Skewed Branch Predictors

Pierre Michaud, André Seznec, Richard Uhlig

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Skewed branch predictors

Pierre Michaud, André Seznec, Richard Uhlig

N ° 2978

Septembre 1996

THÈME 1

ISSN 0249-6399

Skewed branch predictors

Pierre Michaud, André Seznec, Richard Uhlig

Thème 1 — Réseaux et systèmes
Projet CAPS

Rapport de recherche n° 2978 — Septembre 1996 — 18 pages

Abstract:

As modern microprocessors employ deeper pipelines and issue multiple instructions per cycle, they are becoming increasingly dependent on good branch prediction. During the past five years, researchers have shown that branch-prediction accuracy can be improved by basing predictions on the outcome of previous branches. Many such methods have been proposed, but they all share a common characteristic: they require hardware resources to implement the tables and state machines that record the *branch-history* information. Because hardware resources are invariably limited, it is not possible to hold all relevant branch history for all active branches at the same time, especially for larger workloads consisting of multiple processes and operating-system code. The problem that results, commonly referred to as *aliasing* in the branch-predictor tables, is in many ways similar to the misses that occur in finite-sized hardware caches.

The first contribution of this paper is to propose a classification for three different types of branch aliasing (compulsory, capacity and conflict). We argue that although previous research has resulted in reductions in compulsory and capacity aliasing, little has been done to reduce conflict aliasing. Drawing on established work in caches, our second contribution is to propose the *skewed branch predictor*, a multi-bank, tag-less structure, designed specifically to reduce the impact of conflict aliasing. Through both analytical and simulation models, we show that the skewed branch predictor removes a substantial portion of conflict aliasing by introducing redundancy to the branch-predictor tables. Although this redundancy increases capacity aliasing compared to a standard one-bank structure of comparable size, our simulations show that the reduction in conflict aliasing overcomes this effect to yield a net gain in prediction accuracy.

Key-words: Branch Prediction, Aliasing Classification, Skewed Branch Predictor

* pmichaud@irisa.fr
† seznec@irisa.fr
‡ uhlig@eecs.umich.edu

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

(Résumé : tsvp)

Prédicteurs de branchement à redondances

Résumé :

Avec des pipelines de traitement des instructions de plus en plus larges et profonds, les performances des microprocesseurs récents sont conditionnées par un mécanisme de prédiction des branchements performant. Récemment, certains travaux de recherche ont montré qu'on pouvait améliorer la qualité de la prédiction en se basant sur les directions prises dynamiquement par les derniers branchements. Plusieurs variations de ce principe ont été proposées, mais elles ont toutes en commun l'utilisation d'une table de prédicteurs mise à jour dynamiquement. Comme la taille de cette table est limitée par le budget matériel et le temps de cycle, il n'est pas possible de conserver dans cette table toute l'information nécessaire. Le problème qui en résulte, dans les tables de prédicteurs de branchement sans identificateurs, est appelé *aliasing*, et s'apparente aux échecs de lecture dans les caches. Par analogie avec les caches, nous proposons de classer l'aliasing en trois catégories: aliasing de démarrage, de conflit, et de capacité. Nous introduisons dans ce document le *prédicteur de branchement à redondances* ou *skewed branch predictor*, une structure à plusieurs bancs, sans identificateurs. A travers un modèle analytique et des résultats de simulations, nous montrons que le prédicteur de branchement à redondances permet de réduire l'impact de l'aliasing de conflit sur la qualité de la prédiction. Bien que les redondances introduites accroissent l'aliasing de capacité par rapport à une structure mono-banc de même taille, nos résultats de simulation montrent que la réduction de l'aliasing de conflit compense l'augmentation de l'aliasing de capacité, diminuant ainsi le taux global de mauvaises prédiction.

Mots-clé : Prédiction des branchements, Classification de l'aliasing, Predicteur de branchement à redondances

1 Introduction and Related Work

In processors that fetch multiple instructions in parallel and then dispatch them to deep pipelines, dozens of instructions can be speculatively fetched before a branch is resolved. Under these conditions, a mispredicted branch can result in substantial amounts of wasted work and become a bottleneck to exploiting instruction-level parallelism. Accurate branch prediction has come to play an important role in removing this bottleneck. Although branch prediction consists of predicting both the branch direction and the branch target, this paper will deal with predicting only the branch direction for conditional branches.

Many proposals for dynamic branch prediction exist, with each offering certain distinctive features, but most share a common characteristic: they rely on a collection of 1- or 2-bit counters held in a *predictor table*. Each entry records the recent outcomes of a given branch *substream* [13], and is used to predict the direction of future occurrences of that substream. A branch substream might be defined by certain bits of a branch address, by a bit pattern representing previous branch directions (the *branch history*), or by some combination of branch address and branch history [9, 12, 13, 6, 5, 4].

Ideally, we would like to have a predictor table with infinite capacity so that every unique branch substream defined by an (address, history) pair will have a dedicated predictor. Real-world constraints, of course, do not permit this; limited chip die area and access-time constraints limit predictor-table size, and most tables proposed in the literature are further constrained in that they are direct-mapped and tag-less.

Fixed-sized predictor tables lead to a phenomenon called *aliasing* [13], in which multiple (address, history) pairs share the same entry in the predictor table, causing the predictions for two or more branch substreams to be intermingled. Aliasing has been classified as either *destructive* (i.e., a misprediction occurs due to sharing of the predictor-table entry), *harmless* (i.e., it has no effect on the prediction) or *constructive* (i.e., aliasing occasionally provides a good prediction, which would have been wrong otherwise) [13]. Young et al. have shown that constructive aliasing is rare and much smaller in magnitude than destructive aliasing [13]. Moreover, recent studies have shown that large or multi-process workloads with a strong OS component exhibit very high degrees of aliasing [6, 2], and require much larger predictor tables than previously thought necessary to achieve a level of accuracy close to an ideal, unaliased predictor table [6]. We therefore expect that new techniques for removing aliasing effects could provide important gains towards increased branch-prediction accuracy.

Branch aliasing in fixed-size, direct-mapped predictor tables is in many ways analogous to instruction-cache or data-cache misses. This suggests an alternative classification for branch aliasing based on the *three-Cs model* of cache performance first proposed by Hill [3]. As with cache misses, occurances of aliasing can be classified as *compulsory*, *capacity* or *conflict* aliasing. Similarly, as with caches, larger predictor tables reduce capacity aliasing, while associativity in a predictor table could be used to remove conflict aliasing.

Unfortunately, a simple-minded adaptation of cache associativity would require the addition of costly tags, substantially increasing the cost of a predictor table. In this paper we examine an alternative approach, called skewed branch prediction, which borrows ideas from skewed-associative caches [7, 1]. A *skewed branch predictor* is constructed from an odd

number (typically 3 or 5) of *predictor banks*, each of which functions like a standard tag-less predictor table. When performing a prediction, each bank is accessed in parallel but with a different indexing function, and majority vote between the resulting lookups is used to predict the direction of the branch.

In the next section we explain in greater detail our aliasing classification and in section 3 we introduce the skewed branch predictor, a hardware structure designed specifically to reduce branch-conflict aliasing. In section 4, we show how and why the skewed branch predictor removes conflict aliasing effects. Our analysis includes both analytical and simulation models of performance, and considers a range of possible skewed predictor configurations driven by traces from the instruction-benchmark suite (IBS) [11], which includes complete user and operating-system activity.

2 An Aliasing Classification

Throughout this paper, we will focus on global-history adaptative schemes [12] for the sake of conciseness, and because global schemes suffer more from aliasing than local-history adaptative schemes [6, 12]. Global schemes use both the address and a pattern of global history bits, as described earlier. Given a selection of address bits and a fixed history length, the distinguishing feature of different global schemes is the hashing function that is used to map the set of all (address,history) pairs onto the predictor table.

An aliasing metric: To measure aliasing for a particular global scheme, we simulate a table whose size and indexing function is the same as the predictor considered. However, instead of storing branch predictors in the table entries, we store the identity of the last (address, history) pair that accessed a given entry. Aliasing occurs when an entry is read that contains an (address, history) tag different from the (address, history) pair used to index the entry. We defined the *aliasing rate* as the ratio between the number of aliasing occurrences and the number of dynamic conditional branches. When measured in this way, we can begin to see the relationship between branch aliasing and cache misses. Our simulated tagged table is like a cache with a line size of one datum, and an aliasing instance corresponds to a cache miss.

A widely-accepted classification of cache misses is the three-Cs model, first introduced by Hill [3] and later refined by Sugumar [10]. The three-Cs model divides cache misses into three groups, depending on their causes.

- **Compulsory misses** occur when an address is referenced for the first time. These unavoidable misses are required to fill an empty or "cold" cache.
- **Capacity misses** occur when the cache is not large enough to retain all the addresses that will be re-referenced in the future. Capacity misses can be reduced by increasing the total size of the cache.

- **Conflict misses** occur when two memory locations contend for the same cache line in a given window of time. Conflict misses can be reduced by increasing the associativity of a cache, or improving the replacement algorithm.

Aliasing in branch-predictor tables can be classified in a similar fashion:

- **Compulsory aliasing** occurs when a branch substream is encountered for the first time.
- **Capacity aliasing**, like capacity cache misses, are due to program's working set being too large to fit in a predictor table. This form of aliasing can be reduced by increasing the size of the predictor table.
- **Conflict aliasing** occurs when two concurrently-active branch substreams map to the same predictor-table entry. Methods for reducing this component of aliasing, to our knowledge, have not yet appeared in the published literature.

3 The Skewed Branch Predictor

One way to reduce conflict aliasing would be to introduce some associativity to the branch-predictor table. Unfortunately, a straightforward extension of cache-like associativity in a branch-predictor table, encounters two difficulties.

First, some tag bits would have to be stored in a set-associative branch-predictor table, substantially increasing its cost (in most prediction schemes, a predictor table requires just 2 bits of state per entry. Tags composed of (address, history) pairs could require as many as 12 to 32 additional bits per entry).

A second problem concerns what to do on a miss in the predictor table. In tag-less direct-mapped predictor tables, a single entry is read and used for the prediction, regardless of whether or not it corresponds to the actual branch substream that indexed it. In a set-associative table with tags, it is not clear what action should be taken in the case of a miss.

A skewed branch predictor solves both of these problems by avoiding the use of tags altogether. We now describe the operation of this structure in greater detail.

3.1 Execution Model

A skewed branch predictor is illustrated in figure 1. The basic principle of the skewed branch predictor is to use several branch-predictor banks (3 banks in the example illustrated in figure 1), but to index them by different and independent hashing functions computed from the *same* vector V of information (e.g., branch address + global history). A prediction is read from each of the banks and a majority vote is used to select a final branch direction.

The rationale for using different hashing functions for each bank is that two vectors, V and W, that are aliased with each other in one bank are unlikely to be aliased in the other

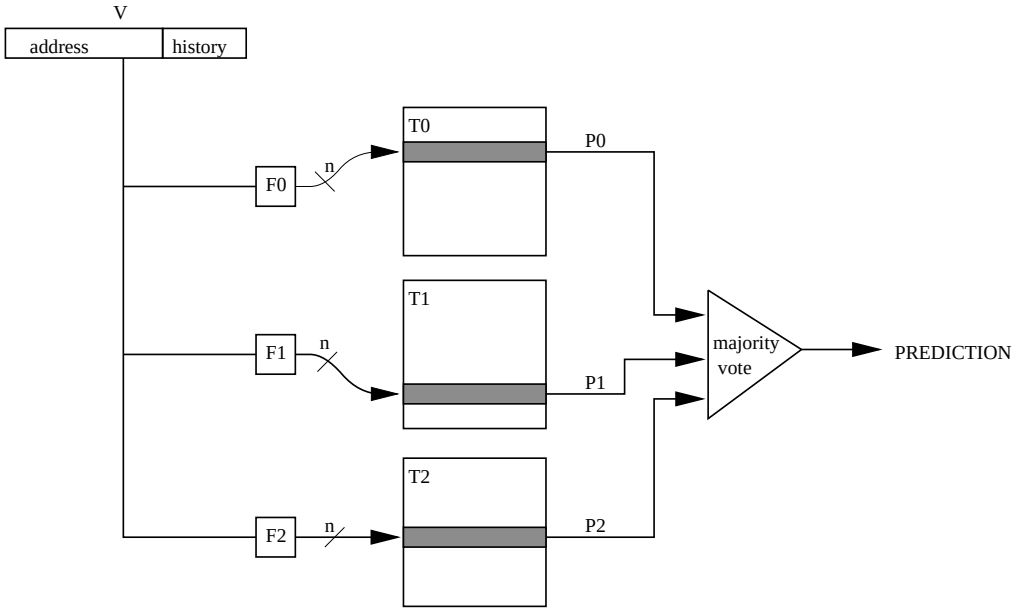

Figure 1: A Skewed Branch Predictor

banks. A destructive aliasing of V by W may occur in a bank, but the overall prediction on V may be correct if V does not suffer from destructive aliasing in the other banks. Using a different hashing function for each bank enables the skewed branch predictor to discriminate between two distinct vectors V and W without requiring tags.

We consider two policies for updating the predictors across multiple banks:

- A **total update** policy: each of the three predictors is updated in the normal way.
- A **partial update** policy: when a bank gives a bad prediction, it is not updated when the overall prediction is good. This wrong predictor is considered to be attached to another (address,history) pair. When the overall prediction is wrong, all banks are updated according to the outcome of the branch.

4 Analysis

Having defined the operation of the skewed branch predictor, we are now in a position to evaluate it. Our analysis includes three components: a description of our experimental setup and the design space explored, an analytical model, and a simulation model.

4-bit history				
Benchmarks	**pattern /branch**	**% comp. aliasing**	**% misp. 1-bit**	**% misp. 2-bit**
groff	1.77	0.17	5.07	3.55
gs	1.81	0.25	6.33	4.78
mpeg play	1.83	0.10	7.78	6.25
nroff	1.70	0.08	4.76	3.64
real gcc	2.22	0.39	8.76	6.76
sdet	1.69	0.16	5.16	3.36
verilog	1.97	0.14	6.53	4.66
video play	1.82	0.14	5.99	4.22
12-bit history				
Benchmarks	**pattern /branch**	**% comp. aliasing**	**% misp. 1-bit**	**% misp. 2-bit**
groff	6.35	0.60	3.71	2.86
gs	6.50	0.89	4.02	3.23
mpeg play	6.09	0.32	5.41	4.41
nroff	4.82	0.22	3.13	2.35
real gcc	10.41	1.83	6.18	5.30
sdet	4.82	0.47	3.01	2.15
verilog	9.22	0.64	4.32	3.27
video play	5.58	0.43	3.43	2.46

Table 1: Unaliased schemes with 4 bits and 12 bits of global history

4.1 Experimental Setup and Design Space

Benchmark Characterization It has been noted in recent studies [2, 6] that the use of the SPEC benchmark suite, and particularly the use of only user-level instructions to evaluate the performance of branch-prediction schemes can lead to false conclusions. The resulting simulations often underestimate the predictor-table sizes required for good performance on general purpose application. To ensure a demanding evaluation of the skewed branch predictor, we conducted all of our trace-driven simulations using the IBS-Ultrix benchmarks [11]. 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, and have been determined by other researchers to be a good test of branch prediction performance on general purpose application [2, 6].

Our method in this study was to fix a history length and then vary the table size. We first simulated an ideal unaliased global scheme (i.e., a predictor table of infinite size). The

misprediction ratios that we obtained are shown in table 1 for a history length of 4 and 12 bits, for both 1-bit and 2-bit predictors. We included unconditional branches as part of the global history bits.

The 2-bit saturating counter gives better prediction accuracy in an unaliased predictor table than the 1-bit predictor. Our intuition is that this difference mainly comes from loop branches. We also measured the number of patterns per static conditional branch, i.e., the average number of (address,history) pairs encountered with the same *address* value, and the percentage of compulsory aliasing, i.e., the ratio between the number of different (address,history) pairs referenced and the total number of dynamic conditional branches.

From these results, we observe that compulsory aliasing, with a history length of 12 bits, generally constitutes less than 1 % of the total of all dynamic conditional branches, except in the case of **real gcc**, which exhibits a compulsory-aliasing rate of 1.83 %. It should be noted that with such a history length, compulsory aliasing may represent a significant fraction of mispredictions in the unaliased branch predictor (more than 20% for 4 out of our 8 benchmarks).

Design Space Chosing the information that is used to divide branches into substreams is an open problem: branch address (and which bits), global history (and which length), path, or other yet to be devised, may be used. The purpose of this paper is not to discuss the relevance of using some combination of information or some other, but to show that most conflict aliasing effects can be removed by using a skewed predictor organization. For the remainder of this paper, the vector of information that will be used for recording branch-prediction information in a 2^n-entry branch-prediction bank will consist of the $2n-k$ lowest-order bits of the branch address and k bits of global history.

The functions f_0, f_1 and f_2 used for indexing the 3 banks in all the experiments illustrated in this paper are the same as those proposed for the skewed-associative cache in [8]:
Consider the decomposition of the binary representation of a vector V in bit substrings V = (V_2, V_1). V_1 and V_2 are two n-bit strings. Let $(y_n, y_{n-1}, .., y_1)$ be the binary representation of $Y = \Sigma_{i=1,n} y_i 2^{i-1}$. Now consider the function H defined as follows:

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

where $\oplus$ is the XOR (exclusive or) operation.
We can now define three different mapping functions as follows:

$$\begin{array}{rlcl} f_0: & \{0,..,2^{2n}-1\} & \longrightarrow & \{0,..,2^n-1\} \\ & (V_2, V_1) & \longrightarrow & H(V_1) \oplus H^{-1}(V_2) \oplus V_2 \end{array}$$

$$\begin{array}{rlcl} f_1: & \{0,..,2^{2n}-1\} & \longrightarrow & \{0,..,2^n-1\} \\ & (V_2, V_1) & \longrightarrow & H(V_1) \oplus H^{-1}(V_2) \oplus V_1 \end{array}$$

$$\begin{array}{rlcl} f_2: & \{0,..,2^{2n}-1\} & \longrightarrow & \{0,..,2^n-1\} \\ & (V_2, V_1) & \longrightarrow & H^{-1}(V_1) \oplus H(V_2) \oplus V_2 \end{array}$$

Further information about these functions can be found in [8]. The most interesting property of these functions is that if two vectors V and W map to the same entry in a bank, they will not conflict in the other banks.

For the purposes of comparison, we will use the *gshare* global scheme [4] for referencing the standard single-bank organization. In the *gshare* scheme, the low-order bits of the address and the global history are XORed to form an indexing value. When the number of history bits is less than the number of indexing bits, the history bits are XORed with *highest order* low-order address bits. This is because low-order address bits have better dispersion [4]. The skewed branch predictor described earlier will also be referred to as *gskewed* in the remainder of this paper.

4.2 Analytical Model

The aim of this section is to model analytically the behavior of the skewed branch predictor in an effort to explain how it removes conflict aliasing. In this discussion, we assume predictions based on 1-bit predictors.

Consider a particular read of the branch predictor with vector V, such that p represents the per-bank *aliasing probability* (which is assumed to be the same for the 3 banks) and b represents the probability that an entry of the unaliased predictor table is set to *taken* (b may change dynamically).

Four cases may occur:

1. With probability $(1-p)^3$, V is not aliased in any of the three banks: the prediction will be the same as the unaliased prediction.
2. With probability $3p(1-p)^2$, V is aliased in one bank, but not in the other two banks: the resulting majority vote will be in the same direction as the unaliased prediction.
3. With probability $3p^2(1-p)$, V is aliased in two banks, but not in the remaining one. With probability $b(1-b)^2 + (1-b)b^2$, predictions for both aliased banks are different from the unaliased prediction: the overall prediction is different from the unaliased prediction.
4. With probability p^3, V is aliased in all three banks.
   With probability $b((1-b)^3 + 3b(1-b)^2) + (1-b)(b^3 + 3(1-b)b^2)$, the predictions are different from the unaliased prediction in at least two prediction banks: the skewed prediction is different from the unaliased prediction.

In summary, the probability that a prediction in our 3-bank skewed predictor differs from the unaliased prediction is :

$$P_{sk} = 3p^2(1-p)b(1-b) + p^3b[3b(1-b)^2 + (1-b)^3] + p^3(1-b)[3(1-b)b^2 + b^3]$$

In contrast, the formula for a conventional 1-bank predictor table is:

$$P_{al} = [b(1-b) + (1-b)b]p = 2b(1-b)p$$

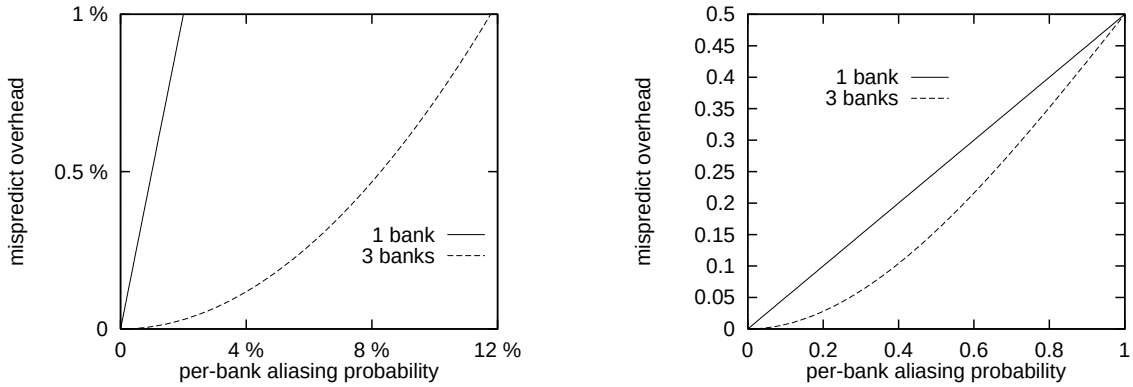

Figure 2: Mispredict overhead as a function of per-bank aliasing probability

P_{sk} and P_{al} are plotted in figure 2 for the worst case $b = 50$ %. The most relevant region of the curve is where the per-bank aliasing probability, p, is low (the left plot magnifies the curve in the region between 0 % and 12 %). The figure clearly shows that the 3-bank *gskewed* scheme supports a per-bank aliasing probability of more than 10 % while keeping under 1 % of mispredict overhead over the unaliased predictor, while the conventional 1-bank scheme supports only 2 % aliasing probability. This comes from the polynomial form of the P_{sk} formula.

At comparable storage resources, a 3-bank scheme has a greater per-bank aliasing probability than a 1-bank scheme, because each bank has a smaller number of entries. However, we will show later that the gain from having a polynomial form overcomes the increase of the per-bank aliasing probability.

It should be noted that the two curves join at 50 % mispredict overhead for an aliasing probability of 100 %. The skewed branch predictor will not remove aliasing effects in this case. In regions of execution where the aliasing probability is naturally high, particularly when there are bursts of capacity aliasing, the skewed branch predictor is not efficient.

To verify if our mathematical model is meaningful, we used measured aliasing rates as estimates for the aliasing probability p. We integrated these formulas over time using actual branch traces from IBS, evaluating p and b in windows of 50 conditional branches at a time, and adding the extrapolated misprediction count to those calculated in previous windows. Finally, we added the unaliased misprediction count. Actually, we cannot fully distinguish mispredictions due to aliasing from unaliased mispredictions. But statistically, if p_u is the unaliased misprediction rate, there is at most a (relative) error p_u on the extrapolated mispredict overhead, so it is less than 10 % on our set of benchmarks.

The results are shown in figure 3 for *gshare* (i.e. 1-bank scheme) and *gskewed* (i.e. 3-bank scheme) with an history length of 4. The notations T and P represent, respectively, a *total* and *partial* update policy. For the *partial update* scheme, we were not able to evaluate the local per-bank aliasing probability, so we only plotted the measured values. Recall that

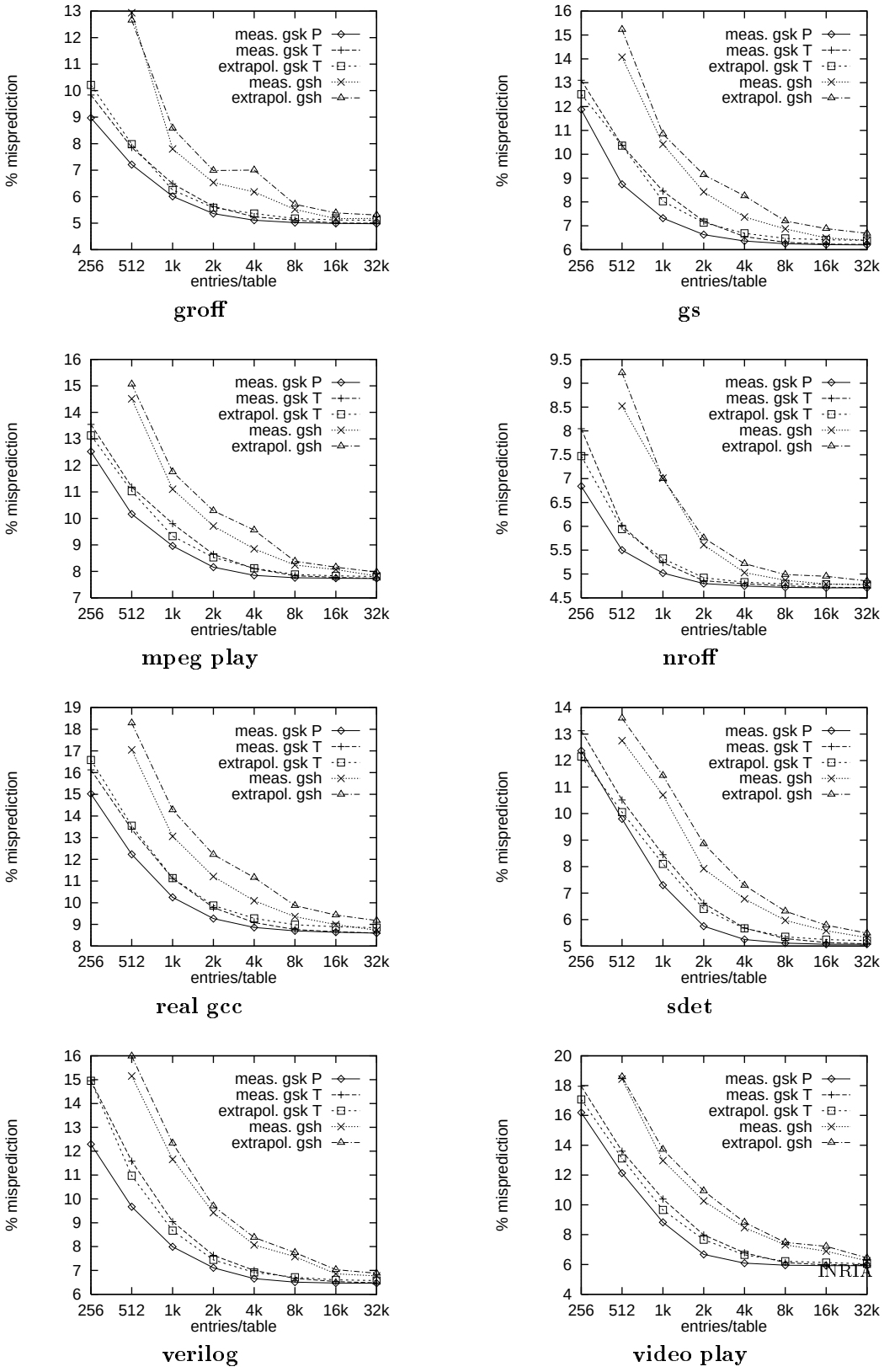

Figure 3: Extrapolated vs. measured misprediction rate

these curves are for one-bit predictors. The horizontal axis is indexed with the number of entries per table. For *gskewed*, we must multiply it by 3 to get the total number of entries. From these curves, we can see that our model follows the actual behavior.

Moreover, it appears that the *partial update* policy gives better results than the *total update* one. This may be explained by the fact that when *partial update* is used, an (address, history) pair can be predicted with only 2 entries, while the third entry can be used to predict another (address, history) pair. Intuitively, the *partial update* policy will result in less capacity aliasing than the *total update* one, thus decreasing per-bank aliasing probability.

4.3 Simulation Results

The analysis in the previous section shows that the skewed branch predictor is likely to significantly decrease conflict aliasing, but this effect could be negated by the redundancy introduced: branch-prediction information is recorded up to three times in three distinct banks.

We used simulation methods to examine configurations beyond the range of our analytical model, and to better understand the tradeoffs between reducing conflict aliasing at the expense of increased capacity aliasing.

We compiled simulation results for a range of *gskewed* and *gshare* table sizes. The results are plotted in figures 4 and 5, for a history size of 4 bits and 12 bits, respectively. We used a *partial update* policy, since it gives better results than the *total update* policy. In these figures, we compare the misprediction rates of *gskewed* with *gshare* at increasing hardware costs. In these simulations, we only used 2-bit saturating counters as predictors.

The interesting region of these curves is the region where the difference between the *gshare* misprediction rate and the unaliased value falls below 0.5 % (absolute). In this region,

> *the skewed branch predictor requires approximately half of the storage resources that a 1-bank scheme requires to achieve the same prediction accuracy.*

Notice that for all benchmarks and for a wide spectrum of predictor sizes, the skewed branch predictor consistently gives better prediction accuracy than the 1-bank predictor. Another effect of the skewed branch predictor is to remove pathological cases. This appears clearly on figure 5 for *nroff*. Such a pathological case may appear when some ping-pong phenomenum occurs in predictor table.

It should be noted that in most cases, a better prediction accuracy is achieved with a 3x16k-entry skewed branch predictor than with the unaliased predictor simulated previously. This can be explained by cold start misprediction on the unaliased predictor (section 4.1), which were counted as always mispredicted, while on real predictors some prediction is performed.

Limitations of the skewed branch predictor: The skewed branch predictor removes only conflict aliasing; it does not remove capacity aliasing. Capacity aliasing occurs in all

the banks. This happens mainly when the processor enters a region of the execution in which the last run is too remote in time for valid predictions to remain in the table. In such cases, there are bursts of aliasing and the aliasing probability is locally close to 100 %, so the skewed branch predictor does not remove these capacity aliasing effects.

However, in regions of execution where temporal locality for (address,history) pairs is high, the skewed branch predictor removes nearly all aliasing effects (figure 2).

Additional predictor banks yield diminishing returns: We also considered skewed configurations with five predictor banks. An analytical model similar to that of the previous section results in a polynomial of degree five instead of degree three, with a corresponding decrease in conflict aliasing. However, our simulations showed that there is very little benefit to increasing the number of prediction banks to five; it appears that a 3-bank skewed branch predictor removes the most significant part of conflict aliasing, and a more cost-effective use of resources is to double the size of the banks (and thus decrease capacity aliasing), rather than increase their number.

5 Conclusions and Future Work

Aliasing effects in branch-predictor tables have been recently identified as a significant contributor to branch misprediction rates. To better understand and minimize this source of prediction error, we have proposed a new branch-aliasing classification, inspired by the three-Cs model of cache performance.

Although previous branch-prediction research has resulted in reductions to compulsory and capacity aliasing, little has been done to reduce conflict aliasing. To that end, we have proposed skewed branch prediction, a technique that distributes predictors across multiple banks using distinct and independent hashing functions; multiple predictors are read in parallel and a majority vote is used to arrive at an overall prediction.

Our analytical model explains why skewed branch prediction works: in a standard one-bank table, the mispredict overhead increases linearly with the aliasing probability, but in an N-bank skewed organization, it increases as an N-th degree polynomial. Because we deal with per-bank aliasing probabilities, that range from 0 to 1, a polynomial growth rate is always preferable to a linear one.

The redundancy in a skewed organization increases the amount of capacity aliasing, but our simulation results show that this negative effect is more than compensated for by the reduction in conflict aliasing. For tables of 2-bit predictors indexed by both 4 and 12 bits of global history, a 3-bank skewed organization outperforms a standard 1-bank organization for all configurations with comparable total storage requirements. We found the update policy to be an important factor, with partial update consistently outperforming total update. Although 5-bank (or greater) configurations are possible, our simulations showed that the improvement over a 3-bank configuration is negligible. We also found skewed branch prediction to be less sensitive to pathological cases (e.g., nroff in Figure 5).

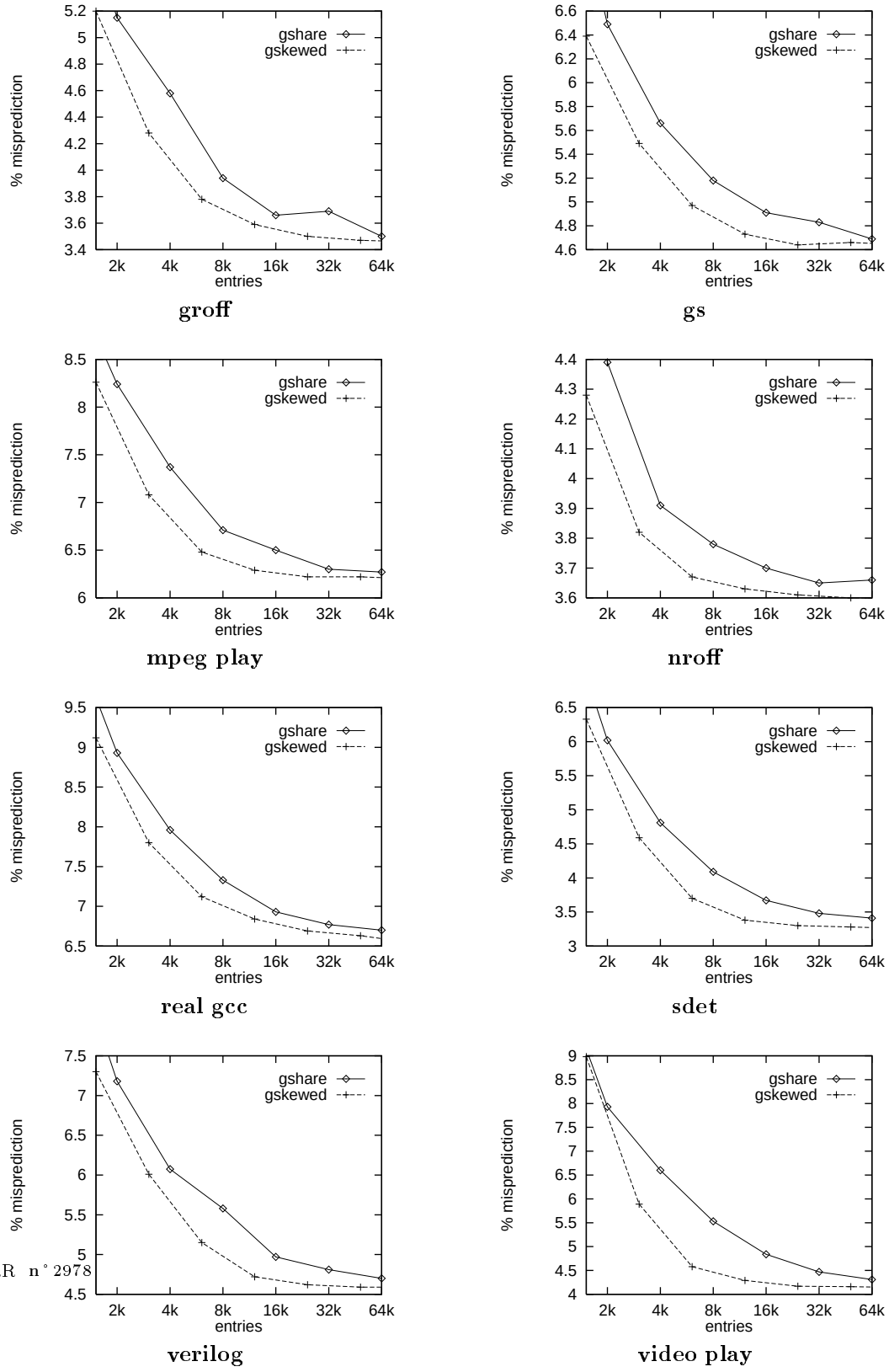

Figure 4: Misprediction percentage with 4 bits history

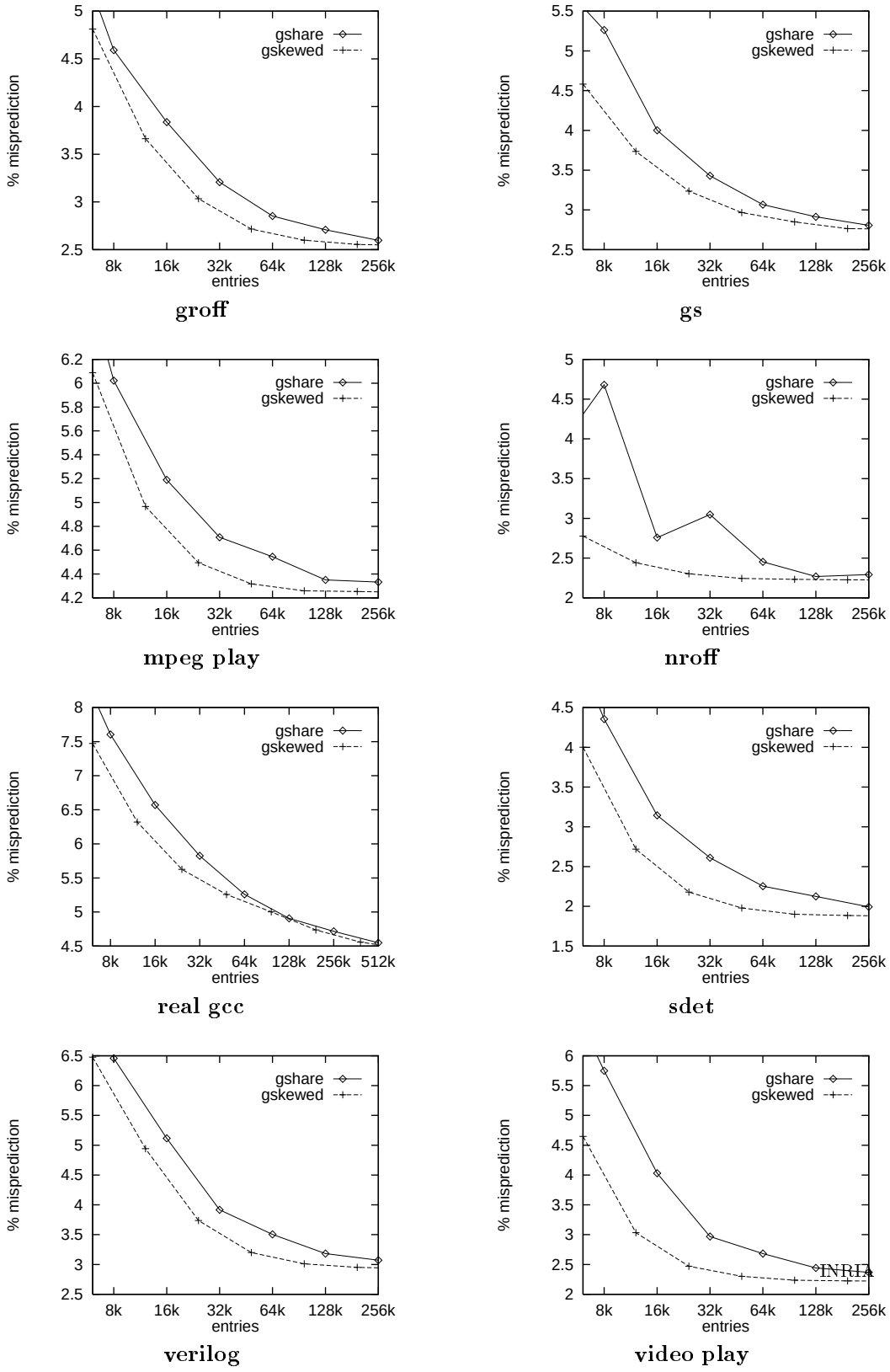

Figure 5: Misprediction percentage with 12-bit history

In addition to these performance advantages, skewed organizations offer a chip designer an additional degree of flexibility when allocating die area. For example, die-area constraints may not permit increasing a 1-bank predictor table from 16K to 32K, but a skewed organization offers a middle point: 3 banks of 8K entries apiece for a total of 24K entries.

Skewed branch prediction raises some new questions in branch-prediction research:

- **Update Policies:** Are there policies other than partial-update and total-update that offer better performance in a skewed branch predictor?
- **Distributed Predictor Encodings:** In our simulations we adopted the standard 2-bit predictor encodings and simply replicated them across 3 or 5 banks. Do there exist alternative "distributed" predictor encodings that are more space efficient, and more robust against conflict aliasing?

References

[1] F. Bodin and A. Seznec. Skewed associativity enhances performance predictability. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, May 1995.

[2] N. Gloy, C. Young, B. Chen, and M.D. Smith. An analysis of dynamic branch prediction schemes on system workloads. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996.

[3] M.D. Hill. *Aspects of Cache Memory and Instruction Buffer Performance*. PhD thesis, University of California, Berkeley, 1987.

[4] Scott McFarling. Combining branch predictors. Technical report, DEC, 1993.

[5] S.T Pan, K. So, and J.T. Rahmeh. Improving the accuracy of dynamic branch prediction using branch correlation. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systemes*, 1992.

[6] S. Sechrest, C.C. Lee, and T. Mudge. Correlation and aliasing in dynamic branch predictors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, May 1996.

[7] A. Seznec. A case for two-way skewed-associative cache. In *Proceedings of the 20th International Symposium on Computer Architecture*, May 1993.

[8] A. Seznec and F. Bodin. Skewed associative caches. In *Proceedings of PARLE' 93*, May 1993.

[9] J.E. Smith. A study of branch prediction strategies. In *Proceedings of the 8th Annual International Symposium on Computer Architecture*, May 1981.

[10] R.A. Sugumar and S.G. Abraham. Efficient simulation of caches under optimal replacement with applications to miss characterization. In *Proceedings of the ACM SIGMETRIC Conference*, 1993.

[11] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Coping with code bloat. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, June 1995.

[12] Tse-Yu Yeh. *Two-level adaptative branch prediction and instruction fetch mechanisms for high performance superscalar processors*. PhD thesis, University of Michigan, 1993.

[13] C. Young, N. Gloy, and M.D. Smith. A comparative analysis of schemes for correlated branch prediction. In *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)
ISSN 0249-6399