A New Case for Skewed-Associativity

André Seznec

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A New Case for Skewed-Associativity

André Seznec

N ° 3208

Juillet 1997

THÈME 1

ISSN 0249-6399

A New Case for Skewed-Associativity

André Seznec

Thème 1 — Réseaux et systèmes
Projet CAPS

Rapport de recherche n ° 3208 — Juillet 1997 — 23 pages

Abstract: Skewed-associative caches have been shown to statisticaly exhibit lower miss ratios than set-associative caches at equal associativity degrees. But for L1 caches, the performance of a four-way set-associative cache is known to be quite close to that of a 16-way set-associative cache.

In this paper, we first show that BTBs and L2 caches featuring long line size may benefit a lot from a high associativity degree on set-associative caches and a LRU replacement policy. Skewed-associative caches with an hypothetic LRU replacement policy "provide" this high associativity but with a low number of cache banks.

Unfortunately, LRU replacement policies can not be easily implemented in hardware on skewed-associative caches. We propose some new replacement heuristics for skewed-associative caches. Those heuristics exhibit slightly worse behavior than a full LRU replacement policy, but still allow a 2-way skewed-associative cache to outperform a 4-way LRU set-associative cache and a 4-way skewed-associative cache to outperform a 16-way LRU set-associative cache. Therefore skewed-associativity allows significant miss reduction on BTBs and L2 caches.

Key-words: skewed-associative caches, replacement policies, set-associative caches, BTBs, L2 caches

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : (33) 02 99 84 71 00 – Télécopie : (33) 02 99 84 71 71

Une nouvelle argumentation pour les structures associatives brouillées

Résumé : Il a été montré que les caches associatifs brouillés exhibent de meilleurs taux de succès que les caches associatifs par ensemble.

Dans ce rapport, nous montrons que les structures associatives brouillées sont particulièrement intéressantes pour les caches de prédictions de branchement et les caches secondaires. De plus nous proposons de nouvelles politiques de remplacement pour les structures associatives brouillées

Mots-clé : caches associatifs brouillés

A New Case for Skewed-Associativity

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

Abstract

Skewed-associative caches have been shown to statisticaly exhibit lower miss ratios than set-associative caches at equal associativity degrees. But for L1 caches, the performance of a four-way set-associative cache is known to be quite close to that of a 16-way set-associative cache.

In this paper, we first show that BTBs and L2 caches featuring long line size may benefit a lot from a high associativity degree on set-associative caches and a LRU replacement policy. Skewed-associative caches with an hypothetic LRU replacement policy "provide" this high associativity but with a low number of cache banks.

Unfortunately, LRU replacement policies can not be easily implemented in hardware on skewed-associative caches. We propose some new replacement heuristics for skewed-associative caches. Those heuristics exhibit slightly worse behavior than a full LRU replacement policy, but still allow a 2-way skewed-associative cache to outperform a 4-way LRU set-associative cache and a 4-way skewed-associative cache to outperform a 16-way LRU set-associative cache. Therefore skewed-associativity allows significant miss reduction on BTBs and L2 caches.

Keywords:

skewed-associative caches, replacement policies, set-associative caches, BTBs, L2 caches

1 Introduction

Skewed-associative caches [6, 7, 1] have been shown to exhibit better behavior than set-associative caches: typically a two-way (resp. four-way) skewed-associative cache has the hardware complexity of a two-way (resp. four-way) set-associative cache, yet it exhibits statistically the same hit ratio as a four-way (resp. eight-way) set associative cache of the same size. These results were obtained assuming a pseudo-replacement policy.

Nevertheless, it has been broadly admitted [3] that there is little benefit in increasing cache associativity over four: it was reported in [3] that increasing the associativity from four to eight reduces cache miss ratio by less than 5%. When admitting this, the potential benefit of artificially "doubling" the associativity of caches with skewed-associative caches is quite low.

The first contribution of this paper is to present simulation results establishing that, in modern processors, there are caches that would benefit more from increasing the associativity than previously admitted: branch target buffers and long line L2 caches. Therefore branch target buffers and L2 caches are good candidates for using skewed-associativity.

Till now the replacement policy issue in skewed-associative caches has not been really adressed. The LRU replacement policy is one of the best replacement heuristics for set-associative caches and can be implemented at a reasonable cost on low associativity degree set-associative caches. Our simulations show that when assuming a full LRU replacement policy, a 2-way skewed associative cache (either BTB, or L2 or L1) would have a slightly better behavior than a 4-way set-associative cache while a full LRU 4-way skewed-associative cache outperforms a 16-way LRU set-associative cache. Unfortunately, LRU replacement policy can not be implemented at reasonable hardware cost on skewed-associative caches.

The second contribution of this paper is to propose some implementable replacement heuristics for skewed-associative caches. Even, if those heuristics exhibits slightly worse behavior than a full LRU replacement policy, the 2-way skewed-associative cache has still a behavior slightly better than a 4-way set-associative cache while a 4-way skewed-associative cache has the behavior of a 16-way LRU set-associative cache.

The remainder of the paper is organized as follows. In Section 2, the basic principles of a skewed-associative cache are recalled. In Section 3, we present the evaluation methodology which will be used in the paper. In Section 4, we present simulation results for random replacement policy and for perfect LRU replacement policy. These simulation results show the potential benefit of using high (or skewed) associativity on BTBs and long lines L2 caches. In Section 5, we recall previously proposed replacement heuristics for skewd-associative caches and analyze their drawbacks; we then propose new implementable pseudo-LRU replacement heuristics. These heuristics are then shown to be quite effective: our enhanced not recently used replacement heuristic only costs 2 bits per cache line, but allows to implement 4-way skewed-associative caches which competes favorably with a 16-way LRU set-associative cache. Section 7 summarizes this study.

2 Skewed-associative caches

2.1 Principle

Skewed associative caches were proposed in [6, 7]. A X-way set-associative cache is built with X distinct banks as illustrated in Figure 1 . The memory block at address D may be physically mapped onto physical line $f(D)$ of any of the distinct banks. This vision of a set-associative cache fits with the physical implementation: X banks of static RAMs.

For a skewed associative cache (Figure 2), a different mapping function is used for each cache bank: A memory block at address D may be mapped onto physical line $f_0(D)$ in bank 0, onto physical line $f_1(D)$ in bank 1, etc.

It has been shown in [6, 7] that for general applications skewed-associative caches exhibit an average lower miss ratio than set-associative caches.

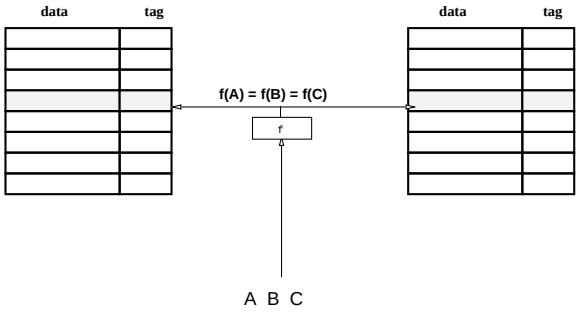


Figure 1: 3 data blocks conflicting for a single set on a two-way set-associative cache. A, B and C compete for only two locations

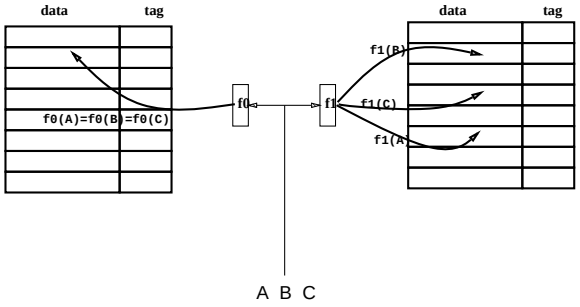


Figure 2: A, B and C compete for the same location in bank 0, but can be present at the same time, as they do not map to the same location in bank 1

2.2 Choosing the skewing functions

In this section, we recall the properties that might exhibit functions chosen for skewing the blocks in the distinct cache banks in order to obtain a good hit ratio (see [6, 1] for further insight). We also present the family of functions that were used in simulations presented in this paper.

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

Inter-bank dispersion In a usual X-way set-associative cache, when (X+1) memory blocks contend for the same set in the cache: one of the blocks must be rejected from the cache (Figure 1), since only X lines are available.

Skewed-associative caches avoid this situation by scattering the data: mapping functions can be chosen so that whenever two memory blocks conflict for a same line in bank i, they have a very low probability to conflict for a location in bank j (Figure 2).

Local dispersion in a single bank Many applications exhibit spatial locality, therefore the mapping functions must also be chosen so that two "almost" neighbor memory blocks do not contend for the same physical line in any cache bank.

The different mapping functions must respect a certain form of local dispersion on their respective bank; the mapping functions f_i must limit the number of conflicts when mapping any region of consecutive memory blocks within bank i.

Simple hardware implementation Mapping functions must have the simplest implementation, so as to introduce very few extra gates and delays.

An example of skewing functions We present here the skewing functions which were used in the simulations that illustrate this paper.

Let us consider a skewed associative cache built with 2 or 4 cache banks, each one consisting of 2^n cache lines of 2^c bytes,
let σ be the perfect-shuffle on n bits [8], ϕ and ϕ' be two permutations on n bits[1], data block at memory address $A_3 2^{c+2n} + A_2 2^{n+c} + A_1 2^c$ may be mapped:

1. on cache line $f_0(A) = \phi(A_1) \oplus \phi'(A_2)$ in cache bank 0
2. *or* on cache line $f_1(A) = \sigma(\phi(A_1)) \oplus \phi'(A_2)$ in cache bank 1
3. *or* on cache line $f_2(A) = \sigma^2(\phi(A_1)) \oplus \phi'(A_2)$ in cache bank 2
4. *or* on cache line $f_3(A) = \sigma^3(\phi(A_1)) \oplus \phi'(A_2)$ in cache bank 3

[1] The perfect-shuffle on a n-bit number is the one-position circular shift on a n-bit number; neither the perfect-shuffle, nor bit permutations require any hardware logic to be performed.

These functions were chosen for the following reasons:

1. They satisfy the previously listed criterions for "good" skewing functions. Each bit in $f_i(A)$ is obtained with a single two-entry XOR. Complete inter-bank dispersion is not achieved; nevertheless, the set of blocks which may be mapped on a precise cache line in one bank is spread over a large number of lines in the other banks.

   Since two blocks with the same highest order bits (A3 and A2) cannot be mapped on the same cache line by function f_i, local dispersion is achieved.

2. They are relatively simple to compute : this limits the simulation time and represents a potential cheap implementation.

The results illustrated in this paper are not specific to this particular choice of skewing functions: the bit permutations used on A2 might be different, the global behavior would be the same. Other skewing functions satisfying the listed criterions were used in [6, 7, 1].

2.3 How a skewed-associative cache handles conflict misses

Two phenomena explain why skewed-associative are less subject to conflict misses than set-associative caches. These phenomena were named **data dispersion** and **self data reorganization** in [1].

Data dispersion After a single read on a sequence of distinct cache blocks, more blocks are in a skewed-associative cache than on a set-associative cache.

Self data reorganization On a set-associative cache, the configuration of data blocks that can be present at the same in the cache is statically determined: the 3 blocks conflicting for a set in Figure 1 can not be alive at the same time in the cache.

On the other hand, on the skewed-associative cache, the configuration of data blocks present at the same time in the cache depends on the precise mapping of each data block in the cache. Let us consider a memory block D present in the cache at time t. Among the other possible locations for a block D, there may be a location occupied by an unuseful block (dead block or block that will not be reused for a long time). Block D may be removed from the cache by a miss. The next time D will be referenced, D can be mapped in an empty location in bank j, thus increasing the number of useful blocks in the cache.

2.4 Replacement policy for skewed-associative caches: a difficult issue

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

LRU replacement policy is generally considered as the most efficient policy. Implementing a LRU replacement policy on a two-way set-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 set-associative cache can be implemented by adding only (X-1) bit tags to each line. With such a scheme, the same information bit "data in bank i is older than data in bank j" is represented twice. One of these two information bits can be saved. Therefore, a LRU replacement policy can be implemented on a N-way set-associative cache using only $\frac{N*(N-1)}{2}$ bits per set. It should be noted that, for large N, the logic needed to manage the LRU policy is quite complex while the size of the tag array becomes quite large.

Unfortunately, finding a simple hardware implementation of a LRU replacement policy on a skewed-associative cache is impossible: as the set of lines on which a block has to be replaced vary with the new block to be introduced, the information needed in order to determine the last referenced line in the set is the complete date of the reference.

Some pseudo-LRU replacement policies were used in [6, 7]. In Section 5, we will present more effective pseudo-LRU policies.

3 Evaluation methodology

3.1 Experimental set-up

In order to evaluate replacement policies on caches, trace driven simulations were conducted. The IBS Ultrix benchmarks were used [9]. These benchmarks were traced using a hardware monitor connected to a MIPS-based DECstation running Ultrix 3.1. The resulting traces include activity from all user-level processes as well as the operating-system kernel.

Three different hardware structures were simulated a split data/instruction L1 cache, a branch target buffer (BTB) and a unified L2 cache using long cache line. These three structures are likely to be integrated in modern preocessors on the same die or at least on the same module. As our study focusses on associativity and replacement policy, we fixed most of the other parameters. These are given below:

- **L1 cache**: 16Kbyte, 16-byte cache line
- **BTB**: 2048 entries. In this study, we are only concerned with the miss ratio on BTB. For an indirect jump, a hit on the BTB does not preclude a good prediction, but for all branches a miss leads to either a misfetch or a misprediction.
- **L2 cache**: 256Kbyte, 128-byte cache line. Requests were filtered by a split direct-mapped 8Kbyte L1 cache with a 16-byte cache line. Inclusion property was enforced on the L1 caches.

	groff	gs	mpeg_play	nroff	real_gcc	sdet	verilog	video_play
L1	3.67	4.95	3.54	1.76	4.37	7.55	7.33	9.91
L2	2.16	2.39	3.08	0.75	3.00	2.83	5.60	4.50
BTB	6.62	7.54	5.87	2.52	7.87	10.45	7.12	12.38

Table 1: Reference miss ratios

3.2 Performance metric

The purpose of this study is not to measure absolute miss ratios, but, first to show that there is a need for high associativity and at second to evaluate replacement policies for skewed-associative caches.

Then, throughout the remainder of the paper, the performance metric for cache C will be:

$P = \frac{miss\ ratio\ on\ C}{reference\ miss\ ratio}$.

where **reference miss ratio** is the miss ratio on a 2-way set-associative cache with LRU replacement for the benchmark and the considered structure (L1 cache, BTB or L2 cache).

For information, reference miss ratios are given in table 1. Reference miss ratios are given in misses per 100 accesses for the BTB, misses per 100 instructions for L1 cache and misses per 100 L2 cache accesses for the L2 cache.

4 Need for high associstivity

In this section, we first point out the high potential benefits than can come from using high associativity particularly for L2 caches and BTBs. We then show the potential of skewed-associative caches in "artificialy" providing such associativity.

A need for high associativity Figure 3 and 4 illustrate the relative performance of skewed-associative caches and set-associative caches on our benchmarks.

It can be noted that the global behavior for the L1 caches, L2 caches and BTBs are different:

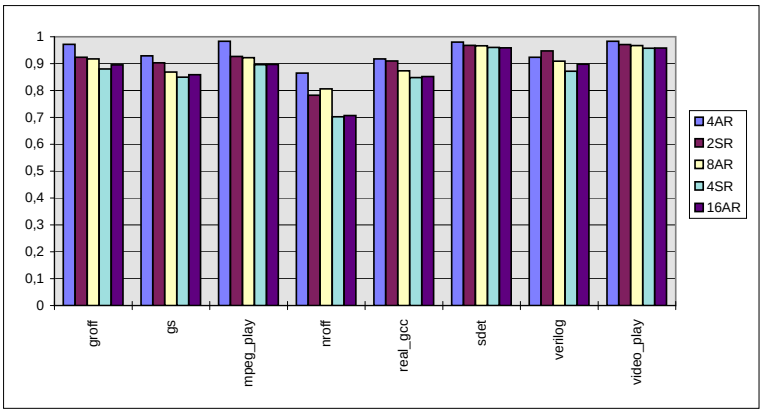
- For 5 of our benchmarks, the potential benefit of increasing associativity degree of L1 caches from 2 to 16 is quite limited: a reduction of less than 20 % of misses is obtained. Only *nroff* gets a reduction in L1 cache misses higher than 40 %. Moreover, as stated in [3], most of the gain is already captured when moving from degree 2 to degree 4, particularly with random replacement policy (Figure 3.a).
- The potential benefit of increasing associativity on L2 caches is clearly high. 6 out of 8 benchmarks obtain a reduction of L2 cache misses in the range of 40 % or plus when using a 16-way LRU set-associative cache instead of a 2-way LRU set-associative cache (Figure 4.c).

- On BTBs, increasing the associativity degree leads to very clear benefits for all our benchmarks.

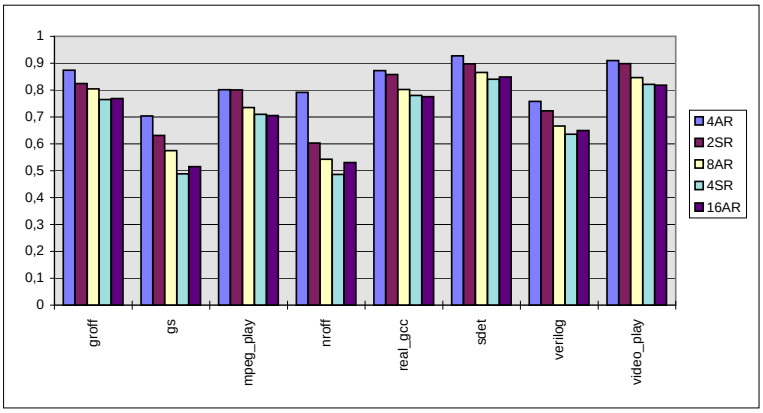
The difference between the L1 cache and the BTB can be explained by spatial locality. Spatial locality exists in instructions and in data, but not really on branches. The use of indexing functions that respect spatial locality limits set conflicts among data (or instruction) blocks: consecutive blocks do not conflict.

A contrario, each branch is a "cache" block in the case of the BTB and branches exhibits very poor spatial locality.

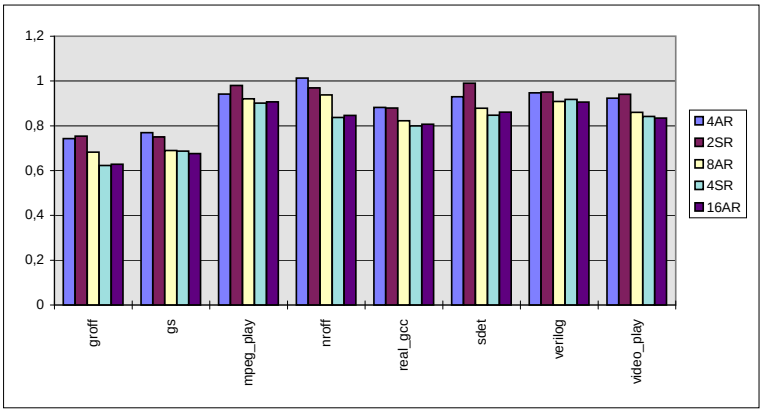
On the other hand, the large benefit from increasing associativity for L2 caches is explained by the long cache line we assume for the L2 cache: when the line size increases, the average number of cache blocks competing for a single position increases.



(a) L1 caches

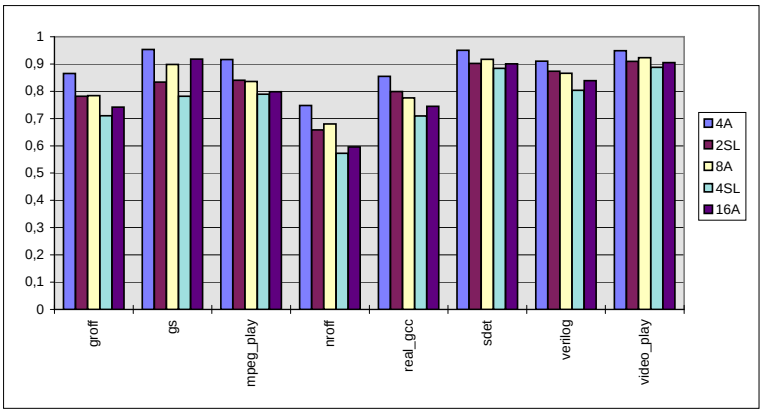


(b) BTBs

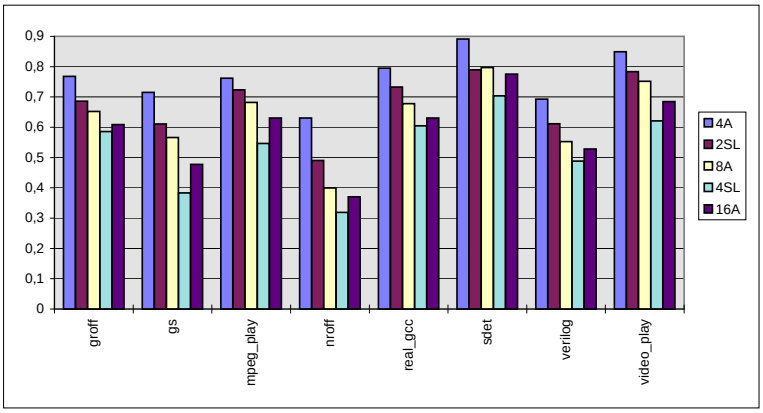


(c) L2 caches

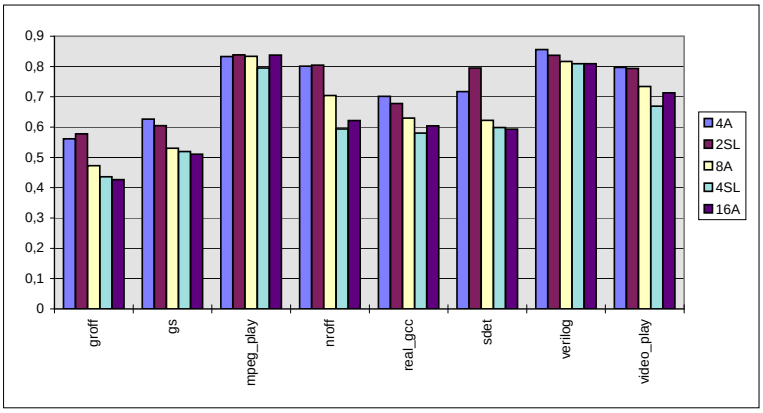
Figure 3: Pseudo-random replacement



(a) L1 caches



(b) BTBs



(c) L2 caches

Figure 4: LRU replacement

Pseudo-random replacement policy In Figure 3, we illustrate the relative performance of skewed-associative caches and set-associative caches when a pseudo-random replacement policy is implemented.

In average, on the L1 cache and the BTB, the 2-way skewed-associative cache slightly outperforms the 4-way set-associative cache, but exhibits consistently higher miss ratio than the 8-way set-associative cache. The 4-way set-associative cache exhibits approximately the behavior of the 16-way set-associative cache for L1 caches and clearly outperforms it for BTBs.

LRU replacement policy In Figure 4, we illustrate the relative performance of skewed-associative caches and set-associative caches when a complete LRU replacement policy is implemented. Comparing Figure 4 and Figure 3 allows to conclude that LRU replacement policy largely outperforms the pseudo-random replacement policy: it is generally better to use a N-way LRU set-associative cache than a 2*N-way set-associative cache using a pseudo-random replacement heuristic.

On our benchmark set, the skewed-associative cache seems to take more benefit from the use of the LRU replacement policy than the set-associative cache. For instance, for BTBs and L1 cache, the 2-way skewed-associative cache clearly outperform 4-way set-associative caches, while the 4-way skewed-associative cache consistently outperforms the 16-way set-associative cache.

LRU is a quite effective replacement heuristic. From our simulation results, it appears that it is even more efficient for skewed-associative caches than for set-associative caches. This is explained by the structure of skewed-associative cache: let us suppose that a useful block B is replaced because it was the older one in the dynamic set associated with the incoming block A [2]. On the next access to block B, block B is brought back in the cache in place of the older block in its own dynamic set (which is different from the dynamic set associated with block A). Then, in several steps the LRU replacement tends to replace "old" data block anywhere in the cache on a skewed-associative cache.

L2 caches One will notice on Figures 3.c and 4.c, that L2 caches, the 2-way skewed-associative cache exhibits slightly higher miss ratios than the 4-way set-associative cache while the 4-skewed associative cache performs as well as a 16-way set-associative cache.

This relative poor behavior (compared to L1 caches and BTBs) is due to the relatively small working set of the traced applications. As the IBS traces were collected on a workstation only featuring 16 Mbytes of main memory,, each application is touching less than 30000 128-byte lines. Then in average each L2 cache line is the possible location for less than 30 memory blocks, and this represents a very limited potential for inter-bank dispersion.

Summary We have shown that there is a major interest in adding associativity, particularly on L2 caches and BTBs. The simulation results presented also show that the replacement policy has a very important impact.

[2] The dynamic set associated with block A is the set of possible locations for block A in the cache

The skewed-associative cache has the potential to be very efficient. This is particularly true with the LRU replacement policy. Unfortunately, implementing a LRU replacement policy on skewed-associative caches cannot be seriously considered: it would require storing a complete date tag in each cache line.

In the next section, we propose and analyze some implementable replacement policies trying to approach the behavior of the LRU replacement policy.

5 Replacement policies for skewed associative caches

5.1 Previous work

We first recall the previous pseudo-LRU replacement policies that were used in previous studies and analyze their drawbacks.

Single-bit replacement policy In [6], the following pseudo-LRU replacement policy based on a single-bit was proposed for 2-way skewed-associative caches.

> 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

The rationale of this policy is that, the last indexing of the cache line in bank 0 determines the "usefullness" of the cache block B it contents.

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.

The major drawback of this replacement policy is to be asymmetric: no information on the cache block stored in bank 1 is used at all, the "usefullnesses" of the two targets are not compared.

It also seems quite difficult to extend this replacement policy to degrees of associativity higher than two.

Not Recently Used policy In [7], the following replacement policy was proposed

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

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

1. Randomly among the blocks for which the RU tag is clear
2. Randomly among the blocks for which the RU tag is set, but which have not been modified since they have been loaded in the cache
3. Randomly among the blocks 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 to memory (or the L2 cache) and therefore limiting memory traffic.

This replacement policy was named Not Recently Used Not Recently Written (NRUNRW).

This replacement is quite interesting because it requires only a single bit per cache line; it also discriminates the set of possible replacement locations between "old" cache lines and recently used cache lines.

This replacement policy was experimented only for L1 caches [7]. Finding a correct period for the reset of RU bits is more troublesome for BTBs or L2 caches. Moreover, experiments on the IBS benchmarks showed poor performance actually worse than the single-bit replacement policy presented above for 2-way skewed-associative cache.

5.2 Alternate replacement policies

Our goal was to find a better replacement heuristic than those previously studied, then many possibilities were explored. Only those leading to significant performance benefits at a reasonable hardware cost are presented here.

"Usefulness" policy First, we tried to adapt the previous single-bit replacement policy to work symmetrically for 2-way skewed-associative caches.

> A tag bit U (for useful) is associated with each line the two banks. When a line is indexed, on a hit the tag bit is updated with the number of the hitting bank. On a miss, the tag bits of the two selected lines are read: when they agree on value b, the missing line is written in bank $1-b$, when they disagree, the number of the target bank is randomly selected

We call this replacement policy, the *"Usefulness" policy.*

This replacement policy is quite simple to implement and costs 1 bit per cache line. It takes a better part of the symmetry of the 2-way skewed-associative cache than the original single-bit replacement heuristic: "usefullnesses" of the two possible replacement targets are used.

Not Recently Used The previously described NRUNRW policy suffers from several drawbacks.

At a first point, the relative numbers of "old" cache blocks and "young" cache blocks present in the cache may vary a lot depending on applications and also on steps in applications: when the same small set of data is accessed iteratively, all informations on other data are becoming lost.

In order to better discriminate between "old" cache lines and "young" cache lines, we propose to modify the Not Recently Used (NRU) policy as follows:

- A bit tag Y (for **young**) is asserted when the cache line is accessed: the cache line becomes **young**.
- a counter registers the number of *new* RU assertions, i.e, counts the number of **young** cache lines. When half the cache lines are **young**, all RU bits are reset i.e., all cache lines become **old**.

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

1. Randomly among the **old** blocks, i.e, the RU tag is clear
2. Randomly among the **young** blocks, i.e the RU tag is set

This NRU replacement policy costs 1 tag bit per cache line and a counter.

We experimentally measured that this NRU policy generally works better than the previously defined NRUNRW policy.

Nevertheless this policy only approaches the LRU policy and suffers from two drawbacks:

- When the Y tag is reset, all cache blocks become **old** for a while: random policy is then applied.
- Discrimination between **old** and **young** cache blocks is quite primitive: in many cases, all replacement targets are **old** (or **young**), then random choice is also applied.

Two other policies were tested in order to limit this random choice.

Not Recently Used + Useful The two previously defined replacement policies may be mixed. Two tag bits Y (for young) and U (for useful) are used. They are asserted just as defined previously, but the choice of the replacement target is now done as with following priority:

1. If one of the block is **old** and the second one is **young** then select the **old** block.
2. If the two U bits agree on value b, then select block in bank $1 - b$.
3. Randomly select one of the two banks.

Enhanced Not Recently Used As already noticed, when using the NRU replacement policy, the reset of Y tag reset creates a lot of randomness in the choice of replacement targets.

In order to such randomness, a second young tag bit Y' is added to each cache line. Y' is asserted whenever the cache block is accessed, and is reset when the counter modulo $\frac{Number\ of\ Cache\ Lines}{2}$ is equal to $\frac{Number\ Cache\ lines}{4}$. The use of this second tag bit allows to classify cache blocks in **very young blocks** (the two tag bits are asserted), **young blocks** (a single asserted tag bit) and **old blocks**.

Replacement target is chosen with the following priorities:

- Randomly select among the **old** blocks
- Randomly select among the **young** blocks
- Randomly select among the **very young** blocks

The cost of this replacement policy consists of 2 tag bits per cache line and a counter modulo $\frac{Number\ of\ Cache\ Lines}{2}$ and compares favorably with the cost of the LRU replacement policy on a 8-way set-associative cache.

5.3 A specific solution for 4-way skewed-associative caches

Till now, we have considered that the banks on the skewed-associative cache are all indexed using distinct skewing functions. Nevertheless, the present paper has shown that being able to implement an efficient replacement policy is also an important issue.

When using a 4-way associative cache, there exists a trade-off: banks may be grouped per pairs and the same skewing function may be used to index the two banks in a pair. We refer to this design as the *hybrid skewed-associative cache*. In experiments illustrated in the next section, the previously defined function f_0 (resp. f_1) is used for indexing banks 0 and 1 (resp. banks 2 and 3).

Then inside each bank pair, a LRU replacement policy may be implemented, while the choice between the two pairs might be done using other heuristics, as for instance the previously defined Not Recently Used heuristic.

6 Experimental results

In this section, we present simulation results for the different replacement heuristics presented in the previous section. The legends of the different figures in this section are given in the following format $< x >< type >< replacement\ policy >$ where x is the associativity degree, $< type >$ is S (for skewed) or A (for set-associative) while $< replacement\ policy >$ are represented in Table 2.

Replacement policy	Legend
LRU	L
Random	R
Single-bit	B
Useful	U
Not Recently Used	Y
Enhanced Not Recently Used	Y2
Not Recently Used + Useful	U+Y
Hybrid	Hy

Table 2: Graphs Legend

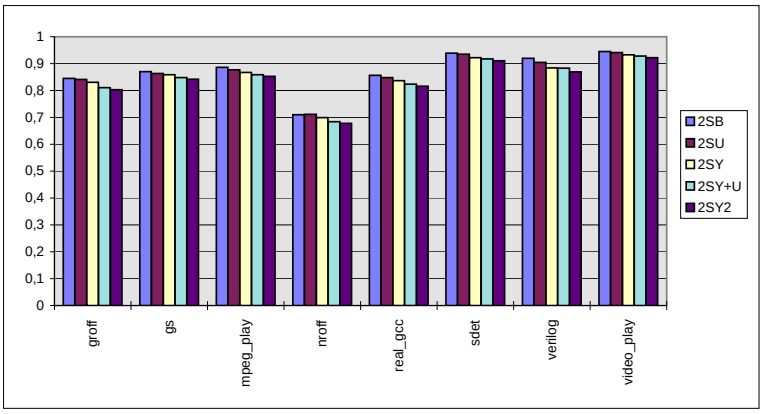
6.1 2-way skewed-associative caches

Simulation results for 5 different replacement heuristics for 2-way skewed-associative caches are reported in Figure 5.

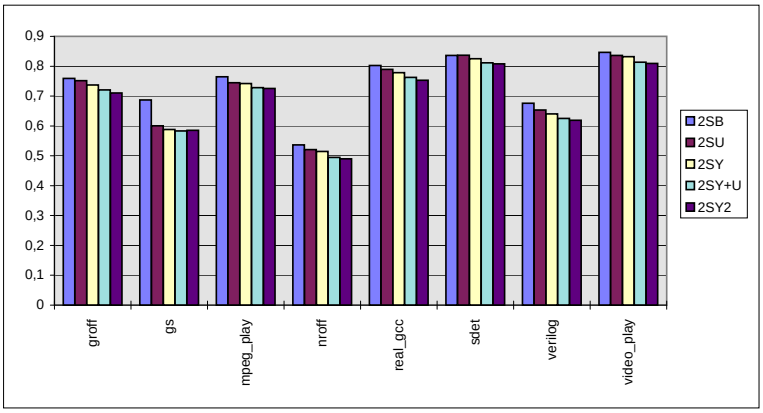
The ranking of the different heuristics is consistent through our 8 benchmarks for L1 caches, L2 caches and BTBs. The Enhanced Not Recently Used policy performs consistently better than the other implementable heuristics.

On Figure 6, we compare the behavior of the 2-way skewed-associative cache with our best implementable heuristics with an hypothetic LRU replacement policies, and with 4-way and 8-way LRU set-associative caches. There remains a small difference between Enhanced Not Recently Used and full LRU replacement policy, but this difference is quite low. On the same figure, the single-bit replacement policy is also plotted. This emphasizes the benefit from the Enhanced Not Recently Used policy: when using the single-bit policy, 2-way skewed is approximately equivalent to 4-way LRU set-associative caches while when using Enhanced Not Recently Used policy, the 2-way skewed-associative cache consistently outperforms the 4-way LRU set-associative cache for L1 caches and BTBs. A contrario, the 4-way LRU set-associative cache slightly outperforms the 2-way skewed-associative cache for L2 caches.

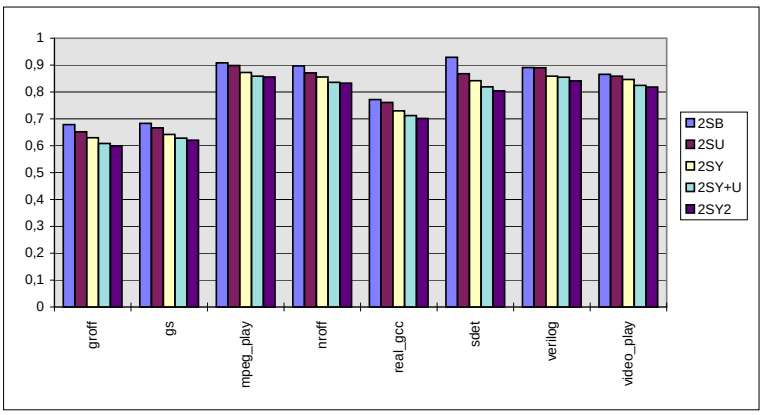
The Enhanced Not Recently Used policy requires only 2 tag bits per cache line, then its hardware cost is quite comparable with the hardware cost of the LRU 4-way set-associative cache (6 bits per set).



(a) L1 caches

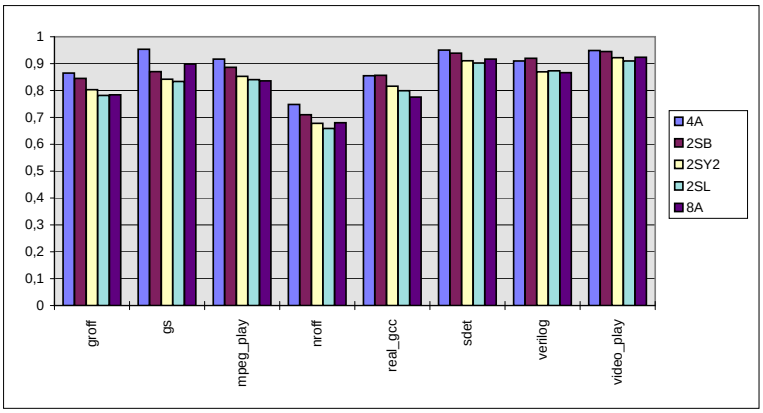


(b) BTBs

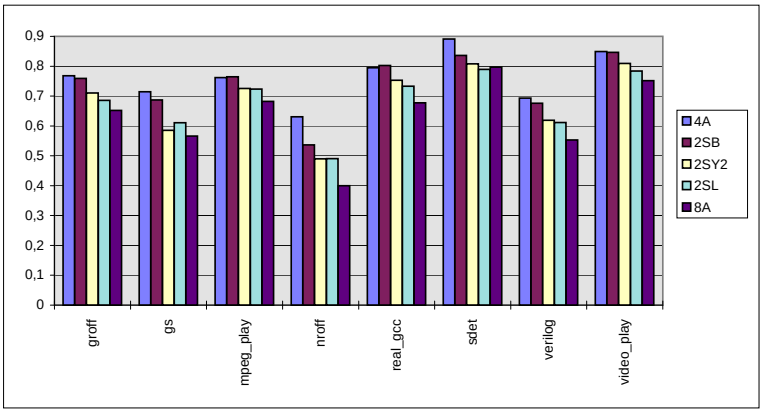


(c) L2 caches

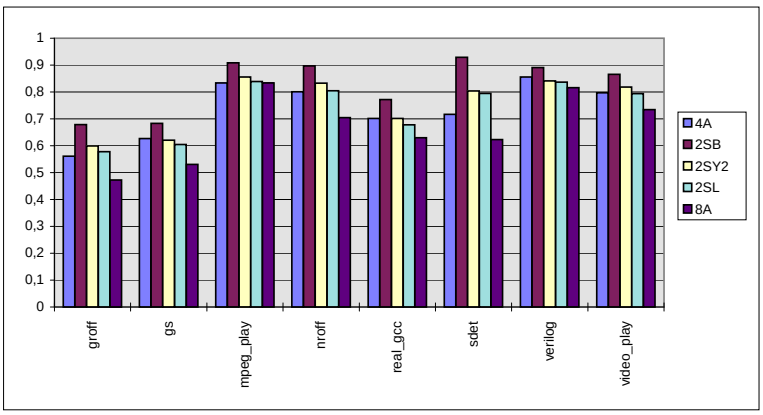
Figure 5: Various replacement heuristics on 2-way skewed-associative caches



(a) L1 caches



(b) BTBs



(c) L2 caches

Figure 6: 2-way skewed versus set-associative caches

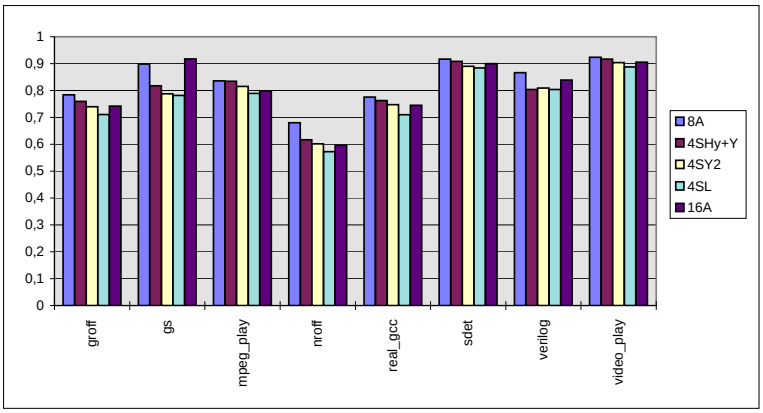
6.2 4-way skewed-associative caches

Figure 7 presents simulation results for 4-way skewed-associative caches using the "hybrid" structure and the usual structure.

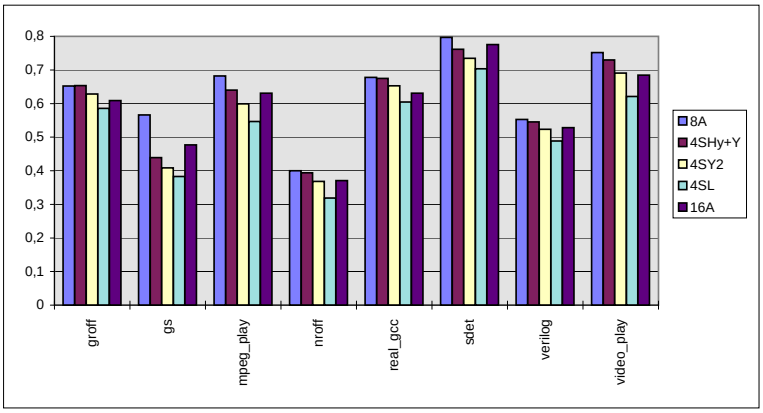
The "hybrid" structure associated with a Not Recently Used replacement policy and LRU replacement inside the bank pairs allows to reach approximately the same performance as a LRU 8-way set-associative cache.

The usual 4-way skewed-associative cache with Enhanced Not Recently Used replacement policy reaches the same level of performance as a 16-way LRU set-associative cache (slightly better for BTBs and L1 caches).

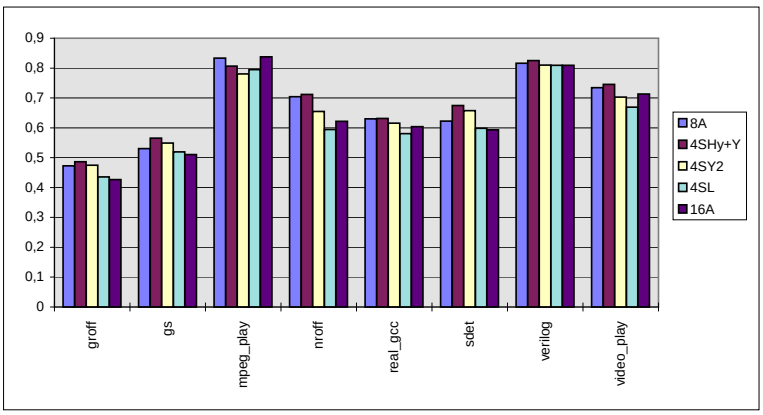
Chosing between the "hybrid" structure and the "full" skewed-associative structure must be done by hardware designers. But we feel that the complexity difference between such an hybrid 4-way skewed-associative cache with Not Recently Used replacement policy and the Enhanced Not Recently Used "full" 4-way skewed-associative cache is not very high.



(a) L1 caches



(b) BTBs



(c) L2 caches

Figure 7: 4-way skewed versus set-associative caches

7 Summary

It was known that the use of skewed-associative caches may allow to significantly reduced conflict misses in numerical applications on dense structures [1, 2]. In particular, on those applications, poor data layout and/or stride distribution may lead to catastrophic and unpredictable behavior when using set-associative cache [4, 5, 1].

Nevertheless, for most non-numeric workloads, four-way set-associative capture the most significant part of conflict misses [3]. Then, using skewed-associative L1 caches instead of set-associative L1 caches would lead to average moderate performancement improvement for such workloads.

We wish to emphasize the two contributions of this paper. First we have pointed out that there exist cache structures in modern processors where high associativity would result in significant cache miss reduction and where typically, 2-way or 4-way set-associativity is not sufficient: BTBs and L2 caches featuring long line size. Second skewed-associativity "provides" such high associativity, with implementable replacement policies.

Needs for higher associativity On L1 caches, the benefit from increasing the associativity over a 2-way set-associative cache exists, but is some where limited and most of the benefit is obtained when moving to a 4-way set-associative cache. On most applications, moving from 4-way set-associative cache to 16-way set-associative caches results in a miss decreasing of less than 10 %, this is coherent with previously published results [3].

The situation is quite different for BTBs and for long lines L2 caches. Increasing associativity results in substantial reduction of cache misses: in average of our benchmarks, using a full LRU 4-way skewed-associative results in less than 55 % (resp. 65 %) of the BTB (resp. L2) misses of a 2-way LRU set-associative cache. The TLB is also likely to be another cache in the processor where skewed-associavity would be useful[3].

Effectivity of skewed-associative caches We have proposed the Enhanced Not Recently Used replacement. This heuristic can be implemented using only 2 tag bits per cache line, but is very effective on approaching LRU replacement policy. A 4-way skewed-associative cache with Enhanced Not Recently Used replacement exhibits the same behavior as a 16-way LRU set-associative cache but at a significantly lower hardware: 4 cache banks against 16 cache banks and a basically less complex replacement policy hardware.

References

[1] F. Bodin, A. Seznec, "Skewed-associativity enhances performance predictability", Proceedings of the 22th International Symposium on Computer Architecture (IEEE-ACM), Santa-Margharita, June 1995.

[3] Unfortunately, we lack access to large workload traces needed to check this assumption

[2] A. Gonzalez, M. Valero, N. Topham, J. Parcerisa " Eliminating Cache Conflict Misses Through XOR-Based Placement Placement Functions", Proceeding of the 11th International Conference on Supercomputing, July 1997

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

[4] M. Lam,, E. Rothberg, M. Wolf, "The Cache Performance and Optimizations of Blocked Algorithms", Proceedings of the Fourth ACM ASPLOS conference, April 91, pp 63-75.

[5] M. Schlansker, R. Shaw, A. Sivaramakrishnan "Randomization and Associativity in the Design of Placement-Insensitive Caches" HP Laboratories Technical Report 93-41, June 1993

[6] A. Seznec, "A case for two-way skewed associative caches", Proceedings of the 20th International Symposium on Computer Architecture, pp 169-178, May 1993

[7] A. Seznec, F. Bodin, "Skewed-associative caches", Proceedings of PARLE' 93, Munich, pp 305-316 June 1993

[8] H.S. Stone, "Parallel processing with the perfect-shuffle", IEEE Transactions on Computers, pp 153-161, Feb. 1971

[9] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, J. Emer,"Coping with Code Bloat", Proceedings of the 22nd Annual International Symposium on Computer Architecture, June 1995

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399