AN EFFICIENT SOLVER FOR CACHE MISS EQUATIONS *

Nerina Bermudo, Xavier Vera, Antonio González, Josep Llosa

Computer Architecture Department
Universitat Politècnica de Catalunya-Barcelona
{nbermudo, xvera, antonio, josepll}@ac.upc.es

ABSTRACT

Cache Miss Equations (CME) [2] is a method that accurately describes the cache behavior by means of polyhedra. Even though the computation cost of generating CME is a linear function of the number of references, solving them is a very time consuming task and thus trying to study a whole program may be infeasible.

This paper presents effective techniques that exploit some properties of the particular polyhedra generated by CME. Such techniques reduce the complexity of the algorithm to solve CME, which results in a significant speed-up when compared with traditional methods. In particular, the proposed approach does not require the computation of the vertices of each polyhedron, which has an exponential complexity.

I Introduction

Cache Miss Equations [2] are a very accurate analytical model of the cache memory. They describe the cache behavior by means of diophantine equations, which allows us to use mathematical techniques to compute the locality of each memory reference. For instance, by solving CME one could compute the different types of cache misses that each reference will cause. Unfortunately, a direct solution of the CME is computationally intractable due to its NP-hard nature.

CME allow us to study each reference in a particular iteration point independently of all other memory references. Deciding whether a reference causes a miss or a hit for a given iteration point is equivalent to deciding whether it belongs to the polyhedra defined by the CME. The number of cache misses can be computed by analyzing either all iteration points [3] or a subset of them through the use of statistical techniques [4].

In this paper, we present efficient techniques to count the number of integer points inside the polyhedra defined by the CME. By exploiting some intrinsic properties of the particular types of polyhedra generated by CME, we reduce the complexity of the algorithm, which results in very high speed-ups. We show that the proposed technique can compute the miss ratio of most SPECfp95 benchmarks just in a few seconds on a typical workstation. This opens the possibility to include this analysis framework in production compilers in order to support many optimizations.

The rest of this paper is organized as follows. In section 2, some background on polyhedra is reviewed. Section 3 summarizes the main features of the CME. Section 4 describes some criteria for detecting empty CME polyhedra. Section 5 presents a technique to compute whether a reference is a miss for a particular iteration point. Section 6 discusses the computation cost of the methods described in the previous sections. Finally, section 7 draws the main conclusions of this work.

II Background

This section reviews the definition of convex polyhedra and a general method to compute the number of integer points inside of it and a technique to identify whether it is empty.

A Definitions

Definition 2.1 Given the points $x_1, \ldots, x_n$ and scalars $\lambda_1, \ldots, \lambda_n$, we define a *convex combination* of $x_1, \ldots, x_n$ as $\sum_{i=1}^{n} \lambda_i x_i$ where $\sum_{i=1}^{n} \lambda_i = 1$ and all $\lambda_i \geq 0$.

Definition 2.2 A *vertex* of a set K is any point in K which cannot be expressed as a convex combination of any other distinct points in K.

Definition 2.3 A set K is *convex* $\iff$ every convex combination of any two points in K is also a point in K.

Definition 2.4 A *convex polyhedron* P is the intersection of a finite family of closed linear half-spaces of the

*This work has been supported by the ESPRIT project MHAOTEU (EP 24942) and the CICYT project 511/98.

form $\{\vec{x}|\vec{a}\vec{x} \geq c\}$ where a is a non-zero row vector and c is a scalar constant.

We only consider bounded convex polyhedra because polyhedra defined by Cache Miss Equations are convex and always bounded. Since the points of a convex bounded polyhedron can be expressed as a convex combination of its vertices, a polyhedron is fully described by its vertices. Therefore, the polyhedra can be given either by a system of linear constraints or a set of vertices [5].

Definition 2.5 We define the *real domain(integer domain)* of a variable x in a polyhedron P as the range of real values (integer values) it takes inside of P.

B Empty Polyhedra

Solving the CME requires to compute whether some polyhedra are empty. A polyhedron is considered empty when it does not contain any integer point, although it may contain real points. Counting the number of integer points in empty polyhedra often takes as much time as doing it for non empty polyhedra. Thus, a criteria for identifying empty polyhedra may be desirable.

For a given polyhedron, if there exists a variable that cannot take any integer value, there will not be any integer points inside the given polyhedron. This gives us a general criterion for detecting empty polyhedra , although it does not detect all of them. For each variable x_k, its definition domain $[a_k, b_k]$ in the polyhedron is calculated. Let lb_k and ub_k be the lower and upper bounds of the correspondent integer domain. If $ub_k < lb_k$ then the polyhedron is empty.

C Counting Integer Points

The method for counting presented next is based on the fact that the vertices of a polyhedron are extreme points. This implies that the greatest and smallest values that any variable can take inside a polyhedron can be found in the vertices. Therefore, the computation of the domain of a variable can be done using its vertices.

Let P be a polyhedron in $\mathbb{R}^p$. We take a variable x_i and calculate its integer domain $[lb_i, ub_i]$. Then, for every integer value z from this domain, we consider the *(p-1)-dimensional* polyhedra that result from giving the variable x_i the value z. This process is repeated recursively, until we have polyhedra defined only by one variable.

Let $P_1^1, \ldots, P_M^1$ be these polyhedra. The number of integer points inside one of them is $ub - lb + 1$, where ub and lb are the upper and lower bounds of the corresponding variable. The total number of integer points in the polyhedron is obtained by adding the points of $P_1^1, \ldots, P_M^1$.

Remarks

1. The selection of the variable to be fixed is not irrelevant. Since in general the domain of a variable in a polyhedron is a function of the other variables, we take every time the variable that has the smallest definition domain in order to minimize the number of nodes in the recurrence tree. Although we do spend some time in choosing the variable, this criterion helps us to reduce the time consumed by counting the number of integer points inside the polyhedron.

2. The domains of the variables are calculated as follows:
   Let x_k be the variable whose domain we want to determine. Let V_P be the set of vertices of P. Then the bounds of the integer domain of the variable in P are:

   $$lb_k = \lceil \min_{v=(v_1,\ldots,v_n)\in V_P} v_k \rceil \qquad ub_k = \lfloor \max_{v=(v_1,\ldots,v_n)\in V_P} v_k \rfloor$$

   Unfortunately, computing the vertices of a polyhedron is a problem with exponential complexity. Our approach avoids this expensive phase of the computation.

III CME Overview

CME [2] are an analysis framework that describes the behavior of a cache memory. The general idea is to obtain for each memory reference a set of equalities and inequalities defined over the iteration space that represent the cache misses. These equations make use of the reuse vectors [6]. Each equation[1] describes the iteration points where the reuse is not realized. This section presents an overview of the CME. Our study is mainly based on the structure of the CME polyhedra. Therefore, the interpretation of the different constants that appear in their definition is avoided except in some special cases, where the meaning of some of them is useful for the development of our techniques. For more details on Cache Miss Equations, the interested reader is referred to the original publications [2, 3].

We assume that $f_1, \ldots, f_m, g_1, \ldots, g_m$ are integer values. For each induction variable i_k $(k = 1, \ldots, m)$, ub_k and lb_k stand for the upper and lower bounds of this variable in the iteration space.

A Cold Miss Equations

These equations describe the iteration points where a reuse does not hold because the reference reuses data from an iteration point outside the iteration space. These polyhedra are defined over the iteration space. This means the

[1] The term equation has been used loosely to represent a set of simultaneous equalities or inequalities.

only variables that appear in their definition (in the linear inequalities that characterize the set), are the induction variables. The Cold Miss Equations constraint the possible values of one of the variables inside the iteration space. They have the following form:

$$(CM) \qquad \begin{array}{ll} i_l \leq d_l, & \text{for a fixed } l \in [1,\ldots,m] \\ lb_k \leq i_k \leq ub_k, & k = 1 \ldots m \end{array}$$

where i_l corresponds to the l-th variable of the iteration space, $d_l \in \mathbb{Z}$. The first equation represents an additional restriction on one of the variables. Note that this equation could introduce a lower bound of the variable i_k, instead of an upper bound. The other $2m$ constraints determine the iteration space.

B Cold Miss Bounds

These equations describe the iteration points where a spatial reuse is not realized because the reference reuses data that is mapped in a different cache line. These polyhedra are defined over $\mathbb{R}^{m+1}$, where m is the dimension of the iteration space. A new variable z is introduced for linearity reasons [1]. In fact, there is a version of the cache miss equations that ignores this variable [3], but we focus on the more precise model that includes it. The equations have the following form:

$$(CMB) \qquad \begin{array}{ll} f_1 i_1 + f_2 i_2 + \cdots + f_m i_m - Lz \geq LB \\ f_1 i_1 + f_2 i_2 + \cdots + f_m i_m - Lz \leq UB \\ lb_k \leq i_k \leq ub_k, & k = 1 \ldots m \end{array}$$

where $LB_1, LB_2, UB \in \mathbb{Z}$, and L is the cache line size.

C Replacement Equations

Given a reference, Replacement Equations represent its interferences with any other reference.

For each pair of references (R_A and R_B), the following expression gives the condition for a cache set contention in a k-way set associative cache:

$$\begin{array}{rcl} Cache\_Set(\vec{i})_{R_A} & = & Cache\_Set(\vec{j})_{R_B} \\ \vec{j} & \in & \mathcal{J} \end{array}$$

where $\mathcal{J}$ represents the iteration points between $\vec{i}$ (the current one) and the iteration point from which R_A reuses.

This identity results in

$$\begin{array}{rcl} Mem_{R_A}(\vec{i}) - Mem_{R_B}(\vec{j}) & = & Cn + b \\ \vec{j} & \in & \mathcal{J} \end{array}$$

This is the type of polyhedron obtained from the CME that has the most complicated topology. A Replacement polyhedron is contained in $\mathbb{R}^{2m+3}$. $2m$ of its variables $(i_1,\ldots,i_m, j_1,\ldots,j_m)$ refer in some way to the iteration space and the remaining variables (b, n and z) are artificial and have been introduced, as in the case of the Cold Miss Bounds, for linearity reasons. Replacement Equations have the following form:

$$(RCM) \qquad \begin{array}{l} -Lz - Cn + f_1 i_1 + \cdots + f_m i_m \geq AL \\ -Lz - Cn + f_1 i_1 + \cdots + f_m i_m \leq AU \\ Lz + g_1 j_1 + \cdots + g_m j_m \geq BL \\ Lz + g_1 j_1 + \cdots + g_m j_m \leq BU \\ n \neq 0 \\ p_k \leq i_k - j_k \leq q_k, \qquad k = 1 \ldots m \\ lb_k \leq i_k \leq ub_k, \qquad k = 1 \ldots m \end{array}$$

where $AU, AL, BU, BL \in \mathbb{Z}$.

D Solving CME

The points inside each CME polyhedron represent the potential cache misses (the number of points is the number of potential cache misses). This leads us to consider several ways for computing them:

- **Solver** Given a reference R with m reuse vectors and n_k equations for the k^{th} reuse vector, the polyhedron that contains all the iteration points that result in a miss is [2]:

  $$Set\_Misses = \cap_{k=1}^{m} \cup_{j=1}^{n_k} Solution\_Set\_Equation_j$$

  This approach implies to count the number of points inside the union of convex polyhedra.

- **Traversing the iteration space** Given a reference, all the iteration points can be tested independently [3]. For this approach, we need to compute whether a polyhedron is empty after substituting the iteration point in the equations.

In a k-way set associative cache, there are k cache line in every set, so k distinct contentions are needed before a cache miss occur. Therefore, the first method can only be applied to direct-mapped caches whereas the second method works for both direct-mapped and set-associative organizations. Our proposal builds upon the second method.

IV Removing Empty Polyhedra

The complexity of both methods mentioned above is a function of the number of CME polyhedra. For this reason it is interesting to reduce the number of them. This section presents some criteria for detecting empty CME polyhedra.

The general criterion that has been presented in section B does not detect all empty polyhedra. In order to increase the number of detected empty polyhedra, specific criteria for each type of polyhedron have been developed. These criteria rely on the structure of the equations and their interpretation in terms of the cache behavior.

A Cold Miss Equations

Since each of these polyhedra consists of the iteration space and an additional constraint on one of the variables, it will be empty if the constraint is incompatible with the iteration space. If the additional restriction has the form $i_l \leq d_l$ and $d_l < lb_l$, there is a contradiction between the two conditions and we conclude that the polyhedron is empty. The same happens when the constraint has the form $i_l \geq d_l$ and $d_l > ub_l$. Hence, the time taken to compute the emptiness is $O(1)$.

B Cold Miss Bounds

Recall the equations that define the Cold Miss Bounds polyhedra. Since the domains of $i_1, \ldots, i_m$ are explicitly given, and they are not constrained by any other equation, the only variable that might have a domain without integer values inside is variable z: Let us observe the constraints involving variable z.

$$f_1 i_1 + \cdots + f_m i_m - UB \leq Lz \leq f_1 i_1 + \cdots + f_m i_m - LB$$

We have that

$$z_{max} = \frac{\max_{(i_1,\ldots,i_m)\in I}\{f_1 i_1 + \cdots + f_m i_m\} - LB}{L}$$

$$z_{min} = \frac{\min_{(i_1,\ldots,i_m)\in I}\{f_1 i_1 + \cdots + f_m i_m\} - UB}{L}$$

where I stands for the iteration space. Then, the integer domain of the variable z in the polyhedron (CMB) is

$$[\lceil z_{min} \rceil, \lfloor z_{max} \rfloor] \cap \mathbb{Z}$$

If there are no integer values inside this interval, we can conclude that the (CMB) polyhedron is empty in $O(m)$. This condition is sufficient, but not necessary. That is, even if the domain of z contains integer values, the polyhedron might be empty.

C Replacement Equations

Different criteria to detect empty Replacement polyhedra have been developed. In this case, not only the information given by the equations is considered, but also its interpretation in terms of the cache behavior.

- **Convex Regions:** In a Replacement polyhedron, there is a subset of equations which relates the variables i_k and j_k for $k = 1, \ldots, m$.

  $$i_k - j_k \geq p_k, \qquad i_k - j_k \leq q_k, \qquad k = 1, \ldots m$$

  These equations appear from the division in convex regions of the domain of the variables $j_1, \ldots, j_m$ [2]. In order to detect empty Replacement polyhedra, it is checked whether these constraints are consistent with the fact that $\vec{i}$ and $\vec{j}$ must belong to the iteration space. The worst case complexity for calculating it is $O(m)$.

- **$Mem_{R_A} - Mem_{R_B}$ and the variable n have different sign:** Recall that Replacement equations result from the following identity:

  $$Mem_{R_A}(\vec{i}) - Mem_{R_B}(\vec{j}) = Cn + b$$

  where R_A and R_B are the references whose interferences are being studied, C stands for $\frac{\text{cache size}}{k}$, where k is the associativity of the cache, n stands for the distance between R_A and R_B in cache size units, and b is the difference between the offset of each reference with respect to the beginning of their respective lines.

  Since the placement of the two references R_A and R_B in the memory is fixed, their relative position will not change, so that $Mem_{R_A}(\vec{i}) - Mem_{R_B}(\vec{j})$ has constant sign for all $\vec{i}, \vec{j}$.

  Besides, this sign must be the same as the sign of the variable n, as this variable represents the distance, in terms of cache size, between the two references. A Replacement polyhedron is empty if the range of feasible values of the expression $Mem_{R_A}(\vec{i}) - Mem_{R_B}(\vec{j})$, (which depends on the variables $i_1, \ldots, i_m$ and $j_1, \ldots, j_m$), causes a contradiction with the constraint that determines the sign of the variable n. This can be done in $O(1)$.

- **Incompatible range of $Mem_{R_A} - Mem_{R_B}$ with the constraint on the variable n:** Depending on the constraint on the variable n, one of the following expression holds:

  $n \leq -1$:
  As this restriction gives an upper bound of n, that is a lower bound of $-n$, we consider the second constraint

  $$\begin{aligned} AU - BU &\geq f_1 i_1 + \ldots + f_m i_m + g_1 j_1 + \ldots + g_m j_m - Cn \\ &\geq f_1 i_1 + \ldots + f_m i_m + g_1 j_1 + \ldots + g_m j_m + C \\ &\geq \min_{\vec{i}\in I, \vec{j}\in J} \{f_1 i_1 + \ldots + f_m i_m + g_1 j_1 + \ldots + g_m j_m\} \\ &\quad + C \end{aligned}$$

  $n \geq 1$:
  In this case the considered inequation is the first one.

  $$\begin{aligned} AL - BL &\leq f_1 i_1 + \ldots + f_m i_m + g_1 j_1 + \ldots + g_m j_m - Cn \\ &\leq f_1 i_1 + \ldots + f_m i_m + g_1 j_1 + \ldots + g_m j_m + C \\ &\leq \max_{\vec{i}\in I, \vec{j}\in J} \{f_1 i_1 + \ldots + f_m i_m + g_1 j_1 + \ldots + g_m j_m\} \\ &\quad + C \end{aligned}$$

  where I stands for the iteration space and J for the domain of $(j_1, \ldots, j_m)$.

  In each of these cases, if the constraint does not hold, we conclude the polyhedron is empty in $O(m)$.

- **The variable n cannot take integer values:** The inequations (1) and (1) are used in order to compute the domain of the variable n. If it contains no integer points, the polyhedron is empty.

Hence, the worst case complexity to decide whether a Replacement Equation is empty or not is $O(m)$.

V Analyzing Iteration Points

This section shows some methods for knowing whether an iteration point $\vec{i_0}$ fulfills a CME. This problem is equivalent to finding out whether the resulting polyhedron after substituting the variables $i_1, \ldots, i_m$ with the values given by the iteration point is empty.

A Cold Miss Equations

Let $\vec{i_0} = (i_{01}, i_{02}, \ldots, i_{0m})$, be the iteration point studied. The only inequality it might not verify is

$$i_l \leq d_l \qquad (1)$$

as the others represents the iteration space. So, $\vec{i_0}$ is a point from the given Cold Miss polyhedron $\iff$ its l-th component verifies inequality (1).

B Cold Miss Bounds

When an iteration point $\vec{i_0}$ is substituted in the Cold Miss Bounds Equations, a *1-dimensional* polyhedron is obtained. Deciding whether $\vec{i_0}$ verifies the equations is equivalent to deciding whether the *1-dimensional* polyhedron

$$(CMB') \qquad LB' \leq -Lz \leq UB'$$

is empty, where $LB' = LB - f_1 i_{01} - \cdots - f_m i_{0m}$ and $UB' = UB - f_1 i_{01} - \cdots - f_m i_{0m}$.

The real domain of the variable z, $\left[\frac{UB'}{L}, \frac{LB'}{L}\right] \subset \mathbb{R}$, is first computed, and then the integer domain of z is obtained from its real domain. By comparing its bounds, it is determined whether it is empty.

C Replacement Equations

After an iteration point $\vec{i_0}$ has been substituted in the equations of a Replacement polyhedron, the problem of deciding whether it is a potential miss depends on the associativity of the cache. When considering a k-way set associative cache, $\vec{i_0}$ fulfills the equations if the polyhedron contains a set of integer points with k different values of the variable n (that represent k distinct contentions, $k \geq 1$).

We propose a method for counting integer points inside Replacement polyhedra that works either for direct mapped caches or for set-associative organizations.

C.1 Counting Integer Points

In this section, a method for counting the Replacement polyhedra will be described. It is based on the general method presented in section C, extended with a new technique to compute the domains of the variables.

When considering a k-way set associative cache, a polyhedron is not empty when it contains a set of integer points with at least k different values of the variable n.

From the definition of (RCM') we can derive the following conclusion:

The domains of the variables $j_1, \ldots, j_m$ are explicitly given in the expression of the polyhedron, so they do not need to be calculated. The domain of the variable n can be calculated by means of the two next inequations:

$$g_1 j_1 + \cdots + g_m j_m - BU' \leq Cn \leq g_1 j_1 + \cdots + g_m j_m - BL' \qquad (2)$$

Let us define

$$n_{max} = \frac{\max_{(j_1,\ldots,j_m)\in J}\{g_1 j_1 + \cdots + g_m j_m\} - BL'}{C}$$

$$n_{min} = \frac{\min_{(j_1,\ldots,j_m)\in J}\{g_1 j_1 + \cdots + g_m j_m\} - BU'}{C}$$

where J is the domain of $\vec{j} = (j_1, \ldots, j_m)$. Then, the integer domain of the variable n in the polyhedron (RCM') is

$$[\lceil n_{min} \rceil, \lfloor n_{max} \rfloor] \cap \mathbb{Z}$$

We can thus conclude that the domains of all variables are easily computed and the explicit computation of the vertices is not needed.

Since the domains of the variables $j_1, \ldots, j_m$ may change when the variable n is fixed, the order in which the variables will be fixed cannot be determined at the beginning. Thus, the real domain of these variables must be recalculated every time. This is done in a similar way to the computation of the domain of n in the initial polyhedron: for every variable j_k, its greatest and lowest values given by the two inequations (eq. 2) are calculated. The actual domain of this variable is the intersection between this interval and the explicit domain given by the equations of the polyhedron.

In order to detect empty polyhedra, the search of empty integer domains must be done for all the variables. Theoretically, the complexity is $O(\#iteration\_points)$, but in practice, it is $O(1.5^m)$ for our benchmarks.

VI Performance Evaluation

We have generated the CME for the SPECfp95 benchmark suite. For each program, we have chosen the most time consuming loop nests that in total represent between the 60-70% of the total execution time using the reference input data.

A Empty Polyhedra

First we evaluate the effectiveness of our proposal for detecting empty polyhedra, assuming a 32K direct mapped cache. Figure 1 compares our method with the technique

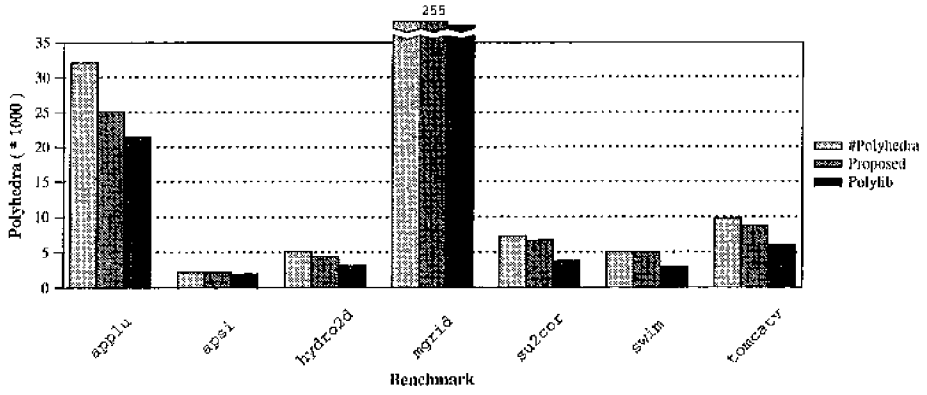

Figure 1. Empty Polyhedra

of the Polylib [5] for detecting empty polyhedra. Only Replacement polyhedra have been considered, as their evaluation is the most time consuming among all CME polyhedra. The first column shows the number of Replacement polyhedra obtained for each SPECfp95 program analyzed. The second column depicts the number of empty polyhedra detected by our approach, whereas the third column shows the number of empty Replacement detected through the Polylib. We can see that our approach detects a significantly higher number of empty polyhedra. This is due to the fact that Polylib only detects polyhedra without any real point inside.

Columns 1 and 2 of table 2 show the execution time required by both methods to check the emptiness of all polyhedra. Due to the complexity of the computation of the vertices of a polyhedron, our proposal is much faster than Polylib's technique. The complexity of the proposed method is $O(m)$. On the other hand, Polylib's method relies on computing the vertices of each polyhedron. The complexity of the algorithm that it uses is $O(\#constraints^{\lfloor \frac{\#variables}{2} \rfloor})$. For Replacement Polyhedra, the number of constraints is $2m + 3$ and the number of variables is $m + 1$, where m is the nesting depth of the loopnest. Thus, the complexity of Polylib's approach is $O(m^{\lfloor \frac{m}{2} \rfloor})$.

B Analyzing Iteration Points

In order to evaluate the techniques proposed for analyzing iteration points, we implemented a solver of the Cache Miss Equations based on traversing a subset of the iteration space through sampling techniques as described in [4]. Next, we evaluate the effectiveness of the proposed technique for both direct mapped and set-associative caches, and it is compared to an algorithm that counts the number of integer points inside the polyhedra by means of the general method for counting presented in section C. The computation of the vertices of the polyhedra needed for this second method (*Vertices*) is done by means of functions from the Polylib library.

Table 1 shows the time in seconds required to analyze the different SPECfp95 for four different organizations of set-associative caches, for both the proposed method and the *Vertices* method.

The speed-up of our approach is very important, due to the different complexities of both algorithms. For a direct mapped cache, it is between 7 and 418 times faster than the *Vertices* method and it is 30 on average. The speed-up for different set-associative configurations is even higher. For instance, the average speed-up for a 4-way set-associative cache is 42.

The difference between these two algorithms relies on the approach to compute the domains of all variables. The proposed method does it with a complexity of $O(m^2)$. The *Vertices* method is split into two steps: first the vertices of the polyhedron are computed with a complexity of $O(m^{\lfloor \frac{m}{2} \rfloor})$, as explained in the previous section. Then, by means of the vertices, the domains of the variables are computed with a complexity of $O(m * \#vertices)$.

Note that most programs can be analyzed by the proposed approach in less than a minute and the most expensive one is applu which takes about 1.5 minutes, whereas the approach based on the *Vertices* method takes several minutes and in the worst case it takes more than one hour.

		Applu	Hydro2d	Mgrid	Su2cor	Swim	Tomcatv
(1)	1 way	99.63s	12.21s	1.89s	1.91s	2.18s	4.57s
	2 way	89s	12s	2.36s	2s	4s	5.2s
	4 way	91s	12.8s	7s	2.02s	7.7s	8.4s
	8 way	97s	14.19s	14.19s	2.17s	15s	15s
(2)	1 way	18m48.55s	1m26.23s	9m6.67s	19.18s	21.22s	31m51s
	2 way	6m44.92s	1m53.27s	9m27.35s	27.6s	26.66s	34m23.73s
	4 way	6m39.62s	2m0.33s	13m32.59s	17.77s	48.32s	1h8m24s
	8 way	6m37.85s	1m52.84s	18m46.30s	15.30s	1m29.6s	1h7m25s
speed-up 1way		11.7	7	283	10	9.7	418.1

(1) Proposed
(2) Vertices

Table 1. Execution time for different cache organizations using an Origin2000.

SPEC	Empty polyhedra	
	Proposed	Polylib
applu	36.70	1933.87
apsi	1.30	17.31
hydro2d	3.16	47.51
mgrid	235.75	5495.10
su2cor	3.08	34.27
swim	3.00	48.71
tomcatv	9.60	280.75
total	292.59	7857.52

Table 2. Execution time (in seconds) using a Sun Ultra Sparc I.

VII Conclusions

Cache Miss Equations provide an analytical and precise description of the cache memory behavior. Unfortunately solving CME by traditional methods based on counting integer points inside polyhedra is a very time consuming task that makes them infeasible for many applications.

In this paper we propose some techniques that exploit some intrinsic properties of the particular polyhedra generated by CME. These techniques significantly reduce the complexity of the algorithms and result in speed-ups of more than one order of magnitude for the SPECfp95 benchmarks. This important speed-up is due to the fact that the proposed approach does not require the computation of the vertices of the associated polyhedra. We have shown that the proposed approach usually takes just a few seconds to analyze a program of the SPECfp95, and it never takes more than 2 minutes for the different cache configurations that have been analyzed. This cost is small enough to allow the technique to be included in a production compiler.

References

[1] P. Clauss. Counting solutions to linear and nonlinear constraints through ehrhart polynomials: applications to analyze ans transform scientific programs. In *ICS96*, pages 278–285, 1996.

[2] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: an analytical representation of cache misses. In *ICS97*, pages 317–324, 1997.

[3] S. Ghosh, M. Martonosi, and S. Malik. Precise miss analysis for program transformations with caches of arbitrary associativity. In *ASPLOS98*, 1998.

[4] X. Vera, J. Llosa, A. Gonzalez, and C. Ciuraneta. A fast implementation of cache miss equations. Technical Report UPC-DAC-1999-50, Universitat Politècnica de Catalunya, November 1999.

[5] D. Wilde. A library for doing polyhedral operations, 1993.

[6] M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In *ACM SIGPLAN91*, pages 30–44, 1991.