ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ
1η ΆΣΚΗΣΗ

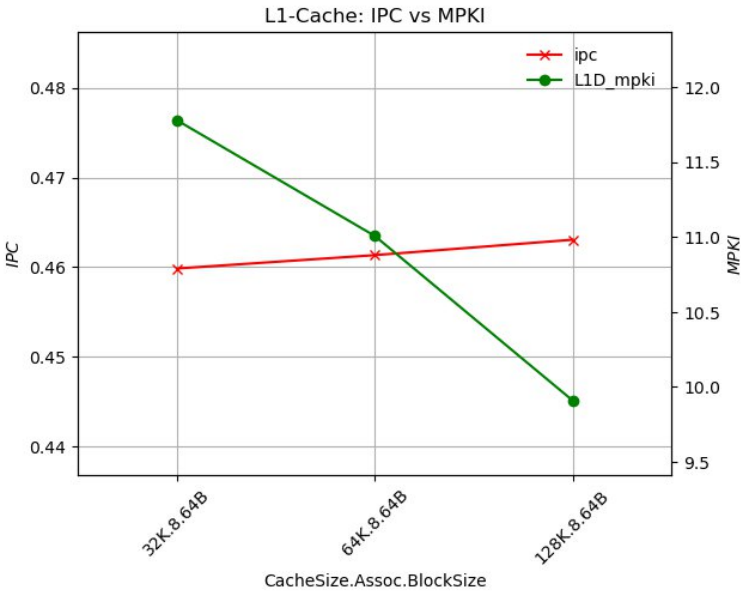
Εμμανουήλ Παντελάκης, 20853

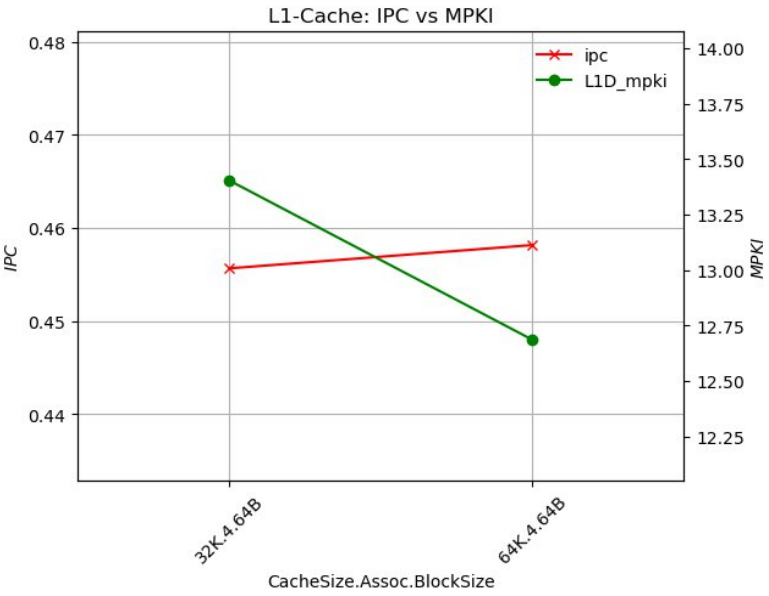
Ζητούμενο 1.1

Για τις διαφορετικές παραμετροποιήσεις της L1 cache θα παρουσιαστούν σε κοινές γραφικές παραστάσεις ο αρμονικός μέσος όρος των τιμών του **IPC** (Instructions Per Cycle) και ο αριθμητικός μέσος όρος των τιμών του **MPKI** (Misses Per 1000-Instructions) που προέκυψαν από τα 8 benchmarks που χρησιμοποιήθηκαν. Ο υπολογισμός των μέσων όρων έγινε με την δημιουργία και χρήση bash scripts, ο κώδικας των οποίων παρατίθεται στο *Παράρτημα Κώδικα* στο τέλος της αναφοράς. Για την εκτέλεση των benchmarks και την δημιουργία των γραφικών παραστάσεων χρησιμοποιήθηκαν τα bash scripts που μας δώθηκαν με μικρές αλλαγές όπου χρειαζόταν κάθε φορά. Σκοπός είναι να μελετήσουμε την επίδραση της μεταβολής συγκεκριμένων παραμέτρων της cache (***Cache Size***, ***Associativity***, ***Block Size***) στην επίδοση των κρυφών μνημών L1 και L2 μέσω των δεικτών IPC, MPKI.

1) L1 Cache

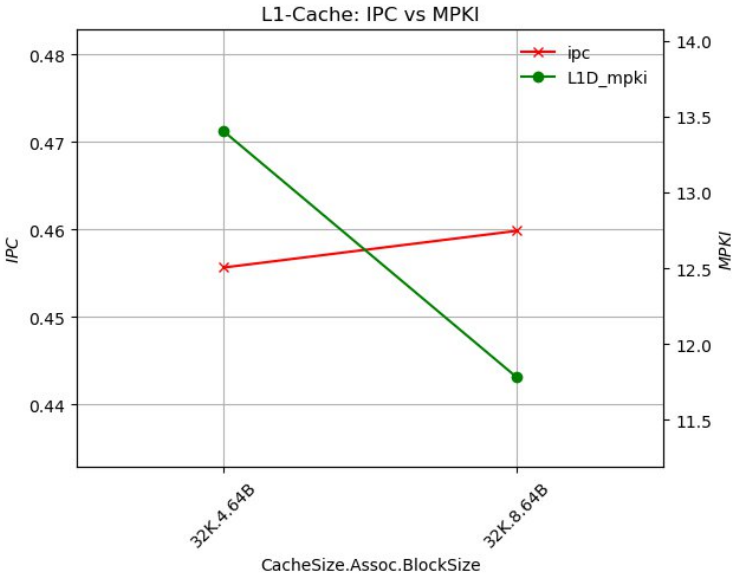
i. Αύξηση του Cache Size

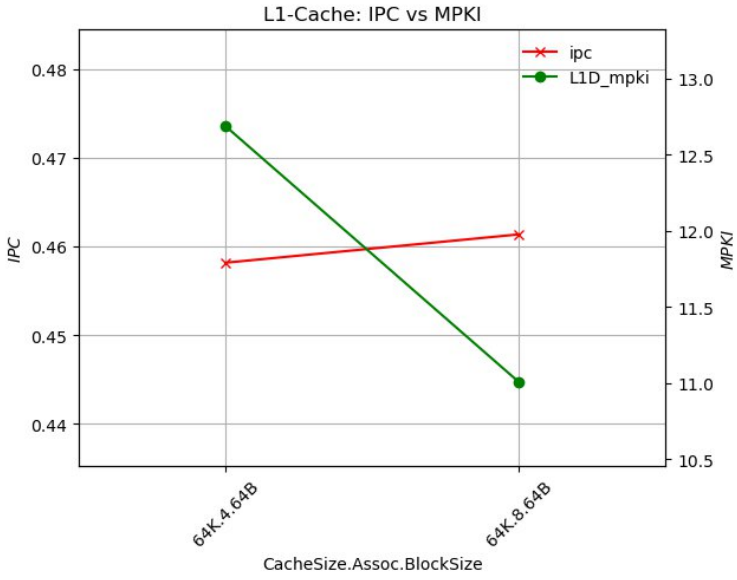




Παρατηρούμε ότι η αύξηση της χωρητικότητας της κρυφής μνήμης οδηγεί σε αύξηση του IPC και μείωση MPKI, λόγω της μείωσης των αστοχιών χωρητικότητας.

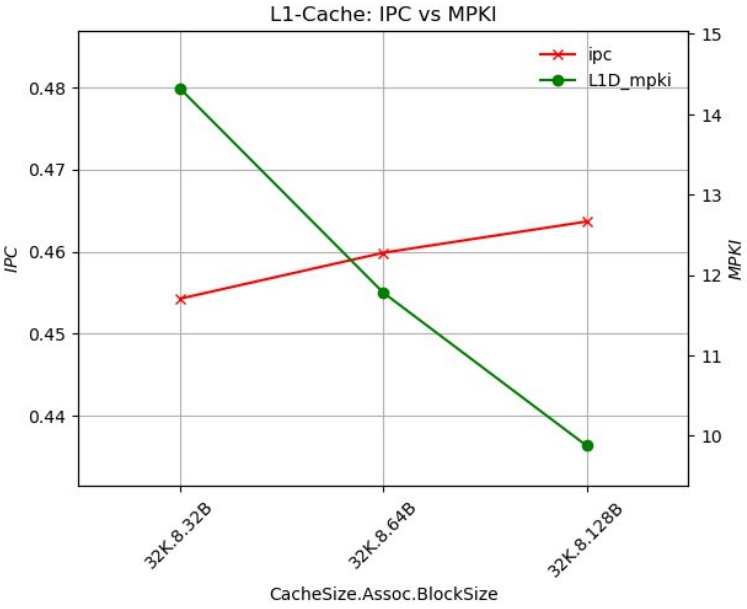
ii. Αύξηση της συσχετιστικότητας

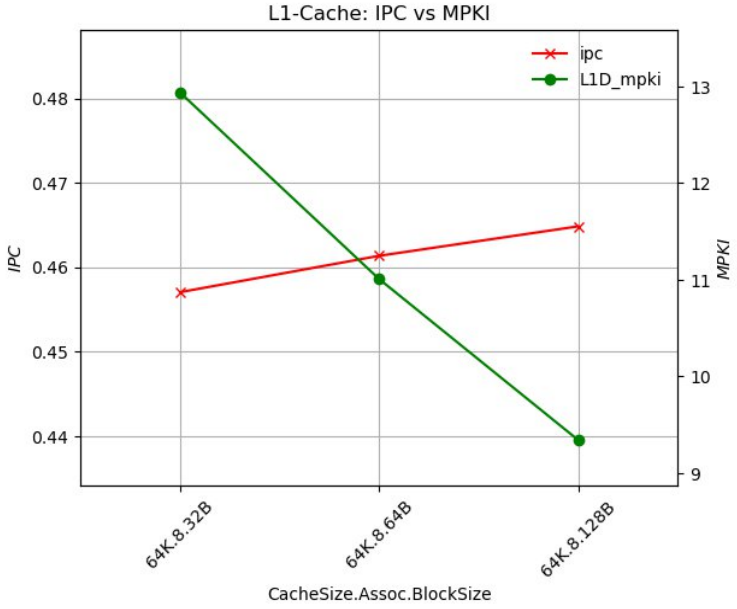


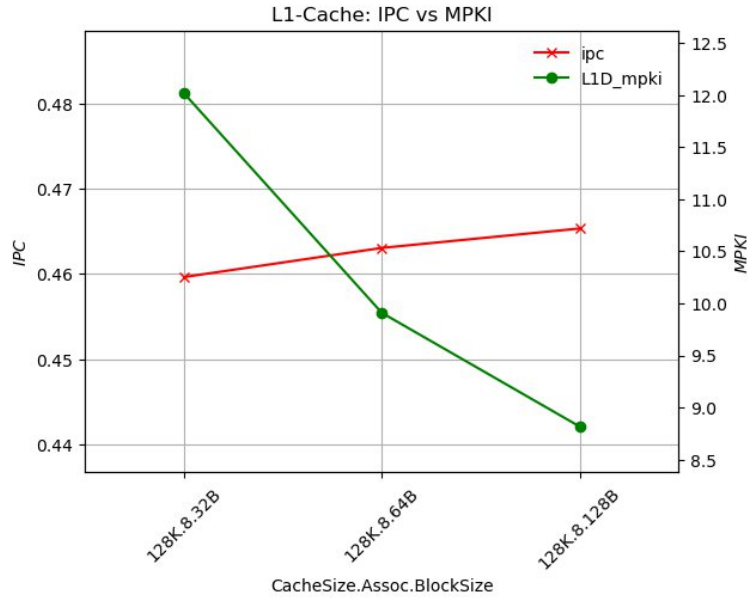


Παρατηρούμε ότι η αύξηση της συσχετιστικότητας οδηγεί σε αύξηση του IPC και μείωση του MPKI. Αυτό εξηγείται από το γεγονός ότι η μεγαλύτερη συσχετιστικότητα μειώνει την πιθανότητα να υπάρξουν αστοχίες διένεξης (conflict misses).

iii. Αύξηση του μεγέθους μπλοκ

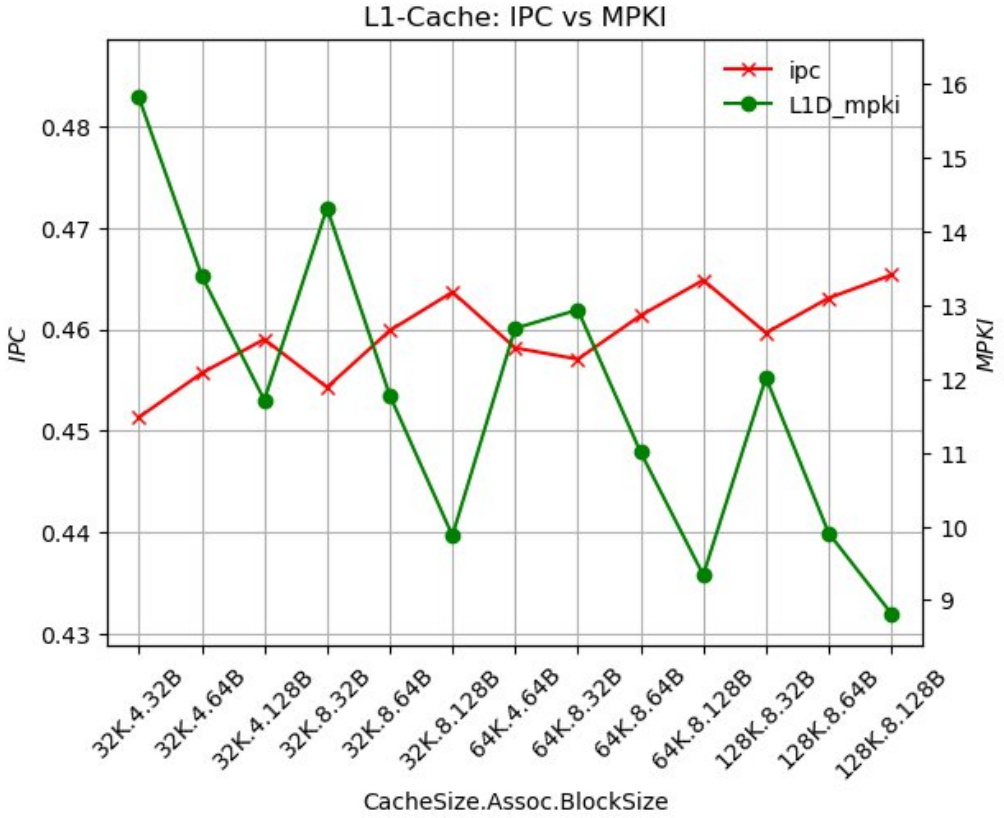






Παρατηρούμε ότι η αύξηση του μεγέθους του μπλοκ οδηγεί σε αύξηση του IPC και μείωση του MPKI, καθώς τα μεγαλύτερα μπλοκ εκμεταλλεύονται τη χωρική τοπικότητα μειώνοντας ταυτόχρονα και τις υποχρεωτικές αστοχίες.

Συνολική παρουσίαση όλων των μετρήσεων για την L1 cache σε κοινή γραφική παράσταση



Στην παραπάνω γραφική παράσταση διαφαίνονται οι παρατηρήσεις που διατυπώθηκαν στις προηγούμενες σελίδες, καθώς και ότι οι παραμετροποιήσεις που έχουν την μεγαλύτερη επίδραση στην επίδοση είναι κυρίως η αύξηση του block size και στη συνέχεια η αύξηση της συσχετιστικότητας. Η αύξηση του μεγέθους της μνήμης προκαλεί βελτίωση της απόδοσης αλλά σε μικρότερο βαθμό.

Ενδεικτικά κάποιες μεταβολές που επιβεβαιώνουν τις παρατηρήσεις μας (με πράσινο χρώμα επισημαίνονται οι μεταβολές μεγαλύτερου μεγέθους):

Cache Size (KB)	Associativity	Block Size (N)	Μεταβολή IPC	Μεταβολή MPKI
32→64	4	64	+0.002499	-0.717431
32	4	32→64	**+0.004356**	**-2.415877**
32	4→8	32	+0.002964	-1.499126
64	4→8	64	+0.00319	-1.678671
64→128	8	64	+0.001719	-1.098756
128	8	32→64	**+0.00343**	**-2.108311**

Για την εκτέλεση των benchmarks παραμετροποιώντας την κρυφή μνήμη L2, επιλέγουμε εκείνες τις τιμές των παραμέτρων της L1 που οδηγούν στον υψηλότερο μέσο όρο για το IPC:

```
Max Mean IPC: 0.465353 (Found in file: 128_8_128)
```

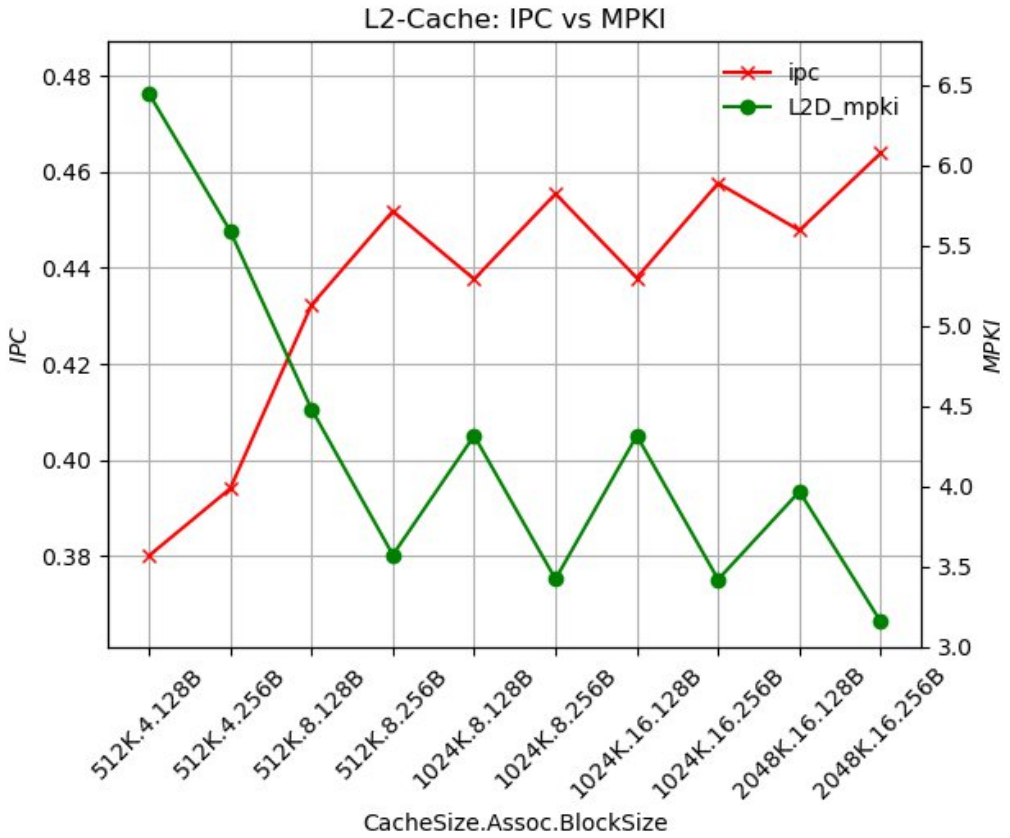
Επομένως επιλέγουμε για την L1:

- Cache Size: 128KB
- Associativity: 8
- Block Size: 128B

2) L2 Cache

Από την περίπτωση αυτή και για τις περιπτώσεις που θα μελετήσουμε στην συνέχεια δεν θα παρουσιάσουμε αναλυτικά τις μεταβολές για κάθε παράμετρο αλλά θα μελετήσουμε και θα σχολιάσουμε τις συνολικές γραφικές παραστάσεις.

Συνολική παρουσίαση όλων των μετρήσεων για την L2 cache σε κοινή γραφική παράσταση



Και στην περίπτωση της L2 cache παρατηρούμε παρόμοια συμπεριφορά στην επίδοση της μνήμης λόγω της αύξησης της χωρητικότητας, της συσχετιστικότητας και του μεγέθους του μπλοκ. Την μεγαλύτερη επίδραση έχουν και πάλι η αύξηση της συσχετιστικότητα και του μέγεθος του μπλοκ.

Ενδεικτικά κάποιες μεταβολές (με πράσινο χρώμα επισημαίνονται οι μεταβολές μεγαλύτερου μεγέθους):

Cache Size (KB)	Associativity	Block Size (N)	Μεταβολή IPC	Μεταβολή MPKI
512→1024	8	128	+0.005366	-0.717431
512	4	128→256	+0.013987	**-0.908062**
512	4→8	128	**+0.052219**	**-1.967216**
1024	8→16	128	+0.000272	-0.000037
1024	16	128→256	**+0.019715**	-0.895717
1024→2048	16	128	+0.009897	-0.347928

Για την εκτέλεση των benchmarks παραμετροποιώντας την κρυφή μνήμη TLB, επιλέγουμε εκείνες τις τιμές των παραμέτρων της L2 που οδηγούν στον υψηλότερο μέσο όρο για το IPC:

```
Max Mean IPC: 0.463929 (Found in file: 2048_16_256)
```

Επομένως επιλέγουμε για την L1:

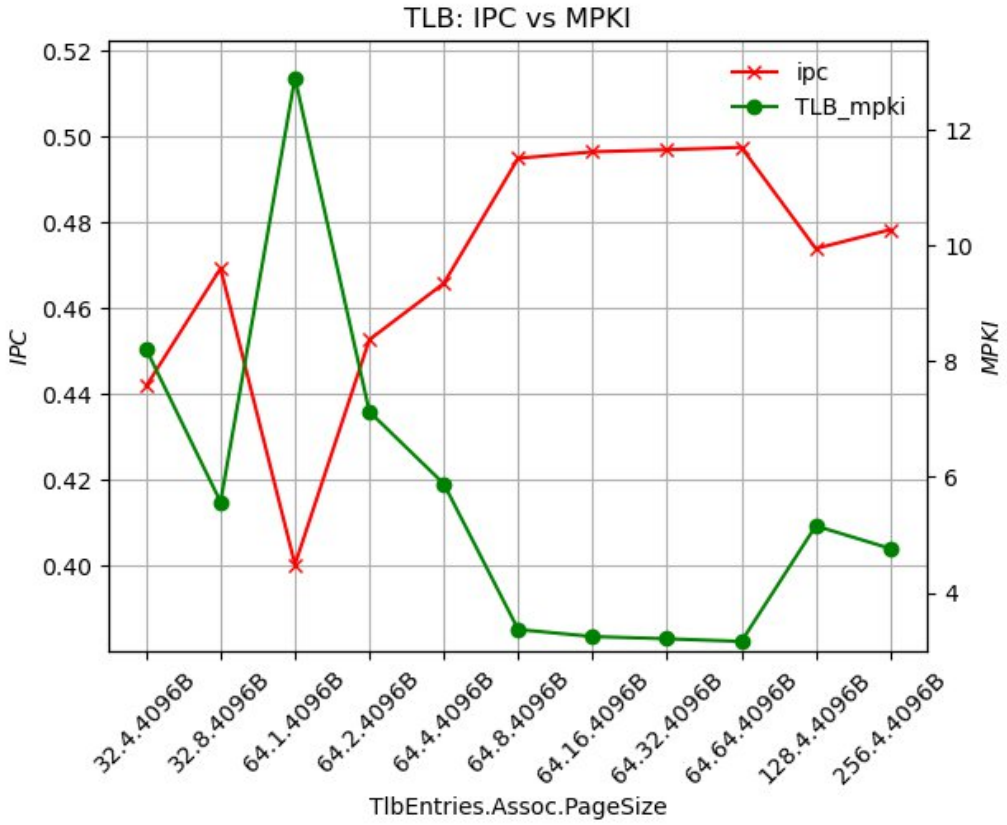
- Cache Size: 128KB
- Associativity: 8
- Block Size: 128B

Και για την L2:

- Cache Size: 2048KB
- Associativity: 16
- Block Size: 256B

3) TLB

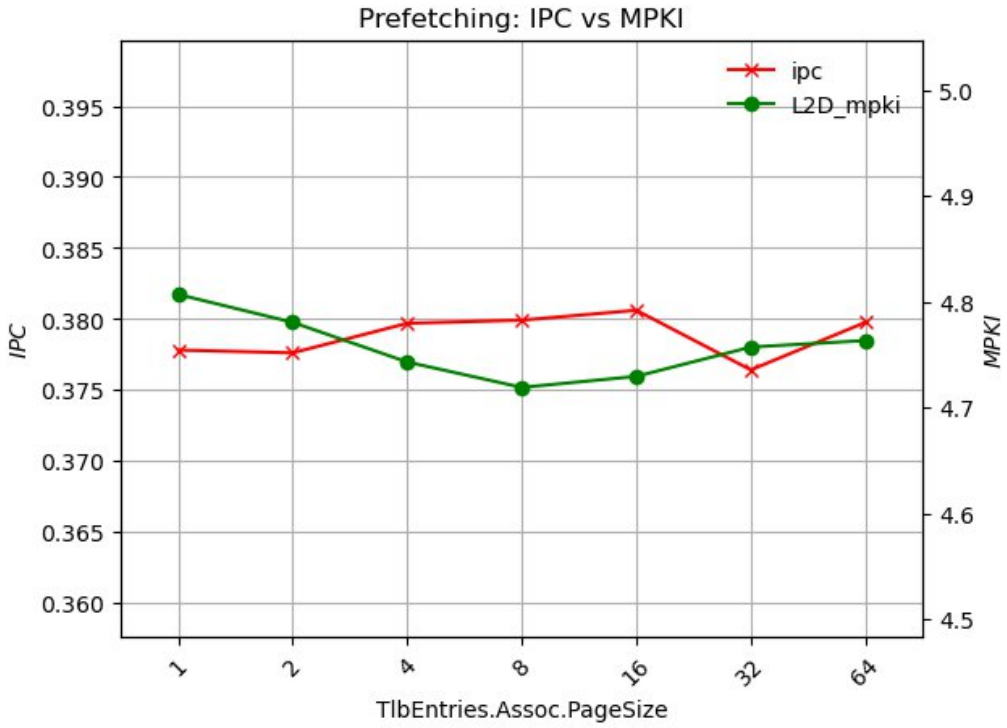
Συνολική παρουσίαση όλων των μετρήσεων για την TLB σε κοινή γραφική παράσταση



Στην περίπτωση της TLB παρατηρούμε ότι η αύξηση των cache size, associativity και block size οδηγεί σε βελτίωση της απόδοσης της μνήμης. Ωστόσο, παρατηρούμε ότι από ένα σημείο και έπειτα η αύξηση της συσχετιστικότητας, και ιδιαίτερα όσο πλησιάζουμε στην πλήρως συσχετιστική μνήμη, δεν προκαλεί ιδιαίτερη μεταβολή στο IPC και το MPKI, ενώ, μάλιστα, η μεγάλη τιμή του μέγεθος του μπλοκ σε σχέση με το μέγεθος της χωρητικότητας της μνήμης οδήγησε και σε μείωση της απόδοσης της μνήμης σε σχέση με προηγούμενες τιμές. Η ιδιαίτερα αυξημένη τιμή και η ιδιαίτερα μειωμένη τιμή που παρατηρούνται για τα MPKI και IPC αντίστοιχα οφείλονται στην άμεσης απεικόνισης μνήμης που εξετάζεται σε εκείνο το σημείο.

4) Προανάκληση (prefetching)

Συνολική παρουσίαση όλων των μετρήσεων για την L2 cache (με NEXT-N-LINE prefetching) σε κοινή γραφική παράσταση για τις διάφορες τιμές του N



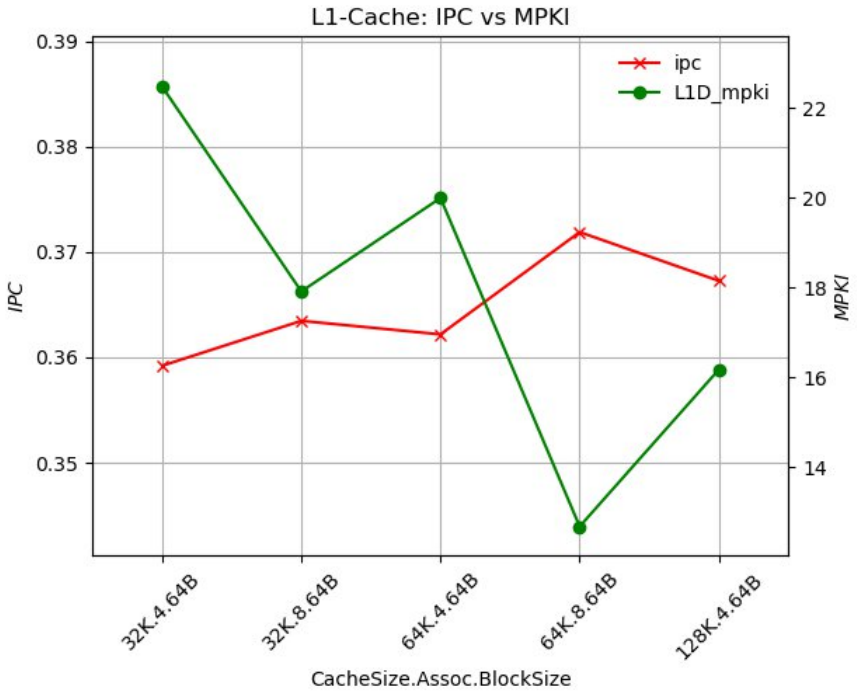
Παρατηρούμε ότι για N=1, 2, 4, 8 σημειώνεται βελτίωση στην απόδοση της κρυφής μνήμης αλλά για τις υπόλοιπες τιμές του N σημειώνεται κυρίως επιδείνωση της απόδοσης. Επομένως η ύπαρξη του ***NEXT-N-LINE prefetching*** είναι επιθυμητή μέχρι ενός συγκεκριμένου πλήθους μπλοκ.

5) Πολιτικές Αντικατάστασης (replacement policies)

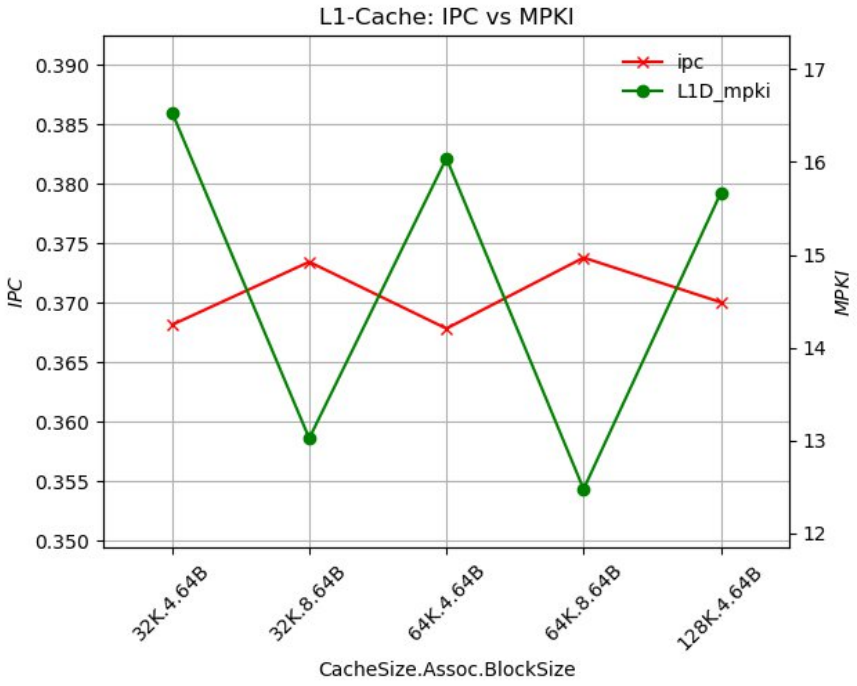
Για να μπορούμε να εξάγουμε κάποια συμπεράσματα θα συγκρίνουμε τις μετρήσεις που προέκυψαν από τις δύο πολιτικές αντικατάστασης που χρησιμοποιήθηκαν (LRU, τυχαίου μπλοκ).

Συνολική παρουσίαση όλων των μετρήσεων για την L1 cache σε κοινή γραφική παράσταση

Πολιτική Αντικατάστασης Τυχαίου Μπλοκ



Πολιτική Αντικατάστασης Least Recently Used (LRU)



Αρχικά παρατηρούμε ότι η παράμετρος που πραγματικά επηρεάζει και βελτιώνει την απόδοση της cache και στις δύο πολιτικές αντικατάστασης είναι η συσχετιστικότητα, καθώς η αύξηση της προκαλεί αύξηση του IPC και μείωση του MPKI. Η αύξηση του μεγέθους της χωρητικότητας της cache δεν επηρεάζει την απόδοσή της. Στην συνέχεια παρατηρούμε ότι σε όλες τις περιπτώσεις παραμετροποίησης η πολιτική αντι-

κατάστασης ***Least Recently Used (LRU)*** εμφανίζει τα καλύτερα αποτελέσματα, το οποίο είναι αναμενόμενο, καθώς εκμεταλλεύεται την χρονική τοπικότητα (σε αντίθεση με τη πολιτική αντικατάστασης ***Τυχαίου Μπλοκ***).

Με βάση όλες τις παραπάνω παρατηρήσεις για κάθε περίπτωση μπορούμε να εξάγουμε τα εξής συμπεράσματα για τα μετροπρογράμματα:

- Η αύξηση της χωρητικότητας της cache δεν προκαλεί ιδιαίτερα βελτιωτική επίδραση στην απόδοση της μνήμης, σε αντίθεση με την αύξηση της συσχετιστικότητας και του μεγέθους του μπλοκ, γεγονός που μπορεί να μας οδηγήσει στο συμπέρασμα ότι τα misses είναι κυρίως conflict misses και όχι compulsory misses,
- Η NEXT-N-LINE προανάκληση δεν προσφέρει ιδιαίτερη βελτίωση της απόδοσης της μνήμης, γεγονός που μπορεί να μας οδηγήσει στο συμπέρασμα ότι οι προσπελάσεις δεν αφορούν συνεχόμενα μπλοκ μνήμης.

Ζητούμενο 1.2

Ως αρχιτέκτονας για να επιλέξω ποιες L1 και L2 κρυφές μνήμες θα πρέπει να λάβω υπόψη μου τρεις κύριες παραμέτρους:

1. Απόδοση
2. Κόστος κατασκευής
3. Κατανάλωση ηλεκτρικής ισχύος.

Αν λάμβανα υπόψη μόνο την απόδοση για τις δύο μνήμες θα επέλεγα τις μνήμες με τα εξής χαρακτηριστικά σύμφωνα με τις μετρήσεις που λάβαμε στο ζητούμενο 1.1:

	L1	L2
Cache Size (KB)	128	2048
Associativity	8	16
Block Size (B)	128	256

Ωστόσο, η μεγάλη χωρητικότητα και η αυξημένη πολυπλοκότητα τείνουν να αυξάνουν το κόστος κατασκευής και την κατανάλωση ηλεκτρικής ισχύος. Για τον λόγο αυτό θα χρειαστεί να κάνω ένα συμβιβασμό και να επιλέξω μνήμες με μειωμένη χωρητικότητα και πολυπλοκότητα αλλά με κοντινή απόδοση στις ιδανικές. Έτσι, μια επιλογή που θα μπορούσα να κάνω λαμβάνοντας υπόψη και τις τρεις παραμέτρους που ανέφερα παραπάνω είναι οι μνήμες με τα εξής χαρακτηριστικά:

	L1	L2
Cache Size (KB)	64	1024
Associativity	8	8
Block Size (B)	128	256

Ζητούμενο 2

Στα συμπεράσματα που εξάγαμε στο ζητούμενο 1.1 είδαμε ότι ο διπλασιασμός κάποιου μεγέθους στην πλειονότητα των περιπτώσεων επέφερε βελτίωση της απόδοσης της μνήμης μέσω της ταυτόχρονης αύξησης του IPC και μείωσης του MPKI. Στην περίπτωση όμως που εισέλθουν καθυστερήσεις κατά την αύξηση των μεγεθών, όσο διπλασιάζονται τα μεγέθη και φτάνουμε σε μεγαλύτερες τιμές, τόσο πιο μεγάλες θα είναι και οι καθυστερήσεις που θα εισέρχονται. Αυτό θα έχει ως αποτέλεσμα, η θετική επίδραση που θα είχε ο διπλασιασμός ενός μεγέθους στην απόδοση της μνήμης να αναιρείται σε ένα βαθμό ή και εξ ολοκλήρου από τις μεγάλες καθυστερήσεις που θα εισάγονται οδηγώντας σε μειωμένη απόδοση σε σχέση με τις μετρήσεις μας στο ζητούμενο 1.1 ή ακόμα και σε επιδείνωση της απόδοσης της μνήμης.

Οι επιλογές μας ως προς την επιλογή των καλύτερων κρυφών μνημών L1 και L2, στην περίπτωση αυτή, θα είναι τελείως διαφορετικές καθώς θα πρέπει να λάβουμε υπόψη τα εξής:

- Ο διπλασιασμός της χωρητικότητας της μνήμης και της συσχετιστικότητας οδηγούν σε καθυστερήσεις,
- Ο διπλασιασμός της χωρητικότητας επιφέρει τις μεγαλύτερες καθυστερήσεις,
- Θα πρέπει να αποφευχθούν τα μεγάλα μεγέθη, καθώς θα προκαλούν μεγάλες καθυστερήσεις,
- Ο διπλασιασμός του block size δεν επιφέρει καθυστερήσεις, οπότε θα πρέπει να επικεντρωθούμε περισσότερο σε αυτή την παράμετρο.

Λαμβάνοντας υπόψη τις παραπάνω παραμέτρους, κάποιες επιλογές που θα μπορούσαν να γίνουν για τις μνήμες L1 και L2 είναι οι εξής:

	L1	**L2**
Cache Size (KB)	32	512
Associativity	8	8
Block Size (B)	128	256

Ωστόσο, αυτές οι επιλογές είναι κυρίως εκτιμήσεις και όχι επιλογές βασισμένες σε συγκεκριμένες μετρήσεις. Προκειμένου να γίνουν οι πραγματικά σωστές επιλογές θα πρέπει να επανεκτελεστούν οι προσομοιώσεις με τις κατάλληλες αλλαγές στην υλοποίηση της μνήμης ώστε να περιλαμβάνει τις καθυστερήσεις που προκαλούν οι αυξήσεις των παραμέτρους της μνήμης.

Παράρτημα Κώδικα

Επέκταση κώδικα του αρχείου cache.h για την υλοποίηση του prefetching

```
// PREFETCHING
ADDRINT prefetch_addr = addr;
for (UINT32 i=0; i < _l2_prefetch_lines; i++) {
    prefetch_addr += L2BlockSize();
    /* ........................ */
    /* Add here prefetching code. */
    /* ........................ */
    SplitAddress(prefetch_addr, L2LineShift(), L2SetIndexMask(), l2Tag, l2SetIndex);
    SET & l2Set = _l2_sets[l2SetIndex];
    l2Hit = l2Set.Find(l2Tag);
    if (!l2Hit) {
        CACHE_TAG l2_replaced = l2Set.Replace(l2Tag);
                if ((L2_INCLUSIVE == 1) && !(l2_replaced == INVALID_TAG)) {
                    ADDRINT replacedAddr = ADDRINT(l2_replaced) << FloorLog2(L2NumSets());
                    replacedAddr = replacedAddr | l2SetIndex;
                    replacedAddr = replacedAddr << L2LineShift();
                    for (UINT32 i=0; i < L2BlockSize(); i+=L1BlockSize()) {
                        ADDRINT newAddr = replacedAddr | i;
                        SplitAddress(newAddr, L1LineShift(), L1SetIndexMask(), l1Tag, l1SetIndex);
                        l1Set = _l1_sets[l1SetIndex];
                        l1Set.DeleteIfPresent(l1Tag);
                    }
            }
    }
}
```

Επέκταση κώδικα του αρχείου cache.h για την υλοποίηση της πολιτικής αντικατάστασης τυχαίου μπλοκ

```
UINT32 Find(CACHE_TAG tag)
{
    for (std::vector<CACHE_TAG>::iterator it = _tags.begin();
         it != _tags.end(); ++it)
    {
        if (*it == tag) { // Tag found, lets make it MRU
            #_tags.erase(it);
            #_tags.push_back(tag);
            return true;
        }
    }

    return false;
}

CACHE_TAG Replace(CACHE_TAG tag)
{
    CACHE_TAG ret = INVALID_TAG;
    _tags.push_back(tag);

    if (_tags.size() > _associativity) {
        srand(20853);
        int randomIndex = rand() % (_tags.size()-1);
            ret = _tags[randomIndex];
            _tags.erase(_tags.begin() + randomIndex);
    }
    return ret;
}
```

Υπολογισμός των μέσω όρων των IPC και MPKI για την L1 cache

```
#!/bin/bash

# Assign directory path to variable
directory="/home/manolis/Downloads/AdvancedArch/parsec-3.0-core/parsec-
3.0/parsec_workspace/outputs/L1"

cd /home/manolis/Downloads/AdvancedArch/parsec-3.0-core/parsec-
3.0/parsec_workspace/results/L1

echo "Reading values from output files..."
# Loop through files in the directory
for file in "$directory"/*; do
    # Check if the file is a regular file
    if [ -f "$file" ]; then
        # Read IPC value from the file
        ipc_value=$(grep -oP 'IPC:\s+\K\d+(\.\d+)?' "$file")
        # Read Total Instructions value from the file
        total_instr=$(grep -oP 'Total Instructions:\s+\K\d+' "$file")
        # Read L1-Total-Misses value from the file
        l1_total_misses=$(grep -oP '^L1-Total-Misses:\s+\K\d+' "$file")
        # Extracting L1 cache parameters
    L1_SIZE=$(grep "Size(KB):" $file | head -n 1 | awk '{print $2}')
    L1_BLOCK_SIZE=$(grep "Block Size(B):" $file | head -n 1 | awk '{print $3}')
    L1_ASSOCIATIVITY=$(grep "Associativity:" $file | tail -n 2 | head -n 1 | awk
'{print $2}')

    filename="${L1_SIZE}_${L1_ASSOCIATIVITY}_${L1_BLOCK_SIZE}"
    if [ ! -e "$filename" ]; then
        echo "L1-Data Cache" >> $filename
            echo "SIZE(KB): $L1_SIZE" >> "$filename"
            echo "BLOCK SIZE(B): $L1_BLOCK_SIZE" >> "$filename"
            echo "ASSOCIATIVITY: $L1_ASSOCIATIVITY" >> "$filename"
        fi
        echo $ipc_value >> $filename
        LC_NUMERIC="C" mpki=$(awk "BEGIN {printf \"%.6f\", $l1_total_misses * 1000
/ $total_instr}")
        if [ ! -d "./MPKI" ]; then
            mkdir ./MPKI
        fi
        echo $mpki >> "./MPKI/$filename"
    fi
done

max_IPC_mean=0
max_IPC_filename=""

echo "Calculating IPC mean values..."
for file in ./*; do
    # Check if the file is a regular file
    if [ -f "$file" ]; then
        #Initialize variables
        total=0
```

```
        count=0
        # Loop through lines 4 to 11 from the file
        while IFS= read -r line; do
            # Check if the line is numeric
            if [[ $line =~ ^[0-9]+([.][0-9]+)?$ ]]; then
                # Add the value to the total
                LC_NUMERIC="C" rline=$(awk "BEGIN {printf \"%.6f\", 1 / $line}")
                total=$(echo "$total + $rline" | bc)
                ((count++))
            fi
        done < <(sed -n '5,12p' "$file")

        # Calculate mean
        if [ $count -gt 0 ]; then
            LC_NUMERIC="C" mean=$(awk "BEGIN {printf \"%.6f\", $count / $total}")
        if [ -n "$mean" ]; then
            printf "Mean IPC ($(basename $file)): %.6f\n" "$mean"
            printf "Mean IPC: %.6f\n" "$mean" >> "$file"
            # Check if current mean IPC value is greater than max_mean
                    if (( $(awk 'BEGIN {print ("'"$mean"'" >
"'"$max_mean"'")}') )); then
                    max_mean=$mean
                    max_file=$file
                    fi
        else
            echo "Error: Mean calculation failed."
        fi
        else
            echo "No valid numeric lines found."
        fi
    fi
done
echo "Calculated IPC mean values."

echo "Calculating MPKI mean values..."
for file in "./MPKI"/*; do
    # Check if the file is a regular file
    if [ -f "$file" ]; then
        #Initialize variables
        total=0
        count=0
        # Loop through lines 4 to 11 from the file
        while IFS= read -r line; do
            # Check if the line is numeric
            if [[ $line =~ ^[0-9]+([.][0-9]+)?$ ]]; then
                # Add the value to the total
                total=$(echo "$total + $line" | bc)
                ((count++))
            fi
        done < <(sed -n '1,8p' "$file")

        # Calculate mean
        if [ $count -gt 0 ]; then
            LC_NUMERIC="C" mean=$(awk "BEGIN {printf \"%.6f\", $total / $count}")
```

```
        if [ -n "$mean" ]; then
            printf "Mean MPKI ($(basename $file)): %.6f\n" "$mean"
            printf "Mean MPKI: %.6f\n" "$mean" >> $(basename $file)
        else
            echo "Error: Mean calculation failed."
        fi
        else
            echo "No valid numeric lines found."
        fi
    fi
done
echo "Calculated MPKI mean values."
rm -r ./MPKI
if [ -n "$max_file" ]; then
    printf "Max Mean IPC: %.6f (Found in file: %s)\n" "$max_mean" "$(basename
"$max_file")"
else
    echo "No valid mean IPC values found."
fi
```

Με αντίστοιχο τρόπο υπολογίζονται οι μέσοι όροι για όλες τις περιπτώσεις που εξετάζουμε.