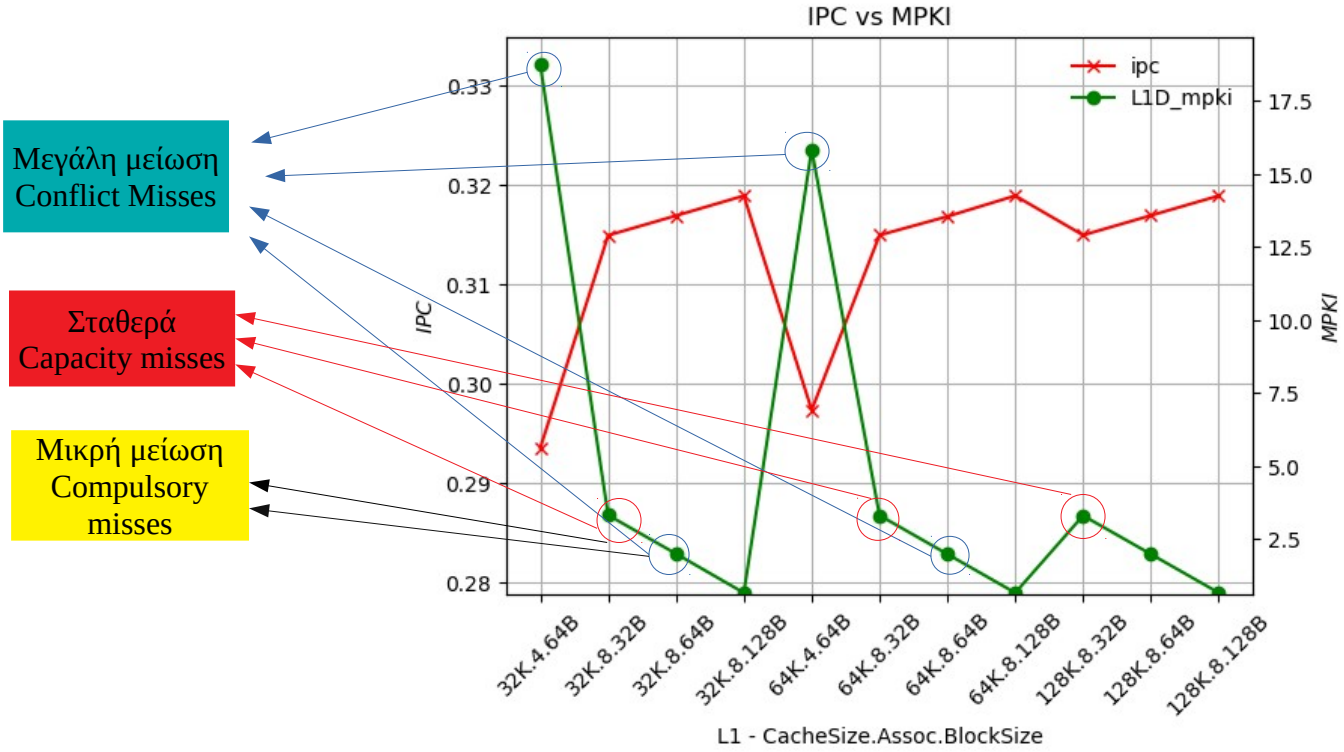
Προηγμένα Θέματα Αρχιτεκτονικής Υπολογιστών

1η Σειρά Ασκήσεων
Μάνος Αραπίδης
ΑΜ: 03116071

- L1 cache
  Σκοπός αυτού του ερωτήματος είναι να μελετήσουμε την επίδραση στην απόδοση ενός προγράμματος, μεταβάλλοντας τις βασικές παραμέτρους (όπως το cache size, associativity και block size) μιας L1 cache μνήμης και κρατώντας σταθερά τα χαρακτηριστικά της L2 cache και TLB. Ακολουθούν γραφικές παραστάσεις για τους διαφορετικούς συνδυασμούς cache size, associativity και block size που ελέγξαμε για κάθε benchmark , σε σχέση με το ipc και mkpi.

Blacksholes

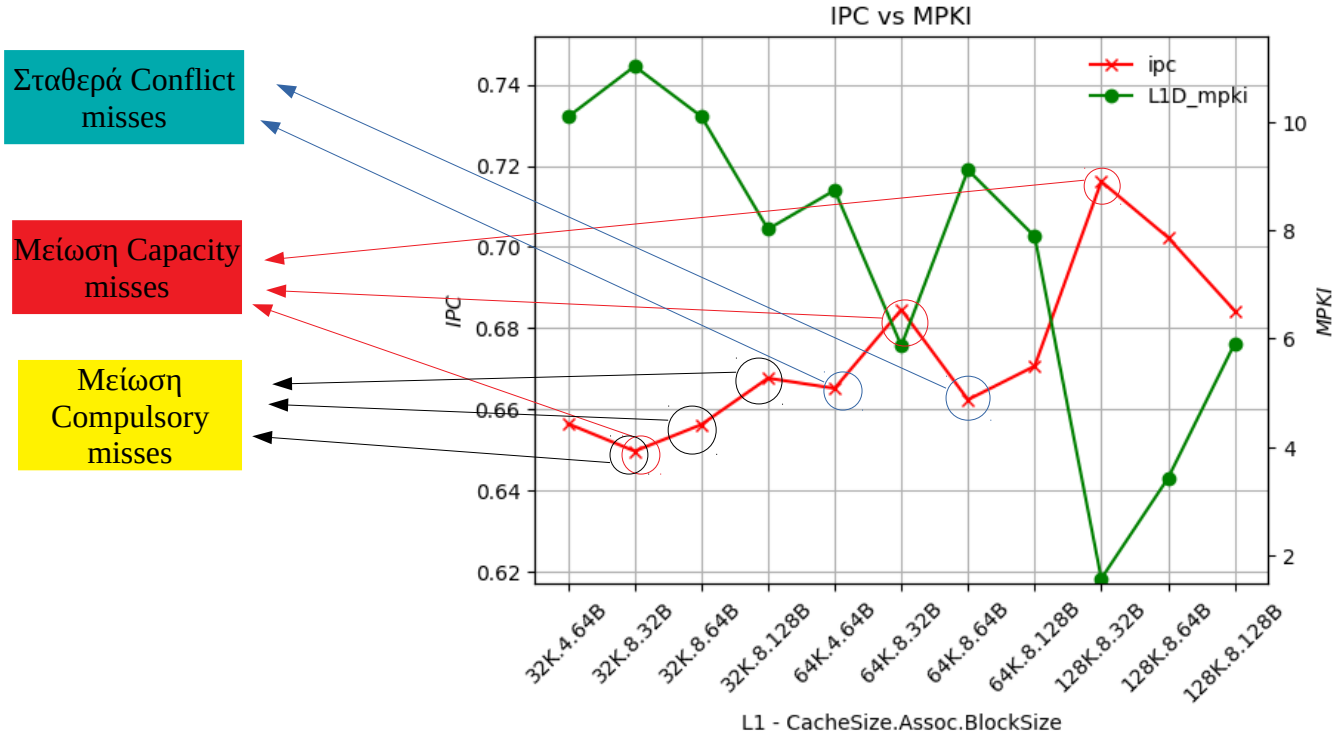


Από την γραφική παρατηρούμε ότι αύξηση του associativity από 4->8 προκαλεί σημαντική μειώσει του miss-rate, οπότε μπορούμε να συμπεράνουμε ότι η εφαρμογή έχει αρκετά conflict misses. Αντίθετα η αύξηση του cache size δε επιρεάζει καθόλου την απόδοση, άρα έχουμε ελάχιστα capacity misses. Τέλος, υπάρχει αύξηση του ipc (αντίστοιχα μείωση του mpki) με κάθε διπλασιασμό του block size, περιορίζοντας τα compulsory misses.

Οπότε μπορούμε να συμπεράνουμε ότι η εφαρμογή θα επωφελούταν πολύ από μια L1 cache με υψηλό associativity και δευτερευόντως από μεγάλο block-size, ενώ το cache-size την αφήνει ανεπηρέαστη.

Την βέλτιστη απόδοση την παρατηρούμε για τους συνδυασμούς 32K-8_128B, 64K-8_128B και 128K-8_128B. Καθώς αποδίδουν το ίδιο, θα επιλέξουμε την αρχιτεκτονική με το μικρότερο cache-size, λόγω του μικρότερου κόστους.

Bodytrack

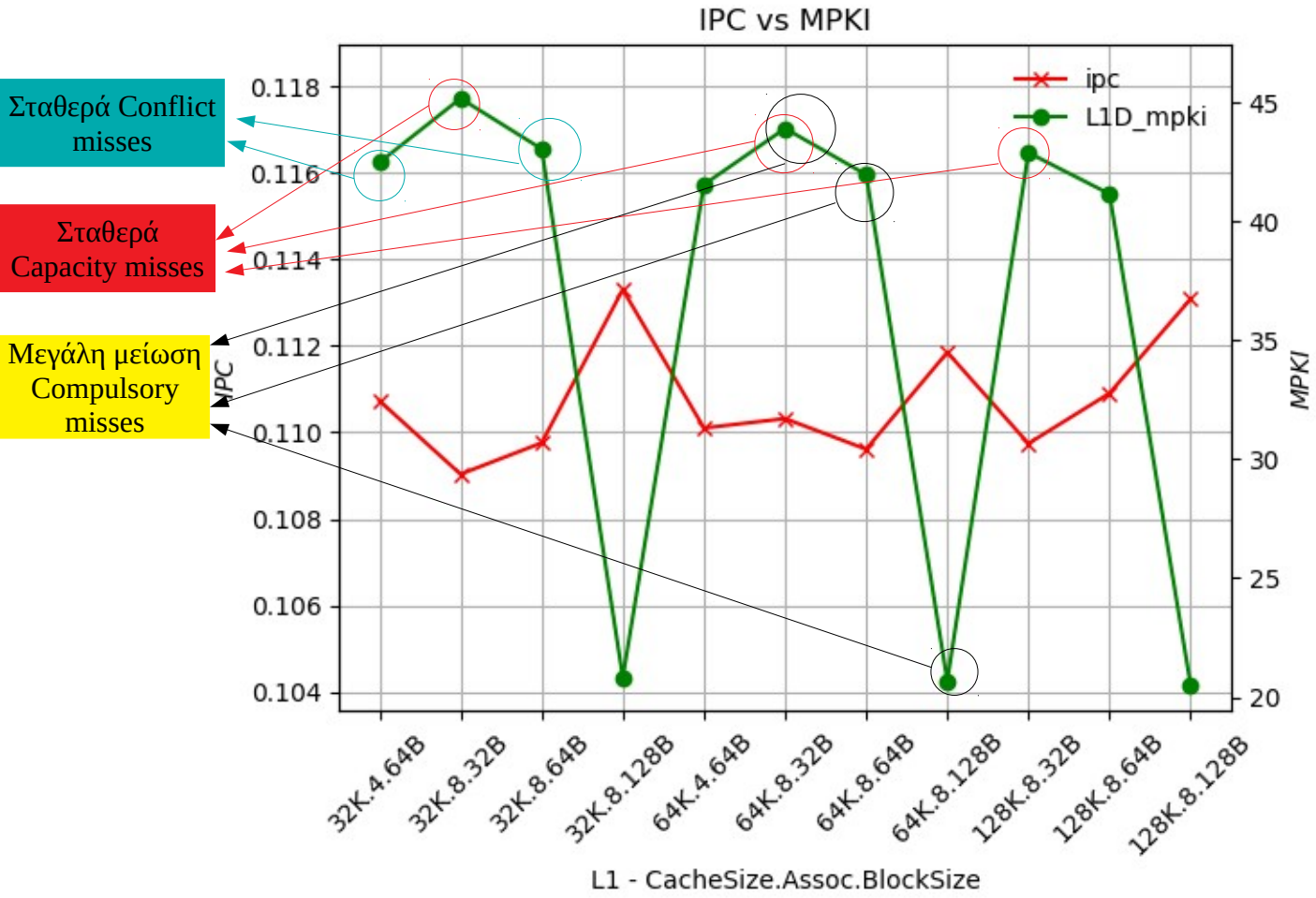


Από το διάγραμμα παρατηρούμε, ότι αυξάνοντας το cache-size και κρατώντας τα σταθερά τα άλλα στοιχεία το ipc αυξάνεται (αντίστοιχα το mpki μειώνεται) , οπότε συμπεραίνουμε ότι έχουμε πολλά capacity-misses. Αντίθετα, η αύξηση του assosiativity δεν επιρεάζει το ipc οπότε έχουμε ελάχιστα conflict misses. Ακόμα, η αύξηση του block-size, για μικρό cache-size(32K), καταφέρνει να μειώσει το misss-rate , μειώνοντας τα compulsory-misses. Ωστόσο, για μεγαλύτερο cache-size δε έχουμε την ίδια συμπεριφορά και σε ορισμένους συνδυασμούς (128K.8.32B-128K.8.64B-128k.8.128B) έχουμε υψηλότερο msis-rate , καθώς αν και μειώνονται τα compulsory-misses αυξάνονται τα conflict-misses.

Την μέγιστη απόδοση την έχουμε για τους συνδυασμούς : 128K.8.32B και 64K.8.32B.

Οπότε η εφαρμογή θα επωφελούνταν κυρίως από μεγάλο cache-size και δευτερευόντως από μικρό block-size, ενώ το assosiativity δεν επηρεάζει ιδιαίτερα. Για καλύτερη απόδοση θα επιλέγαμε την υλοποίηση με 128K.8.32B, ενώ για χαμηλότερο κόστος την 64K.8.32B .

Canneal

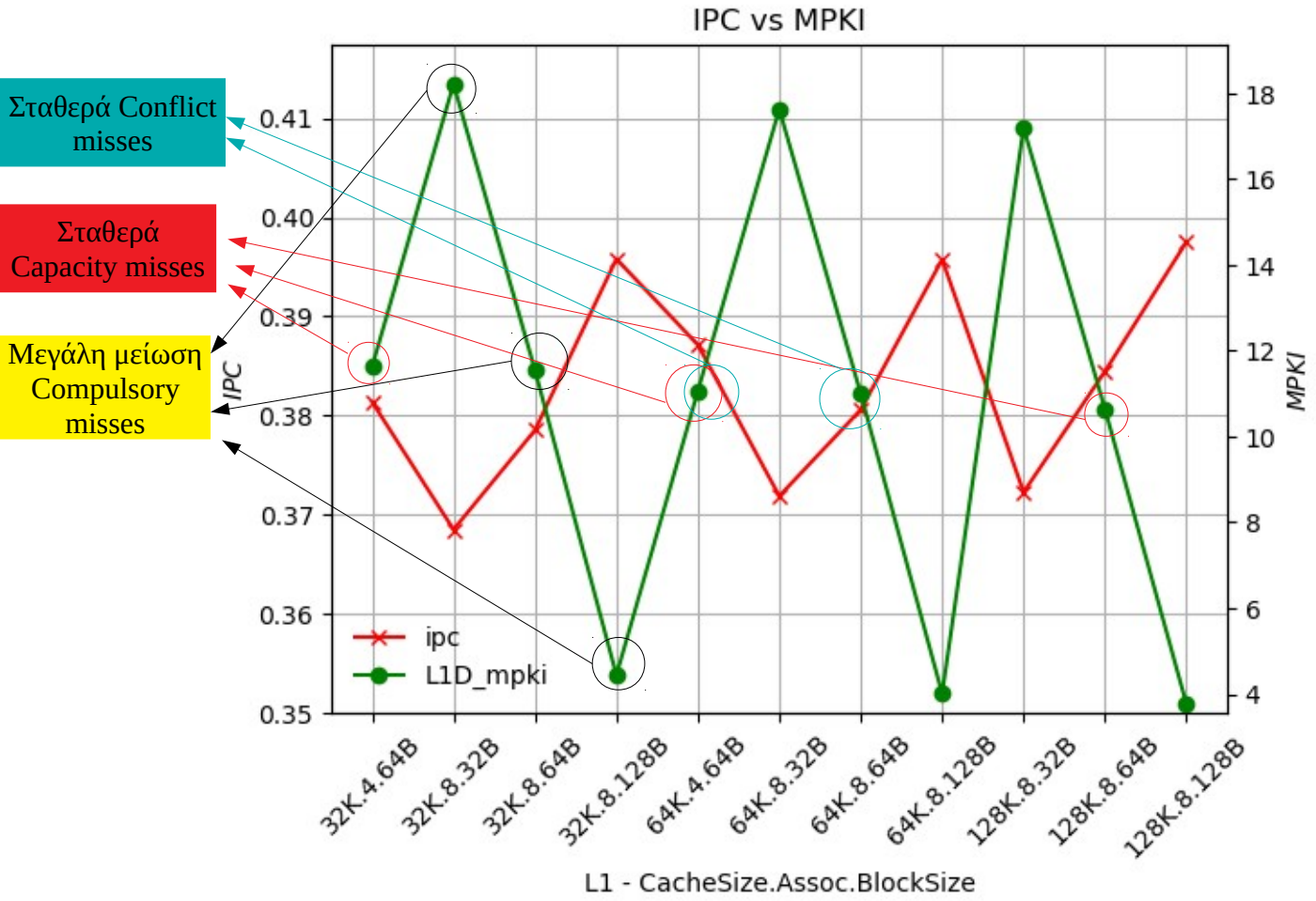


Παρατηρώντας το διάγραμμα βλέπουμε ότι η αύξηση του block-size προκαλεί μεγάλη πτώση του miss-rate, οπότε η εφαρμογή έχει αρκετά compulsory-misses. Αντίθετα, το cache-size και το assosciativity φαίνεται να μην επηρεάζουν την απόδοση, άρα το πλήθος των conlfict και capacity misses φαίνεται να παραμένει σταθερό.

Η απόδοση παρά τις μεταβολές παραμένει σχεδόν, καθώς έχουμε διαφορποιήση στα τελευταία δεκαδικά.

Την μέγιστη απόδοση ipc την πετυχαίνουμε για τους συνδυασμούς: 32K.8.128B και 128K.8.128B. Θα επιλέγαμε την υλοποίηση με το μικρότερο cache-size λόγω του μικρότερου κόστους.

Facesim

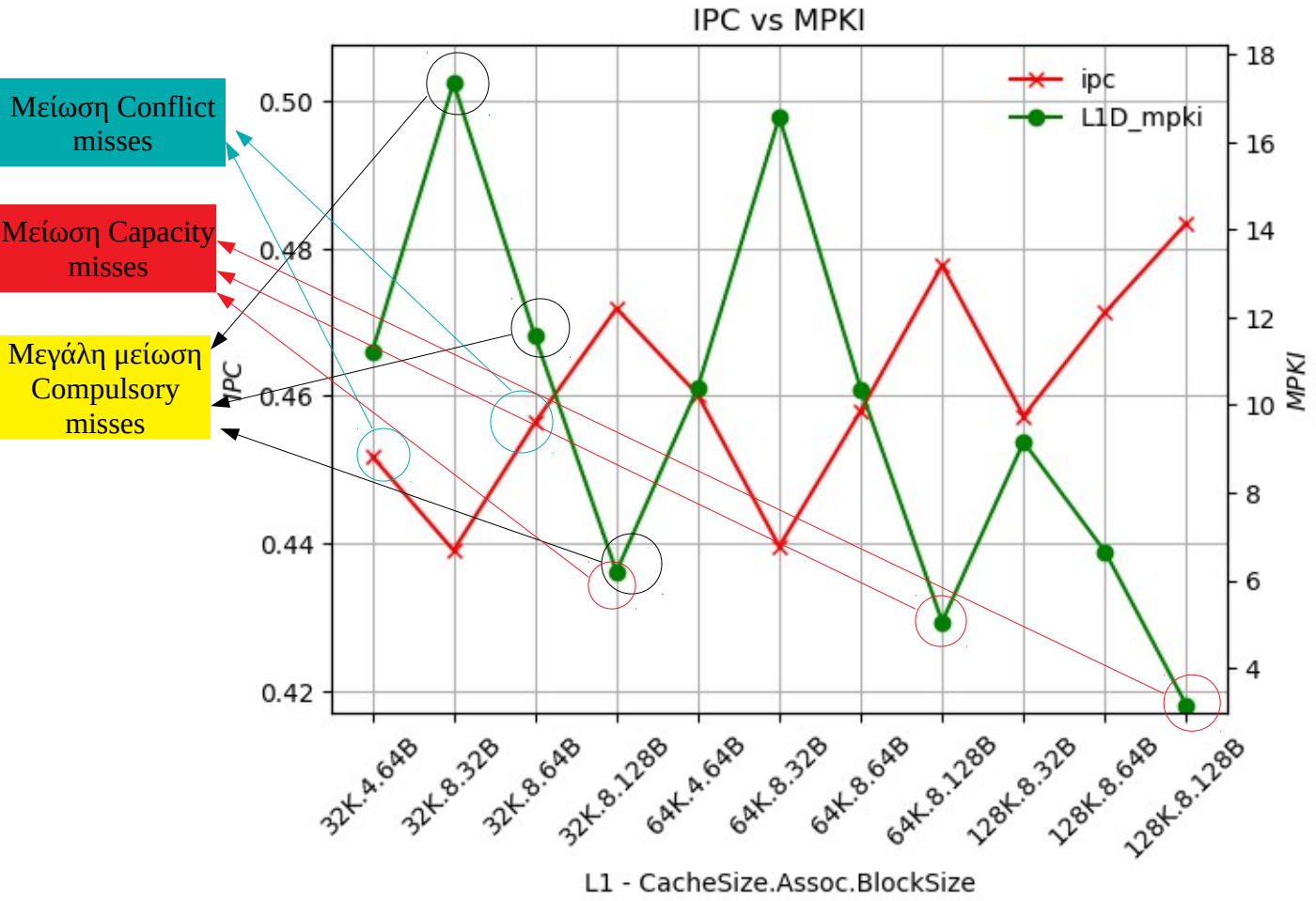


Ομοίως παρομοίως με την προηγούμενη εφαρμογή, η αύξηση του block-size μειώνει σημαντικά το miss-rate, εξαλείφοντας αρκετά compulsory-misses. Ακόμα το cache-size και το associativity φαίνεται να επηρεάζει ελάχιστα τα ipc και mpki. Η αύξηση του cache-size(32K->64K) αυξάνει ελάχιστα το ipc ενώ επιπλέον μέγεθος(64K->128K) δε φαίνεται να φέρει αλλαγή. Αντίθετα με την αύξηση του associativity (64K.4->64.8) έχουμε ελαφρύ μείωση του ipc.

Η εφαρμογή θα επωφελούνταν κατά κύριο λόγο από μεγάλο block-size και σε δεύτερο βαθμό από μεσαίο cache-size και associativity. Ωστόσο, η αύξηση που παρατηρούμε είναι ελάχιστη .

Την βέλτιστη απόδοση την έχουμε για τους συνδιασμούς: 32K.8.128B, 64K.8.128B και 128K.8.128B. Θα προτιμήσουμε την υλοποίηση με το μικρότερο cache-size λόγω του χαμηλότερου κόστους.

Ferret

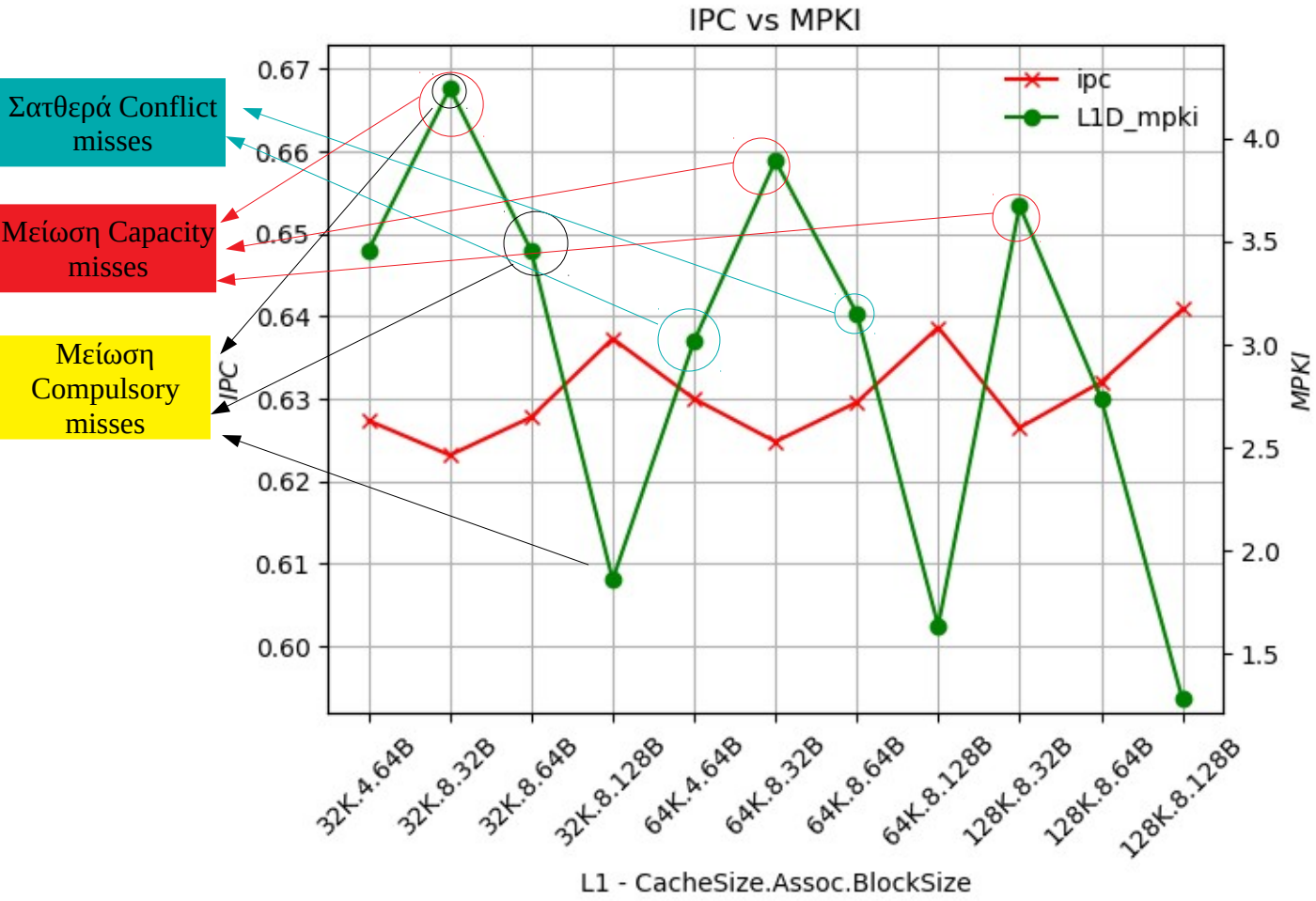


Από το διάγραμμα παρατηρούμε ότι η αύξηση του block-size οδηγεί σε μεγάλη μείωση του miss-rate και αντίστοιχα την αύξηση του ipc. Η αύξηση του associativity(32K.4.64->32K.8.64) καταφέρνει να βελτιώσει ελάχιστα την απόδοση ενώ το cache-size (32K.8.128B-> 64K.8.128B-> 128K.8.128B) φαίνεται να την επηρεάζει περισσότερο.

Η απόδοση της εφαρμογής εξαρτάται κατά κύριο λόγο από μεγάλο block-size, σε μικρότερο βαθμό από το cache-size και ελάχιστα από το associativity. Ωστόσο, η αύξηση που παρατηρούμε είναι μικρή .

Την μέγιστη απόδοση την έχουμε για τους εξής συνδυασμούς: 32K.8.128B, 64K.8.128B και 128K.8.128B. Οι επιδόσεις τους είναι αρκετά κοντά οπότε μπορούμε να επιλέξουμε την μέση περίπτωση με cache-size 64K .

Fluidanimate

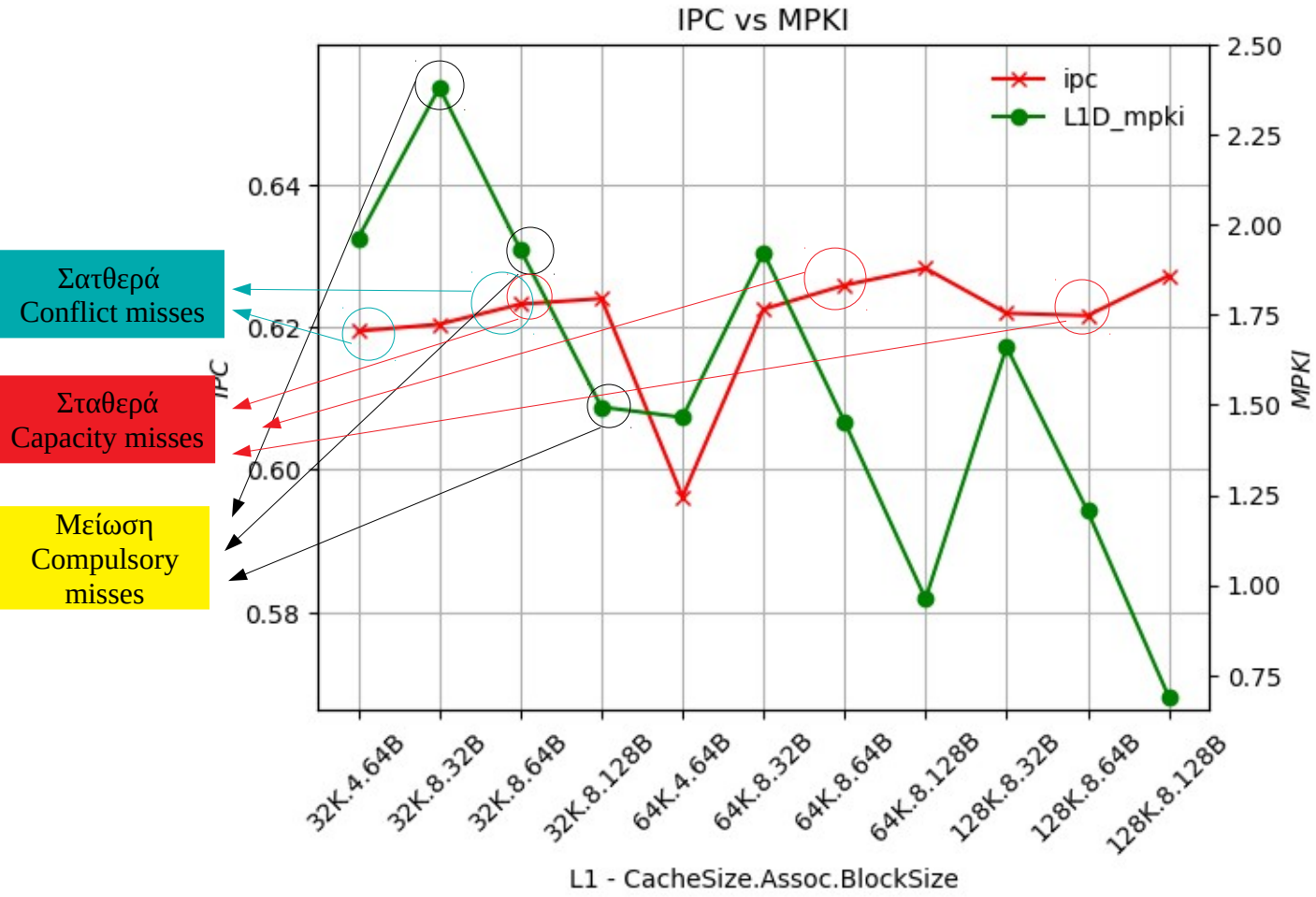


Από το διάγραμμα διαπιστώνουμε ότι η αύξηση του block-size οδηγεί στην μείωση του miss-rate και την αύξηση του ipc. Η αύξηση του cache-size οδηγεί σε μικρή βελτίωση της απόδοσης, αντίθετα οι αλλαγές στο associativity δεν επηρεάζουν το σύστημα.

Η εφαρμογή λειτουργεί αποδοτικότερα με μεγαλό block-size ,το cache-size παίζει ελάχιστο ρόλο ενώ το associtivity καθόλου. Πάλι, η αύξηση της απόδοσης είναι ελάχσιτη κρατώντας την σχεδόν σταθερή .

Την βέλτιστη απόδοση την πετυχαίνουμε με τους συνδυασμούς: 32K.8.128B, 64K.8.128B και 128K.8.128B. Οι διαφορές του είναι ελάχιστες οπότε μπορούμε να διαλέξουμε την εκδοχή με το μικρότερο κόστος, 32K.8.128B .

Freqmine



Σατθερά
Conflict misses

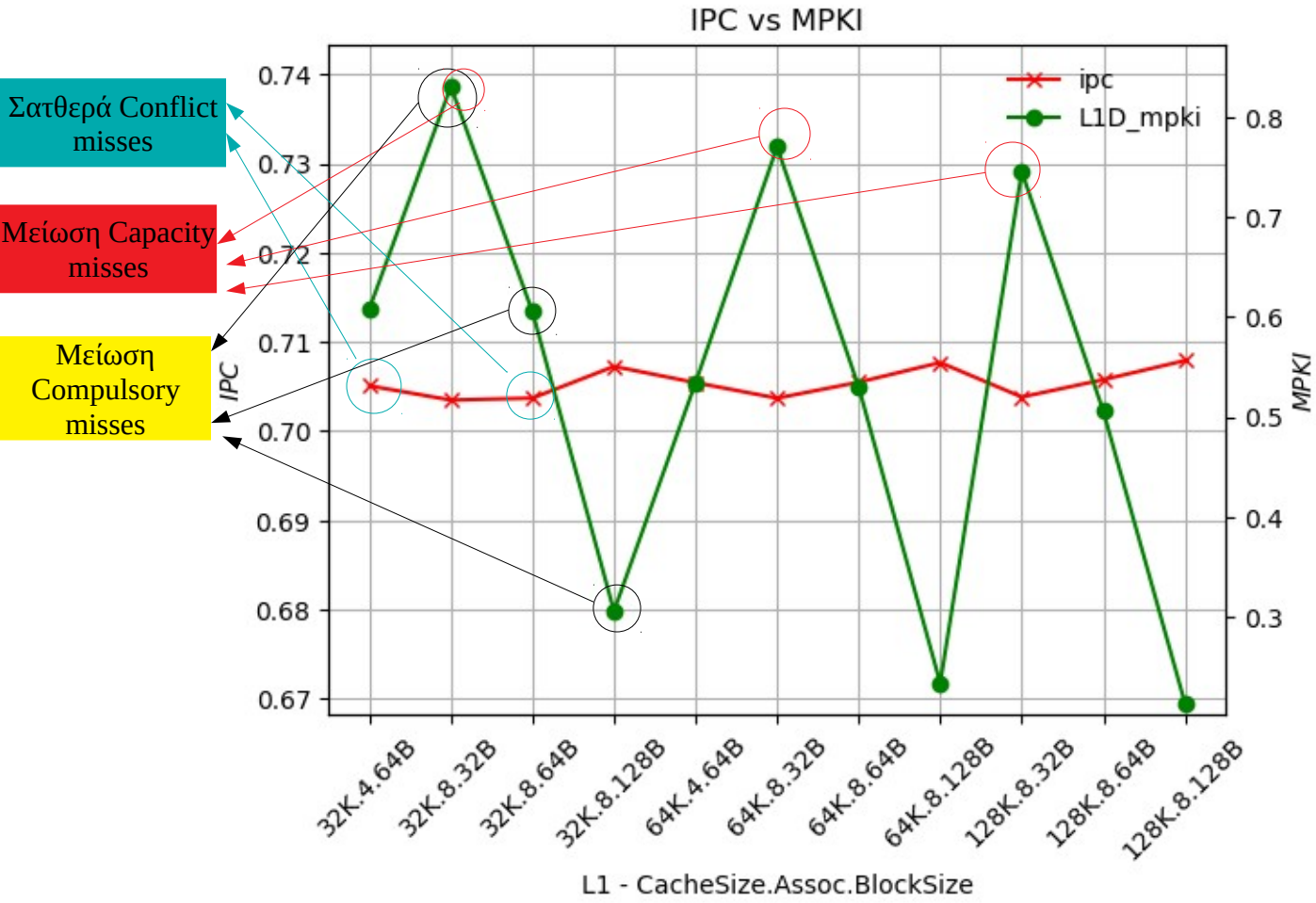
Σταθερά
Capacity misses

Μείωση
Compulsory
misses

Από το διάγραμμα φαίνεται ότι η αύξηση του block-size οδηγεί σε μείωση του miss-rate και αντίστοιχα μικρή αύξηση του ipc. Ακόμα η απόδοση παραμένει σταθερή στις αυξήσεις του cache-size και του βαθμού associativity . Ωστόσο, παρατηρούμε ότι μια αντιφατική συμπεριφορά καθώς έχουμε ταυτόχρονη αύξηση και μείωση των δύο μετρικών ipc και mpki στις μεταβάσεις 32K.8.64B-> 64K.4.64B και 64K.4.64B-> 64K.8.32B . Η συμπεριφορά αυτή δε συμφωνεί με την γενικότερη εικόνα της προσομοίωσης και μας δυσκολεύει να έχουμε σίγουρα συμπεράσματα .

Παρατηρούμε ότι αν εξαιρέσουμε την παραπάνω περίπτωση η απόδοση παραμένει σταθερή, οπότε μια καλή επιλογή θα ήταν 32K.4.64B .

Raytraceview

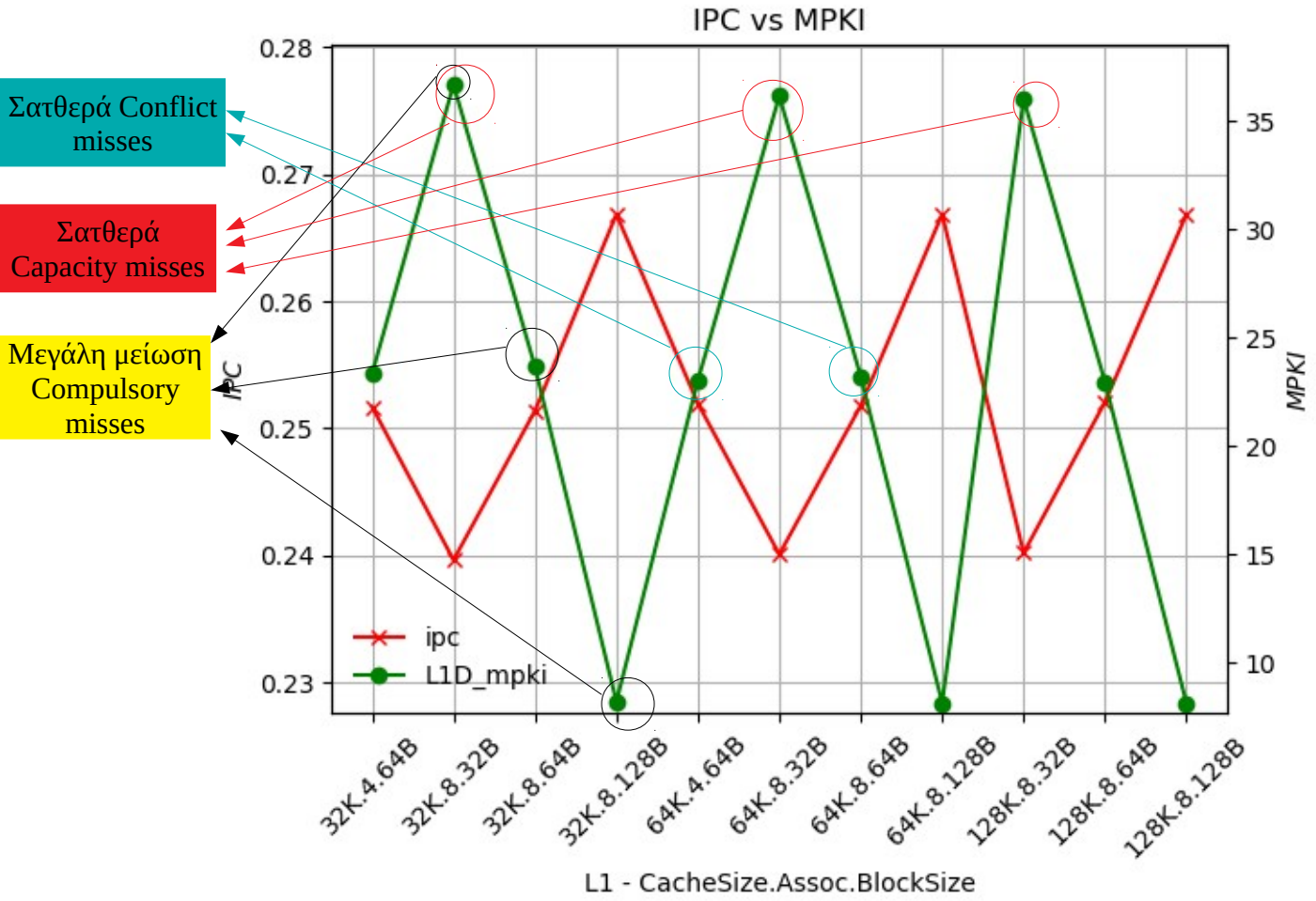


Η αύξηση του block-size έχει την ως αποτέλεσμα την μικρή αύξηση της απόδοσης (ipc) και την μείωση του mss-rate. Παρατηρούμε ότι αν και η αύξηση του cache-size προκαλεί μικρή μείωση στο miss-rate, δε είναι αρκετό για την βελτίωση του ipc. Οι αλλαγές στο associativity δεν επηρεάζουν καθόλου.

Η απόδοση παραμένει πρακτικά σταθερή .

Την βέλτιστη απόδοση την έχουμε για : 32K.8.128B, 64K.8.128B και 128K.8.128B. Ωστόσο δε παρατηρούμε μεγάλες αλλαγές στο ipc για μικρότερο block-size(πχ 32K.8.64B ->32K.8.128B) ή associativity (πχ 32K.4.64B->32K.8.64B), οπότε για πιθανώς μεγαλύτερη ταχύτητα(λόγω μικρότερου associativity) και χαμηλότερο κόστος, μπορούμε να επιλέξουμε 32K.4.64B .

Streamcluster

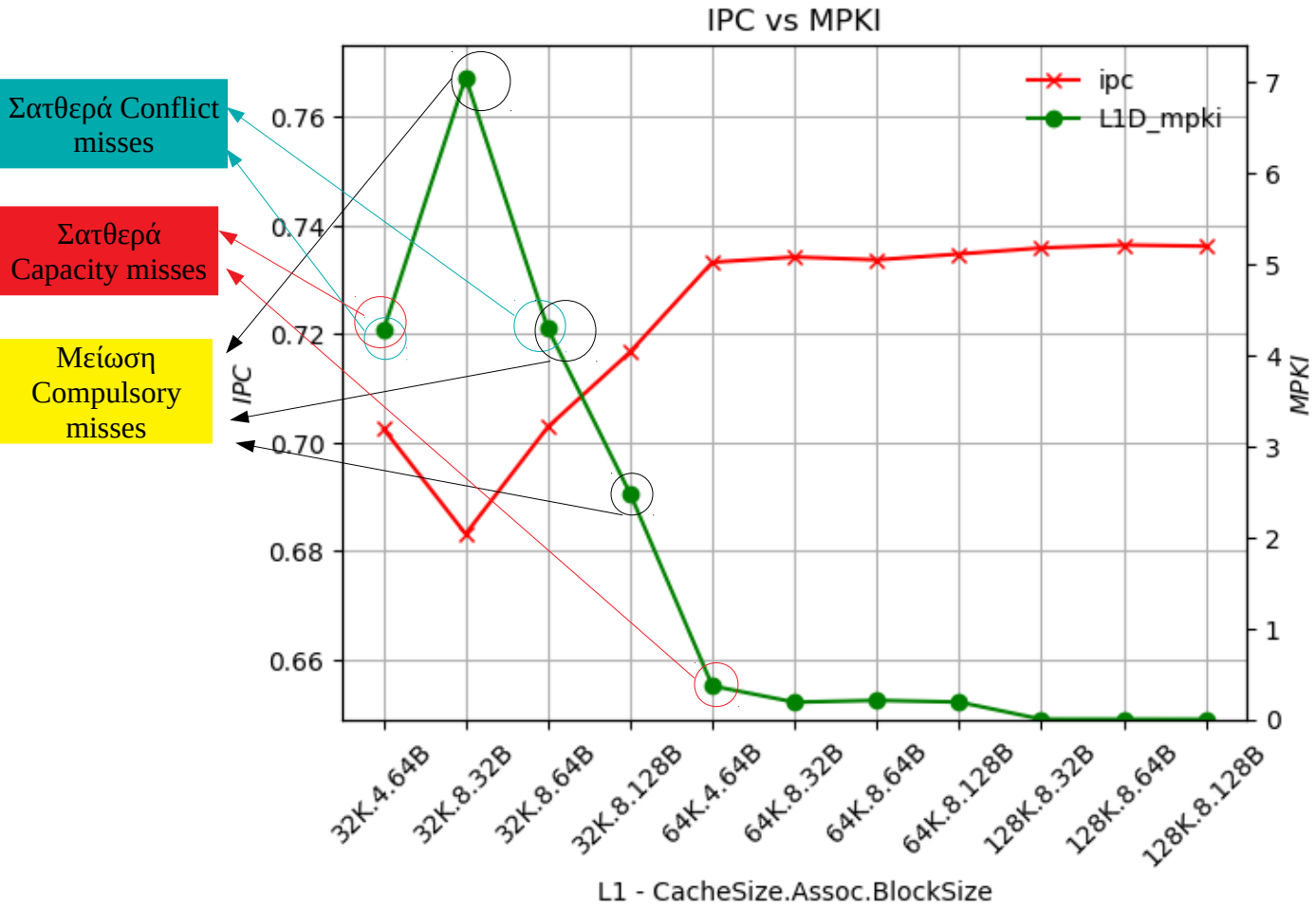


Η αύξηση του block-size προκαλεί αύξηση του ipc αντίθετα οι αλλαγές στο cache-size και associativity δεν επηρεάζουν το σύστημα

Η απόδοση της εφαρμογής βελτιώνεται με μεγάλο block-size ενώ τα cache-size και associativity δεν παίζουν σημαντικό ρόλο. Ωστόσο, η αύξηση που παρατηρούμε είναι μικρή .

Την βέλτιστη απόδοση την πετυχαίνουμε για τους συνδυασμούς : 32K.8.128B, 64K.8.128B και 128K.8.128B. Ωστόσο, αποδίδουν το ίδιο οπότε μπορούμε να διαλέξουμε αυτή με μικρότερο κόστος, 32K.8.128B .

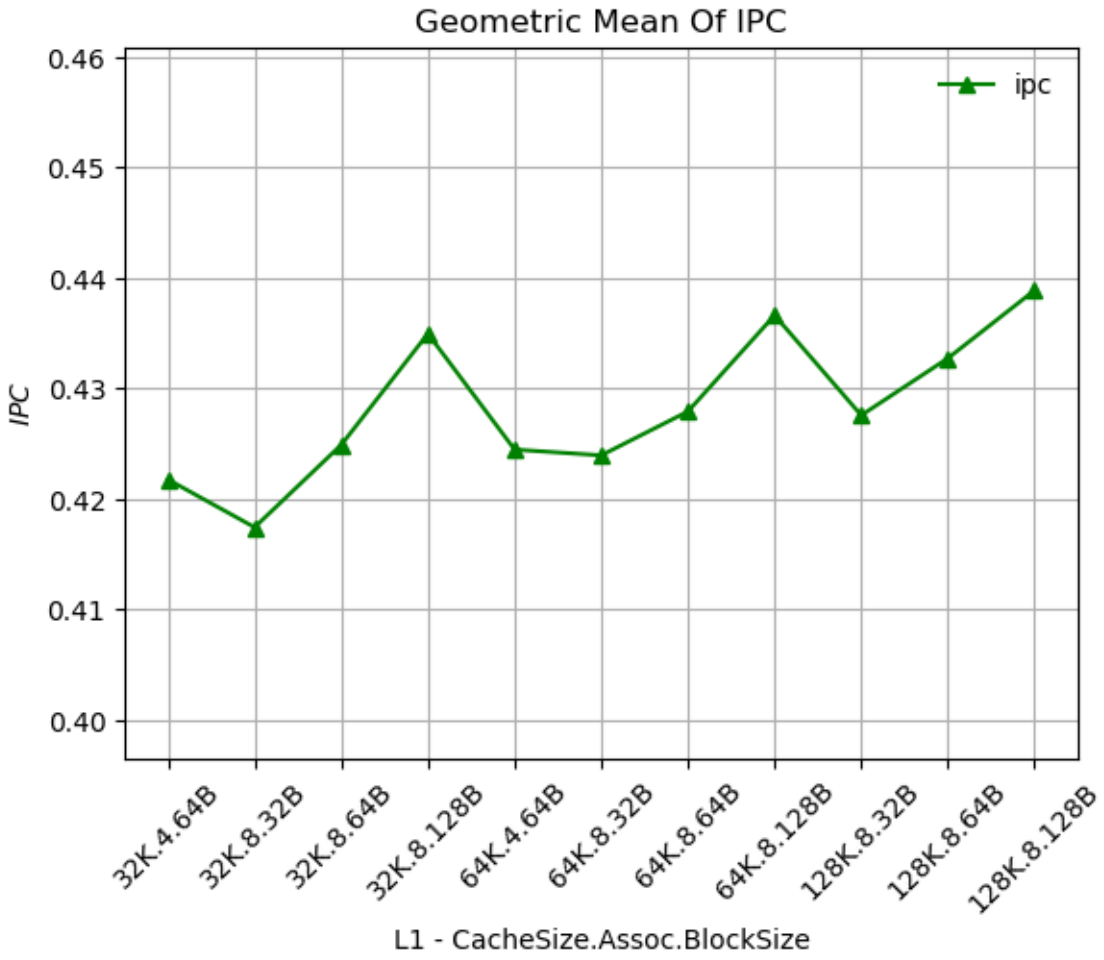
Swaptions



Από το διάγραμμα βλέπουμε ότι έχουμε μικρή βελτίωση του ipc για την αρχική αύξηση του cache-size (32K.4.64B->64K.4.64B). Όμως, περαιτέρω αύξηση στο μέγεθος δεν φέρνει το ίδιο αποτέλεσμα. Η αύξηση του block-size επωφελεί την απόδοση αλλά μόνο στην περίπτωση μικρής cache, μετά δε την επηρεάζει. Ο βαθμός associativity δεν φέρνει κάποια αλλαγή.

Το κυρίαρχο ρόλο στην απόδοση τον παίζει το cache-size, καθώς η αρχική του αύξηση επισκιάζει οποιαδήποτε άλλη αλλαγή στο σύστημα. Οι τιμές για τις 64K και 128K size είναι πολύ κοντά, οπότε θα επιλέξουμε την υλοποίηση με το μικρότερο μέγεθος λόγω του κόστους. Ακόμα, επειδή το associativity δεν επηρεάζει την απόδοση, θα επιλέξουμε βαθμό 4 για να έχουμε μεγαλύτερη ταχύτητα και να κάνουμε την μνήμη πιο απλή. Οπότε θα επιλέξουμε την 64K.4.64B .

Συμπεράσματα



Το παραπάνω διάγραμμα αποτελεί τον γεωμετρικό μέσο του ipc που προέκυψε από τα benchmarks. Η συμπεριφορά της γραφικής συμβαδίζει με την ανάλυση που είχαμε κάνει στο μεγαλύτερο κομμάτι των εφαρμογών. Παρατηρούμε ότι την καλύτερη απόδοση με την αύξηση του block-size (πχ 32K.8.32B-> 32K.8.64B->32K.8.128B). Η αύξηση στο associtivity και στο cache-size επηρεάζουν ελάχιστα, πχ 32K.4.64B-> 32K.8.64B και 32K.8.128B-> 64K.8.128B->128K.8.128B.

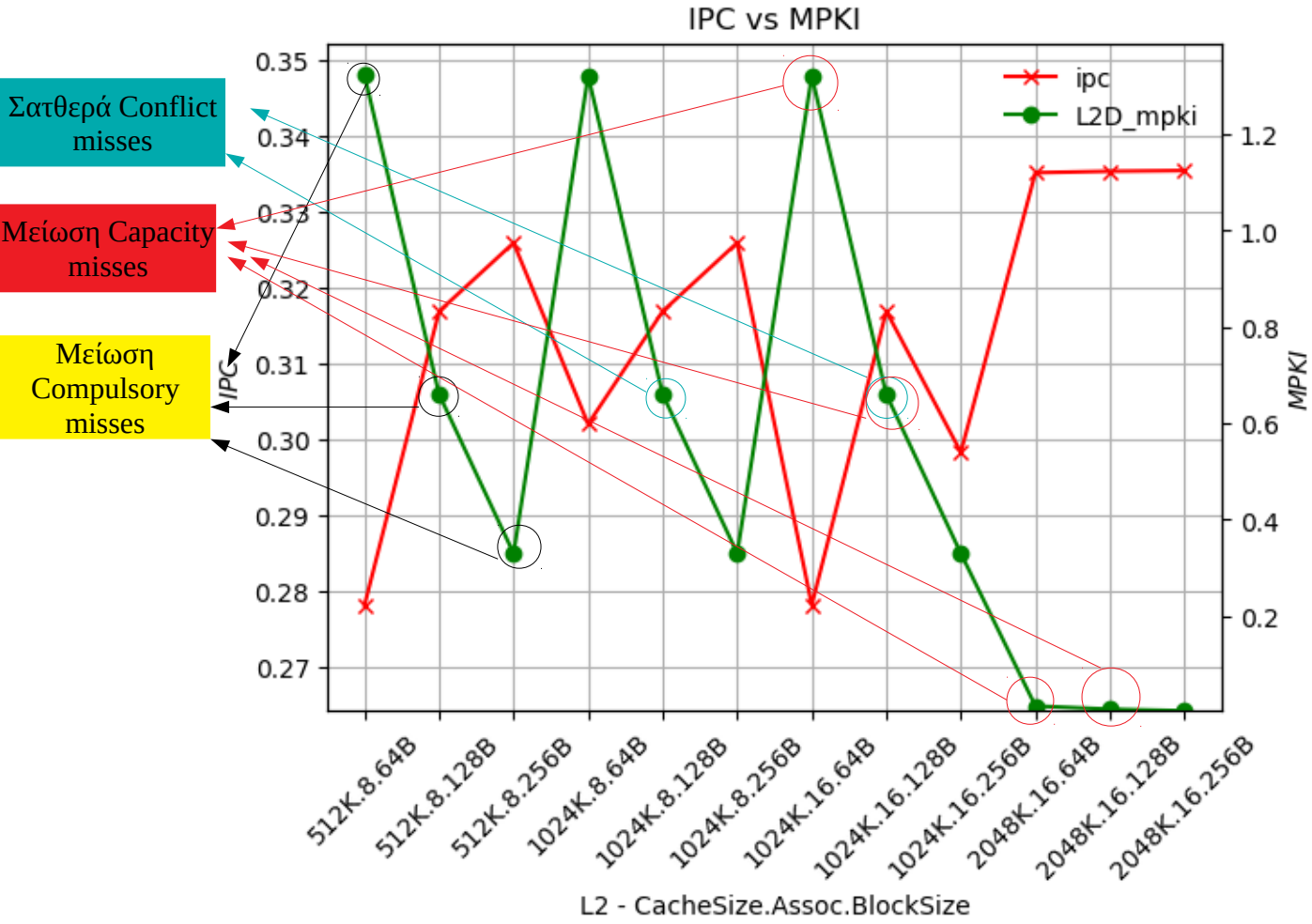
Οπότε, συμπεραίνουμε ότι τον μεγαλύτερο ρόλο για την απόδοση τον έχει το block-size., ενώ το associtivity και το cache-size έχουν ελάχιστο εώς καθόλου.

Σύμφωνα λοιπόν με την παραπάνω η καλύτερη επιλογή θα ήταν 32K.8.128B .

- L2 cache

Σκοπός αυτού του ερωτήματος είναι να μελετήσουμε την επίδραση στην απόδοση ενός προγράμματος, μεταβάλλοντας τις βασικές παραμέτρους (όπως το cache size, associativity και block size) μιας L2 cache μνήμης και κρατώντας σταθερά τα χαρακτηριστικά της L1 cache και TLB. Ακολουθούν γραφικές παραστάσεις για τους διαφορετικούς συνδυασμούς cache size, associativity και block size που ελέγξαμε για κάθε benchmark , σε σχέση με το ipc και mkpi.

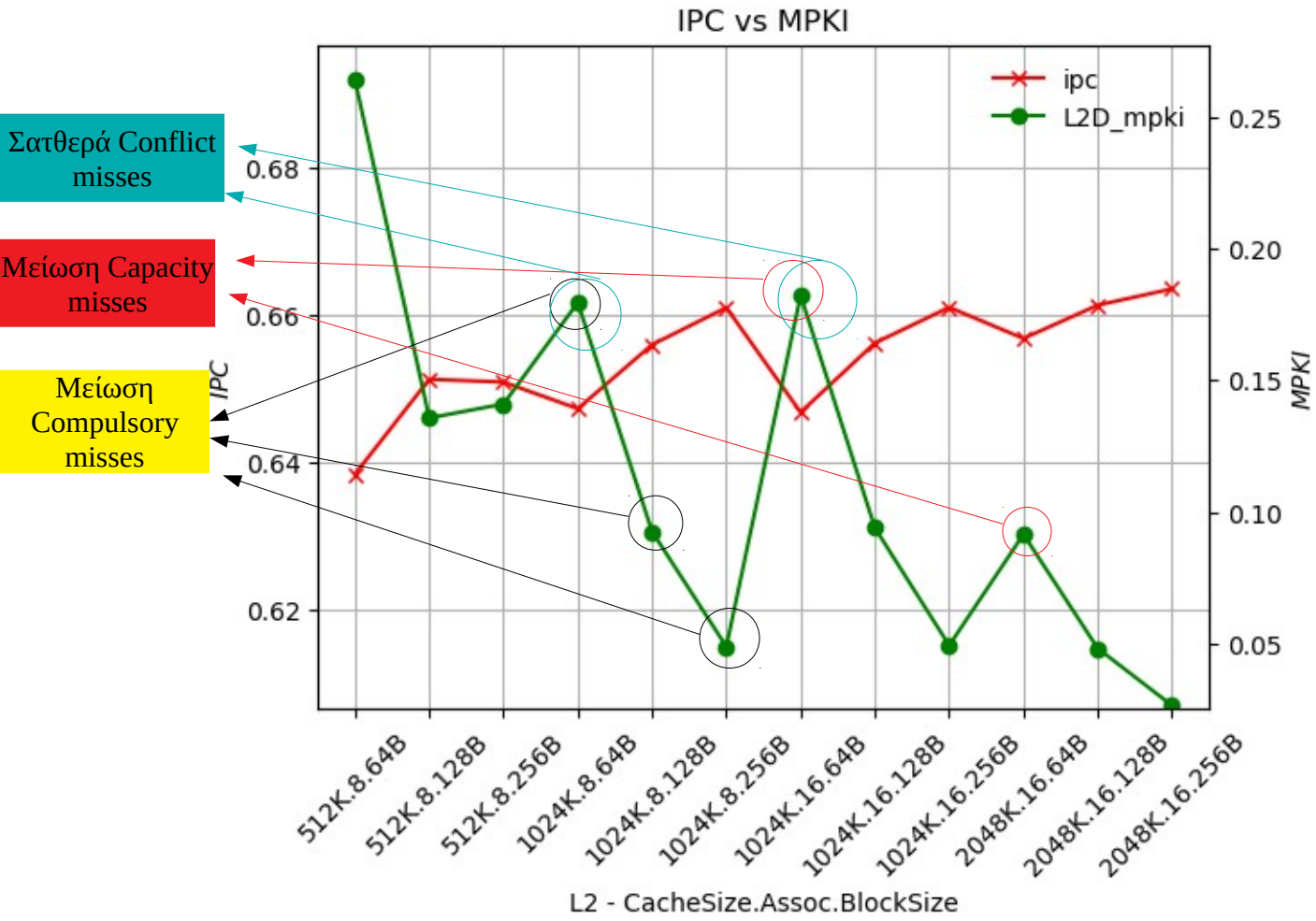
Blacksholes



Από το διάγραμμα διαπιστώνουμε ότι η αύξηση του block-size καταφέρνει να βελτιώσει την απόδοση της μνήμης, μέχρι το μέγεθος της cache να μεγαλώσει αρκετά όπου και τότε δε φέρνει κάποια αλλαγή. Ο βαθμός του associativity δεν φαίνεται να αλλάζει την συμπεριφορά της μνήμης. Η αρχική αύξηση στο cache-size (512K->1024K) δε επηρεάζει καθόλου την απόδοση, ωστόσο μια επιπλέον αύξηση στα 2048K φέρνει τα καλύτερα αποτελέσματα από όλες τις άλλες αλλαγές.

Για την συγκεκριμένη εφαρμογή έχουμε δύο επιλογές ως προς την υλοποίηση της L2, οι οποίες φέρνουν παρόμοια αποτελέσματα. Οι επιλογές μας είναι μικρό cache-size με μεγαλό block-size (512K.8.256B) ή μεγάλο cache-size με μικρό block-size(2048K.16.64B) .

Bodytrack



Σατθερά Conflict misses

Μείωση Capacity misses

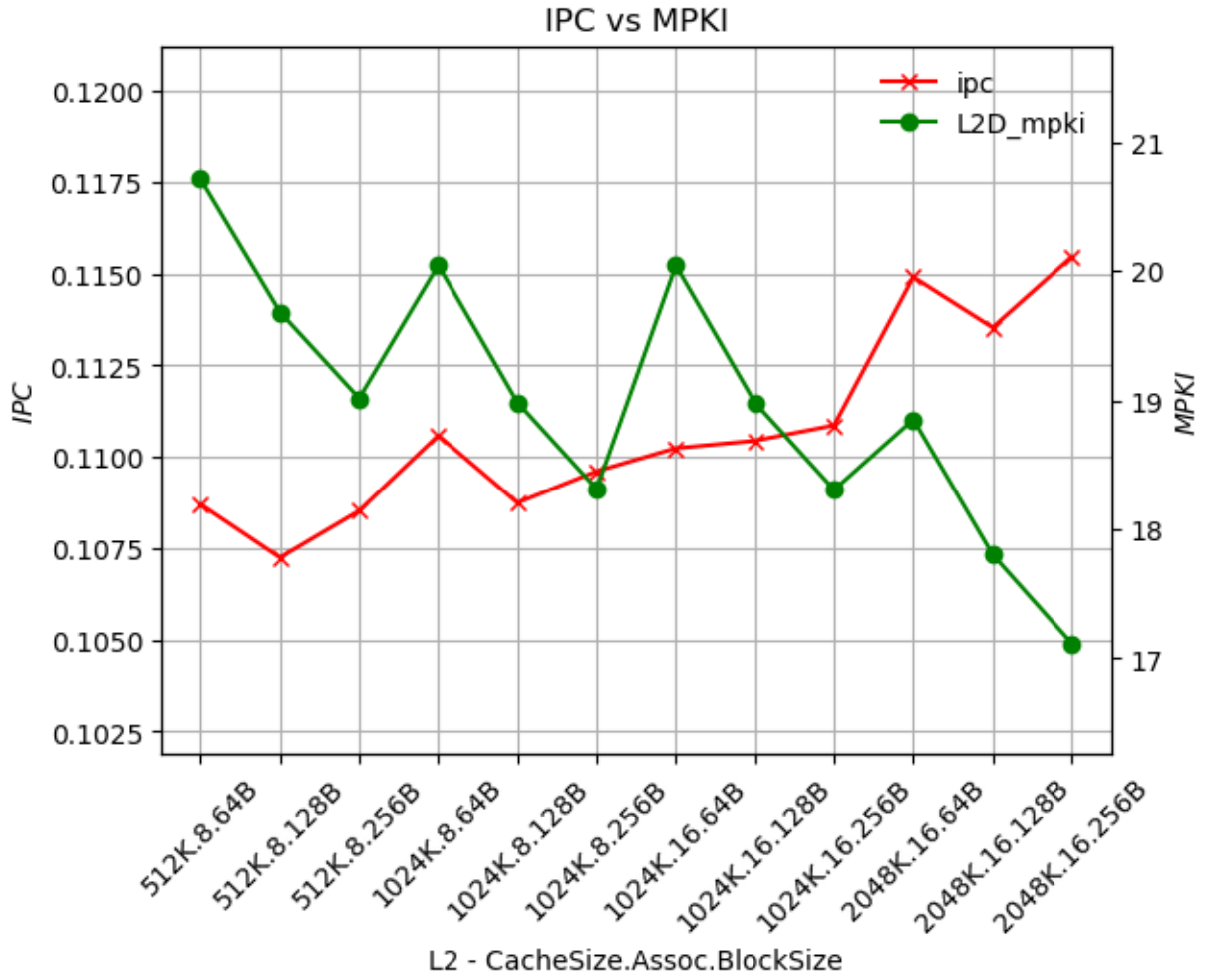
Μείωση Compulsory misses

Από το διάγραμμα διαπιστώνουμε ότι η αύξηση cache-size βελτιώνει την απόδοση της εφαρμογής. Το ίδιο αποτέλεσμα έχουμε με την αύξηση του block-size, αντίθετα ο βαθμός του associativity δε επηρεάζει την απόδοση.

Η εφαρμογή φαίνεται να επωφελείται φαίνεται να επωφελείται στον ίδιο βαθμό από την αύξηση του block-size και cache-size, αντίστοιχα. Ωστόσο, η αύξηση που παρατηρούμε είναι μικρή .

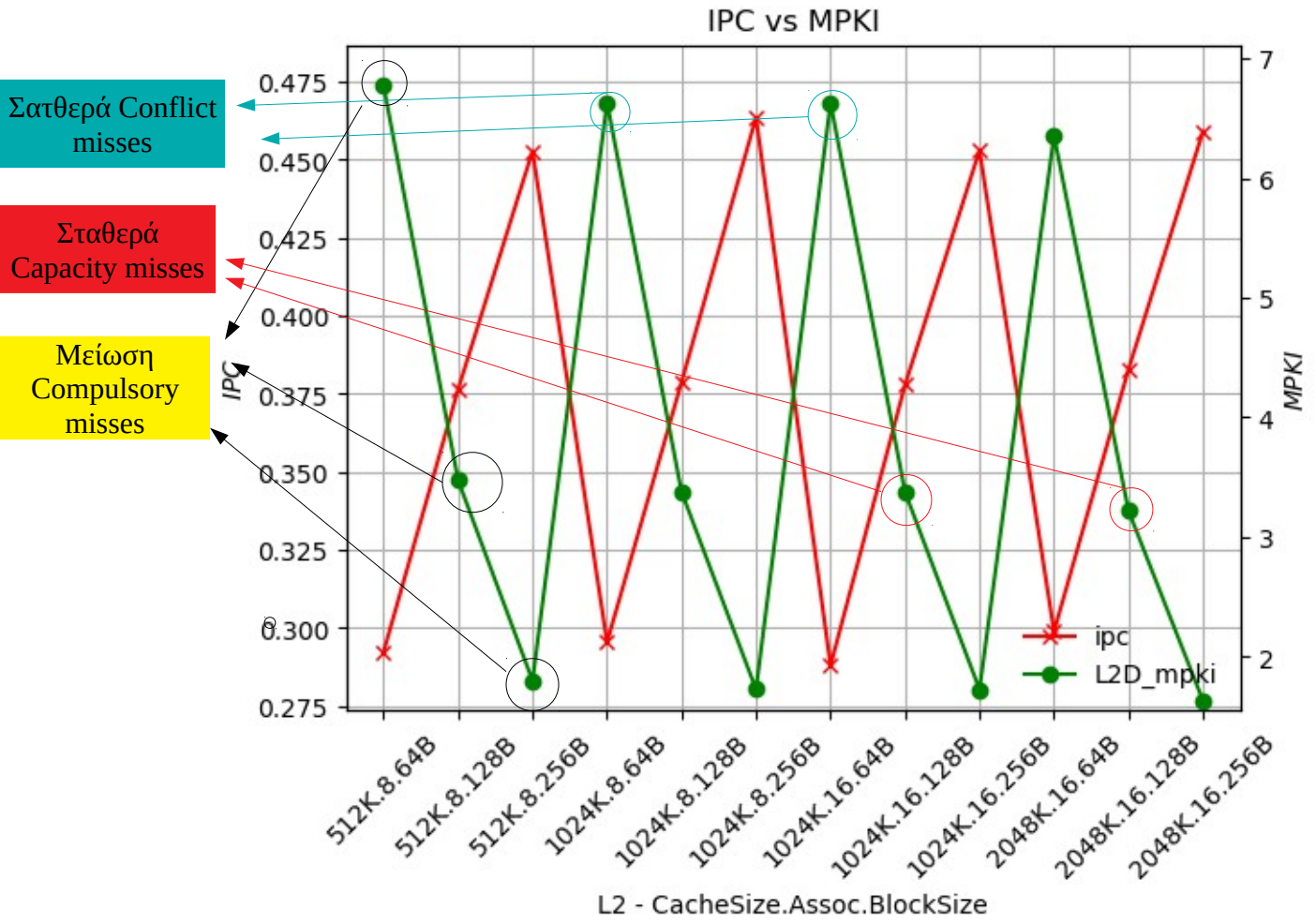
Την βέλτιστη απόδοση την έχουμε για τους συνδυασμούς: 1024K.8.128B, 1024K.16.256B και 2048K.16.256B. Θα επιλξουμε την 1024K.8.128B λόγω του χαμηλότερου κόστους (μικρότερο cache-size) και της μεγαλύτερης ταχύτητας (μικρότερο assocaitivity).

Canneal



Στην συγκριμένη εφαρμογή παρατηρούμε μια ασυμμετρία στην συμπεριφορά των μετρικών μας καθώς η αύξηση του mpki συνοδεύεται από αύξηση του ipc. Είναι δύσκολο να αντλήσουμε συμπεράσματα από την συγκεκριμένη γραφική παράσταση, αν και η αύξηση του cache-size φαίνεται να βελτιώνει την απόδοση. Οι μεταβολές στα block-size και associativity μας δίνουν αντιφατικά αποτελέσματα και δεν μπορούμε να είμαστε σίγουροι. Μια καλή επιλογή είναι 2048K.16.64B .

Facesim



Παρατηρούμε ότι αύξηση του block-size βελτιώνει την απόδοση της εφαρμογής. Αντίθετα οι αλλαγές στο cache-size και associativity δεν επηρεάζουν ιδιαίτερα.

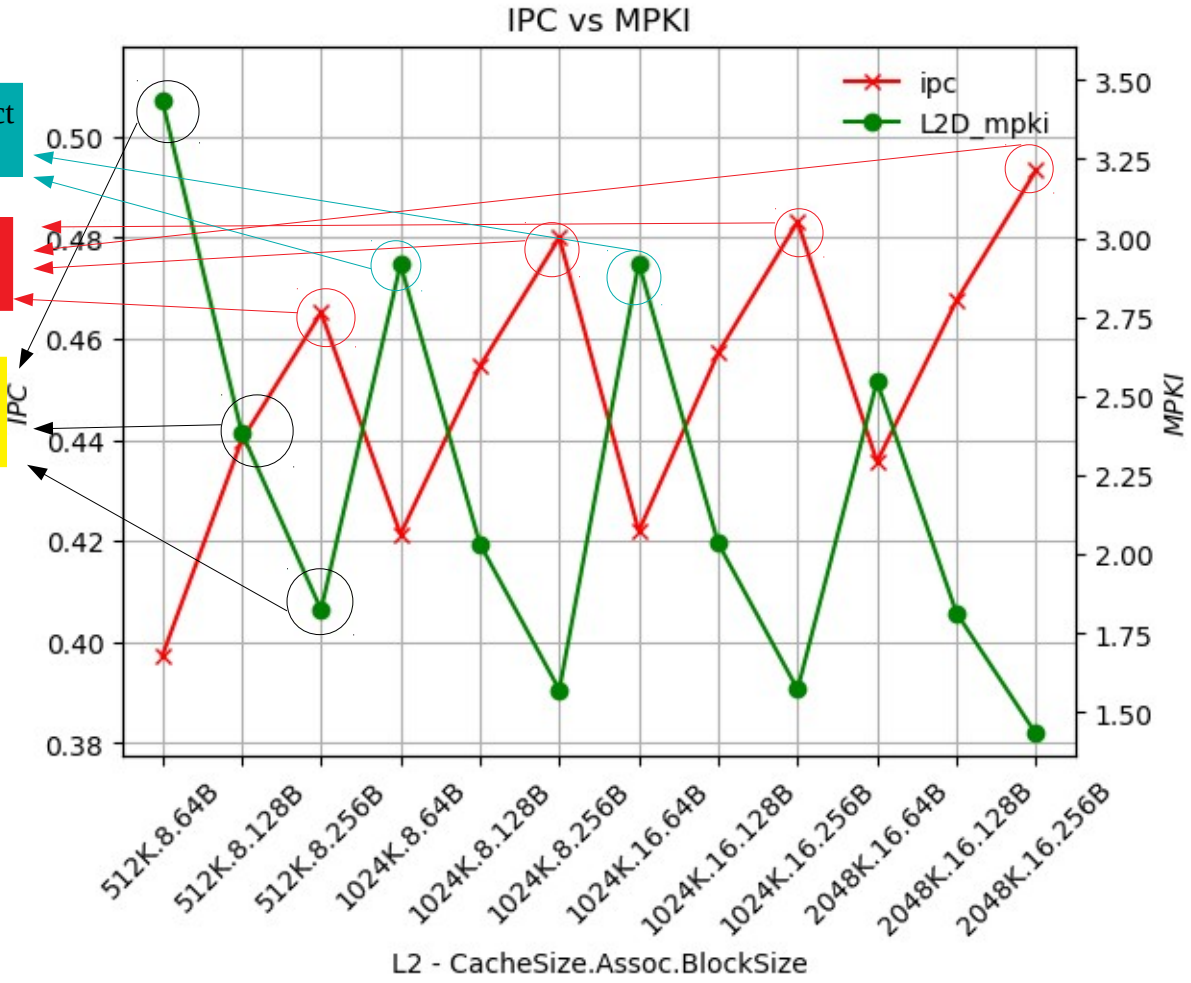
Η εφαρμογή αποδίδει καλύτερα για μεγάλο bolck-size ενω τα cache-size και associativity δεν διαμορφώνουν διαφορετικό αποτέλεσμα. Την βέλτιστη απόδοση την έχουμε για τον συνδυασμό 1024K.8.256B και είναι μια καλή επιλογή, από πλευράς κόστους(μεσαίο cache-size) και ταχύτητας(λόγω associativity).

Ferret

Σατθερά Conflict misses

Μικρή μείωση Capacity misses

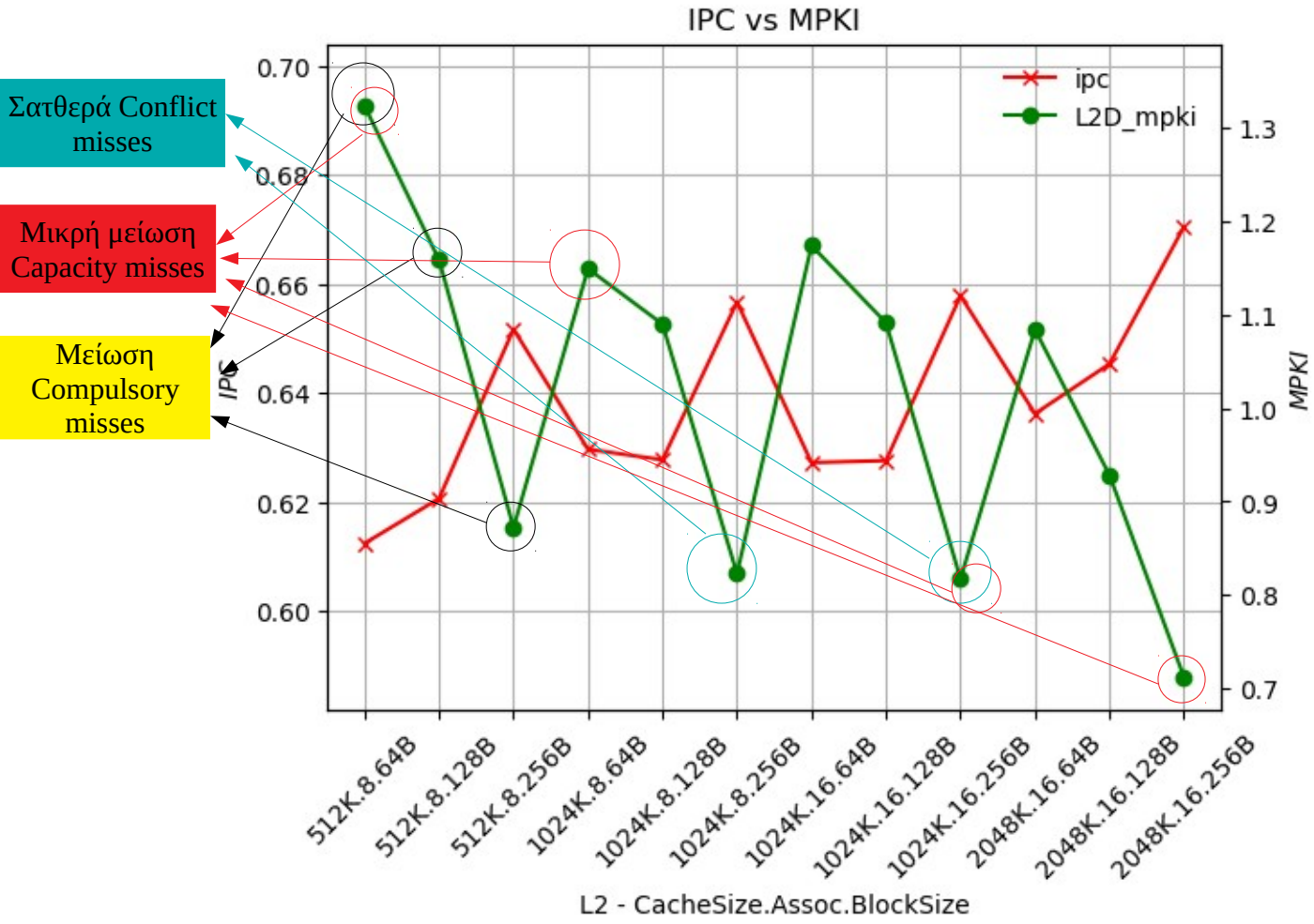
Μείωση Compulsory misses



Από το διάγραμμα παρατηρούμε ότι η αύξηση του block-size βελτιώνει την απόδοση. Μικρή βελτίωση έχουμε και από την αύξηση του cache-size ενώ η αλλαγή του βαθμού του associativity δεν επηρεάζει καθόλου.

Συμπεραίνουμε ότι η απόδοση της εφαρμογής εξαρτάται κυρίως από το block-size, σε μικρότερο βαθμό από το cache-size ενώ καθόλου από το associativity. Την βέλτιστη απόδοση την έχουμε για την 2048K.16.256B ωστόσο μια επιλογή, με παρόμοια αποτελέσματα και χαμηλότερο κόστος θα ήταν 1024K.8.256B .

Fluidanimate

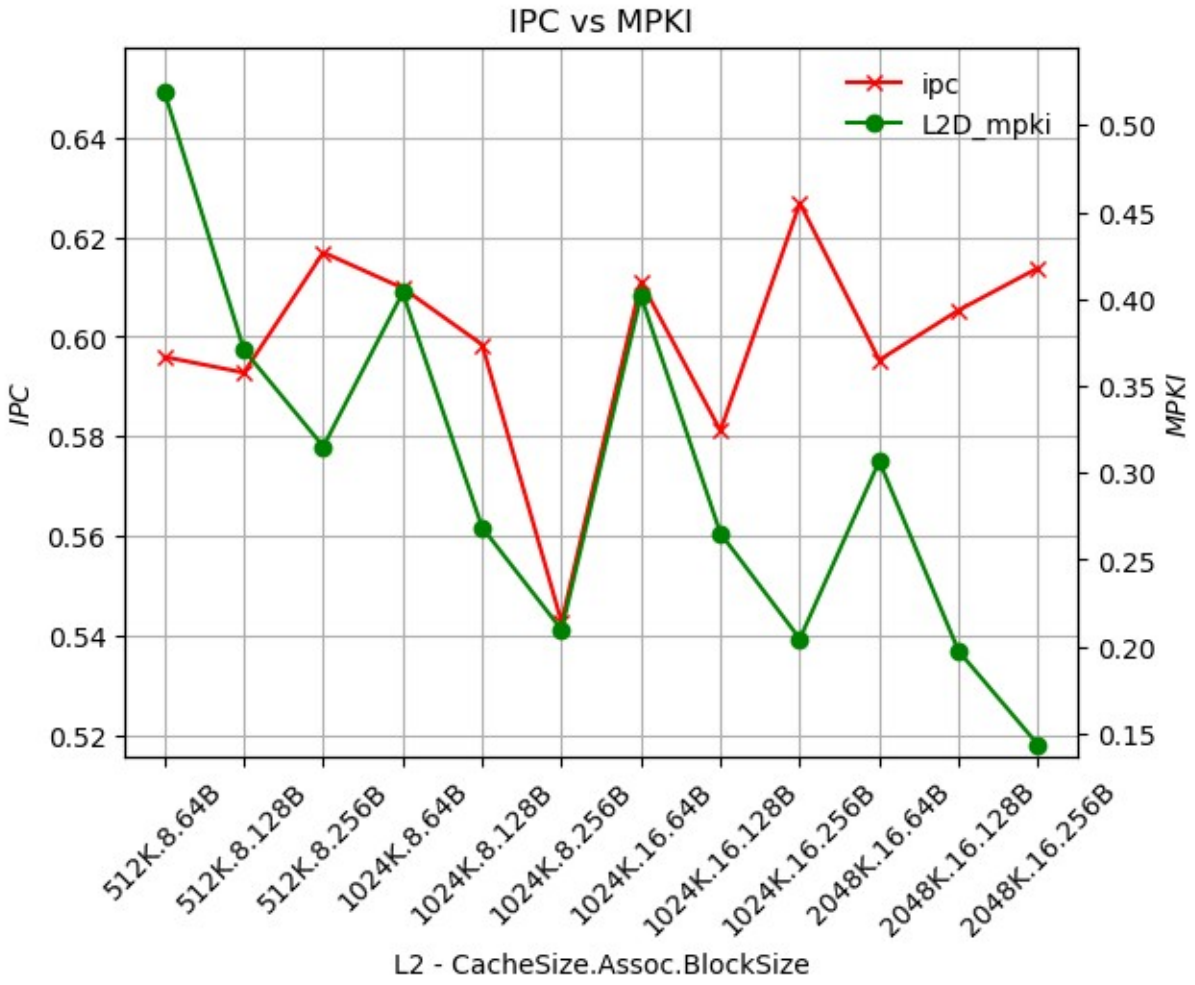


Από το διάγραμμα παρατηρούμε ότι η αύξηση του block-size βελτιώνει την απόδοση της εφαρμογής. Περαιτέρω βελτίωση έχουμε με την αύξηση του cache-size, αντίθετα ο βαθμός associativity δε παρουσιάζει αλλαγές.

Η απόδοση της εφαρμογής εξαρτάται κυρίως από το block-size, σε μικρότερο βαθμό από το cache-size ενώ το associativity δεν την επηρεάζει . Ωστόσο, η αύξηση που παρατηρούμε είναι μικρή .

Την βέλτιστη απόδοση την έχουμε για 2048K.16.256B . Για μικρότερο κόστος (cache-s-ze) και πιθανώς μεγαλύτερη ταχύτητα (μικρότερο associativity) καλή επιλογή είναι 1024K.8.256B .

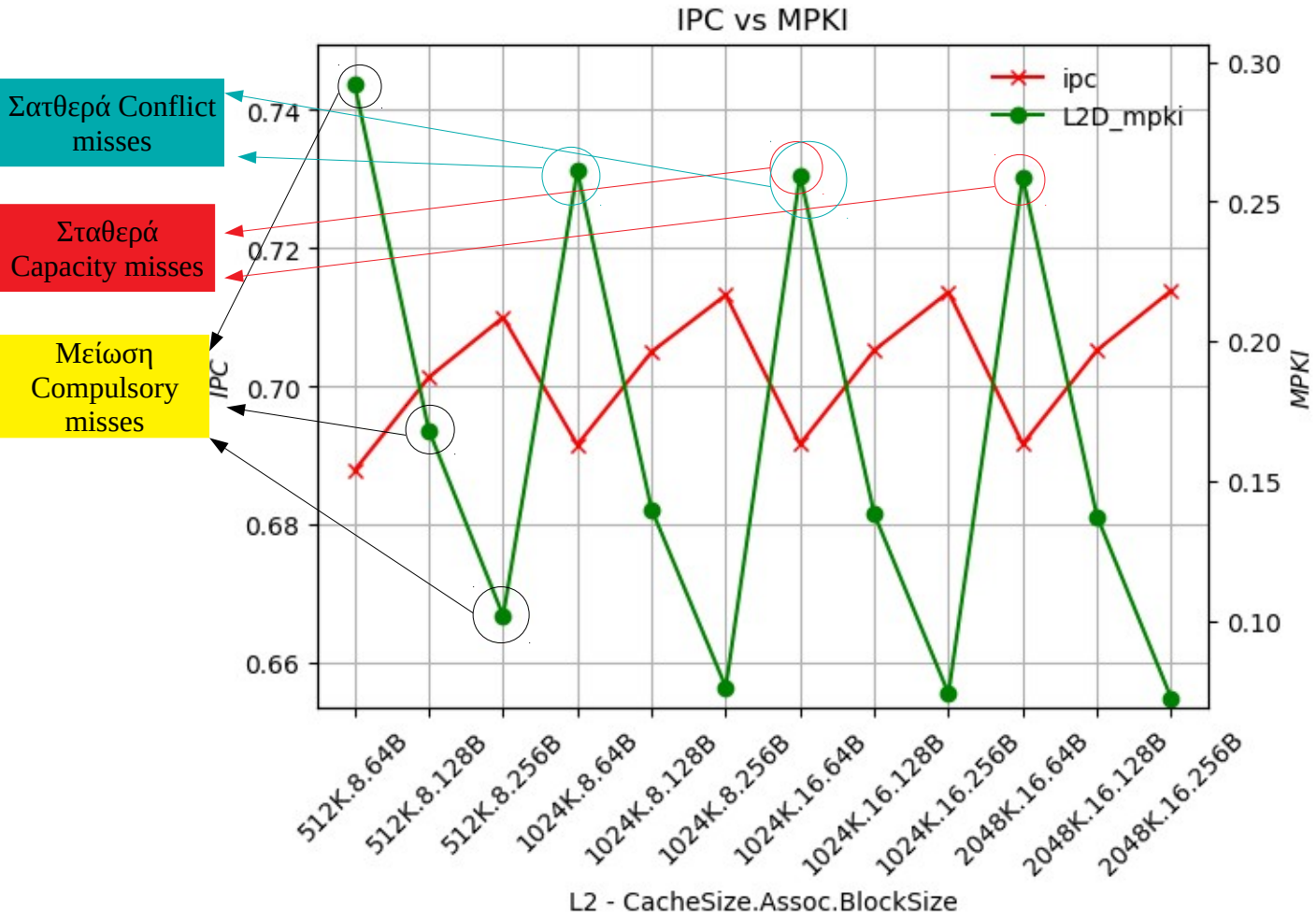
Freqmine



Στην συγκεκριμένη εφαρμογή είναι δύσκολο να οδηγηθούμε σε σωστά συμπεράσματα. Αρχικά παρατηρούμε αντιφατική συμπεριφορά των μετρικών, αφού έχουμε ταυτόχρονη αύξηση των mpki και ipc . Ακόμα, η αύξηση του associativity δεν μας δίνει σταθερά αποτελέσματα καθώς μπορούμε να οδηγηθούμε σε καλύτερες και χειρότερες αποδόσεις. Παρόμοια απόκριση έχουμε για το cache-size και block-size. Δε μπορούμε να αποφανθούμε για το τι βελτιώνει την απόδοση της εφαρμογής.

Την βέλτιστη απόδοση την έχουμε για 1024K.16.256B .

Raytrace

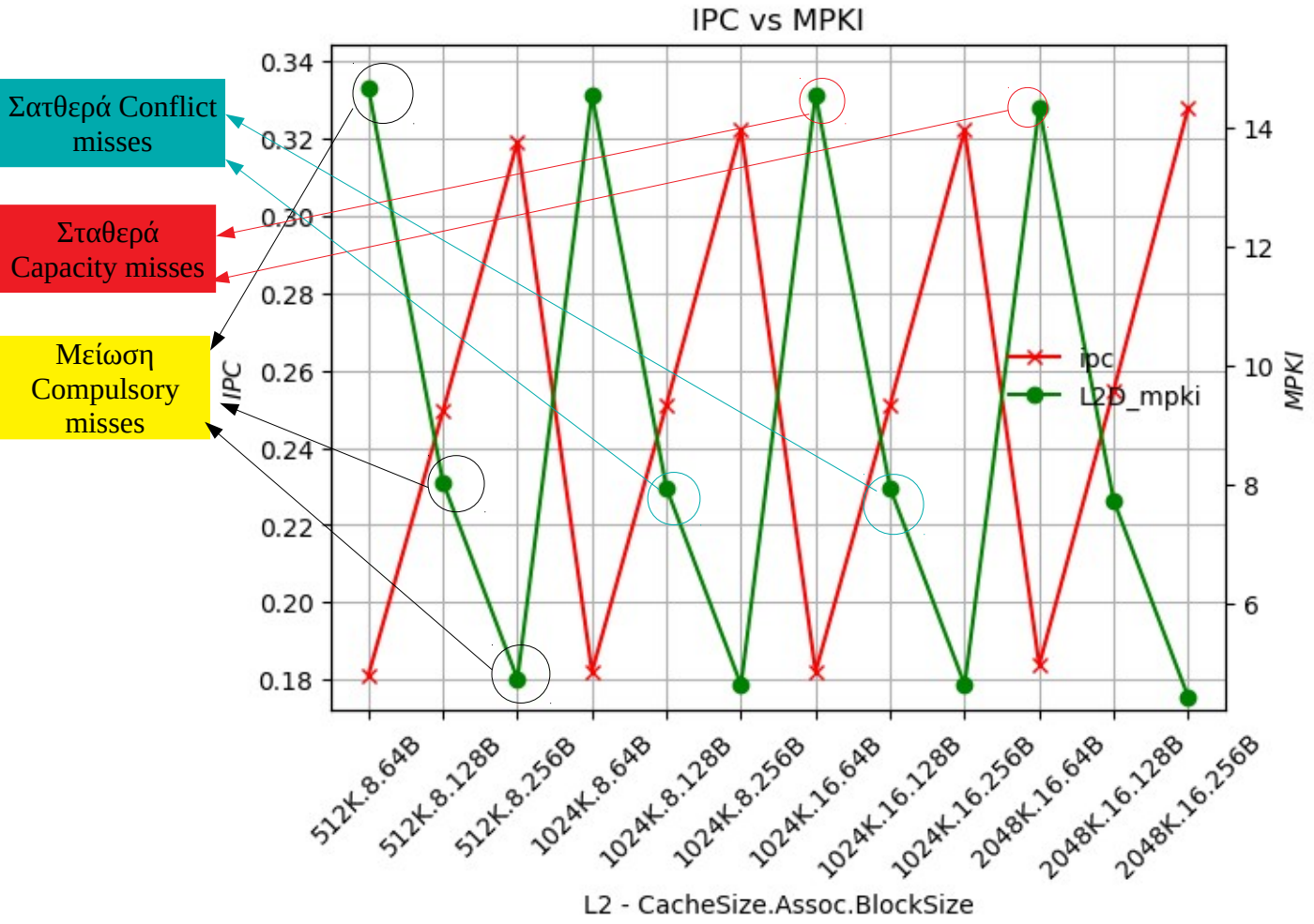


Παρατηρούμε ότι αύξηση του block-size βελτιώνει την απόδοση της εφαρμογής. Αντίθετα οι αλλαγές στο cache-size και associativity δεν επηρεάζουν ιδιαίτερα.

Η εφαρμογή αποδίδει καλύτερα για μεγάλο bolck-size ενω τα cache-size και associativity δεν διαμορφώνουν διαφορετικό αποτέλεσμα. Ωστόσο, η αύξηση που παρατηρούμε είναι μικρή .

Την βέλτιστη απόδοση την έχουμε για τους συνδυασμούς: 1024K.8.256B, 1024K.16.256B και 2048K.16.256B . Μια καλή.επιλογή είναι 1024K.8.256B , από πλευράς κόστους(μεσαίο cache-size) και ταχύτητας(λόγω associativity).

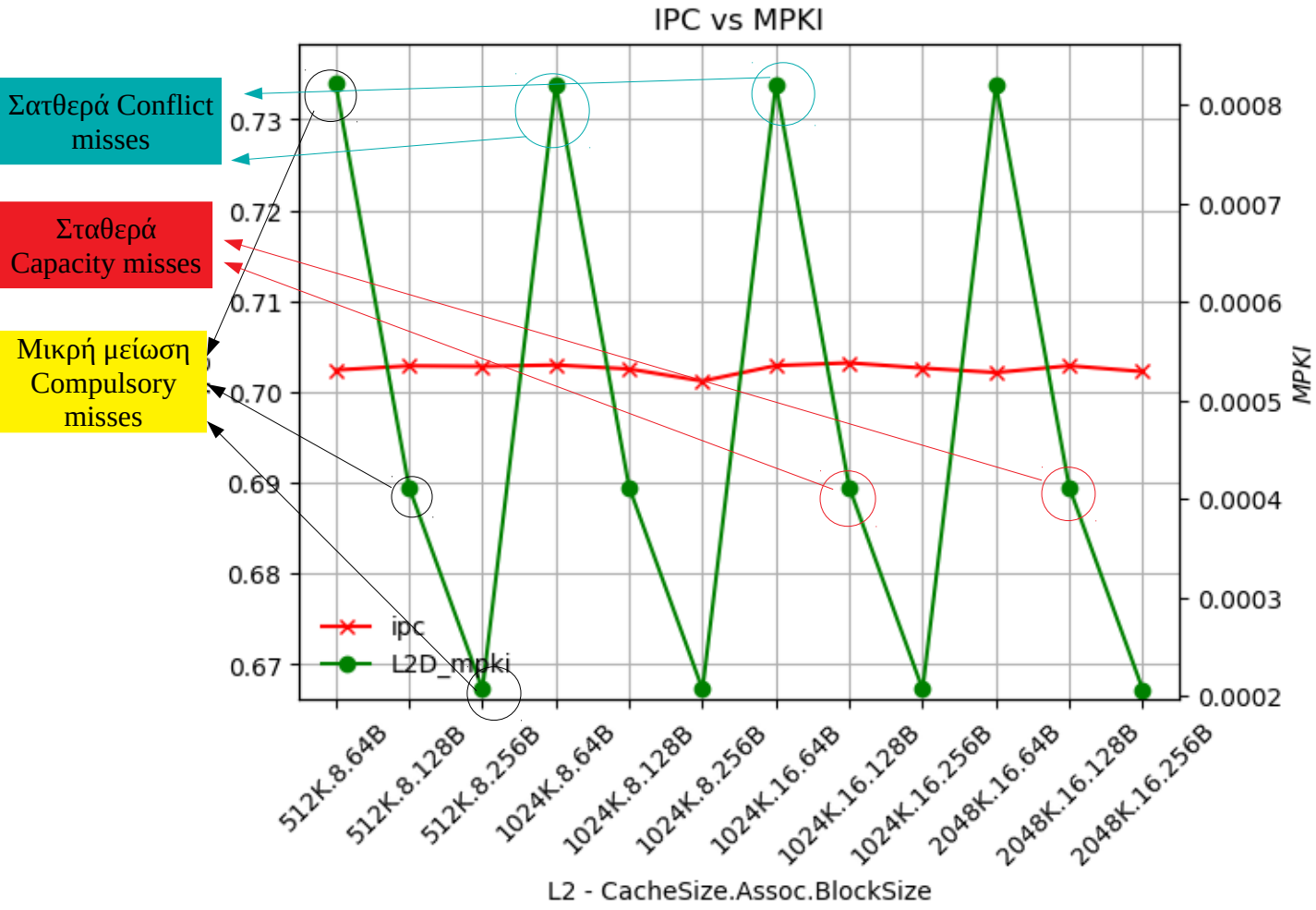
Streamcluster



Ομοίως με την προηγούμενη εφαρμογή .

Την βέλτιστη απόδοση έχουμε για τους συνδυασμούς : 512K.8.256B, 1024K.8.256B, 1024K.16.256B και 2048K.16.256B. Μια καλή επιλογή είναι 512K.8.256B , από πλευράς κόστους(μικρό cache-size) και ταχύτητας(λόγω μικρότερου associativity).

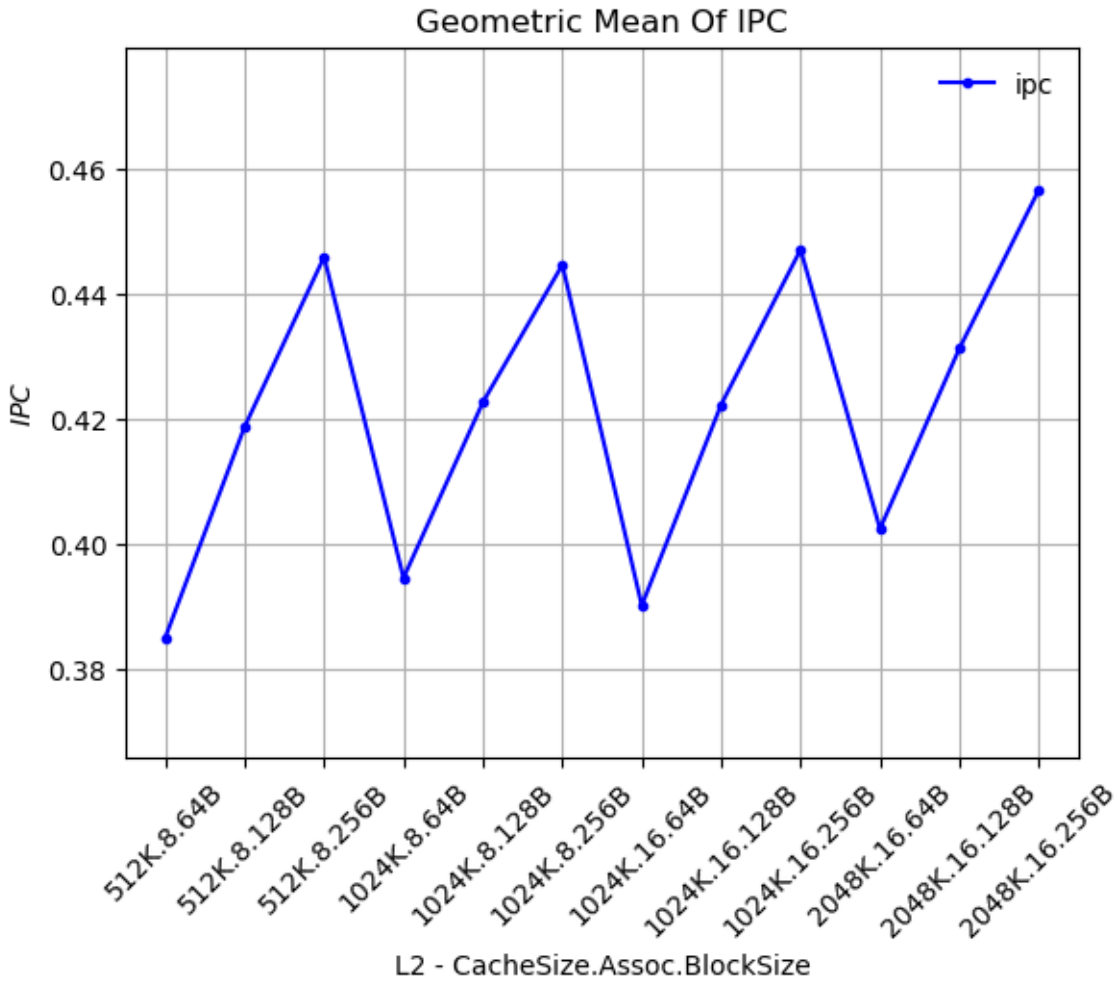
Swaptions



Από την γραφική παράσταση παρατηρούμε ότι αν και η αύξηση του block-size μειώνει το miss-rate, δεν είναι αρκετό για να βελτιώσει την απόδοση του προγράμματος. Ομοίως, οι μεταβολές στα cache-size και associativity δεν επηρεάζουν κάπως.

Επειδή απόδοση της εφαρμογής παραμένει σταθερή και το miss-rate είναι εξαιρετικά μικρό ,παρά τις αλλαγές στην L2 cache, μπορούμε να οδηγηθούμε στο συμπέρασμα ότι γίνεται μεγαλύτερη χρήση της L1 και όχι της L2. Οπότε στην συγκεκριμένη περίπτωση μπορούμε να χρησιμοποιήσουμε την απλούστερη υλοποίηση , 512K.8.64B .

Συμπεράσματα



Αρχικά παρατηρούμε ότι οι μεταβολές για τα χαρακτηριστικά της L2 οδηγούν σε μεγαλύτερο εύρος αποδόσεων από την L1, L2: 0.38-0.46 και L1: 0.42-0.43 . Οπότε συμπεραίνουμε ότι η L2 παίζει πιο καθοριστικό ρόλο, από την L1, στην απόδοση των εφαρμογών.

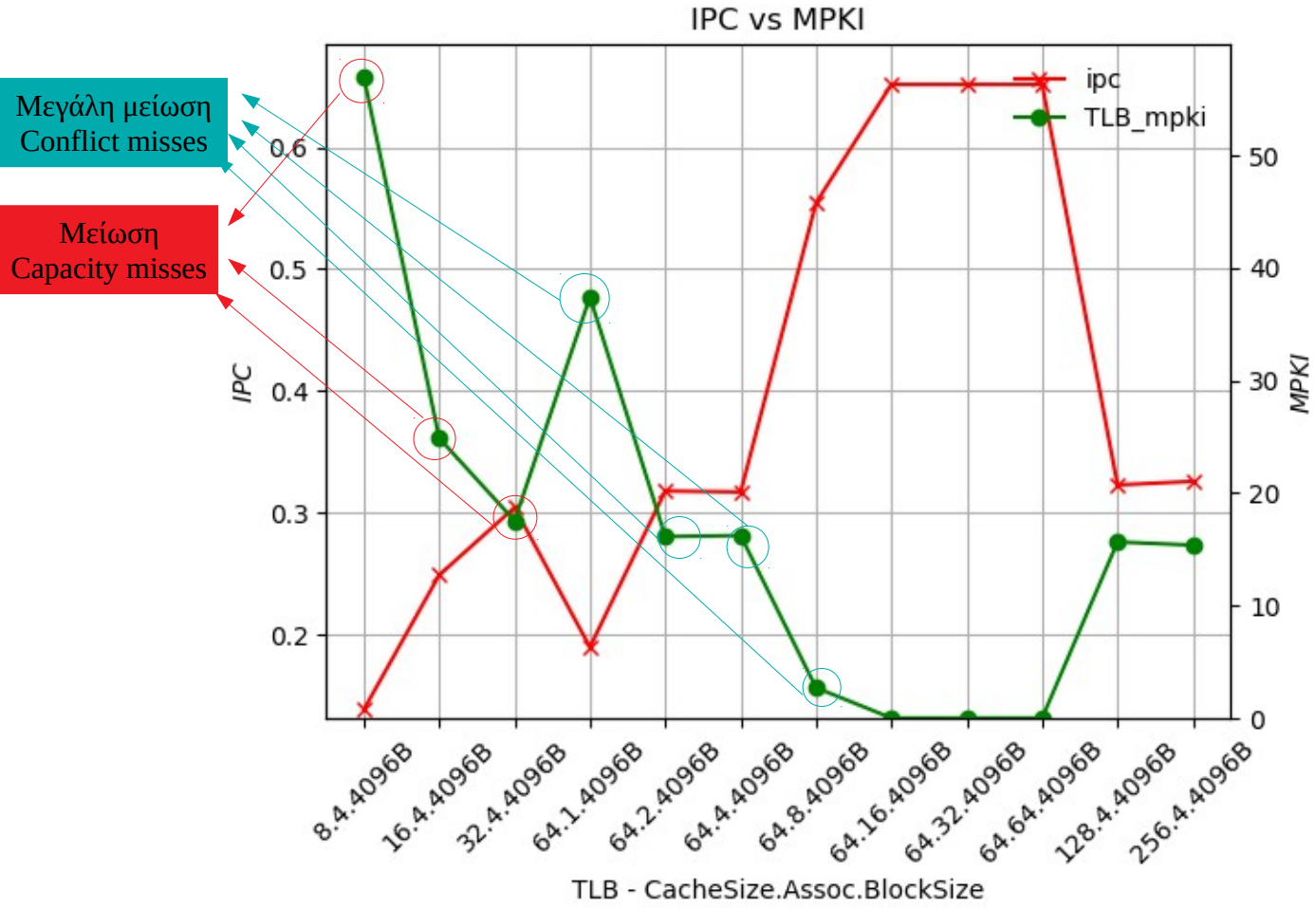
Ομοίως με την L1, η αύξηση του block-size φέρνει τα καλύτερα αποτελέσματα, πχ 2048K.16.64B-> 2048K.16.128B -> 2048K.16.256B . Το cache-size παίζει ελάχιστο ρόλο, αν και μια μεγάλη αύξηση του (διπλασιασμός) καταφέρνει μικρή βελτίωση στην απόδοση του, πχ 1024K.16.256B -> 2048K.16.256B . Ο βαθμός του associativity δεν φαίνεται να επηρεάζει την γενικότερη απόδοση των εφαρμογών.

Η καλύτερη επιλογή για την L2 από πλευράς κόστους φαίνεται να είναι η 512K.8.256B, λόγω του μικρότερου cache-size. Ακόμα, χάριν του μικρότερου βαθμού associativity πετυχαίνουμε καλύτερες ταχύτητες από τις άλλες υλοποιήσεις.

➢ TLB

Σκοπός αυτού του ερωτήματος είναι να μελετήσουμε την επίδραση στην απόδοση ενός προγράμματος, μεταβάλλοντας τις βασικές παραμέτρους (όπως το tlb-size, associativity και page-size) μιας TLB μνήμης και κρατώντας σταθερά τα χαρακτηριστικά των L1 και L2 cache. Ακολουθούν γραφικές παραστάσεις για τους διαφορετικούς συνδυασμούς tlb-size και associativity που ελέγξαμε για κάθε benchmark , σε σχέση με το ipc και mkpi.

Blacksholes

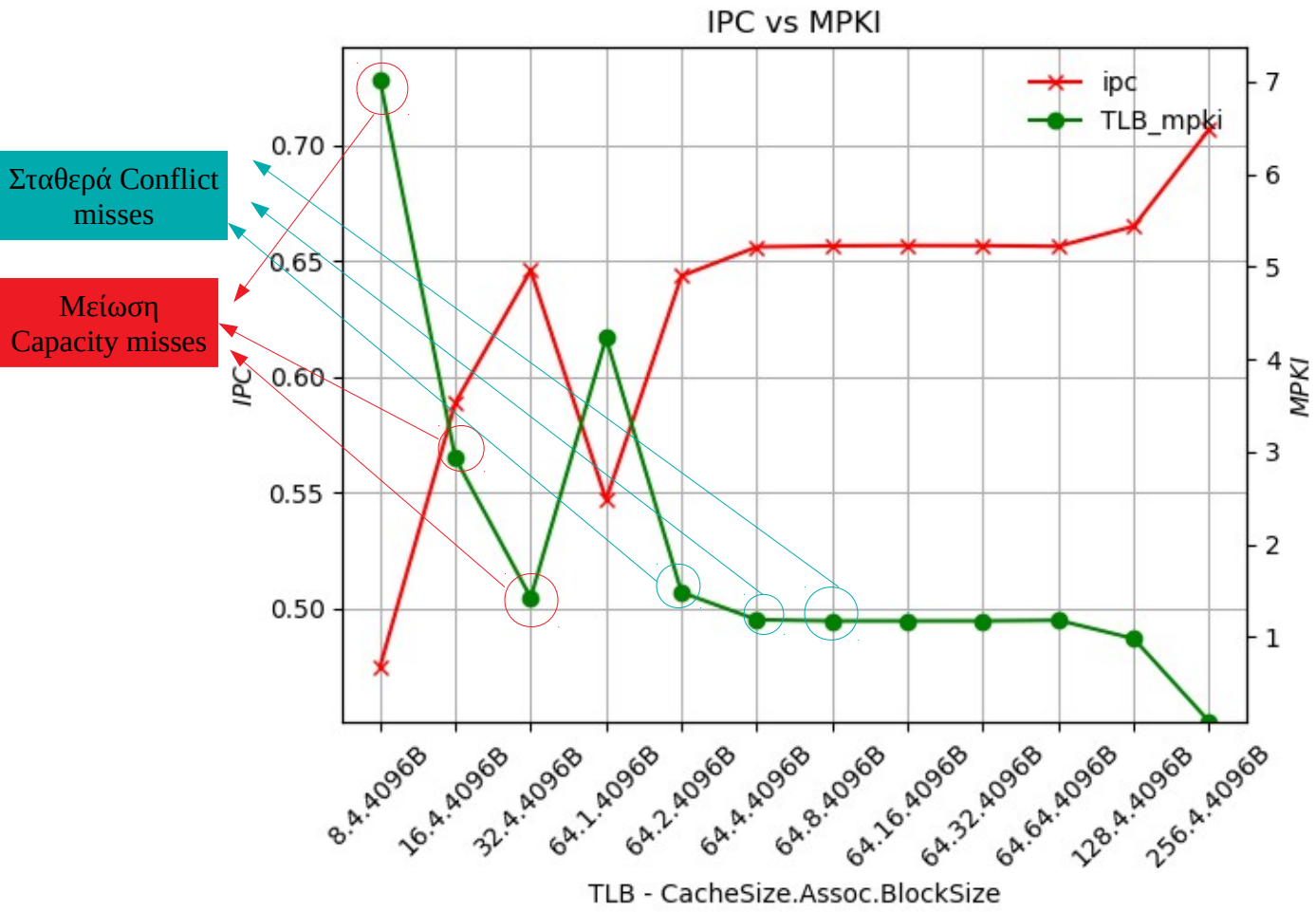


Από το διάγραμμα παρατηρούμε ότι με την αύξηση του associativity βελτιώνει σημαντικά την απόδοση, αν και μετά από βαθμό 16 παραμένει σταθερή. Η αρχικές αυξήσεις του tlb-size φαίνεται να αυξάνει την απόδοση, ωστόσο μετά από ένα σημείο (64 entries) δεν φέρνει κάποια αλλαγή .

Συμπεραίνουμε ότι η απόδοση της εφαρμογής εξαρτάται κυρίως από τον βαθμό associativity και σε μικρότερο επίπεδο από το tlb-size.

Την βέλτιστη απόδοση την πετυχαίνουμε για 64.16.4096B όπου και θα επιλέξουμε.

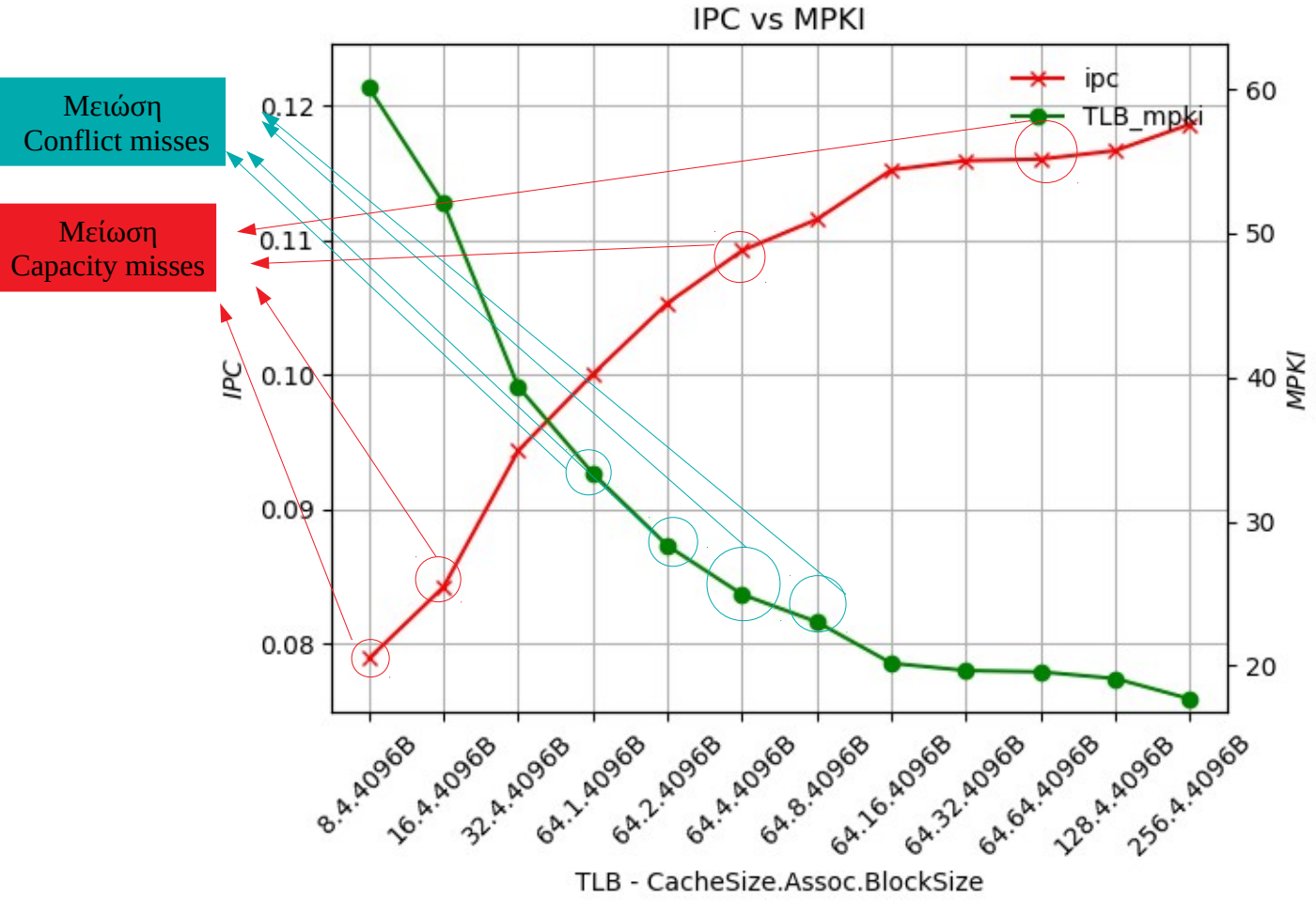
Bodytrack



Από το διάγραμμα παρατηρούμε η αύξηση του tlb-size βελτιώνει σημαντικά την απόδοση. Το associativity επηρεάζει την απόδοση μόνο για μικρές τιμές (βαθμό 1 και 2) , η περαιτέρω αύξηση δεν φέρνει διαφορετικά αποτελέσματα.

Συμπεραίνουμε ότι η απόδοση εξαρτάται κυρίως από το tlb-size και σε μικρότερο βαθμό από το associativity. Για να έχουμε κάποια διαφοροποίηση στην απόδοση, με την αύξηση του tlb-size, για μεγέθη μεγαλύτερα των 32 entries, θα πρέπει να φτάσουμε στις 256 entries. Έχουμε μεγάλη διαφορά κόστους για μικρή βελτίωση στην απόδοση, οπότε καλύτερη επιλογή είναι 32.4.4096 .

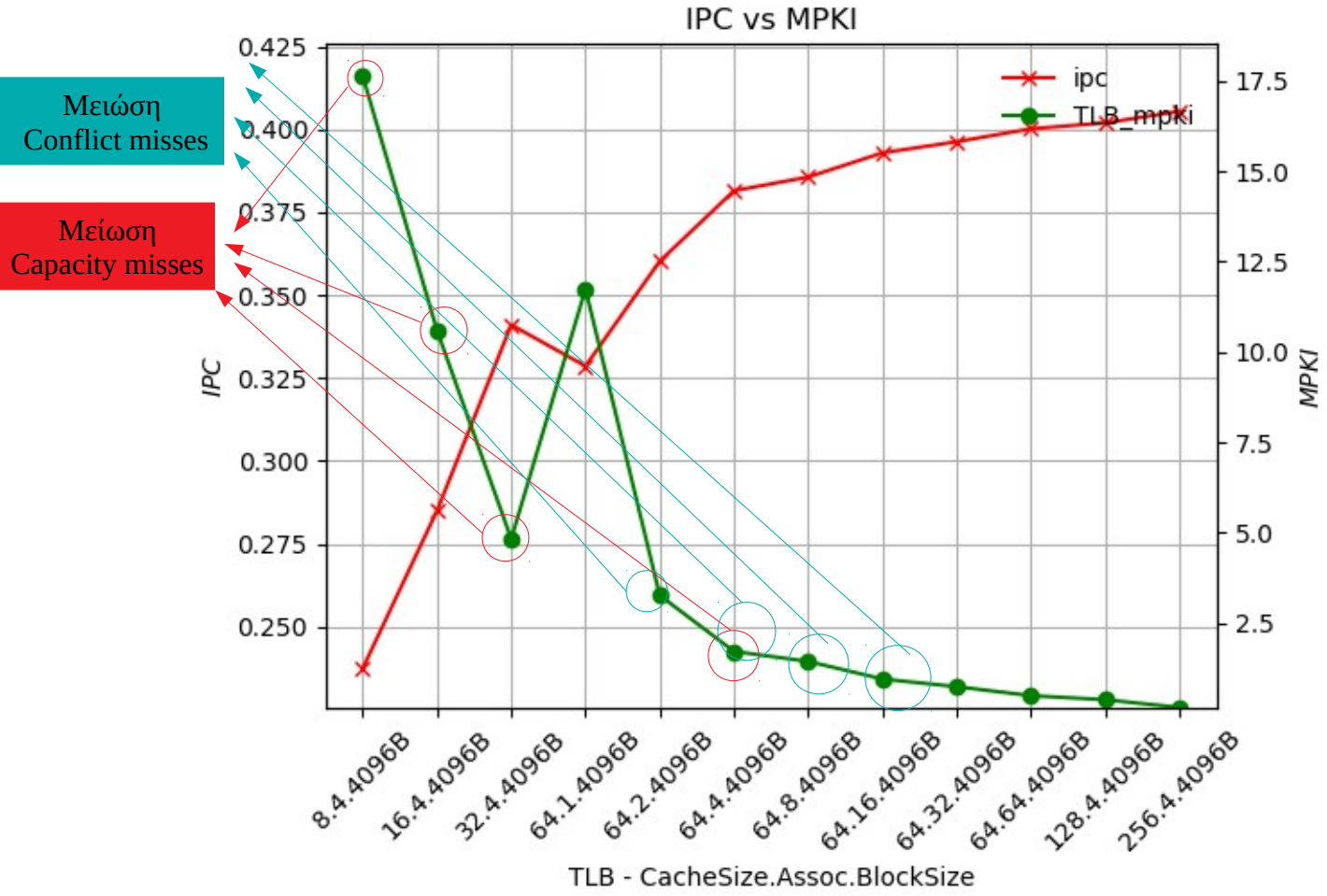
Canneal



Ομοίως με την προηγούμενη εφαρμογή, παρατηρούμε ότι η αύξηση του tlb-size βελτιώνει την απόδοση . Η επίδραση του associativity είναι αντίστοιχη, αν και λιγότερο δραστική. Οπότε συμπεραίνουμε ότι η απόδοση εξαρτάται κυρίως από το tlb-size και σε μικρότερο βαθμό από τον βαθμό associativity.

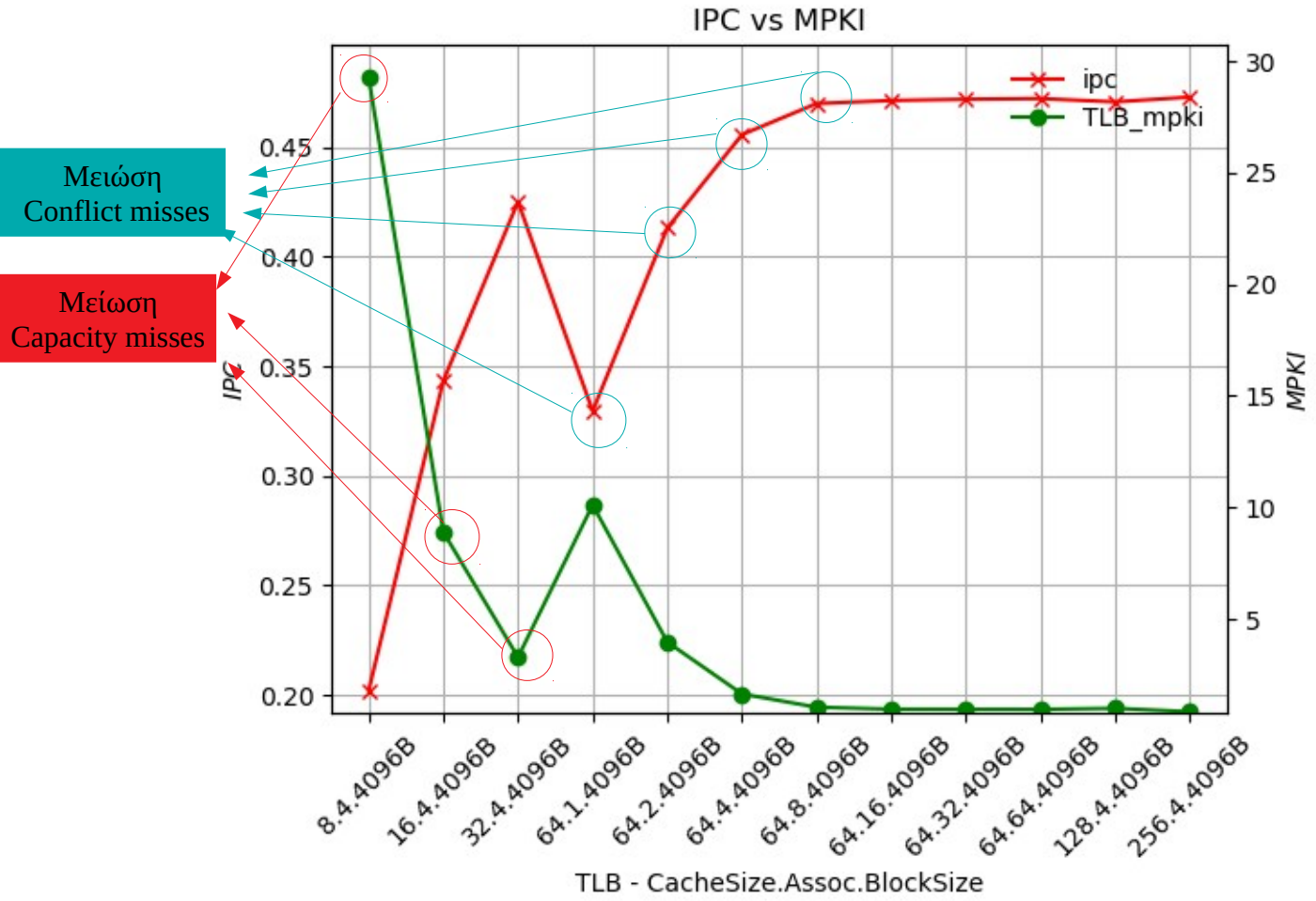
Την βέλτιστη απόκριση (ipc: 0.12) την έχουμε για 256.4.4096B, ωστόσο για σημαντικά λιγότερη μνήμη έχουμε παρόμοιο αποτέλεσμα (ipc: 0.11). Άρα, μία καλή επιλογή είναι 64.4.4096B .

Facesim



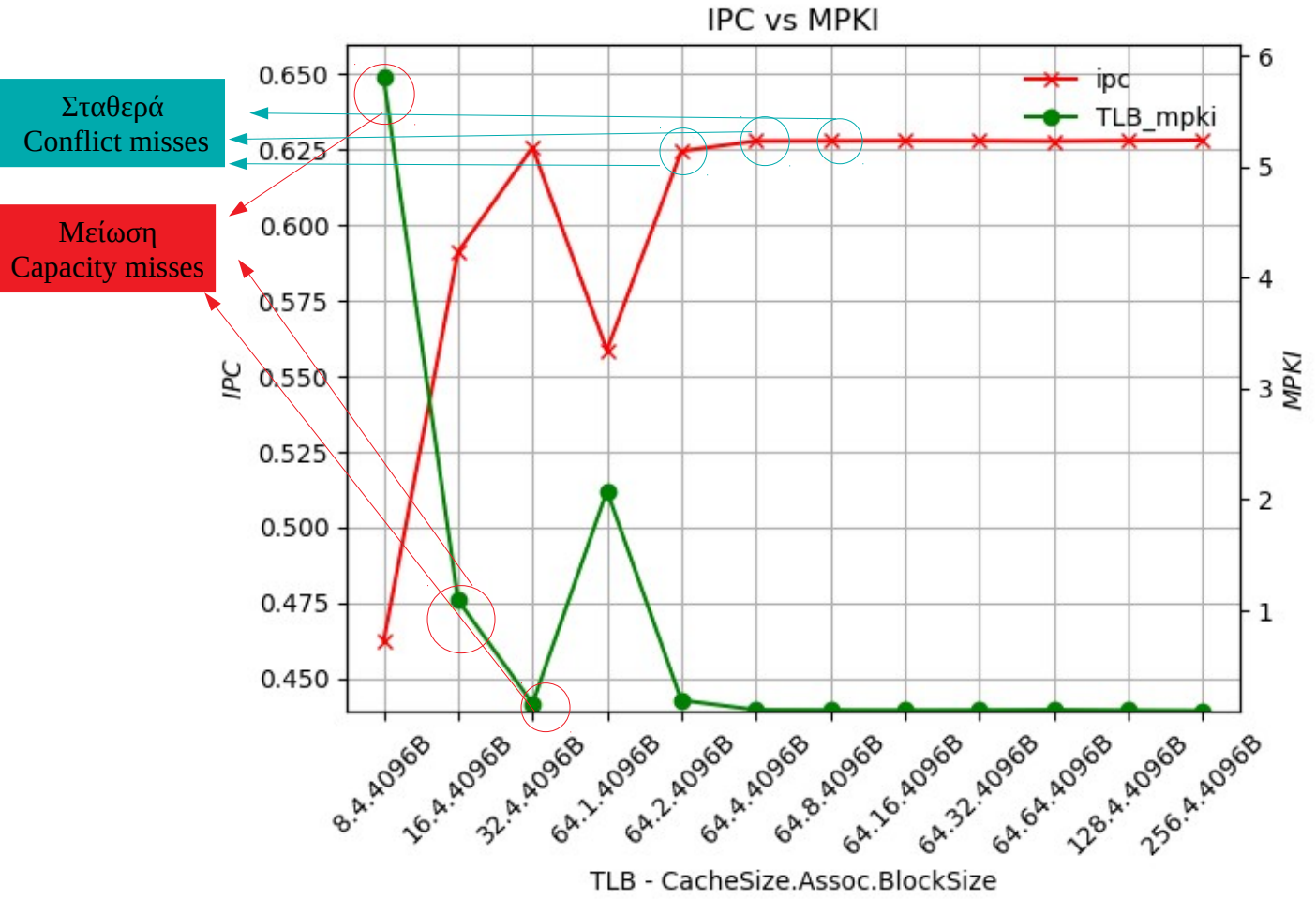
Ομοίως με την προηγούμενη εφαρμογή καθοριστικό ρόλο στην βελτίωση απόδοσης παίζει το tlb-size ενώ σε μικρό βαθμό επηρεάζει το associativity . Όμως για να υπάρχει αλλαγή στην απόκριση, με την αύξηση του tlb-size, θα πρέπει να έχουμε 2-way ή μεγαλύτερη associativity. Μετά από ένα σημείο(64 entries & 4-way), η αύξηση και των δύο δε αλλάζουν σημαντικά το αποτέλεσμα, οπότε μια καλή επιλογή είναι 64.4.4096B .

Ferret



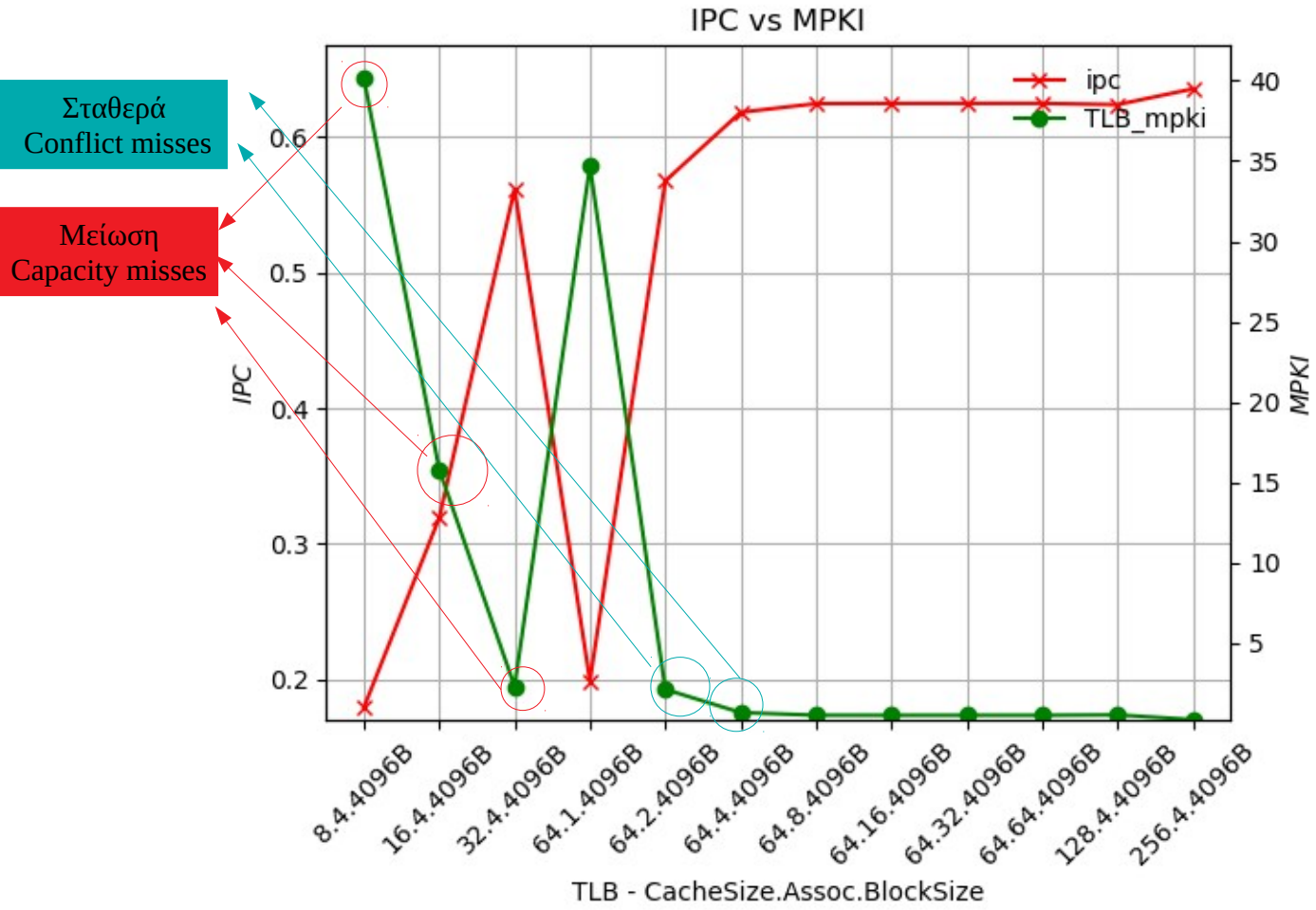
Αντίστοιχη ανάλυση με την προηγούμενη εφαρμογή, μόνο που θα πρέπει ο βαθμός του associativity να είναι μεγαλύτερος ή ίσος του 4. Κατάλληλη επιλογή είναι 64.4.4096B

Fluidanimate



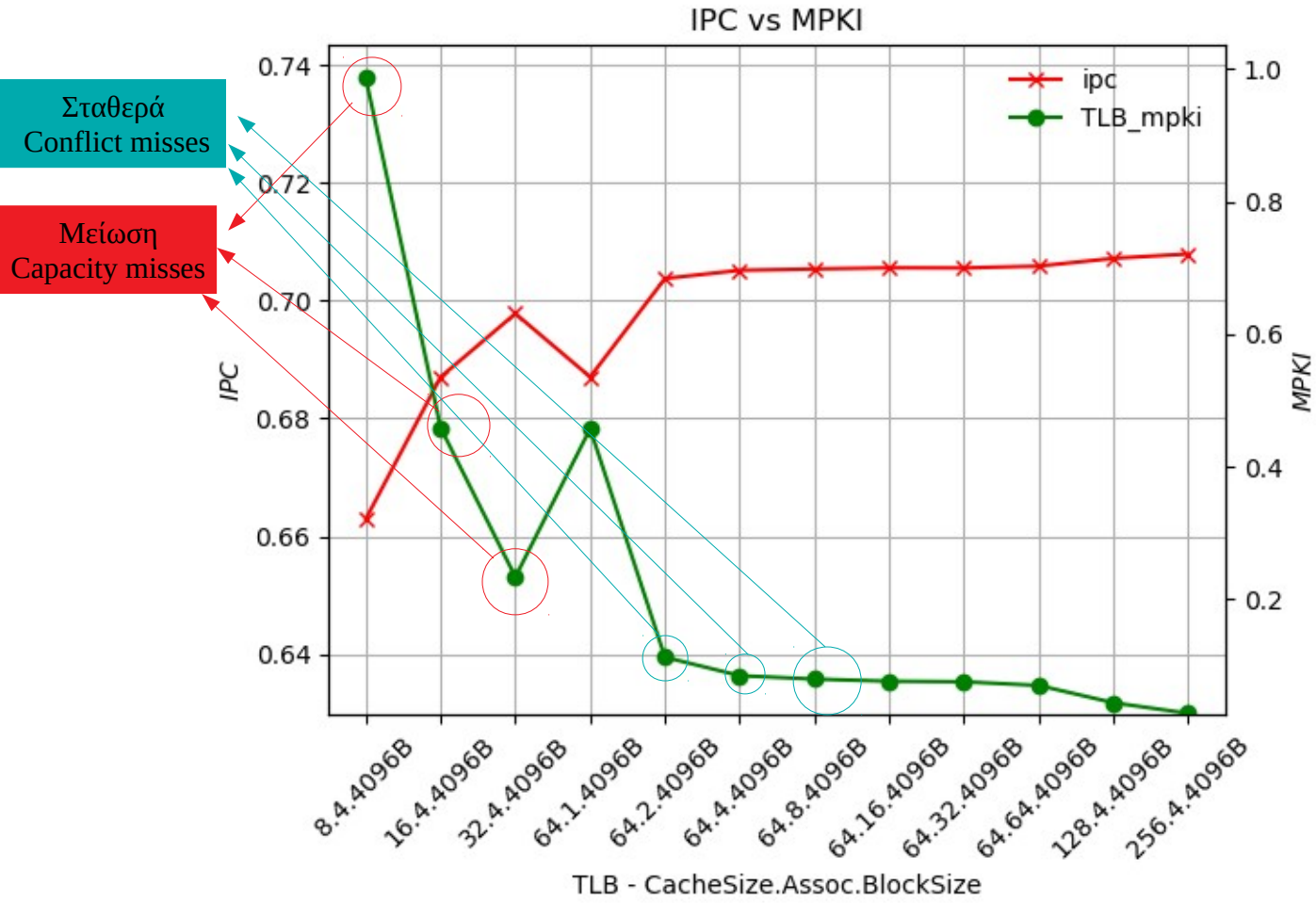
Παρατηρούμε ότι η αύξηση του tlb-size βελτιώνει σημαντικά την απόδοση, όμως δε έχουμε καλύτερη απόκριση για μεγέθη μεγαλύτερα των 32 entries. Ο βαθμός του associativity δεν επηρεάζει την απόδοση, αρκεί να είναι direct-mapped η μνήμη. Μια καλή επιλογή είναι 32.4.4096B .

Freqmine



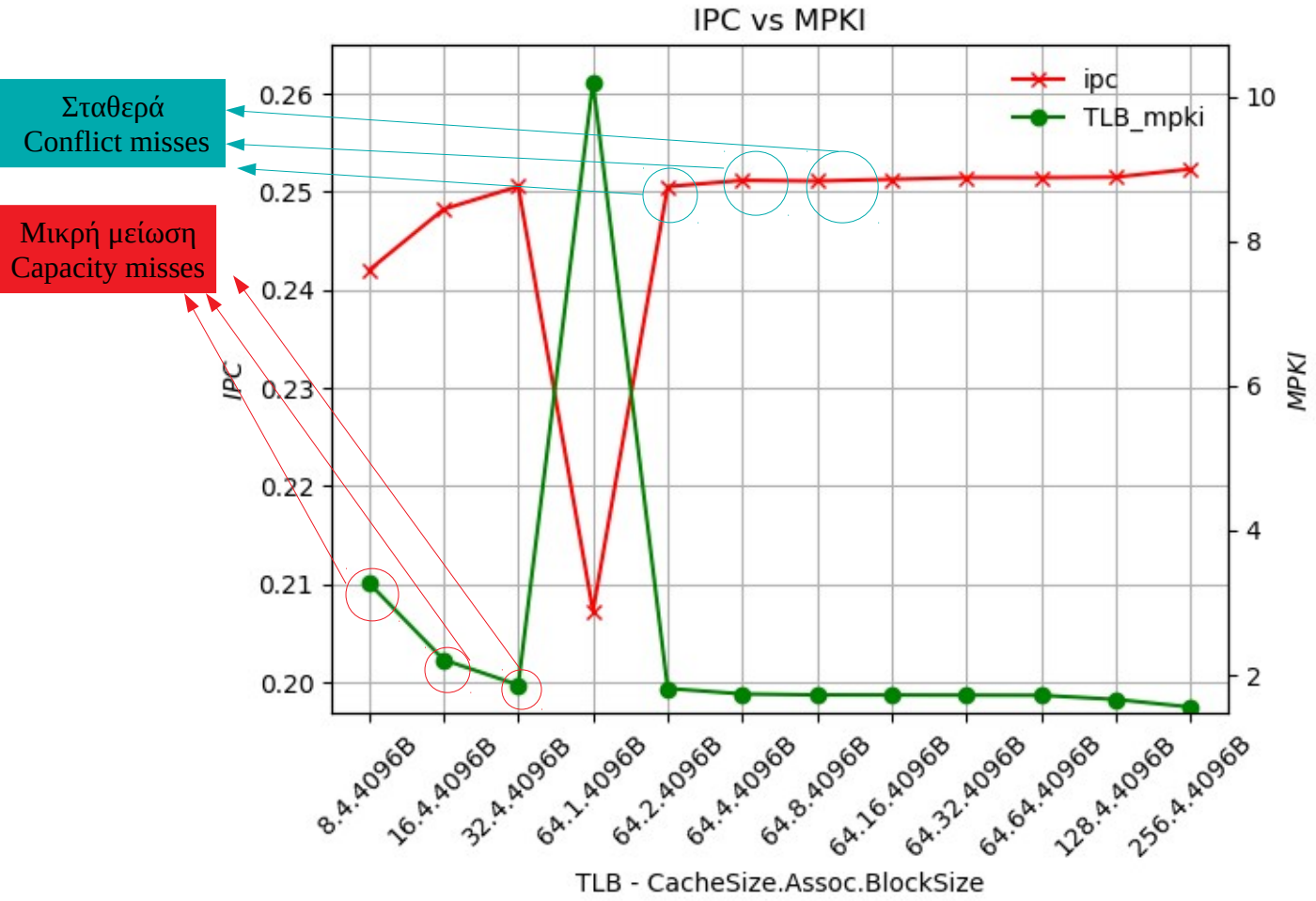
Παρατηρούμε ότι η αύξηση του tlb-size βελτιώνει σημαντικά την απόδοση, όμως δε έχουμε καλύτερη απόκριση για μεγέθη μεγαλύτερα των 64 entries. Ο βαθμός του associativity δεν επηρεάζει την απόδοση, αρκεί να είναι direct-mapped η μνήμη. Μια καλή επιλογή είναι 64.4.4096B .

Raytrace



Ίδια ανάλυση με fluidanimate. Μία καλή επιλογή είναι 32.4.4096B .

Streamcluster

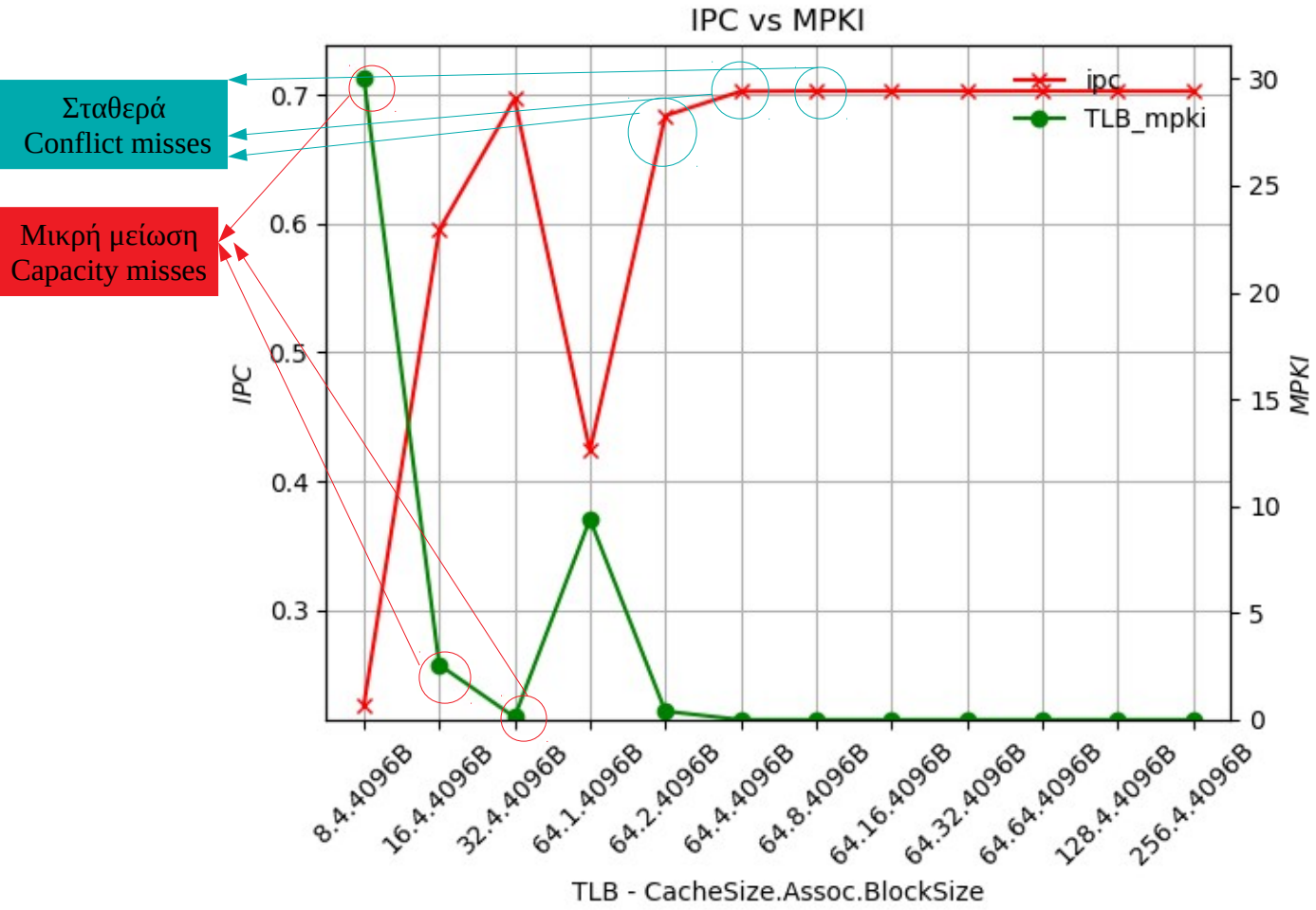


Σταθερά
Conflict misses

Μικρή μείωση
Capacity misses

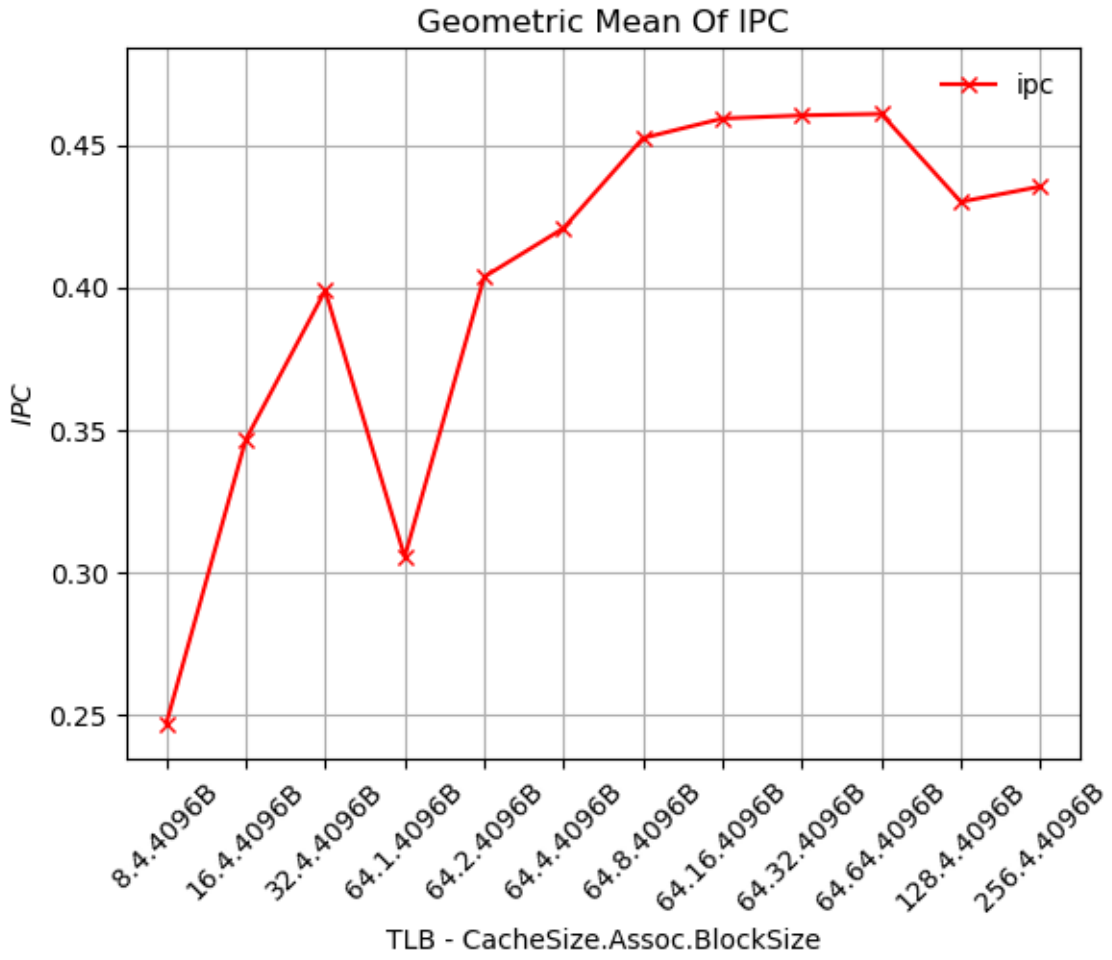
Από το διάγραμμα παρατηρούμε ότι η αρχική αύξηση στο tlb-size προκαλεί μικρή βελτίωση στην απόδοση. Ο βαθμός associativity δεν επηρεάζει την απόδοση, αρκεί να μην είναι direct-map η μνήμη, όπου και πέφτει σημαντικά. Επειδή η αύξηση του tlb-size, μετά των 32 entries κρατά σταθερά τα αποτελέσματα , μία καλή επιλογή θα ήταν 16.4.4096B ή 32.4.4096B .

Swaptions



Ίδια ανάλυση με fulidanimate. Μια καλή επιλογή είναι 32.4.4096B .

Συμπεράσματα



Αρχικά, παρατηρούμε ότι οι μεταβολές των χαρακτηριστικών της TLB μας δίνουν μεγαλύτερο εύρος αποδόσεων από ότι οι cache μνημες L1 και L2, TLB: 0.25-0.45 L2: 0.38-0.46 και L1: 0.42-0.43 . Οπότε συμπεραίνουμε ότι η TLB επηρεάζει περισσότερο την απόδοση από τις L1 και L2 .

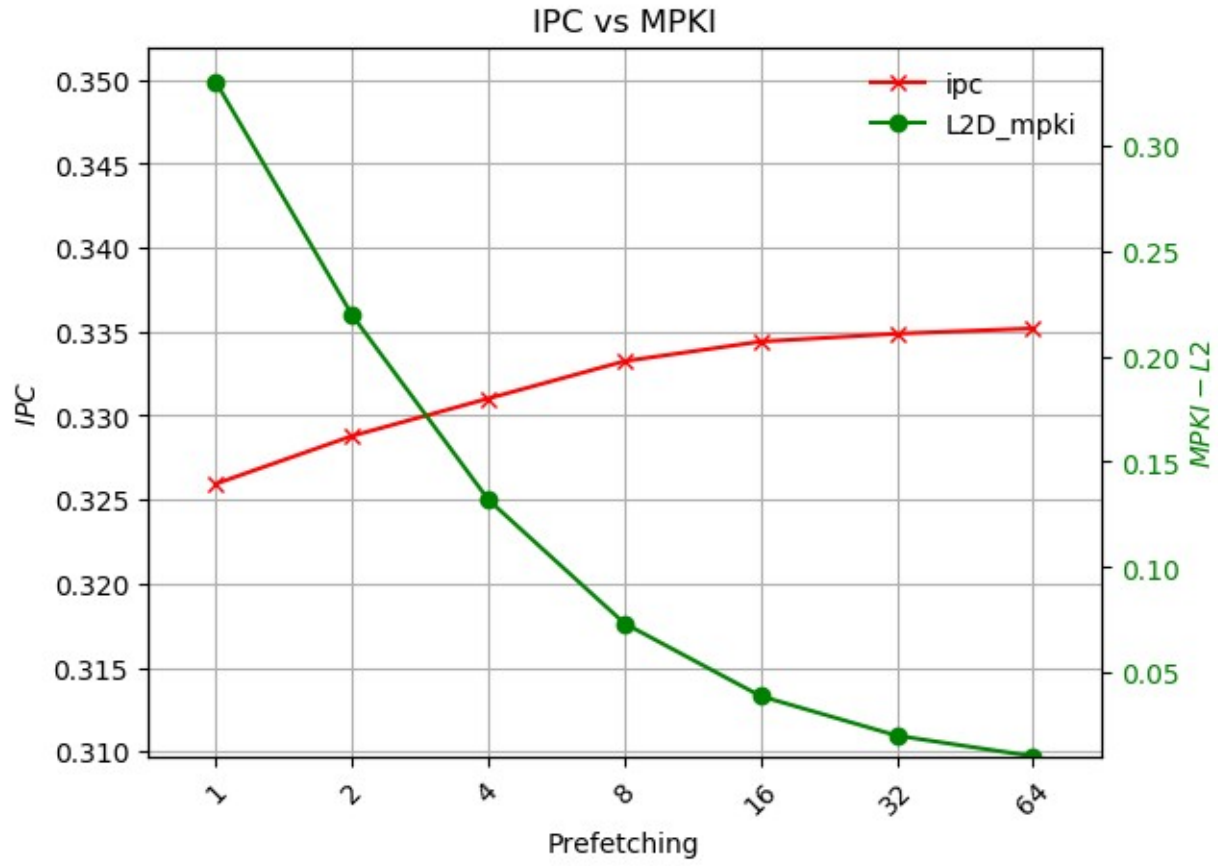
Από την γραφική παρατηρούμε ότι αύξηση του tlb-size οδηγεί σε σημαντική βελτίωση της απόδοσης , πχ 8.4.4096B->16.4.4096B->32.4.4096B . Ωστόσο, για να έχουμε καλύτερα αποτελέσματα από την αύξηση του tlb-size θα πρέπει ο βαθμός του associativity να μεγαλώνει επίσης, αλλιώς η απόδοση δεν διαφοροποιείται αρκετά , πχ 32.4.4096B-> 64.4.4096B->128.4.4096B->256.4.4096B . Ακόμα, η TLB δεν θα πρέπει direct-map γιατί η απόδοση μειώνεται δραστικά . Τέλος, σύμφωνα με το διάγραμμα βαθμός του associativity καταφέρνει να επωφελήσει την μνήμη μέχρι και την τιμή 8, μετά δεν παρατηρείται διαφορά .

Φαίνεται η γραφική παράσταση να συμφωνεί με την ανάλυση που κάναμε για τα περισσότερα benchmark. Η καλύτερη επιλογή από πλευράς απόδοσης, κόστους και ταχύτητας (associativity) είναι 64.8.4096B .

- **Prefetching**

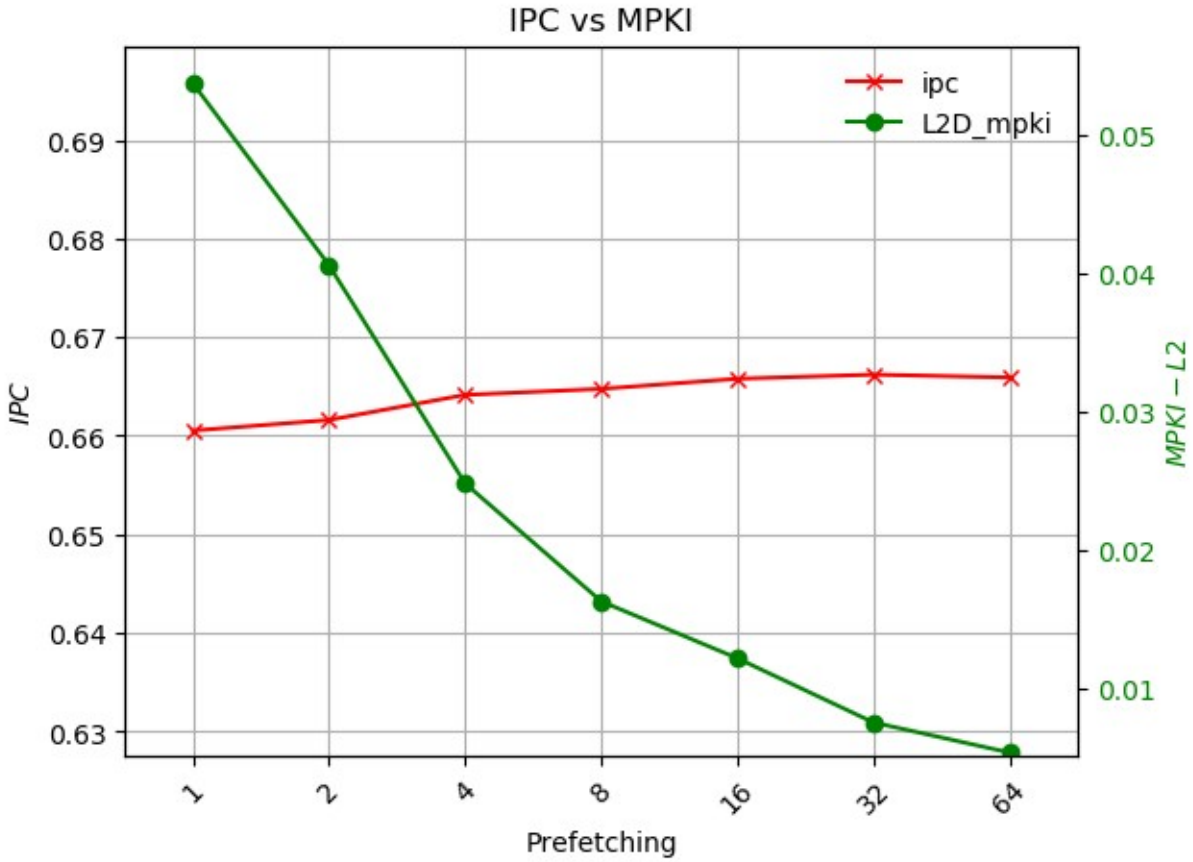
Σκοπός αυτού του ερωτήματος είναι να παρατηρήσουμε τις μεταβολές στην απόδοση ενεργοποιώντας το prefetcing στην L2 cache και κρατώντας τα χαρακτηριστικά των L1, L2 και TLB σταθερά. Πίο συγκεκριμένα, για κάθε L2 miss θα πρέπει να φορτωθεί πέρα από το ζητούμενο block και τα n επόμενα . Ακολουθούν γραφικές παραστάσεις για κάθε benchmark, για τιμές n=1,2,4,8,16,32,64 και η με μετρικές: ipc και L2_mpki .

Blacksholes



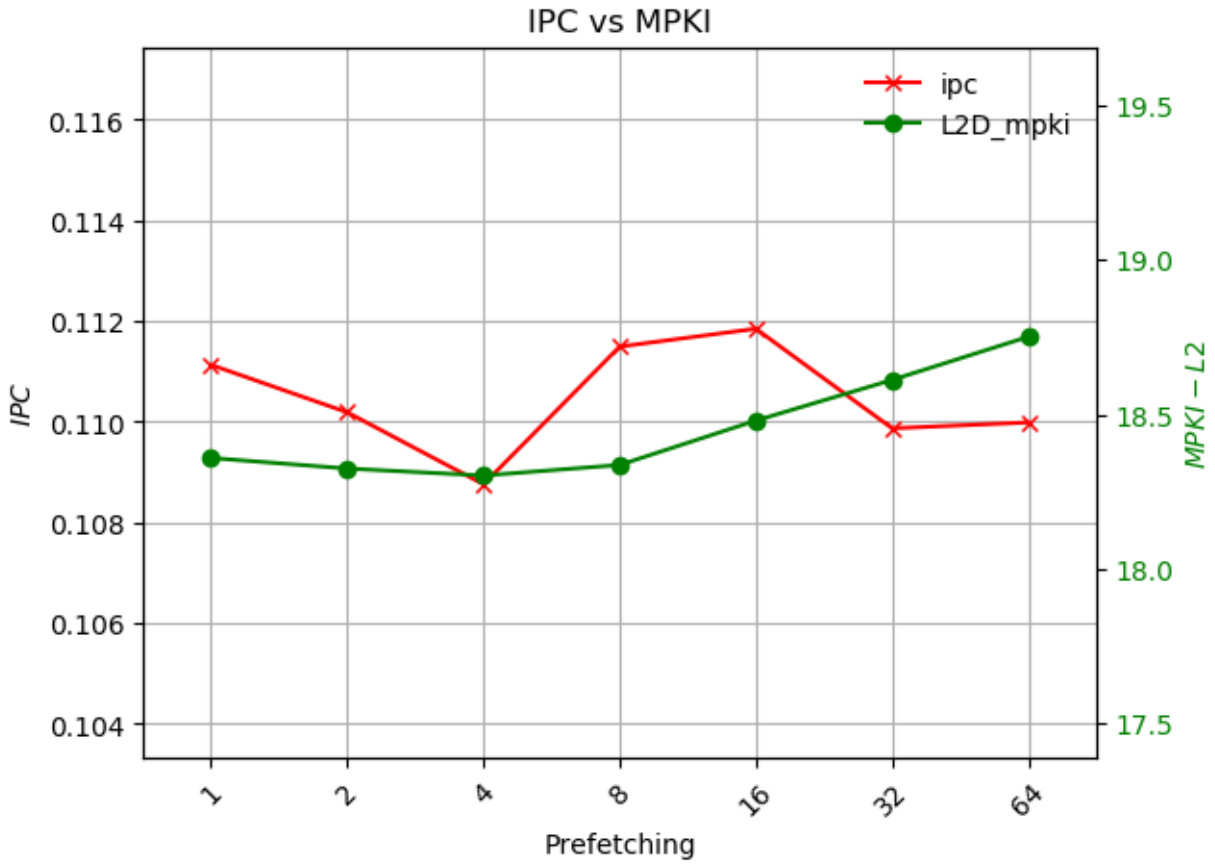
Η ενεργοποίηση του prefetcing στην L2 μειώνει σταθερά το miss-rate της . Μειώνονται τα compulsory-misses καθώς επωφελούνται αρκετά από την spacial-locality των block . Η απόδοση παρουσιάζει μικρή βελτίωση και φαίνεται να σταθεροποιείται για n>=32 .
Οπότε συμπεραίνουμε ότι το prefetcing επωφελεί ελάχιστα την απόδοση.

Bodytrack



Η ενεργοποίση του prefetching προκαλεί την πτώση του miss-rate καθώς έχουμε μείωση των compulsory-misses . Δεν φαίνεται όμως να έχει ιδιαίτερη επίδραση στην απόδοση της εφαρμογής.
Το miss-rate ήταν ήδη εξαιρετικά μικρό (0.05) και η απόδοση παράμενε σταθερή, οπότε δεν έχει νόημα η υλοποίηση prefetcing για την συγκεκριμένη εφαρμογή.

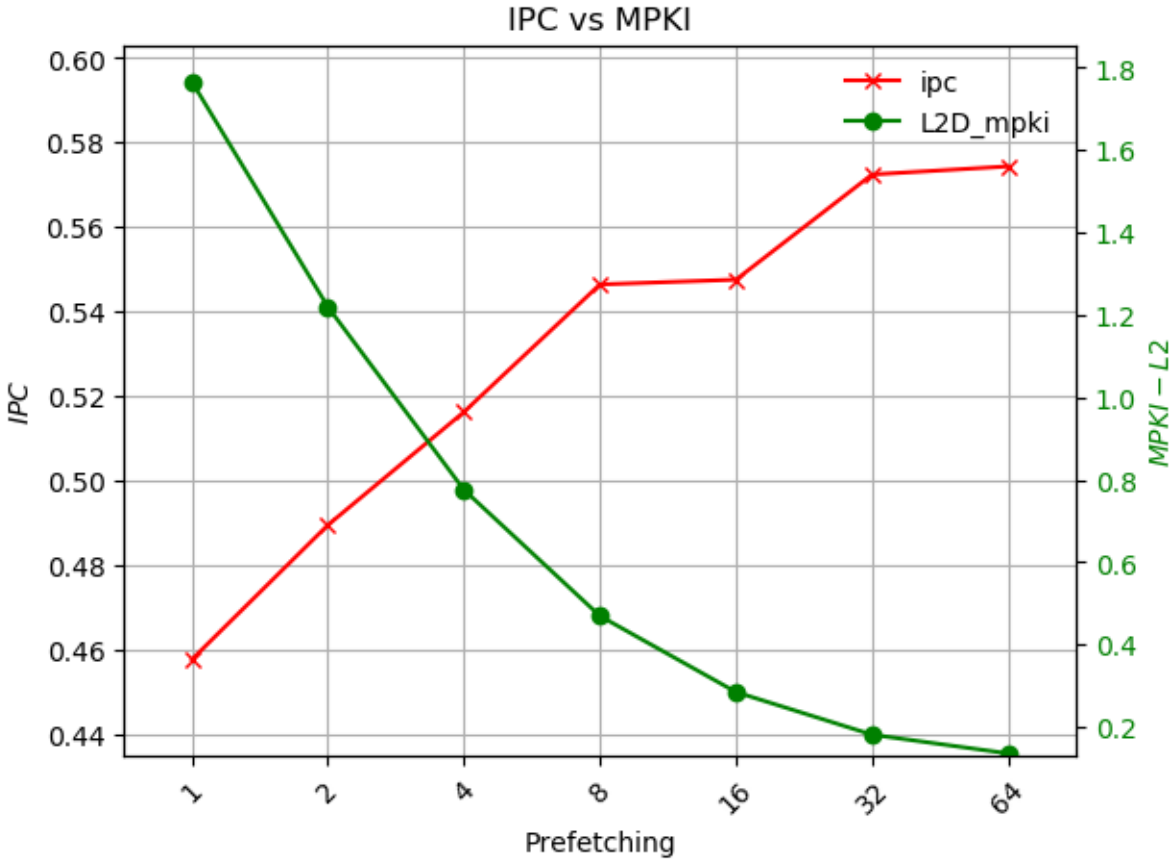
Canneal



Η ενεργοποίση του prefetcing προκαλεί την αύξηση του miss-rate. Ένας πιθανός λόγος είναι ένα κομμάτι των block που κάνουμε prefetch σε κάθε miss, αντικαθιστά χρήσιμα blocks αυξάνοντας έτσι τα capacity ή conflict misses . Παρατηρούμε μια μικρή ταλάντωση για την απόδοση, ωστόσο οι τιμές διαφέρουν στα τελευταία δεκαδικά ψηφία οπότε μπορούμε να την θεωρήσουμε σταθερή .

Άρα η υλοποίηση prefetching δεν βελτιώνει την απόδοση της εφαρμογής.

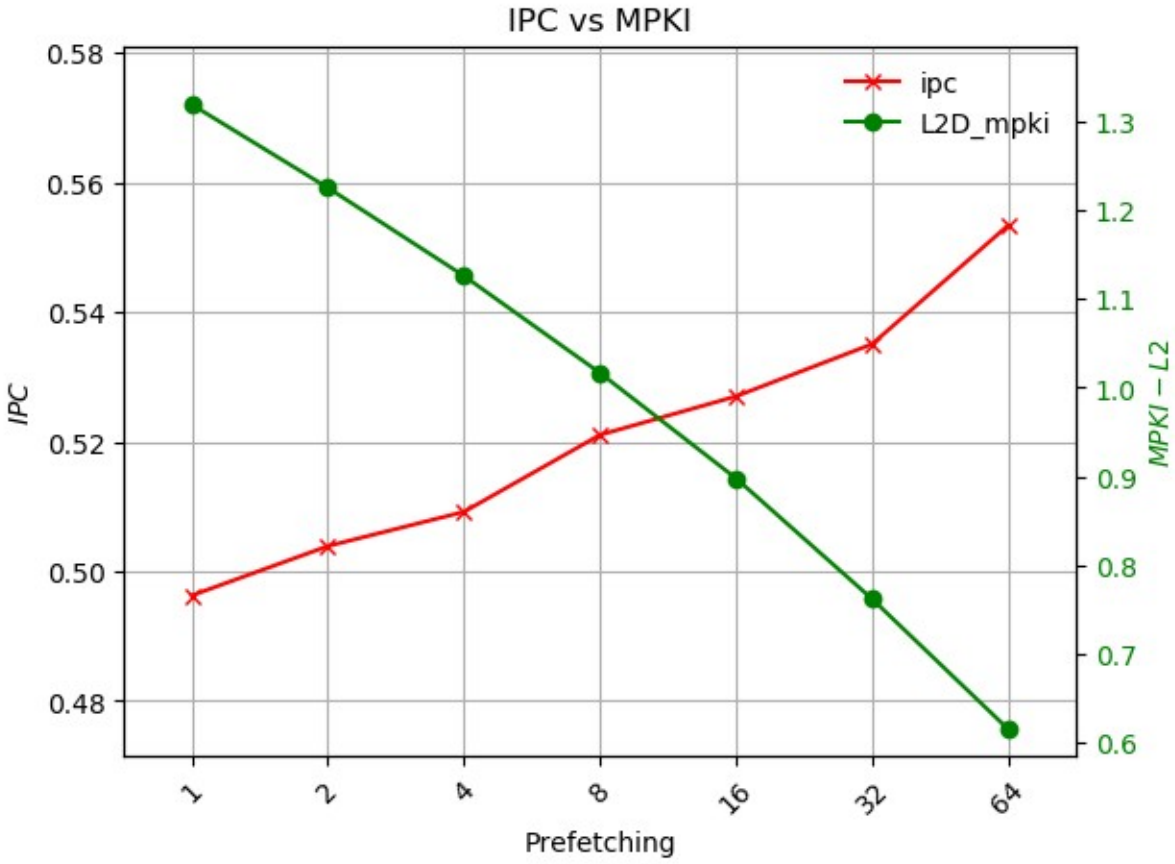
Facesim



Η ενεργοποίση του prefetcing προκαλεί πτώση του miss-rate , εξαλείφοντας τα compulsory-misses . Αντίθετα, παρατηρούμε σημαντική αύξηση της απόδοσης . Όποτε συμπεραίνουμε ότι η εφαρμογή επωφελείται από το prefetcing .

Την βέλτιστη απόδοση την έχουμε για τις τιμές 32 και 64, μπορούμε να επιλέξουμε οποιαδήποτε από τις δύο .

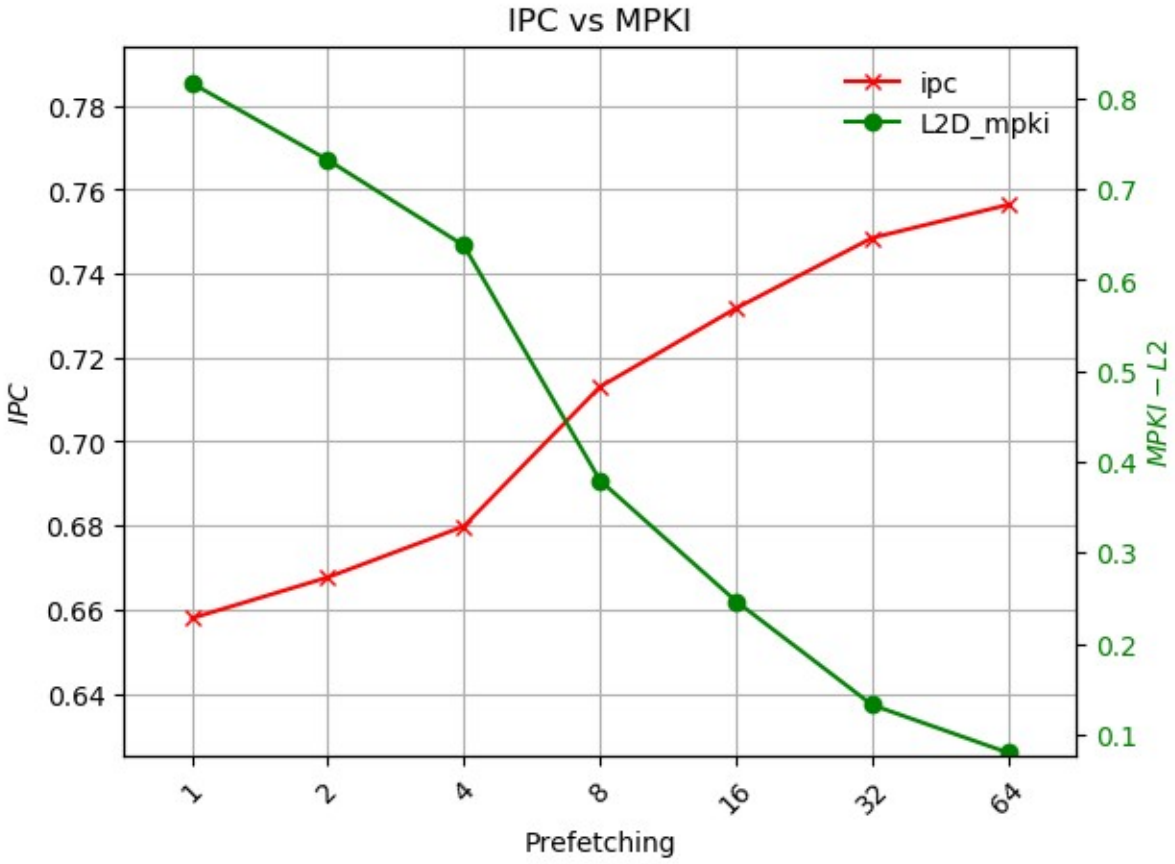
Ferret



Ομοίως με προηγουμένως έχουμε αύξηση της απόδοσης και μείωση του L2 miss-rate . Όμως η επίδραση φαίνεται να είναι μικρότερη .

Η καλύτερη επιλογή είναι 64 block prefetcing .

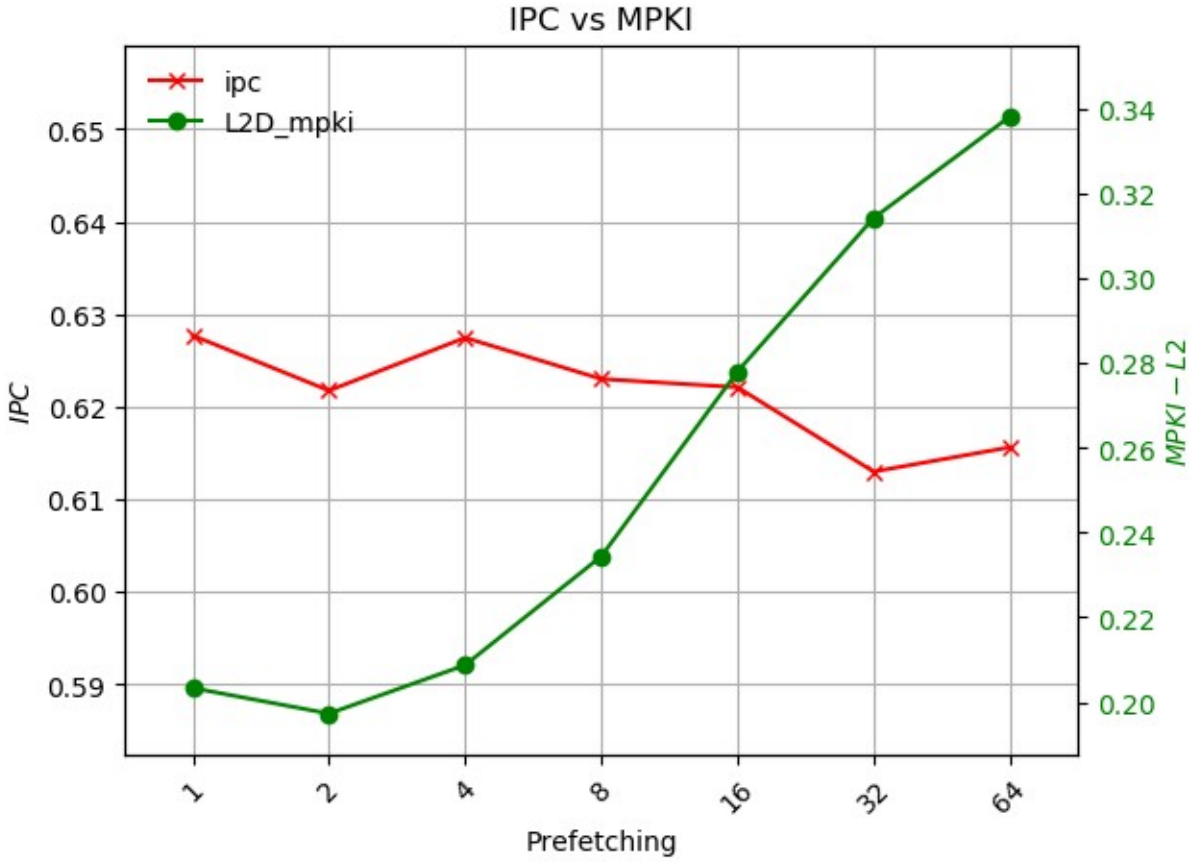
Fluidanimate



Η ενεργοποίση του prefetcing προκαλεί σημαντική πτώση του miss-rate , εξαλείφοντας τα compulsory-misses . Αντίθετα, παρατηρούμε αύξηση της απόδοσης . Όποτε συμπεραίνουμε ότι η εφαρμογή επωφελείται από το prefetcing .

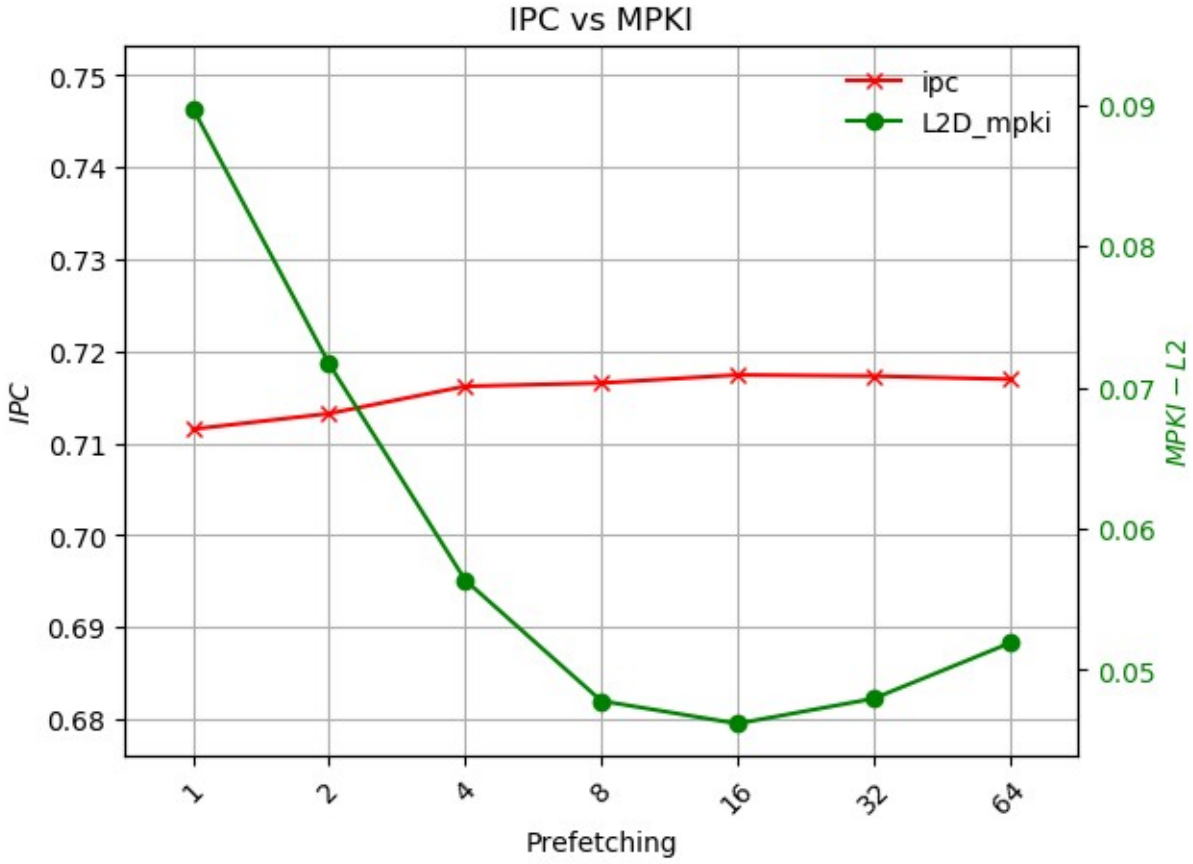
Η καλύτερη επιλογή είναι 64 block prefetcing .

Freqmine



Η ενεργοποίηση του prefetcing στην L2 αυξάνει το miss-rate της . Ένας πιθανός λόγος είναι ότι ένα κομμάτι των block που κάνουμε prefetch σε κάθε miss, αντικαθιστά χρήσιμα blocks αυξάνοντας έτσι τα capacity ή conflict misses . Ακόμα, παρατηρούμε για ορισμένες τιμές αυξομοιώση της απόδοσης, λόγω αυτού δε μπορούμε με βεβαιότητα να συμπεράνουμε την επίδραση του perfetcing. Ωστόσο κατά κύριο λόγο φαίνεται να έχουμε σταθερή απόδοση γύρω από την τιμή 0.62 .

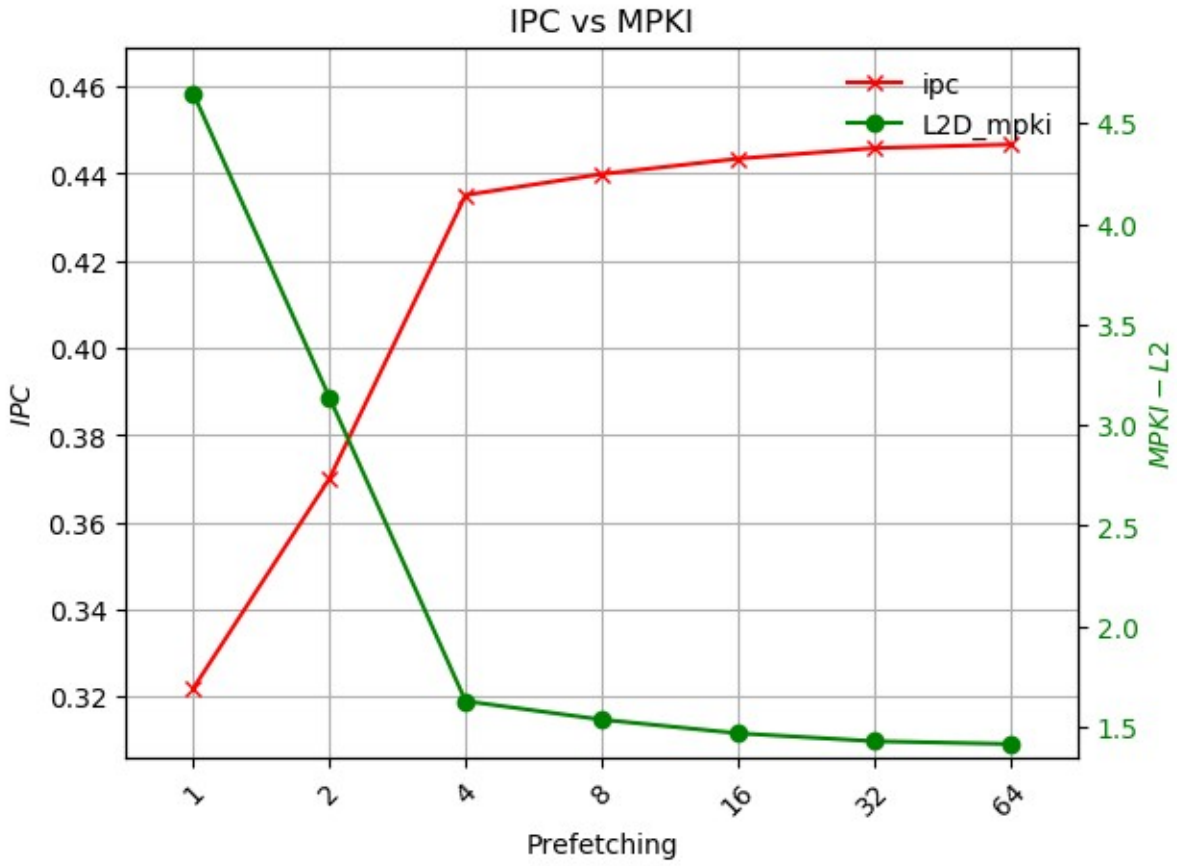
Raytrace



Η ενεργοποίηση του prefetcing προκαλεί μείωση του miss-rate μέχρι την τιμή 16, μετά παρατηρούμε μικρή άνοδο. Ένας πιθανός λόγος για την αύξηση είναι τα conflict (ή capacity) misses που οφείλονται στον αυξημένο αριθμό block που κάνουμε prefetch. Τα capacity είναι πιο απίθανο, λόγο του μικρού miss-rate που έχουμε (max ~ 0,01) . Η απόδοση φαίνεται να παραμένει σταθερή .

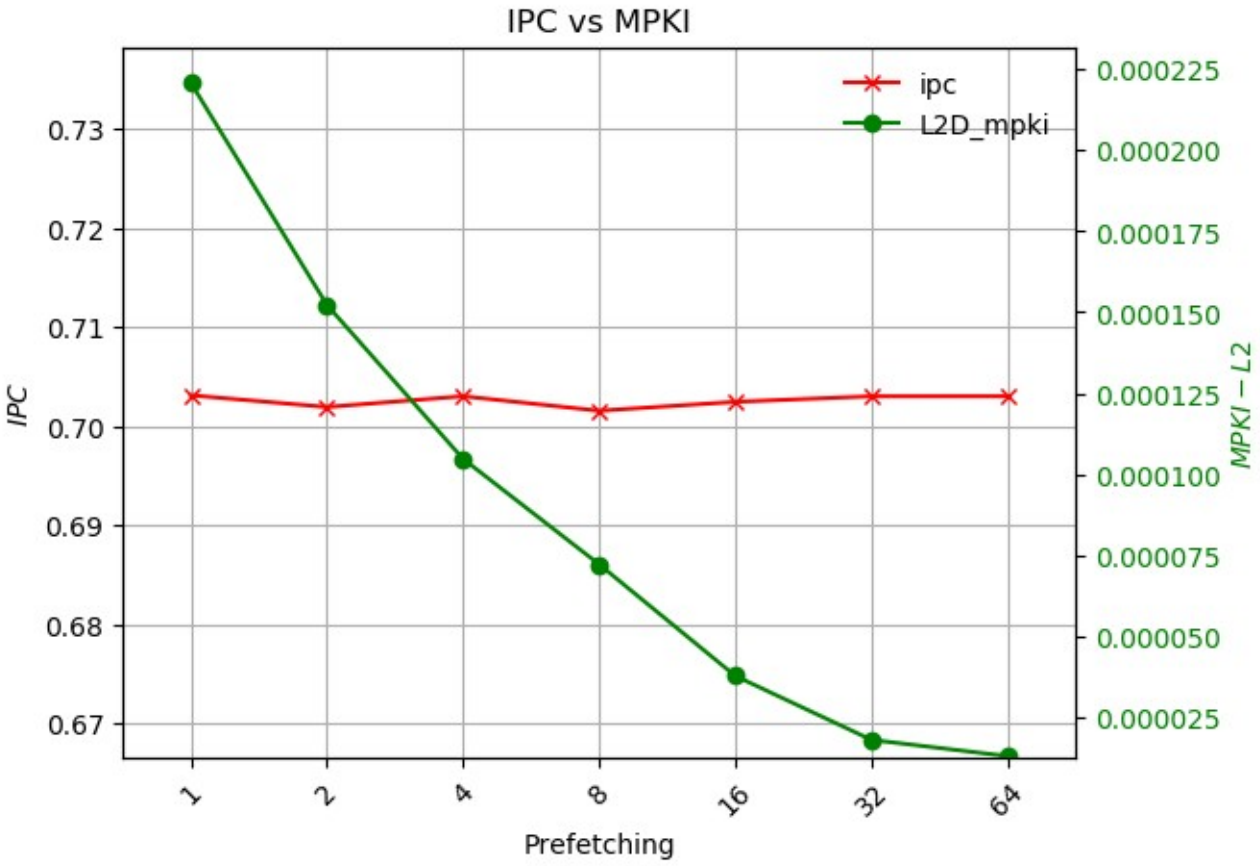
Λόγω της μικρής διαφοράς στο miss-rate (max = 0.04) , που μπορούμε να την θεωρήσουμε αμελητέα και της σταθερής απόδοσης, συμπεραίνουμε ότι το prefetcing δεν επηρεάζει την εφαρμογή .

Streamcluster



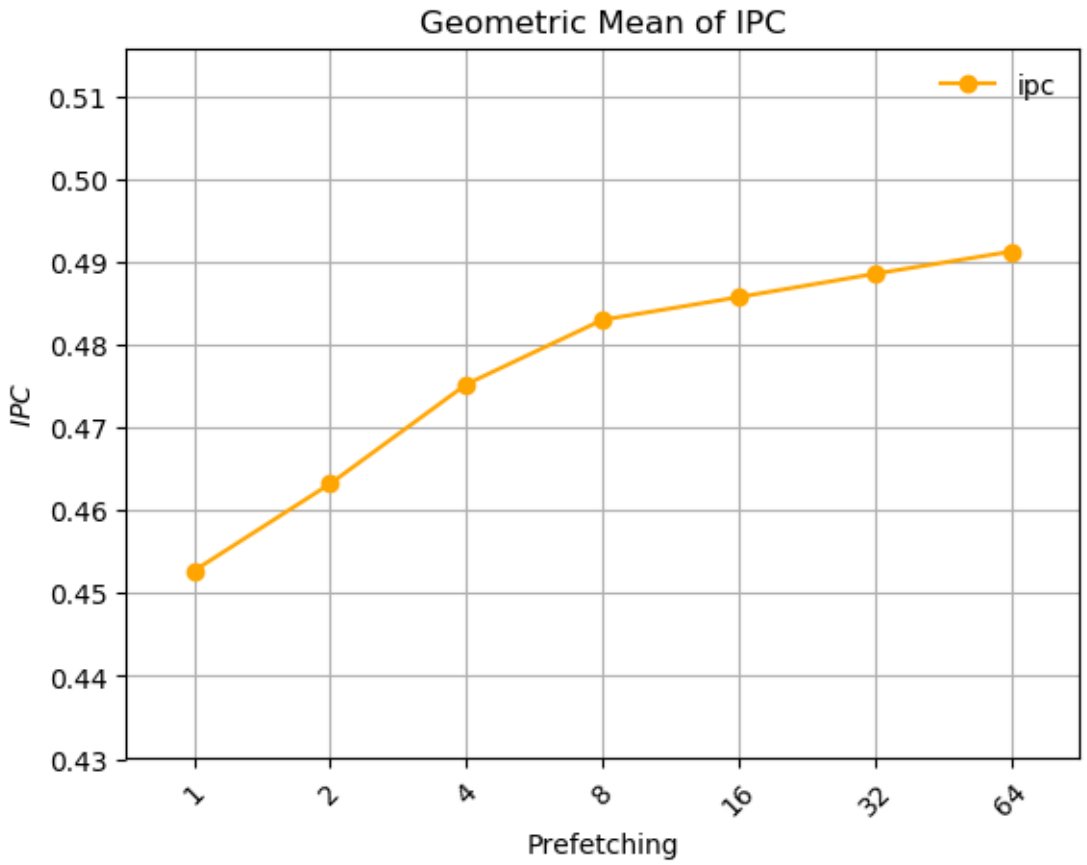
Παρατηρούμε ότι έχουμε ταυτόχρονη άνοδο της απόδοσης και πτώση του miss-rate μέχρι την τιμή 4 . Η αιτία είναι η σημαντική μείωση των compulsory misses στην L2 . Οι δύο μετρικές σχεδόν σταθεροποιούνται για n=>4 . Οπότε συμπεραίνουμε ότι το perfetcing ενισχύει την απόδοση και μια καλή επιλογή είναι οποιαδήποτε μεγαλύτερη ή ίση του 4 block prefetcing .

Swaptions



Η ενεργοποίση του prefetching προκαλεί την πτώση του miss-rate που οφείλεται στην μείωση των compulsory-misses . Δεν φαίνεται όμως να έχει ιδιαίτερη επίδραση στην απόδοση της εφαρμογής. Το miss-rate ήταν ήδη εξαιρετικά μικρό,σχεδόν μηδενικό και η απόδοση παράμενε σταθερή, οπότε δεν έχει νόημα η υλοποίηση prefetcing για την συγκεκριμένη εφαρμογή.

Συμπεράσματα



Η παραπάνω γραφική παράσταση αποτελεί τον γεωμετρικό μέσο της απόδοσης που είχαμε για όλα τα benchmark . Παρατηρούμε αύξηση της απόδοσης για όλες τις τιμές, μπορούμε να πούμε ότι είναι σχεδόν γραμμική αλλά με μικρή κλίση . Τα αποτελέσματα που πήραμε συμφωνεί με την ανάλυση που κάναμε για τις εφαρμογές, όπου στην περισσότερες αυξανόταν ή παράμενε σταθερή απόδοση.

Οπότε καταλήγουμε στο συμπέρασμα ότι το prefetcing βελτιώνει την συνολική απόδοση του συστήματος και παίρνουμε τα καλύτερα αποτελέσματα για 64 block-prefetcing .

- Μεταβολή κύκλου

Στις προηγούμενες αναλύσεις θεωρούσαμε ότι οι μεταβολές στα στοιχεία μνήμης δεν επηρεάζουν τον κύκλο ρολογιού, στην πράξη όμως αυτό δεν συμβαίνει . Στο συγκεκριμένο ερώτημα θα εξετάσουμε πως αλλάζει η απόδοση του συστήματος από την μεταβολή του κύκλου.

Ειδικότερα, θεωρούμε το πρώτο αποτέλεσμα κάθε προσομοίωσης ως αναφορά για το ipc και για κάθε διπλασιασμό του μεγέθους της μνήμης (cache ή tlb size) ή του associativity θα έχουμε 10% ή 5% αύξηση του κύκλου, αντίστοιχα .

Έστω c_{base} ο αρχικός κύκλος και $c_{adjust\text{-}i}$ ο αυξημένος κύκλος για την i-οστή επανάληψη της προσομοίωσης, με $c_{adjust-i} = x * c_{base}, x > 1$. Οι δύο κύκλοι έχουν διαφορετική διάρκεια, ωστόσο το πλήθος κύκλων που χρειάζεται για να ολοκληρωθεί μια εντολή παραμένει σταθερό .

Όπως γνωρίζουμε $IPC = \frac{Instructions}{\#Cycles}$, οπότε ισχύει

$$IPC = \frac{Instructions}{\#Cycles_{adjust-i}} = \frac{Instructions}{\#Cycles_{base}} = \frac{Instructions}{\#Cycles_i} \quad .$$

Άρα το ipc, σαν μέγεθος, για κάθε υλοποίηση δεν επηρεάζεται από την αλλαγή της διάρκειας του κύκλου . Όμως, αν μετράμε το πλήθος κύκλων σαν χρονική διάρκεια, τότε για το ίδιο διάστημα θα έχουμε εκτελέσει περισσότερους κύκλους c_{base} από $c_{adjust\text{-}i}$. Πιο συγκεκριμένα, θα ισχύει $\#Cycles_{adjust-i} = x * \#Cycles_{base}$, δηλαδή στον χρόνο εκτέλεσης της εφαρμογής με κύκλο $c_{adjust\text{-}i}$ θα είχαμε εκτελέσει $x * \#Cycles_{base}$ κύκλους .

Οπότε, $IPC_{adjust-i} = \frac{Instructions}{x * \#Cycles_{base}} = \frac{1}{x} * IPC_{base-i}$. Στην ουσία αλλάζουμε τον ορισμό του ipc, ώστε (για τους διάφορους κύκλους $c_{adjust\text{-}i}$) να δείχνουμε πόσες εντολές μπορούμε να εκτελέσουμε στην διάρκεια ενός κύκλου c_{base} . Ο λόγος για αυτήν την αλλαγή είναι για να μπορούμε να συγκρίνουμε την μεταβολή στην απόδοση του συστήματος .

Για κάθε διπλασιασμό του μεγέθους της μνήμης θα έχουμε:

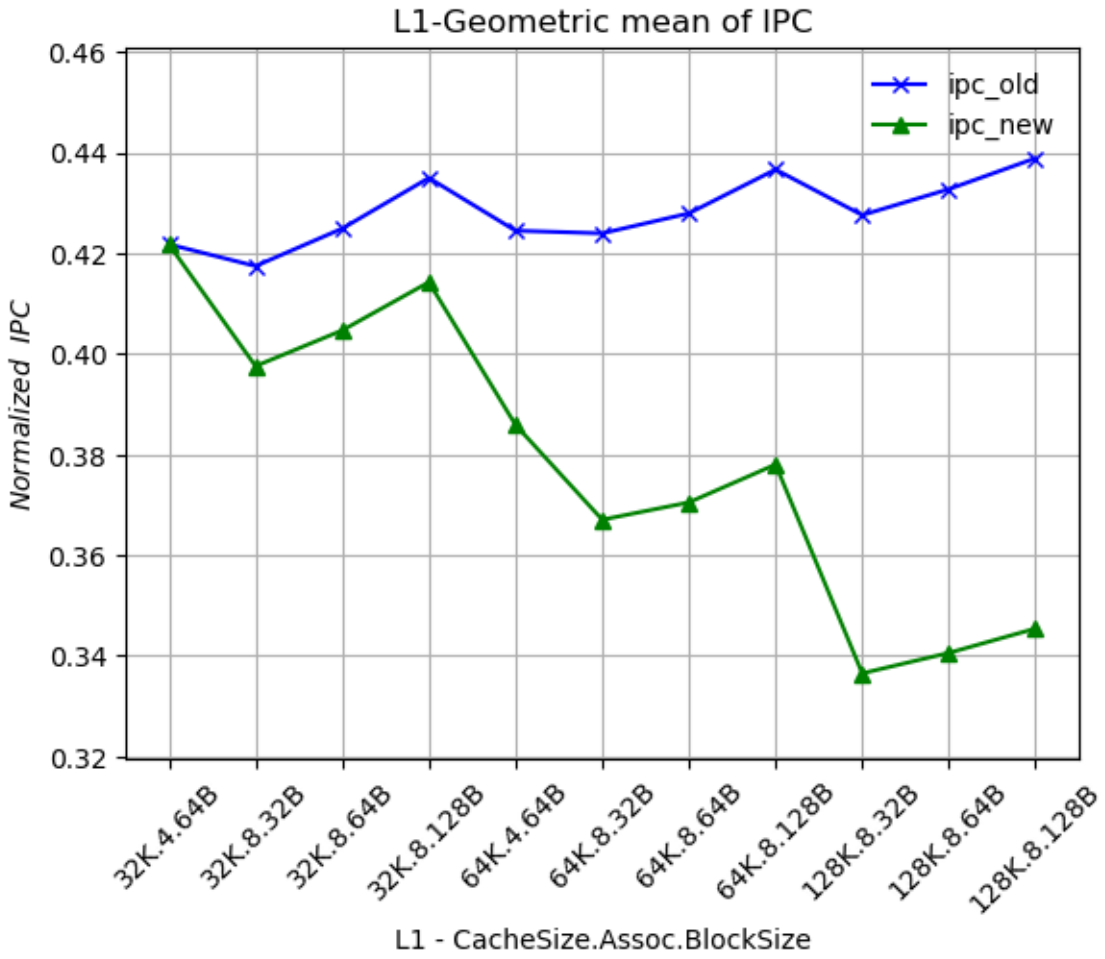
- 1η φορά $c_{adjust-i} = 1.1 * c_{base}$
- 2η φορά $c_{adjust-i} = 1.1 * 1.1 * c_{base} = 1.1^2 * c_{base}$
- n-οστή φορά $c_{adjust-i} = 1.1^n * c_{base}$

Ομοίως για κάθε διπλασιασμό του associativity θα έχουμε:

- 1η φορά $c_{adjust-i} = 1.05 * c_{base}$
- 2η φορά $c_{adjust-i} = 1.05 * 1.05 * c_{base} = 1.05^2 * c_{base}$
- n-οστή φορά $c_{adjust-i} = 1.05^n * c_{base}$

Οπότε ισχύει $c_{adjust-i} = 1.1^n * 1.05^m * c_{base}$, με n και m διπλασιασμό μεγέθους και associativity αντίστοιχα . Άρα $IPC_{adjust-i} = \frac{1}{1.1^n * 1.05^m} * IPC_{base-i}$

- L1 cache



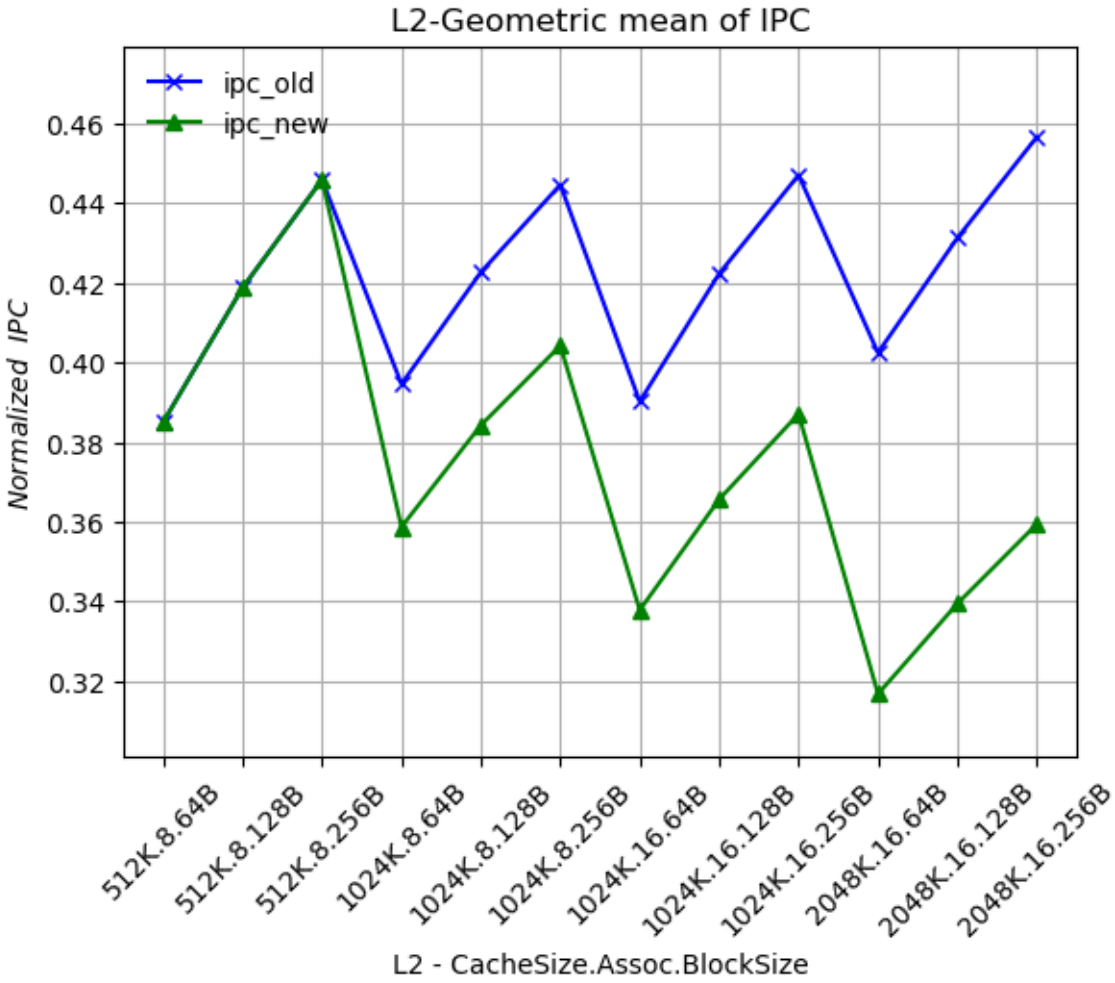
Όπως φαίνεται ξεκάθαρα από την γραφική παράσταση η αύξηση του κύκλου διαφοροποιεί σημαντικά την απόδοση. Αν και αρχικά, θεωρούσαμε ότι η αύξηση του cache-size και associativity επηρέαζε ελάχιστα την απόδοση, διαπιστώνουμε ότι την μειώνουν σε μεγαλύτερο βαθμό από ότι καταφέρνει να την αυξήσει το block-size,όπου είχε και το καθοριστικό ρόλο στην ανάλυση μας .

Παραδείγματα :

- 32K.4.64B -> 32K.8.128B : associativity ↑ , block-size ↑ => απόδοση ↓
- 32K.4.64B -> 64K.4.128B : cache-size ↑ , block-size ↑ => απόδοση ↓
- 32K.4.64B-> 64K.8.128B : associativity ↑ , cache-size ↑ , block-size ↑ => απόδοση ↓

Όποτε, συμπεραίνουμε ότι από πλευράς απόδοσης , δηλαδή χρόνου εκτέλεσης, η καλύτερη επιλογή είναι 32K.4.64B αντίθετα από την αρχική μας 32K.8.128B.

- L2 cache



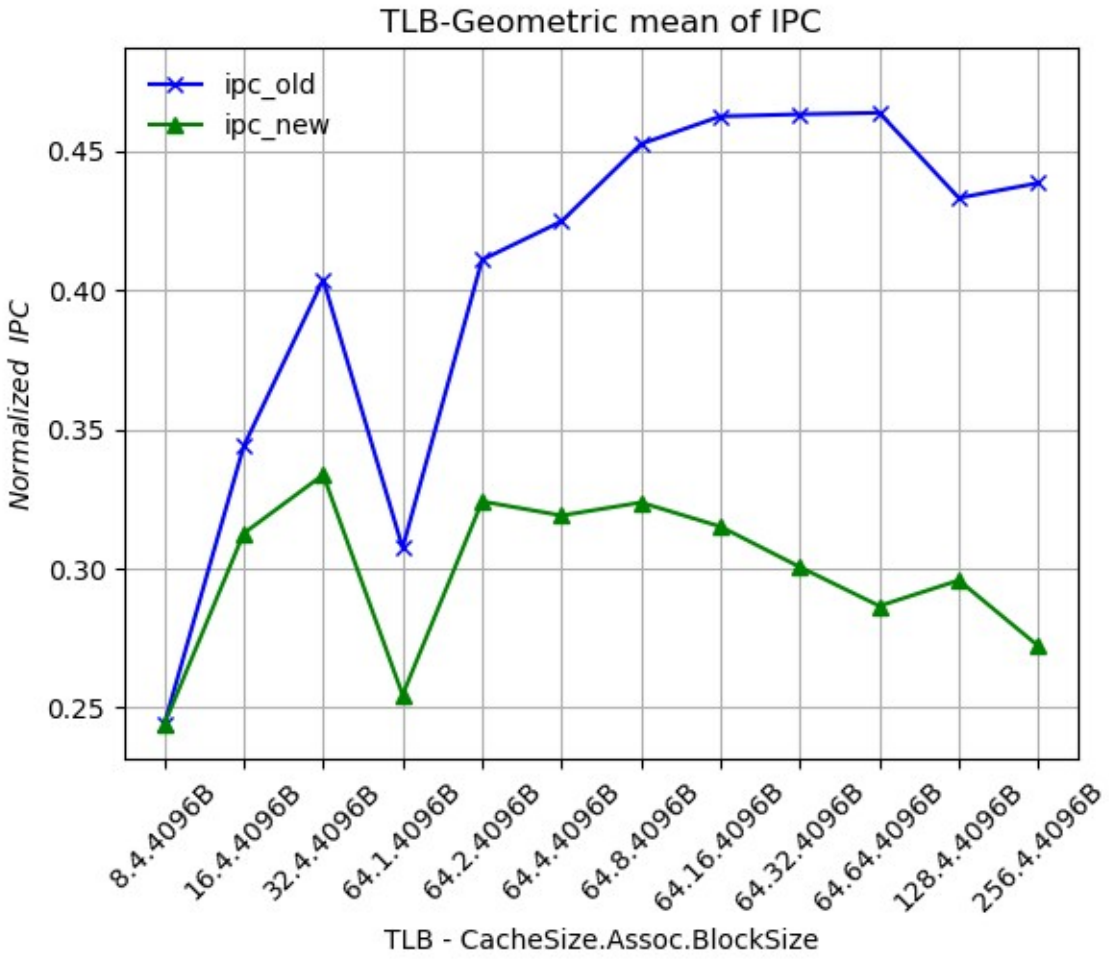
Ομοίως με την L1 παρατηρούμε ότι η αύξηση του κύκλου μειώνει σημαντικά την απόδοση . Πάλι φαίνεται ότι η αρνητική επίδραση των associativity και cache-size εξουδετερώνει την βελτίωση στην απόδοση που αναμέναμε από την αύξηση του block-size .

Παραδείγματα :

- 512K.8.64B -> 1024K.8.256B : cache-size ↑ , block-size ↑ => ελάχιστη αύξηση απόδοσης
- 1024K.8.256B -> 1024K.16.256B : associativity ↑ , block-size ↑ => απόδοση ↓
- 512K.8.64B -> 1024K.16.256B : associativity ↑ , cache-size ↑ , block-size ↑ => απόδοση σταθερη

Οπότε συμπεραίνουμε ότι με την αύξηση των associativity , cache-size και block-size καταφέρνουμε, στην καλύτερη περίπτωση, ελάχιστα μεγαλύτερή ή σταθερή απόδοση από την αρχική μας υλοποιήση (512K.8.64B) . Τέλος, παραμένει καλύτερη επιλογή η 512K.8.256B, όπου λαμβάνουμε τα οφέλη από την αύξηση του block-size , χωρίς τις αντίστοιχες συνέπειες των associativity και cache-size .

- TLB



Στην αρχική μου ανάλυση είχα διατυπώσει ότι τα tlb-size και associativity παίζουν καθοριστικό ρόλο στην απόδοση, πράγμα που επαληθεύεται εν μέρει και εδώ . Παρά την αρνητική επίδραση που έχει ο διπλασιασμός του μεγέθους,λόγω της αύξησης του κύκλου, διαπιστώνουμε ότι δεν είναι αρκετή για να αναιρέσει τα οφέλη από την μείωση των capacity misses . Ωστόσο, αυτό το παρατηρούμε μόνο για τις αρχικές αυξήσεις του tlb-size όπου η συνεισφορά του μεγέθους είναι πιο δραστική , μετά από τα 32 entries έχουμε μείωση της απόδοσης . Αντίθετα με την πρώτη ανάλυση, ο βαθμός του associativity φαίνεται να παίζει μικρό ρόλο, καθώς με την αύξηση του, στην καλύτερη περίπτωση , η απόδοση παραμένει σταθερή ή μειώνεται ελάχιστα. Εξαίρεση αποτελεί πάλι η περίπτωση της direct-map, όπου έχουμε μεγάλη πτώση .

Παραδείγματα :

- 8.4.4096B -> 16.4.4096B -> 32.4.4096B : tlb-size ↑ => απόδοση ↑
- 32. 4.4096B -> 64. 4.4096B -> 128. 4.4096B -> 256. 4.4096B : tlb-size ↑ => απόδοση ↓
- 64. 2.4096B -> 64. 8.4096B : associativity ↑ => απόδοση σταθερή
- 64. 8.4096B -> 64. 16.4096B -> 64. 32.4096B -> 64. 64.4096B : associativity ↑ => απόδοση ↓

Οπότε συμπεραίνουμε ότι μόνο η αύξηση του tlb-size καταφέρνει να βελτιώσει την απόδοση αλλά μέχρι ένα συγκεκριμένο μέγεθος . Η καλύτερη επιλογή, απο πλευράς απόδοσης και κόστους, αλλάζει και είναι 32.4.4096B .