



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΜ&ΜΥ

Προηγμένα Θέματα
Αρχιτεκτονικής Υπολογιστών

2^η Άσκηση
Ακ. έτος 2011-2012

Γρηγόρης Λύρας Α.Μ.: 03109687

27 Μαΐου 2012

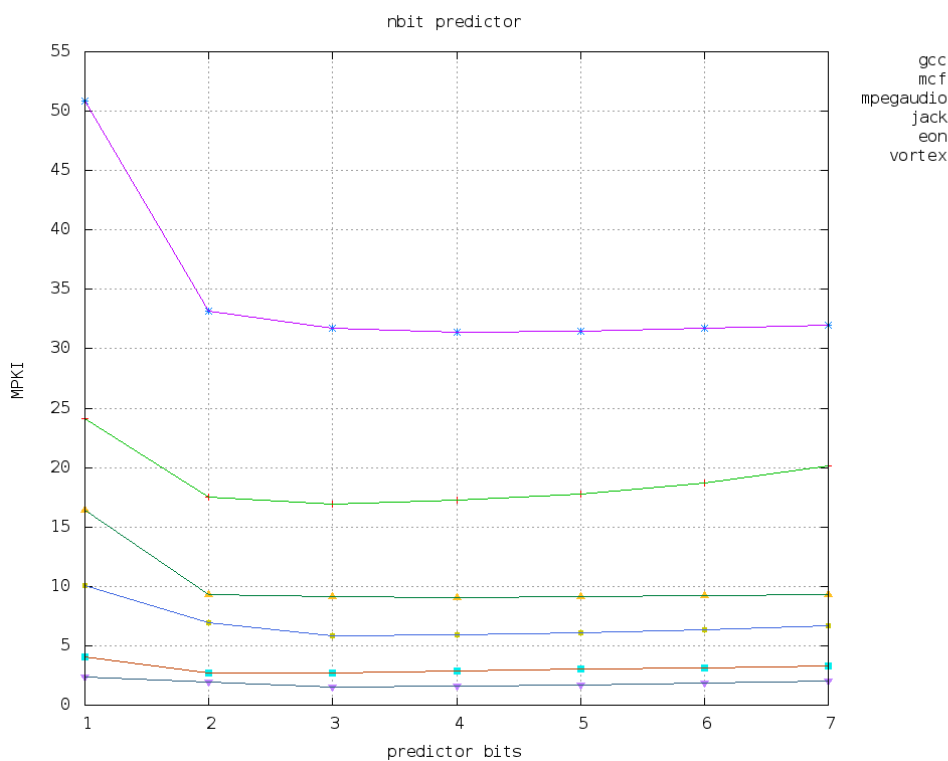
Εισαγωγή

Στην άσκηση αυτή χρησιμοποιήσαμε ένα Framework για την προσομοίωση αλμάτων όπως έχουν καταγραφεί από την εκτέλεση benchmarks από τη σουίτα SPEC2000. Τα traces περιέχουν μόνο εντολές άλματος όπως αυτές πραγματοποιήθηκαν κατά την εκτέλεση 100M εντολών.

Μελέτη των n-bit predictors

A.1

Σε αυτό το τμήμα μελετήσαμε την απόδοση των $\{1..7\}$ -bit predictors χρησιμοποιώντας ως μετρική τα MPKI (Mispredictions Per Thousand Instructions) με 16K BHT entries.



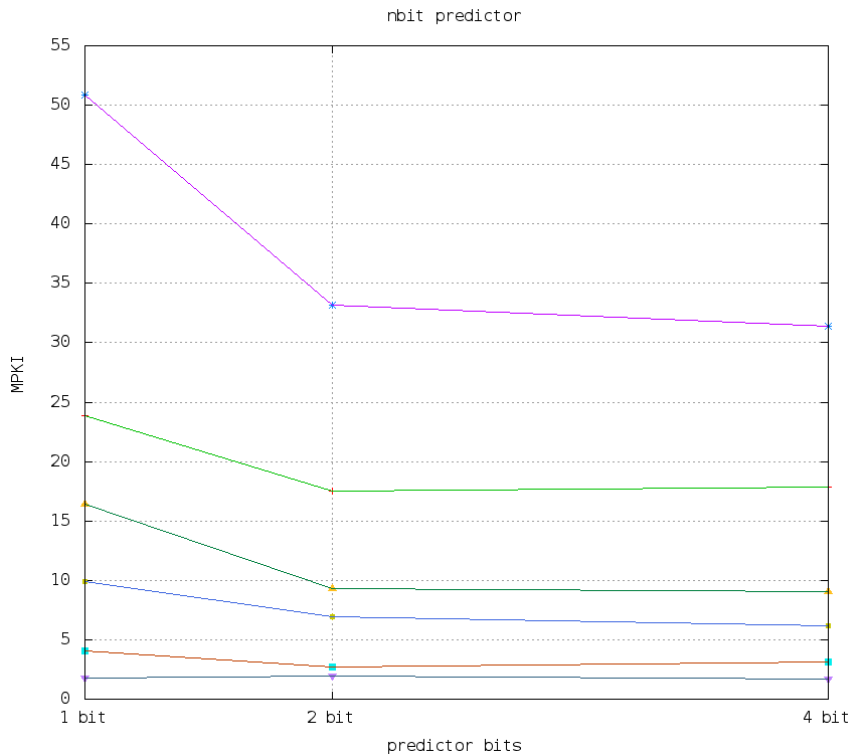
Σχήμα 1: 1-7 bit predictors

Όπως παρατηρούμε από το Σχήμα 1, βέλτιστος προβλέπτης για τα περισσότερα benchmarks είναι ο 4-bit predictor καθώς εκείνος έχει το χαμηλότερο missprediction rate χωρίς να έχει πολύ μεγάλη πολυπλοκότητα στο hardware.

A2

Διαφοροποιούμε την υλοποίηση και μελετάμε {1,2,4}-bit predictors χρησιμοποιώντας ως μετρική τα MPKI (Mispredictions Per Thousand Instructions) με μεταβλητό πλήθος BHT entries ώστε να έχουμε σταθερό μέγεθος hardware και ίσο με 32K.

HW	bits	BHT entries
32K	1	32K
32K	2	16K
32K	4	8K

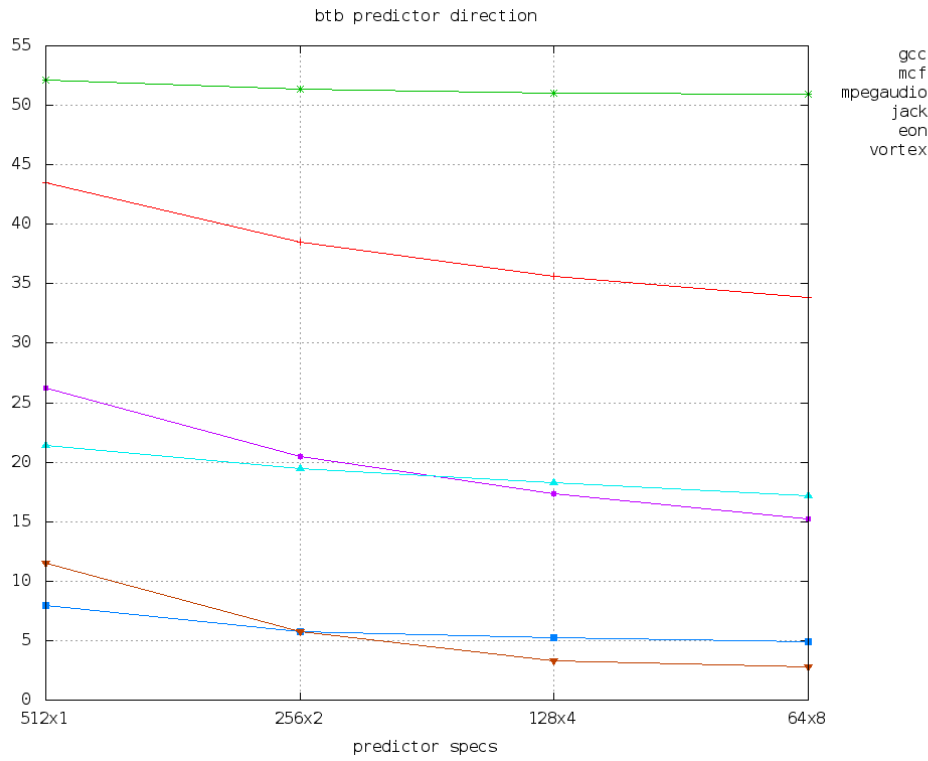


Σχήμα 2: 1,2 και 4 bit predictors

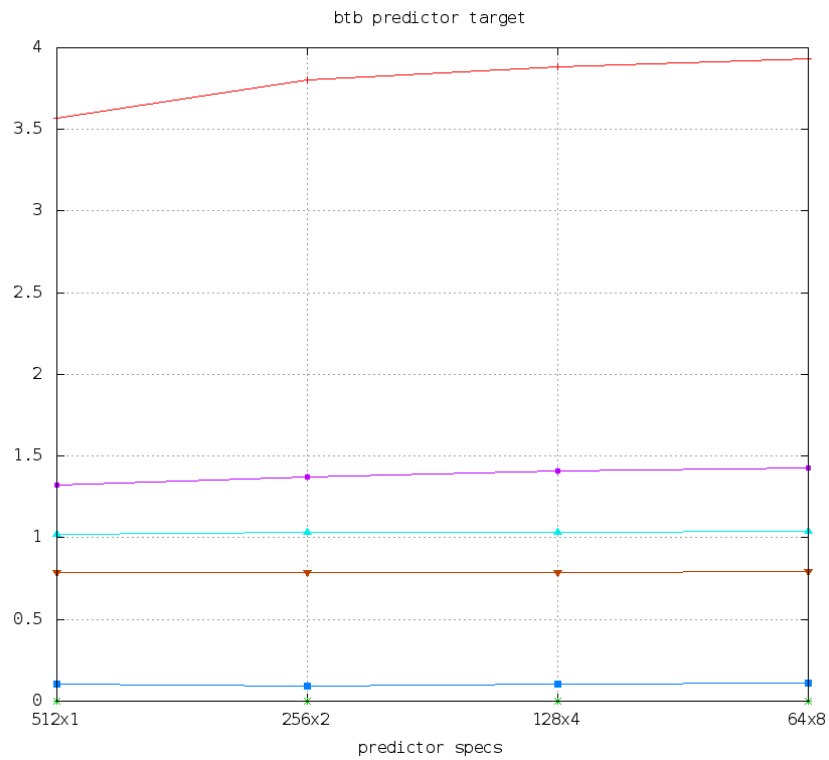
Και σε αυτή την περίπτωση που το hardware είναι περισσότερο (32K) καλύτερα φαίνεται να συμπεριφέρεται ο 4 bit predictor, συνεπώς αυτόν θα επιλέγαμε και σε αυτή τη φορά.

B1. Μελέτη του BTB

bttb_lines	bttb_assoc
512K	1
256K	2
128K	4
64K	8



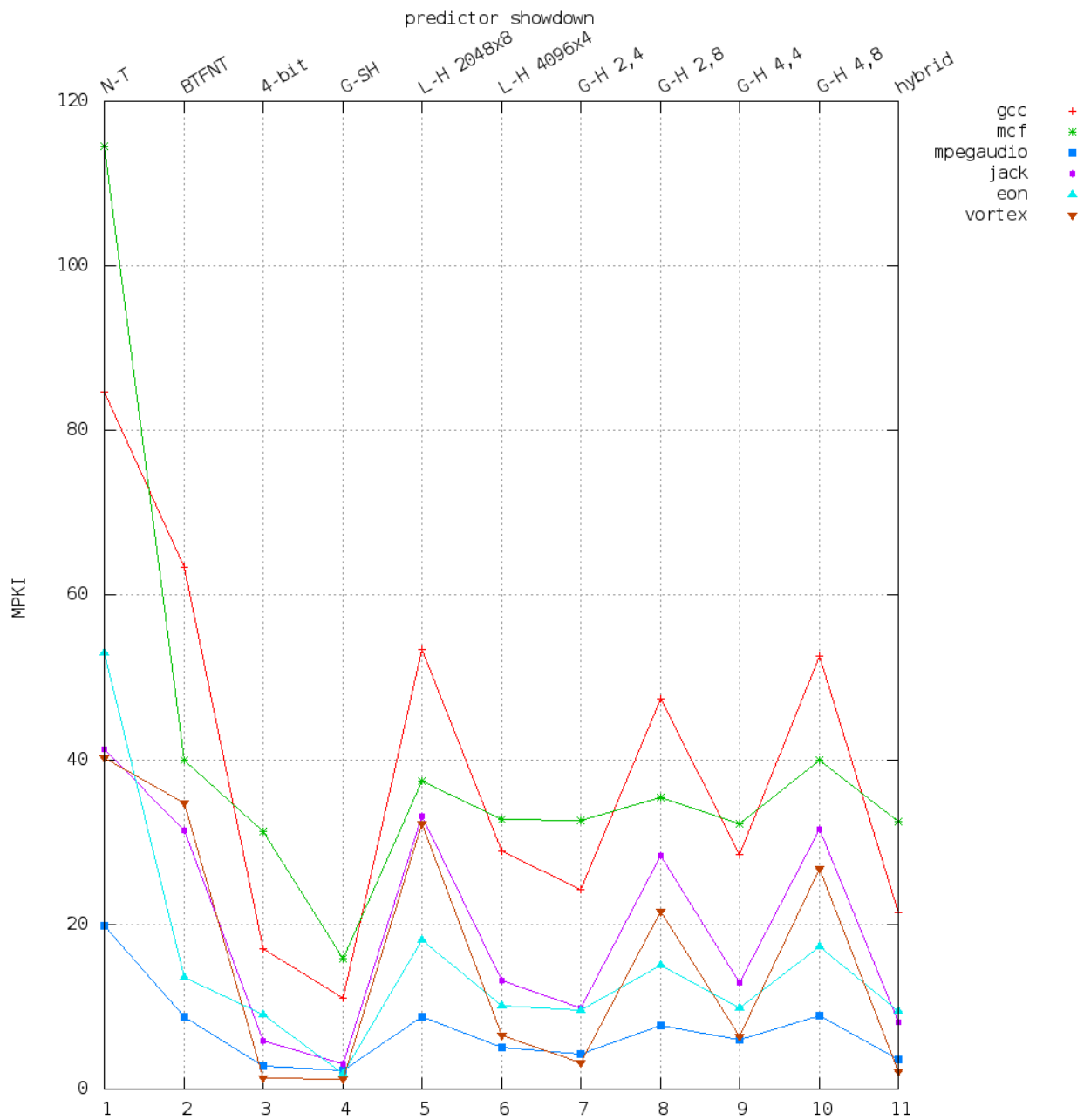
Σχήμα 3: Direction misspredictions (direction MPKI)



Σχήμα 4: Target Misspredictions (target MPKI)

Παρατηρούμε πως το target missprediction παραμένει σταθερό σχεδόν, σε αντίθεση με το direction missprediction που μειώνεται δραστικά στον 64x8 btb predictor. Αυτή είναι και η επιθυμητή οργάνωση για τον btb.

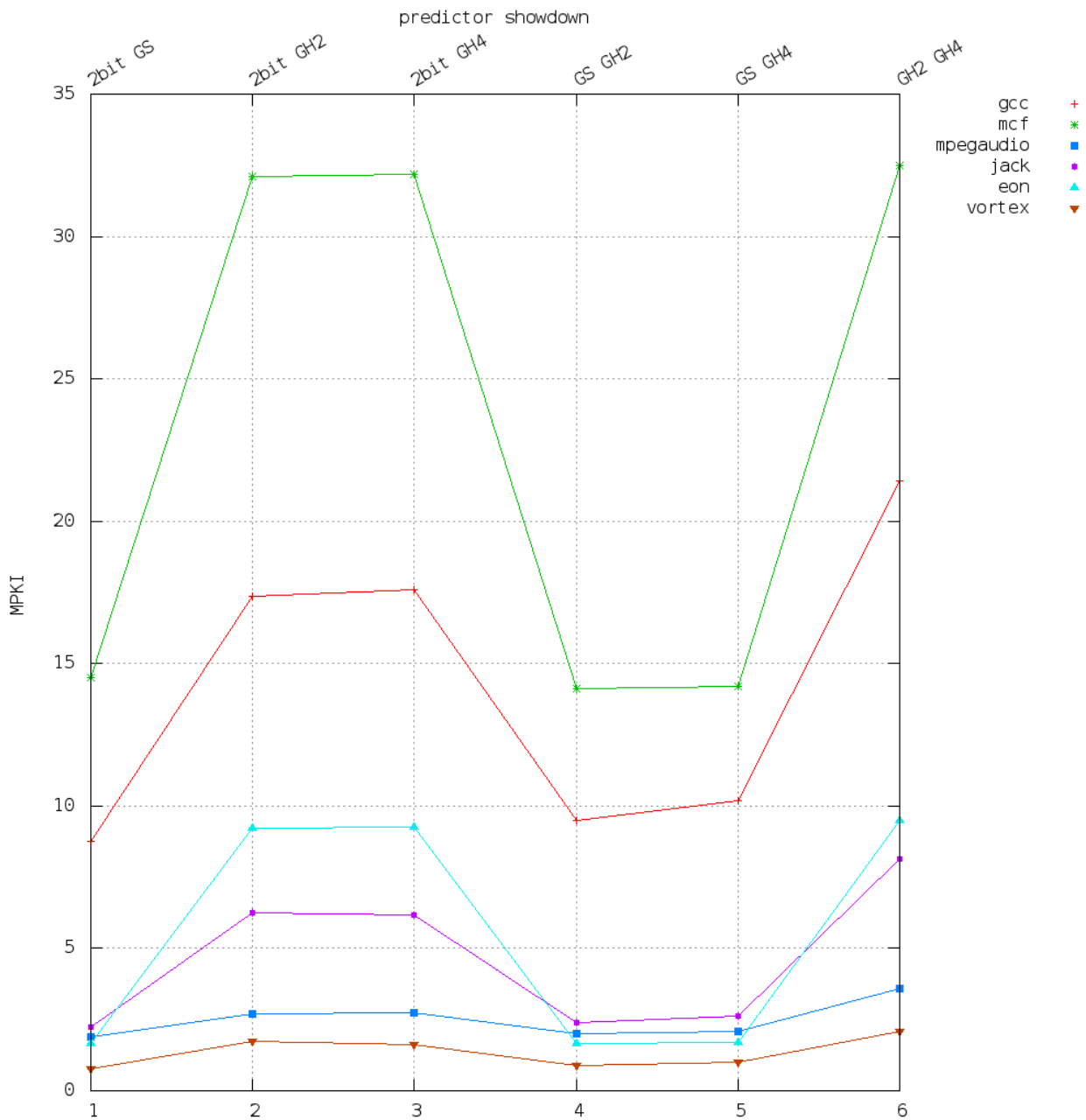
C1. Σύγκριση διαφορετικών predictors



Σχήμα 5: Σύγκριση predictors

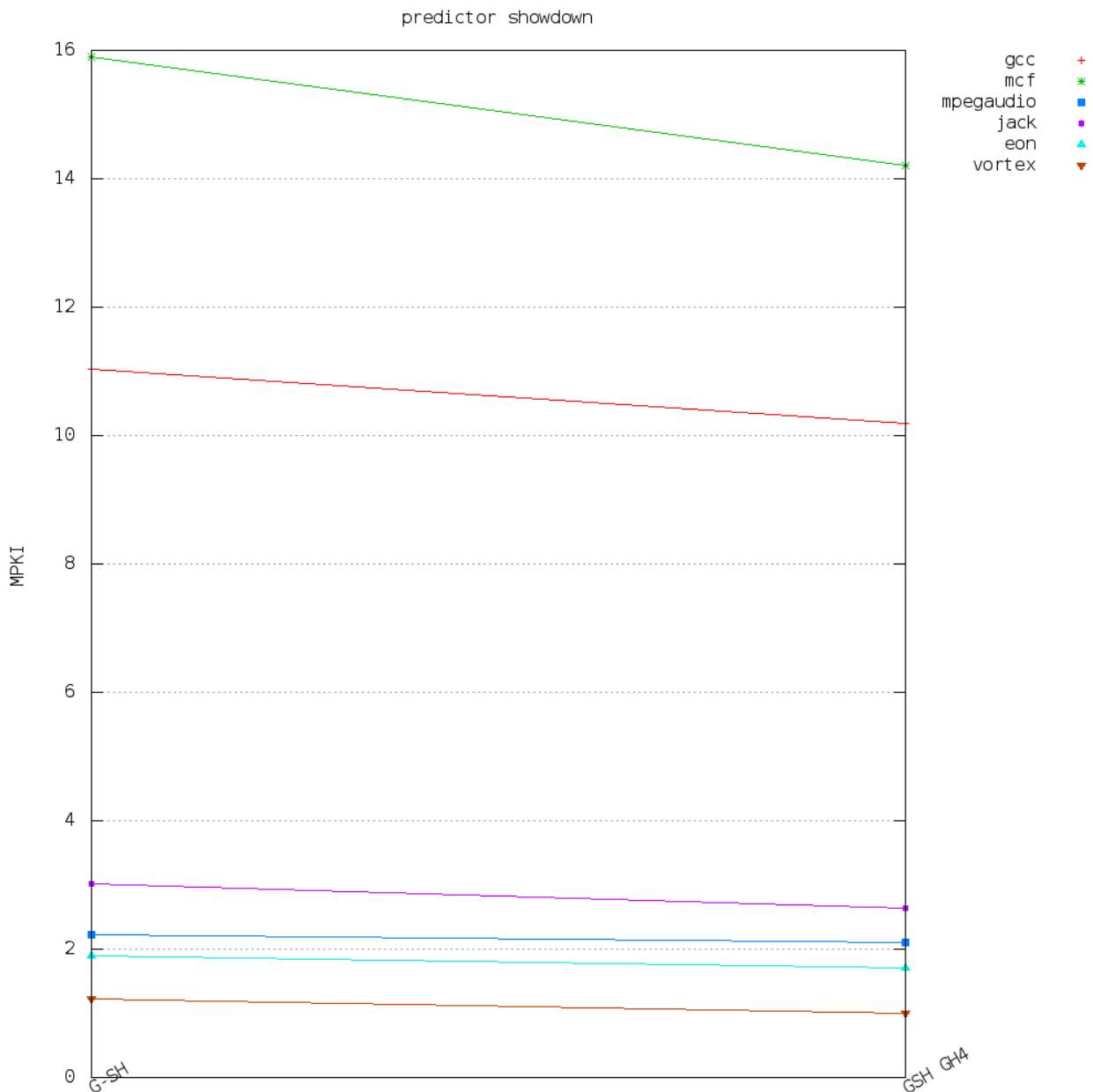
Από αυτούς καταλήγουμε πως καλύτερος είναι ο gshare predictor.

Tournament Hybrid predictors



Σχήμα 6: Σύγκριση hybrid predictors

Από όλους τους hybrid predictors που χρησιμοποιήσαμε, καλύτεροι αποδείχτηκαν οι gshare-globalhistory2 και gshare-globalhistory4 predictors. Τέλος συγκρίνουμε τον hybrid gshare-globalhistory4 και gshare predictor.



Σχήμα 7: Σύγκριση hybrid predictors

Καταλήγουμε πως ο hybrid predictor που δημιουργήσαμε είναι ελαφρώς καλύτερος από τον gshare predictor.

Ο κυρίως κώδικας που χρησιμοποιήσαμε ήταν ο ακόλουθος:

```

1 // predict.cpp
2 // This file contains the main function. The program accepts a single
3 // parameter: the name of a trace file. It drives the branch predictor
4 // simulation by reading the trace file and feeding the traces one at a time
5 // to the branch predictor.
6
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <string.h>
10 #include <assert.h>
11
12
13 #include <algorithm>
14
15 #include "branch.h"

```

```

16 #include "trace.h"
17 #include "predictor.h"
18 #include "btb.h"
19 #include "nt_predictor.h"
20 #include "btfmt_predictor.h"
21 #include "nbit_predictor.h"           //the .h files of the branch predictors' implementations
22 #include "gshare_predictor.h"
23 #include "localhistory_predictor.h"
24 #include "globalhistory_predictor.h"
25 #include "hybrid_predictor.h"
26
27
28 #define NR_PREDICTORS 11
29
30 using namespace std;
31
32 int main (int argc, char *argv[])
33 {
34
35     // make sure there is one parameter
36
37     if (argc != 2) {
38         fprintf (stderr, "Usage: %s <filename>.gz\n", argv[0]);
39         exit (1);
40     }
41
42     // open the trace file for reading
43
44     init_trace (argv[1]);
45
46     // initialize competitor's branch prediction code
47
48     // you can use more than one predictor in an array of predictors!!!
49
50     branch_predictor **p = new branch_predictor*[NR_PREDICTORS];
51
52     p[0] = new nt_predictor();
53     p[1] = new btfmt_predictor();
54     p[2] = new nbit_predictor(4);
55     p[3] = new gshare_predictor();
56     p[4] = new localhistory_predictor(2048, 8);
57     p[5] = new localhistory_predictor(4096, 4);
58     /* X=2 BHR=4 */
59     p[6] = new globalhistory_predictor(16384, 2, 4);
60     /* X=2 BHR=8 */
61     p[7] = new globalhistory_predictor(16384, 2, 8);
62     /* X=4 BHR=4 */
63     p[8] = new globalhistory_predictor(8192, 4, 4);
64     /* X=4 BHR=8 */
65     p[9] = new globalhistory_predictor(8192, 4, 8);
66
67     p[10] = new hybrid_predictor(512);
68
69     long long int
70     tmiss[NR_PREDICTORS],           // number of target mispredictions
71     dmiss[NR_PREDICTORS];          // number of direction mispredictions
72
73     fill( tmiss, tmiss+NR_PREDICTORS, 0);
74     fill( dmiss, dmiss+NR_PREDICTORS, 0);
75     // keep looping until end of file
76
77     for (;;) {
78
79         // get a trace
80
81         trace *t = read_trace ();
82
83         // NULL means end of file
84
85         if (!t) break;
86
87         // send this trace to the competitor's code for prediction
88
89         branch_update *u;
90

```



```

91      /* static not taken */
92      u = p[0]->predict(t->bi);
93      p[0]->update(u, t->taken, t->target);
94      dmiss[0] += u->direction_prediction() != t->taken;
95      tmiss[0] += u->target_prediction() != t->target;
96
97
98      /* backward taken forward not taken */
99      ((btfnt_predictor *)p[1])->set_target(t->target > t->bi.address);
100     u = p[1]->predict(t->bi);
101     p[1]->update(u, t->taken, t->target);
102     dmiss[1] += u->direction_prediction() != t->taken;
103     tmiss[1] += u->target_prediction() != t->target;
104
105
106     /* 4 bit predictor */
107     u = p[2]->predict(t->bi);
108     p[2]->update(u, t->taken, t->target);
109     dmiss[2] += u->direction_prediction() != t->taken;
110     tmiss[2] += u->target_prediction() != t->target;
111
112     /* gshare predictor */
113     u = p[3]->predict(t->bi);
114     p[3]->update(u, t->taken, t->target);
115     dmiss[3] += u->direction_prediction() != t->taken;
116     tmiss[3] += u->target_prediction() != t->target;
117
118     /* local history 2 level predictor X=2048 */
119     u = p[4]->predict(t->bi);
120     p[4]->update(u, t->taken, t->target);
121     dmiss[4] += u->direction_prediction() != t->taken;
122     tmiss[4] += u->target_prediction() != t->target;
123
124     /* local history 2 level predictor X=4096 */
125     u = p[5]->predict(t->bi);
126     p[5]->update(u, t->taken, t->target);
127     dmiss[5] += u->direction_prediction() != t->taken;
128     tmiss[5] += u->target_prediction() != t->target;
129
130     /* global history X=2 BHR=4 */
131     u = p[6]->predict(t->bi);
132     p[6]->update(u, t->taken, t->target);
133     dmiss[6] += u->direction_prediction() != t->taken;
134     tmiss[6] += u->target_prediction() != t->target;
135
136     /* global history X=2 BHR=8 */
137     u = p[7]->predict(t->bi);
138     p[7]->update(u, t->taken, t->target);
139     dmiss[7] += u->direction_prediction() != t->taken;
140     tmiss[7] += u->target_prediction() != t->target;
141
142     /* global history X=4 BHR=4 */
143     u = p[8]->predict(t->bi);
144     p[8]->update(u, t->taken, t->target);
145     dmiss[8] += u->direction_prediction() != t->taken;
146     tmiss[8] += u->target_prediction() != t->target;
147
148     /* global history X=4 BHR=8 */
149     u = p[9]->predict(t->bi);
150     p[9]->update(u, t->taken, t->target);
151     dmiss[9] += u->direction_prediction() != t->taken;
152     tmiss[9] += u->target_prediction() != t->target;
153
154     u = p[10]->predict(t->bi);
155     p[10]->update(u, t->taken, t->target);
156     dmiss[10] += u->direction_prediction() != t->taken;
157     tmiss[10] += u->target_prediction() != t->target;
158 }
159
160 // done reading traces
161
162 end_trace ();
163
164 // give final mispredictions per kilo-instruction and exit.
165 // each trace represents exactly 100 million instructions.

```

```

166
167     for(int i = 0; i < NR_PREDICTORS; i++) {
168         printf("%d\t%.3f\n",i+1,1000.0 * (dmiss[i]/1e8));
169         delete p[i];
170     }
171     delete [] p;
172
173     exit (0);
174 }

```

Static Not-Taken

```

1  /* .....
2  * File Name : nt_predictor.h
3  * Creation Date : 19-05-2012
4  * Last Modified : Sat 19 May 2012 10:09:02 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * .....*/
7
8  // nt_predictor.h
9
10 #ifndef NT_PREDICTOR_H
11 #define NT_PREDICTOR_H
12 #include "predictor.h"
13
14
15 class nt_update : public branch_update
16 {
17 public:
18     unsigned int index;
19 };
20 class nt_predictor: public branch_predictor
21 {
22     nt_update u;
23     branch_update *predict (branch_info & b);
24     void update (branch_update *u, bool taken, unsigned int target);
25 };
26
27 #endif

```

```

1  /* .....
2  * File Name : nt_predictor.cpp
3  * Creation Date : 19-05-2012
4  * Last Modified : Sat 19 May 2012 10:07:46 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * .....*/
7
8  #include "nt_predictor.h"
9
10 branch_update *nt_predictor::predict (branch_info & b)
11 {
12     if (b.br_flags & BR_CONDITIONAL) {
13         u.direction_prediction (false);
14     } else {
15         u.direction_prediction (true);
16     }
17     u.target_prediction (0);
18     return &u;
19 }
20 void nt_predictor::update (branch_update *u, bool taken, unsigned int target)
21 {
22 }

```

Static Backward Taken Forward Not Taken

```

1  /* .....
2  * File Name : btfn_predictor.h
3  * Creation Date : 19-05-2012
4  * Last Modified : Sat 26 May 2012 06:02:21 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * .....*/
7  // btfn_predictor.h

```

```

8
9  #ifndef BTFLT_PREDICTOR_H
10 #define BTFLT_PREDICTOR_H
11
12 #include <math.h>
13 #include "predictor.h"
14
15 class btflt_predictor : public branch_predictor
16 {
17 public:
18
19     branch_update u;
20     branch_info bi;
21     bool jump;
22
23     void set_target(bool t);
24     branch_update *predict (branch_info & b);
25     void update (branch_update *u, bool taken, unsigned int target);
26 };
27
28 #endif

```

```

1  /* .....
2  * File Name : btflt_predictor.cpp
3  * Creation Date : 19-05-2012
4  * Last Modified : Sat 26 May 2012 06:03:49 PM EEST
5  * Created By : Greg Liras <greyliras@gmail.com>
6  .....*/
7  #include "btflt_predictor.h"
8
9  void btflt_predictor::set_target (bool t) {
10     jump = t;
11 }
12 branch_update *btflt_predictor::predict (branch_info & b)
13 {
14     bi = b;
15     if (b.br_flags & BR_CONDITIONAL) {
16         if (jump) {
17             u.direction_prediction (false);
18         }
19         else {
20             u.direction_prediction(true);
21         }
22     } else {
23         u.direction_prediction (true);
24     }
25     u.target_prediction (0);
26     return &u;
27 }
28
29 void btflt_predictor::update (branch_update *u, bool taken, unsigned int target)
30 {
31 }
32

```

4-bit predictor

```

1  // nbit_predictor.h
2  //
3  //
4
5  #ifndef N_BIT_PREDICTOR_H
6  #define N_BIT_PREDICTOR_H
7
8
9  #include <math.h>
10 #include <string.h>
11 #include "predictor.h"
12
13 class nbit_update : public branch_update
14 {
15 public:
16     unsigned int index;

```



```

13     unsigned int index;
14 };
15
16 /*
17  * H klash gshare_predictor kihronomei thn klash
18  * branch_predictor kai kanei override tis me8odous
19  * predict kai update
20  */
21
22 class gshare_predictor : public branch_predictor
23 {
24 public:
25     #define HISTORY_LENGTH      14
26     #define GSP_TABLE_BITS      15
27     gshare_update u;
28     branch_info bi;
29     unsigned int history;
30     unsigned char tab[1<<GSP_TABLE_BITS];
31
32     gshare_predictor (void);
33     branch_update *predict (branch_info & b);
34
35     void update (branch_update *u, bool taken, unsigned int target);
36 };
37 #endif

```

```

1  /* .....
2  * File Name : gshare_predictor.cpp
3  * Creation Date : 20-05-2012
4  * Last Modified : Sat 26 May 2012 06:16:06 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  * .....*/
7
8  #include "gshare_predictor.h"
9  #include <algorithm>
10 gshare_predictor::gshare_predictor (void) : history(0)
11 {
12     std::fill (tab, tab+sizeof (tab),0);
13 }
14 branch_update *gshare_predictor::predict (branch_info & b)
15 {
16     bi = b;
17
18     /*
19      * 0 gshare xrhsimopoieitai mono gia conditional branches.
20      * Ta unconditional ginontai predicted panta TAKEN
21      */
22
23     if (b.br_flags & BR_CONDITIONAL) {
24         u.index = (history << (GSP_TABLE_BITS - HISTORY_LENGTH)) ^ (b.address & ((1<<GSP_TABLE_BITS)-1));
25         u.direction_prediction (tab[u.index] >> 1);
26     } else {
27         u.direction_prediction (true);
28     }
29
30     // 0 gshare den kanei target prediction, gia auto to Setoume sto 0.
31
32     u.target_prediction (0);
33     return &u;
34 }
35 void gshare_predictor::update (branch_update *u, bool taken, unsigned int target)
36 {
37     //0 gshare xrhsimopoieitai mono gia conditional branches
38     if (bi.br_flags & BR_CONDITIONAL) {
39         unsigned char *c = &tab[((gshare_update*)u)->index];
40         if (taken) {
41             if (*c < 3) (*c)++;
42         } else {
43             if (*c > 0) (*c)--;
44         }
45         history <= 1;
46         history |= taken;
47         history &= (1<<HISTORY_LENGTH)-1;
48     }
49 }

```

Local-History two-level predictors

```
1  #ifndef localhistory_predictor_H
2  #define localhistory_predictor_H
3
4  #include <cmath>
5  #include "predictor.h"
6
7  class localhistory_update : public branch_update
8  {
9  public:
10     unsigned int phtindex;
11     unsigned int bhtindex;
12 };
13
14 class localhistory_predictor : public branch_predictor
15 {
16 public:
17
18     localhistory_update u;
19     branch_info bi;
20
21     int p_counter_limit;
22     int b_counter_limit;
23
24     int pht_entries;
25     int pht_nbit;
26
27     int bht_entries;
28     int bht_entry_length;
29
30     unsigned char *pht;
31     unsigned char *bht;
32
33     unsigned int pht_mask;
34
35
36     localhistory_predictor (int x,int z);
37     branch_update *predict (branch_info & b);
38
39     void update (branch_update *u, bool taken, unsigned int target);
40 };
41 #endif

```



```
1  #include "localhistory_predictor.h"
2  #include <algorithm>
3  #include <cstring>
4
5  localhistory_predictor::localhistory_predictor (int x , int z) : bht_entries(x), bht_entry_length(z)
6  {
7      pht_entries=8192;
8      pht_nbit=2;
9
10     pht = new unsigned char [pht_entries];
11     memset (pht, 0, sizeof (pht));
12
13     bht = new unsigned char [x];
14     memset (bht, 0, sizeof (bht));
15
16     pht_mask = ((1<<(((int) log2(pht_entries))-bht_entry_length))-1);
17
18     p_counter_limit = (1<<pht_nbit);
19     b_counter_limit = (1<<bht_entry_length);
20 }
21
22 branch_update *localhistory_predictor::predict (branch_info & b)
23 {
24     bi=b;
25     if (b.br_flags & BR_CONDITIONAL) {
26
27         u.bhtindex = (b.address & (bht_entries-1));
28         u.phtindex = ((b.address & pht_mask)<<bht_entry_length);
29         u.phtindex |= bht[u.bhtindex];
30
31         u.direction_prediction(pht[u.phtindex]>>(pht_nbit-1));

```

```

32     } else {
33         u.direction_prediction(true);
34     }
35     u.target_prediction (0);
36     return &u;
37 }
38
39 void localhistory_predictor::update (branch_update *u, bool taken, unsigned int target)
40 {
41     if (bi.br_flags & BR_CONDITIONAL) {
42
43         unsigned char *c = &pht[((localhistory_update*)u)->phtindex];
44         unsigned char *d = &bht[((localhistory_update*)u)->bhtindex];
45
46         if (taken) {
47             if (*c < p_counter_limit)
48                 (*c)++;
49         }
50         else {
51             if (*c > 0)
52                 (*c)--;
53         }
54         (*d) <= 1;
55         (*d) |= taken;
56         (*d) &= b_counter_limit;
57
58     }
59 }
60

```

Global-History two-level predictors

```

1  #ifndef GLOBALHISTORY_PREDICTOR_H
2  #define GLOBALHISTORY_PREDICTOR_H
3
4  #include <cmath>
5  #include <cstring>
6  #include <cstdio>
7  #include "predictor.h"
8
9  class globalhistory_update : public branch_update
10 {
11 public:
12     unsigned int index;
13 };
14
15 class globalhistory_predictor : public branch_predictor
16 {
17 public:
18
19     globalhistory_update u;
20     branch_info bi;
21
22     int p_counter_limit;
23     int b_counter_limit;
24     unsigned int pht_mask;
25
26     int pht_entries;
27     int pht_nbit;
28
29     int bhr;
30     int bhr_length;
31
32     unsigned char **pht;
33
34     globalhistory_predictor (int x,int y, int z);
35     branch_update *predict (branch_info & b);
36
37     void update (branch_update *u, bool taken, unsigned int target);
38 };
39 #endif
40
41 #include "globalhistory_predictor.h"
42 #include <algorithm>

```

```

3
4 globalhistory_predictor::globalhistory_predictor (int x , int y, int z) :pht_entries(x), pht_nbit(y), bhr_length(z)
5 {
6
7     bhr=0;
8
9     pht = new unsigned char *[1<<bhr_length];
10
11     for( int i = 0; i < (1<<bhr_length); i++)
12     {
13         pht[i] = new unsigned char [pht_entries>>bhr_length];
14     }
15
16     for (int i = 0; i < (1<<bhr_length); i++)
17     {
18         memset(pht[i], 0, sizeof (pht[i]));
19     }
20
21     p_counter_limit = (1<<pht_nbit);
22     b_counter_limit = (1<<bhr_length);
23     pht_mask = ((pht_entries>>bhr_length)-1);
24 }
25
26 branch_update *globalhistory_predictor::predict (branch_info & b)
27 {
28     bi=b;
29     if (b.br_flags & BR_CONDITIONAL) {
30
31         u.index = (b.address & pht_mask);
32
33         u.direction_prediction(pht[bhr][u.index]>>(pht_nbit-1));
34     } else {
35         u.direction_prediction (true);
36     }
37     u.target_prediction (0);
38     return &u;
39 }
40
41 void globalhistory_predictor::update (branch_update *u, bool taken, unsigned int target)
42 {
43     if (bi.br_flags & BR_CONDITIONAL) {
44
45         unsigned char *c = &pht[bhr][((globalhistory_update*)u)->index];
46
47         if (taken) {
48             if (*c < p_counter_limit)
49                 (*c)++;
50         }
51         else {
52             if (*c > 0)
53                 (*c)--;
54         }
55         bhr <= 1;
56         bhr |= taken;
57         bhr &= b_counter_limit;
58     }
59 }
60
61
62 /* .....
63 * File Name : hybrid_predictor.h
64 * Creation Date : 27-05-2012
65 * Last Modified : Sun 27 May 2012 11:04:25 PM EEST
66 * Created By : Greg Liras <gregliras@gmail.com>
67 .....*/
68
69 #ifndef HYBRID_PREDICTOR_H
70 #define HYBRID_PREDICTOR_H
71
72 #include "predictor.h"
73 #include "globalhistory_predictor.h"
74
75 class hybrid_update : public branch_update
76 {
77 public:

```



```

17     branch_update *ups[2];
18     unsigned int index;
19     unsigned int pred;
20
21 };
22 class hybrid_predictor : public branch_predictor
23 {
24 public:
25
26     hybrid_update u;
27     branch_info bi;
28     int counter_limit;
29
30     branch_predictor **preds;
31
32
33     int pht_entries;
34     int pht_bits_length;
35
36     unsigned char *tab;
37
38     hybrid_predictor(int entries);
39     branch_update *predict (branch_info & b);
40     void update (branch_update *u, bool taken, unsigned int target);
41
42
43
44
45 };
46
47
48 #endif

```

```

1  /* .....
2  * File Name : hybrid_predictor.cpp
3  * Creation Date : 27-05-2012
4  * Last Modified : Sun 27 May 2012 11:09:18 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  .....*/
7
8  #include "hybrid_predictor.h"
9
10 hybrid_predictor::hybrid_predictor(int entries) : pht_entries(entries)
11 {
12     tab = new unsigned char[pht_entries];
13     memset(tab,0,sizeof(tab));
14     pht_bits_length = 2;
15     counter_limit = 2;
16     u.pred = 0;
17     u.index = 0;
18     preds = new branch_predictor*[2];
19 }
20
21
22 branch_update *hybrid_predictor::predict (branch_info & b)
23 {
24     bi = b;
25     if (b.br_flags & BR_CONDITIONAL) {
26         u.index = (b.address & (pht_entries-1));
27         u.pred = tab[u.index] >> (pht_bits_length - 1);
28     }
29     u.ups[0] = (branch_update *) preds[0]->predict(b);
30     u.ups[1] = (branch_update *) preds[1]->predict(b);
31     unsigned int thepred = u.pred & 1;
32     u.direction_prediction(u.ups[thepred]->direction_prediction());
33     return &u;
34 }
35 void hybrid_predictor::update (branch_update *u, bool taken, unsigned int target)
36 {
37     if (bi.br_flags & BR_CONDITIONAL) {
38         unsigned char *c = &tab[((hybrid_update*)u)->index];
39         bool pred0_result = ((hybrid_update *)u)->ups[0]->direction_prediction() == taken;
40         bool pred1_result = ((hybrid_update *)u)->ups[1]->direction_prediction() == taken;
41         int meta_update = pred1_result - pred0_result;
42

```

```

43         if (meta_update > 0 && *c < 3)
44             (*c)++;
45         else if (meta_update < 0 && *c > 0)
46             (*c)--;
47         (*c) &= 3;
48         preds[0]->update(((hybrid_update *)u)->ups[0], taken, target);
49         preds[1]->update(((hybrid_update *)u)->ups[1], taken, target);
50
51     }
52 }

```

2-bit global history BHT=2

```

1  /* -.-.-.-.-.
2  * File Name : hybrid_2bit_GH2_predictor.cpp
3  * Creation Date : 27-05-2012
4  * Last Modified : Sun 27 May 2012 10:38:21 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  -.-.-.-.-.*/
7
8
9
10
11 #include "hybrid_2bit_GH2_predictor.h"
12
13
14 hybrid_2bit_GH2_predictor::hybrid_2bit_GH2_predictor(int entries): hybrid_predictor(entries)
15 {
16     preds[0] = new nbit_predictor(2);
17     preds[1] = new globalhistory_predictor(8192,2,2);
18 }
19
20
21
22
23
24
25
26
27
28
29 /* -.-.-.-.-.
30 * File Name : hybrid_2bit_GH2_predictor.h
31 * Creation Date : 27-05-2012
32 * Last Modified : Sun 27 May 2012 10:45:58 PM EEST
33 * Created By : Greg Liras <gregliras@gmail.com>
34 -.-.-.-.-.*/
35
36
37
38
39 #ifndef HYBRID_2BIT_GH2_PREDICTOR_H
40 #define HYBRID_2BIT_GH2_PREDICTOR_H
41
42 #include "predictor.h"
43 #include "gshare_predictor.h"
44 #include "nbit_predictor.h"
45 #include "hybrid_predictor.h"
46
47 class hybrid_2bit_GH2_predictor : public hybrid_predictor
48 {
49 public:
50
51     hybrid_2bit_GH2_predictor(int entries);
52
53 };
54
55 #endif

```

2-bit global history BHT=4

```

1  /* -.-.-.-.-.
2  * File Name : hybrid_2bit_GH4_predictor.cpp
3  * Creation Date : 27-05-2012
4  * Last Modified : Sun 27 May 2012 10:40:56 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  -.-.-.-.-.*/
7
8
9

```



```

17
18 class hybrid_2bit_GS_predictor : public hybrid_predictor
19 {
20 public:
21
22     hybrid_2bit_GS_predictor(int entries);
23
24
25
26
27 };
28
29
30 #endif

```

global history BHT=2,4

```

1  /* ..... */
2  * File Name : hybrid_GH2_GH4_predictor.cpp
3  * Creation Date : 27-05-2012
4  * Last Modified : Sun 27 May 2012 11:16:20 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  .....*/
7
8
9
10
11 #include "hybrid_GH2_GH4_predictor.h"
12
13
14 hybrid_GH2_GH4_predictor::hybrid_GH2_GH4_predictor(int entries): hybrid_predictor(entries)
15 {
16     preds[0] = new globalhistory_predictor(8192,2,2);
17     preds[1] = new globalhistory_predictor(8192,2,4);
18 }

```

```

1  /* .....*/
2  * File Name : hybrid_GH2_GH4_predictor.h
3  * Creation Date : 27-05-2012
4  * Last Modified : Sun 27 May 2012 11:16:25 PM EEST
5  * Created By : Greg Liras <gregliras@gmail.com>
6  .....*/
7
8
9
10
11 #ifndef HYBRID_GH2_GH4_PREDICTOR_H
12 #define HYBRID_GH2_GH4_PREDICTOR_H
13
14 #include "predictor.h"
15 #include "gshare_predictor.h"
16 #include "nbit_predictor.h"
17 #include "hybrid_predictor.h"
18
19 class hybrid_GH2_GH4_predictor : public hybrid_predictor
20 {
21 public:
22
23     hybrid_GH2_GH4_predictor(int entries);
24
25
26 };
27
28
29 #endif

```

gshare global history BHT=2

[illegible]

```

6  |  .....*/
7  |
8  |
9  |
10 |
11 | #include "hybrid_GS_GH2_predictor.h"
12 |
13 |
14 | hybrid_GS_GH2_predictor::hybrid_GS_GH2_predictor(int entries): hybrid_predictor(entries)
15 | {
16 |     preds[0] = new gshare_predictor();
17 |     preds[1] = new globalhistory_predictor(8192,2,2);
18 | }

1  | /* .....
2  |  * File Name : hybrid_GS_GH2_predictor.h
3  |  * Creation Date : 27-05-2012
4  |  * Last Modified : Sun 27 May 2012 11:12:16 PM EEST
5  |  * Created By : Greg Liras <gregliras@gmail.com>
6  |  .....*/
7  |
8  |
9  |
10 |
11 | #ifndef HYBRID_GS_GH2_PREDICTOR_H
12 | #define HYBRID_GS_GH2_PREDICTOR_H
13 |
14 | #include "predictor.h"
15 | #include "gshare_predictor.h"
16 | #include "nbit_predictor.h"
17 | #include "hybrid_predictor.h"
18 |
19 | class hybrid_GS_GH2_predictor : public hybrid_predictor
20 | {
21 | public:
22 |
23 |     hybrid_GS_GH2_predictor(int entries);
24 |
25 |
26 | };
27 |
28 |
29 | #endif

```

gshare global history BHT=4

```

1  | /* .....
2  |  * File Name : hybrid_GS_GH4_predictor.cpp
3  |  * Creation Date : 27-05-2012
4  |  * Last Modified : Sun 27 May 2012 11:14:51 PM EEST
5  |  * Created By : Greg Liras <gregliras@gmail.com>
6  |  .....*/
7  |
8  |
9  |
10 |
11 | #include "hybrid_GS_GH4_predictor.h"
12 |
13 |
14 | hybrid_GS_GH4_predictor::hybrid_GS_GH4_predictor(int entries): hybrid_predictor(entries)
15 | {
16 |     preds[0] = new gshare_predictor();
17 |     preds[1] = new globalhistory_predictor(8192,2,4);
18 | }

1  | /* .....
2  |  * File Name : hybrid_GS_GH4_predictor.h
3  |  * Creation Date : 27-05-2012
4  |  * Last Modified : Sun 27 May 2012 11:15:00 PM EEST
5  |  * Created By : Greg Liras <gregliras@gmail.com>
6  |  .....*/
7  |
8  |
9  |

```

```

10
11 #ifndef HYBRID_GS_GH4_PREDICTOR_H
12 #define HYBRID_GS_GH4_PREDICTOR_H
13
14 #include "predictor.h"
15 #include "gshare_predictor.h"
16 #include "nbit_predictor.h"
17 #include "hybrid_predictor.h"
18
19 class hybrid_GS_GH4_predictor : public hybrid_predictor
20 {
21 public:
22
23     hybrid_GS_GH4_predictor(int entries);
24
25 };
26
27
28
29 #endif

```