Modeling Set Associative Caches Behavior for Irregular Computations*

Basilio B. Fraguela, Ramón Doallo
Dept. Electrónica e Sistemas. Univ. da Coruña (SPAIN)

Emilio L. Zapata
Dept. de Arquitectura de Computadores. Univ. de Málaga (SPAIN)
e-mail: {basilio,doallo}@udc.es, ezapata@ac.uma.es

Abstract

While much work has been devoted to the study of cache behavior during the execution of codes with regular access patterns, little attention has been paid to irregular codes. An important portion of these codes are scientific applications that handle compressed sparse matrices. In this work a probabilistic model for the prediction of the number of misses on a K-way associative cache memory considering sparse matrices with a uniform or banded distribution is presented. Two different irregular kernels are considered: the sparse matrix-vector product and the transposition of a sparse matrix. The model was validated with simulations on synthetic uniform matrices and banded matrices from the Harwell-Boeing collection.

Keywords: Sparse matrix, irregular computation, cache performance, probabilistic model.

1 Introduction

Sparse matrices are in the kernel of many numerical applications. Their compressed storage [2], which permits both operations and memory savings, generates irregular access patterns. This fact reduces and makes hard to predict the performance of the memory hierarchy. In this work a probabilistic model for the prediction of the number of misses on a K-way associative cache memory considering sparse matrices with a uniform or banded distribution is presented. We want to emphasize that an important body of the model is reusable in different algebra kernels.

The most important approach to study cache behavior has traditionally been the use of trace-driven simulations [7], [9], [12] whose main drawback is the large amount of time needed to process the traces. Another possibility is nowadays provided by the performance monitoring tools of modern microprocessors (built-in hardware counters), that make available data such as the number of cache misses. These tools are obviously restricted to the evaluation of the specific cache architectures for which they are available. Finally, analytical models present the advantage that they reduce the times for obtaining the estimations and make the parametric analysis of the cache more flexible. Their weak point has traditionally been their limited precision. Although some models use parameters extracted from address traces [1], more general analytical models have been developed that require no input traces. While most of the previous works focuse on dense algebra codes [5], [11], little attention has been devoted to sparse kernels due to the irregular nature of their access patterns. For example, [10] studies the self interferences on the vector involved in the sparse matrix-vector product on a direct-mapped cache without considering interferences with other data structures and they do not derive a general framework to model this kind of codes.

Nevertheless, as an example of the potential usability of such types of models, we have modeled the cache behavior of a common algebra kernel, the sparse matrix-vector product, and a more complex one, the transposition of a sparse-matrix. This last code includes different access patterns and presents an important degree of data reusability for certain vectors.

The remainder of the paper is organized as follows: Section 2 presents the basic model parameters and concepts. Sparse matrix-vector product cache behavior is modeled in Section 3, while Section 4 is dedicated to the modeling of the transposition of a sparse matrix. Both models are extended to banded matrices in Section 5. In Section 6 the models are verified and the cache behavior they depict is studied. Finally, Section 7 concludes the paper.

2 Probabilistic model

Our model considers three types of misses: intrinsic misses, and self and cross interferences. An intrinsic miss takes place the first time a memory block is accessed. Self interferences are misses on lines that have been previously replaced in the cache by another line that belongs to the same program vector. Cross interferences refer to misses on lines that have been replaced between two consecutive references by lines belonging to other vectors.

In direct mapped caches each memory line is always mapped to the same cache line, so replacements take place whenever accesses to two or more memory lines mapped to the same cache line take place. However, in K-way asssociative caches lines are mapped to a set of K cache lines. In this case the line to be replaced is selected following a

*This work was supported by the Ministry of Education and Science (CICYT) of Spain under project TIC96-1125-C03, Xunta de Galicia under Project XUGA20605B96, and E.U. Brite-Euram Project BE95-1564

C_{S}	Cache size in words
L_{S}	Line size in words
N_{C}	Number of cache lines ($C_{\mathrm{S}}/L_{\mathrm{S}}$)
N_{nz}	Number of non zero elements of the sparse matrix
M	Number of rows of the sparse matrix
N	Number of columns of the sparse matrix
β	Average of non zeros elements per row (N_{nz}/M)
K	Associativity
Nk	Number of cache sets (N_{C}/K)
p_n	Probability that a position in the sparse matrix contains a non zero element (β/N)
p	Probability that there is at least one entry in a group of L_{S} positions of the sparse matrix $1-(1-p_n)^{L_{\mathrm{S}}}$
r	$\frac{\text{size of an integer}}{\text{size of a real}}$

Table 1: Notation used.

random or a LRU criterium. Our model is oriented to K-way associative caches with LRU replacement. In order to replace a line in a cache of this kind, K different new lines mapped to the same set must be accessed before reusing the line considered. When $K = 1$ the behavior is that of direct mapped caches.

The replacement probability for a given line grows with the number of lines affected by the accesses between two consecutive references to it, and the way these lines are distributed among the different sets. This depends on the memory location of the vectors to be accessed, as it determines the set corresponding to a given line. We handle the areas covered by the accesses to a given program vector `V` as an area vector, $S_{\mathrm{V}} = S_{\mathrm{V}0}S_{\mathrm{V}1}\dots S_{\mathrm{V}K}$, where $S_{\mathrm{V}0}$ is the ratio of sets that have received K or more lines of vector `V`, while $S_{\mathrm{V}_i}, 0 < i \leq K$ is the ratio of sets that have received $K-i$ lines from `V`. This means that S_{V_i} is the ratio of cache sets that require i accesses to new different lines to replace all the lines they contained when the access to `V` started.

Besides K, the number of lines in a set, there is a number of additional parameters our model considers. They are depicted in Table 1. By word we mean the logical access unit, this is to say, the size of a real or an integer. We have chosen the size of a real, but the model is totally scalable.

A uniform distribution in an $M \times N$ sparse matrix with N_{nz} non zero elements (entries) allows us to state the following considerations: the number of entries in a group of L_{S} positions belongs to a binomial $\mathrm{B}(L_{\mathrm{S}}, p_n)$ where p_n is the probability of a given position of the matrix containing an entry, that is, $p_n = \beta/N$. This way, the probability of generating an access to a given line of the vector times which we are multiplying the sparse matrix is p, which is calculated as the table shows.

2.1 Area vectors union

An essential issue is the way we obtain the total area vector. We add the area vectors corresponding to the accesses to the different program vectors. Given two area vectors $S_{\mathrm{U}} = S_{\mathrm{U}0}S_{\mathrm{U}1}\dots S_{\mathrm{U}K}$ and $S_{\mathrm{V}} = S_{\mathrm{V}0}S_{\mathrm{V}1}\dots S_{\mathrm{V}K}$ corresponding to the accesses to vectors `U` and `V`, we define the union area vector $S_{\mathrm{U}} \cup S_{\mathrm{V}}$ as

$$\begin{array}{ll}(S_{\mathrm{U}} \cup S_{\mathrm{V}})_0 = \sum_{j=0}^{K} \left(S_{\mathrm{U}j} \sum_{i=0}^{K-j} S_{\mathrm{V}i}\right) & \\ (S_{\mathrm{U}} \cup S_{\mathrm{V}})_i = \sum_{j=i}^{K} S_{\mathrm{U}j} S_{\mathrm{V}(K+i-j)} & 0 < i \leq K \end{array} \quad (1)$$

where $(S_{\mathrm{U}} \cup S_{\mathrm{V}})_i, 0 < i \leq K$ is the ratio of sets that have received $K-i$ lines from these two vectors, and $(S_{\mathrm{U}} \cup S_{\mathrm{V}})_0$ is the ratio of sets that have received K or more lines. Figure 1 illustrates the area vector union process. From now on the symbol $\cup$ will be used to denote the vector union operation. This method makes no assumptions on the relative positions of the program vectors in memory, as it is based in the addition as independent probabilities of the area ratios.

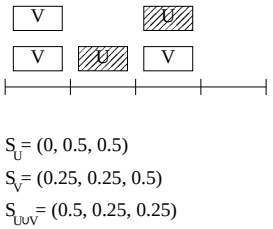

Figure 1: Area vectors corresponding to the accesses to a vector `U` and a vector `V` in a cache with $N_{\mathrm{k}} = 4$ and $K = 2$ and resulting total area vector.

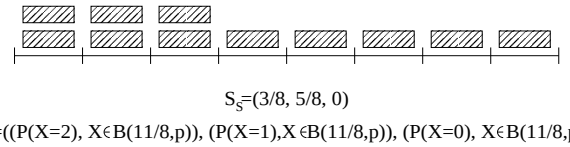

Figure 2: Area vectors corresponding to a sequential access and an access to lines with a uniform reference probability of a vector covering 11 lines in a cache with $N_{\mathrm{k}} = 8$ and $K = 2$.

2.2 Sequential access

The calculation of the area vector depends on the access pattern of the corresponding program vector, so we define an area vector function for each access pattern we find in the codes analyzed. For example, the area vector for the access to n consecutive memory positions is

$$\begin{array}{l} S_{s(K-\lfloor l \rfloor)}(n) = 1 - (l - \lfloor l \rfloor) \\ S_{s(K-\lfloor l \rfloor - 1)}(n) = l - \lfloor l \rfloor \\ S_{si}(n) = 0 \qquad 0 \leq i < K - \lfloor l \rfloor - 1, K - \lfloor l \rfloor < i \leq K \end{array} \quad (2)$$

where $l = (n + L_{\mathrm{S}} - 1)/(L_{\mathrm{S}} N_{\mathrm{k}})$ is the average number of lines that fall into each set. If $l \geq K$ then $S_{s0} = 1, S_{si} = 0, 0 < i \leq K$, as all of the sets receive an average of K or more lines. The term $L_{\mathrm{S}} + 1$ added to n stands for the average extra words brought to the cache in the first and last lines of the access.

2.3 Access to lines with a uniform reference probability

The area vector for an access to an n word vector where each one of the cache lines into which it is divided has the same probability P_l of being accessed may be calculated as

$$\begin{array}{ll} S_{li}(n, P_l) = P(X = K - i) & m < i \leq K \\ S_{lm}(n, P_l) = P(X \geq K - m) & \\ S_{li}(n, P_l) = 0 & 0 \leq i < m \end{array} \quad (3)$$

where $X \in B(n/(L_{\mathrm{S}} N_{\mathrm{k}}), P_l)$, being $\mathrm{B}(n,p)$ the binomial distribution[1] and $m = \max\{0, K - \lceil n/(L_{\mathrm{S}} N_{\mathrm{k}}) \rceil\}$. An example for this access and the sequential access is presented

[1] We define the binomial distribution on a non integer number of elements n as
$P(X = x), X \in B(n,p) = (P(X = x), X \in B(\lfloor n \rfloor, p))(1 - (n - \lfloor n \rfloor)) + (P(X = x), X \in B(\lceil n \rceil, p))(n - \lfloor n \rfloor)$

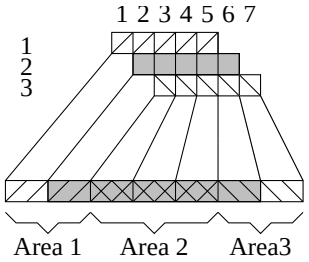

Figure 3: Correspondence between the location of the non zeros in a banded matrix to be transposed and the groups where they are moved to in the vectors that define the output matrix.

in Figure 2. In the case of k accesses of this type, the area vector is $S_l^k(n, P_l) = S_l(n, 1-(1-P_l)^k)$.

2.4 Access to groups of elements with a uniform reference probability

In the transposition of a sparse matrix we find the case of a vector of n words divided into groups of t words where the probability P_g of accessing each group is the same. It happens in the main loop of the algorithm, which accesses the non zeros of the input matrix per rows (sequentially) moving them to the group corresponding to the column they belong to in the vectors that define the output matrix. Each group has as many positions as non zeros has the associated column in the input matrix. Only one of the lines of the group is accessed during the processing of each row of the input matrix, and consecutive accesses to the same group address consecutive memory positions. The area covered by an access of this type is given by

$$S_g(n,t,P_g) = \begin{cases} S_l(n, 1-(1-P_g)^{L_S/t}) & \text{if } t \le L_S \\ S_l(n, L_S/tP_g) & \text{if } t > L_S \end{cases} \quad (4)$$

where if $t \le L_S$, each line of the vector has a uniform probability $1-(1-P_g)^{L_S/t}$ of being accessed. Whereas if $t > L_S$ this probability is L_S/tP_g.

As in the case of S_l, S_g may be extended in order to calculate the area covered by k accesses, $S_g^k(n,t,p)$. Due to space limitations we only show the main expressions of our model; all of them can be found in [4].

2.5 Access to areas displaced with successive references

The model may be extended to consider the case in which the area with some access probability is displaced with successive references. Let $S_{gb}(b,t,P_g)$ be the area vector corresponding to an access to b consecutive groups of t elements with a uniform probability per group P_g of accessing one and only one of the elements of the group. Its value is given by $S_g(b \cdot t, t, P_g)$.

We define $S_{gb}^k(b,t,P_g)$ as the area vector corresponding to k accesses of this type. In the banded sparse matrix transposition case each one of the k accesses corresponds to the processing of a new row of the sparse matrix, as in each row a column leaves the band (to the left), which makes the probability of accessing the group corresponding to it null, and a new one joins the band to the right, thus adding its group to the set of groups that may be accessed during the processing of the row. The situation is depicted in Figure 3 with $b = 5$: during the processing of the first row, groups 1 to 5 may be accessed if there is a non zero in the associated column, while in the second row these sets are 2 to 6, and in the third only sets 3-7 may be accessed.

The accesses following this pattern cover three adjacent areas:

1. A set of $L_v = (\min\{k,b\}-1)t/L_S$ lines with a growing access probability (Area 1 in Figure 3).
2. A set of $L_u = |k-b|t/L_S$ lines with a constant access probability (Area 2 in Figure 3).
3. A last set symmetric to the first one, of the same size, and with a decreasing access probability (Area 3 in Figure 3).

Once the average access probabilities for the lines of this three consecutive areas are known (see [4]), it only remains to combine them to obtain the corresponding area vector.

```
DO I=1, M
  REG = 0
  DO J=R(I), R(I+1)-1
    REG = REG + A(J) * X(C(J))
  ENDDO
  D(I) = REG
ENDDO
```

Figure 4: Sparse matrix-vector product.

2.6 Number of lines in a vector competing for the same cache set

Finally, for the calculation of the self interference probability a function to compute the average number of lines with which a line of the vector competes for the same cache set is needed. This function is defined as

$$C(n) = \begin{cases} 0 & \text{if } v \le 1 \\ \lfloor v \rfloor / v(2v - \lfloor v \rfloor - 1) & \text{if } v > 1 \end{cases} \quad (5)$$

where $v = n/(L_S N_k)$ is the average number of lines of the vector mapped to the same cache set.

3 Modeling the sparse matrix-vector product

The code for this sparse matrix algebra kernel is shown in Figure 4. The format used to store the sparse matrix is the Compressed Row Storage (CRS) [2]: `A` contains the sparse matrix entries, `C` stores the column of each entry, and `R` indicates in which point of `A` and `C` a new row of the sparse matrix starts, which permits knowing the number of entries per row. These three vectors and `D`, the destination vector of the product, present a purely sequential access, thus most of the misses on them are intrinsic. There are no self interferences and very few misses due to cross interferences (specially taking into account that we are considering a K-way associate cache). Therefore the number of misses for these vectors is calculated as the number of cache lines they cover (intrinsic misses).

Nevertheless vector `X` suffers a series of indirect accesses dependent on the location of the entries in the sparse matrix, as it is addressed from the value of `C(J)`. The number of misses accessing this vector is calculated multiplying the

number of lines referenced per dot product by the number of rows of the sparse matrix and the miss probability of one of these accesses. The first value is calculated as $p \cdot N/L_S$, as `X` covers N/L_S lines, and each one has a probability p of being accessed during the dot product times a given row of the sparse matrix, as we have stated in the previous section.

The miss probability is calculated as the opposite of the hit probability. Hits take place whenever the accessed line has been referenced during a previous dot product, and it has suffered no self or cross interferences.

Cross interferences are generated by the accesses to the remaining vectors, which present a sequential access. The cross interference area is given by the total area vector associated with the accesses to vectors `A`, `C`, `R` and `D`. This area vector is calculated in the way explained in Section 2.1, by adding the individual area vectors corresponding to the accesses to each program vector during the period considered. In our case we are interested in calculating the cross interference area vector after i dot products:

$$S_{\text{cross X}}(i) = S_{\text{A}}(i) \cup S_{\text{C}}(i) \cup S_{\text{R}}(i) \cup S_{\text{D}}(i) \qquad (6)$$

where $S_{\text{V}}(i)$ is the area vector covered by the accesses to a vector `V` during i dot products. The four vectors have a sequential access, so expression S_s derived in Section 2.2 is applied:

$$\begin{aligned} S_{\text{R}}(i) &= \quad S_s(i \cdot r) \quad S_{\text{D}}(i) = S_s(i) \\ S_{\text{A}}(i) &= \quad S_s(i \cdot \beta) \quad S_{\text{C}}(i) = S_s(i \cdot \beta \cdot r) \end{aligned} \qquad (7)$$

These values are obtained considering that one element of `D` and `R`, and β components of `A` and `C` are accessed per dot product. A scale factor r is applied to integer vectors in order to take into account that integer data are often stored using less bytes than real ones. This factor is the quotient of the number of bytes required by a real datum and the one required by an integer.

As for the self interference probability, each line of `X` competes on average with $C(N)$ lines of the same vector (see definition of $C(n)$ in (5)) in the same cache set, and they all have the same probability p of being accessed during a dot product times a row of the sparse matrix. As a result, the number of different candidate lines of `X` to replace a given line that have been accessed after i dot products belongs to a binomial $\text{B}(C(N), 1-(1-p)^i)$.

The hit probability in the first access to any line of `X` during the dot product of the j-'th row of the sparse matrix times vector `X` is:

$$P_{\text{hit X}}(j) = \sum_{i=1}^{j-1} p(1-p)^{i-1}(1 - S_{\text{int X}_0}(i)) \qquad (8)$$

where $p(1-p)^{i-1}$ is the probability that the last access to the line has taken place i dot products ago and $S_{\text{int X}}(i)$ is the interference area vector generated by the accesses produced during these i dot products, which is calculated as

$$S_{\text{int X}}(i) = S_l(C(N)\frac{C_S}{K}, 1-(1-p)^i) \cup S_{\text{cross X}}(i) \qquad (9)$$

where the cross interfence area vector obtained in (6) is added to the self interference area vector, which is given by an access with a uniform access probability per line of $1-(1-p)^i$ to the lines of vector `X` that can generate self interferences with another line, which are calculated using (5). Finally, the average hit probability is calculated as:

$$P_{\text{hit X}} = \frac{\sum_{j=1}^{M} P_{\text{hit X}}(j)}{M} \qquad (10)$$

```
1  DO I=2,N+1                    4  DO I=1, M
     RT(I)=0                        DO K=R(I), R(I+1)-1
   END DO                             J=C(K)
2  DO I=1, R(M+1)-1                   P=RT(J)
     J=C(I)+2                         CT(P)=I
     RT(J)=RT(J)+1                    AT(P)=A(K)
   END DO                             RT(J)=P+1
   RT(1)=1                          END DO
   RT(2)=1                        END DO
3  DO I=3, N+1
     RT(I)=RT(I)+RT(I-1)
   END DO
          ↪
```

Figure 5: Transposition of a sparse matrix.

We must point out that the model only takes into account the first access to each line of `X` in each dot product. The other $N_{\text{NZ}} - p \cdot M \cdot (N/L_S)$ accesses have a very low probability of resulting in a miss, as they refer to lines that have been accessed in the previous iteration of the inner loop.

4 Modeling the transposition of a sparse matrix

In this section the model is extended to a operation with a greater degree of complexity, the sparse matrix transposition. As in the previous section, we assume that the sparse matrix and its transposed are stored in a CRS format. Figure 5 shows the transposition algorithm described in [8], where both matrices are represented by vectors(`A`, `C`, `R`) and (`AT`, `CT`, `RT`), respectively. Observe that in loop 4 there are multiple indirection levels both in the left and right side of the sentences.

In what follows we employ a similar approximation to the one developed in the previous section to estimate the number of misses for vectors `AT`, `CT` and `RT`. The remaining vectors have sequential accesses, which have already been considered in the previous algorithm.

4.1 Vectors `AT` and `CT`

These two vectors follow exactly the same access pattern, as may be observed in loop 4 of Figure 5. We will thus explain the estimation of the number of misses for `AT`, as the one for `CT` is identical but taking into account that it is an integer vector. The access pattern is the one modeled by S_g, described in Section 2.4, where $n = N_{\text{NZ}}$, $t = N_{\text{NZ}}/N$ and $P_g = p_n$ for `AT`.

This pattern has similarities with the one explained in Section 3 for vector `X`, as the access probability is uniformly distributed along the vector, being the difference that for vector `X` the probability is constant for each line of the vector, while for vectors `AT` and `CT` it corresponds to sets of as many elements as a column of the original sparse matrix holds. As a result, the general form of the hit probability during the process of the j-'th row of the sparse matrix is very similar to the one in (8), with the following differences:

- The probability of accessing the considered line of the vector during the processing of a row is not p but $1-(1-p_n)^{\max\{1, L_S/t\}}$.
- The probability of accessing another line mapped to the same cache set during the processing of the i previous rows is not $1-(1-p)^i$ but $1-(S^i_{gK}(n,t,P_g))^{L_S N_k / N_{\text{NZ}}}$.

- Vector `AT` has N_{nz} elements, so the number of lines that compete in the set with the line considered is $C(N_{\mathrm{nz}})$ in the equation that calculates the interference area vector.
- When calculating the cross interference probability, the same scheme of adding the area vectors corresponding to the access to the remaining vectors is used. $S_{\mathrm{R}}(i)$, $S_{\mathrm{C}}(i)$ and $S_{\mathrm{A}}(i)$ are calculated according to (7) and the remaining area vectors are

$$\begin{aligned} S_{\mathrm{RT}}(i) &= S_l^i(N \cdot r, p_r) \\ S_{\mathrm{CT}}(i) &= S_g^i(N_{\mathrm{nz}} \cdot r, (N_{\mathrm{nz}} \cdot r/N), p_n) \end{aligned} \quad (11)$$

4.2 Vector `RT`

This vector is referenced in the four loops of the algorithm. In the first loop it has a totally sequential access that produces $(N \cdot r)/L_{\mathrm{S}}$ intrinsic misses.

In the second loop, the accesses to `RT` follow a similar pattern to those of vector `X` in the sparse matrix-vector product. The only differences consist in that there is only one possible source of cross interferences (vector `C`), and that `RT` is an integer vector and not a real value vector. The number of misses in loop 2 is

$$F_{\mathrm{RT_2}} = p_r \cdot M \frac{N \cdot r}{L_{\mathrm{S}}}(1 - (P_{\text{hit RT}_2} + P_{\text{first\_hit RT}_2})) \quad (12)$$

where $P_{\text{hit RT}_2}$ is calculated following expressions (8) and (10) introducing the modifications mentioned above. Vector `RT` has been completely accessed in a sequential manner in the previous loop. For this reason we must add the probability $P_{\text{first\_hit RT}_2}$ of getting a hit due to the existence of portions of `RT` in the cache when initiating the loop. Again, a final expression can be found in [4].

In loop 3, vector `RT` is sequentially accessed without any accesses to other vectors. The estimated number of misses is

$$F_{\mathrm{RT_3}} = Nr/L_{\mathrm{S}}(1 - P_{\text{first\_hit RT}_3}) \quad (13)$$

where $P_{\text{first\_hit RT}_3}$ may be estimated as $P_{\text{first\_hit RT}_2} \cdot p_r \cdot M$.

Finally, in loop 4 the access to `RT` is again similar to the sparse matrix-vector product described in Section 3. In this loop we must also consider a hit probability $P_{\text{first\_hit RT}_4}$ due to a previous load of `RT` generated by loop 3.

5 Extension to banded matrices

A large number of real numerical problems in engineering lead to matrices with a sparse banded distribution of the entries [3]. In this section we present the modifications the model requires in order to describe the cache behavior for matrices of this type. The model parameter p_n (see Table 1) is calculated as $\frac{\beta}{W}$, where W is the bandwidth. Consequently p takes the value $p = 1 - (1 - \beta/W)^{L_{\mathrm{S}}}$.

5.1 Sparse matrix-vector product

The number of misses over vector `X` is the only one affected by the band distribution of the entries. The modeling of the behavior of this vector is identical to the one described in Section 3. The number of accesses to different lines in the M dot products in which vector `X` is involved, $p \cdot M \cdot W/L_{\mathrm{S}}$, is multiplied by the miss probability calculated as $1 - P_{\text{hit X}}$. For the calculation of $P_{\text{hit X}}$ in expression (10) M must be replaced by W, as one line of `X` may be accessed during the product of a maximum of W rows. In addition, the number of lines of `X` that compete with another in a cache set is $C(W)$ instead of $C(N)$, which influences $P_{\text{hit X}}(j)$ in expression (8), as the entries of each row are distributed among W positions instead of N.

5.2 Transposition of a sparse matrix

The calculation of the number of misses on `AT` is modified in a similar way to that of vector `X` in the sparse matrix-vector product: each line of this vector spreads its access probability through the processing of W rows of the sparse matrix, instead of M, thus reducing to this limit the sum that calculates the hit probability. For the same reason, the number of lines with which a given line of this vector competes for the same cache set during its access period is not $C(N_{\mathrm{nz}})$, but $C(N_{\mathrm{nz}}/N \cdot W)$.

Besides, `AT` is accessed following the pattern described by area vector funcion S_{gb} in Section 2.5, so the probability of accessing during the processing of the i previous rows a given line that is mapped to the same cache set as the line we are considering is $1-(S^i_{gbk}(W,t,P_g))^{L_{\mathrm{S}}N_{\mathrm{k}}/(N_{\mathrm{nz}}/N\cdot(W+i))}$. Also the cross interference probability calculation needs to be modified according to the new access patterns.

Similar changes are needed for the estimation of the number of misses on vector `CT`, taking into account that it is made up of integer values.

The prediction of the number of misses on vector `RT` in loops 2 and 4 is based on the model for vector `X` in the sparse matrix-vector product, so the calculation of $P_{\text{hit RT2}}$ and $P_{\text{hit RT4}}$ requires the modifications explained in the previous section, and the number of different lines accesses in each dot product is no longer $N \cdot r/L_{\mathrm{S}}$ but $W \cdot r/L_{\mathrm{S}}$. Some changes are needed also to calculate the probabilities of hit due to a reuse $P_{\text{first\_hit RT}_2}$ and $P_{\text{first\_hit RT}_4}$. Finally, the value of $P_{\text{first\_hit RT3}}$ is now $P_{\text{first\_hit RT2}} \cdot p_r \cdot W$.

6 Results

The model was validated with simulations on synthetic matrices with a uniform distribution of the non zero elements and banded matrices from the Harwell-Boeing collection [3]. Traces for the simulations were obtained by running the algorithms after replacing the original accesses by calls to functions that calculate the position to be accessed and write it to disk. These traces were fed to the dineroIII cache simulator, belonging to the WARTS tools [6].

Tables 2 and 3 show the model accuracy for the sparse matrix-vector product for some combinations of the input parameters for uniform and banded sparse matrices respectively. Tables 4 and 5 display the results for the sparse matrix transposition model. Without any loss of generality we have considered square matrices ($N = M$) in the analysis, and $r = 1$. C_{S} is expressed in Kwords and L_{S} in words.

The maximum error obtained in the trial set was 5.15% for the synthetic matrices and 28.12% for the Harwell-Boeing matrices. The reason for this last result is that the entries distribution in the real matrices is not completely uniform, which produces high deviations for the sparse matrix transposition model (see Table 5). Nevertheless, we consider that such amount of deviation is still acceptable for our analysis purposes. Besides the small size of the matrices of the collection does not favor the convergence of our probabilistic model. We also want to point out that the average error

N	N_{nz}	α	C_s	L_s	K	predicted misses	measured misses	Dev.
1	10	1.00%	2	4	2	6.952	6.948	0.06%
1	10	1.00%	4	4	4	5.779	5.777	0.02%
1	10	1.00%	8	8	2	2.903	2.908	-0.15%
1	10	1.00%	16	8	4	2.875	2.876	-0.02%
10	100	0.10%	8	8	2	69.978	70.199	-0.31%
10	100	0.10%	16	8	4	37.134	36.986	0.39%
10	100	0.10%	32	8	2	30.239	30.440	-0.66%
10	100	0.10%	64	8	4	28.751	28.751	0%

Table 2: Predicted and measured misses and deviation of model for sparse matrix-vector product with a uniform entries distribution. M, N_{nz} and numbers of misses in thousands.

Name	N	W	N_{nz}	α	C_s	L_s	K	predicted misses	measured misses	Dev.
JPWH_991	991	155	6027	3.92%	2	4	2	3788	3826	-0.98%
JPWH_991	991	155	6027	3.92%	2	4	4	3758	3758	0.01%
JPWH_991	991	155	6027	3.92%	4	4	2	3766	3774	-0.22%
JPWH_991	991	155	6027	3.92%	4	4	4	3758	3758	0.02%
BCSSTK05	153	20	2423	79.18%	2	4	2	1327	1330	-0.24%
BCSSTK05	153	20	2423	79.18%	2	4	4	1326	1329	-0.16%
BCSSTK05	153	20	2423	79.18%	4	4	2	1326	1329	-0.19%
BCSSTK05	153	20	2423	79.18%	4	4	4	1326	1329	-0.16%
BCSSTM10	1086	71	22092	28.65%	2	4	2	11893	11934	-0.34%
BCSSTM10	1086	71	22092	28.65%	2	4	4	11864	11862	0.02%
BCSSTM10	1086	71	22092	28.65%	4	4	2	11872	11879	-0.06%
BCSSTM10	1086	71	22092	28.65%	4	4	4	11864	11862	0.02%

Table 3: Predicted and measured misses and deviation of model for sparse matrix-vector product of some Harwell-Boeing matrices.

N	N_{nz}	α	C_s	L_s	K	predicted misses	measured misses	Dev.
1	10	1.00%	2	4	2	30.896	30.917	-0.06%
1	10	1.00%	4	4	4	25.879	25.939	-0.23%
1	10	1.00%	8	4	2	21.137	21.565	-1.98%
1	10	1.00%	8	4	4	20.271	20.619	-1.68%
10	100	0.10%	2	8	4	415.990	416.202	-0.05%
10	100	0.10%	4	8	2	390.848	391.128	-0.07%
10	100	0.10%	32	8	4	245.369	243.793	0.64%
10	100	0.10%	64	8	4	197.390	194.595	1.43%

Table 4: Predicted and measured misses and deviation of model for the transposition of a sparse matrix with a uniform entries distribution. M, N_{nz} and numbers of misses in thousands.

Name	N	W	N_{nz}	α	C_s	L_s	K	predicted misses	measured misses	Dev.
JPWH_991	991	155	6027	3.92%	2	4	2	10896	13025	-16.35%
JPWH_991	991	155	6027	3.92%	2	4	4	10535	12797	-17.67%
JPWH_991	991	155	6027	3.92%	4	4	2	9257	9925	-6.73%
JPWH_991	991	155	6027	3.92%	4	4	4	8607	9308	-7.54%
BCSSTK05	153	20	2423	79.18%	2	4	2	3191	3341	-4.50%
BCSSTK05	153	20	2423	79.18%	2	4	4	3158	3272	-3.48%
BCSSTK05	153	20	2423	79.18%	4	4	2	2941	2978	-1.26%
BCSSTK05	153	20	2423	79.18%	4	4	4	2980	2972	0.24%
BCSSTM10	1086	71	22092	28.65%	2	4	2	31903	33457	-4.65%
BCSSTM10	1086	71	22092	28.65%	2	4	4	29965	32250	-7.09%
BCSSTM10	1086	71	22092	28.65%	4	4	2	29492	30537	-3.42%
BCSSTM10	1086	71	22092	28.65%	4	4	4	28941	30357	-4.67%

Table 5: Predicted and measured misses and deviation of model for the transposition of a sparse matrix of some Harwell-Boeing matrices.

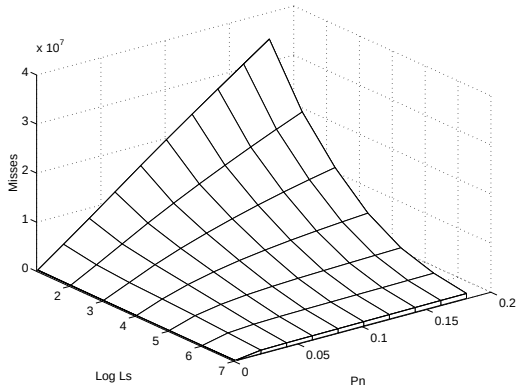

Figure 6: Number of misses in a 4-way associative cache with 2Kw during the sparse matrix-vector product of a $10K \times 10K$ matrix as a function of L_S and p_n.

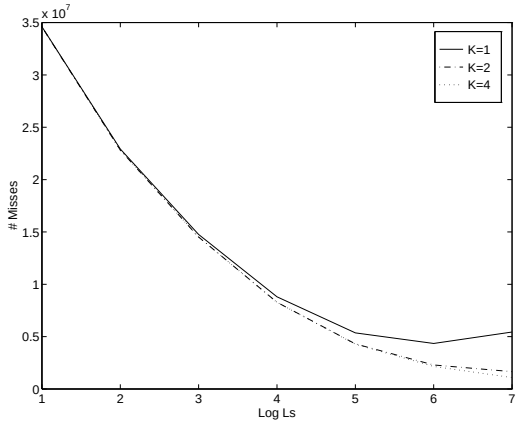

Figure 7: Number of misses during the sparse matrix-vector product of a $10K \times 10K$ matrix with $p_n = 0.18$ as a function of the line size of a 2Kw cache for different associativities.

for the synthetic matrices was 0.65%, and 5% for the real matrices.

In Figures 6-13 we present the relationship between the number of misses and the different parameters introduced in the models. In the case of C_S and L_S we display the base 2 logarithm of the number of cache Kwords and line words respectively.

Figure 6 shows the relationship of the number of misses with L_S and p_n in the sparse matrix-vector product. The evolution is approximately linear with respect to p_n, as the number of accesses is directly proportional to N_{NZ} and most of them follow a sequential pattern. It may be observed how the number of misses significantly decreases as L_S increases because the accesses to all of the vectors except `X` are sequential (see Figure 4) and the larger the lines, the more exploitation of the spatial locality we obtain. Nevertheless, when L_S is very large (> 64 words) and the matrix has a lower degree of sparsity ($p_n > 0.1$) the increase of the self and cross interferences probabilities over vector `X` begins to unbalance the advantages obtained from a more efficient use of the spatial locality exhibited by the remaining vectors for $K = 1$. This effect, shown in Figure 7, increases with the value of p_n.

The relationship between W and C_S for the banded sparse matrix-vector product is shown in Figure 8. In this graph

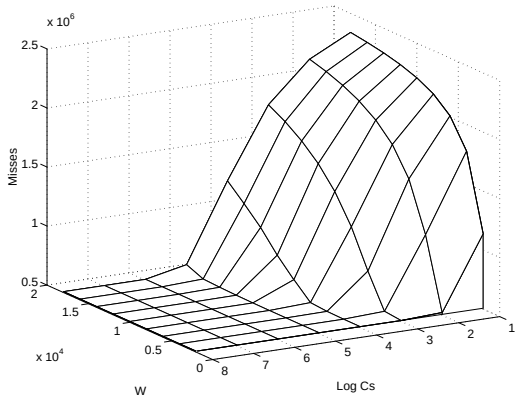

Figure 8: Number of misses during the sparse matrix-vector product for a sparse matrix 20K×20K with $N_{NZ} = 2M$ entries, $L_S = 8$ and $K = 4$ as a function of W and C_S.

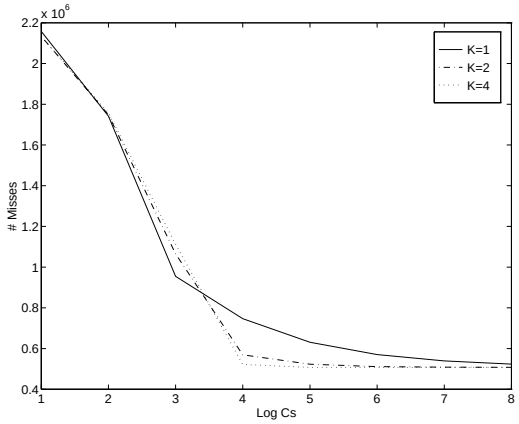

Figure 9: Number of misses during the sparse matrix-vector product of a $20K \times 20K$ matrix with 2M entries and $W = 8000$ as a function of the cache size with $L_S = 8$ for different associativities.

we have considered broad bands in order to illustrate the effect of self interferences in the access to vector `X`. The bandwidth reduction has a large influence on the number of misses because it reduces the self interference probability and increases the reuse probability for the lines of vector `X`, as the non zeros are spread on narrower rows (spatial locality improvement) and columns (temporal locality improvement). A number of misses near the minimum is reached when $C_S \geq 1.5W$, being the increases of C_S beyond that size of little use. It is intuitive that good miss rates can only be reached with $C_S > W$, as with this size there is a line in the cache for each line in the band of the currently processed row. The extra room is needed to avoid the combined effect of self and cross interferences, as we shall now demostrate through an analysis for a fixed bandwidth for different degrees of associativity and cache sizes.

In Figure 9 the number of misses for a matrix with $W = 8000$ is displayed in relation to the cache size for different associativities. We can observe that for $K = 1$ most of the improvement is reached for the 8Kw cache, due to the elimination of the self interferences. The cache size increments from that size on help to gradually reduce the cross interferences. On the other hand, caches with $K > 1$ have

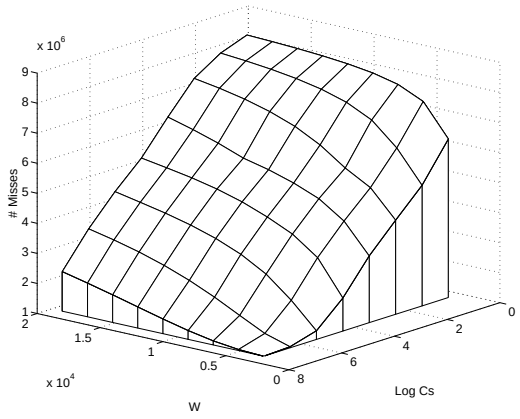

Figure 10: Number of misses during the transposition of a sparse matrix 20K×20K with $N_{\mathrm{nz}} = 2M$ entries, $L_{\mathrm{S}} = 8$ and $K = 4$ as a function of W and C_{S}.

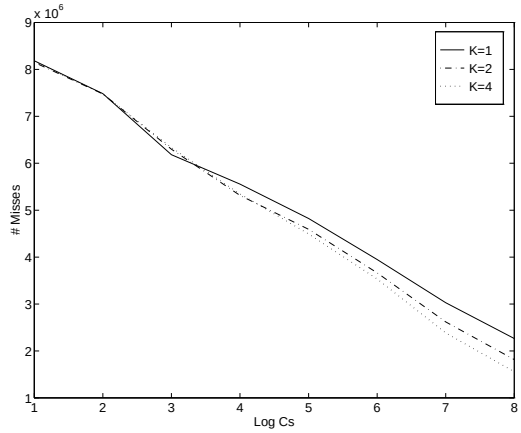

Figure 11: Number of misses during the transposition of a sparse matrix 20K×20K matrix with 2M entries and $W = 8000$ as a function of the cache size with $L_{\mathrm{S}} = 8$ for different associativities.

a different behavior: the misses reduction gradient is very high while $C_{\mathrm{S}} \leq$16Kw. The reason is that in the 8Kw cache there are K different lines of `X` mapped to each cache set. As the lines are usually accessed in the same order and the cache uses a LRU replacement, any cross interference may generate misses for all of the lines of `X` that are mapped to the same cache set. The result is that the cross interferences affect as many lines as a set can hold, thus generating more misses. In fact we can see that the 4-way associate cache performs a little worse than the 2-way cache for this cache size. For caches with $C_{\mathrm{S}} \geq$ 16Kw, the cache sets have enough lines left to absorb the accesses to the vectors that comprise the sparse matrix and the destination vector without increasing the interferences on `X` due to their combination with the lines of this vector that reside in the same cache set. For small cache sizes all the associativities perform similarly due to the large number of interferences. The conclusion is that associative caches help reducing the interference effect when the number of lines that compete for a given cache line is smaller than or equal to K; otherwise the performance is quite similar to that of a direct mapped cache. Moreover, in this case, if the lines mapped to the same cache set are usually accessed in the same order, high associativities may perform worse.

As for the sparse matrix transposition, Figures 10 and 11 represent the same data for this algorithm as Figures 8 and 9 for the sparse matrix-vector product respectively. The first one shows a decrease of the number of misses with the bandwidth reduction, being this more noticeable in the point where C_{S} becomes greater than W. The reasons are those explained for the previous algorithm. Anyway, this reduction is much softer than in the sparse matrix-vector product because the accesses to vector `RT` in loops 2 and 4, which are the most directly favoured by the bandwidth reduction, stand only for a small portion of the misses. On the other hand, the number of misses on vectors `AT`, and `CT`, which account for the vast majority of the misses, decreases slowly when C_{S} increases or W decreases, although they are also heavily dependent on the bandwidth. The reason is that in all of the cases shown in the figure the data belonging to these vectors during the process of all of the columns inside the band of the original matrix do not fit in the cache ($(N_{\mathrm{nz}}/N) \cdot W$ entries). Only for the case when $W = 2000$ and $C_{\mathrm{S}} = 256K$ does this set fit, and we can see that the number of misses becomes stable in this area of the graph. The misses on `C`, `R` and `A` remain almost completely constant due to their sequential access, obtaining as only benefit from the bandwidth reduction a somewhat lower cross interference probability.

Figure 11 shows that the general behavior of the algorithm with respect to K, the associativity degree, although having similarities with that of the sparse matrix-vector product, does not depend only on W to determine the cache sizes for which the hit rate obtains reasonable values. As explained before, the reason is that for $W = 8000$ none of the cache sizes considered contain a considerable portion of the data the algorithm accesses during the W iterations that process a whole band in loop 4, the one that causes most of the misses. Only accesses to `RT`, mainly in loop 2, benefit from the increase of C_{S} to values larger than W. This is specially noticeable for $K = 1$, as in the sparse matrix-vector product. Another similarity is the worse behavior of caches with $K > 1$ for a cache size very close to W due to the added effect of cross interferences with self interferences because of cache lines which are usually accessed in the same order. This penalizes the LRU replacement algorithm. Once the cache size is noticeable larger than the bandwidth, higher associativities perform better, as in Figure 9.

In order to get hints about the values of the cache parameters for which the algorithm stabilizes its misses, Figures 12 and 13 show the same data for a smaller matrix using $L_{\mathrm{S}} = 4$. During the process of a band of this matrix the working set for vectors `AT` and `CT`, those that account for most of the misses, comprise $25W$ elements, as there is an average of 25 elements per column. The hit rate obtains values near to its maximum in Figure 12 when the cache size exceeds this value. Increases of C_{S} beyond that limit provide little performance improvement. These improvements are only noticeable when the cache size increase is very high, due to the strong reduction of the cross interference. We must take into account that this graph is constructed for a 4-way associative cache. Somewhat greater cache sizes would be needed to obtain good cross interference reduction in a direct mapped cache (see Figure 13).

Set associative caches help reducing the miss rate for the small cache sizes in Figure 13 because on average there are always less than 2 lines competing in any set, as during the process of each row there are only about 200 lines of vector

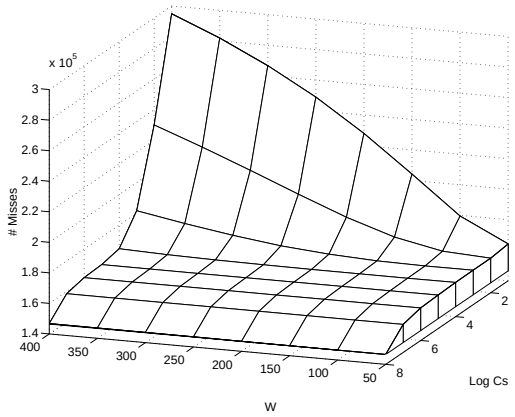

Figure 12: Number of misses during the transposition of a sparse matrix 5K×5K with 125K entries, $L_S = 4$ and $K = 4$ as a function of W and C_S.

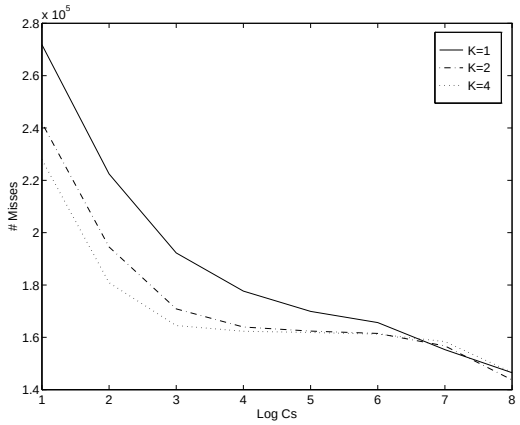

Figure 13: Number of misses during the transposition of a sparse matrix 5K×5K matrix with 125K entries and $W = 200$ as a function of the cache size with $L_S = 4$ for different associativities.

`AT` with a non null access probability, 200 lines belonging to vector `CT` and a few more lines belonging to the other vectors, while the smallest of the caches considered has 512 lines. As expected, the increase of the cache size reduces the hit ratio difference between direct mapped caches and set associate caches. For $C_S \geq 8$Kw the cache size increase helps only by reducing the cross interferences.

Finally, the relationship of the line size and the matrix density with the number of misses for this last algorithm has proved to be the same as in the sparse matrix-vector product, being the only difference that the gradient of the increase of misses with relation to p_n is about three times larger, which is normal, as the number of accesses per non zero in the original matrix is eight, while in the sparse matrix-vector product algorithm it is three.

7 Conclusions and future work

We have presented a fully-parametrizable model for the estimation of the number of misses on a generic K-way associative cache with LRU replacement and we have applied it to the sparse matrix-vector product and the transposition of a sparse matrix. The model deals with all the possible types of misses and has demonstrated a high level of accuracy in its predictions. It considers a uniform distribution of the entries on the whole matrix or on a given band.

N	N_{nz}	α	W	C_s	L_s	K	dineroIII time	model time
100	1000	1%	1000	16	4	2	67.13	0.23
100	1000	10%	100	16	4	2	63.39	0.07
100	1000	1%	1000	16	2	2	76.76	0.23
100	1000	10%	100	16	2	2	73.08	0.08
100	1000	1%	1000	16	4	4	69.10	0.29
100	1000	10%	100	16	4	4	64.19	0.13
1	22	28%	71	2	4	4	1.26	0.01
1	22	28%	71	2	2	2	1.38	0.01

Table 6: Simulation and model user times to calculate the number of misses during the transposition of a banded sparse matrix on a 200 MHz Pentium. N and N_{nz} in thousands, time in seconds.

As Table 6 shows, besides providing more information, the modelization is much faster than the simulation, even when the time required to generate the trace, which takes almost as much time as the execution of the simulator itself, is not included in the table.

We have illustrated how the model may be used to study the cache behavior for this code, and shown the importance of the bandwidth reduction in the case of the banded matrices, even for high degrees of associativity. The model can be also applied to study possible architectural cache parameter modifications in order to improve cache performance.

Future work includes the extension of the model to consider prefetching and subblock placement. On the other hand, we are now working on the modeling for non uniform distributions of the entries in the sparse matrices focusing in the most common patterns that appear in real matrices suites. Finally, in order to obtain more accurate estimations, we are studying the inclusion of the data structures base addresses as a parameter of the model.

References

[1] A. Agarwal, "Analysis of Cache Performance for Operating Systems and Multiprogramming," PhD thesis, University of Stanford, Department of Electrical Engineering, May 1987.

[2] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM Press, 1994.

[3] I.S. Duff, R.G. Grimes and J.G. Lewis, "User's Guide for the Harwell-Boeing Sparse Matrix Collection," CERFACS TR-PA-92-96, Oct. 1992.

[4] B.B. Fraguela, "Cache Miss Prediction in Sparse Matrix Computations," Technical Report UDC-DES-1997/1. Departamento de Electrónica e Sistemas da Univerdade da Coruña, April 1997.

[5] S. Ghosh, M. Martonosi and S. Malik, "Cache Miss Equations: An Analytical Representation of Cache Misses," *Proc. ACM Int'l. Conf. on Supercomputing* (ICS'97), pp. 317-324, July 1997.

[6] A.R. Lebeck and D.A. Wood, "Cache Profiling and the SPEC Benchmarks: A Case Study," *IEEE Computer*, vol. 27, no. 10, pp. 15-26, Oct. 1994.

[7] J.J. Navarro, E. García, J.L. Larriba-Pey and T. Juan, "Block Algorithms for Sparse Matrix Computations on High Performance Workstations," *Proc. ACM Int'l. Conf. on Supercomputing* (ICS'96), pp. 301-309, May 1996.

[8] S. Pissanetzky, *Sparse Matrix Technology.* Academic Press, 1984.

[9] V.E. Taylor, "Sparse Matrix Computations: Implications for Cache Designs," *Proc. IEEE Int'l Conf. on Supercomputing*, pp. 598-607, Nov. 1992.

[10] O. Temam and W. Jalby, "Characterizing the Behaviour of Sparse Algorithms on Caches," *Proc. IEEE Int'l. Conf. on Supercomputing* (ICS'92), pp. 578-587, Nov. 1992.

[11] O. Temam, C. Fricker and W. Jalby, "Cache Interference Phenomena," *Proc. ACM SIGMETRICS*, pp. 261-271, May 1994.

[12] R.A. Uhlig and T.N. Mudge, "Trace-Driven Memory Simulation: A Survey," *ACM Computing Surveys*, 29(2), pp. 128-170, June 1997.