Cache Misses Prediction for High Performance Sparse Algorithms*

Basilio B. Fraguela[1], Ramón Doallo[1], and Emilio L. Zapata[2]

[1] Dept. Electrónica e Sistemas. Univ. da Coruña,
Campus de Elviña s/n, A Coruña, 15071 Spain
{basilio,doallo}@udc.es
[2] Dept. de Arquitectura de Computadores. Univ. de Málaga, Complejo Tecnológico
Campus de Teatinos, Málaga, 29080 Spain
ezapata@ac.uma.es

Abstract. Many scientific applications handle compressed sparse matrices. Cache behavior during the execution of codes with irregular access patterns, such as those generated by this type of matrices, has not been widely studied. In this work a probabilistic model for the prediction of the number of misses on a direct mapped cache memory considering sparse matrices with an uniform distribution is presented. As an example of the potential usability of such types of models, and taking into account the state of the art with respect to high performance superscalar and/or superpipelined CPUs with a multilevel memory hierarchy, we have modeled the cache behavior of an optimized sparse matrix-dense matrix product algorithm including blocking at the memory and register levels.

1 Introduction

Nowadays superscalar and/or superpipelined CPUs provide high speed processing, but global performances are constrained due to memory latencies even despite the fact that a multilevel memory organization is usual. Performances are further reduced in many numerical applications due to the indirect accesses that arise in the processing of sparse matrices, because of their compressed storage format [1].

Several software techniques for improving memory performance have been proposed, such as blocking, loop unrolling, loop interchanging or software pipelining. Navarro et al. [5] have studied some of these techniques for the sparse matrix-dense matrix product algorithm. They have proposed an optimized version of this algorithm as a result of a series of simulations on a DEC Alpha processor. The traditional approach for cache performance evaluation has been the software simulation of the cache effect for every memory access [7]. Different approaches consist in providing performance monitoring tools (built-in counters in current microprocessors), or the design of analytical models.

* 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

This paper chooses this last approach and addresses the problem of the estimation of the total number of misses produced in a direct-mapped cache using a probabilistic model. In a previous work, Temam and Jalby [6] model the cache behavior for the sparse matrix-vector product of uniform banded sparse matrices, although they do not consider cross interferences. Simpler sparse linear algebra kernels than the one this paper is devoted to have been also modeled in [2], being the purpose of this work to demonstrate that it is feasible to extend this type of models to more complex algorithms, such as the one mentioned above. This fact can make possible further improvements for these codes by making easier the study of the effect of these techniques on the memory hierarchy. An extension of the proposed model for K-way associative caches has been introduced in [3].

The remainder of the paper is organized as follows: Section 2 describes the algorithm to be modeled. Basic model aparameters and concepts are introduced in Sect. 3 together with a brief explanation, due to space limitations, of the modeling process. In Sect. 4 the model is validated and used to study the cache behavior of the algorithm as a function of the block dimensions and the cache main parameters. Section 5 concludes the paper.

2 Modeled Algorithm

The optimized sparse matrix-dense matrix product code proposed in [5] is shown in Fig. 1. The sparse matrix is stored using the Compressed Row Storage (CRS) format [1]. This format uses three vectors: vector `A` contains the sparse matrix entries, vector `C` stores the column of each entry, and vector `R` indicates in which point of `A` and `C` a new row of the sparse matrix starts and permits knowing the number of entries per row. As a result the sparse matrix must be accessed row-wise. The dense matrix is stored in `B`, while `D` is the product matrix.

The loops are built so that variable `I` always refers to the rows of the sparse matrix and matrix `D`, `J` refers to the columns of matrices `B` and `D`, and `K` traverses the dimension common to the sparse matrix and the dense matrix. Our code is based on a `IKJ` order of the loops (from the outermost to the innermost loop).

This code uses one level blocking at the cache level selecting a block of matrix `B` with `BK` rows and `BJ` columns. The accesses to the block in the inner loop are row-wise, so a copy by rows to a temporal storage `WB` is desired in order to avoid self interferences and achieve a good exploitation of the spatial locality.

The performance has been further improved by applying blocking at the register level. This code transformation consists in the application of strip mining to one or more loops, loop interchanging, full loop unrolling and the elimination of redundant load and store operations. In our case, inner loop `J` has been completely unrolled and the load and stores for `D` have been taken out of loop `K`. This modification requires the existence of `BJ` registers (`d1`, `d2`, ..., `dbj` in the figure) to store these values, as Fig. 1 shows. The resulting algorithm has a bidimensional blocking for registers, resulting in fewer loads and stores per arithmetic operation. Besides the number of independent floating point operations in the loop body is increased.

```
 1 DO J2=1, H, BJ
 2   LIMJ=J2+MIN(BJ, H-J2+1)-1
 3   DO I=1, M+1
 4     R2(I)=R(I)
 5   ENDDO
 6   DO K2=1, N, BK
 7     LIMK=K2+MIN(BK, N-K2+1)
 8     DO J=1, LIMJ-J2+1
 9       DO K=1, LIMK-K2
10         WB(J,K)=B(K2+K-1,J2+J-1)
11       ENDDO
12     ENDDO
13     DO I=1, M
14       K=R2(I)
15       LK=R(I+1)
16       d1=D(I,J2)
17       d2=D(I,J2+1)
         ...
18       dbj=D(I,J2+BJ-1)
                  ↪
19       DO WHILE (K<LK AND C(K)<LIMK)
20         a=A(K)
21         ind=C(K)
22         d1=d1+a*WB(1,ind-j2+1)
23         d2=d2+a*WB(2,ind-j2+1)
           ...
24         dbj=dbj+a*W(BJ, ind-j2+1)
25         K=K+1
26       ENDDO
27       D(I,J2)=d1
28       D(I,J2+1)=d2
         ...
29       D(I,J2+BJ-1)=dbj
30       R2(I)=K
31     ENDDO
32   ENDDO
33 ENDDO
```

Fig. 1. Sparse matrix-dense matrix product with IKJ ordering and blocking at the memory and register levels

3 Probabilistic Model

We call intrinsic miss the one that takes place the first time a given memory block is accessed. The accesses to the block from that moment on result in hits unless the block is replaced. This happens in a direct mapped cache if and only if another memory block mapped to the same cache line is accessed. If this block belongs to the same vector as the replaced line it is called a self interference, whereas if it belongs to another vector it is called a cross-interference.

The replacement probability grows with the cache area (number of lines) affected by the accesses between two consecutive references to the considered block. This area depends on the memory location of the vectors to be accessed.

In most cases, two consecutive references to a given block are separated by accesses to several vectors that cover different areas which are measured as ratios, as they correspond to line replacement probabilities. The total area covered by these accesses in the cache is calculated adding these areas as independent probabilities[1], operation that we express with symbol $\cup$. This way our probabilistic model does not make any assumptions on the relative location of these vectors in memory.

The main parameters our model employs 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. Integers are considered through the use of parameter r.

[1] Given the independent probabilities p_1 and p_2 of events A_1 and A_2 respectively, the probability of A_1 or A_2 is obtained as $p_1 \cup p_2 = p_1 + p_2 - p_1 p_2$

Table 1. Notation used

C_s	Cache size in words
L_s	Line size in words
N_c	Number of cache lines (C_s/L_s)
M	Number of rows of the sparse matrix
N	Number of columns of the sparse matrix
H	Number of columns of the dense matrix
BJ	Block size in the J dimension
BK	Block size in the K dimension
N_{BJ}	Number of blocks in the J dimension (H/BJ)
N_{BK}	Number of blocks in the K dimension (N/BK)
N_{nz}	Number of entries of the sparse matrix
p_n	Probability that a position in the sparse matrix contains an entry $(\frac{N_{nz}}{M \cdot N})$
r	$\frac{\text{size of an integer}}{\text{size of a real}}$

A main issue of the proposed model we want to point out is the way the cache area affected by the accesses to a given vector is calculated. This area depends on the vector access pattern and the cache parameters. For example, a sequential access to x words loads $(x + L_s - 1)/L_s$ lines on average. The cache area they cover is

$$S_s(x) = \min\{1, (x + L_s - 1)/C_s\} \tag{1}$$

In a similar way different expressions or iterative methods have been developed to estimate the average cache area affected by the different types of accesses to a given vector. Combinig these expressions and adding the areas covered by the accesses to the different vectors the average cross interference probability is estimated. Also the self interference probability may be calculated as a function of the vector size, the cache parameters, and the vector access pattern. Miss rate calculation is directly obtained from the previous probabilities. These are the access patterns found in this code:

1. Sequential accesseses on vectors `A` and `C` within groups of entries (not positions) located in the same row in a set of BK consecutive columns in lines 19-21. There is a hit in reuse probability in the four loops (with index variables `K`, `I`, `K2` and `J2` from the inner to the outer) with decreasing weight. These accesses are slightly irregular because of the different number of entries in each considered subrow and the offset inside the vectors between the data corresponding to entries in consecutive subrows. The difference between these vectors is that `C` is always accessed at least once in each iteration of loop 13-31 in line 19, and it is always accessed once more than vector `A`, as it is used to detect the end of the subrow.
2. Access to matrix `B` in groups of BJ consecutive subcolumns of BK elements each in line 10. Recall that the access is by columns, and FORTRAN stores

matrices by columns, so it is completely sequential in each subcolumn. There is only a real reuse probability in loop 9-10, as each element is only accessed once.

3. Subrows of BJ consecutive positions of matrix `D` read in lines 16-80 and written in lines 27-29. The read operation has a hit in reuse probability in the loop indexed by `I`, and when the cache is large, also in the loop on `K2`. The write operations will result in hits providing they are not affected by the self interferences in the accesses to a row and the cross interferences due to the accesses in lines 19-26.
4. Vectors `R` and `R2` have almost the same access pattern: they both are sequentially accessed in loop 3-5 and also inside the loop indexed by `I` (lines 14, 15 and 30: this last line gives the difference between these vectors). In the first case, only an access to the other vector takes place between two consecutive accesses to the same line, while in the second the cross interference probability is much higher. There is a hit in reuse probability in the first access to each line in the two loops when the cache is large with respect to the block size.
5. Matrix `WB`, which will be studied below, is accessed by rows in loops 8-11 to be filled with the values of the block to process, and by columns -thus, sequentially- in the inner loop, being the column chosen as a function of the column of the considered entry. This last fact makes the selection irregular. When this matrix fits in the cache, there is some hit in reuse probability in the first access to each line due to the previous access in loops 8-11. Conversely, the accesses in line 10 have a hit in reuse probability derived from the accesses in lines 22-24.

As an example, and due to space limitations, we shall only explain the way the number of misses on matrix `WB` has been modeled, as it is usually responsible for most of the misses and it has the more complex access pattern. This matrix contains a transposed copy of the block of matrix `B` that is being processed, so it has BJ rows and BK columns. We calculate first the number of misses for `WB` in the inner loop and then the number of misses during the copying process of the block of matrix `B`.

3.1 Misses on Matrix `WB` in the Inner Loop

The hit probability for a given line of matrix `WB` during the processing of the j-th row is calculated as

$$\begin{aligned} P_{\text{hit WB}}(j) = & \sum_{i=1}^{j-1} P(1-P)^{i-1}(1-P)^{i \cdot n_l}(1 - P_{\text{cross WB}}(i)) \\ & + (1-P)^{j-1}(1-P)^{j \cdot n_l}(1 - P_{\text{cross WB}}(j))P_{\text{surv WB}} \end{aligned} \quad (2)$$

where $n_l = \max\{0, S_s(BJ \cdot BK) - 1\}$ is the average number of lines of `WB` that compete with another line of this matrix for a given cache line and $P = 1 - (1 - p_n)^{Av}$ is the probability that a line of matrix `WB` is accessed during the processing of a subrow of the sparse matrix, being Av the average of different columns of the matrix that have elements in the same line.

In this way, $P(1-P)^{i-1}$ is the probability that the last access to the considered line has taken place i iterations before of the loop on variable `I`. Function $P_{\text{cross WB}}(j)$ gives the cross interference probability generated by the accesses to other matrices and vectors during the processing of j subrows of the sparse matrix. The value is calculated as

$$P_{\text{cross WB}}(j) = S_D(j) \cup S_A(j) \cup S_C(j) \cup S_s(j \cdot r) \cup S_s(j \cdot r) \tag{3}$$

where the last two terms stand for the area covered by the accesses to `R` and `R2`. The areas corresponding to the references to matrix `D` and vectors `A` and `C` are calculated using a deterministic algorithm further developed in [2].

The last addend in expression (2) stands for the probability of the reuse of a line that has not been referenced yet in loop `I` (probability $(1-P)^{j-1}$) but that has remained in the cache since the copying process. This probability of hit in the reuse means excluding both self (probability $(1-P)^{j \cdot n_1}$) and cross interferences ($1 - P_{\text{cross WB}}(j)$) as well as being in cache when the copying has just finished ($P_{\text{surv WB}}$, whose calculation is not included here due to space limitations).

The number of misses on matrix `WB` in the inner loop is calculated multiplying the average miss probability by the number of accesses to the first element of a line in this loop, as only in the access to the beginning of a line can a miss take place:

$$F_{\text{inner WB}} = \left(1 - \frac{\sum_{j=1}^{m} P_{\text{hit WB}}(j)}{M}\right) \left(\frac{BJ+L_s-1}{L_s}\right) \left(\frac{N_{nz} \cdot BK}{N}\right) N_{BJ} \cdot N_{BK} \tag{4}$$

3.2 Misses on Matrix `WB` in the Copying

The number of misses during the copying is obtained multiplying the average number of misses per copy process by the number of blocks, $N_{BJ}N_{BK}$. This value is estimated as

$$F_{\text{copy WB}} = \sum_{j=1}^{BJ} \sum_{i=1}^{\frac{BJ \cdot BK}{L_s}} A \cdot S_s(1) + \left(\frac{L_s}{BJ} - A\right) P_{\text{miss first}}(i,j) \tag{5}$$

where $A = \max\{0, \frac{L_s}{BJ} - 1\}$ is the average number of accesses after the first one to a given line of matrix `WB` during an iteration of the loop on line 8. As the equation shows, the miss probability for these accesses is the associated to an access to a line of `B`. As for $P_{\text{miss first}}(i,j)$, it is the probability of a miss for the first access to line i during iteration j of the loop, which is calculated as

$$\begin{aligned} P_{\text{miss first}}(i,j) = {} & P_{\text{acc}}(j-1)(S_{\text{self}}(BJ, BK) \cup S_s(BK)) \\ & + (1 - P_{\text{acc}}(j-1))((1 - P_{\text{hit WB}}(M)) \cup P_{\text{exp WB}}(i,j)) \end{aligned} \tag{6}$$

where $P_{\text{acc}}(j)$ if the probability of the line having been accessed in the j previous iterations, which is $\min\{1, (L_s + j - 2)/BJ\}$ but for $j = 0$, for which the probability is null. The first access to a line results in a miss if it is not in the cache

when the copying begins (probability $1 - P_{\text{hit WB}}(M)$) or if it has been replaced during the copy of the elements previously accessed, which is $P_{\text{exp WB}}(i, j)$. On the other hand, if the line has been accessed in the previous iteration, only the accesses to BK elements of matrix `B` located in two consecutive columns (whose area we approach by a completely sequential access) and the accesses to a row of matrix `WB` can have replaced the considered line. This last probability is calculated using the deterministic algorithm we have mentioned above.

4 Cache Behavior of the Algorithm

The model was validated with simulations on synthetic matrices made by replacing the references to memory by functions that calculate the position to be accessed and feed it to the dineroIII cache simulator, belonging to the WARTS tools [4]. Table 2 shows the model accuracy for some combinations of the input parameters using synthetic matrices. The average error obtained in the trial set was under 5%.

Table 2. Predicted and measured misses and deviation of model for optimized sparse matrix-dense matrix product with an uniform entries distribution. Order ($M = N$), N_{nz} and H in thousands, C_s in Kwords and number of misses in millions

Order	N_{nz}	p_n	H	BJ	BK	C_s	L_s	predicted misses	measured misses	Dev.
2	20	0.005	0.2	25	500	8	4	1.524	1.516	-0.55%
2	20	0.005	0.2	25	500	16	4	1.162	1.163	0.10%
2	20	0.005	0.2	25	500	32	4	0.962	0.972	1.08%
2	20	0.005	0.2	25	500	8	8	0.887	0.872	-1.70%
2	20	0.005	0.2	25	500	16	8	0.667	0.660	-0.99%
2	20	0.005	0.2	25	500	32	8	0.552	0.553	0.17%
10	100	0.001	10	20	1000	8	4	646	672	4.05%
10	100	0.001	10	20	1000	16	4	608	612	0.72%
10	100	0.001	10	20	1000	32	4	549	546	-0.45%
10	100	0.001	10	20	1000	64	4	490	492	0.41%
10	100	0.001	10	20	1000	8	8	401	419	4.40%
10	100	0.001	10	20	1000	16	8	373	378	1.35%
10	100	0.001	10	20	1000	32	8	321	323	0.65%
10	100	0.001	10	20	1000	64	8	287	290	0.86%

As a first result of our modeling, a linear increase of the number of misses with respect to N and M due to blocking technique is shown. The exception are the accesses to matrix `D`, as they have an access stride M, which makes them increase noticeably when M is a divisor or multiple of C_s. Figure 2 illustrates the evolution of the number of misses with respect to the block size for a given matrix. We have supposed a value of BJ between 8 and 30, this is, that there

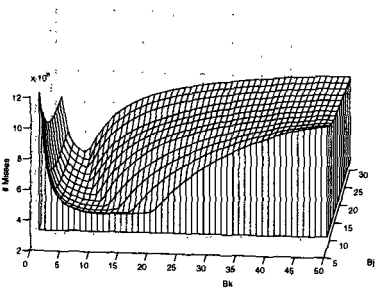

Fig. 2. Number of misses on a 16Kw cache with $L_s = 8$ for a $M = N = 5000$ sparse matrix with $p_n = 0.02$ and $H = 100$ as a function of the block size (BK in hundreds)

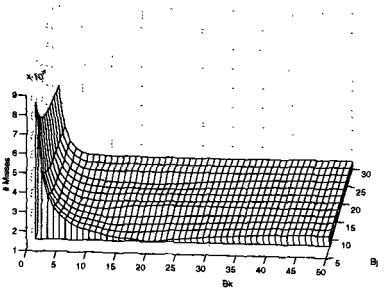

Fig. 3. Number of misses on a 16Kw cache with $L_s = 8$ for a $M = N = 5000$ sparse matrix with $p_n = 0.002$ and $H = 100$ as a function of the block size (BK in hundreds)

are up to 30 registers available for the blocking at the register level. The number of misses grows in the two directions of the axis tending asymptotically to a maximum when $BJ \times BK > C_s$. The minimum number of misses is obtained in a case in which the block fits completely in the cache ($BJ = 22$, $BK = 700$). Besides, among all of the possible block sizes the best has been the one with the greatest value of BJ that multiplied by a multiple of 100 (the variation of the BK value in the figure is 100 and that of BJ is 2) gives the greatest value under C_s. The reason is that the greater BJ is, lesser blocks there are in the J direction, reducing the number of accesses. In addition, accesses to matrix **WB** in the inner loop take place sequentially in the J direction, whose components are stored in consecutive memory positions. As a result exploitation of the spatial locality is improved with the block size increase in this direction. Simulations showed that the actual optimum block was ($BJ = 20$, $BK = 800$), with 5.6% less misses, which is a slight difference due to the small errors of the model.

Although the evolution of the number of misses usually follows the previous structure, we can obtain important variations depending on several factors. For example, Fig. 3 shows a similar plot for a matrix with lesser p_n. Here the optimum block size is ($BJ = 24$, $BK = 2000$), which is much greater than C_s (almost three times). This is because p_n is much smaller in this matrix, reducing remarkably the replacement probability due to self and cross interferences. In this way much greater blocks may be used, reducing the number of accesses and misses. On the other hand, although there are little variations, the number of misses is stabilized when the block size is similar to or greater than C_s because the interference increase is balanced with the lesser number of blocks to be used. The optimum block size was checked with simulations which indicated the same value for BJ and a value of 2100 for BK, with 0.04% less misses.

Finally, in Fig. 4 we study the cache behavior as a function of the cache parameters. *Log Cs* axis stands for the logarithm base two of C_s measured in Kw. There is an increase of the number of misses when the block size is greater

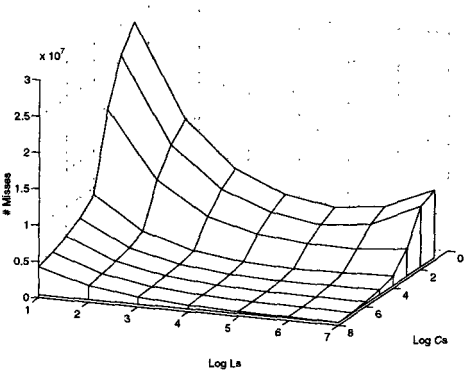

Fig. 4. Number of misses for a $M = N = 5000$ sparse matrix with $p_n = 0.02$ and $H = 100$ using a block $BJ = 22$, $BK = 700$ as a function of C_s and L_s

than C_s. Besides this value is always much greater for small line sizes ($L_s \leq 8$ words). The optimum line size turns out to be 32 when the cache size is greater than or similar to that of the block. This means a balance between the spatial locality exploitation in the access to WB in the inner loop of the algorithm and the self interference probability. As C_s grows, so does the optimum line size due to the reduction in the self and cross interference probabilities.

5 Conclusions

The proposed model may be parameterized and considers matrices with an uniform distribution. It significantly extends the previous models in the literature as it incorporates the three possible types of misses (intrinsic misses, self and cross interferences) in the cache and it takes into account the propagation of data between successive loops of the code (reusability). Besides the modeled code size is noticeably greater than those used in most previous works and their predictions are highly reliable. An effort has been done to structure the model in algorithms and formulae related to typical access patterns so that they can be reused when studying other codes.

The time required by our probabilistic model to get the miss predictions is much smaller than that of the simulations, being this difference larger the larger the dimensions and/or sparsity degree of the considered sparse matrix. For example, the time required to generate the trace corresponding to the product of a matrix with $M = N = 1500$ and 125K entries with a block size ($BJ = 10$, $BK = 500$) by a matrix of the same size (almost 246M references) and process it with dineroIII simulating a 16Kw cache with a line size of 8 words was around 20 minutes on a 433 MHz DEC Alpha processor, while our model required between 0.1 and 0.2 seconds.

As for the time required to obtain the best block size for a given matrix and cache, the time required to do one simulation for each possible pair (BJ, BK)

in Fig. 2 would be around 54.5 hours on this machine. Anyway, as different locations of the vectors in memory generate different numbers of misses, some other simulations would be needed to get an average for each pair. On the other hand our model required only 350.5 seconds.

As shown in Sect. 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 block dimensions and the features of the matrix. Besides the behavior of each vector may be studied in a separate manner.

References

1. Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., H. van der Vorst, H.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM Press (1994).
2. Fraguela, B.B.: Cache Misses Prediction in Sparse Matrices Computations. Technical Report UDC-DES-1997/1. Departamento de Electrónica e Sistemas da Univerdade da Coruña (1997).
3. Fraguela, B.B, Doallo, R., Zapata, E.L.: Modeling Set Associative Caches Behavior for Irregular Computations. To appear in Proc. ACM Sigmetrics/Performance Joint Int'l. Conf. on Measurement and Modeling of Computer Systems, Madison, Wisconsin, (1998).
4. Lebeck, A.R., Wood, D.A.: Cache Profiling and the SPEC Benchmarks: A Case Study. IEEE Computer, **27(10)** (1994) 15–26.
5. Navarro, J.J., García, E., Larriba-Pey, J.L., Juan, T.: Block Algorithms for Sparse Matrix Computations on High Performance Workstations. Proc. ACM Int'l. Conf. on Supercomputing (ICS'96) (1996) 301–309.
6. Temam, O., Jalby, W.: Characterizing the Behaviour of Sparse Algorithms on Caches. Proc. IEEE Int'l. Conf. on Supercomputing (ICS'92) (1992) 578–587.
7. Uhlig, R.A, Mudge, T.N.: Trace-Driven Memory Simulation: A Survey. ACM Computing Surveys **29** (1997) 128–170.