Σχολή Ηλεκτρολόγων Μηχανικών

&

Μηχανικών Υπολογιστών

Προηγμένα θέματα Αρχιτεκτονικής Υπολογιστών

2η Σειρά Ασκήσεων

Μανδηλαράς Νικηφόρος ,Α.Μ. 03112012

Εξάμηνο : 8ο

Παράδοση 23/4/16

Περιγραφή και Σκοπός της Άσκησης

Στη συγκεκριμένη άσκηση κληθήκαμε να υλοποιήσουμε μια σειρά από branch predictors και να αποφανθούμε για την αποδοτικότητα αυτών τρέχοντας μια σειρά μετροπρογραμμάτων, με τη βοήθεια του εργαλείου PIN. Η ανάγκη για την ύπαρξη των branch predictors οφείλεται στο γεγονός πως μια εντολή διακλάδωσης μπορεί να καθυστερήσει αρκετά την εκτέλεση των προγραμμάτων μας καθώς μέχρι να αποφανθούμε για τον προορισμό της, δεν είμαστε σίγουροι για το αν οι νέες εντολές που εισάγονται στο pipeline του επεξεργαστή μας είναι και οι σωστές. Οι predictors αναλαμβάνουν να προσδιορίσουν το σημείο απ' το οποίο θα επιλέξουμε την επόμενη εντολή για εισαγωγή.

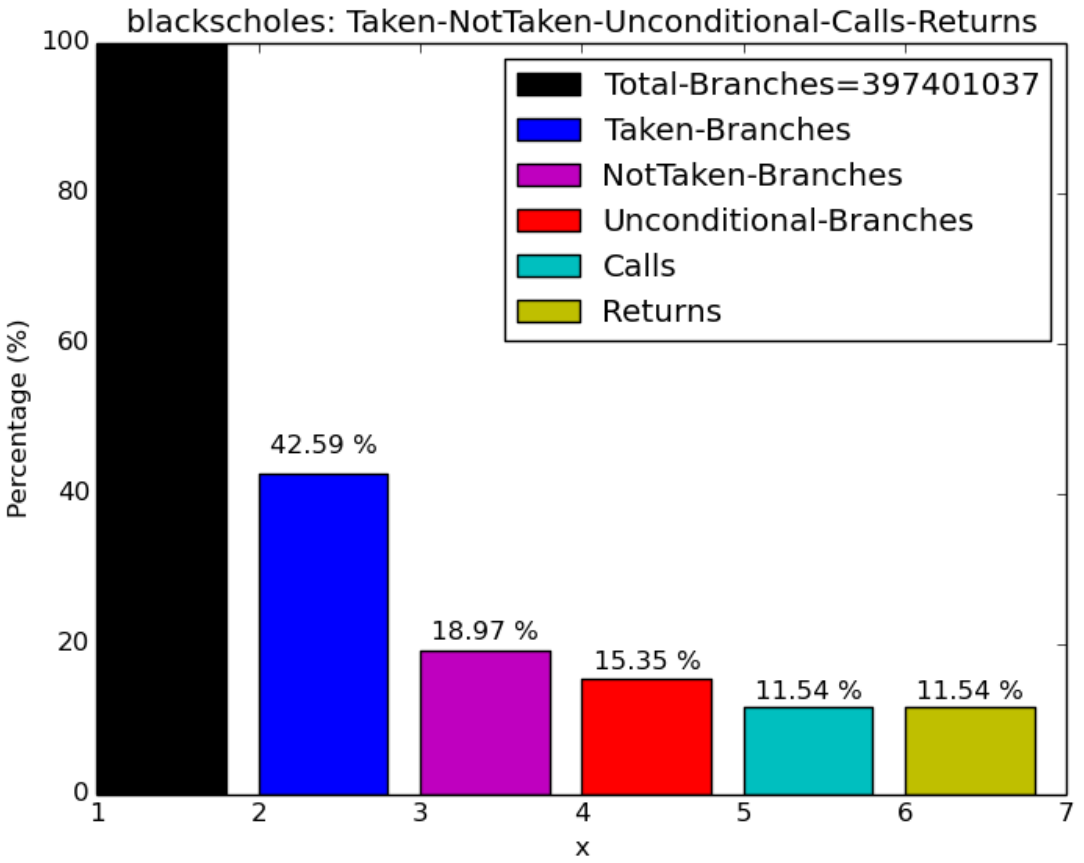
4. Πειραματική Αξιολόγηση

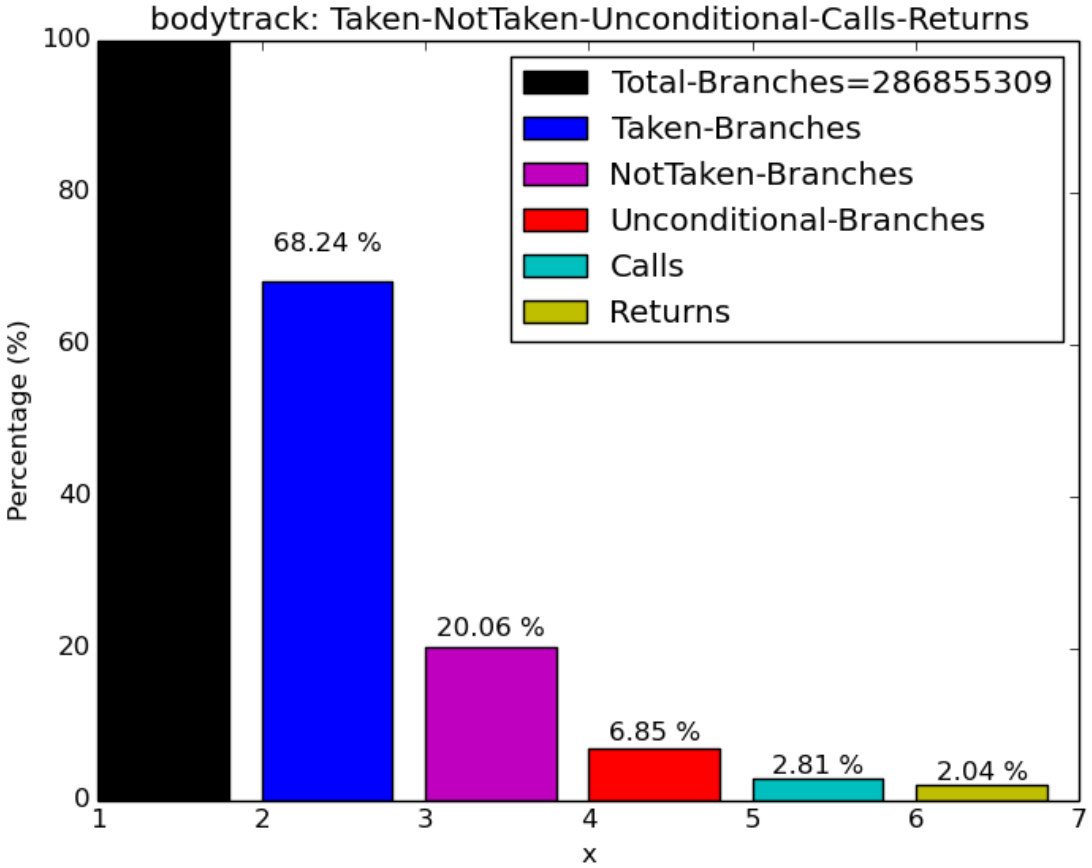
4.1 Μελέτη εντολών άλματος

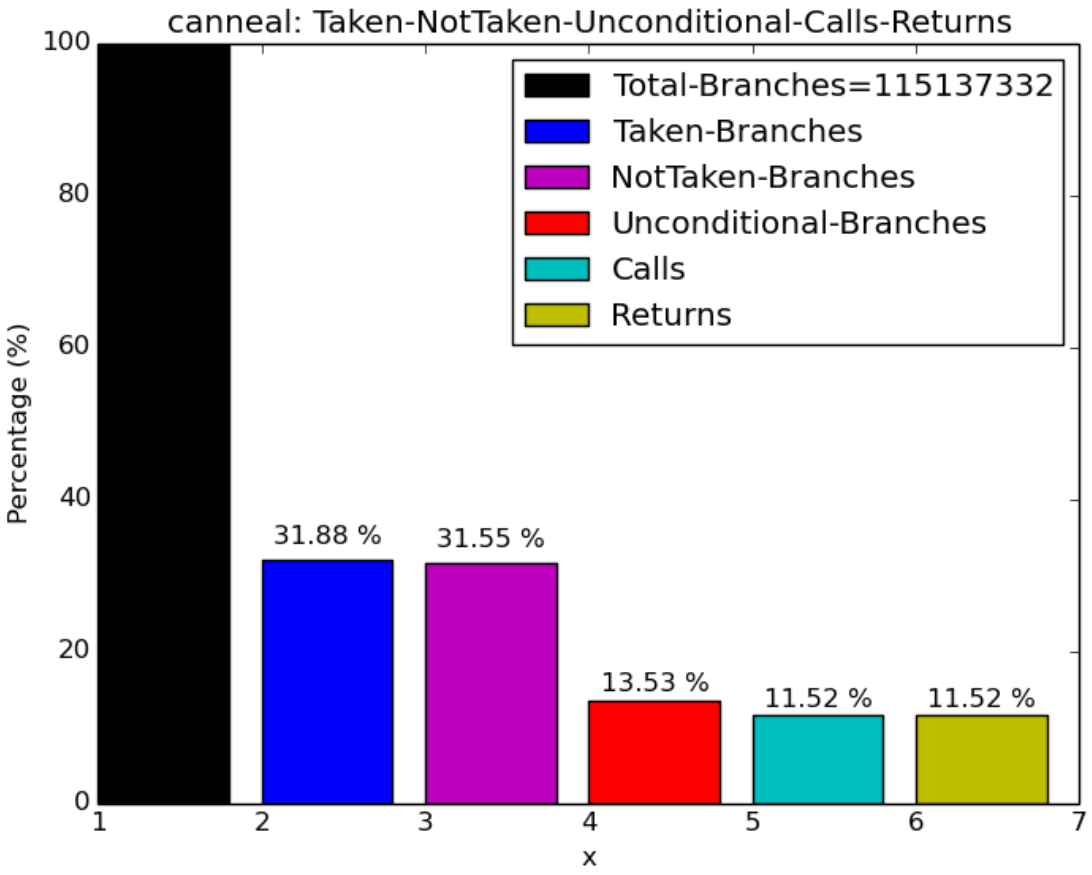
Στο πρώτο σκέλος της άσκησης συλλέξαμε στατιστικά για το είδος των εντολών άλματος που συμβαίνουν σε κάθε μετροπρόγραμμα. ΟΙ κατηγορίες αυτών ήταν :

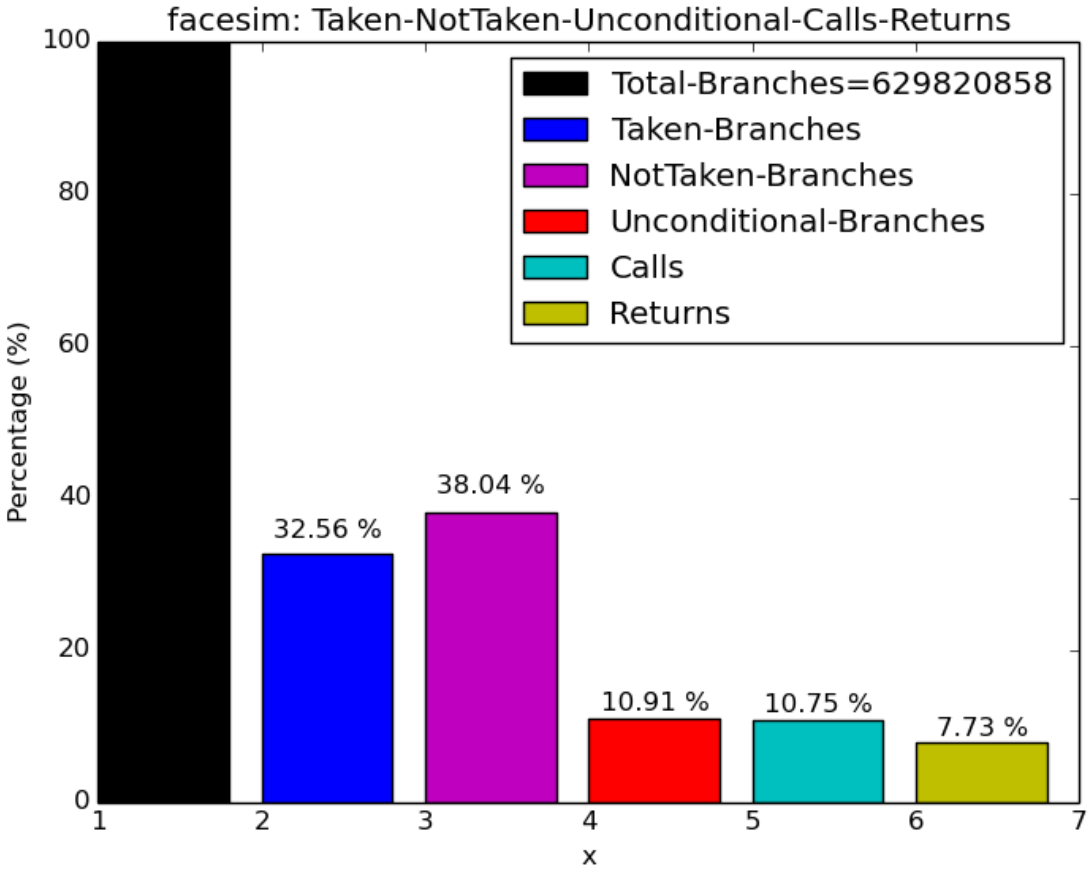
1) Conditional-taken braches
2)Conditional -not-taken branches
3)Unconditioanl – branches
4)Calls
5)Returns

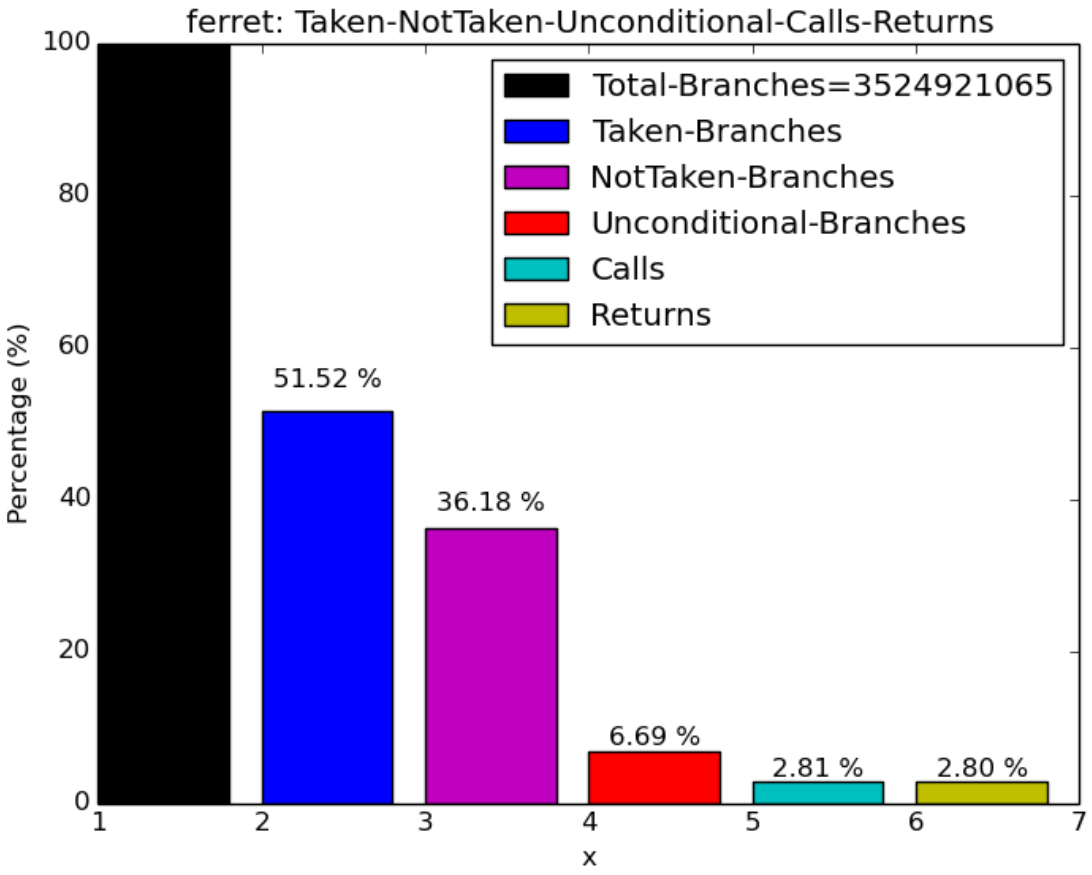
Παρακάτω παρουσιάζονται ο αριθμός των συνολικών branches καθώς και τα ποσοστά αυτών των κατηγοριών ανά benchmark.

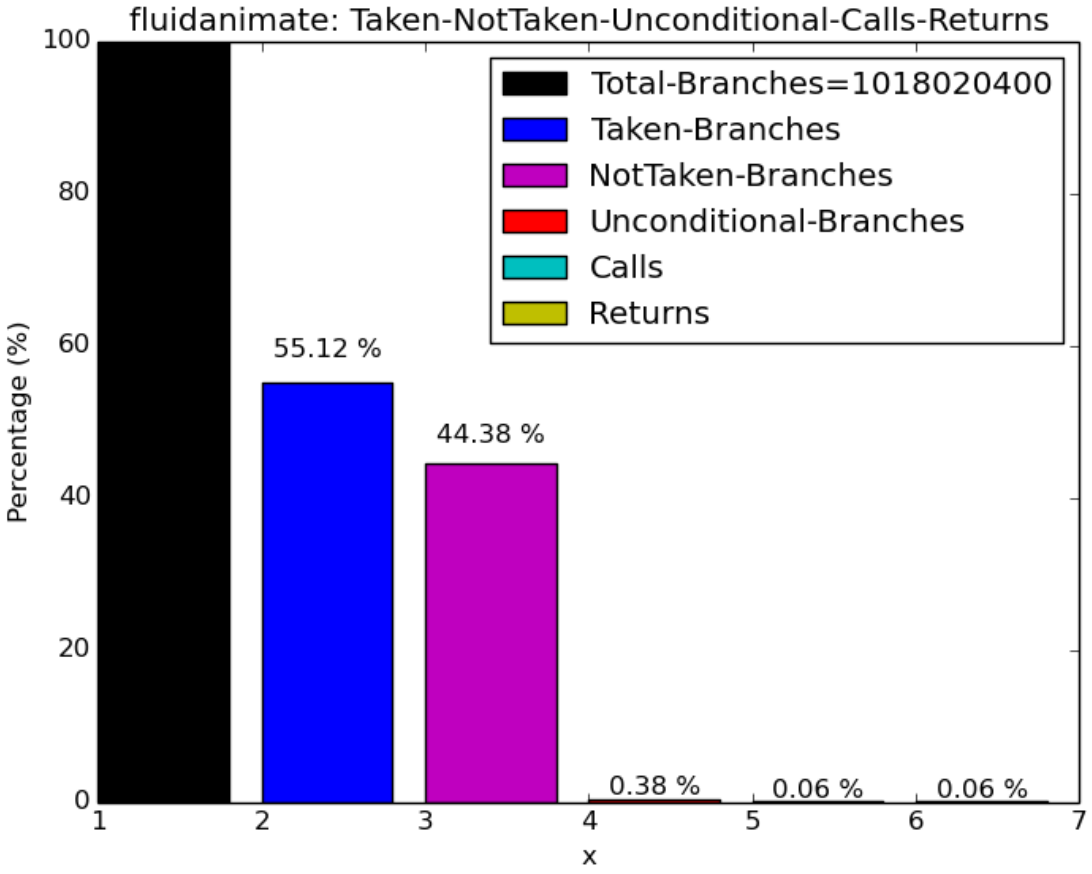

blackscholes: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=397401037
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
42.59 %
18.97 %
15.35 %
11.54 %
11.54 %
Percentage (%)
x

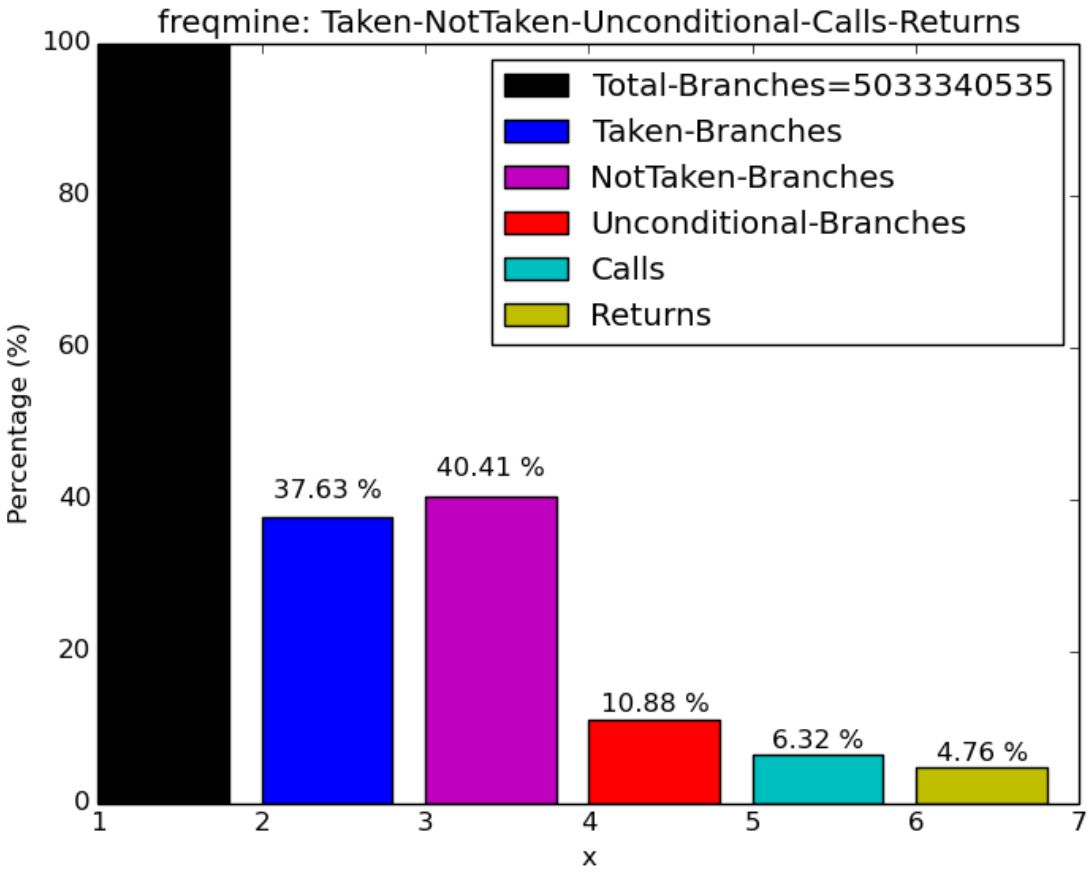

bodytrack: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=286855309
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
68.24 %
20.06 %
6.85 %
2.81 %
2.04 %
Percentage (%)
x

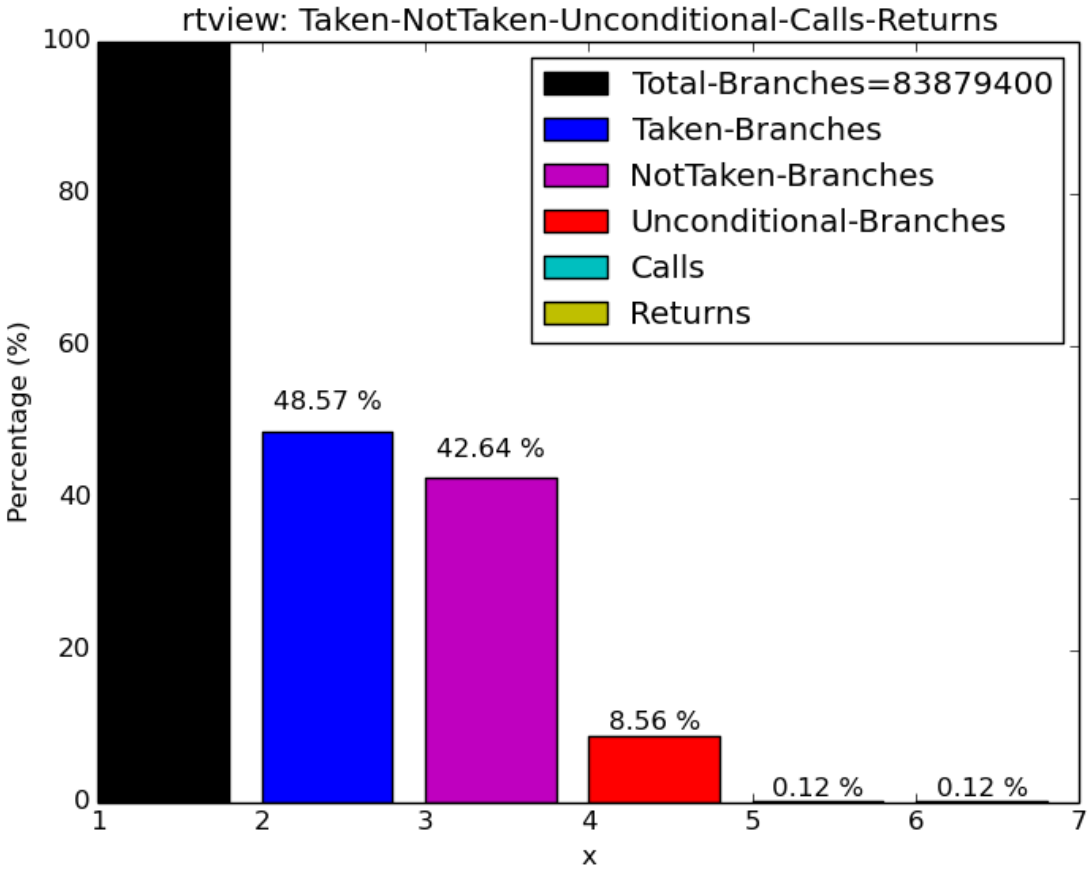

canneal: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=115137332
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
Percentage (%)
x
31.88 %
31.55 %
13.53 %
11.52 %
11.52 %

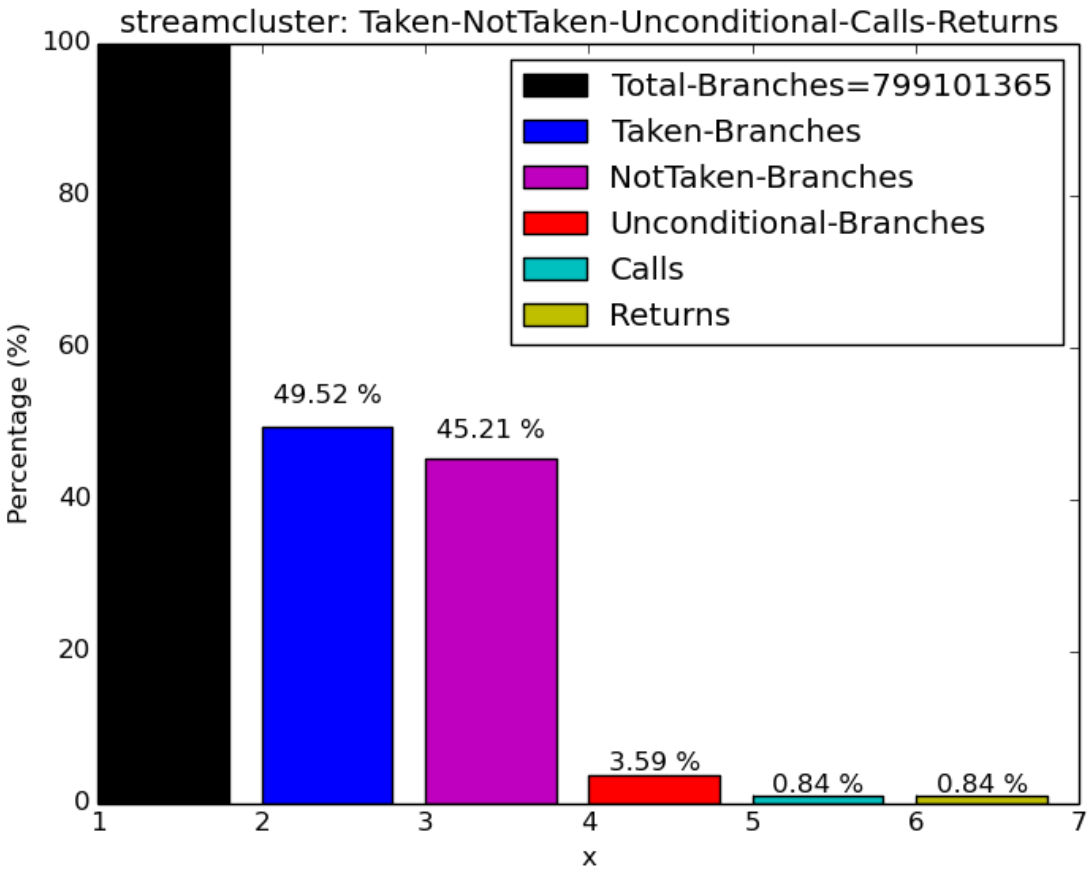

facesim: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=629820858
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
Percentage (%)
x
32.56 %
38.04 %
10.91 %
10.75 %
7.73 %

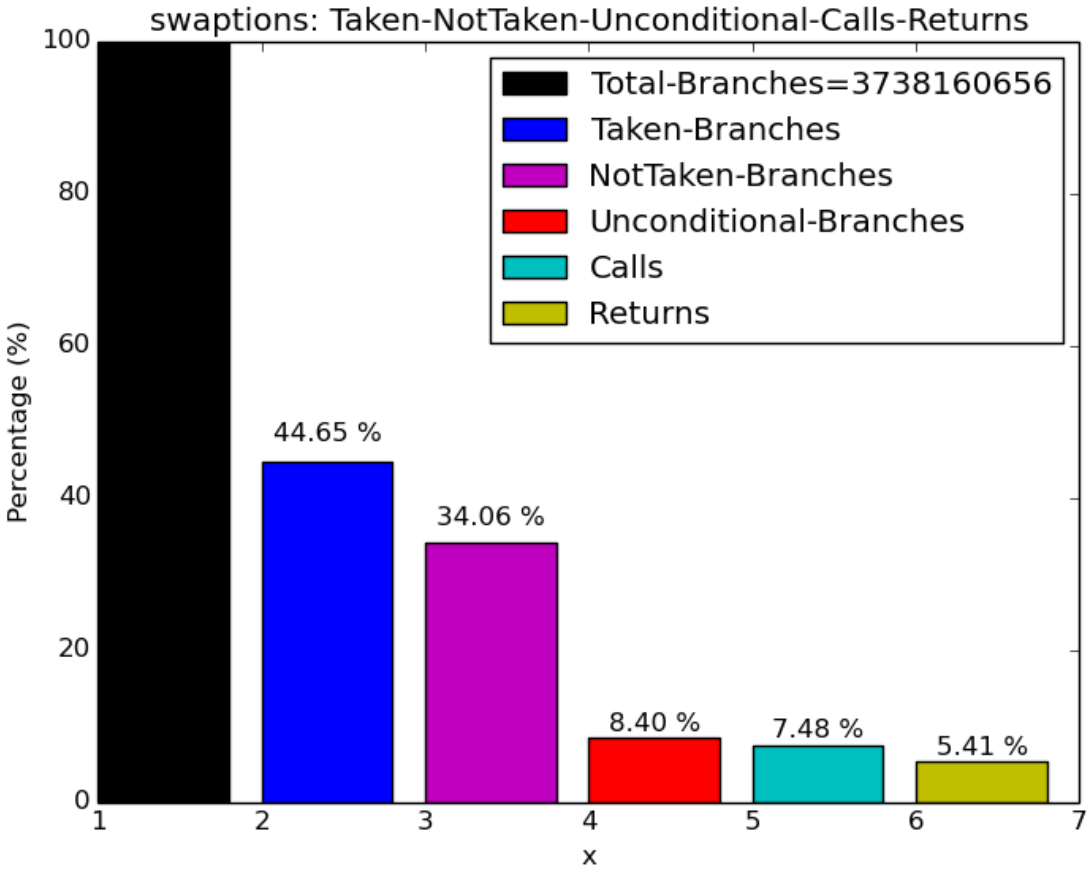

ferret: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=3524921065
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
51.52 %
36.18 %
6.69 %
2.81 %
2.80 %
Percentage (%)
x


fluidanimate: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=1018020400
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
55.12 %
44.38 %
0.38 %
0.06 %
0.06 %
Percentage (%)
x


freqmine: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=5033340535
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
Percentage (%)
x
37.63 %
40.41 %
10.88 %
6.32 %
4.76 %


rtview: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=83879400
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
Percentage (%)
x
48.57 %
42.64 %
8.56 %
0.12 %
0.12 %


streamcluster: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=799101365
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
49.52 %
45.21 %
3.59 %
0.84 %
0.84 %
Percentage (%)
x

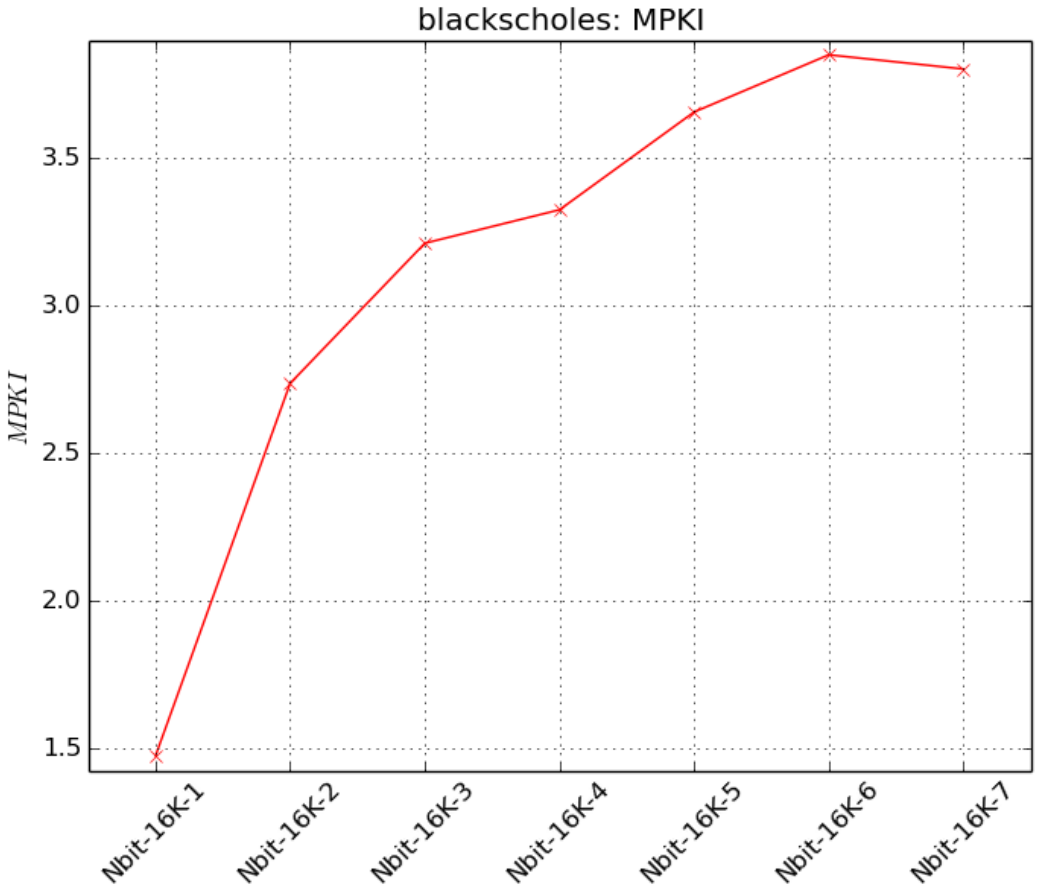

swaptions: Taken-NotTaken-Unconditional-Calls-Returns
Total-Branches=3738160656
Taken-Branches
NotTaken-Branches
Unconditional-Branches
Calls
Returns
44.65 %
34.06 %
8.40 %
7.48 %
5.41 %
Percentage (%)
x

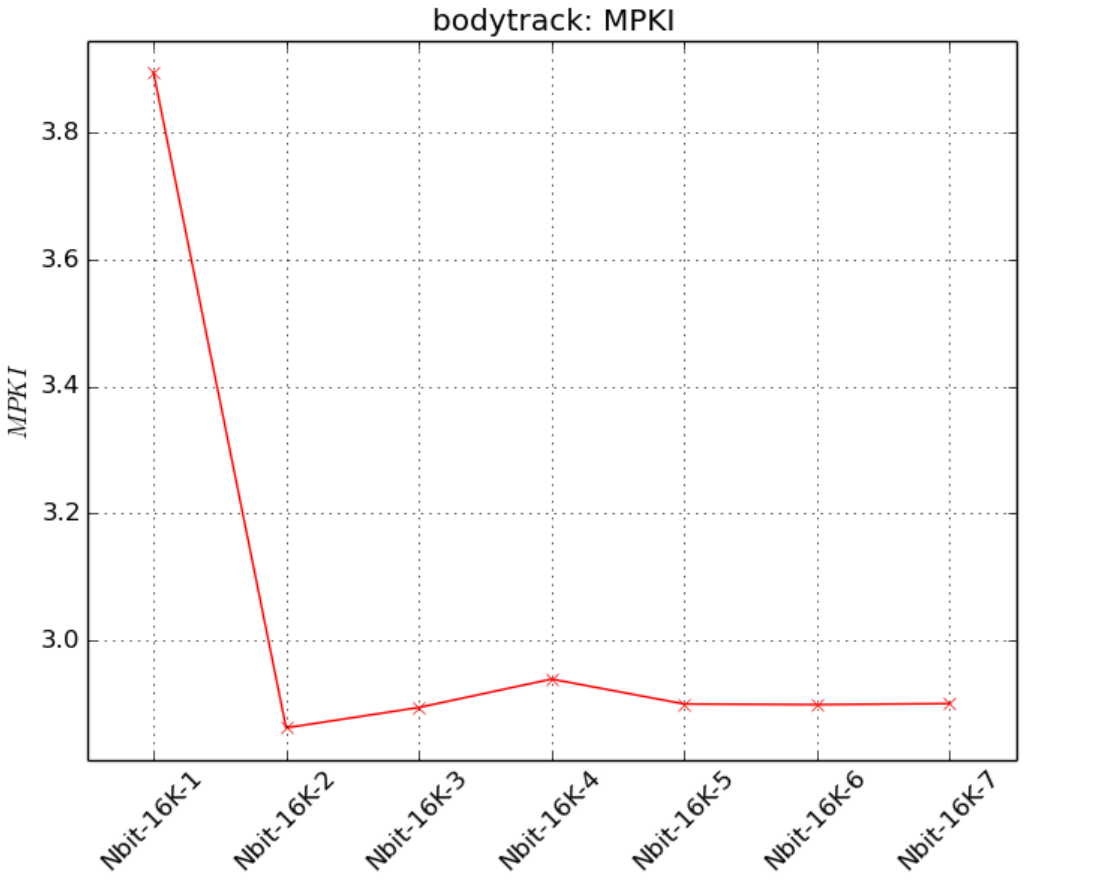
Το πλήθος των conditional branches (Taken – Not Taken) είναι αρκετά μεγαλύτερο από όλων των υπολοίπων. Στα μετροπρογράμματα blackscholes και canneal συγκεκριμένα βλέπουμε αυξημένες τιμές για τις υπόλοιπες κατηγορίες branches σε σχέση με τις υπόλοιπες . Εικάζουμε ότι λόγο αυτού μπορεί να προκύψουν διαφορετικά αποτελέσματα σε μεθόδους όπως η ras ή η btb όπου οι εντολές return από συναρτήσεις έχουν και διαφορετική συμπεριφορά .

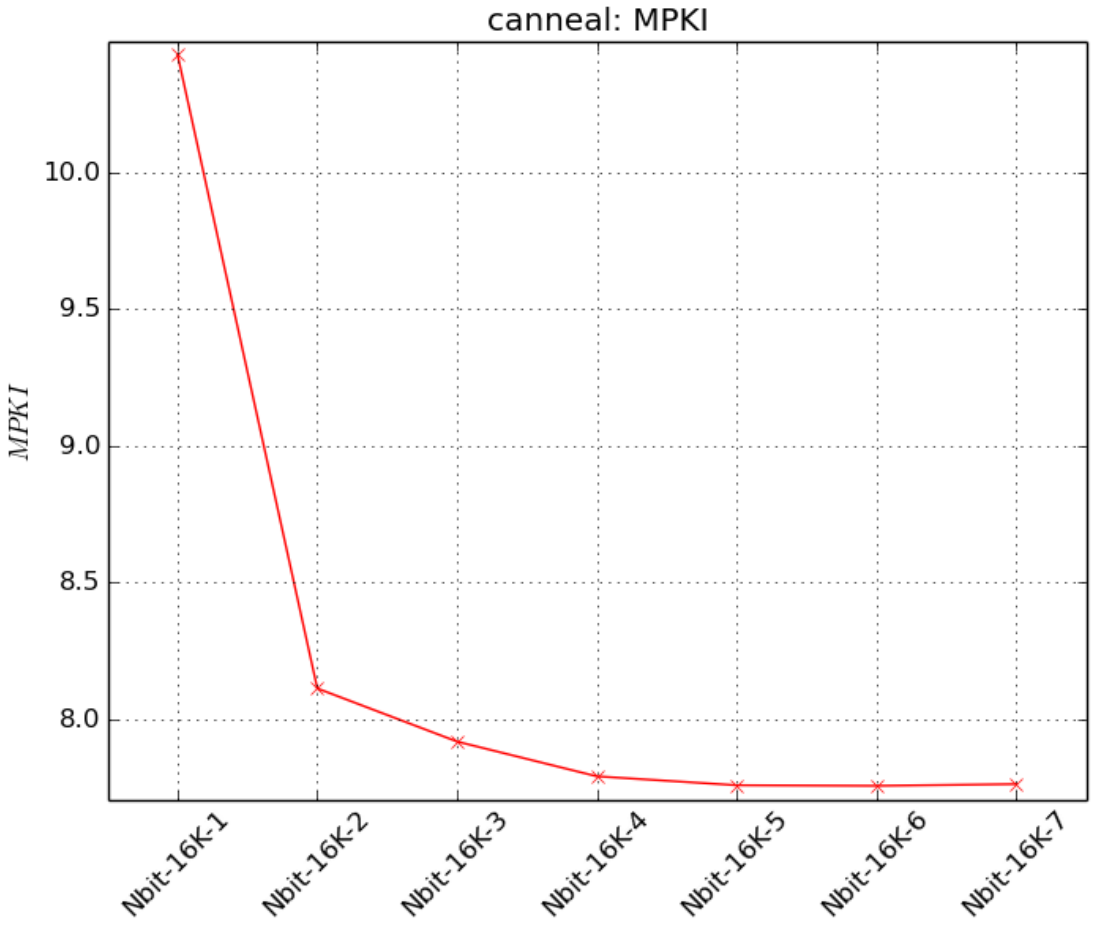
Ως μέτρο αξιολόγησης θα ορίσουμε το MPKI δηλαδή των αριθμό λάθος προβλέψεων ανά χίλιες εντολές.

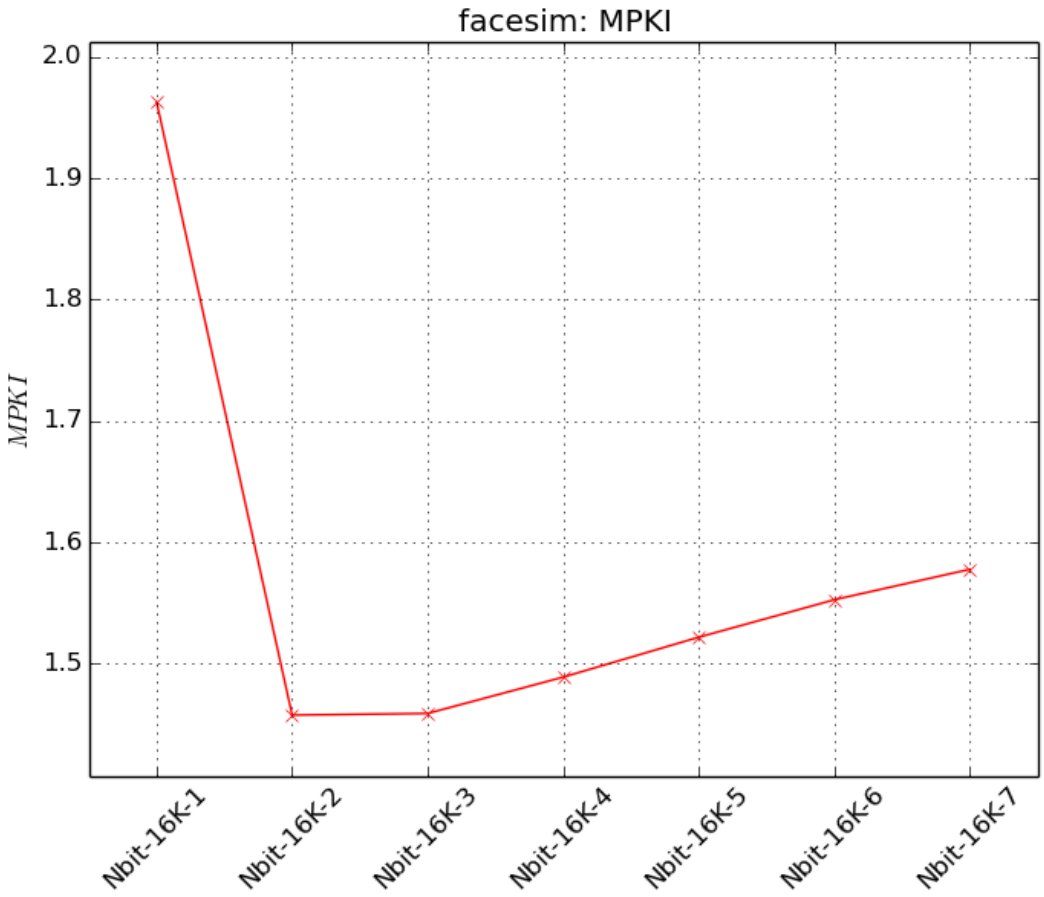
4.2 Μελέτη των N-bit predictors

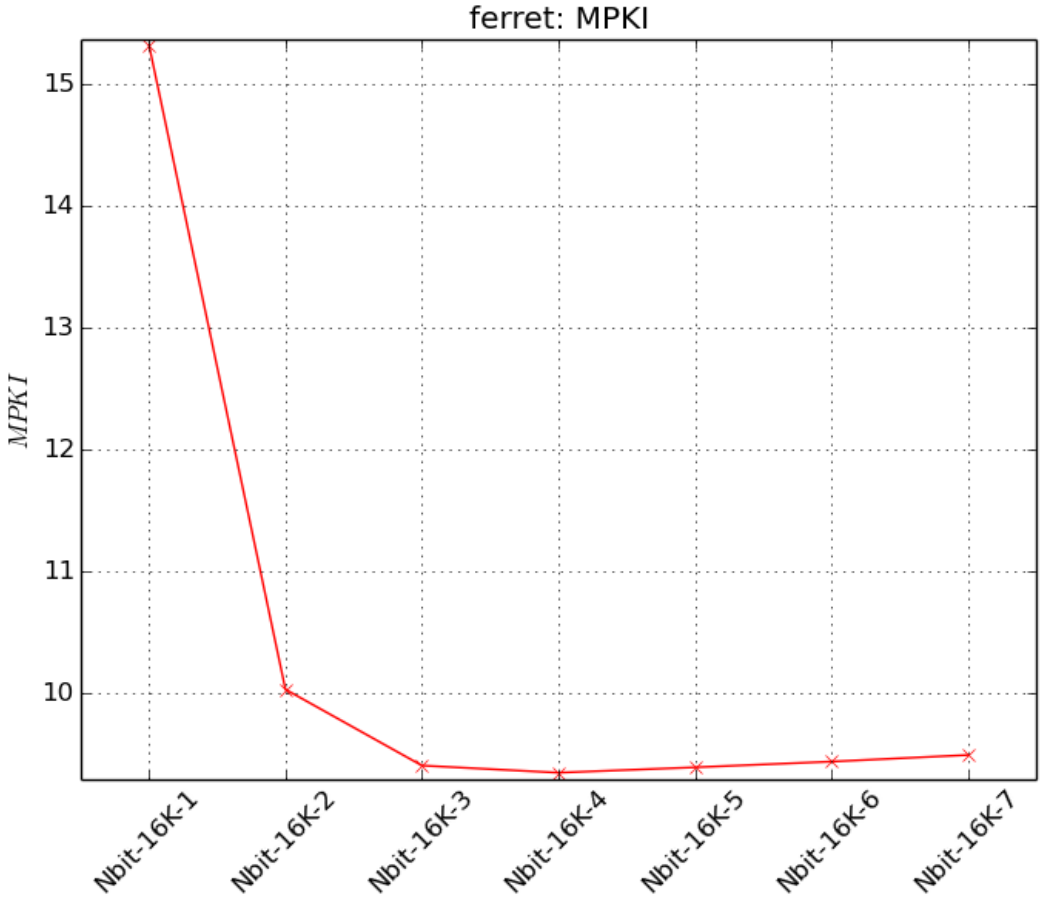
I) Αρχικά διατηρούμε σταθερό τον αριθμό των κελιών του πίνακα BHT αλλά αυξάνουμε διαδοχικά κατά ένα τα bit των predictors. Σ αυτή την περίπτωση έχουμε διαδοχική αύξηση του hardware. Ακολουθούν τα διαγράμματα των 7 αυτών n-bit predictor για κάθε benchmark.

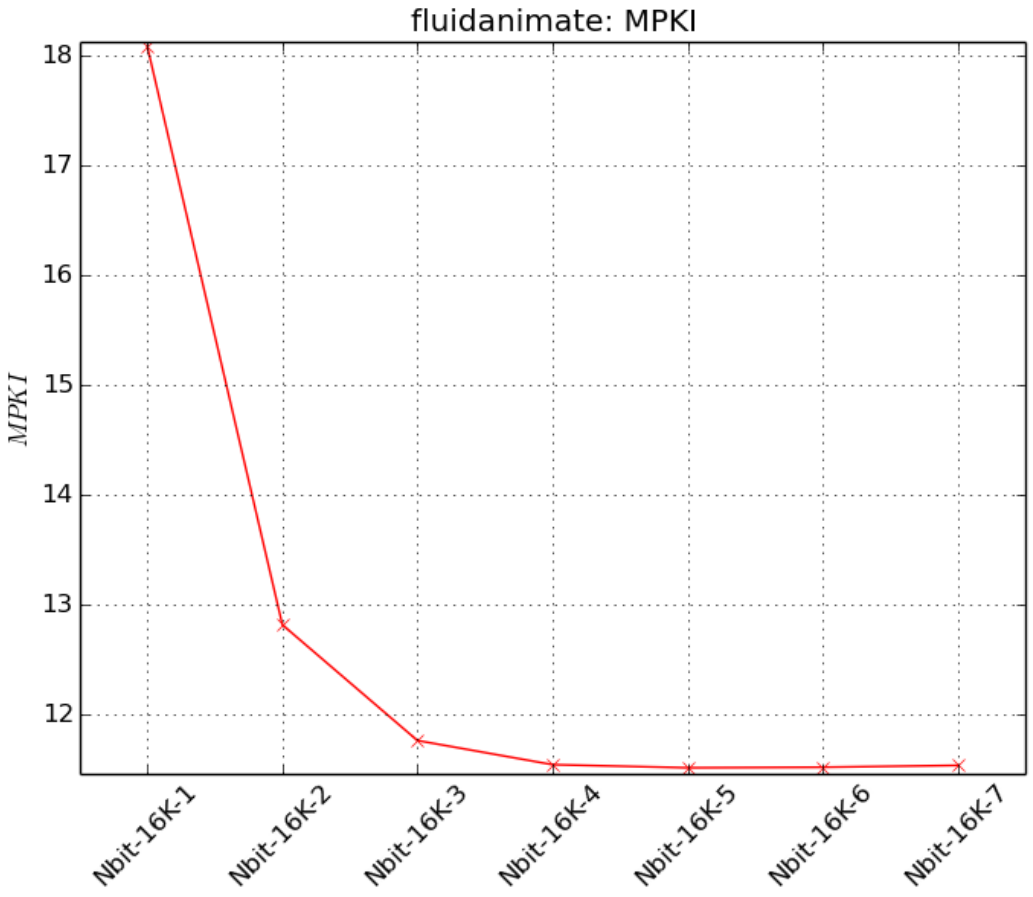


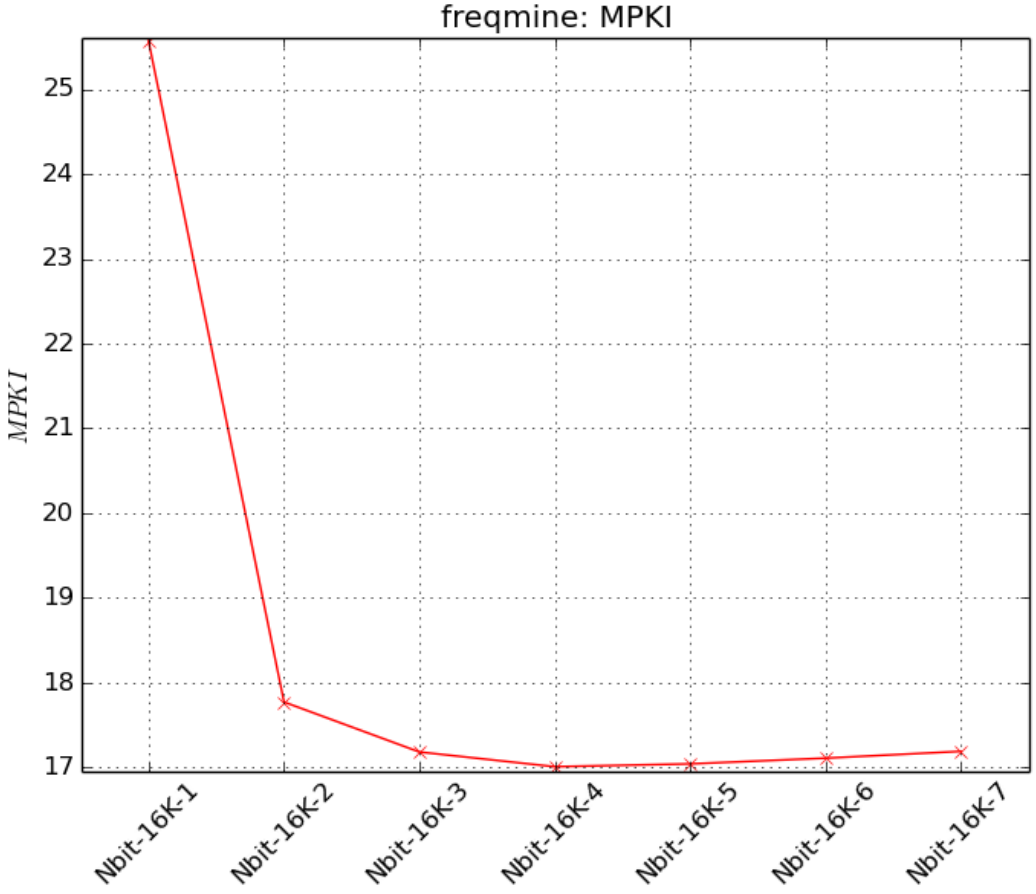

bodytrack: MPKI
MPKI
3.8
3.6
3.4
3.2
3.0
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
Nbit-16K-5
Nbit-16K-6
Nbit-16K-7

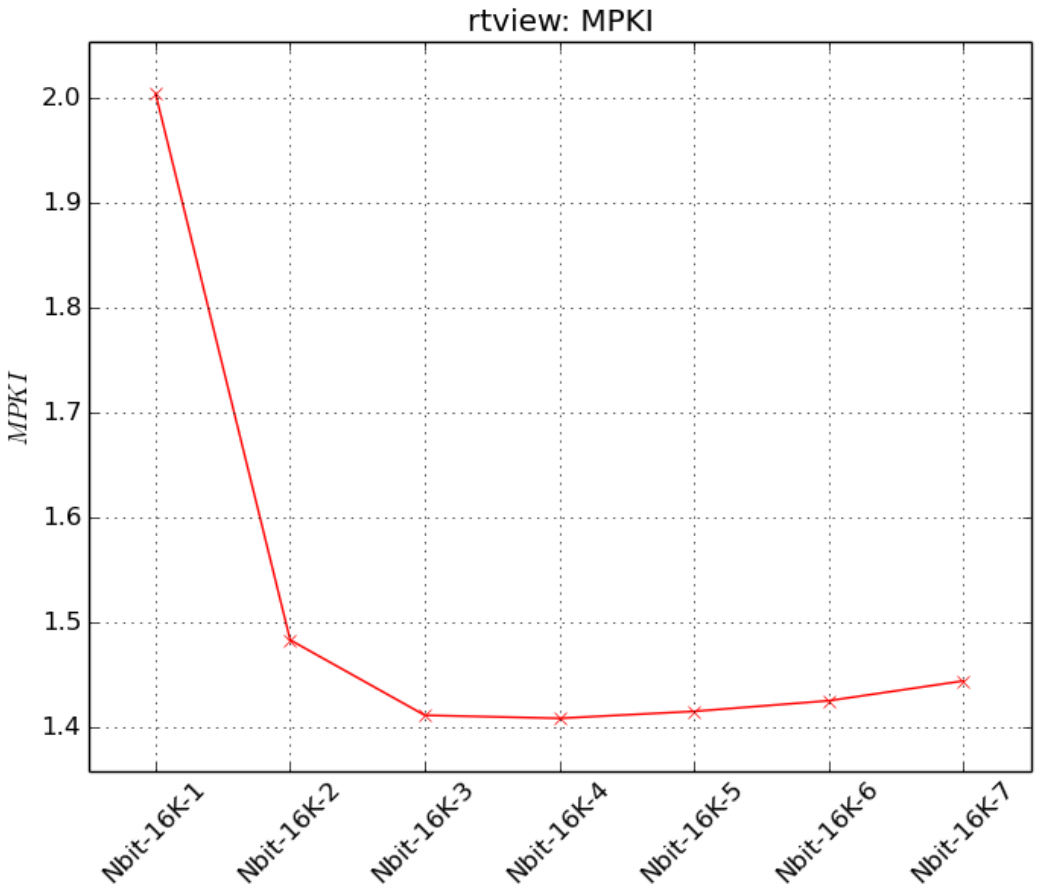

canneal: MPKI
MPKI
10.0
9.5
9.0
8.5
8.0
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
Nbit-16K-5
Nbit-16K-6
Nbit-16K-7

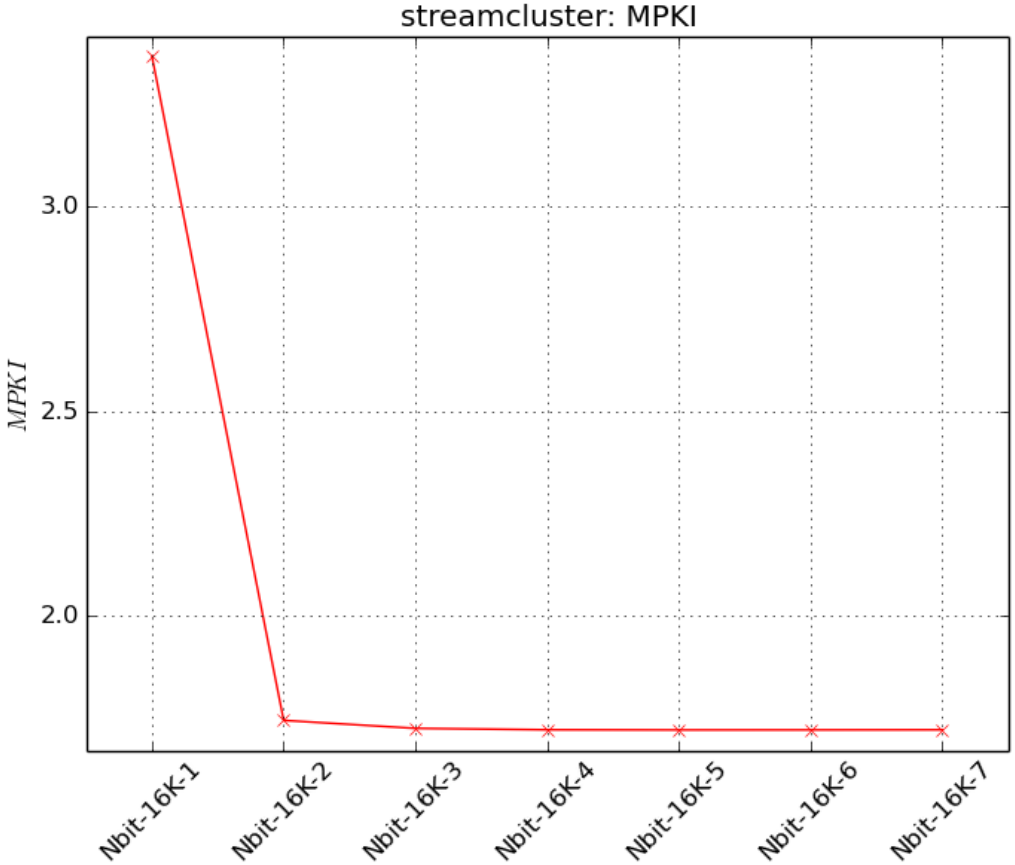

facesim: MPKI
MPKI
2.0
1.9
1.8
1.7
1.6
1.5
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
Nbit-16K-5
Nbit-16K-6
Nbit-16K-7

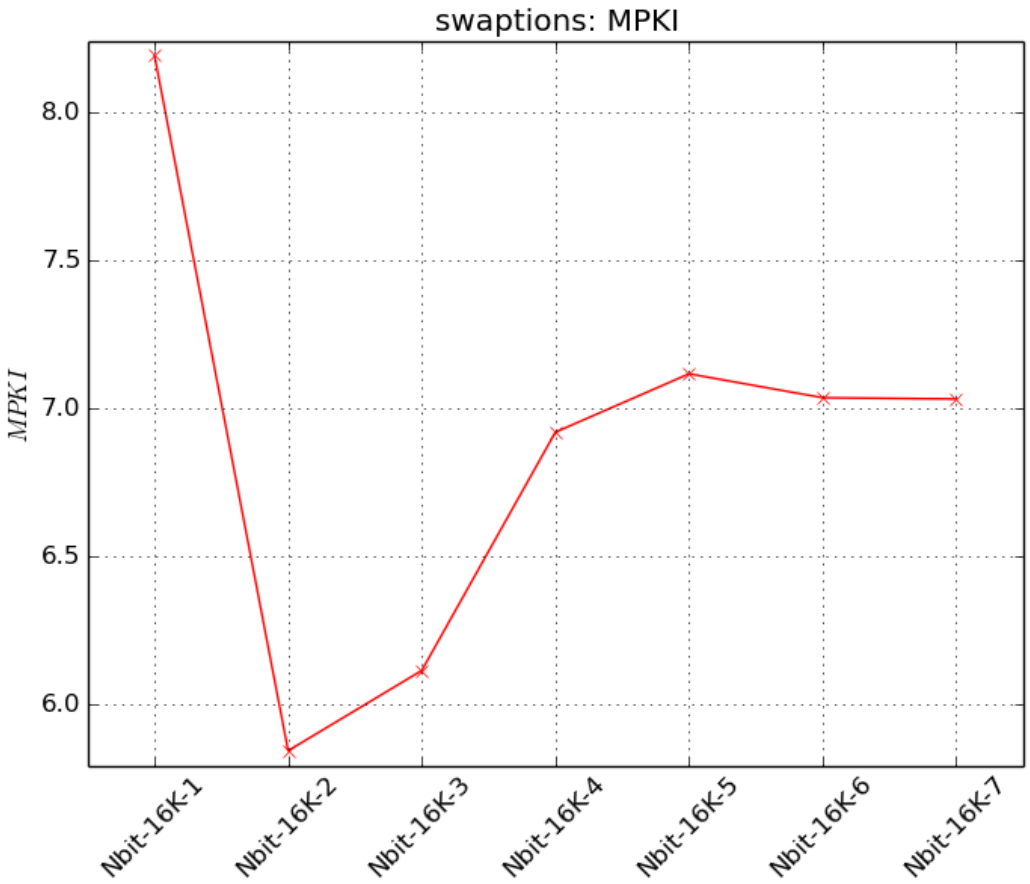

ferret: MPKI
MPKI
15
14
13
12
11
10
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
Nbit-16K-5
Nbit-16K-6
Nbit-16K-7


fluidanimate: MPKI
MPKI
18
17
16
15
14
13
12
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
Nbit-16K-5
Nbit-16K-6
Nbit-16K-7


freqmine: MPKI
MPKI
25
24
23
22
21
20
19
18
17
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
Nbit-16K-5
Nbit-16K-6
Nbit-16K-7


rtview: MPKI
MPKI
2.0
1.9
1.8
1.7
1.6
1.5
1.4
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
Nbit-16K-5
Nbit-16K-6
Nbit-16K-7


streamcluster: MPKI
MPKI
3.0
2.5
2.0
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
Nbit-16K-5
Nbit-16K-6
Nbit-16K-7

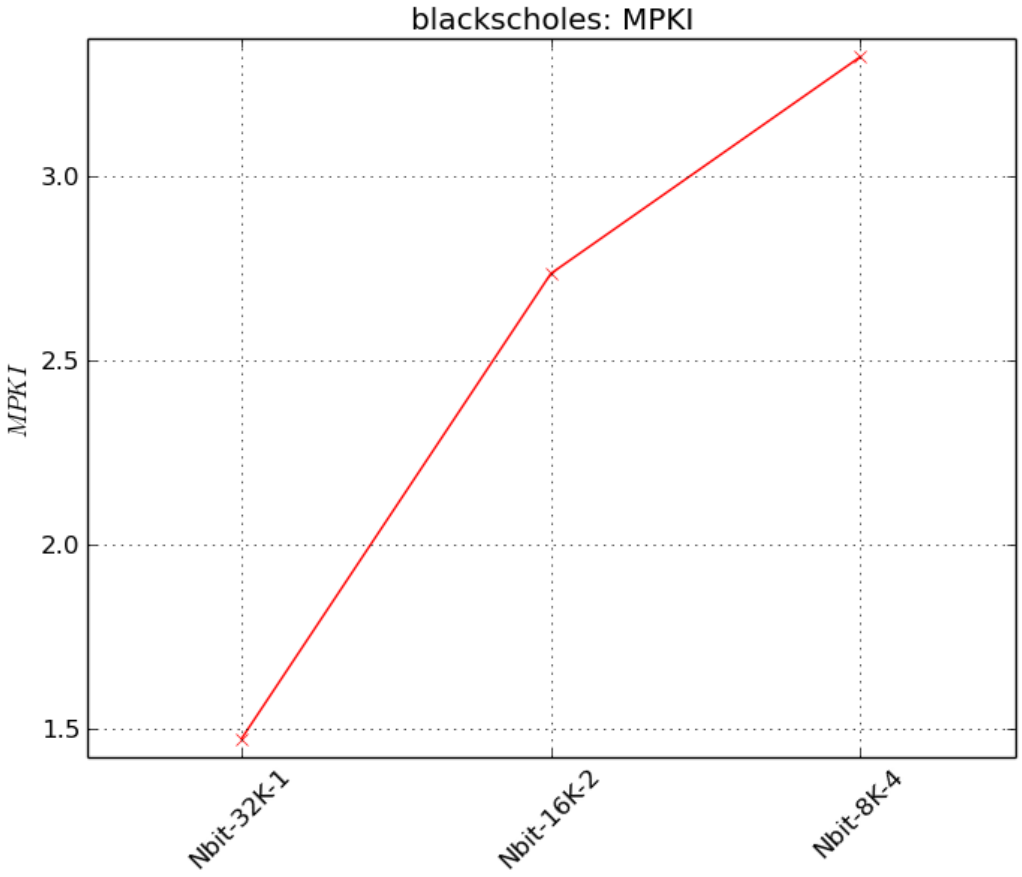


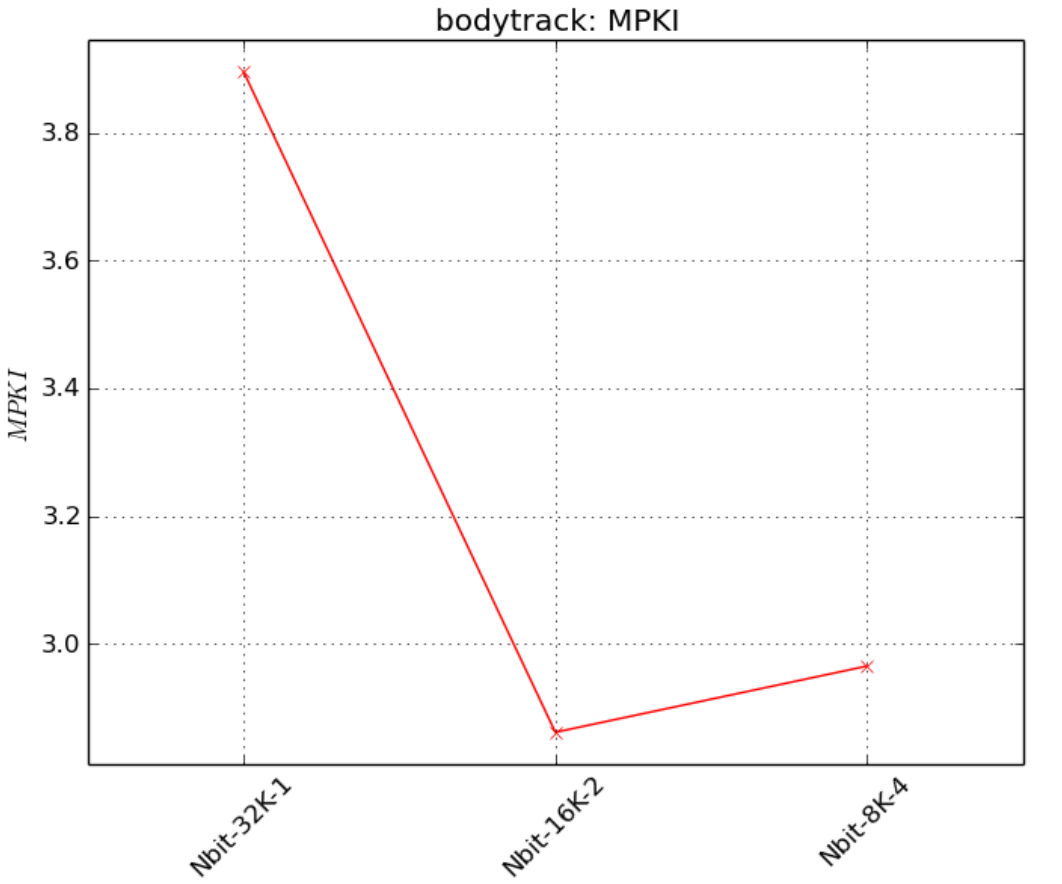
Παρατηρούμε πως ο αριθμός των bit επηρεάζει δραστικά την απόδοση του συστήματος σε κάθε benchmark. Φαίνεται πως μικρός αριθμός bit δίνει σχετικά μικρή απόδοση. Αυξάνοντας την τιμή του N πετυχαίνουμε καλύτερα αποτελέσματα. Αλλά δε πρέπει να ξεγνάμε πως με αυτό τον τρόπο αυξάνουμε συνεχώς το απαιτούμενο υλικό. Γενικά είναι εύκολο κανείς να συμπεράνει πως για μικρά N, έχουμε πιο γρήγορη μεταβολή στην τιμή της πρόβλεψης, Taken ή Not Taken. Αυτό ωστόσο δεν συνεπάγεται πως αυξάνεται και η απόδοση αφού σε ακολουθίες στις οποίες γίνεται συχνή εναλλαγή τιμών τα αποτελέσματα είναι τραγικά. Από την άλλη όσο αυξάνουμε το N πετυχαίνουμε όλο και πιο αργές μεταβάσεις. Συνεπώς αδυνατούμε να μαντέψουμε σωστά απομονωμένα branches τα οποία είναι Taken ή Not Taken αντίστοιχα. Για αυτό το λόγο σε διαγράμματα όπως του blackscholes βλέπουμε να ακολουθείται διαφορετική πορεία από τι στην πλειοψηφία των υπολοίπων. Τέλος Οι ιδανικές τιμές φαίνονται να αντιστοιχούν για N = 2 , 3 ή 4.

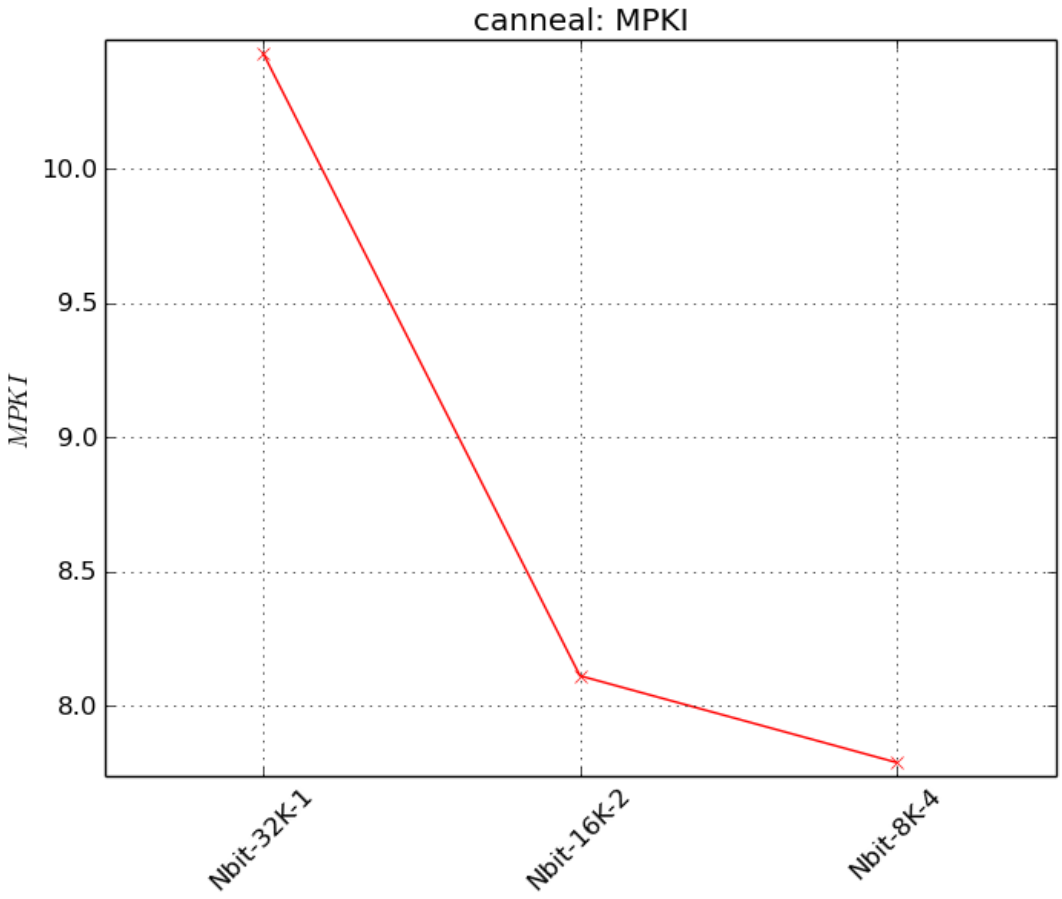
ΙΙ) Στη συνέχεια κρατάμε σταθερό το harware και ίσο με 32K bits. Προκειμένου να το πετύχουμε αυτό μειώνουμε τον αριθμό των entries. Έτσι εκτελούμε

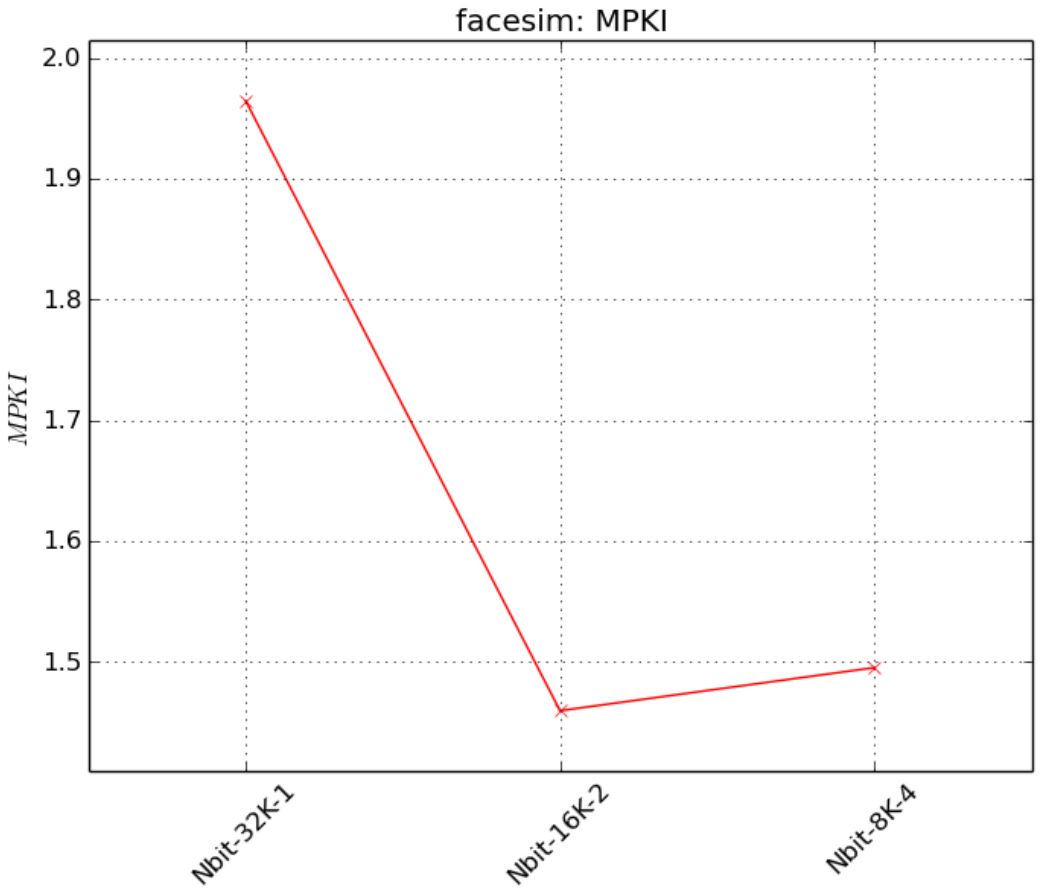
προσωμοιώσεις για τους παρακάτω N- bit predictors :

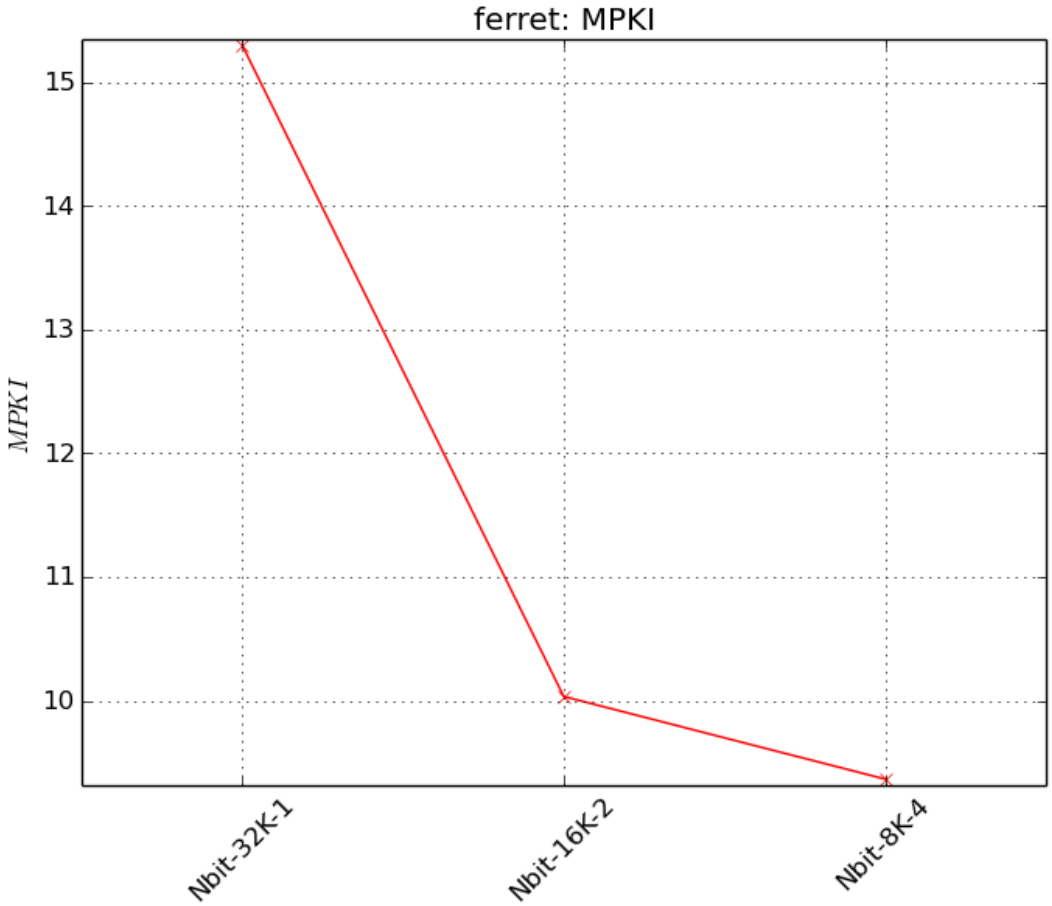
32K-1 , 16K- 2 , 8K -4 όπου το πρώτο νούμερο συμβολίζει τον αριθμό των BHT entries ενώ το δεύτερο το πλήθος των bit που χρησιμοποιούμε για τον κάθε predictor.

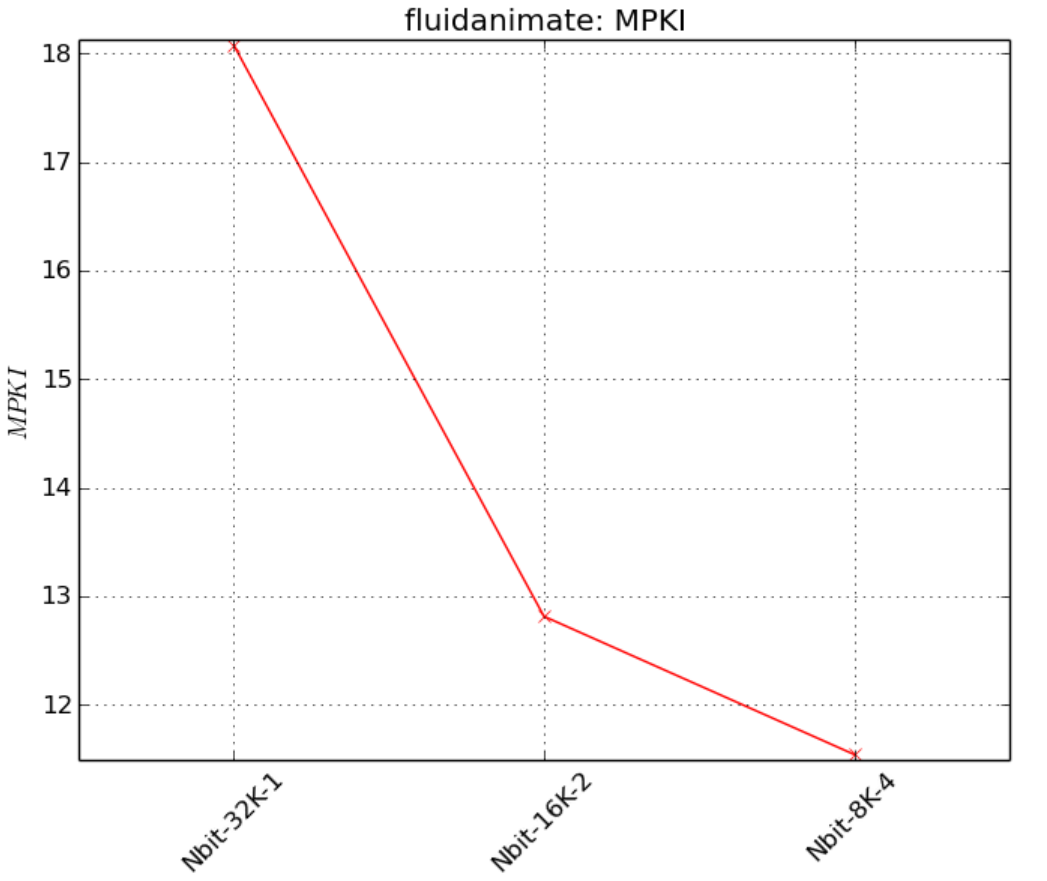


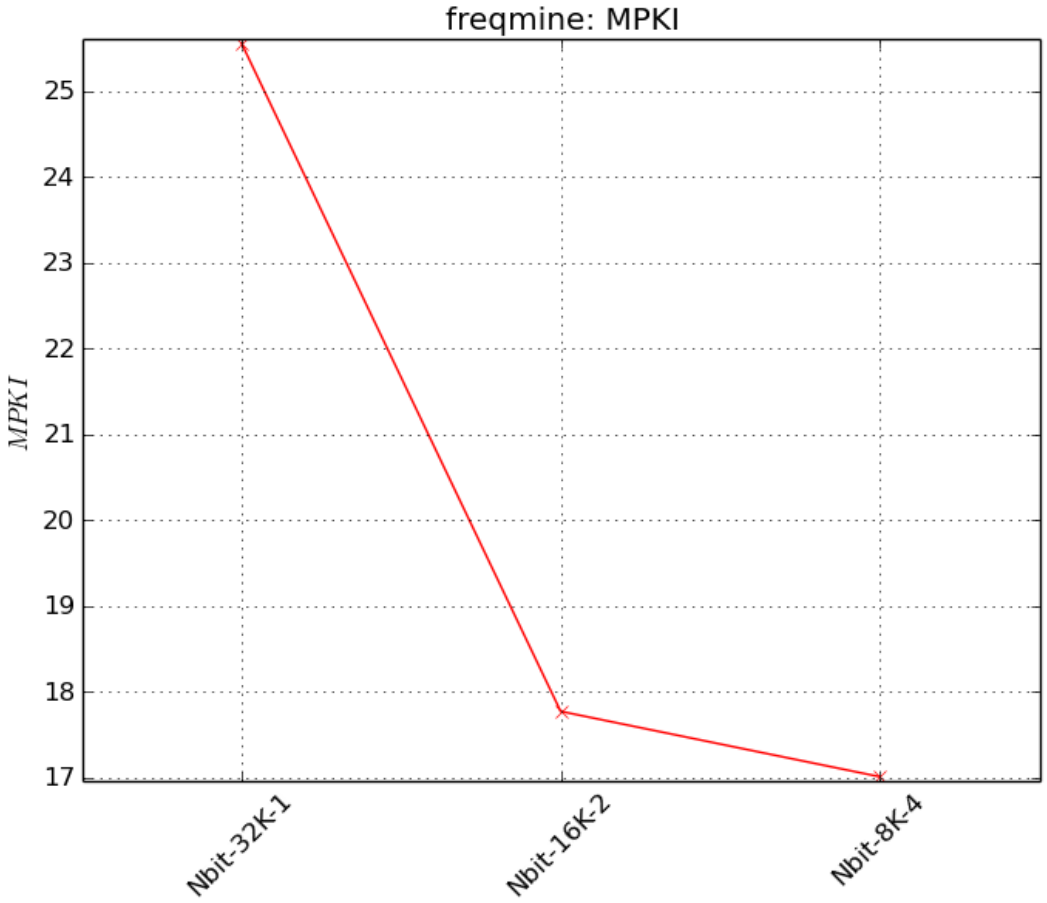

bodytrack: MPKI
MPKI
3.8
3.6
3.4
3.2
3.0
Nbit-32K-1
Nbit-16K-2
Nbit-8K-4

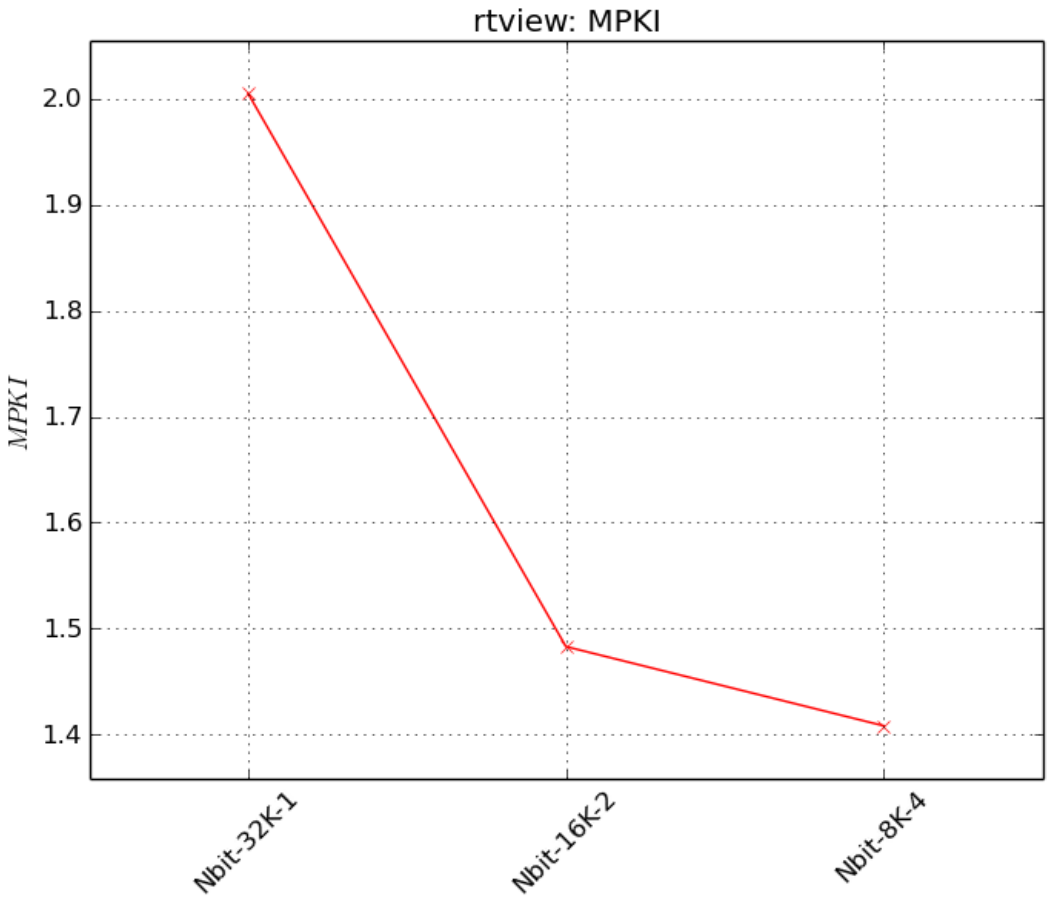

canneal: MPKI
MPKI
10.0
9.5
9.0
8.5
8.0
Nbit-32K-1
Nbit-16K-2
Nbit-8K-4

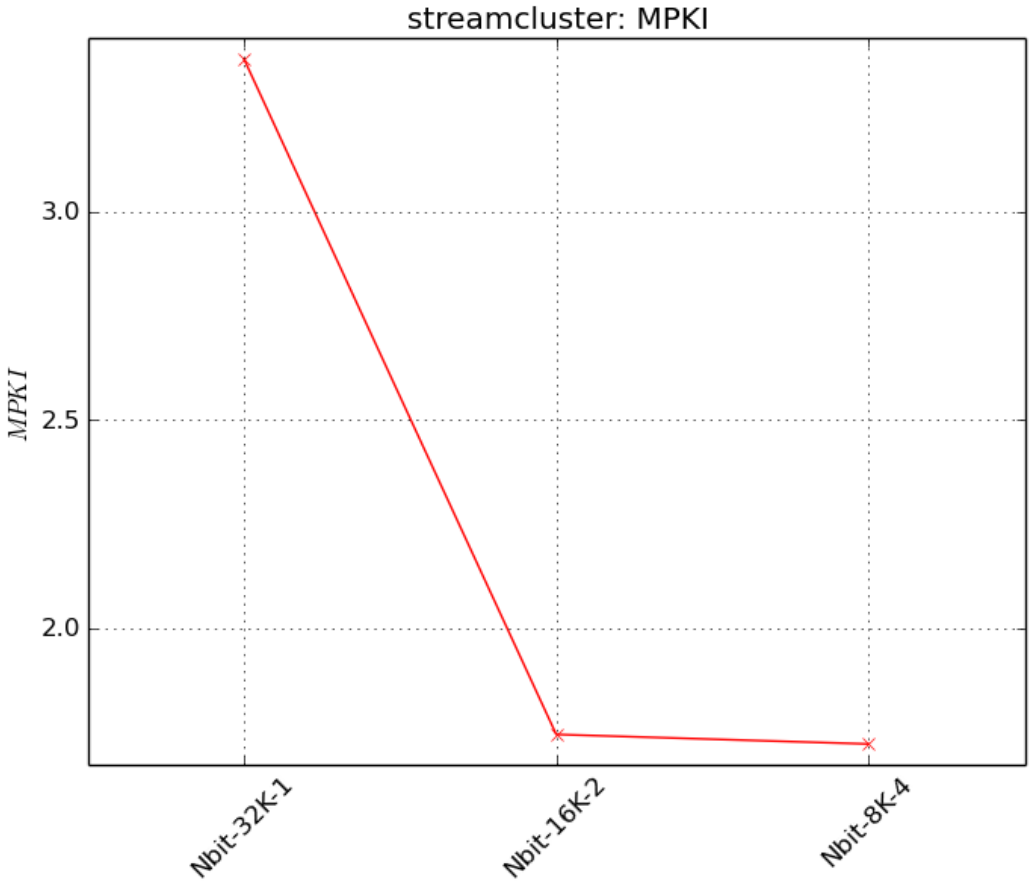

facesim: MPKI
2.0
1.9
1.8
1.7
1.6
1.5
MPKI
Nbit-32K-1
Nbit-16K-2
Nbit-8K-4

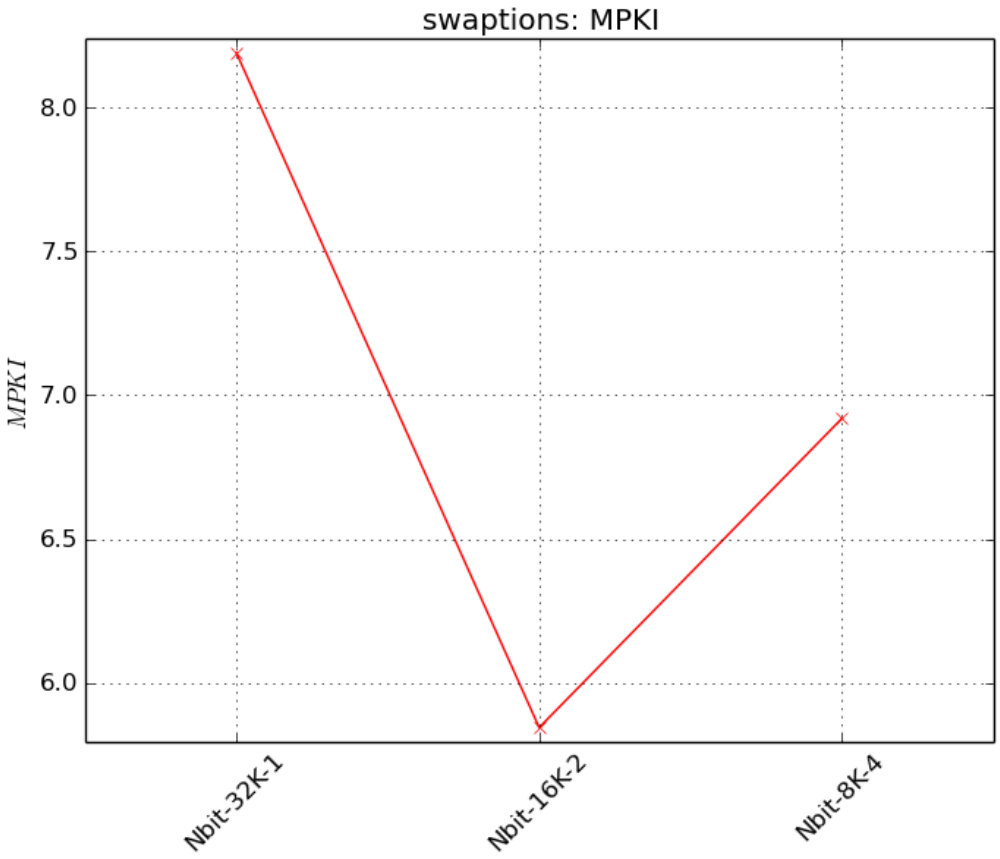

ferret: MPKI
15
14
13
12
11
10
MPKI
Nbit-32K-1
Nbit-16K-2
Nbit-8K-4


fluidanimate: MPKI
MPKI
18
17
16
15
14
13
12
Nbit-32K-1
Nbit-16K-2
Nbit-8K-4


freqmine: MPKI
MPKI
25
24
23
22
21
20
19
18
17
Nbit-32K-1
Nbit-16K-2
Nbit-8K-4


rtview: MPKI
MPKI
2.0
1.9
1.8
1.7
1.6
1.5
1.4
Nbit-32K-1
Nbit-16K-2
Nbit-8K-4


streamcluster: MPKI
MPKI
3.0
2.5
2.0
Nbit-32K-1
Nbit-16K-2
Nbit-8K-4



Στα περισσότερα των διαγραμμάτων βλέπουμε μια υπεροχή του 8K-4 με τον 16K-2 να ακολουθεί έχοντας το χαμηλότερο αριθμό misses σε αρκετά μετροπρογράμματα. Φαίνεται λοιπόν ότι το γεγονός πως έχουμε περισσότερα bits για τους predictors επιδρά θετικότερα σε σχέση με τα περισσότερα entries . Φυσικά αυτό ισχύει μέχρις ενός σημείου, όπου από κει και έπειτα ο αριθμός των entries έχει μειωθεί σημαντικά με αποτέλεσμα τη μείωση της απόδοσης λόγω χρησιμοποίησης του ίδιου entry από πολλές εντολές . Συνεπώς ο predictor που επιλέγουμε είναι 8K-4.

4.3 Μελέτη του BTB

Για διαφορετικές οργανώσεις της BTB cache συγκρίνουμε το πλήθος των λαθασμένων προβλέψεων branch καθώς και το πλήθος τον φορών που επιλέχτηκε η σωστή διεύθυνση target σε περίπτωση taken.

Παραθέτουμε επίσης την υλοποίηση του BTB Predictor. Η BTB cache υλοποιήθηκε

ως ένας τρισδιάστατος πίνακας . Η πρώτη διάσταση αντιπροσωπεύει το set στο οποίο αντιστοιχίζεται η κάθε εντολή branch βάση του Program Counter και το μήκος της καθορίζεται από τα BTB entries κάθε φορά. Η δεύτερη διάσταση αναπαριστά κάποια συγκεκριμένη καταχώρηση εντός του set ,και το μήκος αυτής αντιστοιχεί στο associativity που ορίστηκε. Τέλος η τελευταία διάσταση έχει σταθερό μέγεθος, δύο και το πρώτο κελί κρατά το PC ενός branch που έχει αποθηκευτεί στην cache , ενώ το δεύτερο τον προβλεπόμενο προορισμό διακλάδωσης.

Στη μέθοδο predict αναζητούμε εντός του set όπου αντιστοιχεί στην εντολή, το PC της και αν το βρούμε τότε η πρόβλεψη μας είναι Taken (True). Στην update διακρίνουμε 4 καταστάσεις. Αν η πρόβλεψή μας είναι NT και το ίδιο συμβεί κιόλας τότε δεν χρειάζεται να τροποποιήσουμε τίποτα. Σε περίπτωση που έχουμε προβλέψει NT και τελικά γίνει το branch τότε πρέπει να καταχωρίσουμε την εντολή στο BTB buffer. Διαφορετικά αν έχουμε προβλέψει taken και αυτό δε συμβεί τότε διαγράφουμε την καταχώρηση αυτή από το buffer. Τέλος αν προβλέψαμε σωστά και έχουμε Taken αλλά προς διαφορετικό προορισμό τότε ανανεώνουμε το target για τη συγκεκριμένη καταχώριση. Σε κάθε περίπτωση ανανεώνουμε τους μετρητές Correct , Incorrect και TargetCorrect.

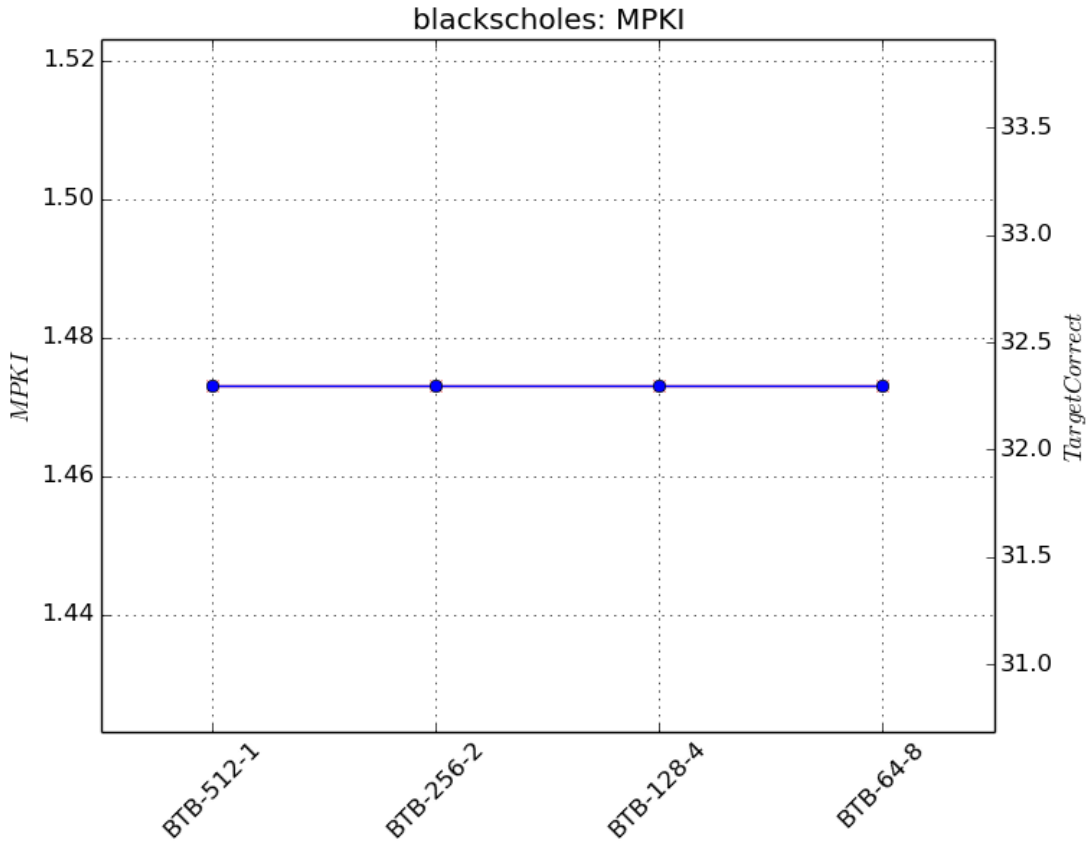
```
class BTBPredictor : public BranchPredictor
{
public:
        BTBPredictor(int btb_lines, int btb_assoc)
           : table_lines(btb_lines), table_assoc(btb_assoc)
        {
                correctTarget_predictions=0;
                TABLE= new unsigned long long**[table_lines];
                if (!TABLE) {printf("error"); exit(1);}
                for(int i=0; i<table_lines; i++){
                    TABLE[i]= new unsigned long long * [table_assoc];
```

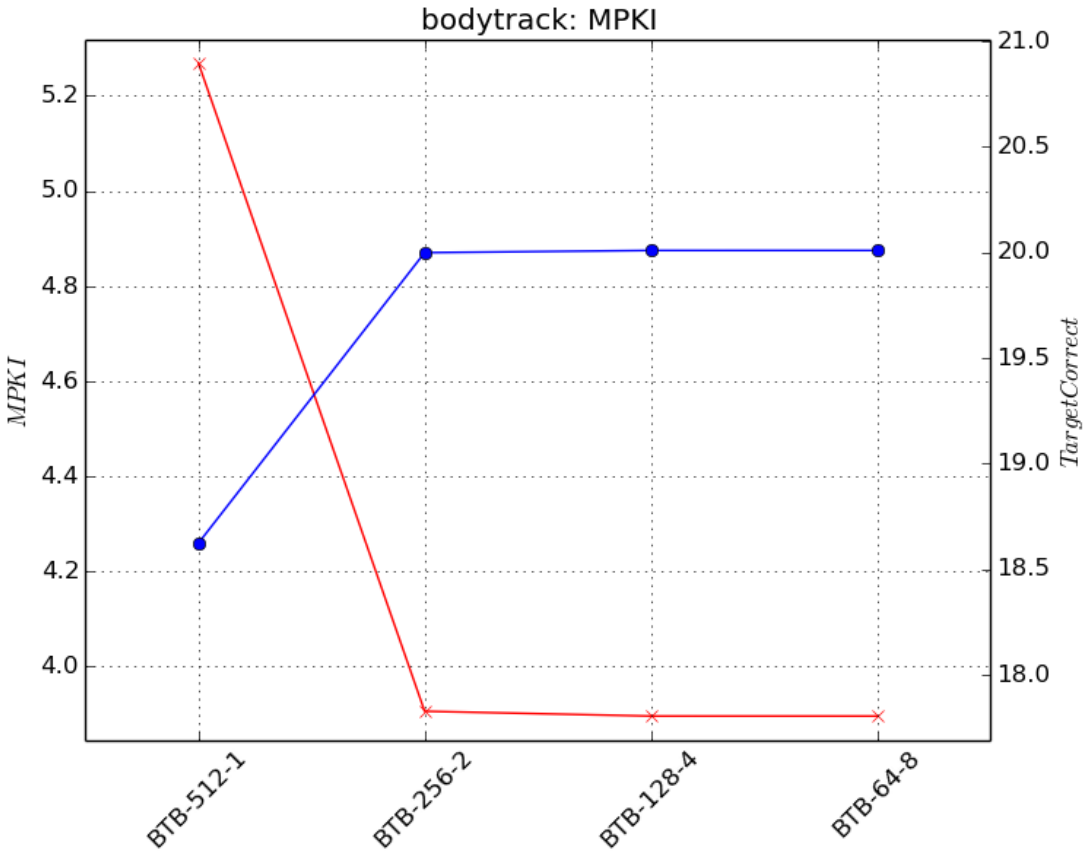
```
                    if (!TABLE[i]) {printf("error"); exit(1);}
                    for (int j=0; j<table_assoc; j++){
                                TABLE[i][j]= new unsigned long long[2];
                                if (!TABLE[i][j]) {printf("error"); exit(1);}
                                memset(TABLE[i][j],0, 2 * sizeof(TABLE[0][0][0]));
                    } }}
        ~BTBPredictor() {
                for (int i=0; i<table_lines; i++) {
                        for (int j=0; j<table_assoc; j++)
                                delete TABLE[i][j];
                        delete TABLE[i];
                }
                delete TABLE; }
virtual bool predict(ADDRINT ip, ADDRINT target) {
        unsigned int ip_table_index = ip % table_lines;
        for (int i=0; i<table_assoc; i++){
                if (TABLE[ip_table_index][i][0] == ip)
                        return true;
        }
   return false;
}
virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {
                updateCounters(predicted,actual);
                unsigned int ip_table_index = ip % table_lines;
                if (actual &&!predicted){ //branch taken but not predicted,
                      //insert branch and it's destination into the btb cache
                        for (int i=0; i<table_assoc; i++){
                                if (TABLE[ip_table_index][i][0]==0) {
                                        TABLE[ip_table_index][i][0]=ip;
```

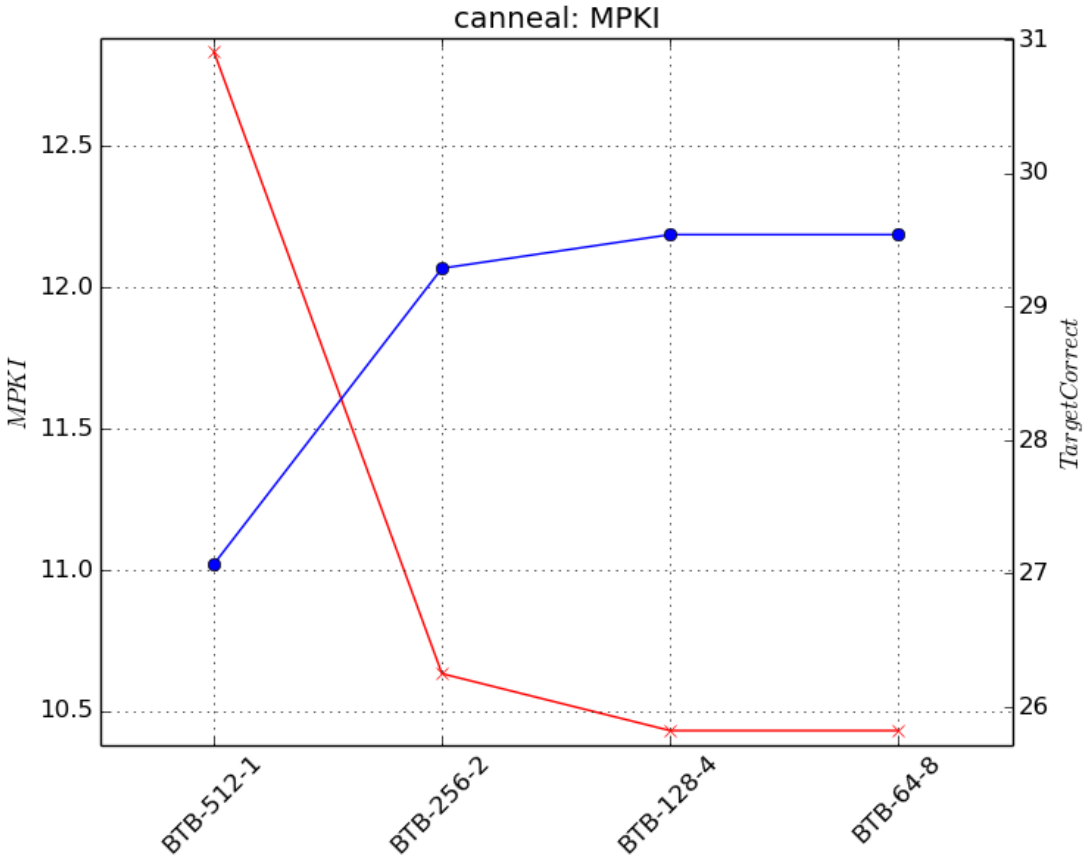
```
                                        TABLE[ip_table_index][i][1]=target;
                                        return ;
                    }}
                        int replace = std::rand() % table_assoc;
                        TABLE[ip_table_index][replace][0]=ip;
                        TABLE[ip_table_index][replace][1]=target;
                }
                else if (!actual &&predicted){ //
                        for (int i=0; i<table_assoc; i++){
                                if (TABLE[ip_table_index][i][0]==ip) {
                                        TABLE[ip_table_index][i][0]=0;
                                        TABLE[ip_table_index][i][1]=0;
                                        return ;
                                }}}
                else if (actual &&predicted){
                        for (int i=0; i<table_assoc; i++){
                                if (TABLE[ip_table_index][i][0]==ip) {
                                        if (TABLE[ip_table_index][i][1]==target)
                                                correctTarget_predictions++;
                                        else
                                                TABLE[ip_table_index][i][1]=target;
                                        return;
                                }}}}
  virtual string getName() {
    std::ostringstream stream;
                stream <<"BTB-"<<table_lines <<"-"<<table_assoc;
                return stream.str();
}
  UINT64 getNumCorrectTargetPredictions() {
```

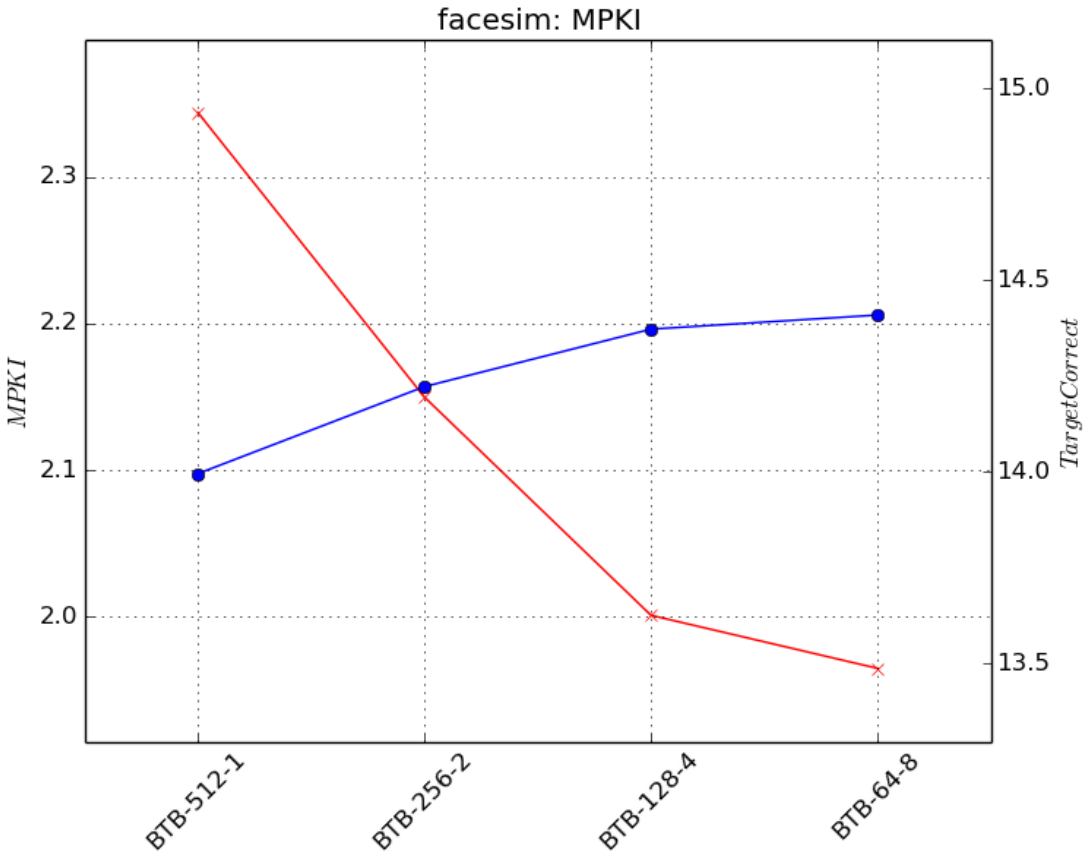
```
                return correctTarget_predictions;
        }
private:
        int table_lines, table_assoc;
        unsigned long long ***TABLE;
    UINT64 correctTarget_predictions;
};
```

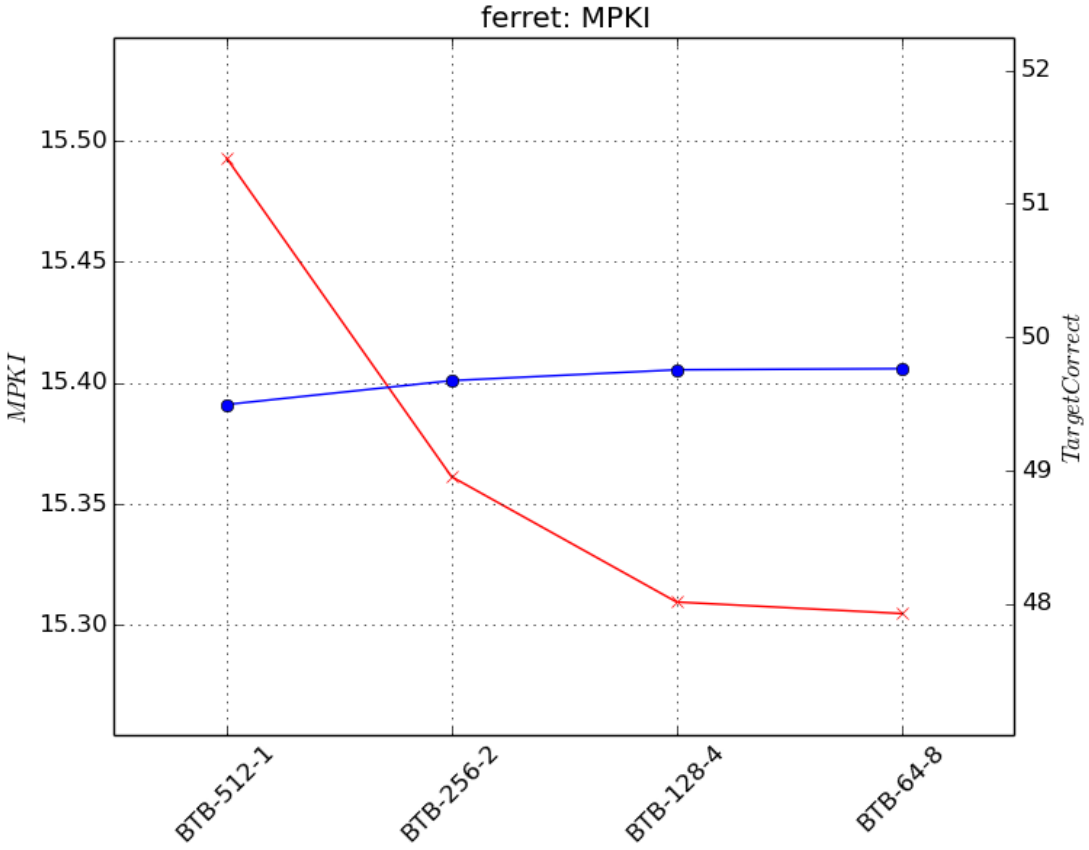
Ακολουθούν τα διαγράμματα όπου αναπαρίστανται με κόκκινο τα MPKI και με μπλε τα Target Correct ανά χίλιες εντολές.

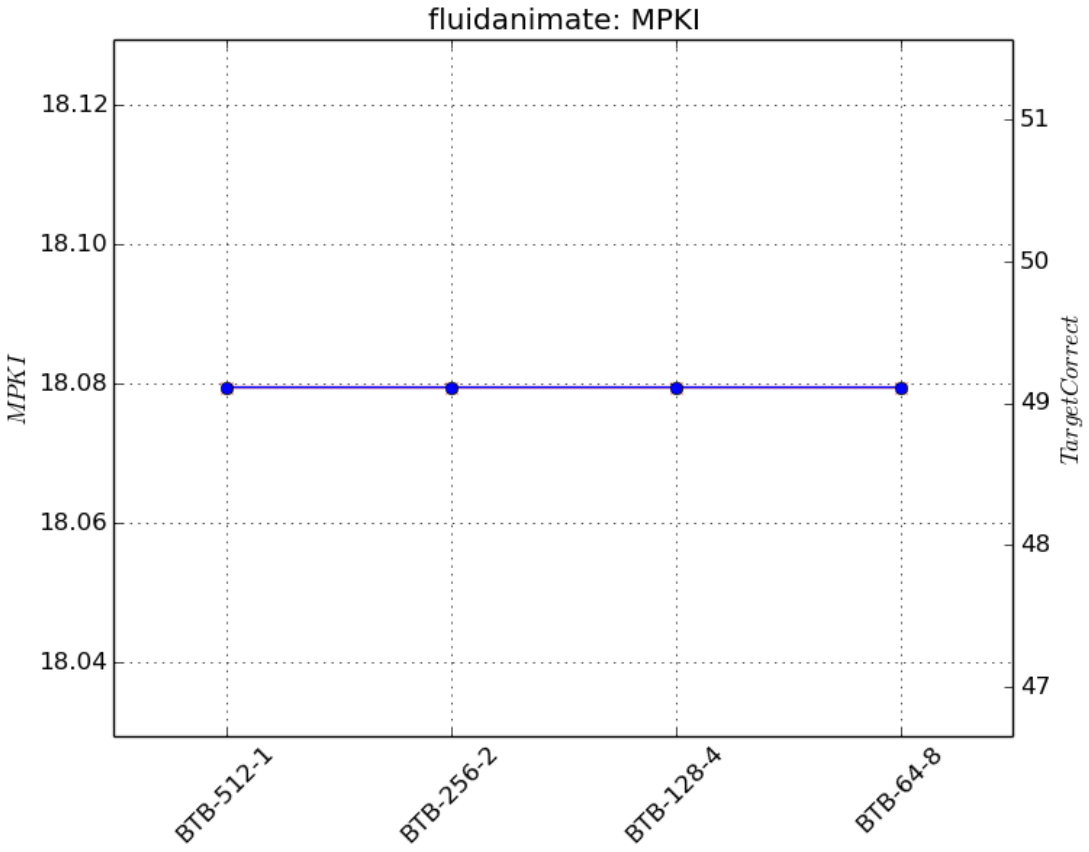


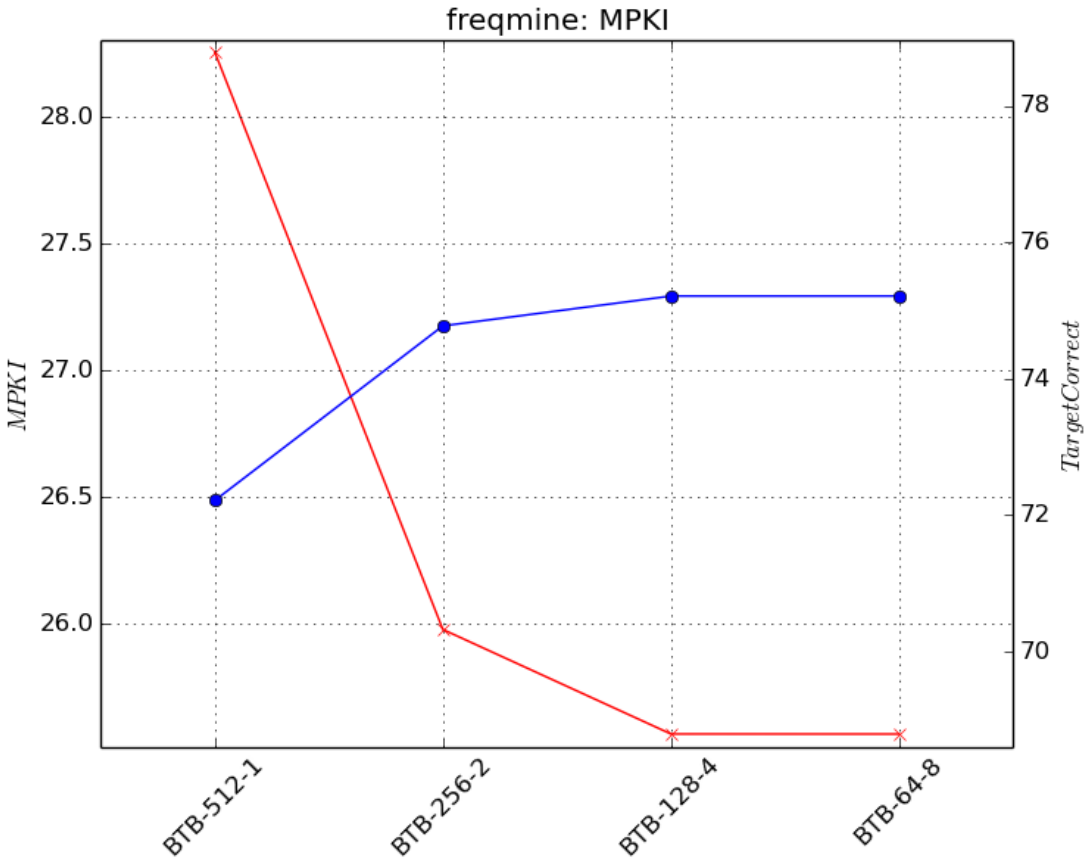

bodytrack: MPKI
MPKI
TargetCorrect
5.2
5.0
4.8
4.6
4.4
4.2
4.0
21.0
20.5
20.0
19.5
19.0
18.5
18.0
BTB-512-1
BTB-256-2
BTB-128-4
BTB-64-8

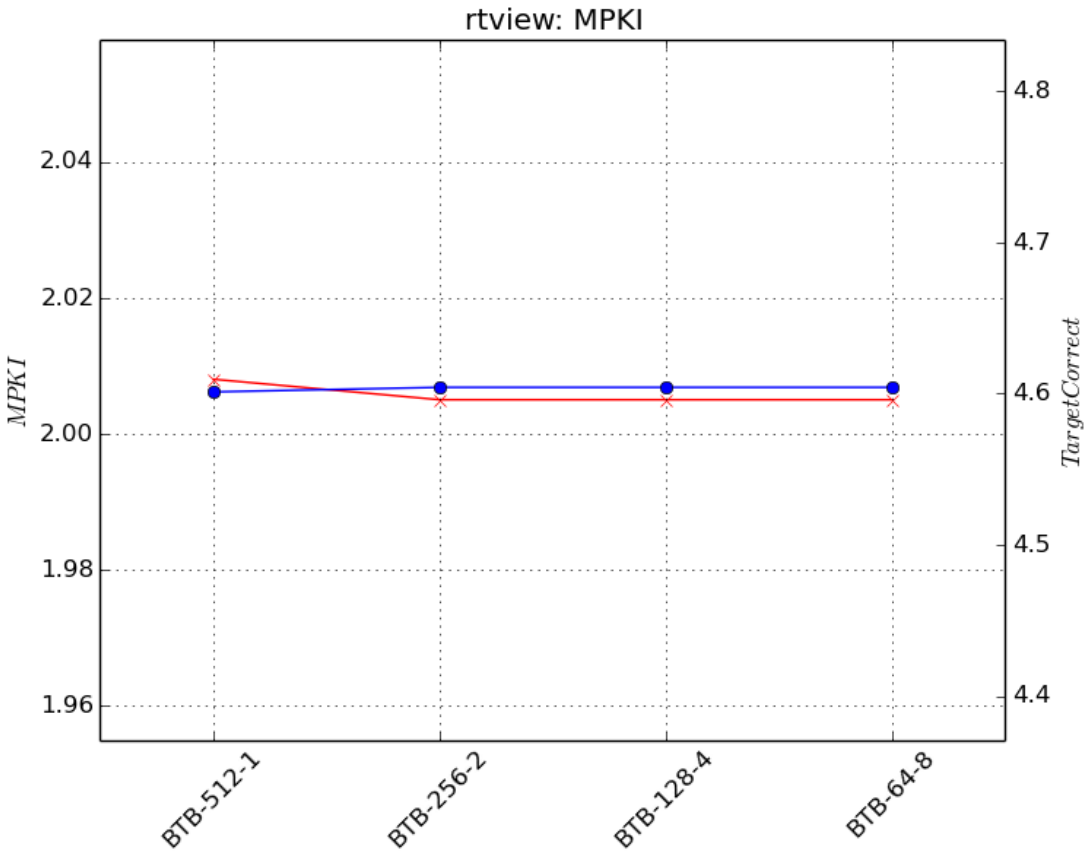

canneal: MPKI
MPKI
TargetCorrect
12.5
12.0
11.5
11.0
10.5
31
30
29
28
27
26
BTB-512-1
BTB-256-2
BTB-128-4
BTB-64-8

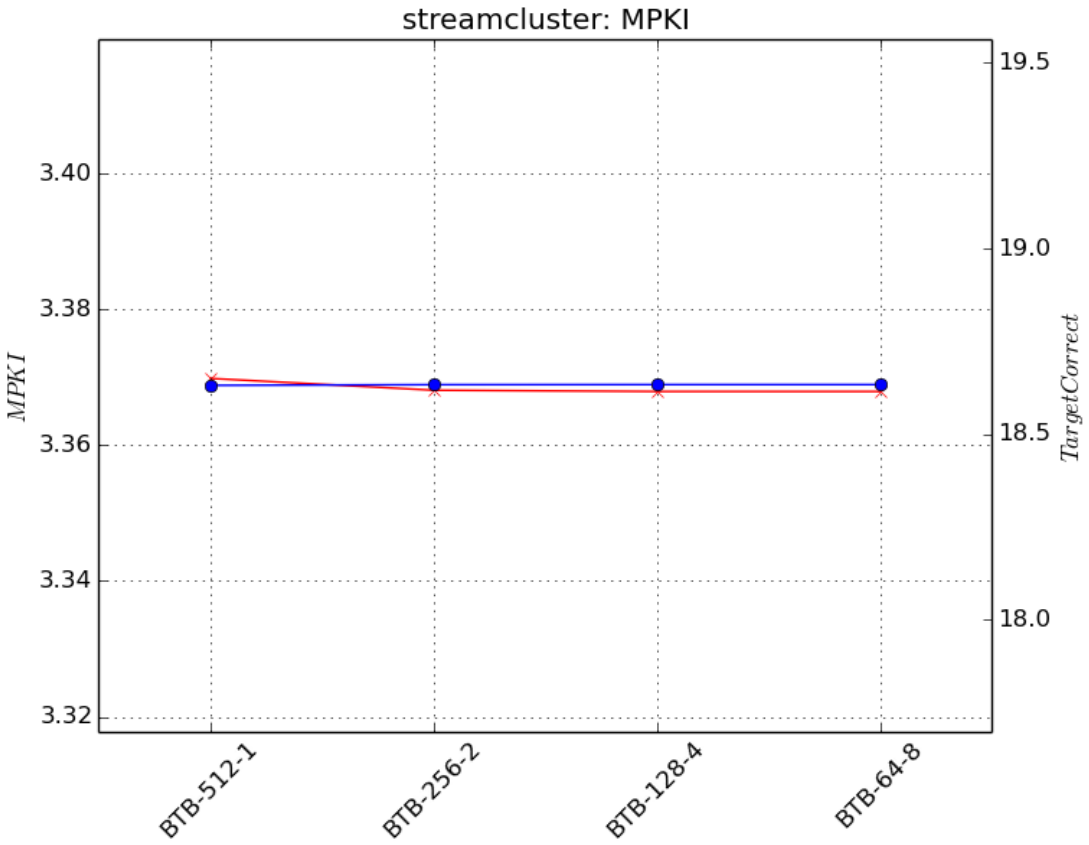

facesim: MPKI
MPKI
TargetCorrect
2.3
2.2
2.1
2.0
15.0
14.5
14.0
13.5
BTB-512-1
BTB-256-2
BTB-128-4
BTB-64-8

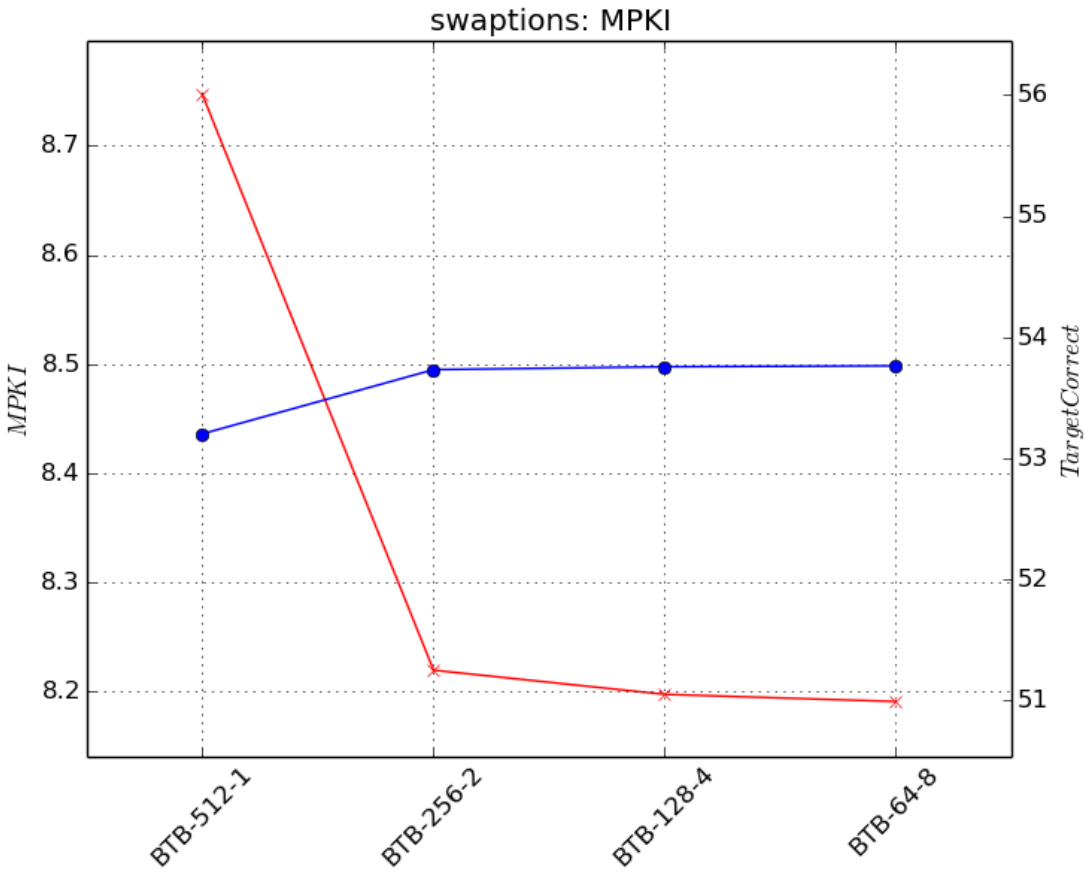

ferret: MPKI
MPKI
TargetCorrect
15.50
15.45
15.40
15.35
15.30
52
51
50
49
48
BTB-512-1
BTB-256-2
BTB-128-4
BTB-64-8


fluidanimate: MPKI
18.12
18.10
18.08
18.06
18.04
MPKI
51
50
49
48
47
TargetCorrect
BTB-512-1
BTB-256-2
BTB-128-4
BTB-64-8


freqmine: MPKI
28.0
27.5
27.0
26.5
26.0
MPKI
78
76
74
72
70
TargetCorrect
BTB-512-1
BTB-256-2
BTB-128-4
BTB-64-8


rtview: MPKI
MPKI
TargetCorrect
2.04
2.02
2.00
1.98
1.96
4.8
4.7
4.6
4.5
4.4
BTB-512-1
BTB-256-2
BTB-128-4
BTB-64-8

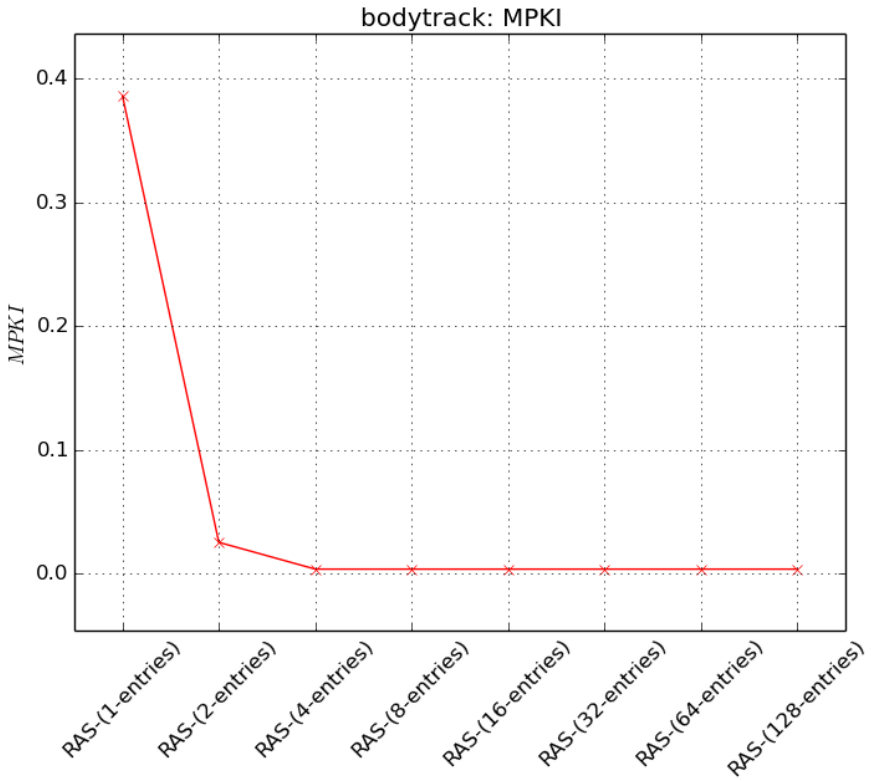

streamcluster: MPKI
MPKI
TargetCorrect
3.40
3.38
3.36
3.34
3.32
19.5
19.0
18.5
18.0
BTB-512-1
BTB-256-2
BTB-128-4
BTB-64-8

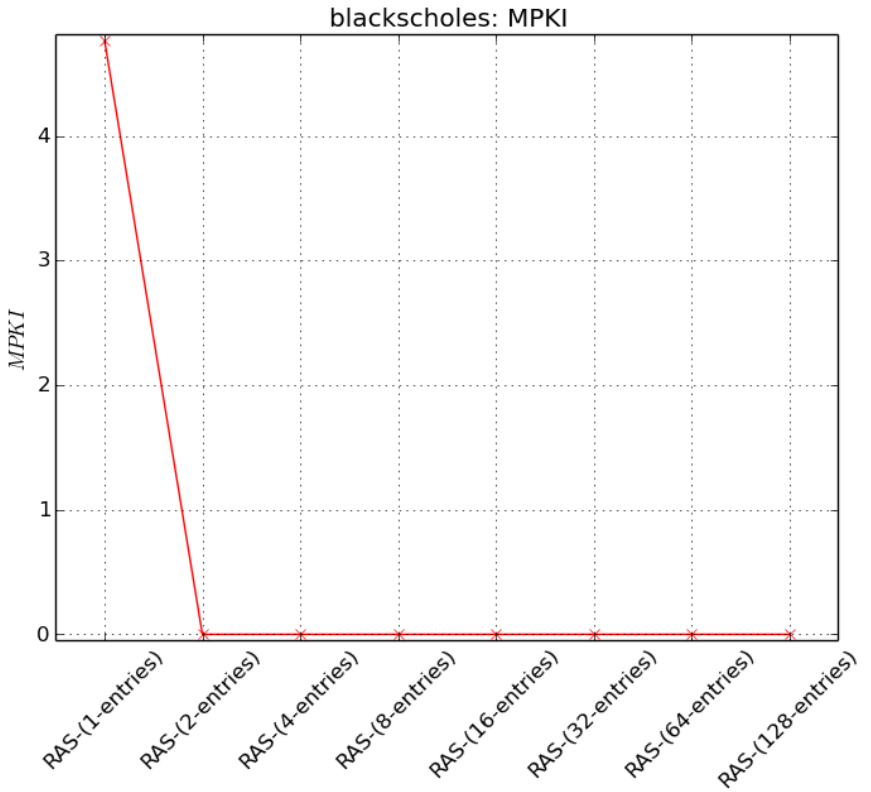


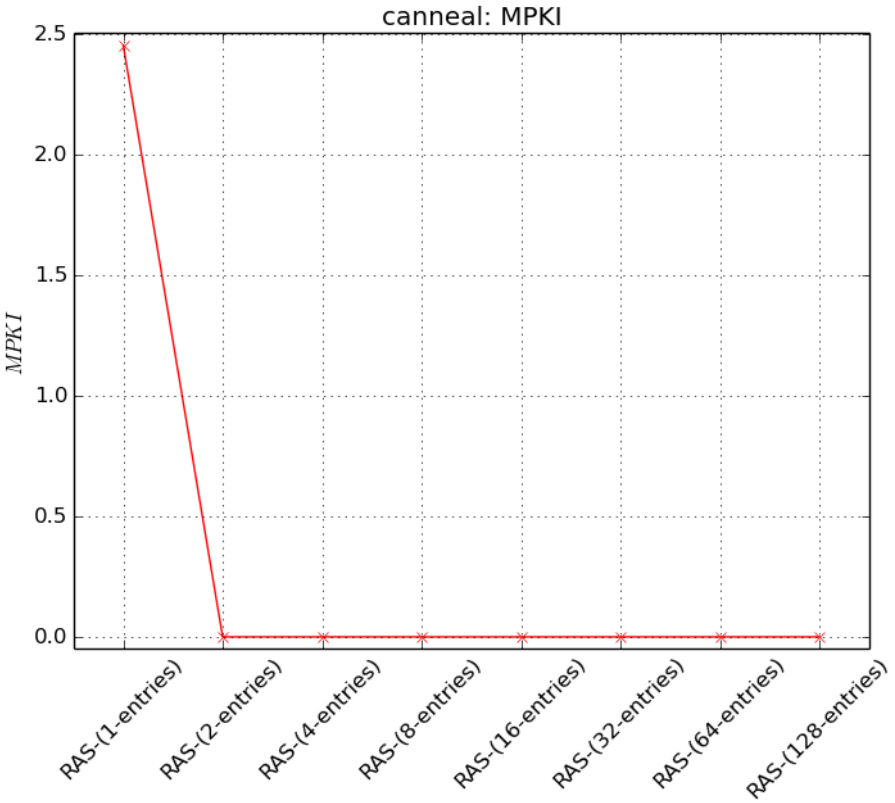
Γενικά παρατηρούμε πως πέρα της περίπτωσης όπου έχουμε οργάνωση μνήμης Direct Mapping , τα misses μειώνονται αισθητά . Οι διαφορές μεταξύ των υπολοίπων οργανώσεων είναι αρκετά μικρές , όμως παρατηρούμε ότι η αύξηση του associativity επιδρά θετικά μειώνοντας τα misses και κατά συνέπεια έχουμε και μια μικρή αύξηση στα TargetCorrect . (Κυρίως λόγο του γεγονότος ότι μειώνονται τα conflicts και έτσι χρήσιμες εγγραφές παραμένουν στη μνήμη).

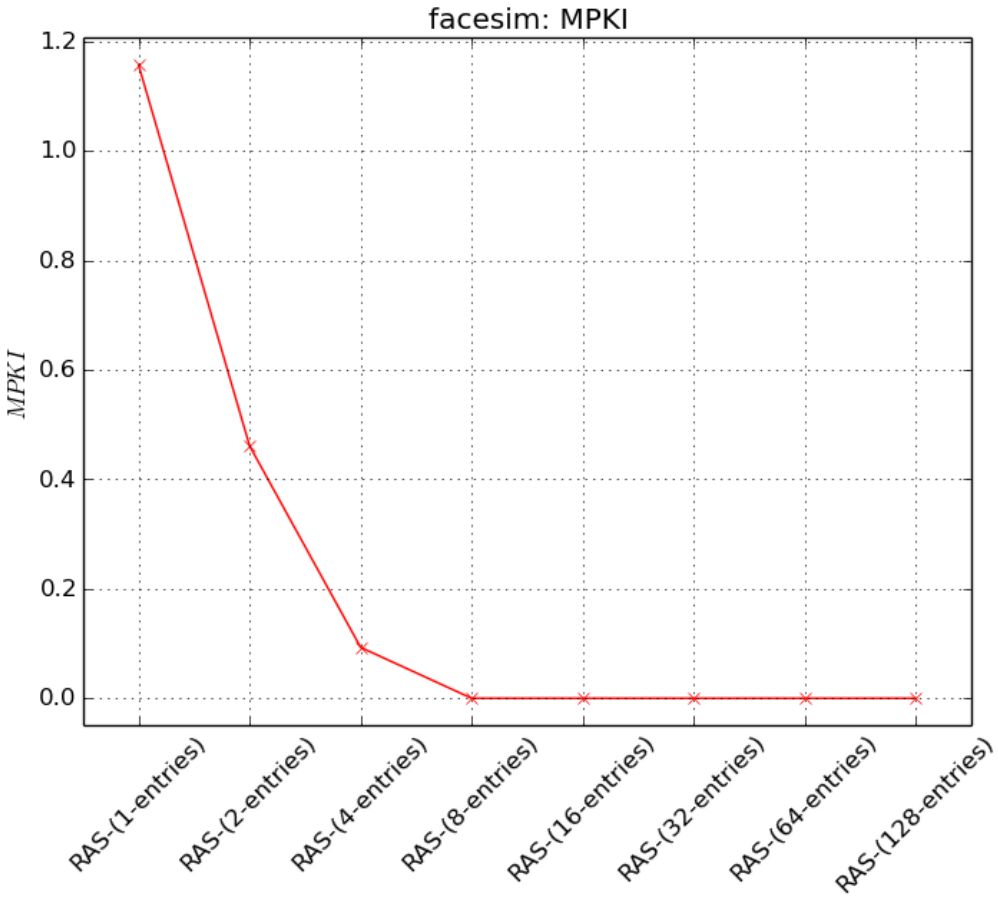
4.4 Μελέτη του RAS

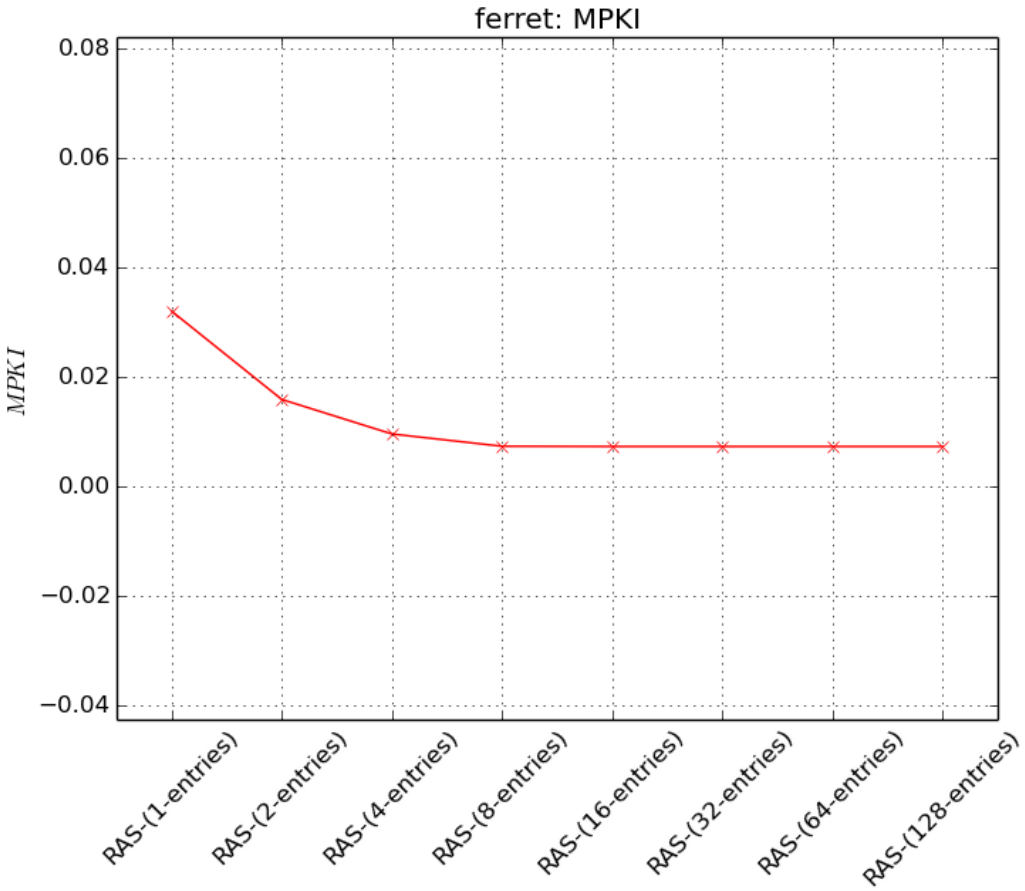
Σ' αυτό το σκέλος της άσκησης στοχεύουμε στις εντολές return. Έτσι μεταβάλουμε το πλήθος των εγγραφών της στοίβας όπου κρατιούνται οι διευθύνσεις επιστροφής των συναρτήσεων. Περιμένουμε μετροπρογράμματα όπως το blackscholes που παρουσίασαν μεγάλο πλήθος τέτοιων εντολών να έχουν και τις μεγαλύτερες διαφοροποιήσεις. Ως μέτρο σύγκρισης επιλέχτηκαν και εδώ τα MPKI.

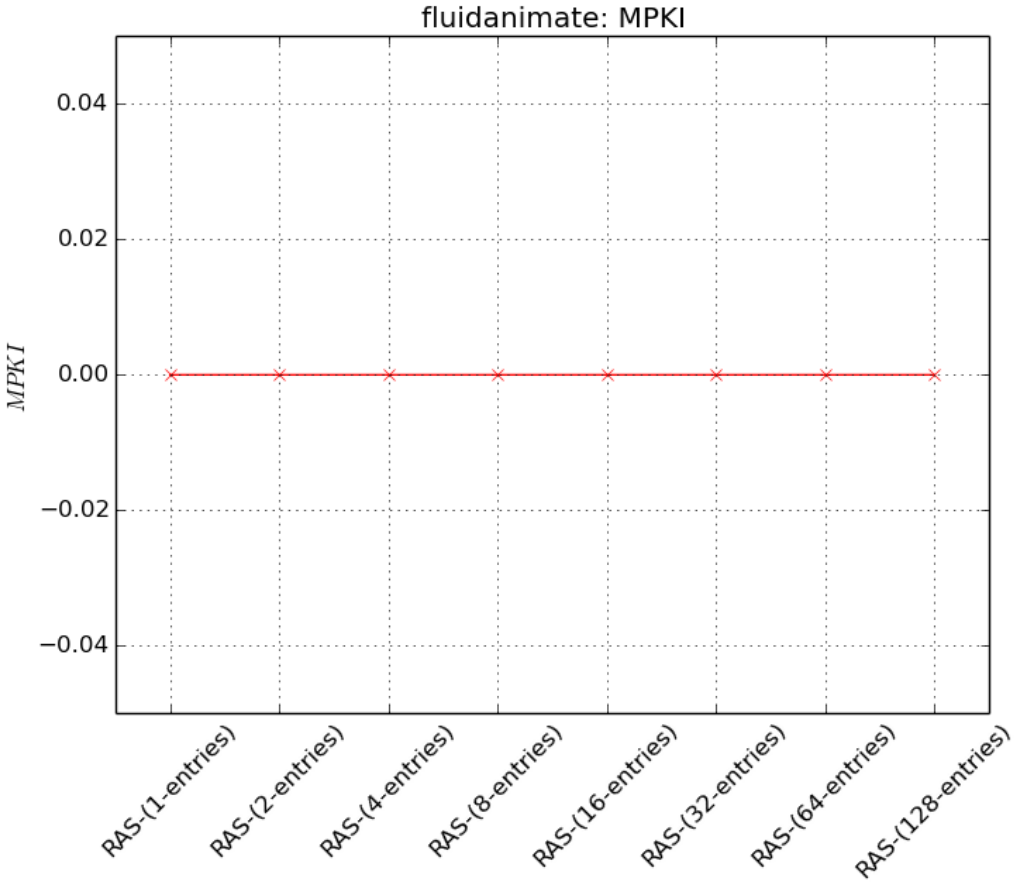

bodytrack: MPKI
MPKI
0.4
0.3
0.2
0.1
0.0
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)

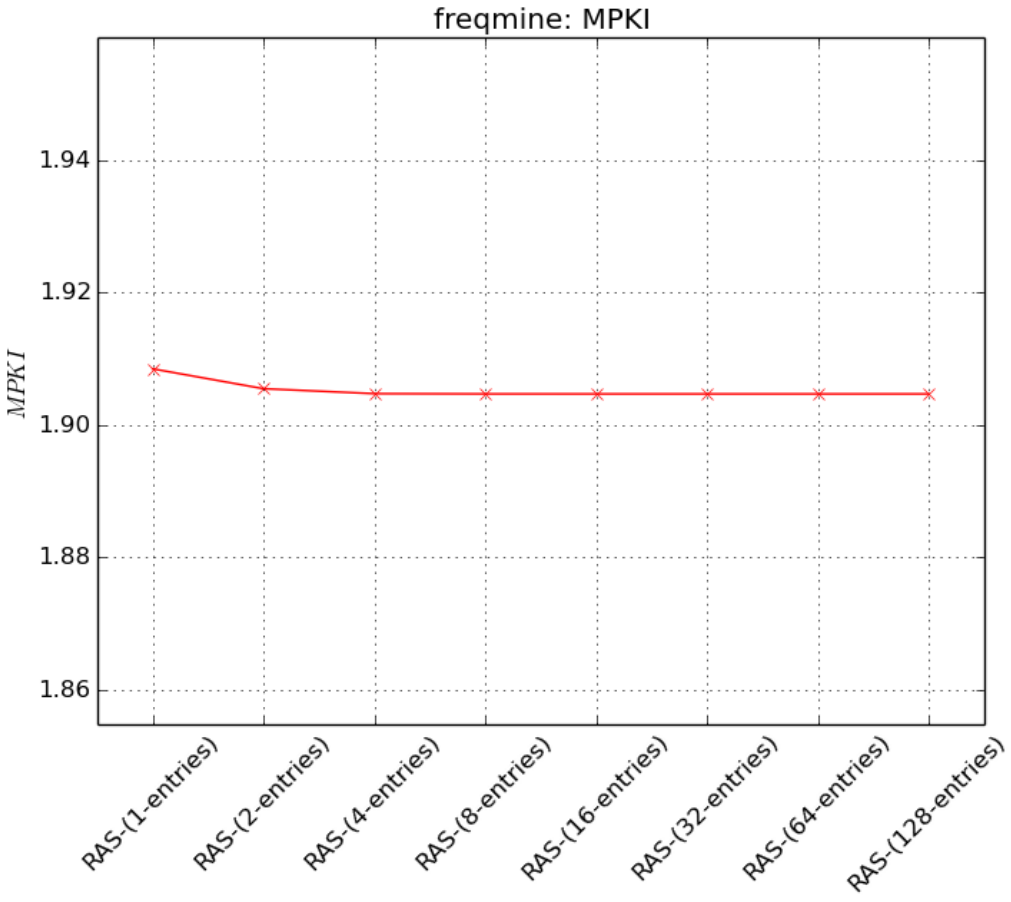

blackscholes: MPKI
MPKI
4
3
2
1
0
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)

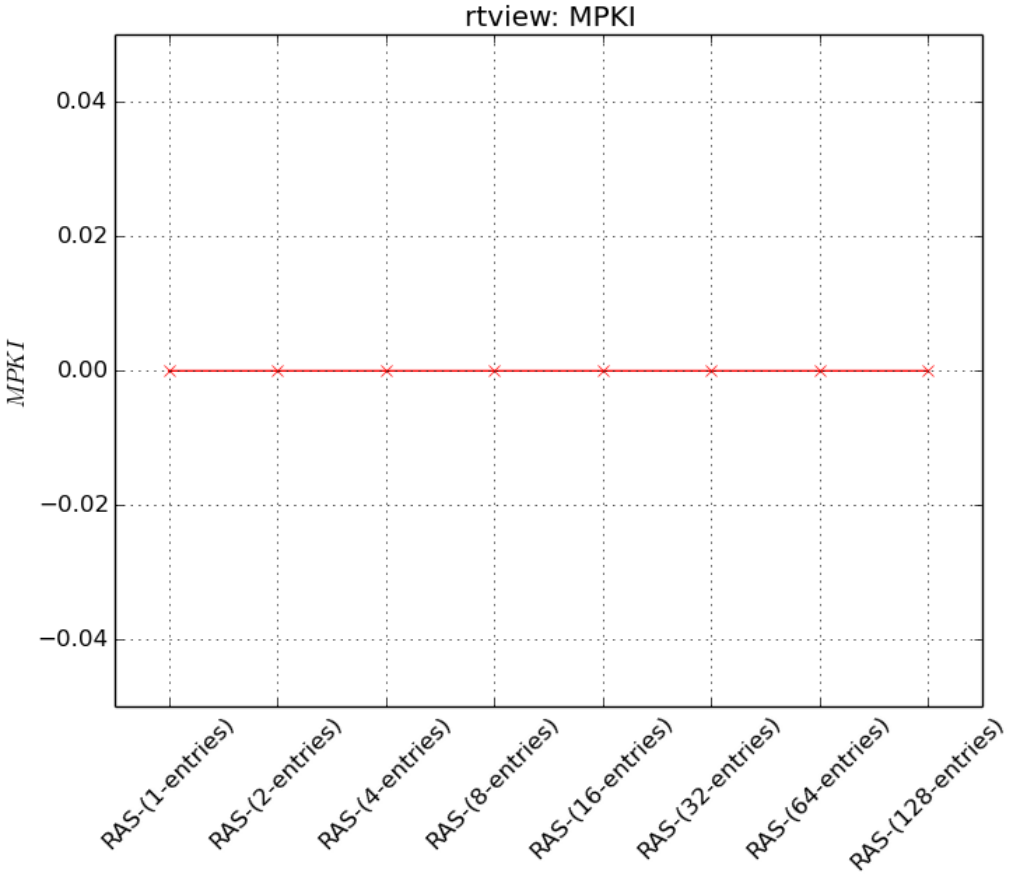

canneal: MPKI
MPKI
2.5
2.0
1.5
1.0
0.5
0.0
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)

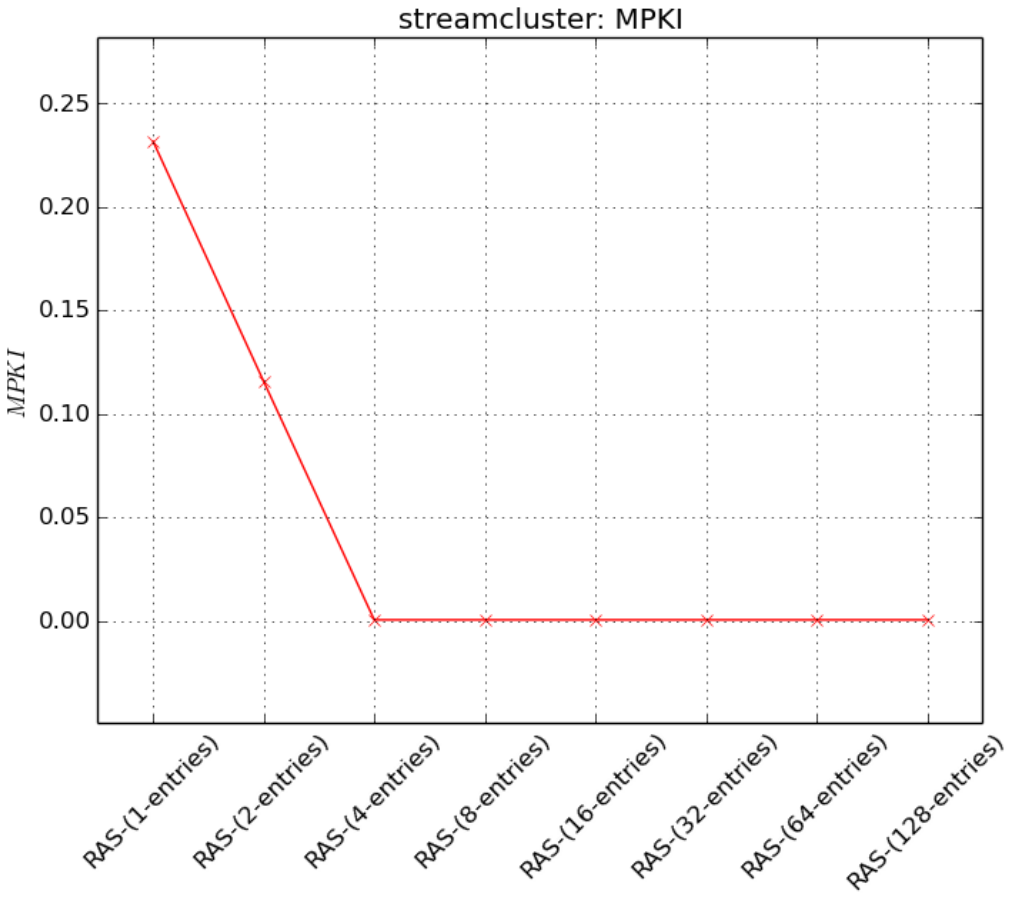

facesim: MPKI
MPKI
1.2
1.0
0.8
0.6
0.4
0.2
0.0
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)

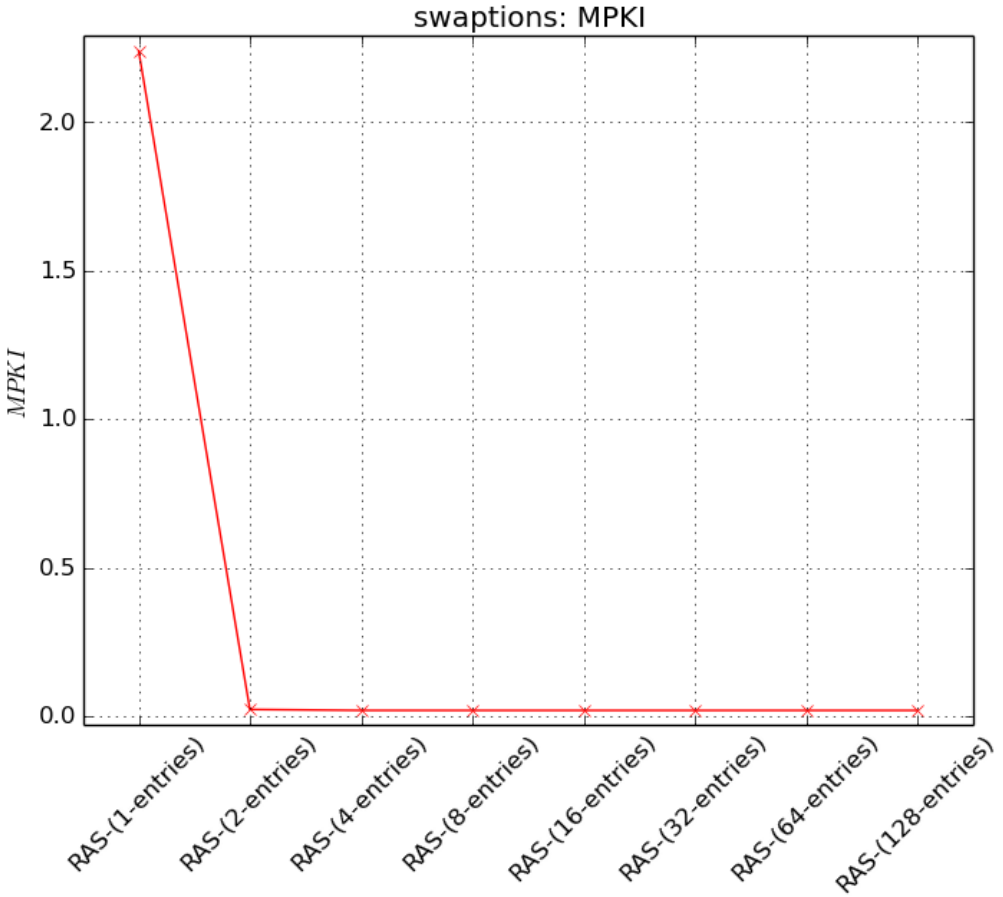

ferret: MPKI
MPKI
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)


fluidanimate: MPKI
MPKI
0.04
0.02
0.00
−0.02
−0.04
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)


freqmine: MPKI
MPKI
1.94
1.92
1.90
1.88
1.86
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)


rtview: MPKI
MPKI
0.04
0.02
0.00
−0.02
−0.04
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)


streamcluster: MPKI
MPKI
0.25
0.20
0.15
0.10
0.05
0.00
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)


swaptions: MPKI
MPKI
2.0
1.5
1.0
0.5
0.0
RAS-(1-entries)
RAS-(2-entries)
RAS-(4-entries)
RAS-(8-entries)
RAS-(16-entries)
RAS-(32-entries)
RAS-(64-entries)
RAS-(128-entries)

Εν τέλει παρατηρούμε πως σε όλα τα μετροπρογράμματα ακόμα και σε αυτά με πολύ μεγάλο αριθμό από misses στην περίπτωση της μίας εγγραφή για την Ras , αρκεί η αύξηση των εγγραφών σε 2 για την ραγδαία βελτίωση της απόδοσης. Από τις 4 εγγραφές και μετά δεν παρατηρούμε ιδιαίτερες διακυμάνσεις , με την απόδοση να έχει σταθεροποιηθεί .(Σχεδόν μηδενικός αριθμός από misses στις περισσότερες των περιπτώσεων.)

4.5 Σύγκριση διαφορετικών predictors

Static Not-Taken , Static BTNFT

Στο τελευταίο σκέλος ζητήθηκε να γίνει μια σύγκριση πολλών διαφορετικών predictors. Στην αρχή υλοποιήσαμε τους δύο στατικούς Not-taken και BTFNT. Στον πρώτο η πρόβλεψη είναι πάντα false ενώ στον δεύτερο συγκρίνουμε τη διεύθυνση προορισμού, με το PC και αν προηγείται αυτού η πρόβλεψη είναι false ειδάλλως true.

Local-History

Στη συνέχεια υλοποιήσαμε τον Local-history two levels predictors. Ο συγκεκριμένος predictor οργανώθηκε με δύο διαφορετικούς τρόπους . Στην πρώτη τα entries του BHT πίνακα ήταν 2048 και κάθε εγγραφή αντιστοιχούσε σε έναν 4-bit predictor ενώ στη δεύτερη είχαμε 4096 εγγραφές και 8-bit predictors.

Συγκεκριμένα τώρα στην συνάρτηση predict για κάθε εντολή διακλάδωσης βρίσκουμε το κελί που αντιστοιχεί σ' αυτή (στον πίνακα BHT). Έπειτα τοποθετούμε το περιεχόμενο αυτής της εγγραφής ως τα LSBs του δείκτη που θα μας προσδιορίζει ποια εγγραφή πρέπει να προσπελάσουμε στον PHT πίνακα. Ο δείκτης αυτός συμπληρώνεται με το κατάλληλο αριθμό (ανά περίπτωση) από LSB του PC.

Στη συνάρτηση update τώρα αν έχουμε taken αυξάνουμε πρώτα την αντίστοιχη εγγραφή του PHT πίνακα και έπειτα το ίδιο πράττουμε και για τον BHT. Στην

περίπτωση όπου δεν έγινε η διακλάδωση μειώνουμε αντιστοίχως τους πίνακες.

Ακολουθεί ο κώδικας της κλάσης :

```
class Lhp : public BranchPredictor
{
public:
  Lhp(unsigned index_bits_, unsigned cntr_bits_)
      : BranchPredictor(), index_bits(index_bits_), cntr_bits(cntr_bits_) {
      table_entries = 1 <<index_bits; // 2^ index  of BHT
      PHT = new unsigned long long[8192];
          if (!PHT) { printf("error"); exit(1); }
          memset(PHT,0, table_entries * sizeof(*PHT));
          BHT = new unsigned long long[table_entries];
          if (!BHT) {printf("error"); exit(1);}
          memset(BHT,0, table_entries * sizeof(*BHT));
      COUNTER_MAX = 3;
          BHT_MAX = (1<<cntr_bits) -1;
   };
   ~Lhp() {  delete BHT; delete PHT; };
   virtual bool predict(ADDRINT ip, ADDRINT target) {
      unsigned int ip_table_index = ip % table_entries;
      unsigned long long ip_BHT_value = BHT[ip_table_index];
      PHT_index = ip;
          int reg;
          if (cntr_bits == 4 )
                    reg = 0b111111111;
          else
                    reg = 0b00011111;
          PHT_index = PHT_index &reg;
```

```
            PHT_index = PHT_index <<cntr_bits;
            PHT_index = PHT_index + ip_BHT_value;
        unsigned long long ip_PHT_value = PHT[PHT_index];
        unsigned long long prediction = ip_PHT_value >>1;
        return (prediction != 0); // true gia 1(take) false gia 0(notTaken)
    };
    virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {       //
predicted einai h timh poy epistrefei h predict
        unsigned int ip_BHT_index = ip % table_entries; //briskoyme pali se poio entry
antistoixei to branch
        if (actual) {
            if (PHT[PHT_index] <COUNTER_MAX)
                PHT[PHT_index]++;
            if (BHT[ip_BHT_index] <BHT_MAX)
                BHT[ip_BHT_index]++;
        } else {//not taken
            if (PHT[PHT_index] >0) // an einai megalytero apo mhden to meiwnoyme
                PHT[PHT_index]--;
            if (BHT[ip_BHT_index] >0)
                BHT[ip_BHT_index]--;
        }
        updateCounters(predicted, actual);
    };
    virtual string getName() {
        std::ostringstream stream;
        stream <<"Local-"<<pow(2,index_bits) / 1024.0 <<"K-"<<cntr_bits;
        return stream.str();
    }
private:
```

```
    ADDRINT PHT_index;
    unsigned int index_bits, cntr_bits;
    unsigned int COUNTER_MAX;
    unsigned int BHT_MAX;
    unsigned long long *BHT,*PHT;
    unsigned int table_entries;
};
```

Global History

Έπειτα υλοποιήσαμε τον Global-history two levels predictors. Εδώ διαφοροποιήσαμε τόσο το BHR length όσο και τα n-bit του PHT . Όσο αφορά τον PHT στην πρώτη περίπτωση ο BHR έχει μήκος 4 και στη δεύτερη 8 bit. Συνεπώς στην πρώτη περίπτωση έχουμε 16 διαφορετικές περιπτώσεις ενώ στη δεύτερη 256. Ο PHT λοιπόν οργανώθηκε σαν δισδιάστατος πίνακας με 16 και 256 στήλες αντίστοιχα . Τέλος το πλήθος των γραμμών του πίνακα εξαρτάται και από το μέγεθος των n-bit predictors . Αναλυτικά οι 4ις συνδυασμοί:

1)Global 1024 – 2 -4 Μέγεθος pht 16K
2) Global 64 – 2 -8 Μέγεθος pht 16K
3) Global 512 – 4 -4 Μέγεθος pht 8K
4) Global 32 – 4 -8 Μέγεθος pht 8K

Συγκεκριμένα τώρα στην συνάρτηση predict για κάθε εντολή διακλάδωσης βρίσκουμε τη γραμμή που αντιστοιχεί σ' αυτή (στον πίνακα PHT). Η στήλη που μας ενδιαφέρει καθορίζεται πάντα από τον bhr.

Στη συνάρτηση update τώρα αν έχουμε taken αυξάνουμε πρώτα την αντίστοιχη εγγραφή του PHT πίνακα και έπειτα το ίδιο πράττουμε και για τον BHR. Στην

περίπτωση όπου δεν έγινε η διακλάδωση μειώνουμε αντιστοίχως τις καταχωρήσεις.

Ακολουθεί ο κώδικας της κλάσης :

```
class Ghp : public BranchPredictor

{

public:

   Ghp(unsigned index_bits_, unsigned cntr_bits_, unsigned bhr_bits_) // entries = 2^ index cntr = n-bit sto bhr kratame tis prohgoumenes katastaseis twn m branch

      : BranchPredictor(), index_bits(index_bits_), cntr_bits(cntr_bits_), bhr_bits(bhr_bits_) {

      table_entries = 1 <<index_bits; // 2^ index grammes

          pos_situations = 1 <<bhr_bits;  // 2^ bhr_bits sthles

      TABLE = new unsigned long long*[table_entries]; // 2d table 2^ m colume each due to m bhr_bits

          if (!TABLE) { printf("error"); exit(1); }

          for (unsigned int i=0; i<table_entries; i++) {

                  TABLE[i] = new unsigned long long[pos_situations];

                  if (!TABLE[i]) {printf("error"); exit(1);}

                  memset(TABLE[i],0,pos_situations * sizeof(TABLE[0][0]));

          }

          bhr=0;

      COUNTER_MAX = (1 <<cntr_bits) - 1; // deixnei ton arithmo katastasewn jekinwntas apo to mhden etsi gia n=2 exoyme max =3

          BHR_MAX = pos_situations -1;

   };

   ~Ghp() { for (unsigned int i=0; i<table_entries; i++) delete TABLE[i]; delete TABLE; };


   virtual bool predict(ADDRINT ip, ADDRINT target){
            unsigned int ip_table_index = ip % table_entries
            unsigned long long ip_table_value = TABLE[ip_table_index][bhr];
unsigned long long prediction = ip_table_value >>(cntr_bits – 1);
      return (prediction != 0); // true gia 1(take) false gia 0(notTaken)
```

```
};

virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {// predicted einai h timh poy epistrefei h predict

    unsigned int ip_table_index = ip % table_entries; //briskoyme pali se poio entry antistoixei to branch

    if (actual) {// an einai taken tote ayjanoyme to sygkekrimeno entry ektos an einai hdh sth megisth katastash gia ta n-bit

        if (TABLE[ip_table_index][bhr] <COUNTER_MAX)

            TABLE[ip_table_index][bhr]++;

            if (bhr <BHR_MAX)

                bhr++;

    } else {//not taken

        if (TABLE[ip_table_index][bhr] >0) // an einai megalytero apo mhden to meiwnoyme

            TABLE[ip_table_index][bhr]--;

            if (bhr >0 )

                bhr--;

    }

    updateCounters(predicted, actual); // kaloyme th synarthsh poy krataei to posa branch exoyme petyxei kai posa oxi

};

virtual string getName() {

    std::ostringstream stream;

    stream <<"Global-"<<pow(2,index_bits) / 1024.0 <<"K-"<<cntr_bits <<"-"<<bhr_bits;

    return stream.str();

}

private:

unsigned int index_bits, cntr_bits,bhr_bits;

unsigned int COUNTER_MAX;

unsigned int BHR_MAX;

unsigned int bhr;
```

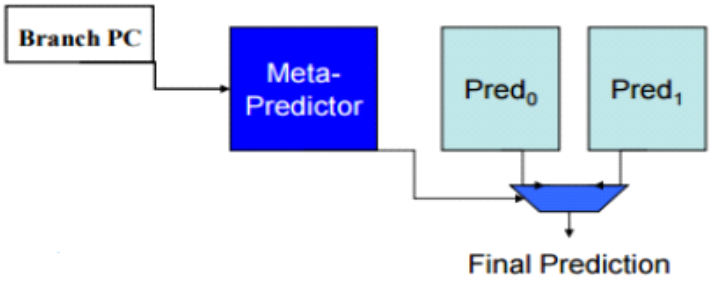
```
    /* Make this unsigned long long so as to support big numbers of cntr_bits. */
    unsigned long long **TABLE;
    unsigned int table_entries,pos_situations;
}
```

Tournament Hybrid predictors

Εδώ υλοποιήσαμε την κλάση Hybrid η οποία δέχεται ως ορίσματα δείκτες σε άλλες κλάσεις predictors. Στη συγκεκριμένη κλάση διατηρούμε ένα πίνακα 512 θέσεων που κάθε μία περιέχει έναν 2-bit predictor.

Πιο συγκεκριμένα στη συνάρτηση predict με βάση την τιμή του πίνακα επιλέγουμε τον predictor του οποίου το αποτέλεσμα θα εμπιστευτούμε .(το κελί του πίνακα επιλέγεται όπως και προηγουμένως με βάση το PC της εντολής)

Στη συνάρτηση update τώρα καταρχάς καλούμε τις αντίστοιχες update συναρτήσεις για τους δύο predictors που χρησιμοποιούμε . Τέλος ενημερώνουμε τον πίνακα του Meta-predictor στις περιπτώσεις που οι δύο predictor έδωσαν διαφορετική πρόβλεψη κατά τον τρόπο που φαίνεται στο παρακάτω σχήμα .



$Pred_0$	$Pred_1$	Meta Update
Λάθος	Λάθος	---
Λάθος	Σωστή	+1
Σωστή	Λάθος	-1
Σωστή	Σωστή	---

Χρησιμοποιήσαμε τελικά 3 meta –predictors με τους predictor αυτών να έχουν μέγεθος 16K ώστε να μην υπάρχει συνολικό overhead σε σχέση με τους υπόλοιπους. Οι συνθέσεις που χρησιμοποιήσαμε είναι οι εξής :

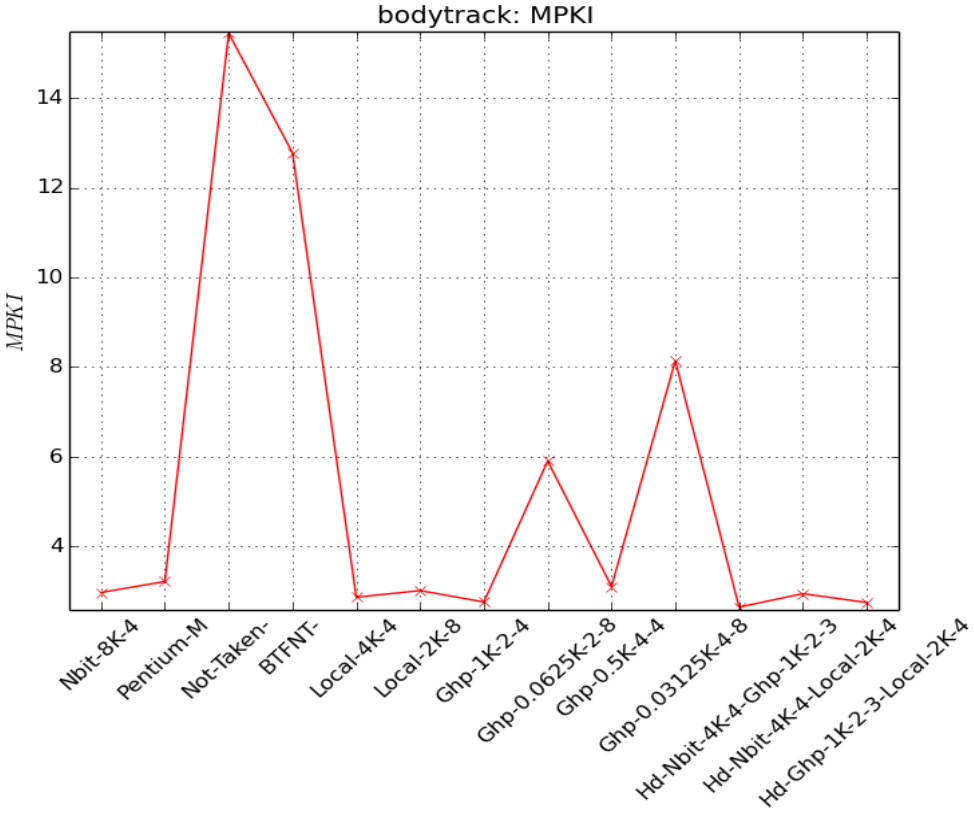
1) N-bit 4K-4 με Global -1k-2-3 (Μέγεθος PHT 8K 3 bit BHR)
2) N-bit 4K-4 με 8192-1 bit 2048-4 bit
3) Global -1k-2-3 με local 8192-1 bit 2048-4 bit

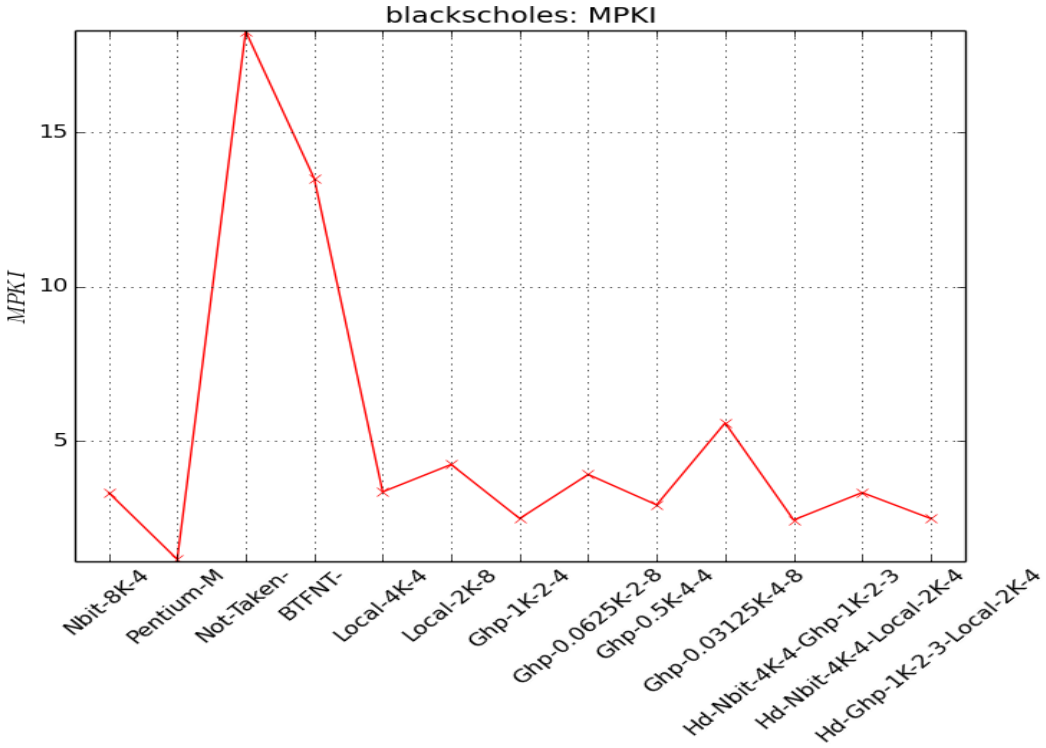
Ακολουθεί ο κώδικας της κλάσης:

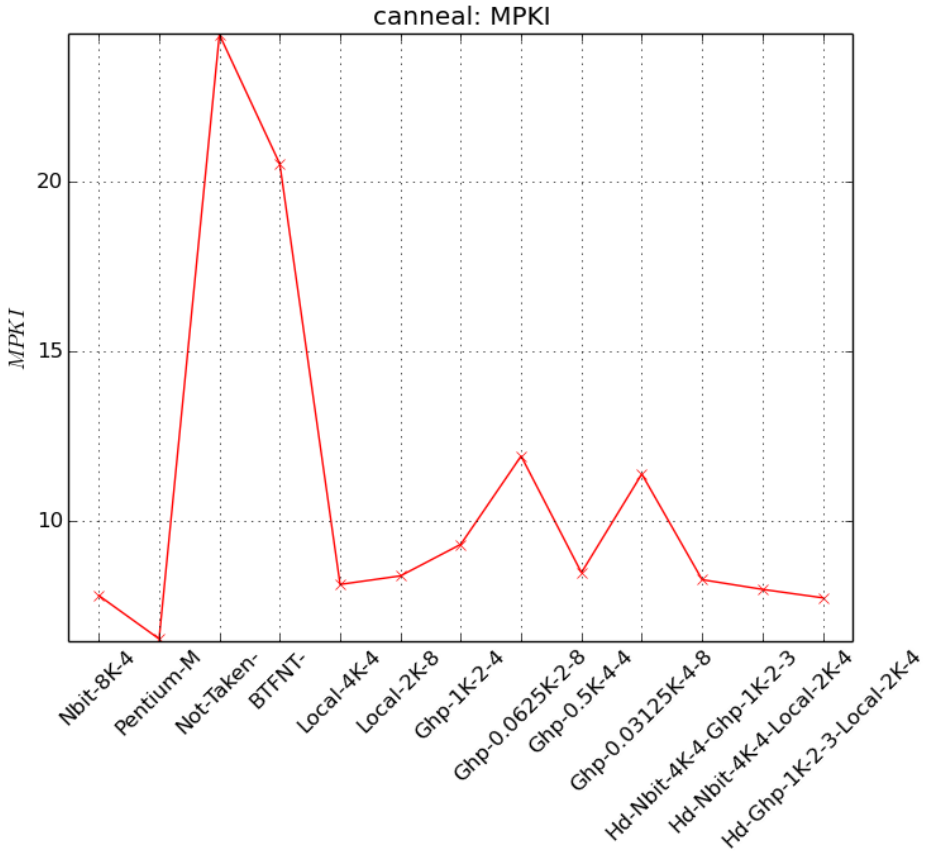
```
class Hybrid : public BranchPredictor

{

public:

   Hybrid(BranchPredictor *curr_predictor_, BranchPredictor *curr_predictorTwo_  )

      : BranchPredictor(), curr_predictor(curr_predictor_),
curr_predictorTwo(curr_predictorTwo_) {

      TABLE = new unsigned long long[512];

      memset(TABLE, 0, 512 * sizeof(*TABLE));
      COUNTER_MAX = 3;

   };

   ~Hybrid() { delete TABLE; };


   virtual bool predict(ADDRINT ip, ADDRINT target){
     // ip current instruction PC, target taken PC address to target de xreiasthke edw

      unsigned int ip_table_index = ip % 512;

      unsigned long long ip_table_value = TABLE[ip_table_index];

      unsigned long long prediction = ip_table_value >>1;

          temp0=curr_predictor->predict(ip, target);

          temp1=curr_predictorTwo->predict(ip, target);

      if (prediction == 0) // true gia 1 (Pred 1) false gia 0 (Pred 0)

                  return temp0;
```

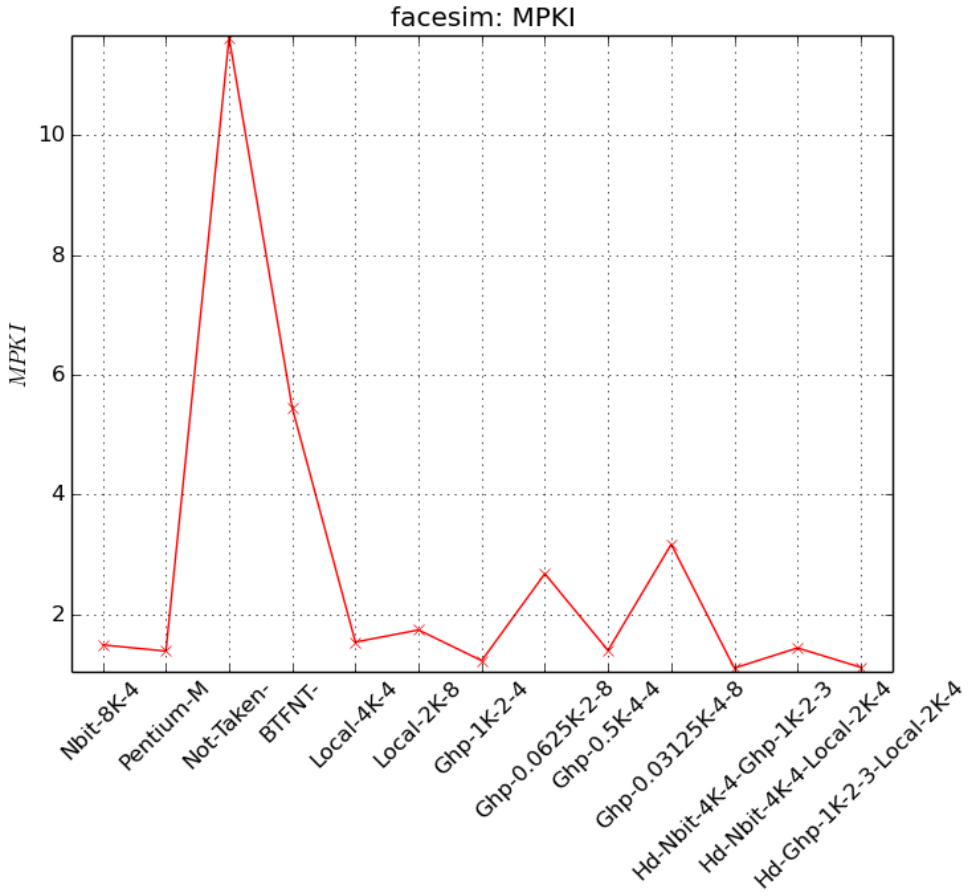
```
        else
                return temp1;
  };
  virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {
    unsigned int ip_table_index = ip % 512;
        if (temp0!=temp1){
                if (temp0==actual){
                        if (TABLE[ip_table_index] >0) // an einai megalytero apo mhden to
meiwnoyme
                        TABLE[ip_table_index]--;
              }
                else{
                     if (TABLE[ip_table_index] <COUNTER_MAX)
                        TABLE[ip_table_index]++;
                }}
        curr_predictor->update(predicted, actual, ip, target);
        curr_predictorTwo->update(predicted, actual, ip, target);
    updateCounters(predicted, actual);    };
  virtual string getName() {
    std::ostringstream stream;
    stream <<"Hybrid-"<<curr_predictor->getName() <<"-"<<curr_predictorTwo-
>getName() ;
    return stream.str();
  }
private:
  BranchPredictor *curr_predictor,*curr_predictorTwo;
  unsigned int COUNTER_MAX;
  bool temp0, temp1;
  unsigned long long *TABLE;
};
```

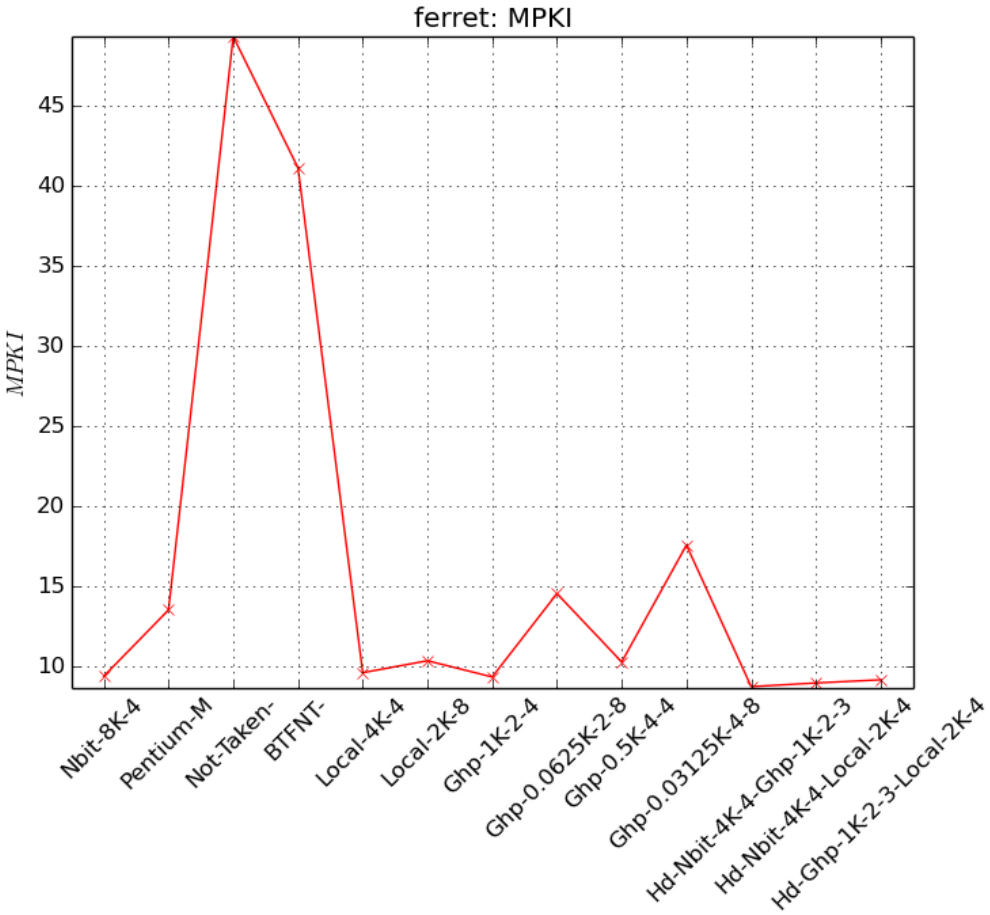
Στη συνέχεια οι γραφικές για όλους τους ζητούμενους predictors . Στους προηγούμενους έχουν προσθεθεί και ο Pentium-M καθώς και ο Nbit-8K-4 που επιλέξαμε στο 2^{ο} ερώτημα.

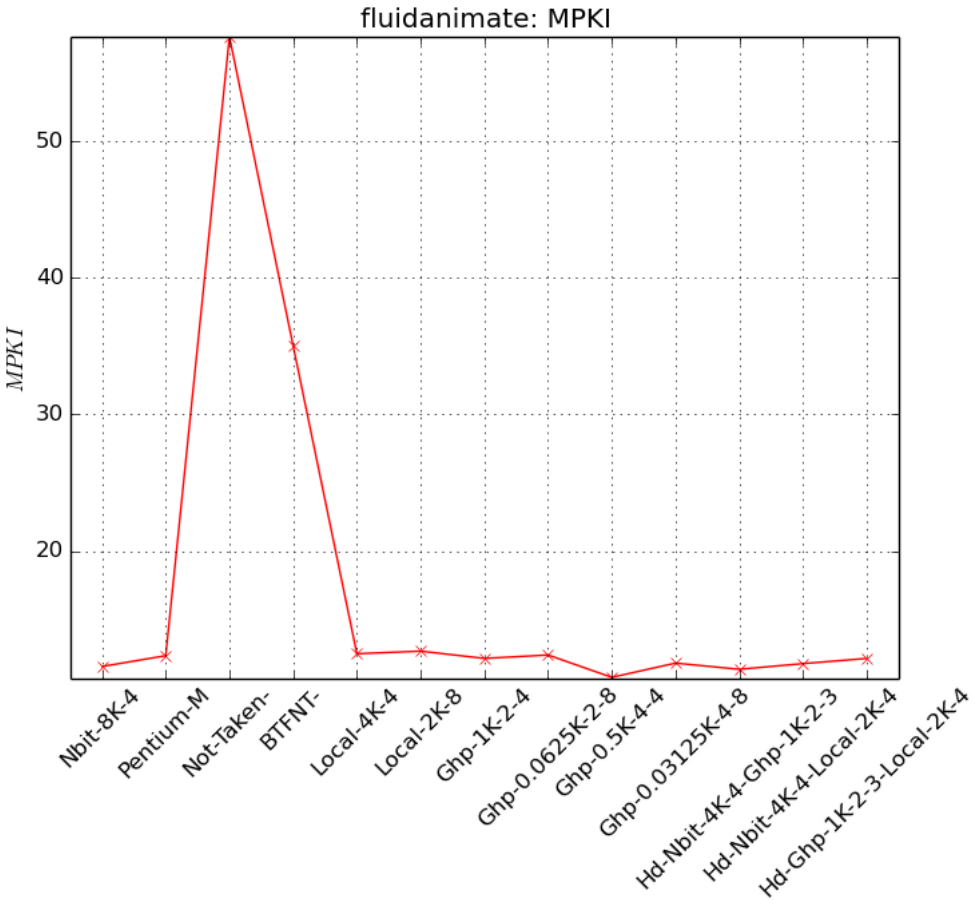


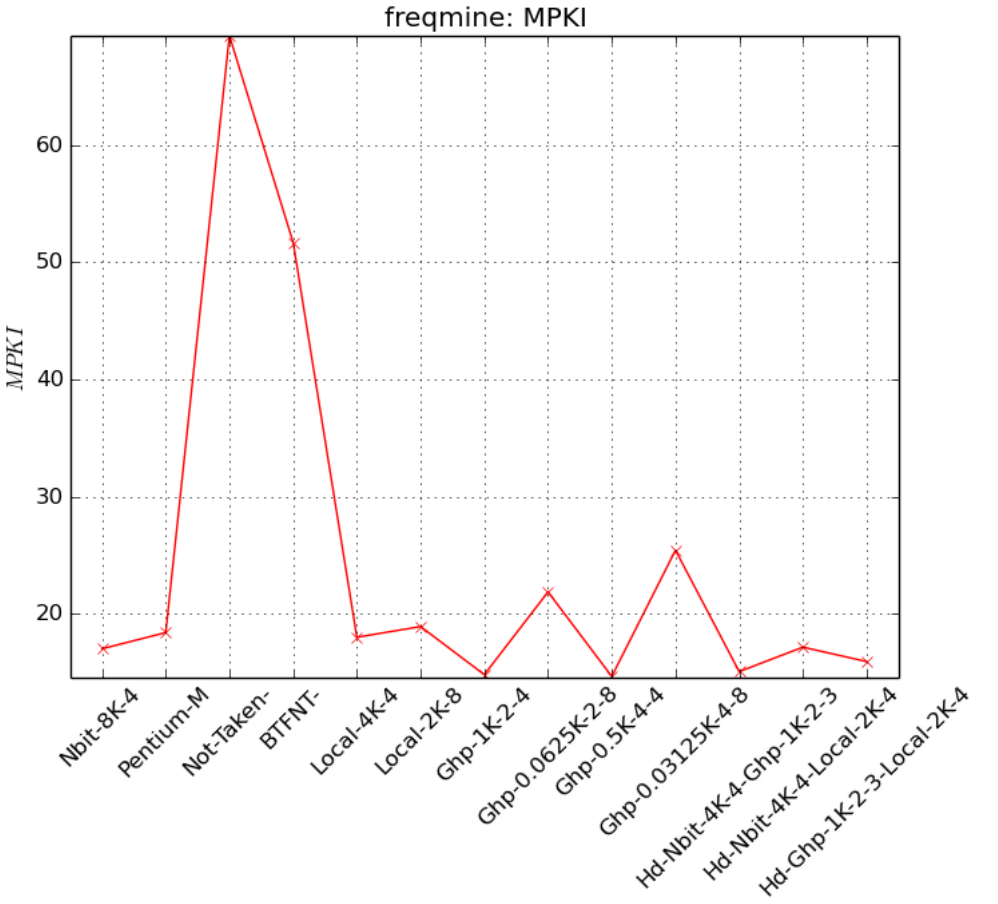


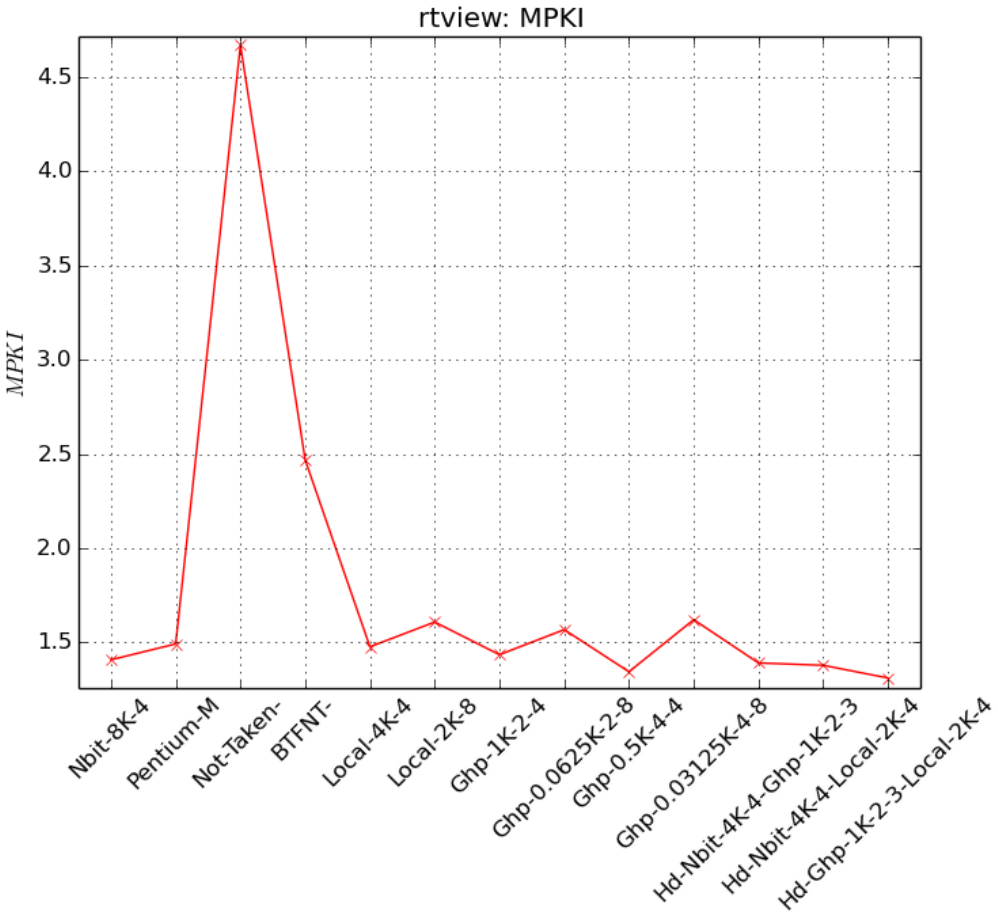

canneal: MPKI
MPKI
20
15
10
Nbit-8K-4
Pentium-M
Not-Taken-
BTFNT-
Local-4K-4
Local-2K-8
Ghp-1K-2-4
Ghp-0.0625K-2-8
Ghp-0.5K-4-4
Ghp-0.03125K-4-8
Hd-Nbit-4K-4-Ghp-1K-2-3
Hd-Nbit-4K-4-Local-2K-4
Hd-Ghp-1K-2-3-Local-2K-4

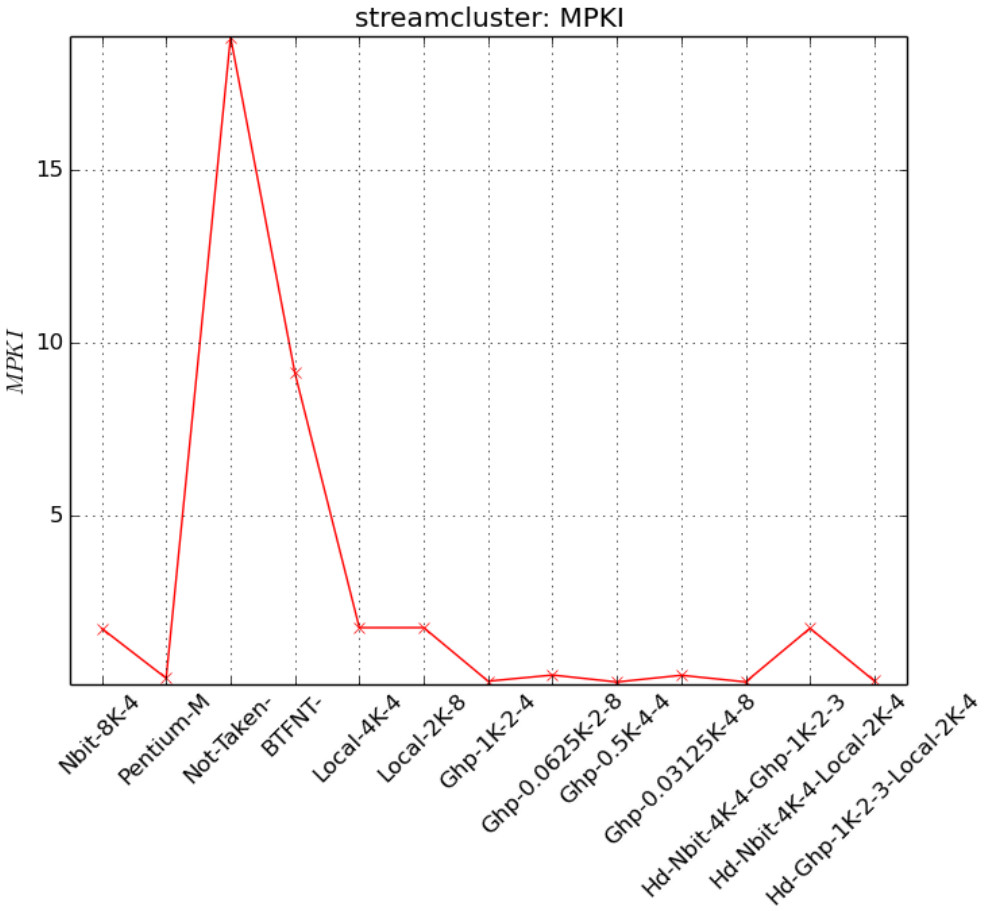

facesim: MPKI
MPKI
10
8
6
4
2
Nbit-8K-4
Pentium-M
Not-Taken-
BTFNT-
Local-4K-4
Local-2K-8
Ghp-1K-2-4
Ghp-0.0625K-2-8
Ghp-0.5K-4-4
Ghp-0.03125K-4-8
Hd-Nbit-4K-4-Ghp-1K-2-3
Hd-Nbit-4K-4-Local-2K-4
Hd-Ghp-1K-2-3-Local-2K-4

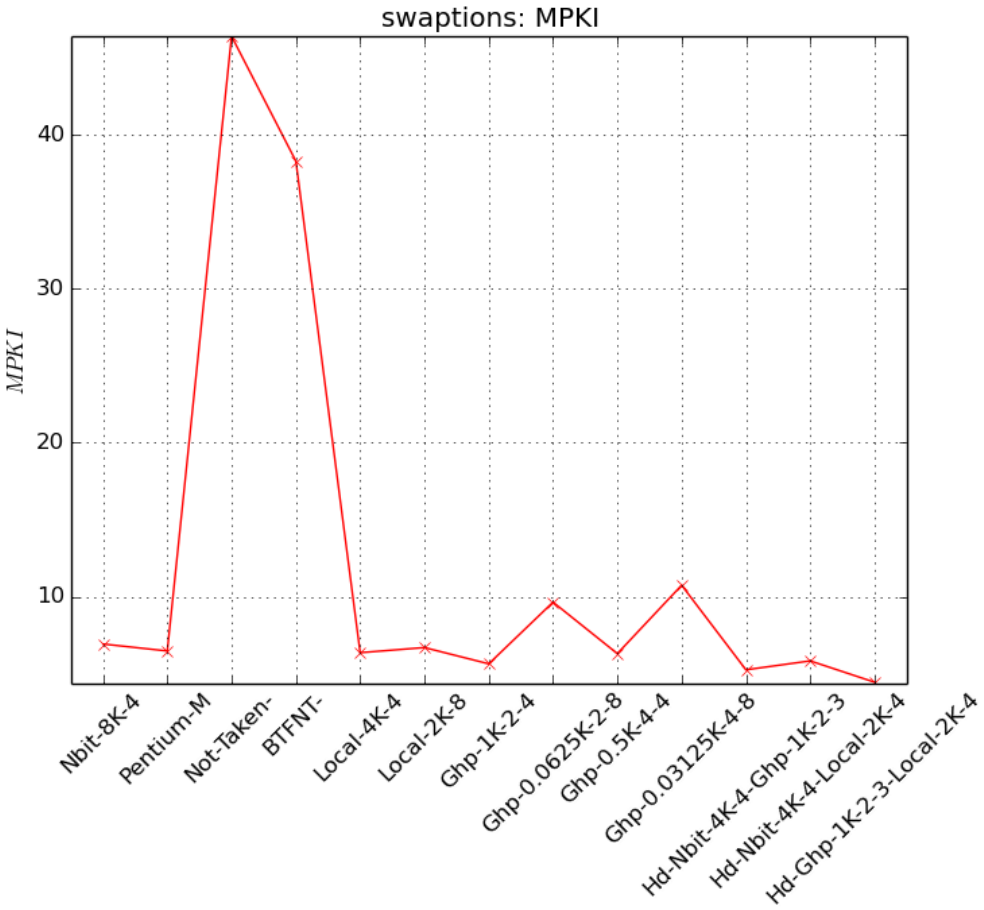

ferret: MPKI
MPKI
45
40
35
30
25
20
15
10
Nbit-8K-4
Pentium-M
Not-Taken-
BTFNT-
Local-4K-4
Local-2K-8
Ghp-1K-2-4
Ghp-0.0625K-2-8
Ghp-0.5K-4-4
Ghp-0.03125K-4-8
Hd-Nbit-4K-4-Ghp-1K-2-3
Hd-Nbit-4K-4-Local-2K-4
Hd-Ghp-1K-2-3-Local-2K-4


fluidanimate: MPKI
MPKI
50
40
30
20
Nbit-8K-4
Pentium-M
Not-Taken-
BTFNT-
Local-4K-4
Local-2K-8
Ghp-1K-2-4
Ghp-0.0625K-2-8
Ghp-0.5K-4-4
Ghp-0.03125K-4-8
Hd-Nbit-4K-4-Ghp-1K-2-3
Hd-Nbit-4K-4-Local-2K-4
Hd-Ghp-1K-2-3-Local-2K-4


freqmine: MPKI
MPKI
60
50
40
30
20
Nbit-8K-4
Pentium-M
Not-Taken-
BTFNT-
Local-4K-4
Local-2K-8
Ghp-1K-2-4
Ghp-0.0625K-2-8
Ghp-0.5K-4-4
Ghp-0.03125K-4-8
Hd-Nbit-4K-4-Ghp-1K-2-3
Hd-Nbit-4K-4-Local-2K-4
Hd-Ghp-1K-2-3-Local-2K-4


rtview: MPKI
MPKI
4.5
4.0
3.5
3.0
2.5
2.0
1.5
Nbit-8K-4
Pentium-M
Not-Taken-
BTFNT-
Local-4K-4
Local-2K-8
Ghp-1K-2-4
Ghp-0.0625K-2-8
Ghp-0.5K-4-4
Ghp-0.03125K-4-8
Hd-Nbit-4K-4-Ghp-1K-2-3
Hd-Nbit-4K-4-Local-2K-4
Hd-Ghp-1K-2-3-Local-2K-4


streamcluster: MPKI
MPKI
15
10
5
Nbit-8K-4
Pentium-M
Not-Taken-
BTFNT-
Local-4K-4
Local-2K-8
Ghp-1K-2-4
Ghp-0.0625K-2-8
Ghp-0.5K-4-4
Ghp-0.03125K-4-8
Hd-Nbit-4K-4-Ghp-1K-2-3
Hd-Nbit-4K-4-Local-2K-4
Hd-Ghp-1K-2-3-Local-2K-4


swaptions: MPKI
MPKI
40
30
20
10
Nbit-8K-4
Pentium-M
Not-Taken-
BTFNT-
Local-4K-4
Local-2K-8
Ghp-1K-2-4
Ghp-0.0625K-2-8
Ghp-0.5K-4-4
Ghp-0.03125K-4-8
Hd-Nbit-4K-4-Ghp-1K-2-3
Hd-Nbit-4K-4-Local-2K-4
Hd-Ghp-1K-2-3-Local-2K-4

Μετά από μία συνολική εποπτεία των παραπάνω ξεχωρίζουμε τα Hybrid Nbit – Global , Hybrid Global – Local καθώς και τα Local 4k-4 Και Global 0,5K -4-4 σε κάποιες εκ των περιπτώσεων . Θα επιλέγαμε λοιπόν έναν εκ των τα Hybrid Nbit – Global και Hybrid Global – Local.

Συμπερασματικά βλέπουμε πως η επιλογή του καταλληλότερου predictor είναι ένα πολυδιάστατο πρόβλημα . Οι μεταβολές στην απόδοση των διαφόρων predictors ανά μετροπρόγραμμα φανερώνουν ότι δεν υπάρχει ο ιδανικός predictor , αλλά ότι απαιτείται ο κατάλληλος συμβιβασμός προκειμένου να ανταποκριθούμε αποδοτικά σε ένα πλήθος εφαρμογών με διαφορετικές απαιτήσεις.

Διαγράμματα

Τέλος παραθέτουμε ενδεικτικά των κώδικα των script που χρησιμοποιήσαμε για τις γραφικές των στατιστικών του 1ου ερωτήματος και των MPKI και Target Correct του 3^{ου}.

4.1

```
#!/usr/bin/env python

import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np

fp = open(sys.argv[1])
line = fp.readline()
while line:
        tokens = line.split()
        if line.startswith("  Total-Branches:"):
                total_bra = float(tokens[1])
```

```
        elif line.startswith("  Conditional-Taken-Branches:"):
                total_taken = float(tokens[1])
        elif line.startswith("  Conditional-NotTaken-Branches:"):
                total_notTaken = float(tokens[1])
        elif line.startswith("  Unconditional-Branches:"):
                total_unco = float(tokens[1])
        elif line.startswith("  Calls:"):
                total_calls = long(tokens[1])
        elif line.startswith("  Returns:"):
                total_returns = float(tokens[1])
        line = fp.readline()
x=[1]
y=[100]
xt=[2]
yt=[100*total_taken/total_bra]
xn=[3]
yn=[100*total_notTaken/total_bra]
xu=[4]
yu=[100*total_unco/total_bra]
xc=[5]
yc=[100*total_calls/total_bra]
xr=[6]
yr=[100*total_returns/total_bra]

onoma=sys.argv[1].split(".")
rects1=plt.bar(x,y,label='Total-Branches=%d' % total_bra, color='k')
rects2=plt.bar(xt,yt,label='Taken-Branches',color='b')
rects3=plt.bar(xn,yn,label='NotTaken-Branches', color='m')
rects4=plt.bar(xu,yu,label='Unconditional-Branches', color='r')
rects5=plt.bar(xc,yc,label='Calls',color='c')
rects6=plt.bar(xr,yr,label='Returns', color='y')

plt.xlabel('x')
plt.ylabel('Percentage (%)')
```

```
plt.title(onoma[0]+': Taken-NotTaken-Unconditional-Calls-Returns')
plt.legend()



def autolabel(rects):
    # attach some text labels
    for rect in rects:
        height = rect.get_height()
        plt.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                "%0.2f %%" % height,
                ha='center', va='bottom')
#autolabel(rects1)
autolabel(rects2)
autolabel(rects3)
autolabel(rects4)
autolabel(rects5)
autolabel(rects6)

name=sys.argv[1]
plt.savefig("L2"+name+".png",bbox_inches="tight")
```

4.3

```
#!/usr/bin/env python


import sys
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np

## For nbit predictors
predictors_to_plot = [ "  BTB" ]


x_Axis = []
```

```
mpki_Axis = []
target_Axis = []

fp = open(sys.argv[1])
line = fp.readline()
while line:
        tokens = line.split()
        if line.startswith("Total Instructions:"):
                total_ins = long(tokens[2])
        else:
                for pred_prefix in predictors_to_plot:
                        if line.startswith(pred_prefix):
                                predictor_string = tokens[0].split(':')[0]
                                correct_predictions = long(tokens[1])
                                incorrect_predictions = long(tokens[2])
                                target_predictions = long(tokens[3])
                                x_Axis.append(predictor_string)
                                target_Axis.append(target_predictions / (total_ins / 1000.0))

                                mpki_Axis.append(incorrect_predictions / (total_ins /
1000.0))

        line = fp.readline()

fig, ax1 = plt.subplots()
ax1.grid(True)

xAx = np.arange(len(x_Axis))
ax1.xaxis.set_ticks(np.arange(0, len(x_Axis), 1))
ax1.set_xticklabels(x_Axis, rotation=45)
ax1.set_xlim(-0.5, len(x_Axis) - 0.5)
ax1.set_ylim(min(mpki_Axis) - 0.05, max(mpki_Axis) + 0.05)
ax1.set_ylabel("$MPKI$")
line1 = ax1.plot(mpki_Axis, label="mpki", color="red",marker='x')
```

```
ax2 = ax1.twinx()
ax2.xaxis.set_ticks(np.arange(0, len(x_Axis), 1))
ax2.set_xticklabels(x_Axis, rotation=45)
ax2.set_xlim(-0.5, len(x_Axis) - 0.5)
ax2.set_ylim(min(target_Axis) - 0.05 * min(target_Axis), max(target_Axis) + 0.05 *
max(target_Axis))
ax2.set_ylabel("$TargetCorrect$")
line2 = ax2.plot(target_Axis, label="TargetCorrect", color="b",marker='o')

lns = line1 + line2
labs = [l.get_label() for l in lns]

onoma=sys.argv[1].split(".")
plt.title(onoma[0]+': MPKI')
name=sys.argv[1]
plt.savefig("L2"+name+".png",bbox_inches="tight")
```