Direct Mapped Cache Performance Modeling for Sparse Matrix Operations[*]

Ramón Doallo, Basilio B. Fraguela
Dept. de Electrónica e Sistemas
Universidade da Coruña
15071 A Coruña (SPAIN)
{doallo,basilio}@udc.es

Emilio L. Zapata
Dept. de Arquitectura de Computadores
Universidad de Málaga
29080 Málaga (SPAIN)
ezapata@ac.uma.es

Abstract

Sparse matrices are in the kernel of numerical applications. Their compressed storage, which permits both operations and memory savings, generates irregular access patterns, reducing the performance of the memory hierarchy. In this work we present a probabilistic model for the prediction of the number of misses of a direct mapped cache memory, considering sparse matrices with a uniform entries distribution. The number of misses is directly related to the program execution time and the memory hierarchy performace. The model considers the three types of standard interferences: intrinsic, self and cross interferences. We explain in detail the modeling of a representative matrix operation such as the sparse matrix-dense matrix product, considering several loop orderings, and include validation results that show the model accuracy.

1 Introduction

There are many numerical applications that lead to sparse matrices. In order to save memory and operations, they are stored using vector based compressed formats [1]. This storage leads to irregular memory access patterns due to the use of indirect addressing, making the efficient exploitation of the spatial and temporal locality in the memory hierarchy more difficult, besides making difficult its performance prediction.

Cache miss rates can be derived by trace-driven simulations [10], which although being very accurate lack flexibility and consume many computing resources. Another approach is that of analytical models that extract their input parameters from address traces [2], [6], thus requiring the program execution or simulation each time any of its parameters changes. The same problem arises when we use the hardware built-in counters some microprocessor architectures implement. In this last case there is the additional limitation that these devices are obviously limited to the study of the architectures where they exist. As for the analytical models based on the code, they have been traditionally devoted to dense codes [5], [8], as the regularity their access patterns exhibit favours their study. Purely analytical modeling has many advantages, being the most relevant flexibility and speed, and one major drawback: limited precision. Little effort has been devoted in this area to irregular access patterns like those we model here. As a previous work, we can mention [9], where a partial modeling of the behavior of one vector in the sparse matrix-vector product on a direct-mapped cache is developed. We mean by partial modeling that they only consider intrinsic misses and self interferences, but not cross interferences.

In this work we present a probabilistic model that predicts the number of misses produced in a direct-mapped cache during the execution of algorithms based on sparse matrices. The model includes all the possible types of interferences.We have applied it to a number of algebra kernels such as sparse matrix-vector product, sparse matrix transposition and sparse matrix-dense matrix product [3], but due to space limitations only the latter will be here described.

2 Probabilistic model

Our model considers a C_S words direct mapped cache with L_S words per line (and $N_C = C_S/L_S$ lines). Three kinds of misses may take place: intrinsic, cross and self interferences. The first one arise when accessing a memory block for the first time, while the other ones are due to line replacements by lines either from other data structures (cross interferences) or from the same data structure the replaced line belongs to (self interferences).

[*]This work was supported by the Comisión Interministerial de Ciencia y Tecnología (CICYT) under project TIC96-1125, Xunta de Galicia under Project XUGA20605B96

In this way, the first access to a memory line results always in a miss. The remaining references have a miss probability directly proportional to the ratio of the cache area affected by the memory accesses since the last time the considered line has been referenced. Our model uses this ratio as the miss probability in the accesses not being the first one, so it is based on the calculation and addition of the cache area ratios affected by the accesses to the different data structures between consecutive references to a given line. As the array base addresses of the arrays are unknown to the model, it adds the area ratios associated to different arrays as independent probabilities. That means that given two arrays X and Y whose accesses affect a ratio S_{X} and S_{Y} respectively, the portion of the cache affected by accesses to any of these vectors is calculated as $S_{\mathsf{X}} \cup S_{\mathsf{Y}} = S_{\mathsf{X}} + S_{\mathsf{Y}} - S_{\mathsf{X}} \cdot S_{\mathsf{Y}}$. From now on we will employ the symbol $\cup$ in order to denote this operation.

For the calculation of the area ratios covered by the accesses to each array, we use a series of functions or algorithms that will depend on the access pattern it undergoes. The simplest one is the access to n consecutive positions of a vector, that is

$$S_s(n) = \min\{1, (n + L_{\mathsf{S}} - 1)/C_{\mathsf{S}}\} \tag{1}$$

where $L_{\mathsf{S}} - 1$ is added to n to stand for the extra words that are brought to cache in the first and last lines of the access. Another access to be considered is the corresponding to an n word vector in which each one of the cache lines into which it is mapped has the same probability P_l of being accessed. Its cache ratio may be calculated as

$$S_l(n, P_l) = 1 + \left(\left(\frac{n}{C_{\mathsf{S}}} - \left\lfloor\frac{n}{C_{\mathsf{S}}}\right\rfloor\right) P_l - 1\right)(1 - P_l)^{\left\lfloor\frac{n}{C_{\mathsf{S}}}\right\rfloor} \tag{2}$$

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 line is needed. This function is defined as

$$C(n) = \begin{cases} 0 & \text{if } v \leq 1 \\ \lfloor v\rfloor/v(2v - \lfloor v\rfloor - 1) & \text{if } v > 1 \end{cases} \tag{3}$$

where $v = n/C_{\mathsf{S}}$ is the average number of lines of the vector mapped to the same cache line.

In this work we shall use as memory unit the word, which is chosen to be the size of a floating point number. Integers will be considered through the use of a scaling constant $r =$ integer size/floating point size.

3 Sparse Matrix-Dense Matrix Product Modeling

Without any loss of generality, we use the Compressed Row Storage (CRS) format [1] in order to store the sparse matrix. In this format one vector stores the sparse matrix entries, A, other one stores the column of each entry, C and one last, R, indicates in which point of A and C a new row of the sparse matrix starts. The dense matrix involved in the product is B, and D is the product matrix.

C_{S}	Cache size in words
L_{S}	Line size in words
N_{C}	Number of lines in the cache $(C_{\mathsf{S}}/L_{\mathsf{S}})$
r	integer size/floating point size
M	Number of rows of the sparse matrix
N	Number of columns of the sparse matrix
N_{nz}	Number of entries of the sparse matrix
β	Average number of entries per row (N_{nz}/M)
α	Matrix density $(N_{\mathsf{nz}}/(M \cdot N))$
p	Probability that there is at least one entry in L_{S} positions of the sparse matrix $(1-(1-\alpha)^{L_{\mathsf{S}}})$
H	Number of columns of the dense matrix

Table 1. Notation used.

There are three basic versions for the sparse matrix-dense matrix product code depending on the order in which the loops are nested: JIK, IJK and IKJ, where the first letter corresponds to the outermost loop and the last letter to the innermost one. The elements of the sparse matrix, the dense matrix B and the product matrix D are referenced by (I,K), (K,J) and (I,J), respectively. We shall use an $M \times N$ sparse matrix with N_{nz} entries uniformly distributed and a dense matrix with H columns. The uniform distribution we consider in this work allows to state that $p = 1-(1-\alpha)^{L_{\mathsf{S}}}$, where $\alpha = N_{\mathsf{nz}}/(M \cdot N)$ is the matrix density, is the probability that there is at least one entry in a group of L_{S} matrix positions. As a result of this uniform distribution, we shall also consider that there are on average $\beta = N_{\mathsf{nz}}/M$ entries per row. Table 1 summarizes the parameters describing the cache and the matrix.

3.1 JIK ordering

The accesses to the three vectors that comprise the sparse matrix and one column of the product matrix are sequential in each iteration of loop J (see Figure 1), thus most of the misses are intrinsic during the product by the first column of matrix B. There are no self interferences and very few misses due to cross interferences. Therefore we can calculate the number of misses for these vectors as the number of cache lines they occupy (intrinsic misses) plus a miss probability, in every access which is not the first to a line, equal to the union of the areas covered by the accesses to the other vectors since the last access to the considered line. For example, for vector A, between two consecutive accesses to a line there is an access to vector C and another to matrix B,

```
DO J=1, H
  DO I=1, M
    REG=0.0
    DO K=R(I), R(I+1)-1
      REG=REG + A(K) * B(C(K), J)
    ENDDO
    D(I,J)= D(I,J)+REG
  ENDDO
ENDDO
```

Figure 1. Sparse matrix-dense matrix product with JIK ordering.

so the affected area between these accesses is:

$$S_{\text{int intf A}} = S_s(1) \cup S_s(1) \tag{4}$$

For the products by the remaining columns of matrix B, matrix D has the same behavior, but vectors A, C and R have a hit probability in the first access to each line that is the opposite of the cache area ratio affected by accesses to vectors since the last access to that line. As an example, this cache ratio for a line of vector A is:

$$\begin{aligned} S_{\text{intf A}} =& S_s(C(N\text{nz})C\text{s}) \cup S_s(N\text{nz}r) \cup \\ & S_s(Mr) \cup S_s(M) \cup \\ & S_l\left(2N, 1 - \frac{1-(1-p)^M}{pM}\right) \end{aligned} \tag{5}$$

where the first four componentes correspond to sequential accesses, modeled by S_s (see eq. (1)). They are associated, respectively, to the possible self interferences during the access to the whole vector A, plus the cross interferences generated by the references to vectors C and R, and one column of matrix D. The formula takes into account that both C and R are made up of integers by multiplying their number of elements by r. On the other hand, portions belonging to two columns of matrix B are accessed between two consecutive accesses to a line of vector A in two iterations of loop J. In this case we handle an irregular access pattern whose influence on the cache may be estimated using an uniform reference probability per line on the $2N$ elements that conform the two columns. The average probability is the average of the probabilities that a line has been accessed during the processing of $0, 1, \ldots, M-1$ rows of the sparse matrix. As the access probability per line is p for each row, for i rows this probability will be $1-(1-p)^i$. The probability in (5) is the average of this value for $i = 0, 1, \ldots, M-1$.

As a result, the number of misses on vector A may be estimated by:

$$M_{\text{A}} = M_{\text{A}}(1) + (H-1)M_{\text{A}}(S_{\text{intf A}}) \tag{6}$$

where $M_{\text{A}}(p)$ gives the the number of misses corresponding to the accesses to vector A during the product of the sparse matrix by one column of matriz B as a function of the probability p of miss in the first accesss to a line of this vector during this processing:

$$M_{\text{A}}(p) = \frac{N\text{nz}}{L\text{s}}(p + (L\text{s} - 1)S_{\text{int intf A}}) \tag{7}$$

However, each column of matrix B suffers indirect accesses dependent on the entries location in the sparse matrix, as it is addressed from the value of `C(J)`. We calculate the number of misses accessing this column 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. If we multiply p by the length of the vector in lines ($N/L\text{s}$) we obtain the number of lines of the column accessed in each dot product.

The hit probability in the first access to a line of matrix B in each dot product depends on the number of dot products carried out since the last access to this line. As a consequence, the hit probability is obtained as a weighted average of the probabilities associated to the different reuse distances, being its value in the j-'th row

$$P_{\text{hit B}}(j) = \sum_{i=1}^{j-1} p(1-p)^{i-1}(1 - S_{\text{intf B}}(i)) \tag{8}$$

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

$$\begin{aligned} S_{\text{intf B}}(i) =& S_l(C(N)C\text{s}, 1-(1-p)^i) \cup \\ & S_s(i\beta) \cup S_s(i\beta r) \cup \\ & S_s(ir) \cup S_s(i) \end{aligned} \tag{9}$$

where the five terms (see eq. (1) and (2)) correspond to the interference area ratios generated by the column itself (each one of the $C(N)$ potentially interferencing lines has a probabililty $1-(1-p)^i$ of having been accessed), vectors A, C and R and the current column of D, respectively. The average hit probability is obtained as

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

The $N\text{nz} - pM(N/L\text{s})$ accesses to matrix B not being the first one to a line during the dot products in which a column of this matrix is involved have an approximate probability

$$S_{\text{int intf B}} = S_s(1) \cup S_s(1) \tag{11}$$

of resulting in a miss, since only one line of vector A and another of C are referenced before the line is reused. So the

```
DO I=1, M
  DO K=R(I), R(I+1)-1
    REG0=A(K)
    REG1=C(K)
    DO J=1, H
      D(I,J)=D(I,J)+REG0*B(REG1,J)
    ENDDO
  ENDDO
ENDDO
```

Figure 2. Sparse matrix-dense matrix product with IKJ ordering.

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

Figure 3. Sparse matrix-dense matrix product with IJK ordering.

number of misses on this matrix is given by expression:

$$M_{\mathsf{B}} = H\Bigg(pM\frac{N}{L_{\mathsf{S}}}(1 - P_{\text{hit }\mathsf{B}}) + \left(N_{\text{nz}} - pM\frac{N}{L_{\mathsf{S}}}\right) S_{\text{int intf }\mathsf{B}}\Bigg) \tag{12}$$

Note that in this ordering the modeling of another basic algebra kernel is embebbed. JIK ordering corresponds to the sparse matrix-vector product repeated as many times as the number of columns, H, in the dense matrix (see Figure 1).

3.2 IKJ ordering

In the algorithm of Figure 2 both matrix D and matrix B are accessed by rows. Therefore, the hit probability on the reuse of their cache lines depends on the relationship between the first dimension of these matrices and the cache size, as several memory lines can be mapped to the same cache lines. In order to take into account this effect we calculate, by means of a deterministic algorithm [3], the cache area covered by the access to a row of each one of these matrices (S_{D} and S_{B}).

If $H \geq 2S_{\mathsf{B}}N_{\mathsf{C}}$ all the lines participating in the access to a row of B replace each other. It is thus impossible to get hit on reuses, the number of misses when accessing a row is H and there is a total of HN_{nz} misses. As a result, we can calculate $q_B = \max\{0, H - 2(H - S_{\mathsf{B}}N_{\mathsf{C}})\}$ as the number of lines that are not replaced by this phenomenon. The probability $P'_{\text{hit B}}$ of a hit over a line of B that has not been replaced by lines in its same row in different iterations of loop I can be calculated using the formulae explained in the previous section. The only change is that the number of lines that may generate interferences is no longer $C(N)$ but $\max\{0, S_{\mathsf{B}}N/L_{\mathsf{S}} - 1\}$. The number of misses on matrix B is

$$M_{\mathsf{B}} = N_{\text{l}}q_B(1 - P'_{\text{hit B}}) + (N_{\text{nz}} - N_{\text{l}})q_B S_{\text{D}} + N_{\text{nz}}(H - q_B) \tag{13}$$

where $N_{\text{l}} = MNp/L_{\mathsf{S}}$. In what follows we will denote as M_{V} the number of misses over a vector V.

For D the process is very similar to the one explained for matrix B. The area covered between two consecutive accesses to the same line of matrix D is $(S_{\text{B}} \cup S_s(1) \cup S_s(1))$, and $q_D = \max\{0, H - 2(H - S_{\text{D}}N_{\mathsf{C}})\}$. In this case the expression providing the number of misses is

$$M_{\mathsf{D}} = (\beta L_{\mathsf{S}} - 1)\,\frac{M}{L_{\mathsf{S}}}q_D(S_{\text{B}} \cup S_s(1) \cup S_s(1)) + \frac{M}{L_{\mathsf{S}}}q_D + N_{\text{nz}}(H - q_D) \tag{14}$$

For A and C there is an intrinsic miss every L_{S} accesses (L_{S}/r for integer vector C). When the reference is not the first to a line, the replacement probability is $S_{\text{B}} \cup S_{\text{D}}$ (between two consecutive accesses to A and C a complete row of B and D are accessed). Therefore, the number of misses for A is

$$M_{\mathsf{A}} = (1/L_{\mathsf{S}} \cup S_{\text{B}} \cup S_{\text{D}} \cup 1/N_{\mathsf{C}})N_{\text{nz}} \tag{15}$$

and M_{C} is calculated substituting $1/L_{\mathsf{S}}$ by r/L_{S} in the previous expression.

Finally, R is an integer vector with a intrinsic miss every L_{S}/r accesses. Between two consecutive accesses, β elements of A and C, one row of D and β rows of B are accessed. Calculating these areas we obtain the number of misses over R in a similar way as for A and C. Nonetheless, given the small portion of misses generated by R with respect to the total number, the model accuracy is almost not affected if we only consider the intrinsic misses ($(M + 1)r/L_{\mathsf{S}}$).

3.3 IJK ordering

In this version, shown in Figure 3, matrix D is still taken by rows, whereas matrix B is accessed by columns in loop K, but only accessing in each column the elements required to perform the inner product with a row of the sparse matrix. There may be reuse in this loop in each cache line if there are several entries in a set of L_{S} consecutive positions of the sparse matrix. With respect to the reuse in different iterations in loop I, matrix B is accessed by rows, there being q_B lines not affected by the self interferences with other lines of the row. Therefore the number of misses is

$$\begin{aligned} M_{\mathsf{B}} =& N_{\mathrm{l}}(H - q_B P\text{'}_{\text{hit B}}) + \\ & (N_{\mathrm{nz}} - N_{\mathrm{l}})H(2/N_{\mathrm{C}}) \end{aligned} \tag{16}$$

where $P\text{'}_{\text{hit B}}$, N_{l} and q_B are calculated as explained in the previous section.

Regarding matrix D, there is a hit probability on the reuses of q_D of each H accesses if they are not replaced due to cross interferences. This cross interference probability is

$$\begin{aligned} S_{\text{cross D}} =& S_l(NH, p) \cup S_s(1) \cup \\ & S_s(2\beta) \cup S_s(2\beta r) \end{aligned} \tag{17}$$

as there is an access with uniform reference probability per line p that affects the whole matrix B (NH words), a read of one element of vector R when starting the process of a new row of the sparse matrix, and the references to the data in vectors A and C that make up two consecutive rows of the sparse matrix, which contain on average 2β elements, and taking into account that vector R needs the use of the coeficient r due to the possibly different size of its elements (integers instead of floating point values). Finally the total number of misses over D is

$$M_{\mathsf{D}} = \frac{M}{L_{\mathsf{S}}}(H + (H - q_D(1 - P_{\text{cross D}}))(L_{\mathsf{S}} - 1)) \tag{18}$$

Vectors A and C are sequentially accessed in groups of elements corresponding to a given row of the sparse matrix. Each one of these groups is accessed H times before accessing the next one. Between two consecutive accesses to a cache line of these vectors, another two lines are accessed. Therefore, the misses obtained when accessing elements that are not the first one of a line are $N_{\mathrm{nz}}\frac{L_{\mathsf{S}}-1}{L_{\mathsf{S}}}H(2/N_{\mathrm{C}})$ for A, and $N_{\mathrm{nz}}\frac{L_{\mathsf{S}}-r}{L_{\mathsf{S}}}H(2/N_{\mathrm{C}})$ for C. Regarding the accesses corresponding to the first component of each line, they constitute intrinsic misses during the multiplication by the first column of B. The other $H - 1$ accesses to the first element of each line of A and C may generate misses with probabilities Pcross A and Pcross C, respectively, that take the following values:

$$P\text{cross A} = S_l(N, p) \cup S_s(1) \cup S_s(\beta r) \tag{19}$$

and

$$P\text{cross C} = S_l(N, p) \cup S_s(1) \cup S_s(\beta) \tag{20}$$

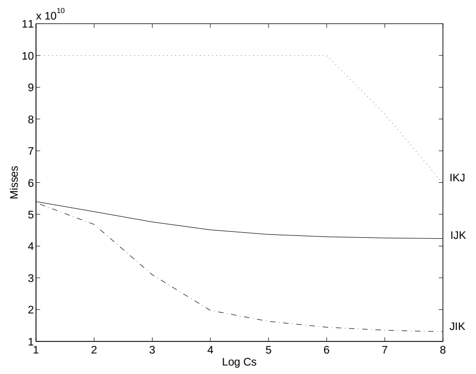

Figure 4. Number of misses during the sparse matrix-dense matrix product as a function of the ordering and C_{S} for a sparse matrix 10K×10K with $\alpha = 0.05$, $H = 10K$ and $L_{\mathsf{S}} = 8$.

where the first component stands for the accesses to a column of matrix dense B during its dot product with a row of a sparse matrix, the second one corresponds to the access to one element of vector R when starting the process of a new row, and the last one describes the acceses to vector C during the processing of a row of the sparse matrix (in Pcross A) or to vector A (in Pcross C).

The number of misses for vector R are estimated in the same way as in section 3.2, as their access pattern in the area covered between two consecutive accesses to the same line of this vector are the same in both cases.

4 Model validation and Ordering Comparison

The model has been validated by means of simulations carried out using synthetic matrices with a uniform distribution of the entries and a local developed simulator that was validated using dineroIII, belonging to the WARTS toolset [7]. Without any loss of generality, we have considered squared matrices ($N = M$) in the analysis, and $r = 1$ (the model is independent of the word size). Table 2 shows the model deviation for the three possible orderings of the sparse matrix-dense matrix product for some input parameters combinations. C_{S} is expressed in Kwords and L_{S} in words. The average errof of the model in the total set of trials performed has been 0.75% for the JIK ordering, 0.6% for IKJ and 0.79% for IJK.

We compare in Figures 4 through 6 the behavior of the three orderings JIK, IKJ and IJK (Figures 1, 2 and 3 respectively) of the sparse matrix-dense matrix product with respect to parameters C_{S} and L_{S}. The largest slope of the number of misses for the JIK ordering in Figure 4 occurs

N	α	H	C_S	L_S	Dev. JIK	Dev. IKJ	Dev. IJK
2000	5%	100	4	4	0.41%	-0.03%	1.02%
2000	5%	100	32	4	-0.03%	-0.04%	0.05%
2000	5%	100	4	8	0.11%	-0.01%	0.58%
2000	5%	100	32	8	-0.06%	-0.02%	0.03%
2000	5%	200	4	4	-1.45%	0.18%	-0.71%
2000	5%	200	32	4	-0.09%	0.00%	0.14%
2000	5%	200	4	8	-1.05%	0.69%	-0.41%
2000	5%	200	32	8	-0.07%	0.00%	0.07%
1000	1%	1000	4	4	-0.25%	-0.28%	0.08%
1000	1%	1000	32	4	2.91%	0.23%	0.17%
1000	1%	1000	4	8	-0.50%	0.00%	0.17%
1000	1%	1000	32	8	2.97%	1.02%	0.20%

Table 2. Deviation of the model for sparse matrix-dense matrix product with the three orderings.

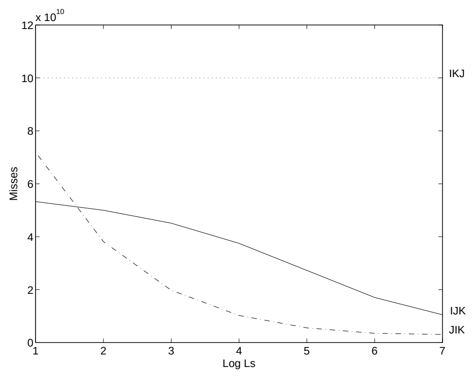

Figure 5. Number of misses during the sparse matrix-dense matrix product as a function of the ordering and L_S for a sparse matrix 10K×10K with $\alpha = 0.05$, $H = 10K$ and $C_S = 16$Kw.

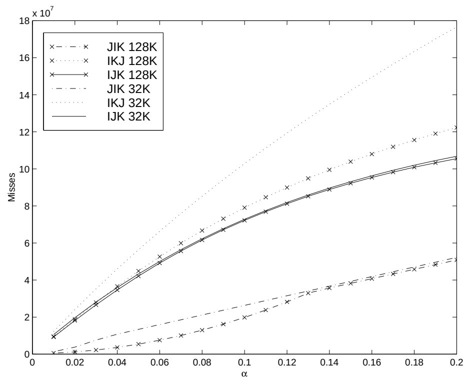

Figure 6. Number of misses during the sparse matrix-dense matrix product as a function of the ordering and α for a sparse matrix 1K×1K with $H = 1K$ and $L_S = 8$ for two possible cache sizes.

when C_S increases from a value much smaller than N to a similar one, as the self interferences in the access to column of matrix B processed in each iteration of loop J disappear and cross interferences are severely reduced. It may be observed in Figure 5 how the number of misses significantly decreases as L_S increases because the accesses to all the vectors except the column of matrix B are sequential (see Figure 1). Only for very large values of L_S the increase of the interference probability begins to unbalance the advantages obtained from a more efficient use of the spatial locality exhibited by the remaining vectors. This effect increases with the value of α.

The IKJ ordering is the only one in which the dense matrix is accessed by rows. As a result, it is very sensitive to self interferences in the access to the lines of a given row. In the case shown in Figure 4, $\text{mcm}(N = 10K, C_S) = 625C_S$, then self interferences appear from column $C_S/16$ on. Thus, only for $C_S \geq 64Kw$ can we take advantage when reusing lines. However, the large probability of cross interference in the reuse of the lines of any vector causes the number of misses for this ordering to be still quite larger than for the other two.

The problem of the self interferences in the reuse of lines of matrix B in products of different rows of the sparse matrix also arises in the IJK ordering. However, given that B is accessed by columns, the interferences between two successive accesses to the same line during the product times a row of the sparse matrix present a low probability. Matrix D is also accessed by columns, there are no self interferences and its cross interference probability is much smaller than that of the IKJ ordering. Besides, the cross interference probability in the accesses to vectors A and C is very small and their access is sequential in each dot product times a column of matrix B. For all of these reasons, the reduction of the number of misses is much more significant with respect to an increase of L_S than to an increase of C_S.

Finally, Figure 6 shows once more that no matter which is the density of our matrix, the evolution on the number of misses favours always the use of the JIK ordering, being the IKJ the worst option. This behavior also makes this last order to be the only one to get real benefits from large increases in the cache size for high densities once the cache can keep as many lines as columns the dense matriz has. The other two orders absorb the increase of accesses by the exploitation of the spatial locality thanks to the consecutive accesses to the lines that make up a column of matrix B during its dot product with a row of the sparse matrix.

5 Conclusions and Future Work

We have presented a model that allows to predict with great accuracy the number of misses during the execution of an algorithm with irregular access patterns, as the sparse matrix-dense matrix product. The model may be parameterized and considers matrices with an uniform entries distribution. It significantly extends the previous models in the literature as it includes the three possible types of cache misses (intrinsic misses, and self and cross interferences).

Order	N	α	H	C_S	L_S	Simulation time	Model time
IKJ	2000	5%	200	4	4	35.46	0.05
IKJ	2000	5%	200	32	4	33.28	0.28
IJK	2000	5%	200	4	4	43.82	0.05
IJK	2000	5%	200	32	4	42.88	0.30
JIK	2000	5%	200	4	4	42.22	0.01
JIK	2000	5%	200	32	4	39.35	0.01

Table 3. Example simulation and model user times on an Origin 200 server with R10000 processors at 180MHz.

As shown in Section 4, it is possible to analyze the behavior of the number of misses with respect to the basic characteristics of the cache (its size and its line size), the nesting order of the loops of the algorithm and the features of the matrix.

The execution time required by the programs that implement the particular models is significantly shorter than that of the execution of the algorithms or their simulations. Table 3 gives an idea of the relation between these times for some input parameter combinations. This difference grows the larger the dimensions and/or density of the matrices and vectors considered are.

The model proposed is modular enough to permit working with different sparse algebra kernels such as matrix transposition [3] or more complex algorithms considering blocking at the memory and register levels [4]. Another extension of the model already implemented but not explained here due to space limitations consists is the consideration of matrices with a non uniform entries distribution.

Finally, we have focused this work on the prediction of the number of misses. However, the model may be applied to the optimization of the performance of a memory hierarchy or a multiprogrammed system, or be used as a base for similar analyses in multiprocessor systems.

References

[1] 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, Philadelphia, 1994.

[2] D. Buck and M. Singhal. An analytic study of caching in computer systems. *Journal of Parallel and Distributed Computing*, 32(2):205–214, Feb. 1996.

[3] B. B. Fraguela. Cache miss prediction in sparse matrix computations. Technical report, Departamento de Electrónica e Sistemas da Univerdade da Coruña, April 1997.

[4] B. B. Fraguela, R. Doallo, and E. L. Zapata. Cache misses prediction for high performance sparse algorithms. In *Proceedings of the 4th Intl. Euro-Par Conference, LNCS*, volume 1470, pages 224–233, Sep 1998.

[5] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: An analytical representation of cache misses. In *Proceedings of the 11th International Conference on Supercomputing (ICS-97)*, pages 317–324, New York, July7–11 1997. ACM Press.

[6] B. L. Jacob, P. M. Chen, S. R. Silverman, and T. N. Mudge. An analytical model for designing memory hierarchies. *IEEE Transactions on Computers*, 45(10):1180–1194, Oct 1996.

[7] A. R. Lebeck and D. A. Wood. Cache profiling and the spec benchmarks: A case study. *IEEE Computer*, 27(10):15–26, Oct 1994.

[8] O. Temam, C. Fricker, and W. Jalby. Cache interference phenomena. In *Proceedings of the Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 261–271, New York, NY, USA, May 1994. ACM Press.

[9] O. Temam and W. Jalby. Characterizing the behaviour of sparse algorithms on caches. In *Proc. IEEE Int'l. Conf. on Supercomputing (ICS'92)*, pages 578–587, Nov 1992.

[10] R. A. Uhlig and T. N. Mudge. Trace-driven memory simulation: A survey. *ACM Computing Surveys*, 29(2):128–170, June 1997.