Handling Memory Cache Policy with Integer Points Countings

Philippe Clauss

ICPS, Université Louis Pasteur, Strasbourg,
Pôle API, Bd Sébastien Brant,
67400 Illkirch France
clauss@icps.u-strasbg.fr
http://icps.u-strasbg.fr

Abstract. We propose an automatic method allowing the computation of useful information on the distribution of data in the memory cache. The number of distinct memory locations and cache lines touched by a loop, and the number of processors resulting from a distribution (CYCLIC, BLOCK, REPLICATE) from array elements are computed. It is shown that these problems are relevant to the same general mathematical problem, which is the counting of the exact number of integer points resulting from linear and non-linear mappings of polytopes.

1 Introduction

The construction of realistic and efficient parallel programs requires the extraction and the use of much information, concerning as well the potential parallelism and the needed resources of the algorithm, as the availability of resources on the target architecture. But these are generally not trivial to determine.

Nowadays processors are designed with a memory hierarchy organized into several levels, each of which is smaller, faster, and more expensive than the level below. The cache is the memory level between CPU and the main memory. It is usually between 1KB to 256KB, and is divided into lines of 4-128 bytes. The unit of data transfer between main memory and cache is a line. If the same data in the cache is reused, this is called *temporal locality*. Since the unit of data in cache is a line, once the line is brought into the cache, any access to the data elements in the line is a *cache hit*. If data in the same cache line are used, this is called *spatial locality*. If a data is not found in the cache, a *cache miss* occurs. The *cache miss rate* is the ratio of cache misses to cache accesses. A cache hit takes usually one cycle, while a cache miss takes 8-32 cycles. It is ideal for all memory references to hit in the cache, however, since cache size is much smaller than main memory, once new data needs to be brought in from main memory, some data in the cache has to be replaced.

In a multiprocessor system with multiple caches, data inconsistency may occur between memory and caches. Multiple caches may have inconsistent copies of the same cache line. The memory and caches may have inconsistent copies as well. All these facts are related to the problem of *cache coherence*.

In this work, we make the following contributions. It is shown how the number of distinct memory locations and cache lines touched by a loop can be automatically determined. Such issues are very important, for example, in order to determine the minimum cache size avoiding any cache miss to occur, or in order to optimize the shape of tiles for minimal communication when partitioning parallel loops [1, 16]. Anyway, the knowledge of such information helps the mapping process to cache miss rate optimization [8], and allows the minimization of the elapsed time waiting for cache lines to be loaded or unloaded, by minimizing the number of cache lines touched.

The presented method can also be used to compute the number of processors resulting from a distribution (CYCLIC, BLOCK, REPLICATE) of array elements for common languages such as *HPF* [10], and can generally be used as a performance prediction tool for parallel programs.

Our application context is the well-known *polytope model*, where any iteration or array element is represented as an integer point whose coordinates are its index values. Hence, we propose a geometrical approach to reach our goals, unlike some other ones as W. Pugh in [15], who proposes the manipulation of Presburger formulas for equivalent objectives.

We show that these problems are relevant to the same general mathematical problem, which is the counting of the exact number of integral points resulting from non-linear mappings of polytopes. The non-linear mappings that we consider are defined as being linear mappings where further functions such as modulos and integer divisions occur.

We have introduced in some recent works [3, 4] an exact counting method of integer points in polytopes, based on the determination of the *Ehrhart polynomials* of any parameterized polytopes, where many applications to the analysis and transformations of programs have been shown. An application to probabilistic memory disambiguation has also been presented by Ju, Collard and Oukbir in [9].

Let us recall this method by considering the simple example of a rectangle whose definition depends on two parameters n and m, $P_{n,m} = \{(i,j) \in \mathcal{Z}^2 | 0 \le i \le \frac{n}{2}, 0 \le j \le \frac{m}{2}\}$. The objective is to compute an analytic formula $EP(n,m)$, giving the number of integer points in $P_{n,m}$ relatively to n and m. Such formulas are called *Ehrhart polynomials* since they are due to the mathematician Eugène Ehrhart [7]. These polynomials are characterized by *periodic coefficients*, i.e. arrays of rational values, for which each dimension is associated with a parameter, and where the indices of a selected value are given by the modulos of the parameters relatively to the numbers of possible values. An upper bound of these numbers of possible values is given by the *denominator of the polytope*, i.e. the *lcm* of the vertices coordinates denominators. For example, $(-1)^{n-m}$ can be defined as being the periodic number $\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}_{n,m}$.The first step is to compute the parametric coordinates of the vertices with the Loechner/Wilde algorithm [13]. The algorithm answer consists in sets of vertices associated with adjacent *parameters validity domains*. In our example, the vertices are $(0,0)$, $(\frac{n}{2},0)$, $(0,\frac{m}{2})$

and $(\frac{n}{2}, \frac{m}{2})$, associated with a single parameters validity domain $\{n \geq 0, m \geq 0\}$. Since the degree of $EP(n,m)$ is equal to the dimension of $P_{n,m}$, it is of the form,

$$EP(n,m) = \begin{bmatrix} c_1 \; c_2 \\ c_3 \; c_4 \end{bmatrix} nm + \begin{bmatrix} c_5 \; c_6 \\ c_7 \; c_8 \end{bmatrix} n + \begin{bmatrix} c_9 \;\; c_{10} \\ c_{11} \; c_{12} \end{bmatrix} m + \begin{bmatrix} c_{13} \; c_{14} \\ c_{15} \; c_{16} \end{bmatrix}$$

$EP(n,m)$ can also be defined as being an Ehrhart polynomial of the form $[c_1, c_2]n + [c_3, c_4]$, where each c_k is also an Ehrhart polynomial in m of the same form. Hence, the coefficients of $EP(n,m)$ are computed by resolving systems of linear equations. For example, the coefficients c_1 and c_3 occurring when $n \; mod \; 2 = 0$ are computed by resolving:

$$\begin{cases} c_3 & = EP(0,m) \\ 2c_1 + c_3 & = EP(2,m) \end{cases}$$

To be able to resolve this system, $EP(0,m)$ and $EP(2,m)$ have first to be determined. For example, $EP(0,m) = [c_5, c_6]m + [c_7, c_8]$ is computed by resolving:

$$\begin{cases} c_7 & = EP(0,0) = 1 \\ c_6 + c_8 & = EP(0,1) = 1 \\ 2c_5 + c_7 & = EP(0,2) = 2 \\ 3c_6 + c_8 & = EP(0,3) = 2 \end{cases}$$

The values of $EP(0,0)$, $EP(0,1)$, $EP(0,2)$ and $EP(0,3)$ are determined by using a loop scanning $P_{0,1}, \ldots, P_{0,3}$. Finally, the following result is obtained:

$$EP(n,m) = \begin{bmatrix} 1/4 \; 1/4 \\ 1/4 \; 1/4 \end{bmatrix} nm + \begin{bmatrix} 1/2 \; 1/4 \\ 1/2 \; 1/4 \end{bmatrix} n + \begin{bmatrix} 1/2 \; 1/2 \\ 1/4 \; 1/4 \end{bmatrix} m + \begin{bmatrix} 1 \;\;\; 1/2 \\ 1/2 \; 1/4 \end{bmatrix}$$

which can be easily simplified to:

$$EP(n,m) = \frac{1}{4}nm + \left[\frac{1}{2}, \frac{1}{4}\right]_m n + \left[\frac{1}{2}, \frac{1}{4}\right]_n m + \begin{bmatrix} 1 \;\;\; 1/2 \\ 1/2 \; 1/4 \end{bmatrix}_{n,m}$$

This parametric formula can be evaluated for any value of n and m. For example, the number of integer points in $P_{4500,2001}$ is determined by $EP(4500, 2001) = \frac{1}{4} \times 4500 \times 2000 + \frac{1}{4} \times 4500 + \frac{1}{2} \times 2001 + \frac{1}{2} = 2252126$.

All of these described calculations are independent of the size parameters of the considered polytope. Hence, the complexity is $\mathcal{O}(1)$ relatively to these parameters. A more precise complexity analysis yields $\mathcal{O}(v(dn)^m)$, where v denotes the number of parameters validity domains, d denotes the denominator of the polytope, n denotes the polytope dimension (the number of free variables), and m denotes the number of parameters. This complexity can be compared to the one of the *Omega test*, also used for equivalent objectives [15], which has an upper bound of $2^{2^{2^{\mathcal{O}(n)}}}$. However, the *Omega test*, implemented in the *Omega calculator*, is dedicated to the resolution of a large field of problems, as exact data dependence analysis for example [14].

The method can also be used to count integer points in non-parametric polytopes. The way is to introduce an artificial parameter n, to compute the corresponding Ehrhart polynomial and then to evaluate this polynomial with the value of n corresponding to the initial problem.

Unfortunately, this method does not handle mappings of polytopes. However, we show in this paper that its fullness allows the construction of a general method reaching our current objectives.

The paper is organized as follows. The second section is devoted to formalize our problem of processors distribution or cache lines touched countings. The method elaboration is then decomposed in first resolving the linear mapping case in the third section, and then resolving the original one, the non-linear mapping case in the fourth section. Both sections are illustrated with an application example. An example dealing with *parametric loop tiling* is also presented.

2 Problem statement

Considering mapping functions from array elements to cache lines, or distributions from array elements to processors, yields the consideration of non-linear functions. These functions are commonly composed of integer divisions $\div$ (BLOCK) and modulo operations *mod* (CYCLIC). Other common used operators (REPLICATE) are linear.

In the polytope model, array elements are represented geometrically as integer points, whose coordinates are their indices, belonging to a polytope P. Hence, any mapping function or distribution can be modeled by a general non-linear mapping function of the form:

$$\begin{aligned} P &\longrightarrow M \\ nonlinmap : X &\longmapsto Y = [(((F_k(X) + f_k) \odot a_{1,k} + b_{1,k}) \odot a_{2,k} + b_{2,k}) \ldots]_{1 \le k \le q} \end{aligned}$$

where F_k is a linear integer combination on X, f_k, $a_{i,k}$ and $b_{i,k}$ are integer constants and $\odot$ symbolizes $\div$, *mod*, or any common arithmetic operation.

Our objective is to determine the number of different values of Y that are reached through *nonlinmap*. The difficulty of this problem stems from the fact that generally, the reached values are non-continuous, i.e. do not define a polytope in a regular lattice of integer points. Analyzing *nonlinmap*, a linear part appears, $F_k(X) + f_k$, composed with non-linear operators $\odot$. Hence, we first resolve the problem of counting the number of integer points resulting from a linear mapping of the form:

$$\begin{aligned} P &\longrightarrow M \\ linmap : X &\longmapsto Y = [F_k(X) + f_k]_{1 \le k \le q} \end{aligned}$$

It turns out that the resolution of this problem is useful to handle the number of memory locations touched by a loop nest, as it is shown in the next section example.

3 Counting in linear mappings of polytopes

The following example is also handled by W. Pugh in [15] and by us in [3]. In this last work, a different approach has been proposed, which does not allow the consideration of any linear mapping, but only from $\mathcal{Z}^n$ to $\mathcal{Z}^{n-1}$.

Example 1. It consists in the following loop nest:

```
for i := 1 to 8 do
    for j := 1 to 5 do
        a(6i+9j-7) = a(6i+9j-7) + 5 ;
```

Computing the number of distinct memory locations touched by this loop is equivalent to count the number of distinct values of `6i+9j-7` subject to $1 \le i \le 8$ and $1 \le j \le 5$. It is also equivalent to count the number of integer points resulting from $linmap(i,j) = 6i + 9j - 7$, applied to all the points of the iteration space $P = \{(i,j) \in \mathcal{Z}^2 | 1 \le i \le 8, 1 \le j \le 5\}$. □

In this example, the mapping is made from dimension 2 to dimension 1. Let n be the dimension of the iteration space and let q be the dimension of the referenced array. The following presented method considers references to any array whose dimension q is greater than one. In cases where $1 \le q < n$, the problem can be seen as a projection of the iteration space into a lower dimension. In cases where $q > n$, the problem can be seen as an "immersion" in a higher dimensional space.

So the objective is to compute the number of integer points resulting from *linmap*. Hence, this problem is equivalent to determine the number of distinct values of $Y = [F_k(X) + f_k]_{1 \le k \le q}$ reached through *linmap*.

Proposition 1. *Let P_Y be the parametric polytope defined by:*

$$P_Y = \{X \in \mathcal{Z}^n | X \in P, Y = [F_k(X) + f_k]_{1 \le k \le q}\}$$

This parametric polytope has the following properties:

- *(1) $\forall\ Y_1, Y_2 \in \mathcal{Z}^q$, P_{Y_1} and P_{Y_2} are supported by parallel subspaces.*
- *(2) $\forall\ Y \in \mathcal{Z}^q$, $\bigcup P_Y = P$.*
- *(3) If $P_Y \neq \emptyset$, $linmap(P_Y) = Y$.*

P_Y is called the partitioning polytope of P through linmap.

Proof. (1) From a geometrical point of view, the points of any non-empty set P_Y characterize an hyperplane H_Y of dimension q, defined by $\{X \in \mathcal{Z}^q \mid [F_k(X) + f_k]_{1 \le k \le q} = Y\}$. Hence, two such hyperplanes H_{Y_1} and H_{Y_2} are parallel, containing respectively P_{Y_1} and P_{Y_2}.

(2) Any point X of P results in a point $Y \in \mathcal{Z}^q$ through *linmap*. Conversely, given a point $Y \in \mathcal{Z}^q$, the set P_Y defines all the points X of P resulting in Y through *linmap*. Hence, P can be partitioned as a union of sets P_Y, since all the reached values Y are considered (Y is any point in $\mathcal{Z}^q$), and is fully defined as the union of all the sets P_Y.

(3) This property is obvious from the definition of P_Y. □

Considering any polytope $P_{Y_0} \neq \emptyset$, its number of integer points characterizes the number of integer points of P resulting in Y_0 through *linmap*. Since our previously recalled counting method allows the computation of the Ehrhart polynomials of the parametric polytope P_Y, we get expressions of this number of points depending on Y and defined on several adjacent domains D of its values.

Since P is a bounded set of integer points, the set of all the values Y_0 reached through *linmap* is also bounded ; so is the union of the adjacent domains associated with the Ehrhart polynomials of P_Y. Unfortunately, these domains also contain values which are not reached through *linmap*. Hence, the problem is to count exclusively the points of this union which are associated with reached values Y_0.

Example 2. (continued) The iteration space of the loop nest is modeled by the polytope $P = \{(i,j) \in \mathcal{Z}^2 | 1 \leq i \leq 8, 1 \leq j \leq 5\}$. The parametric polytope characterizing points of P resulting in a given value $y = 6i + 9j - 7$, is defined by $P_y = \{(i,j) \in \mathcal{Z}^2 | 1 \leq i \leq 8, 1 \leq j \leq 5, y = 6i + 9j - 7\}$.

Using our method, the Ehrhart polynomials of P_y are computed. The answer consists in three adjacent domains D_1, D_2 and D_3 of the parameter values, each are associated with an Ehrhart polynomial $EP_1(y)$, $EP_2(y)$ and $EP_3(y)$:

- $D_1 = \{y \in \mathcal{Z} | 8 \leq y \leq 44\}$,
  $EP_1(y) = \left[0, 0, \frac{1}{18}\right] y + \left[0, 0, -\frac{1}{9}, 0, 0, -\frac{5}{18}, 0, 0, \frac{5}{9}, 0, 0, -\frac{11}{18}, 0, 0, \frac{2}{9}, 0, 0, \frac{1}{18}\right]$
- $D_2 = \{y \in \mathcal{Z} | 44 \leq y \leq 50\}$,
  $EP_2(y) = [0, 0, 3, 0, 0, 2]$
- $D_3 = \{y \in \mathcal{Z} | 50 \leq y \leq 86\}$,
  $EP_3(y) = \left[0, 0, -\frac{1}{18}\right] y + \left[0, 0, \frac{46}{9}, 0, 0, \frac{95}{18}, 0, 0, \frac{49}{9}, 0, 0, \frac{83}{18}, 0, 0, \frac{52}{9}, 0, 0, \frac{89}{18}\right]$

For example, the array element `a(41)` is referenced 2 times by the loop: since $41\ mod\ 3 = 2$ (the third value of the first periodic coefficient is selected) and $41\ mod\ 18 = 5$ (the sixth value of the second periodic coefficient is selected), we have $EP_1(41) = \frac{41}{18} - \frac{5}{18} = 2$. □

At this step of the presentation, a resolution could already have been proposed. It consists in counting the number of non-zero values resulting from the evaluations of the Ehrhart polynomials, for all the values in the union of the domains. But this method is very expensive, since its cost depends on the number of points in the domains, i.e. the number of possible reached values Y. We propose another method. In the following, some useful information is extracted from the Ehrhart polynomials of P_Y.

Let $EP(Y)$ be an Ehrhart polynomial defined on a domain D of possible reached values Y, where the periodic coefficients are expanded to a common size in each given dimension. Let S be the vector of the sizes of any periodic coefficient in all its q dimensions. We denote by $Y\ mod\ S$ the conjunction of all terms $y_k\ mod\ s_k$, $1 \leq k \leq q$. Let $V = (v_k)$ be a q-dimensional vector of integers such that for all $1 \leq k \leq q$, $0 \leq v_k < s_k$.

Since an Ehrhart polynomial is characterized by periodic coefficients, it can also be seen as a “periodic polynomial”. Hence, for any Y such that $Y\ mod\ S =$

V, a given polynomial occurs:

$$EP_V(Y) = \sum_{k_1=0}^{n} \sum_{k_2=0}^{n-k_1} \ldots \sum_{k_q=0}^{n-k_1-k_2-\ldots-k_{q-1}} c_{k_1,k_2,\ldots,k_q} y_1^{i_1} y_2^{i_2} \ldots y_q^{i_q}$$

where any $c_{k_1,k_2,\ldots,k_q}$ is a constant rational number.

Proposition 2. *If all the coefficients of $EP_V(Y)$ are equal to zero, the number of values Y in D reached through linmap and such that Y mod $S = V$ is equal to zero. Otherwise, it is equal to $N_V - N_R$, where N_V and N_R are defined as follows:*

- *N_V is the number of integer points in the polytope defined by:*

$$\{Y \in \mathcal{Z}^q | Y \in D, Y \ mod \ S = V\}$$

- *N_R is the number of integer roots R of $EP_V(Y)$ such that R mod $S = V$ and $R \in D$.*

Proof. It is obvious that if the occurring polynomial $EP_V(Y)$, associated with values Y such that $Y \ mod \ S = V$, has null coefficients, any evaluation of $EP_V(Y)$ results in zero. Otherwise, evaluations of $EP_V(Y)$ result in zero only for some specific values of Y, i.e. the roots of $EP_V(Y)$. Hence, if these roots belong to D, their values are not reached through *linmap* and must not be counted. These roots are representative of a phenomenon well-known by the "polytope model" specialists: a polytope projection results in quasi-regular lattices, disrupted by some "bad holes". These prevent to model any projection by its convex hull, making analysis and transformations more difficult.

For all other values Y which are not equal to these roots, evaluations of $EP_V(Y)$ result in strictly positive integers. Hence, the number of these values is the number of values Y in D reached through *linmap* and such that $Y \ mod \ S = V$. □

Since the domains D of parameters values are adjacent, they must first be transformed into disjoint ones. This is done easily by replacing some large inequalities by strict inequalities. The following result comes.

Proposition 3. *The total number of values reached through linmap is obtained by achieving the previously described counting for all vectors V on each disjoint domain D, and by summing all the results.*

So, our method can be summarized in the following steps:

1. Compute the Ehrhart polynomials of

$$P_Y = \{X \in \mathcal{Z}^n | X \in P, Y = [F_k(X) + f_k]_{1 \leq k \leq q}\}$$

and the associated domains of parameters values. Transform them into disjoint domains.

2. On each domain of parameters values D and for all vectors V, if the associated polynomial $EP_V(Y)$ is not equal to zero for any value of Y, compute $N_V - N_R$.
3. The sum of all the results obtained in 2 is the number of values reached through $linmap$.

The last remaining difficulty is to count integer points in the polytopes defined with modulo constraints $Y \ mod \ S = V$ (proposition 2).

Proposition 4. *Let C be a system of constraints defining integer points $Y = (y_k)$, whose non-linear ones are of the form $y_k \ mod \ s_k = v_k$, where s_k and v_k are integer constants, $0 \leq v_k < s_k$.*

Let C' be the system of linear constraints defining integer points $Z = (z_k)$, deduced from C, where any non-linear constraint $y_k \ mod \ s_k = v_k$ has been deleted and any y_k, in all the other constraints, has been replaced with $s_k z_k + v_k$.

The number of integer points Y defined by C is equal to the number of integer points Z defined by C'.

Proof. Let y_0 be an integer satisfying $y_0 \ mod \ s = v$. This equality implies that there exists an integer z_0 such that $y_0 = sz_0 + v$. Hence, y_0 can be replaced with $sz_0 + v$.

Making this substitution in a system of constraints allows anticipating the satisfaction of $y \ mod \ s = v$. The resulting system C' defines values z satisfying a system of linear constraints. The values y defined by C can be deduced from C' by applying to all z the bijective transformation $sz + v$. Hence, the number of values z satisfying C' is equal to the number of values y satisfying C. □

Example 3. (continued) The domains of the parameters values are transformed into disjoint domains: $D_1 = \{y \in \mathcal{Z} | 8 \leq y \leq 44\}$, $D_2 = \{y \in \mathcal{Z} | 44 < y < 50\}$ and $D_3 = \{y \in \mathcal{Z} | 50 \leq y \leq 86\}$. On each domain D_p and for every extracted polynomial $EP_v(y)$ from $EP_p(y)$, such that it is not equal to zero for any value of y, we define a polytope whose integer points have to be counted. It yields the following polytopes:

$$D_1 \begin{cases} \{8 \leq y \leq 44, y \ mod \ 18 = 2\} \\ \{8 \leq y \leq 44, y \ mod \ 18 = 5\} \\ \{8 \leq y \leq 44, y \ mod \ 18 = 8\} \\ \{8 \leq y \leq 44, y \ mod \ 18 = 11\} \\ \{8 \leq y \leq 44, y \ mod \ 18 = 14\} \\ \{8 \leq y \leq 44, y \ mod \ 18 = 17\} \end{cases} \quad D_2 \begin{cases} \{44 < y < 50, y \ mod \ 6 = 2\} \\ \{44 < y < 50, y \ mod \ 6 = 5\} \end{cases}$$

$$D_3 \begin{cases} \{50 \leq y \leq 86, y \ mod \ 18 = 2\} \\ \{50 \leq y \leq 86, y \ mod \ 18 = 5\} \\ \{50 \leq y \leq 86, y \ mod \ 18 = 8\} \\ \{50 \leq y \leq 86, y \ mod \ 18 = 11\} \\ \{50 \leq y \leq 86, y \ mod \ 18 = 14\} \\ \{50 \leq y \leq 86, y \ mod \ 18 = 17\} \end{cases}$$

Application of the transformation described in proposition 4 yields the following polytopes:

$$D_1 \begin{cases} \{1 \leq z \leq 2\} \\ \{1 \leq z \leq 2\} \\ \{0 \leq z \leq 2\} \\ \{0 \leq z \leq 1\} \\ \{0 \leq z \leq 1\} \\ \{0 \leq z \leq 1\} \end{cases} D_2 \begin{cases} \{7 < z < 8\} \\ \{7 \leq z \leq 7\} \end{cases} D_3 \begin{cases} \{3 \leq z \leq 4\} \\ \{3 \leq z \leq 4\} \\ \{3 \leq z \leq 4\} \\ \{3 \leq z \leq 4\} \\ \{2 \leq z \leq 4\} \\ \{2 \leq z \leq 3\} \end{cases}$$

Since the dimension of the array referenced by the loop is one, counting the number of integer points in these polytopes is obvious. But in the general case, more complicated definitions occur. Hence, counting the number of points in these polytopes yields the use of our counting method on Ehrhart polynomials, applied to non-parametric polytopes, as described in the first section.

In our example, summing all the values N_v on each domain D_p results in $2+2+3+2+2+2+0+1+2+2+2+2+3+2=27$.

For all non-zero polynomials $EP_v(y)$, their roots are computed in order to determine N_r. For example, on domain D_1, the following polynomials occur:

$$D_1 \begin{cases} if\ y\ mod\ 18 = 2, & EP_2(y) = \frac{1}{18}y - \frac{1}{9} & (1) \\ if\ y\ mod\ 18 = 5, & EP_5(y) = \frac{1}{18}y - \frac{5}{18} & (2) \\ if\ y\ mod\ 18 = 8, & EP_8(y) = \frac{1}{18}y + \frac{5}{9} & (3) \\ if\ y\ mod\ 18 = 11, & EP_{11}(y) = \frac{1}{18}y - \frac{11}{18} & (4) \\ if\ y\ mod\ 18 = 14, & EP_{14}(y) = \frac{1}{18}y + \frac{2}{9} & (5) \\ if\ y\ mod\ 18 = 17, & EP_{17}(y) = \frac{1}{18}y + \frac{1}{18} & (6) \end{cases}$$

The computation of the roots of these polynomials can be done using, for example, the function `solve` provided in *Maple*. We get the following results: one valid root is found (11), belonging to D_1. Moreover, one other valid root is found on D_3 (83). Hence, two has to be subtracted to the first counting determined above.

It yields that the number of distinct memory locations touched by the loop nest is equal to 25. This result is the same as the one obtained by W. Pugh in [15]. □

Example 4. (Application to parametric loop tiling) In order to show the usefulness of such a computation, it is used in the following for *parametric loop tiling*. Let us modify the loop nest by introducing a size parameter p as follows:

```
for i := 1 to 8 do
    for j := 1 to p do
        a(6i+9j-7) = a(6i+9j-7) + 5 ;
```

An exact dependence analysis, as described in [5], yields the dependence vector $(3, -2)$. This vector is also called a *reuse vector* in [11] and [12]. Hence, any iteration $(i+3, j-2)$ uses the array element computed during the iteration (i, j).

A current objective, when mapping such a loop nest on a distributed memory multiprocessor, is to minimize the number of occurring communications. Therefore, mapping on a single processor any iterations (i, j) and $(i+3, j-2)$ yields a communication-free computation of the loop nest and avoids the problem of cache coherence, since the array element $a(6i+9j-7)$ is stored in the processor cache. Moreover, this allows to achieve temporal locality.

A current method is to divide the iteration space into blocks, or tiles, and map them onto the processors. Good shape and size of the tiles allow communication-free blocks and optimized use of the processor cache. As shown in [11], the optimal tile size is difficult to predict. We show in this simple example that our method helps for this prediction.

In our example, communication-free blocks are obtained by choosing a parallelogram shape of the tiles such that one side is generated by the dependence vector $(3, -2)$. Since the upper and lower bounds of index i are constants, the height of the tiles are fixed to the height of the iteration space along the i axis.

Since one tile is mapped onto one processor, array elements accessed by its iterations must fit the cache in order to achieve an optimal use of it. This number of accessed elements is connected with the width of the tiles along the j axis. So the question is, given a cache size of s bytes, what is the width w of the tiles yielding the best use of the cache. A parameterized tile is represented on figure 1, yielding the computation of a loop on any processor indexed $(j_0 \div (w+1))+1$, such that the innermost loop enumerates all the points of a line segment generated by the dependence vector $(3, -2)$, and the outermost loop enumerates all such lines.

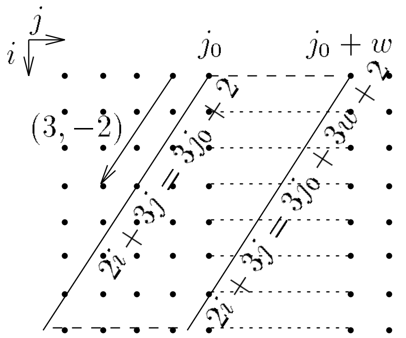

Fig. 1. A parameterized tile

This parameterized tile can be defined as a parametric polytope $T_{j_0,w} = \{(i, j) \in \mathcal{Z}^2 \mid 1 \le i \le 8, 1 \le j_0, 3j_0 + 2 \le 2i + 3j \le 3j_0 + 3w + 2\}$. In order to determine the minimum parametric cache size s_w which is necessary to store all the accessed elements, we use our method to compute the number of distinct memory locations touched by the parameterized tile.

By adding the constraint $y = 6i + 9j - 7$, we define the parametric polytope $T_{j_0,w,y}$ for which the Ehrhart polynomial is computed: $D = \{1 \leq j_0 \leq p - w, 9j_0 - 1 \leq y \leq 9j_0 + 9n - 1\}, EP(j_0, w, y) = [0,0,2,0,0,3,0,0,3]_y$. Then, the number of integer points in $D \cap \{y \ mod \ 9 = 2\}$, $D \cap \{y \ mod \ 9 = 5\}$ and $D \cap \{y \ mod \ 9 = 8\}$ are computed, yielding $w + w + (w+1) = 3w + 1$ distinct memory locations touched. Hence, assuming that any array element is coded on 8 bytes (double float), the minimum cache size allowing an optimal cache utilization, without any occurring cache miss, is $s_w = (3w+1) \times 8$ bytes. For example, a 1KB sized cache yields an optimal tiling characterized by $w = 42$. □

Since the linear case has been resolved, we now focus on non-linear mappings of polytopes in order to be able to compute the number of cache lines touched by a loop from a given mapping function, and the number of processors resulting from distribution of array elements.

4 Counting in non-linear mappings of polytopes

As said in section 2, a mapping function to cache lines or a processor distribution can be modeled by the general non-linear mapping function:

$$\begin{array}{l} \qquad\qquad\quad P \longrightarrow M \\ nonlinmap : X \longmapsto Y = [(((F_k(X) + f_k) \odot a_{1,k} + b_{1,k}) \odot a_{2,k} + b_{2,k}) \ldots]_{1 \leq k \leq q} \end{array}$$

where F_k is a linear integer combination on X, f_k, $a_{i,k}$ and $b_{i,k}k$ are integer constants and $\odot$ symbolizes $\div$, mod, or any common arithmetic operation.

Applying the same reasoning as in the previous section, such a non-linear mapping function yields the consideration of a parametric polytope P_Y defined with some non-linear constraints. However, such constraints can be supported due to some elementary transformations.

Proposition 5. *Let C be a system of constraints defining integer points $X \in \mathcal{Z}^n$, whose non-linear ones contain terms of the form $w_k \div a_k$ or $w_k \ mod \ a_k$, where w_k is an affine linear combination of X and a_k is a positive integer constant.*

The system C' of linear constraints deduced from C, by application of the following rules, defines the same set of integer points:

- *(1) Any term $w_k \div a_k$ is deleted and replaced with a new variable z_k and both constraints $w_k \geq a_k z_k$ and $w_k < a_k(z_k + 1)$ are added.*
- *(2) Any term $w_k \ mod \ a_k$ is deleted and replaced with $w_k - z_k$ and both constraints $w_k \geq a_k z_k$ and $w_k < a_k(z_k + 1)$ are added.*

Proof. W. Pugh introduces the same transformations in [15]. By substituting $w_k \div a_k$ for z_k, both transformations can be easily verified.

Example 5. (continued) Let us consider a mapping from array elements $a(y)$ to cache lines $[(y-1) \div 16]$. Counting the number of cache lines touched by the loop consists in counting the number of distinct values of $((y-1) \div 16)$ reached for all the $a(y)$'s $(y = 6i + 9j - 7)$ touched by the loop.

In a similar way as above, we compute first the Ehrhart polynomials giving the number of iterations touching a given cache line $y = [((6i+9j-7)-1) \div 16]$. Hence, we compute the Ehrhart polynomials associated with $P_y = \{(i,j) \in \mathcal{Z}^2 | 1 \leq i \leq 8, 1 \leq j \leq 5, y = [(6i+9j-8) \div 16]\}$, which is transformed following our transformation rules to $P_y = \{(i,j) \in \mathcal{Z}^2 | 1 \leq i \leq 8, 1 \leq j \leq 5, 6i+9j-8 \geq 16y, 6i+9j-8 < 16(y+1)\}$.

The computation of $EP(y)$ results in:

- $D_1 = \{0\}$, $EP(0) = 2$
- $D_2 = \{1\}$, $EP(1) = 8$
- $D_3 = \{2\}$, $EP(2) = 11$
- $D_4 = \{3\}$, $EP(3) = 11$
- $D_5 = \{4\}$, $EP(4) = 7$
- $D_6 = \{5\}$, $EP(5) = 1$

Finally, application of the same continuation as in the previous section is simple and yields an answer of 6. Hence, 6 cache lines have to be loaded before the execution of this loop and have to be eventually unloaded for some further computations on the processor. Other mapping functions could result in a lower number of cache lines, yielding a lower loading/unloading spending time. □

Example 6. (Application to parametric loop tiling) Let us reconsider the parametric loop tiling of the previous section. Optimized spatial locality is achieved by mapping the accessed array elements to the cache lines such that array elements mapped to the same cache line are accessed during successive iterations. Two successive iterations (x, y) and $(x, y+1)$ of the innermost loop access the same array element $a(3x+5)$. But two successive iterations x and $x+1$ of the outermost loop access respectively array elements $a(3x+5)$ and $a(3x+8)$.

Assuming cache lines of 32 bytes storing 4 double floats, the mapping stating that a reference to array element $a(y)$ references cache line $[((y-7) \div 3+1) \div 4+1]$ ensures optimal spatial locality. Assuming a 1KB sized cache and a 42 tile width, the number of cache lines touched with this mapping function is minimum (32), since the cache is entirely loaded with all the data requested for the tile. A greater number of cache lines would have increased the loading/unloading spending time and lead to cache misses. A lower number of cache lines also would have lead to cache misses, since such a related mapping function overwrites some useful data during the loading process. □

5 Conclusion

We have presented a general method allowing the computation of the number of memory locations and cache lines touched by a loop from a given mapping function, or the number of processors resulting from a given distribution of array elements. While reaching these objectives, the general problem of the number of integer points resulting from any linear or non-linear mapping of a polytope has been resolved. It has many other applications in the parallelization of nested loops, as the number of allocated processors from linear or non-linear allocation of iterations, in the same way as in [2, 6], or the latency of a parallel loop from a given multi-dimensional schedule.

Since we denote by e the complexity given in section 1 and concerning our exact counting method, the complexity of our counting method in linear and non-linear mappings is $\mathcal{O}(c(ve+1))$, where c denotes the number of non-zero polynomials $EP_V(Y)$: the same work is done for all the domains of parameters values, where for each non-zero polynomial, a counting is done using our counting method ; finally, the roots of each of these polynomials are computed.

Exact answers, in the case of symbolic analysis and transformation of programs, enable to avoid rough approximation that can yield sizeable errors: approximating a polynomial of degree 3 only by its term of degree 3 leads to an error of $\mathcal{O}(n^2)$, which is unacceptable in most cases. Moreover, our method is not more expensive than methods giving "acceptable" approximations.

Ferrante, Sarkar and Thrash in [8] present methods for computing the number of distinct memory locations and cache lines accessed by a loop nest. However, their methods cannot handle unions of polytopes, do not compute symbolic answers, often compute only approximations and use expensive methods to handle the cache lines touched by a set of references.

W. Pugh's work in [15], which has been already mentioned in section 1, is based on a noticeable different approach, considering a set of constraints as Presburger formulas. The presented technique consists in a set of formulas manipulations, each dedicated to simplify the initial set of constraints. Applying all these tools would be expensive for important sets of constraints. Moreover, their application is not always automatic: it consists in some cases in adding auxiliary variables, and needs the use of standard formulas for sums of powers (the author expects it will be sufficient to hard code the formulas for powers up to 10). This technique also tends to complicated answers when considering rational polytopes. However, the used manipulations of formulas are included in an ambitious package, the *Omega calculator*, dedicated to the resolution of a large field of problems.

The computation method presented in this paper will be soon added to our implementation on Ehrhart polynomials, in order to complete the objective of an analyzing tool devoted to optimizing parallelization of programs.

For future works, we believe that many other computations, which are useful in automatic parallelization, can be achieved through Ehrhart polynomials. Hence, we are currently studying the construction of some other opportune models and methods.

References

1. A. Agarwal, D.A. Kranz and V. Natarajan. Automatic Partitioning of Parallel Loops and Data Arrays for Distributed Shared-Memory Multiprocessors. *IEEE Trans. on Parall. and Distrib. Syst.*, Vol. 6, NO. 9, Sept. 1995.
2. Ph. Clauss: An efficient allocation strategy for mapping affine recurrences into space and time optimal regular processor arrays. In C. Jesshope, V. Jossifov and W. Wilhelmi editors, Int. Conf. PARCELLA'94, pages 257-266, Potsdam, Germany, Sept. 1994, Akademie Verlag.
`http://icps.u-strasbg.fr/pub-94/pub-94-15.ps.Z`
3. Ph. Clauss: Counting Solutions to Linear and Nonlinear Constraints through Ehrhart polynomials: Applications to Analyze and Transform Scientific Programs. *10th ACM Int. Conf. on Supercomputing*, Philadelphia, May 1996. Also available as Research Report ICPS 96-03.
`http://icps.u-strasbg.fr/pub-96/pub-96-03.ps.gz`
4. Ph. Clauss and V. Loechner: Parametric Analysis of Polyhedral Iteration Spaces. *IEEE Int. Conf. on Application Specific Array Processors, ASAP'96*, Chicago, Illinois, August 1996. Also available as Research Report ICPS 96-04.
`http://icps.u-strasbg.fr/pub-96/pub-96-04.ps.gz`
5. Ph. Clauss and V. Loechner: Parametric Analysis of Polyhedral Iteration Spaces (extended version). To appear in *Journal of VLSI Signal Processing*, Kluwer Academic Pub., 1997.
6. Ph. Clauss and G.-R. Perrin: Optimal mapping of systolic algorithms by regular instruction shifts. Int. Conf. on Application-Specific Array Processors, ASAP'94, pages 224-235, San Francisco, California, IEEE Computer Society Press.
`http://icps.u-strasbg.fr/pub-94/pub-94-08.ps.Z`
7. E. Ehrhart: *Polynômes arithmétiques et Méthode des Polyèdres en Combinatoire.* International Series of Numerical Mathematics, vol.35, Birkhäuser Verlag, Basel/Stuttgart, 1977.
8. J. Ferrante, V. Sarkar and W. Thrash: On estimating and enhancing cache effectiveness. *Advances in Languages and Compilers for Parallel Processing*, pages 328-343. The MIT Press, 1991.
9. D. -C. R. Ju, J.-F. Collard and K. Oukbir: Probabilistic Memory Disambiguation and its Application to Data Speculation. Research Report PRiSM 96-010, 1996.
`ftp://ftp.prism.uvsq.fr/pub/reports/1996/1996.010.ps.gz`
10. C. Koelbel, D. Loveman, R. Schreiber, G. Steel, M. Zosel: The High Performance Fortran Handbook. The MIT Press. 1994.
11. M. S. Lam, E. E. Rothberg and M. E. Wolf: The cache performance and optimizations of blocked algorithms. In *Proc. 4th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, ASPLOS IV*, pages 63-74, Palo Alto, California, April 1991.
12. W. Li: Compiler Optimizations for Cache Locality and Coherence. Technical report 504, University of Rochester, Department of Computer Science, April 1994.
13. V. Loechner and D. K. Wilde: Parameterized polyhedra and their vertices. Research Report ICPS 96-09, 1996.
`http://icps.u-strasbg.fr/pub-96/pub-96-09.ps.gz`
14. W. Pugh: The Omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102-114, August 1992.

15. W. Pugh: Counting Solutions to Presburger Formulas: How and Why. *Proc. of the 1994 ACM SIGPLAN Conference on Programming Language Design and Implementation*, 1994.
16. H. S. Stone and D. Thiebaut: Footprints in the cache. *Proc. ACM SIGMETRICS 1986*, pp. 4-8, May 1986.