Automatic Analytical Modeling for the Estimation of Cache Misses *

Basilio B. Fraguela, Ramón Doallo
Universidade da Coruña
Dept. de Electrónica e Sistemas
Facultade de Informática
Campus de Elviña, 15071 A Coruña, Spain
{basilio,doallo}@udc.es

Emilio L. Zapata
Universidad de Málaga
Dept. de Arquitectura de Computadores
Complejo Politécnico, C. Teatinos,
Apdo. 4114, E-29080 Málaga, Spain
ezapata@ac.uma.es

Abstract

Caches play a very important role in the performance of modern computer systems due to the gap between the memory and the processor speed. Among the methods for studying their behavior, the most widely used by now has been trace-driven simulation. Nevertheless, analytical modeling gives more information and requires smaller computation times that allow it to be used in the compilation step to drive automatic optimizations on the code. The traditional drawback of analytical modeling has been its limited precision and the lack of techniques to apply it systematically without user intervention. In this work we present a methodology to build analytical models for codes with regular access patterns. These models can be applied to caches with an arbitrary size, line size and associativity. Their validation through simulations using typical scientific code fragments has proved a good degree of accuracy.

1 Introduction

Nowadays the performance gap between processors and main memory makes an efficient usage of the memory hierarchy necessary for good program performance. Techniques for analyzing cache performance are required in order to propose improvements to the cache configuration or the code structure. The traditional approach, simulation [14], gives little information on the place where misses happen or the reasons of a certain cache behavior, besides requiring computation times which are typically far greater than the execution time of the program to study. An alternative method is the use of the hardware built-in counters that some microprocessor architectures implement [4], [15]. Although this approach requires less computer resources, it is obviously limited to the study of the architectures where these devices exist, and it still gives little information on the cache behavior.

On the other hand, there are analytical models that extract their input parameters from address traces [2], [10], which requires execution or simulation of the program each time any of its parameters changes, just as the two previous approaches. There are very few analytical models based directly on the code [8], [13], and most of them are oriented to direct mapped caches. Besides very little efforts have been made in order to systematize their construction, which is essential to get a real advantage in time in comparison to the use of simulations or built-in counters. The development of this kind of techniques would allow the integration of the analytical modeling in program optimization environments that would be able to improve automatically the memory hierarchy performance. A work in this direction is [9], based on the construction of *Cache Miss Equations* (CMEs), which are a system of linear Diophantine equations where each solution corresponds to a potential cache miss. The type of modeling developed by the authors is suitable for regular access patterns in isolated perfectly nested loops on set associative caches and presents a good degree of accuracy. Nevertheless, it seems to have heavy computing requirements and it does not take into account the probability of hit in the reuse of data structures referenced in previous loops.

In this work we present a new strategy to develop probabilistic analytical models of the cache behavior. This approach makes feasible the automatic generation of equations that estimate the number of misses a given code generates. Our method can be applied to set-associative caches with an LRU replacement policy. It applies to perfectly nested loops, allowing several references per data structure and loops controlled by other loops. Non perfectly nested loops can also be analyzed, although their modeling requires the fulfillment of certain conditions in order to provide good estimations. The model is able to take into ac-

*This work was supported by the Ministry of Education and Science (CICYT) of Spain under project TIC96-1125-C03

count the probability of hit in the reuses of data which have been accessed in loops belonging to different nests using a conservative approach. The automatically developed models have been validated using simulations of typical code loops found in scientific and engineering applications, and have proved to be very accurate in most of the cases. Anyway, they can be applied to any kind of codes where such kind of loops appear.

The basic ideas of our modeling approach will be introduced in the next section. Then the most common constructions in dense codes will be analyzed, and later more complex variants will be considered. First, the modeling of perfectly nested loops with one reference per data structure is explained in Section 3. The model is extended to consider several references per data structure in the following section. Imperfectly nested loops will be considered in Section 5. The validation of our strategy is performed in Section 6. The final section is devoted to conclusions and future work.

2 Modeling concepts

In a K-way associative cache, a miss when accessing a certain line can be obtained in two situations. The first time a given line is referenced, the access results in an intrinsic miss. The remaining accesses will result in misses if and only if K or more lines mapped to the cache set associated with the line have been referenced since the previous access. We call these misses interference misses. In this way, the probability of getting an interference miss is the likelihood of having placed at least K lines in the set of the studied line since its last access. This probability is computed and represented in our model through area vectors. Given a data structure V, we call $S_{\mathrm{V}} = S_{\mathrm{V}_0}, S_{\mathrm{V}_1}, \ldots, S_{\mathrm{V}_K}$ the area vector associated with the accesses to V during a given period of the program execution. The element in the i-th position of the vector stands for the ratio of sets that have received $K-i$ lines of this structure. Only S_{V_0} has a different meaning, as it is the ratio of sets that have received K or more lines. In [7] and [6] we show different expressions that can be used to calculate the vectors associated with typical access patterns.

Two kinds of interference area vectors are considered, depending on the source of the interference. When the interference is generated by the accesses to the data structure whose cache behavior is being modeled, we call them self interferences. On the other hand, cross interferences are those generated by the accesses to the other structures. The calculation of the area vector associated with a given access pattern is different for cross and self interferences, although they have many similarities.

2.1 Common access patterns

As we shall see in Section 3, we devote our analysis to regular loops in which the subscript expressions of array references are affine combinations of the enclosing loop indices. These are by far the most common loops and references in scientific codes, for example about 72% of the loops in SPECfp verify these conditions [8]. This affine indexing scheme gives place to two main regular access patterns: the sequential and the one associated with a number of regions of consecutive elements which have a constant distance between the start of two such regions.

The sequential access to n consecutive words generates a cross interference area vector $S_s(n)$:

$$
\begin{array}{l}
S_{\mathrm{s}_{(K-\lfloor l \rfloor)}}(n) = 1 - (l - \lfloor l \rfloor) \\
S_{\mathrm{s}_{(K-\lfloor l \rfloor-1)}}(n) = l - \lfloor l \rfloor \\
S_{\mathrm{s}_i}(n) = 0 \qquad 0 \leq i < K - \lfloor l \rfloor - 1, K - \lfloor l \rfloor < i \leq K
\end{array} \tag{1}
$$

where $l = \max\{K, (n+L_{\mathrm{s}}-1)/(L_{\mathrm{s}}N_K)\}$ is the maximum of K and the average number of lines placed in each set. In this expression, L_{s} stands for the line size and N_K for the number of cache sets. The term $L_{\mathrm{s}} - 1$ added to n stands for the average extra words brought to the cache in the first and last accessed lines.

As for the second pattern, the estimation of its area vector is performed through a mixed method that involves the calculation of the starting and ending points of each region on the cache. From these data, an average of the number of lines mapped to each cache set is calculated. The corresponding cross interference area vector is obtained from these averages and is represented by expression $S_{\mathrm{r}}(N_{\mathrm{R}}, T_{\mathrm{R}}, L_{\mathrm{R}})$, where N_{R} is the number of regions, T_{R} is the size of each region and L_{R} is the constant stride between two consecutive regions (see [6] for more details).

2.2 Adding area vectors

In general, several data structures may be referenced between two consecutive accesses to a line of the structure we are studying. This implies the need for a mechanism to add the area vectors associated with the references to each of these structures to get the global interference area vector. Given two area vectors $S_{\mathrm{U}} = (S_{\mathrm{U}_0}, S_{\mathrm{U}_1}, \ldots, S_{\mathrm{U}_K})$ and $S_{\mathrm{V}} = (S_{\mathrm{V}_0}, S_{\mathrm{V}_1}, \ldots, S_{\mathrm{V}_K})$, the union area vector $S_{\mathrm{U}} \cup S_{\mathrm{V}}$ that comprises the accesses corresponding to both area vectors is defined as

$$
\begin{array}{l}
(S_{\mathrm{U}} \cup S_{\mathrm{V}})_0 = \sum_{j=0}^{K} \left(S_{\mathrm{U}_j} \sum_{i=0}^{K-j} S_{\mathrm{V}_i} \right) \\
(S_{\mathrm{U}} \cup S_{\mathrm{V}})_i = \sum_{j=i}^{K} S_{\mathrm{U}_j} S_{\mathrm{V}_{(K+i-j)}} \qquad 0 < i \leq K
\end{array} \tag{2}
$$

This method is based on the addition as independent probabilities of the area ratios, which means that it does not take

into account the relative positions of the program data structures in memory. In order to improve the model precision, this fact is taken into account by modifying the original interference area vector of a given structure using the overlapping coefficient of this structure with the one whose behavior is being modeled. Given two structures A and B with sizes $T_{\rm A}$ and $T_{\rm B}$ the overlapping coefficient between them, $\mathrm{Over}(\mathrm{A},\mathrm{B})$, is defined as follows:

$$\mathrm{Over}(\mathrm{A},\mathrm{B}) = \frac{N_K \mathrm{Com}(\mathrm{A},\mathrm{B})}{\min\{N_K, T_{\rm A}/L_{\rm s}\}\min\{N_K, T_{\rm B}/L_{\rm s}\}} \quad (3)$$

where $\mathrm{Com}(\mathrm{A},\mathrm{B})$ is the number of cache sets that may contain lines belonging to both structures. The value of $\mathrm{Com}(\mathrm{A},\mathrm{B})$ depends both on the sizes and the relative positions of these structures. When adding an area vector $S_{\rm B}$ to the global area vector that stands for the interferences with structure A, the following scaling is performed:

$$\begin{aligned} S'_{\mathrm{B}_j} &= \mathrm{Over}(\mathrm{A},\ \mathrm{B}) S_{\mathrm{B}_j},\ \ 0 \le j < K \\ S'_{\mathrm{B}_K} &= 1 - \textstyle\sum_{j=0}^{K-1} S'_{\mathrm{B}_j} \end{aligned} \quad (4)$$

When both the interfering reference and the reference whose behavior is being modeled have a sequential access, a completely different method may be applied. It consists in using a simple algorithm that gives the average number of interference lines. The algorithm takes into account the relative position of the data structures, so it does not require the calculation of the overlapping coefficient. Both this algorithm and the calculation of $\mathrm{Com}(\mathrm{A},\mathrm{B})$ are not included here due to space limitations (see [6]).

In the following sections a strategy to apply systematically these concepts to codes with regular access patterns is presented.

3 Perfectly nested loops

We shall consider accesses to matrices of arbitrary dimensions where each dimension is indexed by an affine function of the enclosing loop variables. We shall not allow the use of any variable in more than one dimension in order to get regular accesses of one of the two kinds we have mentioned in the previous section. As for the loops, they will have a predetermined number of iterations that will be the same for each execution of the loop. Besides, we consider initially that there is only one reference per data structure. Certain kinds of loops depending on other loops, such as the ones that appear when blocking is applied, can also be modeled using our technique with very few changes.

In order to explain the modeling automation, we begin considering a generic set of perfectly nested FORTRAN DO loops, as those in Figure 1, that have been enumerated from 0, beginning from the inner one. We know that a reference to an element of a matrix A which presents the

```
DO IZ=1, NZ, LZ
 ...
 DO I1=1, N1, L1
   DO I0=1, N0, L0
     A(fA1(IA1), ..., fAdA(IAdA))
     ...
     B(fB1(IB1), ..., fBdB(IBdB))
     ...
   END DO
   ...
   C(fC1(IC1), ..., fCdC(ICdC))
   ...
 END DO
 ...
END DO
```

Figure 1. General perfectly nested DO loops in a dense code.

form A($f_{\rm A1}$($I_{\rm A1}$), ..., $f_{\rm AdA}$($I_{\rm AdA}$)) accesses a memory position that is calculated as:

$$\mathrm{Pos_A} + \sum_{x=1}^{\mathrm{d_A}} \left(f_{\mathrm{A}x}(\mathrm{I}_{\mathrm{A}x}) \prod_{j=1}^{x-1} \mathrm{d}_{\mathrm{A}j} \right) \quad (5)$$

where $\mathrm{Pos_A}$ is the base address of matrix A, $\mathrm{d_A}$ is its number of dimensions and $\mathrm{d}_{\mathrm{A}j}$ is the size of dimension j.

The functions of the indices consist of a constant multiplying one of the variables of the loops enclosing the reference, plus another constant. This means they are of the form:

$$f_{\mathrm{A}x}(\mathrm{I}_{\mathrm{A}x}) = \Delta_{\mathrm{A}x}\mathrm{I}_{\mathrm{A}x} + K_{\mathrm{A}x},\ x = 0, 1, \ldots, \mathrm{d_A} \quad (6)$$

which is by far the most common indexing scheme. Anyway, we obviate the $\Delta_{\mathrm{A}x}$ constants in the following, as their handling would be analogous to that of the steps L_i of the loops (their consideration would just require taking into account that the distance between two consecutive points accessed in dimension x is $\Delta_{\mathrm{A}x} S_{\mathrm{A}x}$ instead of $S_{\mathrm{A}x}$).

3.1 Miss equations

Let $F_i(R,p)$ be the number of misses on reference R at nesting level i considering a miss probability p in the first access to a line of the referenced matrix. To calculate the number of misses on a matrix A generated by this reference, the loops are examined from the inner one containing the reference to the outer one, applying the following rules in each level:

1. If the loop variable is one of the used in the indices of the reference, but not the corresponding to the first dimension, the function that provides the number of misses during the complete execution of the loop of level i as a function of probability p is:

   $$F_i(R,p) = \frac{N_i}{L_i} F_{i-1}(R,p) \quad (7)$$

   This approach is based on the hypothesis that the first dimension of any matrix is greater or equal to the cache line size. This could not hold for caches with large line sizes and matrices with small dimensions, but the conjunction of both factors gives place to very small miss rates in which the error introduced is small and little advantage can be obtained from the application of the automated model.

2. If the loop variable is not any of those used in the indices of the reference, this is a reuse loop for the reference we are studying. In this case the number of misses in this level is estimated as:

   $$F_i(R,p) = F_{i-1}(R,p) + \left(\frac{N_i}{L_i} - 1\right) F_{i-1}(R, S_0(\mathtt{A}, i, 1)) \quad (8)$$

   where $S(\mathtt{Matrix}, i, n)$ is the interference area vector that stands for the lines that may cause interferences with any of the lines of matrix `Matrix` after n iterations of the loop in level i. Function (8) expresses the fact that the first iteration of the loop does not influence the number of misses on matrix `A`, being this value determined by the probability p, which is calculated externally. Nevertheless, in the following iterations the same regions of the matrix are accessed, which means that the interference area vector in the first accesses to each line in this region is composed by the whole set of elements accessed during one iteration of the reuse loop.

3. If the loop variable is the one used in the indexing of the first dimension and $L_i \geq L_s$ we proceed as in case 1, as each reference will take place on a different line. Otherwise, we have:

   $$F_i(R,p) = \frac{N_i}{L_s} F_{i-1}(R,p) + \left(\frac{N_i}{L_i} - \frac{N_i}{L_s}\right) F_{i-1}(R, S_0(\mathtt{A}, i, 1)) \quad (9)$$

   The miss probability in the first access to each referenced line is given by the probability p, externally calculated. The remaining accesses take place on lines referenced in the previous iteration of the loop of level i, so the interference area vector is the corresponding to one iteration of this loop.

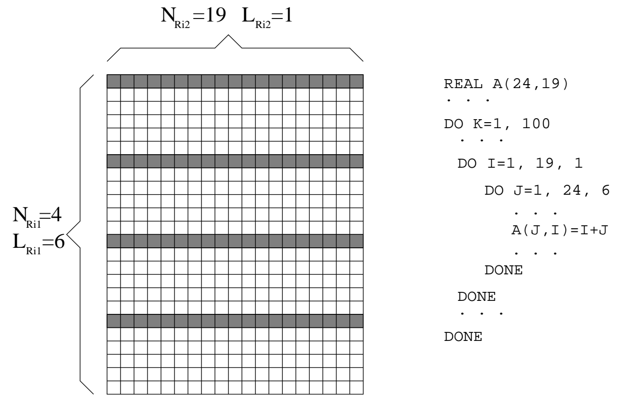

Figure 2. Areas accessed with stride 3 for the columns in a bidimensional matrix `A` during one iteration of the `K` loop.

In the first level containing the reference $F_{i-1}(R,p)$, the number of misses caused by this reference in the closer lower level, is considered to be p. Once calculated the formula for the outer loop, the number of misses is calculated as $F_z(R,1)$ (see Figure 1), which assumes that there are no portions of the matrix in the cache when the code execution begins.

3.2 Interference area vectors calculation

The calculation of the interference area vectors $S(\mathtt{Matrix}, i, n)$ is performed in an automated way by analyzing the references in such a way that:

- The variable used in the indexing of one dimension $\mathtt{I}_h$, is such that $h < i$, then we assign to this dimension a set of N_h/L_h points with a constant distance of L_h points between each two of them.
- On the other hand, if $h > i$, only one point in this dimension is assigned.
- Finally, if $h = i$, n points with a distance L_i are assigned to the dimension.

There is one exception to this rule. It takes place when the variable associated with the considered dimension $\mathtt{I}_h$ belongs to a loop that is a reuse loop for the reference whose number of misses is to be estimated, and there are no non reuse loops for that reference between levels i (not included) and h. In that case only one point of this dimension is considered.

After this analysis, the area of matrix `A` affected by reference R during an iteration of loop i would consist of $N_{\mathrm{R}i1}$ regions of one word in the first dimension, with a constant

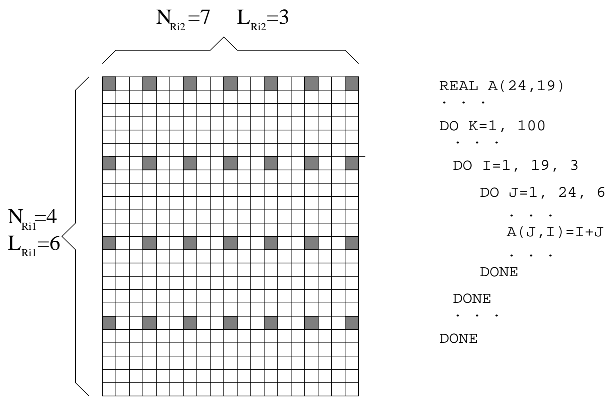

Figure 3. Area accessed with stride 1 for the columns in a bidimensional matrix A during one iteration of the K loop.

stride between each two regions of $L_{\mathrm{R}i1}$ words. In the second dimension, $N_{\mathrm{R}i2}$ elements would have been accessed, with a constant stride between each two consecutive ones of $L_{\mathrm{R}i2}$ groups of $\mathrm{d_{A1}}$ words (size of the first dimension), and so on. This area could be represented as:

$$\mathcal{R}_{\mathrm{R}i} = ((N_{\mathrm{R}i1}, L_{\mathrm{R}i1}), (N_{\mathrm{R}i2}, L_{\mathrm{R}i2}), \dots, (N_{\mathrm{R}i\mathrm{d_A}}, L_{\mathrm{R}i\mathrm{d_A}})) \quad (10)$$

Figures 2 and 3 depict this idea for a bidimensional matrix. This area will typically have the shape of a sequential access or an access to groups of consecutive elements separated by a constant stride, which is what happens in Figure 3 (the stride in Figure 2 is not constant). Both accesses have already been modeled for the calculation of their corresponding cross or self interference area vectors, as explained in Section 2. As an example, the area shown in Figure 3 would be modeled by interference area vector $S_{\mathrm{r}}(N_{\mathrm{R}} = 76, T_{\mathrm{R}} = 1, L_{\mathrm{R}} = 6)$. In this expression we find that:

- The value N_{R}, the number of regions, is obtained as the product of the $N_{\mathrm{R}ij}$ for $j = 1, 2, \dots, \mathrm{d_A}$.
- The size T_{R} of each region is one word, as we are considering only one reference.
- The stride between regions L_{R} is given by the value of $L_{\mathrm{R}ij}$ for the smaller j such that $N_{\mathrm{R}ij} \neq 1$ multiplied by the size of the dimensions lower to j.

If $L_{\mathrm{R}} = 1$ the completely analytical modeling associated with the sequential access to N_{R} words would be applied, which is given by $S_s(N_{\mathrm{R}})$ (see Section 2.1). On the other hand, in the case that the area does not correspond to a regular region as the ones we have studied, the area vector would be calculated through a simulation of the access.

4 Multiple references to one data structure

When there are several references to a given vector or matrix, they usually differ only in one constant added in one of the dimensions of the references. We first consider this case, this is, references of the type `A(..., Ii+Ki1, ...)`, `A(..., Ii+Ki2, ...)` keeping the expressions for the remaining dimensions equal. An approach for modeling the case that there are differences in several dimensions will be introduced later.

4.1 Miss equations for multiple references

In this case the $\Re$ references are sorted in descending order of the value of the constant K_{ij} ($K_{i1} > K_{i2} > \cdots > K_{i\Re}$). The first one is modeled in the way explained in Section 3. For each of the following references, all the loops are modeled in the same way but the one associated with $\mathtt{I}_i$. In this case a procedure that depends on the value of $\delta = K_{i(j-1)} - K_{ij}$ is applied. This procedure is explained in the subsections below.

The references differ in the indexing of the first dimension Let $\epsilon = \mathrm{mcd}(L_{\mathrm{s}}, L_i)$, then the access performed in this loop can be considered as $G = (N_i\epsilon)/(L_i L_{\mathrm{s}})$ groups of L_i/ϵ lines in which each reference accesses L_{s}/ϵ different positions. These positions can be classified in the following way:

- $N_{\mathrm{same}} = \max\{0, \frac{L_{\mathrm{s}}-\delta-q+\epsilon-1}{\epsilon}\}$ positions that are accessed in the same iteration both by the present reference and the one analyzed in the previous step. In this expression q stands for the address of the first access modulus ϵ. For these accesses the miss probability depends on the accesses between both references in the same iteration.

  In the notation used so far only complete iterations of the loops had been considered for the area vectors calculation. Here two references R_1 and R_2 located in a loop at level i are considered, and the area vector $\breve{S}(R_1, R_2)$ associated with the accesses between them is to be estimated.

- $N_{\mathrm{self}} = \max\{0, \frac{L_{\mathrm{s}}-L_i-p-\max\{N_{\mathrm{same}}-1,0\}\epsilon+\epsilon-1}{\epsilon}\}$ positions in which the analyzed reference accesses a line that has been brought to the cache in the previous iteration, but which has not been accessed in the present iteration by the preceding reference, this is, the one corresponding to the immediately greater value of the constant. The area vector corresponding to the accesses to these positions is estimated as the one associated with one iteration of loop i.

- In the remaining $L_\mathrm{s}/\epsilon - N_\mathrm{same} - N_\mathrm{self}$ positions the line has been accessed by the preceding reference $\delta/\max\{L_i, L_\mathrm{s}\}$ iterations ago of the loop.

On the other hand, there are $\delta/\max\{L_i, L_\mathrm{s}\}$ positions that are never accessed by the preceding reference. As a result, the number of misses on them can be estimated as $F_{i-1}(R_j, p)$.

Putting it all together, the number of misses generated by reference R_j in loop i is:

$$
\begin{aligned}
F_i(R_j, p) = & N_\mathrm{same}\breve{S}(R_{j-1}, R_j) + N_\mathrm{self}S_0(\mathtt{A}, i, 1) + \\
& \left(\frac{L_\mathrm{s}}{\epsilon} - N_\mathrm{same} - N_\mathrm{self}\right) S_0\left(\mathtt{A}, i, \frac{\delta}{\max\{L_i, L_\mathrm{s}\}}\right) + \\
& \frac{\delta}{\max\{L_i, L_\mathrm{s}\}} F_{i-1}(R_j, p)
\end{aligned}
\tag{11}
$$

The references differ in the indexing of any other dimension In this case there are two possibilities:

- If $\delta \bmod L_i = 0$, the reference accesses the same line as the preceding reference exactly δ/L_i iterations ago. This means the formula of the number of misses is:

$$
\begin{aligned}
F_i(R_j, p) = & \frac{N_i - \delta}{L_i} F_{i-1}(R_j, S_0(\mathtt{A}, i, \delta/L_i)) + \\
& \frac{\delta}{L_i} F_{i-1}(R_j, p)
\end{aligned}
\tag{12}
$$

- Otherwise these two references never access the same line, so the formula is:

$$
F_i(R_j, p) = \frac{N_i}{L_i} F_{i-1}(R_j, p) \tag{13}
$$

The error rate can be reduced when $\mathrm{d_{A1}}$, the first dimension of the matrix, is smaller than the line size L_s, and so is $\delta\mathrm{d_{A1}}$ (the stride between positions accessed by the references). The miss equation for the loop k controlling the first dimension is modified as follows:

$$
\begin{aligned}
F_k(R_j, p) = & \frac{N_j}{L_\mathrm{s}} F_{k-1}(R_j, p) + \\
& \left(\frac{N_j}{L_j} - \frac{N_j}{L_\mathrm{s}}\right) \frac{\delta\mathrm{d_{A1}}}{L_\mathrm{s}} F_{k-1}(R_j, S_0(\mathtt{A}, j, 1))
\end{aligned}
\tag{14}
$$

4.2 References that differ in several dimensions

If there are several dimensions with differences in the constants added in the functions that index them, the references are sorted in descending order of the position they

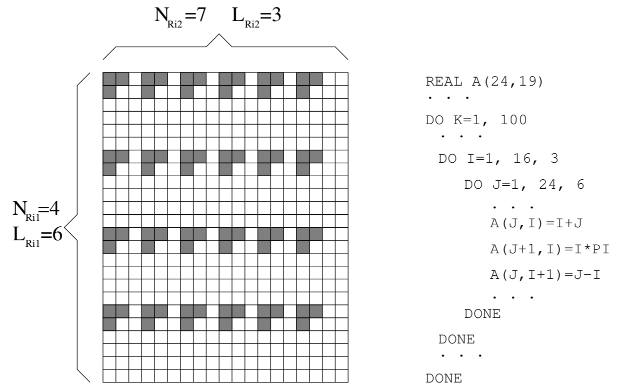

Figure 4. Portions of a bidimensional matrix A with multiple references that have been accessed during one iteration of the K loop.

access, as in Section 4.1. In the same way, the first reference is processed as if it were the only one. The remaining references are processed applying the formulae explained in the previous sections and only the last one of the dimensions that change with respect to the preceding reference is considered. This strategy simplifies the treatment of the problem allowing good estimations when the variations introduced by the differences in the smaller dimensions are much smaller than those produced by the greater dimension with differences, which is the most usual.

Area vectors calculation The existence of multiple references to a given data structure makes more complex the representation of the area they can access, as the pattern is much less regular, as Figure 4 shows. Fortunately, patterns as the one in the figure are not the most common, and a simple extension to the notation introduced in Section 3.2 allows to represent areas generated by accesses belonging to multiple references. It consists in adding a new parameter: the number $T_{\mathrm{R}i}$ of consecutive words accessed in the first dimension of the region associated with the pattern generated by reference R in nesting level i. We do not consider here the possibility of accessing consecutive points in other dimensions that are separated by other sets of non accessed points, which would give place to a parameter of this kind for each dimension. The reason is that this access would not correspond to any of the ones modeled in this work. Anyway, this possibility could be considered in more general implementations.

On the other hand, when calculating the area vectors, the automatic analyzer must take into account the possible overlapping of the regions accessed by several references to a given data structure. For this reason, once these regions have been calculated the analyzer compares them trying to

merge them. Lines that are accessed by different references should not be taken into account several times as source of interferences. If the conditions we have imposed on the references hold, which are the most common, the analyzer is able to merge them in a regular region, this is, one that can be modeled by the algorithms we have developed. In order to perform this merging, one more parameter is used to describe the region affected by a given reference R: the position Q_{R} of the first word it contains. The merging algorithm is not shown here due to space limitations.

5 Imperfectly nested loops and data reuse

Real applications do not consist of an unique set of perfectly nested loops, but of many sets of loops which may have in each level several other loops. On the other hand, matrices are accessed in different loops, there being a probability of hit in the reuse of previously accessed lines.

Our model obtains the miss equation for each reference in a given level of a nest and calculates the miss probability in the access to a line of the considered matrix in that loop. It considers that there can be references to a given matrix in only one of the loops in a set of perfectly nested loops. In general, when this does not hold for a given code, there is only a real access in the greater of the levels that reference it, taking place the references in the lower levels through accesses to a register. In this way, our hypothesis is true from the point of view of the cache accesses.

The existence of imperfectly nested loops in the level where the reference is located or the lower levels only influences the calculation of the interference area vector. The references inside those loops are analyzed to derive the shape of the regions accessed in the corresponding matrices. Then, an attempt is made to merge regions associated with the same matrix, in a way similar to the one mentioned in the preceding section for different references to the same matrix.

5.1 Modeling references in loops in sequence

When considering upper levels, the studied matrix could be referenced in several of the loops they contain. This situation is depicted in general through the code in Figure 5.

```
DO I_{j+1}=1, N_{j+1}, S_{j+1}
  DO I_{j0}=1, N_{j0}, S_{j0}
    ...
    A(f_{A01}(I_{A01}), ..., f_{A0dA}(I_{A0dA}))
    ...
  END DO
  ...
  DO I_{j1}=1, N_{j1}, S_{j1}
    ...
    A(f_{A11}(I_{A11}), ..., f_{A1dA}(I_{A1dA}))
    ...
  END DO
  ...
  DO I_{jn}=1, N_{jn}, S_{jn}
    ...
    A(f_{An1}(I_{An1}), ..., f_{AndA}(I_{AndA}))
    ...
  END DO
END DO
```

Figure 5. Code with references to a given matrix in a set of imperfectly nested loops.

In this case, taking into account that the analysis is performed beginning with the inner loop containing the references, the approach described so far is applied to calculate the number of misses on matrix A in the loops jk with $k = 0, 1, \ldots, n$. As explained in Section 3.1, the miss equation for reference R in nesting level i, $F_i(R,p)$, depends on the parameters for that loop (N_i and L_i), the expression of the number of misses in the lower level, $F_{i-1}(R,p)$, and the interference area vector for the data structure associated with one iteration of loop i, $S(\mathtt{A}, i, 1)$. In this case, as level $j+1$ does not contain only one loop of level j, but n, the calculation of F_{j+1} requires different considerations depending on the loop each reference belongs to.

For the references R inside loops jk where $k > 0$, the following constant is chosen as value for the number of misses they generate during a complete iteration of the loop of level j:

$$F_j(R,p) = F_{jk}(R, (S(\mathtt{A}, \mathrm{Nr}(\mathtt{A}, j(k-1)), \mathrm{It}(\mathrm{Nr}(\mathtt{A}, j(k-1)))) \cup \dot{S}(j(k-1), jk))_0) \quad (15)$$

where $\mathrm{It}(u) = N_u/L_u$ is the number of iterations of loop u and $\mathrm{Nr}(\mathtt{A}, u)$ is the loop of the outer level inside loop u (or loop u) that is not a reuse loop for the accesses to matrix A.

The number of misses is given by the expression associated with the corresponding loop estimating the miss probability in the access to a line of the data structure when the loop starts.

The simplest estimation is the addition of the interference generated during the execution of the preceding loop, given by expression $S(\mathtt{A}, \mathrm{Nr}(\mathtt{A}, j(k-1)), \mathrm{It}(\mathrm{Nr}(\mathtt{A}, j(k-1))))$, and the one for the loops there may be between $j(k-1)$ and jk loops in which this data structure is not referenced. The notation $\dot{S}(j(k-1), jk)$ is introduced to stand for the cross interference area vector generated by the

```
 1 DO J2=1, N, BJ
 2  DO K2=1, N, BK
 3   DO J=J2, J2+BJ-1
 4    DO K=K2, K2+BK-1
 5     WB(J-J2+1,K-K2+1)=B(K,J)
 6    ENDDO
 7   ENDDO
 8   DO I=1, N
 9    DO K=K2, K2+BK-1
10     RA=A(I,K)
11     DO J=J2, J2+BJ-1
12      D(I,J)=D(I,J)+
13              WB(J-J2+1,K-K2+1)*RA
14     ENDDO
15    ENDDO
16   ENDDO
17  ENDDO
18 ENDDO
```

Figure 6. Dense matrix-dense matrix product with blocking and a copy with transposition of the block.

accesses there may be between the end of the execution of loop $j(k-1)$ and the beginning of loop jk.

This approach is good if loop jk contains at least one reuse loop for the references to matrix A or the structures referenced and the order in which these references are performed (mainly those of A) is similar in loops $j(k-1)$ and jk. Otherwise precision can be seriously affected, requiring an analysis similar to the one presented in [7] for the sparse matrix transposition. In this work the hit probability in the reuse is calculated for each line of one vector, studying the accesses that take place between its last access in the preceding loop and its first access in the analyzed loop.

For the references inside loop $j0$, $F_j(R,p)$ cannot be given by a constant, as in being the first loop inside loop $j+1$, it depends on the probabilities of outer or preceding loops. In this case it is just $F_{j0}(R,p)$. Anyway, when calculating the hit probability in the reuses inside loop $j+1$, $S(\mathtt{A}, j+1, 1)$ is estimated as $S(\mathtt{A}, \mathrm{Nr}(\mathtt{A}, jn), \mathrm{It}(\mathrm{Nr}(\mathtt{A}, jn))) \cup \dot{S}(jn, j0)$. The reason is that only the interferences generated since the last loop in this level in which data structure A is referenced need to be taken into account. Remarks about the validity of this approach are analogous to the previous case.

As an example, let us take the references to matrix WB in the code in Figure 6. In order to simplify the explanation, we will refer to the references and the loops as WB_i and DO_i, respectively, where i is the line number where they appear in this code. The number

```
DO I=1, N-1
 DO J=1, N-1
  A(J,I)=A(J,I+1)
         +B(J,I)+B(J+1,I)
         +C(I,J)+C(I+1,J)
 ENDDO
ENDDO
```

Figure 7. Stencil code.

of misses on WB_{13} at loop DO_8 is given by the constant $F_{\mathrm{DO}_8}(\mathrm{WB}_{13}, S_0(\mathtt{WB}, \mathrm{DO}_3, \mathrm{BJ}))$. On the other hand, $F_{\mathrm{DO}_3}(\mathrm{WB}_5, p)$ is calculated following the standard rules. Nevertheless, modeling differs for the outer loops (those of lines 1 and 2), as the area vector corresponding to the interferences generated on WB on a complete execution of loop DO_3 is estimated as $S(\mathtt{WB}, \mathrm{DO}_9, \mathrm{BK})$.

6 Validation of automatic modeling

An automatic analyzer based in this model has been implemented which receives the description of the code through function calls. It accepts vectors and bidimensional matrices accessed in perfectly nested loops, and provides functions to allow the modeling of imperfectly nested loops. This is done through the combination of the miss equations and the area vectors corresponding to the perfectly nested loops they may contain, which can be calculated without the user intervention.

This analyzer has been applied to several codes in order to perform its validation. Here we include three of them:

- a dense matrix-dense matrix product code with blocking and imperfectly nested loops. The code performs a copy and transposition of the block to multiply in order to generate more sequential accesses (Figure 6).
- a Stencil code (Figure 7).
- an equation solver kernel extracted from the Ocean program that uses the Gauss-Seidel method (Figure 8).

In order to simplify the validation, we have always used square matrices of order N. For the first code about one thousand combinations of the input parameters were tried, using values of N ranging from 25 to 400. The second code was validated trying 2400 combinations of the input parameters using matrices of sizes 200 and 400. For these two codes twenty simulations were performed for each combination of the input parameters changing the value of the data structure base addresses using a random generator. We measure the error Δ of the model for each combination as the average of the absolute error in the estimation of the number

```
E=0
WHILE (E.EQ.0)
 D=0
 DO I=2, N-1
  DO J=2, N-1
   T = A(I,J)
   A(I,J)=0.2*(T+A(I,J-1)+A(I-1,J)
                 +A(I,J+1)+A(I+1,J))
   D=D+ABS(A(I,J)-T)
  ENDDO
 ENDDO
 IF (D/(N*N)<TOL) E=1
ENDWHILE
```

Figure 8. Equation solver kernel using Gauss-Seidel method.

of misses generated in each of the twenty simulations. This value is expressed as the percentage of the difference between the number of misses predicted by the model and the number of misses measured in the simulations with respect to the latter.

The average errors have turned out to be 2.68% for the Stencil code and 5.96% for the dense matrix-dense matrix optimized product. The typical deviation σ of the number of measured misses in the simulations expressed as a percentage of the average number of measured misses is 2.22% and 11.25%, respectively. This means our average errors are similar or noticeably smaller than the typical deviation of the real number of misses.

As for the last code, it only uses one matrix, which makes it independent on its base address. As a result, only one simulation has been needed for each combination of the input parameters. On the other hand, the `DO` loop modeled by the algorithm is inside a `WHILE` loop whose number of iterations is unknown in advance, so we have modeled the behavior of this code in one iteration of this loop using 7500 combination of the input parameters, getting an average error of 2.8%. Later 2400 simulations were performed using 2, 3, 5, 10 and 25 iterations of the loop, achieving an average error of 3.7%.

This degree of precision has proved to be good enough to drive an optimization process successfully [5]. Tables 1–3 show the validation data of these algorithms for some combinations of the input parameters. Column C_s stands for the cache size in Kwords.

Large errors can be observed in some cases of Table 2. They are exceptional, as we must recall that the average error for this code is under 6%. They are due to the simplification made for the estimation of the overlapping coefficient of the blocked matrices as the average of the over-

Table 1. Validation data for the automated model for some combinations of the input parameters of the Stencil code.

N	C_s	L_s	K	σ	Δ
50	2	8	2	1.06	4.55
100	16	4	1	1.09	1.31
175	32	4	1	6.51	1.38
200	16	4	1	6.02	0.57
200	8	8	1	0.73	11.34
250	4	8	2	10.87	4.23
250	32	16	2	1.81	2.47
300	1	4	1	2.68	2.48
300	16	4	4	0.00	0.44
375	32	4	4	0.00	0.62
400	8	8	2	1.34	2.13
400	32	8	2	0.67	0.16

Table 2. Validation data for the automated model for some combinations of the input parameters of the dense matrix-dense matrix product with blocking and copy with transposition of the block.

N	BJ	BK	C_s	L_s	K	σ	Δ
200	100	100	16	8	2	8.16	6.23
200	100	100	256	16	2	4.80	2.73
200	200	100	32	8	1	8.81	6.88
200	200	100	128	8	2	3.60	2.86
200	200	100	128	32	2	93.42	44.25
200	50	200	16	4	1	5.05	4.62
200	100	200	32	8	2	16.24	12.51
200	100	200	64	16	1	33.54	3.31
400	100	100	16	8	2	6.18	4.48
400	100	100	256	16	2	4.01	4.26
400	200	100	32	8	1	1.50	2.65
400	200	100	128	8	2	15.04	5.82
400	200	100	128	32	2	57.06	44.68
400	50	200	16	4	1	1.52	2.02
400	100	200	32	8	2	7.86	5.55
400	100	200	64	16	1	7.40	7.12

Table 3. Validation data for the automated model for some combinations of the input parameters of the equation solver using the Gauss-Seidel method.

N	Iterations	C_s	L_s	K	Δ
100	25	16	4	1	3.96
175	1	32	4	1	2.27
200	1	16	4	1	1.99
200	5	16	4	1	1.99
250	1	128	8	2	1.60
300	1	16	4	4	1.33
300	10	16	4	4	1.33
400	1	8	8	2	1.24
400	10	8	8	2	1.24
400	1	32	8	2	1.00
400	25	32	8	2	1.00
400	1	256	32	4	1.00
400	25	256	32	4	1.00

lapping coefficients obtained for each block. The right approach would be the calculation of the number of misses for each block in the matrices using its own overlapping coefficient, and finally add these values. We have not done this because the modeling time would grow noticeably and because the errors, although important, are still inferior to the typical deviation of the number of measured misses. The probabilistic nature of our modeling strategy must be taken into account too: small problems do not favor the convergence of this kind of approaches. In fact we see that the large errors appear in situations with a small number of misses. A 2-way associative cache with 128 Kw is large in relation to the 200x200 or 400x400 matrix product using a block of 200x100 elements. In fact, the average number of misses measured in the simulations is respectively 9264 and 53522. This means that the possible errors introduced by the model when driving compiler optimizations on this code would have a negligible influence on the program execution time. Our explanation is backed up by the fact that using the same cache and block configuration with 2000x2000 matrices gives place to an error of only 1.5% for the model estimations. In this case the average number of misses measured increased to about 6.3×10^6. We have not used large data sets in the validation process, for which cache optimization is much more important, due to the heavy computing requirements of their simulation (see last row of Table 4 for this example).

7 Conclusions and future work

A systematic approach that allows the automated generation of cache analytic models has been presented. The resulting equations provide the number of misses for each reference and loop in a given code. The models have proved a good degree of accuracy besides taking into account all the possible sources of misses. Our models are fully parameterizable and are applicable to K-way associative caches with a LRU replacement policy, besides supporting a wider range of codes than previous automated models. For example, they take into account that portions of data structures accessed in previous loops may be in the cache when they are accessed by another set of nested loops.

Although we have only used the total number of misses to validate model, we have seen from its construction that it can provide much more information. Formulae for the number of misses for each reference and loop are build. The interference area vectors give us an idea of the degree of interference generated by each data structure on the references of any other one or itself in each nesting level. All these data give us a much more detailed picture of the cache behavior and its reasons.

As an added benefit, the computation time required by the models is much shorter than the required by simulations. As an example, the times required by the simulation and the modeling of the dense matrix-dense matrix optimized code on an SGI Origin 200 server with R10000 processors at 180MHz are shown in Table 4. We see there is a difference from two to five orders of magnitude, even when we have used a very simplified simulator locally developed. The validity of our simple simulator has been checked using dineroIII, belonging to the WARTS toolset [12].

We are currently working in the integration of our technique in code analysis environments that support it and allow its effective and fast application to real programs. An analyzer based on Polaris [1] that requires no user intervention is being built. It has already been validated with simple codes, obtaining results similar to those in this work. The feasibility and usefulness of the application of our models to drive compiler optimizations has already been proved [5]. In the experiments we have developed they have always helped the compiler to choose optimal or near optimal code transformations.

At the same time, we plan to extend the automatic modeling to indexing schemes other than the affine one (indirections, multiple index affine, etc.). We have already developed a series of algorithms and formulae to calculate interference area vectors associated to several typical irregular access patterns [7]. One possible approach would be the application of pattern matching techniques in a way similar to the one used in [11], [3] to identify the structure of the accesses generated by the references, both in terms of

Table 4. User time in seconds for the simulation and the execution of the automatically generated model for the dense matrix-dense matrix optimized code.

N	BJ	BK	C_s	L_s	K	Siml. time	Modl. time
300	100	100	32	4	2	14.72	0.009
300	150	150	32	4	2	15.30	0.009
300	100	100	32	4	4	13.81	0.005
300	100	100	32	16	2	13.32	0.007
300	100	100	128	4	1	14.61	0.058
300	100	100	256	8	2	13.22	0.046
500	100	100	32	4	2	68.00	0.009
500	250	250	32	4	2	80.74	0.009
500	250	500	1024	16	2	67.16	0.165
2000	200	100	128	32	2	3900	0.029

general shape and quantitative parameters. With these data we could choose the right formulae between those available and apply them to derive the corresponding interference area vector and misses equations.

References

[1] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, B. Pottenger, R. L., and P. Tu. Parallel programming with polaris. *IEEE Computer*, 29(12):78–82, 1996.

[2] 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.

[3] D. R. Chakrabarti, N. Shenoy, A. Choudhary, and P. Banerjee. An efficient uniform runtime scheme for mixed regular-irregular applications. In *Proc. 12th ACM Int'l. Conf. on Supercomputing (ICS'98)*, pages 61–68, July 1998.

[4] Digital Equipment Corporation. *pfm - The 21064 Performance Counter Pseudo-Device*, 1995. DEC OSF/1 Manual pages.

[5] R. Doallo, B. B. Fraguela, and E. L. Zapata. Set associative cache behavior optimization. In *Proc. EuroPar'99*, Sept. 1999. to be published.

[6] B. B. Fraguela. *Analytical Modeling of the Cache Memories Behavior*. PhD thesis, Dept. Electrónica e Sistemas, Univ. da Coruña, March 1999. (in Spanish).

[7] B. B. Fraguela, R. Doallo, and E. L. Zapata. Modeling set associative caches behavior for irregular computations. *ACM Performance Evaluation Review (Proc. SIGMETRICS/PERFORMANCE'98)*, 26(1):192–201, June 1998.

[8] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: An analytical representation of cache misses. In *Proc. 11th ACM Int'l. Conf. on Supercomputing (ICS'97)*, pages 317–324. ACM Press, July 1997.

[9] S. Ghosh, M. Martonosi, and S. Malik. Precise miss analysis for program transformations with caches of arbitrary associativity. In *Proc. 8th ACM Int'l. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, Oct. 1998.

[10] 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.

[11] A. Lain. *Compiler and Runtime Support for Irregular Computation*. PhD thesis, University of Illinois at Urbana-Champaign, 1995.

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

[13] O. Temam, C. Fricker, and W. Jalby. Cache interference phenomena. In *Proc. Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 261–271. ACM Press, May 1994.

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

[15] M. Zagha, B. Larson, S. Turner, and M. Itzkowitz. Performance analysis using the MIPS R10000 performance counters. In ACM, editor, *Proc. Supercomputing '96 Conference*, pages 17–22. ACM Press and IEEE Computer Society Press, Nov. 1996.