Εθνικο Μετσοβιο Πολυτεχνειο

ΠΡΟΗΓΜΕΝΑ ΘΕΜΑΤΑ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

2η Σειρά Ασκήσεων

Αχλάτης Στέφανος-Σταμάτης (03116149)

<el16149@central.ntua.gr>

Μάιος 2020

Περιεχόμενα

1.Σκοπός της Άσκησης

Η συγκεκριμένη εργαστηριακή άσκηση έχει ως σκοπό την μελέτη της επίδρασης διαφορετικών συστημάτων πρόβλεψης εντολών άλματος και την αξιολόγηση τους με αντικειμενικό κριτήριο τον διαθέσιμο χώρο του chip. Πιο συγκεκριμένα γίνεται εξέτασης της επίδρασης των βασικών predictors στην απόδοση 12 μετροπρογραμμάτων (**benchmarks**). Μετά το πέρας της εξέτασης, θα γίνει επιλογή του καταλληλότερου predictor για παραπάνω 12 μετροπρογράμματα.

2. PIN TOOL

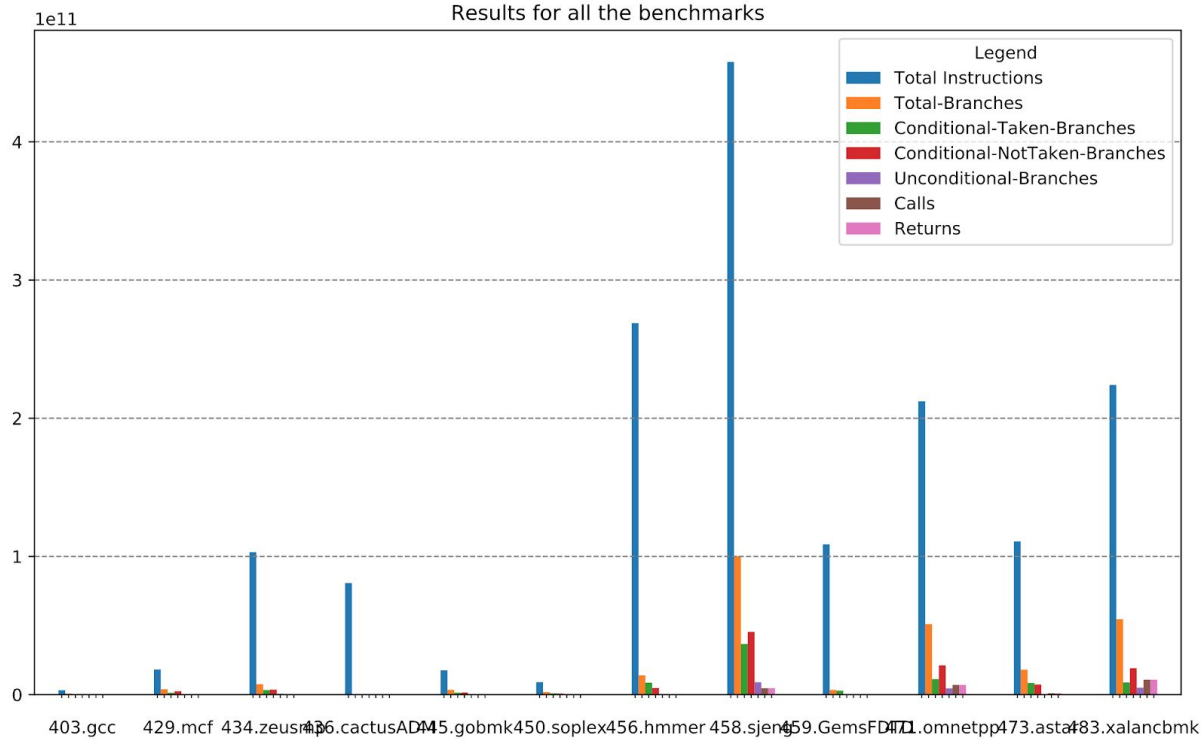
- Έγινε χρήση του εργαλείου Pin, ένα εργαλείο ανάλυσης εφαρμογών που αναπτύσσεται και συντηρείται από την εταιρεία Intel
- Προσφέρει δυνατότητες για dynamic binary instrumentation, ή εναλλακτικά επιτρέπει την δυναμική εισαγωγή και τροποποίηση κώδικα της εφαρμογής κατά την διάρκεια εκτέλεσης των αυτών των μετροπρογραμμάτων. Απώτερος σκοπός είναι η συλλογή πληροφοριών που αφορούν την εκτέλεση, συγκεκριμένα **αριθμός εντολών, αριθμός misses και hits στην cache.**
- Η έκδοση του PIN στην οποία πραγματοποιήθηκε η προσομοίωση είναι η **PIN 98189**, σε **Ubuntu Linux 18.04** με έκδοση **πυρήνα 4.4**.

3. Benchmarks

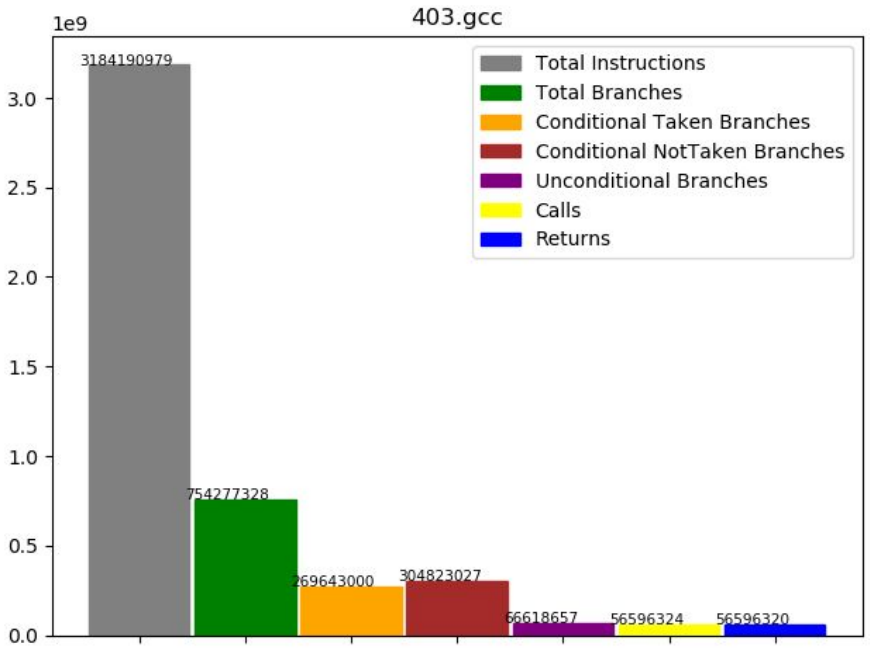
- Τα 12 μετροπρογράμματα (benchmarks) που χρησιμοποιήθηκαν ήταν SPEC CPU2006
- Στα πλαίσια των προσομοιώσεων μας χρησιμοποιήθηκαν 12 από αυτά τα benchmarks. Συγκεκριμένα:
    1. 403.gcc
    2. 429.mcf
    3. 434.zeusmp
    4. 436.cactusADM
    5. 445.gobmk
    6. 450.soplex
    7. 456.hmmer
    8. 458.sjeng
    9. 459.GemsFDTD
    10. 471.omnetpp
    11. 473.astar
    12. 483.xalancbmk

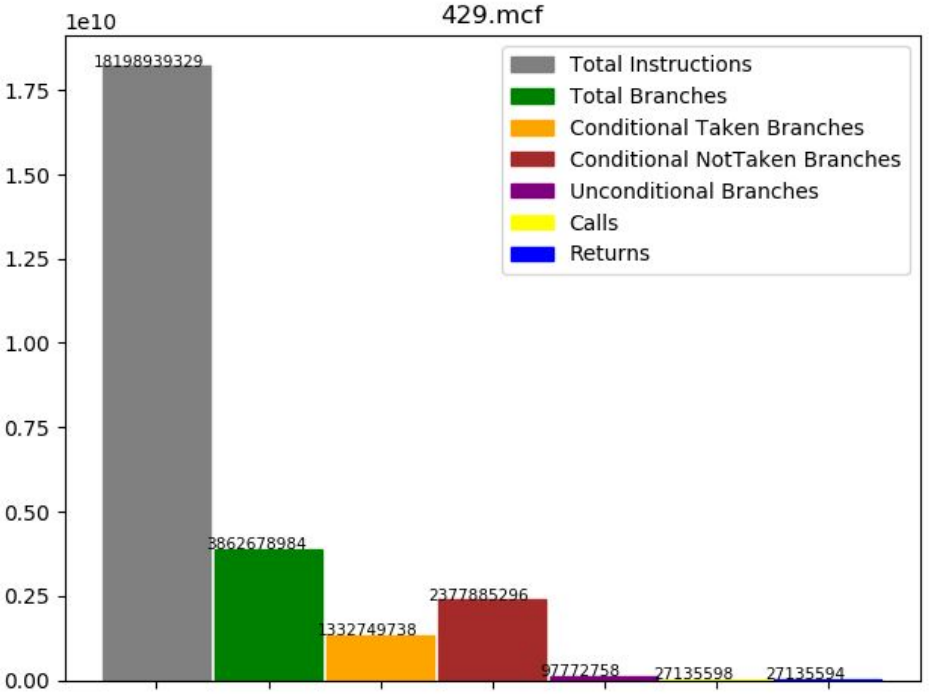
4. Πειραματική Αξιολόγηση
4.1 Μελέτη εντολών άλματος

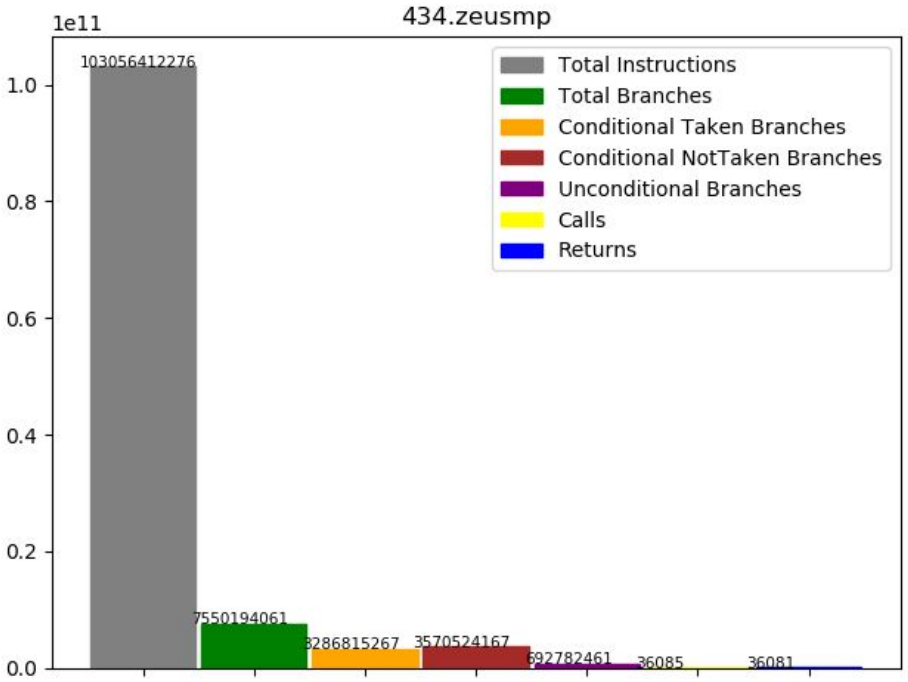
Στα πλαίσια της παρούσας εργασίας, θα διερευνήσουμε τις εντολές άλματος που γίνονται για κάθε benchmark που μας δίνεται. Στη συνέχεια ακολουθεί ένα διάγραμμα με το πλήθος των εντολών άλματος καθώς και τα είδη αυτών. Η πλειοψηφία τους είναι conditional-taken και conditional-notaken.

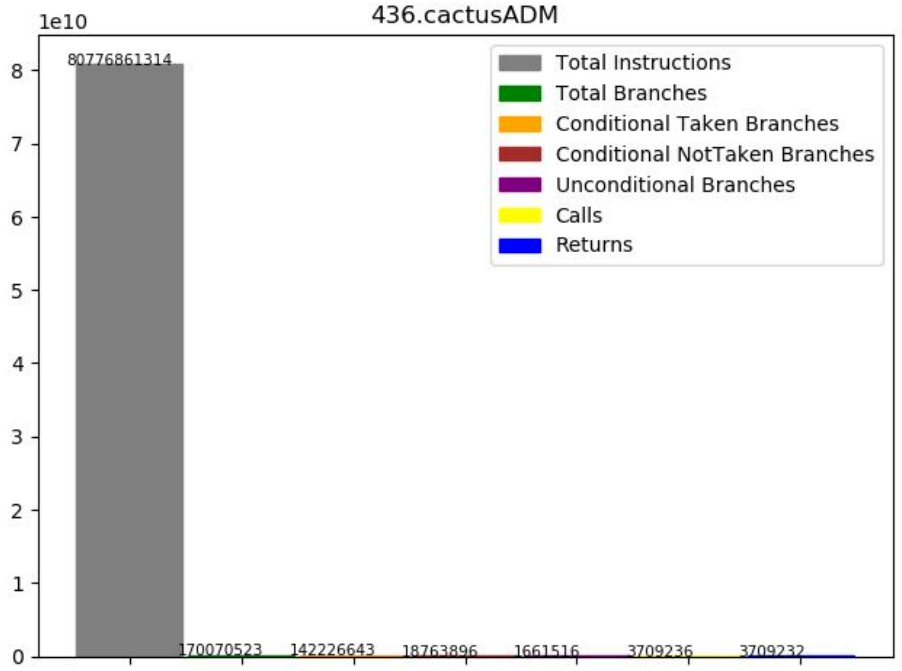


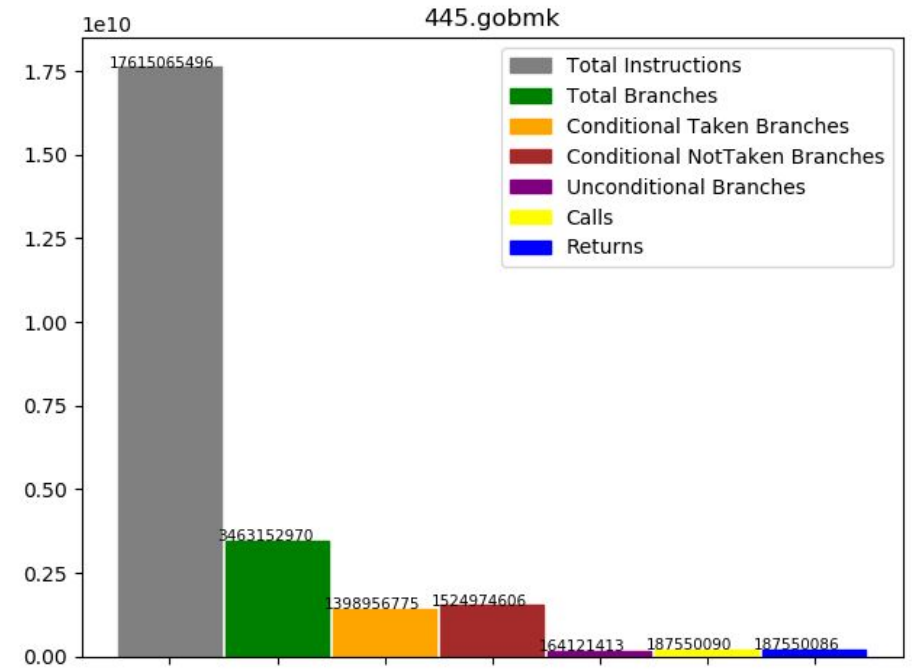
Παρακάτω παρουσιάζονται τα αποτελέσματα ξεχωριστά για κάθε benchmark για να γίνει σαφέστερο το παραπάνω διάγραμμα:

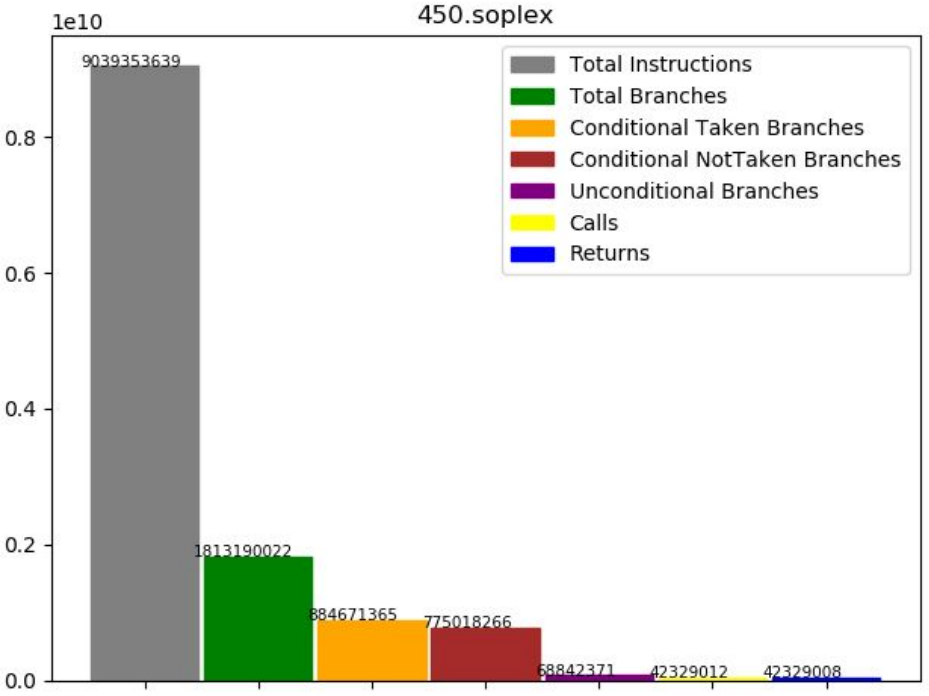

403.gcc
1e9
3184190979
754277328
269643000
304823027
66618657
56596324
56596320
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns
3.0
2.5
2.0
1.5
1.0
0.5
0.0

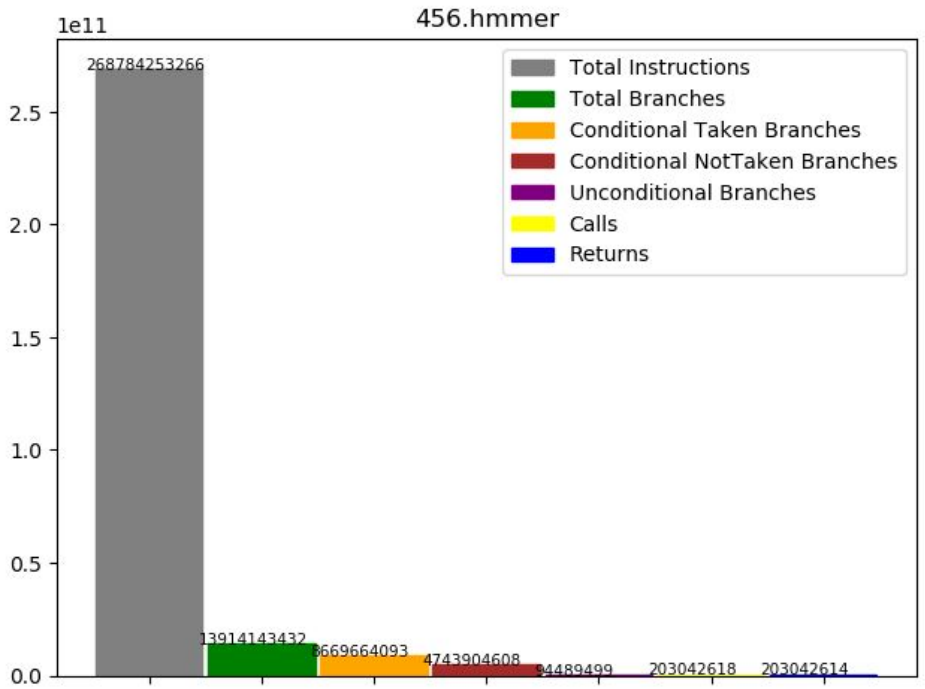

429.mcf
1e10
18198939329
3862678984
1332749738
2377885296
97772758
27135598
27135594
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00

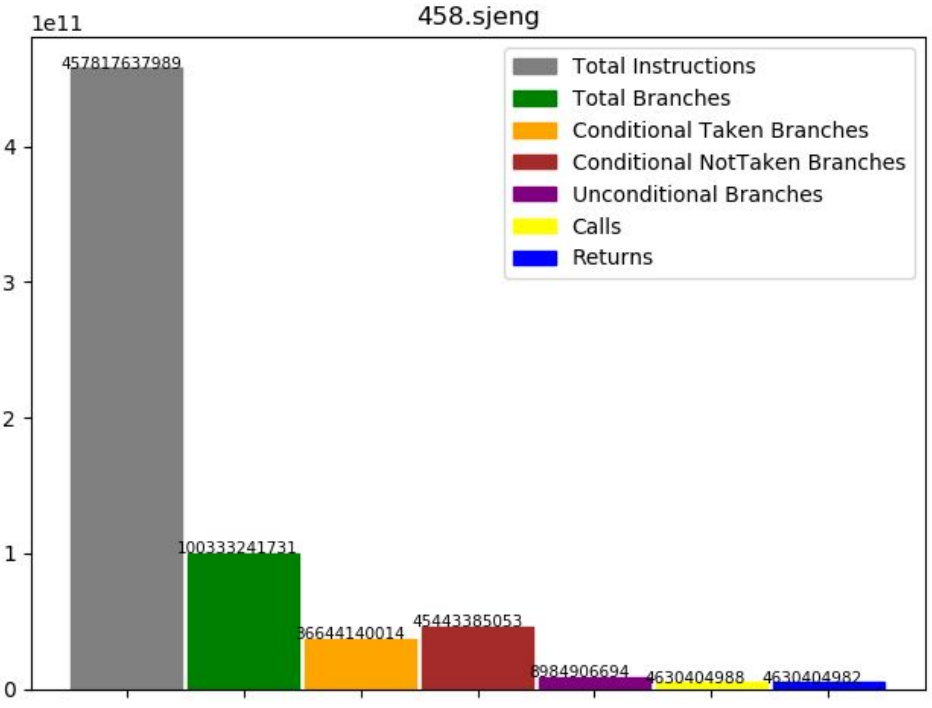

434.zeusmp
1e11
1.0
0.8
0.6
0.4
0.2
0.0
103056412276
7550194061
3286815267
3570524167
692782461
36085
36081
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns

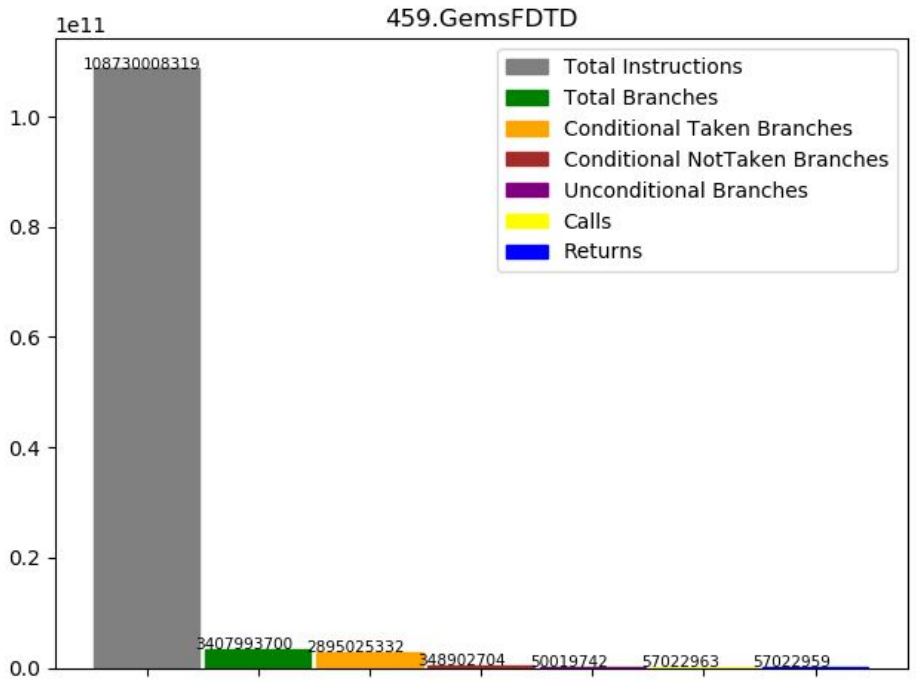

436.cactusADM
1e10
8
7
6
5
4
3
2
1
0
80776861314
170070523
142226643
18763896
1661516
3709236
3709232
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns

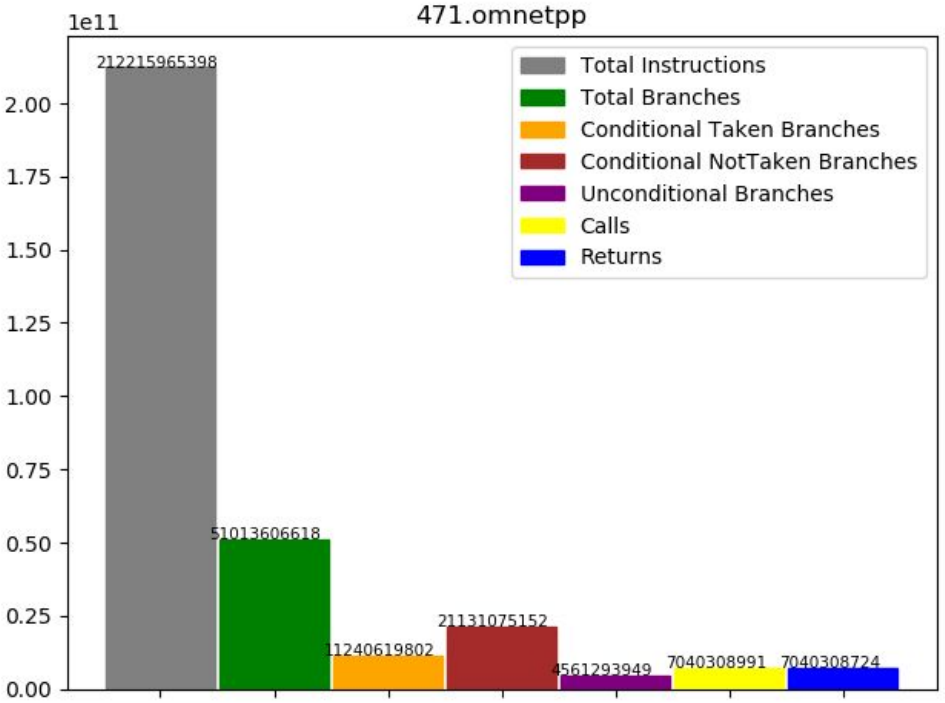

445.gobmk
1e10
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
17615065496
3463152970
1398956775
1524974606
164121413
187550090
187550086
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns

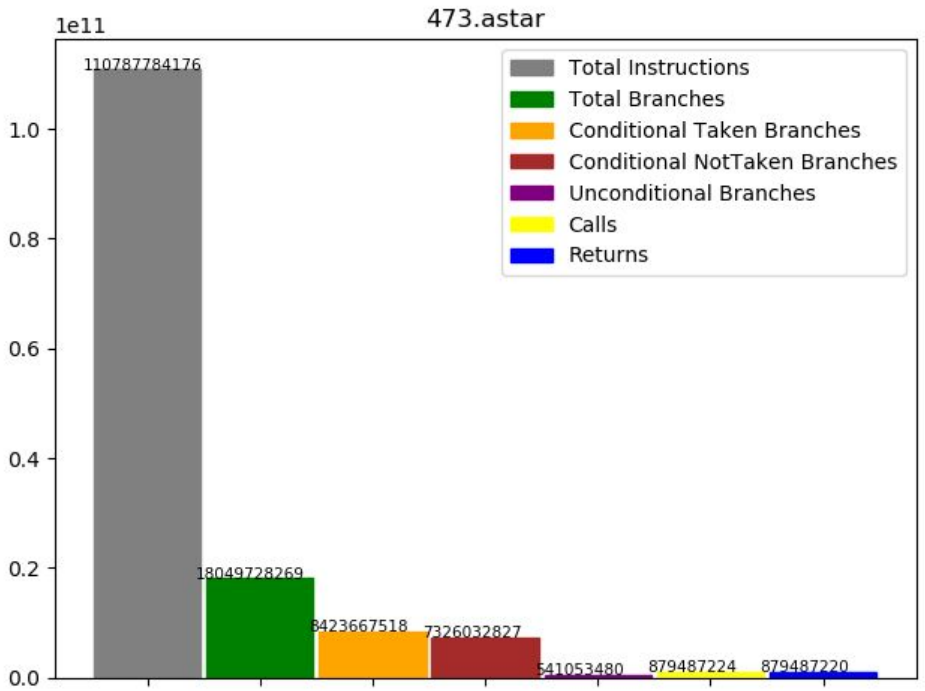

450.soplex
1e10
0.8
0.6
0.4
0.2
0.0
9039353639
1813190022
884671365
775018266
68842371
42329012
42329008
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns

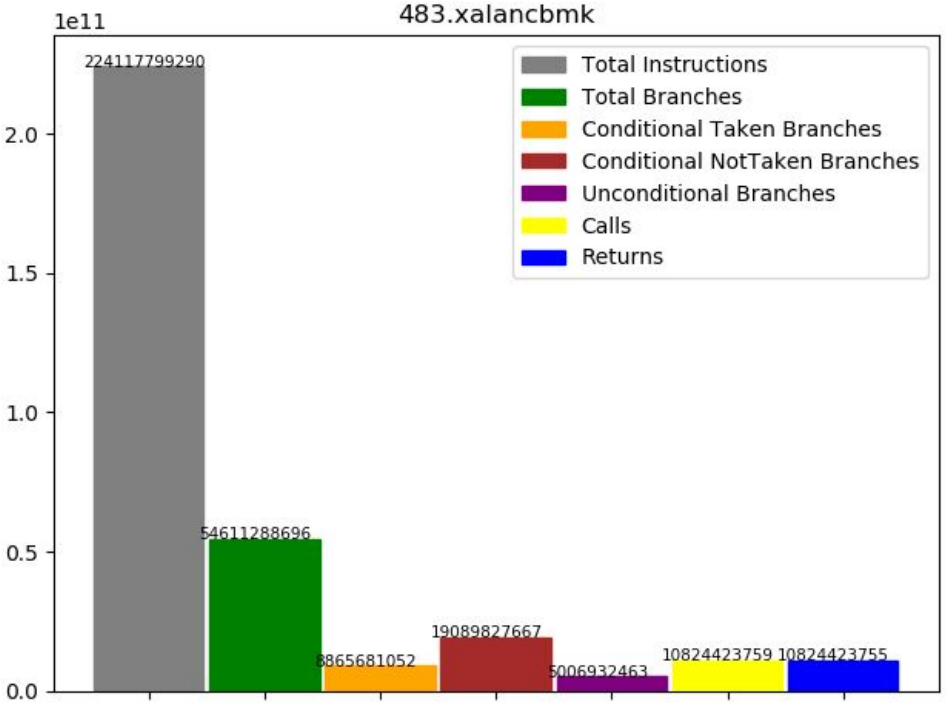

456.hmmer
1e11
2.5
2.0
1.5
1.0
0.5
0.0
268784253266
13914143432
8669664093
4743904608
94489499
203042618
203042614
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns


458.sjeng
1e11
4
3
2
1
0
457817637989
100333241731
36644140014
45443385053
8984906694
4630404988
4630404982
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns


459.GemsFDTD
1e11
1.0
0.8
0.6
0.4
0.2
0.0
108730008319
3407993700
2895025332
348902704
50019742
57022963
57022959
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns


471.omnetpp
1e11
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
212215965398
51013606618
11240619802
21131075152
4561293949
7040308991
7040308724
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns


473.astar
1e11
110787784176
18049728269
8423667518
7326032827
541053480
879487224
879487220
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns
1.0
0.8
0.6
0.4
0.2
0.0


483.xalancbmk
1e11
224117799290
54611288696
6865681052
19089827667
5006932463
10824423759
10824423755
Total Instructions
Total Branches
Conditional Taken Branches
Conditional NotTaken Branches
Unconditional Branches
Calls
Returns
2.0
1.5
1.0
0.5
0.0

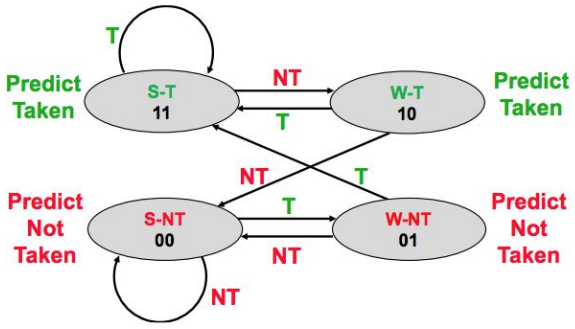
4.1.1 Συμπεράσματα για το 4.1

Όπως αναφέρθηκε και αρχικά, από τα παραπάνω διαγράμματα παρατηρούμε ότι τα branches είναι ένα σεβαστό ποσοστό των συνολικών εντολών που υπάρχουν στα μελετούμενα μετροπρογράμματα. Επίσης, παρατηρούμε ότι τα περισσότερα απο αυτά τα branches είναι είτε conditional-taken, είτε conditional-not-taken.
Παρατηρούμε πολύ λίγα είναι τα unconditional branches, τα calls και τα returns όπως εξάλλου περιμέναμε.
Γενικότερα το μεγαλύτερο ποσοστό εντολών δεν είναι τα branches, κατά μέσο όρο αποτελούν το 15% των συνολικών εντολών του μετροπρογράμματος. Εντολές load και store σημειώνουν υψηλά ποσοστά! Πάντως συγκεκριμένα για τα benchmarks, το hmmer έχει υψηλά ποσοστά εντολών μνήμης, ενώ τα gcc, mcfm xalancbmk, sjeng έχουν υψηλότερα ποσοστά εντολών άλματος.
Παρόλα αυτά, τα υψηλά αυτά ποσοστά δεν αποτελούν ένδειξη για παρουσία περισσότερων bottlenecks. Υπάρχουν benchmarks που σημειώνουν υψηλά ποσοστά και στις δύο κατηγορίες εντολών και εντούτοις διατηρούν υψηλό IPC
Επομένως, αυτά τα benchmarks είναι **κατάλληλα** για την μελέτη που θα κάνουμε πάνω στα διαφορετικά συστήματα πρόβλεψης εντολών άλματος.

4.2 Μελέτη των N-bit Predictors

4.2.1 Μελέτη Α περίπτωσης

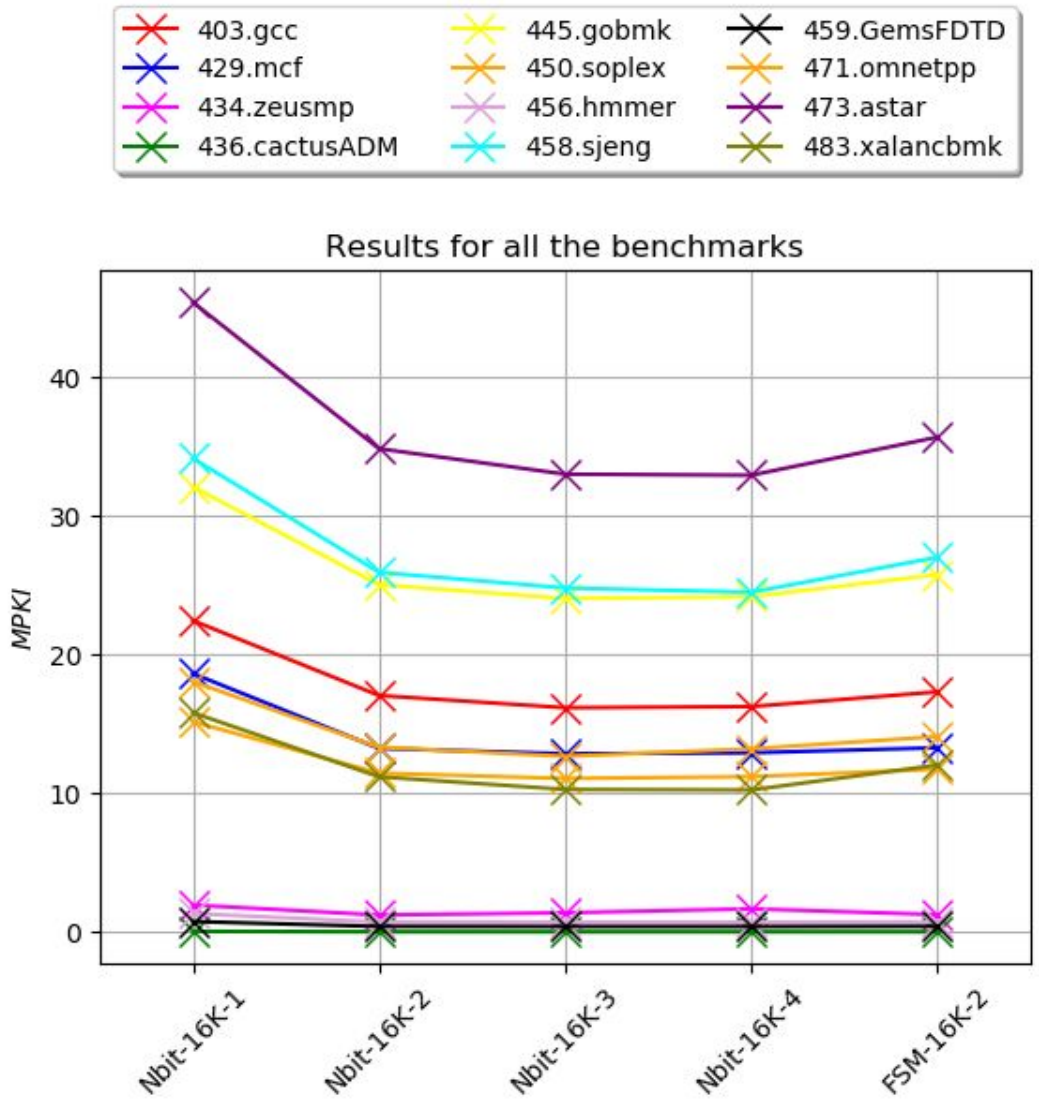
Αρχικά μελετούμε την απόδοση των n-bits predictors χρησιμοποιώντας την υλοποίηση που δίνεται από το cslab στο branch_predictor.h.
Στο πρώτο στάδιο θα διατηρήσουμε σταθερό τον αριθμό των BHT entries και ίσο με entries=16K, και θα προσομοιώσουμε τους n-bit predictors, για N=1,2,3,4 χρησιμοποιώντας τα direction Mispredictions Per Thousand Instructions (direction MPKI).
Επίσης θα προσομοιώσουμε το εξείς FSM για N=2:



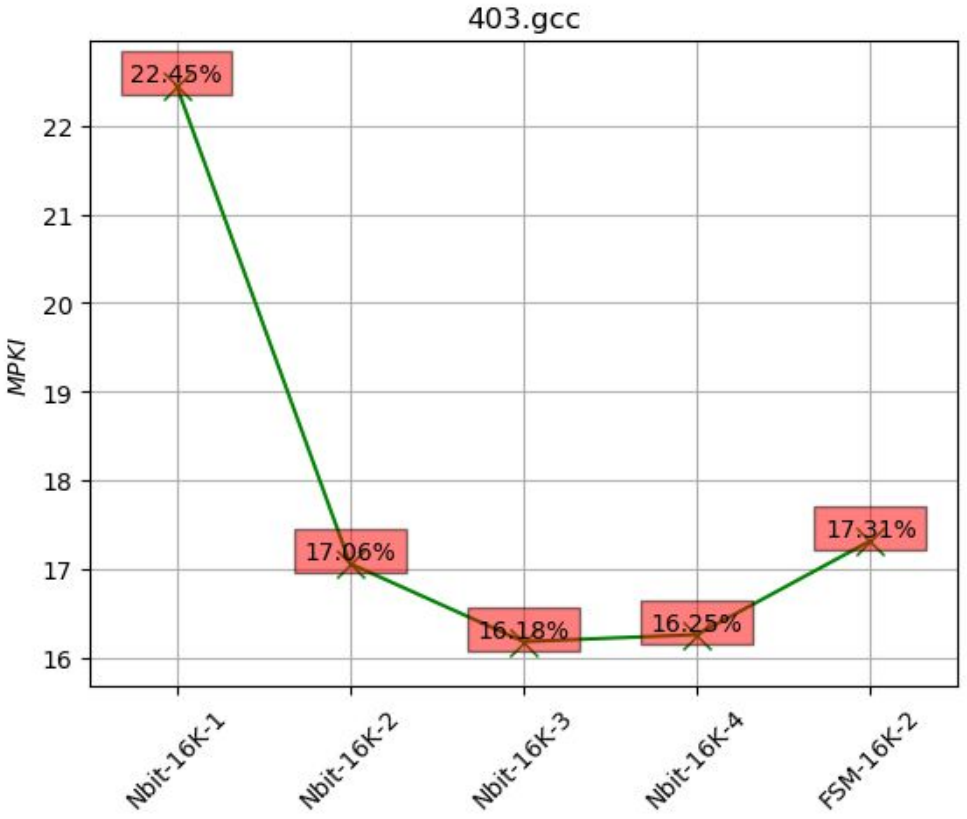
Για να χρησιμοποιήσουμε αυτό το fsm πρέπει να δημιουργήσουμε μια νέα κλάση ακριβώς ίδια με αυτή του n-bit predictor τροποποιόντας την διαδικασία update ως εξής:

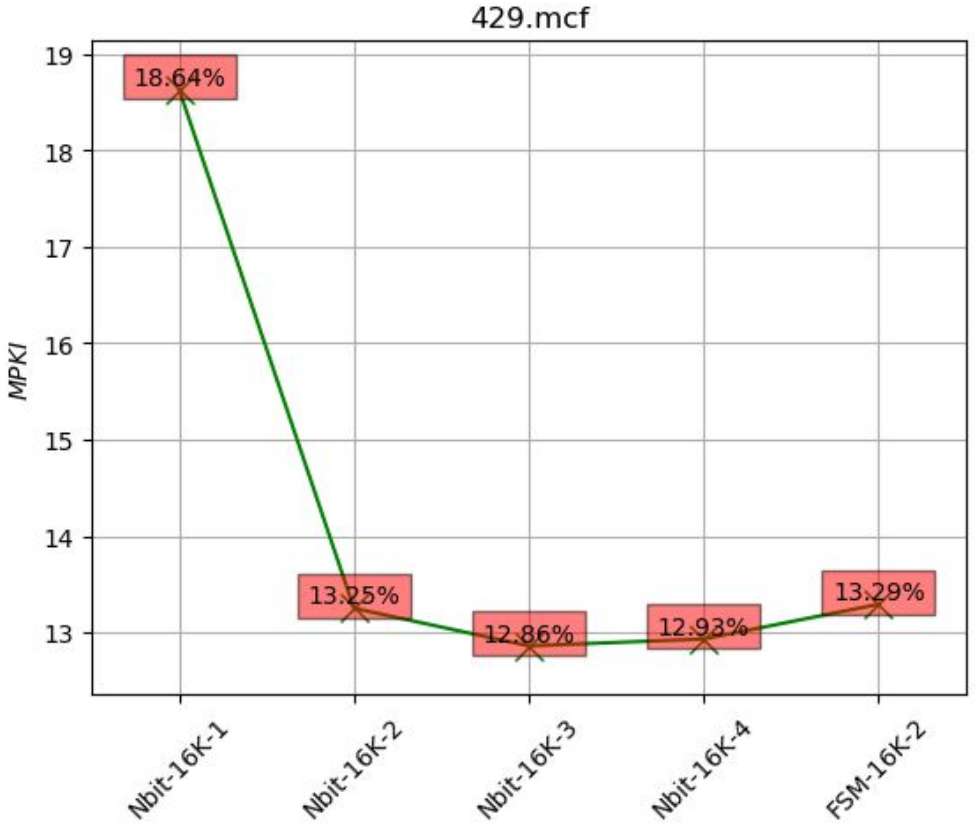
```
virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {
    unsigned int ip_table_index = ip % table_entries;
    if (actual) {
        if(TABLE[ip_table_index]==1) TABLE[ip_table_index]=3;
        else if (TABLE[ip_table_index] < COUNTER_MAX && TABLE[ip_table_index]!=1) TABLE[ip_table_index]++;
    }
    else {
        if (TABLE[ip_table_index]==2) TABLE[ip_table_index]=0;
        else if (TABLE[ip_table_index]>0 && TABLE[ip_table_index] !=2) TABLE[ip_table_index]--;
    }

    updateCounters(predicted, actual);
};
```

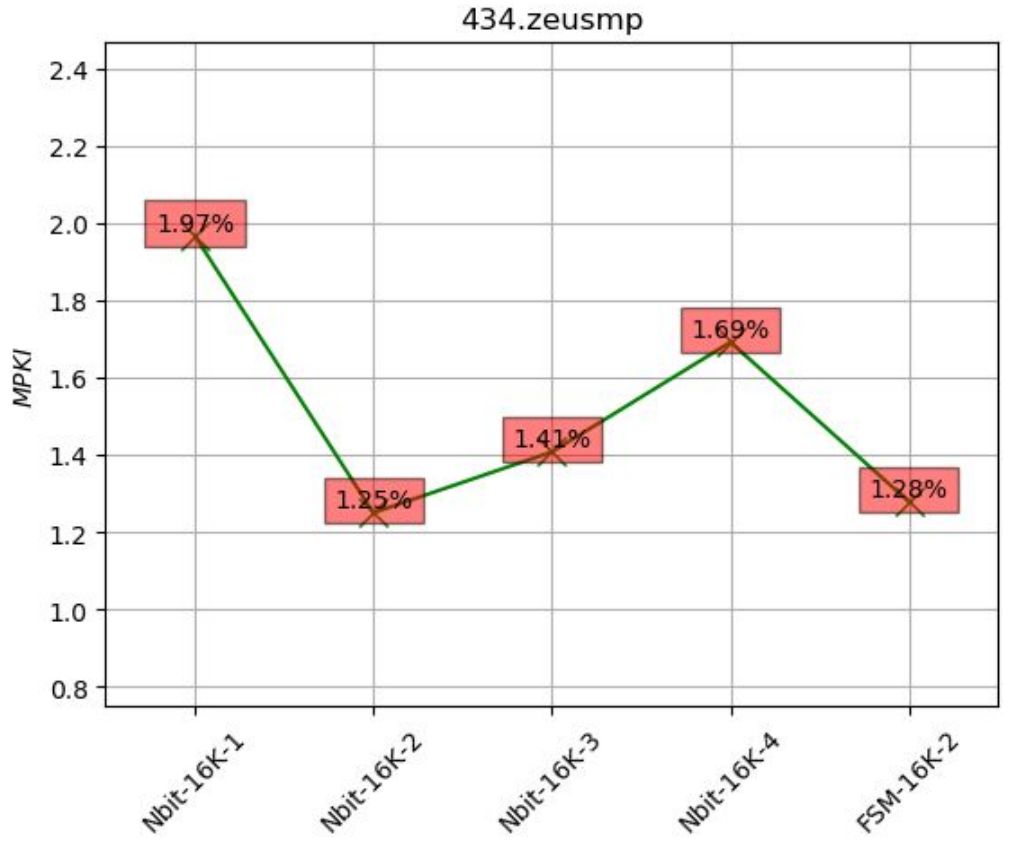
Παρακάτω παρουσιάζεται το συγκεντρωτικό αποτέλεσμα για όλα τα benchmarks:

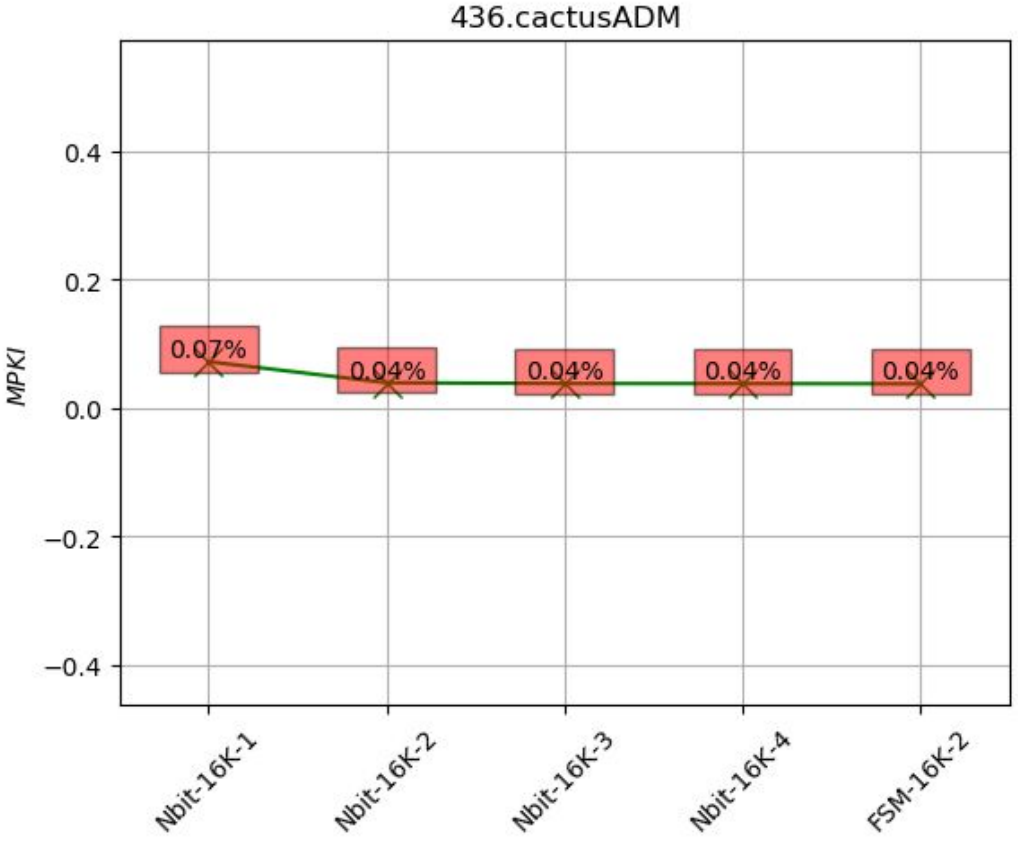


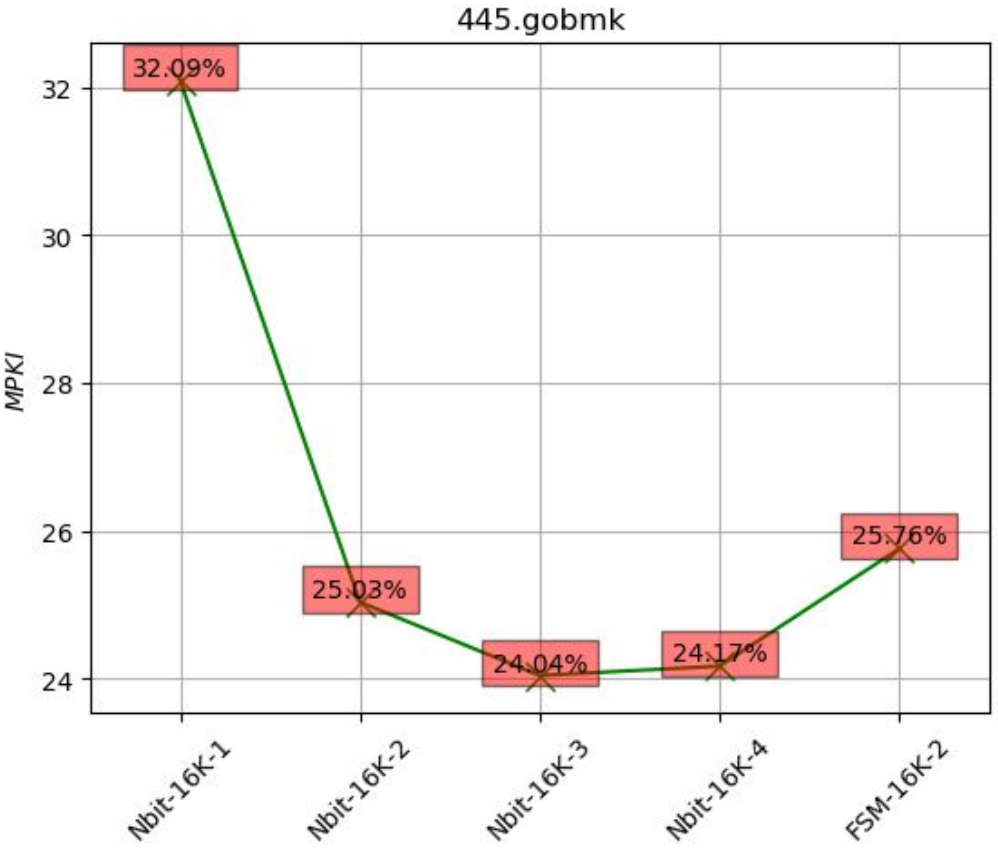
Στην συνέχεια παρουσιάζεται το αποτέλεσμα για καθε ενα απο τα επιμέρους benchmarks έτσι ώστε να έχουμε πιο λεπτομερή μελέτη.

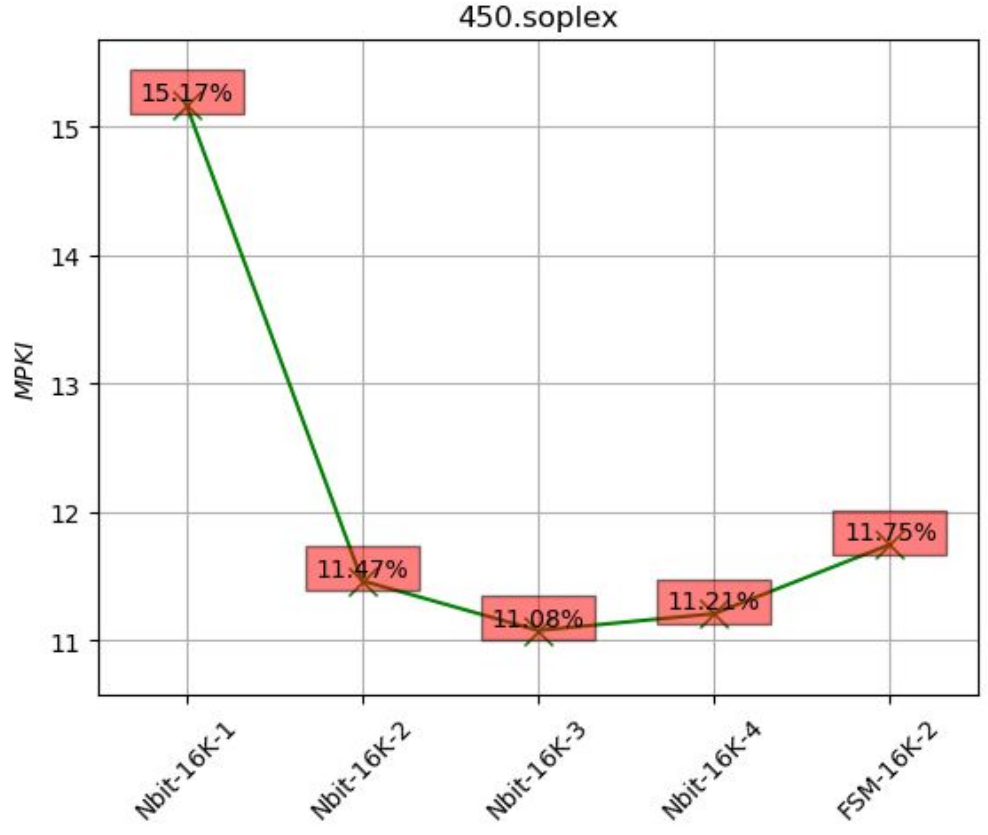

403.gcc
MPKI
22
21
20
19
18
17
16
22.45%
17.06%
16.18%
16.25%
17.31%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2


429.mcf
MPKI
19
18
17
16
15
14
13
18.64%
13.25%
12.86%
12.93%
13.29%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2

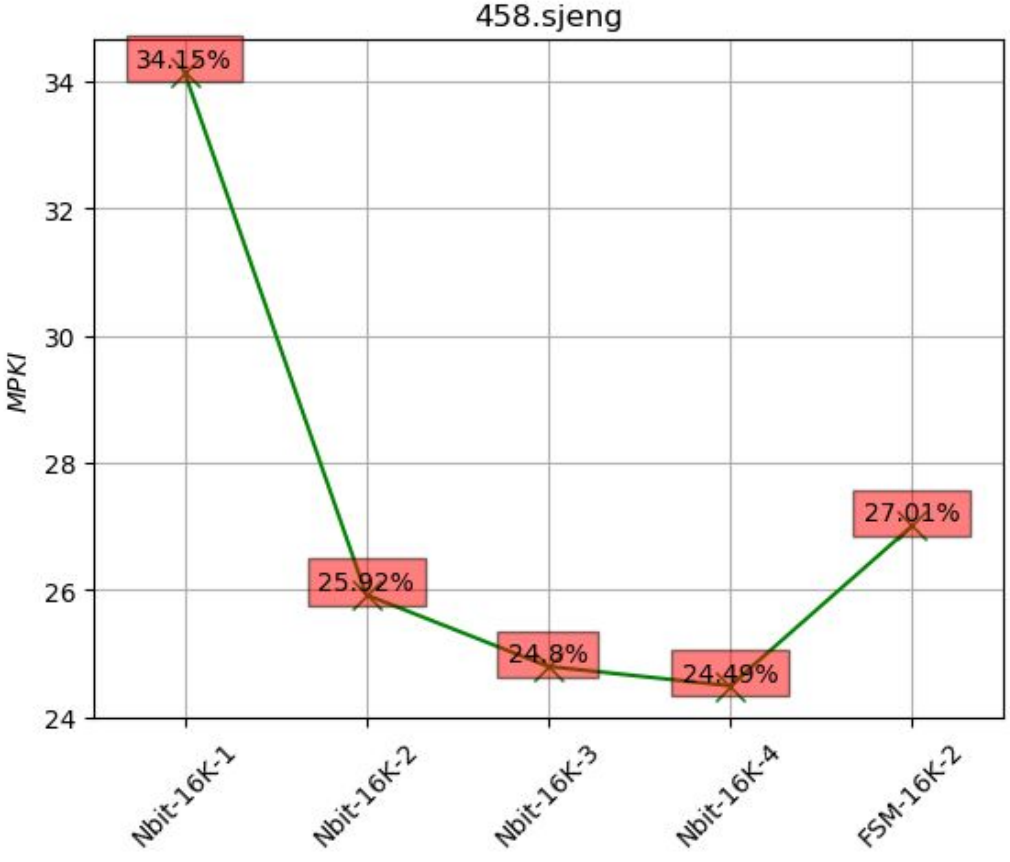

434.zeusmp
MPKI
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
1.97%
1.25%
1.41%
1.69%
1.28%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2

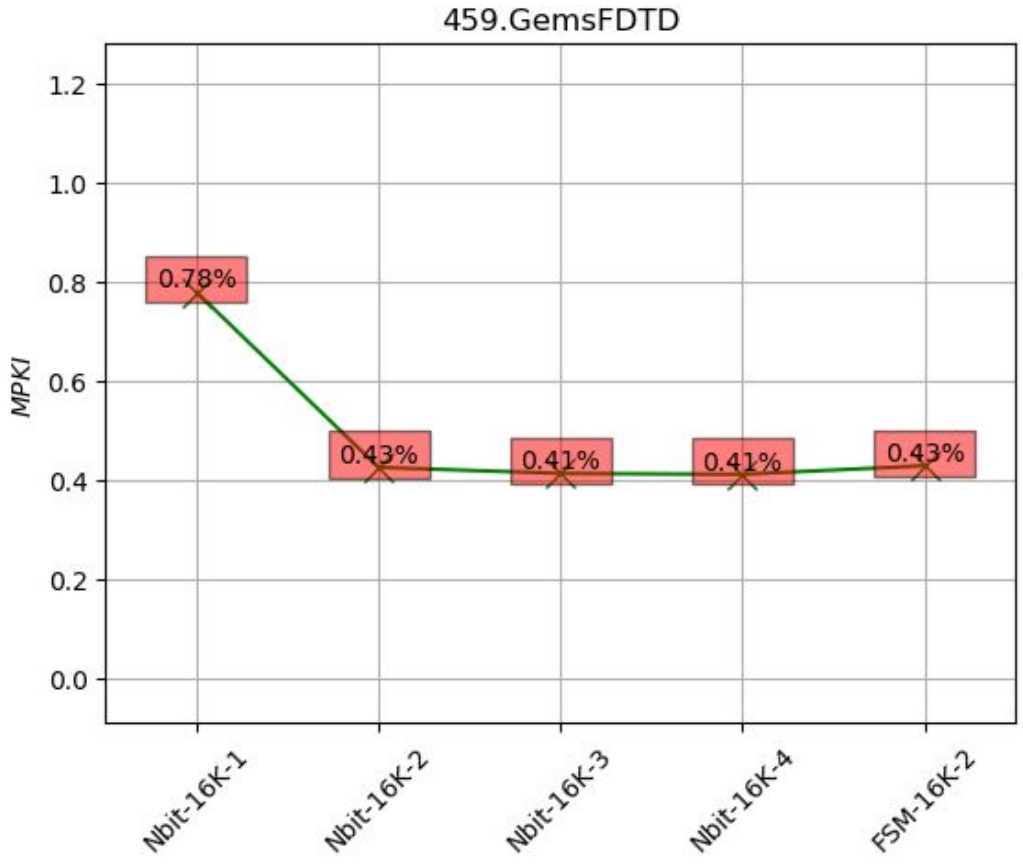

436.cactusADM
MPKI
0.4
0.2
0.0
−0.2
−0.4
0.07%
0.04%
0.04%
0.04%
0.04%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2

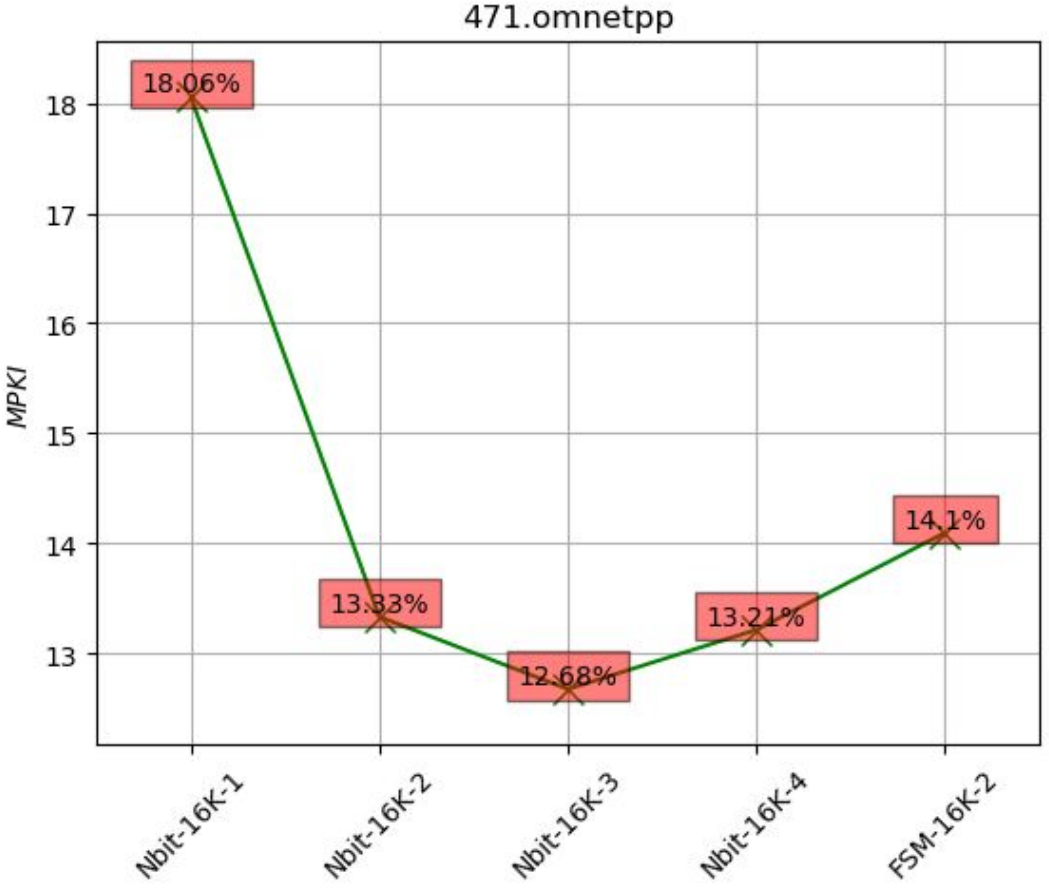

445.gobmk
MPKI
32
30
28
26
24
32.09%
25.03%
24.04%
24.17%
25.76%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2

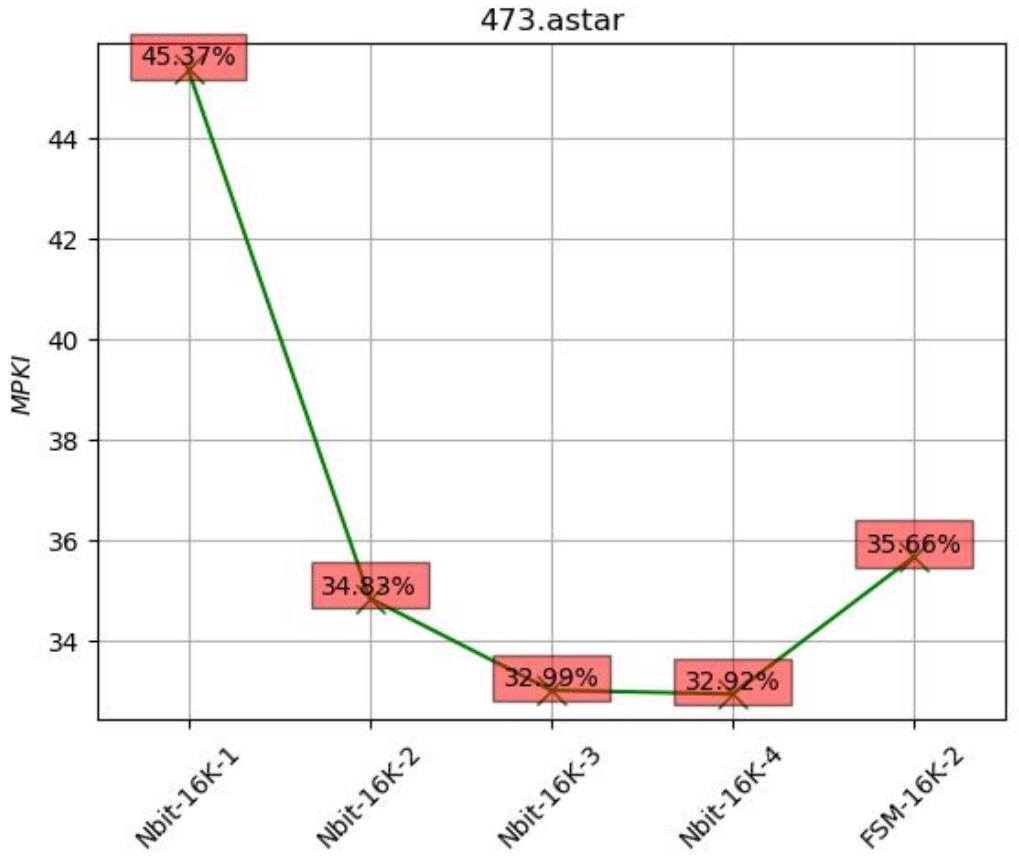

450.soplex
MPKI
15
14
13
12
11
15.17%
11.47%
11.08%
11.21%
11.75%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2

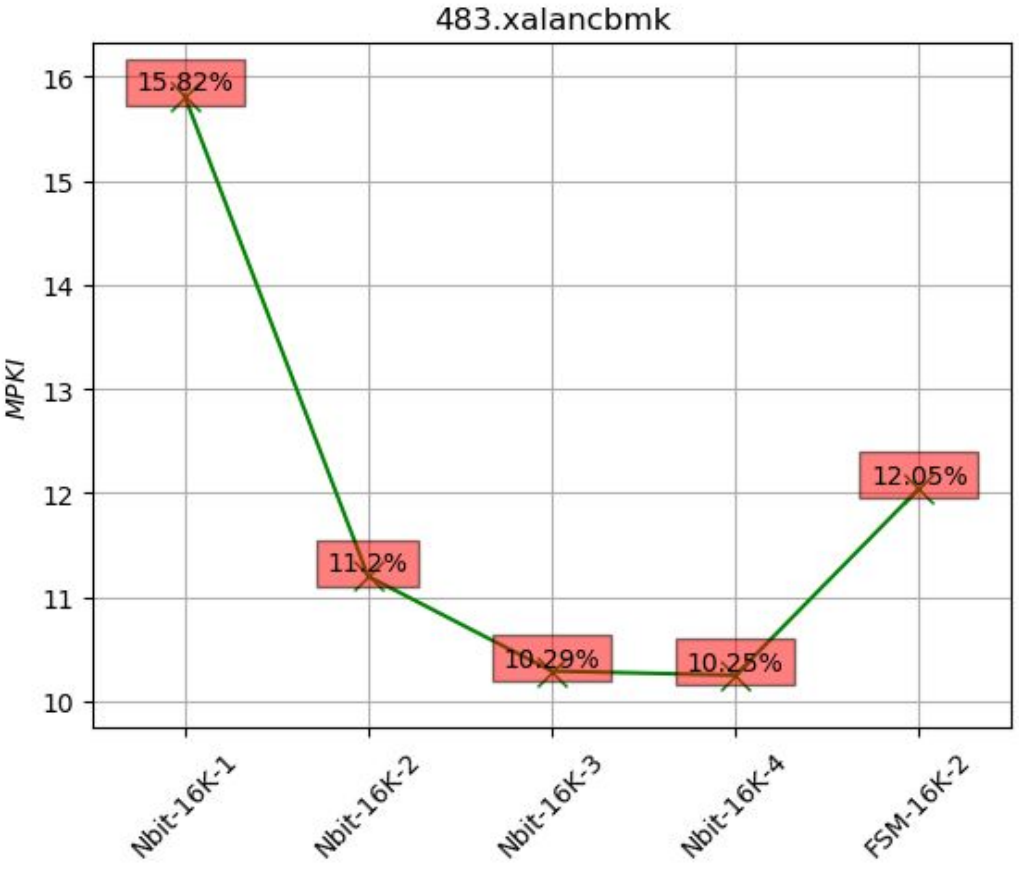

456.hmmer
MPKI
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
1.35%
0.74%
0.73%
0.73%
0.75%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2


458.sjeng
MPKI
34
32
30
28
26
24
34.15%
25.92%
24.8%
24.49%
27.01%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2


459.GemsFDTD
MPKI
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.78%
0.43%
0.41%
0.41%
0.43%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2


471.omnetpp
MPKI
18
17
16
15
14
13
18.06%
13.33%
12.68%
13.21%
14.1%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2


473.astar
MPKI
44
42
40
38
36
34
45.37%
34.83%
32.99%
32.92%
35.66%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2


483.xalancbmk
MPKI
16
15
14
13
12
11
10
15.82%
11.2%
10.29%
10.25%
12.05%
Nbit-16K-1
Nbit-16K-2
Nbit-16K-3
Nbit-16K-4
FSM-16K-2

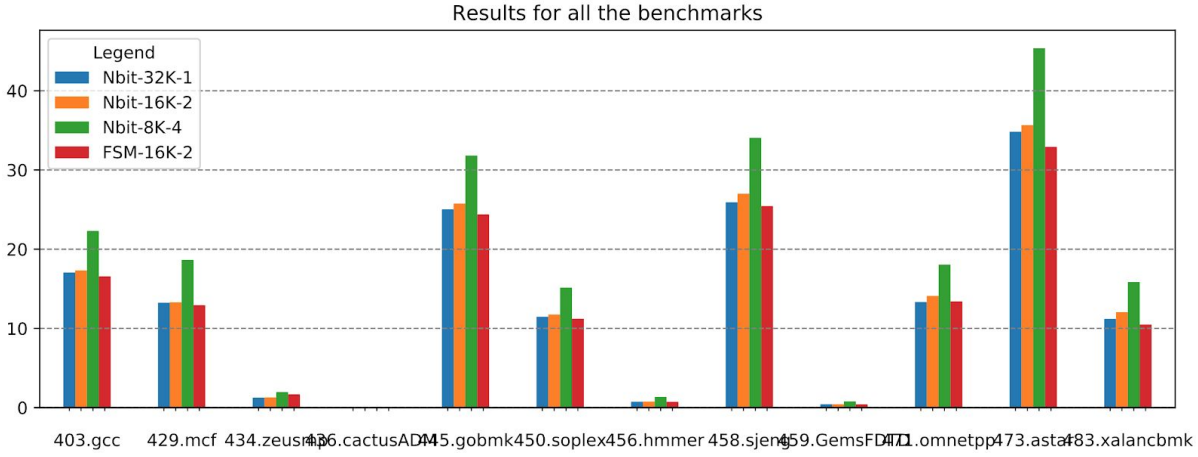
4.2.2 Συμπεράσματα 4.2α:

- Για bit n = 1 bit, παρατηρούμε ότι σχεδόν όλα τα benchmarks έχουν την χειρότερη δυνατή απόδοση, καθώς το MPKI αγγίζει μεγαλύτερες τιμές για n = 1 συγκριτικά με οποιοδήποτε άλλο αριθμό bits. Αναλυτικά, η μετάβαση σε n = 2 bit οδηγεί σε μεγάλη βελτίωση της απόδοσης, ενώ σε n = 3, 4 bits δεν παρατηρείται σημαντική βελτίωση αντίθετα η απόδοση διατηρείται σταθερή (δηλαδή και ο δείκτης MPKI) ή και χειροτερεύει για τα περισσότερα benchmarks (αρκετά μικρή χειροτέρευση)
- Το fsm-16K-2 παρουσιάζει παραπλήσια απόδοση με το αντίστοιχο n bit predictor για Nbit-16K-2, αφού έχουν αντίστοιχα χαρακτηριστικά ,ίδιο hardware και ίδιο n, και πολύ μικρή αλλαγή στην διαδικασία update που τα κάνει να έχουν παραπλήσια συμπεριφορά.
- Πιο αναλυτικά, το fsm-16K-2 παρουσιαζει τις περισσότερες φορές λίγο χειρότερη συμπεριφορά από το Nbit-16K-2. Έτσι και η μετάβαση από Nbit-16K-1 σε fsm-16K-2 παρουσιάζει σχεδόν πάντα σε μεγάλο βαθμό βελτίωσης της απόδοσης.
- Τα χαμηλότερα ποσοστά mispredictions σημειώνονται για τα benchmarks (Κάτω από 2%) 434.zeusmp, 436.cactusADM, 456.hmmer, 459.GemsFDTD.
  Μάλιστα, η τάξη MPKI που σημειώνουν αυτά τα 4 benchmarks είναι αρκετά μικρότερη συγκριτικά με τα υπόλοιπα 6 benchmarks.
- Τέλος, η αύξηση του αριθμού των bits ισοδυναμεί με αύξηση του απαιτούμενου hardware, αφού ο αριθμός των entries του BHT παραμένει σταθερός.

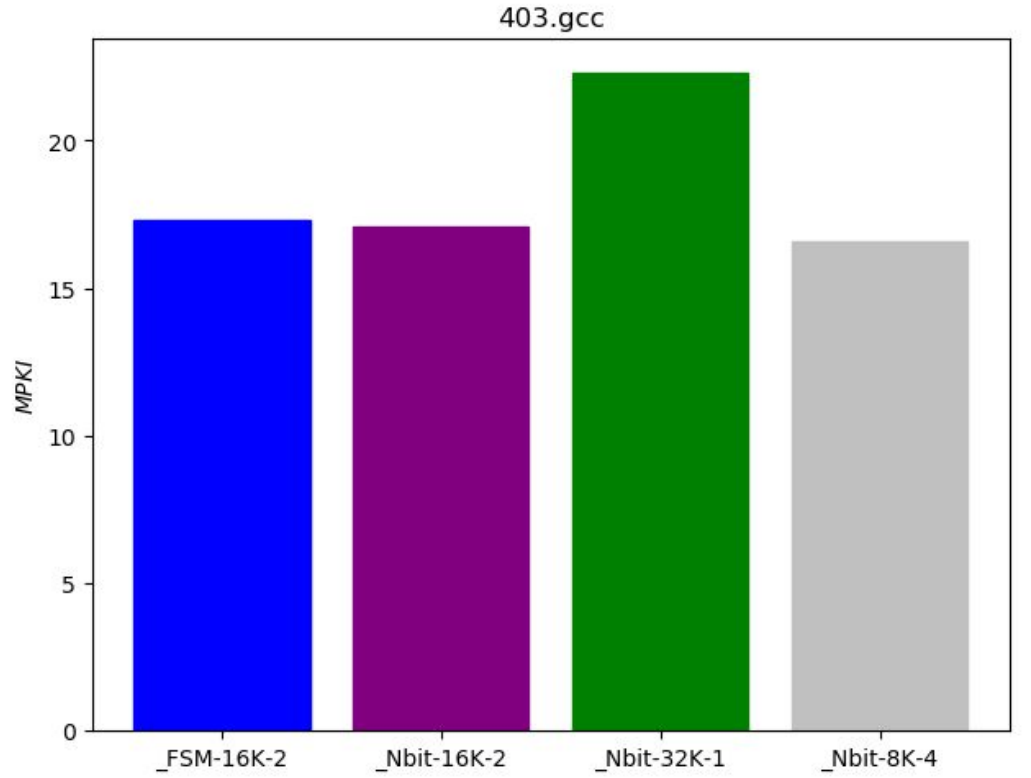
4.2.3 Μελέτη Β περίπτωσης:

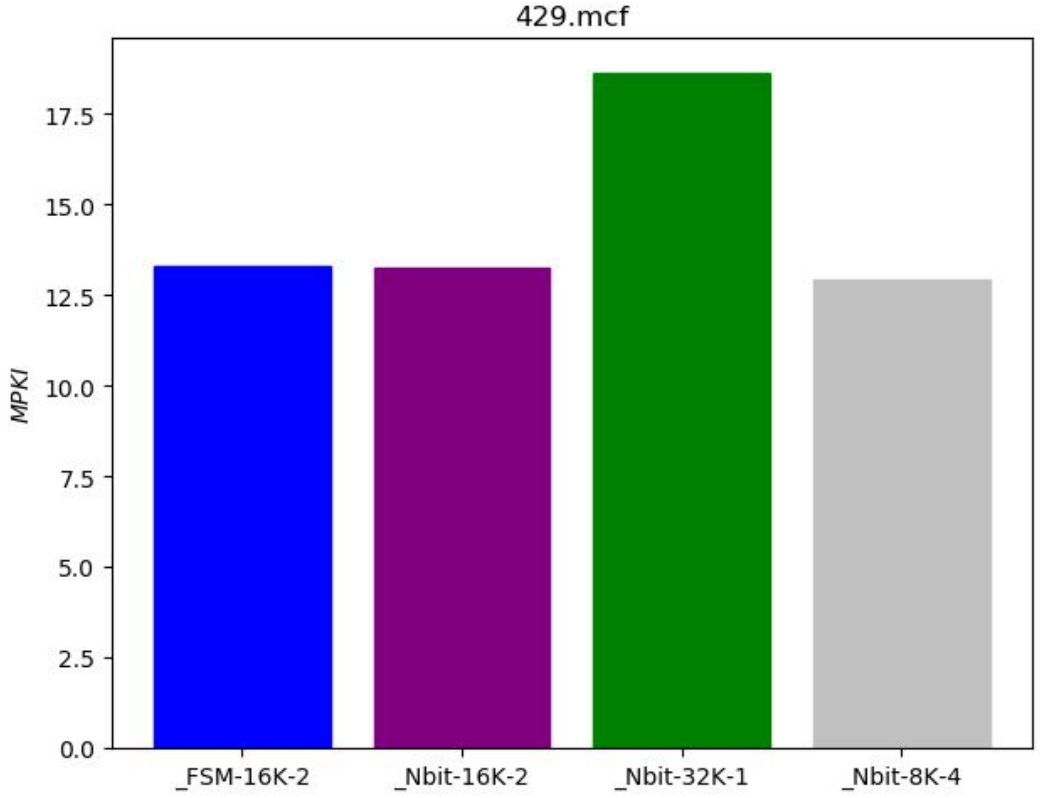
Διατηρώντας τώρα σταθερό το hardware και ίσο με 32K bits, εκτελούμε ξανά τις προσομοιώσεις για τα 12 benchmarks, θέτοντας για τον NBit predictor N=1,2,4 και για το FSM N=2.

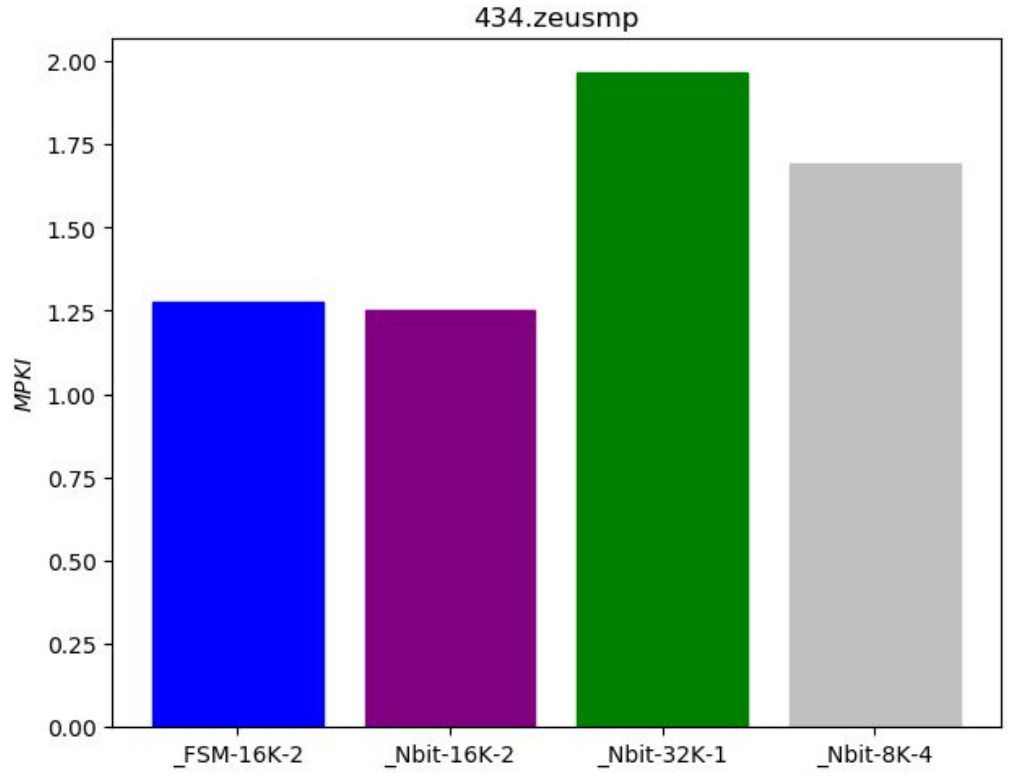
Το συγκεντρωτικό αποτέλεσμα παρουσιάζεται παρακάτω:

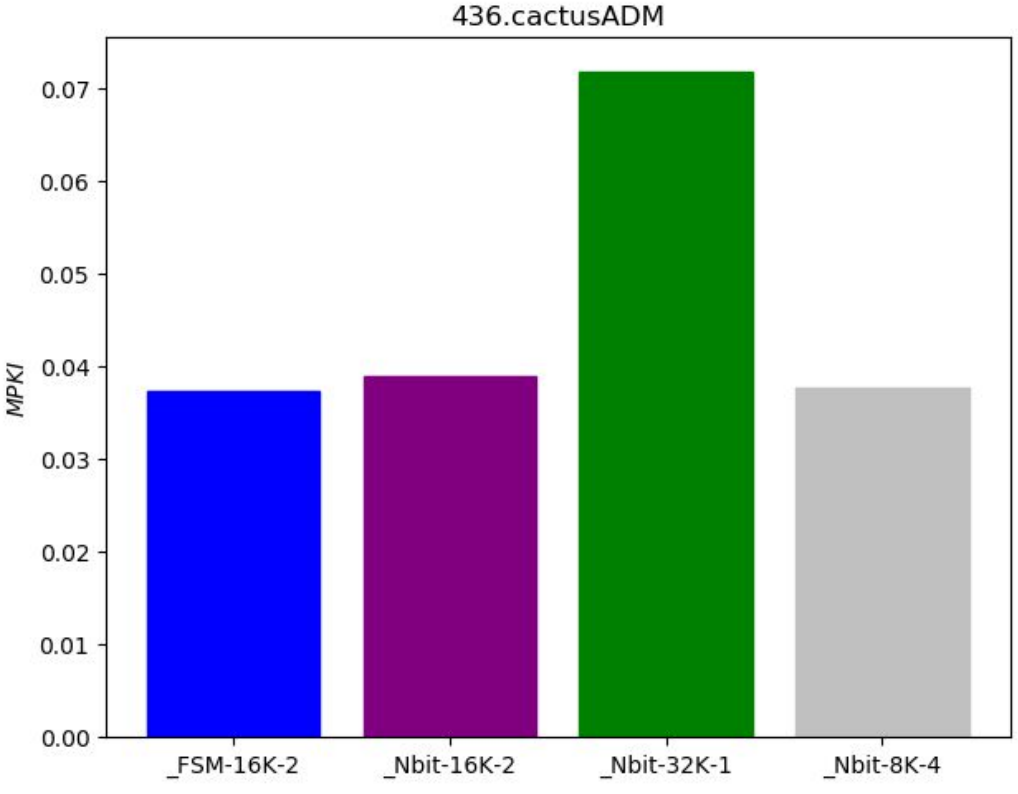


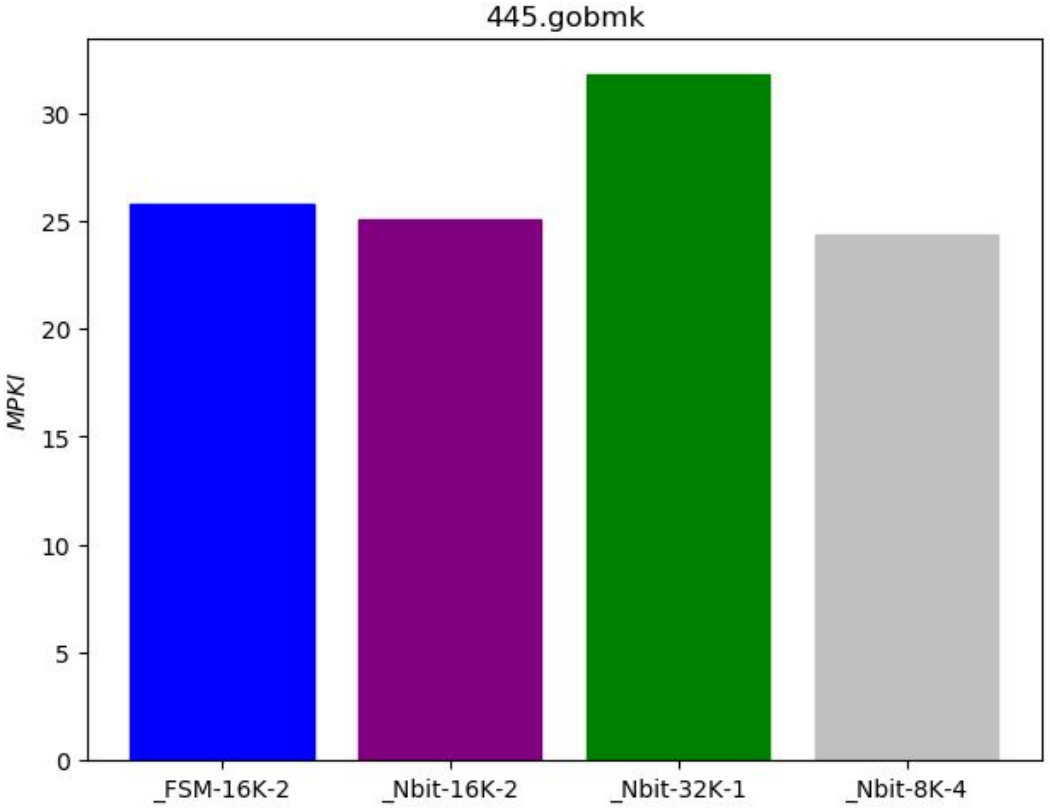
Στην συνέχεια παρουσιάζεται το αποτέλεσμα για καθε ενα απο τα επιμέρους benchmarks έτσι ώστε να έχουμε πιο λεπτομερή μελέτη.

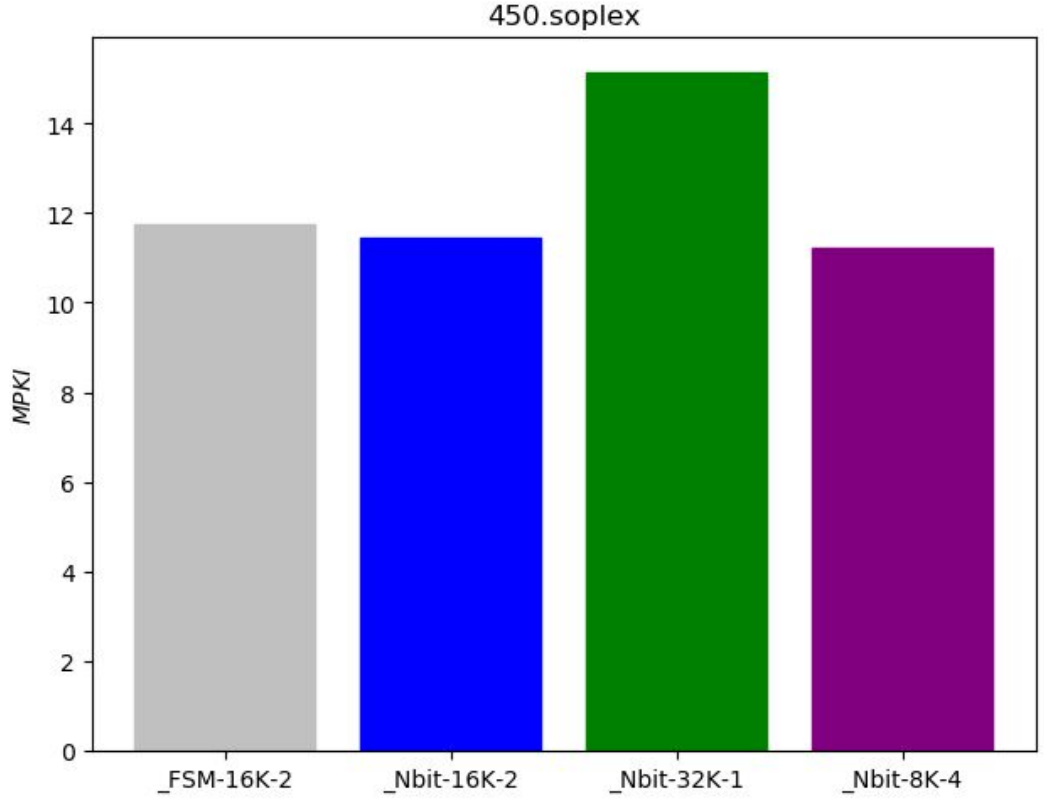

403.gcc
MPKI
0
5
10
15
20
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4

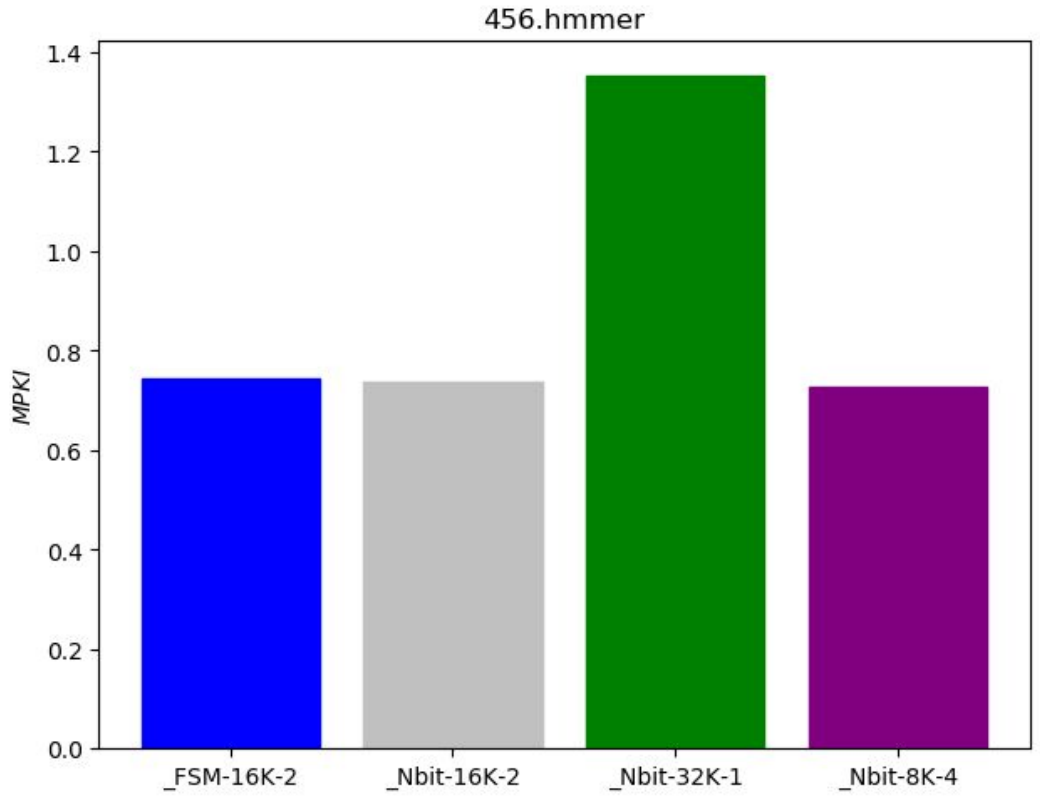

429.mcf
MPKI
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4

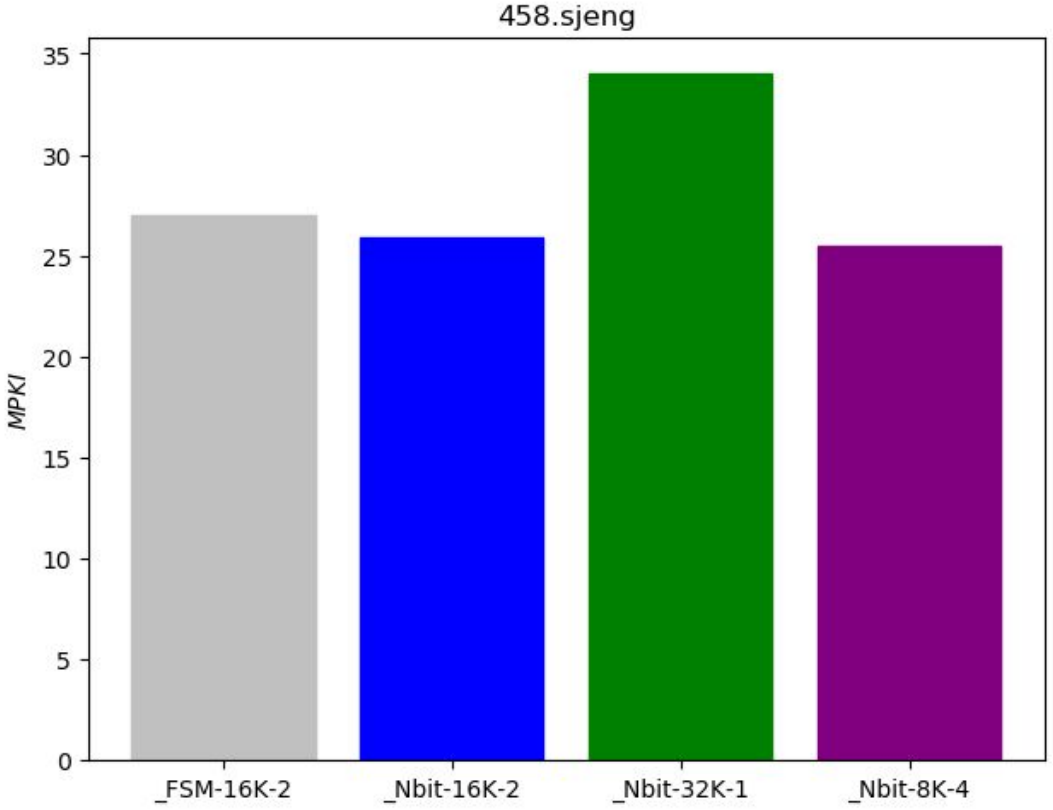

434.zeusmp
MPKI
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4

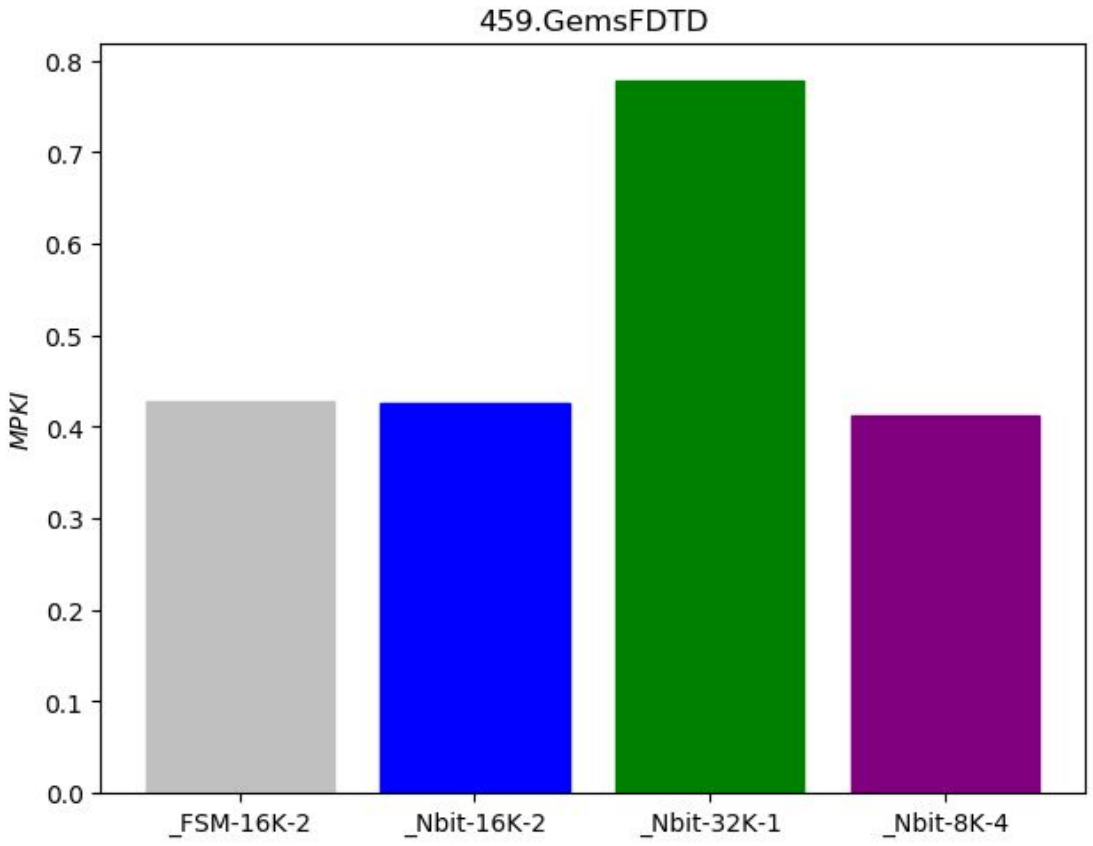

436.cactusADM
MPKI
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4

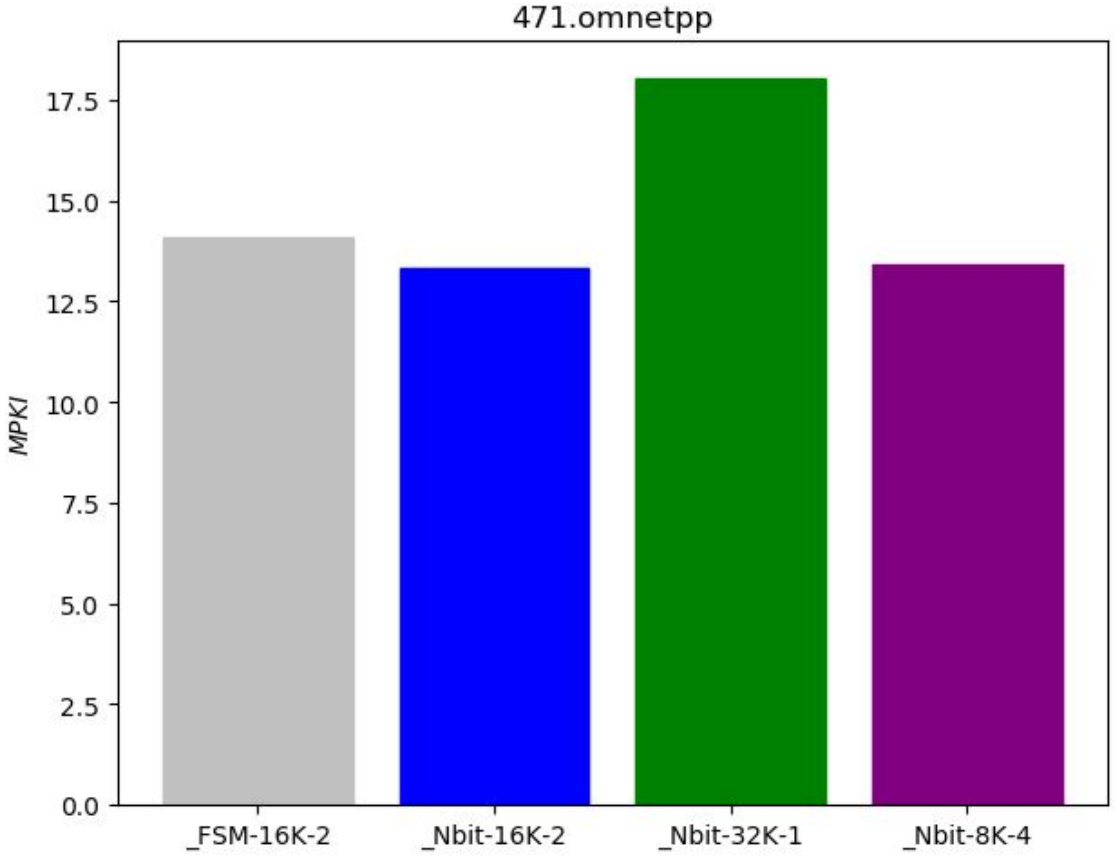

445.gobmk
MPKI
0
5
10
15
20
25
30
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4

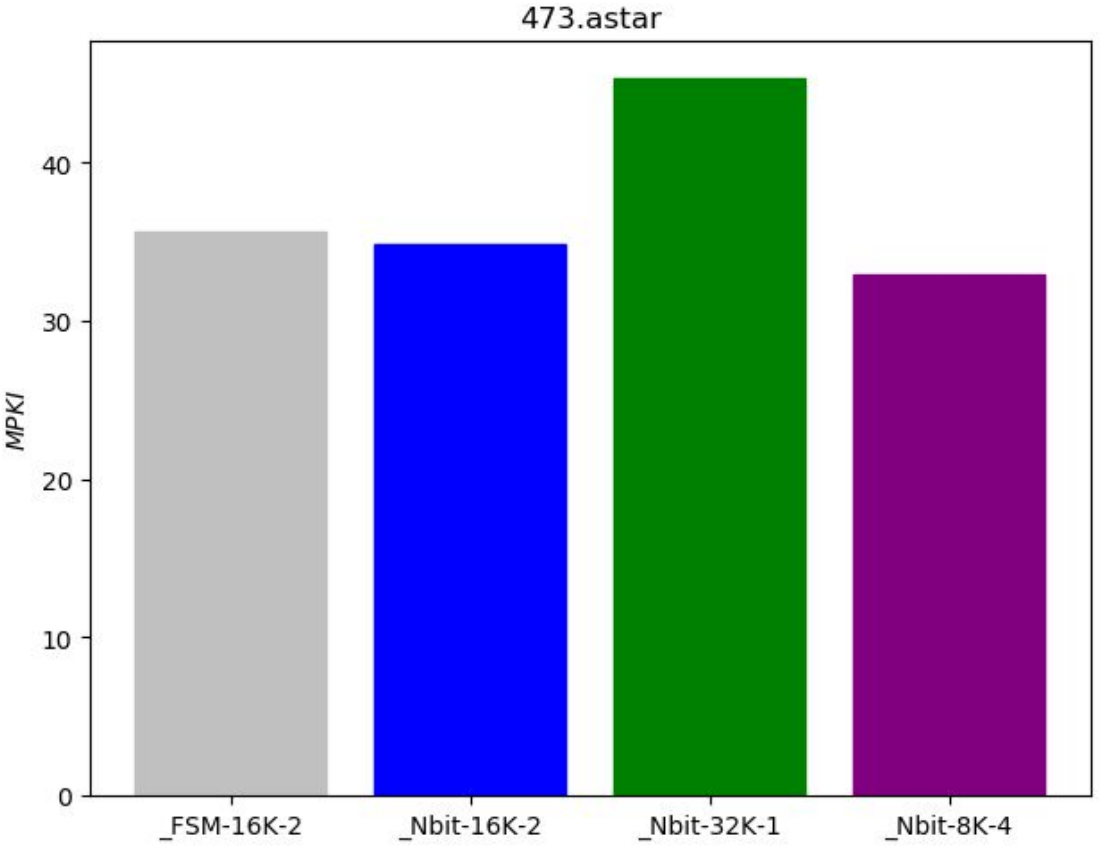

450.soplex
MPKI
0
2
4
6
8
10
12
14
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4

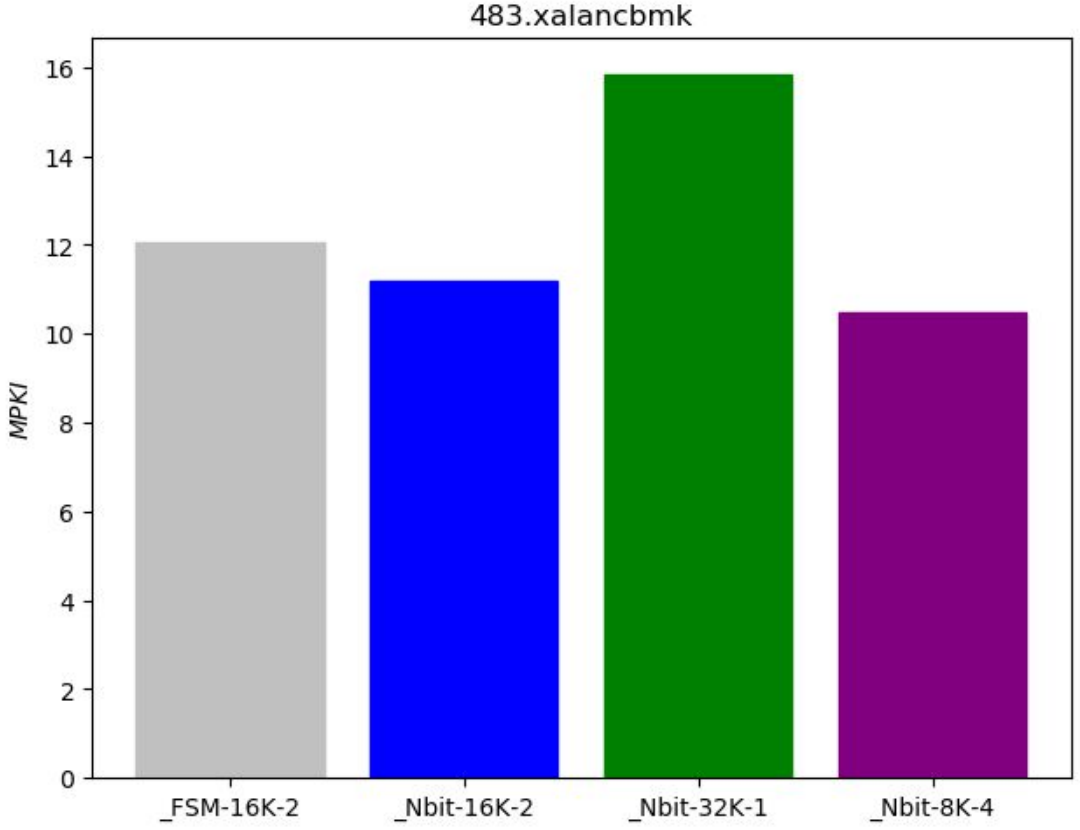

456.hmmer
MPKI
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4


458.sjeng
MPKI
35
30
25
20
15
10
5
0
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4


459.GemsFDTD
MPKI
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4


471.omnetpp
MPKI
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4


473.astar
MPKI
0
10
20
30
40
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4


483.xalancbmk
MPKI
0
2
4
6
8
10
12
14
16
_FSM-16K-2
_Nbit-16K-2
_Nbit-32K-1
_Nbit-8K-4

Συμπεράσματα για το 4.2β:

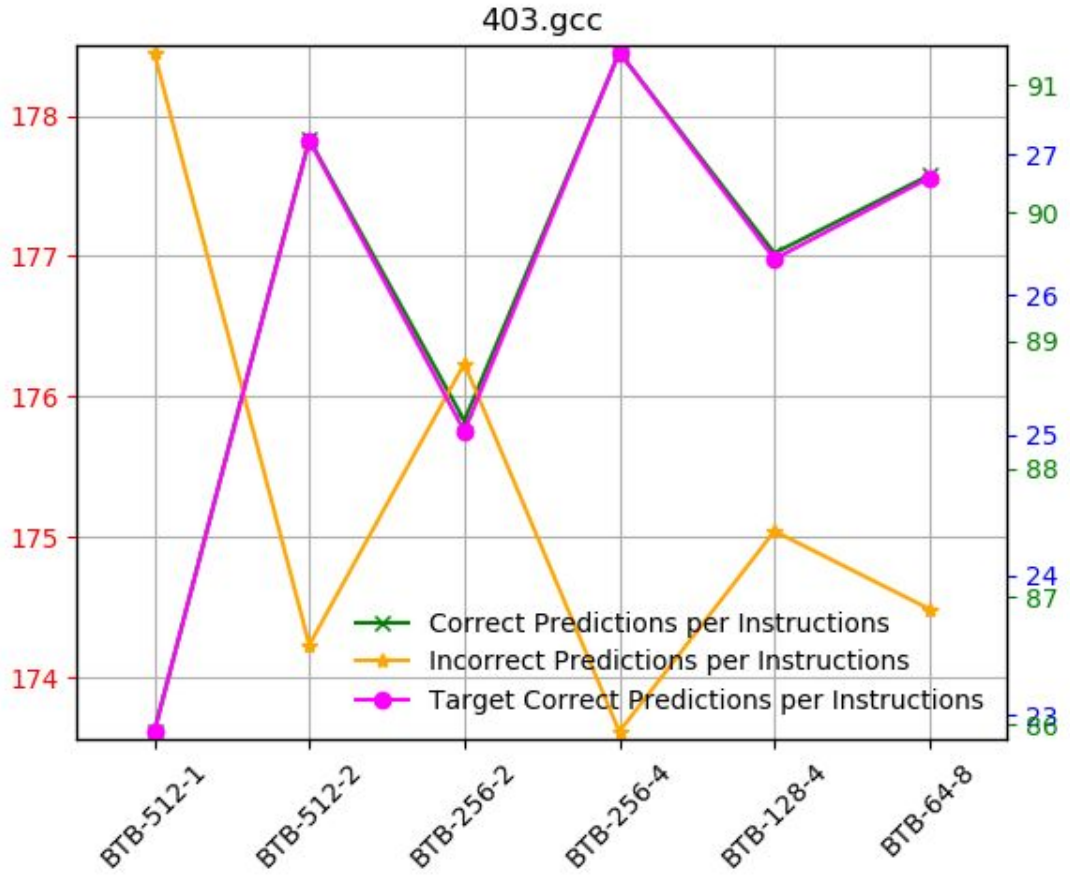
- Σημειώνεται πως η αύξηση του bits συνδέεται άρρηκτα και πλήρως με την αύξηση του απαιτούμενου hardware που απαιτεί η εφαρμογή, καθώς ο αριθμός των BHT entries διατηρείται σταθερός. Επομένως, αυτή η προσομοίωση είναι πιο ρεαλιστική και κοντά στην πραγματικότητα από ότι η προσομοίωση του πρώτου ερωτήματος.
- Όπως και προηγουμένως ο predictor για n = 1 bit έχει την χειρότερη απόδοση σε όλα τα benchmarks. Επιπλέον, η απόδοση για n = 2, n = 4 bits και για τον fsm κυμαίνεται στα ίδια επίπεδα, αλλά είναι σχετικά καλύτερη από αυτή για n = 1 bit.
- Ο fsm-16K-2 που παρουσιάζει όπως και στο προηγούμενο ερώτημα μεγάλες ομοιότητες στην απόδοση με τον Nbit-16K-2. Ωστόσο ο nbit έχει ελαφρώς καλύτερη απόδοση.
- Βάσει της παραπάνω διερεύνησης και αποτελεσμάτων, επιλέγουμε τον **Nbit predictor για n=2** καθώς κατά μέσο όρο παρουσιάζει σχετικά καλύτερη απόδοση από όλους τους υπόλοιπους! Αξίζει να αναφερθεί πάντως ότι κατά μια γενική ομολογία, στην πραγματικότητα αυτό που χρησιμοποιείται είναι κυρίως ένα **συνδυασμό διαφορετικών τύπων** predictors, όπως π.χ. υβριδικό από 2-bit counter και global predictor, το οποίο βελτιώνει την απόδοση, αλλά οδηγεί σε αυξημένες καθυστερήσεις(tradeoff).
- Δεύτερη επιλογή θα είναι ο fsm-16K-2 αφού παρουσιάζει μεγάλες ομοιότητες με τον predictor της πρώτης επιλογής μας.

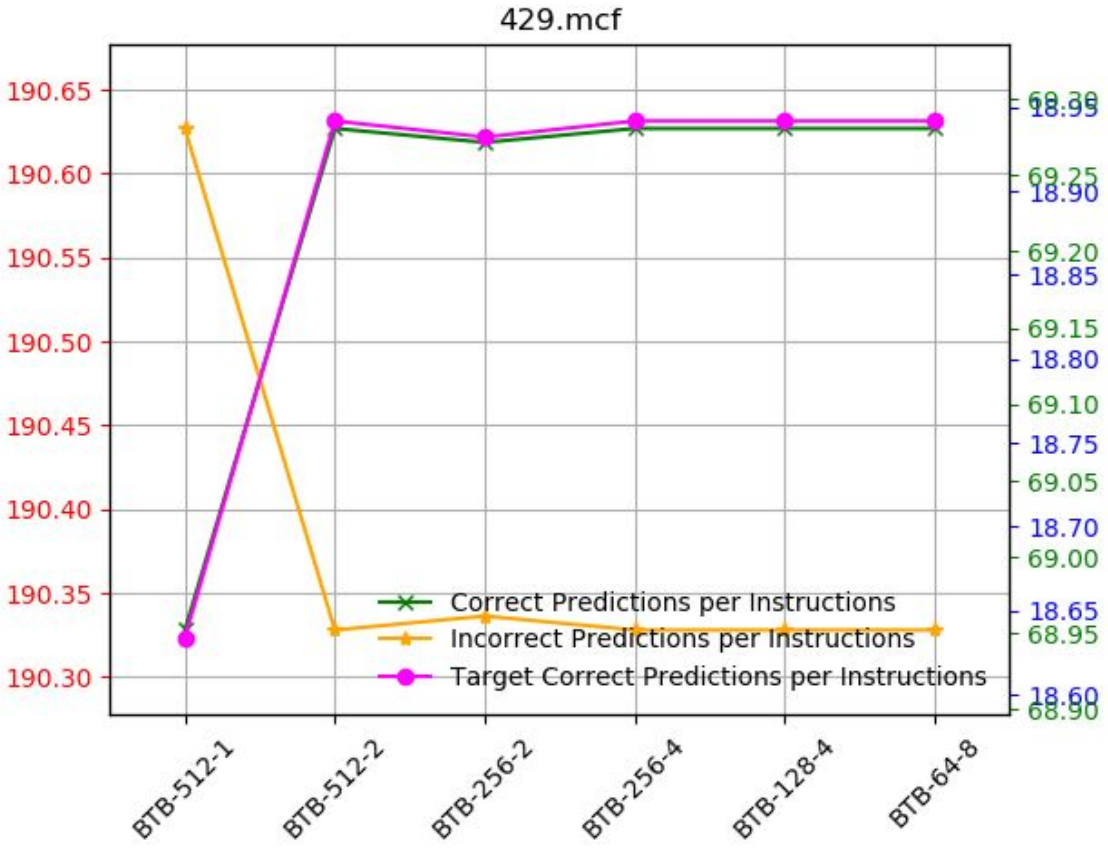
3.3 Μελέτη του BTB

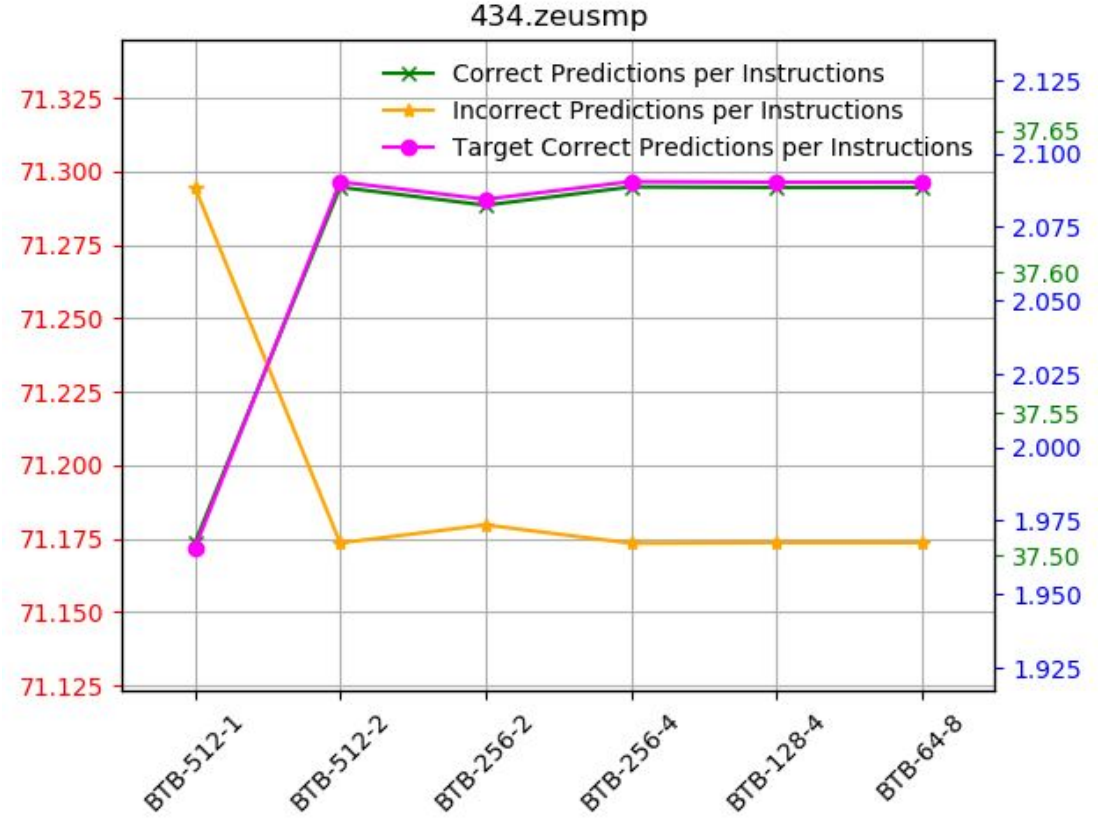
Στο Τρίτο Μέρος της πειραματικής αξιολόγησης για την παρούσα άσκηση, εξετάζουμε την ακρίβεια πρόβλεψης κάνοντας χρήση ενός Branch Target Buffer (BTB). Συγκεκριμένα, εκτελούμε προσομοιώσεις για τις παρακάτω περιπτώσεις:

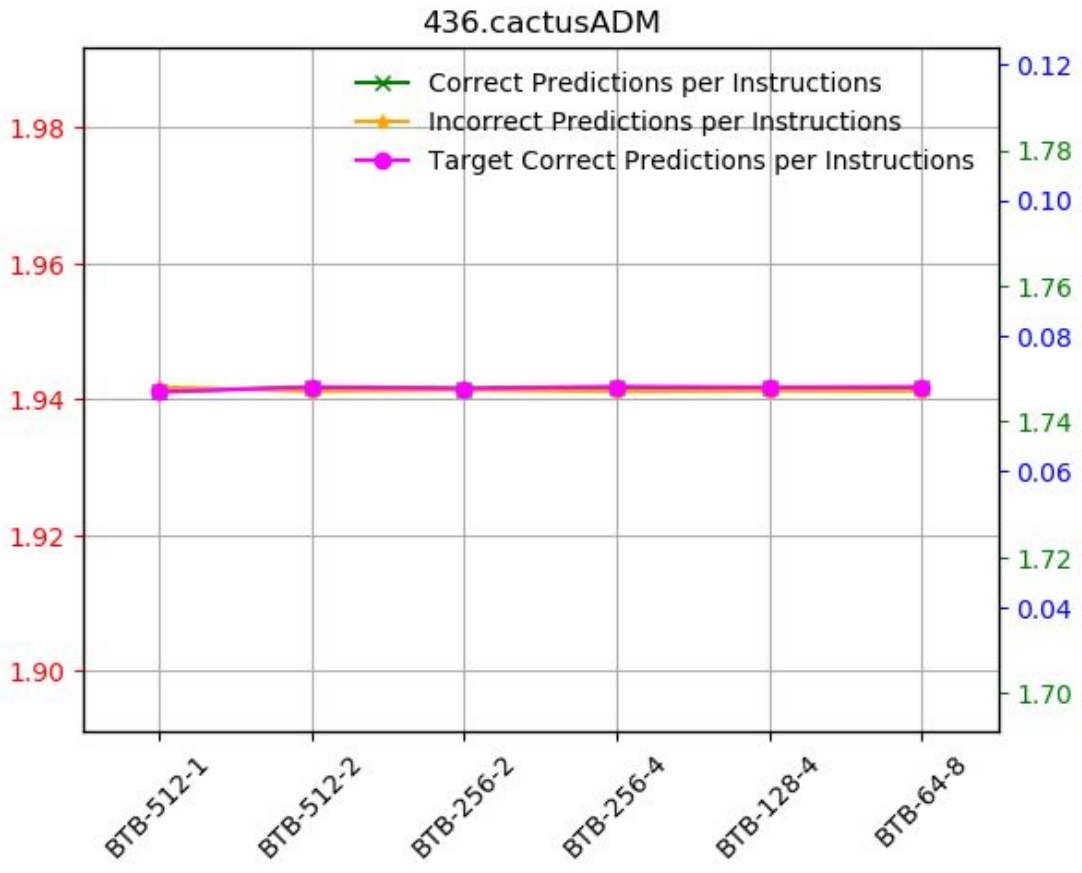
btb entries	btb associativity
512	1, 2
256	2, 4
128	4
64	8

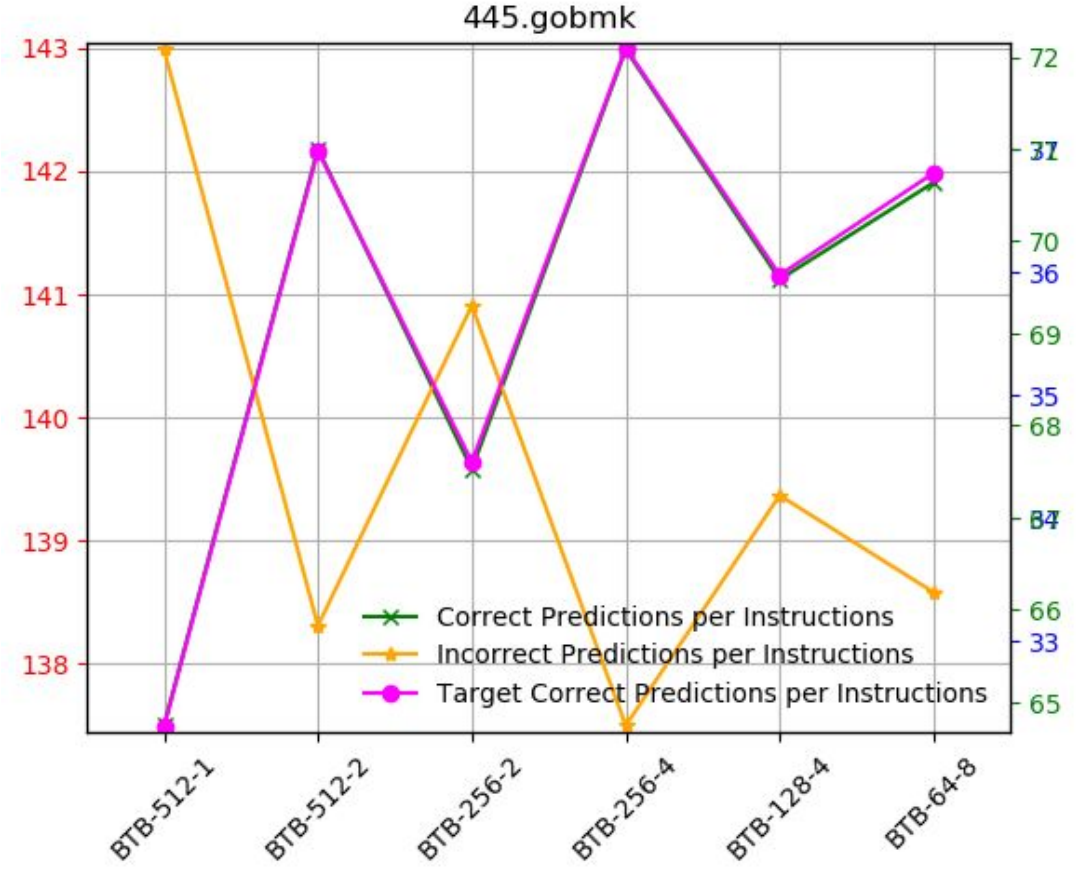
Στην συνέχεια παρουσιάζεται το αποτέλεσμα για καθε ενα απο τα επιμέρους benchmarks έτσι ώστε να έχουμε πιο λεπτομερή μελέτη.

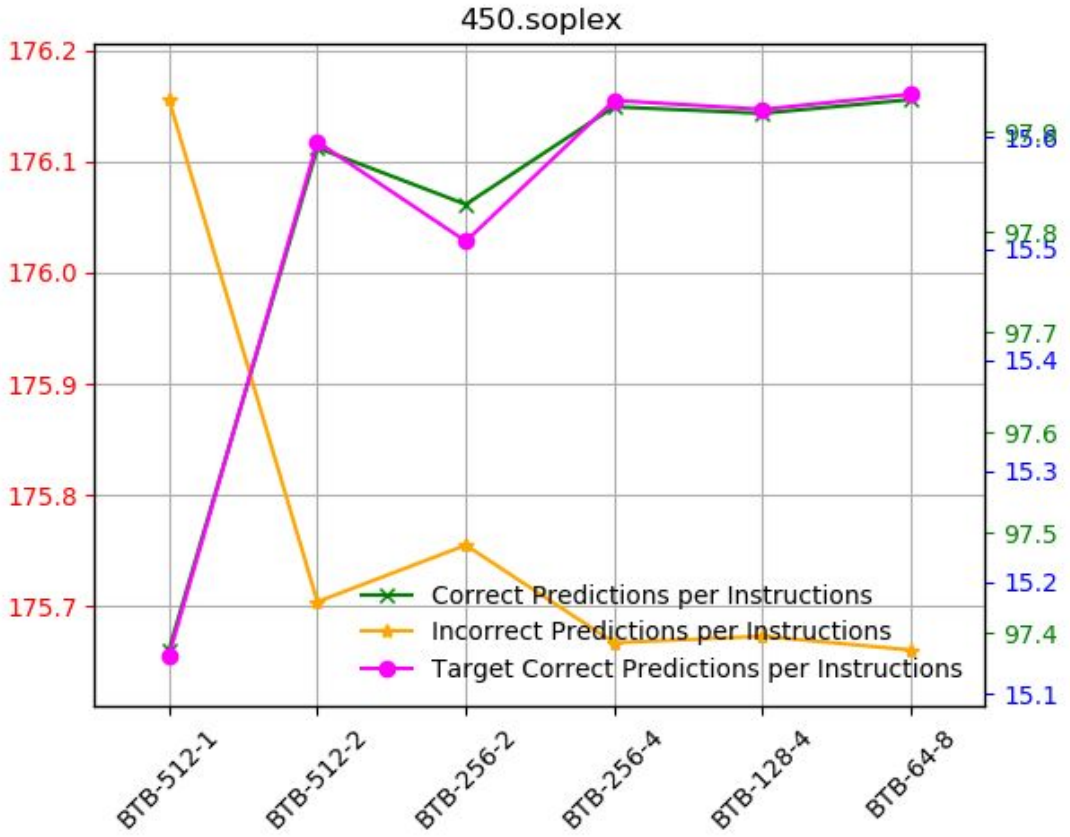


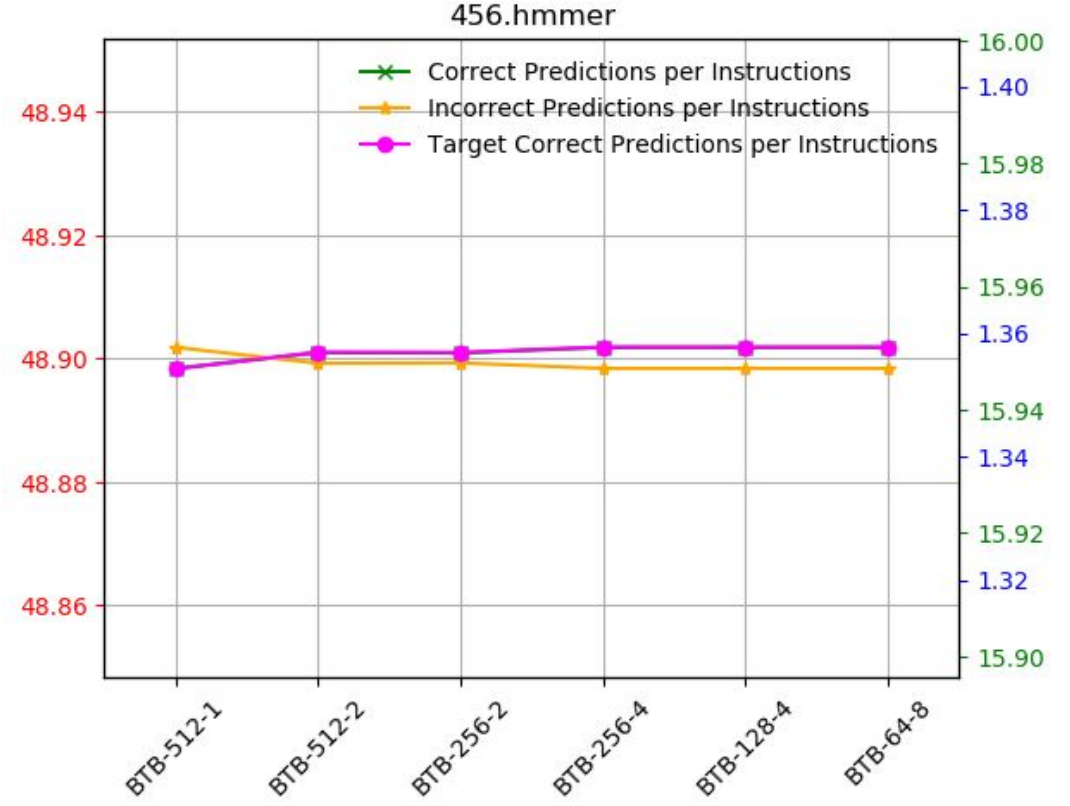

429.mcf
190.65
190.60
190.55
190.50
190.45
190.40
190.35
190.30
69.30
18.95
69.25
18.90
69.20
18.85
69.15
18.80
69.10
18.75
69.05
18.70
69.00
18.65
68.95
18.60
68.90
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8

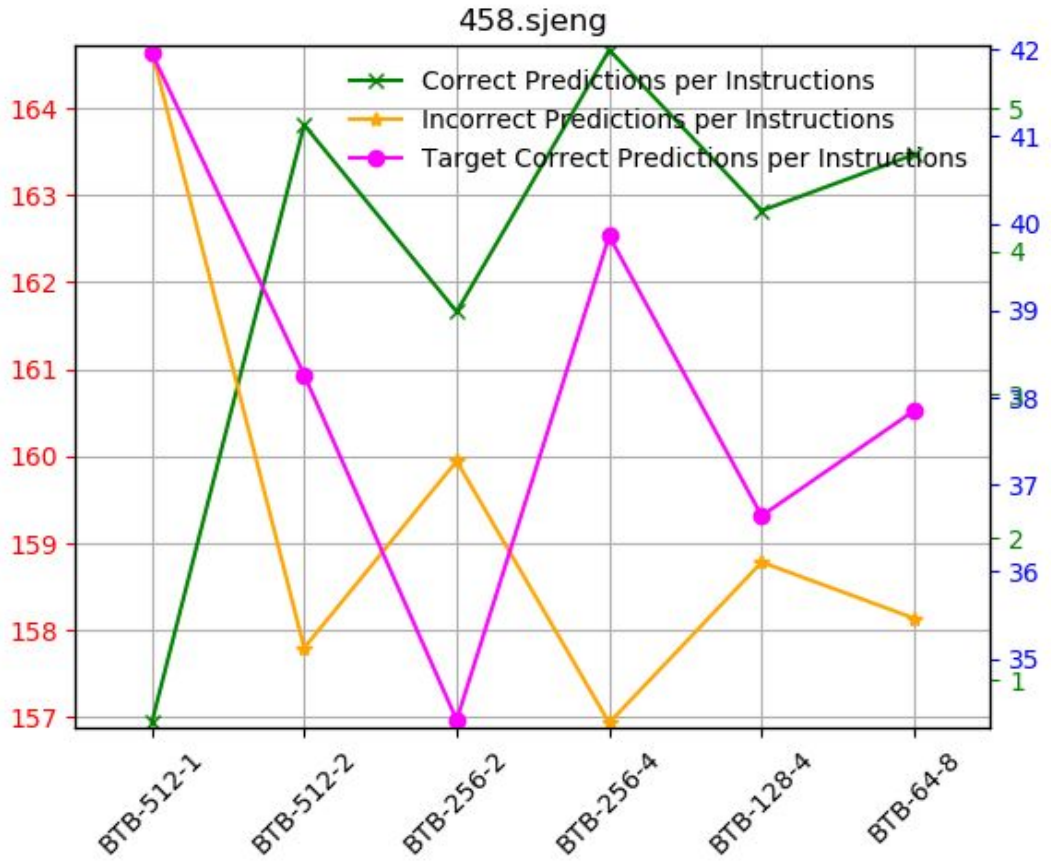

434.zeusmp
71.325
71.300
71.275
71.250
71.225
71.200
71.175
71.150
71.125
2.125
37.65
2.100
2.075
37.60
2.050
2.025
37.55
2.000
1.975
37.50
1.950
1.925
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8

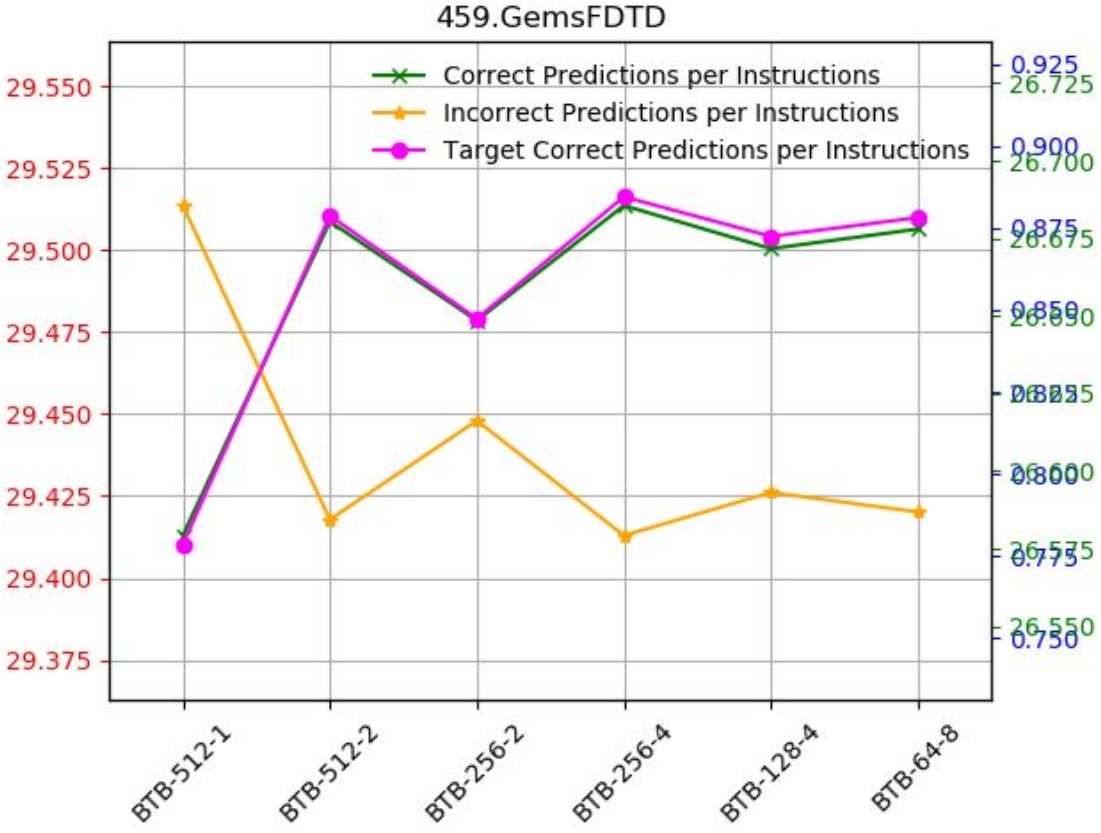

436.cactusADM
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
1.98
1.96
1.94
1.92
1.90
0.12
1.78
0.10
1.76
0.08
1.74
0.06
1.72
0.04
1.70
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8

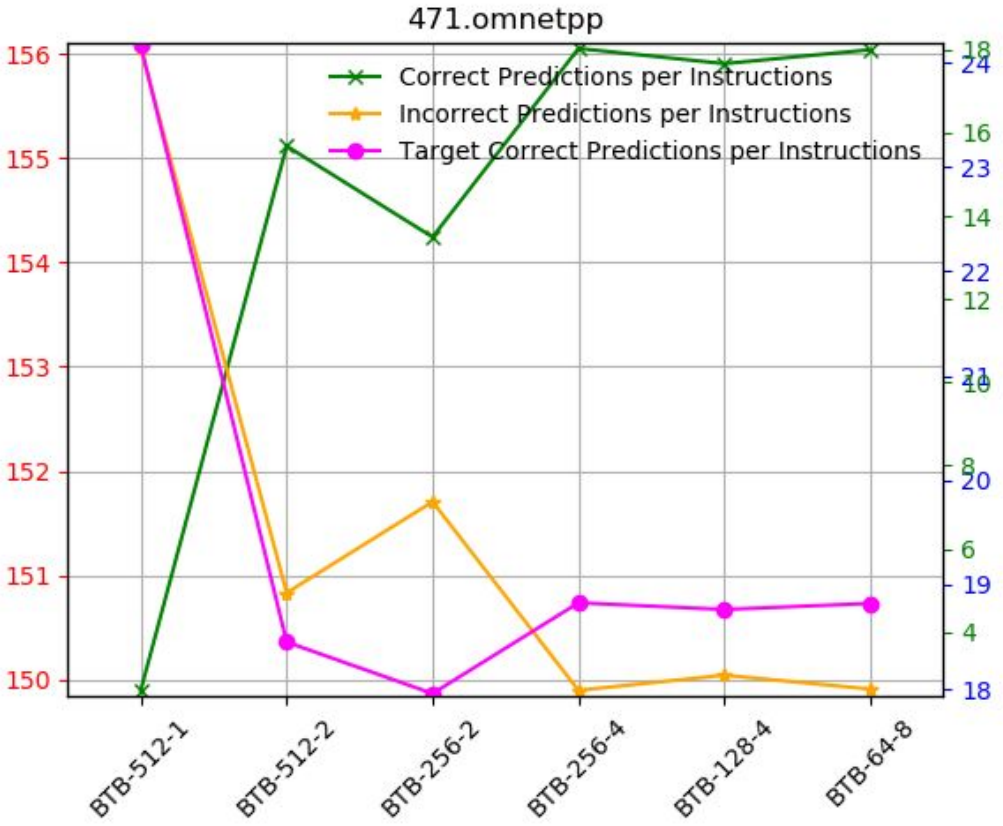

445.gobmk
143
142
141
140
139
138
72
70
36
69
35
68
66
33
65
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8

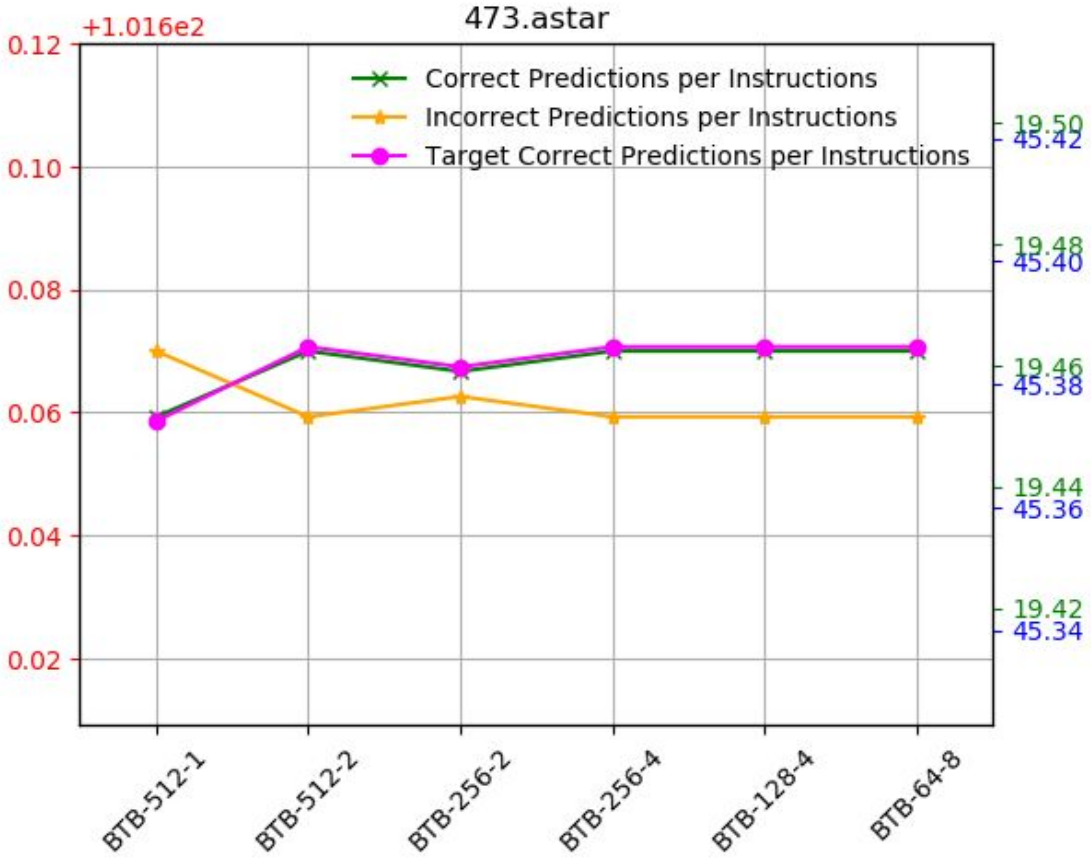

450.soplex
176.2
176.1
176.0
175.9
175.8
175.7
97.8
97.7
97.6
97.5
97.4
15.5
15.4
15.3
15.2
15.1
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8

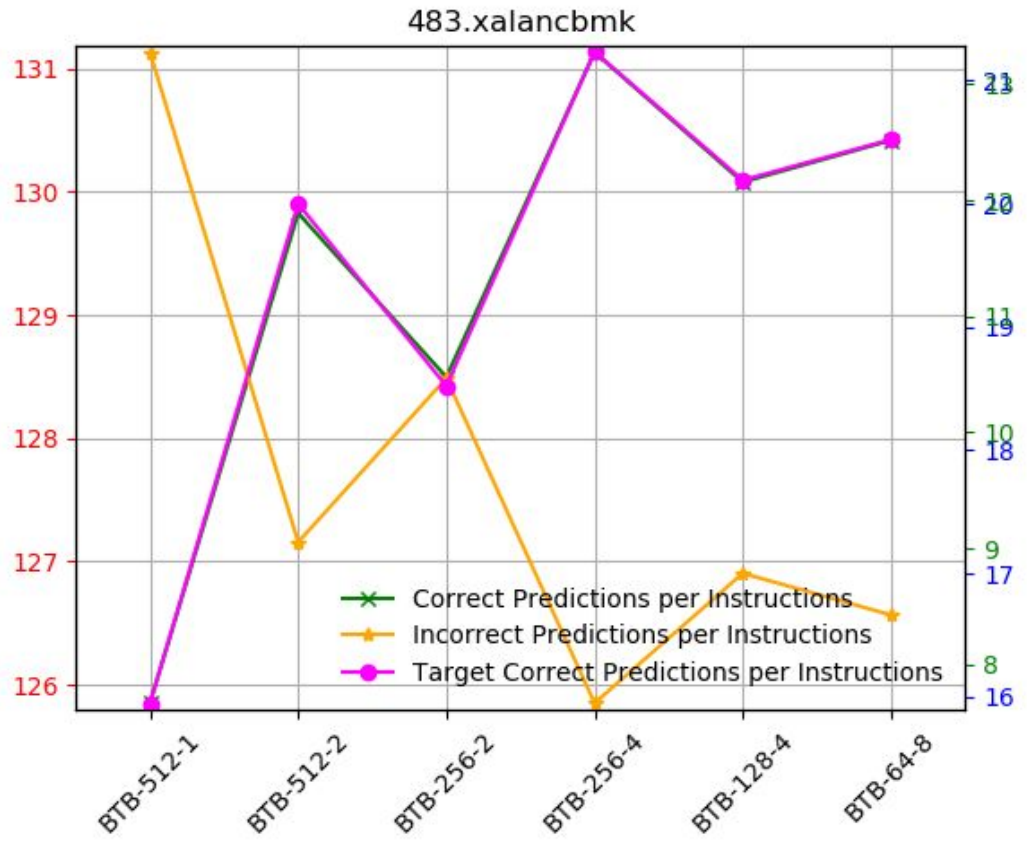

456.hmmer
48.94
48.92
48.90
48.88
48.86
16.00
1.40
15.98
1.38
15.96
1.36
15.94
1.34
15.92
1.32
15.90
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8


458.sjeng
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
164
163
162
161
160
159
158
157
42
5
41
40
4
39
38
37
2
36
35
1
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8


459.GemsFDTD
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
29.550
29.525
29.500
29.475
29.450
29.425
29.400
29.375
0.925
26.725
0.900
26.700
0.875
26.675
0.850
0.800
0.775
26.575
26.550
0.750
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8


471.omnetpp
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
156
155
154
153
152
151
150
18
24
16
23
14
22
12
21
10
8
20
6
19
4
18
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8


473.astar
+1.016e2
Correct Predictions per Instructions
Incorrect Predictions per Instructions
Target Correct Predictions per Instructions
0.12
0.10
0.08
0.06
0.04
0.02
19.50
45.42
19.48
45.40
19.46
45.38
19.44
45.36
19.42
45.34
BTB-512-1
BTB-512-2
BTB-256-2
BTB-256-4
BTB-128-4
BTB-64-8



3.3.1 Συμπεράσματα του 4.3:

Αρχικά, να σημειωθεί πως τα BTB entries έχουν μεγαλύτερο υπολογιστικό κόστος από BHT entries , αλλά κάνουν redirect fetches σε πολύ νωρίτερο στάδιο στο pipeline και μπορούν να επιταχύνουν τα indirect branches. Δεν ξεχνάμε όμως ότι τα BHT entries διατηρούν περισσότερα entries και έχουν μεγαλύτερη ακρίβεια.

Παρόλα αυτά, αυτή η μέθοδος προβλέψεων είναι αρκετά αποτελεσματική. Παρ' όλα αυτά, παρουσιάζονται σφάλματα λανθασμένου στόχου. Αυτό οφείλεται στο ότι ο πίνακας δεν είχε αποθηκευμένο το στόχο της διακλάδωσης, αν και έγινε σωστή απόφαση για τη πορεία της διακλάδωσης. Επίσης, παρατηρούμε ότι ένα μεγάλο ποσοστό των εντολών είχαν και σωστό στόχο στο BTB

Οι predictors φτάνουν **παρόμοια επίπεδα απόδοσης**,κατά μέσο όρο για οποιονδήποτε από τους 6 παραπάνω συνδυασμούς entries και associativity,και αυτό φαίνεται γραφικά επειδή η ροζ και η πράσινη καμπύλη σχεδόν πάντα ταυτίζονται ή η διαφορά τους κυμαίνεται μεταξύ ενός μικρού εύρους τιμών. Επομένως, θα επιλέγαμε έναν από αυτούς τους έξι με ισοδύναμο τρόπο.

Τέλος, ο τρόπος αντικατάστασης στα μεγάλα associativity όπου μία εντολή δεν αντικαθίσταται αμέσως από μία άλλη εντολή που έχουν την ίδια διεύθυνση στο BTB αλλά χρειάζεται αριθμός εντολών διακλάδωσης με την ίδια θέση στο BTB ίσος με το associativity.

3.4 Μελέτη του RAS

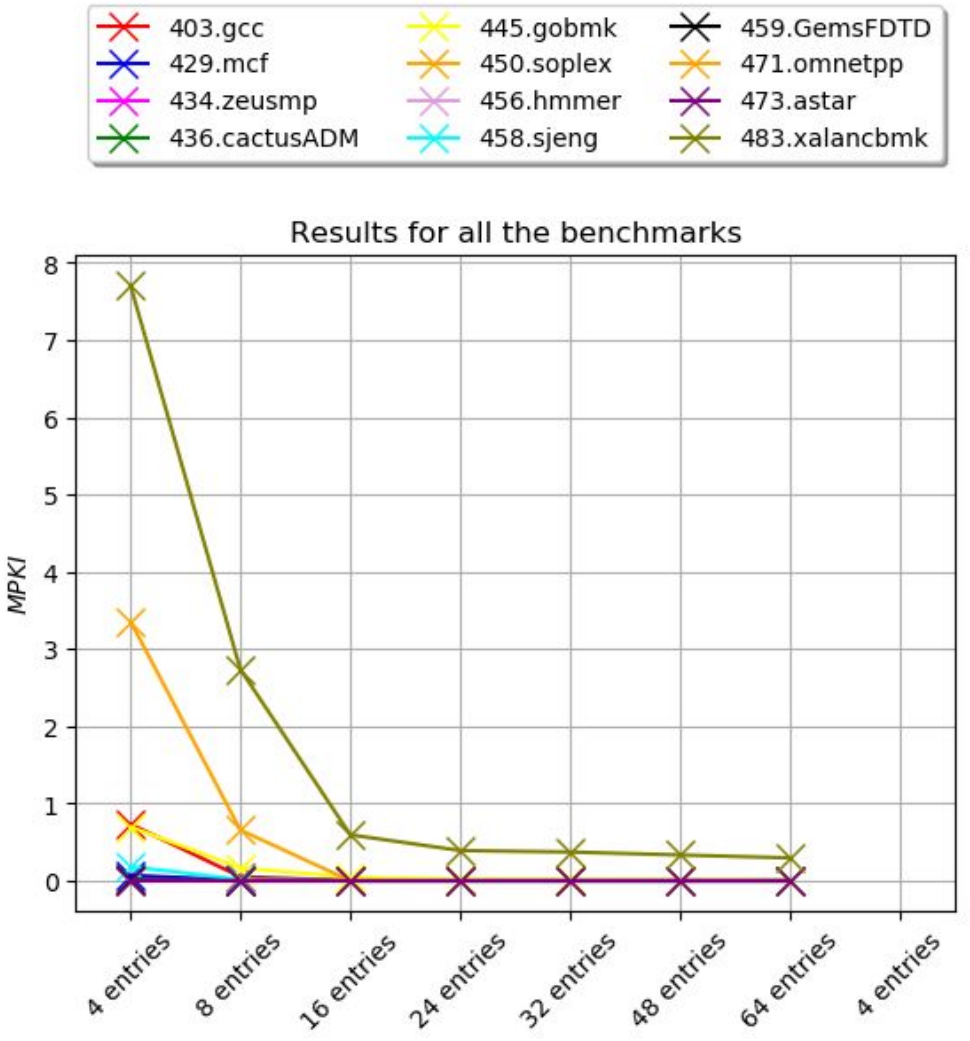
Στο Τέταρτο Μέρος της πειραματικής αξιολόγησης για την παρούσα άσκηση, εξετάζουμε το ποσοστό αστοχίας που προκύπτει κάνοντας χρήση διαφορετικού αριθμού εγγράφων RAS. Για την υλοποίηση της RAS, η οποία είναι πολύ απλή και αποτελεί μέθοδο για την πρόβλεψη εντολών return, χρησιμοποιούμε στο ras.h (με μικρές προσαρμογές) μία στοίβα (FILO) η οποία λειτουργεί ως εξής: Σε κάθε κλήση προσθέτουμε διευθύνσεις και σε κάθε return αφαιρούμε.

Συγκεκριμένα, εκτελούμε προσομοιώσεις για τις παρακάτω περιπτώσεις:

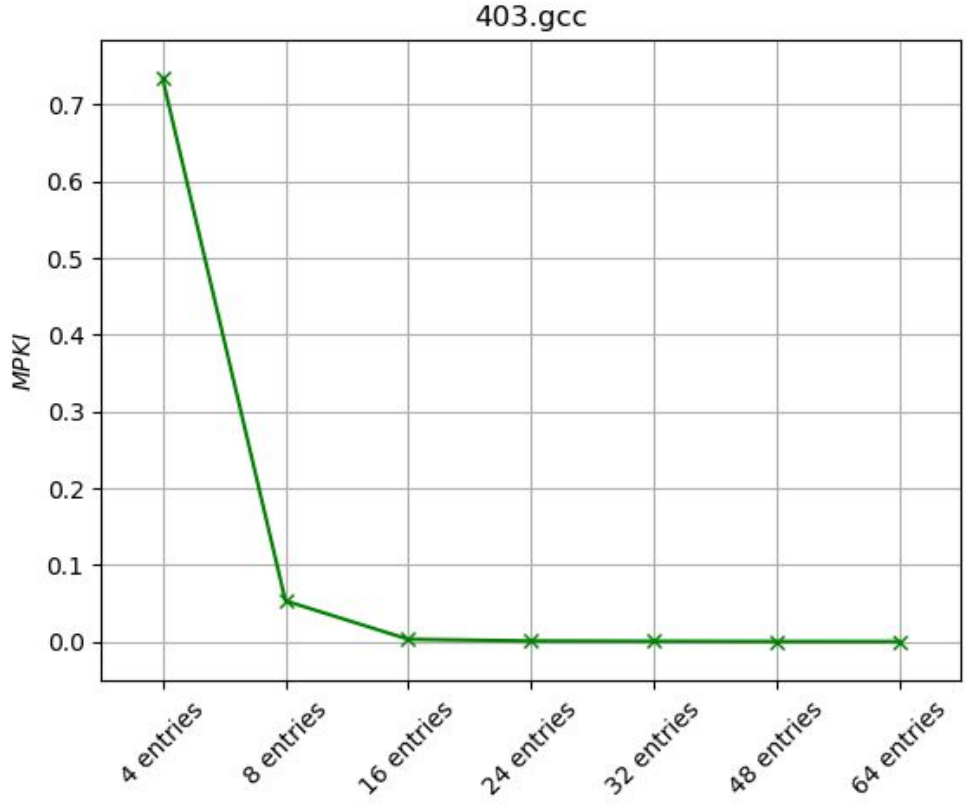
Αριθμός εγγραφών στη RAS
4
8
16
24

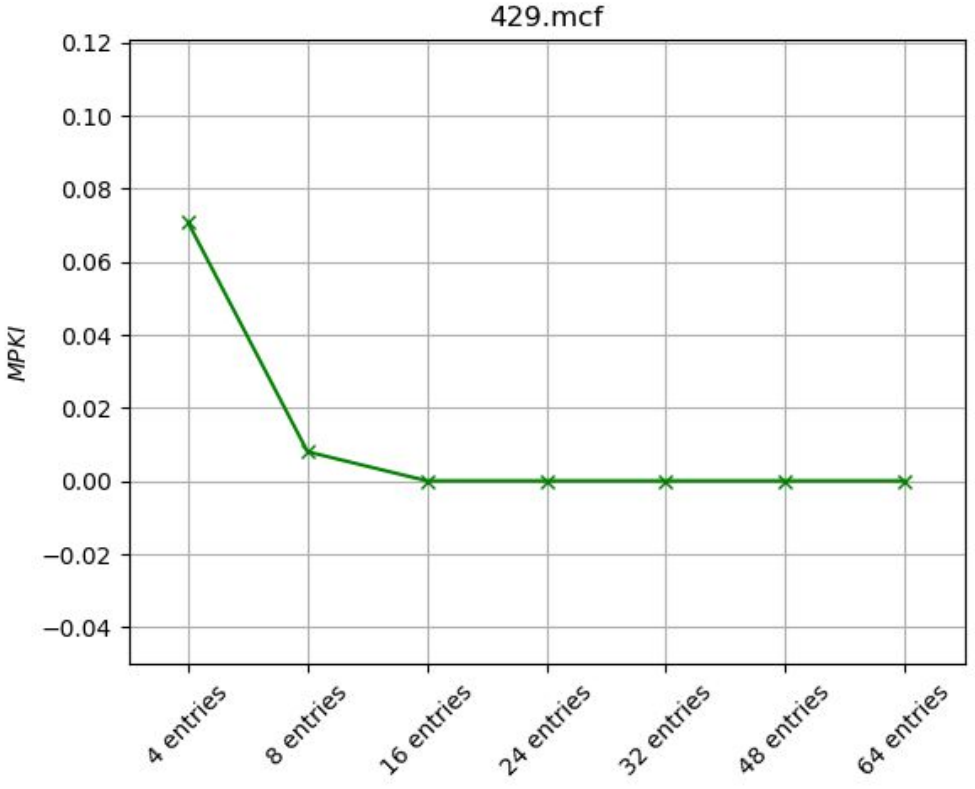
Αριθμός εγγραφών στη RAS
32
48
64

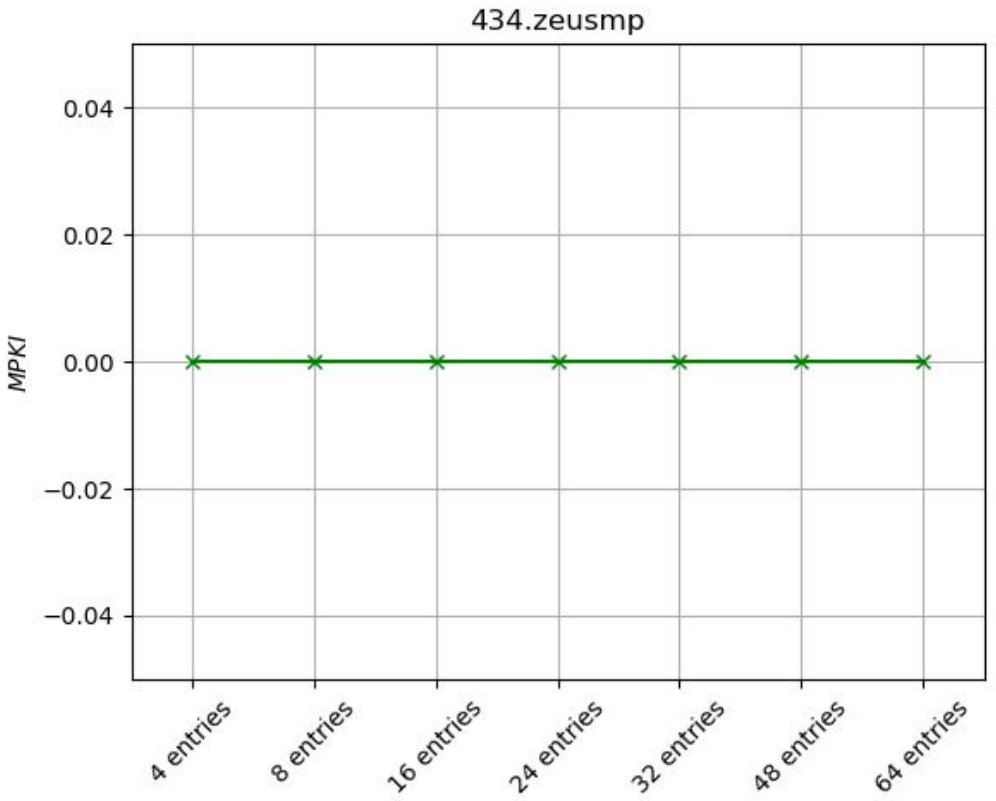
Το συγκεντρωτικό αποτέλεσμα παρουσιάζεται παρακάτω:

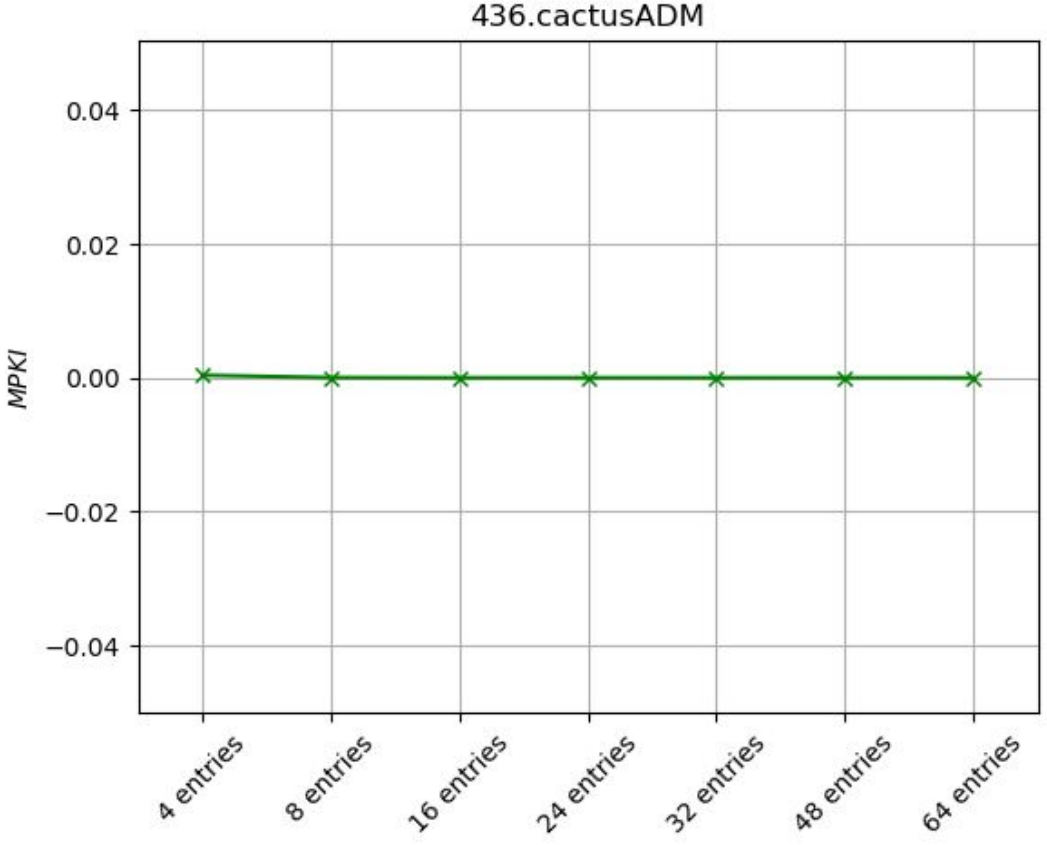


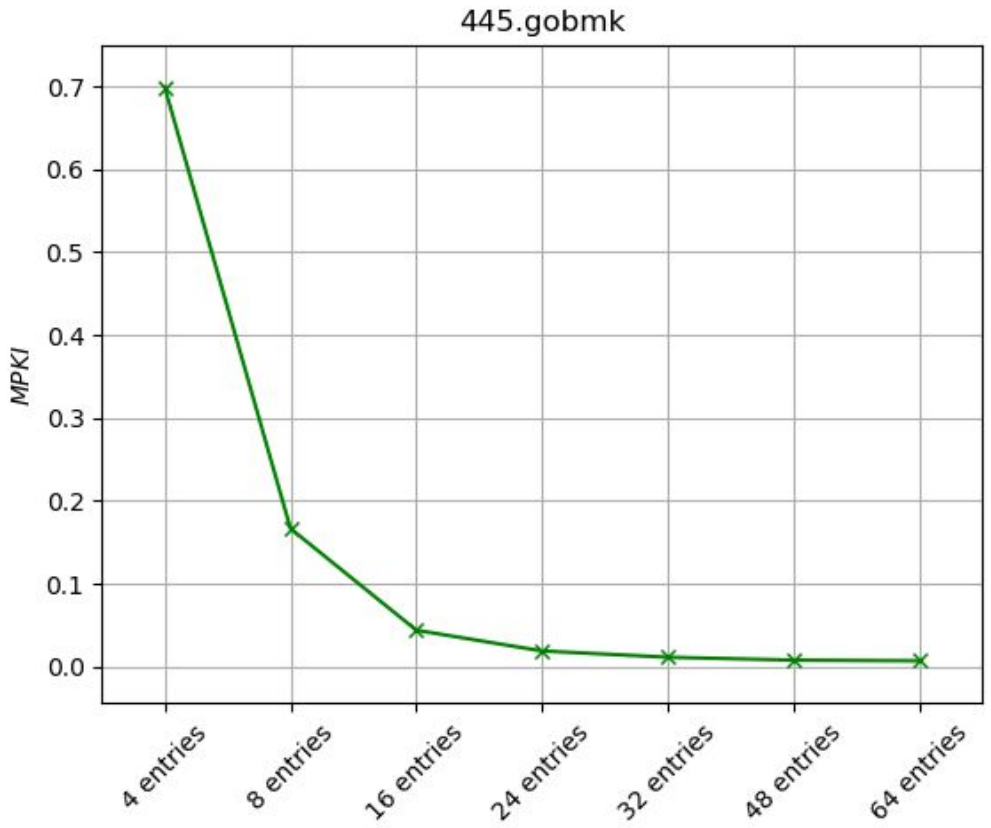
Στην συνέχεια παρουσιάζεται το αποτέλεσμα για καθε ενα απο τα επιμέρους benchmarks έτσι ώστε να έχουμε πιο λεπτομερή μελέτη.

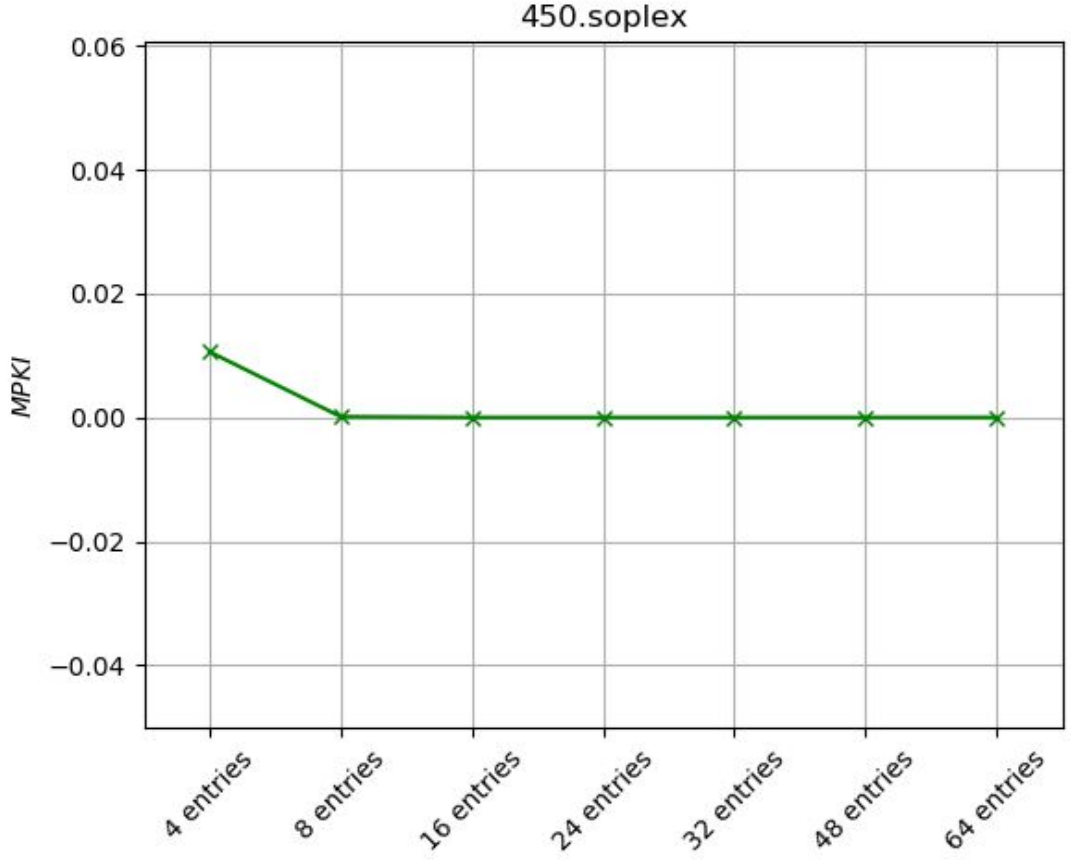


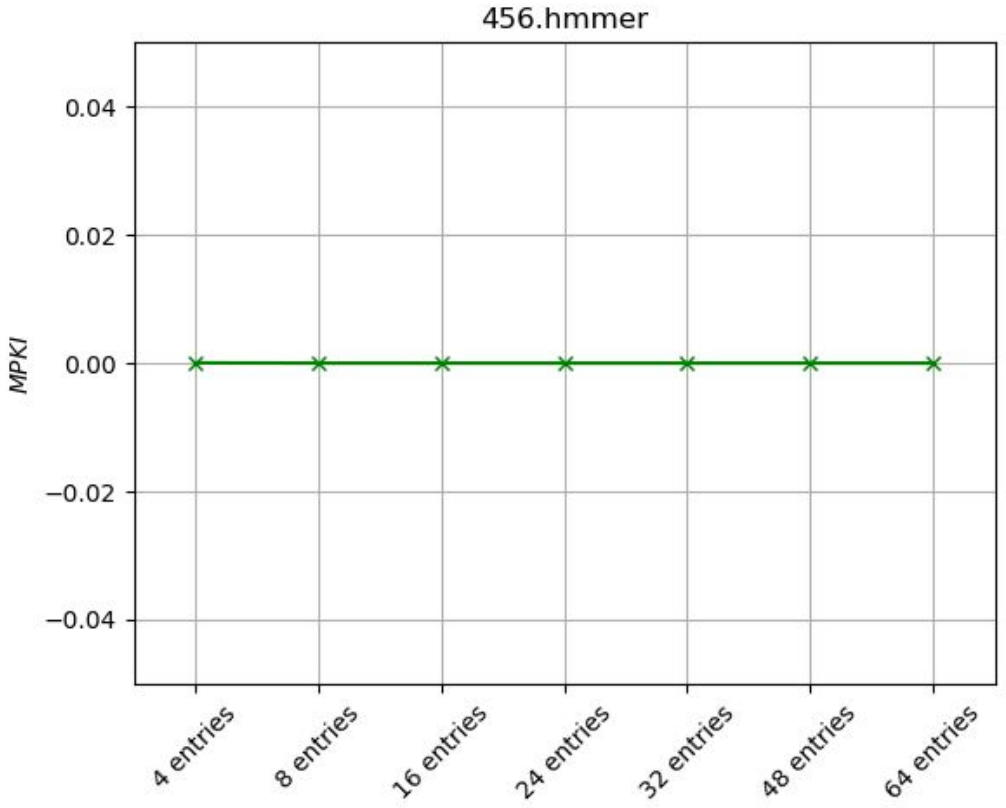


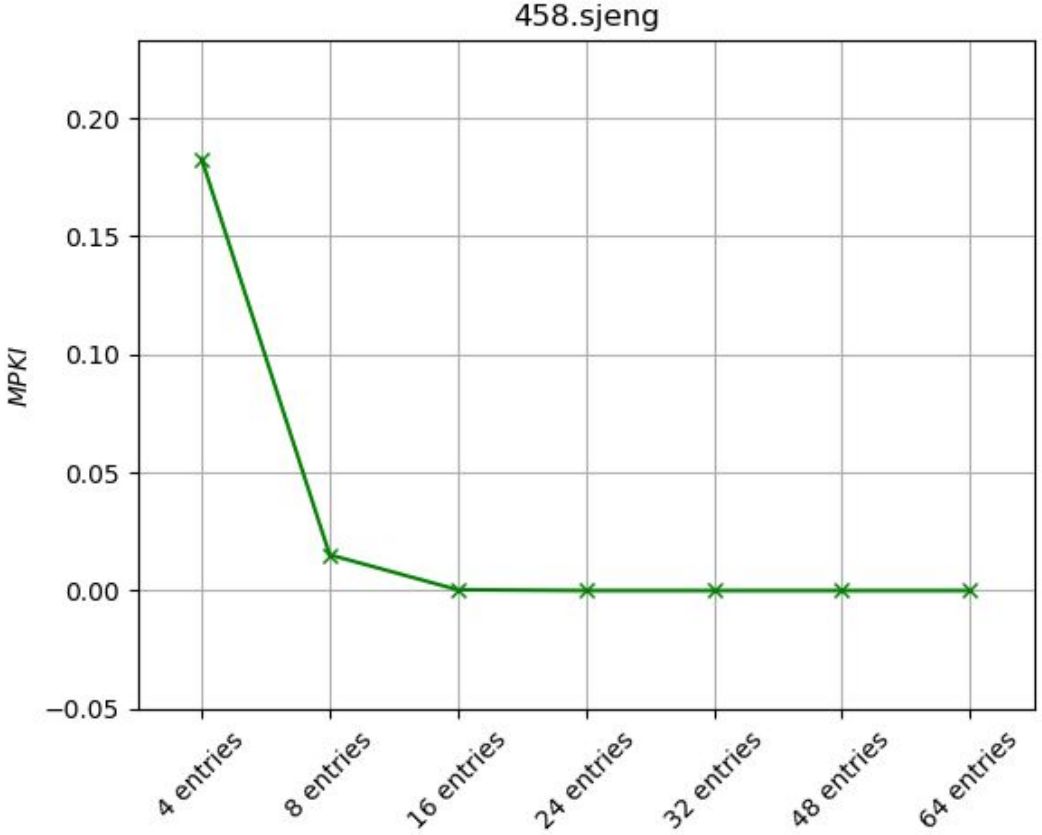

434.zeusmp
MPKI
0.04
0.02
0.00
−0.02
−0.04
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries

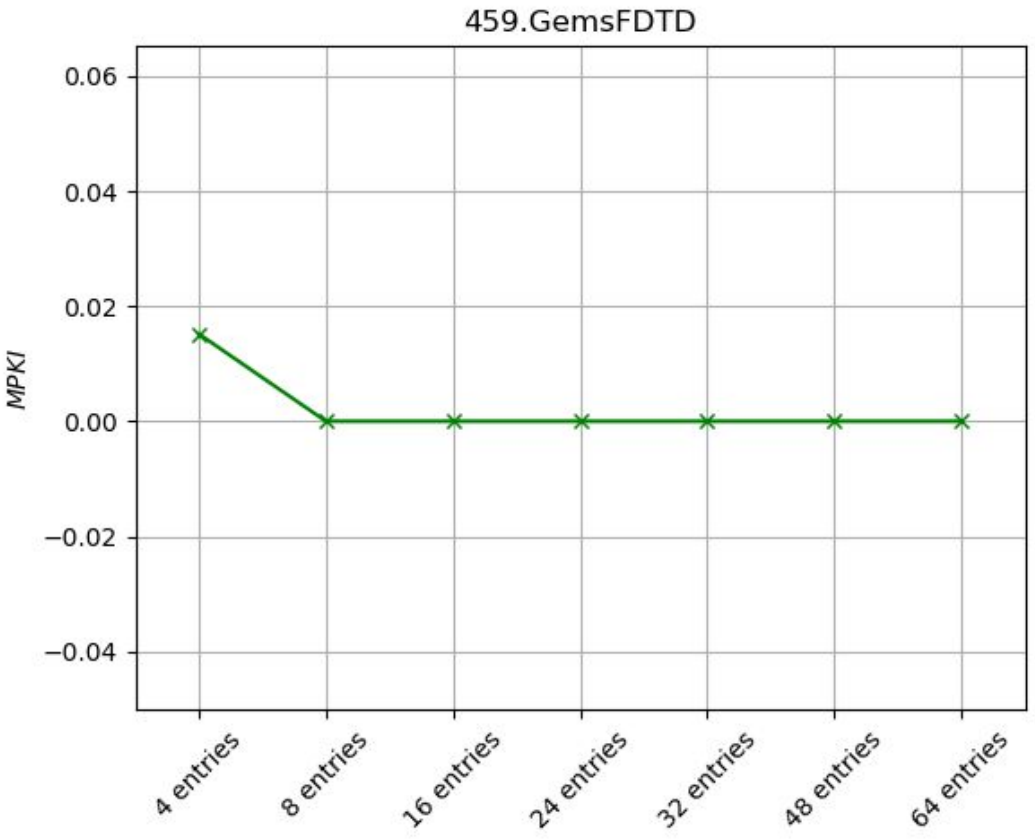

436.cactusADM
MPKI
0.04
0.02
0.00
−0.02
−0.04
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries

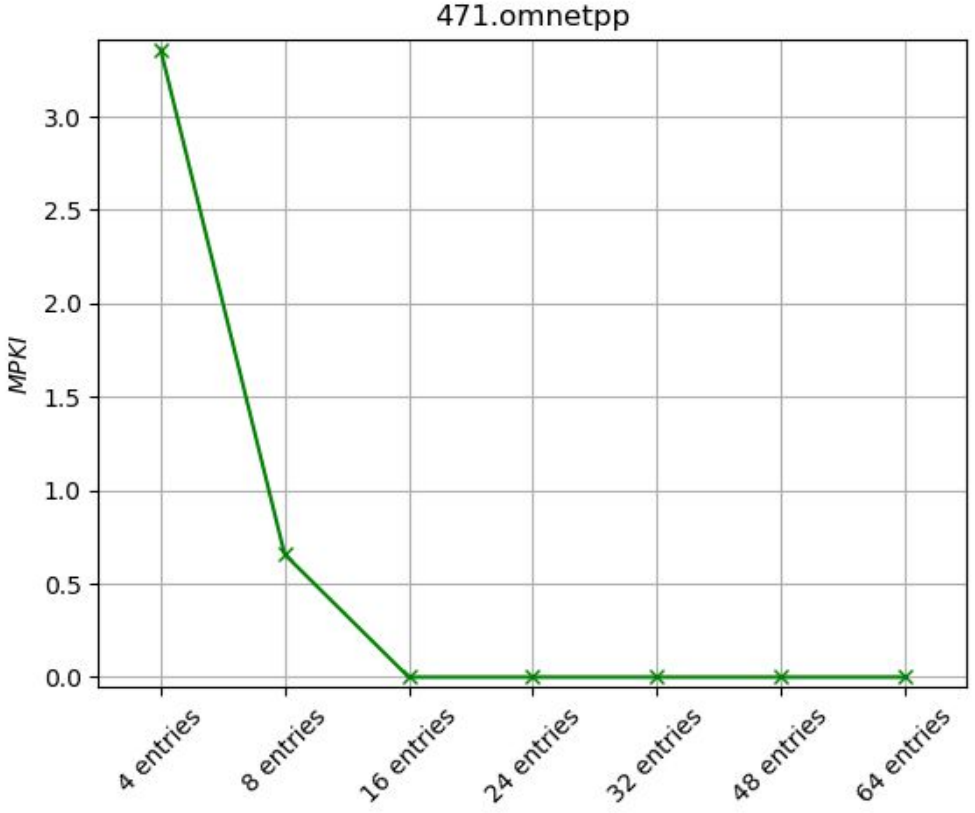

445.gobmk
MPKI
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries

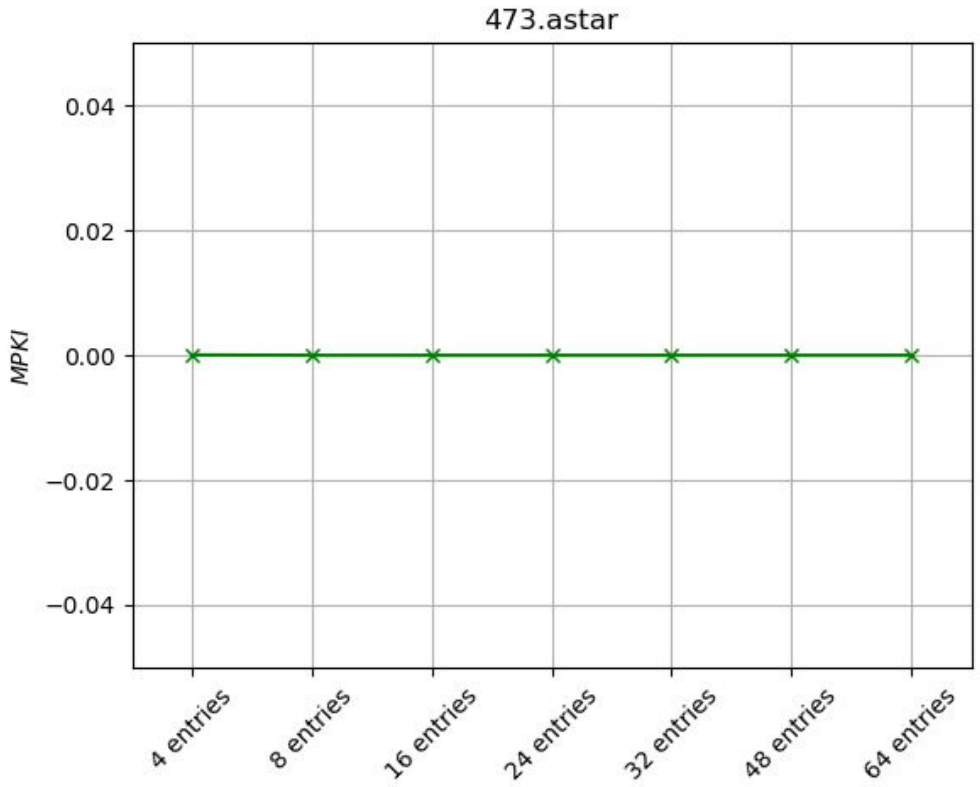

450.soplex
MPKI
0.06
0.04
0.02
0.00
−0.02
−0.04
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries

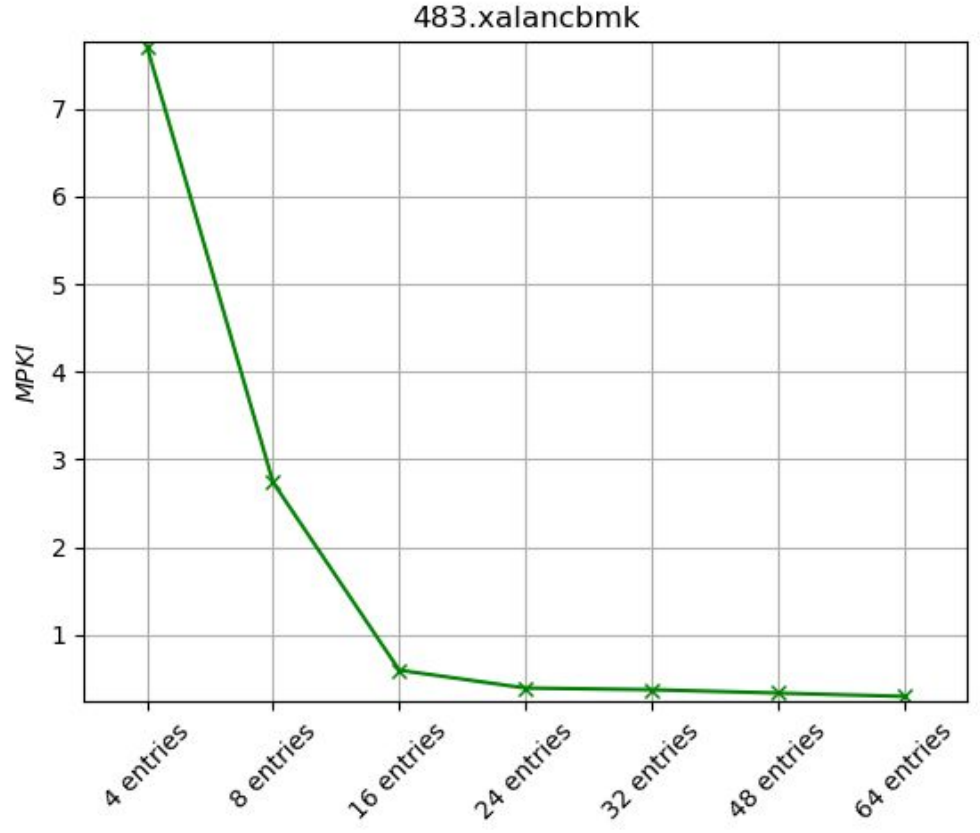

456.hmmer
MPKI
0.04
0.02
0.00
−0.02
−0.04
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries


458.sjeng
MPKI
0.20
0.15
0.10
0.05
0.00
−0.05
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries


459.GemsFDTD
MPKI
0.06
0.04
0.02
0.00
−0.02
−0.04
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries


471.omnetpp
MPKI
3.0
2.5
2.0
1.5
1.0
0.5
0.0
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries


473.astar
MPKI
0.04
0.02
0.00
−0.02
−0.04
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries


483.xalancbmk
MPKI
7
6
5
4
3
2
1
4 entries
8 entries
16 entries
24 entries
32 entries
48 entries
64 entries

Συμπεράσματα 4.4:
Κατά μια γενική ομολογία και κάνοντας μια πολύ απλή σύγκριση με τις γραφικές MPKI που αφορούν την (4.2) Ενότητα - Μελέτη n-bit predictors, παρατηρούμε χαμηλότερη επίπεδα κατά μέσο όρο, με τα χειρότερα benchmarks να αγγίζουν 18% σε αντίθεση με προηγουμένως που κάποια ξεπερνούσαν και το 35%.
Όταν #RAS = 4 , έχουμε παρ' όλα αυτά πολύ μεγάλο MPKI σχεδόν για όλα τα benchmarks, ενώ οταν αυξήσουμε τον αριθμό αυτόν σε 8 , παρατηρείται μεγάλη μείωση του δείκτη σχεδόν στα μισά βενψημαρκσ. Για τα υπόλοιπα μισα benchmarks επειδη το #ras είναι ήδη αρκετά μεγάλο παρατηρούμε ότι έχουν φτάσει ήδη στην βέλτιστη κατάσταση με το 4 και συνεχίζουν να μένουν εκεί όσο το αυξάνουμε, για ορισμένα άλλα είδαμε ότι το αμέσως επόμενο του 4, δηλαδή το 8 οδηγεί το σύστημα στην ιδανική κατάσταση ενώ για λιγότερα το 16. Από εκεί και πέρα οποιαδήποτε αύξηση ειναι αχρειαστη και σπατάλη πόρων για τα δεδομένα προγράμματα με τα δεδομένα input στον δεδομένο υπολογιστική.
Τονίζουμε ότι οι παραπάνω προσομοιώσεις ήταν καταλυτικές για να κατανοήσουμε ότι η αύξηση του # RAS δεν μπορεί να οδηγήσει σε μείωση της απόδοσης σε καμία των περιπτώσεων, άρα παρατηρούμε μόνο διατήρηση ή και βελτίωση της απόδοσης
Με αντικειμενικό κριτήριο το **κόστος του υλικού**, επιλέγουμε υλοποίηση που κάνει χρήση **#RAS = 8 ή 16.** Πιο αναλυτικά 16 για να επιτύχουμε την βέλτιστη απόδοση σε όλα, ενώ 8 για να πετυχουμε την βέλτιστη απόδοση στα περισσότερα(tradeoff).

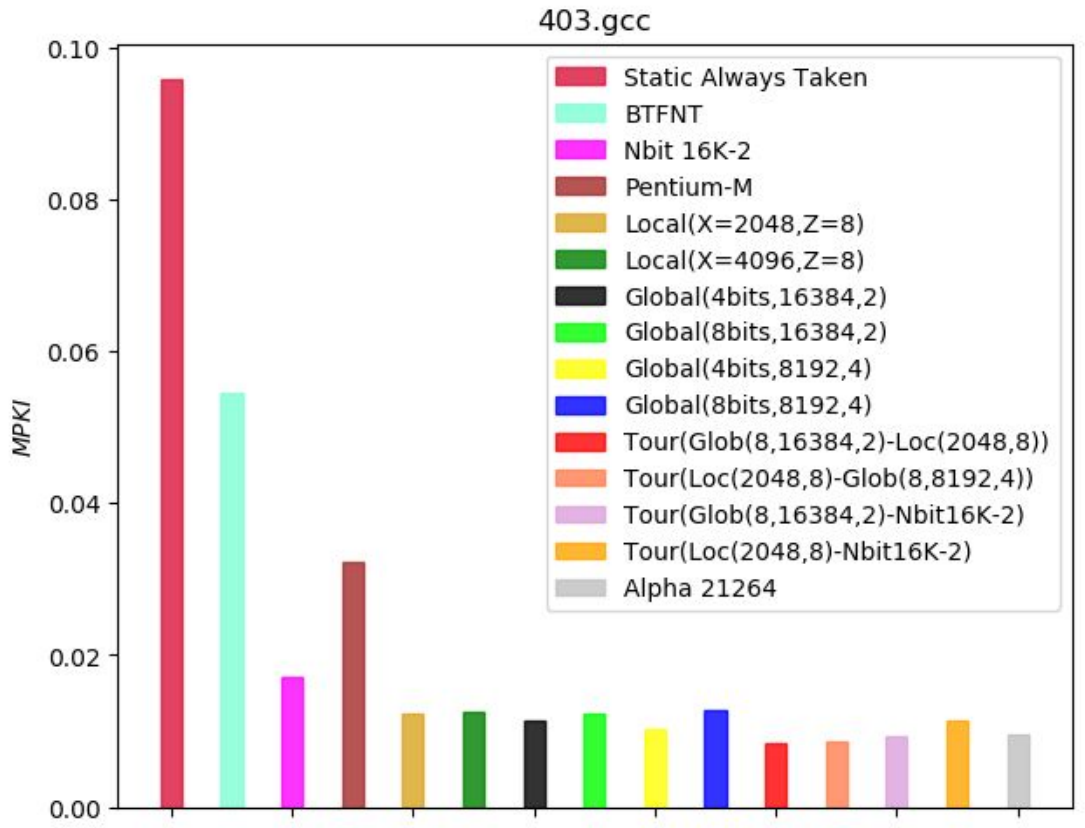
4.5 Σύγκριση διαφορετικών predictors

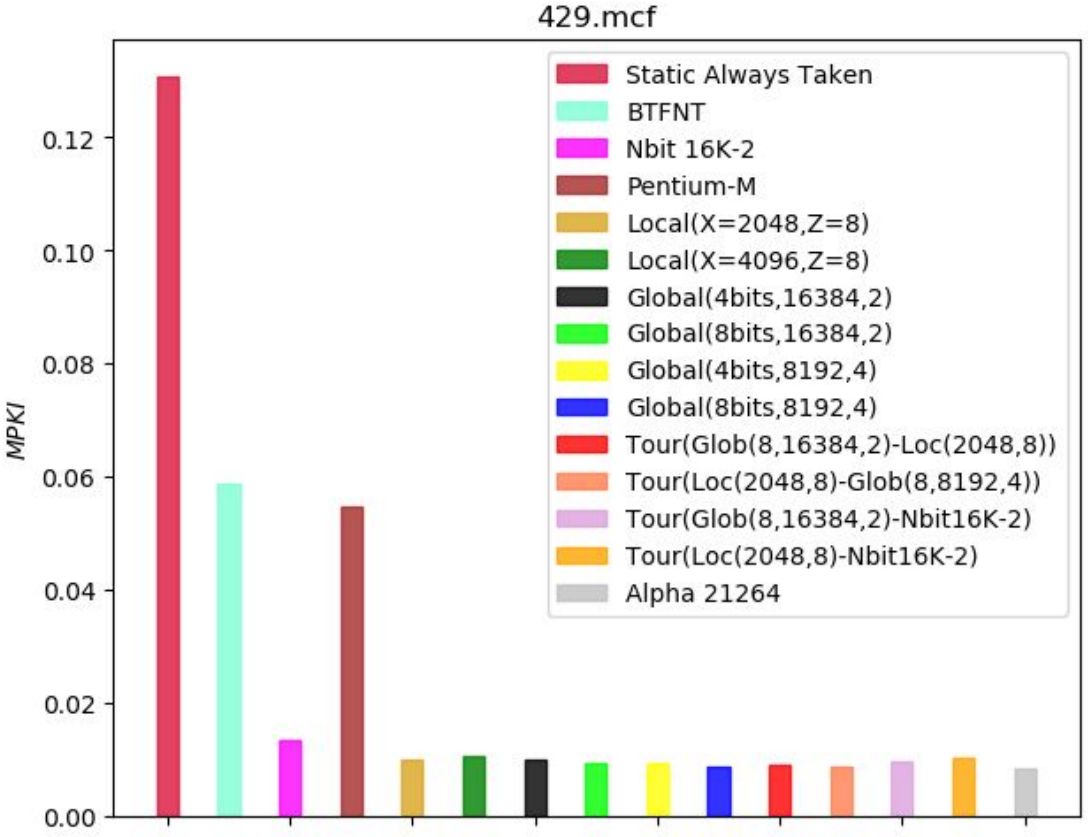
Στο Πέμπτο Μέρος και τελευταίο της πειραματικής αξιολόγησης για την παρούσα άσκηση, εξετάζουμε την απόδοση διαφορετικών predictors.

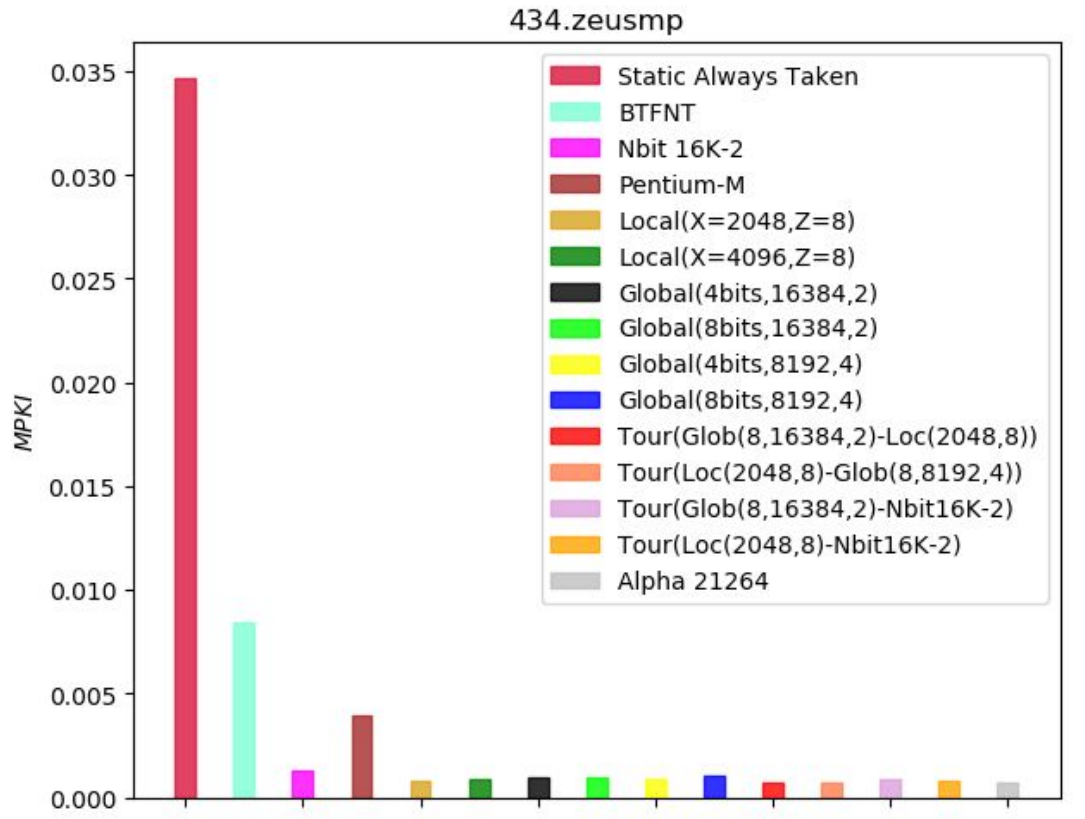
Συγκεκριμένα 15 predictors είναι οι εξείς:

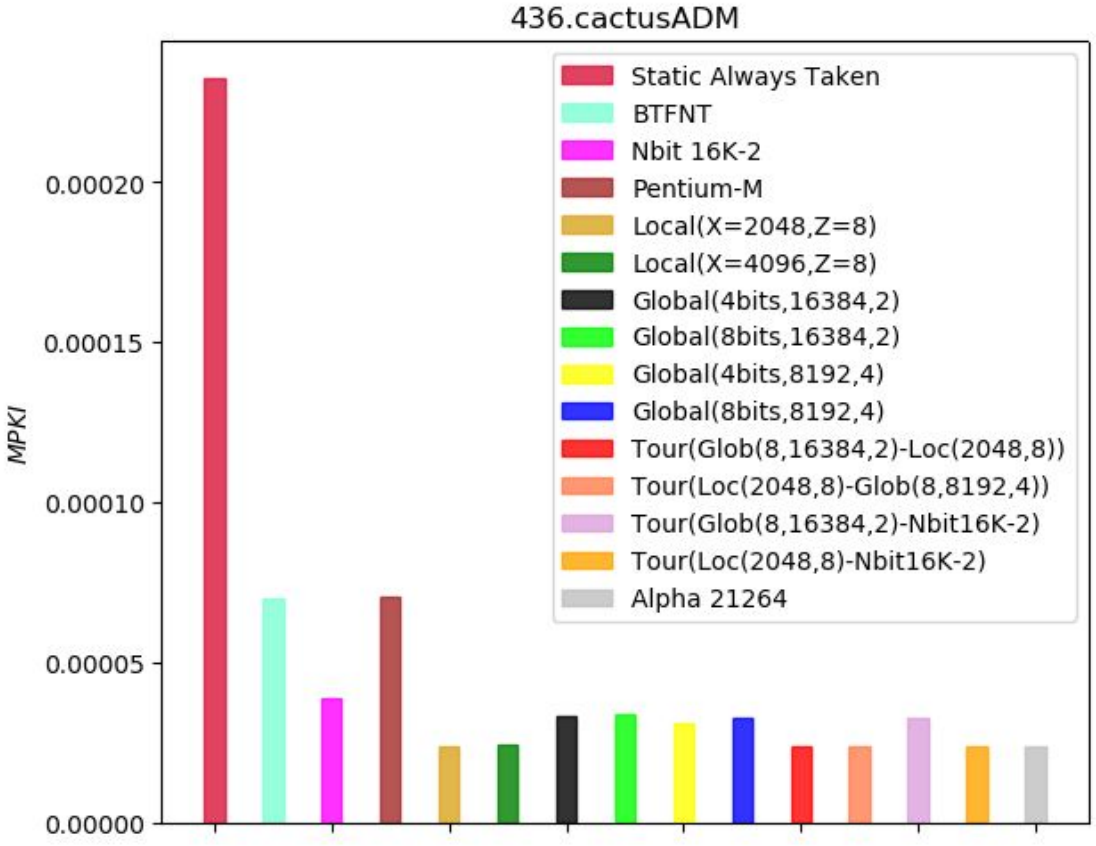
- Static AlwaysTaken
- Static BTFNT (BackwardTaken-ForwardNotTaken)
- Nbit-16K-2
- Pentium-M predictor
- Local-History two-level predictors(X=2048,Z=8)
- Local-History two-level predictors(X=4096,Z=8)
- Global History two-level predictors(4bits,16384,2)
- Global History two-level predictors(8bits,16384,2)
- Global History two-level predictors(4bits,8192,4)
- Global History two-level predictors(8bits,8192,4)
- Tournament Hybrid predictors(Global History two-level predictors(8bits,16384,2) - Local-History two-level predictors(X=2048,Z=8))
- Tournament Hybrid predictors(Local-History two-level predictors(X=2048,Z=8) - Global History two-level predictors(4bits,8192,4))
- Tournament Hybrid predictors(Global History two-level predictors(8bits,16384,2) - Nbit-16K-2)
- Tournament Hybrid predictors(Local-History two-level predictors(X=2048,Z=8) - Nbit-16K-2)
- Alpha 21264 predictor

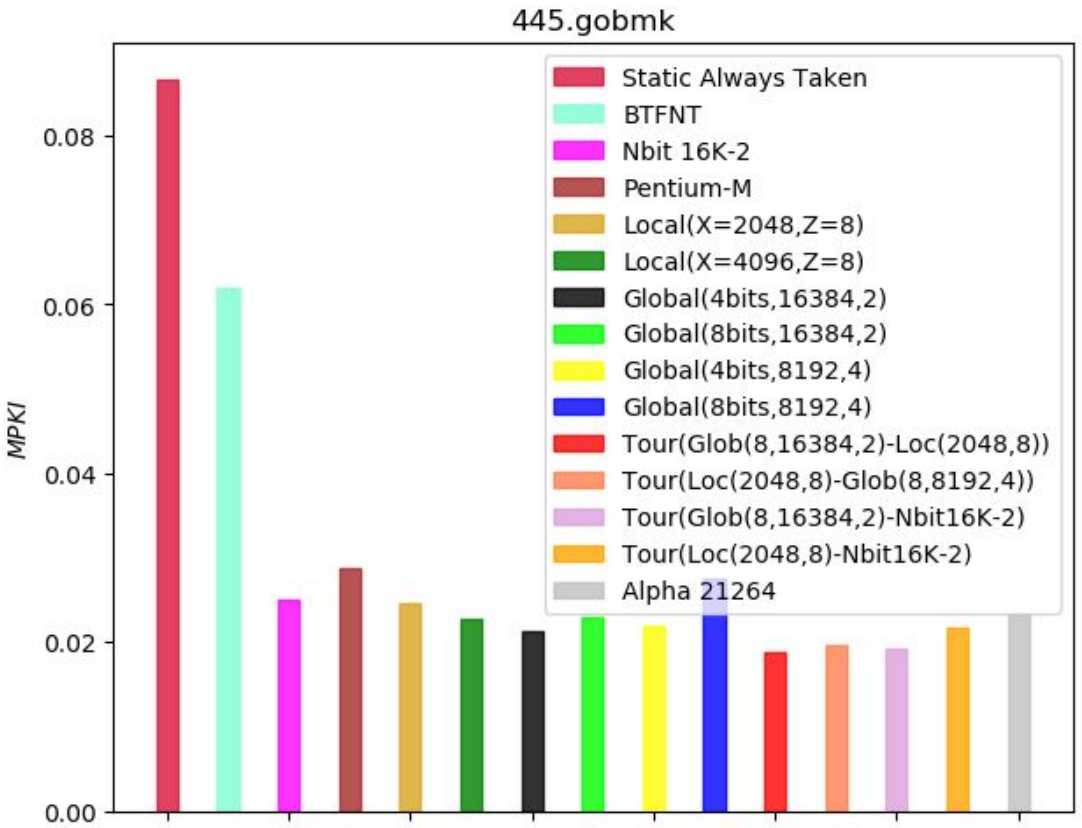
Στην συνέχεια παρουσιάζεται το αποτέλεσμα για καθε ενα απο τα επιμέρους benchmarks έτσι ώστε να έχουμε πιο λεπτομερή μελέτη.

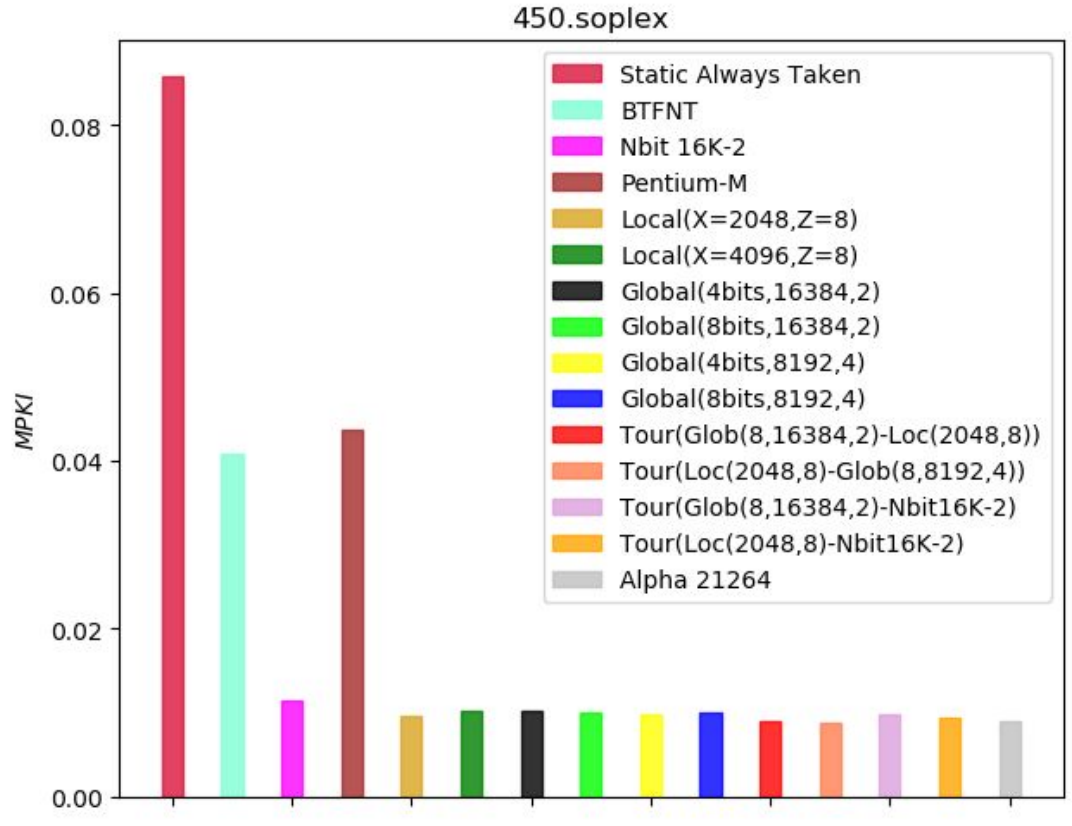

403.gcc
MPKI
0.10
0.08
0.06
0.04
0.02
0.00
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264

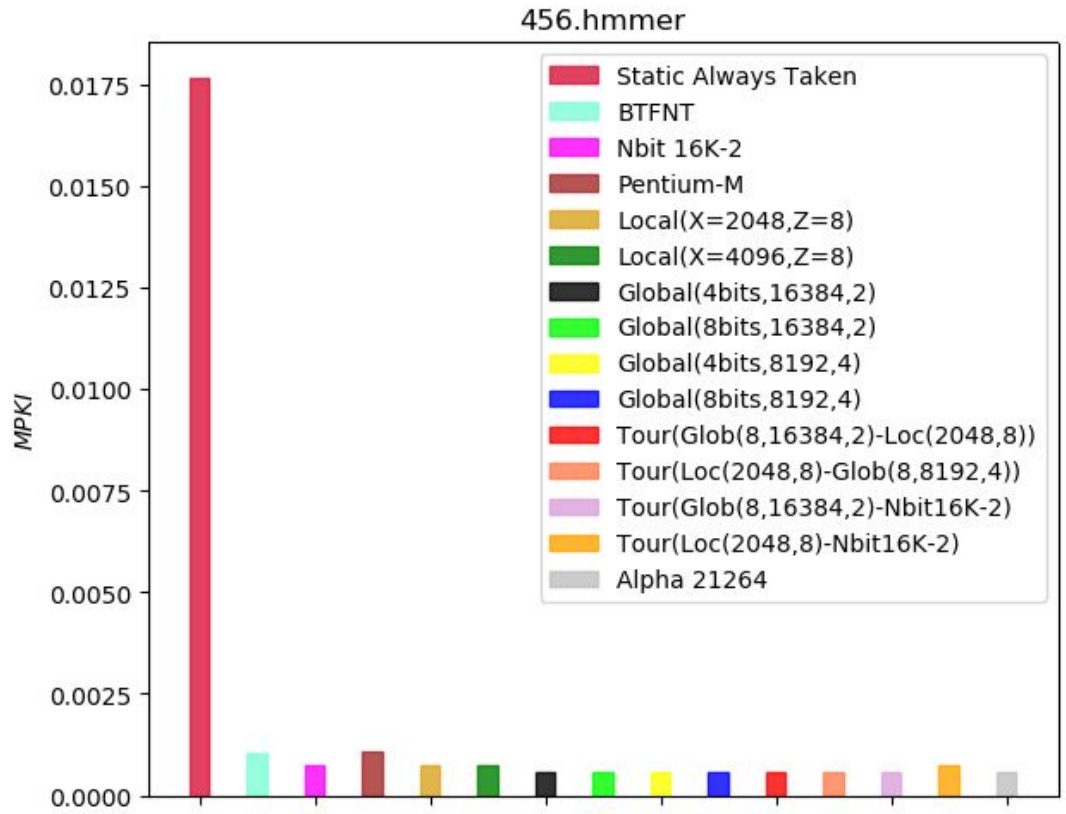

429.mcf
MPKI
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264

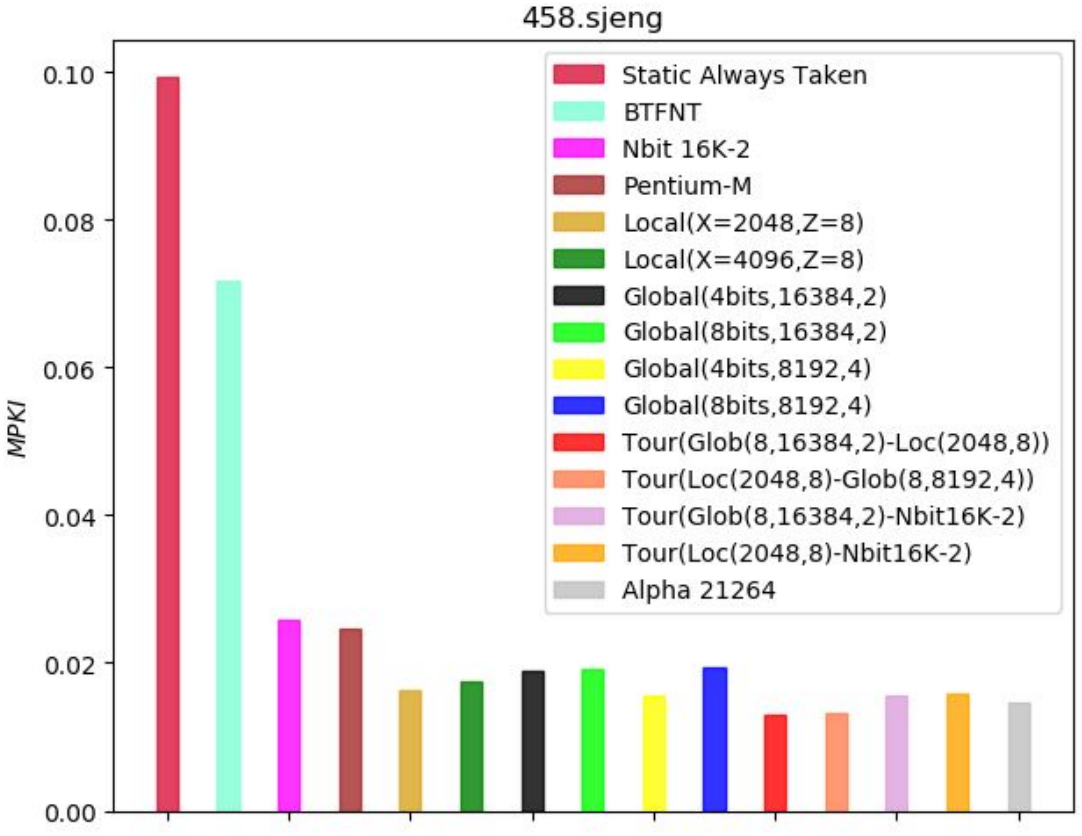

434.zeusmp
MPKI
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264

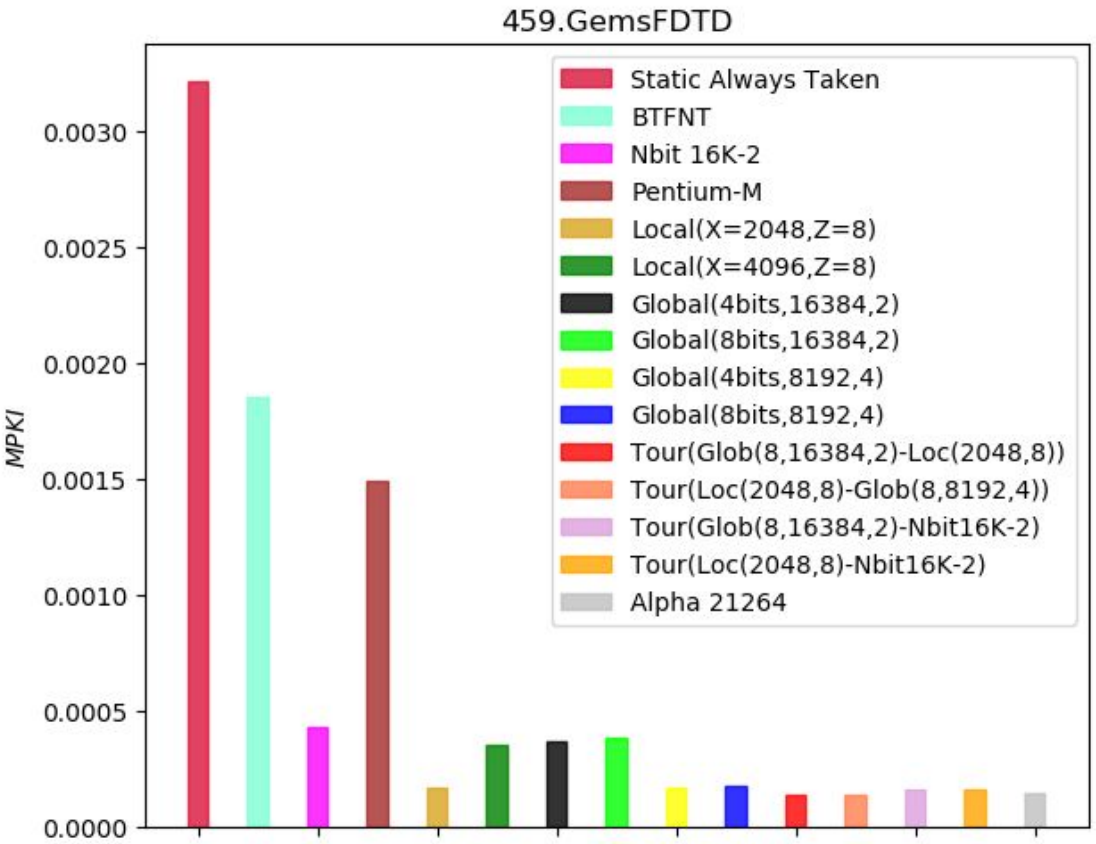

436.cactusADM
MPKI
0.00020
0.00015
0.00010
0.00005
0.00000
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264

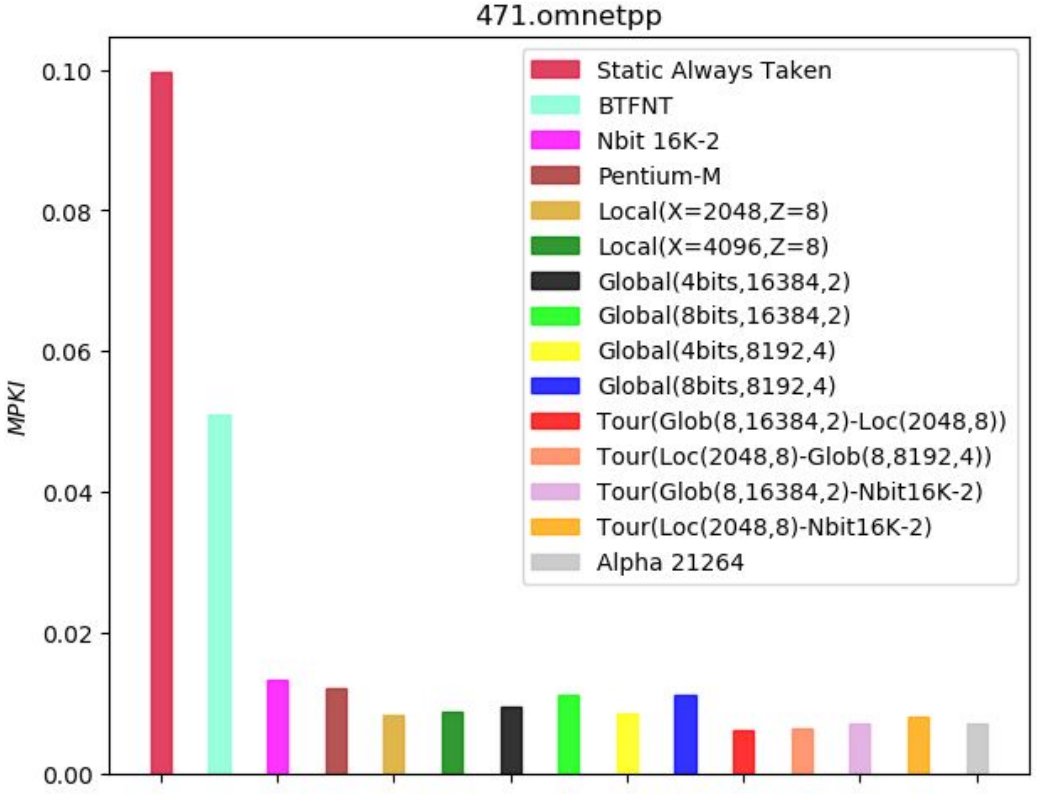

445.gobmk
MPKI
0.00
0.02
0.04
0.06
0.08
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264

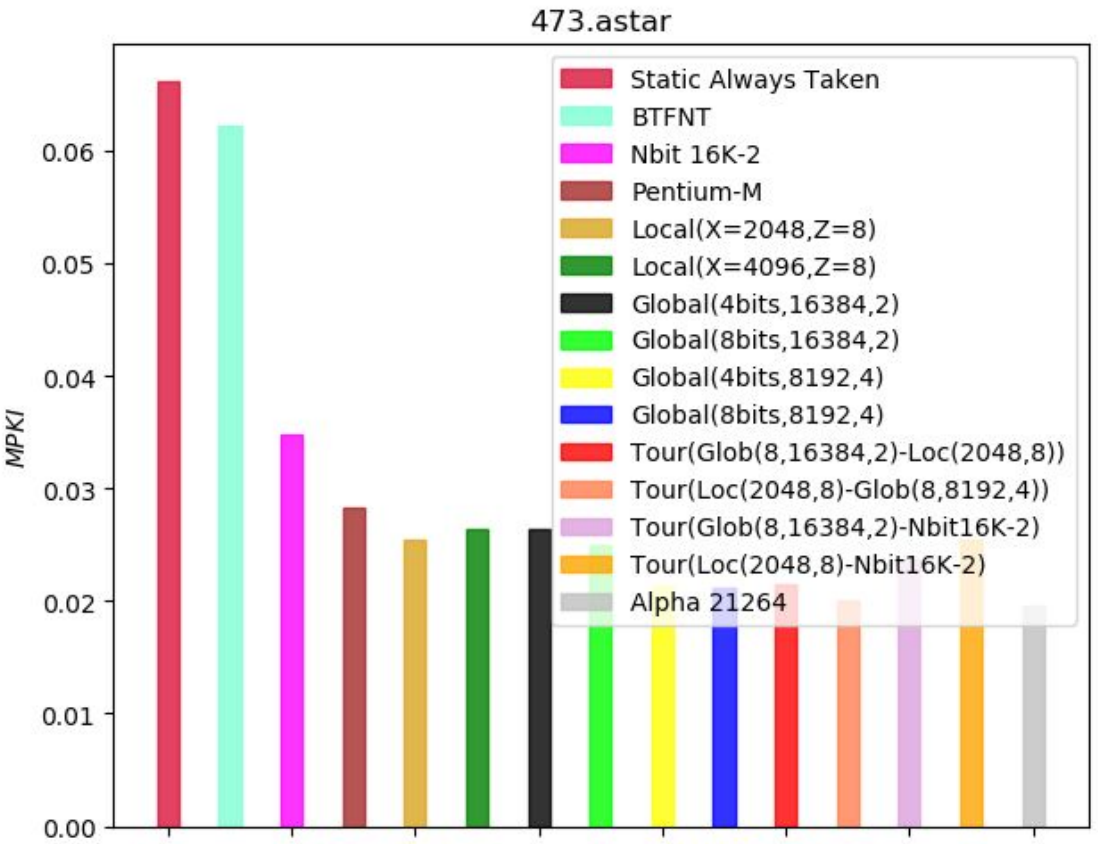

450.soplex
MPKI
0.00
0.02
0.04
0.06
0.08
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264

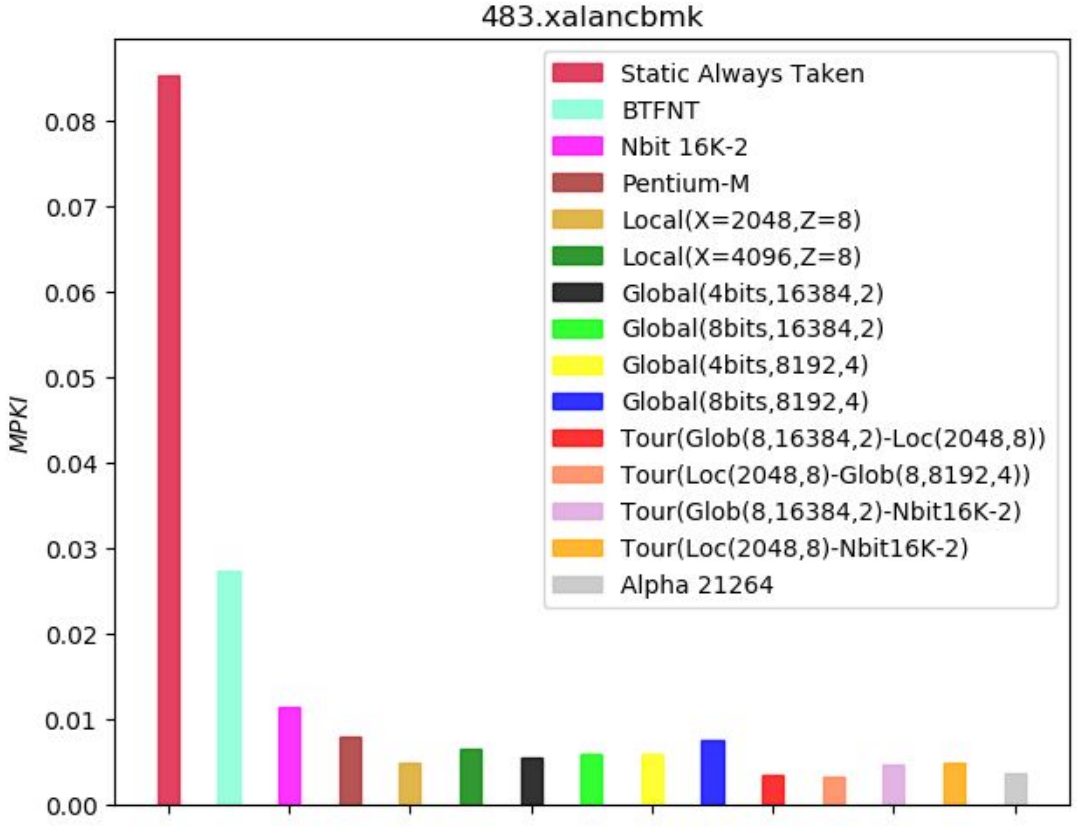

456.hmmer
MPKI
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264


458.sjeng
MPKI
0.10
0.08
0.06
0.04
0.02
0.00
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264


459.GemsFDTD
MPKI
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264


471.omnetpp
MPKI
0.10
0.08
0.06
0.04
0.02
0.00
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264


473.astar
MPKI
0.00
0.01
0.02
0.03
0.04
0.05
0.06
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264


483.xalancbmk
MPKI
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
Static Always Taken
BTFNT
Nbit 16K-2
Pentium-M
Local(X=2048,Z=8)
Local(X=4096,Z=8)
Global(4bits,16384,2)
Global(8bits,16384,2)
Global(4bits,8192,4)
Global(8bits,8192,4)
Tour(Glob(8,16384,2)-Loc(2048,8))
Tour(Loc(2048,8)-Glob(8,8192,4))
Tour(Glob(8,16384,2)-Nbit16K-2)
Tour(Loc(2048,8)-Nbit16K-2)
Alpha 21264

Συμπεράσματα:

- Ο Static Taken predictor έχει την χειρότερη δυνατή απόδοση για όλα τα benchmarks , και αυτό γιατί αλγοριθμικά θεωρεί ότι τα branch δεν θα εκτελεστούν ενώ έχουμε μεγάλου βρόγχους.
- Ο BTFNT predictor παρουσιάζει κατά γενική ομολογία καλύτερη απόδοση για όλα τα benchmarks συγκριτικά με Static Taken predictor, αλλά τα επίπεδα του δείκτη MKPI παραμένουν ακόμα αρκετά υψηλά
- Εν συνεχεία οι N-bit, Pentium-M και όλοι οι Global History και Local History οδηγούν σε πολύ χαμηλότερες τιμές του δείκτη MKPI συγκριτικά με τους 2 προαναφερθέντες, αλλά δεν παρουσιάζεται σαφές pattern για εξαγωγή άμεσου αποτελέσματος για την απόδοση τους, οπότε η ταξινόμηση τους θα γίνεται για την απόδοση του κατά μέσο όρο για όλα τα benchmarks.
- Οι tournament predictors, παρουσιάζουν επίσης καλή απόδοση , είναι υβριδικοί και χρησιμοποιούν μια συνδυασμένη έκδοση Global, Local και N-bit predictors.
- Ο Alpha 21264 predictor φαίνεται ότι τις περισσότερες φορές παρουσιάζει την καλύτερη απόδοση

Άρα, η συνολική στατιστική κατάταξη των predictors θα μπορούσε να είναι αυτη(παρατηρούνται επιμέρους μικροδιαφορές σε καθε benchmark):
Alpha 21264 predictor
Tournament (Local - Global) predictor
Tournament (Global - Local) predictor
Local History (8 bits) predictor
Local History (4 bits) predictor
Tournament (Global - Nbit) predictor
Global History (8 bits) predictor
Tournament (Local - Nbit) predictor
Global History (4 bits) predictor
N-bit 16K-2 predictor
Pentium-M predictor
Static BTFNT predictor
Static Always Taken predictor

Άρα βλέπουμε πόσο σημαντικός είναι ο αλγόριθμος και το hardware του predictor.

5. Παράρτημα:

Η κλάση fsm στο branch_predictors.h

```
class FSM : public BranchPredictor
{
public:
    FSM(unsigned index_bits_, unsigned cntr_bits_)
        : BranchPredictor(), index_bits(index_bits_), cntr_bits(cntr_bits_) {
        table_entries = 1 << index_bits;
        TABLE = new unsigned long long[table_entries];
        memset(TABLE, 0, table_entries * sizeof(*TABLE));

        COUNTER_MAX = (1 << cntr_bits) - 1;
    };
    ~FSM() { 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];
        unsigned long long prediction = ip_table_value >> (cntr_bits - 1);
        return (prediction != 0);
    };

    virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {
        unsigned int ip_table_index = ip % table_entries;
        if (actual) {
            if (TABLE[ip_table_index]==1) TABLE[ip_table_index]=3;
            else if (TABLE[ip_table_index] < COUNTER_MAX && TABLE[ip_table_index]!=1) TABLE[ip_table_index]++;
        }
    else {
            if (TABLE[ip_table_index]==2) TABLE[ip_table_index]=0;
        else if (TABLE[ip_table_index] >0 && TABLE[ip_table_index]!=2) TABLE[ip_table_index]--;
        }

        updateCounters(predicted, actual);
    };

    virtual string getName() {
        std::ostringstream stream;
        stream << "FSM" << pow(2.0,double(index_bits)) / 1024.0 << "K-" << cntr_bits;
        return stream.str();
    }

private:
    unsigned int index_bits, cntr_bits;
    unsigned int COUNTER_MAX;

    unsigned long long *TABLE;
    unsigned int table_entries;
};
```

Η κλάση BTB Entry στο branch_predictors.h

```
class BTBEntry {
   public:
      ADDRINT ip;
      ADDRINT target;
      UINT64 LRUCounter;

      BTBEntry()
         : ip(0), target(0), LRUCounter(0) {}

      BTBEntry(ADDRINT i, ADDRINT t, UINT64 cnt)
         : ip(i), target(t), LRUCounter(cnt) {}
};

class BTBPredictor : public BranchPredictor
{
public:
            BTBPredictor(int btb_lines, int btb_assoc)
                : table_lines(btb_lines), table_assoc(btb_assoc),
correct_target_predictions(0), incorrect_target_predictions(0), timestamp(0)
            {   TABLE = new BTBEntry[table_lines*table_assoc];

            }

            ~BTBPredictor() {
            delete TABLE;

            }

   virtual bool predict(ADDRINT ip, ADDRINT target) {
                        BTBEntry* entry = find(ip);
               if (entry) {
                        entry->LRUCounter = timestamp++;

                        return true;
            }
                        return false;
            }



            virtual void update(bool predicted, bool actual, ADDRINT ip, ADDRINT target) {

                        if (actual && predicted) {

                                    BTBEntry* entry = find(ip);
                                    if (entry) {
                                                if (entry->target == target)
                                                            correct_target_predictions++;
                                                else {
                                                            incorrect_target_predictions++;
                                                            entry->target = target;
                                                }
                                    }
                                    else {
                                    perror("BTB: Entry is not present although the branch is predicted taken");
                                    }
```

```
                }
                else if (actual && (!predicted)) {

                        BTBEntry* entry = replace(ip);
                        entry->ip = ip;
                        entry->target = target;
                        entry->LRUCounter = timestamp++;
                }
                else if ((!actual) && predicted) {

                        BTBEntry* entry = find(ip);
                        entry->ip = 0;
                        entry->target = 0;
                        entry->LRUCounter = 0;
                }

                updateCounters(predicted, actual);
        }

  virtual string getName() {
    std::ostringstream stream;
                stream << "BTB-" << table_lines << "-" << table_assoc;
                return stream.str();
        }

  UINT64 getNumCorrectTargetPredictions() {
        return correct_target_predictions;
        }

private:
        unsigned int table_lines, table_assoc;
        unsigned int correct_target_predictions, incorrect_target_predictions;

        UINT64 timestamp;

        BTBEntry* TABLE;

        BTBEntry* find(ADDRINT ip) {
                unsigned int ip_table_index = ip % table_lines;

                for (unsigned int i = 0; i<table_assoc; i++) {
                        BTBEntry* entry = &TABLE[ip_table_index*table_assoc+i];
                        if (entry->ip == ip) {
                                return entry;
                        }
                }
                return NULL;
        };

        BTBEntry* replace(ADDRINT ip) {
                unsigned int ip_table_index = ip % table_lines;

                BTBEntry* entry;
                BTBEntry* LRUEntry = &TABLE[ip_table_index*table_assoc];

                for (unsigned int i = 0; i < table_assoc; i++) {
                        entry = &TABLE[ip_table_index*table_assoc+i];
```

```
                        if (LRUEntry->LRUCounter > entry->LRUCounter) {
                                LRUEntry = entry;
                        }
                }
                return LRUEntry;
        };
};
```

Οι κλάσεις για το 4.5:

```
class Static Taken Predictor : public Branch Predictor {

public:
        Static Taken Predictor():Branch Predictor() {};

        ~Static Taken Predictor() {};

        virtual bool predicate(ADDRESS ip, ADDRESS target) {
                return true;
        };

        virtual void update(bool predicate, bool actual, ADDRESS ip, ADDRESS target) {
                updateCounters(predicted, actual);
        };

        virtual string getName() {
                std::ostringstream stream;
                stream << "Static Always Taken Predictor";
                return stream.str();
        };

};

class BTF Predictor : public Branch Predictor {
public:
 BTF Predictor ():Branch Predictor() {};
 ~BTFNTPredictor() {};

        virtual bool predicate(ADDRESS ip, ADDRESS target) {
                return target < ip;
        };

        virtual void update(bool predicate, bool actual, ADDRESS ip, ADDRESS target) {
                updateCounters(predicted, actual);
        };

        virtual string getName() {
                std::ostringstream stream;
                stream << "BTF Predictor";
                return stream.str();
        };

};

class Tournament Predictor : public Branch Predictor {
public:
```

```
  Tournament Predictor(iit _entries, Branch Predictor* A, Branch Predictor* B)
  : Branch Predictor(), entries(_entries) {
      PREDICTOR[0] = A;
      PREDICTOR[1] = B;

      // whatever, initial values are never used anyway
      prediction[0] = true;
      prediction[1] = true;

      counter = new int[entries];
      memset(counter, 0, entries * sizeof(*counter));
  }

  virtual bool predicate(ADDRESS ip, ADDRESS target) {
      int cnt = counter[ip % entries];

      prediction[0] = PREDICTOR[0]->predict(ip, target);
      prediction[1] = PREDICTOR[1]->predict(ip, target);

      // 0,1 -> prediction[0] --- 2,3 -> prediction[1]
      if (cnt < 2)
          return prediction[0];
      else if (cnt < 4)
          return prediction[1];
      else {
          cerr << "ERROR, SOMETHING WENT WRONG, TRUST NOTHING BEYOND THIS LINE" << endl;
          return false;
      }
  }

  virtual void update(bool predicate, bool actual, ADDRESS ip, ADDRESS target) {
      updateCounters(predicted, actual);
      int index = ip % entries;

      PREDICTOR[0]->update(prediction[0], actual, ip, target);
      PREDICTOR[1]->update(prediction[1], actual, ip, target);

      if (prediction[0] == prediction[1])
          return;

      if ((prediction[0] == actual) && (counter[index] > 0))
          counter[index] --; // favour p0 for this entry
      else if (counter[ip % entries] < 3)
          counter[index] ++; // favour p1 for this entry
  }

  virtual string getName() {
      std::ostringstream stream;
      stream << "Tournament(" << PREDICTOR[0]->getName() << "," << PREDICTOR[1]->getName() << ")";
      return stream.str();
  }

  ~Tournament Predictor() {
      delete counter;
  }

private:
```

```
    Branch Predictor *PREDICTOR[2];
    bool prediction[2];

    int entries;
    int *counter;
};


class Local History Predictor : public Branch Predictor {
public:
    Local History Predictor(int bht_entries_, int bht_bits_, int index_bits_, int bits)
    : Branch Predictor(), pht_entries(1 << (index_bits_+bht_bits_)), pht_bits(nbits), bht_entries(bht_entries_),
bht_bits(bht_bits_) {
        bhrmax = 1 << bht_bits;

        BHT = new int[bht entries];
        for (int i = 0; i < bhrmax; i++) {
            PHT.push_back(new NbitPredictor(index_bits_, pht_bits));
        }
    }

    ~Local History Predictor() {
        PHT.clear();
        delete BHT;
    }

    virtual bool predicate(ADDRESS ip, ADDRESS target) {
        int bha = BHT[ip % bht entries];
        return PHT[bhr]->predict(ip, target);
    }

    virtual void update(bool predicate, bool actual, ADDRESS ip, ADDRESS target) {
        updateCounters(predicted, actual);

        int & bha = BHT[ip % bht entries];
        PHT[bhr]->update(predicted, actual, ip, target);

        bhr = (bhr << 1) % bhrmax;      // shift left and drop MSB
        if (actual & bht bits) bar;  // set LSB to 1 if branch was taken
    }

    virtual string getName() {
        ostringstream st;
        st << "Local History-PHT(" << pht entries << "," << pht_bits << ")-BHT(" << bht entries << "," << bht bits << ")";
        return st.str();
    }

protected:
    int pht_entries, pht_bits;
    int bht_entries, bht_bits;

private:
    int *BHT;                   // use BHT[ip % bht entries]
    vector<Nbtc Predictor*> PHP;  // use PHP[BHT[ip % bht entries]]

    int bhrmax;                 // max entry for BHT
};
```

```
class Global History Predictor : public Local History Predictor {
public:
   GlobalHistoryPredictor(int _bhr_bits, int index_bits, int nbits)
   : Local History Predictor(1, _b hr_bits, index bits, n bits)
   {}

   virtual string getName() {
      ostringstream st;
      st << "Global History-PHT(" << pht_entries << "," << pht_bits << ")-BHR(" << bht_bits << ")";
      return st.str();
   }
};

class lh_2lpalpha: public Branch Predictor {
            public:

                        lh_2lpalpha(int entries,int entry length) : Branch Predictor(), bht entries(entries), entry length(entry
length), cnt_max( int(pow(2.0,entry length*1.0)) -1 )  {
                                    pt cl=2;
                                    bht entries = 1024;
                                    bht_table = new int[entries];
                                    pht table = new int[bht entries];
                                    for (int i=0; i<entries; i++)
                                                bht table[i]=0;
                                    for (int i=0; i<pht_entries; i++){
                                                pht table[i]=0;
                                    }
                        }

                        virtual bool predicate(ADDRESS ip, ADDRESS target){
                                    int bit index = ip % bht entries;
                                    int bit value = bht table[bht index];
                                    int php index = by value;
                                    return (php table[php index]>=4);

                        }

                        virtual void update(bool predicate, bool actual, ADDRESS ip, ADDRESS target){
                                    int bit index = ip % bht entries;
                                    int bit value = bht table[bht index];
                                    int php index = by value;
                                    if (actual){
                                                if (php table[php index]<7)
                                                            pht table[php index]++;
                                                bht table[bht index] = ((bht value << 1) & int_max) +1;
                                    }
                                    else{
                                                if (php table[php index]>0)
                                                            pht table[php index]--;
                                                bht table[bht index] = ((by value << 1) & int_max);
                                    }
                                    updateCounters(predicted,actual);
                        }

                        ~lh 2lp alpha(){
                                    delete[] bht table;
```

```
                        delete[] php table;
                }
                virtual string getName() {
                        std::ostringstream stream;
                        stream << "Alpha Local History 2 level predictor-PHT_entries("<< pht_entries << ")-BHT
entries(" << bht_entries << ")-BHT entry length(" << entry_length <<")";
                        return stream.str();
                }

        private:
                int overhead;
                int bht entries;
                int pht_entries ;
                int pht_cl;
                int entry_length; //length = 16*1024/entries
                int *bht table;
                int *pht table;
                int int_max;


};

class alpha: public Branch Predictor{

        public:
                alpha(){
                        p0 =  new lh 2lp alpha(1024,10);
                        p1 =  new gh_2lp(4096,2,12);
                        table = new int[4096];
                }
                ~alpha() {
                        delete[] table;
                }

                virtual bool predicate(ADDRESS ip, ADDRESS target){
                        bool pre d0 = p0->predict(ip,target);
                        bool pred1 = p1->predict(ip,target);
                        int meta_index = static_cast<gh_2lp*>(p1)->get_pht_addr(ip);
                        int dec = table[meta_index % 4096];
                        if (dec<2)
                                return pr0;
                        else
                                return pred1;
                }

                virtual void update(bool predicate, bool actual, ADDRESS ip, ADDRESS target){
                        bool pre d0 = p0->predict(ip,target);
                        bool pred1 = p1->predict(ip,target);

                        int meta_index = static_cast<gh_2lp*>(p1)->get_pht_addr(ip);
                        int dec = table[meta_index % 4096];
                        if (pred0==actual && pred1!=actual)
                                dec--;
                        else if (pred0!=actual && pred1==actual)
                                dev c++;
                        if (dec<0)
                                dec = 0;
```

```
                        if (dec>3)
                                dec = 3;
                        table[meta index % 4096] = dec;
                        updateCounters(predicted, actual);
                        p0->update(predicted, actual, ip, target);
                        p1->update(predicted, actual, ip, target);
                }

                virtual string getName(){
                        std::ostringstream stream;
                        stream << "Alpha 21264 predictor";
                        return stream.str();
                }

        private:
                Branch Predictor *p0;
                Branch Predictor *p1;
                int *table;



};
```