Βαμβακίδης Δημήτρης - 9501
Δημητρίου Μάριος - 9235

Αρχιτεκτονική Υπολογιστών

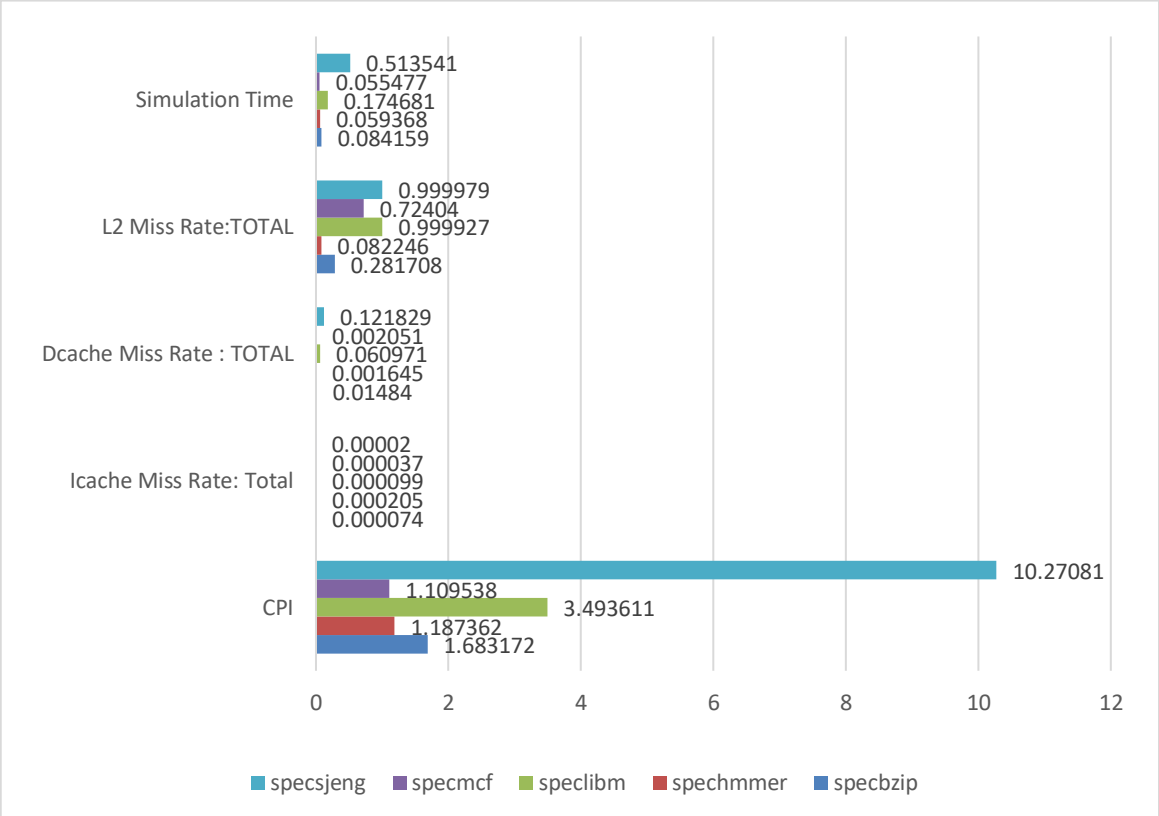
Εργαστήριο 2

Βήμα 1 – Ερώτημα 1

Ανατρέχοντας στα αρχεία που δημιουργούνται από την εκτέλεση των benchmarks που μας δόθηκαν, μπορούμε να βρούμε αναλυτικά τα χαρακτηριστικά του κάθε επεξεργαστή. Για το κάθε benchmark έχουμε τα παρακάτω χαρακτηριστικά σύμφωνα με το config.ini αρχείο.

L1 Instruction cache size	32KB
L1 Instruction Associativity	2
L1 Data cache size	64KB
L1 Data Associativity	2
L2 Cache Size	2MB
L2 Associativity	8
Cache Line	64

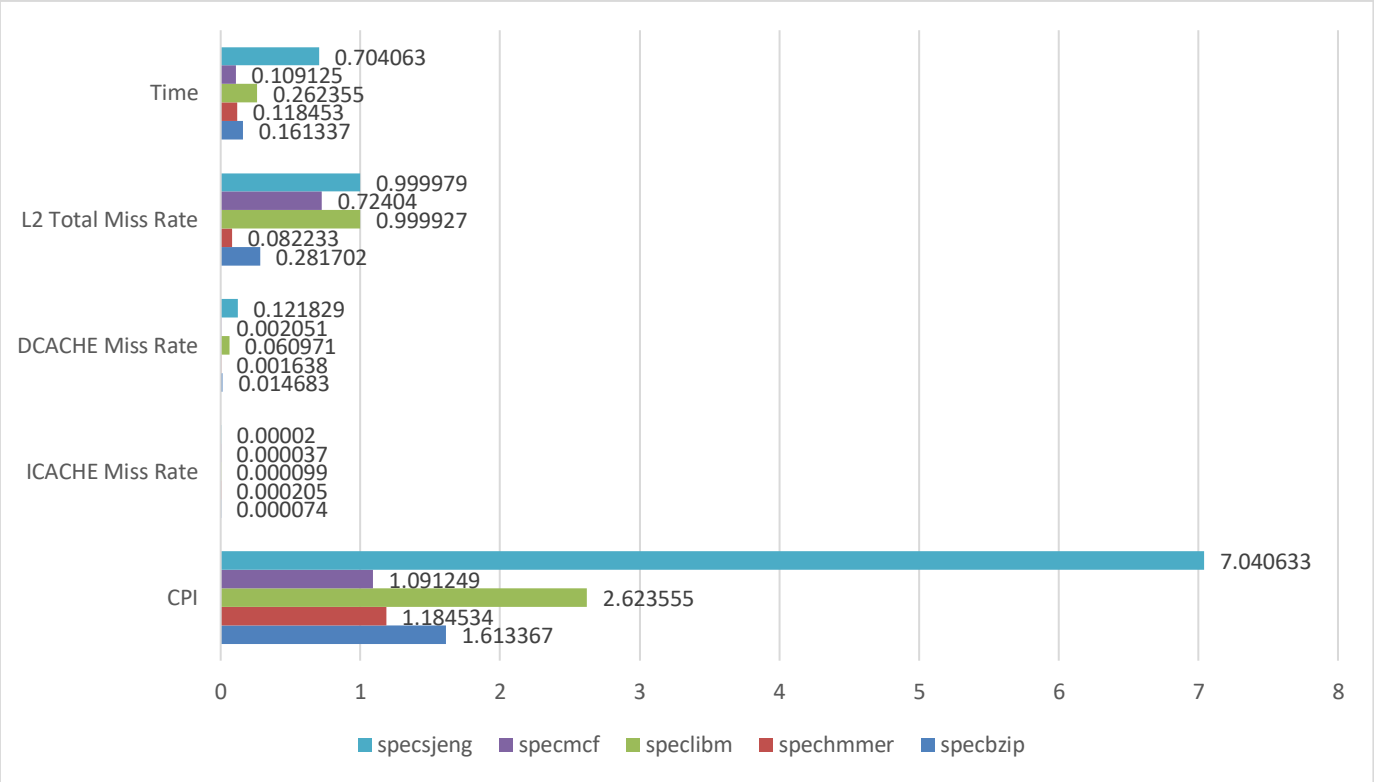
Ερώτημα 2



Όπως φαίνεται και από το παραπάνω γράφημα ο καθοριστικός παράγοντας για την αύξηση του CPI στα specsjeng και speclibm benchmarks είναι τα misses στην L2 Cache τα οποία αγγίζουν τη μονάδα για τα δύο αυτά benchmarks. Το παραπάνω επιφέρει και την αντίστοιχη αύξηση στο πραγματικό χρόνο εκτέλεσης του εκάστοτε προγράμματος. Όσο αφορά το miss rate της dcache αυτό αυξάνεται μόνο για το specbjeng benchmark.

Βαμβακίδης Δημήτρης - 9501
Δημητρίου Μάριος - 9235

Ερώτημα 3



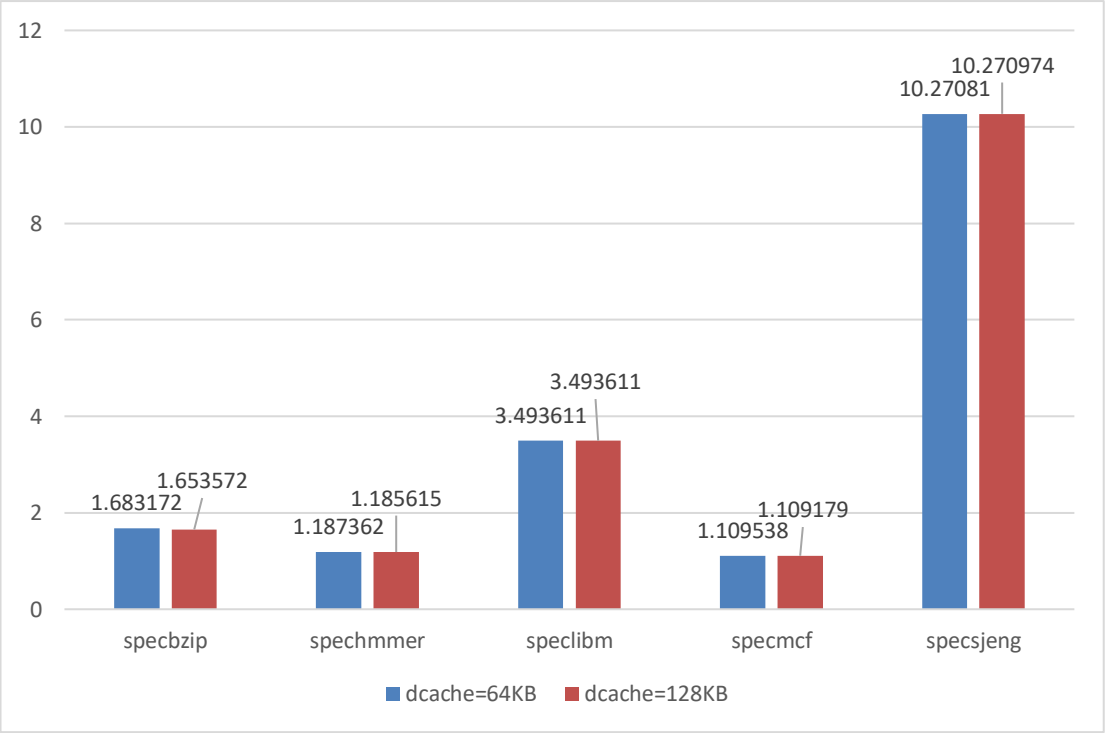
Από τις παραπάνω μετρήσεις φαίνεται ότι με υποδιπλασιασμό του χρονισμού του επεξεργαστή πράγματι αυξάνεται ο χρόνος εκτέλεσης του προγράμματος, σε ορισμένες περιπτώσεις μάλιστα, πολύ κοντά στο 2. Αξίζει να σημειωθεί ότι το CPI φαίνεται να μειώνεται ωστόσο δεν επαρκής η εξέταση μόνο αυτού του στοιχείου αλλά και η αντιπαραβολή του με το ρολόι του επεξεργαστή και κατεπέκταση με το συνολικό αριθμό ticks simulated που υποδηλώνουν ακριβώς τη καθυστέρηση που προαναφέρθηκε στην εκτέλεση του benchmark. Όσο αφορά τα δύο διαφορετικά ρολόγια που συναντιούνται στο stats.txt η παράμετρος που χρησιμοποιήθηκε επηρεάζει μόνο το cpucluster.clk_domain.clock το οποίο αναφέρεται στο χρονισμό του επεξεργαστή καθώς και του bus επικοινωνίας του επεξεργαστή με την L1 cache. Το system.clk_domain.clock αποτελεί το χρονισμό του συστήματος και για τη περίπτωση που μας ενδιαφέρει, το χρονισμό του bus της επικοινωνίας L2-L1 cache. Μπορεί δηλαδή να αποτελέσει bottleneck στην εκτέλεση του προγράμματος σε σχέση με τις δυνατότητες του επεξεργαστή. Για το λόγο αυτό ο συνεχόμενος πολλαπλασιασμός του ρολογιού ακόμα και μόνο διπλασιασμός του δεν οδηγεί σε τέλειο scaling αφού άλλοι παράμετροι του συστήματος καθυστερούν την εκτέλεση του προγράμματος(system clock). Τέλος, σε περίπτωση πρόσθεσης κι' άλλου επεξεργαστή αυτός θα υιοθετήσει σύμφωνα με τη δημιουργία του cpu-cluster το ρολόι των 2GHz.

Βήμα 2

Στόχος του συγκεκριμένου βήματος ήταν η βελτιστοποίηση των αρχιτεκτονικών επιλογών ενός MinorCPU μοντέλου για την επίτευξη του χαμηλότερου δυνατού CPI(όσο πιο κοντά στο 1 γίνεται). Καθώς οι πιθανές επιλογές ήταν αρκετές κριτήριο για την τακτική που ακολουθήθηκε αποτέλεσαν τα αποτελέσματα της πρώτης σειράς από benchmarks με default τιμές για τα υπόλοιπα στοιχεία και 2GHz base clock.

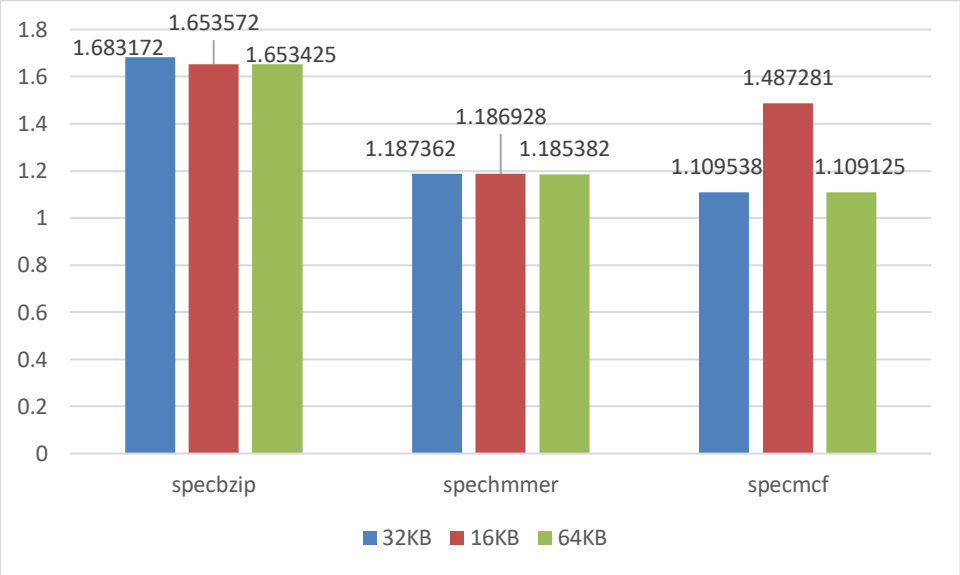
Benchmarks	CPI	ICACHE Miss Rate	ICACHE MISSES	ICACHE Accesses	DCACHE Miss Rate	DCACHE Misses	DCACHE Accesses
specbzip	1.683172	0.000074	757	10198809	0.01484	774129	52164486
spechmmer	1.187362	0.000205	3516	17142404	0.001645	72178	43865006
speclibm	3.493611	0.000099	589	5959253	0.060971	2975792	48806500
specmcf	1.109538	0.000037	998	27009248	0.002051	72151	35173755
specsjeng	10.27081	0.00002	629	31871080	0.121829	10523876	86382246

Όπως φαίνεται και παραπάνω η μεγαλύτερη καθυστέρηση προέκυψε από τα misses στη Dcache πράγμα που δείχνει και ο φυσικός αριθμός αλλά και το ποσοστό επί των dcache accesses. Έτσι πρώτη απόπειρα που έγινε ήταν η αλλαγή (αύξηση) του μεγέθους της εν λόγω cache καθώς μεγαλύτερη cache θα επέτρεπε την αποθήκευση περισσότερων στοιχείων στη γρηγορότερη μνήμη του συστήματος και άρα καλύτερη εκμετάλλευση του spatial-locality. Παρακάτω φαίνονται οι αντίστοιχες μετρήσεις:



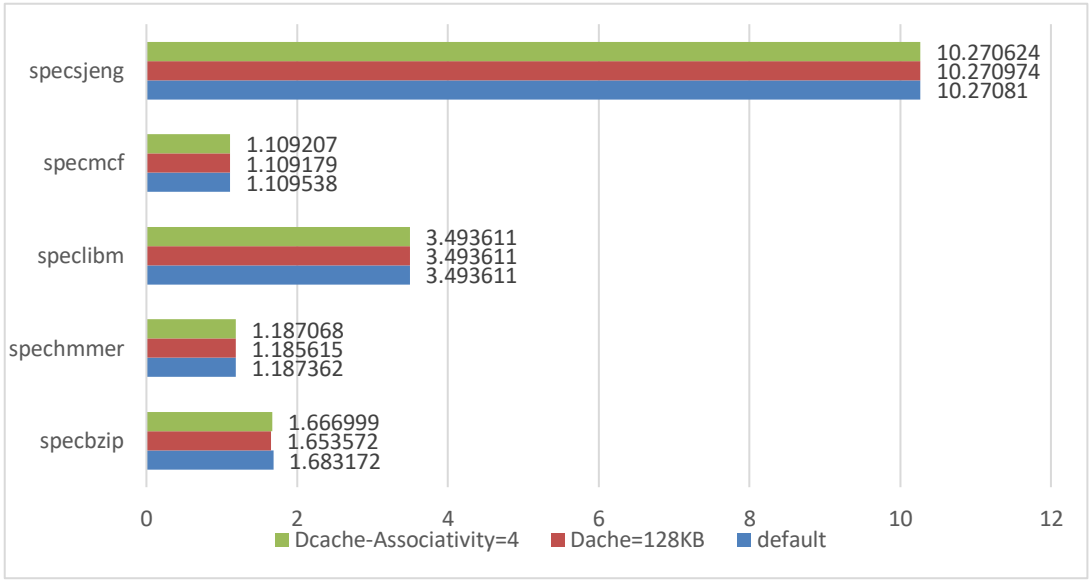
Όπως είναι εμφανές, οι διαφορές στο CPI για διπλάσιο μέγεθος dcache, δεν δικαιολογούν το διπλάσιο κόστος της μνήμης cache(θα επανέλθουμε στο ζήτημα αυτό με τη συνάρτηση κόστους παρακάτω). Να σημειωθεί ότι οι μεταβολές σε όλα τα παρακάτω και παραπάνω μεγέθη έγιναν σε δυνάμεις του 2 καθώς αυτό ήταν επιτρεπτό από το gem5 αλλά και λογικό σχεδιαστικά.

Με παρόμοιο σκεπτικό έγινε μεταβολή του μεγέθους της icache πρώτα με μειωτική τάση με απώτερο σκοπό την εξοικονόμηση υλικού χωρίς την απώλεια απόδοσης και στη συνέχεια αύξησης του μεγέθους ώστε να διαπιστωθεί πιθανό περιθώριο κέρδους σε απόδοση. Ακολουθούν οι αντίστοιχες μετρήσεις:



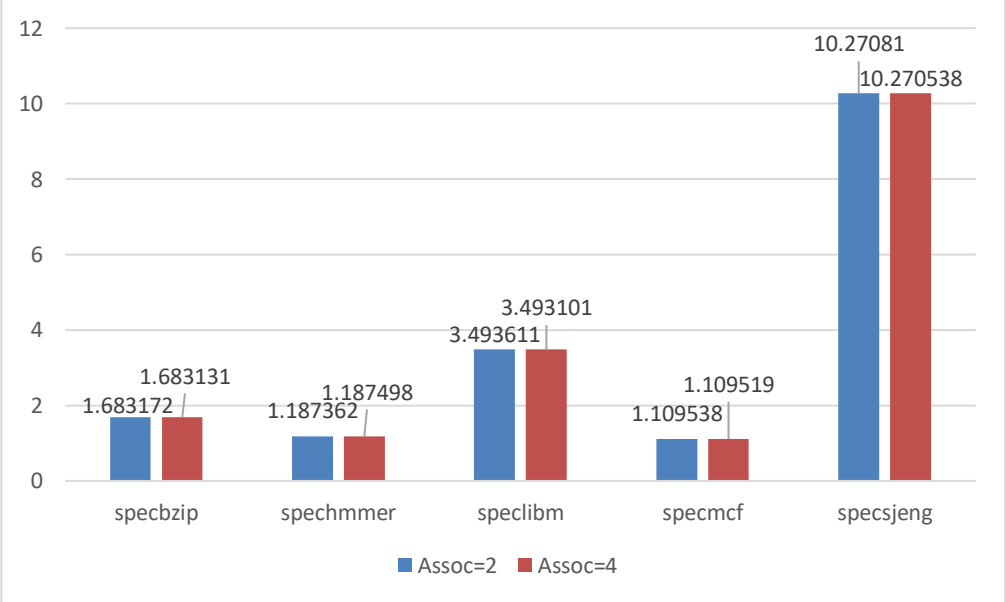
Φαίνεται από τα τρία παραπάνω benchmarks, όπως ήταν αναμενόμενο εξετάζοντας τα misses στην icache ότι μια αύξηση του μεγέθους αυτής δε προκαλεί ουσιαστική διαφορά του cpi ωστόσο επιβεβαιώνεται ότι μείωση του μεγέθους αυτής προκαλεί τέτοια “απώλεια” cpi που δεν είναι δυνατό να καλυφθεί ούτε από άλλες φθηνότερες τροποποιήσεις (π.χ. μεταβολή του cachline – μεταβολή του associativity της L1 icache). Να σημειωθεί ότι οι παραπάνω μετρήσεις γίνανε με σταθερό μέγεθος dcache(128KB) καθώς ενδιέφερε η σχετική επίδραση μόνο της icache.

Όπως αναφέρθηκε και παραπάνω, εξοικονόμηση μεγέθους πρώτα στη dcache και σε δεύτερη φάση στην icache επιχειρήθηκε μέσω αλλαγής του associativity. Η ιδέα πίσω από αυτή την αλλαγή προήλθε από προσπάθεια κάλυψης των συγκρούσεων θέσεων μνήμης κατά το mapping τους μέσα στη cache με χρήση της τυχαιότητας που προσφέρει το μεγαλύτερο associativity. Όπως θα αναλυθεί και παρακάτω(βλ. συνάρτηση κόστους) το κόστος της συγκεκριμένης επιλογής τόσο όσον αφορά τη λογική(πύλες) όσο και την επιφάνεια πυριτίου είναι πολύ μικρότερο από διπλασιασμό του μεγέθους της D/Icache. Παρακάτω παρουσιάζονται σε διαγράμματα οι παραπάνω ισχυρισμοί για associativity 2 και 4.
Το 8-way-associativity δε παρουσιάζεται καθώς τα αποτελέσματα ήταν ταυτόσημα με το 4-way.



Όπως φαίνεται και στο παραπάνω chart, καλύτερα αποτελέσματα από τη default περίπτωση επιτυγχάνονται έστω και ελάχιστη και με την αλλαγή του associativity, επιλογή πολύ πιο αποδοτική οικονομικά από διπλασιασμό του μεγέθους της cache, αν ειδικά λάβει κανείς υπόψιν του την αναλογία κόστους-απόδοσης(ανάλυση παρακάτω).

Ακολουθούν τα ίδια αποτελέσματα για την περίπτωση της icache με όμοια συμπεράσματα:



Σε αυτό το σημείο προκειμένου να μειώσουμε τον αριθμό των συνδυασμών που θα προσομοιώναμε στη συνέχεια και όπως αναφέρθηκε και παραπάνω, δημιουργήσαμε μια συνάρτηση κόστους ώστε να μπορούμε αν αποτιμήσουμε τη σκοπιμότητα της κάθε αλλαγής και άρα και της πορείας που ακολουθήθηκε. Η φιλοσοφία της συνάρτησης ήταν η εξής: Προκειμένου να επιτευχθεί καλύτερο resolution οι δυνατές μεταβολές σταθμίστηκαν με ένα συντελεστή που αντιστοιχεί στην αναλογική μεταβολή του καθαρού κόστους(χωρίς να λαμβάνεται υπόψιν η απόδοση) της επιλογής αυτής. Έτσι πρώτα ορίστηκαν τα καθαρά κόστη που αντιστοιχούν στη default συνάρτηση κόστους. Τα κόστη αυτά αναφέρονται σε μια απόλυτη μονάδα κόστους με εύρος 10 έως 70. Η επιλογή του εύρους έγινε με γνώμονα την

ανάγκη για αναπαράσταση του αντίκτυπου του κόστους κάθε επιλογής, δηλαδή αρκετά μεγάλη τιμή και ανάγκη για διαχωρισμό του κόστους και ορθή τήρησης των αναλογιών, δηλαδή αρκετά μεγάλο εύρος. Σε δεύτερη ανάλυση το βασικό κόστος της icache είναι μικρότερο από αυτό της dcache καθώς είναι μικρότερη σε μέγεθος(η σύγκριση των δύο τιμών δεν είναι απόλυτα αναλογική καθώς θεωρήθηκε ύπαρξη τρίτων παραμέτρων κατά τη κατασκευή του ολοκληρωμένου και όχι μόνη παράμετρος το απόλυτο μέγεθος της L1). Το κόστος της L2 είναι μικρότερο ανά μονάδα από το κόστος της L1 αλλά η L2 είναι πολύ μεγαλύτερη από την πρώτη οπότε το κόστος αντικατοπτρίζει και τις δύο παραμέτρους. Το associativity έχει πολύ μικρότερο κόστος από το μέγεθος των cache, ωστόσο μια μεγαλύτερου μεγέθους cache περιέχει περισσότερα blocks και άρα απαιτεί περισσότερες πύλες για την υλοποίηση της n-way associativity γεγονός που η συνάρτηση κόστους προσπαθεί να αποδώσει. Τέλος το cacheline αν και σαν λογική πυλών θεωρήθηκε ίδιας πολυπλοκότητας με το associativity(αναζήτηση λέξεων μέσα στο block), απαιτούνται περισσότερες συνδέσεις μεταξύ των cache για την αναγκαστική μεταφορά περισσότερων δεδομένων ανά request(μεγαλύτερο block) και άρα έχει ψηλότερο κόστος. Στον παρακάτω πίνακα φαίνονται τα απόλυτα κόστη για τις διαθέσιμες σχεδιαστικές επιλογές:

Part	**Cost index (10-70)**
L1 Instruction cache size	42
L1 Instruction Associativity	14
L1 Data cache size	70
L1 Data Associativity	20
L2 Cache Size	55
L2 Associativity	27
Cacheline	32

Έχοντας διαθέσιμη πλέον τη συνάρτηση κόστους εξήγαμε το γινόμενο TotalCostIndex*mean(CPI)και με βάση τους καλύτερους προχωρήσαμε στις επόμενες αλλαγές. Έτσι προέκυψαν οι εξής λόγοι:

Parts Changed	Mean CPI	Total Cost Index (TCI)	TCI*mean(CPI)
default	2.399642876	260	623.9071478
Dcache=128KB	2.390291802	330	788.7962947
Icache=64KB | Dcache=128KB	2.390049776	372	889.0985167
DCache Associativity = 4 | default	2.394743389	280	670.5281489
Cacheline = 128	2.0567645	292	600.575234

Όπως φαίνεται από τα παραπάνω οι δύο αποδοτικότεροι συνδυασμοί όσο αφορά το κόστος ανά μονάδα IPC είναι τα default sizes και το cacheline=128. Αυτό ήταν αναμενόμενο καθώς με μεγαλύτερο cacheline για ένα miss επιτρέπετε το fetching περισσότερων λέξεων σε κοντινές θέσεις μνήμης με την αναζητούμενη, από ότι για ένα μικρότερο block, κάνοντας καλύτερη εκμετάλλευση του locality ακόμα και από μεγαλύτερη cache. Λειτουργεί δηλαδή σαν ένα είδος pre-fetching. Να σημειωθεί ότι ως ανώτατο όριο cacheline θεωρήθηκε το 128 τόσο λόγο μείωσης πιθανών συνδιασμών για τα benchmarks αλλά και με το σκεπτικό της αδυναμίας μεταφοράς υπερβολικού αριθμού bit το ένα δίπλα στο άλλο(καθαρός περιορισμός χώρου όπως και σε μια motherboard).

Με αυτό το σκεπτικό στα υπόλοιπα benchmarks φαίνεται να είναι συμφερότερη η χρήση αυτού του μοντέλου ως βάση, ωστόσο και πάλι για τα benchmarks που ακολουθούν προσομοιώθηκαν παραπάνω μεταβολές ώστε να αποτυπωθεί ο εν λόγω ισχυρισμός. Να σημειωθεί ότι ο λόγος που επιλέχθηκε το TCI*mean(CPI) είναι ότι παράγει καλύτερα αποτελέσματα τόσο διαισθητικά όσο και μαθηματικά από τον λόγο TCI/mean(CPI). Ο τελευταίος για μικρότερο CPI δίνει λανθασμένα την εντύπωση αναλογικά χειρότερης υλοποίησης.

Σε επόμενη προσπάθεια βελτίωσης απόδοσης επιχειρήθηκε μεταβολή μεγέθους και associativity της L2 cache για λόγους ακριβώς ίδιους με αυτούς που οδήγησαν στα αντίστοιχα πειράματα για L1 cache.

Parts Changed	**Mean CPI**	**Total Cost Index (TCI)**	**TCI*CPI**
L2=4MB | CacheLine=128	2.0602879	347	714.9199033
L2=4MB	2.391073	315	753.1879922
L2=1MB | L2Asso.=16	2.4096572	260	626.5108778
L2=1MB | L2Asso. = 16 | CL = 128	2.0730157	292	597.32059

Όπως φαίνεται από όλα τα παραπάνω η πιο αποδοτική επιλογή που συνδυάζει χαμηλό κόστος και όσο το δυνατό χαμηλότερο CPI είναι CacheLine=128, L2=1MB, L2Asso.=16 με γινόμενο TCI*CPI = 597.32059.

Προφανώς θα μπορούσαν να συνδυαστούν περισσότεροι παράγοντες με σκοπό την επίτευξη χαμηλότερου CPI, όπως μεγαλύτερη cacheline, μεγαλύτερη L1/L2 cache, ταυτόχρονα. Ωστόσο όλες αυτές οι βελτιώσεις δεν επιφέρουν καλύτερη απόδοση αναλογική με το κόστος τους οπότε είναι ασύμφορες επιλογές για τη διάθεση τους στην αγορά.