Cache Probabilistic Modeling for Basic Sparse Algebra Kernels involving Matrices with a Non Uniform Distribution *

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

A probabilistic model to estimate the number of misses on a set associative cache with an LRU replacement algorithm is introduced. Such type of modeling has been done by our group in previous works for sparse matrices with an uniform distribution of the non zero elements. In this paper we present new results focusing in different types of distributions that usually appear in some well-known real matrices suites, such as Harwell-Boeing or NEP.

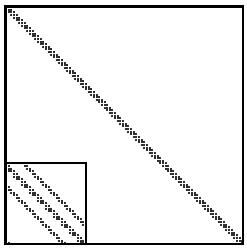

Figure 1. CRY10000 matrix, belonging to the NEP collection, and detail of the band

1. Introduction

A large number of scientific applications work with sparse matrices, this is to say, matrices with a very low percentage of non zero elements or entries. These matrices are stored in compressed formats [2] in order to reduce the operations and memory needed. These formats generate irregular patterns of references to memory due to the indirect addressing, thus reducing the performance of the memory hierarchy and making hard to analyze the cache behaviour.

In this paper a general model to study cache behaviour under such kind of workloads, presented in [6] for sparse matrices with a completely uniform distribution of the entries, is applied to a classic algebra kernel such as sparse matrix-vector product using real matrices from the Harwell-Boeing [4] and NEP [1] collections. Trace driven simulations are a common method for the estimation of code performance. Another usual approach is the use of models that derive their input parameters from traces [3], [7], [9]. Probabilistic models reduce estimation times and provide more flexibility for the parametric study of the cache. There are few previous works on this type of modeling of cache performance for sparse codes and their results are quite limited. For example, [10] models the self interference misses on the indirectly accessed vector of the sparse matrix-vector product on a direct-mapped cache, and only for a completely uniform distribution of the entries. Our approach considers caches with an arbitrary size, line size and associativity using a LRU replacement algorithm and takes into account all the kinds of misses on all of the data structures. The set of matrices we consider in this work are banded square matrices where the entries are uniformly and indepently distributed along the diagonals of the band, there being different densities of entries in the diagonals. An example of the type of matrices we are talking about is shown in Figure 1.

Basic model concepts and expressions are introduced in the next section. The modeling of the sparse matrix-vector product is developed in Section 3, and its extension to several loop orderings in the sparse matrix-dense matrix product is further explained in Section 4. The validation of the model is carried out by means of simulations on real matrices in Section 5. Finally, Section 6 concludes the paper.

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

2. Model basics

In our model there are two kinds of misses: intrinsic misses, which are those that take place the first time a memory block corresponding to a cache line is accessed, and interference misses, which are misses on lines that have been replaced from the cache due to the interferences generated by the accesses to other lines. When these lines belong to the same program vector as the considered one, they produce a self interference, while when they belong to another vectors, cross interferences are generated.

On the other hand, in order to produce an interference miss on a given line in a K-way asssociative cache with an LRU replacement algorithm, which is the one our model considers, at least K different lines mapped to the cache set where this line resides must be accessed between two consecutive accesses to it. Notice that when $K = 1$ the behavior is that of direct mapped caches.

This means that the number of misses on any vector can be estimated by knowing for each line of this vector the number of different lines mapped to its cache set that are referenced between consecutive accesses to this line. We formalize this idea by means of the area vector. Given a vector V, we define the area vector $S_{\mathsf{V}} = S_{\mathsf{V}0} S_{\mathsf{V}1} \ldots S_{\mathsf{V}K}$, where $S_{\mathsf{V}0}$ is the ratio of sets that have received K or more lines of vector V, while $S_{\mathsf{V}i}, 0 < i \leq K$, is the ratio of sets that have received $K - i$ lines. This means that $S_{\mathsf{V}i}$ is the ratio of cache sets that require accesses to i new different lines to replace all the lines they contained when the access to V started.

General expressions may be developed to calculate the area vector corresponding to a given access pattern. For example, the area vector $S_{\mathsf{S}}(n)$ for a sequential access to n consecutive memory words is

$$\begin{aligned} &S_{s(K-\lfloor l \rfloor)}(n) = 1 - (l - \lfloor l \rfloor) \\ &S_{s(K-\lfloor l \rfloor - 1)}(n) = l - \lfloor l \rfloor \\ &S_{si}(n) = 0 \;\; 0 \leq i < K - \lfloor l \rfloor - 1, K - \lfloor l \rfloor < i \leq K \end{aligned} \tag{1}$$

where $l = (n + L_{\mathrm{s}} - 1)/(L_{\mathrm{s}} N_K)$ is the average number of lines that are mapped to each set. If $l \geq K$ then $S_{\mathsf{S}0} = 1, S_{\mathsf{S}i} = 0, 0 < i \leq K$, 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.

The area vectors corresponding to all of the vectors accessed between consecutive references to a line of a given vector must be composed to get the miss probability for these references. Given two area vectors $S_{\mathsf{U}} = S_{\mathsf{U}0} S_{\mathsf{U}1} \ldots S_{\mathsf{U}K}$ and $S_{\mathsf{V}} = S_{\mathsf{V}0} S_{\mathsf{V}1} \ldots S_{\mathsf{V}K}$ corresponding to the accesses to vectors U and V, we define the union area vector $S_{\mathsf{U}} \cup S_{\mathsf{V}}$, which includes the lines belonging to both vectors, as

$$\begin{aligned} &(S_{\mathsf{U}} \cup S_{\mathsf{V}})_0 = \textstyle\sum_{j=0}^{K} \left(S_{\mathsf{U}j} \sum_{i=0}^{K-j} S_{\mathsf{V}i} \right) \\ &(S_{\mathsf{U}} \cup S_{\mathsf{V}})_i = \textstyle\sum_{j=i}^{K} S_{\mathsf{U}j} S_{\mathsf{V}(K+i-j)} \qquad 0 < i \leq K \end{aligned} \tag{2}$$

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.

Table 1 summarizes the input parameters for the model.

C_{s}	Cache size in words
L_{s}	Line size in words
K	Associativity
N_K	Number of cache sets $(C_{\mathrm{s}}/(L_{\mathrm{s}} \cdot K))$
N_{nz}	Number of entries of the sparse matrix
N	Dimension of the sparse matrix
W	Bandwitdh
H	Number of columns of the dense matrix
d_i	Probability that a position in diagonal i of the sparse matrix contains an entry

Table 1. Notation used

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

Figure 2. Sparse matrix-vector product

3. Modeling the Sparse Matrix-Vector Product

The code for this algebra kernel is shown in Figure 2. The sparse matrix is stored in Compressed Row Storage (CRS) format [2]: A contains the matrix entries, C stores the column of each entry, and R points the location in A and C where a new row of the sparse matrix starts. These three vectors and D, the destination vector of the product, exhibit flawless locality due to their sequential access. There are no self interferences and very few misses due to cross interferences, as only a few memory positions are referenced between consecutive accesses to any of these vectors. Therefore it is a good approach to consider that there are only intrinsic misses on them. Anyway, our model considers these cross interferences, although their calculation will not be included here due to space limitations.

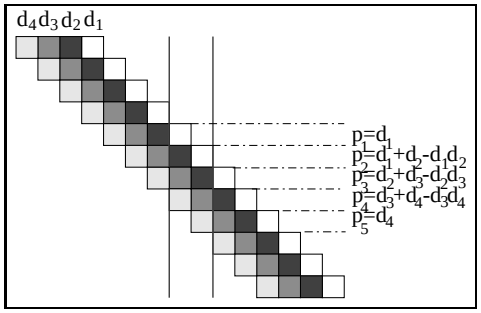

Figure 3. Probabilities that there is at least one entry in each of the $W + L_s - 1$ rows of the band that cross $L_s = 2$ given consecutive columns, being $W = 4$

On the other hand, vector X is indirectly addressed trough array C, giving place to an irregular access pattern. The number of misses on this vector is estimated calculating the average miss probability for each line accessed during the dot product of a row of the sparse matrix by this vector. The model only takes into account the first access to each line in each dot product, as the remaining accesses have a very low probability of resulting in a miss, because they refer to lines that have been accessed in the previous iteration of the inner loop. The access probabilities to a line of vector X corresponding to the $W + L_s - 1$ rows of the matrix that contain entries whose column correspond to this line are calculated from the densities of the diagonals that cross each row in a set of L_s consecutive columns. Figure 3 shows the case for a band with $W = 4$ and a line size of two words. These probabilities are called $p_1, p_2, \ldots p_{W+L_s-1}$, in increasing order of the row they are associated to.

The hit probability in the first access to a given line of vector X during the dot product of the j-th row within the band of the sparse matrix with the mentioned vector is calculated as:

$$P_{\text{hit X}}(j) = \sum_{i=1}^{j-1} p_i \prod_{k=i+1}^{j-1} (1 - p_k)\,(1 - P_{\text{intf X}_0}(i,j)) \quad (3)$$

being $P_{\text{intf X}}(i,j)$ the area vector corresponding to the interferences generated during the process between rows i and j, which is calculated as

$$P_{\text{intf X}}(i,j) = P_{\text{self X}}(i,j) \cup P_{\text{cross X}}(j-i) \quad (4)$$

where $P_{\text{cross X}}(i)$ is the area vector corresponding to the cross interferences generated during i dot products. Is is calculated as the union of the area vectors correponding to the sequential accesses to i elements of vectors R and D and $i \cdot N_{\text{nz}}/N$ elements of vectors A and C (see general expressions in Section 2). On the other hand $P_{\text{self X}}(i,j)$ is the area vector corresponding to the self interferences vector X generated during the process between rows i and j. In order to calculate this vector it must be taken into account that there are $N_{\text{R}}(j) = \lfloor \frac{j-2}{N_K L_s} \rfloor$ lines of vector X to the right of the one considered that are mapped to the same cache set and may be accessed during the process of the rows between i and j, being their access probabilities

$$P_{\text{R}lN_K}(i,j) = 1 - \prod_{k=\max\{1, i-lN_K L_s\}}^{j-lN_K L_s-1} (1 - p_k), l = 1, \ldots, N_{\text{R}}(j) \quad (5)$$

and other $N_{\text{L}}(i) = \lfloor \frac{W+L_s-2-i}{N_K L_s} \rfloor$ lines to the left whose access probabilities $P_{\text{L}lN_K}(i,j)$ can be calculated in a similar way to expression (5).

As the $N_{\text{R}}(j)+N_{\text{L}}(i)$ lines that are mapped to the considered cache set have different access probabilities, we cannot apply the binomial distribution to calculate the associated area vector, as it was the case in a band with an uniform distribution of the non zero elements [6]. We have followed the approach of calculating the average number of lines to be accessed $\overline{L}(i,j)$, which is equal to the sum of the access probabilities of the considered lines:

$$\overline{L}(i,j) = \sum_{l=1}^{N_{\text{R}}(j)} P_{\text{R}lN_K}(i,j) + \sum_{l=1}^{N_{\text{L}}(i)} P_{\text{L}lN_K}(i,j) \quad (6)$$

From this average value the corresponding area vector $P_{\text{self X}}(i,j)$ is calculated as $S_{\text{S}}(\overline{L}(i,j)L_s N_K)$. The final expression of the number of misses on vector X is

$$M_{\text{X}} = \frac{N}{L_s} \sum_{j=1}^{W+L_s-1} p_j (1 - P_{\text{hit X}}(j)) \quad (7)$$

4 Modeling the Sparse Matrix-Dense Matrix Product

This product has three possible orderings considering a CRS storage scheme. If I is the variable that designates the rows of the sparse and product matrices, J is associated to the columns of the product and the dense matrices and K refers to the dimension common to the sparse and the dense matrices, the three possibilities are JIK, IKJ and IJK, being the first variable the one corresponding to the outer loop.

Once the sparse matrix-vector product has been introduced in the previous section, the modeling extension to the JIK ordering is direct, as it consists in making a product of this type for each column in the dense matrix. The only difference is that the first access to a line of vectors A, C and R during the process of each column but the first one does not result in a sure miss. Then we must consider separate

Matrix	Order	N_{nz}	W	K	σ	Dev.
bcsstk03	112	640	15	1	30.49	-2.40
bcsstk03	112	640	15	2	1.15	-0.16
bcsstk03	112	640	15	4	0.00	-0.12
bcsstk06	420	7860	93	1	3.09	1.48
bcsstk06	420	7860	93	2	0.77	1.03
bcsstk06	420	7860	93	4	0.20	1.01
bcsstk10	1086	22070	75	1	3.90	0.83
bcsstk10	1086	22070	75	2	1.31	-0.44
bcsstk10	1086	22070	75	4	0.55	-0.48
cry10000	10000	49699	201	1	4.11	1.45
cry10000	10000	49699	201	2	0.69	-0.01
cry10000	10000	49699	201	4	0.68	-0.01

Table 2. Deviation of the model for the sparse matrix-vector product of some example matrices on a 1Kw cache with $L_{\text{s}} = 4$

hit probabilities for this three vectors. Due to space limitations we do not explain here the modeling for the `IKJ` and `IJK` orderings (see [5] for a more detailed explanation and validation results).

5. Validation of the model

The code for this kernel was rewritten replacing the accesses to memory by functions that write the referenced address to a trace file which is later processed by dineroIII, one of the WARTS tools [8]. Table 2 shows the average errors of the model using some matrices of the Harwell-Boeing and NEP collections with an approximate uniform distribution of the entries within each diagonal of their bands. C_{s} is expressed in Kwords and L_{s} in words. For each matrix and cache 50 simulations were performed changing the initial addresses of the involved data structures. The σ column gives the average deviation of the number of measured misses in the simulations as a percentage of the average number of measured misses, while the Dev column gives the deviation of the model prediction with respect to this average measured value.

The table shows that the model is not so sensitive to the variation in the starting addresses of the vectors and matrices as the simulations, and although there is a certain deviation with respect to the real number of misses, we think that it is acceptable.

6. Conclusions

Cache behavior estimation by means of a probabilistic modeling has been done for a significant set of real matrices. The validation has shown both a reasonable accuracy of the model and an advantage over the simulations, consisting in its independence of the initial address of the data structures.

On the other hand the model has several other advantages over the use of traces such as giving the number of misses for each vector and matrix or requiring much shorter CPU times. Besides the processing time and the space in disk of the trace grow linearly with the number of accesses so it is proportional to N_{nz}, while the model execution time is rather independent of the input parameters.

We think the type of modeling we are introducing can be applied in two different directions for improving cache performance: introduction of possible architectural cache parameters modifications, and a framework to be inserted in future compilation tools dealing with irregular access patterns.

References

[1] Z. Bai, D. Day, J. Demmel, and J. Dongarra. A test matrix collection for non-Hermitian eigenvalue problems, release 1.0, Sep 1996.

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

[4] I. S. Duff, R. G. Grimes, and J. G. Lewis. User's guide for the harwell-boeing sparse matrix collection. Technical report, CERFACS, Oct 1992.

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

[6] B. B. Fraguela, R. Doallo, and E. L. Zapata. Modeling set associative caches behaviour for irregular computations. In *Proc. ACM SIGMETRICS*, June 1998. to appear.

[7] 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.

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

[9] R. W. Quong. Expected i-cache miss rates via the gap model. In *Proc. 21st Int'l. Symp. on Computer Architecture (ISCA'94)*, pages 372–383, April 1994.

[10] 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.