A case for two-way skewed-associative caches*

André Seznec
IRISA-INRIA, Campus de Beaulieu
35042 Rennes Cedex, FRANCE
e-mail:seznec@irisa.fr
http://www.irisa.fr/caps

Copyright Notice

This paper appears in Proceedings of the 20th International Symposium on Computer Architecture, San Diego, May 1993

Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for crating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

*This work was partially supported by CNRS (PRC-ANM)

Abstract

We introduce a new organization for multi-bank caches: the skewed-associative cache. A two-way skewed-associative cache has the same hardware complexity as a two-way set-associative cache, yet simulations show that it typically exhibits the same hit ratio as a four-way set associative cache with the same size. Then skewed-associative caches must be preferred to set-associative caches.

Until the three last years external caches were used and their size could be relatively large. Previous studies have showed that, for cache sizes larger than 64 Kbytes, direct-mapped caches exhibit hit ratios nearly as good as set-associative caches at a lower hardware cost. Moreover, the cache hit time on a direct-mapped cache may be quite smaller than the cache hit time on a set-associative cache, because optimistic use of data flowing out from the cache is quite natural.

But now, microprocessors are designed with small on-chip caches. Performance of low-end microprocessor systems highly depends on cache behavior. Simulations show that using some associativity in on-chip caches allows to boost the performance of these low-end systems.

When considering optimistic use of data (or instruction) flowing out from the cache, the cache hit time of a two-way skewed-associative (or set-associative) cache is very close to the cache hit time of a direct-mapped cache. Therefore two-way skewed associative caches represent the best tradeoff for today microprocessors with on-chip caches whose sizes are in the range of 4-8K bytes.

Keywords: microprocessors, cache, set-associative cache, skewed-associative cache.

1 Introduction

For a few years, the direct-mapped cache organization has been considered as the best organization for microprocessor caches [7, 14]. But technology has changed, large external caches which *were* associated with first generation RISC microprocessors are now replaced by small on-chip caches.

In section 2.1, we introduce a new organization for multi-bank cache: the skewed-associative cache. A two-way skewed-associative cache has the same hardware complexity as a two-way set-associative cache. Simulations presented in section 3 show that a two-way skewed-associative cache typically exhibits the same hit ratio as a four-way set associative cache with the same size: two-way skewed-associative caches must be preferred to two-way or four-way set-associative caches.

Then we compare using on-chip two-way skewed-associative caches in place of on-chip direct-mapped caches. In low-end microprocessor systems, the miss penalty is high; increasing clock frequency does not lead to a significant performance improvement. Using a two-way skewed associative cache rather than a direct-mapped cache improves the performance of the system even when it slightly reduces the clock frequency.

On high-end microprocessor systems, a fast second-level cache is used; the performance depends more directly on the clock frequency. In order to reduce cycle time, optimistic use of data flowing out from the caches may be used. Using this technic on microprocessors with on-chip two-way skewed-associative caches will lead to better performance than using on-chip direct-mapped caches.

Finally, we show that skewed-associative caches may be used for implementing physically indexed caches as well as virtually indexed caches.

2 Skewed-associative caches

In this section, we present a new organization for a multi-bank cache: the *skewed-associative cache*.

2.1 Background

High-end microprocessors based systems are built with a secondary external cache: a miss on the primary cache will be served in a few cycles when the line is present in the secondary cache. As the cost of a fast secondary cache is prohibitive, there will not be secondary caches in low-end systems; the miss penalty will be very high (20 or even 50 CPU cycles) (see Figure 1).

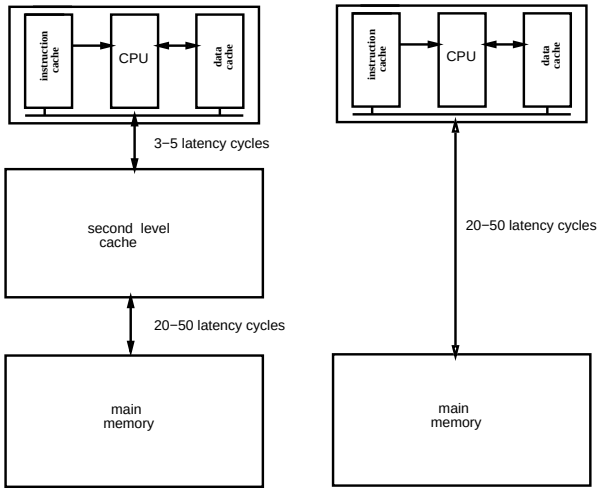

Figure 1: Basic implementations of microprocessor based systems

Let us recall some technological parameters:

microprocessor	Frequency	Instruction per cycles	Miss penalty with 2nd level cache	Miss penalty without 2nd level cache
DEC 21064	200 Mhz	2	24	100
MIPS R4000	100 Mhz	1	6	25
TI SuperSparc	50 Mhz	3	9	37
PowerPC	40 Mhz	2	5	20

Figure 2: Miss penalties converted in Instruction Delays

1. Access time to the first word of a line in main memory is generally higher than 250 ns.
2. Access time to the first word of a line in the second level cache is around 60 ns (assuming 12-15ns static memory chips): a throughput of one word per 20 ns may then be obtained.[1]

Figure 2 shows the miss penalties converted in Instruction Delays for some of the recently announced microprocessors when assuming these access delays.

Reducing miss ratio on on-chip caches has become a key issue for performance on all microprocessor systems.

2.2 Skewing on caches: principle

A set-associative cache is illustrated by Figure 3: a X way set-associative cache is built with X distinct banks. A line of data with base address D may be physically mapped on physical line $f(D)$ in any of the distinct banks. This vision of a set-associative cache fits with its physical implementation: X banks of static memory RAMs.

We propose a very slight modification in this design as illustrated in Figure 4:

> Different mapping functions are used for the distinct cache banks i.e., a line of data with base address D may be mapped on physical line $f_0(D)$ in cache bank 0 or in $f_1(D)$ in cache bank 1, etc.

We call a multi-bank cache with such a mapping of the lines onto the distinct banks: a *skewed-associative cache.*

This hardware modification incurs a very small hardware over cost when designing a new microprocessor with on-chip caches since the mapping functions can be chosen so that the implementation uses a very few gates. But we shall see that this may help to increase the hit ratio of caches and then to increase the overall performance of a microprocessor using a multi-bank cache structure.

Related works

In 1977, Smith [15] considered set-associative caches and suggested selecting the set by hashing the main memory address; this approach corresponds to figure 3: a *single* hashing function is used.

More recently Agarwal [2] (Chapter 6.7.2) studied hash-rehash caches.

> As in a conventional cache, the address indexes into a cache set, and the word of data is sent to the processor if it is valid in the set. This case is called a first time hit. On a miss, the address again indexes into the cache but using a different hashing function. If the second access is also unsuccessful, the data must be fetched from memory.

[1] This corresponds to TI SuperSparc second-level cache specification[19]

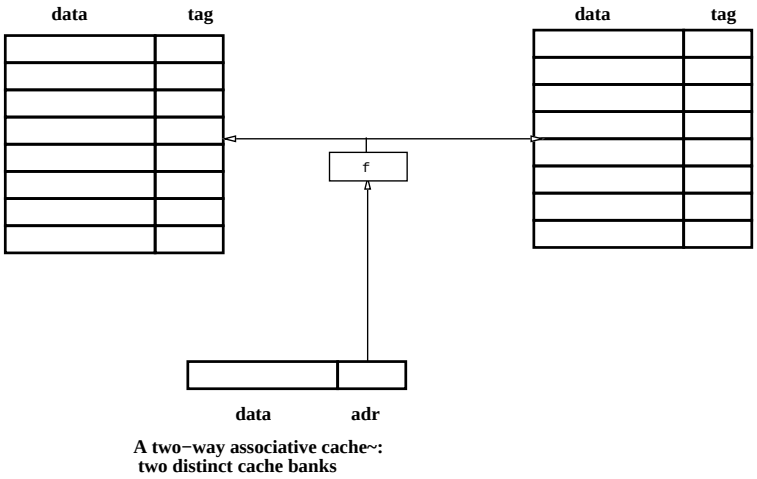

Figure 3: A two-way associative cache

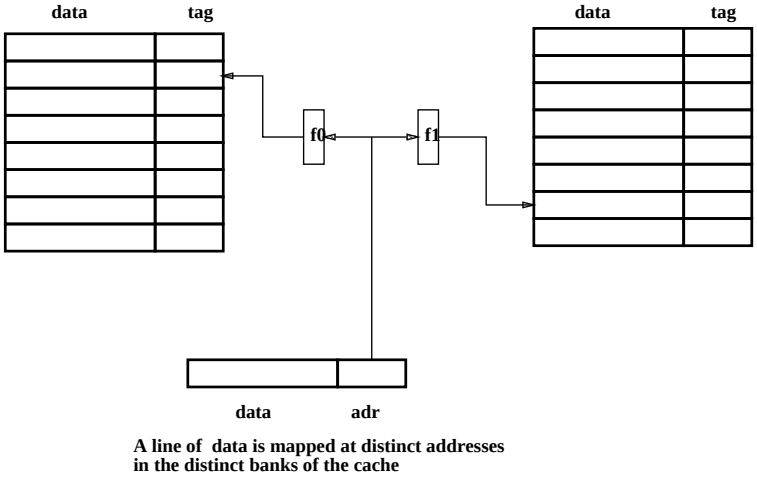

Figure 4: A two-way skewed-associative cache

Hash-rehash caches present better overall hit ratios than direct-mapped caches. But Agarwal remarked that for a 64 Kbytes cache, hash-rehash caches induce longer execution time than two-way set-associative caches.

In hash-rehash caches, the primary cache itself is used as a secondary cache after a first time miss. However in skewed-associative caches, different hashing functions are used *at the same time* for indexing the distinct cache banks.

2.3 Choosing the skewing functions

In this section, we give some insight on the properties that might exhibit functions chosen for skewing the lines in the distinct cache banks in order to obtain a good hit ratio.

2.3.1 Equitability

First of all like in classical caches, for each line in the cache, the numbers of lines of data in the main memory that may be mapped onto this cache line must be equal.

2.3.2 Inter-bank dispersion

In a usual X-way set-associative cache, when (X+1) lines of data contend for the same set in the cache, they are all conflicting for the same place in the X cache banks: one of the lines must be rejected from the cache (Figure 5).

We have introduced skewed-associative caches to avoid this situation by scattering the data: mapping functions can be chosen such that whenever two lines of data conflict for a single location in cache bank i, they have very low probability to conflict for a location in cache bank j (Figure 6).

Ideally, mapping functions may be chosen such as the set of lines that might be mapped on a cache line of bank i will be equaly distributed over all the lines in the other cache banks.

2.3.3 Local dispersion in a single bank

Many applications exhibit spatial locality, therefore the mapping functions must be chosen so as to avoid having two "almost" neighbor lines of data conflicting for the same physical line in cache bank i.

The different mapping functions must respect a certain form of local dispersion on a single bank; the mapping functions f_i must limit the number of conflicts when mapping any region of consecutive lines of data in a single cache bank i.

2.3.4 Simple hardware implementation

A key issue for the overall performance of a microprocessor is the pipeline length. Using distinct mapping functions on the distinct cache banks will have no effects on the performance, as long as the computations of the mapping functions can be added to a non critical stage in the pipeline and do not lengthen the pipeline cycle. Let us notice that in most of the new generation microprocessors, the address computation stage is not the critical stage in the pipeline (e.g. in TI SuperSparc, two cascaded ALU operations may be executed in a single cycle).

In order to achieve this, we have to chose mapping functions whose hardware implementations are very simple: as few gate delays as possible.

In [20], we exhibited a family of four functions f_i mapping which respects the previous properties. For an address A, each bit of $f_i(A)$ is obtained by Exclusive-ORing at most 4 bits of the binary decomposition of A.

2.4 Skewing on two-way associative caches

In this paper, we focus only on two-way skewed-associative caches. Our basic goals here are to minimize hardware implementation cost and extra delay on cache hit time.

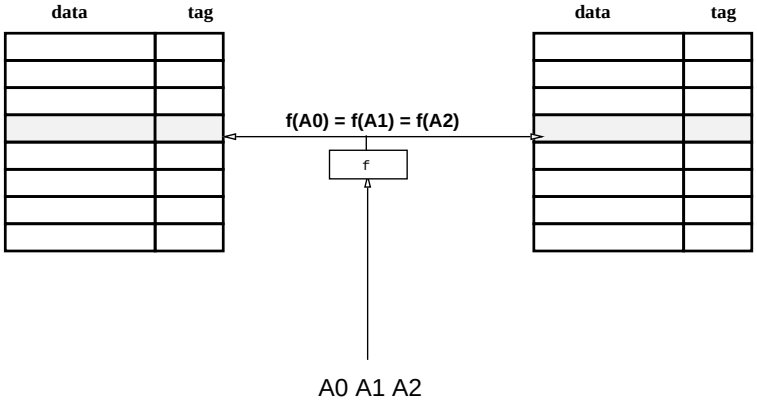

Figure 5: 3 data conflicting for a single set on a two-way set-associative cache

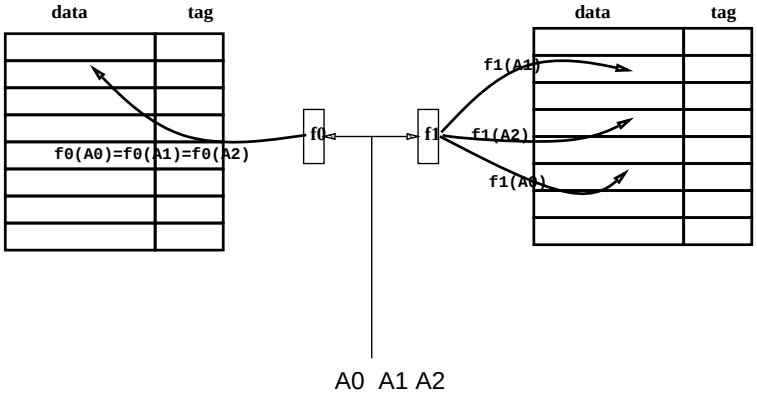

Figure 6: Data conflicting for a cache line on bank 0, but not on bank 1 on a skewed-associative cache

An example

Let us consider the particular case of 64 bytes lines and 4096 bytes cache banks. $(a_n, \ .. \ , a_0)$ is the binary representation of the address. A possible organization for a two-way skewed associative cache is illustrated in Figure 9.

The skewing functions used in this example verify the previously mentioned criterions for "good" skewing functions.

On this example, only three two entries XOR gates are added to the classical cache bank design. In the proposed design, the access time to the cache bank is only slightly increased by the delay for crossing a XOR gate when using skewing functions.

We believe that the access time may even be exactly the same as in a classical two-way set-associative cache:

1. In a microprocessor, when using a one-cycle access cache, the cache hit time generally determines the machine cycle. The address computation is performed in a less critical stage: the XOR gates may be added at the end of that stage.

2. When using a pipelined cache, row selection may be done in the second cycle.

A formal description of the family of skewing functions

Let 2^c be the size of the line.

Let 2^n be the number of cache lines in a cache bank and let us consider the decomposition of a binary representation of an address A in bit substrings A = $(A_3$,A_2, A_1, $A_0)$, A_0 is a c bit string: the displacement in the line. A_1 and A_2 are two n bits strings and A_3 is the string of the $q-(2*n+c)$ most significant bits.

Let us consider T an integer such that $0 \leq T < 2^n$ and $\bar{T}$ its binary opposite,$(\bar{T} = 2^n - 1 - T)$.

Let ϕ be a Bit Permute Permutation on the set $\{0, .., 2^n - 1\}$ (e.g. Identity, Perfect Shuffle, Bit Reversal). We consider the mapping functions defined respectively by:

$$\begin{array}{lll} F_0^{T,\phi}: & S & \longrightarrow \{0, .., 2^{n+c} - 1\} \\ & (A_3, A_2, A_1, A_0) & \longrightarrow (A_1 \oplus (\phi(A_2) \cdot T), A_0) \end{array}$$

$$\begin{array}{lll} F_1^{T,\phi}: & S & \longrightarrow \{0, .., 2^{n+c} - 1\} \\ & (A_3, A_2, A_1, A_0) & \longrightarrow (A_1 \oplus (\phi(A_2) \cdot \bar{T}), A_0) \end{array}$$

$\oplus$ is the exclusive OR and $\cdot$ is the bitwise product.

These functions satisfy the criterions for "good" skewing functions defined in the paper (Equitability, inter-bank dispersion and local dispersion).

Each bit of the $F_0^{T,\phi}(A)$ or $F_1^{T,\phi}(A)$ is either directly a bit of the binary representation of address A or the XOR of two bits of this binary representation.

T may be chosen in order to allow symmetric design of the two cache banks: when n is even, having the same number of bits equal to one and zero seems an interesting approach.

In the previous example in figure 9, T= 44 (binary decomposition 101010) and the Bit Permute Permutation is the identity.

2.5 Replacement policy on a two-way skewed-associative cache

When a miss occurs on a X-bank cache, the line of data to be replaced must be chosen among X lines. Different replacement policies may be used. LRU replacement policy or pseudo-random replacement policy are generally used in set-associative caches.

The pseudo-random replacement policy is the simplest to implement. But LRU replacement policy generally works better and may be implemented at a reasonable hardware cost. Implementing a LRU replacement policy on a two-way set-associative cache is quite simple: a single bit tag per set sufficient. More generally a LRU replacement policy for a X-way associative is feasible with adding only $X*(X-1)/2$ bit tags to each set.

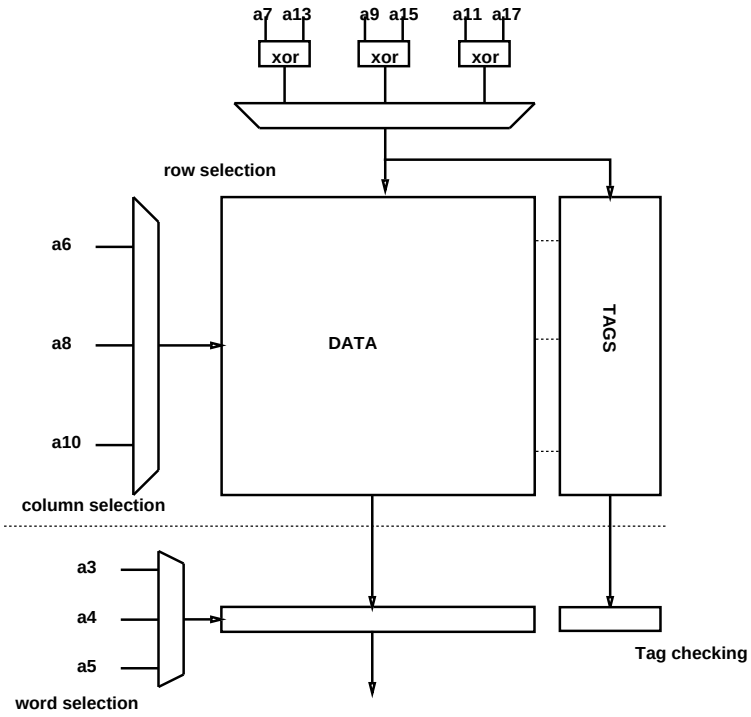

Figure 7: Cache bank 0

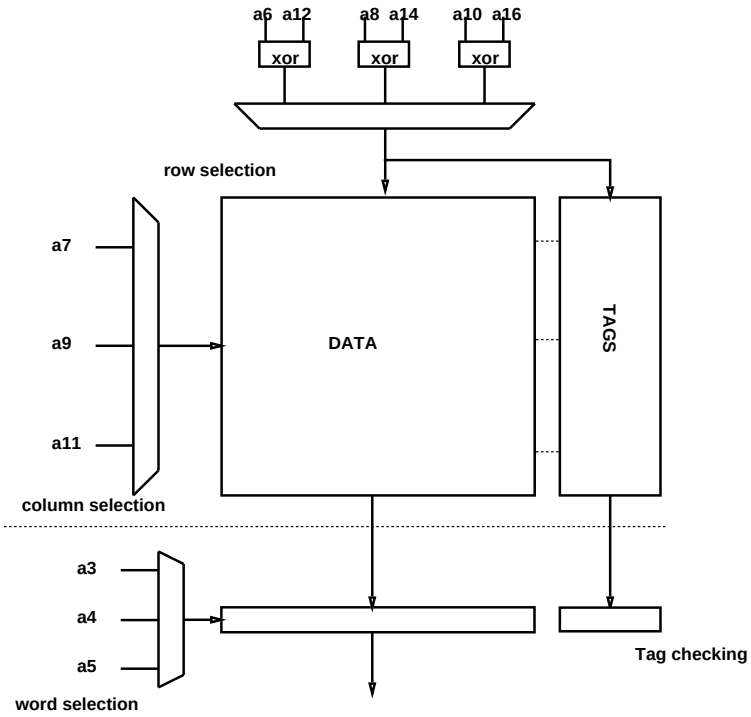

Figure 8: Cache bank 1

Figure 9: An example of a two way skewed-associative cache

Unfortunately, we have not been able to find concise information to associate with a cache line which would allow a simple hardware implementation of a LRU replacement policy on a skewed-associative cache.

Nevertheless, for two-way skewed-associative caches, a pseudo-LRU replacement policy may work fine:

> A tag bit is associated with each line in bank 0: when the line is indexed, the tag bit is asserted when the data was in bank 0 and deasserted when the data is in bank 1.
>
> On a miss, the tag of the line selected in bank 0 is read: when this tag is 1, the missing line is written in bank 1 otherwise the missing line is written in bank 0

Notice that implementing this replacement policy on a two-way skewed-associative cache requires the same hardware as implementing a LRU replacement policy on a two-way set-associative cache.

3 Simulations

Cache simulations have been conducted in order to compare skewed-associative caches with usual cache organizations.

Presented results show that for equal sizes, a two-way skewed-associative cache exhibits approximately the same miss ratio as a four-way set-associative cache.

3.1 Traces

Our set of traces is composed with two distinct sets.

The first set is composed with the three traces from the Hennessy-Patterson software obtained from the DLX simulator [5] (gcc, TeX and Spice).

Seven other traces were generated using the SparcSim simulator facility [18]; this set was composed with:

- RESEAU: the simulator of a particular interconnection network
- POISSON : a Poisson solver
- STRASSEN: a matrix-matrix multiply using the Strassen algorithm.
- LINPACKUD: part of the LINPACK benchmark
- CACHE : The cache simulator itself
- CPTC : A Pascal-to-C translator
- SPARSE: multiply of a sparse matrix by a full vector.

In the results presented in the paper, all the benchmarks were valued with the same weight.

For direct-mapped caches, a victim cache of four lines was assumed. A similar mechanism was evaluated for multi-way associative caches, but as it did not bring significant hit improvement, we do not consider it in the paper.

Virtual indexing of the cache was assumed for the simulations. Physical indexing will be discussed in section 5.

3.2 Miss ratios

Different cache configurations have been simulated: mixed data/instructions and split caches. Results are reported in figure 14. Notice that these results are certainly very optimistic: single process traces, exceptions not simulated (TLB miss, etc.), no system, .. Effective miss ratios will certainly be higher.

For separate instruction and data caches, the microprocessor is assumed to execute one instruction per cycle. The miss ratios reported in Figures 12 and 13 is the average number of misses per sequenced instruction (i.e $\frac{Instruction\ Misses+Data\ Misses}{Number\ of\ Instructions}$).

For direct-mapped caches, a victim buffer [9] of four lines was assumed. The ratio of misses which effectively induce an access on memory or on a second level cache is reported in column "Miss-Vict"[2].

LRU replacement policy was used for set-associative caches. The pseudo-LRU replacement policy described in the previous section was used for two-way skewed-associative caches.

These results show that at equal sizes a two-way skewed-associative cache exhibits approximately the same miss ratio as a four-way set-associative cache. Figure 17 shows that this conclusion is valid on the two sets of traces.

At this point of the study, we make the following recommendation:

> A two-way skewed associative cache must be preferred to a usual two-way or four-way set-associative caches.

Remark 1: Other simulations were conducted assuming a LRU policy replacement on a two-way skewed-associative cache. Better hit ratios than with the pseudo-LRU policy were obtained, but unfortunately LRU policy cannot be implemented at a reasonable hardware cost.

3.3 Skewing versus hashing

Hewlett-Packard recently introduced the HP7100 microprocessor. In this microprocessor, addresses are hashed before accessing a direct-mapped cache. On the HP7100, the whole virtual address including process number is hashed and a very large *external* cache is used (greater than 128 Kbytes); the microprocessor designers claimed that this technic improves the average hit ratio on a virtually-indexed large cache when running multiprocess workloads. This phenomenum was also observed by Agarwal [2].

Set-associative caches indexed with the function f_0 instead of the usual bit-selection have been simulated for associativity degree 1,2 and 4. Results for split 8192 bytes caches are shown in figure 18. On our benchmark set and for a small cache size, hashing the addresses does not lead to very significant hit ratio improvement.

These results show that the improveement of performance of skewed-associative over set-associative caches is mostly due to the inter-bank distribution property (see section 2.3.2) and not to a simple hashing on the addresses.

3.4 Local dispersion

In order to illustrate why, the local dispersion property (see section 2.3.3) is very important, we simulated a two-way skewed-associative cache where skewing functions f_0 and f_1 are two independent random functions. Average miss ratio for these simulations are given in figure 18 in row "skewed-associative RANDOM". As f_0 and f_1 are independant random functions, there may be local interferences on a single cache bank. These interferences affect a lot the hit ratio.

4 Influence on performance

In this section, we show that, for 8K bytes cache, using an associative cache structure will lead to better performance than using a direct-mapped cache structure.

4.1 Caches for low-end microprocessor systems

In this section, we consider low-end microprocessor systems.

[2] Victim caching does not lead to significant hit improvement for skewed-associative or set-associative caches

Organization	Miss	Miss - Vict
Direct mapped	0.074086	0.064618
2-way set-associative	0.050133	
4-way set-associative	0.041690	
2-way skewed-associative	0.043938	

Figure 10: mixed data/instruction cache: 4096 bytes

Organization	Miss	Miss - Vict
Direct mapped	0.046719	0.041846
2-way set-associative	0.029362	
4-way set-associative	0.024265	
2-way skewed-associative	0.024287	

Figure 11: mixed data/instruction cache: 8192 bytes

Organization	Miss	Miss - Vict
Direct mapped	0.058757	0.054152
2-way set-associative	0.041994	
4-way set-associative	0.036830	
2-way skewed-associative	0.037562	

Figure 12: two 4096 bytes split caches

Organization	Miss	Miss - Vict
Direct mapped	0.037876	0.034900
2-way set-associative	0.025992	
4-way set-associative	0.021844	
2-way skewed-associative	0.020865	

Figure 13: two 8192 bytes split caches

Figure 14: Miss ratios for different cache sizes and organizations; Line size is 16 bytes

Organization	Miss	Miss - Vict
Direct mapped	0.033929	0.031123
2-way set-associative	0.022641	
4-way set-associative	0.018305	
2-way skewed-associative	0.016981	

Figure 15: SparcSim traces

Organization	Miss	Miss - Vict
Direct mapped	0.047087	0.043712
2-way set-associative	0.033811	
4-way set-associative	0.030103	
2-way skewed-associative	0.029928	

Figure 16: DLX traces

Figure 17: Cache Size 8192 bytes, split Instruction/Data caches

Organization	Miss	Miss - Vict
Direct mapped	0.037876	0.034900
Direct mapped hashed	0.037586	0.032970
2-way set-associative	0.025992	
2-way set-associative hashed	0.025632	
4-way set-associative	0.021844	
4-way set-associative hashed	0.020895	
2-way skewed-associative	0.020865	
2-way skewed-associative RANDOM	0.024202	

Figure 18: Skewing versus hashing

When cache hit time does not determine the microprocessor clock

From now, we resent performance using *Cycle Per Instruction* or *Time per Instruction* assuming separate instruction cache and data cache.

Figure 19 illustrates the performance in Cycle Per Instruction on our benchmark set for different memory latencies. Both cache sizes are 8192 bytes and length of a cache line is 16 bytes.

Formula (1) is used for deriving Figure 19 from Figure 13.

$$3 * (miss - vict) * Cycle + vict + Latency * (miss - vict) \quad (1)$$

where *miss* is the miss ratio of the cache, *vict* is is the ratio of misses that hits in the victim cache [3] and *Latency* is the latency for accessing the missing line in the main memory (or second level cache). The internal delay in the microprocessor is assumed to be 3 cycles.

When the cache hit time does not determine directly the clock frequency of the microprocessor (e.g. when cache access is pipelined), then an associative structure must be used for the cache: performance benefits for memory latency of the order of 20 cycles is about 20 %.

[3] for direct mapped caches only

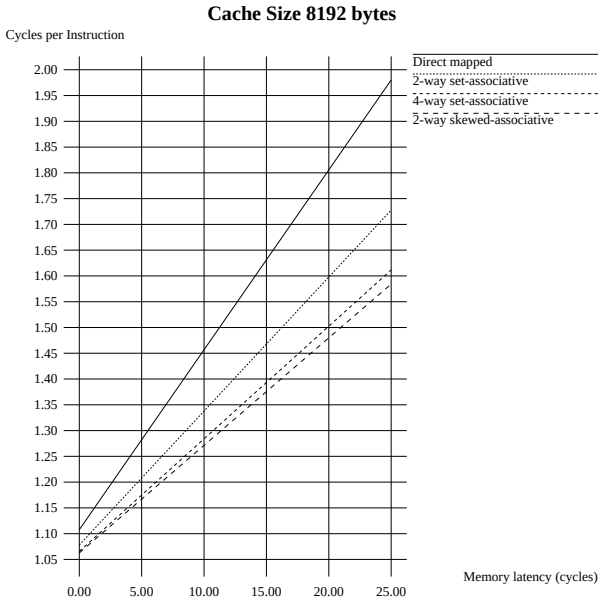

Figure 19: Separate instruction cache and data cache

Unfortunately, on many microprocessors, the cache hit time is determined by the clock frequency of the microprocessor.

When cache hit time determines the microprocessor clock

For illustrating why on-chip caches used in low-end microprocessors must be associative, we consider two microprocessors directly accessing a main memory with a 250 ns access time [4]:

- Processor P1 is a single-issue RISC microprocessor with two on-chip 8K bytes direct-mapped caches.
- Processor P2 is a single-issue RISC microprocessor with two on-chip 8K bytes two-way skewed-associative caches.

Figure 20 illustrates the clock needed on each of the two processors for achieving a performance of one instruction per X ns on our set of benchmarks.

Remark that Processor P2 achieved one instruction per 20 ns with a 14 ns clock, while Processor P1 needs a 10 ns clock to achieve the same performance level.

This example clearly indicates that for low-end microprocessor systems, the structure of the caches have to be associative:

Using a two-way skewed-associative cache is the reasonable choice.

4.2 Caches for high-end microprocessor systems

In high-end microprocessor systems, second level caches will be used. For getting back the first word of data of a missing line from this second level cache, a delay around 60 ns seems realistic with today technology.

As for low-end microprocessor systems, an associative structure of cache must be used when the cache hit time does not determine the basic clock of the processor (see Figure 19).

As in the previous section, in Figure 21, we compare the clocks needed on the hypothetic processors P1 and P2 for achieving a constant performance when they are connected to a second level cache [5]

[4] In this example, the important parameter is the sequencing rate, a single-issue microprocessor at frequency F may be replace by a dual-issue microprocessor at frequency F/2

[5] For simplicity, we do not consider second level cache misses.

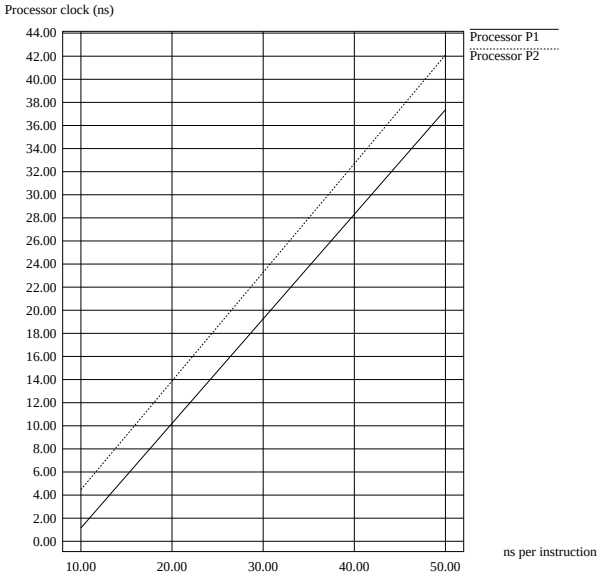

Figure 20: Comparing P1 and P2 connected to main memory

On this figure, one will notice that the *effective* performance of processor P1 and processor P2 at a given frequency are quite close:

> With a 10 ns clock, processor P2 achieves one instruction per 12 ns, while processor P1 achieves this level of performance with a 9 ns clock.

When data flowing out from the cache cannot be used before tag check, the cache hit time on a two-way skewed-associative cache and on a direct mapped cache are within a very few per cent [6], then using two-way skewed-associative caches will lead to slightly better overall performance of the system than using direct-mapped caches.

Optimistic execution

In order to increase clock frequency, data flowing out from the cache may be used before tag check with direct-mapped caches. Checking the validity of the data word may be executed in parallel with the other activities in the pipeline. The current cycle in the pipeline will be canceled if the data (or instruction) is found to be unvalid.

We shall refer to this technic as *optimistic execution*.

Optimistic execution was also proposed for a set-associative cache: the most-recently-used data (MRU) in the selected set [3] can be systematically used.

> For a 128 Kbytes cache, assuming a 12 cycles penalty on a global miss, and a one cycle penalty on a miss on the MRU region, but a hit on the global cache, the cache access time was shown to be within 4% of the performance of a true one-cycle 4-way set-associative cache[7].

Although a 32-way set-associative cache was initially considered,, the authors claimed that reducing associativity degree to four and using the optimistic MRU policy dramatically reduce the cache hit time in the range of 30-35%.

Using optimistic execution on a direct-mapped cache and on a set-associative (or skewed-associative) cache seem to lead to very close cache hit times.

[6] 2 % was reported by Hill [6]

[7] considering a constant cycle

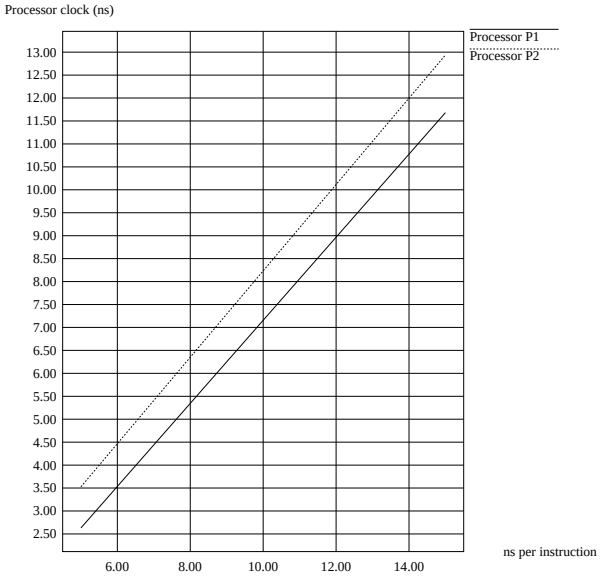

Figure 21: Comparing P1 and P2 both connected with a second level cache

When using a skewed-associative associative cache, the (pseudo) Most Recently Used data is selected. We assume that a miss on pseudo MRU region which hits in the other cache bank costs one stall cycle on the processor [8].

Notice that, on a four-way set-associative cache, the hit ratio on the MRU region corresponds to the hit ratio on a direct-mapped cache which size is only the fourth of the original cache size, while on a two-way skewed-associative cache, this hit ratio corresponds approximately to the hit ratio on a direct-mapped cache which size is half of the original cache size. There a two-way skewed-associative cache will achieve better performance than a four-way set-associative cache.

In Figure 22, we compare the clocks needed on the hypothetic processors P1 and P2 for achieving a constant performance, but here we assume optimistic execution on both processors.

Performance of processor P1 and performance of processor P2 are quite close:

> With a 8 ns clock, processor P2 achieves one instruction per 10 ns, while processor P1 achieves this performance with a 7.2 ns clock.

As, cache hit times for the two processors would be in a very few per cent (may be 5 %), using a two-way skewed associative cache would lead to slightly better performance.

5 Skewed-associative caches and virtual memory

5.1 Virtual indexing or physical indexing

As already mentionned, cache hit time is one of the critical path in the microprocessor. Caches may be virtually or physically indexed. When sharing pages between processes, physical addressing of the cache allows to avoid multiple copies of the same line of data in the cache and then the data in the cache always remain coherent; physical addressing of the cache may be considered as very desirable. Unfortunately, for physically indexed caches, the virtual-to-physical address translation must precede the cache indexing, thus increasing the overall cache hit time.

[8] This assumption is quite pessimistic, when the read data (or instruction) is not used directly by the next instruction, no cycle is lost

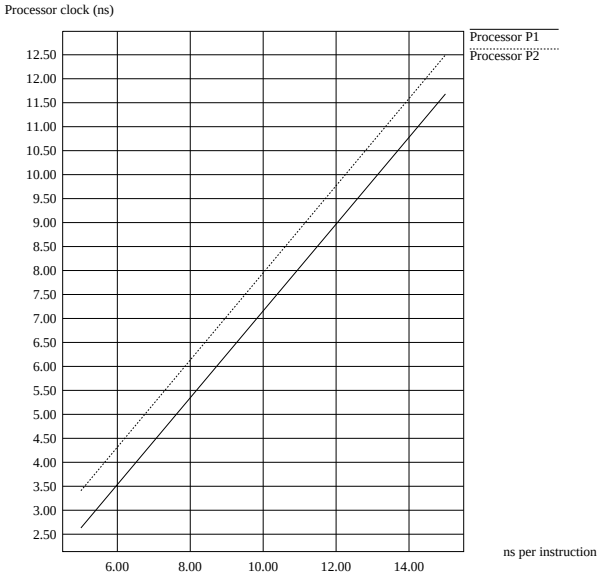

Figure 22: Optimistic execution on P1 and P2, both connected with a second level cache

Virtually indexed caches are used in HP7xxx, MIPS R4000 for example. When the cache is virtually indexed, using a skewed-associative cache in place of a set-associative cache will not lenghten the cache hit time.

5.1.1 An artifice for physical indexed caches

Physically indexed caches are used in Dec 21064, TI SuperSparc, IBM Power for example. When the virtual-to-physical address translation effectively precedes the cache indexing, using a skewed-associative cache in place of a set-associative cache would not lenghten the overall hit time.

But, on the Dec 21064 and the TI SuperSparc, an artifice has been used in order to allow to execute in parallel the indexing of the cache and the virtual-to-physical address translation. The size of the cache bank is chosen equal to (or inferior than) the minimum size of a page of the virtual memory; on a direct-mapped or a set-associative cache, the bits required for indexing the cache are not translated: virtual-to-physical address translation and cache indexing may be executed in parallel.

Such an artifice cannot be used for skewed-associative caches: in order to enable inter-bank dispersion, some extra bits of th address are used for computing the distinct mapping functions for the distinct cache banks e.g. for computing the skewing functions F_0 and F_1 proposed in section 2.4 for a 8Kbytes two-way skewed-associative cache with a 16 bytes cache line, the 20 lowest significant bits of the address must be known!

5.1.2 Uncomplete address translation

Some recent works [13, 12] have shown that, when the page allocation algorithm is implemented in such a way that virtual-to-physical address translation does not affect the lowest significant bits of the virtual page number, the miss ratio on a physically indexed cache is lower than when usual page allocation algorithm is used.

This result holds for medium range primary caches (16 KB to 512 KB) [13], as well as for large secondary caches [12]. On the other hand, a too large number of untranslated bits would lead to a significant increase of page faults, particularly on low-end microprocessor systems; e.g. on a 8 Megabyte system, having 20 bits untranslated is not realistic.

But keeping 15 or even 18 bits untranslated, even on a low-end system would not lead to a significant increase of the number of page faults:

> Let us consider a 8 Megabytes physical memory and a virtual page size of 4Kbytes, with the usual page allocation, the physical memory acts as a 2048 lines fully-associative cache, while if the 15 lowest significant bits must not be affected by the virtual-to-physical address translation, it would act as a 256-way set-associative cache. It is well known that the behavior of these two structures of caches are very close!

5.2 Partial skewing

In this section, we assume that the page allocation algorithm is implemented in such a way that it does not affect a few of the lowest significant bits of the page number.

If the computation of mapping functions on the cache banks uses only the untranslated address bits, then the artifice described in section 5.1.1 may be used for executing in parallel the indexing of a skewed-associative cache and the virtual-to-physical address translation.

In this section, we show that skewed-associative cache may also perform well in the case where only a limited number of bits are skewed.

A case study Let us consider that the 15 lowest significant bits of the virtual addresses not translated i.e the virtual address and the physical address are equal modulo 32K.

Let us consider the mapping functions defined by:
if $X = 2^{15}X_4 + 2^{12}X_3 + 2^9X_2 + 2^4X_1 + X_0$ then

$$\begin{array}{lll} f_0: & S & \longrightarrow \{0,..,4095\} \\ & (X_4, X_3, X_2, X_1, X_0) & \longrightarrow (X_2, X_1, X_0) \end{array}$$

$$\begin{array}{lll} f_1: & S & \longrightarrow \{0,..,4095\} \\ & (X_4, X_3, X_2, X_1, X_0) & \longrightarrow (X_3 \oplus X_2, X_1, X_0) \end{array}$$

f_0 is the usual bit truncation; f_1 does not change the nine lowest significant bits, and the three highest significant bits are simply obtained a XOR.

f_0 and f_1 may be used as mapping functions for a two-way 8Kbytes skewed-associative cache.

Notice that when using these functions, the inter-bank dispersion of data will only be partial: the set of data that can be mapped (by mapping function f_0) onto a given line in bank 0 is distributed among only 8 lines on cache bank 1 by mapping function f_1.

Simulations were conducted using these skewing functions for 8K bytes two-way skewed-associative caches; miss ratios are reported in figure 25. The miss ratios obtained when using this partial skewing are in the same range as the miss ratios observed when using the complete skewing described in section 2.1.

Our simulation results tend to show that there is no significative hit ratio improvement when increasing the inter-bank dispersion degree over 8 on a two-way skewed-associative cache as there is no significative hit ratio improvement when increasing the associativity degree over 4 or 8 on a set-associative cache.

This result associated with the results presented in [13, 12] may encourage microprocessor designers to implement physically indexed skewed-associative caches and to impose implementing operating systems with page allocation algorithms respecting the 15 or may be 18 lowest significant bits of the virtual address.

6 Conclusion

We have introduced a new multi-bank cache organization: the skewed-associative cache. The two-way skewed-associative cache has the same hardware complexity as a two-way set-associative, but exhibits a miss ratio close to the miss ratio of a four-way set-associative cache:

> A two-way skewed-associative cache must be preferred to a two-way or four-way set associative cache.

Organization	Miss
2-way partial skewed-associative	0.024503
2-way complete skewed-associative	0.024287

Figure 23: unified 8192 bytes cache

Organization	Miss
2-way partial skewed-associative	0.020891
2-way complete skewed-associative	0.020865

Figure 24: two 8192 bytes split caches

Figure 25: Miss ratios for partial and complete skewed-associative caches

Today, microprocessors are built with relatively small on-chip caches. In 1992, 8 Kbytes is the current size for on-chip caches. For this size, miss ratios on direct-mapped caches are significantly higher than on associative caches. In section 4, we have pointed out that, for low-end microprocessor systems, some associativity on on-chip cache(s) will enhance performance: using direct-mapped cache may allow to increase clock frequency, but when the miss penalty becomes high, using a skewed associative cache with a slower clock will lead to a better over all system performance.

Peak performance on high-end systems is a major commercial argument for microprocessor vendors. In high-end systems, large very fast second level caches are used. Performance in these systems highly depends on the clock frequency. In order to reduce the clock frequency, the cache access may be pipelined (e.g. in MIPS R4000 [10] or in DEC 21064[4]) and optimistic execution may be used (e.g. MIPS R4000). Optimistic execution on skewed-associative caches will allow to reach a clock frequency within a few per cent of the clock reachable when using direct-mapped caches. In section 4.2, we have pointed that using such an optimistic execution will lead to slightly better performance when using a two-way skewed associative cache.

At last, in section 5, we have shown that skewed-associative caches may be used for implementing physical caches as well as virtual caches without lenghtening the cache hit time.

As most of the microprocessor chips are designed to built both high-end microprocessor systems and low-end microprocessor systems, using two-way skewed-associative cache structure seems a very interesting trade-off.

References

[1] A. Agarwal, M. Horowitz, J. Hennesy "Cache performance of operating systems and multiprogramming workloads" ACM Transactions on Computer Systems, Nov. 1988

[2] A. Agarwal *Analysis of Cache Performance for Operating Systems and Multiprogramming*, Kluwer Academic Publishers, 1989

[3] J.H. Chang, H. Chao, and K. So "Cache Design of A Sub-Micro CMOS System/370" pp208-213, Proceedings of the 14th International Symposium on Computer Architecture (IEEE-ACM), May 1987.

[4] "DECChip 21064-AA RISC Microprocessor, Preliminary Data Sheet" Digital Equipment Corporation, 1992

[5] J.L. Hennessy, D.A. Patterson *Computer Architecture a Quantitative Approach*, Morgan Kaufmann Publishers, Inc. 1990

[6] M.D. Hill, "Aspects of Cache Memory and Instruction Buffer Performance", Ph.D Thesis, University of Berkeley, 1987

[7] M.D. Hill, "A case for direct-mapped caches", IEEE Computer, Dec 1988

[8] M.D.Hill, A.J. Smith "Evaluating Associativity in CPU Caches" IEEE Transactions on Computers, Dec. 1989

[9] N.P. Jouppi, "Improving Direct-Mapped Cache Performance by the addition of a Small Fully-Associative Cache and Prefetch Buffers" Proceedings of the 17^{th} International Symposium on Computer Architecture, June 1990

[10] G. Kane *MIPS RISC Architecture* Prentice-Hall, 1988

[11] G. Kane, J. Heinrich *MIPS RISC Architecture* Prentice-Hall, 1992

[12] R.Kessler, M. Hill "Miss Reduction in Large Real-Indexed Caches," Technical Report No 940, Dpt of Computer Science, University of Wisconsin-Madison, June 90.

[13] W.L. Lynch, B.K. Bray, M.J. Flynn "The Effect of Page Allocations on Caches" Proceedings of MICRO 25 , December 1992

[14] S.A. Przysbylski "Performance-Directed Memory Hierarchy design" PhD Thesis, Stanford University, 1988

[15] A.J. Smith "A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory" IEEE Transactions on Sofware Engineering, March 1978

[16] A.J. Smith "Cache memories" ACM Computing Surveys, Sept. 1982

[17] A.J. Smith "Line (block) size choice for CPU cache memories" IEEE Transactions on Computers, Sept. 1987

[18] SparcSim Manual, SUN Inc, Dec 1989

[19] "TMS390Z55 Cache Controller, Data Sheet", Texas Instrument, 1992

[20] "Skewed Associative Caches" *same authors*