An Algorithm for Optimally Exploiting Spatial and Temporal Locality in Upper Memory Levels

Olivier Temam

Abstract—In this study, we present an extension of Belady's MIN algorithm that optimally and simultaneously exploits spatial and temporal locality. Thus, this algorithm provides a performance upper bound of upper memory levels. The purpose of this algorithm is to assess current memory optimizations and to evaluate the potential benefits of future optimizations. We formally prove the optimality of this new algorithm with respect to minimizing misses and we show experimentally that the algorithm produces nearly minimum memory traffic on the SPEC95 benchmarks.

Index Terms—Optimal memory management, local memory, cache, Belady.

1 Introduction

Many research studies focus on hardware optimizations for cache memories or, more generally, for the upper memory level. To drive future research, we wish to get hints at the performance upper bound of an upper memory level. In this study, we design an algorithm that provides the best performance that can be achieved with a given amount of memory space expressed in bytes, assuming we abrogate all hardware cost constraints and that memory performance is defined as memory traffic. We present and formally prove the optimality of an algorithm that simultaneously exploits temporal and spatial locality.

Our algorithm is based on Belady's MIN algorithm [2]. We extend Belady's algorithm to define a new algorithm that optimally exploits temporal and spatial locality. Belady's MIN algorithm provides the minimum miss ratio when temporal locality is optimally exploited and does not deal with spatial locality, thus providing incomplete information. We extend Belady's algorithm to include spatial locality and we formally prove the optimality of this new algorithm. This algorithm minimizes *miss ratios*, but we experimentally show that optimally exploiting temporal and spatial locality simultaneously, with miss ratio as a cost function, also leads to near-minimal *memory traffic*. Besides setting performance upper bounds, this algorithm can be used as an analysis tool. It is designed so that a number of strategies for exploiting spatial and temporal locality can be implemented and tested.

In Section 2, we outline the principles of our algorithm; the proof of optimality is provided in the Appendix. In Section 3, we present experimental results using our algorithm.

1.1 Limitations of the Study

In this study, we have ignored hardware cost and timing issues. Consequently, we wish to acknowledge that, even if we can physically implement an upper memory level that exhibits the same threshold miss ratio and memory traffic ratio as obtained in this study, such a memory may not achieve maximal performance when cost and, especially, timing issues are considered. Moreover, Belady's algorithm and the Extended Belady's algorithm proposed in this article rely on knowledge of future references, therefore, they are not likely to be used for practical applications, even though the compiler can extract some future reference information.

In the current implementation of the algorithm, as in the regular MIN algorithm [2], we do not discriminate between loads and stores. Consequently, we only optimize the fetch traffic and we do not optimize the write-back traffic.

1.2 Terminology

For the sake of clarity, we define a few terms which are used throughout the study.

- The term *reuse* corresponds to *achieved reuse*, while the term *locality* corresponds to *potential reuse* whether it is achieved or not.
- Temporal/Spatial reuse/locality *distance* is the number of memory requests between the source and target of reuse/locality.
- M_S denotes the upper memory level size, in words or bytes.
- B_S denotes the block size, in words or bytes.
- *Traditional caches* correspond to current cache implementations.
- The term *cache line* only applies to traditional caches.
- *Local memory* is a generic term defining upper memory levels and includes caches and any other type of memories. Local memories make no assumption on hardware implementation except for their size.
- *Blocks and block size* are generic terms used to denote the set of words brought simultaneously in local memory. In contrast to cache lines, they are not tied to caches and apply to any type of local memory.
- *Cache miss* only applies to caches, while *miss* applies to any type of local memory. Similarly, we use the terms *cache miss ratio* and *miss ratio*.

• *The author is with PRiSM, Computer Science Department, Versailles University, France. E-mail: temam@prism.uvsq.fr.*

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 108228.

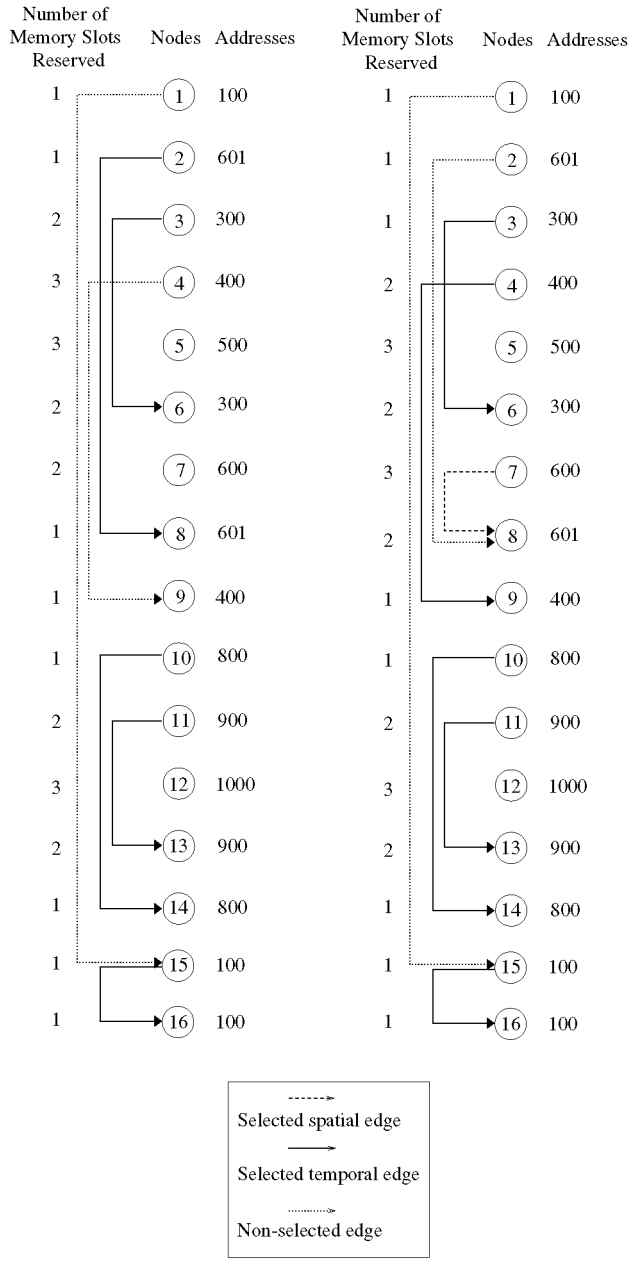

Fig. 1. Locality edge selection with Belady's and Extended Belady's algorithm.

2 An Algorithm for Optimally Exploiting Both Temporal and Spatial Locality

In this section, we first recall the main principles of Belady's algorithm and, then, we show how to extend it to spatial locality.

2.1 Belady's Algorithm

Belady's algorithm is a replacement algorithm for local memory that exploits temporal locality. Consider an address A that is referenced twice in a program trace at times t_1 and t_2, $t_2 > t_1$. Belady's algorithm determines whether A is kept in local memory between t_1 and t_2 so as to exploit the corresponding temporal locality. However, the decision to keep A is postponed until t_2. The criterion is the following: If, at any time t between t_1 and t_2, the number of local memory entries already reserved for storage is equal to the memory capacity, then the algorithm decides a posteriori not to keep A at t_1; otherwise, A is systematically kept.

We now provide a graph representation of this algorithm that is more convenient for introducing spatial locality.

2.2 A Graph Representation of Belady's Algorithm

Let us consider the address trace on the left of Fig. 1 and a local memory size $M_S = 3$ words. The issue is to select which words to keep in local memory and which ones to discard in order to minimize the total number of memory fetches. The graph represents temporal locality using edges. Two nodes referencing the same address, like nodes 1 and 15, are linked with an edge, called a *locality edge*. Note that only the last reference to an address can be a source of locality, i.e., there is no edge between node 1 and node 16. Deciding whether an address should be kept in local memory for later reuse is then equivalent to *selecting a locality edge* in our graph representation. Minimizing memory fetches is the same as maximizing the number of locality edges selected. Each time an edge is selected, a memory slot is reserved for the corresponding address at all nodes between the source and target nodes of the locality edge. An edge *cannot be selected* if the number of memory slots reserved at any of these nodes is already equal to the memory size M_S. We call $\sigma(i)$ **the number of memory slots reserved at node** i.

Let us illustrate Belady's algorithm on our example. At any node, one memory slot must be reserved to fetch the node address. Nodes are examined sequentially and if the current node is the target node of a locality edge, we need to decide whether the edge is selected or not. When node 6 is examined, we find it is the target node of edge (3, 6). At this time, $\sigma(4)$ and $\sigma(5)$ are equal to 1 (one memory slot for 400 and 500, respectively). One memory slot is necessarily reserved at nodes 3 and 6 to load 300, so $\sigma(3)$ and $\sigma(6)$ need not be checked. Since σ does not reach $M_S = 3$ at nodes 4 or 5, edge (3, 6) can be selected. The main intuition of Belady's algorithm is that *if a locality edge can be selected, selecting it always leads to an optimal solution*. Mattson et al. [12] later proved this assertion and the optimality of Belady's algorithm. Note that (3, 6) is selected at node 6, i.e., 300 is referenced at node 3 but the decision to keep it in local memory is only taken at node 6; that is why Belady's algorithm requires knowledge of future references. Similarly, edge (2, 8) can be selected when node 8 is examined. Note, however, that $\sigma(5)$ is then equal to 3 (M_S: one memory slot for 500, one for 601, and one for 300. Consequently, edge (4, 9) cannot be selected when node 9 is examined. Similarly, (10, 14), (11, 13), and (15, 16) are selected but (1, 15) is not selected. The total number of memory fetches is equal to 11 and is the minimum number of memory fetches for this trace. Suppose, for instance, that (1, 15) were selected. Then, at most (3, 6) (or (2, 8) or (4, 9)) and (11, 13) (or (10, 14)) could be selected. The total number of memory fetches would then be equal to 12.

2.3 Introducing Spatial Locality Edges

The only parameter that characterizes the exploitation of temporal locality in a local memory is the memory size M_S.

Symmetrically, we can introduce a parameter B_S, the block size, that characterizes the exploitation of spatial locality: *B_S is the maximum number of words that can be fetched together in local memory*; for caches, B_S is the cache line size. However, spatial locality is also characterized by a function F, where *$F(A)$ defines the set of words fetched along with word at address A*. For caches, the function F is the following (recall B_S is the cache line size in this case):

$$F_{cache} : A \rightarrow \{A_0, \ldots, A_{B_S-1}\},$$

$$\text{with } A_0 = (A - A \mod B_S), A_{B_S-1} = (A_0 + B_S - 1).$$

2.4 Extending Belady's Algorithm to Spatial Locality

When an address A is fetched at time t_1 in memory, the Extended Belady's algorithm can also fetch one or more additional addresses, A_i. The number and set of neighbor addresses that are eligible for simultaneous fetching are, respectively, defined by B_S and the function F. In the Extended Belady's algorithm, an address A_i is fetched together with A only if it is later referenced at time t_2 and if the number of local memory slots reserved between t_1 and t_2 is smaller than the local memory size. As long as the above criterion is met, up to the B_S neighboring addresses defined by $F(A)$ are fetched along with A. Note that, unlike with a cache line, only part of the B_S words of a block can be fetched.

We illustrate this extended algorithm below using the graph representation and, in Fig. 2, we provide a formal description of the algorithm.

2.5 Spatial Locality Edge Selection

For our example, let us use $B_S = 2$ and $F : A \rightarrow \{A, A+1\}$. At node 7, 600 must be fetched from memory. Together with 600, it is useful to fetch 601, which is later used at node 8. Therefore, there is a new locality edge between node 7 and node 8, as shown on the right of Fig. 1. While all other edges are **temporal locality edges**, this edge is a **spatial locality edge**. Let us apply Belady's algorithm until node 8. If edge (7, 8) is selected instead of edge (2, 8), then $\sigma(3)$ to $\sigma(6)$ are decreased by 1. At node 7, one memory slot is naturally used by 600, but we must also immediately reserve one memory slot for 601 since it is loaded *together* with 600. Unlike temporal locality edges, selecting spatial locality edges increases σ at the source node. With a B_S-word block size, σ at the source node can increase by up to $B_S - 1$. If (7, 8) is selected instead of (2, 8), $\sigma(5) = 2$ instead of 3 and, at node 9, it is now possible to select (4, 9). The other edge selections are unchanged and it is still not possible to select (1, 15). The total number of words fetched is still equal to 11, but the total number of misses is now equal to 10. Indeed, when (2, 8) is replaced by (7, 8), 600 is fetched from memory instead of being reused. There is no additional miss but one additional memory fetch. On the other hand, since (4, 9) is now selected, both a miss and a fetch are avoided. In our algorithm, we exploit temporal and spatial locality with the purpose of minimizing the overall *miss ratio*, not the memory traffic. However, we will experimentally show in Section 3 that, when a local memory is managed with this algorithm, memory traffic is close to the minimum. Basically, when temporal locality edges are traded for shorter spatial locality edges, memory traffic increases but σ decreases at several nodes, providing more opportunities for temporal locality exploitation, which, in turn, compensates for additional memory traffic due to spatial reuses, as in the above example.

2.6 Extended Algorithm

In Fig. 2, we present a formal description of the algorithm. We detail the main principles of the algorithm below.

1. Each node corresponds to a trace entry, i.e., a load/store reference. Nodes are examined sequentially. i is the node currently examined and $addr(i)$ its address.
2. We seek the source node $s_T(i)$ of the shortest temporal locality edge, i.e., the closest previous node which referenced $addr(i)$. Between nodes $s_T(i)$ and i, σ, the number of memory slots reserved, must never reach M_S; otherwise, it would not be possible to reserve an additional memory slot for $addr(i)$.
3. Similarly, we seek the source node $s_S(i)$ of the shortest spatial locality edge, i.e., the closest previous node where a miss occurred and where $addr(i)$ can be loaded together with $addr(s_S(i))$. Formally, the set of addresses that can be loaded at node $s_S(i)$ is $F(addr(s_S(i)))$. Again, σ must never reach M_S between nodes $s_S(i)$ and i.
4. If both $s_T(i)$ and $s_S(i)$ are empty, then $addr(i)$ must be fetched in memory as it is the target of neither a temporal nor a spatial locality edge that can be selected.
5. If a spatial locality edge between $j = s_S(i)$ and i is selected, a memory slot at nodes j up to $i-1$ must be reserved, while, for a temporal locality edge starting at node $j = s_T(i)$, a memory slot at nodes $j+1$ up to i must be reserved. A memory slot must be reserved at the source node of a spatial locality edge because $addr(i) \neq addr(j)$, while, at the source node of a temporal locality edge, $addr(i) = addr(j)$ and one memory slot is already reserved for $addr(i)$. If there are both a candidate temporal locality edge and a candidate spatial locality edge, the shortest edge, i.e., the one that increases the memory reservation at the smallest number of nodes, is selected.
6. To reserve one memory slot, we increase σ at each node between the source node and the target node ($j+1$ to $i-1$ for a temporal edge, j to $i-1$ for a spatial edge).

2.7 Differences Between Belady's Algorithm and the Extended Belady's Algorithm

Introducing spatial locality in Belady's algorithm raises several issues that we now outline using the previous example:

- Any two consecutive references to an address A define a temporal locality edge. On the other hand, there is a spatial locality edge between a node n and any other node only *if a miss occurred at node n*. Consequently, spatial locality edges are defined

```
Let M_S = local memory size.
Let N = number of nodes.
Let i = current node.
Let addr(i) = address at node i.

for (i = 0; i < N; i++)
   /* Step 1 */
   SelectNode (i)
endfor

procedure SelectNode (Node i)
   /* Step 2 */
   s_T(i) = max_{j<i} {j / addr(j) = addr(i)
        and ∀k ∈ ]j,i[ , σ(k) < M_S}
   /* Step 3 */
   s_S(i) = max_{j<i} {j / miss at node j
        and addr(i) ∈ F(addr(j)) and ∀k ∈ [j,i[ , σ(k) < M_S}
   /* Step 4 */
   if (s_T(i) = ∅ and s_S(i) = ∅) return
   /* Step 5 */
   if (s_T(i) ≥ s_S(i) − 1 or s_S(i) = ∅) then
      SelectEdge (s_T(i) + 1, i − 1)
   else
      SelectEdge (s_S(i), i − 1)
      addr(i) is fetched at node s_S(i)
   endif

procedure SelectEdge (Node kBegin, Node kEnd)
   /* Step 6 */
   for (k = kBegin; k <= kEnd; k++)
      increment σ(k)
   endfor
```

Fig. 2. Extended Belady's algorithm.

dynamically as the trace is scanned while temporal locality edges are defined *statically*. In other words, *the existence of spatial locality edges depends on edge selections at previous nodes*.

There is a spatial locality edge between node 7 and node 8 because 600 is only used once and necessarily induces a miss at node 7. Suppose now 600 is referenced at node 5, then there is a temporal locality edge between node 5 and node 7. In that case, there is a spatial locality edge between node 7 and node 8 only if node 7 is a miss, i.e., only if edge (5, 7) is not selected. Otherwise, there is a spatial locality edge between node 5 and node 8.

- In Belady's algorithm, each node can be the source and target of a single edge. With spatial locality, each node can be the source of up to $B_S - 1$ locality edges, where B_S is the block size and the target of two locality edges, a temporal and a spatial edge. In Belady's algorithm, the issue is to either select or not select an edge; with spatial locality, the issue is to decide whether an edge must be selected and to choose among multiple edges.
- Temporal and spatial locality edges have different properties. When selecting a temporal locality edge, σ is not modified at the source and target nodes. When selecting a spatial locality edge, σ at the source node is increased by one.

In summary, we have obtained and proved the following results:

1. Spatial locality edges can be dynamically built as the trace is examined without loss of optimality. Thanks to this property, many different types of F functions can be used.
2. We provide a simple formulation of temporal and spatial locality edges that enables the algorithm to treat them the same.
3. When a node is the target of multiple nodes, we prove that selecting the shortest locality edge always leads to an optimal solution.
4. We prove that Belady's selection criterion, i.e., greedily selecting edges always leads to an optimal solution, remains true under these new hypotheses.

Our algorithm is essentially defined by criteria 3 and 4. Intuitively, both Belady's and the Extended Belady's algorithms lead to an optimal selection because they favor the shortest locality edges, i.e., the edges that least increase the nodes' memory reservation count. Even though the two criteria are rather intuitive, the main issue was to formally prove the optimality of the selection algorithm driven by these two criteria. The proof is given in the Appendix.

3 Optimal Behavior of Local Memories

As mentioned in Section 2, spatial locality is characterized both by the block size B_S and by the function F, where $F(A)$ is the set of addresses fetched together upon miss on address A. In this section, $F = F_{cache}$, i.e., we use the same grouping function as in caches. B_S is varied from 8 bytes (1 word) to 512 bytes (64 words).

3.1 Experimental Framework

We use all SpecFp95 and SpecInt95 codes with the `ref` data sets. For each experiment, we collect 1 billion data address references, skipping the first 1 billion, on a DEC 21164 processor using ATOM [5]. In the current implementation of the algorithm, we do not discriminate between loads and stores. Therefore, memory traffic is solely defined by the number of *words fetched*. In the next sections, we essentially use two metrics: miss ratio and memory traffic ratio. Memory traffic ratio is defined as the ratio of the total number of words fetched over the total number of references. The word size is 64 bits and, for the sake of simplicity, all load/store requests are considered to be word requests. In all experiments, the local memory size is 32KB; caches are 4-way set associative. See [18] for more details on the experimental framework.

As we analyze strategies that simultaneously minimize miss ratio and memory traffic for different block size values, we report experiments in graphs where the x-axis corresponds to the miss ratio and the y-axis to the memory traffic ratio and each dot in a curve corresponds to a different block size value. Therefore, these graphs are projections of 3D graphs, where the block size B_S would be the z-axis. Because these graphs are projections, there are dots with different y-axis values and same x-axis value. Experiments with integer codes are reported in Fig. 3a and experiments with floating-point codes are reported in Fig. 3b. The dots corresponding to lowest/highest block size value are labeled with the block size in bytes. The following strategies are presented in Fig. 3:

1. Cache. 32KB cache, 4-way set associative.

2. Belady. Temporal locality is optimally exploited (Belady's replacement policy) and spatial locality is exploited as with normal cache lines, i.e., all words defined by F_{cache} are fetched upon miss; $M_S = 32$KB.
3. Extended Belady. Both temporal and spatial locality are optimally exploited using the algorithm of Section 2. The set of words that can be fetched upon miss is also defined by F_{cache} but, within this block, only those words selected for spatial reuse by the algorithm are fetched, the other words in the block are not fetched; $M_S = 32$KB.

3.2 The Extended Belady Strategy Achieves the Minimum Miss Ratio and Near-Minimum Memory Traffic

It is important to understand that the Belady strategy with $B_S = 1$ word (8 bytes) defines the minimum memory traffic for a given local memory size. With this strategy, temporal locality is optimally managed. Exploiting spatial locality in addition to temporal locality cannot further reduce memory traffic. Assume, ideally, that all remaining misses are eliminated by exploiting spatial locality without decreasing the number of temporal reuses. Then, the memory traffic would be the same: Exploiting a spatial locality removes a miss but the corresponding word is still fetched from memory so memory traffic is not reduced. Therefore, exploiting spatial locality can only *degrade* memory traffic: If temporal reuses are traded for spatial reuses, overall miss ratio can decrease but memory traffic increases.

The main issue is to maximize bandwidth utilization by finding the best trade-off between miss ratio and memory traffic. We now know that any strategy which attempts to maximize bandwidth utilization by exploiting both temporal and spatial locality would exhibit an equal or higher memory traffic than Belady with $B_S = 1$ word (8 bytes). Consequently, *this latter strategy defines a memory traffic lower bound* for any strategy that exploits either or both temporal and spatial locality. In Section 5, we only prove that our strategy minimizes miss ratio by simultaneously exploiting temporal and spatial locality. Fig. 3 provides experimental evidence that *the memory traffic induced by this strategy is very close to the memory traffic lower bound*. We cannot provide a more precise assertion: It is possible this strategy does achieve minimum memory traffic for each block size B_S but this assertion is neither proved nor particularly intuitive. However, we have shown that it is possible to simultaneously manage temporal and spatial locality in such a way that both miss ratio and memory traffic are close to their minimum.

3.3 Local Memories Crave Flexibility

Optimally exploiting temporal locality alone is sufficient to bring the integer codes miss ratio down to 1-5 percent in most cases. For such codes, exploiting spatial locality is not only almost useless, but it can significantly increase memory traffic. Conversely, optimally exploiting spatial locality even with 256 or 512-byte blocks brings floating-point codes miss ratio down from 27 percent ($B_S = 1$) to 1-2 percent with no significant additional memory traffic. This high discrepancy among programs' spatial locality properties suggests that local memories are used for incompatible types of codes, and that the cache line paradigm used in Cache and Belady is too rigid. This paradigm is studied in more detail in [18].

4 Related Work

Several previous studies on optimal memory management [1], [3] rely on Belady's optimal replacement algorithm [2]. Sugumar and Abraham [1] use Belady's algorithm to accurately characterize the notion of capacity and conflict misses. Burger et al. [3] use Belady's algorithm to define traffic inefficiency as the ratio between cache traffic and a perfectly managed cache using Belady's MIN algorithm.

Recently, several studies have attempted to better characterize spatial and temporal locality exploitation in current cache memories using either dynamic or static analysis [9], [14], [17]. Many studies also exist on improving spatial and temporal locality exploitation in cache memory. Several techniques attempt to better exploit temporal locality through a reduction of cache misses by affecting cache placement or replacement algorithms [15], [13]. Spatial locality is mostly exploited with different forms of hardware prefetching [6], [10]. Bypassing and selective caching has also been successfully investigated [7]. Sectored caches [8] attempt to reduce the fraction of useless words fetched by splitting cache lines in smaller sublines and only fetching necessary sublines. Seznec [16] also proposed *Decoupled Sectored Caches*, an original implementation of sectored caches that combines reduced memory traffic with low miss ratio by allowing several sublines to reside in different cache locations. Kumar and Wilkerson [11] vary the granularity of miss requests based on the data and load/store address to exploit spatial locality more selectively. However, few sophisticated techniques to improve spatial and temporal locality exploitation have actually been implemented in processors, with the notable exception of the *Assist-Cache* in the HP-7200 [4]. This mechanism combines aggressive spatial locality exploitation using prefetching with reduced cache pollution by discarding cache lines that are tagged *spatial-only*; this tag is provided by the compiler.

5 Conclusions

We have extended Belady's algorithm to develop an algorithm for determining the minimal miss ratio achieved with a given local memory space when temporal and spatial locality are optimally exploited. We have formally proven this algorithm is optimum, and we have also experimentally shown that this algorithm achieves close to minimum memory traffic. Therefore, with respect to these metrics, we have defined a theoretical upper bound of local memory performance. Using this algorithm, we show that very large block sizes can be used in small local memories if they are properly managed, without inducing significant additional memory traffic.

The Extended Belady's algorithm can be used to assess hardware optimizations. Software optimizations require altering the load/store sequence and, unless we can introduce the load/store schedule as a parameter in the

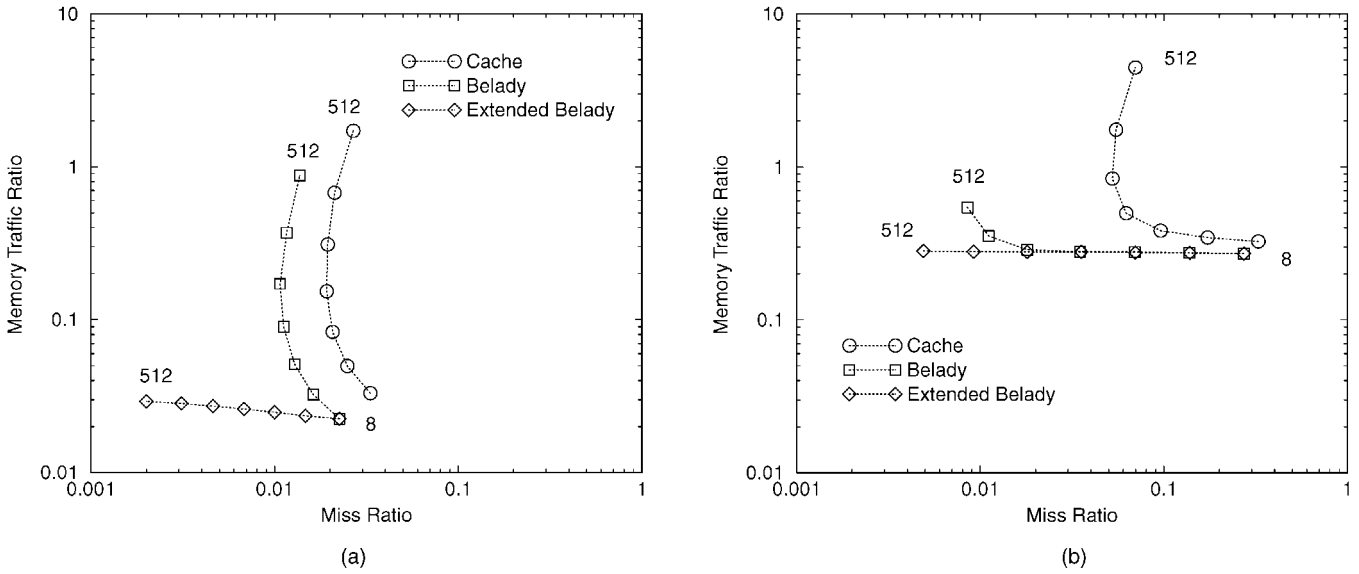

Fig. 3. Comparison of Cache, Belady, and Extended Belady strategies: (a) integer, (b) floating-point.

algorithm, we can only deal with a fixed address trace order.

APPENDIX

OPTIMALITY OF THE EXTENDED BELADY'S ALGORITHM

Definitions: Let M_S be the memory size in words. Let N be the number of trace entries. Let $\sigma_{Set}(j)$ be **the number of local memory slots reserved at node j assuming all the locality edges in Set have been selected**, where Set is any set of locality edges. Since the algorithm examines nodes sequentially, at node j, all locality edges which target node i are such that $i \leq j$ have been examined. So, we define $\Sigma(j)$ the **set of all locality edges that have been selected between node 1 and node j** (node j included). $\Sigma(0) = \emptyset$. Therefore, $\sigma_{\Sigma(j)}(m)$ indicates the number of local memory slots necessary at node m considering that all locality edges with a target node index lower or equal to j have been selected whenever possible.

Source of a locality edge: Let $i, i'/\exists R(i \rightarrow i')$. If $R(i \rightarrow i')$ is selected and R is a *temporal* locality edge, the memory requirements of nodes $]i; i'[$ are increased by 1. If i is selected and R is a *spatial* locality edge, the memory requirements of nodes $[i; i'[$ are increased by 1. $[i; i'[$ can be also denoted $]i-1; i'[$. Therefore, we can define $\lambda(R(i \rightarrow i'))$, **the lower bound of the open interval of nodes which memory requirements are incremented if locality edge R is selected**. $\lambda(R(i \rightarrow i')) = i$ if R is a temporal locality edge and $\lambda(R(i \rightarrow i')) = i - 1$ if R is a spatial locality edge.

General formulation of a locality edge selection: Let $i, i'/\exists R(i \rightarrow i')$. If $R(i \rightarrow i')$ is selected, the memory requirements of all nodes in $]\lambda(R(i \rightarrow i')); i'[$ are increased by 1.

Proof of Optimality. The optimality of the extended Belady's algorithm can be simply phrased as follows: Minimizing memory fetches is equivalent to maximizing the number of locality edges selected, i.e., maximizing $card(\Sigma(N))$.[1] **A maximum set of selected locality edges is denoted** Σ^{max}. Note that there may exist several maximal sets. The Extended Belady's algorithm is presented below, and the theorem asserting optimality is formulated next.

Extended Belady's Algorithm:

Label:

Let j be the current node. Let $\Sigma(j-1)$ be the set of locality edges already selected. Let $T(i) = \{i'/\exists R(i \rightarrow i')\};$. Let

$$I(j) = \{i/j \in T(i) \text{ and } \forall m \in]\lambda(R(i \rightarrow j)); j[, \sigma_{\Sigma(j-1)}(m) < M_S\},$$

the set of locality edges with j as target node that can be selected. Let

$$I_{M_S}(j) = \{i/j \in T(i) \text{ and } \exists m \in]\lambda(R(i \rightarrow j)); j[/\sigma_{\Sigma(j-1)}(m) = M_S\},$$

the set of locality edges with j as target node that cannot be selected. Finally, let $i_{sup} = \max_{i \in I(j) \cup I_{M_S}(j)} i$.

The general selection criterion distinguishes three cases:

1. $I(j) = \emptyset$ and $I_{M_S}(j) = \emptyset$, then $\Sigma(j) = \Sigma(j-1)$.
2. $I(j) \neq \emptyset$, then $\Sigma(j) = \Sigma(j-1) \cup R(i_{sup} \rightarrow j)$.
3. $I(j) = \emptyset$ and $I_{M_S}(j) \neq \emptyset$, then $\Sigma(j) = \Sigma(j-1)$.

j is increased by 1. Goto *Label* if $j \leq N$.

The meaning of the selection criteria is the following:

1. No locality edge can be selected since j is not the target node of any locality edge.
2. If the number of local memory slots reserved at any node within $]\lambda(R(i_{sup} \rightarrow j)); j[$ is strictly smaller than M_S, then $R(i_{sup} \rightarrow j)$ is selected.
3. If for all locality edges $R(i \rightarrow j)$ there exists at least one node within $]\lambda(R(i \rightarrow j)); j[$ where M_S local memory slots are already used, then no locality edge can be selected.

1. Recall $card(S)$ denotes the number of elements in set S.

THEOREM OF OPTIMALITY. *$\Sigma(N)$ is a maximal selection.*

PROOF. The theorem is proven by induction on j ($1 \leq j \leq N$), the current node index. First, we prove that if we build $\Sigma(N)$ using the above selection criterions, there exists a maximal selection that contains $\Sigma(N)$. Then we prove that $\Sigma(N)$ is a maximal selection.

Step $j = 1$: According to criterion 1, $\Sigma(1) = \emptyset$ and $\forall \Sigma^{max}, \emptyset \subset \Sigma^{max}$, so $\exists \Sigma^{max} / \Sigma(1) \subset \Sigma^{max}$.

Step $j-1$: Induction hypothesis.

Step j:

1. No locality edge can be selected, so $\Sigma(j) = \Sigma(j-1)$.
2. For the proof, we need to consider separately the case where $i_{sup} = j-1$, because then $j = \emptyset$. So the first case is (a) $\forall i \in I(j), i < j-1$. The second case is (b) $i_{sup} = j-1$.
   a. The goal is to show that there exists a maximal selection that contains $\Sigma(j-1) \cup (i_{sup} \rightarrow j)$.[2] For that purpose, we pick one maximal selection $\Sigma^{max}_{-i_{sup}}$ that does not contain i_{sup} and we distinguish two cases: i) the memory requirements of all nodes within $]\lambda(R(i_{sup} \rightarrow j));\ j[$ are strictly smaller than M_S and we will show this hypothesis is impossible; ii) there exists nodes within $]\lambda(R(i_{sup} \rightarrow j));\ j[$, where M_S local memory slots are reserved. In this latter case, we will build a maximal selection that contains $R(i_{sup} \rightarrow j)$ by replacing one of the locality edges within $\Sigma^{max}_{-i_{sup}}$ by $R(i_{sup} \rightarrow j)$.
      i. The set of nodes which memory requirements are increased by selection of $R(i_{sup} \rightarrow j)$ is $]\lambda(R(i_{sup} \rightarrow j));\ j[$. Assume that

$$\forall m \in\ ]\lambda(R(i_{sup} \rightarrow j));\ j[, \sigma_{\Sigma^{max}_{-i_{sup}}}(m) <\ M_S,$$

then since $\sigma_{\Sigma^{max}_{-i_{sup}} \cup R(i_{sup} \rightarrow j)}(m) = \sigma_{\Sigma^{max}_{-i_{sup}}}(m) + 1$ for all

$$m \in\ ]\lambda(R(i_{sup} \rightarrow j));\ j[,$$
$$\forall m \in\ ]\lambda(R(i_{sup} \rightarrow j));\ j[,$$

$\sigma_{\Sigma^{max}_{-i_{sup}} \cup R(i_{sup} \rightarrow j)}(m) \leq M_S$. Therefore, $\Sigma^{max}_{-i_{sup}} \cup R(i_{sup} \rightarrow j)$ is a valid selection of locality edges since memory capacity is never exceeded. However,

$$card\Big(\Sigma^{max}_{-i_{sup}} \cup R(i_{sup} \rightarrow j)\Big) = card\Big(\Sigma^{max}_{-i_{sup}}\Big) + 1$$

which is impossible since $\Sigma^{max}_{-i_{sup}}$ is a maximal selection, hence the contradiction.

      ii. Therefore,

$$M_{max}(j) = \{m \in\ ]\lambda(R(i_{sup} \rightarrow j));\ j[/ \sigma_{\Sigma^{max}_{-i_{sup}}}(m) = M_S\}$$

is not empty. However, note that, by hypothesis,

$$\forall m \in\ ]\lambda(R(i_{sup} \rightarrow j));\ j[, \sigma_{\Sigma(j-1)}(m) <\ M_S.$$

Therefore, there necessarily exist locality edges within $\Sigma^{max}_{-i_{sup}} \backslash \Sigma(j-1)$,[3] that saturate the memory slot reservations of nodes in $M_{max}(j)$. Formally, let us show that

$$L_{max}(j) = \{l \in \Sigma^{max}_{-i_{sup}} \backslash \Sigma(j-1), l' \in T(l) / \exists m \in M_{max}(j)\ \cap\ ]\lambda(R(l \rightarrow l'));\ l'[\ \}$$

is not empty. Assume $L_{max}(j)$ is empty, then

$$\forall l \in \Sigma^{max}_{-i_p} \backslash \Sigma(j-1),$$
$$\forall l' \in T(l),]\lambda(R(l \rightarrow l')); l'[\ \cap\ ]\lambda(R(i_{sup} \rightarrow j)); j[\ = \emptyset.$$

Therefore,

$$\forall m \in\ ]\lambda(R(i_{sup} \rightarrow j));\ j[,$$
$$\sigma_{\Sigma^{max}_{-i_{sup}}}(m) = \sigma_{\Sigma(j-1)}(m) <\ M_S$$

which is contradictory with $M_{max}(j) \neq \emptyset$. So, $L_{max}(j) \neq \emptyset$.

Now, we need to show that there exists a locality edge $R(m \rightarrow m')$ in $\Sigma^{max}_{-i_{sup}} \backslash \Sigma(j-1)$ that increases the memory requirements of at least all nodes in $]\lambda(R(i_{sup} \rightarrow j));\ j[$, i.e., such that

$$]\lambda(R(i_{sup} \rightarrow j));\ j[\ \subset]\lambda(R(m \rightarrow m'));\ m'[.$$

If we can find such a locality edge, then it would be valid to replace $R(m \rightarrow m')$ with $R(i_{sup} \rightarrow j)$ in $\Sigma^{max}_{-i_{sup}}$.

Note that the target node of any locality edge in $\Sigma^{max}_{-i_{sup}} \backslash \Sigma(j-1)$ is necessarily greater or equal than j since a locality edge is selected only when its target node is examined. Now, we pick the locality edge in $L_{max}(j)$ which source node is the farthest away from node j. Because its target node is necessarily beyond or at node j, this locality edge should encompass all nodes in $M_{max}(j)$. Let us now prove this assertion.

We showed that $L_{max}(j) \neq \emptyset$, so let

$$l_{min} \in L_{max}(j),$$
$$l'_{min} \in T(l_{min}) / \lambda(R(l_{min} \rightarrow l'_{min})) = min_{l \in L_{max}(j), l' \in T(l)} \lambda(R(l \rightarrow l')).$$

Then, since $\forall l \in L_{max}(j), \forall l' \in T(l)$, $l' \geq j$,

2. Note that the selection criterion does not say a maximal selection is obtained *only* if $R(i_{sup} \rightarrow j)$ is selected, just that there necessarily exists a maximal selection which contains this locality edge.

3. Recall $A \setminus B$ denotes $x \in A / x \notin A \cup B$.

$$
\begin{aligned}
&]\lambda(R(l \to l')); \\
&l[\ \cap \ ]\lambda(R(i_{sup} \to j)); j[\\
&\subset \ ]\lambda(R(l_{min} \to l'_{min})); \\
&l'_{min}[\ \cap \ ]\lambda(R(i_{sup} \to j)); j[.
\end{aligned}
$$

Since we showed above that

$$
\begin{aligned}
&\forall m \in M_{max}(j), \\
&\exists l \in L_{max}(j), \\
&l' \in T(l)/m \in \ ]\lambda(R(l \to l')); l'[,
\end{aligned}
$$

then

$$
\begin{aligned}
&M_{max}(j) \subset \cup_{l \in L_{max}(j), l' \in T(l)} \ ]\lambda(R(l \to l')); \\
&l'[\ \cap \ ]\lambda(R(i_{sup} \to j)); j[\\
&= \ ]\lambda(R(l_{min} \to l'_{min}); \\
&l'_{min}[\ \cap \ [\lambda(R(i_{sup} \to j)); j[.
\end{aligned}
$$

We can then show that replacing $R(l_{min} \to l'_{min}))$ with i_{sup} in $\Sigma^{max}_{-i_{sup}}$ breeds a valid and maximal selection.

Note that $\forall m \in \ ]\lambda(R(l_{min} \to l'_{min})); l'_{min}[$, $\sigma_{\Sigma^{max}_{-i_{sup}} \setminus R(l_{min} \to l'_{min})}(m) = \sigma_{\Sigma^{max}_{-i_{sup}}}(m) - 1$. Considering that

$$
\begin{aligned}
&M_{max}(j) \subset \ ]\lambda(R(l_{min} \to l'_{min}; \\
&l'_{min}[\ \cap \ ]\lambda(R(i_{sup} \to j)); j[, \\
&\forall m \in \ ]\lambda(R(i_{sup} \to j)); \\
&\lambda(R(l_{min} \to l'_{min}))], \\
&\sigma_{\Sigma^{max}_{-i_{sup}}}(m) = \sigma_{\Sigma(j-1)}(m) < M_S.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
&\forall m \in \left]\lambda(R(i_{sup} \to j)); \lambda(R(l_{min} \to l'_{min}))\right], \\
&\sigma_{\Sigma^{max}_{-i_{sup}} \cup R(i_{sup} \to j)}(m) \leq M_S,
\end{aligned}
$$

so $\sigma_{\Sigma^{max}_{-i_{sup}} \setminus R(l_{min} \to l'_{min}) \cup R(i_{sup} \to j)}(m) \leq M_S$. And

$$
\forall m \in \left]\lambda(R(l_{min} \to l'_{min})); \lambda(R(i_{sup} \to j))\right[,
$$

$$
\begin{aligned}
&\sigma_{\Sigma^{max}_{-i_{sup}} \setminus R(l_{min} \to l'_{min}) \cup R(i_{sup} \to j)}(m) \\
&= \sigma_{\Sigma^{max}_{-i_{sup}}}(m) \leq M_S.
\end{aligned}
$$

Therefore,

$$
\Sigma^{max}_{-i_{sup}} \setminus R(l_{min} \to l'_{min}) \cup R(i_{sup} \to j)
$$

is a valid selection, i.e., memory capacity is never exceeded.

Since

$$
\begin{aligned}
&card(\Sigma^{max}_{-i_{sup}} \setminus R(l_{min} \to l'_{min}) \cup R(i_{sup} \to j) \\
&\quad = card\Sigma^{max}_{-i_{sup}}
\end{aligned}
$$

and $\Sigma^{max}_{-i_{sup}}$ is a maximal selection,

$$
\Sigma^{max}_{-i_{sup}} \setminus R(l_{min} \to l'_{min}) \cup R(i_{sup} \to j)
$$

is also a maximal selection. Therefore, a maximal selection containing $\Sigma(j-1) \cup R(i_{sup} \to j)$ was built, hence, the proposition.

b. Let $\Sigma^{max}_{-i_{sup}}$ be a maximal selection that does not contain $R(i_{sup} \to j)$. The only consequence of selecting $R(i_{sup} \to j)$ is that one additional memory slot must be reserved at all nodes within $]\lambda(R(i_{sup} \to j)); j[$. If

$$
\begin{aligned}
&]\lambda(R(i_{sup} \to j); j[= \emptyset, \\
&\forall m \in \ ]\lambda(R(i_{sup} \to j)); j[/\sigma_{\Sigma^{max}_{-i_{sup}} \cup R(i_{sup} \to j)}(m) \\
&= \sigma_{\Sigma^{max}_{-i_{sup}}}(m) \leq M_S.
\end{aligned}
$$

Therefore, $\forall \Sigma^{max}_{-i_{sup}}$, $\Sigma^{max}_{-i_{sup}} \cup R(i_{sup} \to j)$ is a valid selection. Since

$$
card(\Sigma^{max}_{-i_{sup}} \cup R(i_{sup} \to j)) = card(\Sigma^{max}_{-i_{sup}}) + 1
$$

and $\Sigma^{max}_{-i_{sup}}$ is maximal, there is a contradiction. So, there is no maximal selection that does not contain $R(i_{sup} \to j)$ in this case.

3. For this criterion, we need to show that there exists a maximal selection that contains $\Sigma(j-1)$ and that does not contain any locality edge in $I_{M_S}(j)$. By induction hypothesis, there exists a maximal selection Σ^{max} that contains $\Sigma(j-1)$. If $\forall i \in I_{M_S}(j), R(i \to j) \notin \Sigma^{max}$, then the desired result is obtained. Let

$$
i \in I_{M_S}(j)/R(i \to j) \in \Sigma^{max}.
$$

According to the definition of $I_{M_S}(j)$, $\exists m \in \ ]\lambda(R(i \to j)); j[/\sigma_{\Sigma(j-1)}(m) = M_S$. Therefore, $\sigma_{\Sigma(j-1) \cup R(i \to j)}(m) = M_S + 1$ and, since $\Sigma(j-1) \cup R(i \to j) \subset \Sigma^{max}$,

$$
\sigma_{\Sigma^{max}}(m) = M_S + 1,
$$

which is impossible (memory capacity exceeded), hence, the contradiction.

$\Sigma(N)$ is a maximal selection: Up to now we have proven that, at any step j, if the selection criterions are applied there exists a maximal selection Σ^{max} that contains $\Sigma(j)$. So, there exists Σ^{max} that contains $\Sigma(N)$. We are going to prove $\Sigma^{max} \subset \Sigma(N)$.

Let $\Sigma^{max}/\Sigma(N) \subset \Sigma^{max}$ and $card(\Sigma(N)) < card(\Sigma^{max})$, i.e., $\Sigma(N)$ is not a maximal selection. Then, there exists a node j where a locality edge was selected in Σ^{max} and no locality edge was selected in $\Sigma(N)$. Let $j \in [1; N]/\exists i/j \in T(i), R(i \to j) \in \Sigma^{max}$ and $R(i \to j) \notin \Sigma(N)$. If $I(j) = \emptyset$, then, since $\Sigma(j-1) \subset \Sigma^{max}$, no locality edge with j as target node can be selected in Σ^{max} and the assumption is incorrect. If $I(j) \neq \emptyset$, then a locality edge has been selected for inclusion in $\Sigma(j)$ according to the selection criteria and the assumption is incorrect. Therefore, $\Sigma(N)$ is a maximal selection. □

COROLLARY 1. *The theorem is still true if all locality edges with source node $j, j \geq i$ are not known when node i is examined.*

PROOF. The selection criterion of the extended algorithm makes no assumption on any locality edges with source node $j, j \geq i$ when node i is examined. □

ACKNOWLEDGMENTS

This work was supported by the Esprit Agency DG XIII under grant Esprit LTR MHAOTEU, No. 24942.

REFERENCES

[1] S.G. Abraham and R.A. Sugumar, "Efficient Simulation of Caches Under Optimal Replacement with Applications to Miss Characterization," *Proc. ACM Int'l Conf. Measurement and Modeling of Computer Systems,* 1993.
[2] L.A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer," *IBM Systems J.,* vol. 5, no. 2, pp. 78-101, 1966.
[3] D. Burger, A. Kägi and J.R. Goodman, "Memory Bandwidth Limitations of Future Microprocessors," *Proc. 23rd ACM Int'l Symp. Computer Architecture,* Philadelphia May 1996.
[4] K.K. Chan, C.C. Hay, J.R. Keller, G.P. Kurpanek, F.X. Schumacher, and J. Sheng, "Design of the HP PA-7200 CPU," *Hewlett-Packard J.,* Feb. 1996.
[5] Digital Equipment Corp., "Atom, User Manual," technical report, Digital Equipment Corp., Maynard, Mass., June 1995.
[6] J.W.C. Fu, J.H. Patel, and B.L. Janssens, "Stride Directed Prefetching in Scalar Processors," *Proc. 26th IEE Int'l Symp. Microarchitecture,* 1992.
[7] E.D. Granston and A.V. Veidenbaum, "An Integrated Hardware/Software Solution for Effective Management of Local Storage in High-Performance Systems;" *Proc. Int'l Conf. Parallel Processing,* vol. II, pp. 83-90, Aug. 1991.
[8] J.L. Hennessy and D.A. Patterson *Computer Architecture: A Quantitative Approach, second ed.* Morgan Kaufmann 1996.
[9] A.S. Huang and J.P. Shen, "The Intrinsic Bandwidth Requirements of Ordinary Programs," *Proc. Seventh Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* pp. 105-114, Cambridge, Mass., 1-5 Oct. 1996.
[10] N.P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small, Fully-Associative Cache and Prefetch Buffers," *Proc. 17th ACM Int'l Symp. Computer Architecture,* pp. 364-373, May 1990.
[11] S. Kumar and C. Wilkerson, "Exploiting Spatial Locality in Data Caches Using Spatial Footprints," *Proc. 25th Int'l Symp. Computer Architecture,* Barcelona, Spain, June 1998.
[12] R.L. Mattson, J. Gecsei, D.R. Slutz, and I.L. Traiger, "Evaluation Techniques for Storage Hierarchies," *IBM Systems J.,* vol. 9, no. 2, pp. 78-117, 1970.
[13] S. McFarling, "Cache Replacement with Dynamic Exclusion," *Proc. 19th Ann. Int'l Symp. Computer Architecture,* pp. 191-200, Gold Coast, Australia, 19-21 May 1992, also in *Computer Architecture News,* vol. 20, no. 2, May 1992.
[14] K.S. McKinley and O. Temam, "A Quantitative Analysis of Loop Nest Locality," *Proc. Seventh Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* Cambridge Oct. 1996.
[15] A. Seznec, "A Case for Two-Way Skewed-Associative Caches," *Proc. 20th Int'l Symp. Computer Architecture,* pp. 169-178, San Diego, Calif., May 1993.
[16] A. Seznec, "Decoupled Sectored Caches: Conciliating Low Tag Implementation Cost and Low Miss Ratio," *Proc. 21st Int'l Symp. Computer Architecture,* pp. 384-393, 1994.
[17] F.J. Sanchez, A. Gonzalez, and M. Valero, "Static Locality Analysis for Cache Management," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques (PACT-97),* San Francisco, Nov. 1998.
[18] O. Temam, "Investigating Optimal Local Memory Performance," *Proc. Eighth Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* San Jose, Calif., Oct. 1998.

Olivier Temam received his PhD from Rennes University, France, in 1993. He has been an assistant professor at Versailles University, France, since 1994. His research interests are in memory and processor architectures, with a special emphasis on performance analysis of memory hierarchies.