

Two-Level Address Storage and Address Prediction

Enric Morancho, José María Llabería and Àngel Olivé

Abstract—The amount of information recorded in the prediction tables of the proposed address predictors turns out to be comparable to the current on-chip cache sizes; for instance, a last-address predictor that records 4.096 64-bit effective addresses uses a 32 Kbytes storage space.

Area cost of address predictors is proportional to address width. To reduce their area cost, we will take advantage of the spatial-locality property of the memory references, that is, the high-order bits of several effective addresses recorded in the prediction table are equal. Then, the effective addresses are split in two parts: the high-order bits and the low-order bits. The High-Address Table records the high-order bits and the Low-Address Table records the low-order bits and a link between tables. This organization allows addresses with the same high-order bits to share one entry in the High-Address Table.

We use this organization in a last-address predictor and our evaluations show that it produces significant area-cost reductions (28%-60%) without performance losses.

Keywords—address prediction, address locality, area cost

I. INTRODUCTION

A characteristic of the current tendencies in processor design is the replication of resources (functional units, execution pipelines,...). To take full advantage of these resources, the processor must exploit the instruction-level parallelism (ILP) available in the programs, but the capacity of exploiting the ILP is limited by the dependencies between operations.

Dependencies can be classified into three classes: control, name and true-data dependencies. Lot of work have been performed to reduce the restrictions imposed by control [15] and name dependencies [11], but true-data dependencies remain as the major bottleneck for exploiting ILP. Recently, some works [10][13][22] have proposed the use of techniques that have been successfully applied to overcoming control dependencies, to also overcoming true-data dependencies. These techniques are based in the prediction and the speculative execution of the operations. They predict the result of an operation prior to execute it, and the prediction is used to issue speculatively its dependent operations. On a right prediction, the effective latency of the predicted operation can be reduced in several processor cycles; on a misprediction, the dependent operations must be flushed out and reissued to recover the correct processor state.

Usually, two different instructions are involved in a true-data dependence, but there also exist true-data dependencies between the basic operations that compose the execution of a single instruction. For instance, in the load instructions, there is a true-data dependence between the computation of the effective address and the

memory access. This dependence contributes to the large latency of the load instructions and can affect the processor performance. Then, some works have proposed the use of prediction methods to reduce the negative impact of this true-data dependence. They predict the effective address that will be computed by the load instructions and they access memory speculatively.

Address predictors use on-chip prediction tables that record up to 4.096 64-bit effective addresses [5][19], that is, 32 Kbytes for a last-address predictor; that is comparable to the current on-chip cache sizes. However, the previous designs do not exploit the locality of the addresses. In this paper, we will take advantage of the spatial-locality property of the addresses to reduce the area cost of the predictors. The addresses are split in two parts and they are recorded in two different tables connected by a link. The High-Address Table (HAT) records the high-order bits and the Low-Address Table (LAT) records the low-order bits and a link between both tables; we name this organization *Two-Level Address Storage* (TLAS). TLAS allows addresses with the same high-order bits to share one HAT entry.

We will use the TLAS organization in a typical last-address predictor and in an enhanced address predictor (the *Looking-Backward Predictor* [17]) and we will show that the performance of both predictors is not affected and, moreover, we obtain a significant area-cost reduction. Also, TLAS can be used in value prediction and in more complex prediction models as stride based[2], context based [22] or hybrid ones [5].

This paper is organized as follows. Section II shows the prediction model used in this work and a typical address predictor proposed in the literature: the *Base Last-Address Predictor*. Section III presents an evaluation of the locality of the addresses recorded by the *Base Last-Address Predictor* (BP). Section IV introduces a BP that uses the *Two-Level Address-Storage* (TLAS) organization; moreover it describes some implementation issues. Section V shows that the previous predictor reduces the area-cost of the BP without affecting its performance. Section VI evaluates the inclusion of TLAS in an enhanced last-address predictor. Finally, the conclusions of this work are summarized in Section VII.

Related works

The spatial-locality property of the memory references had been used to reduce the address-bus width [8]. More recently, it has also been used to reduce the energy consumed by the address bus [18].

Works closer to the scope of this paper propose an organization that reduces the area cost of address tags in caches [23][25]. These organizations are similar to TLAS but our proposal has a simpler control logic. It does not need to maintain the exact correspondence between both tables because it will be applied to a predictor, then a recovery mechanism on mispredictions is yet supplied.

Rychlick et al. [21] propose the Popular Last-Value Predictor that exploits the temporal locality in the results produced by register-writing instructions; they report results for 4.096-entry LAT's. Differences between our work and Rychlick's work arise in LAT mapping and management of the HAT entries. We take explicit trace of unused entries in HAT to reduce capacity misses.

Another difference is that we add a filtering ability to reduce capacity misses in HAT: we review a filtering idea presented in [16], and extend it to reduce pollution in HAT by filtering out unpredictable load instructions. Also, in this paper we evaluate the temporal and the spatial locality of the addresses computed by predictable load instructions for a large range of high-order bits.

A similar organization to TLAS is proposed in [24] to share page-number information between several processor devices. Then, replacement in shared tables is guided for devices that share it.

Another approach to reduce area cost is presented by Bekerman et al. [3]. They have observed that the addresses computed by load instructions related to Recursive Data Structures (RDS) only differ by some constants. Then, they introduce the idea of global correlation and record in the prediction table base addresses, defined as the real address minus the immediate offset as specified in the load-instruction operation code. This organization allows all loads associated to the same RDS to share the same prediction-table entry. Furthermore area-cost reduction can be obtained applying TLAS.

II. LAST-ADDRESS PREDICTION MODEL

The last-address predictor [22] predicts the effective addresses that will be computed by the load instructions. It performs a trivial computational operation: the predicted address is equal to the previous address computed by the same load instruction. These previous addresses are recorded in a prediction table where each load instruction is related to a prediction-table entry.

Mispredictions could have a penalty of some processor cycles. To reduce the amount of mispredictions, the addition of a two-bit saturated counter to every prediction-table entry has been proposed [10][13][14]; in this paper, these counters will be named confidence counters. Every time a load instruction computes the same address than the one computed in its previous execution, its confidence-counter value is increased by one, otherwise it is decreased by one. A load instruction

is predicted only when its confidence-counter value is greater than one. To surpass this threshold value, the load instruction must show during some executions that it is predictable by the last-address prediction model. Consequently, some executions that would be correctly predicted are not predicted. This is a conservative assumption about the load-instruction behaviour. So, confidence counters represent a trade-off between the number of right predictions and the amount of mispredictions performed by the predictor [16].

Proposed implementations of address predictors based in the last-address prediction model [10][13] employ a direct-mapped table, named Address Table (AT), indexed with the least-significant bits of the PC. Every AT entry contains the last address computed by the most-recently executed load instruction mapped to the entry, a two-bit confidence-counter value, and a tag used to detect mapping conflicts in the AT entries. Load instructions are allocated in the AT using the *always allocate* policy; on a miss, the load instruction is not predicted, the address field is updated and its confidence-counter value is initialized to one. This initialization represents a trade-off between the number of mispredictions and the number of correct predictions performed by the predictor. Figure 1 shows the scheme and the pseudo-code of this predictor. In this paper, it will be named *Base Last-Address Predictor* (BP). To evaluate the performance of an address predictor, we use two measures: the address predictability captured by the predictor and the accuracy of the predictor. The address predictability is defined as the percent of correct predictions out of the number of executed load instructions. The accuracy is defined as the percent of correct predictions out of the number of predictions performed. Table I shows the address predictability and the accuracy obtained using an *BP* with an unbounded AT in the integer benchmarks of the spec-95 benchmark suite. In this work we have focused on integer codes because they are more sensitive to the effective latency of load instructions than floating-point codes [11].

Binaries used in this work have been obtained compiling with the `-O4` switch of the `cc` native compiler of the machine (an Alpha 21164 processor, with OSF1 V3.2). Then, they have been instrumented with ATOM (only user-level code) to evaluate the performance of the predictors. Benchmarks were run until completion, and loads to zeroed registers have been ignored.

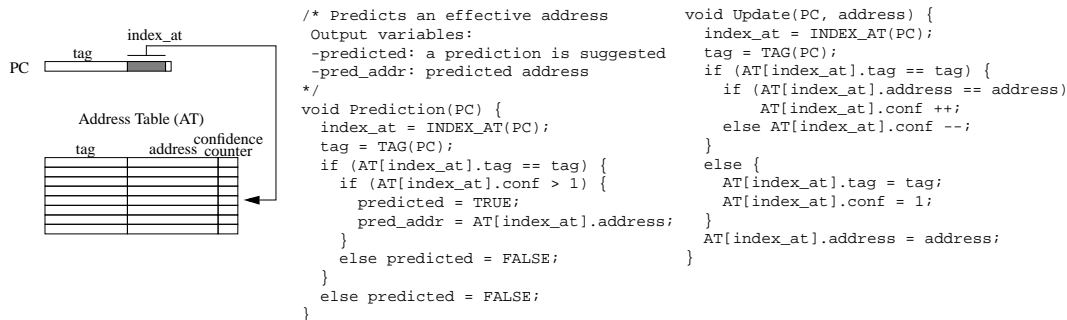


Fig. 1. Base Last-Address Predictor scheme and pseudo-code (++ and -- operators update counters in a saturated way).

Table I shows that the BP predicts correctly a significant percent of addresses computed by load instructions: between 19% and 79%. Moreover, the accuracy of the BP is between 92% and almost 100%.

TABLE I Benchmark description, input data set, number of static load instructions (present in the binary file and executed at least once), number of dynamic load instructions, and performance (maximum address predictability and accuracy) obtained using the *Base Last-Address Predictor* with an unbounded AT. Measures taken on an Alpha processor.

Bench.	Description	Input	Static Loads present/exec. ($\times 10^3$)	Dyn. loads ($\times 10^9$)	Last addr. pred..	Accur.
<i>go</i>	GO game player	Ref.	21 / 16.3	8.5	47.81	92.40
<i>mksim</i>	Processor simulator	Ref.	13 / 3.7	19.3	69.53	96.89
<i>gcc</i>	C compiler	cp-decl.i	97 / 21.3	0.1	60.11	95.03
<i>compress</i>	Data compressor	Ref/	4 / 0.7	12.5	57.27	99.72
<i>perl</i>	Perl interpreter	primes.pl	43 / 5.1	3.0	79.19	97.61
<i>vortex</i>	Database program	Ref.	43 / 19.5	22.8	56.28	93.90
<i>li</i>	Lisp interpreter	Ref.	7.7 / 2.4	18.6	30.20	94.11
<i>jpeg</i>	JPEG encoder	Ref.	15 / 3.9	7.1	19.11	93.14

III. EFFECTIVE-ADDRESS LOCALITY ANALYSIS

Many works have analysed a property of memory references: the locality. It has been defined as the program's tendency to reference memory in non-uniform, highly localized patterns [4]. Temporal locality and spatial locality are the main classes of locality present in memory references. Temporal locality is the tendency to reference a memory location that had been referenced in the past; spatial locality is the tendency to reference a memory location that is close to another memory location referenced in past.

The addresses computed by the load instructions exhibit both kinds of locality. This fact can produce a certain degree of redundancy in the contents of the AT:

a) Temporal redundancy: As the BP assigns an AT entry to every static load instruction, load instructions that compute the same address record it redundantly at several AT entries. For instance, the accesses to the same global variable from different routines, and the stack accesses performed by routines called from the same stack height produce this effect.

b) Spatial redundancy: Some addresses recorded in AT will be close then, in most cases, only will differ in their low-order bits. That produces redundancy because some addresses recorded in the AT have the same value in their high-order bits. For instance, global variables stored in consecutive addresses and stack accesses produce this effect.

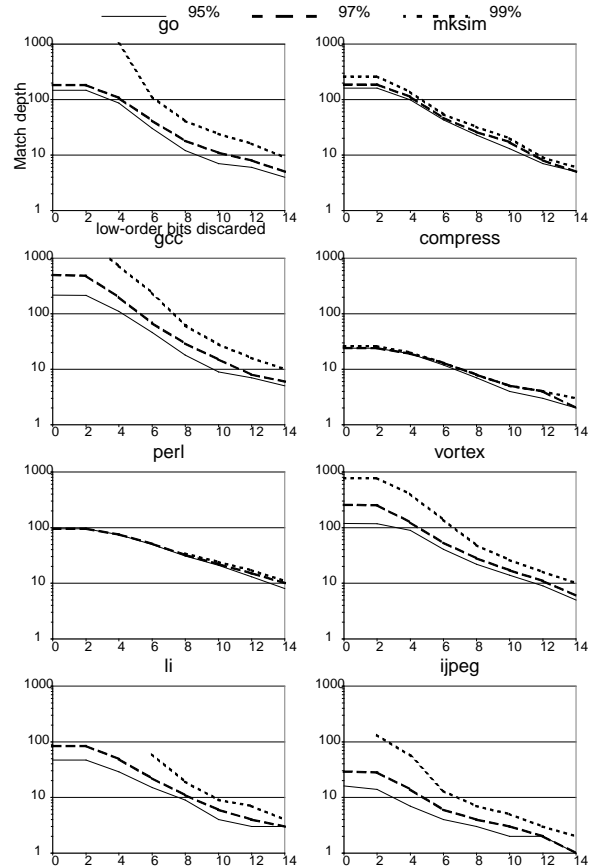
Locality analysis

We will present an analysis of the locality of the addresses recorded in AT. To perform this analysis, the last reference time of every address is used to maintain a temporal ordering on a stack of slots; every slot records an address and an address is recorded in only one slot. When an address recorded in the stack is not the last address computed by any load instruction, its slot becomes an empty slot. Then, if an address is not found in the stack, the address is recorded in the most recently empty slot and it is moved to the top of the stack.

We define the match depth of the Most Recently Used

(MRU) address as one, the match depth of the second MRU address as two, and so on. Then, for every dynamic load instruction, we evaluate the match depth of its computed address; if it is not found in the stack, its match depth is infinite.

Many works [9][16][22] have observed that some static load instructions are highly unpredictable. Then, unpredictable load instructions are filtered to not update the stack of slots. To decide if a dynamic load instruction is unpredictable, we will use the confidence-counter value of an unbounded BP: the execution of a load instruction will be classified as predictable if its confidence-counter value is greater than one. We consider that the match depth of the addresses computed by unpredictable load instructions is zero.



2. Dynamic-load-instruction distributions according to their match depth; dynamic load instructions have been filtering out considering their confidence information. Vertical axis stands for the base-10 logarithm of the match depth, horizontal axis stand for the number of low-order bits discarded, and every graph is related with a % of dynamic load instructions.

To analyse the temporal locality of the effective addresses, we must discard zero low-order bits of the addresses, and to evaluate the spatial locality in AT we must discard some low-order bits of them. In Figure , vertical axes stand for the match depth, horizontal axes stand for the number of low-order bits discarded, and every graph is related with a percent of dynamic load instructions. The meaning of every graph is that the addresses referenced by this percentage of dynamic load instructions have a match depth smaller or equal than the vertical height. For instance, in benchmark *compress*, discarding 6 low-order bits of the addresses, the addresses referenced by the 99% of dynamic load

instructions have a match depth ≤ 13 . This value reflects the number of different high-order portions that have been referenced recently by the predictable load instructions.

The low-order bits of the addresses computed by predictable load instructions use to be not significant (Figure) in consequence of the Alpha architecture, because most load-instruction operation codes produce eight-byte aligned addresses and load instructions that are not aligned use to be unpredictable. As the number of low-order bits discarded grows, the match depth related to the percentage of load instructions decreases.

The working-set size of static load instructions¹ is much bigger than the maximum match depth of the 99% of dynamic load instructions. For instance, in benchmark *go*, the working-set size of static load instructions is larger than 2.048 load instructions, but discarding 12 low-order bits of the addresses, 99% of dynamic load instructions have a match depth ≤ 16 .

Results suggest an organization of the predictor where high-order bit portions of addresses are shared between several low-order bits portions in a many-to-one mapping. Then, exploiting the spatial locality in address portions can be valuable to reduce the area cost of predictor.

IV. INCLUSION OF TWO-LEVEL ADDRESS STORAGE IN THE BASE PREDICTOR

In this section, we replace the address field of the BP by the suggested *Two-Level Address Storage* (TLAS) organization; the new predictor will be named *Two-Level Address Predictor* (2LAP). We describe the 2LAP, and perform evaluations of the management of High-Address-Table entries and the filtering-out of some allocations. Also, we evaluate two replacement algorithms.

A. 2LAP design

We have shown that there is a certain degree of redundancy in the information recorded in the AT of the BP. To reduce the redundancy in the prediction table of an address predictor, we use the TLAS organization. The AT is split in two parts (Figure Fig.): the Low-Address Table (LAT) and the High-Address Table (HAT). The low-order bits of the effective addresses will be recorded in the LAT and the high-order bits in the HAT; moreover, each LAT entry has a link to reference one HAT entry. Then, a HAT entry can be shared by several LAT entries: a one-to-many relationship.

To predict a load instruction, the 2LAP indexes LAT using the PC of the load instruction; this access retrieves the low-order bits of the predicted address and a link to HAT. After that, the predictor accesses HAT using the link; this access obtains the high-order bits of the predicted address. The predicted address is formed by

the concatenation of the low-order bits and the high-order bits obtained from both tables.

The 2LAP must access both tables sequentially to predict a load instruction. This does not imply an implementation restriction because the LAT can be accessed very early in the pipeline and, in current processors, the number of pipeline stages before issuing a instruction is so large that both prediction tables can be accessed before issuing the dependent operations; for instance, the Alpha 21264 processor spends 5 pipeline stages before issuing an instruction [1]. Moreover, we could reduce the critical path of the predictor by recording in the LAT enough low-order bits for indexing the cache.

A.1 LAT and HAT management

The 2LAP updates the LAT like the BP updates the AT, that is, using the *always-allocate* policy.

To link a LAT to an HAT entry, the 2LAP applies a hash function to the high-order bits of the computed address and searches them in the HAT. The hash function used depends on the number of HAT entries. Fully associative lookups are possible using a HAT with a small number of entries; as a reference, current microprocessors perform fully associative lookups in up to 96-entry TLB's [12].

To reduce the eviction from HAT of useful information on a HAT miss, the 2LAP searches HAT entries that are related to zero LAT entries (empty HAT entries). To detect them, we will associate to every HAT entry a link counter that reflects the number of LAT entries linked to it. If none empty HAT entry is found, the 2LAP picks randomly a HAT entry but the MRU one and replaces it (no-MRU algorithm). These actions can be performed in parallel before selecting the replaced entry.

On a LAT replacement, or a update due to a change in the high-order portion, the link counter of the previously related HAT entry is decreased by one (could empty the HAT entry). When the LAT entry is linked to a HAT entry, its link counter is updated: it is increased if the high-order portion was yet allocated (HAT hit), and set to one otherwise.

On a replacement in HAT, the LAT entries that were related to the evicted HAT entry are not invalidated because the cost is considerable and require more complex logic. Notice that these incoherencies do not break the program correctness because the predictor belongs to an speculative mechanism; on the other hand, these incoherencies could decrease the performance of the predictor.

In a later section we will show the performance decrease if the detection of empty HAT entries is not considered on HAT replacements. Also we will present the differences between the no-MRU and the LRU policies. Moreover, we will discuss the link-counter width.

A.2 Filtering out HAT allocations

It is not useful to record high-order bits related to unpredictable load instructions because these load instructions will not be predicted. The 2LAP will allocate in LAT these loads but will avoid the allocation in HAT of high-order bits computed by them, that is,

1. We define the working-set size of static load instructions of a benchmark as the minimum two-power number of AT entries needed to achieve, at most, a 1% miss rate in a fully associative AT with LRU (Least Recently Used) replacement policy. The working-set size of the analysed benchmarks is ≤ 128 (*compress*), 256-512 (*li*, *jpeg*, *perl*), 1.024 (*mksim*) and ≥ 2.048 (*go*, *gcc*, *vortex*).

their LAT entries will not be linked to any HAT entry.

To classify the load instructions, the 2LAP will use a small chunk of the effective addresses (it will be recorded in LAT). This idea has been proposed in [16] using the low-order bits of the addresses and also applied in [3] in the context of correlated predictors.

Most load instructions can be classified properly by checking the low-order bits of their effective addresses, but for some load instructions it is not sufficient. For instance, for loads with a stride multiple of 2^b , where b is the number of low-order bits analysed, the classification will be wrong and can produce mispredictions. However, a proper classification can be performed by analysing other bits of the addresses.

Then, in this paper we propose to select dynamically the effective-address chunk that must be recorded in the LAT entry to classify each load instruction. As LAT entries record b bits of the addresses, we divide the addresses in several non-overlapped b -bit chunks. The 2LAP will assume that two addresses are the same if the address-chunk contents are equal, and it will update the confidence counter consequently.

When a load instruction is allocated in LAT, its classification is initialized as unpredictable and the 2LAP uses the low-order bits of its addresses to classify it.

For load instructions classified as predictable, all bits of the address are used to update their confidence counter. However, when the classification of a load instruction changes from predictable to unpredictable, the predictor breaks the link with the HAT entry, decreases its link counter, selects the lower chunk of the computed address that is not equal to the one of the predicted address, and records it in the LAT entry. In next executions, classification is performed by checking the selected address chunk; for this, every LAT entry contains a field that identifies the address chunk recorded in it. When the classification changes to

predictable (the confidence-counter value of the load instruction goes from 1 to 2) the link is established.

In a later section we will show the performance decrease when filtering is not used and we will present the effect of the chunk selection on the accuracy of the 2LAP.

Figure Fig. shows the scheme and pseudo-code of the 2LAP.

B. HAT implementation issues

In this subsection we evaluate different policies in HAT replacement algorithm. For this, unbounded LAT's have been used.

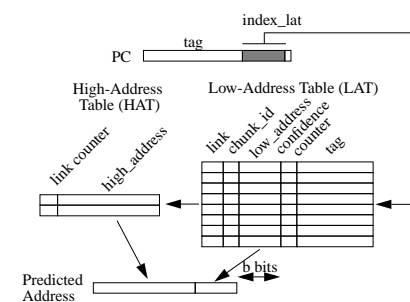
To perform a prediction, HAT is accessed after LAT, close to the fetch stage, and in a later stage it is looked-up and updated. In this paper we consider that access time of large HAT's can be a restriction. Then, we evaluate HAT sizes with an access time close to the one of the register file; we use 16, 32 and 64-entry HAT's. Also, for these sizes, associative lookups can be performed. For these table sizes, Figure suggests the number of low-order bits that should be recorded in the LAT entries (b): 10, 12 and 14. Without HAT access-time restriction, other designs can be considered for bigger HAT's as n-way associative mapping and sequential lookups [26].

First we perform evaluations using the LRU replacement algorithm in HAT. Later, we present performance differences of the LRU versus the no-MRU algorithm.

Also, we show accuracy differences between classifying load instructions checking the low-order bits versus allowing a dynamic chunk selection. On the other hand, differences in captured predictability between both classifiers are negligible.

B.1 Filtering HAT allocations and managing empty HAT entries

To evaluate the effect of managing empty HAT entries



```
void Prediction(PC) {
  index_lat = INDEX_LAT(PC);
  tag = TAG(PC);
  if (LAT[index_lat].tag == tag) {
    if (LAT[index_lat].conf > 1) {
      predicted = TRUE;
      index_hat = LAT[index_lat].link;
      pred_addr = CONCATENATE(
        HAT[index_hat].high_address,
        LAT[index_lat].low_address);
    }
    else predicted = FALSE;
  }
  else predicted = FALSE;
}
```

```
void Update(PC, address, pred_addr, index_lat, index_hat, tag) {
  if (LAT[index_lat].tag == tag) {
    if (LAT[index_lat].conf > 1) {
      if (address == pred_addr)
        LAT[index_lat].conf ++;
      else {
        LAT[index_lat].conf --;
        if (LAT[index_lat].conf == 1) { /* Transition 2 -> 1 */
          LAT[index_lat].chunk_id = INDEX_DIF_CHUNK(address, pred_addr);
          HAT[index_hat].links--;
        }
      }
    }
    else {
      chunk = CHUNK(address, LAT[index_lat].chunk_id);
      if (chunk == LAT[index_lat].low_address)
        LAT[index_lat].conf ++;
      else LAT[index_lat].conf --;
      if (LAT[index_lat].conf == 2) /* Transition 1 -> 2 */
        LAT[index_lat].chunk_id = 0;
    }
  }
  else {
    if (LAT[index_lat].conf > 1) HAT[index_hat].links--;
    LAT[index_lat].tag = tag;
    LAT[index_lat].conf = 1;
    LAT[index_lat].chunk_id = 0; /* stands for the [0, b-1] bit-range chunk */
    LAT[index_lat].low_address = CHUNK(address, LAT[index_lat].chunk_id);
    if (LAT[index_lat].conf > 1) {
      index_hat = INSERT(HAT, HIGH_ADDRESS(address));
      LAT[index_lat].link = index_hat;
    }
  }
}
```

Fig. 3. 2LAP scheme and pseudo-code (++ and -- operators update the counter in a saturated way).

and the effect of filtering out the allocation in HAT of address portions computed by unpredictable load instructions, we have compared four 2LAP designs. We have evaluated the influence of managing empty HAT entries when HAT allocations are not filtered out. Also, we have evaluated the influence of managing empty HAT entries when HAT allocations are filtered out.

Our results show that the management of empty HAT entries increases the predictability captured by a 2LAP. Moreover, without the use of filtering, the performance increase is bigger; that is, HAT allocations are more sensitive to the management of empty HAT entries. This effect is specially noticeable in benchmark *compress*. In *compress*, a load instruction accesses a large hash table (up to 2^{19} bytes); this load instruction is unpredictable and pollutes the HAT with different values. By managing empty HAT entries, the HAT entry related to this load instruction becomes an empty entry, and it is reused to record the new address portion.

Figure Fig. shows the influence of both policies on the predictability captured by the 2LAP in benchmark *compress*. The vertical axis stands for the predictability captured by the predictors, and the horizontal axis shows different predictor configurations and benchmarks. Results for configurations with the same number of low-order bits in LAT entries are grouped; the difference between them is the number of HAT entries. The bottom side of each bar stands for the predictability captured by the 2LAP configuration that does not filter, and the top side stands for the predictability increment due to filtering. We can observe that, without managing empty HAT entries, the influence of filtering for $b=10$ and 16-entry HAT is significant; on the other hand, managing empty HAT entries, its influence is almost insignificant.

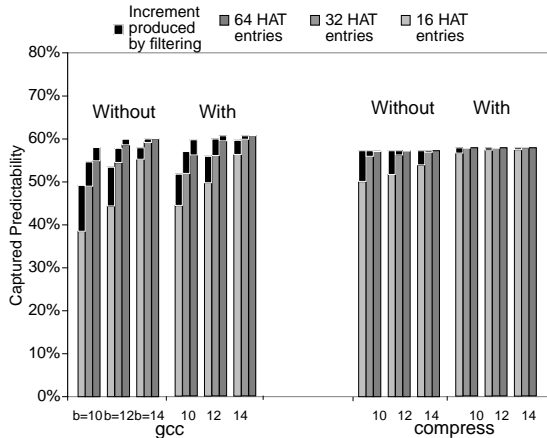


Fig. 4. Effect on the captured predictability of filtering-out HAT allocations without managing empty HAT entries (without) and managing empty HAT entries (with). Results presented for benchmarks *gcc* and *compress*.

Filtering is valuable to reduce the working set of different high-order address bits; predictable load instructions can use HAT entries that could have been related to unpredictable load instructions. Also, medium-predictable load instructions can unlink a HAT entry in an unpredictable burst. The increase in performance is noticeable in benchmarks with large number of unpredictable static load instructions; for instance *gcc* (Figure Fig.). Observe that filtering

increases the predictability captured by the 2LAP independently of the management of empty HAT entries; but combining both characteristics, the 2LAP captures more predictability.

From previous observations, the predictability increase is obtained by managing empty HAT entries and by filtering-out HAT allocations; each characteristic is useful for some benchmarks. Using 2LAP configurations that manage empty HAT entries, Figure Fig. shows the captured predictability without filtering and filtering-out HAT allocations for all the evaluated benchmarks.

Configurations with $b=14$ and 32 HAT entries or 64 HAT entries, and $b=12$ and 64 HAT entries capture as much predictability as the BP with an unbounded AT (except in *gcc*). In the remaining configurations, we can appreciate a captured-predictability decrease of the 2LAP respect the BP. The decrease is significant in benchmarks *go*, *gcc* and *vortex*; these results are coherent with the match-depth evaluations (Figure) because these benchmarks present the largest match depths.

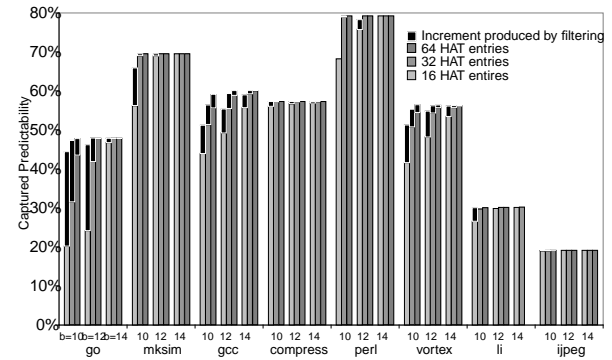


Fig. 5. Effect on the captured predictability of filtering-out HAT allocations managing empty HAT entries.

B.2 Influence of chunk selection on the accuracy

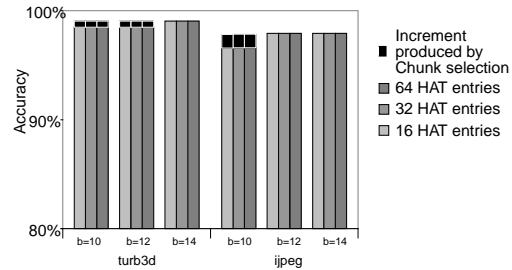


Fig. 6. Influence of dynamic chunk selection on the accuracy of the 2LAP using unbounded LAT's.

Chunk selection almost does not influence on the predictability captured by the 2LAP but the accuracy is sensitive to it. Figure Fig. presents an evaluation of the accuracy of the 2LAP in two benchmarks that compute addresses with large strides: *ijpeg* (up to 2^{11} bytes) and *turb3d* (up to 2^{18} bytes). The bottom side of each bar shows the accuracy of a 2LAP that always selects the low-order bits of the addresses; the top side of each bar, reflects the increment on the accuracy when dynamic selection of the proper chunk is performed. With chunk selection, the accuracy of the predictor saturates for all the analysed configurations.

In the remaining SPECT-INT benchmarks, chunk

selection almost does not influence on the accuracy of the 2LAP because these benchmarks present strided addresses with a short stride.

B.3 No-MRU versus LRU algorithm

When no empty HAT entry is found, we have proposed a replacement policy that selects randomly a HAT entry but the MRU one (no-MRU algorithm) because the implementation of the LRU algorithm is complex and expensive for large tables. Figure Fig. shows the performance difference between both replacement policies. The bottom side of each bar stands for the predictability captured using the no-MRU algorithm, and the top side of each bar stands for the increment due to the LRU algorithm.

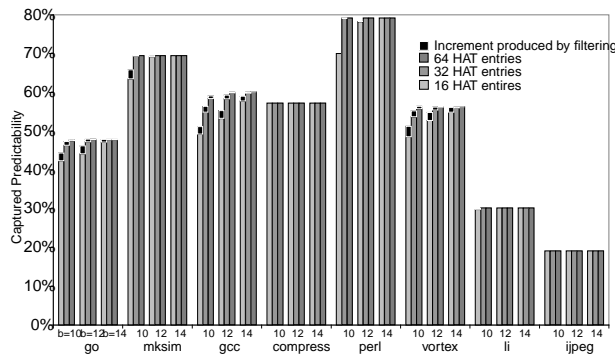


Fig. 7. Influence of the algorithm used by the replacement policy of HAT on the predictability captured by the 2LAP.

The performance of the 2LAP is not saturated in some configurations with the no-MRU replacement algorithm because there are a significant amount of capacity misses in HAT. In these cases, the LRU algorithm is better than no-MRU algorithm and the predictability decrease is limited by a 2.4%. For configurations with $b=12$ and 64 HAT entries, and with $b=14$ and 32 or 64 HAT entries, both the LRU and the no-MRU algorithms exhibit the same performance.

B.4 Link-counter width

Our experiments show that using three-bit link counters the performance of the 2LAP is almost saturated. Note that the goal of these counters is to detect empty HAT entries, and three-bit counters estimate the empty HAT entries with a high correctness.

V. TWO-LEVEL ADDRESS PREDICTOR VERSUS BASE LAST-ADDRESS PREDICTOR

This section presents an evaluation of three characteristics of the 2LAP using bounded prediction tables: area cost, captured predictability and accuracy, and compares them with the ones of the BP.

Bounded LAT's can reduce the pressure over HAT, decreasing the amount of capacity misses. However, even using 64 HAT entries, the accuracy of some 2LAP configurations decreases respect the one of the BP. Consequently, results showed in this paper will focus on 64-entry HAT's. Moreover, we will use 2LAP configurations that manage empty HAT entries, 3-bit link counters, no-MRU replacement algorithm in HAT and $b=10, 12$ or 14 . Working-set size of static load instructions of the benchmarks [17] justifies that the selected LAT-size range is from 256 to 4.096 entries.

A. Area cost of the predictors

We evaluate the area cost of an address predictor as the number of bits of information that it records. Following expressions show the area cost of the BP and the 2LAP with the no-MRU replacement policy as a function of the number of prediction-table entries. We have assumed the use of 64-bit logical addresses, t -bit tags, 3-bit link counters in the HAT entries, and b -bit address chunks in each LAT entry. The last component of the 2LAP area cost is needed to record the index of the MRU HAT entry.

$$\begin{aligned} \text{BasePredictor AreaCost} &= (t + 64 + 2) \times \text{ATentries} \\ \text{2LAP AreaCost} &= (3 + (64 - b)) \times \text{HATentries} + \\ &+ \left(\log_2 \text{HATentries} + \left\lceil \log_2 \frac{64}{b} \right\rceil + b + 2 + t \right) \times \text{LATentries} + \log_2 \text{HATentries} \end{aligned}$$

The number of tag bits influences on the accuracy of the predictor. In the analysed benchmarks, Morancho et al. show in [17] that the accuracy of the BP saturates when the number of index bits plus tag bits is 17. We will use 2LAP configurations with this number of tag bits. A similar behaviour is observed in the context of branch-instruction identification [7].

The area-cost reduction from an BP configuration to a 2LAP configuration with the same number of AT and LAT entries depends on the number of LAT entries, the number of HAT entries and b . In the evaluated configurations the reduction ranges from 37% (256 LAT entries, $b=14$) up to 60% (4.096 LAT entries, $b=10$).

B. Captured address predictability

Figure Fig. shows the predictability captured by the 2LAP and the BP in the benchmarks with the largest working-set size of static load instructions and match depths. The horizontal axes stand for the area-cost of the predictor and the vertical axes stand for the predictability captured by the predictor. The leftmost-top graph is labelled with the number of AT and LAT entries of the predictor configurations.

The area-cost reduction from a 2LAP to a BP configuration with the same number of LAT and AT entries do not represent a performance loss. Continuous oval surrounds, in benchmark *go*, configurations with AT entries=LAT entries=4.096. In these cases, the load instructions allocated in LAT are also allocated in AT.

Similar area-cost for BP and 2LAP configurations are obtained when the number of LAT entries doubles the number of AT entries. These configurations are surrounded by a dashed oval, in benchmark *go*, for LAT entries=2×AT entries=2.048. In this case, the 2LAP configurations outperform the BP configuration because LAT has less capacity misses than AT.

The remaining benchmarks present a similar behaviour but in a different AT/LAT-size range.

C. Accuracy

The accuracy is not a growing function on the number of AT entries as the predictability: its behaviour is irregular. Conflicts in AT can increase the accuracy because less predictions are performed. When there are few conflicts in AT, accuracy depends on the ability of confidence counter to characterize the load behaviour and to prevent

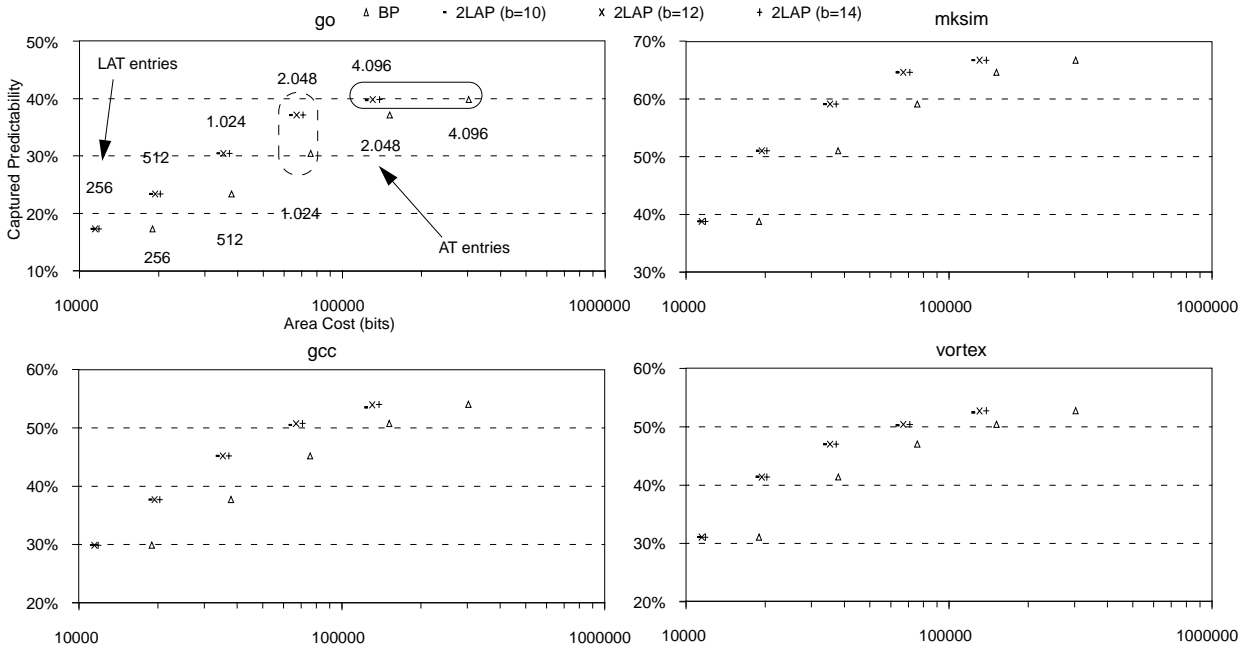


Fig. 8. Predictability captured by the *Base Predictor* and the 2LAP in several benchmarks. Horizontal axes stand for the base-10 logarithm of the area cost of the predictors, vertical axes stand for the captured predictability.

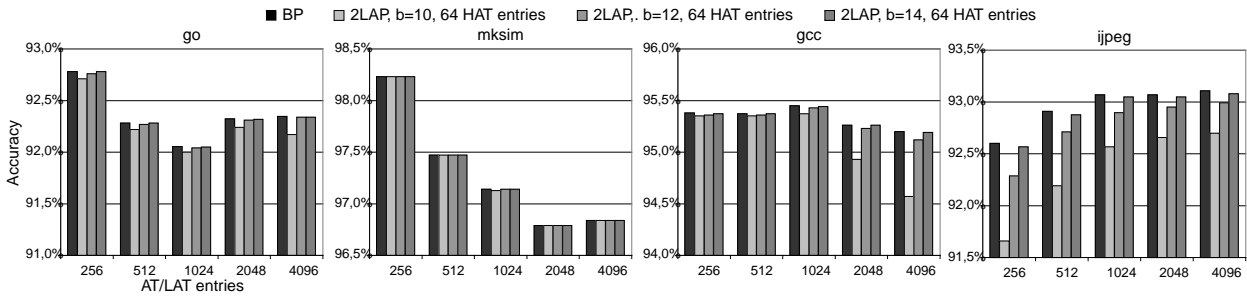


Fig. 9. Accuracy of the BP and the 2LAP. Horizontal axes present predictor configurations, vertical axes stand for the accuracy.

some predictions. In this work we use two-bit saturating counters as a confidence mechanism. More conservative confidence mechanisms can increase the accuracy but also reduce the captured predictability; moreover, they can decrease the pressure over the HAT. That is, our results for HAT-entry requirement are an upper limit for the HAT-entry requirement of these confidence mechanisms. However, the evaluation of other confidence estimators is out of the scope of this paper.

Figure Fig. compares the accuracy of both the BP and the 2LAP in several benchmarks. The vertical axes stand for the accuracy of the predictors and the horizontal axes present the same predictor configurations that in Figure Fig. (without showing area cost); configurations with the same number of AT and LAT entries are grouped.

In 2LAP, there is another characteristic that introduces mispredictions. When an HAT entry is replaced, the 2LAP does not invalidate the LAT entries linked previously to this HAT entry. Then, the next execution of a load instruction allocated at these LAT entries can be mispredicted.

For small LAT's, some of these mispredictions are removed because a LAT replacement invalidates the link; in this case, a slightly accuracy decrease is observed. For large LAT's, the working-set size of values

for the high-order bits of the addresses can be larger than the number of HAT entries; in this case, an accuracy decrease is observed. For instance, for 2LAP configurations with $b=10$ the difference is limited by a 0.7% (*gcc*).

Accuracy increases as b grows because the pressure over HAT decreases, then, more replacements are performed using empty HAT entries. For 2LAP configurations with $b=14$, 2LAP configurations are as accurate as the BP.

Benchmark *jpeg* presents a sharp decrease for 256 LAT entries and $b=10$. This behaviour is due to load instructions with large stride. The allocation of one of these load instructions in HAT produces a misprediction before classifying it as unpredictable. Its eviction from LAT by conflicts and subsequent reallocation reproduces the previous behaviour.

When LAT size doubles the AT size, configurations with a similar area cost are obtained. In these cases, the captured predictability is bigger in 2LAP but accuracy usually is lower, because the accuracy of the 2LAP ($b=14$) is similar to the one of the BP with same AT size, and the irregular behaviour of the accuracy of BP

VI. TWO-LEVEL ADDRESS STORAGE APPLIED

TO THE LOOKING-BACKWARD PREDICTOR

Some works have proposed predictors with filtering capacity. The idea of these predictors is to avoid the allocation of unpredictable instructions in the prediction table. For this, they delay the allocation of an instruction when it collides few times with an allocated predictable instruction that is being executed. Some proposals delay the allocation of any instruction [6][20], other delay the allocation only of the unpredictable ones [17]. In later case, instructions are classified dynamically.

We will evaluate the influence of TLAS organization on the performance of an last-address predictor with filtering capacity. The evaluation is performed on the *Looking-Backward Predictor* [17]; this predictor improves the performance of the BP with twice AT entries, and reduces its area cost around a 40%.

Results show that, the *Looking-Backward Predictor* that uses TLAS needs as many HAT entries as the 2LAP to achieve the performance of the original *Looking-Backward Predictor*. Although the *Looking-Backward Predictor* is filtering the allocation of unpredictable load instruction in LAT, the 2LAP is also filtering the allocation in HAT of high-order bits computed by unpredictable load instructions. Moreover, the chained filtering of the *Looking-Backward Predictor* that uses TLAS does not increase sufficiently the performance of the predictor with fewer HAT entries to equalize the performance using 64 HAT entries.

Also, filtering HAT allocations using LAT information can not be suppressed because the predictors with filtering capacity do not guarantee the exclusion of unpredictable load instructions from LAT. Moreover, the information used to filter HAT allocations is more accurate than the one used to filter LAT allocations.

The area cost reduction for a *Looking-Backward Predictor* when TLAS organization is used ranges from 29% (256 AT/LAT entries) to 40% (4.096 entries).

VII. CONCLUSIONS

We have shown that the spatial-locality property of the memory references produces redundancy in the address field of the prediction tables of the address predictors, because the number of different values for the high-order bits of the addresses recorded in the tables is small.

We have taken advantage of this fact to reduce the amount of information recorded in the prediction tables. Our proposed organization splits the addresses computed by the load instructions in two parts: the high-order bits and the low-order bits. Addresses with the same high-order bits share the only copy of these bits.

Also, we show that: a) management of empty entries in the table that records the high-order bits (HAT), and b) filtering-out the allocations of high-order bits related to unpredictable load instruction reduce capacity misses in HAT and improves the performance of the organization.

The inclusion of this organization and control in a typical last-address predictor or in an enhanced address predictor with filtering capacity reduces the area-cost of the predictor without performance loss.

For $b=14$ and 64 HAT entries, both the BP and the 2LAP are predicting correctly and mispredicting the

same dynamic load instructions. Then, a processor that implements the 2LAP will obtain the same IPC improvement than the one obtained by implementing the BP with a significant area-cost reduction. The IPC improvement obtained by address prediction have been reported in [3][5][10][19].

Other prediction models (stride-based, context-based and hybrid) can also take advantage of the spatial-locality of the addresses to reduce their area cost.

ACKNOWLEDGEMENTS

This work was supported by the Ministry of Education and Science of Spain under grant CICYT TIC98-0511-C02-01, and by the CEPBA (European Centre for Parallelism of Barcelona).

REFERENCES

- [1] Alpha 21264 MicroProcessor Data Sheet. (1999). Compaq Computer Corporation (EC-R4CFA-TE).
- [2] J.L. Baer and F.T. Chen. (1991). An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty. In *Supercomputing'91*, pp. 176-186
- [3] M. Bekerman, S. Jourdan, R. Ronen, G. Kirshenboim, L. Rappoport, A. Yoaz and U. Weiser. (1999). Correlated Load-Address Predictors. ISCA-26.
- [4] L.A. Belady. (1966). A Study of Replacement Algorithms for Virtual Storage Computers. In *IBM Systems Journal*, Vol. 5 (2), pp. 78-101.
- [5] B. Black, B. Mueller, S. Postal, R. Rakvic, N. Utamaphethai and J.P. Shen. (1998). Load Execution Latency Reduction. In *12th IC3*, pp. 29-36
- [6] B. Calder, G. Reinman and D.M. Tullsen. (1999). Selective Value Prediction. In *Proceedings of the 26th ISCA*.
- [7] B. Fagin. (1995). Partial Resolution in Branch Target Buffers. In *Proceedings of the MICRO-28*, pp. 193-198.
- [8] M. Farrens and A. Park. (1991). Dynamic Base Register Caching: A Technique for Reducing Address Bus Width. In *ISCA-18*.
- [9] F. Gabbay and A. Mendelson. (1997). Can Program Profiling Support Value Prediction? In *Proceedings of the 30th Micro*, pp. 270-280
- [10] J. González and A. González. (1997). Speculative Execution via Address Prediction and Data Prefetching. In *11th IC3*, pp. 196-203
- [11] J. L. Hennessy and D. Patterson. (1995). *Computer Architect a Quantitative Approach*. Second Edition. Morgan Kaufman Publishers, Inc.
- [12] B. Jacob and T. Mudge. (1998). Virtual Memory in Contemporary Microprocessors. *IEEE Micro*, Vol 18(4), pp. 60-75.
- [13] M.H. Lipasti, C. B. Wilkerson and J.P. Shen. (1996). Value Locality and Load Value Prediction. In *Proceedings of the 7th ASPLOS*, pp. 138-147
- [14] M.H. Lipasti and J.P. Shen. (1996). Exceeding the Dataflow Limit via Value Prediction. In *Proceedings of the 29th Micro*, pp. 226-237
- [15] S. McFarling. (1993). *Combining Branch Predictors*. Western Research Laboratory (Digital). WRL-TN-36
- [16] E. Morancho, J.M. Llaberia and À. Olivé. (1998). Split Last Address Predictor. In *Proceedings of the PACT'98*, pp. 230-237.
- [17] E. Morancho, J.M. Llaberia and À. Olivé. (1999). *Looking History to Filter Allocations in Prediction Tables*. UPC-DAC-1999-25
- [18] E. Musoll, T. Lang and J. Cortadella. (1997). Exploiting the locality of memory references to reduce the address bus energy. In *Proceedings of the International Symposium on Low Power Design and Electronics*.
- [19] G. Reinman and B. Calder. (1998). Predictive Techniques for Aggressive Load Speculation. In *Proceedings of the 31st Micro*.
- [20] B. Rychlik, J.W. Faistl, B.P. Krug and J.P. Shen. (1998). Efficacy and performance impact of value prediction. In *Proceedings of the PACT'98*.
- [21] B. Rychlik, J.W. Faistl, B.P. Krug, A.Y. Kurland, J.J. Sung, M. Velez and J.P. Shen. (1998). *Efficient and Accurate Value Prediction Using Dynamic Classification*. Carnegie Mellon University, CMU-ART-1998-01.
- [22] Y. Sazeides and J.E. Smith. (1996). The Predictability of Data Values. In *Proceedings of the 29th Micro*, pp. 238-247
- [23] A. Seznec. (1994). Decoupled sectored caches: reconciling low tag volume and low miss rate. In *Proceedings of the 21st ISCA*, pp. 384-393.
- [24] A. Seznec. (1996). Don't use the page number, but a pointer to it. In *Proceedings of the 23rd ISCA*, pp. 104-113.
- [25] H. Wang, T. Sung and Q. Yang. (1997). Minimizing Area Cost of On-Chip Cache Memories by Caching Address Tags. *IEEE Transactions on Computers*, 46 (11): 1187-1201.
- [26] C. Zhang, X. Zhang and Y. Yan. (1997). Two Fast and High-Associativity Cache Schemes. *IEEE Micro*, Vol 17(5), pp. 40-49.