Address-Independent Estimation of the Worst-case Memory Performance

Basilio B. Fraguela, Diego Andrade and Ramón Doallo *Member, IEEE*

***Abstract*—Real-time systems are subject to temporal constraints and require a schedulability analysis to ensure that task execution finishes within lower and upper specified bounds. Worst-case memory performance (WCMP) plays a key role in the calculation of the upper bound of the execution time. Data caches complicate the calculation of the WCMP, since their behavior is highly dependent on the sequence of memory addresses accessed, which is often not available. For example, the address of a data structure may not be available at compile-time, and it may change between different executions of the program. We present an analytical model that provides fast, safe and tight estimations of the worst-case memory performance (WCMP) component of the worst-case execution time, using no information about the data base addresses. The address-independent absolute WCMP for codes with references that follow the same access pattern can be very high with respect to the average behavior because those references may be aligned with respect to the cache, thus generating systematic interferences among them. Our model can also provide a tighter and safe estimation for the WCMP for these codes when the user avoids these alignments.**

***Index Terms*—Cache memories, Worst-case analysis, Performance Analysis and Design Aids**

I. Introduction

PROGRAMS with real-time constraints are subject to a schedulability analysis that requires tight and safe calculations of the upper termination bound [1], which is the worst-case execution time (WCET). The WCET guarantees a termination time for a given task to avoid blocking between tasks. The calculation of the WCET is particularly difficult in the presence of data caches [2] because its memory performance component, the worst-case memory performance (WCMP), is highly dependent on the exact sequence of memory addresses accessed by the program. This sequence may not be determinable at compile-time due to the presence of irregular access patterns and/or to the absence of the base address information of one or more data structures. In a static analysis of the cache behavior there are several reasons why the base addresses of the data structures may not be available, such as program modules and libraries compiled separately, stack variables, and dynamically allocated memory. Moreover, base addresses can vary between different executions of the same program, and the number of cache misses can be highly dependent of these base addresses.

As far as we know, [3] was the first analytical model to tackle the prediction of the WCMP in the presence of data caches without requiring the base addresses of the data structures. This model is based on the Probabilistic Miss Equations (PME) model [4] and is applicable to codes with regular computations. It provides fast and precise estimations, which are not absolute maxima for any base address combination, but which reflect realistic WCMP in practice. Thus it is particularly valuable for soft RTS [5] and non-RTS designers interested in knowing a probable WCMP of regular codes when data addresses are unknown at compile time. According to [3], the main reason why it sometimes fails to give a valid WCMP prediction is because it does not consider base addresses combinations that produce alignments with respect to the cache of references that follow the same access pattern. Such alignments, called full alignments in [3], lead these references to collide systematically in the same cache sets, which increases sharply the cache miss rate. They account for a very small percentage of the possible base addresses combinations, and they should be avoided using padding or extra buffering.

This paper improves over [3] in five ways. First, it identifies the source of the unsafeness of the predictions of [3] and provides a general model for it in Section V-B4, thus enabling a totally safe WCMP prediction. In short, the underpredictions stemmed from not considering the worst-case overlapping (i.e. mapping to the same cache sets) between the data to be reused and the data that can interfere with those reuses. The problem of the full alignments discussed in [3] is a specific instance of this more general and probable one, which we call worst-case overlapping. The second contribution is the identification and modular modeling of this difference in a separate modeling stage, which allows to provide two kinds of WCMP prediction: the absolute and the conditioned one. The absolute WCMP predicted is always safe, and it is tight with respect to the actual WCMP observed. Now, the potential absolute WCMP for codes with references that follow the same access pattern and caches with small associativity can be very high. The reason is that for some combinations of base addresses those references could be aligned with respect to the cache (full alignment), thus colliding systematically in the same cache sets. The conditioned WCMP prediction of the model is a safe and tighter upper bound of the memory performance when the user has taken steps to avoid full alignments. The third and fourth contributions are the description of the modeling of worst-case reuses among different references, and the modeling of strided accesses. A last contribution is a more extensive validation than the one in [3] using more codes and considering more alignments of the data structures; in fact, all the alignments for most codes.

This paper is organized as follows. Section II describes the scope of application of the model presented in this paper. Sec-

The authors are with the University of A Coruña, Spain, e-mail: {basilio.fraguela, diego.andrade, ramon.doallo}@udc.es

tion III introduces the PME model. Sections IV and V describe the two main tasks of the model, namely, the construction of formulas with worst-case reuse distances, and the estimation of the worst-case miss rate associated to a reuse distance, respectively. Section VI shows the experimental results, Section VII is devoted to the related work and Section VIII summarizes the conclusions.

II. Scope of Application

The inputs of the model are the cache configuration and the source code to analyze. It supports any associative cache with a Least Recently Used (LRU) replacement policy. As for codes, their loops can be nested in any arbitrary way, and the references to memory can be found in any nesting level. References must follow regular access patterns defined by affine functions of the loop indexes. Also, the number of iterations of each loop, or at least an upper bound, must be known to perform the analysis. This number of iterations must be constant and the same in each execution of the loop. If this information cannot be inferred from the code even after applying standard compiler techniques such as inlining and constant propagation, it can be provided by the user, for example through compiler directives, or obtained through profiling. All the loops are normalized to have step 1 before the analysis, and they are numbered according to their nesting level, 0 being the outermost one.

The only conditionals allowed inside the portions of code analyzed in this paper are those that only guard accesses to registers or to the latest data item accessed before the branch, so that the data-dependent flow cannot modify the cache behavior. These restrictions are common in the compile-time analytical models of the data cache behavior [8][9][10], and they still allow the modeling of complete real world benchmarks or at least their most time consuming routines, as seen in [4]. Inlining, either symbolic or actual, allowed the PME model in which ours is based to model inter-routine cache effects in [4] and can be applied in the same way to the extensions proposed here.

Applications that exhibit irregular access patterns, such as those arising from the usage of pointers, indirections or more complex conditional statements, can be made analyzable for the model by locking the cache before such patterns arise and unlocking it after them. This technique is commonly used to enable cache predictability, particularly for enabling a tight computation of the WCET [10]. Another popular technique equally complementary of this model is software cache partitioning [11], which divides the cache into disjoint partitions, which are assigned to different concurrent tasks. This facilitates the model of multitasking environments, since the model can analyze the behavior of the program executed by each task independently considering only its cache partition.

III. The Probabilistic Miss Equations model

Contrary to other models, the Probabilistic Miss Equations (PME) model [4] estimates separately the number of misses generated by each static reference R in the code. This is a very interesting property, as this allows to identify hot spots and references. To achieve this, the behavior of each reference is studied in each loop that encloses it, beginning in the innermost one and proceeding outwards. In each loop the model builds a formula for each reference called Probabilistic Miss Equation (PME). Definitions of new or frequently used terms such as this one will be introduced throughout the paper.

Definition III.1. *A* **Probabilistic Miss Equation** F_{Ri} *is an estimator of the number of misses that a static reference* R *generates during an execution of the loop at nesting level* i *that encloses* R*.*

The estimator simply counts the number of accesses generated by the reference during an isolated execution of the loop, and multiplies each one of them by an estimation of the probability it results in a miss. The accesses generated by a reference during the execution of a loop can be classified in two groups: first-time and non first-time accesses to a line during the execution of the loop. Non-first accesses to a line can result in cache hits if the lines brought to the cache since the immediately previous access to their line have not evicted it from the cache. Thus non-first accesses are potential successful reuses of a line in the cache. The probability these reuse attempts result in misses depend on the cache footprint of the memory regions accessed during their reuse distance.

Definition III.2. *The* **reuse distance** *of a reuse attempt on a line is the portion of code executed since the immediately previous access to that line.*

Definition III.3. *A* **memory region** *is the set of memory positions accessed by a reference during a reuse distance.*

Definition III.4. *The* **cache footprint** *of a memory region is the distribution on the cache of the lines that comprise the memory region.*

Definition III.5. *The* **miss probability** *associated to a reuse distance is the probability that the cache footprint of the memory regions accessed during this distance have evicted from the cache the line whose potential reuse is being analyzed. This is the probability the reuse attempt results in a miss.*

Therefore the number of misses generated by these non-first accesses within the loop is given by a summation with one term per reuse distance found. Each term is the product of the number of reuse attempts that have that reuse distance by the corresponding miss probability. As for the first-time accesses, they need not be misses necessarily. An access that is the first one to a line during one execution of a loop cannot exploit a reuse distance within that loop, but it could have a reuse distance associated to outer or preceding loops. For this reason a PME F_{Ri} is always written as a function of the reuse distance RD that the first-time accesses of R in the loop could exploit. This reuse distance is found when outer or preceding loops are analyzed, as we will see in the following sections. Altogether, the general form of a PME is

$$F_{Ri}(\mathrm{RD}) = \mathrm{FirstTimeAccs} \cdot MissP(\mathrm{RD}) + \sum_{j=1}^{\mathrm{NRD}} \mathrm{NumAccs}_j \cdot MissP(\mathrm{RD}_j) \quad (1)$$

where FirstTimeAccs is the number of first-time accesses of R within loop i, $MissP(\mathrm{RD})$ the miss probability for reuse distance RD, NRD is the number of different reuse distances and $\mathrm{NumAccs}_j$ the number of accesses that can enjoy the reuse distance RD_j within the loop.

The PME model [4] seeks to estimate the average number of misses generated by a code during its execution. Thus it relies on probabilities and its estimations are also averages in general. In order to compute a safe upper limit of the number of misses the model must be modified to (1) compute worst-case reuse distances, i.e., ensure the actual distance for a reuse will be shorter than or equal to the one used in the estimation; and (2) compute worst-case miss rates instead of miss probabilities for each reuse distance. The first task is accomplished during the construction of the PMEs, explained in Section IV and Appendix A. Section V is devoted to the second task.

IV. Worst-case PME construction

As the preceding Section explains, the model builds a PME F_{Ri} for each reference R and loop at nesting level i that encloses it. The PMEs are built beginning with the one for the innermost loop that contains R and proceeding outwards up to the outermost loop. The PME for each nesting level captures all the reuse distances within that level. In particular, the construction of F_{Ri} discovers the reuse distances that are specifically associated to this loop. The PME is written in terms of the PME $F_{R(i+1)}$ for the immediately inner level, which carries the reuse distances within the inner loops.

Before building F_{Ri}, the model verifies whether R can exploit a reuse with respect to other references whose reuse distance is associated to this loop. The procedure is described in Appendix A-A, including a brief description of the construction of the worst-case PME to model reuse among different references. If there is no such reuse, the PME F_{Ri} is built taking into account the potential reuses of R with respect to its own accesses. The procedure is described in detail in Section IV-A. Once F_{Ri} has been built, the model verifies whether there are preceding loop nests at nesting level i where there are references from which R could exploit reuse. If this is the case, F_{Ri} is modified to account for such reuses. The steps to model the reuse among different loop nests are explained in Appendix A-B.

A. Worst-case PME without reuse from other references

A key component required to derive this PME F_{Ri} is the stride of reference R with respect to loop i.

Definition IV.1. *The* **stride** *of a reference with respect to a loop is the distance between the positions accessed by the reference in two consecutive iterations of the loop.*

Let us recall that loops are normalized to have step 1. Consider loop i has N_i iterations and its control variable is $\mathtt{I}_i$. Also, since affine functions $\alpha_{Rj}\mathtt{I}_j+\delta_{Rj}$ are used for indexing each dimension j in R, the reference has a constant stride S_{Ri} with respect to any enclosing loop i which is computed as

$$S_{Ri} = \begin{cases} 0 & \text{if } \mathtt{I}_i \text{ does not index } R \\ |\alpha_{Rj}| \cdot D_{aj} & \text{if } \mathtt{I}_i \text{ indexes dimension } j \text{ of } R \end{cases} \quad (2)$$

TABLE I
Notation used.

AV	Area vector
$\mathrm{AV_s}$	AV associated to a $\mathrm{Reg_s}(M)$ region
$\mathrm{AV_r}$	AV associated to a $\mathrm{Reg_r}(groups, size, stride)$ region
$\mathrm{AV_{Reg}}$	AV associated to region `Reg`
α_{Rj}	Constant that multiplies a loop variable in the index of dimension j of reference R
C_s	Cache size
C_sk	Size of a cache way (C_s/k)
$d_{\mathrm{a}j}$	Size of the j-th dimension of array `a`
$D_{\mathrm{a}j}$	Cumulative size of the j-th dimension of the array `a`
δ_{Rj}	Constant added in the index of dimension j of reference R
F_{Ri}	PME for reference R at nesting level i
$\mathrm{Iter}_i(n)$	A reuse distance of n iterations of the loop at nesting level i
k	Associativity of the cache
L_{Ri}	# of lines that an iteration point of reference R at level $i+1$ accesses during the whole execution of the loop at level i
L_s	Line size
N_i	# of iterations of loop at nesting level i
$\mathrm{Reg_s}(M)$	Memory region derived from the access to M consecutive elements
$\mathrm{Reg_r}(groups, size, stride)$	Memory region derived from the access to $groups$ sets of $size$ consecutive elements each, separated by a distance $stride$
S	Number of cache sets
S_{Ri}	Stride of reference R with respect to the loop at level i
Z	Nesting level of the innermost loop that contains a reference

where `a` is the data structure referenced by R and D_{aj} is the cumulative size of dimension j of array `a`. If `a` is a n-dimensional array of size $d_{a1}\times d_{a2}\times\cdots\times d_{an}$ with row-major layout (as in the C language), $D_{aj} = \prod_{i=j+1}^{n} d_{ai}$. Notice that S_{Ri} is non-negative because the absolute value of α_{Rj} is used to compute it. This simplifies the treatment for negative strides. Table I depicts these parameters and others that will be referenced during the explanation of the model. For simplicity, in all the terms and formulas, sizes and strides are expressed in elements of the array whose access is being analyzed rather than in bytes.

Definition IV.2. *An* **iteration vector** *of n loops is a vector of n components where each component is a concrete value of the control variable of each loop.*

Definition IV.3. *An* **iteration point** *of a reference R at level i is an iteration vector of the loops i and the $Z-i$ loops nested inside it that enclose reference R.*

There are $\prod_{j=i}^{Z} N_j$ iteration points at level i. Each one of them defines a different access of R during one execution of the loop at nesting level i. R can access the same element in different iteration points.

In any loop i that encloses R, the reference has a stride S_{Ri} with respect to the loop during N_i iterations. In each one of these iterations, R performs $\prod_{j=i+1}^{Z} N_j$ accesses; one per iteration point at level $i+1$. Let $Addr$ be the arbitrary memory address that R accesses in one of those iteration points in the first iteration of loop i. In the subsequent iterations of

loop i that iteration point will access positions $Addr + S_{Ri}$, $Addr + 2S_{Ri}$, ..., $Addr + (N_i - 1)S_{Ri}$. If L_s is the size of a cache line in elements of the considered access, then in the worst case, this iteration point accesses

$$L_{Ri} = min \left\{ N_i, \left\lceil \frac{S_{Ri}(N_i - 1) + L_s}{L_s} \right\rceil \right\} \quad (3)$$

different lines during the execution of this loop. The actual exact number depends on the alignment of the initial address $Addr$ with a cache line. Expression (3) assumes that always $Addr \bmod L_s = L_s - 1$, that is, the first point is the last one in a line. This is the situation that gives place to the access to more different lines, and thus to fewer reuses within loop i. If $L_{Ri} < N_i$, then there are $N_i - L_{Ri}$ iterations of loop i in which the iteration point is accessing the same line it was accessing in the previous iteration of this loop. It is necessarily the same line, since there is a constant stride S_{Ri} between the addresses accessed by the iteration point in consecutive iterations.

These reasonings hold for every iteration point of R at level $i+1$. Thus there are two kinds of iterations of loop i for every iteration point at level $i+1$:

- In L_{Ri} iterations a new line is accessed, giving place to first-time accesses in this loop. T the potential reuse distance RD for such accesses is unknown in this nesting level. This RD may be found in outer levels or even do not exist, which would turn those accesses into compulsory misses.
- In the other $N_i - L_{Ri}$ iterations every iteration point can enjoy a reuse distance of one iteration of loop i, which we denote by $\mathrm{Iter}_i(1)$.

As a result, the worst-case PME for R at nesting level i, $F_{Ri}(\mathrm{RD})$, can be written as

$$F_{Ri}(\mathrm{RD}) = \begin{cases} L_{Ri} \cdot MissR(\mathrm{RD}) + \\ (N_i - L_{Ri}) \cdot MissR(\mathrm{Iter}_i(1)) & \text{if } i = Z \\ \\ L_{Ri} \cdot F_{R(i+1)}(\mathrm{RD}) + \\ (N_i - L_{Ri}) \cdot F_{R(i+1)}(\mathrm{Iter}_i(1)) & \text{if } i < Z \end{cases} \quad (4)$$

where $MissR$ yields the wort-case miss rate associated to a reuse distance RD. Section V explains how to compute it.

Expression (4) is very intuitive in the innermost loop that contains R (nesting level $i = Z$). The $N_i - L_{Ri}$ accesses that can enjoy a reuse distance $\mathrm{Iter}_i(1)$ generate at worst $(N_i - L_{Ri}) \cdot MissR(\mathrm{Iter}_i(1))$ misses, while the reuse distance RD is yet to be found for the other L_{Ri} accesses. The equation models the worst case situation because (a) the RD for the L_{Ri} first-time accesses to lines in the loop has a miss rate necessarily larger than that of $\mathrm{Iter}_i(1)$, and (b) the expression uses the largest possible value for L_{Ri}, computed according to Eq. (3).

As for the validity of Eq. (4) for the outer loop levels ($i < Z$), let us remember that $F_{R(i+1)}(RD)$ is the number of misses generated by R during the execution of the loop at level $i+1$, that is, one iteration of loop i. The estimator is a function of the reuse distance RD for the first-time accesses of an isolated execution of loop $i+1$. Following the reasonings developed above, each iteration point of R within an execution of loop $i+1$ accesses a new line in L_{Ri} iterations of loop i, while it accesses again the line it accessed in the previous iteration in the other $N_i - L_{Ri}$ iterations. Thus in Eq. (4) $F_{R(i+1)}(RD)$ is multiplied by L_{Ri} to account for L_{Ri} iterations in which there is no reuse in this nesting level, and which give place to have L_{Ri} times more first-time accesses. Regarding the other $N_i - L_{Ri}$ iterations, the number of misses generated by R in each iteration is $F_{R(i+1)}(\mathrm{Iter}_i(1))$ because each first-time access within loop $i+1$ reuses a line with a RD of one iteration of loop i.

```
for(j=0; j<4; j++)   // Level 0
  for(i=0; i<4; i++) // Level 1
    a[j][i] = b[i][j]
```

Fig. 1. Tranposition of a 4×4 matrix

It may be also interesting to unfold Eq. (4) to see how the PME captures all the reuses of the reference within loop i and the ones it contains:

$$F_{Ri}(\mathrm{RD}) = \left(\prod_{j=i}^{Z} L_{Rj} \right) \cdot MissR(\mathrm{RD}) + \sum_{j=i}^{Z} \left(\prod_{l=i}^{j-1} N_l \cdot (N_j - L_{Rj}) \cdot \prod_{l=j+1}^{Z} L_{Rj} \cdot MissR(\mathrm{Iter}_j(1)) \right) \quad (5)$$

The first term relates to the iteration points at level i that cannot exploit reuse within loop i or the loops it contains. They are the product of $L_{Rj}, i \leq j \leq Z$, and their reuse distance is unknown at this point, so their miss rate depends on the input reuse distance RD to this PME. The second term gathers the other $\prod_{j=i}^{Z} N_j - \prod_{j=i}^{Z} L_{Rj}$ iteration points at level i that can exploit reuse within that loop, multiplied by the miss rate that corresponds to their reuse distance. As a result, the second term is a constant. So when in Eq. (4) the PME $F_{R(i+1)}$ is evaluated with different reuse distances in the two terms, this does not affect the iteration points that captured their reuse distance inside loop $i+1$: in the end they are just multiplied by N_i. Only the $\prod_{j=i+1}^{Z} L_{Rj}$ iteration points who have not found their reuse distance inside loop $i+1$ are affected. The outcome for them is that in $N_i - L_{Ri}$ iterations they can enjoy a reuse distance of one iteration of loop i, while in the other L_{Ri} iterations their reuse distance is yet unknown.

Example IV.1. *Let us analyze the behavior of reference $R =$ `b[i][j]` in the matrix transposition of a 4×4 matrix in Fig. 1. A line size of $L_s = 2$ elements is assumed. If matrices are stored by rows, the cumulative sizes of their dimensions are $D_{a1} = D_{b1} = 4$ and $D_{a2} = D_{b2} = 1$ array elements according to the computation of D_{aj} under Eq. (2).*

First, the PME F_{R1} for the innermost loop is derived. The loop variable (`i`) indexes the first dimension of array `b` in this reference, the affine function for the indexing being $1 \cdot \mathtt{i} + 0$. Thus applying Eq. (2), $S_{R1} = |\alpha_{R1}| \cdot D_{b1} = 1 \cdot 4 = 4$. If we replace this value, $L_s = 2$, and the number of iterations of this loop $N_1 = 4$ in Eq. (3) we get $L_{R1} = 4$. Finally, applying Eq. (4) for the case $i = Z$ (because this is the innermost loop, $Z = 1$, that contains the reference), we get

$$F_{R1}(\mathrm{RD}) = 4 \cdot MissR(\mathrm{RD}) + 0 \cdot MissR(\mathrm{Iter}_1(1))$$

This means that the reference accesses different lines in the four iterations of this loop. Let us now derive the PME F_{R0} for

the outermost loop. This loop indexes the second dimension of array `b` *with the function* $1 \cdot \mathtt{j} + 0$*. This way, the stride of the reference with respect to this loop is* $S_{R0} = |\alpha_{R2}| \cdot D_{\mathrm{b}2} = 1 \cdot 1 = 1$ *according to Eq. (2). With this value,* L_{s} *and* $N_0 = 4$ *iterations, Eq. (3) yields* $L_{R0} = 3$*. Eq. (4) for nesting level* $i < Z$ *can then be written as*

$$F_{R0}(\mathrm{RD}) = 3 \cdot F_{R1}(\mathrm{RD}) + 1 \cdot F_{R1}(\mathrm{Iter}_0(1))$$

that is, in the worst case, each iteration point of R *at level 1 access 3 different lines during the whole execution of the outermost loop. When the equations are composed, the final number of misses for reference* `b[i][j]` *can be calculated as*

$$F_{R0}(\mathrm{RD}) = 12 \cdot MissR(\mathrm{RD}) + 4 \cdot MissR(\mathrm{Iter}_0(1))$$

The equation indicates that 12 of the accesses of `b[i][j]` *cannot exploit reuses within the loop. Thus their reuse distance RD depends on previous loop nests. The other 4 accesses try to reuse the line accessed in the previous iteration of loop 0. Thus their miss rate depends on the footprint on the cache of the data accessed during an iteration* $\mathrm{Iter}_0(1)$ ■

V. Estimation of the worst-case miss rate for a reuse distance

The second modification the model needs to make safe WCMP predictions is to ensure it provides an upper bound of the miss rate associated to each reuse distance (RD). The miss rate of a group of cache lines that can enjoy a RD is the ratio of these lines that are evicted from the cache by the data accessed during such RD. Let us recall that in [4], miss probabilities were used instead of miss rates. Probabilities are suitable to estimate the average performance but not the WCMP. A key observation on which the PME model is based is that in a k-way associative cache with LRU replacement policy, a line is evicted during its RD if and only if k or more different lines are placed on its cache set during that RD. Thus, the strategy followed by the model is to compute the distribution of the number of lines placed per cache set by the accesses that take place during the RD. The associated miss rate is then estimated as the worst-case rate of lines to reuse that are mapped to a cache set in which k or more interfering lines have been placed during the RD. Our model follows three steps to calculate the worst-case miss rate associated to a RD: access pattern identification, cache impact estimation and area vectors union. They are discussed now in turn.

A. Access pattern identification

The access patterns followed by the references in a RD are inferred from the indexing functions of these references and the shape of the loops that enclose them. This task is accomplished in three steps:

1) For each reference, the number of points it accesses during the RD in each dimension of the data structure it refers to, and the stride between each two consecutive accesses is computed. The restriction to affine indexing functions in the references considered by the model simplifies this task. This uniquely identifies the memory region accessed by each reference. The access pattern followed by each reference can be derived immediately from this information.
2) When there are several references to the same data structure, the regions they access often overlap. The model tries to merge regions that overlap in order to avoid considering the overlaps several times as source of interference. Concretely, the model only merges the regions accessed by uniformly generated references, that is, references with the same stride for every loop that encloses them, the only difference between them being their initial offset. The algorithm is very simple and it ensures the resulting region is a superset of the actual region accessed, so that the safeness of the process is ensured. It is not described here due to space reasons. Notice that if regions that overlap are not merged, the model overpredicts the interference generated in the reuse distance. As a result, not merging regions that overlap does not endanger safeness, it reduces tightness.
3) From the shape of each resulting memory region, an associated access pattern is inferred. From now on we will use indistinctively the terms memory region and access pattern: the shape of a memory region defines an access pattern, and an access pattern defines the memory region comprised by the elements it accesses. The PME model represents access patterns as functions whose output is a mathematical representation of the footprint of the access on the cache called *area vector*.

   Definition V.1. *An* **area vector** *is a mathematical representation of the cache footprint of a memory region*

   This representation will be described in Section V-B. Two patterns have been found in the codes considered in [4] and in this paper: the sequential access to M elements, denoted as $\mathrm{Reg}_{\mathrm{s}}(M)$, and the access to $groups$ sets of $size$ consecutive elements each, separated by a constant stride $stride$, $\mathrm{Reg}_{\mathrm{r}}(groups, size, stride)$.

This first step of the miss rate estimation process, described in detail in [6], needs no changes for the WCMP prediction.

Example V.1. *The final PME obtained in Example IV.1 requires* $MissR(\mathrm{Iter}_0(1))$*, the worst-case miss rate associated to a reuse distance of one iteration of the outermost loop in Fig. 1. The first step to compute it is to identify the access patterns found in that reuse distance. During one iteration of loop* `j` *the reference* `a[j][i]` *performs 4 accesses in four consecutive memory positions. The reason is that matrices are stored by rows and during this reuse distance there are 4 iterations of the innermost loop, which indexes the columns of the matrix. This way, this access is* $\mathrm{Reg}_{\mathrm{s}}(4)$*, the access to* M *=4 consecutive elements. The innermost loop indexes the row index in* `b[i][j]`*, which gives place to a stride* $S_{R1} = 4$ *of this reference with respect to the iterations of this loop, as Example IV.1 explains. Thus* `b[i][j]` *accesses 4 groups (one per iteration) of a single element each (*$size = 1$*) with a* $stride = S_{R1} = 4$ *between each two consecutive groups. This way, the resulting access pattern is* $\mathrm{Reg}_{\mathrm{r}}(4, 1, 4)$ ■

B. Cache impact quantification

The PME model in [4] analyzes the accesses performed during a reuse distance to estimate the probability they evict from the cache the lines whose reuse is being studied. This happens when these accesses place k or more lines in the cache sets of the lines to reuse, k being the associativity of the cache. To compute this probability, the cache footprint of each access pattern is characterized by a vector V of $k+1$ elements called *area vector* (AV). Each component of an AV is a ratio or probability of interference, that is, it is the probability a line to be reused conflicts in its cache set with a given number of lines from the access pattern that the AV characterizes. This second step of the miss probability estimation process obtains the AV associated to each access pattern found in the reuse distance. The model considers two kinds of area vectors:

- A **Cross interference area vectors (Cross-AVs)** is an AV that represents the impact on the cache of the considered access pattern as viewed by lines not involved in it. In these vectors, the first component, V_0, is the ratio of cache sets where no additional lines are needed to fill the set. That is, it is the ratio of sets where the access pattern has placed k or more lines that compete with the attempts of reuse. V_1 is the ratio of sets which will be filled if just one additional line is placed in the set, i.e., the sets that have received $k-1$ lines. The enumeration would finish with V_k, the ratio of cache sets in which the access pattern placed no lines that compete with the potential reuse.
- A **Self interference area vectors (Self-AVs)** is an AV that represents the impact of the footprint on the probability of reuse for the lines it involves. In these vectors, V_0 is the ratio of lines of the footprint that compete in their cache set with other k or more lines of the footprint. For $i > 0$, V_i is the ratio of lines of the footprint that share their cache set with other $k-i$ lines of the access.

Example V.2. *Let us consider a 2-way associative cache with 4 sets and a reference that has accessed 7 consecutive lines. As a result, three of the four sets contain two of the lines referenced, while the other set contains just one line. The Cross-AV for this access is* $(3/4, 1/4, 0)$*, as 3 out of the 4 sets have received two or more lines from the access; only one set received a single line, and no sets received zero lines. These ratios are conversely the probabilities a randomly chosen set has two or more, one, or zero lines in it, respectively.*

The Self-AV for this access is $(0, 6/7, 1/7)$*. The first component is zero, as none of the lines involved in the access has to compete for its cache set with other two or more lines from the footprint. The second component is the ratio of lines of the footprint that share their cache set with exactly one line (6 out of 7). Finally, according to the third component, only one of the seven lines of the footprint does not share its set with any other line of the footprint. These ratios are conversely the probabilities a randomly chosen line of the footprint has to compete in its set with two or more, one, or no lines, respectively* ■

1) Ensuring safeness moving from probabilities to rates: At this point, a discussion on why the model predicts safe bounds may be in order. This discussion is also useful to explain which are the changes this part of the model in [4] requires and why. The PME model uses the ratios in the AVs as probabilities of interference with each line to be reused. This approach is straightforward in the case of Self-AVs, since if $P\%$ of the lines to reuse fulfill a given property, then there is a $P\%$ probability a given line of this set fulfills this property. Regarding Cross-AVs, the model uses the ratio of *all* the sets in the cache that receive Y lines from that access pattern as probability of interference with Y lines of that access pattern. It would be more accurate to use the ratio of the lines to reuse that experience that competition. Unfortunately, since the base addresses of the data structures are not an input to the model, the relative position in the cache of the lines accessed in different data structures is unknown. As a result, it is impossible to match the mappings of lines from different data structures to sets, and a probabilistic approach has to be followed. Thus, if X of the S cache sets receive Y lines during a reuse distance, the PME model estimates that there is a probability X/S a line to reuse has to compete with Y lines. Now, this probability corresponds to a rate that is computed following a deterministic process and which is defined on a finite number of sets S. As a result, if the S sets of the cache are taken into account in the analysis, which the model does, it is certain that $S \cdot (X/S) = X$ sets will be found that fulfill the property, there being no possible deviation from this value. A similar reasoning can be done for self-interference AVs.

The mapping of access patterns to AVs of the original PME model is not valid for a safe WCMP prediction because it provides average, not worst-case, values for the AVs. The reason is that the access pattern representation of this model does not inform on the relative offset in a line of the element where the access pattern begins, and the AV can vary with this offset. Let us consider for example the access to two consecutive elements: they might be in the same line or in two different lines, depending on the relative offset in a line of the first one. The PME model in [4] provides AVs that are an average of the ones an access pattern would generate with all the line offsets that the first element accessed may have. For the WCMP prediction, only the worst one of these AVs will be considered.

Another important observation is that in the original PME model, the ratios of the AVs of the interfering regions were used as average probabilities of interference with the lines to be reused. That is, not only they were used as probabilities a randomly chosen set receives a number Y of lines, as we have just said: they were also used as probabilities a line to be reused has to compete in its set with those Y lines. The reason for this is the lack of information on the addresses of the data structures. The model cannot know the relative mapping of the lines whose reuse is analyzing with respect to the lines that come from other access patterns. Thus, it takes a probabilistic approach estimating that if X out of S sets received Y lines from that access pattern Reg, then, a line whose reuse is being studied has to compete with Y lines from pattern Reg in its set with probability X/S.

In this paper, and this is a novelty also with respect to [3], this probabilistic approach is dropped in favor of a worst-case deterministic one. Now the worst-case overlapping of the lines to be reused and the interfering lines is computed. This is the mapping that places the largest possible number of lines to reuse in the sets that receive the largest number of lines from the considered interfering access pattern. Then, the ratio of lines of the reuse region that have to compete with a given number of interfering lines in their set in this worst case overlapping is used as component of the modified AV. So, for example, component 0 of the modified AV is the ratio of lines to be reused that are mapped, in the very worst case, to sets that received k or more lines from the interfering access pattern considered. Since the components of the modified AVs are actually (maximal) rates of elements that fulfill a given property out of a finite set (the lines whose reuse is being analyzed) and all the lines are used in the study, the outcome is safe.

Altogether, the algorithms applied in this process remove any uncertainty in the safeness of the prediction. As we have just discussed, every probabilistic approach followed in the original PME model has been replaced by an algorithm that yields safe worst-case rates. This is also backed by the exhaustive evaluation in Section VI.

The modifications to the original model in [4] for the calculation of the cache impact quantification step are now explained in turn. The algorithms to compute the worst-case AV for the sequential and the strided accesses are described in Sections V-B2 and V-B3, respectively. Section V-B4 is devoted to the algorithm to derive the worst-case overlapping between the lines involved in a reuse and the sets affected by each access pattern. The tightly related issue of references that can be aligned with respect to the cache, giving place to systematic conflicts among them, is treated in Section V-B5.

2) Worst-case cache impact estimation for the sequential access: Section V-A explained that one of the access patterns found in regular codes is the access to M consecutive elements, $\mathrm{Reg_s}(M)$. Its worst-case AV is the one that corresponds to the placement of these M consecutive elements that brings more lines to the cache. This happens, as in the case of the L_{Ri} value calculated using Equation 3 introduced in Section IV-A, when the first element of the access pattern is the last element of a line. In this situation, $1+\lceil (M-1)/L_\mathrm{s} \rceil$ lines are brought to the cache. Since a set can hold a maximum of k lines (the associativity), if there are S sets, the average number of lines placed in each set for this worst-case alignment is[1] $l = \min\left\{k, \frac{1+\lceil (M-1)/L_\mathrm{s} \rceil}{S}\right\}$. Based on it, the worst-case area vector $\mathrm{AV_s}(M)$ for this access pattern can be computed as

$$\begin{array}{l} \mathrm{AV}_{\mathrm{s}_{(k-\lfloor l \rfloor)}}(M) = 1-(l-\lfloor l \rfloor) \\ \mathrm{AV}_{\mathrm{s}_{(k-\lfloor l \rfloor-1)}}(M) = l-\lfloor l \rfloor \\ \mathrm{AV}_{\mathrm{s}_i}(M) = 0, \;\; 0 \leq i < k-\lfloor l \rfloor -1, k-\lfloor l \rfloor < i \leq k \end{array} \quad (6)$$

3) Worst-case cache impact estimation for the strided access: The second access pattern identified in Section V-A was the access to $groups$ sets of $size$ consecutive elements each, separated by a constant stride $stride$, $\mathrm{Reg_r}(groups, size, stride)$. Again, the actual AV for the access depends on the relative mapping in a line of its first access. Let us calculate AV^q_r, $0 \leq q < L_\mathrm{s}$, the AV if the first element has relative offset q in a line. In the first step, the positions B_i and E_i corresponding to the beginning and the end of each region of $size$ consecutive elements in a cache way are calculated:

$$\begin{array}{l} B_0 = q \\ B_i = (B_{i-1} + stride) \bmod C_\mathrm{sk}, \; 0 < i \leq groups \\ E_i = (B_i + size - 1) \bmod C_\mathrm{sk}, \; 0 \leq i \leq groups \end{array} \quad (7)$$

where C_s is the cache size, k the degree of associativity, and $C_\mathrm{sk} = C_\mathrm{s}/k$ is the size of a cache way. In two vectors BV and EV of size C_sk, initialized to zero, we add one unit for each position associated with a B_i or an E_i, respectively. They are then analyzed calculating the number of lines of the access corresponding to each group of L_s consecutive positions in these vectors, which correspond to a line of a cache set. For this, we know that the elements in positions sL_s to $(s+1)L_\mathrm{s}-1$ of a cache way are associated to the s-th cache set. The number of lines mapped to set s is calculated as

$$\mathrm{Lines}_s = \left\lfloor \frac{size-1}{C_\mathrm{sk}} \right\rfloor groups + \sum_{i=sL_\mathrm{s}}^{(s+1)L_\mathrm{s}-1} \mathrm{BV}(i) + L_\mathrm{G}(sL_\mathrm{s}) \quad (8)$$

where the first term corresponds to the $\lfloor (size-1)/C_\mathrm{sk} \rfloor$ lines that fall in every cache set for sure from all the $groups$ regions if $size \geq C_\mathrm{sk}$; the second term adds one line for each one of the BV(i) regions that start in the L_s positions of the set; and $L_\mathrm{G}(sL_\mathrm{s})$ are the lines, that come from regions that start before the first position in the cache way associated with this set (sL_s), whose end has not been reached. Its value can be computed as

$$\begin{array}{l} L_\mathrm{G}(0) = \sum_{j=C_\mathrm{sk}-(size-1) \bmod C_\mathrm{sk}}^{C_\mathrm{sk}-1} \mathrm{BV}(j) \\ L_\mathrm{G}(i) = L_\mathrm{G}(i-1) + \mathrm{BV}(i-1) - \mathrm{EV}(i-1), \; 0 < i < C_\mathrm{sk} \end{array} \quad (9)$$

Once Lines_s is computed for the S sets in the cache, the area vector for this offset q is given by

$$\begin{array}{l} \mathrm{AV}^q_{\mathrm{r}_0}(groups, size, stride) = \#\{\mathrm{Lines}_s, 0 \leq s < S / \mathrm{Lines}_s \geq k\}/S \\ \mathrm{AV}^q_{\mathrm{r}_i}(groups, size, stride) = \#\{\mathrm{Lines}_s, 0 \leq s < S / \mathrm{Lines}_s = k-i\}/S, \\ \qquad\qquad\qquad\qquad\qquad\qquad 0 < i \leq k \end{array} \quad (10)$$

where $\#$ is the cardinality of a set. Thus, as the definition of AV implies, component 0 is ($\mathrm{AV}^q_{\mathrm{r}_0}$) the ratio of sets that receive k or more lines, and any other component ($\mathrm{AV}^q_{\mathrm{r}_i}$) is the ratio of lines that receive $k-i$ lines.

There remains the issue of choosing the AV^q_r, $0 \leq q < L_\mathrm{s}$ that will give place to the worst case cache behavior. The answer is that, contrary to the situation of the sequential access, where it is clear that the worst mapping is the one in which more consecutive lines are accessed, for the strided access pattern no a priori approach can be taken. If this were the only access pattern found in a reuse distance, the AV with the largest component 0 would be the worst one. When more patterns appear, which is the usual case, AVs with a smaller component 0 can give place to final higher miss ratios because they may help fill more sets not completely filled by other access patterns when the AVs from all the access patterns are

[1] max was wrongfully used instead of min in Eq. (14) in [4]

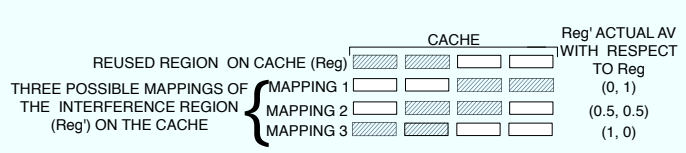

Fig. 2. Miss rate depending on the relative positions of the reused and the interfering memory regions

merged (see Section V-C). Thus the L_s area vectors AV_r^q are passed and tried in the next stages, and the one that gives place to the highest miss rate is chosen in the end. This also implies that if n patterns of this kind appear in a reuse distance, $(L_\mathrm{s})^n$ merges corresponding to all the combinations of offsets, must be tried. Fortunately the area vector union algorithm presented in Section V-C is extremely fast, as all the other ones of this model, so the actual performance penalty is negligible.

4) Worst-case overlapping between reuse and interfering regions: The cache impact quantification stage provides an AV for each interfering memory region `Reg'` found in a reuse distance. Its components are ratios of cache sets that receive a given number Y of lines from this region. The original PME model uses directly these ratios as the ratios of lines of the region whose reuse we are studying, which we will call reuse region `Reg` in what follows, that compete in their cache set with Y lines from `Reg'`. This is a fair average estimation, as the relative placement in the cache of the lines from different memory regions is unknown to the model. Nevertheless, the actual ratios of interference between `Reg` and `Reg'` can be much higher depending on the actual placement, and corresponding overlapping, of both sets of lines on the cache.

Example V.3. *Figure 2 represents the mapping on a one way (direct mapped) cache with four cache sets of a reuse region* `Reg` *and three possible mappings of an interference region* `Reg'`*. Since* `Reg'` *fills two of the four cache sets and leaves empty the other two, its AV is (0.5, 0.5). This suggests an average 50% probability of conflict. However, the actual interference with the lines of* `Reg` *depends on the relative mapping of both sets of lines in the cache and is represented by the AV placed on the right side of each mapping in the figure. In this AV, component 0 is the ratio of lines of* `Reg` *that collide with* `Reg'` *for this mapping, and component 1 is the ratio of lines that do not collide. This way, the first mapping does not interfere with the reuse of* `Reg`*, the second mapping only interferes with the reuse of one of its lines, and the third one avoids both reuses, which leads to 0, 50 and 100% miss rate, respectively* ■

As we see in the previous example, the AVs must be modified to provide worst-case conflict rates. These are ratios of lines of the reuse region `Reg` that collide in their set with a given number of lines from `Reg'`, for the worst-case overlapping of both regions in the cache. This overlapping is the one in which the largest possible number of lines from region `Reg` compete with the largest possible number of lines from `Reg'` in their cache set. That is, it is the situation in which the largest possible number of lines from `Reg` are

```
function worstAV(sim_[n], AV_Reg'[k+1]) {
  interfSets_i = AV_Reg'i * S, 0 ≤ i ≤ k
  lines = Σ_{i=0}^{n-1} sim_i.nsets * sim_i.nlines
  i=j=0
  while (i < k and j < n) {
    tmp = min(interfSets_i, sim_j.nsets)
    AVwc_Reg'_i = AVwc_Reg'_i + (tmp * sim_j.nlines)/lines
    interfSets_i = interfSets_i - tmp
    sim_j.nsets = sim_j.nsets - tmp
    if sim_j.nsets = 0 then j = j + 1
    if interfSets_i = 0 then i = i + 1
  }
  AVwc_Reg'_k = 1 - Σ_{i=0}^{k-1} AVwc_Reg'_i
  return AVwc_Reg'
}
```

Fig. 3. Calculation of worst-case AV

mapped to the $AV_{Reg'_0} \cdot S$ sets that receive k or more lines from `Reg'`, where $AV_{Reg'}$ is the AV for `Reg'` and S is the number of sets in the cache. Component 0 of $AVwc_{Reg'}$, the AV for `Reg'` considering this worst-case alignment, will be then the ratio of lines of `Reg` mapped to these full sets and which thus have to compete with k or more lines from `Reg'`. Once those full sets are exhausted, then the largest possible number of lines from `Reg` are mapped to the $AV_{Reg'_1} \cdot S$ sets that receive $k-1$ lines from `Reg'`, and their ratio on the total number of lines of `Reg` will be $AVwc_{Reg'_1}$, and so on. As we see this algorithm requires $AV_{Reg'}$ as well as the distribution of lines of `Reg` per set in order to match them with the lines from `Reg'`. Unfortunately AV_{Reg} does not suffice for this because its component 0 does not provide the exact number of lines per set, just that there are k or more lines, which is not enough to estimate the ratios. If for example in Fig. 2 region `Reg` had occupied three sets with a single line each, its AV would have been $(0.75, 0.25)$, and since two of them would have collided in the worst case with `Reg'`, $AVwc_{Reg'}$ would have been $(0.66, 0.33)$. Now, if `Reg` had mapped two lines to set 0, another two to set 1 and another line to set 2, its AV would have also been $(0.75, 0.25)$, but since 4 out of its 5 lines could collide with `Reg'` in the worst case, $AVwc_{Reg'}$ would have been $(0.8, 0.2)$. Thus the calculation of $AVwc_{Reg'}$ requires a simulation of the distribution of the lines of `Reg` on the cache whose output is a vector of pairs sim. Each element sim_i of this vector has two components $sim_i.nsets$ and $sim_i.nlines$, such that $nsets$ cache sets received $nlines$ cache lines from region `Reg`. The elements of the vector are sorted in decreasing order of $sim_i.nlines$.

The algorithm `worstAV` in Fig. 3 calculates $AVwc_{Reg'}$ from vector sim and $AV_{Reg'}$ following the procedure explained. In this function, array indexing is 0-based and the input vectors are subindexed with their size. This way, n is the size of vector sim while k stands for the associativity of the cache, $k+1$ being the size of the input AV. The algorithm matches the cache sets that receive the maximum number of lines of the reuse region, as sim is processed rightwards, with the cache sets that receive the maximum number of lines from the interference region, as $AV_{Reg'}$ is also processed rightwards. Each component of the resulting AV, $AVwc_{Reg'}$, represents the ratio of lines of `Reg` that are mapped to a cache set that receives a certain number of lines from region `Reg'`.

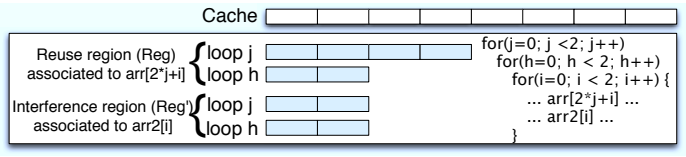

Fig. 4. Mapping of the reuse and the interference region in different nesting levels

a) Tightening the worst-case AV calculation: The output of the `worstAV` algorithm is a safe upper bound of the interference AV, but its tightness can be improved. The algorithm overlaps in the worst possible way the footprint of the reuse region `Reg` with that of the interfering region `Reg'`. But it can be very difficult, or even impossible that `Reg'` always overlaps in the worst possible way with `Reg` in every occurrence of the reuse distance considered.

Example V.4. *Figure 4 shows a ($C_s = 8$, $L_s = 1$, $k = 1$) cache, and a code in which we study the reuse of region* `Reg` *associated to reference $R =$* `arr[2 * j + i]` *in a loop* `h`*, nested inside an outer loop* `j`*. In a single iteration of loop* `h`*, two consecutive lines of* `Reg` *and another two consecutive lines of the region* `Reg'` *associated to reference $R' =$* `arr2[i]` *are accessed. Thus the AV for both of them is $(0.25, 0.75)$. Following the algorithm* `worstAV` *in Fig. 3, the worst-case alignment for their accesses would be the one that puts these lines in the same sets. This yields $AVwc_{Reg'} = (1, 0)$, since with this alignment the lines from* `Reg` *are always replaced by a line from* `Reg'` *before their reuse. Now,* `arr[2 * j + i]` *accesses a different pair of cache sets in each one of the two iterations of loop* `j`*, while* `arr2[i]` *accesses the same pair of cache sets. Thus, if there is full interference during one of the iterations of loop* `j`*, then there can be no interference during the other one. As a result, the maximum interference rate between both regions across all the executions of the loops in this example is actually 50%.* ■

Our example points out that if the reuse region `Reg` changes its relative position in the cache in different iterations of outer loops, it may be impossible that the worst-case overlapping takes place in all the iterations of those loops. An analogous example could be done based on the interfering region, if it were the one to change its position with the iterations of outer loops. Thus tightness can be improved taking into account the freedom of placement of `Reg` and `Reg'` due to the change of their relative position in the cache in different iterations of outer loops. When both references have the same stride with respect to the cache in a given loop i, the relative position of their footprints does not change across the iterations of that loop. This happens when $S_{Ri} \mod C_{\mathrm{sk}} = S_{R'i} \mod C_{\mathrm{sk}}$, S_{Ri} being the stride of the reference R associated to region `Reg` with respect to loop i as defined in Eq. (2), and $C_{\mathrm{sk}} = C_s/k$ the size of a cache way. Thus, the outermost loop where this condition does not hold, which we call loop `zero`, is the one that gives us the largest freedom of relative placement of the regions. In this loop, the minimum portion of the cache affected by the footprint of either R or R' accesses is $r_{zero} = \max\{1 - AV_{Regzero_k}, 1 - AV_{Regzero'_k}\}$, where `Regzero` and `Regzero'` are the regions accessed by R and R' during the whole execution of loop `zero`, respectively. This expression is based on the fact $1 - AV_{Regzero_k}$ is the ratio of cache sets that receive lines from region `Regzero` in the loop. Now, in the loop where the reuse is being studied, the minimum portion of the cache affected by the accesses to either `Reg` or `Reg'` is $r = \max\{1-AV_{Reg_k}, 1-AV_{Reg'_k}\}$. We are interested in the minimum portion of the cache which is affected by any of the regions because when they overlap in the worst possible way, their footprints have the largest possible number of cache sets in common. Thus the ratio of cache sets that receive lines from at least of the two footprints is minimized. The relation between r and r_{zero} gives the freedom of movement of the regions we are considering in loop i inside the minimum area of the cache on which they can be spread due to the iterations in loop `zero`. The appropriate way to take this fact into account is to reduce proportionally by r/r_{zero} all the elements of $AV_{Reg'}$ except the k-th component. Thus tightness is improved by using as input to `worstAV` in Fig. 3 $AV'_{Reg'}$ instead of $AV_{Reg'}$, which is calculated as

$$\begin{aligned} AV'_{Reg'_i} &= AV_{Reg'_i} \cdot r/r_{zero} \qquad 0 \le i < k \\ AV'_{Reg'_k} &= 1 - \textstyle\sum_{i=0}^{k-1} AV_{Reg'_i} \cdot r/r_{zero} \end{aligned} \tag{11}$$

5) Treatment of full alignments: The method presented in the previous section estimates safely and tightly the maximum overlapping between the reuse and the interference region. However, its predictions can be very far from the average overlapping observed in codes with references whose accesses may collide systematically in the same cache sets, a situation we call full alignment.

Definition V.2. *Two references* `R` *and* `R'` *are* **potentially fully aligned** *when $S_{Ri} \mod C_{\mathrm{sk}} = S_{R'i} \mod C_{\mathrm{sk}}, \forall\, 0 \le i \le Z$, where S_{Ri} is the stride of reference R with respect to loop i as defined for Eq. (4), $C_{\mathrm{sk}} = C_s/k$ and Z is the innermost loop containing both references.*

The full alignment actually takes place when the addresses accessed by the references are aligned with respect to the cache, i.e., mapped to the same cache sets. Fully aligned references collide cyclically in the same cache sets. In the worst case they will be mapped to the same cache set in every iteration, in the best case in one out of L_s iterations. The corresponding accesses will result in misses when the number of lines involved per set is larger than the associativity.

a) Disabling the modeling of full alignments: The modular nature of our model allows to disable selectively the modeling of full alignments when desired. This is achieved by disabling the worst-case overlapping adjustments described in the preceding section only for the potentially fully aligned references in the innermost loop containing them. The user may wish to do this for two reasons. One is that s/he may have applied techniques such as padding or buffering to avoid the full alignments. In this situation the prediction will still be safe, but it will be much tighter. The other reason is to get an unsafe, but much more probable and tighter WCMP prediction, as the percentage of base address combinations that lead to full alignments is fortunately very small. This

would be very interesting for non-RTS and soft RTS [5]. Also, the probability there are full alignments in this second case can be quantified statistically. For example, in the benchmarks used in Section VI full alignments can only appear between references that follow a sequential access pattern, the most widely used access pattern. When two references follow this pattern they can collide systematically in the cache when the starting address for their accesses is within a distance of $L_s - 1$ positions in a cache way in one direction or $L_s - 2$ in the other, where L_s is the line size. This totals $2L_s - 2$ conflicting positions out of the $C_{sk} = C_s/k$ in a cache way. If a loop presents n sequential references, the probability full alignments take place is equal to the probability that more than k of them are aligned with respect to the cache, as k or fewer would fit in the set. This probability is $\frac{C_{sk}}{2L_s-2} \cdot P(x > k)$, where x is the number of references aligned, which belongs to a binomial of n elements with probability $\frac{2L_s-2}{C_{sk}}$.

C. Worst-case area vectors union

Only if k or more interfering lines are mapped to the cache set of a line whose reuse is being analyzed during its reuse distance, will the reuse attempt of the line fail. Despite this, area vectors (AV) keep ratios of interference with less than k lines. We must remember that each AV built at this point of the modeling only represents the cache footprint of one of the memory regions found within a reuse distance. Lines from different memory regions, thus represented in different AVs, can be mapped to the same set. Their combination can then yield the k or more lines that preclude reuses in that set. This last stage of the worst-case miss rate estimation process combines the AVs from all the individual memory regions found in the reuse distance into a global one that represents their joint impact on the cache.

Example V.5. *Let us consider a 2-way cache and a reuse distance in which two interfering memory regions (*`Reg` *and* `Reg'`*) appear. The AVs for both regions are* $AV_{Reg} = AV_{Reg'} = (0, 0.5, 0.5)$*. This means that in the worst-case each region can interfere with half of the lines whose reuse is being analyzed by putting a single interfering line in their cache set. Since the cache has two ways per set,* `Reg` *or* `Reg'` *alone cannot evict any of the lines to reuse. Now, if the lines from both regions are always mapped to the same cache sets, then they can place two interfering lines per set, generating misses in the reuse attempts in those sets. Since each region can, in the worst case, place one line in the sets of half of the reuse attempts, their worst-case combination can place two interfering lines in those sets. The AV that represents this worst-case joint effect is (0.5, 0, 0.5), since half of the reuse attempts miss (first component), while the other half experience no interference at all.* ■

The union of the AVs of the memory regions found in the reuse distance is done in the original model [4] using the ratios of cache sets of the AVs as probabilities of independent events. The approach is adequate, since the ignorance of the base address of the data structures makes it impossible to know the relative mappings to sets of lines accessed in different structures. The outcome of that situation is that the events that a set receives X lines from a data structure, and that it receives Y lines from another one, are independent. Since [4] tries to estimate average miss rates, the union of the AVs it performs does not consider the worst-case combinations of the input AVs, but an average one that generates a probable global AV. That method is thus inadequate to compute the worst-case joint effect of the AVs computed, which is the one that gives place to the largest possible miss rate. This miss rate is the component 0 of the global AV built.

In the computation of the WCMP, the algorithm in [4] is replaced with a deterministic algorithm called `maxUnionAV` which was introduced and explained in detail in [3]. This algorithm combines the ratios of interference of the AVs found in the reuse distance in the worst possible way. This way is the one that gives place to the largest possible component 0 for the global resulting AV, which is the worst-case miss rate for the reuse distance, as we have just explained. The details of this algorithm are not included in this paper due to space limitations.

TABLE II
CHARACTERISTICS OF THE CACHES USED IN THE EXPERIMENTS. C_s IS THE CACHE SIZE, L_s IS THE LINE SIZE, k IS THE ASSOCIATIVITY, AND HIT AND MISS ARE THE HIT AND MISS TIME IN CYCLES.

System	C_s	L_s	k	Hit	Miss
MicroSPARC II-ep	8KB	16B	1	1	10
PowerPC 604e	16KB	32B	4	1	38
MIPS R4000	16KB	16B	1	1	40
IDT79RC64574	32KB	32B	2	1	16

VI. EXPERIMENTAL RESULTS

We have validated our model using trace-driven simulations. The model, which is integrated in a compiler framework and provides its predictions always in less than one second, was applied automatically to ten codes: the average, sum and difference of the values stored in two arrays (ST); a 1D stencil calculation (STENCIL); the sum of all the values in a matrix (CNT); a matrix transposition (TRANS); the calculation of the first `N` fibonacci numbers (FIBONACCI) and five codes from the DSPStone benchmark suite [13]: convolution, fir, lms, matrix1 and n_real_updates. Pointer-based memory accesses were replaced with equivalent array accesses and functions were inlined. These codes have been gathered from similar works in the bibliography [3], [14], [10], [15].

The experiments were performed for each code considering a data size of 500 elements per dimension. The complexity of the matrix1 code, and thus its simulation time, is $O(n^3)$, so a smaller number of elements per dimension (200) was used in this case. Each code was tested using the cache configurations present in a MicroSPARC II-ep [16], a PowerPC 604e [17], a MIPS R4000 [18] and a IDT79RC64574 [19], which have been used in [3] and [10] too. Table II summarizes the main characteristics of these caches, including the cache hit and miss times.

The main contribution of this model is the estimation of a base address-independent WCMP. Thus its validation is based

TABLE III
$\overline{\text{MP}}$, $\Delta_{\text{WCMP\%}}$ AND $\Delta_{\text{W}\overline{\text{MP}}\%}$ FOR FOUR DIFFERENT CACHE CONFIGURATIONS.

Code	MicroSPARC II-ep			PowerPC 604e			MIPS R4000			IDT79RC64574		
	$\overline{\text{MP}}$	$\Delta_{\text{WCMP\%}}$	$\Delta_{\text{W}\overline{\text{MP}}\%}$	$\overline{\text{MP}}$	$\Delta_{\text{WCMP\%}}$	$\Delta_{\text{W}\overline{\text{MP}}\%}$	$\overline{\text{MP}}$	$\Delta_{\text{WCMP\%}}$	$\Delta_{\text{W}\overline{\text{MP}}\%}$	$\overline{\text{MP}}$	$\Delta_{\text{WCMP\%}}$	$\Delta_{\text{W}\overline{\text{MP}}\%}$
ST$_*$	8256	0.00	202.80	14155	0.00	571.14	27159	0.00	268.19	7225	0.00	453.58
STENCIL	4286	0.00	183.11	6699	0.00	0.55	11867	0.00	286.90	3905	0.38	0.77
CNT	812500	0.96	0.96	1406250	1.97	1.97	2687500	0.99	0.99	718750	1.57	1.57
TRANS	2045300	0.12	0.57	6151565	17.64	17.74	6292982	0.03	0.91	1762070	17.37	17.83
FIBONACCI	1625	0.55	0.55	2794	2.65	2.65	5375	0.73	0.73	1430	2.10	2.10
convolution	3276	0.00	205.19	5689	0.00	0.81	10836	0.00	269.11	2901	0.51	1.16
fir	4276	0.00	183.52	6689	0.00	0.69	11836	0.00	287.56	3901	0.38	0.87
lms	8244	0.00	203.22	14386	0.76	1.74	27280	0.00	266.56	7318	0.00	446.54
matrix1	44290990	2.11	4.07	61416512	2.42	2.42	107316817	2.83	7.03	39195460	2.69	5.64
n_real_updates$_*$	6618	0.00	202.20	11365	4.53	5.17	22045	0.00	262.88	5799	0.00	451.77

TABLE IV
$\overline{\text{MP}}$, $\Delta_{\text{WCMP\%}}$, $\Delta_{\text{W}\overline{\text{MP}}\%}$ AND $\text{A}_s(\text{A}_t)$ FOR FOUR DIFFERENT CACHE CONFIGURATIONS WHEN EXCLUDING FULL ALIGNMENTS.

Code	MicroSPARC II-ep				PowerPC 604e			
	$\overline{\text{MP}}$	$\Delta_{\text{WCMP\%}}$	$\Delta_{\text{W}\overline{\text{MP}}\%}$	$\text{A}_s(\text{A}_t)$	$\overline{\text{MP}}$	$\Delta_{\text{WCMP\%}}$	$\Delta_{\text{W}\overline{\text{MP}}\%}$	$\text{A}_s(\text{A}_t)$
ST$_*$	8125	0.98	2.55	1.93(2.91)	14155	1.49	6.80	0.0(0.0)
STENCIL	4268	0.21	0.42	0.29(0.29)	6691	0.00	0.66	0.0(0.0)
convolution	3262	0.55	0.72	0.29(0.29)	5680	0.00	0.98	0.0(0.0)
fir	4264	0.42	0.51	0.29(0.29)	6691	0.00	0.66	0.0(0.0)
lms	8190	0.74	0.99	0.48(0.87)	14432	0.76	1.41	0.0(0.0)
n_real_updates$_*$	6500	2.08	2.08	1.16(1.75)	11324	4.53	5.55	0.0(0.0)
Code	MIPS R4000				IDT79RC64574			
	$\overline{\text{MP}}$	$\Delta_{\text{WCMP\%}}$	$\Delta_{\text{W}\overline{\text{MP}}\%}$	$\text{A}_s(\text{A}_t)$	$\overline{\text{MP}}$	$\Delta_{\text{WCMP\%}}$	$\Delta_{\text{W}\overline{\text{MP}}\%}$	$\text{A}_s(\text{A}_t)$
ST$_*$	26875	0.14	2.03	0.97(1.46)	7225	1.63	3.74	0.003(0.01)
STENCIL	11828	0.00	0.33	0.19(0.14)	3905	0.38	0.77	0.0(0.0)
convolution	10804	0.36	0.58	0.14(0.14)	2900	0.51	1.19	0.0(0.0)
fir	11812	0.33	0.46	0.14(0.14)	3908	0.38	0.69	0.0(0.0)
lms	27148	0.29	0.58	0.24(0.43)	7225	1.64	2.70	0.0(0.0)
n_real_updates$_*$	21500	1.63	1.63	0.58(0.87)	5780	3.37	3.37	0.0015(0.004)

in simulating each code in each cache configuration using all the possible combinations of relative positions with respect to the cache of its data structures. Our simulation environment facilitates this, as it allows to specify the base addresses of the data structures and it is highly optimized. The variations in the cache behavior in the simulations are due exclusively to the changes in the base addresses of the data structures, since the data-dependent conditionals modeled cannot modify the cache behavior, as Section II explains. The number of combinations of relative cache offsets of the data structures in a code is very large (for example, in a direct mapped cache of 16 KB, each vector of elements for 4 bytes can present 16 KB/4=4096 different offsets) and it grows exponentially with the number of data structures. Thus two kinds of validations have been performed. For the codes with up to three data structures, *all* the relative address combinations were simulated systematically. For ST and n_real_updates, the only codes with more data structures, simulations using random offset combinations were run for 3200 hours (≈4 and 1/2 months) in a 1.6 GHz Itanium Montvale processor. For this reason these two codes appear in the validation tables with an asterisk.

Table III contains for each code and cache configuration, the average memory performance observed along the simulations expressed in cycles, $\overline{\text{MP}}$. The memory performance in each simulation is calculated as $NM \cdot mt + (ACCS - NM) \cdot ht$, NM being the numer of misses, $ACCS$ the number of accesses, and ht and mt the hit time and miss time of the studied cache, extracted from Table II. $\Delta_{\text{WCMP\%}}$ is the difference between the WCMP predicted by the model and the actual WCMP observed along the simulations, expressed as a percentage of this latter value. The non-negativity of the $\Delta_{\text{WCMP\%}}$ column shows the safeness of the prediction, while its small value shows its tightness for most codes. $\Delta_{\text{W}\overline{\text{MP}}\%}$ is the difference between the WCMP predicted by the model and the average memory performance observed along the simulations, $\overline{\text{MP}}$, expressed as a percentage of this latter value. The large value of $\Delta_{\text{W}\overline{\text{MP}}\%}$ for ST, STENCIL, convolution, fir, lms and n_real_updates indicates that for these codes the WCMP predicted (and also the actual WCMP measured, which is very near according to $\Delta_{WCMP\%}$) is far from the average value observed in the simulations. Sometimes it is up to 7 times larger. The reason for this large difference between the average and the worst-case memory performance for these codes, is the existence of full alignments of more than k (the associativity of the cache) references, which causes systematic cache misses. In the TRANS code the predictions of the WCMP are not very tight in two of the caches. The reason is that the overlapping adjustments described in Section V-B4 consider worst-case overlappings that do not actually take place for the data sizes and cache configurations used in our experiments.

A. Impact of disabling full-alignments modeling

A WCMP prediction closer to the average behavior for the codes where full alignments may appear can be obtained by disabling the modeling of the wort-case overlapping. The prediction would be still safe if the programmer avoids explicitly full alignments by using buffering or extra padding, for example. The results obtained using this new prediction of the WCMP are summarized in Table IV. This table only contains thus the codes where full alignments may appear, and its statistics are referred only to the simulated cases where full alignments did not finally occur. This way, $\overline{MP}$ is the average memory performance (expressed in cycles) observed in these simulations, and $\Delta_{\mathrm{WCMP}\%}$ and $\Delta_{\mathrm{W}\overline{\mathrm{MP}}\%}$ are calculated considering only simulations without full alignments. The values of $\Delta_{\mathrm{WCMP}\%}$ are similar to those in Table III and the values of $\Delta_{\mathrm{W}\overline{\mathrm{MP}}\%}$ are fairly smaller, which indicates that the WCMP is now closer to the average value observed. The table includes a column $\mathrm{A}_s(\mathrm{A}_t)$, where A_s is the percentage of all the simulations where full alignments appeared, and A_t is the estimation of the probability of full aligments predicted as described in Section V-B5. The large correlation between A_s and A_t indicates that the percentage of full alignments observed in the simulations is very close to the one predicted analytically. Only in the STENCIL code when using the MIPS R4000 cache, the percentage observed in the simulations is larger that the one predicted. The reason is that in this code one of the data structures presents two sequential references to two consecutive elements instead of just one. This increases slightly the probability of a systematic collision with the other data structure involved in the code because there is one more position in the cache where full alignments can appear.

VII. Related work

There are many works focused in the calculation of the WCET in the presence of data caches. Several of them have used analytical methods to calculate the WCMP in the presence of caches. The modeling of instruction-caches [20], [21] has had a lot of success, even recently in multicore systems with shared L2 instruction caches [22]. There are also many works devoted to the study of data caches. White et al. [15] bounds, using an static analysis, the worst-case performance of set-associative instruction caches and direct-mapped data-caches. The analysis of data caches needs to determine the base addresses of the involved data structures. Relative address information is used in conjunction with control-flow information by an address calculator to obtain this information. The analysis classifies the accesses in one of four categories: always miss, always hit, first miss and first hit. The validation is performed considering only one cache configuration.

Lundqvist and Stenström [23] distinguish between data structures that exhibit a predictable cache behavior, which is automatically and accurately determined, and those with an unpredictable cache behavior, which bypass the cache. Only memory references, whose address reference can be determined statically, are considered to be predictable. The predictability of a reference is determined considering the storage type (global, stack or heap) and the access type (scalar, regular, irregular or input data dependent). Nevertheless, they do not present an experimental results section.

TABLE V
BCMP, WCMP and $\overline{\mathrm{MP}}$ of the TRANS and the matrix1 benchmarks for 20×20 matrices in the MicroSPARC II-ep cache

Code	BCMP	WCMP	$\overline{\mathrm{MP}}$
TRANS	2600	3158	2663
matrix1	27100	41554	28162

Ramaprasad and Mueller [14] use the cache miss equations (CMEs) [8], which need the data addresses for their predictions, as a basis for the WCMP estimation. Non-perfectly nested and non-rectangular loops are covered using loop transformations like the forced loop fusion which involves the insertion of loop index-dependent conditionals in the code. Loop index-dependent conditionals are modeled using an extra analysis stage. The validation shows almost perfect predictions of the WCMP but only two (direct-mapped) cache configurations are considered.

Vera et al. [10] use also the data address-dependent cache miss equations (CMEs) to predict the WCMP in a multitasking environment. Their work combines the static analysis, provided by the CMEs, with cache partitioning for eliminating intertask interferences, and cache locking to make predictable the cache behavior of those pieces of code outside the scope of application of the CMEs. Good predictions of the WCMP are achieved for codes that use the cache locking in order to improve the WCMP predictability.

Our model is the only one to our knowledge that does not require the base addresses of the data structures. As a result, it is difficult to compare it with the previous works in the bibliography. The other models can only predict the WCMP for one specific possible combination of the base addresses of the data structures out of the millions that are possible. The WCMP for a given base-address combination can be very far from the one obtained with other combinations. This is indicated by the large values observed in the $\Delta_{\overline{\mathrm{MP}}\%}$ column in Table III for the codes with potential full alignments. Codes without full alignments can also experience wide variations of the cache behavior. For example, Table V shows the Best-Case (BCMP), Worst-Case (WCMP) and average-case ($\overline{\mathrm{MP}}$) memory performance for the TRANS and matrix1 codes used in Section VI, which have no full alignments, working on 20×20 matrices in the MicroSPARC II-ep cache. The memory performance is 21% worse in the WCMP than in the BCMP for TRANS and a 53% for matrix1. A single execution of our model, which takes less tan one second, provides a base-address independent prediction of the WCMP. However, such a prediction can only be provided by the other models by making the individual predictions for all the possible base-address combinations. In this case, assuming the predictions of those models are safe, the $\Delta_{\mathrm{WCMP}\%}$ parameter in Tables III and IV is an upper bound of the difference between those predictions and the one of our model.

VIII. Conclusions

This paper presents a model to predict a safe and tight upper bound of the cache performance whose main novelty is that

it is the only one that requires no information about the base addresses of the data structures. This property is very interesting, since base addresses are sometimes unavailable at compile time, and they can change between different executions. The model can provide two kinds of WCMP predictions. The absolute prediction is safe and tight with respect to the actual WCMP for any possible combination of base addresses of the data structures. The conditioned prediction is safe provided the programmer has avoided systematic collision of different references. In this case, the conditioned prediction will be besides much tighter. This prediction is also useful to know a very probable WCMP for soft-RTS and non RTS, as full alignments usually appear in a fairly small percentage of the base address combinations.

An extensive validation using trace driven simulations that required more than 30000 hours of CPU time shows that this approach yields safe and tight values of the WCMP.

In the future, we intend to extend our model to consider irregular access patterns and caches shared by several cores. Finally, we will consider using as input the base addresses of those data structures that are available in order to tighten the predictions.

APPENDIX A
REUSE AMONG DIFFERENT REFERENCES

Equation (4) in Sect. IV-A uses safe upper bounds of the reuse distances of the accesses of a reference with respect to its own previous accesses. Relying only in this equation to estimate the WCMP in a code is safe, but the predictions can lose tightness in codes with references that carry reuse between them, as the equation ignores those reuses. The PME model distinguishes the modeling of reuse among references found in the same loop nest and in different loop nests. We propose here modifications to the modeling of both kinds of reuse that improve the tightness of the WCMP prediction without compromising its safeness.

A. Reuse among references in the same loop nest

The model in [4] considers reuse among references in the same loop nest provided they are uniformly generated. This means that they have the same affine indexing functions $\alpha_{Rj}\mathtt{I}_j + \delta_{Rj}$ in every dimension j, except possibly in the δ_{Rj} constants. This is the most common kind of reuse among references in a loop nest by far, and it sufficed in [4] to model accurately many real codes. Let us also remember that if the references do not hold this condition, the lack of modeling of their reuse may worsen the tightness of the prediction, but it does not compromise its safeness. In order to model the reuse, the references are first sorted in descending order of the iteration of the loop nest in which they can access a line. This can be done building a vector $\overrightarrow{\delta_R}$ of n elements for each reference R in a loop nest of depth n such that

$$\overrightarrow{\delta_{Ri}} = \begin{cases} \delta_{Rj} & \text{if } \mathtt{I}_i \text{ indexes dimension } j \text{ of } R \text{ and } \alpha_{Rj} > 0 \\ -\delta_{Rj} & \text{if } \mathtt{I}_i \text{ indexes dimension } j \text{ of } R \text{ and } \alpha_{Rj} < 0 \\ 0 & \text{otherwise } (\mathtt{I}_i \text{ does not index } R) \end{cases}$$

Given two references R and R' in a loop nest of depth n, R can access a line before R' in the loop nest only if $\overrightarrow{\delta_R} \succ \overrightarrow{\delta_{R'}}$, which is defined as

$$\overrightarrow{\delta_R} \succ \overrightarrow{\delta_{R'}} \iff \exists i,\ 0 \le i < n \ /\ (\forall j,\ 0 \le j < i,\ \overrightarrow{\delta_{Rj}} = \overrightarrow{\delta_{R'j}}) \wedge (\overrightarrow{\delta_{Ri}} > \overrightarrow{\delta_{R'i}})$$

The first reference in the ordering is the first one to access any line in the loop nest, so its PMEs are derived following the method and equations previously described in Sect. IV-A. For each one of the following references, the PMEs for all the loops are also built with Eq. (4) except possibly the one for the loop associated with the longest reuse distance the reference can enjoy. This loop is the outermost one for which $\overrightarrow{\delta_{Ri}} \neq \overrightarrow{\delta_{R'i}}$. Concretely, its nesting level is the smallest i for which $\overrightarrow{\delta_{Ri}} \neq \overrightarrow{\delta_{R'i}}$. If there are other positions $j, j \neq i / \overrightarrow{\delta_{Rj}} \neq \overrightarrow{\delta_{R'j}}$, the loop is also modeled using the method in Sect. IV-A. Otherwise the PME for uniformly generated references derived in [4] is used to model R in this loop level i, with two changes to ensure the safeness of the WCMP prediction that we describe now.

The equation that models the reuse among uniformly generated references in [4] considers all the possible reuse distances, and it computes the average number of reuses for each reuse distance. The actual number of reuses depends on the alignment of the references with respect to a cache line at the beginning of the iterations of the loop nest, very much like the value of L_{Ri} that was maximized with Eq. (3). The transformation of this PME into a worst-case one is thus also achieved calculating the number of reuses for each possible reuse distance using the L_{s} possible initial alignments, and taking the largest one of the values computed. This approach is original in that, rather than maximizing the reuse distance for each access, it maximizes the number of accesses per reuse distance. As a result more accesses than the ones that will actually take place may be modeled, but the safeness of the WCMP prediction is ensured. The second change is that the number G of groups of lines used in the equation, which is an average in [4], is rounded up to the nearest largest integer.

B. Reuse among references in different nests

Let us consider two loop nests X and Y (X preceding Y) at a nesting level i which reference a data structure `S`, there being no intermediate loop nests between them that access this structure. The memory regions of `S` affected by the references in both loops are compared to determine the number n of lines they have in common. The first-time accesses to those lines in loop Y can enjoy a potential reuse with respect to the accesses in loop X. As Eq. (5) shows, a PME $F_{Ri}(\mathrm{RD})$ is always a expression of the kind $a \cdot MissR(\mathrm{RD}) + b$, where a is the number of first-time accesses to lines in the execution of the loop, i.e., the number of different lines accessed, and b is the number of misses in the reuses within the loop. As a result, if n out of these a lines are known to have been accessed in a preceding loop nest, $F_{Ri}(\mathrm{RD})$ can be rewritten as $(a-n) \cdot MissR(\mathrm{RD}) + n \cdot MissR(\text{RD from X to Y}) + b$. Let us notice that the number of lines n in common between the regions accessed in the two loops may vary with the L_{s}

possible initial alignments of the data structure s. Thus the model calculates the values of n for the L_s possible alignments and takes the smallest one. This is the worst case because the $a - n$ lines that cannot experience reuse in this nesting level will have necessarily longer reuse distances in outer loops, or even no reuse distance if they are cold misses.

There remains the problem of estimating the worst-case reuse distance between the loops X and Y used in the second term of this expression. A simple answer is to use the whole execution of both loops as well as the code between them as reuse distance. Most times the actual reuse distance will be of course shorter, but in worst-case situations this could be indeed the reuse distance: consider a loop that accesses the elements of a vector in increasing order, followed by a loop that accesses them in decreasing order. The reuse distance for the access to the first element of the vector in the second loop includes the whole execution of both loops.

ACKNOWLEDGMENT

This work was supported by the Xunta de Galicia under project INCITE08PXIB105161PR and the Ministry of Science and Innovation, cofunded by the FEDER funds of the European Union, under the grant TIN2007-67536-C03-02. We also want to acknowledge the valuable feedback obtained from Gabriel Rodríguez and the reviewers, as well as the Centro de Supercomputación de Galicia (CESGA) for the usage of its computers. The authors are members of the HiPEAC network.

REFERENCES

[1] R. Wilhelm *et al.*, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, 2008.

[2] K. D. Bosschere *et al.*, "High-performance embedded architecture and compilation roadmap," *Trans. on HIPEAC*, vol. 1, no. 3, pp. 5–29, 2006.

[3] D. Andrade, B. B. Fraguela, and R. Doallo, "Static prediction of worst-case data cache performance in the absence of base address information," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2009, pp. 45–54.

[4] B. B. Fraguela, R. Doallo, and E. L. Zapata, "Probabilistic Miss Equations: Evaluating Memory Hierarchy Performance," *IEEE Trans. on Comp.*, vol. 52, no. 3, pp. 321–336, March 2003.

[5] S. Manolache, P. Eles, and Z. Peng, "Optimization of soft real-time systems with deadline miss ratio constraints," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2004, pp. 562–570.

[6] D. Andrade, B. B. Fraguela, and R. Doallo, "Precise automatable analytical modeling of the cache behavior of codes with indirections," *ACM Trans. Archit. Code Optim.*, vol. 4, no. 3, p. 16, 2007.

[7] D. Andrade, B. B. Fraguela, and R. Doallo, "Analytical modeling of codes with arbitrary data-dependent conditional structures," *Journal of Systems Architecture*, vol. 52, pp. 394–410, July 2006.

[8] S. Ghosh, M. Martonosi, and S. Malik, "Cache Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior," *ACM Trans. on Programming Lang. and Sys.*, vol. 21, no. 4, pp. 703–746, July 1999.

[9] J. Xue and X. Vera, "Efficient and accurate analytical modeling of whole-program data cache behavior," *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 547–566, 2004.

[10] X. Vera, B. Lisper, and J. Xue, "Data cache locking for tight timing calculations," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 1, 2007.

[11] F. Mueller, "Compiler support for software-based cache partitioning," in *the ACM SIGPLAN workshop on Languages, compilers, & tools for real-time systems*. New York, NY, USA: ACM, 1995, pp. 125–133.

[12] D. Gannon, W. Jalby, and K. Gallivan, "Strategies for cache and local memory management by global program transformation," *Journal of Parallel and Distributed Computing*, vol. 5, no. 5, pp. 587–616, Oct. 1988.

[13] Zivojnović *et al.*, "DSPSTONE: A DSP-oriented benchmarking methodology," in *International Conference on Signal Processing Applications and Technology*, 1994.

[14] H. Ramaprasad and F. Mueller, "Bounding worst-case data cache behavior by analytically deriving cache reference patterns," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2005, pp. 148–157.

[15] R. White, C. Healy, D. Whalley, F. Mueller, and M. Harmon, "Timing analysis for data caches and set-associative caches," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 1997, pp. 192–202.

[16] S. Microelectronics, "microSPARC-IIep User's Manual," Tech. Rep., 1997.

[17] M. Inc, "PowerPC 604e RISC Microprocessor Technical Summary," Tech. Rep., 1996.

[18] M. Technologies, "MIPS32 4Kp- Embedded, MIPS Processor Core," Tech. Rep., 2001.

[19] I. D. Technologies, "79RC64574/RC64575 Data Sheet," Tech. Rep., 2001.

[20] M. Alt, C. Ferdinand, F. Martin, and R. Wilhelm, "Cache behavior prediction by abstract interpretation," in *the International Symposium on Static Analysis*. London, UK: Springer-Verlag, 1996, pp. 52–66.

[21] C. Healy *et al.*, "Bounding pipeline and instruction cache performance," *IEEE Trans. Computers*, vol. 48, no. 1, pp. 53–70, 1999.

[22] J. Yan and W. Zhang, "Wcet analysis for multi-core processors with shared l2 instruction caches," in *IEEE Real-Time and Embedded Technology and Applications Symposium*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 80–89.

[23] T. Lundqvist and P. Stenström, "A method to improve the estimated worst-case performance of data caching," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 1999, pp. 255–262.

[24] J. Engblom and A. Ermedahl, "Modeling complex flows for worst-case execution time analysis," in *IEEE Real-Time Systems Symposium*, 2000, pp. 163–174.

Basilio B. Fraguela received the M.S. and the Ph.D. degrees in computer science from the University of A Coruna, Spain, in 1994 and 1999, respectively. He is an associate professor in the Department of Electronics and Systems of the University of A Coruna since 2001. His primary research interests are in the fields of performance evaluation and prediction, analytical modeling, design of high performance processors and memory hierarchies, and compiler transformations.

Diego Andrade received the M.S. and the Ph.D. degrees in computer science from the University of A Coruna, Spain, in 2002 and 2007, respectively. He is a lecturer in the Department of Electronics and Systems of the University of A Coruña since 2006. His research interests focuses in the fields of performance evaluation and prediction, analytical modeling and compiler transformations.

Ramon Doallo, Ph.D in Physics (Univ. Santiago de Compostela) is a Full Professor in Computer Architecture and Technology, and the Head of the Computer Architecture Research Group at University of A Coruña, Spain. He has 22 years of experience in research and development in the area of High Performance Computing (HPC), covering a wide range of topics such as compilers and programming languages for HPC, parallel and distributed algorithms and applications, management of HPC infrastructures, cluster and grid computing, processor architecture, computer graphics, and distributed Geographic Information Systems. He has published more than 120 technical papers on these topics.