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

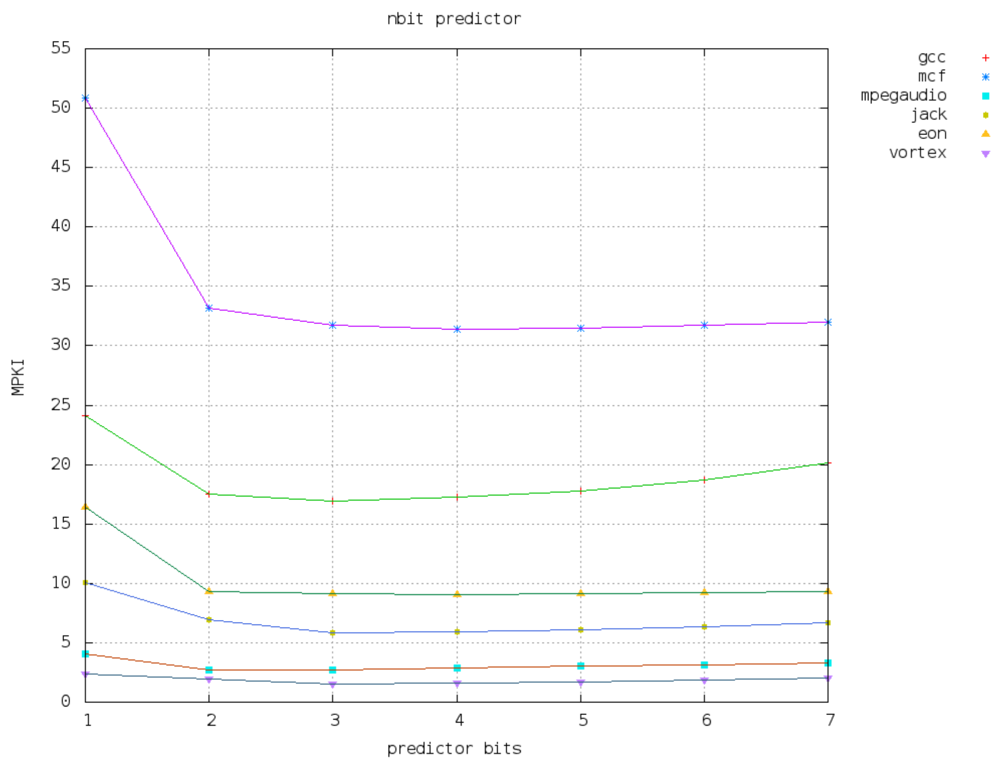
Εισαγωγή

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.



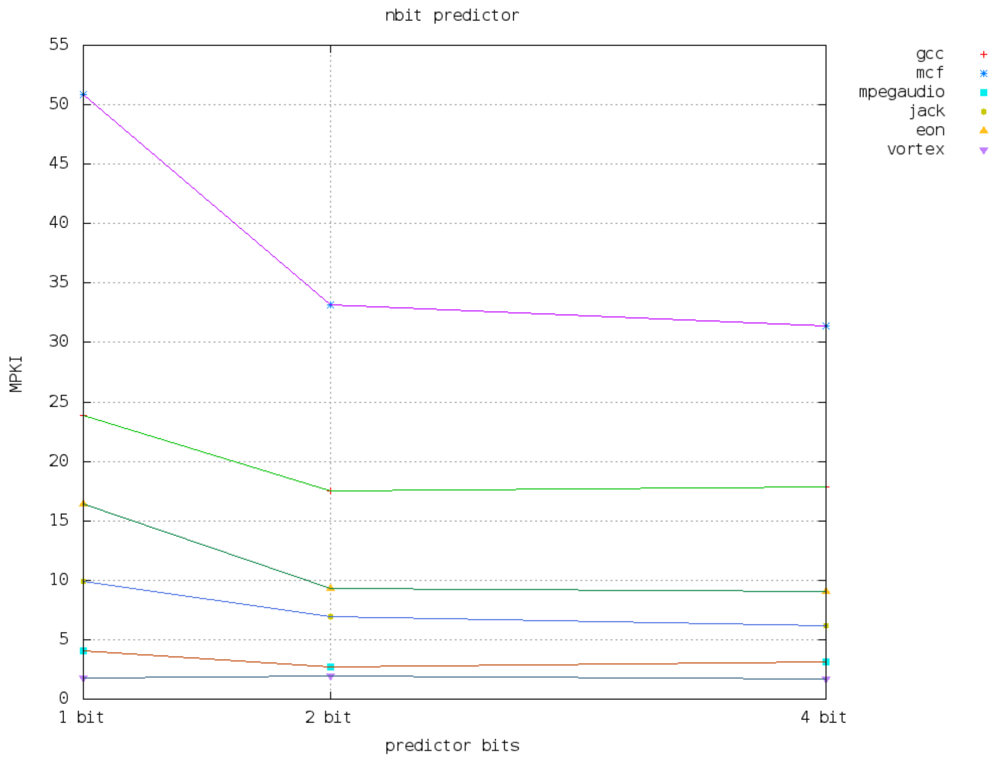
Σχήμα 1: 1-7 bit predictors

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

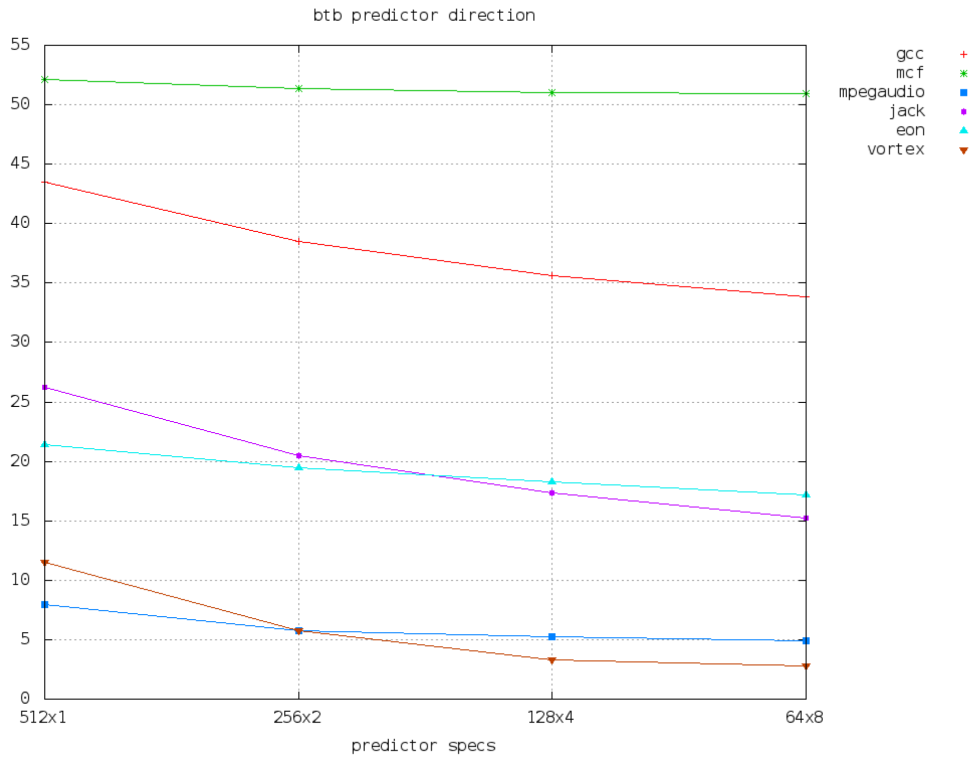


Σχήμα 2: 1,2 και 4 bit predictors

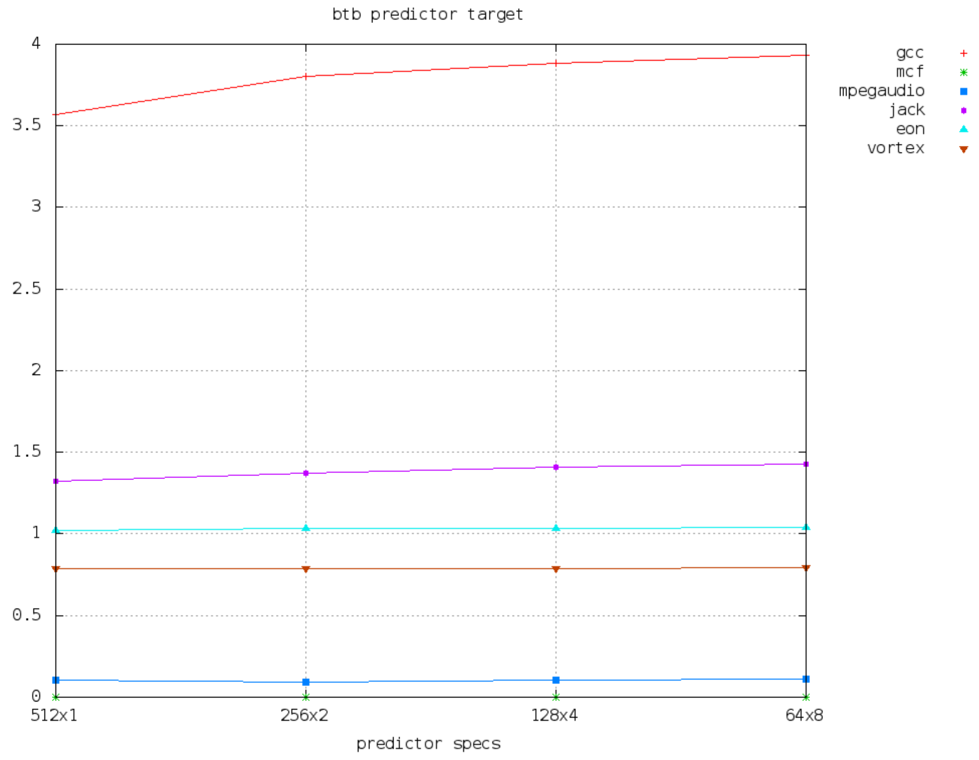
Και σε αυτή την περίπτωση που το hardware είναι περισσότερο (32K) καλύτερα φαίνεται να συμπεριφέρεται ο 4 bit predictor, συνεπώς αυτόν θα επιλέγαμε και σε αυτή τη φορά.

B1. Μελέτη του BTB

btb_lines	btb_assoc
512K	1
256K	2
128K	4
64K	8



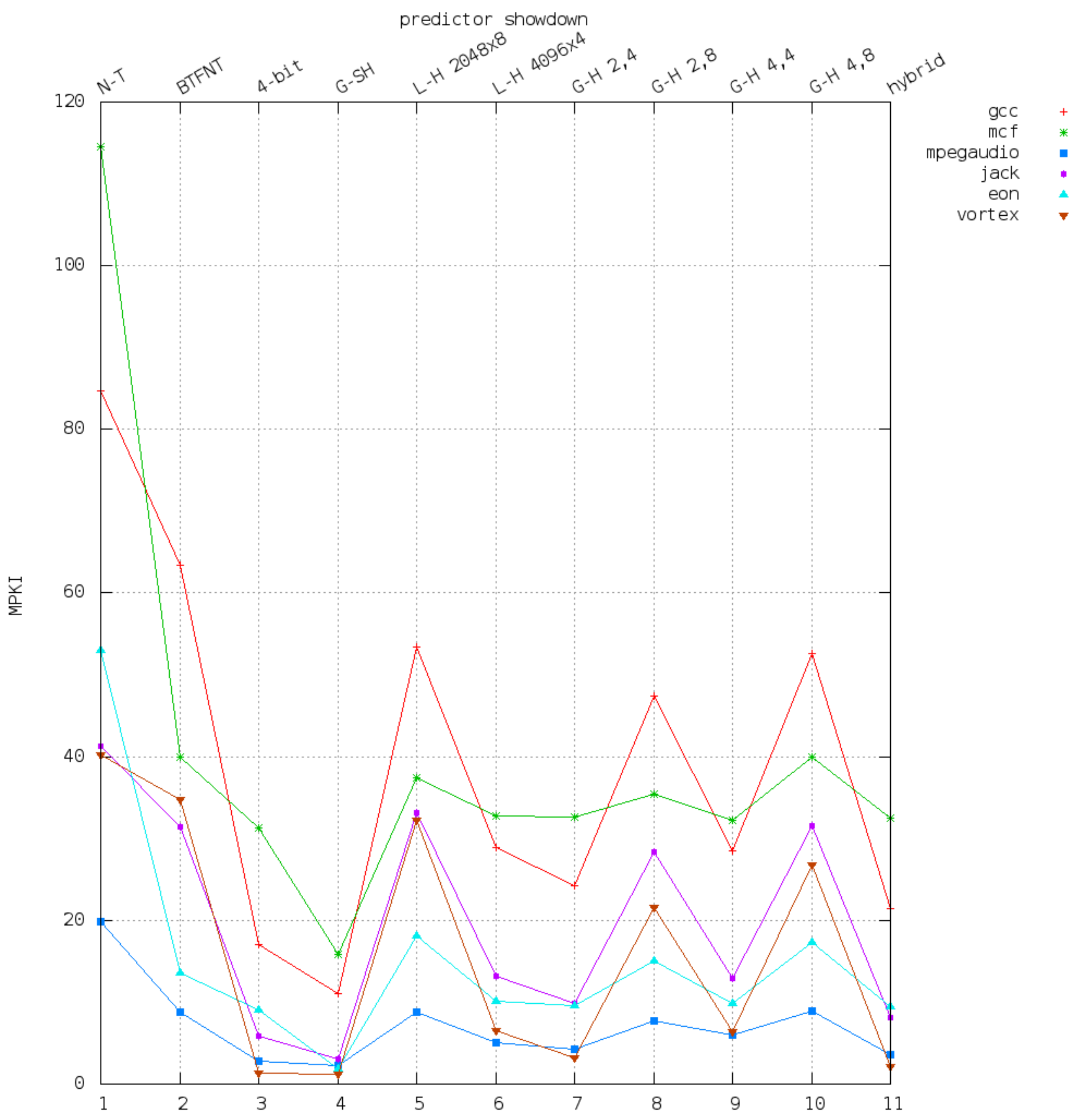
Σχήμα 3: Direction misspredictions (direction MPKI)



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.

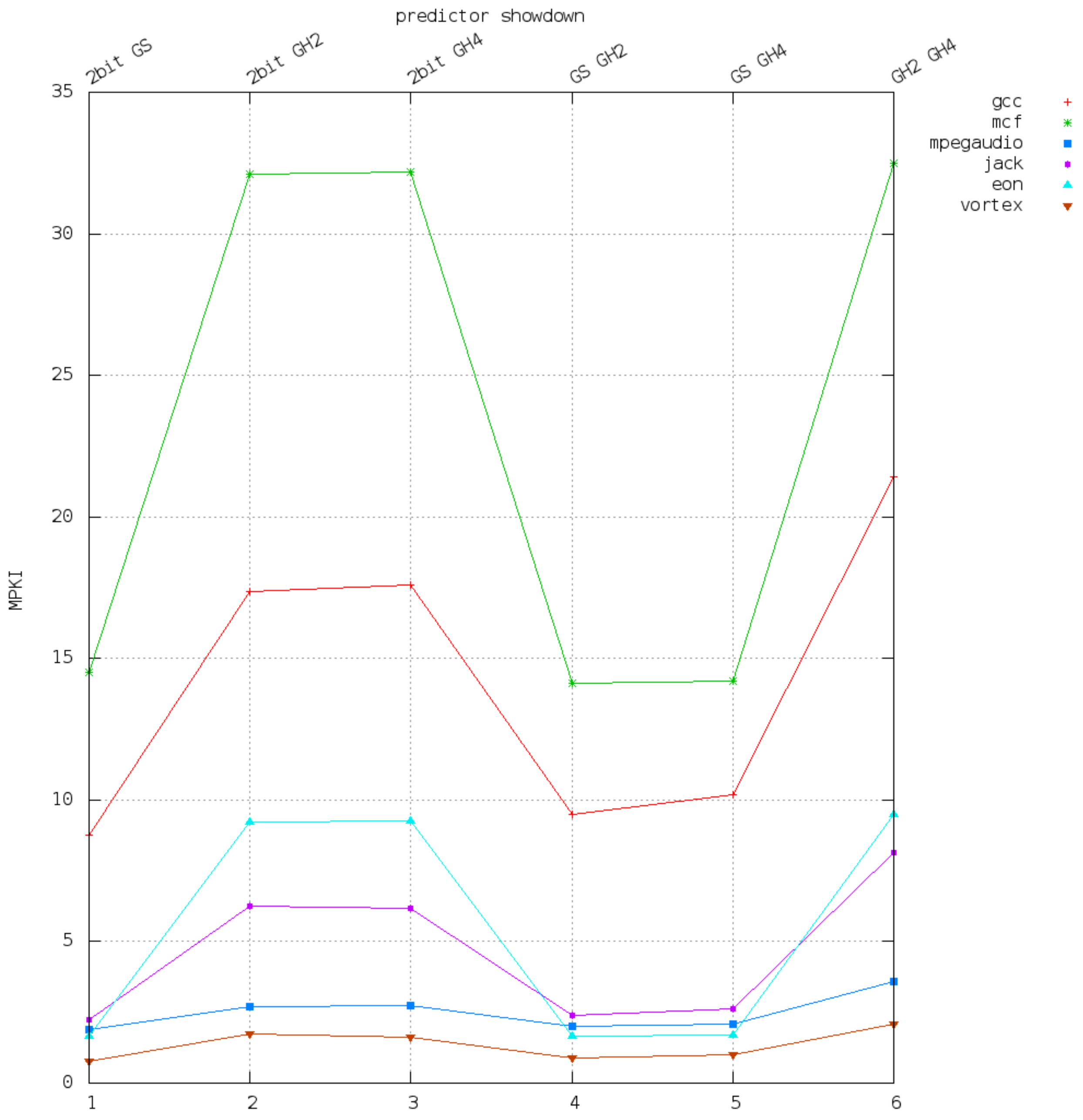
C1. Σύγκριση διαφορετικών predictors



Σχήμα 5: Σύγκριση predictors

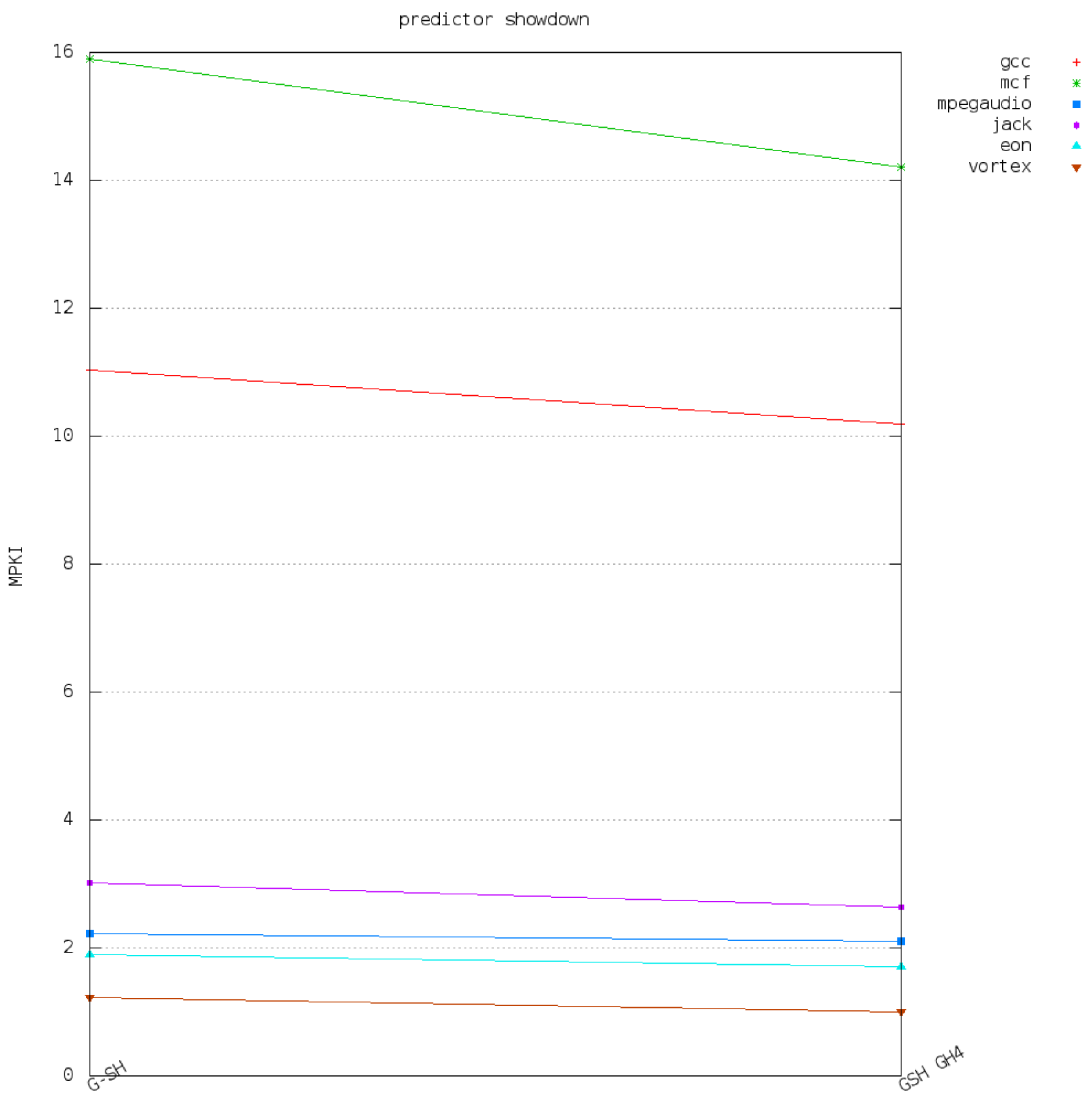
Από αυτούς καταλήγουμε πως καλύτερος είναι ο gshare predictor.

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 ελαρφώς καλύτερος από τον gshare predictor.

Ο πηγαίος κώδικας που χρησιμοποιήσαμε ήταν ο ακόλουθος:

```
// predict.cpp
// This file contains the main function.  The program accepts a single
// parameter: the name of a trace file.  It drives the branch predictor
// simulation by reading the trace file and feeding the traces one at a time
// to the branch predictor.

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>


#include <algorithm>

#include "branch.h"
```

```
#include "trace.h"
#include "predictor.h"
#include "btb.h"
#include "nt_predictor.h"
#include "btfnt_predictor.h"
#include "nbit_predictor.h"                //the .h files of the branch predictors' implementations
#include "gshare_predictor.h"
#include "localhistory_predictor.h"
#include "globalhistory_predictor.h"
#include "hybrid_predictor.h"


#define NR_PREDICTORS 11

using namespace std;

int main (int argc, char *argv[])
{

    // make sure there is one parameter

    if (argc != 2) {
        fprintf (stderr, "Usage: %s <filename>.gz\n", argv[0]);
        exit (1);
    }

    // open the trace file for reading

    init_trace (argv[1]);

    // initialize competitor's branch prediction code

    // you can use more than one predictor in an array of predictors!!!

    branch_predictor **p = new branch_predictor*[NR_PREDICTORS];

    p[0] = new nt_predictor();
    p[1] = new btfnt_predictor();
    p[2] = new nbit_predictor(4);
    p[3] = new gshare_predictor();
    p[4] = new localhistory_predictor(2048, 8);
    p[5] = new localhistory_predictor(4096, 4);
        /* X=2 BHR=4 */
    p[6] = new globalhistory_predictor(16384, 2, 4);
        /* X=2 BHR=8 */
    p[7] = new globalhistory_predictor(16384, 2, 8);
        /* X=4 BHR=4 */
    p[8] = new globalhistory_predictor(8192, 4, 4);
        /* X=4 BHR=8 */
    p[9] = new globalhistory_predictor(8192, 4, 8);

    p[10] = new hybrid_predictor(512);

    long long int
    tmiss[NR_PREDICTORS],        // number of target mispredictions
          dmiss[NR_PREDICTORS];          // number of direction mispredictions

    fill( tmiss, tmiss+NR_PREDICTORS, 0);
    fill( dmiss, dmiss+NR_PREDICTORS, 0);
    // keep looping until end of file

    for (;;) {

        // get a trace

        trace *t = read_trace ();

        // NULL means end of file

        if (!t) break;

        // send this trace to the competitor's code for prediction

        branch_update *u;

```

```
                /* static not taken */
        u = p[0]->predict(t->bi);
        p[0]->update(u, t->taken, t->target);
        dmiss[0] += u->direction_prediction() != t->taken;
        tmiss[0] += u->target_prediction() != t->target;


                /* backward taken forward not taken */
        ((btfnt_predictor *)p[1])->set_target(t->target > t->bi.address);
        u = p[1]->predict(t->bi);
        p[1]->update(u, t->taken, t->target);
        dmiss[1] += u->direction_prediction() != t->taken;
        tmiss[1] += u->target_prediction() != t->target;


                /* 4 bit predictor */
        u = p[2]->predict(t->bi);
        p[2]->update(u, t->taken, t->target);
        dmiss[2] += u->direction_prediction() != t->taken;
        tmiss[2] += u->target_prediction() != t->target;

                /* gshare predictor */
        u = p[3]->predict(t->bi);
        p[3]->update(u, t->taken, t->target);
        dmiss[3] += u->direction_prediction() != t->taken;
        tmiss[3] += u->target_prediction() != t->target;

                /* local history 2 level predictor X=2048 */
        u = p[4]->predict(t->bi);
        p[4]->update(u, t->taken, t->target);
        dmiss[4] += u->direction_prediction() != t->taken;
        tmiss[4] += u->target_prediction() != t->target;

                /* local history 2 level predictor X=4096 */
        u = p[5]->predict(t->bi);
        p[5]->update(u, t->taken, t->target);
        dmiss[5] += u->direction_prediction() != t->taken;
        tmiss[5] += u->target_prediction() != t->target;

                /* global history X=2 BHR=4 */
        u = p[6]->predict(t->bi);
        p[6]->update(u, t->taken, t->target);
        dmiss[6] += u->direction_prediction() != t->taken;
        tmiss[6] += u->target_prediction() != t->target;

                /* global history X=2 BHR=8 */
        u = p[7]->predict(t->bi);
        p[7]->update(u, t->taken, t->target);
        dmiss[7] += u->direction_prediction() != t->taken;
        tmiss[7] += u->target_prediction() != t->target;

                /* global history X=4 BHR=4 */
        u = p[8]->predict(t->bi);
        p[8]->update(u, t->taken, t->target);
        dmiss[8] += u->direction_prediction() != t->taken;
        tmiss[8] += u->target_prediction() != t->target;

                /* global history X=4 BHR=8 */
        u = p[9]->predict(t->bi);
        p[9]->update(u, t->taken, t->target);
        dmiss[9] += u->direction_prediction() != t->taken;
        tmiss[9] += u->target_prediction() != t->target;

        u = p[10]->predict(t->bi);
        p[10]->update(u, t->taken, t->target);
        dmiss[10] += u->direction_prediction() != t->taken;
        tmiss[10] += u->target_prediction() != t->target;
    }

    // done reading traces

    end_trace ();

    // give final mispredictions per kilo-instruction and exit.
    // each trace represents exactly 100 million instructions.

    for(int i = 0; i < NR_PREDICTORS; i++) {
        printf("%d\t%0.3f\n",i+1,1000.0 * (dmiss[i]/1e8));
        delete p[i];
    }
    delete [] p;

    exit (0);
}
```

Static Not-Taken

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : nt_predictor.h
* Creation Date : 19-05-2012
* Last Modified : Sat 19 May 2012 10:09:02 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_-._-._-._-._-._-._-._-._-._-._-._-._-._-._-._-.*/

// nt_predictor.h

#ifndef NT_PREDICTOR_H
#define NT_PREDICTOR_H
#include "predictor.h"


class nt_update : public branch_update
{
public:
    unsigned int index;
};
class nt_predictor: public branch_predictor
{
    nt_update u;
    branch_update *predict (branch_info & b);
    void update (branch_update *u, bool taken, unsigned int target);
};

#endif
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : nt_predictor.cpp
 * Creation Date : 19-05-2012
 * Last Modified : Sat 19 May 2012 10:07:46 PM EEST
 * Created By : Greg Liras <gregliras@gmail.com>
 _-._-._-._-._-._-._-._-._-._-._-._-._-._-._-._-.*/

#include "nt_predictor.h"

branch_update *nt_predictor::predict (branch_info & b)
{
    if (b.br_flags & BR_CONDITIONAL) {
        u.direction_prediction (false);
    } else {
        u.direction_prediction (true);
    }
    u.target_prediction (0);
    return &u;
}
void nt_predictor::update (branch_update *u, bool taken, unsigned int target)
{
}
```

Static Backward Taken Forward Not Taken

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : btfnt_predictor.h
 * Creation Date : 19-05-2012
 * Last Modified : Sat 26 May 2012 06:02:21 PM EEST
 * Created By : Greg Liras <gregliras@gmail.com>
 _-._-._-._-._-._-._-._-._-._-._-._-._-._-._-._-.*/
// btfnt_predictor.h
```

```

#ifndef BTFNT_PREDICTOR_H
#define BTFNT_PREDICTOR_H

#include <math.h>
#include "predictor.h"

class btfnt_predictor : public branch_predictor
{
public:

    branch_update u;
    branch_info bi;
    bool jump;

        void set_target(bool t);
    branch_update *predict (branch_info & b);
    void update (branch_update *u, bool taken, unsigned int target);
};

#endif
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : btfnt_predictor.cpp
 * Creation Date : 19-05-2012
 * Last Modified : Sat 26 May 2012 06:03:49 PM EEST
 * Created By : Greg Liras <gregliras@gmail.com>
 _._._._._._._._._._._._._._._._._._._._._._._.*/
#include "btfnt_predictor.h"

void btfnt_predictor::set_target (bool t) {
        jump = t;
}
branch_update *btfnt_predictor::predict (branch_info & b)
{
    bi = b;
    if (b.br_flags & BR_CONDITIONAL) {
        if (jump) {
                        u.direction_prediction (false);
                }
        else {
                        u.direction_prediction(true);
                }

    } else {
        u.direction_prediction (true);
    }
    u.target_prediction (0);
    return &u;
}
void btfnt_predictor::update (branch_update *u, bool taken, unsigned int target)
{

}
```

4-bit predictor

```
// nbit_predictor.h
//
//

#ifndef N_BIT_PREDICTOR_H
#define N_BIT_PREDICTOR_H


#include <math.h>
#include <string.h>
#include "predictor.h"

class nbit_update : public branch_update
{
public:
    unsigned int index;
};

class nbit_predictor : public branch_predictor
{
public:
#define NBP_TABLE_BITS        15  //number of entries = 2^15
    nbit_update u;
    branch_info bi;
    int counter_limit;
    int N_COUNTER_LENGTH;

    unsigned char tab[1<<NBP_TABLE_BITS];

    nbit_predictor (int length);
    branch_update *predict (branch_info & b);
    void update (branch_update *u, bool taken, unsigned int target);
};

#endif
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
 * File Name : nbit_predictor.cpp
 * Creation Date : 15-05-2012
 * Last Modified : Tue 15 May 2012 05:25:25 PM EEST
 * Created By : Greg Liras <gregliras@gmail.com>
 _._._._._._._._._._._._._._._._._._._._._._.*/
#include "nbit_predictor.h"

nbit_predictor::nbit_predictor (int length) :N_COUNTER_LENGTH(length)
{
    memset (tab, 0, sizeof (tab));
    counter_limit = ((int) pow(2.0, N_COUNTER_LENGTH)) - 1;
}

branch_update *nbit_predictor::predict (branch_info & b)
{
    bi = b;
    if (b.br_flags & BR_CONDITIONAL) {
        u.index =  (b.address & ((1<<NBP_TABLE_BITS)-1));
        u.direction_prediction (tab[u.index] >> (N_COUNTER_LENGTH-1));
    } else {
        u.direction_prediction (true);
    }
    u.target_prediction (0);
    return &u;
}

void nbit_predictor::update (branch_update *u, bool taken, unsigned int target)
{
    if (bi.br_flags & BR_CONDITIONAL) {
        unsigned char *c = &tab[((nbit_update*)u)->index];
        if (taken) {
            if (*c < counter_limit) (*c)++;
        } else {
            if (*c > 0) (*c)--;
        }
    }
}
```

gshare predictor

```
// gshare_predictor.h
// This file contains a sample my_predictor class.
// It is a simple 32,768-entry gshare with a history length of 15.

#ifndef GSHARE_PREDICTOR_H
#define GSHARE_PREDICTOR_H

#include "predictor.h"

class gshare_update : public branch_update
{
public:
    unsigned int index;
};

/*
 * H klash gshare_predictor klhronomei thn klash
 * branch_predictor kai kanei override tis me8odous
 * predict kai update
 */

class gshare_predictor : public branch_predictor
{
public:
#define HISTORY_LENGTH        14
#define GSP_TABLE_BITS        15
    gshare_update u;
    branch_info bi;
    unsigned int history;
    unsigned char tab[1<<GSP_TABLE_BITS];

    gshare_predictor (void);
    branch_update *predict (branch_info & b);

    void update (branch_update *u, bool taken, unsigned int target);
};
#endif
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : gshare_predictor.cpp
* Creation Date : 20-05-2012
* Last Modified : Sat 26 May 2012 06:16:06 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._.*/

#include "gshare_predictor.h"
#include <algorithm>
gshare_predictor::gshare_predictor (void) : history(0)
{
    std::fill (tab, tab+sizeof (tab),0);
}
branch_update *gshare_predictor::predict (branch_info & b)
{
    bi = b;

    /*
     * O gshare xrhsimopoieitai mono gia conditional branches.
     * Ta uncoditional ginontai predicted panta TAKEN
     */

    if (b.br_flags & BR_CONDITIONAL) {
        u.index = (history << (GSP_TABLE_BITS - HISTORY_LENGTH)) ^ (b.address & ((1<<GSP_TABLE_BITS)-1));
        u.direction_prediction (tab[u.index] >> 1);
    } else {
        u.direction_prediction (true);
    }

    // O gshare den kanei target prediction, gia auto to 8etoume sto 0.

    u.target_prediction (0);
    return &u;
}
void gshare_predictor::update (branch_update *u, bool taken, unsigned int target)
{
    //O gshare xrhsimopoieitai mono gia conditional branches
    if (bi.br_flags & BR_CONDITIONAL) {
        unsigned char *c = &tab[((gshare_update*)u)->index];
        if (taken) {
            if (*c < 3) (*c)++;
        } else {
            if (*c > 0) (*c)--;
        }
        history <<= 1;
        history |= taken;
        history &= (1<<HISTORY_LENGTH)-1;
    }
}
```

Local-History two-level predictors

```
#ifndef localhistory_predictor_H
#define localhistory_predictor_H

#include <cmath>
#include "predictor.h"

class localhistory_update : public branch_update
{
public:
    unsigned int phtindex;
    unsigned int bhtindex;
};

class localhistory_predictor : public branch_predictor
{
public:

    localhistory_update u;
    branch_info bi;

        int p_counter_limit;
        int b_counter_limit;

        int pht_entries;
        int pht_nbit;

        int bht_entries;
        int bht_entry_length;

    unsigned char *pht;
    unsigned char *bht;

        unsigned int pht_mask;


    localhistory_predictor (int x,int z);
    branch_update *predict (branch_info & b);

    void update (branch_update *u, bool taken, unsigned int target);
};
#endif
```

```
#include "localhistory_predictor.h"
#include <algorithm>
#include <cstring>

localhistory_predictor::localhistory_predictor (int x , int z) : bht_entries(x), bht_entry_length(z)
{
        pht_entries=8192;
        pht_nbit=2;

        pht = new unsigned char [pht_entries];
        memset (pht, 0, sizeof (pht));

        bht = new unsigned char [x];
        memset (bht, 0, sizeof (bht));

        pht_mask = ((1<<(((int) log2(pht_entries))-bht_entry_length))-1);

        p_counter_limit = (1<<pht_nbit);
        b_counter_limit = (1<<bht_entry_length);
}

branch_update *localhistory_predictor::predict (branch_info & b)
{
        bi=b;
        if (b.br_flags & BR_CONDITIONAL) {

                u.bhtindex = (b.address & (bht_entries-1));
                u.phtindex = ((b.address & pht_mask)<<bht_entry_length);
                u.phtindex |= bht[u.bhtindex];

                u.direction_prediction(pht[u.phtindex]>>(pht_nbit-1));
        } else {
                u.direction_prediction(true);
        }
        u.target_prediction (0);
        return &u;
}

void localhistory_predictor::update (branch_update *u, bool taken, unsigned int target)
{
    if (bi.br_flags & BR_CONDITIONAL) {

                unsigned char *c = &pht[((localhistory_update*)u)->phtindex];
                unsigned char *d = &bht[((localhistory_update*)u)->bhtindex];

                if (taken) {
                        if (*c < p_counter_limit)
                                (*c)++;
                }
                else {
                        if (*c > 0)
                                (*c)--;
                }
                (*d) <<= 1;
                (*d) |= taken;
                (*d) &= b_counter_limit;


        }
}
```

Global-History two-level predictors

```
#ifndef GLOBALHISTORY_PREDICTOR_H
#define GLOBALHISTORY_PREDICTOR_H

#include <cmath>
#include <cstring>
#include <cstdio>
#include "predictor.h"

class globalhistory_update : public branch_update
{
public:
    unsigned int index;
};

class globalhistory_predictor : public branch_predictor
{
public:

    globalhistory_update u;
    branch_info bi;

        int p_counter_limit;
        int b_counter_limit;
        unsigned int pht_mask;

        int pht_entries;
        int pht_nbit;

        int bhr;
        int bhr_length;

    unsigned char **pht;

    globalhistory_predictor (int x,int y, int z);
    branch_update *predict (branch_info & b);

    void update (branch_update *u, bool taken, unsigned int target);
};
#endif
```

```
#include "globalhistory_predictor.h"
#include <algorithm>
```

```

globalhistory_predictor::globalhistory_predictor (int x , int y, int z) :pht_entries(x), pht_nbit(y), bhr_length(z)
{

        bhr=0;

        pht = new unsigned char *[1<<bhr_length];

        for( int i = 0; i < (1<<bhr_length); i++)
        {
                pht[i] = new unsigned char [pht_entries>>bhr_length];
        }

        for (int i = 0; i < (1<<bhr_length); i++)
        {
                memset(pht[i], 0, sizeof (pht[i]));
        }

        p_counter_limit = (1<<pht_nbit);
        b_counter_limit = (1<<bhr_length);
        pht_mask = ((pht_entries>>bhr_length)-1);
}

branch_update *globalhistory_predictor::predict (branch_info & b)
{
        bi=b;
        if (b.br_flags & BR_CONDITIONAL) {

                u.index = (b.address & pht_mask);

                u.direction_prediction(pht[bhr][u.index]>>(pht_nbit-1));
        } else {
                u.direction_prediction (true);
        }
        u.target_prediction (0);
        return &u;
}

void globalhistory_predictor::update (branch_update *u, bool taken, unsigned int target)
{
    if (bi.br_flags & BR_CONDITIONAL) {

                unsigned char *c = &pht[bhr][((globalhistory_update*)u)->index];

                if (taken) {
                        if (*c < p_counter_limit)
                                (*c)++;
                }
                else {
                        if (*c > 0)
                                (*c)--;
                }
                bhr <<= 1;
                bhr |= taken;
                bhr &= b_counter_limit;

        }
}
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_predictor.h
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 11:04:25 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_-._-._-._-._-._-._-._-._-._-._-._-._-._-._-._-.*/

#ifndef HYBRID_PREDICTOR_H
#define HYBRID_PREDICTOR_H

#include "predictor.h"
#include "globalhistory_predictor.h"

class hybrid_update : public branch_update
{
public:
        branch_update *ups[2];
        unsigned int index;
        unsigned int pred;

};
class hybrid_predictor : public branch_predictor
{
public:

        hybrid_update u;
        branch_info bi;
        int counter_limit;

        branch_predictor **preds;


        int pht_entries;
        int pht_bits_length;

        unsigned char *tab;

        hybrid_predictor(int entries);
    branch_update *predict (branch_info & b);
    void update (branch_update *u, bool taken, unsigned int target);




};


#endif
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_predictor.cpp
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 11:09:18 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.-._.*/

#include "hybrid_predictor.h"

hybrid_predictor::hybrid_predictor(int entries) : pht_entries(entries)
{
        tab = new unsigned char[pht_entries];
        memset(tab,0,sizeof(tab));
        pht_bits_length = 2;
        counter_limit = 2;
        u.pred = 0;
        u.index = 0;
        preds = new branch_predictor*[2];
}


branch_update *hybrid_predictor::predict (branch_info & b)
{
    bi = b;
    if (b.br_flags & BR_CONDITIONAL) {
        u.index =  (b.address & (pht_entries-1));
        u.pred = tab[u.index] >> (pht_bits_length - 1);
    }
        u.ups[0] = (branch_update *) preds[0]->predict(b);
        u.ups[1] = (branch_update *) preds[1]->predict(b);
        unsigned int thepred = u.pred & 1;
        u.direction_prediction(u.ups[thepred]->direction_prediction());
        return &u;
}
void hybrid_predictor::update (branch_update *u, bool taken, unsigned int target)
{
    if (bi.br_flags & BR_CONDITIONAL) {
        unsigned char *c = &tab[((hybrid_update*)u)->index];
                bool pred0_result = ((hybrid_update *)u)->ups[0]->direction_prediction() == taken;
                bool pred1_result = ((hybrid_update *)u)->ups[1]->direction_prediction() == taken;
                int meta_update = pred1_result - pred0_result;

```

```
                    if (meta_update > 0 && *c < 3)
                                    (*c)++;
                    else if (meta_update < 0 && *c > 0)
                                    (*c)--;
                    (*c) &= 3;
                    preds[0]->update(((hybrid_update *)u)->ups[0], taken, target);
                    preds[1]->update(((hybrid_update *)u)->ups[1], taken, target);

        }
}
```

2-bit global history BHT=2

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_2bit_GH2_predictor.cpp
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 10:38:21 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._._._.*/




#include "hybrid_2bit_GH2_predictor.h"


hybrid_2bit_GH2_predictor::hybrid_2bit_GH2_predictor(int entries): hybrid_predictor(entries)
{
        preds[0] = new nbit_predictor(2);
        preds[1] = new globalhistory_predictor(8192,2,2);
}
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_2bit_GH2_predictor.h
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 10:45:58 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._._._.*/




#ifndef HYBRID_2BIT_GH2_PREDICTOR_H
#define HYBRID_2BIT_GH2_PREDICTOR_H

#include "predictor.h"
#include "gshare_predictor.h"
#include "nbit_predictor.h"
#include "hybrid_predictor.h"

class hybrid_2bit_GH2_predictor : public hybrid_predictor
{
public:

        hybrid_2bit_GH2_predictor(int entries);


};


#endif
```

2-bit global history BHT=4

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_2bit_GH4_predictor.cpp
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 10:40:56 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._._._.*/



```

```

#include "hybrid_2bit_GH4_predictor.h"


hybrid_2bit_GH4_predictor::hybrid_2bit_GH4_predictor(int entries): hybrid_predictor(entries)
{
        preds[0] = new nbit_predictor(2);
        preds[1] = new globalhistory_predictor(8192,2,4);
}
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_2bit_GH4_predictor.h
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 10:46:01 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._.*/




#ifndef HYBRID_2BIT_GH4_PREDICTOR_H
#define HYBRID_2BIT_GH4_PREDICTOR_H

#include "predictor.h"
#include "gshare_predictor.h"
#include "nbit_predictor.h"
#include "hybrid_predictor.h"

class hybrid_2bit_GH4_predictor : public hybrid_predictor
{
public:

        hybrid_2bit_GH4_predictor(int entries);


};


#endif
```

2-bit gshare

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_2bit_GS_predictor.cpp
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 10:34:58 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._.*/

#include "hybrid_2bit_GS_predictor.h"


hybrid_2bit_GS_predictor::hybrid_2bit_GS_predictor(int entries): hybrid_predictor(entries)
{
        preds[0] = new nbit_predictor(2);
        preds[1] = new gshare_predictor();
}
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_2bit_GS_predictor.h
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 10:46:04 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._.*/



#ifndef HYBRID_2BIT_GS_PREDICTOR_H
#define HYBRID_2BIT_GS_PREDICTOR_H

#include "predictor.h"
#include "gshare_predictor.h"
#include "nbit_predictor.h"
#include "hybrid_predictor.h"
```

```

class hybrid_2bit_GS_predictor : public hybrid_predictor
{
public:

        hybrid_2bit_GS_predictor(int entries);




};


#endif
```

global history BHT=2,4

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_GH2_GH4_predictor.cpp
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 11:16:20 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._._.*/




#include "hybrid_GH2_GH4_predictor.h"


hybrid_GH2_GH4_predictor::hybrid_GH2_GH4_predictor(int entries): hybrid_predictor(entries)
{
        preds[0] = new globalhistory_predictor(8192,2,2);
        preds[1] = new globalhistory_predictor(8192,2,4);
}
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_GH2_GH4_predictor.h
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 11:16:25 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._._.*/




#ifndef HYBRID_GH2_GH4_PREDICTOR_H
#define HYBRID_GH2_GH4_PREDICTOR_H

#include "predictor.h"
#include "gshare_predictor.h"
#include "nbit_predictor.h"
#include "hybrid_predictor.h"

class hybrid_GH2_GH4_predictor : public hybrid_predictor
{
public:

        hybrid_GH2_GH4_predictor(int entries);


};


#endif
```

gshare global history BHT=2

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_GS_GH2_predictor.cpp
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 11:12:22 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
```

```
_._._._._._._._._._._._._._._._._._._._._._.*/



#include "hybrid_GS_GH2_predictor.h"


hybrid_GS_GH2_predictor::hybrid_GS_GH2_predictor(int entries): hybrid_predictor(entries)
{
        preds[0] = new gshare_predictor();
        preds[1] = new globalhistory_predictor(8192,2,2);
}
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_GS_GH2_predictor.h
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 11:12:16 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._.*/



#ifndef HYBRID_GS_GH2_PREDICTOR_H
#define HYBRID_GS_GH2_PREDICTOR_H

#include "predictor.h"
#include "gshare_predictor.h"
#include "nbit_predictor.h"
#include "hybrid_predictor.h"

class hybrid_GS_GH2_predictor : public hybrid_predictor
{
public:

        hybrid_GS_GH2_predictor(int entries);


};


#endif
```

gshare global history BHT=4

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_GS_GH4_predictor.cpp
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 11:14:51 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._.*/



#include "hybrid_GS_GH4_predictor.h"


hybrid_GS_GH4_predictor::hybrid_GS_GH4_predictor(int entries): hybrid_predictor(entries)
{
        preds[0] = new gshare_predictor();
        preds[1] = new globalhistory_predictor(8192,2,4);
}
```

```
/* -.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-.
* File Name : hybrid_GS_GH4_predictor.h
* Creation Date : 27-05-2012
* Last Modified : Sun 27 May 2012 11:15:00 PM EEST
* Created By : Greg Liras <gregliras@gmail.com>
_._._._._._._._._._._._._._._._._._._._._._.*/


```

```
#ifndef HYBRID_GS_GH4_PREDICTOR_H
#define HYBRID_GS_GH4_PREDICTOR_H

#include "predictor.h"
#include "gshare_predictor.h"
#include "nbit_predictor.h"
#include "hybrid_predictor.h"

class hybrid_GS_GH4_predictor : public hybrid_predictor
{
public:

        hybrid_GS_GH4_predictor(int entries);


};


#endif
```