Skewed-associative Caches *

André Seznec, Francois Bodin
IRISA, Campus de Beaulieu
35042 Rennes Cedex
FRANCE
tel : (33) 99 84 73 36
FAX : (33) 99 38 38 32
e-mail : seznec@irisa.irisa.fr

Abstract. During the past decade, microprocessor peak performance has increased at a tremendous rate using RISC concept, higher and higher clock frequencies and parallel/pipelined instruction issuing. As the gap between the main memory access time and the potential average instruction time is always increasing, it has become very important to improve the behavior of the caches, particularly when no secondary cache is used (i.e on all low cost microprocessor systems). In order to improve cache hit ratios, set-associative caches are used in some of the new superscalar microprocessors.
In this paper, we present a new organization for a multi-bank cache : the skewed-associative cache. Skewed-associative caches have a better behavior than set-associative caches: typically a two-way skewed-associative cache has the hardware complexity of a two-way set-associative cache, yet simulations show that it exhibits approximatively the same hit ratio as a four-way set associative cache of the same size.
Keywords: microprocessors, cache, set-associative cache, skewed-associative cache.

1 Introduction

Performance of commercial microprocessors is increasing at a tremendous rate: 100 Mhz clocks were announced on the MIPS R4000 processor in september 1991, a 200 Mhz clock processor is offered by the end of 1992 by DEC, etc. At the same time, the architecture complexity of microprocessors is also increasing. First generation RISC microprocessors such as the MIPS R2000 or the first SUN Sparc microprocessors rapidly become obsolete. In association with technological advances which have allowed faster and faster clock speeds and larger and larger transistor counts, two major architectural techniques have been used in order to improve performance of the microprocessors: superscalar issuing of the instructions (IBM Power, SUN SuperSparc, Intel i860, etc) and "superpipelining" (i.e. increasing clock frequency by deepening the pipeline as in the MIPS R4000) [6, 15].

Most of the newly introduced commercial microprocessors have been designed to address a very large segment of the market: constructors generally claimed to address low cost embedded systems as well as high end file servers or workstations with the same basic microprocessor architecture.

In order to feed these new microprocessors with both instructions and data, a large memory is needed, and the effective performance of the whole system highly depends of the performance of the memory system. Unfortunately, over the last ten years, the main memory access cycle time has not decreased at the same rate as the processor cycle time. On a superscalar microprocessor such as the IBM Power e.g., the demand on memory instruction throughput may be up to 4 instructions per cycle and the demand on memory data throughput may be very close to one word of data per cycle. When the penalty for a cache miss is about 20 cycles, the performance may dramatically decreased when the number of cache misses increases even very smoothly (Amdhal's law).

Increasing the hit ratio of both data and instruction cache is a key issue in order to improve the effective performance of microprocessors and particularly for low-end systems.

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

In section 2, we propose a new data mapping on a partially associative cache: the skewed-associative cache. Then some properties wished on skewing functions are characterized and a family of "good" skewing functions is presented.

Trace driven simulations results presented in section 3 shows that two-way skewed associative exhibits the same hit ratio as a four-way set associative cache of the same size, but at the hardware cost of a two-way set-associative cache.

2 Skewed-associative caches

2.1 Skewing on caches: principle

A set-associative cache is illustrated in Figure 1: a X way set-associative cache is built with X distinct banks of $\frac{L}{X}$ cache lines. Then a line of data with base address D may be physically mapped on physical line $f(D)$ in any of the distinct banks.

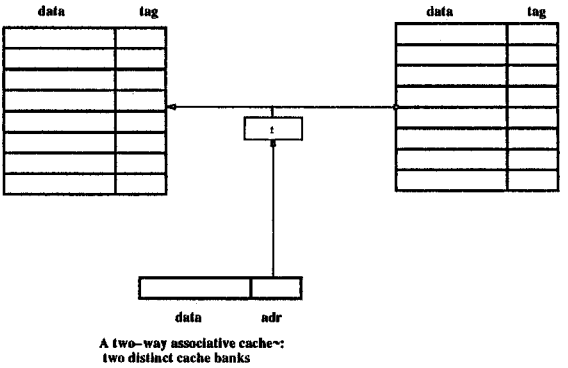

Fig. 1. An alternative vision of a two-way associative cache

We propose a very slight modification to this cache organization illustrated in Figure 2:

Different mapping functions are used for the distinct cache banks i.e., line of data with base address D may be mapped on physical line $f_0(D)$ in cache bank 0, 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 is very slight. It will induce a very marginal hardware upper cost when designing a new microprocessor with on-chip caches: we may choose f_i which may be implemented with 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.

Note that skewed-associative caches may be used for internal primary caches as well as for external caches (primary or secondary).

Related works In 1977, Smith [12] considered set-associative caches and proposed to select the set by hashing the main memory address; this approach corresponds to figure 1: a *single* hashing function is used.

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

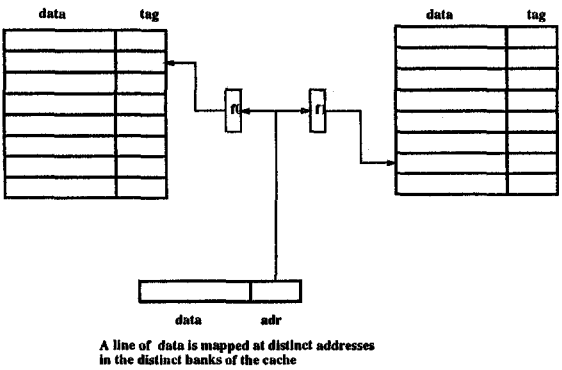

Fig. 2. A two-way skewed-associative cache

As, in a conventional cache, the address indexes into a cache set, and data is available to the processor if it is present in that 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 then the data must be fetched from memory.

Hash-rehash caches present better overall hit ratios than direct-mapped caches approaching overall hit ratios for two-way set-associative caches, but Agarwal [2] shows that hash-rehash caches induces 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 hit, while in skewed-associative caches, different hashing functions are used for indexing *at the same time* the distinct cache banks.

2.2 Choosing skewing functions

In this section, we give some highlight on properties that might exhibit functions chosen for skewing the lines in the distinct cache banks in order to enable a good hit ratio. First we give some notations used:

- D is a line of data in main memory. For sake of simplicity, we shall also note D the address of the first byte in the line.
- f_i is the mapping function from S to CL_i.

Equitability First of all, for each line in the cache, the numbers of lines of data in the main memory that may be mapped on the cache line must be equal.

Inter-bank dispersion On classical associative set caches, when two lines of data would be mapped on the same set in the cache, they are conflicting for the same place in the X cache banks.

We have introduced skewed-associative caches to avoid this situation by scattering the data: we may chose mapping functions such that when 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.

These two situations are illustrated in figure 3 and figure 4.

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

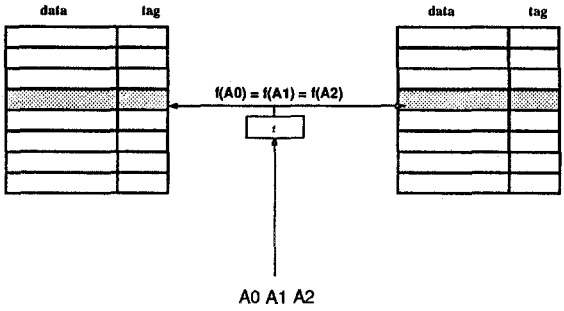

Fig. 3. 3 data conflicting for a single set on a two-way set-associative cache

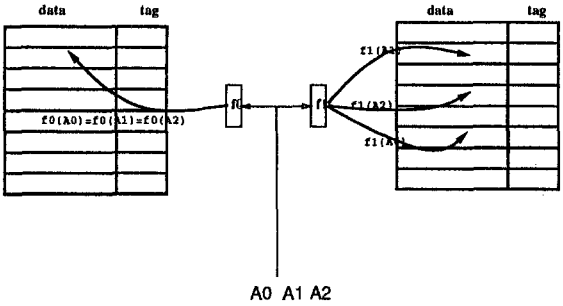

Fig. 4. Data conflicting for a cache line on bank 0 but not on bank 1

Local dispersion in a single bank Many applications exhibit spacial locality, then mapping functions must be chosen in order 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.

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, if these computations can be added to a non critical stage in the pipeline and can be computed without lengthening 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 a single cycle).

In order to achieve this, we have to chose mapping functions whose hardware implementation are very simple: a very few gates delay if possible.

2.3 A family of mapping functions

The family of mapping functions presented in this section exhibits most of the properties listed previously. The family of mapping is based on manipulations on bit strings in addresses of data.

From now, we consider that a cache bank is built with 2^n cache lines of 2^c bytes. We also suppose that the memory consists in 2^q bytes and that $q \geq 2 * n + c$.

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 bits 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 $(y_n, y_{n-1}, .., y_1)$ be the binary representation of $Y = \Sigma_{i=1,n} y_i 2^{i-1}$. Let us consider the function H defined as follows:

$$\begin{aligned} H : \{0, .., 2^n - 1\} &\longrightarrow \{0, .., 2^n - 1\} \\ (y_n, y_{n-1}, .., y_1) &\longrightarrow (y_n \oplus y_1, y_n, y_{n-1}, .., y_3, y_2) \end{aligned}$$

where $\oplus$ is the XOR (exclusive or) operation.
We consider the four mapping functions defined respectively by[2]:

$$\begin{aligned} f_0 : S &\longrightarrow \{0, .., 2^n - 1\} \\ (A_3, A_2, A_1, A_0) &\longrightarrow H(A_1) \oplus H^{-1}(A_2) \oplus A_2 \end{aligned}$$

$$\begin{aligned} f_1 : S &\longrightarrow \{0, .., 2^n - 1\} \\ (A_3, A_2, A_1, A_0) &\longrightarrow H(A_1) \oplus H^{-1}(A_2) \oplus A_1 \end{aligned}$$

$$\begin{aligned} f_2 : S &\longrightarrow \{0, .., 2^n - 1\} \\ (A_3, A_2, A_1, A_0) &\longrightarrow H^{-1}(A_1) \oplus H(A_2) \oplus A_2 \end{aligned}$$

$$\begin{aligned} f_3 : S &\longrightarrow \{0, .., 2^n - 1\} \\ (A_3, A_2, A_1, A_0) &\longrightarrow H^{-1}(A_1) \oplus H(A_2) \oplus A_1 \end{aligned}$$

Property 2.1 *As H is a bijection, the mapping of data using functions f_1, f_2, f_3 and f_4 is equitable.*

Property 2.2 *It is easy to prove that this set of functions f_1, f_2, f_3 and f_4 exhibits the dispersion property previously defined when the function $H^2 \oplus H \oplus Id$ is a bijection. $H^2 \oplus H \oplus I$ is a bijection for n=3,4,6,7,9,10,12,13,15,16*[3].

Property 2.3 *As H and $H \oplus Id$ are bijections, the local dispersion of data in a cache bank is quite optimum: for any cache line C, in any set L of $K * 2^n$ consecutive slices of data, there are at most $K + 1$ lines of data in L conflicting for the physical line C in the cache.*

Hardware implementation The hardware needed for implementing the proposed mapping functions is very simple:

each bit of $f_i(A)$ is deduced from the binary decomposition of A by XORing at most four bits!

Moreover, the four mapping functions may be implemented with similar hardware parts implementing $H(x) \oplus H^{-1}(y) \oplus z$ where x, y and z are chains of n bits (figure 2.3).

---

[2] a line in main memory is represented by the address of its first byte

[3] Using a slightly distinct basic function H allows to obtain the dispersion property for n= 5, 8, 11, 14.

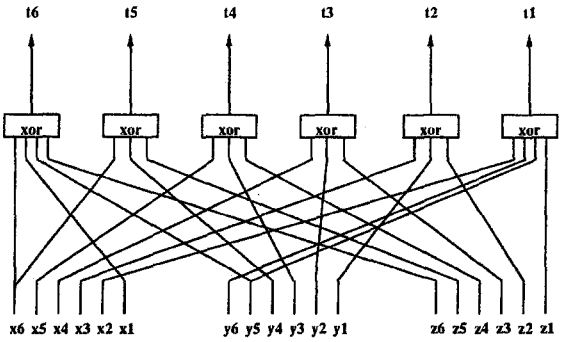

Fig. 5. Computing the skewing functions in hardware

2.4 An example

On a very simple example, we try to illustrate the benefits that can be expected when using a X-way skewed-associative cache in place of a X-way set-associative cache.

We give here an example with X=4 cache banks consisting of 8 lines. For sake of simplicity, a cache line of one word was assumed. A sequence of 32 addresses was randomly generated in $\{0, ..., 2^{20} - 1\}$. Let us assume that this set of data is the working set of an application.

The difference between the classical set-associative cache and the skewing mapping is illustrated in Figure 9[4]:

1. **Data dispersion:**
   Let us consider that the sequence of references is issued only one time:
   - In the set-associative cache, on set 3, 8 data are conflicting, then four data must be rejected from the cache (cf Figure 6).
   - In the skewed-associative cache, 8 data are conflicting for location 1 in bank 1, but as the same data are not conflicting on the other banks, more data may be alive in the cache at the same time: 28 words after one reference to each data (cf Figure 7).

   A better dispersion of data among the cache lines is achieved on skewed-associative caches.
2. **Self data reorganization:**
   Let us now consider that the sequence of references is issued many times, as if this sequence corresponds to the working set of an application:
   - In the set-associative cache, during the whole application, only 25 words will be alive at the same time in the cache. In each set, there is no change to the number of alive cache lines compared with the Figure 6.
   - In the skewed-associative cache, the number of data present at the same time in the cache depends on the precise mapping of each data in the cache: among the other possible locations for a data D present in the cache at time t, there may be an empty location; the data D may be removed from the cache by a miss on an other data D'; in this case, the next time D will be referenced, D can be mapped in the empty location and thus the number of data alive at the same time in the cache can increase. In our example, all the data become alive at the same time after 22 references to each data (cf Figure 8).

[4] Number of data conflicting for a single location are indicated in bold for each set in the set-associative cache and for each location in the skewed-associative cache.

Set	Bank 0	Bank 1	Bank 2	Bank 3
0 (3)	311792	371000	590664	xx
1 (5)	869201	882529	411905	770697
2 (3)	696578	953178	324610	xx
3 (8)	544923	159243	76507	1007147
4 (2)	727204	749916	xx	xx
5 (1)	639421	xx	xx	xx
6 (5)	761790	298390	770462	234166
7 (5)	278911	1043919	246639	143631

Fig. 6. Mapping on a set-associative cache: after a single reference to each data

Address	Bank 0	Bank 1	Bank 2	Bank 3
0	770462 (2)	311792 (3)	411905 (3)	502185 (3)
1	544923 (5)	76507 (8)	546971 (6)	297638 (7)
2	278911 (3)	1043919 (5)	371000 (6)	810219 (7)
3	159243 (3)	727204 (2)	882529 (4)	xx (3)
4	696578 (7)	xx (1)	246639 (4)	937059 (2)
5	234166 (2)	761790 (5)	xx (2)	xx (4)
6	639421 (4)	953178 (3)	298390 (2)	590664 (5)
7	1007147 (6)	869201 (5)	143631 (5)	770697 (1)

Fig. 7. Mapping on a skewed-associative cache: after a single reference to each data

Address	Bank 0	Bank 1	Bank 2	Bank 3
0	770462 (2)	311792 (3)	696578 (3)	502185 (3)
1	544923 (5)	546971 (8)	953178 (6)	76507 (7)
2	278911 (3)	1043919 (5)	701631 (6)	1007147 (7)
3	159243 (3)	749916 (2)	882529 (4)	727204 (3)
4	761790 (7)	639421 (1)	246639 (4)	937059 (2)
5	297403 (2)	297638 (5)	234166 (2)	371000 (4)
6	411905 (4)	324610 (3)	298390 (2)	590664 (5)
7	810219 (6)	869201 (5)	143631 (5)	770697 (1)

Fig. 8. Mapping on a skewed-associative cache: after 22 references to each data

Fig. 9. An example of data mapping

The behavior of the skewed-associative cache, illustrated by the example, should enhance performance of blocked algorithms that iterate computations on small working set smaller than the cache size in which interference misses may degrade performance a lot [8].

2.5 Which replacement policy for skewed-associative cache

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

LRU replacement policy is generally considered as the most efficient policy. Implementing a LRU replacement policy on a two-way associative cache is quite simple. A single bit tag per cache line is sufficient: when a line is accessed, this tag is asserted and the tag of the second line of the set is deasserted. More generally a LRU replacement policy for a X-way associative is feasible with adding only X bit tags to each line.

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: as the set of lines on which a line has to be replaced vary with the new line to be introduced, the information needed in order to determine the last referenced line in the set is the complete date of the reference.

Using a pseudo-random policy replacement generally induces more misses on caches than using a LRU replacement policy. We propose here a very simple replacement policy which requires only one tag bit per cache line and for which we have experimentally obtained very interesting behavior:

- The bit tag RU (Recently Used) is asserted when the cache line is accessed
- Periodically the bit tags RU of all the cache lines are zeroed: we experimentally determine that a good period is each $\frac{\text{cache size in bytes}}{4}$ accesses to the cache.

When a line misses in the cache, the replaced line is chosen among the X possible lines in the following priority order:

1. Randomly among the lines for which the RU tag is deasserted
2. Randomly among the lines for which the RU tag is asserted, but which have not been modified since they have been loaded in the cache[5]
3. Randomly among the lines for which the RU tag is asserted and which have been modified.

This replacement policy is quite simple to implement in hardware. An interesting property of this replacement policy is to limit the copy back of data on the main memory (or the secondary cache) and then to limit the traffic on memory.

We call this replacement policy: Not Recently Used Not Recently Written (NRUNRW).

3 Simulations

In the previous section, we have pointed out that there is a potential improvement on performance of multi-way caches when skewing addresses.

In order to verify that skewing addresses on multi-bank caches will improve the performance on effective applications, we have simulated the primary cache behavior on traces generated by 7 medium size applications (range from half a million data references to 12 millions references). This set was composed with :

1. OPACgen : a microcode generator for a hardware prototype of floating-point microcoded coprocessor
2. RESEAU : a simulator of a specific interconnection network
3. cptc : A Pascal-to-C translator
4. Cache : the cache simulator itself
5. Poisson : a Poisson solver
6. Sparse : a sparse matrix-vector multiply
7. Mulmat : a matrix multiply (60*60 by 60*60)

The first 4 applications are standard C applications. The last three applications are numeric applications.

The traces were generated by using the Abstract Execute software [9] targeted for a SPARC processor. Unfortunately system calls such as fprintf, fclose, etc, were not traced, neither exception managements, then the effective instruction miss ratios would certainly be higher than those obtained in our simulations (for results on influence of operating systems on cache performance see [1]).

[5] For the instruction cache, there no third choice

A single process execution was supposed : performance of caches in a time-sharing environment would be certainly worse than the results shown here [3, 10].

Simulations results have been normalized in order to give the same relative weight to each of the benchmarks. Only the geometric mean of all benchmarks is presented here since there is no significative dispersion of the results over the different programs. However it should be noted that numeric applications show, as expected, less different behaviors between set-associative cache and skewed-associative cache.

Some comments on simulations

Results presented in this paper are given for a cache line size of 64 bytes: other cache line sizes (16, 32 and 128 bytes) were also simulated, but 64 bytes was the size which gave globally the better results in terms of hit ratios on our set of benchmarks (this is coherent with results presented in [14]).

In [7], Jouppi pointed out that a significant improvement of the hit ratio of a direct-mapped cache may be obtained by adding a small fully associative cache (called a victim cache) in order to store the last lines rejected from the major primary cache.

In order to compare the results with on multi-bank caches with direct-mapped cache, such a victim buffer of two lines was simulated for all the configurations: as stated in [7], it improves significantly direct-mapped cache hit ratio. When a cache miss induces a hit in the victim cache, it does not induce any access to the main memory or secondary cache, they are not considered as misses in the rest of the paper.

Simulation results

Miss ratios on respectively the data cache and respectively the instruction cache on our benchmarks set are given respectively in Table 1 and Table 2. Cache sizes from 4096 bytes to 16384 bytes have been simulated; the replacement policies that were simulated are pseudo-random, LRU and NRUNRW (see section 2.5).

Cache Size (bytes)	4096	8192	16384
Direct-mapped	0.076040	0.062770	0.046019
LRU Standard 2 banks	0.063827	0.052571	0.040727
LRU Standard 4 banks	0.051429	0.041835	0.028765
LRU Standard 8 banks	0.048802	0.040838	0.027502
NRUNRW Standard 2 banks	0.065198	0.051921	0.040034
NRUNRW Standard 4 banks	0.053915	0.041629	0.028182
Random Standard 2 banks	0.065672	0.052012	0.039693
Random Standard 4 banks	0.055918	0.042340	0.028900
LRU Skewed 2 banks	0.050388	0.040764	0.027134
LRU Skewed 4 banks	0.048278	0.040066	0.025842
NRUNRW Skewed 2 banks	0.051740	0.040962	0.027514
NRUNRW Skewed 4 banks	0.049648	0.039819	0.026245
Random Skewed 2 banks	0.054219	0.042352	0.028633
Random Skewed 4 banks	0.053011	0.041384	0.027987

Table 1. Data cache miss ratio

These tables clearly show that a two banks skewed-associative cache has a behavior close to a four way set-associative cache. The behavior of a four banks skewed-associative cache is slightly better than the behavior of a four-way associative cache and seems close to the behavior of an eight-way set associative cache, but as pointed out previously the decreasing of the miss ratio obtained with an eight-way set-associative cache beside a four-way set-associative is quite marginal.

Cache Size (bytes)	4096	8192	16384
Direct-mapped	0.022611	0.008704	0.004133
LRU Standard 2 banks	0.016931	0.005847	0.001499
LRU Standard 4 banks	0.013598	0.006238	0.000538
LRU Standard 8 banks	0.011801	0.002425	0.000332
NRUNRW Standard 2 banks	0.016439	0.006238	0.001573
NRUNRW Standard 4 banks	0.013399	0.003748	0.000528
LRU Skewed 2 banks	0.012600	0.003378	0.001044
LRU Skewed 4 banks	0.011187	0.001726	0.000276
NRUNRW Skewed 2 banks	0.013893	0.003423	0.000968
NRUNRW Skewed 4 banks	0.012245	0.001983	0.000304
Random Skewed 2 banks	0.014433	0.003748	0.001158
Random Skewed 4 banks	0.013117	0.002562	0.000454

Table 2. Instruction cache miss ratio

Normalize execution time

In order to illustrate the influence of the miss ratio on the performance, we define the *normalized execution time* as the ratio of the effective execution time on the execution time which would have been reached if there was no time penalty on cache misses.

Definition 3.1 *For sake of simplicity to illustrate the stress on the caches, we shall assume that the* **normalized execution time** T *of an application is defined by:*

$$T = 1 + (1.5 * \tau_I + 0.5 * \tau_D) * LAT \quad (1)$$

where τ_I is the miss ratio of the instruction cache and τ_D is the miss ratio of the data cache.

Formula 1 corresponds to an approximately realist execution on a superscalar microprocessor:

- an average of 1.5 instructions executed per cycle,
- an average of 1 load or store each 2 cycles.

The Figure 10 illustrates the potential benefit of using skewed-associative caches in terms of normalized execution times (see Definition 3.1) for different cache sizes and assuming equal sizes of the two caches:

- There is marginal performance benefit in using a partially associative cache organization when the main memory latency is small (e.g. 5 cycles), but when the memory latency becomes higher, using a partially associative cache organization allows very interesting performance improvement: for 8192 bytes and a 20 cycles memory latency, the normalized execution times vary from 1.46 to 1.89 for respectively a four banks skewed associative cache and the direct mapped cache [6].
- Using a four-way skewed associative cache in place of a classical four-way set associative cache improves performance: about 5% for a cache size of 8192 bytes and a 20 cycles memory latency.
- Using a two-way skewed associative cache seems very attractive: approximately equivalent performance as on a classical four way set-associative is obtained i.e. about 11 % performance improvement on a classical two-way set-associative for a cache size of 8192 bytes and a 20 cycles memory latency.

[6] A victim cache [7] of two lines is supposed for all configurations

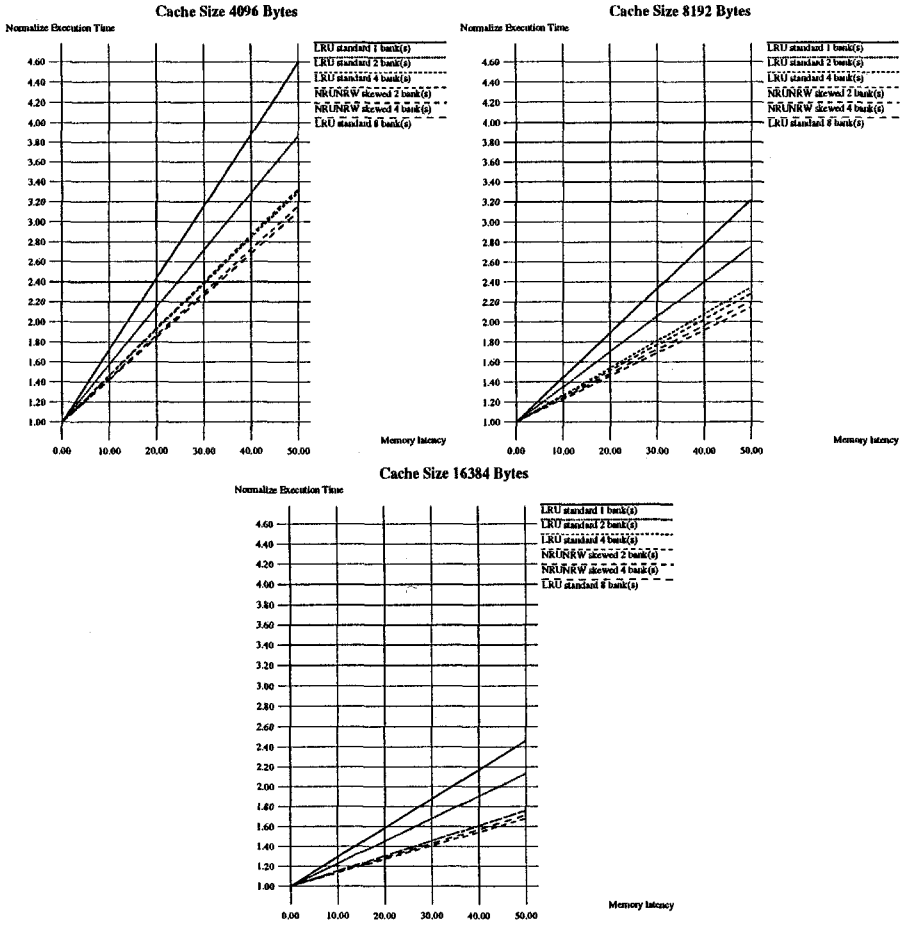

Fig. 10. Normalized Execution Times for different cache sizes

4 Conclusion

During the past decade, microprocessors potential performance has increased at a tremendous rate using RISC concept, higher and higher clock frequencies and parallel instruction issuing. On the other hand, larger and larger main memories are needed in order to feed microprocessors with both data and instructions. But the main memory access time has not decreased at the same rate. Then *effective* performance of a microprocessor on an application essentially depends on the behavior of the whole memory hierarchy: primary instruction and data caches, secondary caches (when available) and main memory system.

As the gap between the main memory access time and the potential average instruction time is always in-

creasing, it has become very important to improve the behavior of the caches, particularly when no secondary cache is used (i.e on all low cost microprocessor systems). In order to improve cache hit ratios, set-associative caches are used in some of the new microprocessors (IBM Power, TI SuperSparc, Motorola 88110).

Set-associative caches are build with separate cache banks; a line of data may be mapped on any of the cache banks, but at the same address in the cache bank. The design of a X-way skewed-associative cache is obtained by a very slight modification of the design of X-way set-associative cache: each line of data has one possible location in any of the cache banks, but the addresses of these possible locations are different in the different cache banks. These different addresses are computed by skewing the addresses.

We have presented a family of skewing functions that exhibits interesting properties and particularly the dispersion property (lines conflicting for the same location in a cache bank are equitably distributed when mapped on another cache bank) and simple implementation hardware implementation (only a few XOR gates).

Simulations have shown that skewed-associative caches have a better behavior than set-associative caches: typically a two-way skewed-associative cache exhibits the same hit ratio as a four-way set-associative cache with the same number of cache lines, but has the same hardware complexity as a two-way set-associative cache.

Simpler skewing functions for the particular case of two-way skewed associative cache may be found in [11].

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. T.E. Anderson, H.M. Levy, B.N Bershad, E.D. Lazowska "The interaction of architecture and operating system design" Proceedings of ASPLOS IV, April 1991
4. M.D. Hill, "A case for direct-mapped caches", IEEE Computer, Dec 1988
5. M.D.Hill, A.J. Smith "Evaluating Associativity in CPU Caches" IEEE Transactions on Computers, Dec. 1989
6. N.P. Jouppi, D.W. Wall "Available instruction-level parallelism for superscalar and superpipelined machines " Proceedings of ASPLOS III, April 1989
7. 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
8. M. Lam, E. Rothberg and M. Wolf, "The Cache Performance and Optimizations of Blocked Algorithms", Proceedings of ASPLOS IV, April 91
9. J.R.Larus, "Abstract execution: a technique for Efficiently Tracing Programs" Technical Report, Computer Sciences Departement, University of Wisconsin-Madison, May 1990
10. J.C. Mogul, A. Borg "The effect of context switches on cache performance" Proceedings of ASPLOS IV, April 1991
11. A. Seznec, "A case for two-way skewed-associative caches", Proceedings of the 20^{th} International Symposium on Computer Architecture, May 1993
12. 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
13. A.J. Smith "Cache memories" ACM Computing Surveys, Sept. 1982
14. A.J. Smith "Line (block) size choice for CPU cache memories" IEEE Transactions on Computers, Sept. 1987
15. M.D. Smith, M. Johnson, M.A. Horowitz "Limits on multiple instruction issue" Proceedings of ASPLOS III, April 1989