ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA
Universitat Rovira i Virgili

[DΣIM]

Arquitectura de Computadors
Pràctica 2
curs 2024-25

Estudiant: Gemma Goitia Gál i Ivan Garcia Pallares

Grup Laboratori: LX

Professor/a: Carles Aliagues

Data de lliurament: 20/11/2024

Índex

Fase 1: estudi dels predictors

Gràfiques amb les diferents configuracions i paràmetres

Predictors de salt estàtics

Els processadors de salts estàtics realitzen la predicció en temps de compilació, és a dir, abans de que el programa s'executi. Aquests realitzen la predicció basant-se en la informació sobre el comportament que hi ha hagut amb anterioritat. Són més simples d'implementar que els dinàmics i per això també menys precisos.

Els predictors estàtics estudiants són el Taken, Not Taken i el Perfect.

Taken → sempre prediu que el salt serà pres, per aquest motiu té un % alt d'errors.

Not Taken → a diferència del Taken, aquest sempre prediu que el salt no serà pres.

Perfect → és un predictor amb una precisió teórica màxima, ja que sap exactament quien serà el resultat del salt cada vegada. Un predictor com aquest és pràcticament impossible d'implementar ja que requeriria informació futura i una complexitat gran i molts recursos.

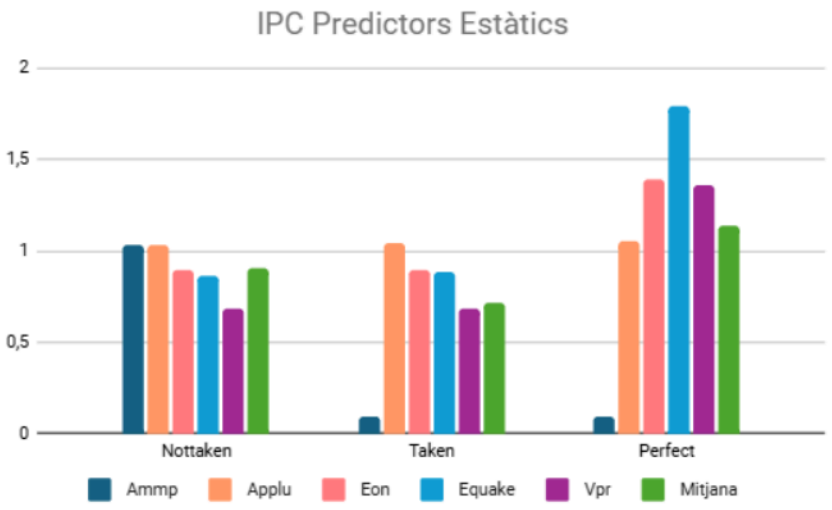
Les comandes executades han sigut les següents:

Hem posat l'exemple del Not Taken per no haver de posar 15 comandes. La diferència amb els altres dos predictors ha sigut el paràmetre ***-bpred*** *que s'ha modificat* ***-bpred perfect*** *per o* ***-bpred taken****.*

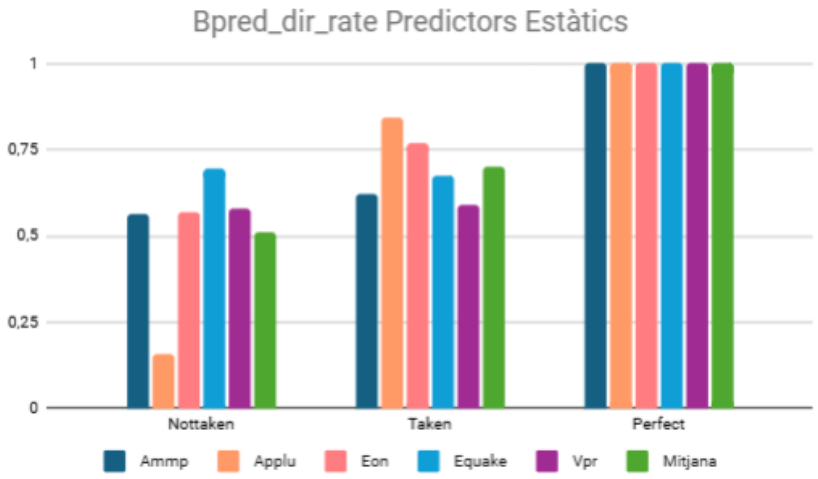
```
#ammp

sim-outorder -fastfwd 100000000 -max:inst 100000000 -bpred nottaken
-mem:width 32 -mem:lat 300 2 ../../exe/ammp.exe < ammp.in


#applu

sim-outorder -fastfwd 100000000 -max:inst 100000000 -bpred nottaken
-mem:width 32 -mem:lat 300 2 ../../exe/applu.exe < applu.in
```

```
#eon

sim-outorder -fastfwd 100000000 -max:inst 100000000 -bpred nottaken
-mem:width 32 -mem:lat 300 2 ../../exe/eon.exe chair.control.cook
chair.camera chair.surfaces chair.cook.ppm ppm pixels_out.cook


#equake

sim-outorder -fastfwd 100000000 -max:inst 100000000 -bpred nottaken
-mem:width 32 -mem:lat 300 2 ../../exe/equake.exe < inp.in


#vpr

sim-outorder -fastfwd 100000000 -max:inst 100000000 -bpred nottaken
-mem:width 32 -mem:lat 300 2 ../../exe/vpr.exe net.in arch.in place.out
dum.out -nodisp -place_only -init_t 5 -exit_t 0.005 -alpha_t 0.9412
-inner_num 2
```

A continuació es mostraran les gràfiques fetes amb els resultats de les simulacions realitzades:



Podem veure com el IPC del perfect és el més alt dels 3 predictors, mentre que el del Taken i el Not Taken es mostren practicament igual de baixos. Això és degut a que el percentatge d'encerts en general és baix i afecta de manera negativa al rendiment ja que s'ha de parar a buidar el pipeline per cada salt mal predit i això té un cost.



El predictor Perfect sempre tindrà un percentatge d'encerts del 100% tal i com el seu nom diu, ja que simula com si encertés tots els salts.

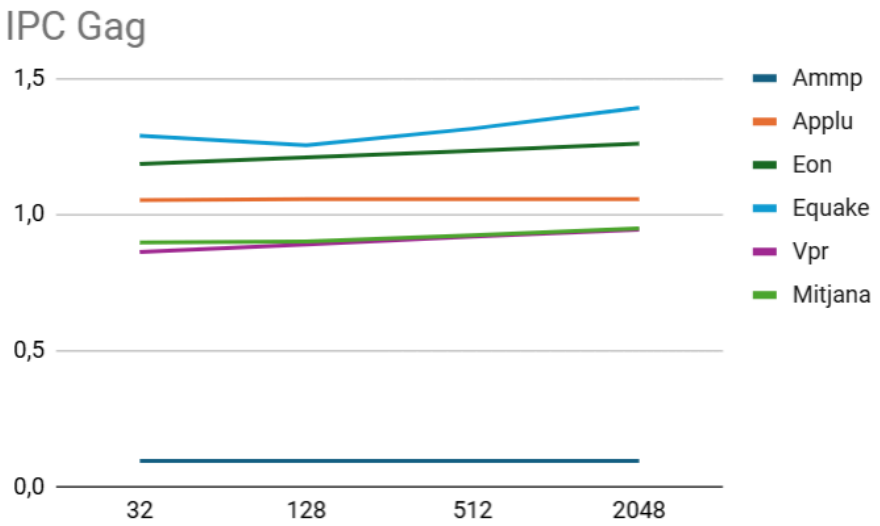
Podem observar com amb el benchmark Applu el predictor Taken té una taxa d'encerts importantment baixa, a diferència del Not Taken, que amb aquest mateix aconsegueix els valors més alts, cas invers. La diferència entre els dos predictors és molt petita. La lògica que hauria de seguir és complementar els resultats del predictor contrari, ja que un prediu taken i l'altre not taken, és a dir, tots els salts inversos al predictor, aquest els fallara (exemple: el NotTaken fallarà tots els salts taken).

Predictors de salts dinàmics

Els predictors de salts dinàmics realitzen la predicció durant l'execució del programa. Utilitzen informació sobre el patró d'execució recent per prendre decisions sobre si saltarà o no. Molts d'ells guarden la informació del historial.

Els predictors dinàmics estudiats són el Gag, el Gshare, el Bimodal i el Pag.

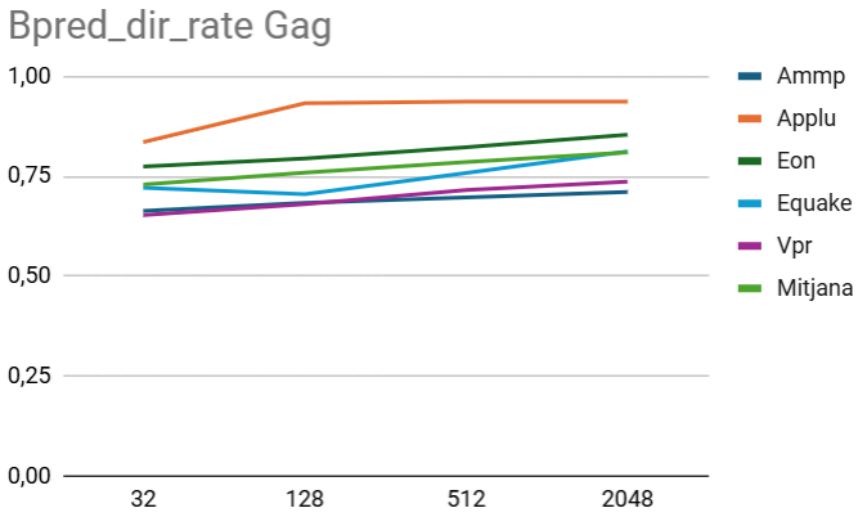
Gag → és un predictor utilitzat principalment per predir els salts indirectes, on es necessita calcular la direcció destí del salt abans de prendre'l. *Per entendre, un salt indirecte és una instrucció que no especifica directament la direcció destí, sinó que conté una direcció que apunta a una altre.* Aquest predictor requereix taules o buffers per emmagatzemar la informació sobre els patrons d'execució dels salts. Els patrons d'execució dels salts són les seqüències de resultats observats en les repetides execucions d'una instrucció (*exemple: sempre salta o mai salta, el salt canvia entre execucions, etc.*.).



Per la majoria dels benchmarks l'IPC augmenta lleugerament a mesura que ho fa el tamany de la finestra. A mesura que la finestra d'instruccions augmenta, ens permet examinar més instruccions paral·lelament.

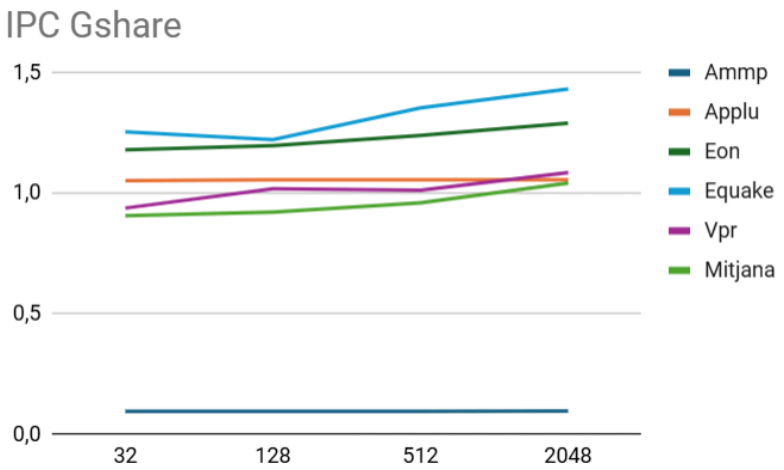
Applu té un IPC constant en totes les mides de la finestra. Això podria indicar que aquest benchmark té un nivell de paral·lelisme bastant limitat i el tamany de la finestra no afecta en el seu rendiment. L'ammp s'ha comportat d'una manera similar però amb uns resultats bastant més baixos.

Els demés es beneficien del augment de tamany per millorar l'IPC. L'equake sobretot.



Respecte el percentatge d'encerts, Applu mostra un gran augment entre el tamany de 32 i 128 després és mostra completament constant arran del 0,9, indicant que el Gag és molt precís per l'Applu i que la mida de la finestra no és afecta gaire al % d'encerts. Tots els demés augmenten lleugerament a mesura que ho fa el tamany.

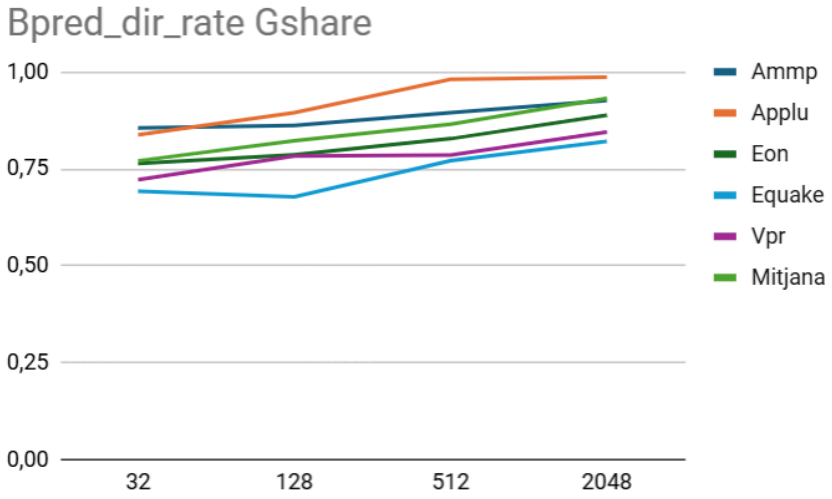
Gshare → utilitza una combinació del historial guardat de les execucions anteriors i el pc per fer la predicció. Explicat millor, Gshare té una taula de predicció amb entrades de n bits, aquesta guarda l'historial dels salts anteriors per a poder predir els futurs salts amb més precisió. A part, té un registre *Global Branch History Register* que guarda l'historial dels salts com una sequència de bits (*exemple: 1010 → significa que els ultims quatre salts havien seguit un patró de cada dos no saltava*). Aquest predictor realitza una XOR entre el registre i la direcció de la instrucció (pc) per obtenir un índex amb el qual consultar la taula en aquell índex calculat i decidir que fer segons el contingut. Un cop s'ha predit el salt, s'actualitza l'entrada de la taula segons el resultat real. S'utilitza en processadors com AMD i Intel Core.



Tornem a veure com el benchmark Ammp torna a ser el més baix mantenint-se constant respecte l'augment del tamany. Això indica que el seu rendiment no millora ni amb el canvi de la finestra ni amb el canvi e predictor de salts. Està bastant limitat.

Applu també mostra un comportament similar al predictor anterior.

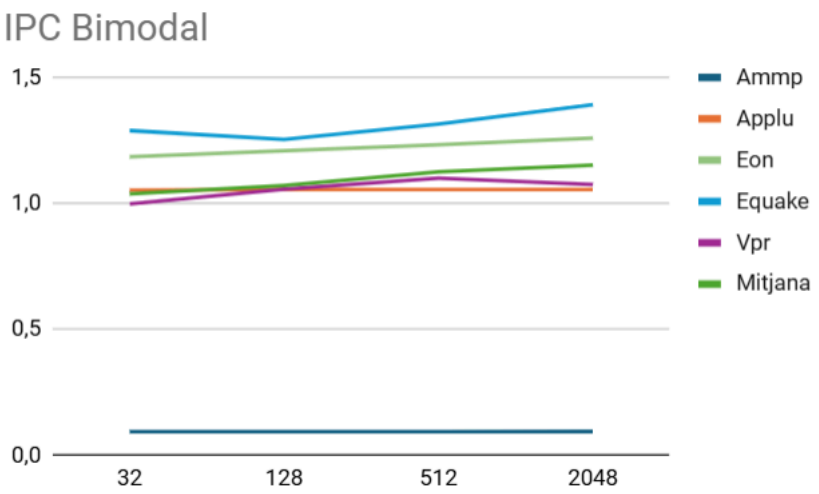
Equake mostra una augment significatiu a partir del tamany 128 lo qual significa que com més gran la finestra millor rendiment tindrà. Eon creix lentament a mesura que ho fa el tamany, mentre que l'Vpr creix entre el 32-128 i 512-2048, quedant-se constant entre 128-512.



Tots els benchmarks produeixen millors resultats amb tamanys grans de la finestra d'instruccions. El Gshare mostra un mitjà major al Gag.

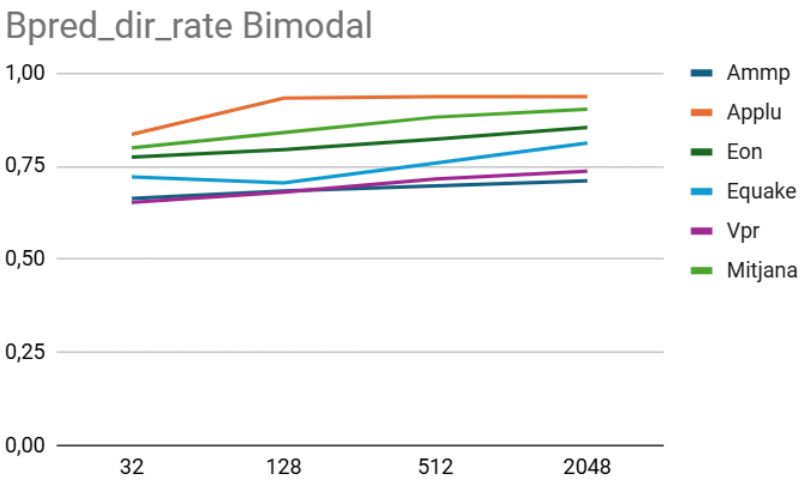
Bimodal → utilitza una taula de de prediccions que guarda estats, indexada només per la direcció del salt. Utilitza 2 estats diferents (taken o not taken) per predir el salt. Amb dos contadors binaris representa els estats. Quan entra una direcció, el predictor agafa uns bits del pc per utilitzar-los com a índex de la taula. Allí mirarà el contingut i decidirà quin pas realitzar. Un cop executat el salt o no, s'actualitzarà el contingut deixant-lo igual si ha estat correcte o canviant-lo per lo contrari si ho ha fet malament.

```
def bimodal_predictor(branch_history):
        if branch_history == '00':
            return 'not_taken'
        elif branch_history == '01':
            return 'weakly_not_taken'
        elif branch_history == '10':
            return 'weakly_taken'
        else:
            return 'strongly_taken'
```



Tornem a veure com el benchmark Ammp és completament independent al canvi de tamany, mentre que ens els demés augmenten lleugerament a mesura que ho fa la mida. L'Vpr arriba un punt en que no aconsegueix millorar el seu IPC i comença a disminuir de nou. El tamany més gran dificulta el seu rendiment.

Respecte el percentatge d'encerts, l'Applu encara i ser el més alt, arriba un punt en que s'estanca i es manté constant.

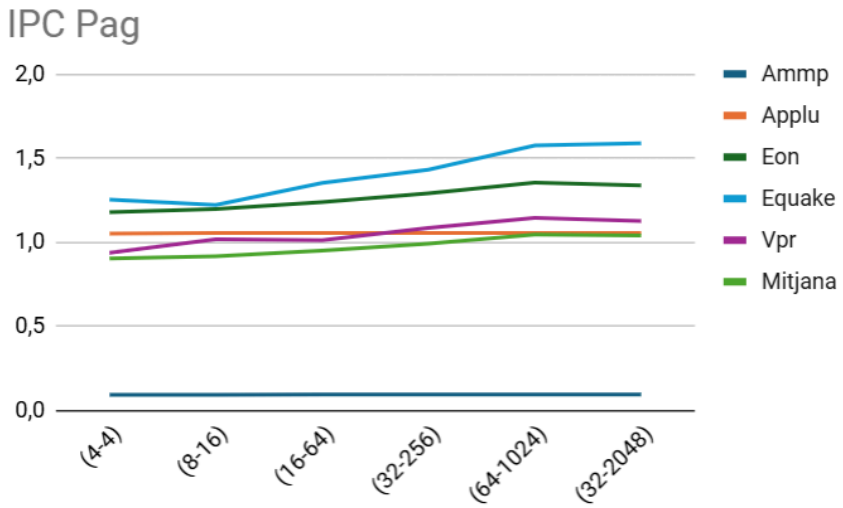


Pag → el predictor Pag utilitza informació sobre patrons d'execució recents. Guarda aquesta informació en una *pattern history table.* Cada entrada de la taula correspon a una combinació específica de bits del pc de la instrucció, i a més, cada una d'elles conté un contador binari. Com el Gshare, també té un registre d'historial de salts. Quan una instrucció s'executa, l'estat del contador corresponent s'actualitza. La taula manté un registre de patrons permetent ajustar la predicció en funció d'aquesta informació.

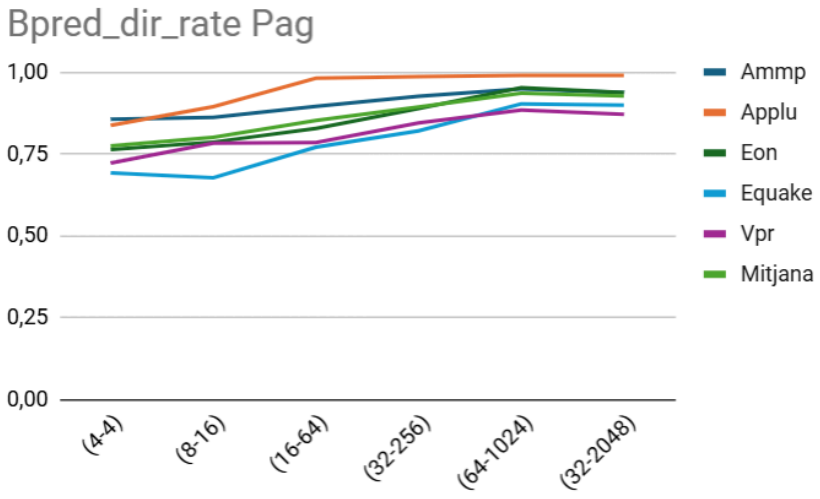
La diferència entre el Pag i el Gshare, ja que hem vist que són bastant gual respecte el funcionament, és primentament la taula, el Gshare té una taula de predicció compartida per totes les instruccions mentre que el Pag té una taula específica per a cada instrucció. Com a segon punt tenim el registre d'historial que en el Pag és l'índex per accedir a cada taula mentre que en el Gshare, el registre es combina amb la direcció per obtenir l'índex de la taula. El Gag també es similar a aquests 2, té una única taula compartida i utilitza l'historial per accedir a ella.

Taula comparativa dels 3 predictors demanda al ChatGPT i creada per ell:

Característica	GAg	Gshare	PAg
Historial utilizado	Global compartido	Global compartido	Global compartido
Tabla de predicción	Única tabla compartida para todos los saltos	Única tabla compartida para todos los saltos	Una tabla específica para cada salto
Conflictos en la tabla	Posibles debido a la tabla compartida	Reducidos con la operación XOR	Menos frecuentes por usar tablas específicas
Precisión en patrones	Buena en patrones globales	Buena en patrones globales y mezcla algunos patrones locales	Mejor para patrones individuales de cada salto
Consumo de memoria	Menor (solo una tabla)	Menor (solo una tabla)	Mayor (una tabla por cada salto)



A mesura que arribem a tamanys com 1024 i 2048 l'IPC s'estabilitza en xifres constants. Encara i així, l'augmente que es produeix a mesura que s'augmenta el tamany tampoc és gaire significatiu per Pag, menys pel Equake que és el que més creix. Un cop més, l'Ammp aconsegueix els pitjors resultats entre tots.



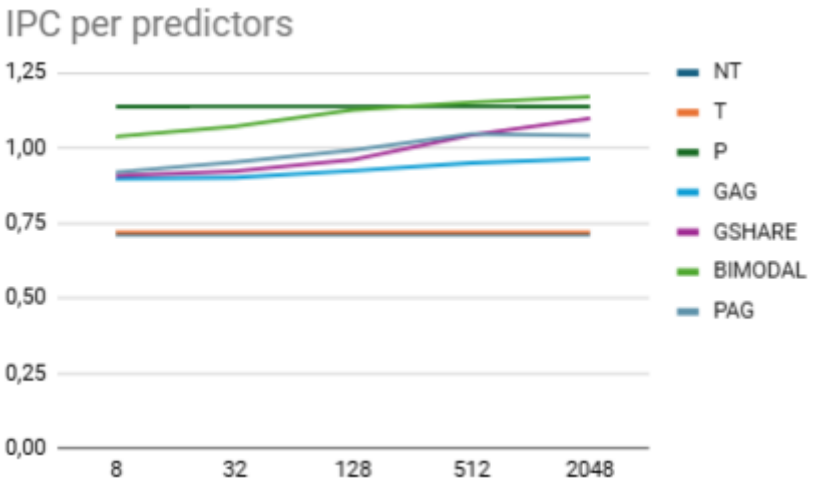
El benchmark Applu arriba a un percentatge d'encerts quasi total pels tamanys més grans. Pels demés benchmarks podem observar que en l'interval 32-2048 es produeix un petit descens, considerant així el pic més alt d'encerts entre 64 i 1024.

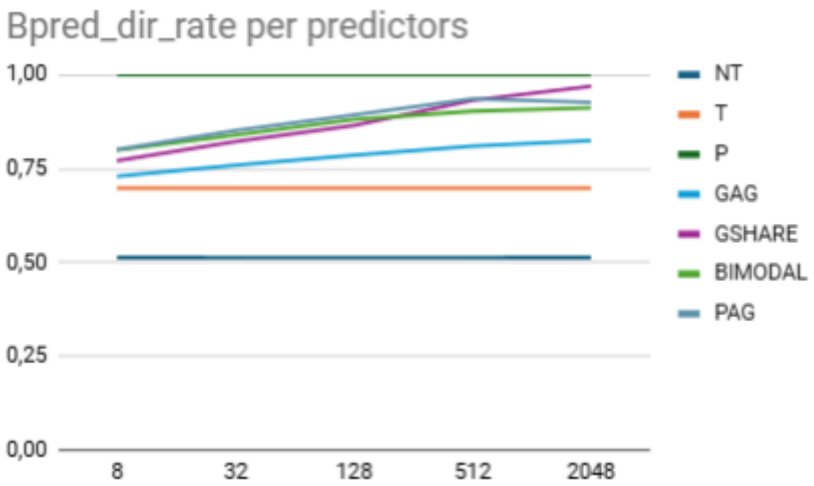
Gràfiques dels comportaments

Aquestes gràfiques s'han tret mitjançant les mitjanes calculades, és a dir la mitjana entre tots els benchmarks per cada predictor.

Pel Taken, Not Taken i Perfect s'ha utilitzat l'únic valor que hi havia i s'ha repetit per cada tamany ja que al demanar-se els estàtics junts no tenim les dades per a aquells valors.

Pel Pag, com que s'havia calculat per intervals enlloc de tamanys fixos, la columna del (4-4) no s'ha tingut en compte. La del (8-16) s'ha col·locat al tamany 8, (16-64) al 32 ja que està dins l'interval, (32-256) a 128, (64-1024) a 512 i (32-2048) a 2048.





Conclusions de l'estudi

IPC

Analitzant les gràfiques del apartat anterior, el que veiem que ha proporcionat millor rendiment ha estat el Bimodal superant al Perfect en el tamany de la finestra més gran estudiat. Seguidament trobem el Gshare mostrant un augment més significatiu a partir del 128 juntament amb el Pag però aquest mostrant-se constant a partir del 512.

Com ja s'esperava els predictors estàtics són els que proporcionen els pitjors resultats ja que no tenen en compte l'historial d'aquests ni el seu comportament.

Bpred_dir_rate

El predictor amb el percentatge més alt ha estat el Perfect ja que com diu el seu nom sempre tindrà un 100% d'encerts. Seguidament ha estat el Pag encara que Gshare l'ha superat en el tamany més gran simulat. Not Taken a penes supera el 50% d'encerts sent així el pitjor dels 7.

Podem concloure que guardar un historial global del comportament dels salts realitzats és efectiu al tenir mides de taules grans. El fet de fallar una predicció té un alt cost respecte el rendiment ja que s'ha de parar l'execució, buidar el pipeline i tornar a carregar tot de nou. Aquest fet ens deixa veure que majoritàriament l'IPC i el % d'encerts són directament proporcionals, ja que si fallem molt, patirem les conseqüències moltes vegades, i si encertem molt podrem executar més instruccions sense interrupcions.

Fase 2: modificació pel nou predictor

Per entendre aquesta fase necessitarem entendre les estructures que mencionen al principi de la pràctica que son BTB, RAS, BHR, GBHR, PaBHR, PHT.

BTB

El BTB és un buffer que emmagatzema l'adreça de destinació d'un salt condicional o incondicional que ja s'ha executat. Quan el processador troba un salt, cerca en el BTB l'adreça de destinació. Si la troba salta directament a aquesta direcció i si no, el processador la calcula.

Estructura:

Dirección PC del salto	Dirección de Destino
0x0040ABCD	0x0040BEF1
0x0040D123	0x0040D456

RAS

El RAS és una pila especial que guarda les direccions de retorn de trucadores a funcions. Cada vegada que es diu a una fusió, la seva adreça de retorn s'apila en el RAS, i en tornar de la funció s'extreu (pop) la direcció del RAS per a saber a on tornar.

Estructura:

PILA
0x0040BEF1 (retorn funcio 1)
0x0040D456 (retorn funcio 2)
0x0040ABCD (retorn funcio 3)

BHR

El BHR guarda un historial únic dels últims salts taken o no taken d'unes instruccions en general. Això s'usa per a preveure el comportament de salts futurs. L'historial sol representar-se com una seqüència de bits (1 per a salt taken, 0 per a not taken).

Estructura:

BHR	1010

GBHR

En GBHR és un BHR compartit globalment, és a dir, un sol registre d'historial per a tots els salts en el processador. Aquest registre captura el comportament de salts globals, no d'instruccions individuals.

PaBHR

El PaBHR manté un historial específic per a cada direcció de salt. Així, el predictor aprèn el patró de cada salt en particular, la qual cosa és útil quan uns certs salts tenen comportaments predictibles.

Estructura:

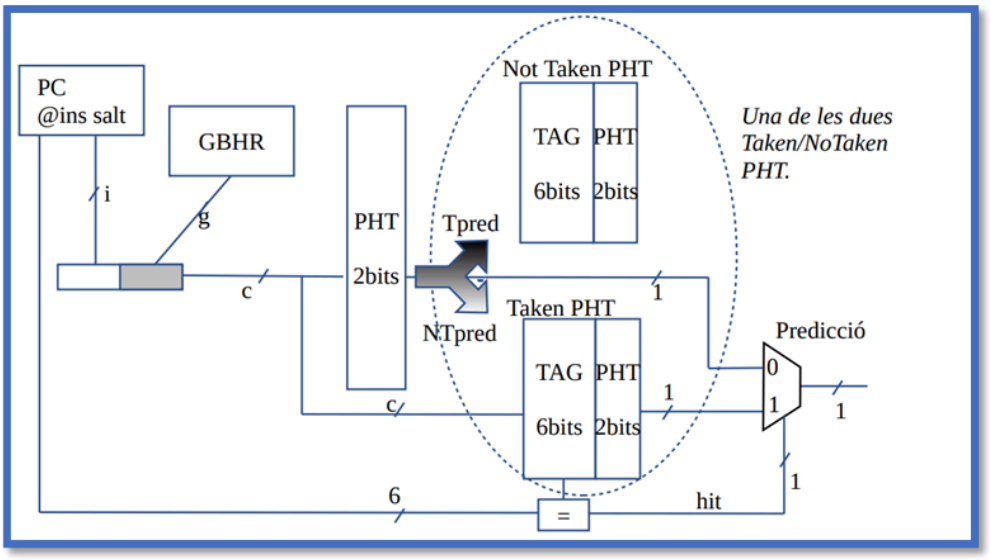
PaBHR para 0x0040D123	1011
PaBHR para 0x0040ABCD	0010

PHT

La PHT és una taula que utilitza l'historial de salts de (de BHR, GBHR o PaBHR) per a indexar patrons de predicció. Cada entrada en la PHT registra la probabilitat de prendre el salt (per exemple, mitjançant un comptador).

Historial	Contador
1010	11 (taken)
1101	01 (not taken)

Una cop tenim les estructures que crearem i/o modificarem en questa part toca entendre el predictor a afegir un nou predictor, YAGS (Yet Another Global Scheme) dins del 'simplesim-3.0_acx2'. Dins del enunciat ens deixen una estructura visual per poder entendre el seu disseny junt amb les funciones.



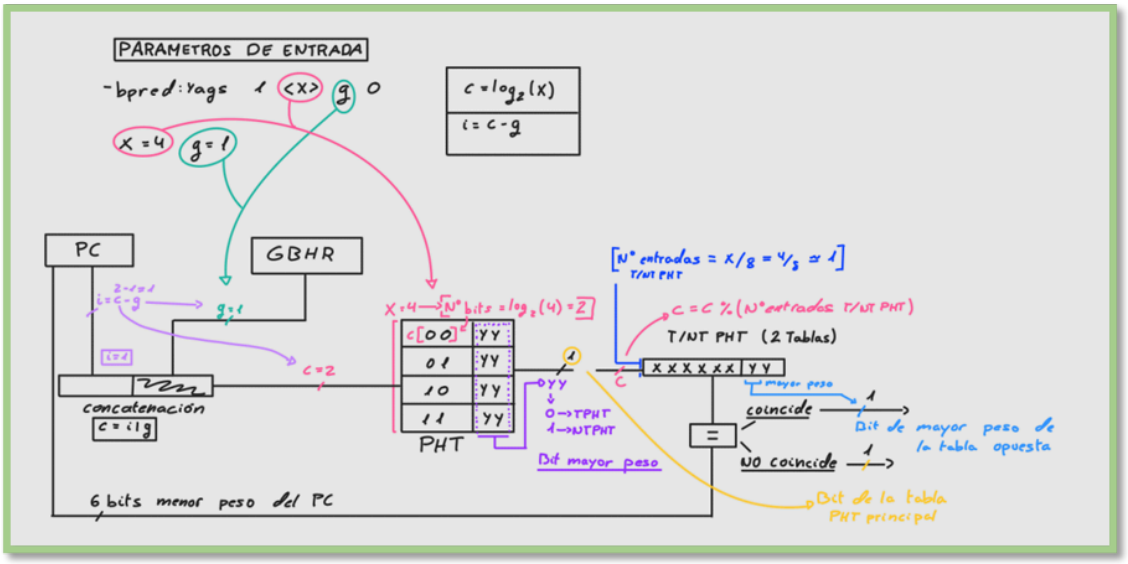
Per començar entendrem com funcionen les estructures que ens demanen:

1. Un GBHR guardarà els últims 'g' salts sent 0 (Not-Taken) i 1 (Taken). Internament serà un registre de 32 bits on obtindrem els 'g' bits de menor pes.
2. PHT Taula on emmagatzemarem un comptador de 2 bits, mirant sempre el bit de major pes. I el qual s'anirà actualitzant si és taken o not taken.
3. TakenPHT i NotTakenPHT, dues taules que emmagatzemen el comportament dels salts que realment han saltat o no. S'usarà només una de les taules, aquella que és contrària a la predicció del PHT. És a dir, la TakenPHT si el PHT surt NT i viceversa.

   Tenen dos camps per entendre: TAG (6 bits baixos del PC) i 2BitCounter (Comptador).

   Farem cas a la predicció d'aquesta taula si coincideixen els tags.

Funcionament de la estructura:



En aquest diagrama podem veure l'estructura més el càlculs per poder crear la taula i saber com accedir a cadascuna.

Per començar tindrem els paràmetres "-bpred:yags 1 <X> g 0" on X serà en número d'entrades de la PHT principal [4, 16, 64, 256, 1024] i g seran el número de bits que mirarem del registre de 32 bits del GBHR [1, 2, 3, 4, 5].

Amb aquests valors haurem de treure la resta d'informació i crear les taules i màscares corresponents.

Per començar haurem d'obtenir el les màscares de bits de 'c', 'g', i 'i'.

- g → Per treure aquest valor tindrem que moure un 1 tants bits a la dreta com el valor que ens arribi per paràmetre g i restar 1.
  (ex/ g = 5 → 00000001 << 5 → 00100000 - 1 → 00011111)
- c → Aquest valor serà la quantitat de bits que necessitarem per accedir a una direcció de la taula PHT. per lo que si tenim una taula de 4 entrades les posición serán 00, 01, 10, 11 per lo que c haura de ser 2. Per lo que l'algoritme será fer el logaritme en base 2 del número d'entrades de la taula PHT que será el valor X que obtenim per paràmetre.
  Per fer-ho sense afegir la llibreria math.h el que farem será moure el valor que obtenim per X un bit a la dreta fins que sigui 0 augmentant un comptador cada vegada menys 1. Així mirarem la posició del bit amb major pes que ens donarà

el número de bits que necessitarem per asignar una posició de la taula PHT. (ex/ 16 = 0001 0000 → posició 1 = 5 - 1 = 4).

- i → Aquest valor asignarà el bits de meys pes que necessitarem del PC per concatenar-lo amb els bits de g i obtenir una posició valida de taula PHT amb l'objectiu d'obtenir el contador saturat.

Un cop obtés els paràmetres que necessitem i obtenim el c per obtenir el contador saturat, mirem el bit de major pes, dictant aquet si fem taken (1) o no (0). Per com aquest predictor és més elaborat en comptes de saltar o no mirarem les taules T/NT-PHT.

Un cop obtés la decisió de salt mirem la taula oposada d'aquest és a dir, si tenim un salt taken (1) haurem de mirar dins de la tauna NotTaken-PHT la posició c d'aquesta taula. Però com el número d'entrades d'aquesta taula és més petit haurem de fer-li el mòdul a c per accedir-hi. Un cop dins mirem si el tag de 6 majors bits coincideix amb el 6 últims bits del PC fem cas al comptador saturat de na nova taula. En cas contrari fem cas al comptador de la PHT principal.

Actualització:

A l'hora d'actualitzar la estructura segons el resultat haurem de primeramente moure la taula GBHR un bit a l'esquerra y si el taken a sigut 1 fer-li una 'or' a l'últim bit d'1.

Un cop actualitzat el valor GBHR cal actualitzar la taula PHT per lo que hem d'obtenir c y augmentar el comptador saturat (si es menor de 3) en cas de taken o disminuir-lo (si major de 0). I fem el mateix amb la direcció de de taules T/NT-PHT.

Bpred.h

```
/* 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,
  BPredYAGS     /* Predictor yags*/ /*MODIFICADO*/
};
```

```
121  /* direction predictor def */
122  struct bpred_dir_t {
123    enum bpred_class class; /* type of predictor */
124    union {
125      struct {
126        unsigned int size;  /* number of entries in direct-mapped table */
127        unsigned char *table; /* prediction state table */
128      } bimod;
129      struct {
130        int l1size;     /* Cantidad de entradas del GBHR */ //SIEMPRE 1
131        int l2size;     /* Numero de entradas del PHT */
132        int shift_width;       /* Numero de bits del GBHR */
133        int xor;        /* Siempre 0 ya que querremos concatenar para ser mas especificos */
134        int *shiftregs;     /* Tabla con numero de entradas del l1size haremos malloc */
135        unsigned char *l2table;   /* Nuemro de entradas del PHT haciendo malloc con l2size */
136        unsigned char *Taken_pht;    /* Dirección del PHT Taken*/
137        unsigned char *NotTaken_pht;  /* Dirección del PHT NOT Taken*/
138        int mascara_i;
139        int mascara_g;
140      } two;
141    } config;
142  };
```

Sim-outorder.c

```
126  /* CREAMOS LAS VARIABLES */
127  static int yags_nelt = 4;
128  static int yags_config[4] = {1, /*Num entradas GBHR*/
129                               1024, /*Num entradas PHT principal y serán 128 TPHT y NTPHT*/
130                               5, /*Longitud GBHR*/
131                               0/*XOR*/};
```

```
677  /*
678   Creamos la llamada para nuestro predictor yags
679   El cual tendrá de entradas
680
681   1 -> Cantidad de filas del GBHR
682   X -> Cantidad de entradas que tendrá en PHT principal y X/8 entradas del TPHT y NTPHT si X es 4 asignar 1.
683        valores [4, 16, 64, 256, 1024]
684   g -> Cantidad de bits que usaremos en el GBHR 1, 2, 3, 4, 5
685   0 -> No queremos XOR ya que concatenamos
686
687  */
688  opt_reg_int_list(odb, "-bpred:yags",
689                   "2-level predictor config "
690      "(1 <Entradas PHT X> <Longitud GBHR g> 0)",
691                   yags_config, yags_nelt, &yags_nelt,
692      /* default */yags_config,
693                   /* print */TRUE, /* format */NULL, /* !accrue */FALSE);
694
```

```
980    else if (!mystricmp(pred_type, "yags"))
981      {
982        /* 2-level adaptive predictor, bpred_create() checks args */
983        if (yags_nelt != 4)
984    fatal("bad 2-level pred config (<l1size> <l2size> <hist_size> <xor>)");
985        if (btb_nelt != 2)
986    fatal("bad btb config (<num_sets> <associativity>)");
987
988        pred = bpred_create(BPredYAGS,
989          /* bimod table size */0,
990          /* 2lev l1 size */yags_config[0],
991          /* 2lev l2 size */yags_config[1],
992          /* meta table size */0,
993          /* history reg size */yags_config[2],
994          /* history xor address */yags_config[3],
995          /* btb sets */btb_config[0],
996          /* btb assoc */btb_config[1],
997          /* ret-addr stack size */ras_size);
998      }
```

Bpred.c

bpred_create i bpred_dir_crete

```
116     //Creamos el caso del BPredCreate sabemos que l1size siempre será 1 y el xor será 0
117   case BPredYAGS:
118     pred->dirpred.twolev = bpred_dir_create(class, l1size, l2size, shift_width, xor);
119     break;
```

```
125   /* allocate ret-addr stack */
126   switch (class) {
127   case BPredYAGS: //añadimos nuestro caso
128   case BPredComb:
129   case BPred2Level:
130   case BPred2bit:
131     {
```

```
265   case BPredYAGS:
266
267     /*
268       Aqui tenemos que crear la estructura guardando espacios de memoria,
269       y inicializar los valores de las tablas, los cuales serán 1 para empezar siempe con taken.
270
271       Pasamos la estructura pred_dir y los parametros de entrada.
272     */
273     predictor_yags(pred_dir, l1size, l2size, shift_width, xor);
274
275     break;
```

```c
static void predictor_yags(struct bpred_dir_t *pred_dir, unsigned int l1size, unsigned int l2size, unsigned int shift_width, unsigned int xor){
  /*
    Para evitar errores de entrada filtramos todos los parametros
    l1size -> Numero de entradas del GBHR (siempre 1)
    l2size -> Numero de entradas del PHT principal [4, 16, 64, 256, 1024]
    shift_width -> Longitud del GBHR [1, 2, 3, 4, 5]
    xor -> Condición de XOR o Concatenación (siempre 0)
  */

  if(l1size != 1)
    fatal("El primer valor de entrada l1size '%d' tiene que ser 1.", l1size);
  if(l2size != 4 && l2size != 16 && l2size != 64 && l2size != 256 && l2size != 1024)
    fatal("El segundo valor de entrada l2size '%d' tiene que ser [4, 16, 64, 256, 1024]", l2size);
  if(shift_width < 1 || shift_width > 5)
    fatal("El tercer valor de entrada shift_width '%d' tiene que estar entre 1 y 5.", shift_width);
  if(xor != 0)
    fatal("El cuarto valor de entrada xor '%d' tiene que ser 0.", xor);

  // Inicializamos los datos sobre la estructura
  pred_dir->config.two.l1size = l1size;
  pred_dir->config.two.l2size = l2size;
  pred_dir->config.two.shift_width = shift_width;
  pred_dir->config.two.xor = xor;

  // Creamos la tabla de GBHR con 32 bits
  pred_dir->config.two.shiftregs = calloc(l1size, sizeof(int));
  if(!pred_dir->config.two.shiftregs)
    fatal("No se puede distribuir en memoria la tabla GBHR");
      // Inicializamos la tabla GBHR poniendo el número de bits shift_width a 1
  int bits_gbhr_inicializados = (1 << shift_width) - 1;
  pred_dir->config.two.shiftregs[0] = bits_gbhr_inicializados;

  // Creamos la tabla de PHT
  pred_dir->config.two.l2table = calloc(l2size, sizeof(unsigned char));
  if(!pred_dir->config.two.l2table)
    fatal("No se puede distribuir en memoria la tabla PHT");
      // Inicializamos los valores a 1
  for(int i = 0; i < l2size; i++){
    pred_dir->config.two.l2table[i] = 0x3;
  }

  // Creamos las tablas TakenPHT NotTakenPHT

      // El tamaño de esta tabla será (l2size / 8)
  if(l2size == 4)
    pred_dir->config.two.size_pht2 = 1;
  else
    pred_dir->config.two.size_pht2 = l2size >> 3;
      // Asignamos el tamaño requerido
  pred_dir->config.two.Taken_pht = calloc(pred_dir->config.two.size_pht2, sizeof(unsigned char));
  pred_dir->config.two.NotTaken_pht = calloc(pred_dir->config.two.size_pht2, sizeof(unsigned char));

  if(!pred_dir->config.two.Taken_pht)
    fatal("No se puede distribuir en memoria la tabla TakenPHT");

  if(!pred_dir->config.two.NotTaken_pht)
    fatal("No se puede distribuir en memeoria la tabla NotTakenPHT");
      // Inicializamos los 8 bits a 1 (6 tags || 2 contador)
  for(int i = 0; i < pred_dir->config.two.size_pht2; i++){
    pred_dir->config.two.Taken_pht[i] = 0xFF;
    pred_dir->config.two.NotTaken_pht[i] = 0xFF;
  }

  // Tablas inicializadas toca calcular los bits de 'c', 'g' y 'i'
  /*
  c -> log2(l2size)
  g -> shift_width
  i -> i = c -g (ya que c será la concatenación de i||g)
  */

      // Se ha calculado la mascara anteriormente cuando inicializamos
      // los bits correspondientes del shiftregs[0] a 1.
  pred_dir->config.two.mascara_g = bits_gbhr_inicializados;

      // Calculamos c haciedo el logaritmo en base 2,
      // haciendo lsr de l2size hasta ser 0 (posición del bit más alto)
  int contador_log = l2size;
  int c = 0;
  while(contador_log > 1){
    contador_log >>= 1;
    c++;
  }

      // Ahora con los valores los restamos y creamos la mascara de i
  int valor_i = c - shift_width;
  pred_dir->config.two.mascara_i = (1 << valor_i) - 1;

}
```

Config_dir_create

```
393     // Generamos los parametros de salida tras la configuración
394   case BPredYAGS:
395     fprintf("pred_dir: %s, Entradas del PHT: %d, Entradas del T/NT PHT: %d, longitud GBHR, %d, %s XOR",
396     name,
397     pred_dir->config.two.l2size,
398     pred_dir->config.two.size_pht2,
399     pred_dir->config.two.shift_width,
400     pred_dir->config.two.xor ? "" : "no");
401   break;
```

Bpred_config

```
438     case BPredYAGS:
439       bpred_dir_config (pred->dirpred.twolev, "yags", stream);
440       fprintf(stream, "btb: %d sets x %d associativity",
441         pred->btb.sets, pred->btb.assoc);
442       fprintf(stream, "ret_stack: %d entries", pred->retstack.size);
443       break;
```

Bpred_reg_stats

```
491       case BPredYAGS: // Configuramos su nombre
492         name = "bpred_yags";
493         break;
```

Bpred_dir_lookup

```
666
667       // Creamos nuestro algoritmo de predicción en base al manual, pasando por parametro
668       // la estructura creada y la dirección del salto (PC).
669     case BPredYAGS:
670       p = algoritmo_prediccion_yags(pred_dir, baddr);
671     break;
```

```c
static char *algoritmo_prediccion_yags(struct bpred_dir_t *pred_dir, md_addr_t baddr){

  // Declaramos los valores para trabajar con los bits de entrada del PC, GBHR para obtener la posición del PHT
  int bits_g = pred_dir->config.two.shiftregs[0] & pred_dir->config.two.mascara_g;
  int bits_i = baddr & pred_dir->config.two.mascara_i;

    // Con los valores i y g los concatenamos para obtener la posición de entrada de la tabla PHT
  int bits_c = (bits_i << pred_dir->config.two.shift_width) | bits_g;

  // Declaramos información para las predicciones
  char contador_pht, direccion, tag_pht;

  // Creamos el valor de salida
  unsigned char *p = NULL;

  // Nos guradamos los 6 bits de menor peso para mirar si coincide con el tag de las tablas TPHT y NTPHT.
  int tag = baddr & 0x3F;


  contador_pht = pred_dir->config.two.l2table[bits_c];
  // Guardamos directamente el resultado del PHT
  pred_dir->config.two.contador[0] = contador_pht & 0x3;
  p = &(pred_dir->config.two.contador[0]);
  contador_pht = (contador_pht & 0x3) >> 1;

  if(contador_pht){
    /*
      Si el bit de mayor peso del contador es 1 tenemos que irnos a la tabla NotTakenPHT
      y obtener el tag almacenado del PHT 'tag_pht' y compararlo con los 6 ulitmos bits del PC 'tag'
    */
    direccion = pred_dir->config.two.NotTaken_pht[bits_c % pred_dir->config.two.size_pht2];
    tag_pht = (direccion & 0xFC) >> 2;
    /*
      Si entramos quiere decir que existe el tag y obtenemos el salto de ese contador especifico actualizando 'contador_hpt',
      sino tocará hacer el salto con el valor sin modificar del 'contador_pht'
    */
    if(tag_pht == tag){
      pred_dir->config.two.contador[0] = pred_dir->config.two.NotTaken_pht[bits_c % pred_dir->config.two.size_pht2] & 0x3;
      p = &(pred_dir->config.two.contador[0]);
      return p;
    }
  }else{
    /*
      Si el bit de mayor peso del contador es 0 tenemos que irnos a la tabla TakenPHT
      y obtener el tag almacenado del PHT 'tag_pht' y compararlo con los 6 ulitmos bits del PC 'tag'
    */
    direccion = pred_dir->config.two.Taken_pht[bits_c % pred_dir->config.two.size_pht2];
    tag_pht = (direccion & 0xFC) >> 2;

    /*
      Si entramos quiere decir que existe el tag y obtenemos el salto de ese contador especifico actualizando 'contador_hpt',
      sino tocará hacer el salto con el valor sin modificar del 'contador_pht'
    */
    if(tag_pht == tag){
      pred_dir->config.two.contador[0] = pred_dir->config.two.Taken_pht[bits_c % pred_dir->config.two.size_pht2] & 0x3;
      p = &(pred_dir->config.two.contador[0]);
      return p;
    }
  }

  // En caso de haber accedido a una de las tablas y no haber encontrado el tag devolvemos el salto origianl del PHT
  return p;

}
```

bpred_lookup

```c
814       case BPredYAGS:
815         if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND))
816       {
817         dir_update_ptr->pdir1 =
818           bpred_dir_lookup (pred->dirpred.twolev, baddr);
819       }
820         break;
```

bpred_update

```
/* Crear la actualización de toda la estructura del YAGS */

if ((MD_OP_FLAGS(op) & (F_CTRL|F_UNCOND)) != (F_CTRL|F_UNCOND) &&
  (pred->class == BPredYAGS))
{
  // tenemos que mover los bits del GBHR y añadir el resultado
  pred->dirpred.twolev->config.two.shiftregs[0] << 1;

  //miramos si el salto ha sido taken o no y lo actualizamos en el GBHR
  if(taken)
    pred->dirpred.twolev->config.two.shiftregs[0] | 1;

  //Obtenemos los 6 bits de menor peso de la dirección
  int direccion = baddr & 0x3F;

  // Actualizamos el contador saturado de la tabla PHT principal
  int bits_g = pred->dirpred.twolev->config.two.shiftregs[0] & pred->dirpred.twolev->config.two.mascara_g;
  int bits_i = baddr & pred->dirpred.twolev->config.two.mascara_i;
  int bits_c = (bits_i << pred->dirpred.twolev->config.two.shift_width) | bits_g;

  int contador = pred->dirpred.twolev->config.two.l2table[bits_c] & 0x3;
  if(taken && ( contador < 3 ))
    contador++;

  if(!taken && ( contador > 0 ))
    contador--;
      // Valor actualizado
  pred->dirpred.twolev->config.two.l2table[bits_c] = contador;
  // Actualizamos los contadores de las tablas T/NT PHT y añadimos la dirección.
  if(taken){
    contador = pred->dirpred.twolev->config.two.Taken_pht[bits_c % pred->dirpred.twolev->config.two.size_pht2] & 0x3;
    if(contador < 3)
      contador++;

    contador |= (direccion << 2);
    pred->dirpred.twolev->config.two.Taken_pht[bits_c % pred->dirpred.twolev->config.two.size_pht2] = contador;
  }
  else{
    contador = pred->dirpred.twolev->config.two.NotTaken_pht[bits_c % pred->dirpred.twolev->config.two.size_pht2] & 0x3;
    if(contador > 0)
      contador--;

    contador |= (direccion << 2);
    pred->dirpred.twolev->config.two.NotTaken_pht[bits_c % pred->dirpred.twolev->config.two.size_pht2] = contador;
  }

}
```

Resultats:

Un cop tenint el codi hem de testejar-ho, amb els paràmetres que ens especifiquen al PDF:

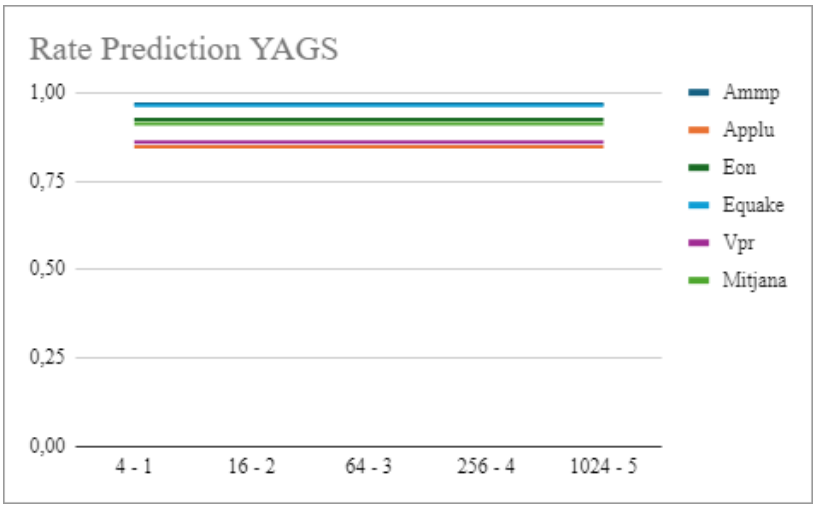
Yags: opció -bpred yags

- Mida del GBHR 1 i del PHT<l2-size> es crearan tres taules de mida: 4, 16, 64, 256 i 1024 per a la PHT principal i per a les T/NotTakenPHT seran de 1,2,8,32 i 128 entrades respectivament. Paràmetre X.
- Ample del GBHR <g> que seran: 1, 2, 3, 4 i 5.
- Així la configuració de c=i+g quedaria: (1+1→ 2), (2+2→ 4), (3+3→ 6), (4+4→ 8), (5 +5→10).
- -bpred:yags 1 <X> <g> 0

Que el que ens interessa és el "-bpred:yags 1 <X> <g> 0"

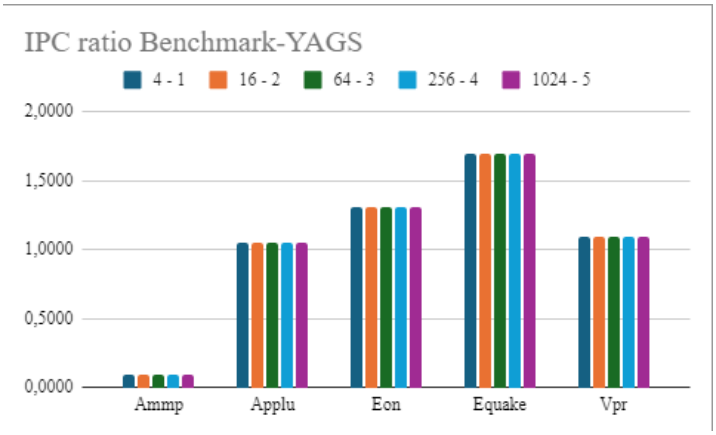
Els resultats obtinguts han sigut els següents:

	Yags	.bpred_dir_rate			
	4 - 1	*16 - 2*	*64 - 3*	*256 - 4*	*1024 - 5*
Ammp	0,9669	0,9669	0,9669	0,9669	0,9669
Applu	0,8463	0,8463	0,8463	0,8463	0,8463
Eon	0,9232	0,9232	0,9232	0,9232	0,9232
Equake	0,9639	0,9639	0,9639	0,9639	0,9639
Vpr	0,8598	0,8598	0,8598	0,8598	0,8598
Mitjana	0,91202	0,91202	0,91202	0,91202	0,91202



Com podem observar el percentatge de predicció ha estat molt alt en tots els casos sent el màxim un 96.69% d'encerts en el ammp i el més baix un 84.63% en el applu.

Però hem de observar que no varien els percentatges entre un mateix benchmark i el paràmetres afegits. Per això mirarem per si decás els IPC de cadascun per veure si varien o no.



Aquí corroborem que els IPC tampoc varien l'única informació notable es que el IPC del ammp es quasi 0 sent (0.0969).

Per lo que he afegit un segon script a per veure si els paràmetres que tracta el predictor son els que passem o uns altres.

```
Resultados/ammp_1024_5.txt:-bpred:yags      1 1024 5 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/ammp_16_2.txt:-bpred:yags      1 16 2 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/ammp_256_4.txt:-bpred:yags      1 256 4 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/ammp_4_1.txt:-bpred:yags      1 4 1 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/ammp_64_3.txt:-bpred:yags      1 64 3 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/applu_1024_5.txt:-bpred:yags      1 1024 5 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/applu_16_2.txt:-bpred:yags      1 16 2 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/applu_256_4.txt:-bpred:yags      1 256 4 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/applu_4_1.txt:-bpred:yags      1 4 1 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/applu_64_3.txt:-bpred:yags      1 64 3 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/eon_1024_5.txt:-bpred:yags      1 1024 5 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/eon_16_2.txt:-bpred:yags      1 16 2 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/eon_256_4.txt:-bpred:yags      1 256 4 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/eon_4_1.txt:-bpred:yags      1 4 1 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/eon_64_3.txt:-bpred:yags      1 64 3 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/equake_1024_5.txt:-bpred:yags      1 1024 5 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/equake_16_2.txt:-bpred:yags      1 16 2 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/equake_4_1.txt:-bpred:yags      1 4 1 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/equake_64_3.txt:-bpred:yags      1 64 3 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/vpr_1024_5.txt:-bpred:yags      1 1024 5 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/vpr_16_2.txt:-bpred:yags      1 16 2 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/vpr_256_4.txt:-bpred:yags      1 256 4 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/vpr_4_1.txt:-bpred:yags      1 4 1 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
Resultados/vpr_64_3.txt:-bpred:yags      1 64 3 0 # 2-level predictor config (1 <Entradas PHT X> <Longitud GBHR g> 0)
```

Però veiem que no ens tracta els mateixos paràmetres sinó que realment fa execució amb els que li enviem sent tots diferents. Per últim per intentar veure si podem arribar a fer que varïn ha sigut afegir a la crida del predictor el troç “-bpred yags” radere de “-bpred:yags 1 X g 0”. Pero ens apareixia un error : “Violación de segmento”, cosa que no hem arribat a trobar una solució.

Hem provat a afegir “-bpred 2lev” i en aquest cas si executaba però seguia donant els mateixos números per benchmark, per lo que hem repassat tot el codi per que funcioni amb el yags encara que donin els mateixos percentatges, però no hem trobat cap diferència i segueix donant el mateix error ja mencionat en el paràgraf anterior per lo que hem decidit descarta-ho i deixar-ho com està, tenint en compte que hauria de fer una minima millora lineal i creixent a mida que se li afegeixen més paràmetres sobre l’estructura fins arribar a un punt mig adequat amb l'eficiència.

Observacions i Conclusions sobre el predictor:

El esquema YAGS (Yet Another Global Scheme) està dissenyat específicament per a reduir el problema del aliasing en el Pattern History Table (PHT), on dues instruccions poden accedir a la mateixa posició a causa de col·lisions. Aquest predictor introdueix la innovadora incorporació de tags en les dues taules PHT (Taken PHT i Not Taken PHT), la qual cosa permet una gestió més precisa de les prediccions. Els tags identifiquen de manera única les instàncies dels condicionals, eliminant aliasing entre condicionals consecutius i reduint la quantitat d'entrades necessàries sense comprometre la informació clau sobre els resultats dels condicionals.

A partir de les nostres proves, hem constatat que, encara que YAGS introdueix una major complexitat estructural, aquest increment es justifica àmpliament per la millora en el rendiment. La reducció de aliasing i l'optimització de l'emmagatzematge permeten un percentatge significativament superior de prediccions de salt correctes, la qual cosa reforça la seva eficàcia com a predictor avançat. En particular, la capacitat d'emmagatzemar únicament excepcions a les tendències dels condicionals resulta en un ús eficient dels recursos, compensant amb escreix el cost addicional de les etiquetes.

En conclusió, els resultats obtinguts mostren que YAGS és una solució robusta i eficient, capaç d'equilibrar la complexitat tècnica amb un notable increment en la precisió de les prediccions. Això ho converteix en un enfocament prometedor per a arquitectures modernes que demanden predictors d'alt rendiment.

Comparació amb un altre Predictor de salt

Per a començar durant la pràctica trobem un document oficial sobre YAGS i la seva implementació on esmenta que la seva creació ve en base de parts específiques que caracteritzen a altres predictors. Entre els quals podem trobar que esmenten el Gshare, ja que ells unifiquen el YAGS a través d'una XOR.

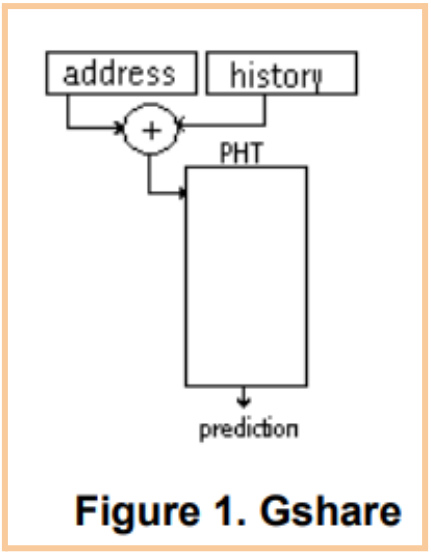


Figure 1. Gshare

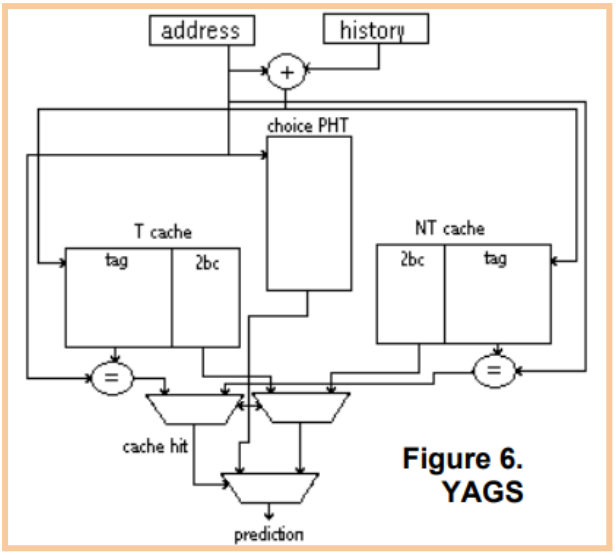
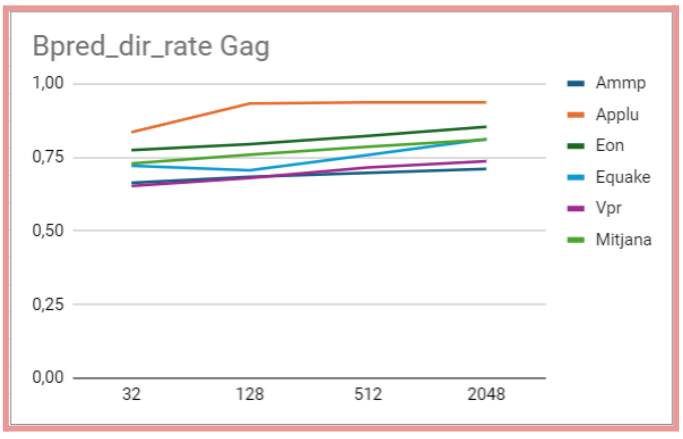
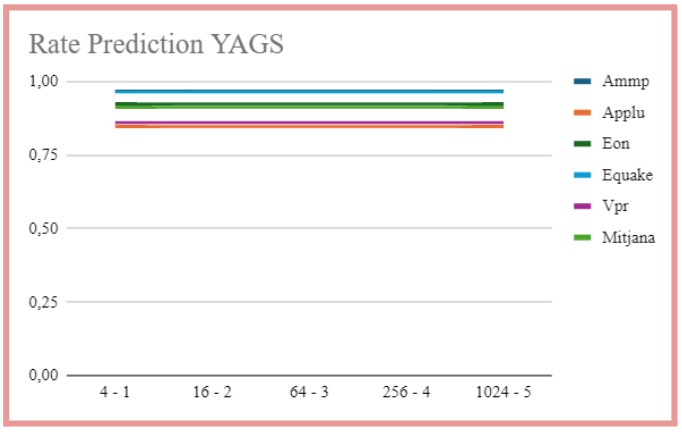


Figure 6. YAGS

Enllaç del document:
https://people.eecs.berkeley.edu/~kubitron/courses/cs252-F99/handouts/papers/mudge_yags.pdf

Però com nosaltres en el YAGS usem concatenació i en Gshare fem XOR, ho compararem pel que ho compararem amb el Gag (Gselect) que és la mateixa estructura però concatenant, que és el que necessitem.





Aquí tenim la comparació del percentatge d'encerts entre el Gag i el YAGS. Fàcilment podem veure que entre aquests dos predictors el YAGS és bastant superior en tots els benchmarks.

Respecte als paràmetres d'entrada hem decidit mantenir els de les proves ja realitzades ja que en YAGS es caracteritza per tenir un GBHR més reduït i centrar-se més en les 3 taules PHT i el Gag té una grandària de GBHR molt més extens i un únic PHT pel que encara així podem veure com el YAGS soluciona els problemes de aliasing que té el Gag aportant millors resultats.