Skewed-associative caches

André Seznec, François Bodin

UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

1992

25ème anniversaire

N° 1655

Programme 1
Architectures parallèles, Bases de données,
Réseaux et Systèmes distribués

SKEWED-ASSOCIATIVE CACHES

André SEZNEC
François BODIN

Avril 1992

RR_1655

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Télex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Skewed-associative Caches

André Seznec, Francois Bodin
Programme 1
Projet CALCPAR

20 mars 1992
Publication Interne n° 645 - Mars 1992 - 20 pages

Antémémoire associative brouillée

Résumé

Depuis 1980, les performances potentielles des microprocesseurs ont cru de manière exponentielle utilisant le concept RISC, des fréquences d'horloge de plus en plus élevées ainsi que le séquencement pipeline d'instructions en parallèle.

Le fossé entre le temps d'accès à la mémoire principale et le temps de cycle du processeur se creuse de plus en plus, il devient donc de plus en plus important d'améliorer le comportement des antémémoires, en particulier quand un seul niveau d'antémémoire est utilisé (c-à-d sur tous les systèmes, excepté les systèmes haut de gamme). Des antémémoires associatives par ensemble sont utilisées dans la plupart des nouveaux microprocesseurs superscalaires.

Dans cet article, nous présentons une nouvelle organisation d'antémémoire multi-banc: l'antémémoire associative brouillée. L'antémémoire associative brouillée présente un meilleur comportement qu'une antémémoire associative par ensemble: de manière typique, une antémémoire associative brouillée à deux bancs présente le même taux de succès qu'une antémémoire associative par ensemble à quatre bancs, mais a la complexité matérielle d'une antémémoire associative par ensemble à deux bancs.

Abstract

During the past decade, microprocessors potential 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 most 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-banks skewed-associative cache has the hardware complexity of a two-way set-associative cache, but exhibits the same hit ratio as a four-way set associative cache of the same size.

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 planned current 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) [JOU89, SMI89].

All the 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 (on current low-end IBM Risc 6000 workstations), 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.

In section 2, we recalled how caches are generally organized. Then we explain why improving cache hit ratios has became so crucial.

In section 3, 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.

Simulations results presented in section 4 shows that two-banks 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 Why such a stress on caches?

"Largé' main memory can be built at relatively low cost[1]. But these memories have a long latency access time compare to the microprocessor cycle.

Fast cache memories are used in order to avoid paying the latency penalty when the application exhibits some temporal locality (data or instructions are used several times in a "short" delay) or some spatial locality (data or instructions stored at consecutive addresses have strong probability to be accessed with "short" inter-access delays).

On the new RISC microprocessors (MIPS R4000, SUN SuperSparc, Intel i860, ..), primary caches and the CPU lies on the same chip: there is no mean to extend the sizes of these primary caches with external static RAM chips.

In Figure 1, we illustrate the two basic microprocessors based systems. High-end microprocessors based systems will be built with a secondary external cache: a miss on the primary cache will be served in a few cycles (may be 3-5) if 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 cost of a miss will be very high (20 or even 50 CPU cycles).

[1] One must remember that the memory of the Cray 1 was only 8 megabytes

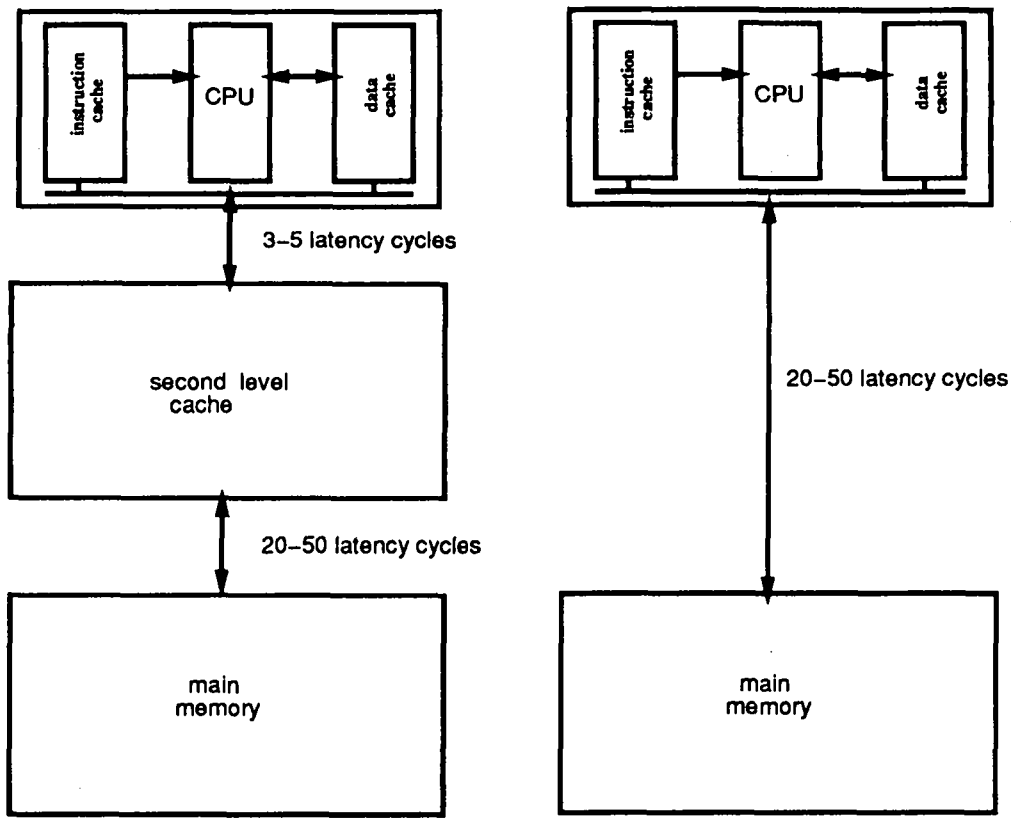


Figure 1: Basic implementations of microprocessor based systems

2.1 Cache organization

Definition 2.1 *In the following we denote* **L** *the number of line in a cache,* **SL** *the length in byte of a cache line,* **X** *the number of banks in the cache (or the degree of associativity),* **S** *the set of data lines in the main memory,* $\mathbf{CL_i}$ *the set of lines in cache bank i. The cardinal of a set is be denoted* ||set||. *The memory latency is denoted* **LAT**.

A cache is divided in a set of L lines of SL bytes of data, as illustrated in Figure 2. SL is generally referred to as the length the cache line. $SL * L$ is the size of cache. To retrieve data in the cache, some complementary information is also stored with the data: tags are needed in order to determine the effective address of the line of data in memory, some information is also needed for checking the data validity, etc.

Organizations of caches in different processors mainly differ with their degree of associativity. For example, very small caches may be fully associative i.e. each line of data may be mapped on any physical line of the cache (cf Figure 2).

At the other extremity, the cache may be direct mapped i.e. a line of data can be mapped on only one physical line in the cache, the number of this line is deduced from the address of the data, usually by taking the lower significant bits in the address (cf Figure 3).

At equal sizes, a fully associative cache generally exhibits a higher hit ratio than a direct-mapped cache, by avoiding conflict misses that is data mapped on the same address in the cache.

But implementing a large fully associative cache is not quite realistic because it would require a huge volume of transistors and because the access time would be unacceptable: in each physical line, the address

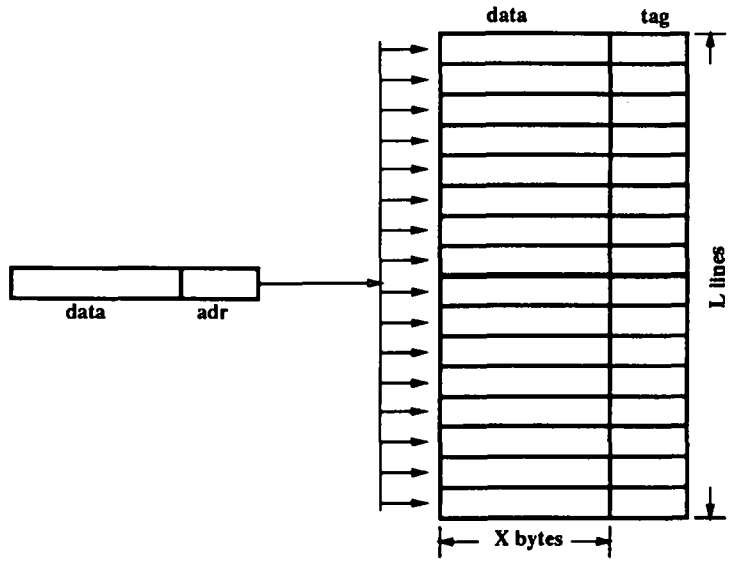


Figure 2: Data mapping on a fully associative cache

tags have to be read and compared with the address to be accessed.

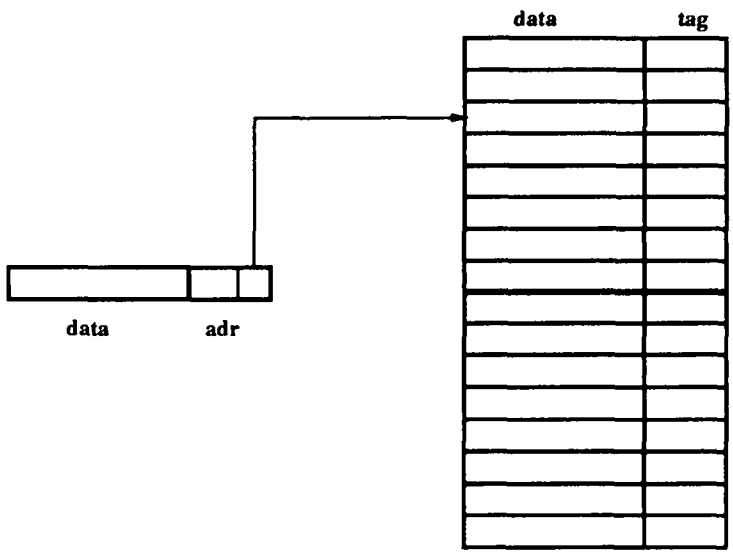
On the other hand, the direct-mapped organization of the cache is very simple: a single word and its associated tags are read, then the address tags are checked against the address. But in some applications direct-mapped caches present a very bad hit ratio due to conflicts in the data mapping onto the cache. For example, a ping-pong phenomenon may occur in the following loop when base addresses for arrays A and B have the same least significant bits:

$$\begin{array}{l} for\ (i = 0;\ i < n;\ i++) \\ \quad A[i]\ =\ B[i] + C[i]; \end{array}$$

In order to avoid some of these conflict misses, set-associative caches are used in many microprocessor designs (IBM Power uses a four way associative data cache, Intel i860 uses a two way associative data cache, etc). In a X-way set-associative cache, the physical lines are grouped in $\frac{L}{X}$ sets and a line of data may be mapped on any line in one of this set (cf Figure 4). The number of this set is determined by the address of the line: generally the number of the set is directly deduced from the least significant bits of the address.

Computer architects generally agree (and experiments show that) that there is no significant improvement of the hit ratio when increasing the degree of associativity over 4 or 8 [HIL89]. As the complexity of the implementation of the cache increases with the degree of associativity, the associativity degree of the current commercial microprocessors are in 1 (direct mapped caches), 2 or 4. But there is still an open debate on whether or not the cache has to be set-associative:

- Implementing direct-mapped cache minimizes the transistor count. It also minimizes the access time and possibly allows faster clock when the cache access time is equal to one cycle time.
- Set-associative caches generally exhibit higher hit ratios; then the pipelines are less often stalled.



Mapping of the line of data is deduced from the address.

Figure 3: Data mapping on a direct-mapped cache

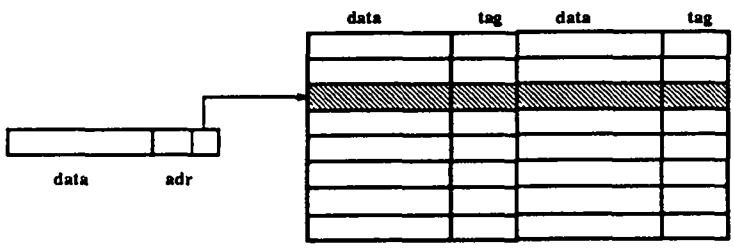
2.2 Increasing miss ratio = decreasing performance

In order to illustrate the direct 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 2.2 *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.*



A line of data may be mapped in any of the lines in the set.

Figure 4: Data mapping on a set-associative cache: a line of data may be mapped on any of the lines in the set.

Definition 2.2 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.

Figure 5 illustrates the normalized execution time for some values of τ_I and τ_D. When no secondary cache is used, the execution time may be dominated by the service of cache misses and thus any decreasing of the miss ratio on the caches will allow to increase the system performance by approximately the same factor.

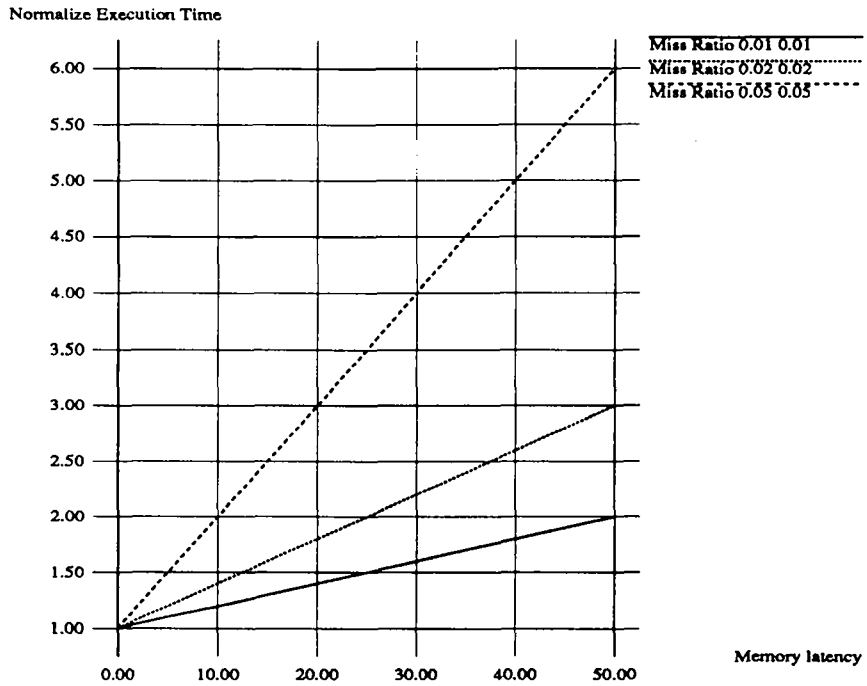


Figure 5: Normalized execution times for $\tau_I = \tau_D =$ equal 0.01, 0.02 and 0.05

2.3 Stress on main memory throughput

Formula (1) does not entirely reflect the stress on memory bandwidth that can occur when the memory latency is high:

it does not include the write accesses for updating data in the main memory.

Two alternative strategies are used for updating data in the main memory:

1. **Write through:** on every store, the word of data is written on both the cache and the main memory.
2. **Write back:** on a store, the word of data is only written in the cache. The line of data is updated in the main memory, when the line is flushed from the cache.

The write through strategy is most simple to implement, but it rapidly saturates the main memory bandwidth:

When considering one load/store instruction per 3 instructions, one store per 3 load/store instructions, at least one write on the main memory is executed when issuing 9 instructions. When the main memory access latency is about 20 or 50 cycles, it automatically limits the performance.

In consequence, all new generation microprocessors have been implemented with a Write back strategy.

In order to measure the demand on memory throughput on a microprocessor, we define the normalized memory busy time as the ratio besides the effective busy time of the main memory on the execution time which would have been reached with an infinite size cache.

Definition 2.3 *The normalized memory busy time of an application is defined by:*

$$T = (1.5 * \tau_I + 0.5 * \tau_D * (1 + \tau_{WB})) * Mdel \quad (2)$$

where τ_I is the miss ratio of the instruction cache, τ_D is the miss ratio of the data cache, and τ_{WB} is the ratio of data misses inducing the write back of the replaced line. Mdel is the main memory busy time for the access to a line of data (i.e. the latency LAT plus the delay for obtaining the whole line).

When formulae (1) and (2) give close results, formula (1) is quite optimistic: the effective normalize execution time is certainly higher.

Minimizing main memory busy time is important in order to be able to built single-bus shared-memory multiprocessors.

3 Skewed-associative caches

3.1 Skewing on caches: principle

An alternative vision of a set-associative cache is illustrated by Figure 6: 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. This vision of a set-associative cache fits with its physical implementation: X banks of static memory RAMs.

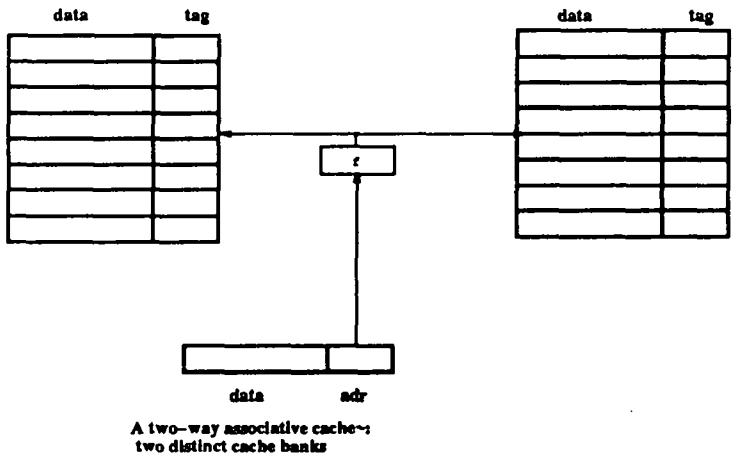


Figure 6: An alternative vision of a two-way associative cache

On this vision of a set-associative cache, we propose a very slight modification illustrated in Figure 7:

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

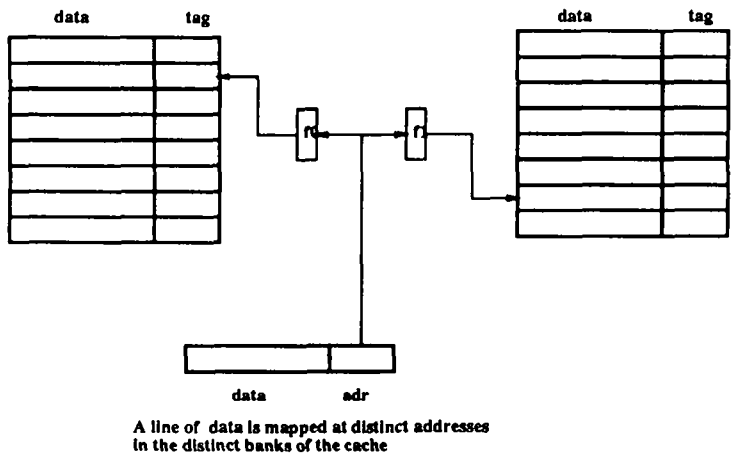


Figure 7: A two-banks skewed-associative cache

cache with such a lines mapping on the distinct banks: a **skewed-associative cache**.

This hardware modification is very slight and induces a very marginal hardware upper cost; but we shall see that this may help to increase the hit ratio of a data cache and then to increase the overall performance of a microprocessor using a multi-bank cache structure at a very marginal hardware cost: we may choose f_i which may be implemented with a very few gates.

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

About skewing schemes

Many authors [BUD71, FRA85, HAR86, HAR87, NOR87, RAU89] have studied the use of skewed storage schemes in order to improve the throughput of interleaved parallel memories on particular pipelined or parallel access patterns (e.g. constant stride vectors) on vector or parallel supercomputers. The goal is to distribute data among the distinct relatively "slow" memory banks in order to allow parallel accesses to a specific structure of data.

Here, the goal is quite different; in current state-of-the-art microprocessors, data caches have been designed in order to deliver data to the processors at the rate of one per cycle. Additional banks have been added in the cache for increasing hit ratio, but when successive accesses hit in the same cache bank, no cycles are hidden.

But if the goals of address skewing in the two cases are quite different, the choice of the skewing functions may be quite similar.

3.2 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.

3.2.1 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. In more mathematical terms a mapping function is equitable when:

$$\forall\, C \in CL_i, \|f_i^{-1}(C)\| = \frac{\|S\|}{\|CL_i\|}$$

3.2.2 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. 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. In more mathematical terms dispersion is achieved when:

$$\forall\, D \in S,\ \forall\, i \neq j\ ,\ \|\{d \in S / f_i(D) = f_i(d)\ \&\ f_j(D) = f_j(d)\}\| = \frac{\|S\|}{\|CL_i\|^2}$$

3.2.3 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.

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

3.3 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:

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

where $\oplus$ is the XOR (exclusive or) operation.

We consider the four mapping functions defined respectively by[2]:

$$
\begin{array}{lll}
f_0: & S & \longrightarrow \{0,..,2^n-1\} \\
 & (A_3,A_2,A_1,A_0) & \longrightarrow H(A_1)\oplus H^{-1}(A_2)\oplus A_2 \\
 & & \\
f_1: & S & \longrightarrow \{0,..,2^n-1\} \\
 & (A_3,A_2,A_1,A_0) & \longrightarrow H(A_1)\oplus H^{-1}(A_2)\oplus A_1 \\
 & & \\
f_2: & S & \longrightarrow \{0,..,2^n-1\} \\
 & (A_3,A_2,A_1,A_0) & \longrightarrow H^{-1}(A_1)\oplus H(A_2)\oplus A_2 \\
 & & \\
f_3: & S & \longrightarrow \{0,..,2^n-1\} \\
 & (A_3,A_2,A_1,A_0) & \longrightarrow H^{-1}(A_1)\oplus H(A_2)\oplus A_1
\end{array}
$$

Property 3.1 *As H is a bijection, the mapping of data using functions f_1, f_2, f_3 and f_4 is equitable.*

Property 3.2 *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 $H^2 \oplus H \oplus I$ 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 3.3 *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 the same hardware part implementing $H(x) \oplus H^{-1}(y) \oplus z$ where x, y and z are chains of n bits.

3.4 An example

In order to illustrate how skewing may improve the performance of a multi-bank cache, we give here an example with X=4 cache banks. For sake of simplicity, a cache line of one word was assumed. A sequence of 32 addresses was randomly generated in $\{0,...,2^{20}-1\}$. The difference between the classical set-associative cache and the skewing mapping is illustrated in Figure 11. The mapping of data in the cache is given for two cases[4]:

1. The sequence of references is issued 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 8).
   - 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 9).
2. The sequence of references is issued many times, as if this sequence corresponds to the working set of an application:

[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.

[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.

- 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 compare with the Figure 8.
- 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 10).

The behavior of the skewed-associative cache, illustrated by the example, should enhance performance of blocked algorithms that iterate computations on small working set, in which interference misses may degrade performance a lot [LAM91].

3.5 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 memory busy time.

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

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

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

Figure 8: 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)

Figure 9: 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)

Figure 10: Mapping on a skewed-associative cache: after 22 references to each data

Figure 11: An example of data mapping

4 Simulations

In the previous section, we have pointed out that there is a potential improvement on performance of multi-banks 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 [LAR90] 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 [AGA88]).

A single process execution was supposed : performance of caches in a time-sharing environment would be certainly worse than the results shown here [AND91, MOG91].

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 simulation:

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 [SMI87]).

In [JOU90], 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 mechanism was simulated for all the configurations: as stated in [JOU90], it improves significantly direct-mapped cache hit ratio (about 25 % data miss were removed on our benchmark set), but also set-associative caches hit ratio(about 10 % data miss removed) and even skewed-associative caches hit ratio (about 5 % data miss removed). 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 3.5).

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.

The Figure 12 illustrates the potential benefit of using skewed-associative caches in terms of normalized execution times (see Definition 2.2) for different cache sizes and assuming equal sizes of the two caches:

- There is marginal performance benefit in using a multi-bank cache organization when the main memory latency is small (e.g. 5 cycles), but when the memory latency becomes higher, using a multi-banks 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.

- Using a four-banks 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-banks 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.

Normalize memory busy time

We have already pointed out that there is also some stress on the main memory.

This stress is illustrated by the normalized memory busy time (see Definition 2.3) in Figure 13 for different cache sizes and organizations. Formula 2 was used with $Mdel = LAT + 7$ i.e. LAT cycles are necessary to obtain the first 8 bytes of the line in the memory, then a 8 bytes word flows out from memory on each cycle.

As previously mentioned, the NRUNRW replacement policy decreases the ratios of miss inducing Copy Back on the memory and then decreases the stress on memory. This may be particularly important when building a single-bus shared memory multi-microprocessor system.

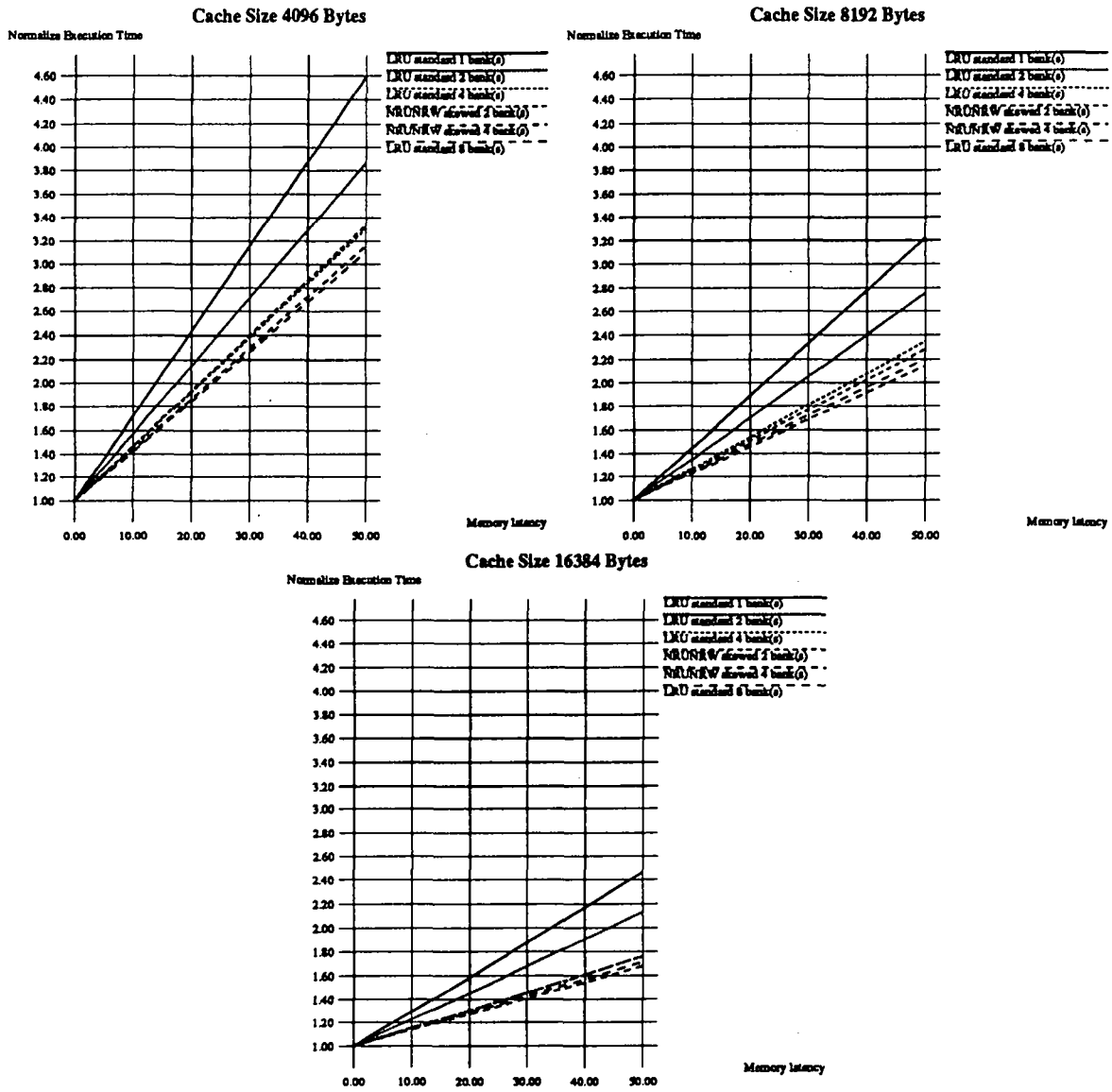


Figure 12: Normalized Execution Times for different cache sizes

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

5 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 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 most of the new superscalar microprocessors (IBM Power, SUN Viking, 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-banks 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).

Unfortunately, LRU replacement policy is quite difficult to implement in hardware for skewed-associative caches. We have proposed the Not Recently Used Not Recently Written replacement policy which is easier to implement in hardware. This replacement policy induces approximately the same miss ratio as the LRU replacement policy, but induces less copy back on the main memory (and then less memory traffic).

Simulations have shown that skewed-associative caches have a better behavior than set-associative caches: typically a two-banks 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.

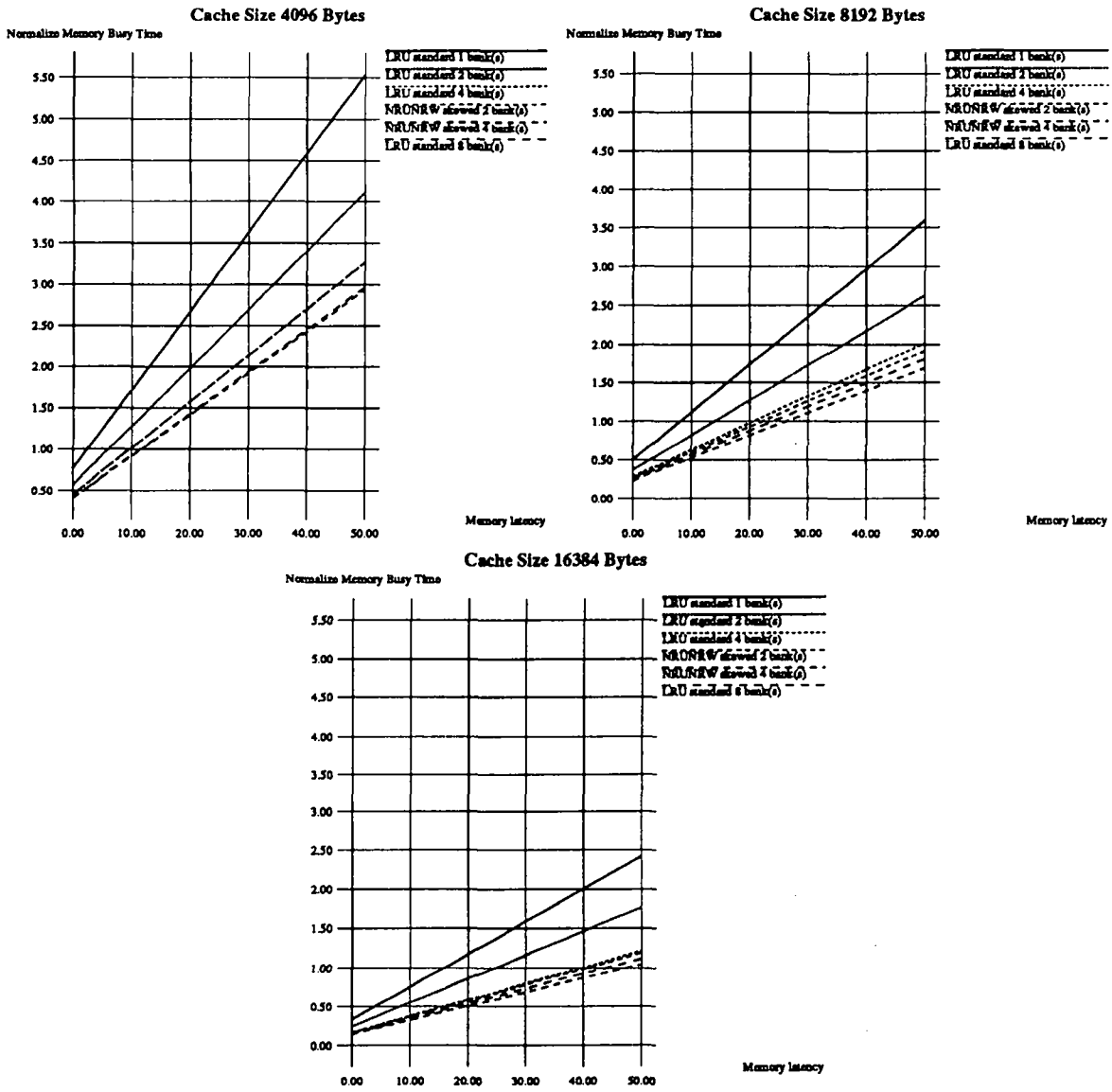


Figure 13: Normalized Memory Busy Times for different cache sizes

References

[AGA88] A. Agarwall, M. Horowitz, J. Hennesy "Cache performance of operating systems and multiprogramming workloads" ACM Transactions on Computer Systems, Nov. 1988

[AGA89] A. Agarwall, M. Horowitz, J. Hennesy "Cache performance of operating systems and multiprogramming workloads" ACM Transactions on Computer Systems, May 1989

[AND91] 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

[BUD71] P.Budnick, D.Kuck "The organization and use of parallel memories" IEEE Transaction On Computers, Dec. 1971

[FRA85] J.M. Frailong, W.Jalby, J.Lenfant "XOR-schemes: a flexible organization in parallel memories" Proceedings of 1985 International Conference on Parallel Processing, Aug. 1985

[HAR86] D.T. Harper, J.R. Jump "Performance evaluation of vector accesses in parallel memories using a skewed storage scheme ", Proceedings of the 13^{th} International Symposium on Computer Architecture, June 1986

[HAR87] D.T. Harper, J.R. Jump "Vector accesses in parallel memories using a skewed storage scheme " IEEE Transactions on Computers, Dec. 1987

[HIL89] M.D.Hill, A.J. Smith "Evaluating Associativity in CPU Caches" IEEE Transactions on Computers, Dec. 1989

[JOU89] N.P. Jouppi, D.W. Wall "Available instruction-level parallelism for superscalar and superpipelined machines " Proceedings of ASPLOS III, April 1989

[JOU90] 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

[LAM91] M. Lam, E. Rothberg and M. Wolf, "The Cache Performance and Optimizations of Blocked Algorithms", Proceedings of ASPLOS IV, April 91

[LAR90] J.R.Larus, "Abstract execution: a technique for Efficiently Tracing Programs" Technical Report, Computer Sciences Departement, University of Wisconsin-Madison, May 1990

[MOG91] J.C. Mogul, A. Borg "The effect of context switches on cache performancé' Proceedings of ASPLOS IV, April 1991

[NOR87] A.Norton, E.Melton "A class of boolean linear transformations for conflict-free power-of-two stride access", Proceedings of the International Conference on Parallel Processing, 1987

[RAU89] B.Rau, M.Schlander, D. Yen " The Cydra 5 stride insensitive memory system", Proceedings of the International Conference on Parallel Processing, 1989

[SMI82] A.J. Smith "Cache memories" ACM Computing Surveys, Sept. 1982

[SMI87] A.J. Smith "Line (block) size choice for CPU cache memories" IEEE Transactions on Computers, Sept. 1987

[SMI89] M.D. Smith, M. Johnson, M.A. Horowitz "Limits on multiple instruction issué' Proceedings of ASPLOS III, April 1989

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 635 A NOTE ON CHERNIKOVA'S ALGORITHM
Hervé LE VERGE
Février 1992, 28 pages.

PI 636 ENSEIGNER LA TYPOGRAPHIE NUMERIQUE
Jacques ANDRE, Roger D. HERSCH
Février 1992, 26 pages.

PI 637 TRADE-OFFS BETWEEN SHARED VIRTUAL MEMORY AND MESSAGE-PASSING ON AN iPSC/2 HYPERCUBE
Thierry PRIOL, Zakaria LAHJOMRI
Février 1992, 26 pages.

PI 638 RUPTURES ET CONTINUITES DANS UN CHANGEMENT DE SYSTEME TECHNIQUE
Alan MARSHALL
Mars 1992, 510 pages.

PI 639 EFFICIENT LINEAR SYSTOLIC ARRAY FOR THE KNAPSACK PROBLEM
Rumen ANDONOV, Patrice QUINTON
Mars 1992, 20 pages.

PI 640 TOWARDS THE RECONSTRUCTION OF POSET
Dieter KRATSCH, Jean-Xavier RAMPON
Mars 1992, 22 pages.

PI 641 MADMACS : A TOOL FOR THE LAYOUT OF REGULAR ARRAYS
Eric GAUTRIN, Laurent PERRAUDEAU
Mars 1992, 12 pages.

PI 642 ARCHE : UN LANGAGE PARALLELE A OBJETS FORTEMENT TYPES
Marc BENVENISTE, Valérie ISSARNY
Mars 1992, 132 pages.

PI 643 CARTESIAN AND STATISTICAL APPROACHES OF THE SATISFIABILITY PROBLEM
Israël-César LERMAN
Mars 1992, 58 pages.

PI 644 PRIME MEMORY SYSTEMS DO NOT REQUIRE EUCLIDEAN DIVISION BY A PRIME NUMBER
André SEZNEC, Yvon JEGOU, Jacques LENFANT
Mars 1992, 10 pages.

PI 645 SKEWED-ASSOCIATIVE CACHES
André SEZNEC, François BODIN
Mars 1992, 20 pages.

PI 646 INTERLEAVED PARALLEL SCHEMES : IMPROVING MEMORY THROUGHTPUT ON SUPERCOMPUTERS
André SEZNEC, Jacques LENFANT
Mars 1992, 14 pages.

PI 647 COMMUNICATING PROCESSES AND FAULT TOLERANCE : A SHARED MEMORY MULTIPROCESSOR EXPERIENCE
Michel BANATRE, Maurice JEGADO, Philippe JOUBERT, Christine MORIN
Mars 1992, 40 pages.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249 - 6399