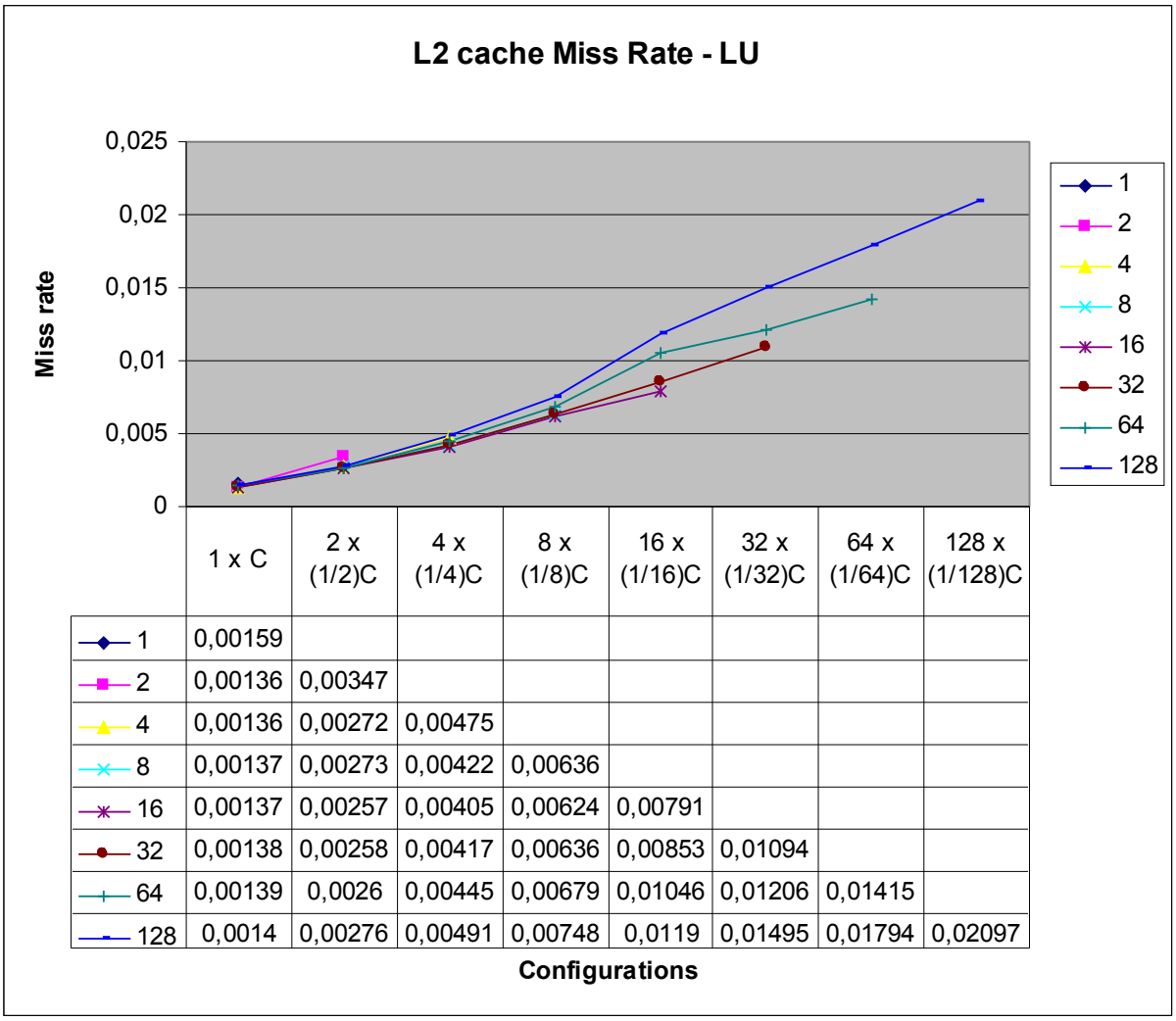
Στη γραφική παράσταση που ακολουθεί (*εικόνα 20*) βλέπουμε τον χρόνο εκτέλεσης των διαφόρων configurations μας για το σενάριο όπου εκτελούμε την εφαρμογή μας με 128 CPUs.Όπως είδαμε και στην γραφική παράσταση που μας δείχνει συνολικά τους χρόνους εκτέλεσης το σενάριο αυτό μας δίνει τον μικρότερο χρόνο για το συγκεκριμένο benchmark.



Εικόνα 20: Χρόνος εκτέλεσης για το benchmark LU για 128CPUs

Αναλυτικότερα, από την γραφική παράσταση βλέπουμε ότι όταν τρέχουμε το benchmark με 128 επεξεργαστές, με την συνολική χωρητικότητα της L2 cache να φτάνει τα 256MB , τα configurations που μας δίνουν του μικρότερους χρόνους εκτέλεσης είναι αυτά που η μνήμη L2 διαμοιράζεται ανά 2 (8,142 ms) , 4 (8,188 ms) και 8 (8,362 ms) CPUs.Παρατηρούμε επίσης ότι για τρέχοντας το πρόγραμμα μας για 128 επεξεργαστές όσο αυξάνουμε τους επεξεργαστές που μοιράζονται την ίδια L2 cache τόσο αυξάνεται ο χρόνος εκτέλεσης. Η αύξηση του execution time οφείλεται στην αύξηση του access time προς την μνήμη L2 καθώς μεγαλώνει η χωρητικότητα της. Περιγράφουμε αναλυτικά το πώς αυξάνεται το access time προς την L2 cache μνήμη στο κεφάλαιο όπου περιγράφουμε τα configurations και της συνθήκες που εκτελέσαμε τα πειράματα μας.

Στη συνέχεια βλέπουμε την γραφική παράσταση για το Miss Rate της L2 cache (*εικόνα 21)* και πως αυτό μεταβάλλεται καθώς εκτελούμε την εφαρμογή με διαφορετικά configurations αλλά και διαφορετικό αριθμό επεξεργαστών. Όπως περιγράψαμε και σε άλλο κεφάλαιο της διπλωματικής μας καθώς αυξάνεται ο αριθμός των επεξεργαστών αυξάνουμε ανάλογα και την συνολική χωρητικότητα της L2 cache που χρησιμοποιούμε για τα simulation μας. Με αυτό τον τρόπο το configuration με 128 επεξεργαστές έχει 128 φορές περισσότερη μνήμη από το πείραμα για ένα επεξεργαστή.



	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	0,00159							
2	0,00136	0,00347						
4	0,00136	0,00272	0,00475					
8	0,00137	0,00273	0,00422	0,00636				
16	0,00137	0,00257	0,00405	0,00624	0,00791			
32	0,00138	0,00258	0,00417	0,00636	0,00853	0,01094		
64	0,00139	0,0026	0,00445	0,00679	0,01046	0,01206	0,01415	
128	0,0014	0,00276	0,00491	0,00748	0,0119	0,01495	0,01794	0,02097

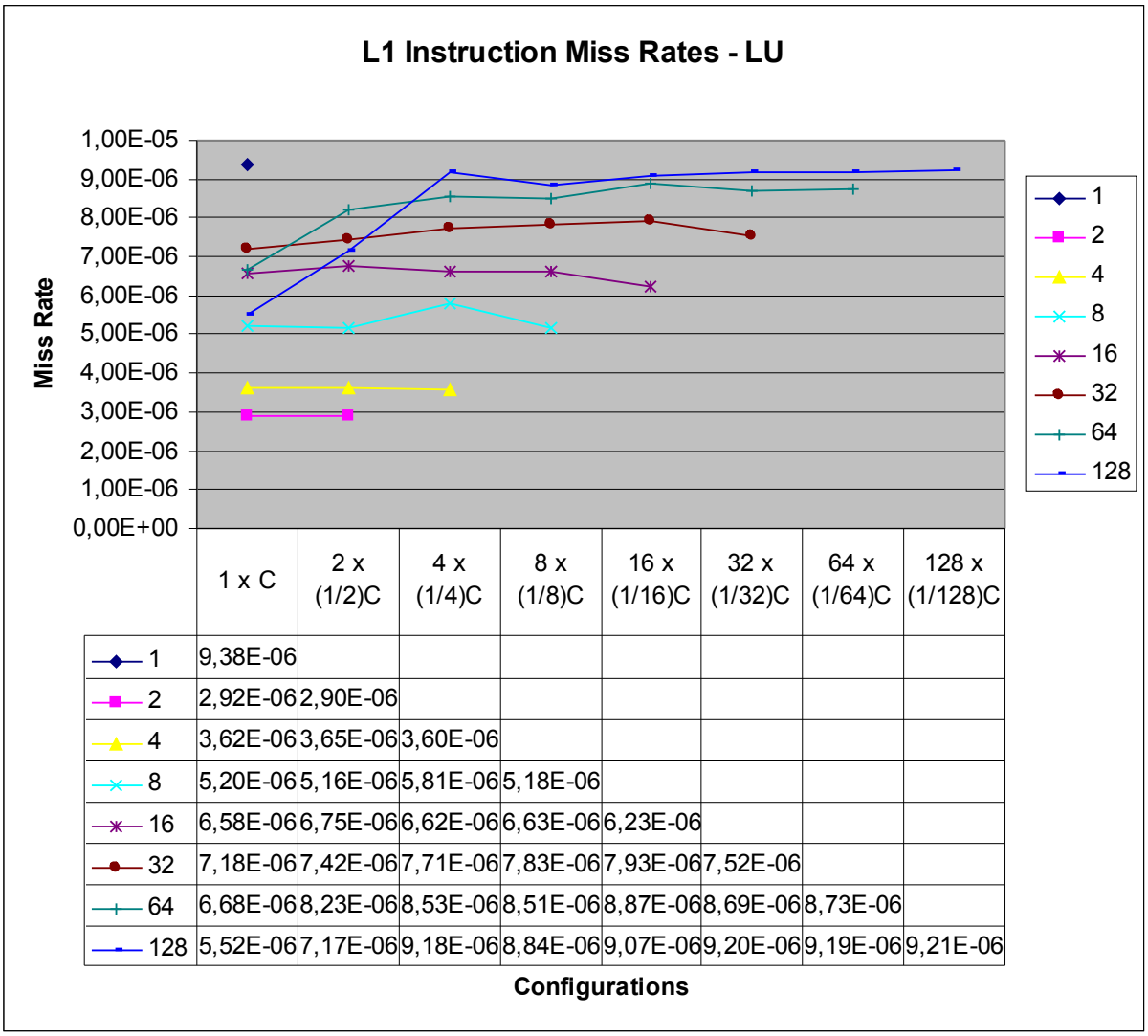
Εικόνα 21:L2 cache Miss Rate για το benchmark LU.

Όπως παρατηρούμε το μικρότερο Miss Rate υπάρχει στα configurations όπου η κρυφή μνήμη επιπέδου 2 διαμοιράζεται από πολλούς επεξεργαστές. Εξαρχής γνωρίζαμε ότι για την ίδια ποσότητα μνήμης θα είχαμε μικρότερο miss rate αν την χρησιμοποιούσαν όλοι μαζί οι επεξεργαστές παρά αν την διαιρούσαμε στον αριθμό τους. Αυτό συμβαίνει διότι σε μία μεγαλύτερη cache έχουμε λιγότερα conflict και capacity misses.Επίσης με μια ενιαία cache ελαχιστοποιούμε και τα compulsory misses αφού τα δεδομένα μας χρειάζονται να γίνουν fetch στην μνήμη μόνο μία φορά. Όπως γνωρίζουμε και από την θεωρία στα cache misses και των δυο επιπέδων , L1 και L2, οφείλονται σε μεγάλο μέρος τα stalls του επεξεργαστή που καθυστερούν την εκτέλεση του προγράμματος. Παρατηρούμε επίσης ότι εξαρχής είχαμε χαμηλό αριθμό από misses για μία ενιαία L2 , κάτι που σημαίνει ότι η μνήμη που εξαρχής είχαμε κάλυπτε επαρκώς της ανάγκες της εφαρμογής μας. Παρόλα αυτά όταν αυξήσαμε του επεξεργαστές από ένα σε δυο είδαμε ότι για το ίδιο configuration το miss rate μειώθηκε

πράγμα που σημαίνει ότι με την αύξηση της μνήμης μειώθηκαν κάποια από τα conflict και capacity misses που είχαμε. Μπορούμε επίσης να δούμε ότι περαιτέρω αύξηση της χωρητικότητας της μνήμης δεν μας έδωσε καλύτερα αποτελέσματα στο miss rate. Το γεγονός αυτό μας κάνει να πιστέψουμε ότι τα misses μειώθηκαν στον μεγαλύτερο δυνατό βαθμό. Επίσης βλέπουμε ότι για τα σενάρια όπου η μνήμη L2 cache δεν είναι κοινή για όλα τα CPUs έχουμε αύξηση των Misses, η οποία οφείλεται στα επιπλέον compulsory misses που έχουμε. Τα misses αυτά είναι κυρίως υπεύθυνα για την αύξηση του miss rate.

Στις γραφικές που ακολουθούν βλέπουμε πώς επηρεάζεται το L1 Data Miss Rate και το L1 Instruction Miss Rate καθώς αλλάζουμε τον αριθμό των επεξεργαστών που εκτελούν το πρόγραμμα καθώς και την χωρητικότητα της μνήμης L2 cache.

Στη συνέχεια βλέπουμε την γραφική παράσταση για τα miss rates για την μνήμη L1 Instruction (*εικόνα 22*) .Αρχικά παρατηρούμε ότι το μεγαλύτερο miss rate το έχει το σενάριο όπου εκτελούμε την εφαρμογή μας με ένα μόνο επεξεργαστή, ενώ το μικρότερο miss rate το έχουμε όταν αυξάνουμε τους επεξεργαστές σε δυο. Με βάση την παραπάνω παρατήρηση μπορούμε να βγάλουμε το συμπέρασμα ότι τα 32 KB L1 cache δεν ήταν αρκετά για το πρώτο σενάριο και για αυτό είχαμε αυξημένα misses.Στη συνέχεια όμως με το διπλασιασμό των επεξεργαστών, αλλά και της συνολικής μνήμης L1 η εφαρμογή μας είχε στη διάθεση της όση μνήμη χρειαζόταν και έτσι μειώσαμε τα misses δραματικά. Όπως μπορούμε να δούμε και από την γραφική παράσταση περεταίρω αύξηση της μνήμης L1 cache για το instruction μέρος δεν μας δίνει καλύτερα αποτελέσματα όσο αφορά το miss rate. Αυτό εξηγείτε αν λάβουμε υπόψη μας ότι αυξάνοντας της L1 αυξάνουμε αναγκαστικά και τα compulsory misses που παίζουν σημαντικό ρόλο, ιδιαίτερα όταν έχουμε αρκετά μεγάλη μνήμη ώστε να μην έχουμε capacity και conflict misses. Για αυτό τον λόγω μετά τους δυο επεξεργαστές έχουμε αύξηση του miss rate με το χειρότερο πολυπύρηνο σενάριο να είναι αυτό με τους 128 επεξεργαστές που όπως είναι λογικό έχει τα περισσότερα compulsory misses.

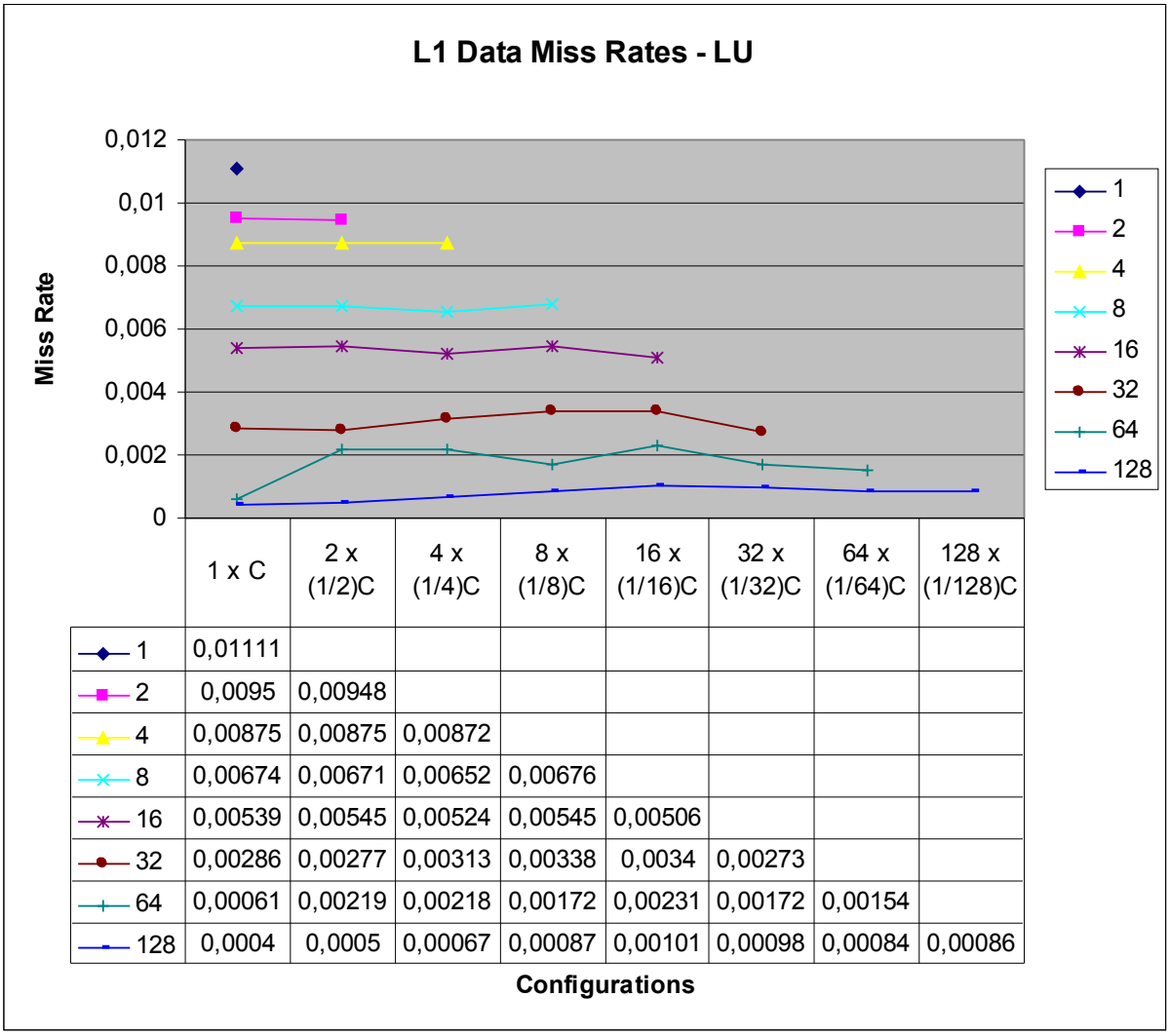


	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	9,38E-06							
2	2,92E-06	2,90E-06						
4	3,62E-06	3,65E-06	3,60E-06					
8	5,20E-06	5,16E-06	5,81E-06	5,18E-06				
16	6,58E-06	6,75E-06	6,62E-06	6,63E-06	6,23E-06			
32	7,18E-06	7,42E-06	7,71E-06	7,83E-06	7,93E-06	7,52E-06		
64	6,68E-06	8,23E-06	8,53E-06	8,51E-06	8,87E-06	8,69E-06	8,73E-06	
128	5,52E-06	7,17E-06	9,18E-06	8,84E-06	9,07E-06	9,20E-06	9,19E-06	9,21E-06

Εικόνα 22:L1 Instruction cache Miss Rate για το benchmark LU.

Παρατηρώντας στη συνέχεια την γραφική με τα miss rates για την μνήμη L1 Data (*εικόνα 23*), βλέπουμε ότι η αύξηση των επεξεργαστών ελαττώνει κατά πολύ το miss rate. Η ελάττωση αυτή μπορεί να έχει πολλές αιτίες. Κατά την γνώμη μας η αύξηση των επεξεργαστών και άρα της συνολικής L1 μνήμης σε συνδυασμό με την σωστή εκμετάλλευση από το πρόγραμμα του temporal locality μας δίνουν αυτά τα αποτελέσματα. Αναλυτικότερα, το kernel LU είναι ειδικά σχεδιασμένο έτσι ώστε να εκμεταλλεύεται το temporal locality μεταξύ των τιμών μικρότερων κομματιών του κάθε πίνακα καθώς κάνει υπολογισμούς για να βρει το γινόμενο των μεγαλύτερων πινάκων. Όσο μεγαλώνει ο αριθμός των επεξεργαστών τόσο μικρότερα κομμάτια πληροφορίας αναλαμβάνει να επεξεργαστεί το κάθε CPU και αυξάνεται επίσης η μνήμη L1 που αντιστοιχεί σε αυτό το κομμάτι πληροφορίας. Με αυτό τον τρόπο χρειαζόμαστε να κάνουμε fetch από την L2 cache λιγότερες πληροφορίες ανά

CPU, και μειώνουμε τα misses. Μπορούμε να παρατηρήσουμε επίσης, ότι για τον ίδιο αριθμό επεξεργαστών δεν έχουμε ιδιαίτερες διαφορές στο miss rate καθώς η χωρητικότητα της L2 cache δεν επηρεάζει την λειτουργία της L1 όσο αφορά τα misses.

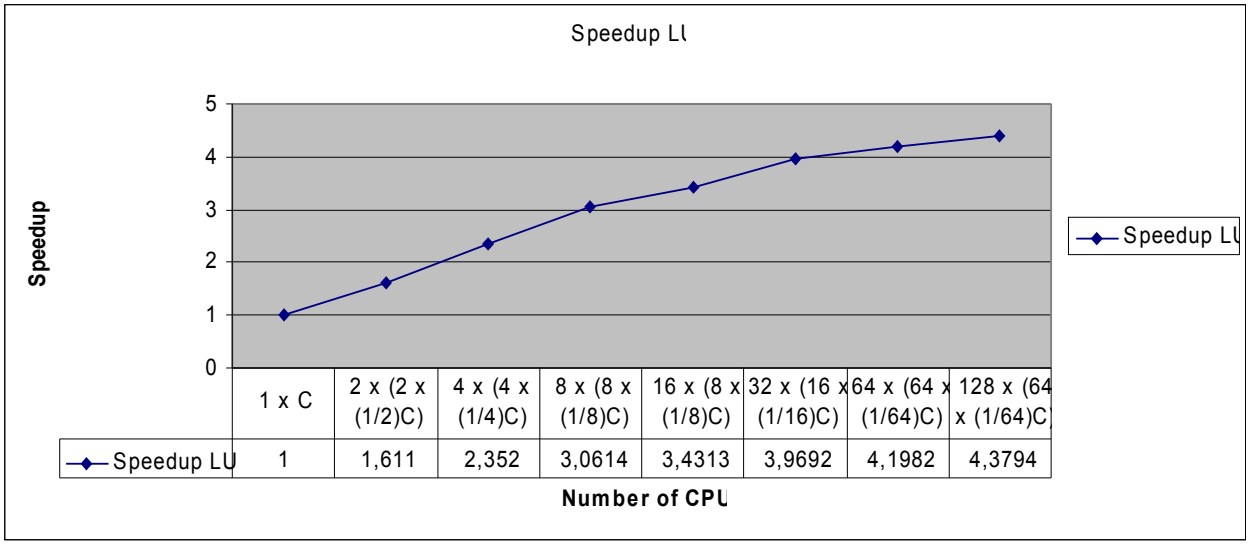


	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	0,01111							
2	0,0095	0,00948						
4	0,00875	0,00875	0,00872					
8	0,00674	0,00671	0,00652	0,00676				
16	0,00539	0,00545	0,00524	0,00545	0,00506			
32	0,00286	0,00277	0,00313	0,00338	0,0034	0,00273		
64	0,00061	0,00219	0,00218	0,00172	0,00231	0,00172	0,00154	
128	0,0004	0,0005	0,00067	0,00087	0,00101	0,00098	0,00084	0,00086

Εικόνα 23:L1 Data cache Miss Rate για το benchmark LU.

Όπως παρατηρήσαμε και σε προηγούμενη γραφική παράσταση όσο αυξάνει ο αριθμός των επεξεργαστών έχουμε μικρότερο χρόνο εκτέλεσης. Η γραφική που ακολουθεί μας δείχνει το speedup (*εικόνα 24)* για το LU benchmark σε συνάρτηση με τον αριθμό των επεξεργαστών που χρησιμοποιούμε για να το εκτελέσουμε. Για να δημιουργήσουμε την γραφική παράσταση για το Speedup χρησιμοποιήσαμε τον μικρότερο χρόνο εκτέλεσης από τα σενάρια που έχουμε για κάθε αριθμό επεξεργαστών. Παρατηρώντας λοιπόν την γραφική μας παράσταση, βλέπουμε ότι όσο μεγαλώνει ο αριθμός των επεξεργαστών που χρησιμοποιούμε

για να εκτελέσουμε το πρόγραμμα τόσο μεγαλύτερο speedup έχουμε, κάτι που είναι λογικό για παράλληλες εφαρμογές. Αναλυτικότερα, βλέπουμε ότι έχουμε την μέγιστη αύξηση στο speedup όταν χρησιμοποιούμε τους περισσότερους δυνατούς επεξεργαστές, δηλαδή 128 CPUs.



	1 x C	2 x (2 x (1/2)C)	4 x (4 x (1/4)C)	8 x (8 x (1/8)C)	16 x (8 x (1/8)C)	32 x (16 x (1/16)C)	64 x (64 x (1/64)C)	128 x (64 x (1/64)C
Speedup LU	1	1,611	2,352	3,0614	3,4313	3,9692	4,1982	4,3794

Εικόνα 24: Speedup για το benchmark LU.

Βλέπουμε επίσης, ότι αν αυξήσουμε των αριθμό των επεξεργαστών από ένα σε οχτώ έχουμε speedup 3 που είναι περίπου το 70 % του συνολικού speedup που μπορούμε να πάρουμε. Το συμφέρον αυτό speedup το παίρνουμε όταν κάθε μνήμη έχει το δική της ξεχωριστή μνήμη cache L2, όπως μπορούμε να δούμε και από τον πίνακα με τους χρόνους. Παρατηρούμε επίσης ότι ο παραλληλισμός της συγκεκριμένης εφαρμογής δεν μας βοήθησε ιδιαίτερα στην επίτευξη ενός πολύ καλύτερου χρόνου εκτέλεσης. Σημαντικό είναι επίσης να πούμε ότι διαμοιράζοντας την μνήμη L2 cache σε πάνω από 2 επεξεργαστές δεν μας έδωσε σε κανένα από τα σενάρια μας βέλτιστο χρόνο εκτέλεσης για τον συγκεκριμένο αριθμό επεξεργαστών, ενώ αντίθετα όταν τρέξαμε την εφαρμογή μας με ξεχωριστή L2 cache για κάθε CPU πήραμε βέλτιστο χρόνο εκτέλεσης για τα πειράματα που χρησιμοποιήσαμε 2,4 και 8 επεξεργαστές. Στα υπόλοιπα σενάρια το configurations που μας έδωσε τον καλύτερο χρόνο ήταν αυτό όπου η μνήμη L2 ήταν κοινή για κάθε δυο επεξεργαστές.

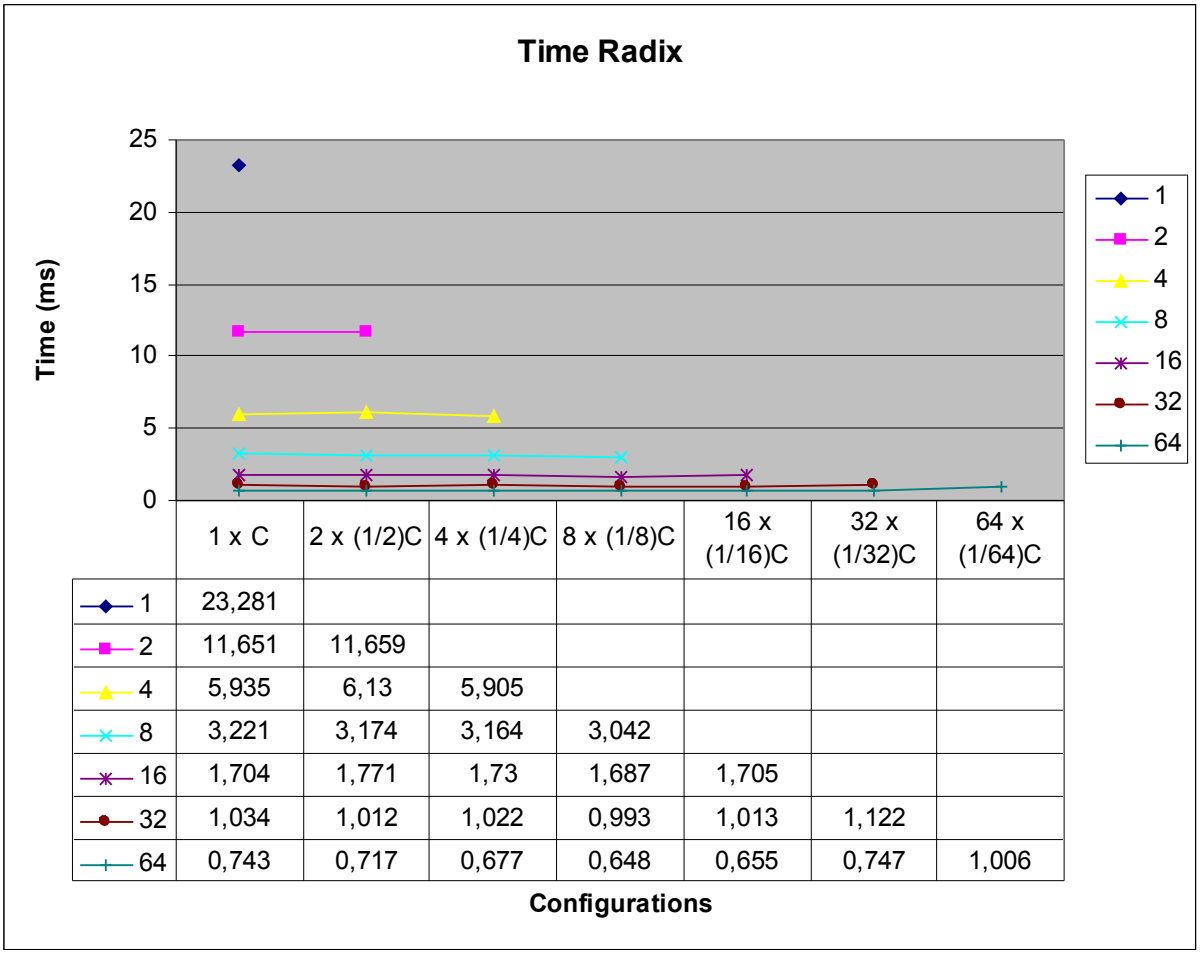
Συμπερασματικά, για αυτό το benchmark βλέπουμε ότι αν αυξήσουμε την μνήμη cache L2 καθώς και τον αριθμό των επεξεργαστών που εκτελούν το πρόγραμμα μπορούμε να βελτιώσουμε τον χρόνο εκτέλεσης. Επιπλέον, όσο περισσότεροι επεξεργαστές

διαμοιράζονται την μνήμη L2 cache τόσο μειώνεται το miss rate της μνήμης κάτι που περιμέναμε να μας δώσει καλύτερο χρόνο εκτέλεσης για το πρόγραμμα μας, κάτι που όμως δεν συμβαίνει γιατί αυξάνοντας την χωρητικότητα της μνήμης αυξάνουμε και το access time προς αυτή. Βλέποντας τα αποτελέσματα μας καταλαβαίνουμε ότι η αύξηση αυτή του access time είναι καταλυτική στην αύξηση του χρόνου εκτέλεσης. Με βάση τα παραπάνω μας φαίνεται φυσιολογικό που τα configurations που μας δίνουν τις καλύτερες επιδόσεις είναι αυτά όπου η μνήμη cache μοιράζεται ανά δυο επεξεργαστές ή όταν κάθε CPU έχει την δική του ξεχωριστή L2. Σε αυτά τα configuration η μνήμη είναι αρκετή ώστε να έχουμε αρκετά χαμηλό miss rate και ταυτόχρονα να έχουμε μικρό delay από το access time.

4.3 Benchmark Radix

Σε αυτό το κομμάτι της διπλωματικής μας εργασίας μας παρουσιάζουμε τα αποτελέσματα για το benchmark Radix. Σε αυτό το benchmark προσομοιώνουμε την ταξινόμηση ακεραίων αριθμών σύμφωνα με τον αλγόριθμο που περιγράφεται από τον G. E. Blelloch, στο άρθρο A Comparison of Sorting Algorithms for the Connection Machine CM-2.

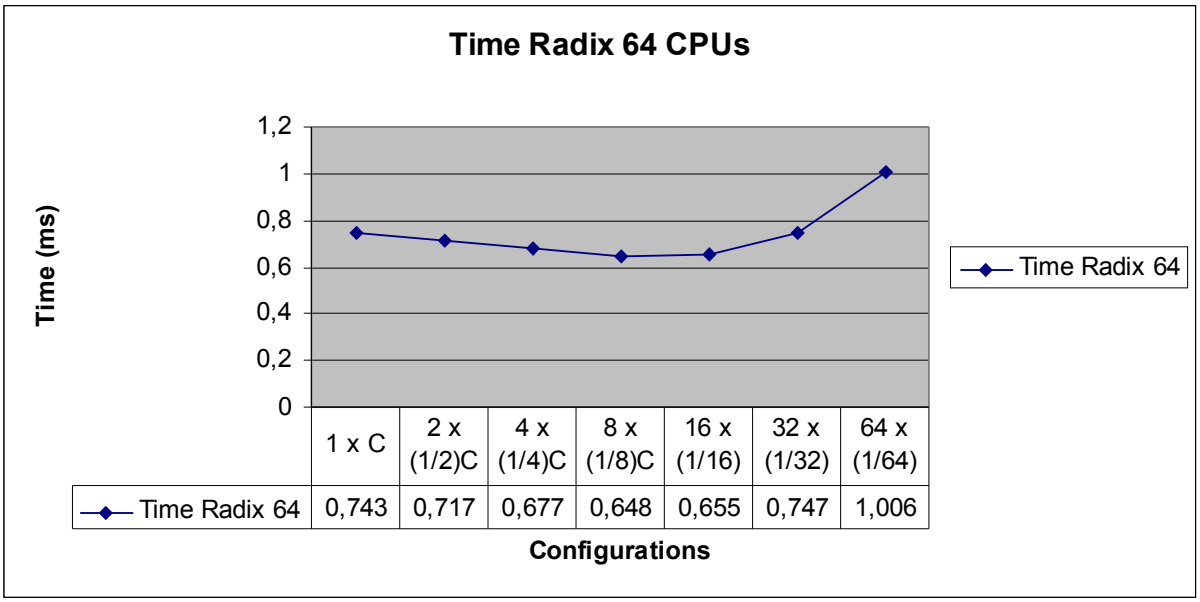
Στη γραφική παράσταση που ακολουθεί βλέπουμε τον χρόνο εκτέλεσης για το benchmark Radix (*εικόνα 25)* για διαφορετικό αριθμό επεξεργαστών αλλά και για διαφορετικά configuration της μνήμης L2 cache.



	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C
1	23,281						
2	11,651	11,659					
4	5,935	6,13	5,905				
8	3,221	3,174	3,164	3,042			
16	1,704	1,771	1,73	1,687	1,705		
32	1,034	1,012	1,022	0,993	1,013	1,122	
64	0,743	0,717	0,677	0,648	0,655	0,747	1,006

Εικόνα 25: Χρόνος εκτέλεσης για το benchmark Radix.

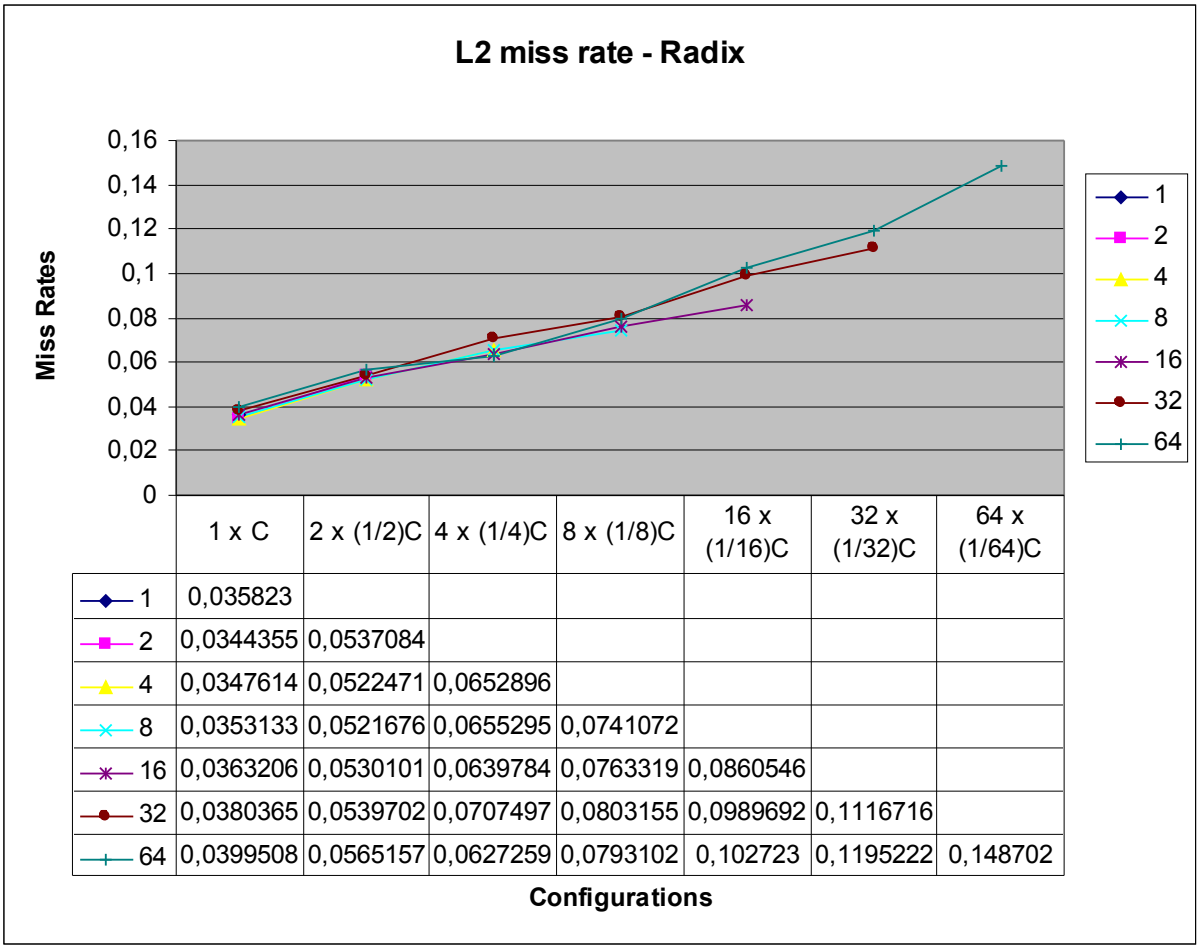
Όπως μπορούμε να δούμε όσο αυξάνεται ο αριθμός των επεξεργαστών τόσο μικρότερο χρόνο εκτέλεσης έχουμε. Παρατηρούμε από τον πίνακα δεδομένων της γραφικής μας παράστασης ο μικρότερος χρόνος εκτέλεσης (0,648 ms) για την δοκιμή μας παρατηρείται όταν χρησιμοποιούμε για την εκτέλεση του benchmark, 64 επεξεργαστές με την μνήμη cache επιπέδου 2 να είναι διαμοιρασμένη ανά 8 CPUs.



Εικόνα 26: Χρόνος εκτέλεσης για το benchmark Radix για 128CPUs.

Αναλυτικότερα, όταν χρησιμοποιούμε 64 επεξεργαστές για την εκτέλεση του προγράμματος μας η συνολική μνήμη cache L2 που χρησιμοποιείται έχει χωρητικότητα 128MB. Βλέποντας τους χρόνους για την εφαρμογή μας (*εικόνα 26)* όταν την εκτελούμε για 64 CPUs παρατηρούμε ότι η κοινή χρήση της μνήμης κρυφού επιπέδου δυο, βοηθάει στην μείωση του χρόνου εκτέλεσης. Ο μικρότερος χρόνος (0,648 ms) επιτυγχάνεται όταν η μνήμη L2 διαμοιράζεται μεταξύ 8 επεξεργαστών, ενώ κοντινούς στον καλύτερο χρόνο εκτέλεσης χρόνους, πετυχαίνουμε διαμοιράζοντας την μνήμη cache L2 ανά τέσσερις (0,655ms) και δεκαέξι (0,677 ms) επεξεργαστές. Παρατηρώντας επίσης τα αποτελέσματα μας για την εφαρμογή μας όταν χρησιμοποιούμε 32 επεξεργαστές, βλέπουμε ότι έχουμε τον μικρότερο χρόνο εκτέλεσης για το configuration όπου η μνήμη cache είναι κοινή για κάθε τέσσερις επεξεργαστές (0,993 ms). Βλέπουμε επιπλέον, ότι τα configurations όπου η μνήμη διαμοιράζεται ανά 2 και 16 επεξεργαστές οι χρόνοι εκτέλεσης είναι αρκετά κοντά με τον μικρότερο,1,013ms και 1,012ms αντίστοιχα. Επίσης παρατηρούμε ότι τρέχοντας το benchmark μας για 16 CPUs έχουμε τον καλύτερο χρόνο εκτέλεσης για το configuration όπου η μνήμη L2 cache διαμοιράζεται ανά δυο επεξεργαστές (1,687 ms) ενώ και τα υπόλοιπα configurations μας δίνουν χρόνους εκτέλεσης που δεν απέχουν ιδιαίτερα από τον μικρότερο. Στις δοκιμή μας για 8 και 4 επεξεργαστές μπορούμε να δούμε ότι το configuration που μας δίνει τον καλύτερο χρόνο είναι αυτό όπου η μνήμη L2 cache είναι ξεχωριστή για κάθε επεξεργαστή (3,042 ms και 5,905 ms αντίστοιχα). Παρατηρώντας γενικά τους χρόνους εκτέλεσης βλέπουμε ότι για τον ίδιο αριθμό επεξεργαστών τα διάφορα

configurations της μνήμης L2 cache δεν μας δίνουν χρόνους εκτέλεσης όπου οι τιμές τους διαφέρουν σε σημαντικό βαθμό.



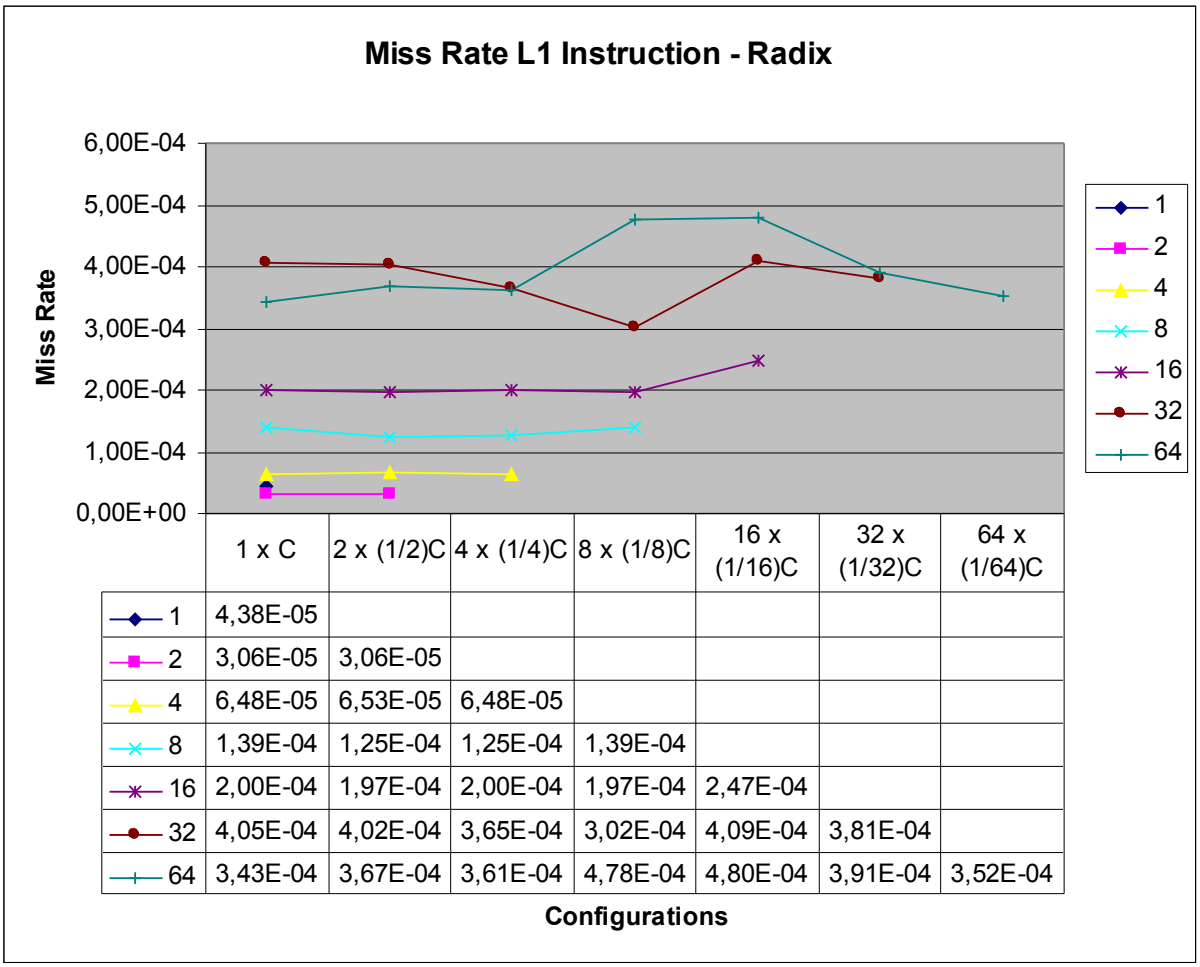
	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C
1	0,035823						
2	0,0344355	0,0537084					
4	0,0347614	0,0522471	0,0652896				
8	0,0353133	0,0521676	0,0655295	0,0741072			
16	0,0363206	0,0530101	0,0639784	0,0763319	0,0860546		
32	0,0380365	0,0539702	0,0707497	0,0803155	0,0989692	0,1116716	
64	0,0399508	0,0565157	0,0627259	0,0793102	0,102723	0,1195222	0,148702

Εικόνα 27:L2 cache Miss Rate για το benchmark Radix.

Στη συνέχεια θα δούμε την γραφική παράσταση για τα miss rates που εμφανίζονται στην μνήμη L2 cache (*εικόνα 27)* σε συνδυασμό με τα configurations και τον αριθμό των επεξεργαστών που χρησιμοποιούμε. Όπως περιμέναμε έχουμε τον μικρότερο αριθμό miss rates για το configuration όπου η μνήμη cache διαμοιράζεται σε όλους τους επεξεργαστές. Παρατηρούμε επίσης ότι έχουμε το χαμηλότερο δυνατό miss rate για το configuration όπου η μνήμη L2 είναι κοινή για όλους τους επεξεργαστές ανεξάρτητα από τον αριθμό των επεξεργαστών που χρησιμοποιούμε. Αυτό σημαίνει ότι η cache των 2MB που έχουμε αρχικά επαρκεί για την εφαρμογή μας, έχουμε περιορισμένα conflict και capacity misses, τα οποία δεν μειώνονται περεταίρω με την αύξηση της μνήμης, και κυρίως τα misses που έχουμε είναι τα compulsory misses, όπου φέρνουμε τα δεδομένα που χρειαζόμαστε στην L2 cache

μνήμη από το main memory. Το συμπέρασμα μας αυτό ενισχύει το γεγονός ότι όσο μειώνουμε τον αριθμό των επεξεργαστών που έχουν κοινή μνήμη τόσο αυξάνεται ο αριθμός του συνολικού miss rate για την μνήμη L2 cache.

Ακολούθως θα δούμε την γραφική παράσταση για τα miss rates για την μνήμη L1 cache instruction (*εικόνα 28)* και την L1 cache Data (*εικόνα 29)*.

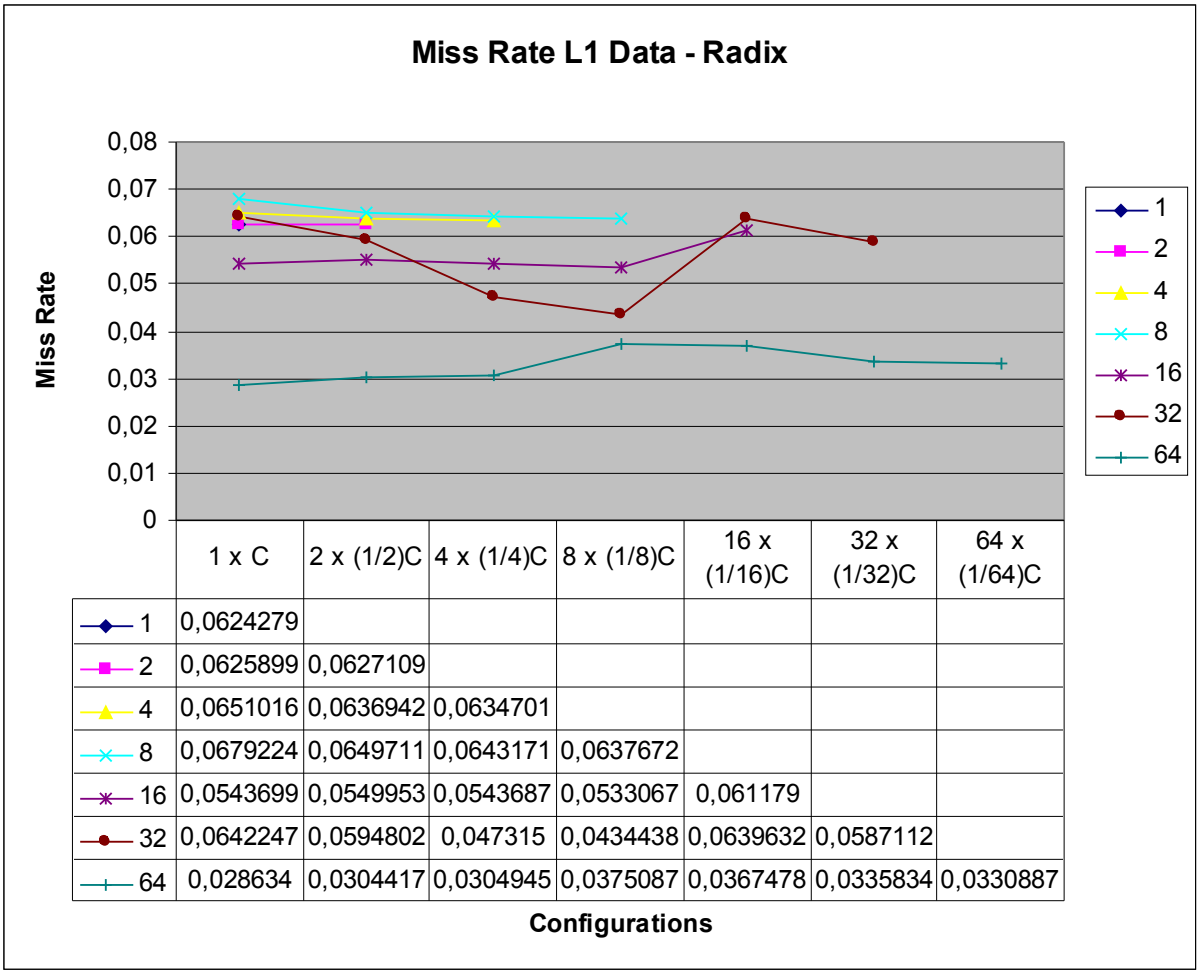


	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C
1	4,38E-05						
2	3,06E-05	3,06E-05					
4	6,48E-05	6,53E-05	6,48E-05				
8	1,39E-04	1,25E-04	1,25E-04	1,39E-04			
16	2,00E-04	1,97E-04	2,00E-04	1,97E-04	2,47E-04		
32	4,05E-04	4,02E-04	3,65E-04	3,02E-04	4,09E-04	3,81E-04	
64	3,43E-04	3,67E-04	3,61E-04	4,78E-04	4,80E-04	3,91E-04	3,52E-04

Εικόνα 28:L2 Instruction cache Miss Rate για το benchmark Radix.

Στη συνέχεια βλέπουμε την γραφική παράσταση για τα miss rates για την μνήμη L1 Instruction.Αρχικά παρατηρούμε ότι εκτελώντας την εφαρμογή μας για ένα επεξεργαστή έχουμε μεγαλύτερο miss rate από ότι εκτελώντας την εφαρμογή για δυο. Περεταίρω αύξηση του αριθμού των επεξεργαστών δεν μας δίνει κάποια μείωση στο miss rate. Με βάση την παραπάνω παρατήρηση μπορούμε να βγάλουμε το συμπέρασμα ότι αρχικά η μνήμη δεν επαρκούσε για την εφαρμογή μας δίνοντας αρκετά conflict και capacity misses, ενώ

διπλασιάζοντας την συνολική μνήμη , στο σενάριο με τους δυο επεξεργαστές, μειώσαμε τα misses αυτά στο ελάχιστο σημείο. Η αύξηση της συνολικής μνήμης από αυτό το σημείο και μετά δεν επέφερε κάποια επιπλέον μείωση στο miss rate αφού αυξάνοντας τους επεξεργαστές και την μνήμη, αυξήθηκαν αναγκαστικά και τα compulsory misses τα οποία είναι υπεύθυνα για την αύξηση του miss rate μετά από τους δυο επεξεργαστές.

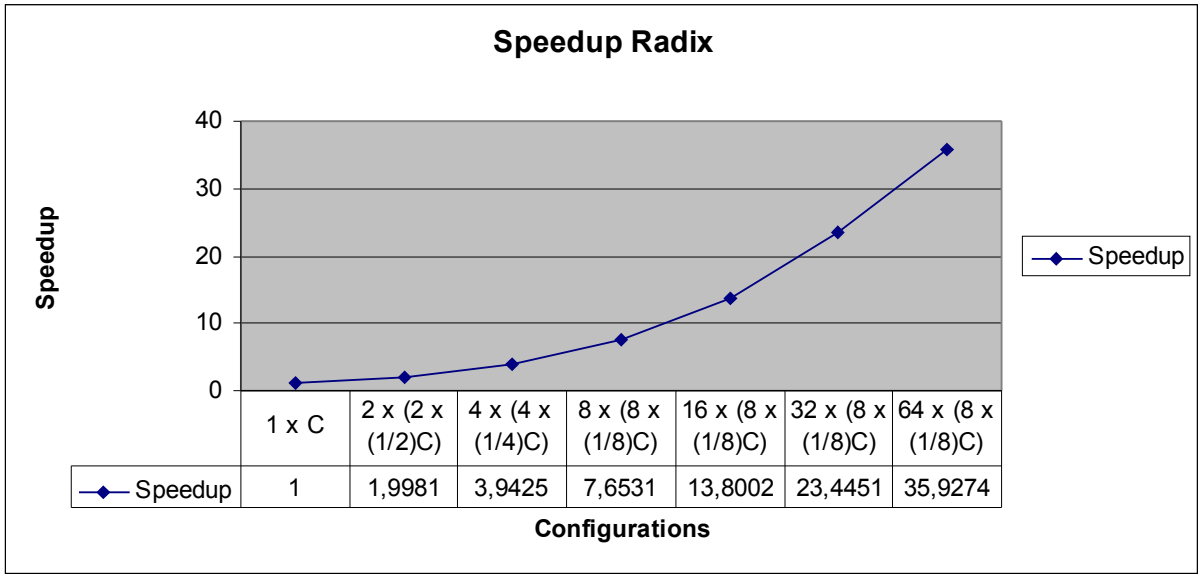


	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C
1	0,0624279						
2	0,0625899	0,0627109					
4	0,0651016	0,0636942	0,0634701				
8	0,0679224	0,0649711	0,0643171	0,0637672			
16	0,0543699	0,0549953	0,0543687	0,0533067	0,061179		
32	0,0642247	0,0594802	0,047315	0,0434438	0,0639632	0,0587112	
64	0,028634	0,0304417	0,0304945	0,0375087	0,0367478	0,0335834	0,0330887

Εικόνα 29:L2 Data cache Miss Rate για το benchmark Radix.

Σε αυτή την γραφική παράσταση (*εικόνα 29)* παρατηρούμε ότι όσο μεγαλώνει ο αριθμός των επεξεργαστών τόσο ελλατώνεται το miss rate για το Data κομμάτι της L1 cache.Όπως γνωρίζουμε ό κάθε επεξεργαστής έχει την δική του L1 cache για Data και Instruction άρα όσο αυξάνουμε τον αριθμό των επεξεργαστών αυξάνεται και το συνολικό μέγεθος της μνήμης. Γνωρίζοντας τα παραπάνω μας φαίνεται λογικό που στο συγκεκριμένο benchmark έχουμε αυτή την μείωση στα miss rates όσο αυξάνονται οι επεξεργαστές. Γνωρίζουμε ότι κάθε επεξεργαστής αναλαμβάνει να εκτελέσει ένα συγκεκριμένο κομμάτι από το συνολικό

sort που πρέπει να κάνει ο αλγόριθμός. Όταν έχουμε περισσότερους επεξεργαστές και περισσότερη μνήμη L1 τα δεδομένα που πρέπει να διαχειριστεί ένα κομμάτι μνήμης είναι λιγότερα από τα αντίστοιχα δεδομένα που έπρεπε να κάνει fetch όταν είχαμε ένα επεξεργαστή για παράδειγμα. Με αυτό τον τρόπο μειώνονται τα capacity και conflict misses ενώ η αύξηση των compulsory misses δεν φαίνεται να επηρεάζει το αποτελέσματα μας.



Εικόνα 30:Speedup για το benchmark Radix.

Η τελευταία γραφική παράσταση που θα σχολιάσουμε για το συγκεκριμένο benchmark είναι αυτή που μας δείχνει το Speedup (*εικόνα 30*). Είχαμε δει και στην γραφική παράσταση για τον χρόνο εκτέλεσης του προγράμματος μας ότι όσο αυξάνεται ο αριθμός των επεξεργαστών που εκτελούν την εφαρμογή τόσο καλύτερο χρόνο εκτέλεσης έχουμε , άρα είναι φυσικό να έχουμε και speedup.Παρατηρούμε αρχικά, ότι αυξάνοντας τους επεξεργαστές σε 2, 4 και 8 φτάνουμε το θεωρητικό βέλτιστο speedup που είναι 2, 4 και 8 αντίστοιχα. Αυτό μας δείχνει ότι η εφαρμογή μας έχει καλό παράλληλο κώδικα που εκμεταλλεύεται σωστά της δυνατότητες του hardware. Παρατηρούμε επίσης ότι τα σενάρια αυτά πετυχαίνουν αυτό το speedup με μνήμη L2 cache που είναι ξεχωριστή για τον κάθε επεξεργαστή. Στη συνέχεια όμως όσο αυξάνονται οι επεξεργαστές τόσο τα configurations όπου η μνήμη διαμοιράζεται μεταξύ των CPUs μας δίνει καλύτερα αποτελέσματα. Αναλυτικότερα για 16 επεξεργαστές πετυχαίνουμε speedup 13,8 , με την μνήμη κρυφού επιπέδου δυο να είναι κοινή ανά δυο επεξεργαστές και το σενάριο αυτό να πετυχαίνει χρόνο εκτέλεσης 1,687ms. Συνεχίζοντας αυξήσαμε τους επεξεργαστές σε 32 και πήραμε το καλύτερο speedup (23,4) όταν

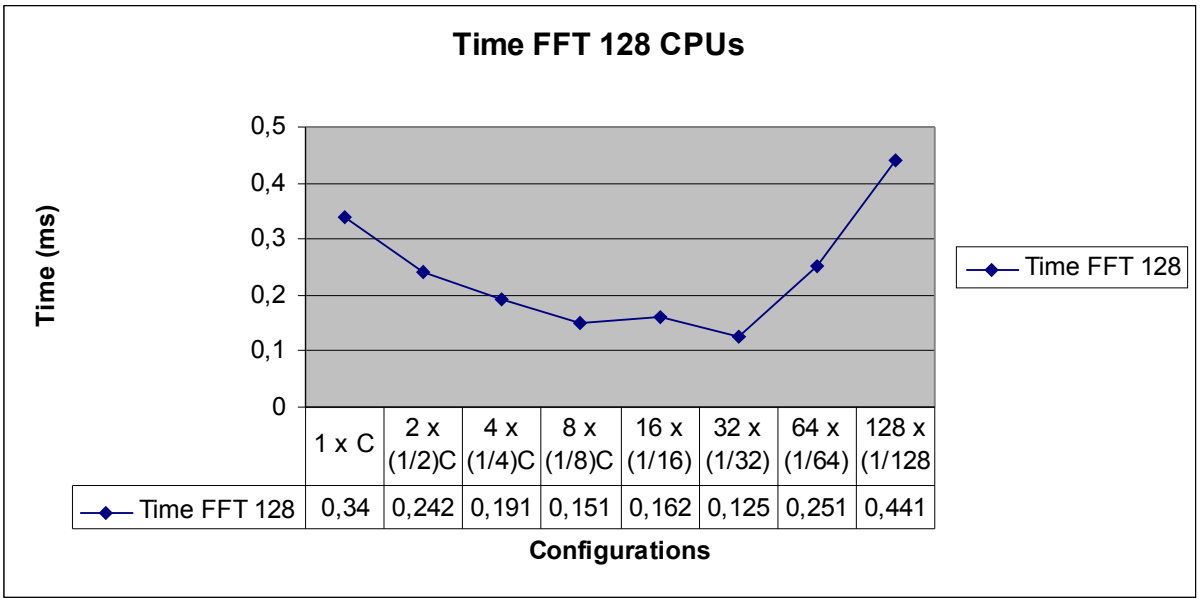
διαμοιράσαμε την μνήμη L2 cache ανά 4 επεξεργαστές. Τέλος όταν εκτελέσαμε την εφαρμογή μας για 64 CPUs πετύχαμε τον μικρότερο χρόνο εκτέλεσης (0,648 ms) όταν η μνήμη L2 ήταν κοινή για κάθε 8 επεξεργαστές.

Συνοψίζοντας τα ευρήματα μας, η αύξηση των επεξεργαστών μας έδωσε καλύτερους χρόνους εκτέλεσης για την εφαρμογή μας. Όπως είδαμε και από της γραφικές μας παραστάσεις δεν είχαμε κάποιο συγκεκριμένο configuration μνήμης το οποίο για κάθε αριθμό επεξεργαστών να μας δίνει τον καλύτερο χρόνο. Παρατηρήσαμε επίσης ότι αν και το configuration όπου η μνήμη cache ήταν κοινή για όλα τα CPUs, μας έδωσε το μικρότερο miss rate δεν μας έδωσε ποτέ τον καλύτερο χρόνο. Αυτό όπως έχουμε πει οφείλεται στο μεγάλο access time που έχουμε προς την μνήμη όταν αυξάνεται η χωρητικότητα της. Για αυτό τον λόγω, τους καλύτερους χρόνους εκτέλεσης τους πετυχαίνουν σενάρια όπου έχουν μνήμη L2 cache αρκετά μεγάλη ώστε να έχουν μειωμένα miss rates και μικρό σχετικά access time. Τα σενάρια αυτά είναι συνήθως αυτά όπου η μνήμη διαμοιράζεται μεταξύ δυο και οχτώ επεξεργαστών.

4.4 Benchmark FFT

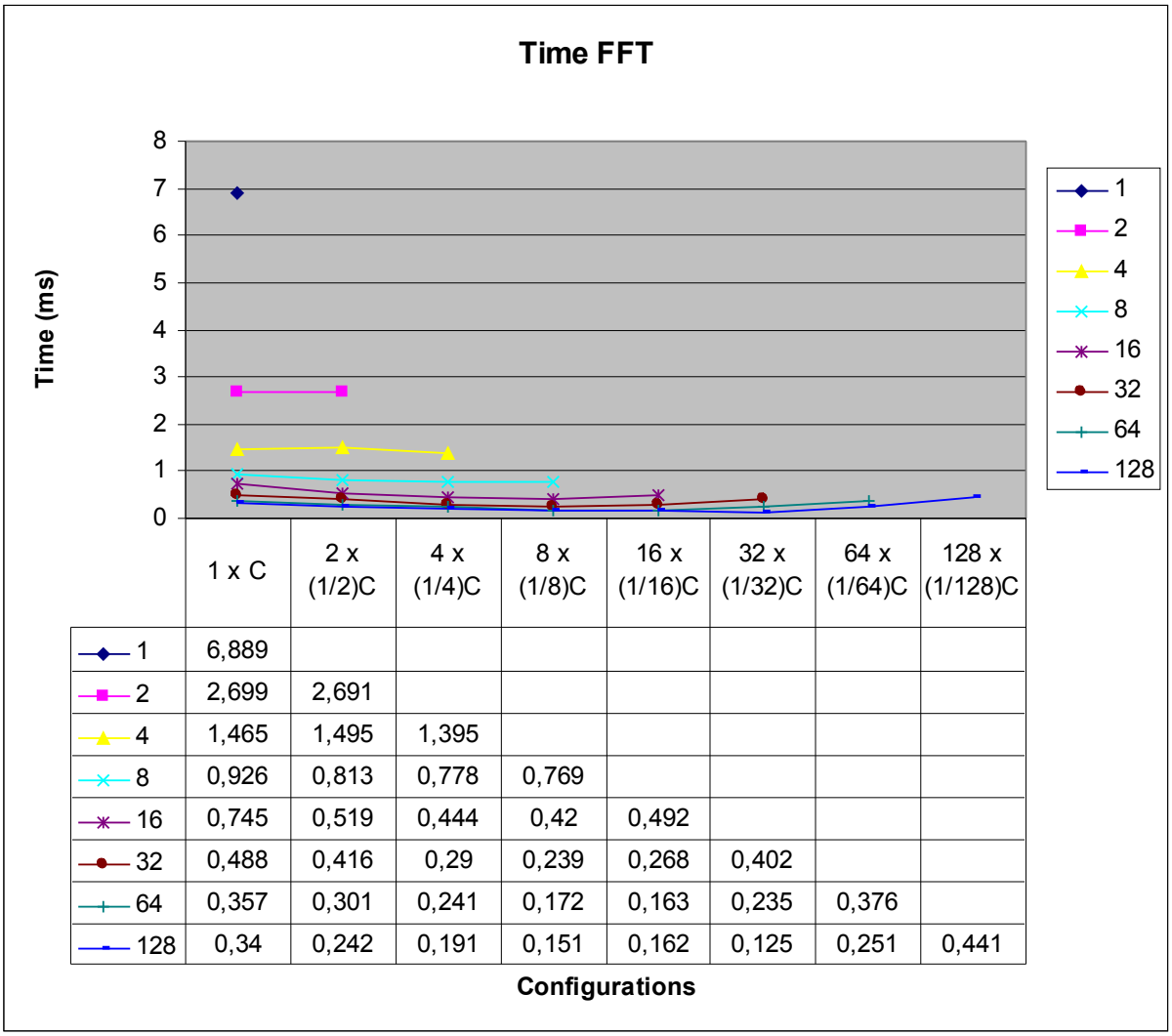
Σε αυτό το κομμάτι της διπλωματικής μας εργασίας παρουσιάζουμε τα αποτελέσματα για το benchmark FFT. Το FFT είναι μια πολύπλοκή μονοδιάστατη έκδοση του “Six-Step” FFT που περιγράφεται στο άρθρο ,του D.H.Bailey, *FFTs in External or Hierarchical Memory.*

Αρχικά βλέπουμε την γραφική παράσταση που παρουσιάζει τους χρόνους εκτέλεσης (*εικόνα 31)* που είχαμε τρέχοντας την εφαρμογή μας για 128 επεξεργαστές και στη συνέχεια την γραφική παράσταση για τα διάφορα configurations που δημιουργήσαμε αλλά και για διαφορετικούς αριθμούς επεξεργαστών (*εικόνα 32)*. Όπως μπορούμε να διακρίνουμε όσο αυξάνεται ο αριθμός επεξεργαστών, και μαζί με τον αριθμό των CPUs η συνολική χωρητικότητα της L1 και της L2, μειώνεται και ο χρόνος εκτέλεσης. Με δεδομένο ότι η εφαρμογή FFT είναι παράλληλη, περιμέναμε να έχουμε καλύτερο χρόνο εκτέλεσης.



Εικόνα 31: Χρόνος εκτέλεσης για το benchmark FFT για 128 CPUs.

Αναλυτικότερα, ο καλύτερος χρόνος εκτέλεσης παρατηρείται όταν χρησιμοποιούμε 128 επεξεργαστές και η μνήμη L2 cache είναι κοινή ανά 4 επεξεργαστές (0,125 ms). Για τον ίδιο αριθμό επεξεργαστών, είχαμε επίσης καλούς χρόνους εκτέλεσης όταν η κρυφή μνήμη επιπέδου δυο διαμοιραζόταν σε 16 (0,151ms) και 8 (0,162 ms) επεξεργαστές. Σημαντικό είναι να υπενθυμίσουμε ότι όταν εκτελούμε μια εφαρμογή με 128 επεξεργαστές η συνολική χωρητικότητα της μνήμης cache L2 έχει οριστεί στα 256MB ενώ για 64 και 32 επεξεργαστές η μνήμη έχει οριστεί στα 128 και 64 ,MB αντίστοιχα.



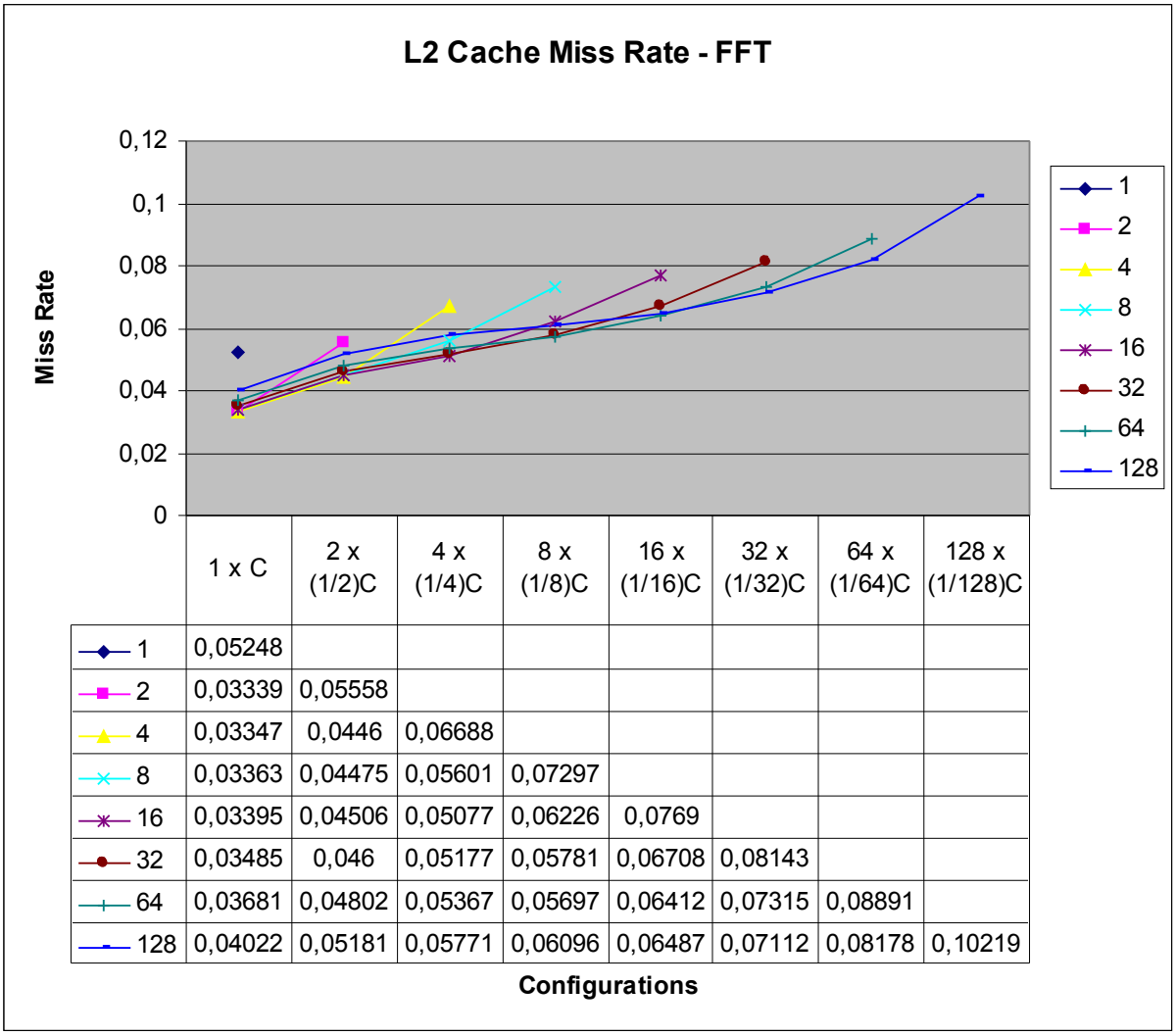
	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	6,889							
2	2,699	2,691						
4	1,465	1,495	1,395					
8	0,926	0,813	0,778	0,769				
16	0,745	0,519	0,444	0,42	0,492			
32	0,488	0,416	0,29	0,239	0,268	0,402		
64	0,357	0,301	0,241	0,172	0,163	0,235	0,376	
128	0,34	0,242	0,191	0,151	0,162	0,125	0,251	0,441

Εικόνα 32: Χρόνος εκτέλεσης για το benchmark FFT.

Όσο αφορά τον χρόνο εκτέλεσης της εφαρμογής μας όταν χρησιμοποιούμε 64 επεξεργαστές, μπορούμε να δούμε και από την γραφική μας παράσταση (*εικόνα 32)* ότι τον μικρότερο χρόνο εκτέλεσης τον έχουμε όταν διαμοιράζουμε την μνήμη L2 cache ανά 4 επεξεργαστές (0,163 ms) ,ενώ επίσης κοντινό στον καλύτερο χρόνο έχουμε όταν η μνήμη είναι κοινή ανά 8 CPUs(0,172 ms). Παρατηρώντας επίσης τους χρόνους εκτέλεσης για την δοκιμή όπου τρέξαμε για 32 επεξεργαστές , βλέπουμε ότι και για αυτό τον αριθμό επεξεργαστών το configuration που μας δίνει τον μικρότερο χρόνο εκτέλεσης είναι αυτό όπου η μνήμη L2 cache είναι κοινή για κάθε 4 επεξεργαστές (0,239 ms). Μπορούμε να δούμε επίσης ότι σχετικά καλούς χρόνους έχουν επιτύχει και τα configurations όπου η μνήμη διαμοιράζεται ανά 2 (0,268ms) και 8 (0,29ms) επεξεργαστές. Επιπλέον παρατηρούμε ότι τρέχοντας την δοκιμή μας για 16 επεξεργαστές έχουμε τον καλύτερο χρόνο εκτέλεσης όταν διαμοιράζεται η

μνήμη L2 ανά 2 επεξεργαστές, ενώ για το configuration όπου κάθε επεξεργαστής έχει ξεχωριστή L2 cache, επιτυγχάνουμε τον καλύτερο χρόνο εκτέλεσης όταν εκτελούμε την εφαρμογή μας για 8,4,2 και 1 επεξεργαστή. Παρατηρώντας γενικά την γραφική με τους χρόνους εκτέλεσης για την εφαρμογή μας, βλέπουμε ότι το configuration όπου η μνήμη κρυφού επιπέδου δυο, είναι κοινή για όλους τους επεξεργαστές μας δίνει πάντα το χειρότερο χρόνο εκτέλεσης. Αυτό συμβαίνει διότι τα οφέλη σε χρόνο από τα misses που αποκομίζουμε ενώνοντας την μνήμη, είναι μικρότερα από την καθυστέρηση που έχουμε στο access time όσο μεγαλώνει η μνήμη.

Στη γραφική παράσταση που ακολουθεί βλέπουμε το miss rate της L2 cache (*εικόνα 33)*. Παρατηρώντας τα αποτελέσματα των δοκιμών μας όπως φαίνονται στην γραφική αυτή παράσταση μπορούμε να δούμε ότι το μικρότερο miss rate το επιτυγχάνουμε όταν η μνήμη L2 cache είναι ενιαία και διαμοιράζεται από όλους τους επεξεργαστές. Γνωρίζαμε εξαρχής από τη θεωρητικές μας γνώσεις ότι ενοποιώντας την κρυφή μνήμη δευτέρου επιπέδου και αυξάνοντας με αυτό τον τρόπο την χωρητικότητα της θα μειώναμε τα capacity και conflict misses .



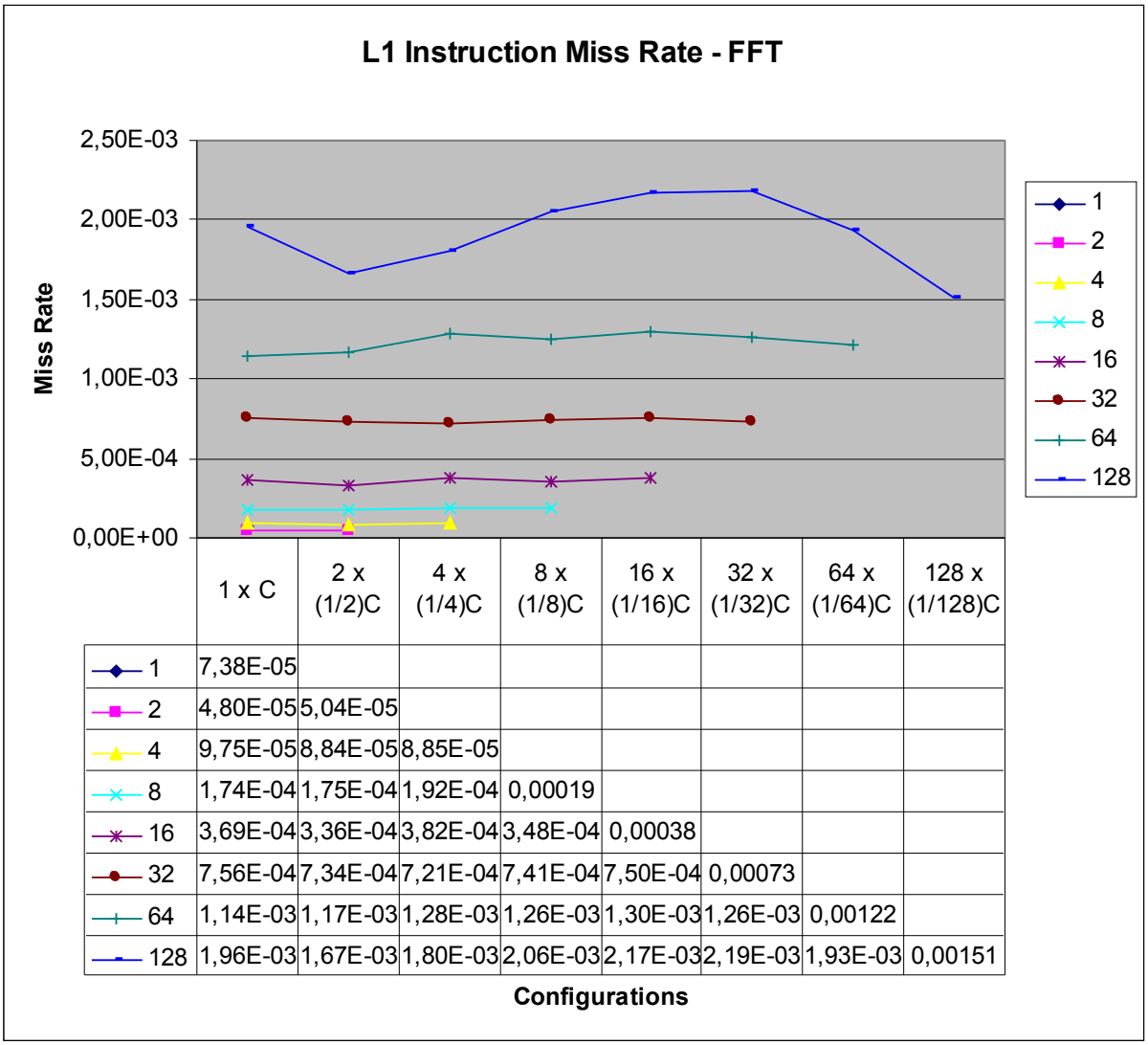
	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	0,05248							
2	0,03339	0,05558						
4	0,03347	0,0446	0,06688					
8	0,03363	0,04475	0,05601	0,07297				
16	0,03395	0,04506	0,05077	0,06226	0,0769			
32	0,03485	0,046	0,05177	0,05781	0,06708	0,08143		
64	0,03681	0,04802	0,05367	0,05697	0,06412	0,07315	0,08891	
128	0,04022	0,05181	0,05771	0,06096	0,06487	0,07112	0,08178	0,10219

Εικόνα 33:L2 Miss Rate για το benchmark FFT.

Δημιουργώντας λοιπόν μια ενιαία μεγάλη L2 cache αντί για πολλές μικρότερες εξοικονομούμε τον χρόνο που χρειάζεται για να γίνουν fetch κομμάτια που είναι κοινά για όλους τους επεξεργαστές που εκτελούν το πρόγραμμα, φέρνοντας τα μόνο μια φορά και δίνοντας την δυνατότητα σε όλους τους επεξεργαστές να χρησιμοποιούν αυτή την μεγάλη μνήμη, με το ανάλογο κόστος σε access time. Μπορούμε επίσης, να δούμε ότι το συγκεκριμένο benchmark έχει αρκετές ανάγκες σε μνήμη. Αυτό το συμπέρασμα το εξάγουμε από το γεγονός όταν το πρόγραμμα εκτελείται από 1 επεξεργαστή έχουμε miss rate 0,05248 που μειώνεται καθώς αυξάνουμε την συνολική χωρητικότητα της μνήμης. Στη συνέχεια βλέπουμε ότι για όλες τις δοκιμές που εκτελέσαμε και είχαμε κοινή μνήμη για όλους τους επεξεργαστές είχαμε για το Miss Rate τιμές που δεν διαφέρουν σημαντικά. Με βάση τα παραπάνω μπορούμε να πούμε ότι αυξάνοντας την μνήμη L2 cache από 2MB (για 1 CPU) σε

4MB (2 CPUs) σχεδόν εκμηδενίσαμε τα capacity και τα conflict, ενώ το miss rate που έχουμε οφείλεται στα compulsory misses. Παρατηρούμε επίσης ότι για τα υπόλοιπα configurations της μνήμης έχουμε μεγαλύτερα miss rates από αυτό που είχαμε όταν η μνήμη είναι κοινή. Η αύξηση των miss rate οφείλεται κυρίως στην αναγκαστική αύξηση των compulsory misses αφού, μοιράζοντας την μνήμη cache σε περισσότερα κομμάτια στην αρχή κάθε εκτέλεσης θα πρέπει να φέρουμε τα δεδομένα που χρειαζόμαστε περισσότερες φορές. Τέλος, από την παραπάνω γραφική παράσταση μπορούμε να βγάλουμε το συμπέρασμα ότι αν το miss rate ήταν το μόνο που μας επηρέαζε το χρόνο εκτέλεσης των εφαρμογών μας θα έπρεπε να χρησιμοποιούσαμε για της υλοποιήσεις μας, επεξεργαστές που η μνήμη L2 cache θα ήταν κοινή για όλα τα CPUs ή που έστω θα ήταν κοινή για τους περισσότερους δυνατούς επεξεργαστές.

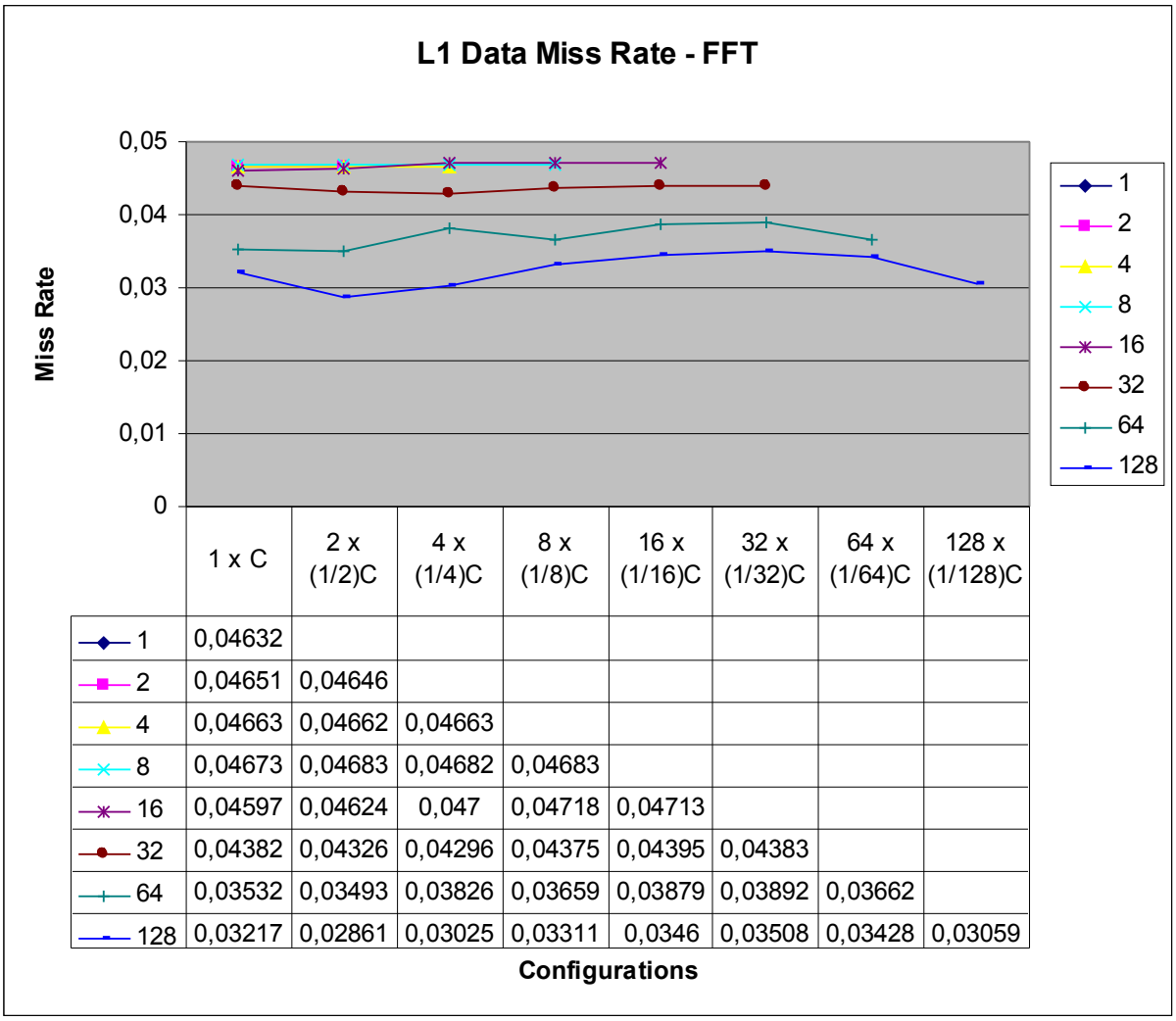
Στη γραφική παράσταση που ακολουθεί βλέπουμε το miss rate για το κομμάτι των εντολών της μνήμης L1 cache (*εικόνα 34)*.Όπως μπορούμε να παρατηρήσουμε όσο αυξάνουμε τον αριθμό των επεξεργαστών με τους οποίους εκτελούμε την εφαρμογή μας, τόσο μειώνεται το miss rate της μνήμης L1 cache. Το συμπέρασμα αυτό είναι λογικό καθώς όταν αυξάνουμε τον αριθμό των επεξεργαστών αυξάνουμε και τον αριθμό των L1 caches όπου πρέπει να φέρουμε τα δεδομένα, στη συγκεκριμένη περίπτωση εντολές, από την μνήμη χαμηλότερου επιπέδου. Έτσι μπορούμε να πούμε ότι η αύξηση στο miss rate που παρατηρούμε όσο αυξάνονται οι επεξεργαστές οφείλονται κυρίως στα compulsory misses και όχι τόσο στα conflict και capacity misses. Επίσης παρατηρούμε ότι το miss rate όταν εκτελούμε την εφαρμογή μας για ένα επεξεργαστή είναι μεγαλύτερο από το αντίστοιχο όταν τρέχουμε για δυο επεξεργαστές. Αυτό μας δείχνει ότι εκτελώντας για ένα επεξεργαστή είχαμε conflict και capacity misses τα οποία με τον διπλασιασμό της συνολικής μνήμης L1 δεν ξανασυνέβησαν, ενώ τα επιπλέον compulsory misses δεν επηρέασαν τόσο το miss rate για τη δοκιμή μας με τους δυο επεξεργαστές. Τέλος παρατηρούμε ότι περαιτέρω αύξηση της μνήμης δεν μας βοήθησε στο να μειώσουμε το miss rate σε μεγαλύτερο βαθμό, πράγμα που σημαίνει ότι δεν μπορούμε να μειώσουμε σε μεγαλύτερο βαθμό τα conflict και capacity misses.



	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	7,38E-05							
2	4,80E-05	5,04E-05						
4	9,75E-05	8,84E-05	8,85E-05					
8	1,74E-04	1,75E-04	1,92E-04	0,00019				
16	3,69E-04	3,36E-04	3,82E-04	3,48E-04	0,00038			
32	7,56E-04	7,34E-04	7,21E-04	7,41E-04	7,50E-04	0,00073		
64	1,14E-03	1,17E-03	1,28E-03	1,26E-03	1,30E-03	1,26E-03	0,00122	
128	1,96E-03	1,67E-03	1,80E-03	2,06E-03	2,17E-03	2,19E-03	1,93E-03	0,00151

Εικόνα 34:L1 Instruction Miss Rate για το benchmark FFT.

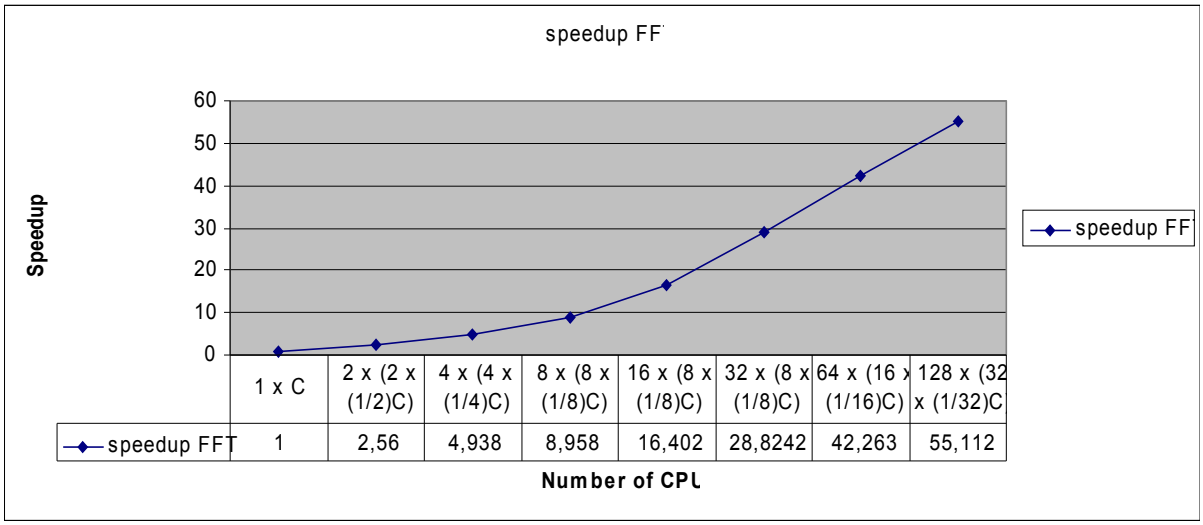
Συνεχίζουμε το σχολιασμό μας στις γραφικές που σχετίζονται με το benchmark FFT σχολιάζοντας την γραφική παράσταση που μας δείχνει το miss rate για το Data κομμάτι της L1 cache μνήμης (*εικόνα 35*). Όπως παρατηρούμε όσο αυξάνουμε τον αριθμό των επεξεργαστών με τους οποίους εκτελούμε την εφαρμογή μας τόσο μειώνεται το miss rate. Με βάση αυτό μπορούμε να βγάλουμε το συμπέρασμα ότι αυξάνοντας την συνολική χωρητικότητα της L1 data cache μνήμης περιορίσαμε τα conflict misses όπως και τα capacity misses. Επίσης μπορούμε να πούμε ότι υπάρχουν περιθώρια για περεταίρω αύξηση της χωρητικότητας της μνήμης. Εάν δεν υπήρχαν αυτά τα περιθώρια οι τιμή του miss rate για το σενάριο δοκιμής με την μεγαλύτερη σε χωρητικότητα μνήμης θα ήταν μεγαλύτερο ή ίση με το αντίστοιχο miss rate των σεναρίων δοκιμής με μικρότερες L1 μνήμες.



	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	0,04632							
2	0,04651	0,04646						
4	0,04663	0,04662	0,04663					
8	0,04673	0,04683	0,04682	0,04683				
16	0,04597	0,04624	0,047	0,04718	0,04713			
32	0,04382	0,04326	0,04296	0,04375	0,04395	0,04383		
64	0,03532	0,03493	0,03826	0,03659	0,03879	0,03892	0,03662	
128	0,03217	0,02861	0,03025	0,03311	0,0346	0,03508	0,03428	0,03059

Εικόνα 35:L1 Data Miss Rate για το benchmark FFT.

Στη συνέχεια βλέπουμε την γραφική παράσταση για το Speedup για το benchmark FFT (*εικόνα 36)*.



Εικόνα 36:Speedup για το benchmark FFT.

Στη γραφική παράσταση αυτή παρατηρούμε ότι έχουμε καθώς αλλάζουμε τα configurations μας αλλά και τον αριθμό των επεξεργαστών έχουμε αύξηση στο speedup. Πιο συγκεκριμένα βλέπουμε ότι το μέγιστο speedup (55,11) το επιτυγχάνουμε όταν χρησιμοποιούμε 128 επεξεργαστές για να εκτελέσουμε την εφαρμογή μας ενώ η μνήμη L2 cache είναι κοινή για κάθε δυο CPUs. Παρατηρούμε επίσης ότι μέχρι και το σενάριο με τους 16 επεξεργαστές έχουμε speedup μεγαλύτερο από αυτά που περιμέναμε με την αύξηση των επεξεργαστών που εκτελούν την εφαρμογή μας. Αυτό συμβαίνει διότι έχουμε μια εφαρμογή η οποία εκτελείται παράλληλα και έχει την δυνατότητα να μας δώσει καλύτερους χρόνους με περισσότερα CPUs αλλά και επειδή μεγαλώνοντας τον χωρητικότητα της L1 και L2 μειώνονται τα miss rates, μειώνοντας έτσι και τον συνολικό χρόνο εκτέλεσης. Βλέποντας την παραπάνω γραφική λοιπόν, μπορούμε να πούμε ότι αυξάνοντας τον αριθμό των επεξεργαστών έχουμε την μείωση που επιθυμούμε στον χρόνο εκτέλεσης.

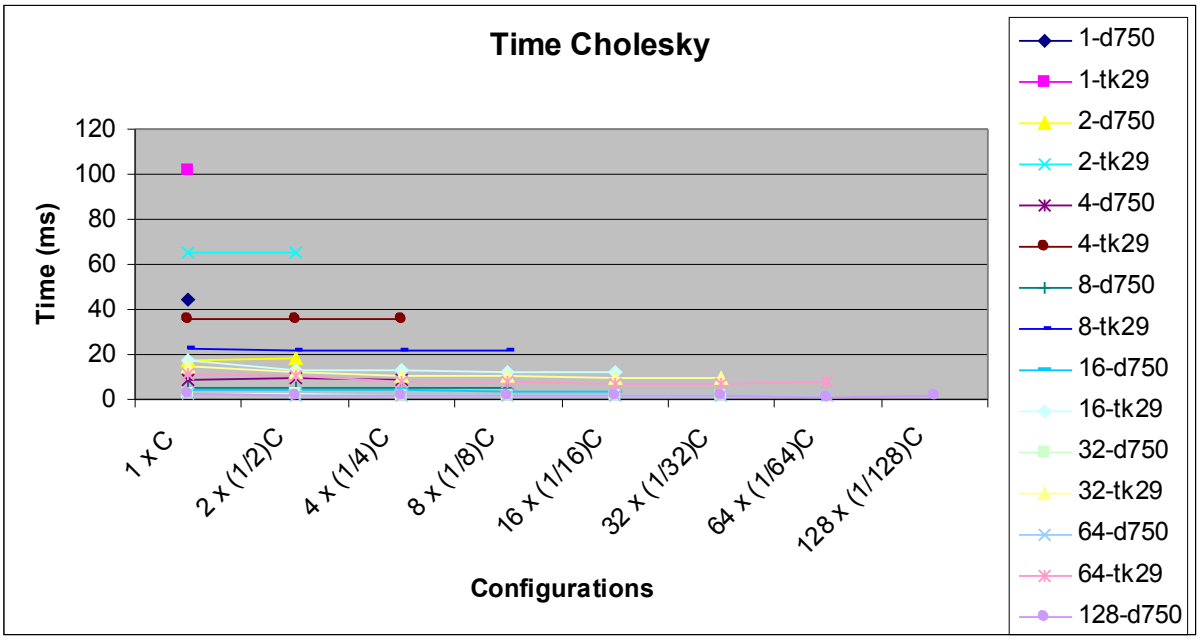
Συμπερασματικά, μπορούμε να πούμε ότι όσο αυξάνεται ο αριθμός των επεξεργαστών πετυχαίνουμε καλύτερο χρόνο εκτέλεσης για το benchmark μας. Επίσης αυξάνοντας τους επεξεργαστές και την συνολική χωρητικότητα της L1 cache , έχουμε μεγάλη μείωση στο miss rate για το κομμάτι Data της L1 και αύξηση για το αντίστοιχο Instruction κομμάτι. Επιπλέον με την αύξηση της συνολικής χωρητικότητας της L2 μνήμης μειώσαμε το miss rate, ιδιαίτερα για το configuration όπου η L2 είναι κοινή για όλους τους επεξεργαστές. Παρόλο που πετύχαμε χαμηλό miss rate με αυτό το σενάριο δεν καταφέραμε να πετύχουμε καλό χρόνο εκτέλεσης λόγω του αυξημένου access time προς την μνήμη L2. Όσο αφορά τις

δοκιμές μας, αρχικά, για 2,4,και 8 επεξεργαστές , τους καλύτερους χρόνους εκτέλεσης πετύχαινε το configuration όπου η μνήμη L2 cache ήταν ξεχωριστή για κάθε επεξεργαστή. Στη συνέχεια όπως είδαμε, καλύτερο χρόνο για τα 16 CPUs, πέτυχε το σενάριο όπου η μνήμη διαμοιράζεται ανά δυο επεξεργαστές, ενώ για τα υπόλοιπες δοκιμές στο configuration με το οποίο πήραμε τα καλύτερα αποτελέσματα , διαμοιράζαμε την κρυφή μνήμη επιπέδου δυο ανά τέσσερις επεξεργαστές. Όπως καταλαβαίνουμε όσο αυξάνεται ο αριθμός των επεξεργαστών τα configurations όπου η μνήμη L2 είναι ξεχωριστή για κάθε CPU έχει μεγάλο miss rate που καθυστερεί την εκτέλεση του προγράμματος, ενώ τα configurations όπου η μνήμη L2 διαμοιράζεται από πολλούς ή ακόμα και από όλους τους επεξεργαστές δεν επιτυγχάνουν καλές επιδόσεις λόγω του αυξημένου access time προς την μνήμη. Έτσι λοιπόν τα configuration που έχουν την καλύτερη επίδοση είναι αυτά που καταφέρνουν να συνδυάσουν μικρό miss rate και μικρό access time προς την μνήμη.

4.5 Benchmark Cholesky

Σε αυτή την ενότητα της διπλωματικής μας εργασίας βλέπουμε τα αποτελέσματα του benchmark Cholesky το οποίο εκτελεί μια blocked Cholesky factorization σε ένα sparse matrix σύμφωνα με τον αλγόριθμο που περίγραψε ο E. Rothberg στη διδακτορική του διατριβή στο πανεπιστήμιο του Stanford με τίτλο, Exploiting the Memory Hierarchy in Sequential and Parallel Sparse Cholesky Factorization. Το συγκεκριμένο benchmark επιλέξαμε να το εκτελέσουμε με δυο διαφορετικά inputs ώστε να δούμε την συμπεριφορά του. Αναλυτικότερα, χρησιμοποιήσαμε τα inputs d750 και tk29 όπου το δεύτερο είναι ελάχιστα μεγαλύτερο σε όγκο (1,4MB έναντι 1,6MB).

Αρχικά θα δούμε στην γραφική παράσταση που ακολουθεί τις επιδόσεις σε χρόνο που είχαμε εκτελώντας την εφαρμογή μας για τα διάφορα σενάρια και configurations , όπως και για τα δυο διαφορετικά inputs (*εικόνα 37)*.



Εικόνα 37: Χρόνος εκτέλεσης για το benchmark FFT

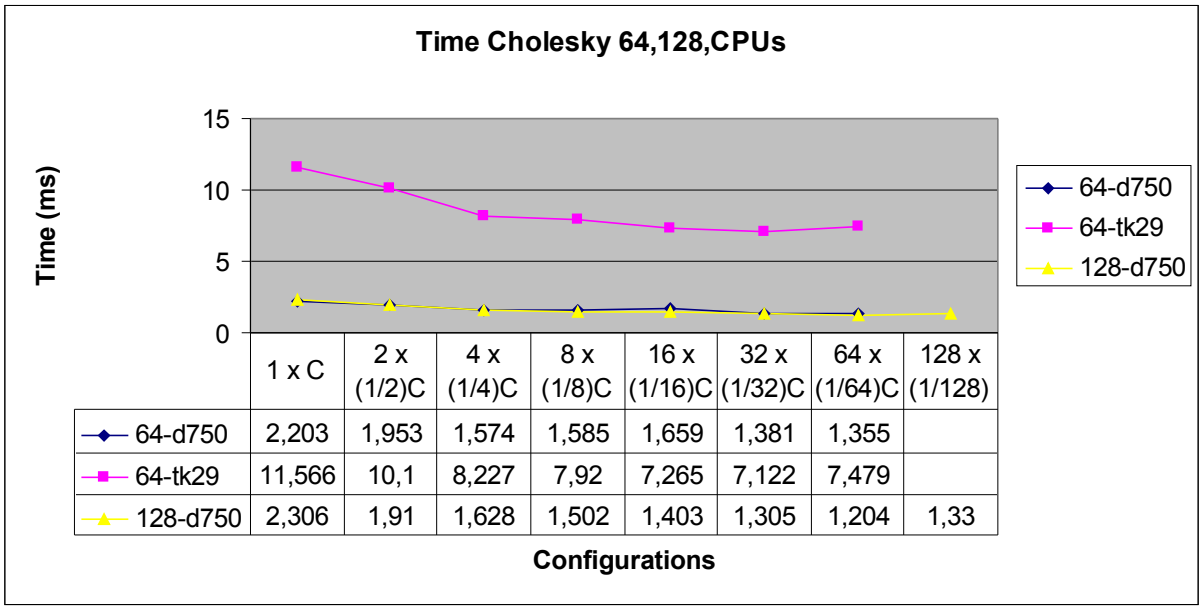
	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1-d750	44,6							
1-tk29	102,169							
2-d750	17,429	18,145						
2-tk29	65,288	65,463						
4-d750	8,839	9,157	8,767					
4-tk29	35,775	36,071	35,832					
8-d750	5,218	5,305	5,306	5,047				
8-tk29	22,701	21,943	21,978	21,688				
16-d750	4,542	3,927	4,137	3,855	3,807			
16-tk29	17,577	12,953	12,718	12,264	12,186			
32-d750	2,779	2,302	2,093	2,109	1,989	1,93		
32-tk29	14,422	11,824	10,668	10,496	9,849	9,969		
64-d750	2,203	1,953	1,574	1,585	1,659	1,381	1,355	
64-tk29	11,566	10,1	8,227	7,92	7,265	7,122	7,479	
128-d750	2,306	1,91	1,628	1,502	1,403	1,305	1,204	1,33

Πίνακας 3: Χρόνος εκτέλεσης του Benchmark Cholesky για inputs d750,tk29.

Όπως μπορούμε να παρατηρήσουμε από την παραπάνω γραφική (*εικόνα 37)* το input tk29 δίνει χειρότερους χρόνους εκτέλεσης από το input d750.Εξαρχής περιμέναμε ότι δίνοντας μεγαλύτερο input θα είχαμε μεγαλύτερο χρόνο εκτέλεσης, λόγω του μεγαλύτερου όγκου

δεδομένων που πρέπει να επεξεργαστεί η εφαρμογή. Σημαντικό είναι να παρατηρήσουμε ότι για το tk29 δεν μπορούμε να τρέξουμε την εφαρμογή για 128 επεξεργαστές πράγμα που μπορούμε για το d750. Μπορούμε να δούμε βλέποντας τους χρόνους εκτέλεσης ότι για το input d750 όταν χρησιμοποιούμε 2 επεξεργαστές για την εκτέλεση του προγράμματος μας καλύτερη επίδοση μας δίνει το configuration όπου η μνήμη είναι ενιαία και για τα 2 CPUs, ενώ όταν αυξήσουμε τους επεξεργαστές σε 4 (ενώ διπλασιάζουμε και την συνολική χωρητικότητα για τις μνήμες L1, L2) έχουμε τον καλύτερο χρόνο εκτέλεσης όταν κάθε CPU έχει την δική του ξεχωριστή κρυφή μνήμη επιπέδου δυο. Αντίστοιχα για το input tk29, καλύτερο χρόνο για 2 και 4 επεξεργαστές έχουμε, όταν η μνήμη είναι κοινή για όλα τα CPUs. Όταν αυξήσουμε τον αριθμό των επεξεργαστών σε 8, παρατηρούμε ότι για το d750 έχουμε τον καλύτερο χρόνο εκτέλεσης (5,047ms) όταν η μνήμη L2 είναι ξεχωριστή για κάθε επεξεργαστή, ενώ ο δεύτερος καλύτερος χρόνος (5,218 ms) για αυτό τον αριθμό επεξεργαστών παρατηρείται όταν η μνήμη είναι κοινή για όλα τα CPUS. Αντίστοιχα παρατηρούμε ότι και για το tk29 έχουμε τον καλύτερο χρόνο (21,688 ms) όταν η μνήμη L2 είναι ξεχωριστή για κάθε CPUs, ενώ τον δεύτερο καλύτερο χρόνο (21,943 ms) όταν η κρυφή μνήμη επιπέδου δυο διαμοιράζεται ανά τέσσερις επεξεργαστές. Συνεχίζοντας βλέπουμε ότι τρέχοντας το benchmark Cholesky με το input d750 για 16 ,32 και 64 επεξεργαστές, έχουμε τον καλύτερο χρόνο εκτέλεσης (3,807 ms, 1,93 ms , 1,355 ms)όταν η μνήμη L2 cache είναι ξεχωριστή για κάθε CPU, ενώ και το configuration όπου η μνήμη L2 μοιράζεται ανά δυο επεξεργαστές μας δίνει καλό execution time.Αντίθετα για το input tk29 παρατηρούμε ότι όταν τρέχουμε για 16 CPUs την εφαρμογή μας το configuration με την ξεχωριστή L2 για κάθε επεξεργαστή μας δίνει καλύτερο χρόνο (12,186 ms), ενώ όταν αυξήσουμε τον αριθμό των CPUs σε 32 και 64 έχουμε τους καλύτερους χρόνους όταν η κρυφή μνήμη διαμοιράζεται ανά 2 (9,849 ms, 7,122 ms) επεξεργαστές. Χαρακτηριστικό είναι ότι για τα δύο ιinputs τα σενάρια στα οποία εκτελούμε την εφαρμογή μας με παραπάνω από 4 επεξεργαστές έχουμε τους χειρότερους χρόνους εκτέλεσης όταν η κρυφή μνήμη επιπέδου 2 είναι κοινή για όλα τα CPUs. Αυτό εξηγείται καθώς όσο αυξάνουμε τον αριθμό των επεξεργαστών που χρησιμοποιούμε για την εκτέλεση της εφαρμογής μας , αυξάνουμε και την συνολική χωρητικότητα της μνήμης L2 cache. Η αύξηση της χωρητικότητας έχει σαν αποτέλεσμα εκτός από την μείωση των miss rates , την αύξηση του access time μπρος την μνήμη. Η αύξηση αυτή στο access time πιστεύουμε ότι είναι υπεύθυνη για τον υψηλό χρόνο εκτέλεσης που παρατηρούμε.

Στη συνέχεια βλέπουμε την γραφική παράσταση για τους χρόνους για τα δύο σενάρια όπου εκτελείται το benchmark Cholesky για 64 και 128 επεξεργαστές για τα δύο inputs (*εικόνα 38)*.



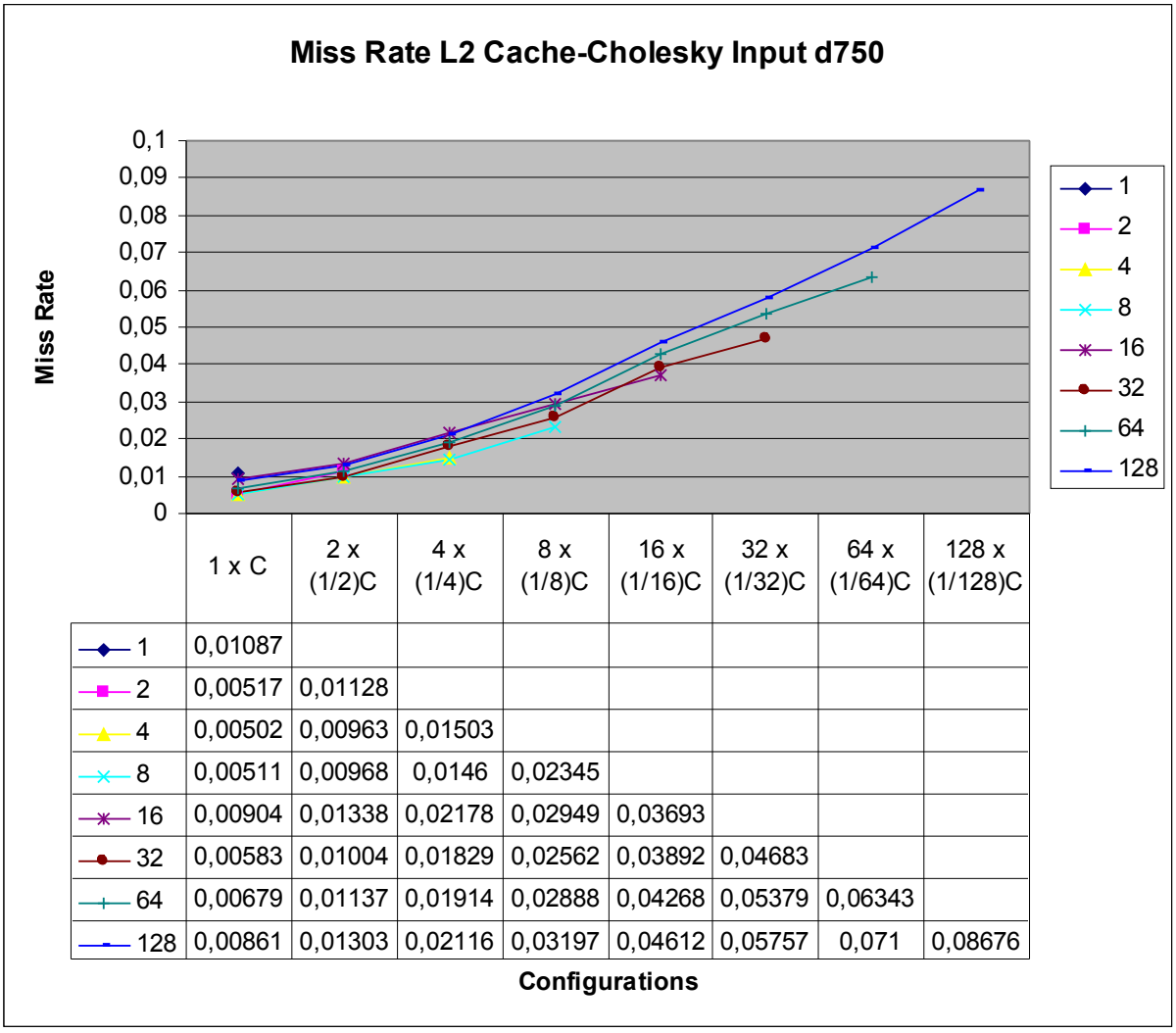
	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)
64-d750	2,203	1,953	1,574	1,585	1,659	1,381	1,355	
64-tk29	11,566	10,1	8,227	7,92	7,265	7,122	7,479	
128-d750	2,306	1,91	1,628	1,502	1,403	1,305	1,204	1,33

Εικόνα 38: Χρόνος εκτέλεσης για το benchmark FFT για 64 και 128 CPUs

Παρατηρούμε σε αυτή την γραφική παράσταση την μεγάλη διαφορά στον χρόνο εκτέλεσης που έχουμε για τα δύο inputs. Το input tk29 μας δίνει χρόνο εκτέλεσης μεγαλύτερο από αυτό που παίρνουμε από το d750. Μπορούμε να δούμε επίσης ότι για το input tk29 όταν χρησιμοποιούμε 64 επεξεργαστές για την εκτέλεση της εφαρμογής μας πετυχαίνουμε τον καλύτερο χρόνο εκτέλεσης όταν η μνήμη L2 cache είναι κοινή για κάθε δυο επεξεργαστές. Βλέπουμε επίσης ότι καλούς χρόνους εκτέλεσης πετυχαίνουμε όταν η μνήμη L2 είναι ξεχωριστή για κάθε CPU και όταν διαμοιράζεται ανά τέσσερις επεξεργαστές. Θυμίζουμε ότι για το input tk29 δεν μπορούμε να εκτελέσουμε την εφαρμογή για 128 επεξεργαστές. Είναι εμφανές επίσης από την γραφική παράσταση ότι για το input d750 όταν τρέχουμε την εφαρμογή για 64 και 128 επεξεργαστές αντίστοιχα δεν έχουμε μεγάλη διαφοροποίηση στον χρόνο εκτέλεσης. Βλέπουμε ότι για τους 64 επεξεργαστές το configuration που μας δίνει τον μικρότερο χρόνο εκτέλεσης είναι αυτό όπου η μνήμη L2 είναι ξεχωριστή για κάθε επεξεργαστή. Για τον ίδιο αριθμό επεξεργαστών κοντινό στον καλύτερο χρόνο εκτέλεσης παίρνουμε όταν διαμοιράζουμε την L2 cache ανά δυο CPUs. Επιπλέον παρατηρούμε ότι καθώς τρέχουμε την εφαρμογή μας για 128 επεξεργαστές έχουμε τον καλύτερο χρόνο

εκτέλεσης όταν μοιράζουμε την L2 cache μνήμη ανά δυο επεξεργαστές ενώ τον δεύτερο καλύτερο χρόνο τον πετυχαίνουμε όταν έχουμε κοινή μνήμη για κάθε τέσσερις επεξεργαστές. Τέλος βλέπουμε ότι όσο αυξάνουμε τον αριθμό των επεξεργαστών που μοιράζονται την μνήμη L2 πέρα του αριθμού 4 έχουμε μεγάλη αύξηση και στον χρόνο εκτέλεσης που οφείλεται κυρίως στην αύξηση του access time προς την L2.

Στις δυο γραφικές παραστάσεις που ακολουθούν βλέπουμε το miss rate της L2 cache για το benchmark μας για καθένα από τα δύο inputs.



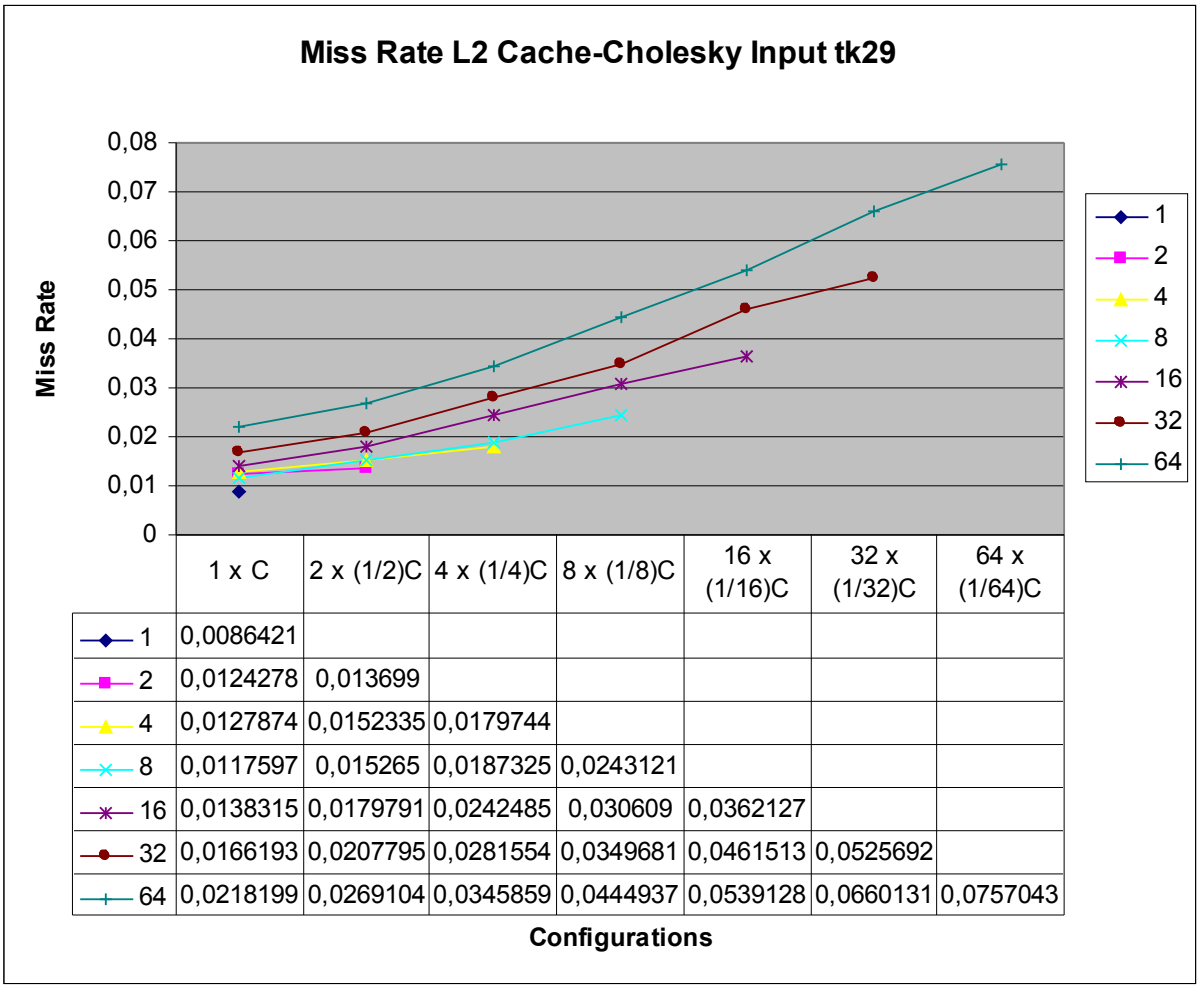
	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	0,01087							
2	0,00517	0,01128						
4	0,00502	0,00963	0,01503					
8	0,00511	0,00968	0,0146	0,02345				
16	0,00904	0,01338	0,02178	0,02949	0,03693			
32	0,00583	0,01004	0,01829	0,02562	0,03892	0,04683		
64	0,00679	0,01137	0,01914	0,02888	0,04268	0,05379	0,06343	
128	0,00861	0,01303	0,02116	0,03197	0,04612	0,05757	0,071	0,08676

Εικόνα 39:L2 Cache Miss Rate για το benchmark Cholesky με input d750

Αρχικά βλέπουμε πως επηρεάζει η αύξηση των επεξεργαστών και της συνολικής χωρητικότητας της L2 cache το miss rate για την εφαρμογή μας όταν έχουμε δώσει σαν input

το d750 (*εικόνα 39)*. Όπως αναμέναμε και από τα άλλα πειράματα που έχουμε κάνει, το configuration όπου η μνήμη είναι ενιαία για όλους τους επεξεργαστές μας δίνει τα καλύτερα αποτελέσματα. Επίσης βλέπουμε ότι σε κάθε περίπτωση το configuration όπου ο κάθε επεξεργαστής έχει την δική του ξεχωριστή cache έχει το μεγαλύτερο miss rate από τα άλλα configurations.Παρατηρούμε επίσης ότι όσο περισσότεροι επεξεργαστές μοιράζονται την μνήμη τόσο μικρότερο miss rate έχουμε. Από την μορφή της γραφικής παράστασης μας μπορούμε να πούμε ότι η συγκεκριμένη εφαρμογή με αυτό το input δεν έχει απαιτήσεις για L2 cache πολύ πέρα των 2MB, που είναι και η χωρητικότητα για τη δοκιμή μας με τον ένα επεξεργαστή. Το συμπέρασμα αυτό το βγάζουμε από το γεγονός ότι όσο και να μεγαλώσουμε την ενιαία L2 cache δεν έχουμε μεγάλη μείωση του miss rate. Αναφέρουμε την ενιαία μνήμη κρυφού επιπέδου διότι αυτή ξέρουμε από την θεωρεία να μας δίνει το μικρότερο miss rate αφού ελαχιστοποιεί τα conflict και capacity misses. Επιπλέον, αν μοιράσουμε την μνήμη μας σε παραπάνω από ένα επεξεργαστή έχουμε περισσότερα compulsory misses.Τα compulsory misses πιστεύουμε ότι ευθύνονται κυρίως για το αυξημένο miss rate στα configurations όπου η μνήμη cache διαμοιράζεται σε παραπάνω από 1 επεξεργαστές, με το μεγαλύτερο miss rate να το έχουμε όταν τρέχουμε την εφαρμογή μας για 128 CPUs και το κάθε CPU έχει την δική του μνήμη L2 cache.

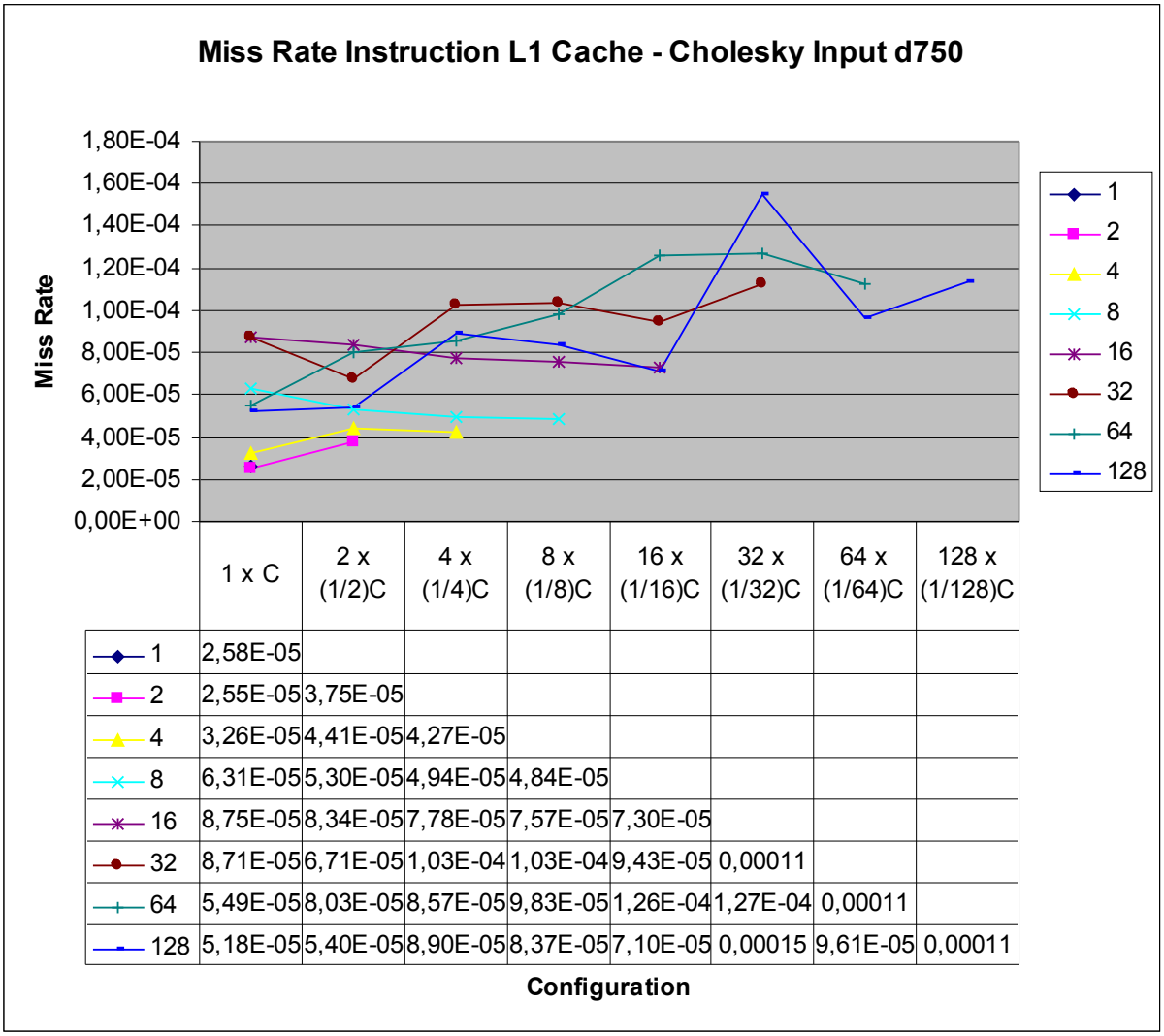
Αντίστοιχα αποτελέσματα παρατηρούμε και για την γραφική παράσταση που παρουσιάζει το miss rate για την εφαρμογή μας όταν έχουμε δώσει σαν input το tk29 (*εικόνα 40)*.



	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C
1	0,0086421						
2	0,0124278	0,013699					
4	0,0127874	0,0152335	0,0179744				
8	0,0117597	0,015265	0,0187325	0,0243121			
16	0,0138315	0,0179791	0,0242485	0,030609	0,0362127		
32	0,0166193	0,0207795	0,0281554	0,0349681	0,0461513	0,0525692	
64	0,0218199	0,0269104	0,0345859	0,0444937	0,0539128	0,0660131	0,0757043

Εικόνα 40: L2 Cache Miss Rate για το benchmark Cholesky με Input tk29

Στη συνέχεια παρουσιάζουμε την γραφική παράσταση για το miss rate για την L1 cache Instruction και για τα δυο inputs.

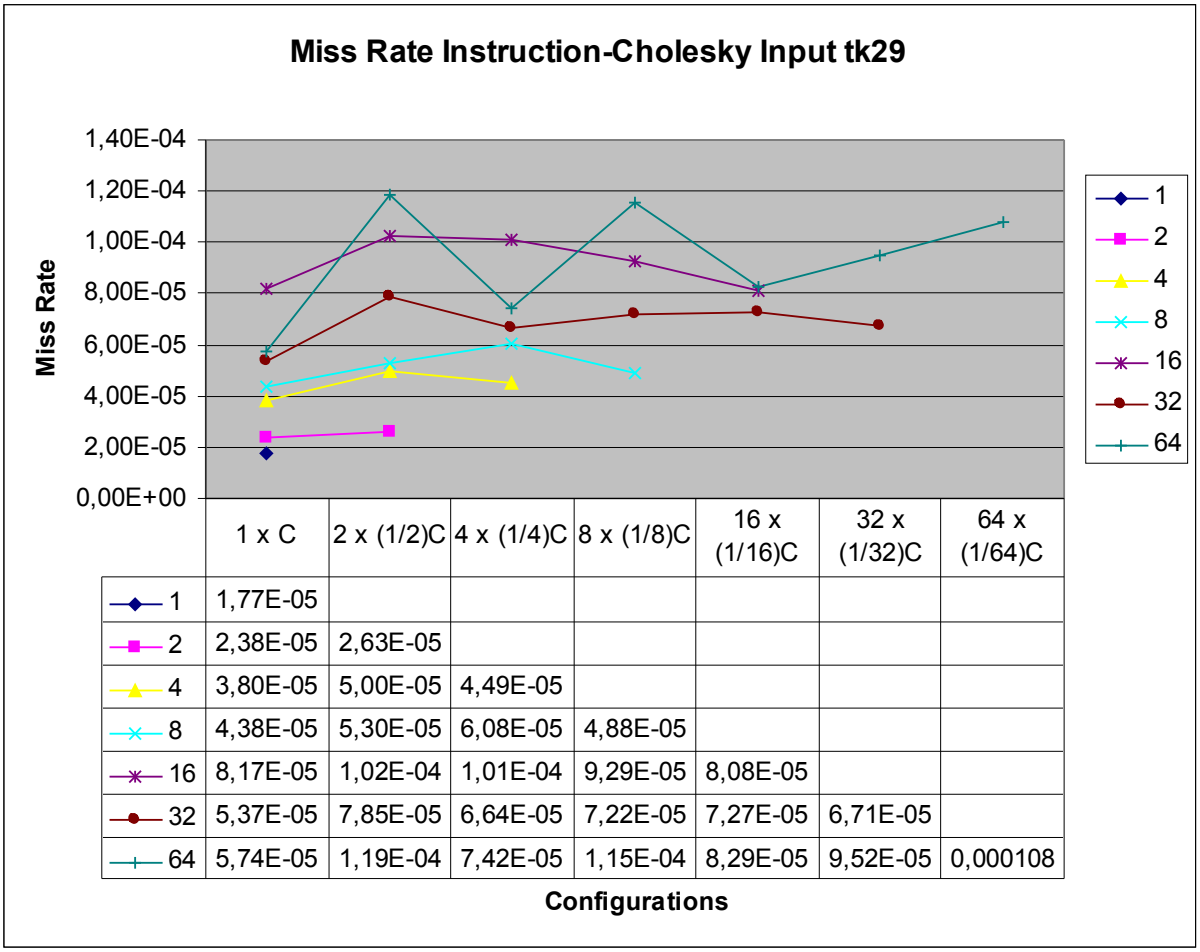


	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	2,58E-05							
2	2,55E-05	3,75E-05						
4	3,26E-05	4,41E-05	4,27E-05					
8	6,31E-05	5,30E-05	4,94E-05	4,84E-05				
16	8,75E-05	8,34E-05	7,78E-05	7,57E-05	7,30E-05			
32	8,71E-05	6,71E-05	1,03E-04	1,03E-04	9,43E-05	0,00011		
64	5,49E-05	8,03E-05	8,57E-05	9,83E-05	1,26E-04	1,27E-04	0,00011	
128	5,18E-05	5,40E-05	8,90E-05	8,37E-05	7,10E-05	0,00015	9,61E-05	0,00011

Εικόνα 41: L1 Instruction Cache Miss Rate για το benchmark Cholesky με Iinput d750

Παρατηρούμε ότι στη γραφική μας για το input d750 (*εικόνα 41*) ότι υπάρχει η τάση όσο μεγαλώνει ο αριθμός των επεξεργαστών να μεγαλώνει και το miss rate. Βλέπουμε ότι το ελάχιστο miss rate το έχουμε όταν εκτελούμε την εφαρμογή μας με ένα ή δυο επεξεργαστές και το μεγαλύτερο όταν τρέχουμε για 128 CPUs. Από τα παραπάνω μπορούμε να πούμε ότι η αρχική ποσότητα μνήμης L1 μας ήταν αρκετή αφού μπορέσαμε να κάνουμε fetch όσες εντολές χρειαστήκαμε για να εκτελέσουμε το πρόγραμμα μας και να έχουμε ταυτόχρονα το χαμηλό miss rate. Το χαμηλότερο miss rate παρατηρήθηκε όταν τρέξαμε την εφαρμογή μας για 2 επεξεργαστές, πράγμα που σημαίνει ότι όταν διπλασιάσαμε τους επεξεργαστές και μαζί και την συνολική χωρητικότητα της L1 cache μειώσαμε τα conflict και capacity misses αρκετά έτσι ώστε να μειωθεί το miss rate παρόλο που αναγκαστικά αυξήθηκαν τα compulsory misses. Αντίστοιχη μείωση με αυτή δεν παρατηρήσαμε για τα υπόλοιπα σενάρια

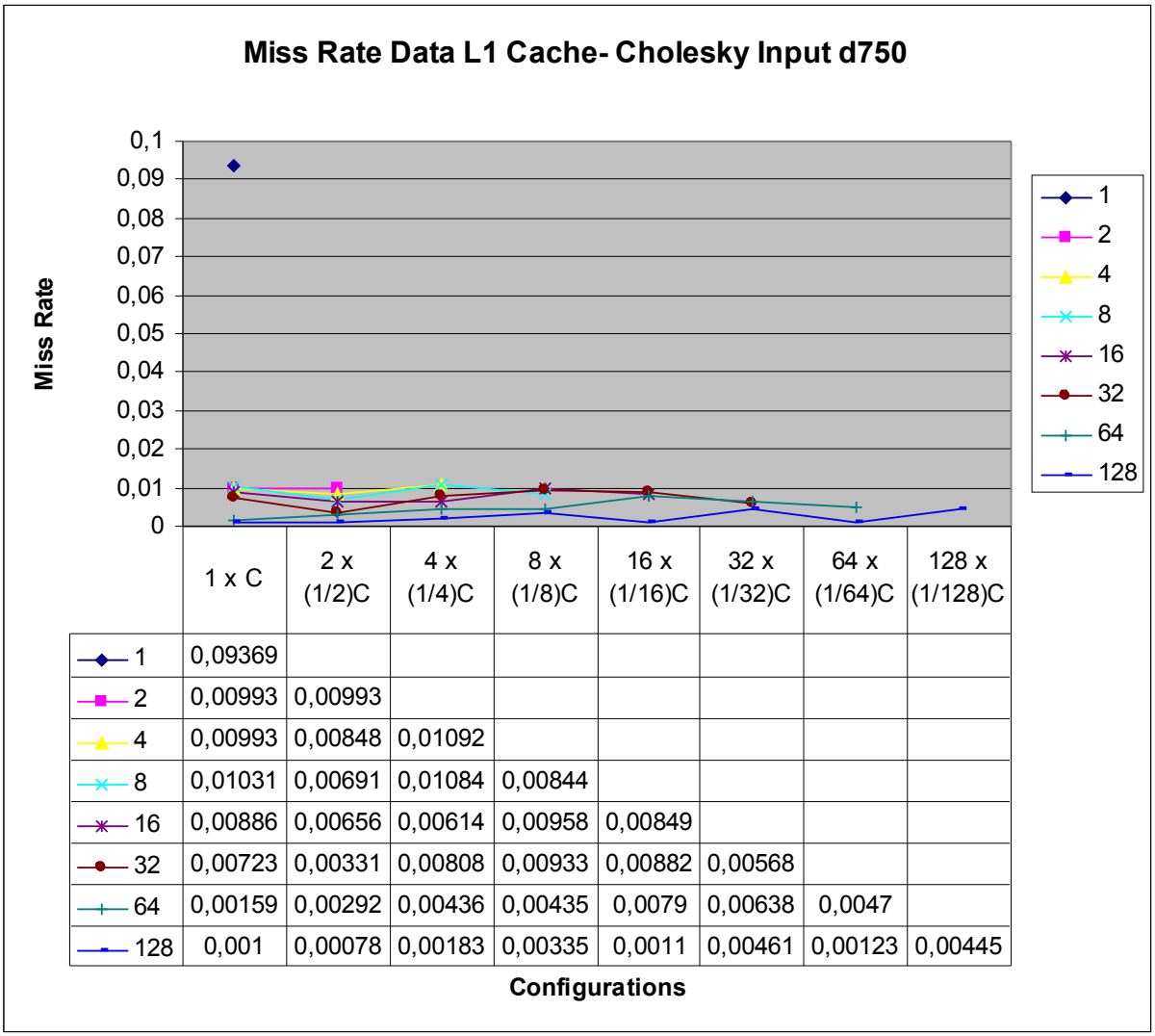
της δοκιμής μας, ενώ μπορούμε να πούμε ότι η αύξηση του miss rate οφείλεται κατά πολύ στα compulsory misses που προστίθενται κάθε φορά που δοκιμάζουμε να τρέξουμε την εφαρμογή μας για περισσότερους από 2 επεξεργαστές. Αντίστοιχη συμπεριφορά βλέπουμε να έχει και το miss rate για το input tk29 (*εικόνα 42)* όπως βλέπουμε στην γραφική παράσταση που ακολουθεί. Η κύρια διαφορά που μπορούμε να εντοπίσουμε είναι ότι για το νέο input το μικρότερο miss rate το πετυχαίνουμε όταν εκτελούμε την εφαρμογή μας για ένα επεξεργαστή. Αυτό σημαίνει ότι μεγαλύτερη L1 Instruction δεν μας μειώνει τόσο τα conflict και capacity misses ώστε να έχουμε μείωση του συνολικού miss rate,εφόσον γνωρίζουμε ότι προσθέτοντας νέες μονάδες L1 μνήμης μαζί με νέους επεξεργαστές αναπόφευκτα έχουμε και compulsory misses. Επίσης είναι πιθανό είδη από το την εκτέλεση του προγράμματος με ένα CPU να είχαμε τα λιγότερα δυνατά conflict και capacity misses, πράγμα που σημαίνει ότι η εφαρμογή μας έχει απαιτήσεις σε μνήμη L1 instruction cache χαμηλότερες από αυτές που προσφέρουμε εξαρχής και οποιαδήποτε αύξηση της μνήμης αυτής είναι άσκοπη.



	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C
1	1,77E-05						
2	2,38E-05	2,63E-05					
4	3,80E-05	5,00E-05	4,49E-05				
8	4,38E-05	5,30E-05	6,08E-05	4,88E-05			
16	8,17E-05	1,02E-04	1,01E-04	9,29E-05	8,08E-05		
32	5,37E-05	7,85E-05	6,64E-05	7,22E-05	7,27E-05	6,71E-05	
64	5,74E-05	1,19E-04	7,42E-05	1,15E-04	8,29E-05	9,52E-05	0,000108

Εικόνα 42: L1 Instruction Cache Miss Rate για το benchmark Cholesky με Input tk29

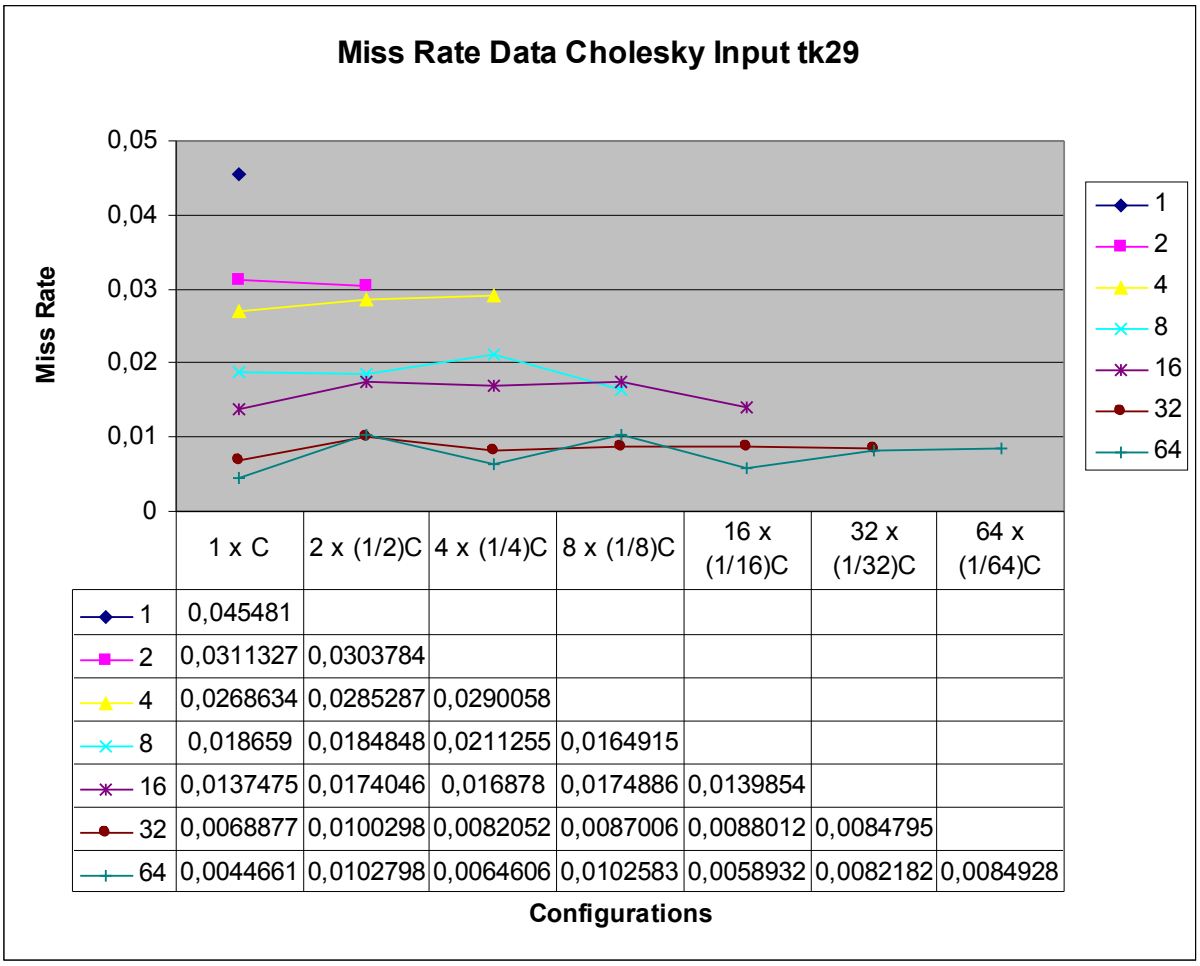
Ακολούθως βλέπουμε την γραφική παράσταση για το data L1 cache miss rate για την εφαρμογή Cholesky για το input d750 (*εικόνα 43)*. Όπως παρατηρούμε το μεγαλύτερο miss rate της εφαρμογής το έχουμε όταν εκτελούμε το πρόγραμμα μας με ένα επεξεργαστή. Βλέπουμε επίσης ότι αυξάνοντας την συνολική μνήμη L1, με την αύξηση των επεξεργαστών, επιτυγχάνουμε πολύ μικρότερο miss rate. Μπορούμε να πούμε συγκρίνοντας τα αποτελέσματα μας ότι αρχικά είχαμε μεγάλο miss rate λόγω της μικρής μνήμης που είχαμε, η οποία προκαλούσε πολλά conflict και capacity misses. Στη συνέχεια μειώσαμε τα misses αυτά με την προσθήκη μνήμης. Επίσης γνωρίζουμε ότι κάθε επεξεργαστής έχει αρμοδιότητα να εκτελέσει ένα συγκεκριμένο κομμάτι, υποσύνολο, του συνολικού προγράμματος. Έτσι η αντίστοιχη L1 Data cache πρέπει να έχει κάνει fetch το αντίστοιχο υποσύνολο των συνολικών δεδομένων μειώνοντας έτσι τον όγκο δεδομένων που πρέπει να υπάρχει ανά μνήμη L1 Data συμβάλλοντας στην περεταίρω μείωση του miss rate. Με αυτό τον τρόπο όσο αυξάνεται ο αριθμός των επεξεργαστών τόσο μειώνεται το miss rate για την L1 Data Cache.



	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C	128 x (1/128)C
1	0,09369							
2	0,00993	0,00993						
4	0,00993	0,00848	0,01092					
8	0,01031	0,00691	0,01084	0,00844				
16	0,00886	0,00656	0,00614	0,00958	0,00849			
32	0,00723	0,00331	0,00808	0,00933	0,00882	0,00568		
64	0,00159	0,00292	0,00436	0,00435	0,0079	0,00638	0,0047	
128	0,001	0,00078	0,00183	0,00335	0,0011	0,00461	0,00123	0,00445

Εικόνα 43: L1 Data Cache Miss Rate για το benchmark Cholesky με Input d750

Στη γραφική παράσταση που ακολουθεί βλέπουμε τα miss rates για την L1 Data cache για την εφαρμογή Cholesky ενώ έχουμε δώσει σαν input το tk29 (*εικόνα 44)*.



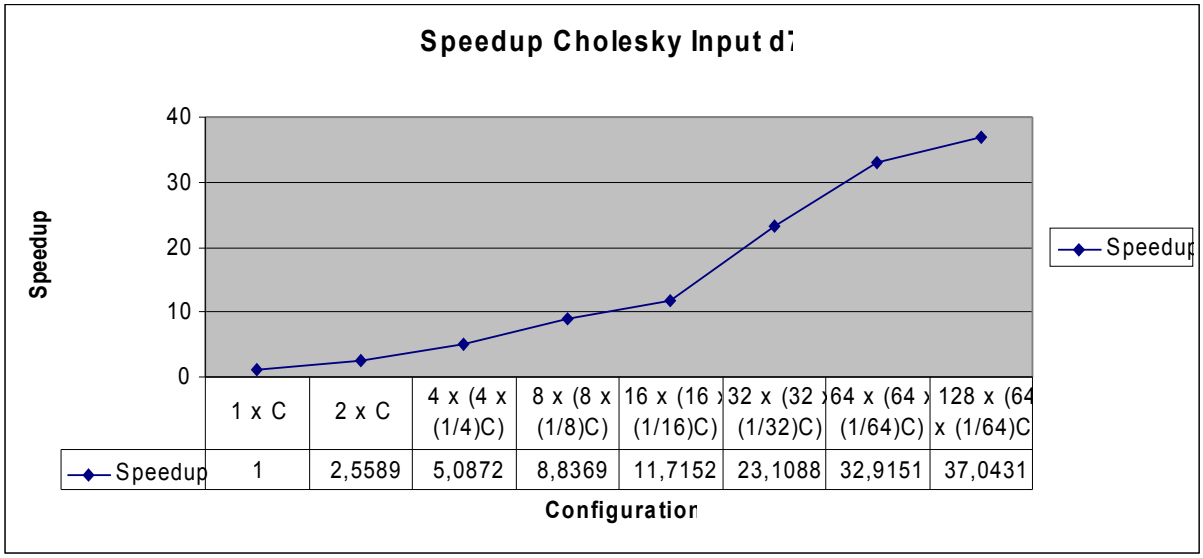
	1 x C	2 x (1/2)C	4 x (1/4)C	8 x (1/8)C	16 x (1/16)C	32 x (1/32)C	64 x (1/64)C
1	0,045481						
2	0,0311327	0,0303784					
4	0,0268634	0,0285287	0,0290058				
8	0,018659	0,0184848	0,0211255	0,0164915			
16	0,0137475	0,0174046	0,016878	0,0174886	0,0139854		
32	0,0068877	0,0100298	0,0082052	0,0087006	0,0088012	0,0084795	
64	0,0044661	0,0102798	0,0064606	0,0102583	0,0058932	0,0082182	0,0084928

Εικόνα 44: L1 Data Cache Miss Rate για το benchmark Cholesky με Input tk29

Από αυτή την γραφική μπορούμε να βγάλουμε τα ίδια συμπεράσματα όπως και στην προηγούμενη. Ιδιαίτερο ενδιαφέρον όμως έχει το γεγονός ότι το μεγαλύτερο input που χρησιμοποιήσαμε σε αυτή την δοκιμή μας μειώνει με διαφορετικό ρυθμό το miss rate.Αναλυτικότερα, στην προηγούμενη γραφική είδαμε ότι διπλασιάζοντας την συνολική μνήμη L1 Data μειώσαμε αρκετά το miss rate σε σημείο που περεταίρω αύξηση της μνήμης δεν μας άλλαξε ιδιαίτερα τα αποτελέσματα μας. Σε αυτή όμως την γραφική βλέπουμε ότι διπλασιάζοντας την μνήμη δεν μειώνουμε αρκετά το miss rate. Βλέπουμε λοιπόν ότι το miss rate μειώνεται σε κάθε αύξηση της μνήμης που κάνουμε εωσότου φτάσουμε στο σενάριο όπου χρησιμοποιούμε 32 επεξεργαστές. Από τα παραπάνω μπορούμε να βγάλουμε το συμπέρασμα ότι η μνήμη δεν είναι αρκετή, δημιουργώντας conflict και capacity misses, για τον μεγάλο όγκο των δεδομένων που χρειάζεται η εφαρμογή όταν για input δίνουμε το tk29, και αρκετή μνήμη έχουμε μόνο όταν εκτελέσουμε το πρόγραμμα μας για 32 επεξεργαστές. Βλέπουμε επίσης ότι μεγαλύτερη αύξηση της μνήμης δεν μας δίνει καλύτερα αποτελέσματα .

Πιθανό είναι η μείωση στα conflict και capacity misses που έχουμε να είναι κοντινή με την αύξηση των compulsory misses και για αυτό τον λόγω να μην έχουμε κάποια διαφορά στα αποτελέσματα μας. Είδαμε τέλος, ότι διαφορετικά inputs διαφοροποιούν την συμπεριφορά της L1 data cache μνήμης.

Στη συνέχεια βλέπουμε τις γραφικές παραστάσεις για το Speedup της εφαρμογής μας για τα δυο διαφορετικά inputs.

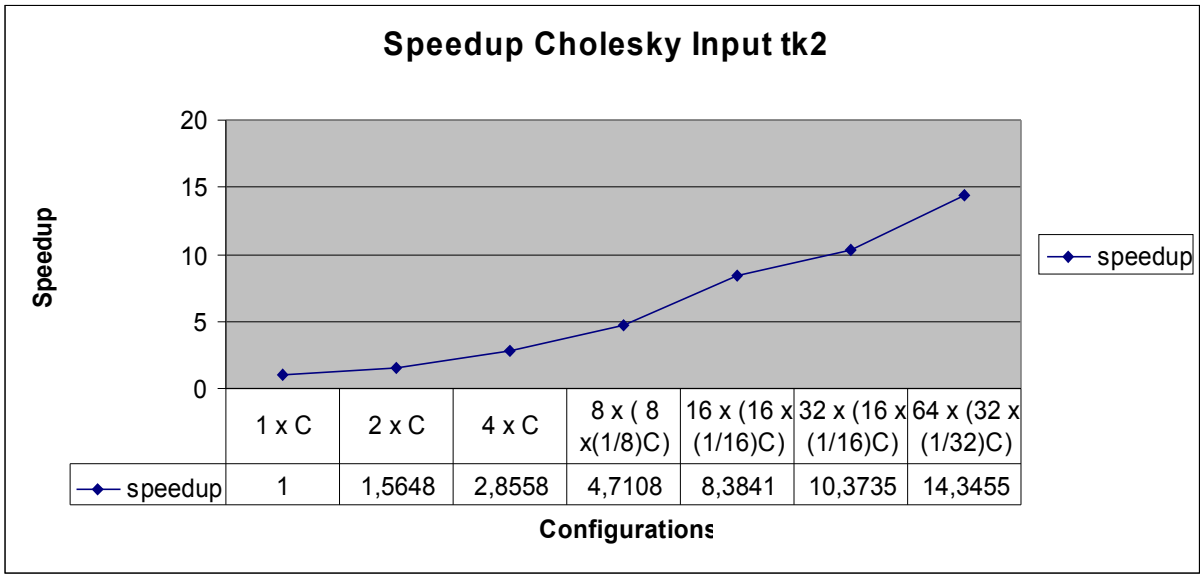


Εικόνα 45: Speedup για το benchmark Cholesky με Input d750

Αρχικά έχουμε το Speedup του προγράμματος μας για το input d750 (*εικόνα 45)*.Όπως παρατηρούμε με την αύξηση των επεξεργαστών που εκτελούν την εφαρμογή πετυχαίνουμε καλυτέρους χρόνους εκτέλεσης. Αναλυτικότερα, διπλασιάζοντας τον αριθμό των επεξεργαστών από ένα σε δυο, πετυχαίνουμε speedup ίσο με 2,5, για το configuration όπου οι δυο επεξεργαστές έχουν κοινή μνήμη L2 cache. Στη συνέχεια για 8 επεξεργαστές πετυχαίνουμε speedup 5 ενώ για 8 CPUs έχουμε speedup 8,8. Όπως είδαμε αρχικά έχουμε speedup που ξεπερνάει το μέγιστο speedup που μπορούμε θεωρητικά να πετύχουμε. Αυτό συμβαίνει διότι στη θεωρεία λαμβάνουμε υπόψη μας μόνο την προσθήκη επεξεργαστικής ισχύς ενώ εμείς σε κάθε διπλασιασμό του αριθμού των επεξεργαστών διπλασιάζουμε επίσης και την χωρητικότητα των L1 και L2 cache. Παρατηρούμε επίσης ότι για 16 επεξεργαστές έχουμε speedup 11,7 ενώ για 32 και 64 CPUs πετυχαίνουμε 23,1 και 32,9 speedup αντίστοιχα. Σημαντικό είναι να πούμε ότι για τα σενάρια 2,4,8,16,32 και 64 επεξεργαστών το

configuration μνήμης που μας δίνει τον καλύτερο χρόνο εκτέλεσης υλοποιείται με την μνήμη L2 cache να είναι ξεχωριστή για κάθε επεξεργαστή. Τέλος εκτελώντας την εφαρμογή για 128 CPUs πετύχαμε speedup 37 ενώ η κρυφή μνήμη επιπέδου 2 είναι κοινή για κάθε δυο επεξεργαστές.

Ακολούθως βλέπουμε το Speedup της εφαρμογής Cholesky για το input tk29 (*εικόνα 46)*. Παρατηρούμε ότι η διαφορά στο input έχει επηρεάσει το speedup της εφαρμογής μας. Βλέπουμε ότι αυξάνοντας τους επεξεργαστές σε 2 και 4 πετυχαίνουμε speedup 1,5 και 2,8 αντίστοιχα. Όπως είδαμε με μικρότερο το input d750 είχαμε speedup 2,5 και 5 για τον ίδιο αριθμό επεξεργαστών. Σημαντικό είναι να πούμε ότι για το input tk29 το speedup για τους 2 και 4 επεξεργαστές επιτυγχάνεται όταν η μνήμη L2 cache είναι κοινή για όλα τα CPUs. Speedup ίσο με 4,7 και 8,3 πετυχαίνουμε όταν εκτελούμε την εφαρμογή μας για 8 και 16 επεξεργαστές όπως βλέπουμε και από την γραφική μας. Για τους 8 και 16 επεξεργαστές η μνήμη L2 cache είναι ξεχωριστή για κάθε CPU. Αντίστοιχα για το προηγούμενο input για 8 και 16 επεξεργαστές είχαμε speedup 8,8 και 11,7.Τέλος για 32 και 64 επεξεργαστές έχουμε speedup 10,3 και 14,3 ενώ αντίστοιχα με το μικρότερο input d750 είχαμε 23,1 και 32,9. Για το input tk29 τους χρόνους εκτέλεσης που μας δίνουν speedup 10,3 για 32 επεξεργαστές και 14,3 για 64 CPUs τα πετύχαμε μοιράζοντας την L2 cache ανά 2 επεξεργαστές.



Εικόνα 46: Speedup για το benchmark Cholesky με Input tk29

Τα πειράματα που κάναμε για το benchmark Cholesky μας έδειξαν ότι μεγαλύτερα inputs αυξάνουν τον χρόνο εκτέλεσης. Επίσης είδαμε ότι σε κάθε input έχουμε διαφορετικά configurations όπου συμπεριφέροντε καλύτερα, το καθένα ανάλογα με τις ανάγκες του σε μνήμη. Γενικά είδαμε ότι και σε αυτή την εφαρμογή η αύξηση του access time προς την μνήμη, με την αύξηση της χωρητικότητας της, παίζει καταλυτικό ρόλο στον χρόνο εκτέλεσης. Αυτό έχει σαν αποτέλεσμα τα configuration όπου η μνήμη διαμοιράζεται μεταξύ πολλών επεξεργαστών να μην πετυχαίνουν καλούς χρόνους εκτέλεσης ειδικά όσο μεγαλώνει ο αριθμός των επεξεργαστών. Τέλος τα configurations που είδαμε ότι πετυχαίνουν τους καλύτερους χρόνους εκτέλεσης είναι το configuration όπου ο κάθε επεξεργαστής έχει το δικό του L2 cache και το configuration όπου η μνήμη L2 cache διαμοιράζεται ανά δυο επεξεργαστές. Το παραπάνω συμβαίνει διότι όσο ο αριθμός των επεξεργαστών μικρός τα configurations δεν έχουν μεταξύ τους ιδιαίτερες διαφορές στο miss rate για την μνήμη L2. Καθώς όμως αυξάνεται ο αριθμός των CPUs και μαζί με αυτόν και το miss rate στην L2 cache, λόγω των compulsory misses, το κόστος για τα misses αυτά είναι μεγάλο και το μετριάζουμε διαλέγοντας ένα configuration που έχει λιγότερα misses, παρόλο που η νέα διάταξη έχει λίγο μεγαλύτερο access time προς την μνήμη.

Κεφάλαιο 5

Συμπεράσματα – Μελλοντική εργασία

5.1 Συμπεράσματα 73
5.2 Μελλοντική εργασία 75

5.1 Συμπεράσματα

Σε αυτή την εργασία εκτελέσαμε μια σειρά από benchmarks με μια πληθώρα από διαφορετικά configuration μνήμης L2 cache με σκοπό να βρούμε το πιο αποδοτικό σενάριο.

Αρχικά παρατηρήσαμε ότι αύξηση της χωρητικότητας της L2 μνήμης έχει σαν αποτέλεσμα την μείωση των miss rates σε μεγάλο βαθμό. Επίσης αύξησης της συνολικής χωρητικότητας της L1 cache έχει σαν αποτέλεσμα την μείωση των miss rates για το κομμάτι των δεδομένων και την αύξηση για το κομμάτι των εντολών. Τα παραπάνω , περιμέναμε ότι θα τα συναντήσουμε από τις θεωρητικές γνώσεις που έχουμε αποκτήσει κατά την διάρκεια των σπουδών μας.

Είδαμε επίσης ότι η αύξηση της επίδοσης διέφερε από εφαρμογή σε εφαρμογή. Αυτό μας έκανε να συμπεράνουμε ότι δεν έχουν όλα τα προγράμματα τόσο αποδοτικό παράλληλο κώδικα ώστε να μπορούν να αποδώσουν το ίδιο καλά σε ενδεχόμενη αύξηση των επεξεργαστών.

Ένα άλλο χαρακτηριστικό των εφαρμογών που εκτελέσαμε είναι ότι αλλάζοντας το input που δίνετε για εκτέλεση αλλάζει σημαντικά το execution time της εφαρμογής. Το δεδομένο αυτό μας κάνει να συμπεράνουμε ότι εκτός από τον κώδικα του προγράμματος σημαντικό ρόλο στην διαμόρφωση του χρόνου εκτέλεσης παίζουν και τα δεδομένα που δέχεται η εφαρμογή σαν είσοδο και πόσο συχνά αυτά χρειάζονται.

Με βάση το παραπάνω προσπαθήσαμε να βρούμε ένα configuration μνήμης το οποίο θα αποθηκεύει με τον καλύτερο δυνατό τρόπο (έχοντας δηλαδή τα λιγότερα misses) τα δεδομένα που χρειάζεται η κάθε εφαρμογή. Κατά την εκτέλεση των διαφόρων benchmarks είδαμε ότι κάθε benchmark έχει δική του συμπεριφορά και δεν υπάρχει μια μόνο απάντηση στο ποιο είναι το πιο αποδοτικό configuration για όλες τις εφαρμογές.

Εξαρχής γνωρίζαμε ότι τα configurations τα οποία διαμοιράζουν μεγάλα σε όγκο κομμάτια μνήμης στους επεξεργαστές πιθανόν να δώσουν χειρότερο χρόνο εκτέλεσης από configurations όπου η μνήμη έχει μικρότερο όγκο. Το παραπάνω συμβαίνει λόγω του ότι μεγαλώνει σε όγκο μια μνήμη μεγαλώνει επίσης και το access time πράγμα που επηρεάζει οποιαδήποτε ενέργεια κάνουμε προς την μνήμη και στην περίπτωση μας το Hit Delay όπως και το Miss Delay.Όντως το configuration όπου η μνήμη ήταν κοινή για όλους τους επεξεργαστές μας έδινε πάντα τον χειρότερο χρόνο εκτέλεσης ειδικά για τις δοκιμές όπου είχαμε μεγάλο αριθμό επεξεργαστών.

Είδαμε λοιπόν, ότι τα σενάρια τα οποία απόδωσαν καλύτερα στις δοκιμές μας ήταν τα σενάρια όπου η μνήμη L2 cache ήταν ξεχωριστή για κάθε επεξεργαστή ή ήταν κοινή για μικρό αριθμό επεξεργαστών. Αναλυτικότερα, από τα πειράματα μας είδαμε ότι όταν εκτελούσαμε τις εφαρμογές μας για μικρό αριθμό επεξεργαστών (μέχρι 8 επεξεργαστές) το καλύτερο configuration ήταν συνήθως αυτό όπου η μνήμη ήταν ξεχωριστή για κάθε CPU. Για τα πειράματα όμως που εκτελούνταν για παραπάνω από 16 επεξεργαστές είδαμε ότι πετυχαίναμε καλύτερους χρόνους εκτέλεσης όταν η μνήμη cache διαμοιραζόταν κυρίως σε δυο και τέσσερις επεξεργαστές.

	LU	Radix	FFT	Cholesky d750	Cholesky tk29
2 CPUs	0	0	0	0	2
4 CPUs	0	0	0	0	4
8 CPUs	0	0	0	0	0
16 CPUs	2	2	4	0	0
32 CPUs	2	4	4	0	2
64 CPUs	0	8	4	0	2
128 CPUs	2		4	2	

Πίνακας 4: Αριθμός επεξεργαστών που διαμοιράζονται την μνήμη στο configuration που πετυχαίνει τον καλύτερο χρόνο εκτέλεσης για κάθε αριθμό επεξεργαστών και κάθε benchmark.

Παρατηρώντας και τον παραπάνω πίνακα μπορούμε να πούμε ότι για μικρό αριθμό επεξεργαστών είχαμε καλύτερα αποτελέσματα όταν η μνήμη L2 ήταν ξεχωριστή για κάθε επεξεργαστή, επειδή σε αυτά τα σενάρια το miss rate για το συγκεκριμένο configuration είναι σε χαμηλά επίπεδα και κοντά στο miss rate που έχουν τα configurations όπου η μνήμη διαμοιράζεται, χωρίς όμως να έχει το υψηλό access time αυτών των configurations. Μεγαλώνοντας όμως τον αριθμό των επεξεργαστών που εκτελούν την κάθε εφαρμογή βλέπουμε ότι τα configurations που δίνουν τον καλύτερο χρόνο εκτέλεσης είναι αυτά όπου πλέον η μνήμη L2 διαμοιράζεται μεταξύ 2 , 4 ή και σε μια περίπτωση 8 επεξεργαστών. Αυτό συμβαίνει διότι όπως έχουμε δει όταν αυξάνουμε τους επεξεργαστές που εκτελούν τις εφαρμογές μας, το configuration που η μνήμη είναι ξεχωριστή για κάθε CPU μας δίνει το μεγαλύτερο miss rate. Το miss rate αυτό, παίζει πλέον σημαντικό ρόλο στη διαμόρφωση του χρόνου εκτέλεσης και έτσι έχουμε configurations που πετυχαίνουν καλύτερους χρόνους εκτέλεσης λόγω του μικρότερου miss rate, ανεξαρτήτως του γεγονότος ότι έχουν μεγαλύτερο access time προς την μνήμη και άρα μεγαλύτερο Hit και Miss Delay.

Πιστεύουμε επίσης ότι αν εκτελούσαμε πειράματα με εφαρμογές που είχαν ακόμα μεγαλύτερες ανάγκες σε μνήμη από τις εφαρμογές που χρησιμοποιήσαμε, θα βρίσκαμε ότι τα configurations που θα μας έδιναν την καλύτερη δυνατή επίδοση θα ήταν αυτά όπου η μνήμη θα διαμοιραζόταν από μεγαλύτερο αριθμό επεξεργαστών από αυτών που συναντήσαμε στα πειράματα μας.

Με βάση τα παραπάνω μπορούμε να πούμε ότι δεν υπάρχει ένα καλύτερο configuration, αλλά το configuration το οποίο είναι καλύτερο για τις συνθήκες της κάθε δοκιμής και τις ανάγκες κάθε εφαρμογής.

5.2 Μελλοντική Εργασία

Πιστεύουμε ότι στο μέλλον θα μπορούσαν να γίνουν πειράματα με διαφορετικά configurations από αυτά που χρησιμοποιήσαμε εμείς για την διπλωματική εργασία μας.

Ένα πιθανό σενάριο που θα μπορούσε να εξεταστεί στο μέλλον θα ήταν η χωρητικότητα της L2 cache να παραμένει σταθερή ανεξάρτητα με τον αριθμό των επεξεργαστών που χρησιμοποιούμε για την εκτέλεση των benchmark μας. Επίσης θα μπορούσαμε εκτός από

την μνήμη L2 cache να τροποποιούμε κατάλληλα και την μνήμη L1 cache ώστε να δούμε πως πιθανή αλλαγή της, επηρεάζει την συμπεριφορά των προγραμμάτων μας. Αναλυτικότερα θα μπορούσαμε να μοιράζεται η μνήμη L1 μεταξύ των επεξεργαστών, σε διαφορετικό αριθμό CPUs κάθε φορά. Αρχικά , θα μπορούσε να έχει κάθε επεξεργαστής δική του μνήμη L1 cache και όσο αλλάζουμε τα configurations να έχουμε κοινή L1 cache για 2, 4, 8 , …,128 επεξεργαστές.

Βιβλιογραφία

[1] R. Varada, M. Sriram, K. Chou, and J. Guzoo , Design and Integration Methods for a Multi-threaded Dual Core 65nm Xeon Processor, 2006, www.acm.org.

[2] R. Boneti, J. Cazorla, R. Gioiosa ,A. Buyuktosunoglu, ,Ch. Cher, and M. Valero, Software-Controlled Priority Characterization of POWER5 Processor, 2008, IEEE, www.acm.org.

[3] T. Mattson, Wijnagaart and Rvd. Frumkin, M.Programming the Intel 80-core network-on-a-chip Terascale Processor, 2007, www.acm.org.

[4] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, P. Juan and P. Hanrahan. Larrabee: A Many-core x86 Architecture for Visual Computing, 2008, www.acm.org.

[5] L.A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qaderer, B. Sano, S. Smith, R.Stete and B. Verghese, Piranha: A scalable Architecture Based on Single-Chip Multiprocessing , 2000, www.acm.org

[6] K. Barker, K. Davis, A. Hoise, D. Kerbyson, M. Lang, S. Pakin, S. Sancho, Entering the Penataflop Era: The Architecture and Performance of Roadrunner, www.acm.org

[7] B. Rogers, A. Krishna, G. Bell, K. Vu, X. Jiang ,Y. Solihin, Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling, www.acm.org.

[8] J. L. Hennessy and D. Patterson, Computer Architecture : A Quantitative Approach, 4th edition, Morgan Kaufmann Publishers,

Παράρτημα Α

Στη συνέχεια παραθέτουμε ένα configuration file. Το συγκεκριμένο μας είναι για 8 επεξεργαστές όπου η μνήμη L2 cache είναι κοινή ανά 4 επεξεργαστές.

```
procsPerNode  = 8
cacheLineSize = 32

issue        = 4     # processor issue width
cpucore[0:$(procsPerNode)-1] = 'issueX'

#<shared.conf> (contents below)

################################
# SYSTEM                       #
################################

enableICache   = true
NoMigration    = true
tech           = 0.10
pageSize       = 4096
fetchPolicy    = 'outorder'
issueWrongPath = true

technology = 'techParam'

################################
# clock-panalyzer input        #
################################
[techParam]
clockTreeStyle = 1               # 1 for Htree or 2 for balHtree
```

```
tech       = 32                    # nm
frequency  = 5e9                   # Hz
skewBudget = 20                            # in ps
areaOfChip = 200                   # in mm^2
loadInClockNode = 20                       # in pF
optimalNumberOfBuffer = 3

################################
# PROCESSORS' CONFIGURATION  #
################################

[issueX]
frequency       = 5e9
areaFactor      = ($(issue)*$(issue)+0.1)/16  # Area compared to Alpha264 EV6
inorder         = false
fetchWidth      = $(issue)
instQueueSize   = 2*$(issue)
issueWidth      = $(issue)
retireWidth     = $(issue)+1
decodeDelay     = 6
renameDelay     = 3
wakeupDelay     = 6                # -> 6+3+6+1+1=17 branch mispred. penalty
maxBranches     = 16*$(issue)
bb4Cycle        = 1
maxIRequests    = 4
interClusterLat = 2
intraClusterLat = 1
cluster[0]      = 'FXClusterIssueX'
cluster[1]      = 'FPClusterIssueX'
stForwardDelay  = 2
maxLoads        = 10*$(issue)+16
maxStores       = 10*$(issue)+16
regFileDelay    = 3
```

```
robSize         = 36*$(issue)+32
intRegs         = 32+16*$(issue)
fpRegs          = 32+12*$(issue)
bpred           = 'BPredIssueX'
dtlb            = 'FXDTLB'
itlb            = 'FXITLB'
dataSource      = "DMemory DL1"
instrSource     = "IMemory IL1"
enableICache    = true
OSType          = 'dummy'


# integer functional units

[FXClusterIssueX]
winSize    = 12*$(Issue)+32 # number of entries in window
recycleAt  = 'Execute'
schedNumPorts = 4
schedPortOccp = 1
wakeUpNumPorts= 4
wakeUpPortOccp= 1
wakeupDelay   = 3
schedDelay    = 1 # Minimum latency like a intraClusterLat
iStoreLat  = 1
iStoreUnit = 'LDSTIssueX'
iLoadLat   = 1
iLoadUnit  = 'LDSTIssueX'
iALULat    = 1
iALUUnit   = 'ALUIssueX'
iBJLat     = 1
iBJUnit    = 'ALUIssueX'
iDivLat    = 12
iDivUnit   = 'ALUIssueX'
```

```
iMultLat   = 4
iMultUnit  = 'ALUIssueX'

[LDSTIssueX]
Num = $(issue)/3+1
Occ = 1

[ALUIssueX]
Num = $(issue)/3+1
Occ = 1

# floating point functional units

[FPClusterIssueX]
winSize    = 8*$(issue)
recycleAt  = 'Execute'
schedNumPorts = 4
schedPortOccp = 1
wakeUpNumPorts= 4
wakeUpPortOccp= 1
wakeupDelay   = 3
schedDelay    = 1 # Minimum latency like a intraClusterLat
fpALULat   = 1
fpALUUnit  = 'FPIssueX'
fpMultLat  = 2
fpMultUnit = 'FPIssueX'
fpDivLat   = 10
fpDivUnit  = 'FPIssueX'

[FPIssueX]
Num = $(issue)/2+1
Occ = 1
```

```
# branch prediction mechanism

[BPredIssueX]
type         = "hybrid"
BTACDelay    = 0
l1size       = 1
l2size       = 16*1024
l2Bits       = 1
historySize  = 11
Metasize     = 16*1024
MetaBits     = 2
localSize    = 16*1024
localBits    = 2
btbSize      = 2048
btbBsize     = 1
btbAssoc     = 2
btbReplPolicy = 'LRU'
btbHistory   = 0
rasSize      = 32

# memory translation mechanism

[FXDTLB]
deviceType = 'cache'
size       = 64*8
assoc      = 4
bsize      = 8
numPorts   = 2
replPolicy = 'LRU'

[FXITLB]
deviceType = 'cache'
size       = 64*8
```

```
assoc      = 4
bsize      = 8
numPorts   = 2
replPolicy = 'LRU'

################################
# MEMORY SUBSYSTEM           #
################################

# instruction source
[IMemory]
deviceType    = 'icache'
size          = 32*1024
assoc         = 2
bsize         = $(cacheLineSize)
writePolicy   = 'WT'
replPolicy    = 'LRU'
numPorts      = 2
portOccp      = 1
hitDelay      = 2
missDelay     = 1               # this number is added to the hitDelay
MSHR          = "iMSHR"
lowerLevel    = "L1L2Bus L1L2"

[iMSHR]
type = 'single'
size = 32
bsize = $(cacheLineSize)

# data source
[DMemory]
deviceType    = 'cache'
size          = 32*1024
```

```
assoc        = 4
bsize        = $(cacheLineSize)
writePolicy  = 'WT'
replPolicy   = 'LRU'
numPorts     = $(issue)/3+1
portOccp     = 1
hitDelay     = 2
missDelay    = 1              #this number is added to the hitDelay
maxWrites    = 8
MSHR         = "DMSHR"
lowerLevel   = "L1L2Bus L1L2"

[DMSHR]
type = 'single'
size = 64
bsize = $(cacheLineSize)

# bus between L1s and L2
[L1L2Bus]
deviceType = 'bus'
numPorts   = 1
portOccp   = 1                # assuming 256 bit bus
delay      = 1
lowerLevel = "L2Cache L2 sharedBy 4"

# private L2
[L2Cache]
deviceType   = 'smpcache'
size         = 8192*1024
assoc        = 8
bsize        = $(cacheLineSize)
writePolicy  = 'WB'
replPolicy   = 'LRU'
```

```
protocol      = 'MESI'
numPorts      = 2               # one for L1, one for snooping
portOccp      = 2
hitDelay      = 9
missDelay     = 11              # exclusive, i.e., not added to hitDelay
displNotify   = false
MSHR          = 'L2MSHR'
lowerLevel    = "SystemBus SysBus sharedBy 2"

[L2MSHR]
size =  64
type = 'single'
bsize = $(cacheLineSize)

[SystemBus]
deviceType    = 'systembus'
numPorts      = 1
portOccp      = 1
delay         = 1
lowerLevel    = "MemoryBus MemoryBus"
BusEnergy     = 0.03

[MemoryBus]
deviceType    = 'bus'
numPorts      = 1
portOccp      = $(cacheLineSize) / 4   # assuming 4 bytes/cycle bw
delay         = 15
lowerLevel    = "Memory Memory"

[Memory]
deviceType    = 'niceCache'
size          = 64
assoc         = 1
```

```
bsize        = 64
writePolicy   = 'WB'
replPolicy    = 'LRU'
numPorts      = 1
portOccp      = 1
hitDelay      = 500 - 31  # 5.0GHz: 100ns is 500 cycles RTT - 16 busData
missDelay     = 500 - 31  # - 15 memory bus => 500 - 31
MSHR          = NoMSHR
lowerLevel    = 'voidDevice'

[NoMSHR]
type = 'none'
size = 128
bsize = 64

[voidDevice]
deviceType    = 'void'

############################
#     BEGIN MIPSEMUL       #
############################

[FileSys]
mount=""
```