The Difference-bit Cache*

Toni Juan, Tomas Lang‡ and Juan J. Navarro

Department of Computer Architecture
Universitat Politècnica de Catalunya
Gran Capità s/n, Modul D6
E-08034 Barcelona, (Spain)

e-mail: {antonioj,juanjo}@ac.upc.es

‡Department of Electrical and
Computer Engineering
University of California at Irvine

e-mail: tlang@uci.edu

Abstract

The difference-bit cache is a two-way set-associative cache with an access time that is smaller than that of a conventional one and close or equal to that of a direct-mapped cache. This is achieved by noticing that the two tags for a set have to differ at least by one bit and by using this bit to select the way. In contrast with previous approaches that predict the way and have two types of hits (primary of one cycle and secondary of two to four cycles), all hits of the difference-bit cache are of one cycle. The evaluation of the access time of our cache organization has been performed using a recently proposed on-chip cache access model.

1 Introduction

Since the cycle time of a pipelined processor is usually determined by the cache access time [4], [10], [2], the best performance is obtained with a direct-mapped first-level cache [11], [7], [5], even though for most programs the miss ratio of this cache is somewhat greater than that of a set-associative cache [13], [5], [6], [3]. A clear performance improvement could be obtained if it is possible to have a cache with the access time of the direct-mapped cache and the miss ratio of the set-associative cache.

Cache organizations that modify a set-associative cache to achieve an average access time close to that of a direct-mapped cache are presented in [9], [14], [12], [1] and a design framework is presented in [16]. All these proposals are based on the same idea, namely, a candidate line is selected in a time corresponding to the direct access, while it takes longer to determine whether it is the correct line. Because of the speculative nature of the initial selection, these schemes have two kinds of hits: primary hits having a latency of one cycle, and secondary hits with a latency from two to four cycles. As a consequence, the average hit time is somewhat larger than that of a direct-mapped cache. The proposals mentioned differ in the function used to predict the candidate line. Other related approaches are the victim cache [8] and the virtual lines [15]. In the victim cache the miss rate is the same of the direct-mapped cache but there are two kinds of misses, the normal ones and faster ones served by a small fully-associative cache placed between the first and the second level caches. The proposal of virtual lines extends the victim cache mainly by reducing the line size of the main cache to increase the temporal locality and by increasing the line size in the auxiliar cache to improve the spatial locality. We further describe previous approaches in Section 3.

In the organization presented here the hit time is faster than that of a two-way set-associative cache and close or equal to that of a direct-mapped cache, the actual value depending on the technology. In contrast with previous approaches that predict the way and have two types of hits (primary of one cycle and secondary of two to four cycles), all hits of the difference-bit cache are of one cycle. Moreover, the miss rate is equal to a two-way set-associative cache. On the other hand, this proposal is only well suited for the two-way case, whereas the previous ones do not have this limitation. However, the reduction in miss rate obtained by a higher associativity is small [6], [2]. Furthermore, increasing the degree of associativity in the other proposals increases the number of secondary hits, worsening the average memory access time.

The organization presented requires a cache with virtual addresses and tags, since the bits needed would not be available in time if the addresses or tags have to be translated.

1.1 Multiarray implementation

The organization we propose is based on the fact that an optimal (fastest) realization of the data part of a cache memory consists of several subarrays as shown in Figure 1 [17], [18]. The number of subarrays and their size are a function of the cache size and of technology characteristics and implementation restrictions [1]. For this realization, in the direct-mapped case (see Figure 2) the index bits are partitioned into two parts: one part is used to access a row from each subarray and the other to select the desired subarray. Moreover, the word bits are used to select the word from the line.

*This work was supported by the Ministry of Education and Science of Spain (CICYT TIC-0429/95) and by the EU (ESPRIT Project APPARC 6634)

ISCA '96 5/96 PA, USA
© 1996 ACM 0-89791-786-3/96/0005...$3.50

[1] Shown in Figure 1 is only the partition of the bit line, although an actual implementation would also partition the word line. We do not show this partition because it does not have an effect in our proposal.

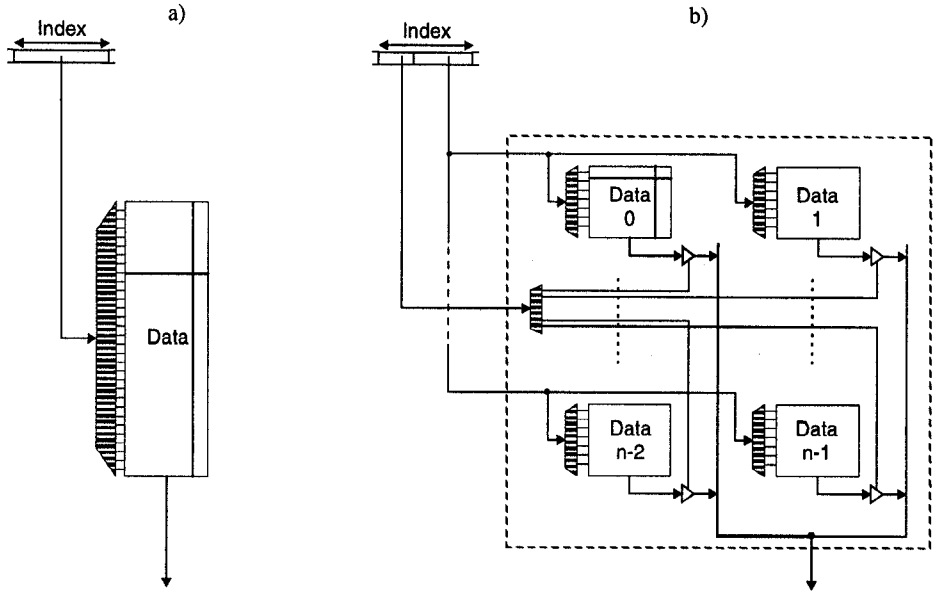

Figure 1: Ideal memory array (a) and optimal aspect ratio for same memory (b).

The same partitioning of the index bits is used in the w-way set-associative case (see Figure 3 for $w = 2$), with the difference that w bits of the second part are obtained from the tag comparisons, instead of from the index (after decoding). The access time of the set-associative cache is larger than that of a direct-mapped cache because usually the critical path is through the tag part and includes the access of tags and the comparisons [5], [18].

From the multiarray organizations shown, the following characteristics can be observed:

- For the direct-mapped cache, the delay of the signals to enable the tri-state gates is smaller than the time to access the data from the memory. Consequently, the time to obtain the resulting data line is equal to the time in which the data is available at the output of the subarray plus the delay of the tri-state gate. Moreover, the tag comparator is not in the critical path since the data can be transferred to the next stage of the pipeline without knowing whether the access is a hit. This information is only required before the use of the data, usually to store it in a processor register.

- To have a set-associative cache with the same word access time as the direct-mapped cache of the same capacity and line size, it is sufficient that the enable signals of the tri-state gates are obtained with a delay which is smaller than the access time of the data subarray. To achieve this, schemes [14], [9], [12] have been proposed in which the correct way is predicted. However, because of the prediction, there are two types of hits and the average hit time is somewhat larger than that of a direct-mapped cache.

In this paper we describe a new cache organization which achieves the hit time of the direct-mapped cache for a two-way set-associative cache. That is,

- We determine the enable signals of the the tri-state gates with a suitably small delay, and

- We do not perform prediction, but select the correct word if there is a hit. The comparisons of the tags are only used to determine whether there is a hit, but not to choose the line. Consequently, all hits are of one cycle.

2 The difference-bit cache

Our realization of a two-way set-associative cache is based on the fact that the two stored tags that correspond to a set have to differ in at least one bit. We call **diff-index** the position in the tag of the least-significant bit in which these two tags differ and **diff-value** the value of the bit in the tag of way 0 of the set. These diff-index and diff-value are used to determine the enable signals of the tri-state gates as shown in Figure 4. To do this, the pairs (diff-index, diff-value) are stored in the Diff memory of size $S \times r$, where S is the number of sets of the two-way associative cache and the value r depends on the code used to represent diff-index. If t is the number of bits of the tag, the minimum value of r is $\lceil log_2 t \rceil + 1$, with the binary code, and the maximum $t + 1$, with the 1-out-of-t code; intermediate values are obtained with other codes, as discussed later.

The enable signals of the tri-state gates are obtained as follows:

1. The corresponding entry of the Diff memory is read, simultaneously with the data (and with the tags, although these are not in the critical path).

2. The obtained diff-index is used to select the corresponding bit of the tag portion of the address.

3. The selected bit and diff-value are used to determine the way: if the bits are equal then way 0 of the set is selected, whereas if they are different way 1 is selected.

4. The way bits (for way 0 and for way 1) are used to drive the enable signal of the tri-state gates that pass the corresponding word.

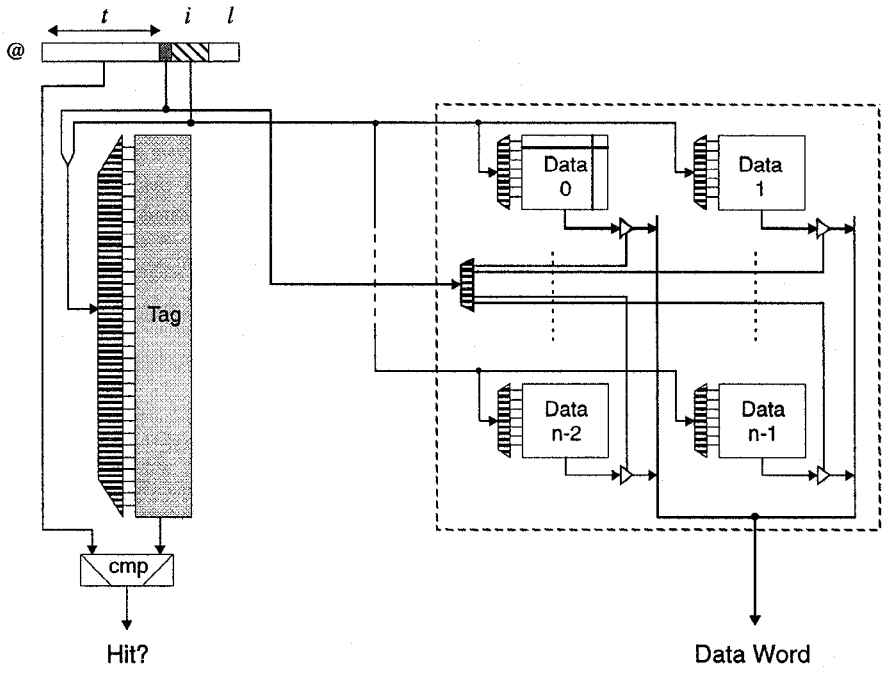

Figure 2: Multi-array realization of a direct-mapped cache (only the partitioning of the data part is shown).

Note also that only one tag comparator is needed since the tag to compare can be selected as shown in Figure 4.

In a miss, it is necessary to determine the new diff-index and diff-value. The simple hardware required is not shown in the Figure. It consists of an array of xor gates, to compare the bits of the tags, and a priority encoder, for the particular code used for the diff-index. For the replacement policy, there are the same choices as for the conventional two-way set-associative cache, resulting in the same miss ratio.

2.1 Determination of critical path

We now determine the critical path in order to argue that it is plausible that the access time of the described two-way set-associative cache is equal to that of a direct-mapped cache of the same capacity and line size. The critical path is

$$t_c = max(t_{data}, t_{enable}) + t_{tri}$$

as shown in Figure 4. Consequently, the access time corresponds to that of a direct-mapped cache if

$$t_{enable} \leq t_{data}$$

As described above,

$$t_{enable} = t_{diff} + t_{select} + t_{way} + t_{drive}$$

where t_{diff} is the access time of the Diff memory.

The terms t_{diff} and t_{select} are related and depend on the code used to represent diff-index. In general, if the code has more bits t_{diff} is larger since the memory is wider but t_{select} is smaller because the decoding is simpler. The optimal combination depends on the increase of t_{diff} with the memory width and on the complexity of the corresponding decoder.

Is $t_{enable} \leq t_{data}$? This depends on the technology and on the implementation restrictions. Although an implementation or circuit-level modeling is required to give a definitive answer, we claim that this is reasonable because the Diff memory is significantly smaller than one subarray of Data. This is so because the data memory has a width of one line (L bits), whereas the Diff memory has a width of r bits (and $r \ll L$). Consequently, in practical cases, the width of Diff is several times smaller than the width of Data; for example, for a processor with $t = 30$ and $L = 256$ resulting in a Diff width of between 6 and 31 bits, depending on the code, and a Data width of 256 bits. Moreover, the number of rows of Diff is one half of that of Data. Finally, the optimal partition of Data using [18] produces from two to eight subarrays, so that Diff is significantly smaller than one Data subarray. As a consequence, the access time of Diff is smaller than that of a Data subarray, so that the way selection can be performed in time.

To confirm that it is plausible to conclude that the resulting hit access time corresponds to that of a direct-mapped cache, we evaluate the delay using the detailed analytical access model for on-chip caches presented in [18] and apply it to a cache of the characteristics similar of that of the Alpha processors (8 Kbytes, line of 32 bytes, tag of 31 bits) and to caches with twice and four times this capacity. The evaluation of the direct-mapped cache and the conventional two-way set-associative cache are performed using `cacti`, the software associated with [18], whereas the evaluation of the difference-bit cache is obtained analytically using the expressions given in [18].

Following the approach used in [18], in Figure 5 we develop an implementation for the fully-decoded scheme of the general idea shown in Figure 4 (we do not include the tag memory and the tag comparators, since this does not affect the data selection part). The word-selection part of the implementation has the following components:

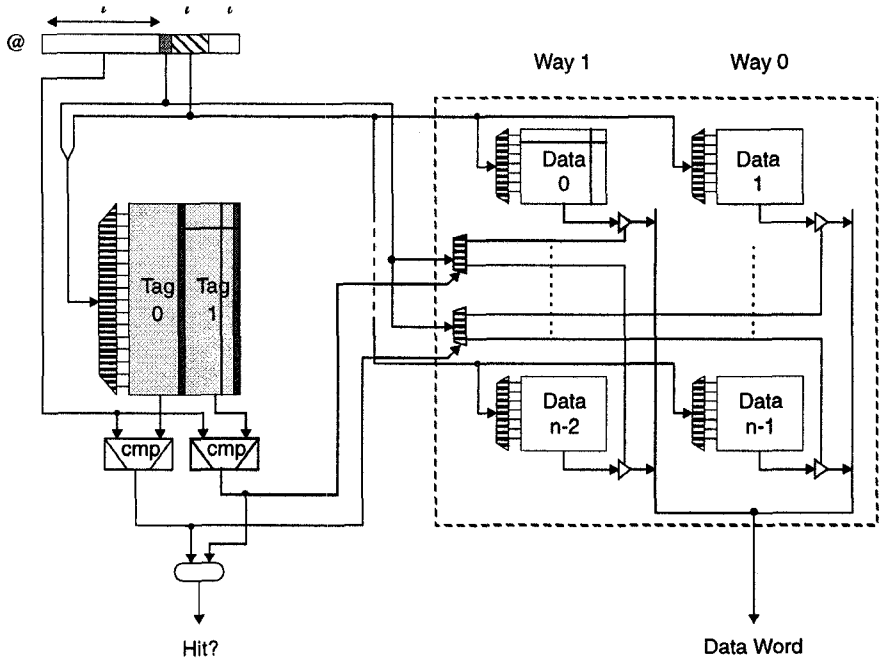

Figure 3: Multi-array realization of a conventional two-way set-associative cache (only the physical partitioning of the data part is shown).

- The Diff memory.
- A decoder to decode the diff-index (this decoder does not appear in Fig 5 since it is not necessary for the fully-decoded case).
- A selector of the corresponding tag bit. This is implemented as a column of a memory, with one bit line and a sense amplifier. To obtain the bit and its complement, two sense amplifiers are used.
- A 2x2 crossbar to obtain the way0 and way1 bits. This crossbar is controlled by the diff-value.
- The driver of the enable signals of the word tri-state gates. As in [18], this is implemented in three levels of gates. Moreover, we include here the delay of an inverter that is part of the tri-state gate.

The evaluation of the critical path is done for lines of 4 words, words of 64 bits, addresses of 43 bits and the technology parameters of the model presented in [18] (for a $.8\mu m$ CMOS technology)[2]. The only varying parameter is the cache capacity. Nevertheless, for the delay of the enable signals in the difference-bit cache, we have considered three coding options: fully-encoded (5 bits), fully-decoded (from 29 to 31 bits depending on the cache capacity) and partially-encoded (6+6 = 12 bits and a decoding of one level of two-input gates).

To simplify the presentation of the results, we divide the delay of the enable signal into an invariant part (not dependent on the capacity nor on the coding) and a variable part. The components of the invariant part correspond to the following times:

[2] According to [18] numbers for a $.5\mu m$ technology can be obtained dividing all delays by 1.6, so that the conclusions remain the same.

Capacity Kb	Coding [ns]	t_{diff} [ns]	t_{decode} [ns]	$t_{invariant}$ [ns]	t_{enable} [ns]
8	5+1	1.8	0.7	2.9	**5.4**
	12+1	2.0	0.3	2.9	**5.2**
	31+1	2.2	0.0	2.9	**5.1**
16	5+1	2.1	0.7	2.9	**5.7**
	12+1	2.2	0.3	2.9	**5.4**
	30+1	2.5	0.0	2.9	**5.4**
32	5+1	2.4	0.7	2.9	**6.0**
	12+1	2.5	0.3	2.9	**5.7**
	29+1	2.7	0.0	2.9	**5.6**

Table 1: Delay for 8, 16 and 32 Kbytes difference-bit caches for the fully-encoded (5+1), partially-encoded (12+1) and fully-decoded (31+1, 30+1 and 29+1) schemes.

- selection of the tag bit, $t_{select} = 0.6$ nsec.
- crossbar, $t_{way} = 0.2$ nsec.
- driver of enable signals, $t_{drive} = 2.1$ nsec.

This total invariant part is then of 2.9 nsec.

Table 1 gives the delay of the enable signal for different capacities and coding schemes. Using this data, in table 2, we compare the delay of the enable signal for a conventional two-way set-associative cache and for the difference-bit cache with the delay of the data part of the cache. From these tables we conclude that the difference-bit cache is considerably faster than the conventional two-way set-associative cache and that, choosing a suitable encoding, it is reasonable to argue that the access time of the difference-bit cache is equal to that of a direct-mapped cache.

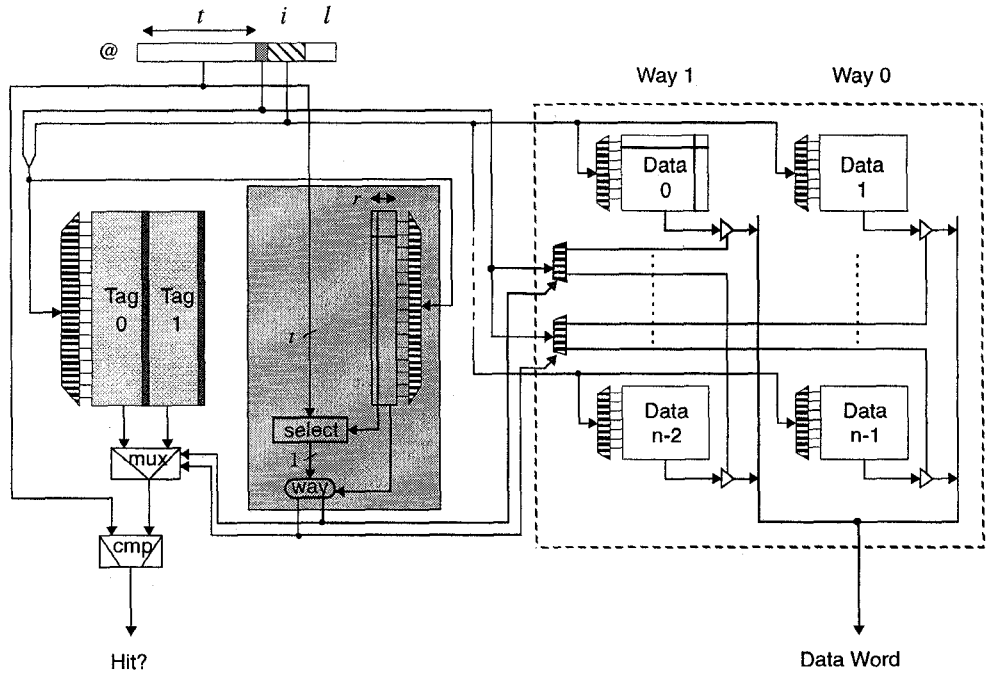

Figure 4: The difference-bit cache.

Capacity Kb	Conv. two-way [ns]	Best Diff [ns]	Data [ns]
8	8.3	**5.1**	5.2
16	8.7	**5.4**	6.2
32	9.7	**5.6**	6.5

Table 2: Delay of the enable signal for a conventional two-way set-associative cache and for the best difference-bit cache and delay of the data subarray.

Capacity Kb	t_{tag} [ns]	t_{cmp} [ns]	t_{drive} [ns]	t_{conv} [ns]
8	2.7	2.9	2.7	**8.3**
16	3.1	2.9	2.7	**8.7**
32	4.3	2.8	2.6	**9.7**

Table 3: Delay of the critical path for conventional two-way set-associative caches ($t_{conv} = t_{tag} + t_{cmp} + t_{drive}$).

To clarify further the difference between the two-way conventional cache and the difference-bit cache, in Table 3 we give a breakdown of the times of the former.

The idea can be applied to any cache and line sizes. For a given technology, the access time would be closer to that of a direct-mapped cache for larger cache size since the invariant part of the delay in the difference-bit cache becomes less significant when the size increases. This is also the case for longer lines (for the same capacity) since the number of sets is reduced, resulting in a shorter Diff array.

2.2 Area increase

The area requirements of the new two-way set-associative cache implementation are somewhat larger than those of the conventional two-way set-associative cache. This extra area corresponds to the shaded portion of Figure 4. The main contribution to this area is the Diff memory of size $S \times r$ bits. In comparison, the Data cache has $2S \times L$ bits (L is the line size in bits) and the tag memory has area $S \times (2t)$ so that the fraction of increase is

$$\frac{r}{2(L+t)}$$

For practical cases this fraction is small; for example for the values used in the evaluation, similar to those of the Alpha family of processors, it is between 0.01 and 0.06 depending on the Diff implementation. Table 4 shows other typical values.

Since the added area depends on the width of the Diff memory, it is convenient to choose the minimum width that achieves the required access time. From the access time data given in Table 1, we would choose $r = 13$ for the 8K case (a 2% area increase) and $r = 6$ for the 16K and 32K cases (a 1% area increase).

3 Related work

As mentioned in the Introduction, several previous proposals have considered a set-associative cache with the access time of a direct-mapped cache. The common denominator among these proposals is that a prediction of the way is performed and the corresponding word is selected. Later, after the tag comparisons, the correct way is determined and a new selection has to be performed if the prediction failed. Consequently, there are two types of hits as follows:

- Primary hits, which occur when the prediction is correct. These hits are served in one cycle.

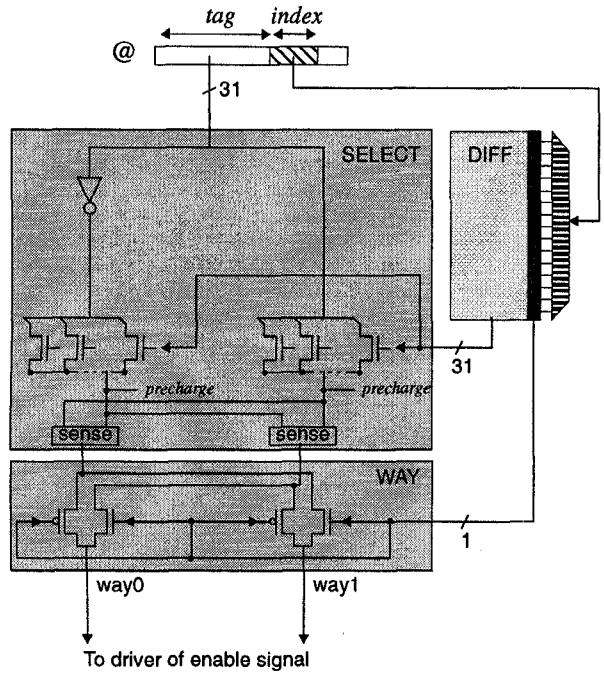

Figure 5: A possible hardware implementation for the tag selection and way signals.

r	$\lceil log_2 t \rceil + 1$		$2\lceil \sqrt{t} \rceil + 1$		$t+1$	
L	128	256	128	256	128	256
t						
20	0.02	0.01	0.04	0.02	0.07	0.04
31	0.02	**0.01**	0.04	**0.02**	0.10	**0.06**
40	0.02	0.01	0.04	0.03	0.12	0.07

Table 4: Fraction of area increase of the difference-bit cache compared with a direct-mapped cache with the same capacity and line size.

- Secondary hits, when the prediction fails. In this case, another selection has to be performed, so that the hit requires from two to four cycles.

The proposals differ in the function used for selection and on the replacement policy. In contrast, the difference-bit cache proposed here achieves the miss rate of a two-way set-associative cache and the hit time of a direct-mapped cache with all the hits being primary hits.

We now describe in somewhat more detail these previous proposals.

In the MRU cache [14] the predicted line is the most-recently used one of the set. Secondary hits require two cycles. This scheme can be used for any degree of associativity but as the associativity increases the probability to have a primary hit decreases, worsening the average hit time.

In the column-associative cache [1] two hashing functions are applied to an address. The data is accessed using the first hashing function (similar to direct mapping). If this first function misses, a second function is used for a secondary hit. If the second function is a hit, the lines corresponding to these two functions are swapped. In a miss, the last line referenced is placed according to the first hashing function. A secondary hit requires three cycles. Moreover, due to the sequential application of two hashing functions, the miss cycle time is increased in three cycles.

Another proposal is the DASC cache [12]. This is a set-associative cache in which the prediction is done assuming a direct-mapped cache. If the tag side detects a hit in another position of the set, the data use is aborted and the line in the accessed position and in the correct position are swapped. A secondary hit requires four cycles. In case of miss, the line is written according to the replacement algorithm and then is swapped with the line that is accessed in a direct-mapped cache. Again it can be used for any degree of associativity but the probability of first-time hit decreases.

The last proposal is the PAD cache [9]. The tag side is divided into two parts. The first part holds the k least-significant bits of the tags and the other part keeps the remaining bits. The way is predicted comparing the tags of the first part. In case of more than one hit in this part, any of the ways (for example the most-recently used) is accessed while the second part of the tags are compared to determine if the correct way was predicted. The penalty of secondary hits is of one additional cycle. It can be used for any degree of associativity but the probability of primary hit decreases.

4 Conclusions

We have presented the difference-bit cache, a new organization of a two-way set-associative cache with the access time of a direct-mapped cache of the same capacity and line size. This access time is obtained by separating the selection of the proper way from the detection of a hit, and selecting the way using the least-significant bit in which both tags of a set differ. The performance obtained with the difference-bit cache is better than the performance obtained with a direct-mapped cache, a conventional two-way set-associative cache and with any of the previous proposals that cause two types of hits.

Our proposal has been evaluated using the implementation approach and the detailed cache model of [18]. The results of this evaluation show that the desired access time is achieved for parameters corresponding to practical first-level caches.

The additional area of the selection mechanism is small and has been estimated at about 3% of the cache area.

The difference-bit cache can be directly used for virtual-indexed/virtual-tagged caches. It requires a cache with virtual tags, since the delay of the address translation would not allow a selection of the way in time. Moreover, the index has to be virtual to permit a fast data access. However, these virtual-indexed/virtual-tagged caches have two drawbacks: a) a context switch may invalidate all cache lines unless the cache lines are tagged with identifiers of their address space, and b) two or more virtual addresses can map to the same real address introducing synonym problems [19].

Virtual-indexed but real-tagged (V/R) caches are preferred because they do not suffer from the context-switching problem. Moreover, the synonyms problem is minimized. We are investigating the possibility of adapting the difference-bit cache to V/R caches. Notice that the translated tag is not needed until the bit selection so that the translation can overlap with the access to the Diff memory. It is not clear at this point whether this overlap is sufficient to get a suitable access time.

The difference-bit idea can be applied to any degree of associativity but the additional area required increases significantly and also the access time.

Acknowledgements

We thank Dr. Andre Seznec for his help in clarifying the applicability for the V/R case and the anonymous reviewers for their useful comments.

References

[1] Anant Agarwal and Steven D. Pudar. Column-associative caches: A technique for reducing the miss rate of direct-mapped caches. In *Proc. of the Int. Symp. on Computer Architecture*, pages 179–190, 1993.

[2] Keith Boland and Apostollos Dollas. Predicting and precluding problems with memory latency. *IEEE Micro*, pages 59–67, Aug 1994.

[3] Jeffrey D. Gee, Mark D. Hill, Dionisios N. Pneumatikatos, and Alan Jay Smith. Cache performance of the spec92 benchmark suite. *IEEE Micro*, Vol. 13(4):8–16, Aug 1993.

[4] John L. Hennessy and Norman P. Jouppi. Computer technology and architecture: An evolving interaction. *IEEE Computer*, pages 18–29, Sep 1991.

[5] Mark D. Hill. A case for direct-mapped caches. *IEEE Computer*, pages 25–40, Dec 1988.

[6] Mark D. Hill and Alan Jay Smith. Evaluating associativity in cpu caches. *IEEE trans. on Computers*, Vol. 38(12):1612–1630, Dec 1989.

[7] Norman P. Jouppi. Tradeoffs in the design of the multititan cpu. In *Proc. of the Int. Symp on Computer Architecture*, pages 281–289, 1989.

[8] Norman P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of the Int. Symp. on Computer Architecture*, pages 364–373, 1990.

[9] Lishing Liu. Partial address directory for cache access. *IEEE trans. on Very Large Scale Integration Systems*, Vol. 2(2):226–239, Jun 1994.

[10] Kunle Olukotun, Trevor Mudge, and Richard Brown. Performance optimization of pipelined primary caches. In *Proc. of the Int. Symp. on Computer Architecture*, pages 181–190, 1992.

[11] Steven A. Przybylski, Mark Horowitz, and John Hennessy. Performance tradeoffs in cache design. In *Proc. of the Int. Symp on Computer Architecture*, pages 290–298, 1988.

[12] Andre Seznec. DASC cache. In *Proc. of the 1st Int. Symp on High-Performance Computer Architecture*, pages 134–143, Jan 1995.

[13] Alan Jay Smith. Cache memories. *Computing Surveis*, Vol. 14(3):473–529, Sep 1982.

[14] Kimming So and Rudolph N. Rechtschaffen. Cache operations by MRU change. *IEEE trans. on Computers*, Vol. 37(6):700–709, Jun 1988.

[15] O. Temam and Y. Jegou. Using virtual lines to enhance locality exploitation. In *Proc. of the Int. Conf. on Supercomputing*, pages 1–12, 1994.

[16] Kevin B. Theobald, Herbert H. J. Hum, and Guang R. Gao. A design framework for hybrid-access caches. In *Proc. of the 1st Int. Symp. on High-Performance Computer Architecture*, pages 144–153, Jan 1995.

[17] Tomohisa Wada, Suresh Rajan, and Steven A. Przybylski. An analytical access time model for on-chip cache memories. *IEEE Journal of Solid-State Circuits*, Vol. 27(8):1147–1156, Aug 1992.

[18] Steven J.E. Wilton and Norman P. Jouppi. An enhanced access and cycle time model for on-chip caches. Research Report 93/5, Digital WRL, Jul 1994.

[19] C. Eric Wu, Yarsun Hsu, and Yew-Huey Liu. A quantitative evaluation of cache types for high-performance computer systems. *IEEE trans. on Computers*, Vol. 42(10):1154–1162, Oct 1993.