UNIVERSITAT ROVIRA I VIRGILI

Simulació Processador Superescalar

Predictor de Salt

Cristòfol Daudén Esmel

Aleix Mariné Tena

Fase 1:

Sobre la relació entre IPC i el % d'encert dels predictors:

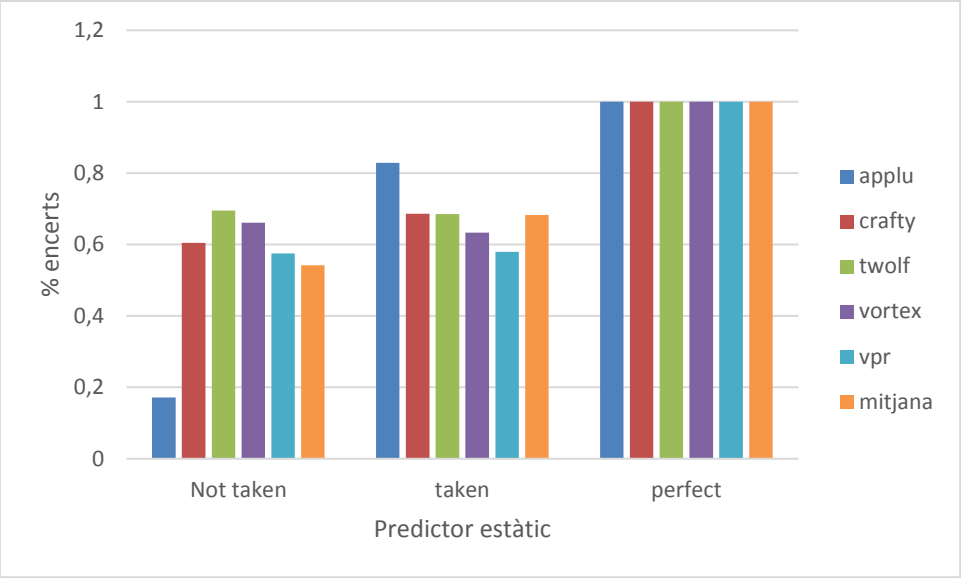
Quan una instrucció de salt es descodificada s'utilitzen mètodes per a predir la direcció i el resultat del salt. D'aquesta manera, es pot fer *prefetching* de les instruccions situades a partir de la direcció de memòria predita com a destí del salt abans que el salt sigui resolt.
A aquest fenomen se'l anomena **execució especulativa** ja que s'executen instruccions sense saber si son realment les que s'han d'executar.

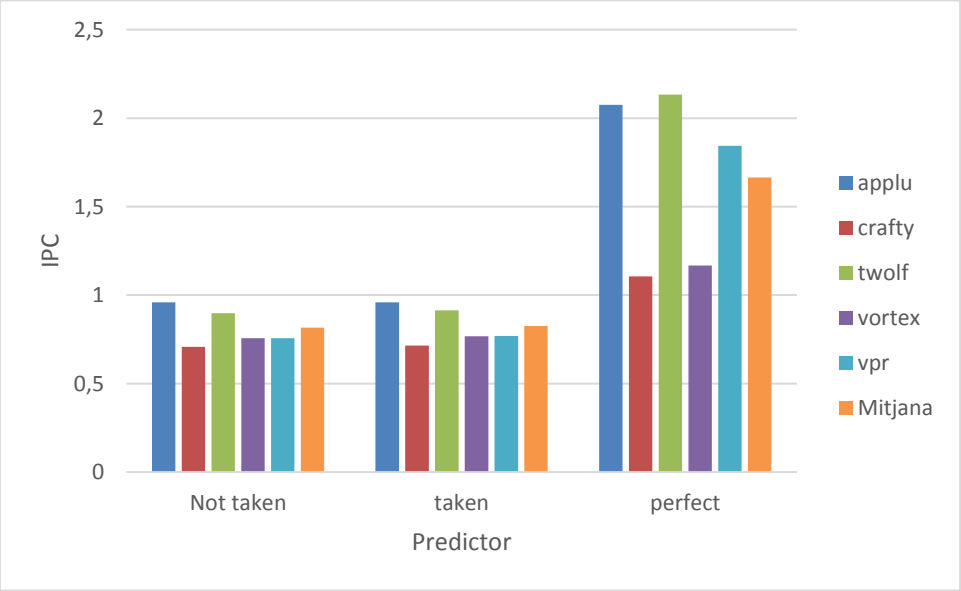
Tot i així els predictors de salt no tenen una fiabilitat del 100%, pel que de vegades cometen errors i prediuen el resultat o direcció d'un salt de manera incorrecta, fent *prefetching* d'instruccions equivocades. En quant el processador rep en la fase d'execució la instrucció de salt i detecta que s'ha executat especulativament les instruccions de la branca equivocada; es provoca el buidat del *pipeline* i es replena amb les instruccions procedents de la branca correcta. Aquest buidat i replenat del *pipeline* dura molts cicles i causa una baixada de rendiment.

Efectivament, un predictor de salts més efectiu provoca que hi hagi menys buidats i replenats del pipeline, per tant, fa augmentar el nombre d'instruccions per cicle, augmentant el rendiment.

Predictors estàtics:

		Not taken	taken	perfect
applu	IPC	0,9589	0,9595	2,0757
	% encerts	0,1714	0,8286	1
crafty	IPC	0,7071	0,7146	1,105
	% encerts	0.6051	0,6867	1
twolf	IPC	0,8983	0,914	2,1338
	% encerts	0,695	0,6853	1
vortex	IPC	0,7563	0,7675	1,1677
	% encerts	0,6615	0,6333	1
vpr	IPC	0,7564	0,7699	1,8432
	% encerts	0,5751	0,58	1
Mitjana	IPC	0,8154	0,8251	1,66508
	% encerts	0,4206	0,68278	1





Com ja sabem aquests predictors no recopilen ni utilitzen la informació de les instruccions de salts en temps d'execució, sinó que sempre es comporten igual front al mateix salt.

Els predictor "taken" i "not taken" sempre prediuen tots els salts com a presos o com a no presos respectivament. Per estadística podem veure que la majoria dels salts solen ser presos, pel que en la majoria dels programes el predictor "taken" sol tenir més taxa d'encerts que el "not taken".

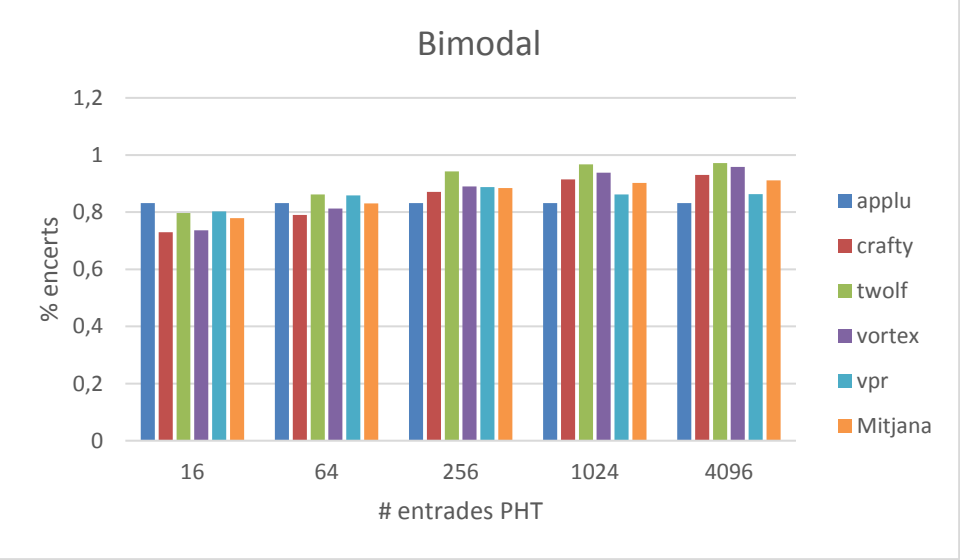
El predictor "perfect" en canvi, té una taxa d'encerts del 100 %, pel que, serà el predictor amb el que obtindrem un IPC més gran degut a que no tindrem mai cap penalització de buidat de pipeline degut a un branch miss.

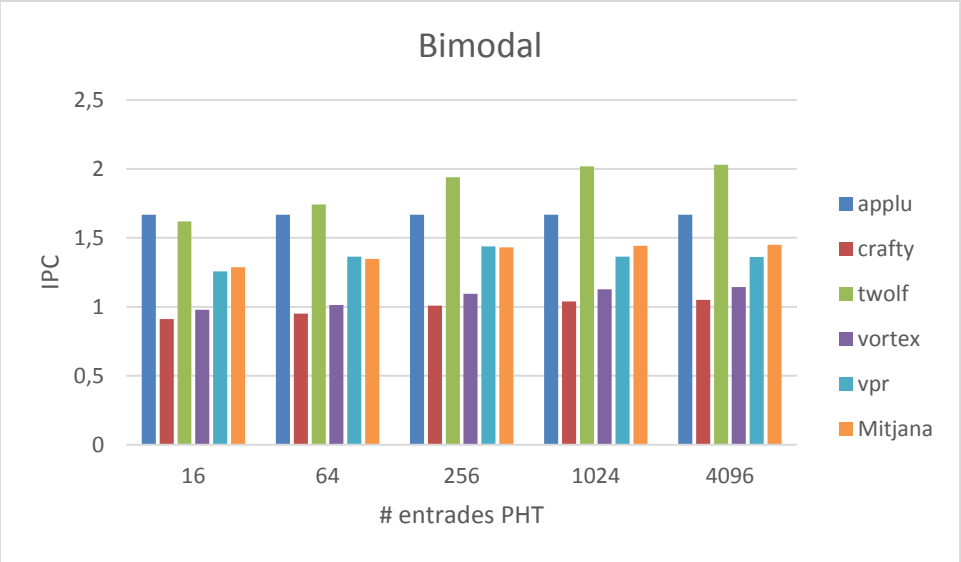
Podem observar que en els benchmarks *Applu*, *twolf* i *vpr* el IPC varia molt segons si utilitzem el predictor perfect o algun dels altres dos. Això ens indica que en aquests benchmarks hi ha un nombre d'execució de salts major que en els altres, pel que en utilitzar un predictor amb una taxa d'encerts baixa, el IPC es veu molt ressentit degut a la major quantitat de branch misses.

Predictors dinàmics

Bimodal:

		Bimodal (en funció de # d'entrades de la taula de predicció de 2-bits saturados PHT)				
		16	64	256	1024	4096
applu	IPC	1,6679	1,6679	1,6679	1,6679	1,6679
	% encerts	0,8315	0,8315	0,8315	0,8315	0,8315
crafty	IPC	0,9119	0,9515	1,0079	1,0397	1,051
	% encerts	0,7302	0,7899	0,8707	0,9147	0,9308
twolf	IPC	1,6188	1,7421	1,9403	2,0185	2,0291
	% encerts	0,7973	0,8616	0,9422	0,9672	0,9717
vortex	IPC	0,9785	1,013	1,0953	1,1272	1,1425
	% encerts	0,737	0,8125	0,8905	0,9379	0,9588
vpr	IPC	1,2566	1,3638	1,439	1,3631	1,3621
	% encerts	0,8021	0,8588	0,8876	0,8625	0,8626
Mitjana	IPC	1,28674	1,34766	1,43008	1,44328	1,45052
	% encerts	0,77962	0,83086	0,8845	0,90276	0,91108





En el predictor bimodal la PHT conté els *2-bit saturating counters* (2bsc) per a dur a terme la predicció del salt. Quan la PHT és petita (el nombre d'instruccions de salt en el codi és major que el nombre d'entrades) es produeixen col·lisions degut a que dues instruccions de salts diferents utilitzen el mateix comptador per a dur a terme la seva predicció de salt. Això resulta en una eficiència baixa ja que estem utilitzant el mateix comptador per a predir el resultat de més d'una instrucció de salt.

Aquests salts òbviament poden tenir comportaments diferents, pel que la predicció resulta incongruent.

A mesura que anem augmentant la PHT, disminueixen les col·lisions, donant lloc a que menys instruccions de salt utilitzin el mateix comptador per a dur a terme la seva predicció pel que augmenta la taxa d'encert en la predicció i disminueixen els misses. Això provoca una pujada en el IPC.

La situació ideal en aquest predictor es que cada instrucció de salt tingui la seva pròpia entrada en la PHT. Aconseguit això, la taxa d'encerts entraria en una fase estacionària respecte a la longitud de la PHT, ja que no podem tenir una millor predicció.

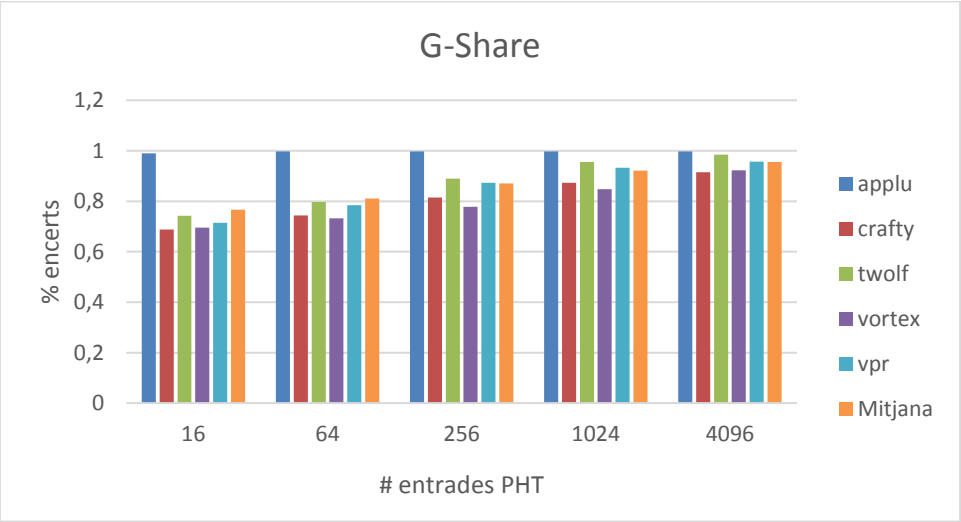
Podem observar aquest fenomen en el gràfic: En el benchmark *applu* la taxa d'encerts és manté gairebé constant respecte al nombre d'entrades de la PHT. Podem pensar doncs, que es tracta d'un benchmark amb poques instruccions de salt.

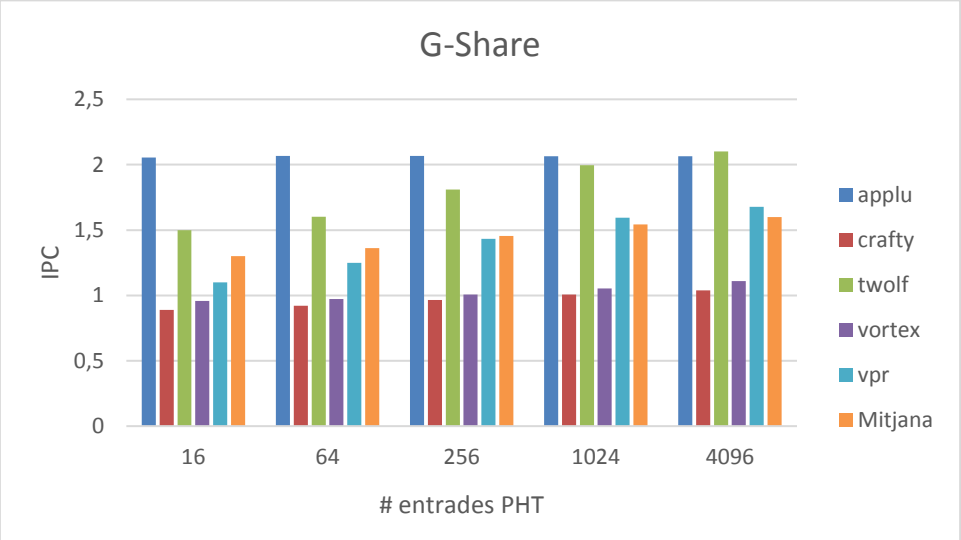
En la resta de benchmarks podem observar que els resultats de la taxa de predicció són més o menys els mateixos amb una PHT de 1024 o 4096. Podem deduir doncs que amb una PHT de 1024 entrades és suficient per a tenir una entrada per a cada instrucció i que no es produeixin col·lisions. Augmentar la mida fins a 4096 resulta en tenir la major part de la taula buida.

Podem seguir veient una correlació directa entre la taxa d'encerts i el IPC.

G-Share

		Gshare (en funció de # de entrades de la Taula de predicció de 2-bits saturats PHT)				
		16	64	256	1024	4096
applu	IPC	2,0549	2,0658	2,0658	2,0657	2,0657
	% encerts	0,9897	0,9971	0,9972	0,9972	0,9971
crafty	IPC	0,8881	0,9211	0,9655	1,0065	1,0388
	% encerts	0,688	0,744	0,8141	0,8734	0,9148
twolf	IPC	1,4999	1,6013	1,8089	1,9949	2,1007
	% encerts	0,7426	0,797	0,89	0,9557	0,9843
vortex	IPC	0,957	0,9727	1,0055	1,0522	1,1093
	% encerts	0,6959	0,7328	0,7782	0,8472	0,9227
vpr	IPC	1,1001	1,249	1,4337	1,5935	1,6764
	% encerts	0,7139	0,7845	0,8725	0,9322	0,9565
Mitjana	IPC	1,3	1,36198	1,45588	1,54256	1,59818
	% encerts	0,76602	0,81108	0,8704	0,92114	0,95508



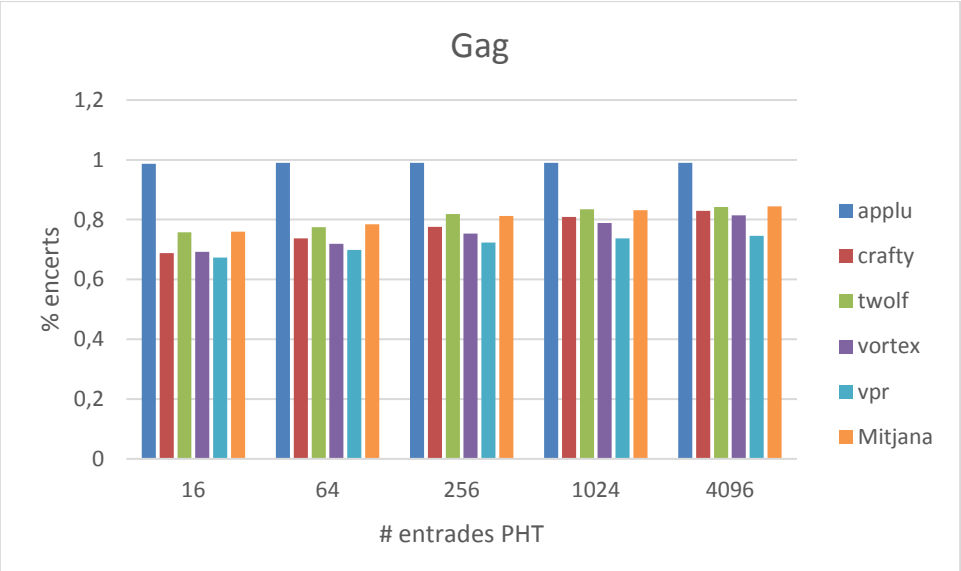


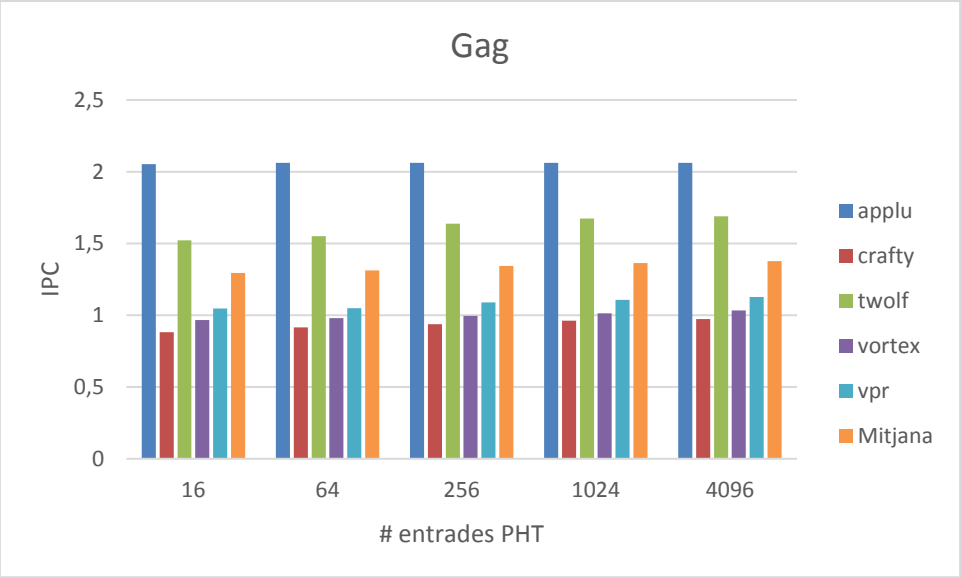
En el predictor G-Share s'utilitza una porta XOR entre el valor de PC quan apunta a la instrucció de salt a predir i el valor de BHR per a generar l'índex per a accedir a la taula del PHT.

Com que ara també utilitzem els bits del BHR, una sola instrucció pot accedir a diferents entrades de la PHT segons l'historial de salts. Això millora l'eficiència de predicció front a patrons repetitius de salt, però també augmenta el nombre de col·lisions inclús amb un nombre d'entrades del PHT major que el nombre d'instruccions de salt. Degut a això, i comparant amb el predictor bimodal podem veure que inclús en passar de 1024 a 4096 entrades la taxa d'encerts segueix augmentant de manera substancial. Per tant el predictor G-Share té una taxa d'encerts molt gran, però que podria augmentar encara més si disposéssim de més entrades encara.

Gag (G-Select)

		Gag (en funció de # de entradas de la tabla de predicció de 2-bits saturats PHT)				
		16	64	256	1024	4096
applu	IPC	2,0532	2,0618	2,0618	2,0618	2,0618
	% encerts	0,9869	0,9898	0,9898	0,9898	0,9898
crafty	IPC	0,8822	0,9146	0,937	0,9611	0,9742
	% encerts	0,6884	0,7369	0,7761	0,8087	0,829
twolf	IPC	1,5226	1,552	1,6383	1,6745	1,6884
	% encerts	0,7573	0,7746	0,8183	0,8352	0,8419
vortex	IPC	0,967	0,9797	0,996	1,0142	1,0331
	% encerts	0,692	0,7196	0,7536	0,7889	0,8149
vpr	IPC	1,0462	1,0485	1,09	1,108	1,1264
	% encerts	0,673	0,6988	0,7232	0,7373	0,7463
Mitjana	IPC	1,29424	1,31132	1,34462	1,36392	1,37678
	% encerts	0,75952	0,78394	0,8122	0,83198	0,84438





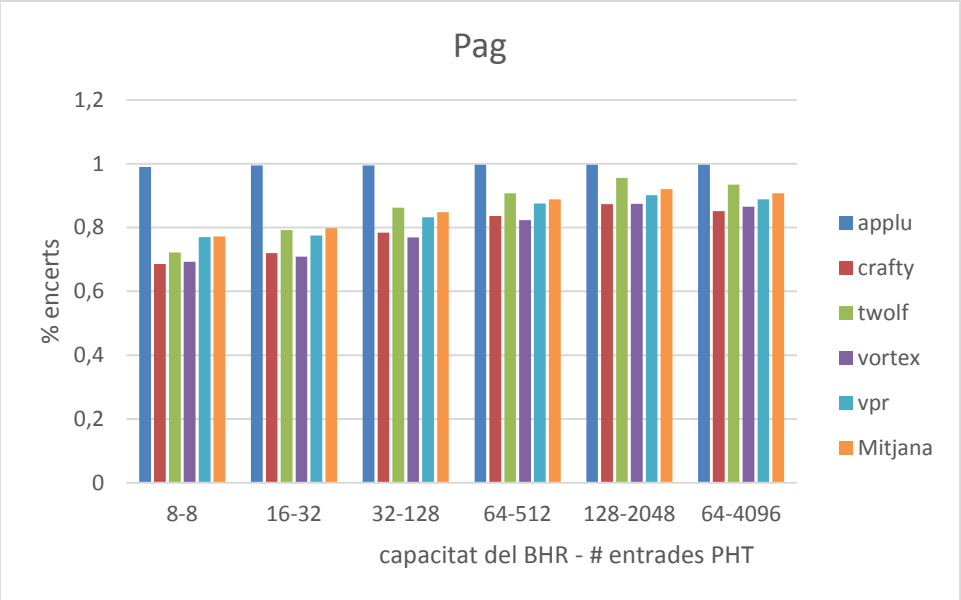
La única diferència entre aquest predictor i el G-Share és que aquí per a generar l'índex a la taula de PHT utilitzem la concatenació (i no la XOR) del BHR i el PC quan apunta a la instrucció de salt de la qual volem predir el salt. En aquest cas la no utilització de la porta XOR com a funció de suma fa que el nombre de col·lisions augmenti respecte al predictor G-Share. I es per això que la seva taxa d'encerts disminueix respecte a aquest.

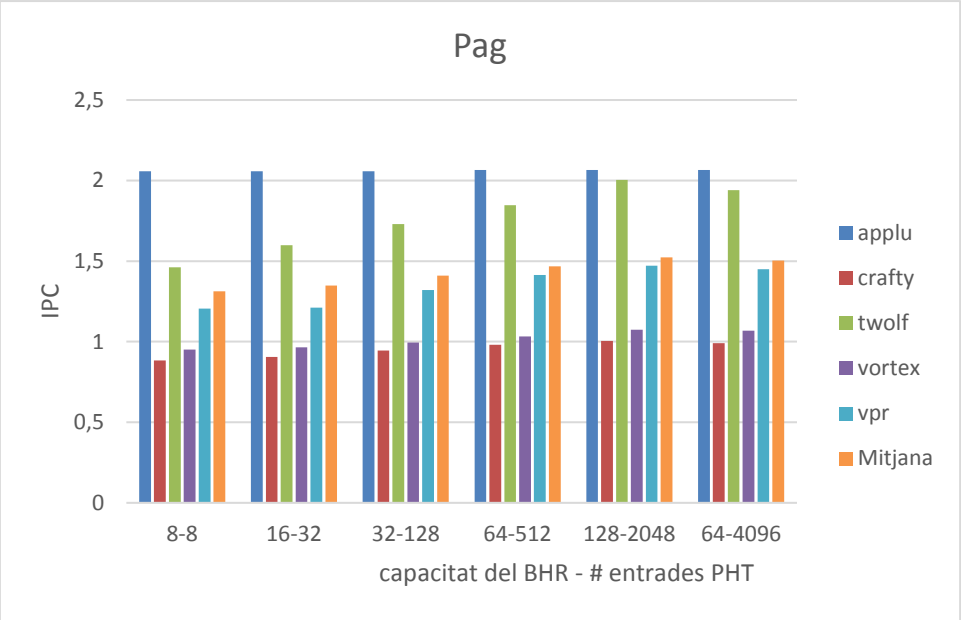
Trobem doncs, que un dels factors més limitant per a un predictors de salt per a obtenir bona taxa d'encerts és que el nombre de col·lisions sigui baix, de tal manera que la predicció de cada salt està més individualitzada i per tant és més acurada.

Sobre la relació entre la PHT i el % d'encerts podem observar que efectivament a mesura que augmentem la mida de la PHT també ho fa el % d'encerts. Tot i que aquest predictor fa prediccions pitjors podem observar igualment una certa tendència a estabilitzar-se per a mides de PHT grans.

Pag (P-select)

		Pag (en funció de # de BHR (BHT) i # d'entrades de la taula de predicciooó de 2-bits saturados PHT)					
		8-8	16-32	32-128	64-512	128-2048	64-4096
applu	IPC	2,0574	2,0576	2,0576	2,0659	2,0659	2,0659
	% encerts	0,9896	0,9943	0,9943	0,9972	0,9971	0,9971
crafty	IPC	0,8838	0,9061	0,9453	0,9804	1,0058	0,9908
	% encerts	0,6861	0,7203	0,7843	0,8366	0,8733	0,8515
twolf	IPC	1,4623	1,5987	1,7306	1,8468	2,0038	1,941
	% encerts	0,7221	0,7921	0,8622	0,9079	0,9553	0,9346
vortex	IPC	0,9513	0,9647	0,9941	1,032	1,075	1,068
	% encerts	0,6929	0,709	0,7688	0,8232	0,8748	0,865
vpr	IPC	1,2064	1,2106	1,3215	1,4148	1,471	1,45
	% encerts	0,77	0,7753	0,8324	0,8749	0,9011	0,888
Mitjana	IPC	1,31224	1,34754	1,40982	1,46798	1,5243	1,50314
	% encerts	0,77214	0,7982	0,8484	0,88796	0,92032	0,90724





El predictor “Pag” (P-Select) funciona de manera molt similar al predictor “G-Select”, però en aquest cas en lloc de tenir un BHR tenim un BHT, de tal manera que podem guardar les últimes n ocurrències d’un determinat nombre d’instruccions de salts. És a dir, utilitzem un BHR de manera local per a un nombre determinat d’instruccions, en lloc de un BHR global per a totes les instruccions.

Com més gran sigui el BHT menys instruccions hauran de compartir el mateix BHR per tant la precisió serà major ja que tindrem les últimes ocurrències de menor nombre de salts. La situació ideal, seria que hi hagués un BHR per cada instrucció de salt.

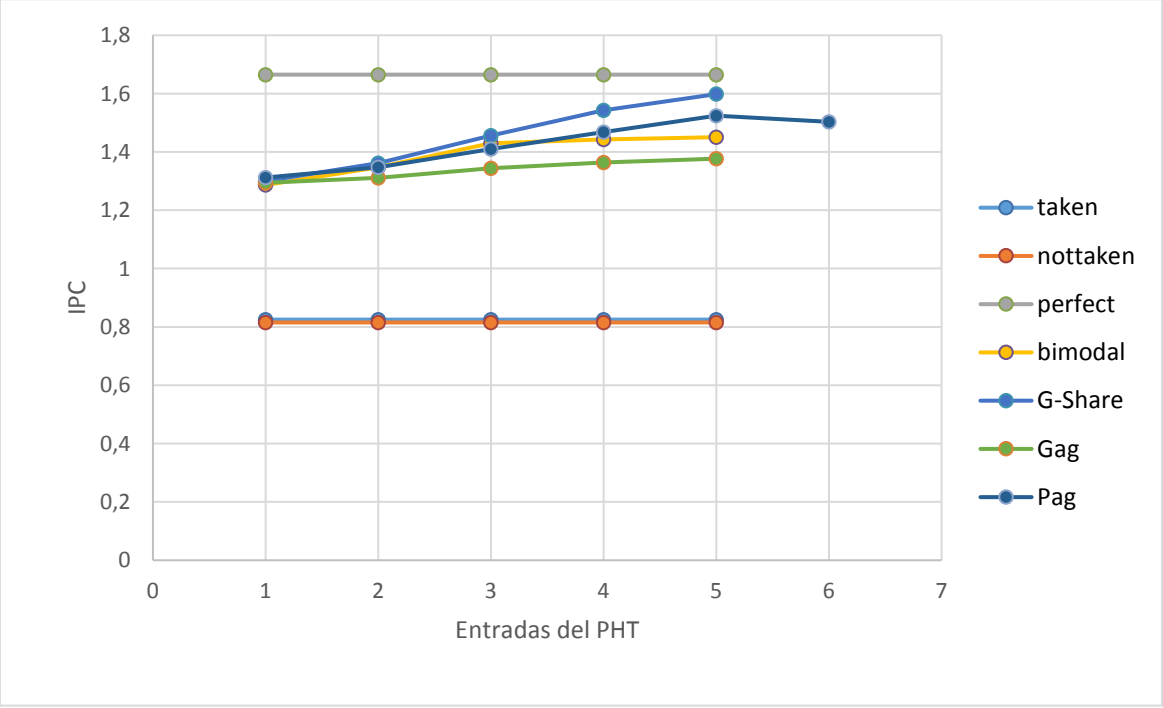
Després, el contingut de l’entrada del BHT corresponent s’opera amb una XOR junt amb la direcció de la instrucció de salt actual per a generar un índex per accedir a la PHT i predir el salt.

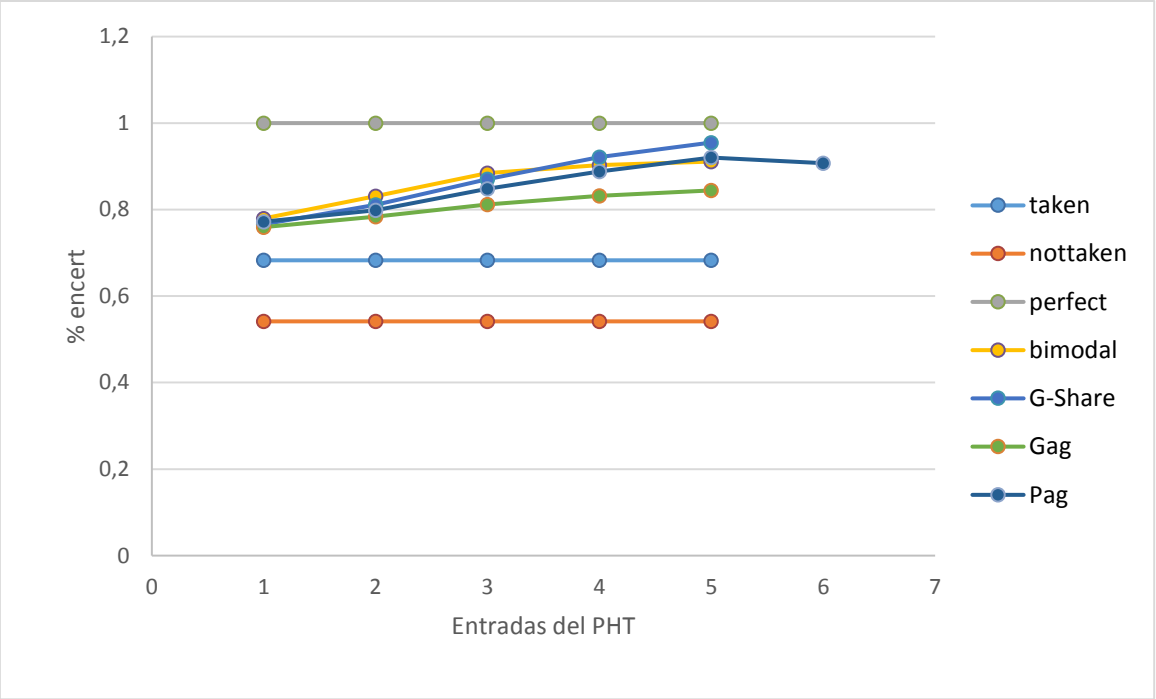
Podem observar que entre el cas 128-2048 i 64-4096, hi ha una millor predicció quan la BHT té una capacitat major tot i que la PHT es redueixi a la meitat. Per tant, resulta tan important tenir una PHT suficientment gran per a que no hi hagin col·lisions, com tenir una BHT gran per

a que hi hagin poques instruccions de salt que comparteixin el mateix registre. Això ens permet predir de manera més efectiva patrons repetitius en els salts.

Resum

		Resum					
		16 (8-8)	64 (16-32)	256 (32-128)	1024 (64-512)	4096 (128-2048)	64-4096
taken	IPC	0,8251	0,8251	0,8251	0,8251	0,8251	-
	% encerts	0,68278	0,68278	0,68278	0,68278	0,68278	-
no taken	IPC	0,8154	0,8154	0,8154	0,8154	0,8154	-
	% encerts	0,54162	0,54162	0,54162	0,54162	0,54162	-
perfect	IPC	1,66508	1,66508	1,66508	1,66508	1,66508	-
	% encerts	1	1	1	1	1	-
bimodal	IPC	1,28674	1,34766	1,43008	1,44328	1,45052	-
	% encerts	0,77962	0,83086	0,8845	0,90276	0,91108	-
G-Share	IPC	1,3	1,36198	1,45588	1,54256	1,59818	-
	% encerts	0,76602	0,81108	0,8704	0,92114	0,95508	-
Gag	IPC	1,29424	1,31132	1,34462	1,36392	1,37678	-
	% encerts	0,75952	0,78394	0,8122	0,83198	0,84438	-
Pag	IPC	1,31224	1,34754	1,40982	1,46798	1,5243	1,50314
	% encerts	0,77214	0,7982	0,8484	0,88796	0,92032	0,90724



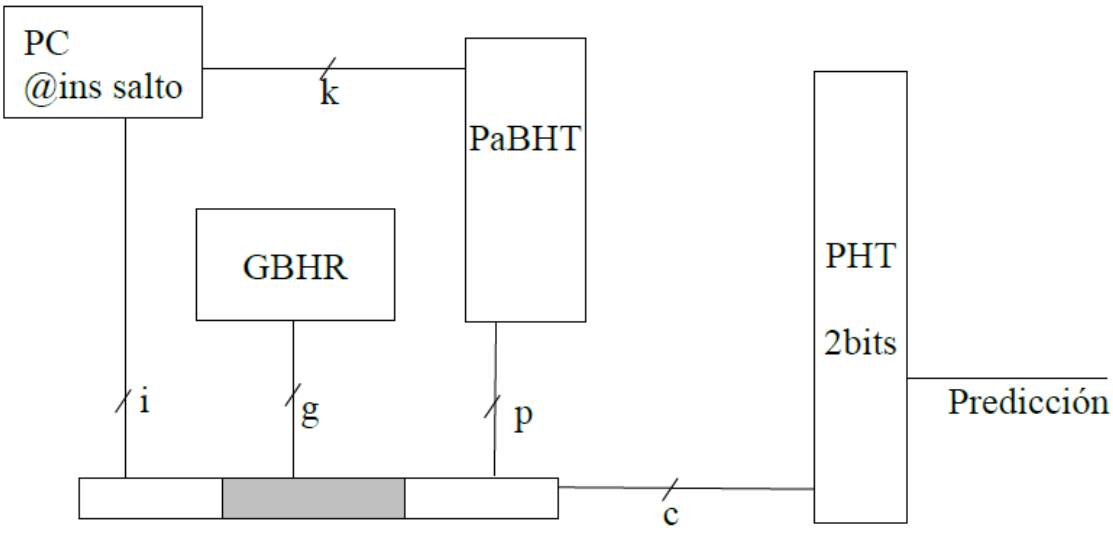


En el gràfic podem observar que el predictor G-Share és el predictor que té el major creixement a mesura que la PHT es fa gran. En el bimodal podem veure efectivament una corba asimptòtica respecte a la resta de predictor dinàmics, aquests presenten una correlació directa entre el % d'encerts, l'IPC i la mida de la PHT. Sobre els predictor taken i not taken veiem que efectivament el taken té major % d'encerts, però el IPC, per alguna raó (podria ser algun problema amb el simulador) no augmenta de manera proporcionada a aquest % d'encerts.

Podríem concloure dient que el predictor G-Share resulta el més efectiu a l'hora de predir el salt, i també que resulta un dels més òptims ja que tampoc necessita uns requeriments hardware elevats, concretament no necessita una BHT.

Fase 2: Implementació del predictor de salts ALLOYED

El predictor que implementarem tindrà aquesta estructura:



- Un registre d'història de salts global (GBHR)
- Una taula amb les històries de salt de cada salt (PaBHT)
- La Patern History Table (PHT) → en cada entrada emmagatzema dos bits per a implementar un comptador saturat (2BitCounter), on el bit de més pes ens indicarà el resultat de la predicció (1 Taken, 2 Not Taken).

Implementació del predictor en el SimpleScalar:

En el bpred.h:

- Afegim el predictor a la llista de enum bpred_class:

```
/* branch predictor types */
enum bpred_class {
  BPredComb,                    /* combined predictor (McFarling) */
  BPred2Level,                  /* 2-level correlating pred w/2-bit counters */
  BPred2bit,                    /* 2-bit saturating cntr pred (dir mapped) */
  BPredTaken,                   /* static predict taken */
  BPredNotTaken,                /* static predict not taken */
  BPred_NUM,
  BPredALLOYED                  /*Predictor implementat per nosaltres*/
};

/* branch predictor def */
struct bpred_t {
   enum bpred_class class;       /* type of predictor */
   struct {
     struct bpred_dir_t *bimod;    /* first direction predictor */
     struct bpred_dir_t *twolev;   /* second direction predictor */
     struct bpred_dir_t *alloy;    /* third direction predictor */
     struct bpred_dir_t *meta;     /* meta predictor */
   } dirpred;
```

- Afegim les noves variables i apuntadors extra necessaris per al nostre predictor:

```
/* direction predictor def */
struct bpred_dir_t {
  enum bpred_class class;       /* type of predictor */
  union {
    struct {
      unsigned int size;        /* number of entries in direct-mapped table */
      unsigned char *table;     /* prediction state table */
    } bimod;
    struct {
      int l1size;               /* level-1 size, number of history regs */
      int l2size;               /* level-2 size, number of pred states */
      int shift_width;          /* amount of history in level-1 shift regs */
      int xor;                  /* history xor address flag */
      int *shiftregs;           /* level-1 history table */
      unsigned char *l2table;   /* level-2 prediction state table */
      //elements adicionals per al nostre predictor
      int hist_width;           /*mida del GBHR*/
      int *shiftGlobregs;       /*punter GBHR*/
    } two;
  } config;
};
```

En aquest fitxer també haurem de modificar l'especificació de les funcions que posteriorment modificarem en el bpred.c

```
/* create a branch predictor */
struct bpred_t *                        /* branch predictory instance */
bpred_create(enum bpred_class class,    /* type of predictor to create */
             unsigned int bimod_size,   /* bimod table size */
             unsigned int l1size,       /* level-1 table size */
             unsigned int l2size,       /* level-2 table size */
             unsigned int meta_size,    /* meta predictor table size */
             unsigned int shift_width,  /* history register width */
////////////////////////////////////////////////////////////////////////////////
             unsigned int hist_width,   /* history register width */
////////////////////////////////////////////////////////////////////////////////
             unsigned int xor,          /* history xor address flag */
             unsigned int btb_sets,     /* number of sets in BTB */
             unsigned int btb_assoc,    /* BTB associativity */
             unsigned int retstack_size);/* num entries in ret-addr stack */

/* create a branch direction predictor */
struct bpred_dir_t *           /* branch direction predictor instance */
bpred_dir_create (
  enum bpred_class class,      /* type of predictor to create */
  unsigned int l1size,         /* level-1 table size */
  unsigned int l2size,         /* level-2 table size (if relevant) */
  unsigned int shift_width,    /* history register width */
  unsigned int xor,            /* history xor address flag */
////////////////////////////////////////////////////////////////////////////////
  unsigned int hist_width);    /*global history register width*/
////////////////////////////////////////////////////////////////////////////////
```

En el sim-outorder.c:

- En el sim_reg_options registrem els paràmetres de configuració del predictor:

```
////////////////////////////////////////////////////////////////////////////
/* 3-level predictor config (<l1size> <l2size> <hist_size> <hist_width>) */
static int alloy_nelt = 4;
static int alloy_config[4] =
  { /* l1size */128, /* l2size */2048, /* hist */4, /* Ghist */4};
////////////////////////////////////////////////////////////////////////////

  opt_reg_int_list(odb, "-bpred:alloy",
                   "alloy predictor config "
                   "(<l1size> <l2size> <hist_size> <GBHR_size>)",
                   alloy_config, alloy_nelt, &alloy_nelt,
                   /* default */alloy_config,
                   /* print */TRUE, /* format */NULL, /* !accrue */FALSE);
```

l1size i l2size serà el nombre d'entrades de la PaBHT i la PHT respectivament i hist_size i GBHR_size serà el nombre de bits que emmagatzemarà (tindrem nosaltres en compte, ja que les entrades són de 32 bits) cada entrada de la PaBHT i el GBHR respectivament.

- Afegim en el sim_check_options la lectura dels paràmetres del nostre predictor i cridem la funció per crear-lo (bpred_create()):

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
  else if (!mystricmp(pred_type, "alloy"))
    {
        /*alloy predictor, bpred_create() checks args*/
      if (alloy_nelt != 4)
        fatal("bad alloy pred config (<l1size> <l2size> <hist_size> <GBHR_size>)");
      if (btb_nelt != 2)
        fatal("bad btb config (<num_sets> <associativity>)");

      pred = bpred_create(BPredALLOYED,
                /* bimod table size */0,
                /* alloy l1 size */alloy_config[0],            //fixem mida de PaBHT
                /* alloy l2 size */alloy_config[1],            //fixem mida de PHT
                /* meta table size */0,
                /* history reg size */alloy_config[2],         //nombre de prediccons emmagatemades en PaBHT
                /* alloy GBHR reg size*/alloy_config[3],       //nombre de prediccons emmagatemades en GBHR
                /* history xor address */0,
                /* btb sets */btb_config[0],
                /* btb assoc */btb_config[1],
                /* ret-addr stack size */ras_size);

    }
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

En el bpred.c:

- Afegit al bpred_create i bpred_dir_create un nou cas d'inicialització on a partir dels paràmetres definits del predictor es reservi la memòria necessària i s'inicialitzin les variables i taules als valors necessaris.

Afegim la nova variable necessària per al nostre predictor:

```
struct bpred_t *                        /* branch predictory instance */
bpred_create(enum bpred_class class,    /* type of predictor to create */
             unsigned int bimod_size,   /* bimod table size */
             unsigned int l1size,       /* 2lev l1 table size */
             unsigned int l2size,       /* 2lev l2 table size */
             unsigned int meta_size,    /* meta table size */
             unsigned int shift_width,  /* history register width */
             unsigned int hist_width,   /* history register width */
             unsigned int xor,          /* history xor address flag */
             unsigned int btb_sets,     /* number of sets in BTB */
             unsigned int btb_assoc,    /* BTB associativity */
             unsigned int retstack_size) /* num entries in ret-addr stack */
{
  struct bpred_t *pred;
```

Nou cas en el bpred_create:

```
case BPredALLOYED:
  pred->dirpred.alloy =
    bpred_dir_create(class, l1size, l2size, shift_width, 0, hist_width);

  break;
```

Tenim en compte que hem afegit una nova variable d'entrada em el bpred_dir_create (com s'observarà més avall) per tant haurem de modificar la crida d'aquesta funció per tots els altres predictors afegint com a 6é paràmetre d'entrada un 0.

Assignem el ret stack i el BTB per al nostre predictor:

```
  /* allocate ret-addr stack */
  switch (class) {
  case BPredComb:
  case BPred2Level:
  case BPred2bit:
//////////////////////////////////////////////////
  case BPredALLOYED:
//////////////////////////////////////////////////
```

Afegim la variable necessària per al nostre predictor:

```
/* create a branch direction predictor */
struct bpred_dir_t *            /* branch direction predictor instance */
bpred_dir_create (
  enum bpred_class class,       /* type of predictor to create */
  unsigned int l1size,          /* level-1 table size */
  unsigned int l2size,          /* level-2 table size (if relevant) */
  unsigned int shift_width,     /* history register width */
  unsigned int xor,             /* history xor address flag */
////////////////////////////////////////////////////////////////////////////////
  unsigned int hist_width)      /*global history register width*/
////////////////////////////////////////////////////////////////////////////////
{
```

Nou cas en el bpred_dir_create:

```
case BPredALLOYED:
    {
      if (!l1size || (l1size & (l1size-1)) != 0)
        fatal("level-1 size, `%d', must be non-zero and a power of two",
              l1size);
      pred_dir->config.two.l1size = l1size;

      if (!l2size || (l2size & (l2size-1)) != 0)
        fatal("level-2 size, `%d', must be non-zero and a power of two",
              l2size);
      pred_dir->config.two.l2size = l2size;

      if (!shift_width || shift_width > 30)
        fatal("shift register width, `%d', must be non-zero and positive",
              shift_width);
      pred_dir->config.two.shift_width = shift_width;

      if (!hist_width || hist_width > 30)
        fatal("shift global register width, `%d', must be non-zero and positive",
              hist_width);
      pred_dir->config.two.hist_width = hist_width;

      pred_dir->config.two.shiftregs = calloc(l1size, sizeof(int));
      if (!pred_dir->config.two.shiftregs)
        fatal("cannot allocate shift register table");

   pred_dir->config.two.l2table = calloc(l2size, sizeof(unsigned char));
   if (!pred_dir->config.two.l2table)
     fatal("cannot allocate second level table");

   pred_dir->config.two.shiftGlobregs = calloc(1, sizeof(unsigned char));
   if (!pred_dir->config.two.shiftGlobregs)
     fatal("cannot allocate global history register");

   /* initialize counters to weakly this-or-that */
   flipflop = 1;
   for (cnt = 0; cnt < l2size; cnt++)
     {
       pred_dir->config.two.l2table[cnt] = flipflop;
       flipflop = 3 - flipflop;
     }

   break;
 }
```

- Afegim en el bpred_config i bpred_dir_config un nou cas per a mostrar per la sortida del simulador la configuració del predictor.

Nou cas en el bpred_dir_config:

```
////////////////////////////////////////////////////////////////////////////////////////////////
  case BPredALLOYED:
    fprintf(stream,
      "pred_dir: %s: 2-lvl: %d l1-sz, %d bits/ent, %i GBHR bits, %d l2-sz, direct-mapped\n",
      name, pred_dir->config.two.l1size, pred_dir->config.two.shift_width,
      pred_dir->config.two.hist_width, pred_dir->config.two.l2size);
    break;
////////////////////////////////////////////////////////////////////////////////////////////////
```

Nou cas en el bpred_config:

```
case BPredALLOYED:
  bpred_dir_config (pred->dirpred.alloy, "alloy", stream);
  fprintf(stream, "btb: %d sets x %d associativity",
          pred->btb.sets, pred->btb.assoc);
  fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
  break;
```

- Afegim en el bpred_reg_stats un nou cas per poder veure el resultat de la simulació:

```
case BPredALLOYED:
  name = "bpred_alloyed";
  break;
```

- Afegim en el bpred_lookup y bpred_dir_lookup un nous cas per obtenir la direcció de la següent instrucció atenent a la predicció que es realitza a partir de la instrucció de salt que ha realitzat el fetch (sempre que trobi el BTB).

  Aclariment: bpred_dir_lookup torna la direcció de l'entrada de la taula PHT (taula l2) i el bpred_lookup la guarda en el camp pdir1 de la pròpia instrucció per a que en la següent crida a bpred_update ja tingui calculada la posició en la PHT.

Afegim un cas per al nostre predictor en el bpred_dir_lookup:

```
case BPredALLOYED:
  {
    int index, l2index;

    /* traverse 2-level tables */
    index = (baddr >> MD_BR_SHIFT) & (pred_dir->config.two.l1size - 1);     /*Obtenim el punter a la posicó de la PaBHT*/
    index = pred_dir->config.two.shiftregs[index];                          /*Obtenim el valor en la PaBHT*/
    /*ens quedem amb els shift_width bits de menys pes, per fer-ho elevem 2 a shift_width
    (1 << pred_dir->config.two.shift_width) i restem 1 per obtenir la màscara necessària*/
    l2index = index & ((1 << pred_dir->config.two.shift_width) - 1);        /*obtenim p, els bits de menys pes del punter a la PHT*/

    index = pred_dir->config.two.shiftGlobregs[0];                          /*Obtenim el valor en el GBHR*/
    index = index & ((1 << pred_dir->config.two.hist_width) - 1);           /*obtenim g*/
    l2index = l2index | (index << pred_dir->config.two.shift_width);        /*index a la PHT (c) ja té dos dels valors g-p*/

    /*Obtenim un punter de 32 bits a la PHT on els bits de menys pes venen de la PaBHT, els mitjans del GBHR i els de més pes del PC*/
    l2index = l2index | ((baddr >> MD_BR_SHIFT) << (pred_dir->config.two.shift_width+pred_dir->config.two.hist_width));

    /*Obtenim el punter a la PHT amb els bits necessaris*/
    l2index = l2index & (pred_dir->config.two.l2size - 1);

    /* get a pointer to prediction state information */
    p = &pred_dir->config.two.l2table[l2index];
  }
  break;
```

Afegim un cas per al nostre predictor en el bpred_lookup:

```
////////////////////////////////////////////////////////////////////////////////
    case BPredALLOYED:
      if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
        {
          dir_update_ptr->pdir1 =
            bpred_dir_lookup (pred->dirpred.alloy, baddr);
        }
      break;
////////////////////////////////////////////////////////////////////////////////
```

- Afegir a bpred_update el cas per actualitzar el GBHR, el PaBHT (L1 Table) i la PHT (L2 Table).

```
////////////////////////////////////////////////////////////////////////////////////////////////
 if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND) &&
     (pred->class == BPredALLOYED))
    {
      int llindex, shift_reg;

      /* also update appropriate L1 history register */
      llindex = (baddr >> MD_BR_SHIFT) & (pred->dirpred.alloy->config.two.l1size - 1);
      shift_reg = (pred->dirpred.alloy->config.two.shiftregs[llindex] << 1) | (!!taken);
      pred->dirpred.alloy->config.two.shiftregs[llindex] =
                              shift_reg & ((1 << pred->dirpred.alloy->config.two.shift_width) - 1);
      /* also update appropriate GBHR */
      shift_reg = (pred->dirpred.alloy->config.two.shiftGlobregs[0] << 1) | (!!taken);
      pred->dirpred.alloy->config.two.shiftGlobregs[0] =
                              shift_reg & ((1 << pred->dirpred.alloy->config.two.hist_width) - 1);
    }
////////////////////////////////////////////////////////////////////////////////////////////////
```

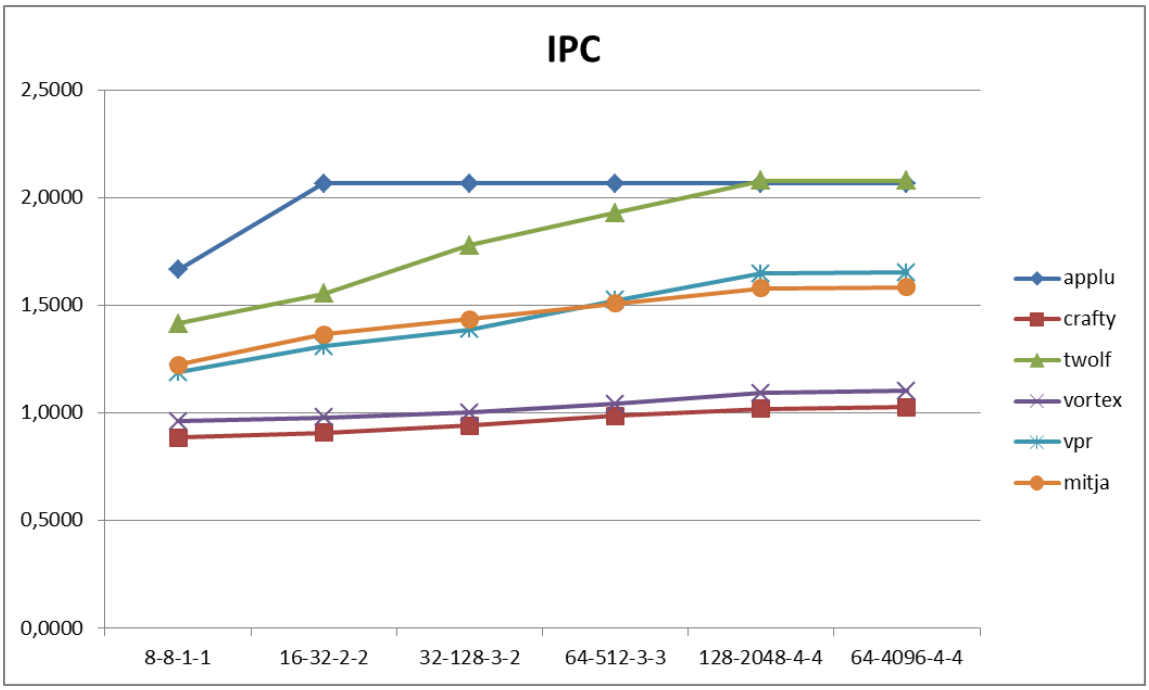
Solució d'errors:

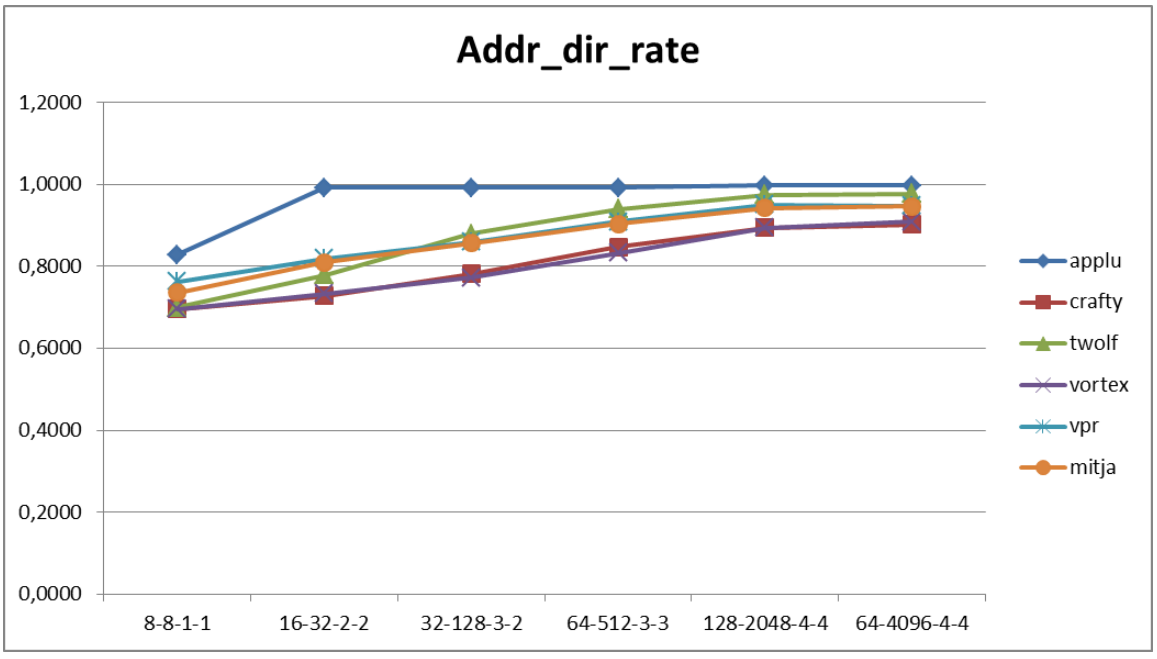
El primer error que he trobat, és que en el fitxer sim-outorder.c no hem afegit una variable més com a zero en la funció bpred_create, de forma que al cridar-la ens saltava un error del tipus: *Too few arguments to function: bpred_create*.

```
  else if (!mystricmp(pred_type, "taken"))
    {
      /* static predictor, not taken */
      pred = bpred_create(BPredTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    }
  else if (!mystricmp(pred_type, "nottaken"))
    {
      /* static predictor, taken */
      pred = bpred_create(BPredNotTaken, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
    }
```

Resultats:

	8-8-1-1		16-32-2-2		32-128-3-2		64-512-3-3		128-2048-4-4		64-4096-4-4	
	IPC	dir_rate	IPC	dir_rate	IPC	dir_rate	IPC	dir_rate	IPC	dir_rate	IPC	dir_rate
applu	1,6652	0,8286	2,0658	0,9924	2,0658	0,9924	2,0658	0,9924	2,0658	0,9972	2,0659	0,9972
crafty	0,8853	0,6952	0,9088	0,7275	0,9421	0,7815	0,9867	0,8472	1,0200	0,8937	1,0262	0,9021
twolf	1,4152	0,6997	1,5550	0,7785	1,7799	0,8800	1,9297	0,9398	2,0777	0,9744	2,0800	0,9772
vortex	0,9605	0,6961	0,9798	0,7328	1,0016	0,7726	1,0420	0,8322	1,0921	0,8947	1,1017	0,9087
vpr	1,1876	0,7634	1,3108	0,8193	1,3875	0,8595	1,5250	0,9091	1,6472	0,9499	1,6517	0,9479
mitja	1,2228	0,7366	1,3640	0,8101	1,4354	0,8572	1,5098	0,9041	1,5806	0,9420	1,5851	0,9466





Podem veure que els resultats d'aquest predictor són força semblants al del predictor Pag. Això probablement és degut a que el mecanisme de predicció és força semblant. La única diferència és que en el predictor ALLOYED trobem que els bits de pes mitjà de l'índex per accedir a la taula procedeixen d'un BHR global. De igual forma, podem observar com en el predictor Pag, la disminució del BHT implica directament una baixada del % d'encerts encara que augmentem la PHT en compensació (es pot veure al passar del cas 128-2048 a 64-4096). En el predictor ALLOYED, en canvi, el fet de tenir aquests bits de pes mitjà

implica que la disminució de la BHT no afecti de manera tan dràstica al % d'encerts. En l'última columna, tot i haver disminuït la BHT, el % d'encerts no es veu afectat.
Podem observar també la mateixa tendència d'augment del % d'encerts respecte a la mida del PHT i BHT en els dos predictors.