

Experiences with Enumeration of Integer Projections of Parametric Polytopes

Sven Verdoolaege *Kristof Beyls*
Maurice Bruynooghe *Francky Catthoor*

Report CW 395,



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Experiences with Enumeration of Integer Projections of Parametric Polytopes

Sven Verdoolaege *Kristof Beyls*
Maurice Bruynooghe *Francky Catthoor*

Report CW 395,

Department of Computer Science, K.U.Leuven

Abstract

Many compiler optimization techniques depend on the ability to calculate the number of integer values that satisfy a given set of linear constraints. This count is a function of the symbolic parameters that typically appear in the constraints and it is known as the enumeration of a parametric polytope. In an extended problem, some of the variables that appear in the constraints may be existentially quantified and then the enumerated set corresponds to the projection of the integer points in a parametric polytope. We refer to this problem as the enumeration of the integer projection of a parametric polytope.

This paper shows how the enumeration of the integer projection of parametric polytopes can be reduced to the enumeration of parametric polytopes. Two approaches are described and experimentally compared. Differences are small, but both methods can solve problems that previously were considered very difficult to solve analytically.

Keywords : Ehrhart quasi-polynomial, integer projection, parametric polytope, polyhedral model

CR Subject Classification : D.3.4 Compilers

Experiences with Enumeration of Integer Projections of Parametric Polytopes

Sven Verdoolaege¹, Kristof Beyls², Maurice Bruynooghe¹, and Francky Catthoor³

¹ K.U.Leuven Departement Computerwetenschappen, Celestijnenlaan 200A, B-3001 Leuven, Belgium

² Departement of Electronics and Information Systems, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium

³ IMEC, Kapeldreef 75, B-3001 Leuven, Belgium

1 Introduction

Many compiler optimization techniques require the enumeration of objects of a certain class. Examples include counting the number of calculations, accessed memory locations or statement executions in a loop nest or parts thereof [6, 21, 23, 30, 31, 40]; calculating the number of cache misses in a loop [12, 16, 25]; computing the number of dynamically allocated bytes [11]; enumerating the number of live array elements at a given iteration (i, j) [29, 42]; counting how many parallel processing elements can be used when executing a loop on an FPGA [5, 22, 26]; computing the amount of communication for a given schedule of parallel tasks on a distributed computing system [9, 27, 39].

These counts are used to drive optimizations such as increasing parallelism [40], minimizing memory size [1, 2, 29, 40, 42], estimating worst case execution time [30], increasing cache effectiveness [6, 16], high-level transformations for DSP applications [23], converting software loops into parallel hardware implementations [5, 18, 22, 26, 40] and minimizing communication overhead in distributed applications [9, 27, 39].

In many of these analyses and optimizations, the objects/events to be counted are modeled as the integer solutions to systems of linear inequalities, i.e., as the elements of a set $S = \{\mathbf{x} \in \mathbb{Z}^d \mid A\mathbf{x} + \mathbf{c} \geq \mathbf{0}\}$, with $A \in \mathbb{Z}^{n \times d}$ and $\mathbf{c} \in \mathbb{Z}^n$. Furthermore, many of these analyses need the count in function of a vector of parameters \mathbf{p} (e.g., in the presence of symbolic loop bounds):

$$S_{\mathbf{p}} = \{\mathbf{x} \in \mathbb{Z}^d \mid A\mathbf{x} + B\mathbf{p} + \mathbf{c} \geq \mathbf{0}\}. \quad (1)$$

An efficient algorithm which computes the function which maps specific values of \mathbf{p} to the number of elements in $S_{\mathbf{p}}$ has recently been presented in [41]. In this paper, we present an extension to a yet more general counting problem, where some of the variables can be existentially quantified. We propose a general solution for counting the number of elements (in function of parameters \mathbf{p}) for sets which can be expressed in the form

$$\left\{ \mathbf{x} \in \mathbb{Z}^d \mid \exists \mathbf{y} \in \mathbb{Z}^{d'} : A\mathbf{x} + D\mathbf{y} + B\mathbf{p} + \mathbf{c} \geq \mathbf{0} \right\}. \quad (2)$$

Computing the number of elements in such a set is, amongst others, needed in the program analyses described in [1, 2, 5, 6, 9, 12, 16, 26, 27, 39, 42]. Practical examples are discussed in Section 4.

1.1 Example

Consider an example adapted from [14] (Figure 1). Assume we want to know the total number of array elements accessed by the statement in the inner loop as

a function of the symbolic parameter p . This problem is equivalent to counting the number of elements in the set

$$S_p = \{l \in \mathbb{Z} \mid \exists i, j \in \mathbb{Z} : l = 6i + 9j - 7 \wedge 1 \leq j \leq p \wedge 1 \leq i \leq 8\}, \quad (3)$$

which can be written in the same form as (2):

$$\left\{ l \in \mathbb{Z} \mid \exists \begin{pmatrix} i \\ j \end{pmatrix} \in \mathbb{Z}^2 : \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} l + \begin{pmatrix} -6 & -9 \\ 6 & 9 \\ 0 & 1 \\ 0 & -1 \\ 1 & 0 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} p + \begin{pmatrix} 7 \\ -7 \\ -1 \\ 0 \\ -1 \\ 8 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right\}.$$

Figure 2 shows the array elements that are accessed for $p = 3$. These elements do not correspond to the integer points in a polytope. Even after shifting and scaling by 3 it still contains two “holes” (marked by \times on the figure). These holes complicate the enumeration.

```

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

```

Fig. 1. Example Program

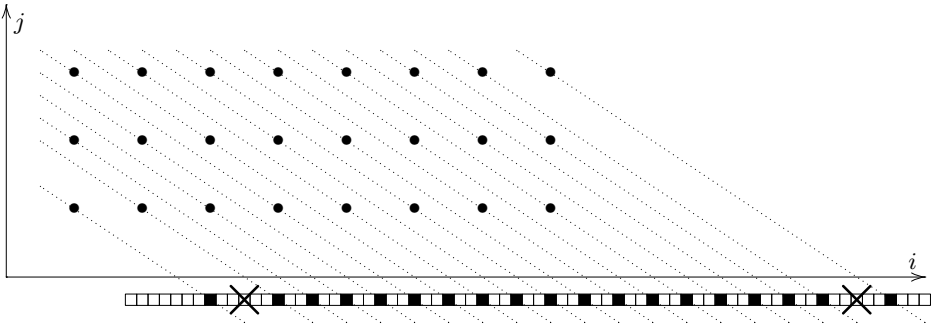


Fig. 2. Array elements accessed for $p = 3$

For $p = 3$, the set S_p contains 19 points, see Figure 2. In general, the number of points in S_p can be described by the function

$$c(S_p) = \begin{cases} 8 & \text{if } p = 1 \\ 3p + 10 & \text{if } p \geq 2 \end{cases}.$$

Different polynomials represent the count in different regions of the parameter space. Following [38], we call these regions *chambers*. In general, the count in each chamber is described by a *quasi-polynomial*, as defined in Section 2.

1.2 Related Work

The solution to a counting problem is called the enumerator of the set of constraints. When there are no parametric variables in the counting problem, the

enumerator is an integer; otherwise the enumerator is a function which maps the values of the parametric variables to an integer. Different applications for different types of constraints has led to different proposals. Below, when we refer to the complexity of algorithms, we always mean for a fixed number of variables. To the best of our knowledge, all methods are exponential in the number of variables.

Linear inequalities. Barvinok [4] was first to propose an algorithm computing enumerators for sets of linear inequalities that is polynomial-time.

Parametric linear inequalities. Ehrhart [19] showed that the general form of the enumerator for a certain form of parametric linear inequalities with a single parameter is a quasi-polynomial. Clauss et al. [15] extended this theory to handle the more general form shown in Equation (1), albeit in exponential time complexity. De Loera et al. [17] implemented Barvinok’s [4] algorithm for counting linear inequalities in polynomial time and extended it to compute the Ehrhart series corresponding to the dilation nP of a polytope P . Finally, Verdoolaege et al. [41] obtained a polynomial time algorithm for the counting problem of Equation (1).

Linear inequalities with existential variables. Barvinok and Woods [3] propose a polynomial time algorithm. No implementation has been reported, and the extension for parameters is not obvious.

Parametric linear inequalities with existential variables. Boulet [9] proposes to compute the enumerator of a set of parametric linear inequalities with existential variables in two steps. First, parametric integer programming (PIP) [20] is used to eliminate the existential variables, after which Clauss’s [15] method is used to enumerate the resulting set of linear inequalities. However, no extensive evaluation has been reported and the appendix in [10] indicates that the method cannot compute the enumerator fully automatically. Meister [32] proposes a similar technique using his more general periodic polyhedra instead of PIP. No implementation has been reported. Clauss [13] proposed a method (recently implemented [37]) based on “thick facets” that works for a single existential variable.

Non-parametric Presburger formula. Presburger formulas consist of linear inequalities of integer variables, combined by existential and universal quantifiers, disjunction, conjunction and negation. ($\exists, \forall, \vee, \wedge, \neg$). Recently, two methods [8, 33] have been presented to count the number of integer solutions to these formulas, both based on representing the formula as a finite automaton.

Parametric Presburger formula. In [35], a number of rewrite rules are proposed to compute the enumerator of a parametric Presburger formula. However, the rules seem ad-hoc and have apparently not been implemented to date. Therefore, it is hard to evaluate their usefulness in practice.

This paper investigates the combination of PIP with our method for parametric polytopes [41]. This combination can handle the parametric counting problems with existential variables reported in [2, 6, 9, 12] that were previously considered difficult or even unsolvable. Since PIP is worst-case exponential, we also investigate an alternative method that uses a number of simple polynomial rewriting rules to eliminate existential quantifiers. While all existential quantifiers are eliminated in our experiments, some could remain. In that case, PIP can be used to solve the reduced problem. Theoretically, parametric Presburger formulas can be transformed into a disjoint union of sets of the form (2). For the majority of the parametric Presburger formulas we considered, this transformation could be performed efficiently and automatically by the Omega library.

Section 2 discusses the necessary background on parametric polytopes and enumerators. The two extensions for handling existential variables are presented

in Section 3. Section 4 briefly describes the main categories of compiler optimizations that can profit from using the presented techniques. An experimental evaluation is performed in Section 5, and concluding remarks follow in Section 6.

2 Parametric Polytopes

Before tackling integer projections of parametric polytopes, we review the results on enumeration of parametric polytopes. We refer to [41] for the details.

Definition 1. A rational polyhedron $P \in \mathbb{Q}^d$ is a set of rational d -dimensional vectors \mathbf{x} defined by linear inequalities

$$P = \{ \mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} + \mathbf{c} \geq \mathbf{0} \}, \quad (4)$$

with $A \in \mathbb{Z}^{m \times d}$ and $\mathbf{c} \in \mathbb{Z}^m$.

Definition 2. A rational polytope is a bounded rational polyhedron.

Definition 3. A rational parametric polytope $P_{\mathbf{p}}$ with n parameters \mathbf{p} is a set of rational d -dimensional vectors \mathbf{x} defined by linear inequalities on \mathbf{x} and \mathbf{p}

$$P_{\mathbf{p}} = \{ \mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} + B\mathbf{p} + \mathbf{c} \geq \mathbf{0} \} \quad (5)$$

with $A \in \mathbb{Z}^{m \times d}$, $B \in \mathbb{Z}^{m \times n}$ and $\mathbf{c} \in \mathbb{Z}^m$, and such that for each fixed value \mathbf{p}_0 of \mathbf{p} , $P_{\mathbf{p}_0}$ defines a (possibly empty) rational polytope in \mathbb{Q}^d .

All the polyhedra in this paper are rational. If the parametrization of a polytope is clear from the context, we will omit the subscript \mathbf{p} . Note that the same equations that define a parametric polytope also define a $(d+n)$ -dimensional polyhedron in the combined data and parameter space.

$$P' = \{ (\mathbf{x}, \mathbf{p}) \in \mathbb{Q}^{d+n} \mid A\mathbf{x} + B\mathbf{p} + \mathbf{c} \geq \mathbf{0} \}$$

This polyhedron need not be bounded.

Definition 4. The enumerator $c(P_{\mathbf{p}})$ of a parametric polytope $P_{\mathbf{p}}$ is a function from the set of n -dimensional integer vectors \mathbb{Z}^n to the set of natural numbers \mathbb{N} .⁴ The function value at \mathbf{p}_0 , denoted $c(P_{\mathbf{p}}; \mathbf{p}_0)$, is the number of integer points in the polytope $P_{\mathbf{p}_0}$.

$$\begin{aligned} c(P_{\mathbf{p}}) : \mathbb{Z}^n &\rightarrow \mathbb{N} \\ \mathbf{p}_0 &\mapsto c(P_{\mathbf{p}}; \mathbf{p}_0) \\ c(P_{\mathbf{p}}; \mathbf{p}_0) &= \# (\mathbb{Z}^d \cap \{ \mathbf{x} \in \mathbb{Q}^d \mid A\mathbf{x} + B\mathbf{p}_0 + \mathbf{c} \geq \mathbf{0} \}) \end{aligned}$$

Definition 5. An (Ehrhart) quasi-polynomial is a polynomial in the lower integer parts (floors) of rational affine functionals.

Proposition 1 ([41]). For fixed dimensions d and n , an explicit representation of the enumerator of a parametric polytope can be computed in a time which is polynomial in the input size. In particular, the computed representation consists of a set of chambers,⁵ which form a subdivision of the parameter space, each with an associated quasi-polynomial. The size of this representation is also polynomial in the input size.

The input size is the number of bits needed to represent the input [36].

⁴ In [41], the symbol \mathcal{E} is used instead of c .

⁵ Chambers are also called validity domains in some publications.

Example 1. Consider the parametric polytope P_p

$$\{(x, y) \in \mathbb{Q}^2 \mid x + 3y \leq 8 \wedge x + 2y + 1 \leq 0 \wedge x + 2y + p \geq 0 \wedge x + 3p + 11 \leq 0\}.$$

Figure 3 shows P_p for different values of p . The number of points is given by

$$c(P_p) = \begin{cases} 5 & \text{if } p \geq 3 \\ -\frac{3}{4}p^2 + \frac{15}{4}p + \frac{1}{2}\lfloor \frac{1}{2}p \rfloor & \text{if } 1 \leq p \leq 2 \end{cases}.$$

This enumerator has two chambers: $\{p \mid p \geq 3\}$ and $\{p \mid 1 \leq p \leq 2\}$. The quasi-polynomial associated with the first chamber is a constant. For the second chamber, we obtain a polynomial in the lower integer parts of p and $\frac{1}{2}p$. Note that this is only one of the possible representations of $c(P_p)$. For this particular example, a much simpler representation exists with chambers $\{p \mid p \geq 2\}$ and $\{1\}$, and the constants 5 and 3 for the corresponding functions.

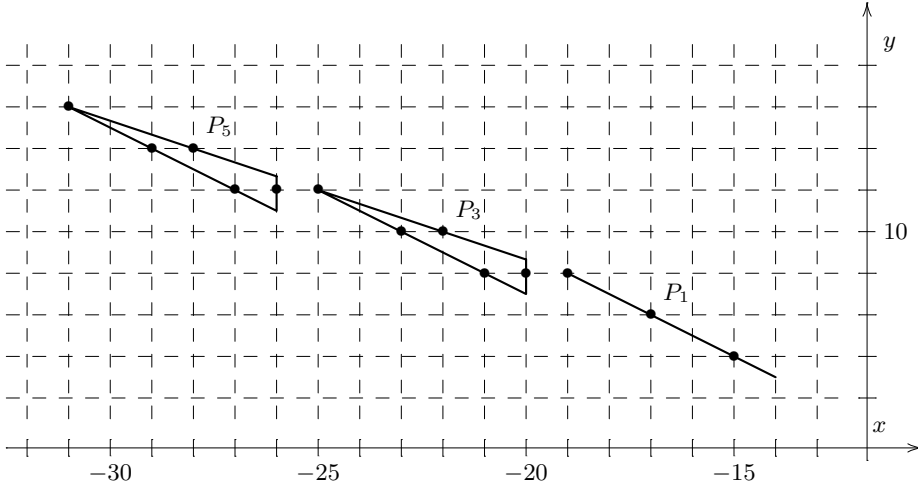


Fig. 3. Different instantiations of the parametric polytope from Example 1.

3 Existential Variables

In this section we consider the extension of the enumeration of parametric polytopes to the case where some of the variables that occur in the linear constraints are existentially quantified. The general form of these counting problems, given in Equation 2, is equivalent to

$$\#\pi_d \left(\mathbb{Z}^{(d+d')} \cap \left\{ (\mathbf{x}, \mathbf{y}) \in \mathbb{Q}^{(d+d')} \mid \mathbf{A}\mathbf{x} + \mathbf{D}\mathbf{y} + \mathbf{B}\mathbf{p} + \mathbf{c} \geq \mathbf{0} \right\} \right)$$

where π_d is the projection onto the first d dimensions. This parametric count corresponds to the number of points in the projection of the integer points in a parametric polytope, or integer projection of a parametric polytope for short.

Note that we cannot simply ignore the existential quantifier and count the number of points as if the set was a parametric polytope. Since for any particular value of \mathbf{x} there may be several values of \mathbf{y} that satisfy the constraints, we would be counting the same element several times. We also cannot simply project out the existential variables since there may exist values of \mathbf{x} in this projection for which there is no *integer* value of \mathbf{y} satisfying the constraints. If we project, for

example, P_5 in Figure 3 onto the x -axis, then this projection will contain the value -30 , while there is no integer y such that $(-30, y) \in P_5$.

We consider three techniques for eliminating existential variables. The first detects situations where an existential quantifier can be dropped without affecting the count of the set; the second detects situations where an existentially quantified variable is redundant and can be projected out without affecting the count; finally, the third technique searches for a split in two non-overlapping sets, each describing a simpler problem. Each of these techniques is polynomial in the input size (for fixed dimensions). However, a split does not always exist. In that case, one can fall back upon parametric integer programming to count the set. This always works but is worst-case exponential, even for fixed dimensions.

3.1 Elimination

Unique Existential Variables The existential quantifiers introduced by tools that automatically extract counting problems from source code can sometimes be redundant. This occurs when for each \mathbf{x} in the corresponding set, there is at most one y_i that satisfies the constraints. In such a case, the existential quantifier for y_i can be omitted without affecting the count of the set.

Many such cases can be detected when there is a constraint that involves y_i but none of the other existential variables $\bar{\mathbf{y}}$. Without loss of generality, we will assume the constraint establishes a lower bound on the variable y_i , i.e., it is of the form

$$n_l y_i + \langle \mathbf{a}_l, \mathbf{x} \rangle + \langle \mathbf{b}_l, \mathbf{p} \rangle + c_l \geq 0 \quad (6)$$

with $n_l \in \mathbb{N}$, and $\langle \mathbf{v}, \mathbf{w} \rangle$ the inproduct of \mathbf{v} and \mathbf{w} . Combining this constraint with an upper bound

$$-n_u y_i + \langle \mathbf{a}_u, \mathbf{x} \rangle + \langle \bar{\mathbf{d}}_u, \bar{\mathbf{y}} \rangle + \langle \mathbf{b}_u, \mathbf{p} \rangle + c_u \geq 0 \quad (7)$$

we obtain

$$-n_u (\langle \mathbf{a}_l, \mathbf{x} \rangle + \langle \mathbf{b}_l, \mathbf{p} \rangle + c_l) \leq n_u n_l y_i \leq n_l (\langle \mathbf{a}_u, \mathbf{x} \rangle + \langle \bar{\mathbf{d}}_u, \bar{\mathbf{y}} \rangle + \langle \mathbf{b}_u, \mathbf{p} \rangle + c_u). \quad (8)$$

The number of distinct integer values for $n_u n_l y_i$ is given by the upper bound minus the lower bound plus one. If this number is smaller than $n_u n_l$, then the two constraints admit at most one integer value for y_i . That is, if

$$n_l (\langle \mathbf{a}_u, \mathbf{x} \rangle + \langle \bar{\mathbf{d}}_u, \bar{\mathbf{y}} \rangle + \langle \mathbf{b}_u, \mathbf{p} \rangle + c_u) + n_u (\langle \mathbf{a}_l, \mathbf{x} \rangle + \langle \mathbf{b}_l, \mathbf{p} \rangle + c_l) + 1 \leq n_l n_u \quad (9)$$

for all integer values that satisfy the constraints, then y_i is uniquely determined by \mathbf{x} and \mathbf{p} and can therefore be treated as a regular variable, without existential quantification. It is independent of the other existential variables because of our assumption that one of the constraints does not involve these other variables. Condition (9) can easily be checked by adding the negation to the existing set of constraints and testing for satisfiability. Note that it is sufficient to find one such pair to be able to drop the existential quantification of the variable.

Example 2. Consider the set S

$$\{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} : x + 3y \leq 8 \wedge x + 2y + 1 \leq 0 \wedge x + 2y + p \geq 0 \wedge x + 3p + 11 \leq 0\}.$$

This is the same set that appeared in Example 1, except that y is now an existential variable. Since there is only a single existential variable, all constraints are independent of the “other existential variables”. Using $x + 2y + p \geq 0$ and $-x - 3y + 8 \geq 0$ as constraints, condition (9) yields

$$x + 3p + 17 \leq 6. \quad (10)$$

All elements of the set satisfy this constraint so we can remove the existential quantification and the set S is then exactly the same as the set P_p from Example 1.

Even if there is no single existential variable that is unique, some linear combination of existential variables may still be unique. To avoid enumerating all possible combinations, we only consider this case if we have two constraints that are “parallel in the existential space”, i.e., such that $\mathbf{d}_l = n_l \mathbf{d}$ and $\mathbf{d}_u = -n_u \mathbf{d}$ for some positive integers n_l and n_u and an integer vector \mathbf{d} with greatest common divisor (gcd) 1. We compute condition (9) from (6) and (7) with y_i replaced by $\langle \mathbf{d}, \mathbf{y} \rangle$ ($\bar{\mathbf{d}}_u$ is $\mathbf{0}$ in this case). If this condition holds, we perform a change of basis such that $y'_1 = \langle \mathbf{d}, \mathbf{y} \rangle$, which we now know to be unique. Such a change of basis can be obtained through transformation by the unimodular extension of \mathbf{d} [7].

Example 3. Consider the set S (3) from Section 1.1. This set satisfies the equality $l = 6i + 9j - 7$, which means that $2i + 3j$ is unique. Transforming this set using the unimodular extension of $\mathbf{d} = (2, 3)$

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ -1 & -1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

we obtain

$$S = \{ l \in \mathbb{Z} \mid \exists x, y \in \mathbb{Z} : l = 3x - 7 \wedge -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8 \}.$$

Since equation $l = 3x - 7$ provides an upper and a lower bound on x that are equal, Equation (9) is trivially satisfied and $\exists x$ can be removed. Since l is now redundant, we removed it for simplicity:

$$S' = \{ x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} : -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8 \}. \quad (11)$$

Redundant Existential Variables Consider once more a lower bound on the existential variable y_i :

$$n_l y_i + \langle \mathbf{c}_l, \mathbf{w} \rangle \geq 0,$$

where we used $\mathbf{c}_l := (\mathbf{a}_l, \bar{\mathbf{d}}_l, \mathbf{b}_l, c_l)$ and $\mathbf{w} := (\mathbf{x}, \bar{\mathbf{y}}, \mathbf{p}, 1)$ for brevity. Since we are only interested in integer values of y_i , this is equivalent to

$$n_u (n_l y_i + \langle \mathbf{c}_l, \mathbf{w} \rangle) + n_u - 1 \geq 0,$$

for any positive integer n_u . Similarly, for an upper bound we obtain

$$n_l (-n_u y_i + \langle \mathbf{c}_u, \mathbf{w} \rangle) + n_l - 1 \geq 0.$$

The range in (8) can therefore be expanded to

$$-n_u \langle \mathbf{c}_l, \mathbf{w} \rangle - n_u + 1 \leq n_u n_l y_i \leq n_l \langle \mathbf{c}_u, \mathbf{w} \rangle + n_l - 1.$$

If this range is larger than $n_u n_l$, i.e., if

$$n_l \langle \mathbf{c}_u, \mathbf{w} \rangle + n_u \langle \mathbf{c}_l, \mathbf{w} \rangle + n_l - 1 + n_u - 1 + 1 \geq n_l n_u, \quad (12)$$

then there is *at least* one integer value for each given value of the other variables. If this holds for all pairs of constraints, then variable y_i does not restrict the solutions in any way and can simply be eliminated. This is known as the Omega test [34]. Note that unlike the case of unique existential variables, the constraints need not be independent of the other variables.

Example 4. Consider the set

$$S = \{x \in \mathbb{Z} \mid \exists y \in \mathbb{Z} : -x - p \leq 2y \leq -x - 1 \wedge x \leq -11 \wedge -x + 1 \leq 3y \leq -x + 8 \wedge x + 3p + 10 \leq 0 \wedge p \geq 3\}.$$

This set is shown (■) in Figure 4. Pairwise combining the two upper and two lower bounds to form condition (12), we obtain $2p + 1 \geq 4$, $26 \geq 9$, $-x - 1 \geq 6$ and $x + 20 + 3p \geq 6$. All of these are true in S . (Note that in practice we would use the least common multiple (lcm) of n_l and n_u instead of their product.) Variable y can therefore be eliminated and we obtain

$$S = \{x \in \mathbb{Z} \mid x \leq -11 \wedge p \geq 3 \wedge x + 3p + 10 \geq 0\}.$$

Independent Splits If neither of the two heuristics above apply, we can split the set into two or more parts by cutting the polyhedron in the combined space along a hyperplane. We only consider hyperplanes that are independent of the existential variables. This ensures that the enumerator of the original set is the sum of the enumerators of the parts. A cut that depends on existential variables, on the other hand, would result in sets that may intersect, requiring the computation of a disjoint union.

In particular, we consider all pairs of a lower and an upper bound on an existential variable that do not depend on other existential variables, i.e., they are of the form (6). If neither condition (9) nor condition (12) is satisfied over the whole set, then we cut off that part of the set where condition (9) does hold. In the remaining part, condition (12) holds for this particular pair of constraints. Since the number of pairs of constraints is polynomial in the input size, the number of sets we split off is also polynomial and so the whole technique, if it applies, is polynomial in the input size (for fixed dimension). As a special case, this technique always applies if there is only a single existential variable.

Example 5. Consider once more the set S' (11) from Example 3. The bottom of Figure 4 shows the projection of the corresponding polyhedron in the combined data-parameter space onto the xp -plane and the top shows the xy -slice at $p = 4$. The two constraints we considered in Example 2 also appear in this set. Condition (10) does not hold for the whole set, but instead is used to cut off the part that we considered in Example 2. This is the leftmost part (■) in Figure 4. Using the other constraints, we further split off $p \leq 2$ and $x \geq -10$. The remaining part is the set discussed in Example 4.

3.2 Parametric Integer Programming

Parametric integer programming (PIP) [20] is a technique for computing the lexicographical minimum of a parametric polytope as a function of the parameters. The solution is defined by rational linear expressions in both the original parameters and possibly some extra parameters, defined as the lower integer parts of rational linear expressions of other parameters. Different solutions may exist in different parts of the parameter space, each defined by linear inequalities in the parameters (both original and extra).

To see how parametric integer programming helps in the enumeration of integer projections of parametric polytopes, consider a set S (2) with d regular variables, d' existential variables and n parameters. Compute the lexicographical

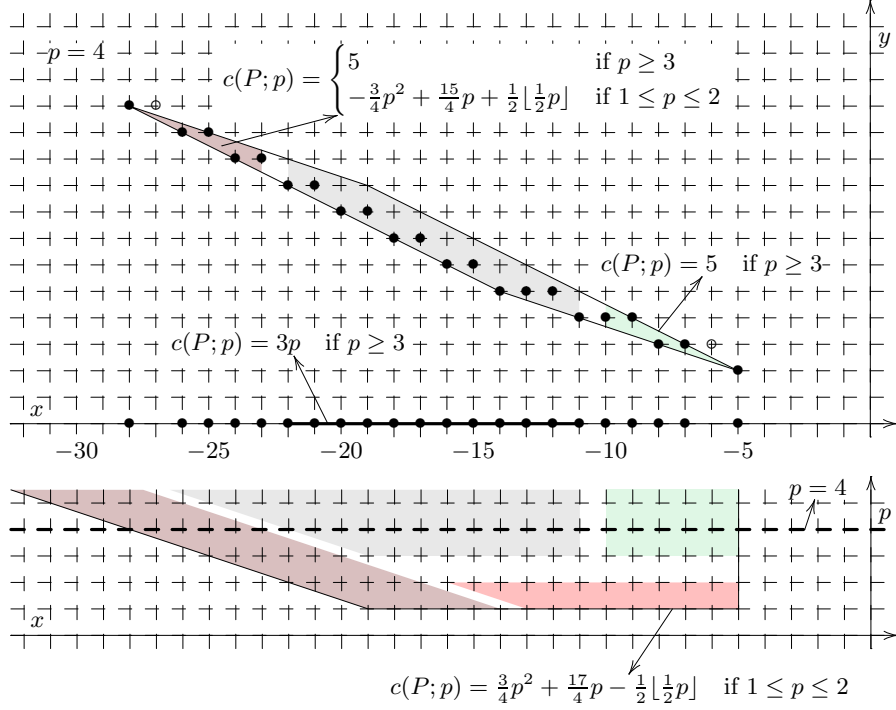


Fig. 4. Decomposition of the set from Example 5.

minimum of the d' existential variables where both the regular variables and the original parameters are considered as parameters, i.e.,

$$\mathbf{y}_{(\mathbf{x}, \mathbf{p})}^{\text{m}} = \text{lexmin} \left\{ \mathbf{y} \in \mathbb{Z}^{d'} \mid \mathbf{A}\mathbf{x} + \mathbf{D}\mathbf{y} + \mathbf{B}\mathbf{p} + \mathbf{c} \geq \mathbf{0} \right\}.$$

Replacing \mathbf{y} by $\mathbf{y}_{(\mathbf{x}, \mathbf{p})}^{\text{m}}$ in the definition of S does not change the number of solutions. However, $\mathbf{y}_{(\mathbf{x}, \mathbf{p})}^{\text{m}}$ is unique (it satisfies Equation (9)) and the quantifier can be dropped. The extra parameters that may appear in the solution can be handled by considering them as extra (unique) existential variables in the set S .

The advantage of using PIP is that it always applies and that it introduces d' equalities, reducing the total dimension. The disadvantage of PIP is that it is worst-case exponential, even for fixed dimension, and that it adds extra existential variables, increasing the total dimension. The total dimension is important since the enumeration technique for parametric polytopes is only polynomial for fixed dimension. Whether the final dimension is greater or smaller than the dimension of the original problem depends on whether the number of extra variables is greater or smaller than the number of existential variables in the original problem.

Example 6. Consider once more the set S' (11) from Example 3. The solution of

$$y_{(x,p)}^{\text{m}} = \text{lexmin} \{ y \in \mathbb{Z} \mid -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8 \}$$

is

$$y_{(x,p)}^{\text{m}} = \begin{cases} 1 - x - \lfloor \frac{2-x}{3} \rfloor & \text{if } x + 3p + 2 \geq 0 \\ -x - \lfloor \frac{p-x}{2} \rfloor & \text{otherwise} \end{cases}.$$

The set S' can thus be written as the (disjoint) union of two sets $S_1 \sqcup S_2$. E.g., S_1 is defined as

$$S_1 = \{ x \in \mathbb{Z} \mid \exists y, q \in \mathbb{Z}^2 : y = 1 - x - q \wedge 2 - 2x \leq 3q \leq 4 - 2x \wedge$$

$$x + 3p + 2 \geq 0 \wedge -x - p \leq 2y \leq -x - 1 \wedge -x + 1 \leq 3y \leq -x + 8 \},$$

where q is the new “parameter” $q = \lfloor (2 - 2x)/3 \rfloor$. Note that both S_1 and S_2 have exactly one additional (unique) existential variable, which means that the total dimension remains constant in this example.

4 Applications

4.1 Reuse Distances

Our main application is the calculation of reuse distances [6]. First, *reuse pairs* are computed, which are pairs of memory accesses to the same cache line, without intervening accesses to that cache line. All the reuse pairs for which the first access is generated by reference r and the second access is generated by reference s , is described by the following Presburger formula (I_r is the iteration at which the first access occurs, J_s is the iteration of the second access):

$$\forall r, s \in \mathcal{R} : \text{reuse}(r \rightarrow s) = \{ (I_r, J_s) \in \mathbb{Z}^n : \text{subject to conditions (13a)–(13d)} \}$$

$$I_r \in \text{IS}(r) \wedge J_s \in \text{IS}(s) \quad (\textit{iteration space}) \quad (13a)$$

$$I_r \prec J_s \quad (\textit{execution ordering}) \quad (13b)$$

$$r@I_r = s@J_s \quad (\textit{same location}) \quad (13c)$$

$$\forall t \in \mathcal{R} : \neg(\exists K_t \in \text{IS}(t) : I_r \prec K_t \prec J_s \wedge t@K_t = r@I_r) \quad (\textit{no intervening access}) \quad (13d)$$

Given a reuse pair, its *accessed data set* is the set of all cache lines accessed in between. The corresponding *reuse distance* is the number of lines in the accessed data set and is an indication of the likelihood for the data to remain in the cache between the two accesses. The definition of the accessed data set can also be given as a Presburger formula [6, Ch.4] which is simplified using the Omega tool [28] to a (disjoint) union of sets of the form (2). Such an expression can be obtained for each pair of references in the program. The iterators of the surrounding loops appear as parameters in these expressions, and the equations (13c), (13d). Furthermore, the Presburger formula representing the accessed data set give rise to existentially quantified variables. The reuse distance is obtained by counting the number of integer solutions to these expressions.

4.2 Other Applications

A number of other applications of enumerating integer projections of polytopes are summarized below.

Calculating Cache Behavior In [12], Chatterjee et al. propose to model the cache behavior of nested loops using Presburger formulas. As an example, in their paper a large formula is presented [12, Fig. 4] which represents the conflict misses in a matrix-vector multiply loop, where the start address of the matrix is a parametric variable. In Section 5.3, we demonstrate that our method computes the number of integer solutions of that formula, and therefore computes the number of conflict misses in that loop in function of the start address of the matrix.

In [16], D’Alberto et al. present a method to compute the number of cache misses in function of the cache line size. Their goal is to choose a cache line size for

a given program, so that its energy dissipation due to cache misses is minimized. In their compiler analysis, they need to count the number of integer solutions in the projection of a polytope. They use heuristics to estimate this number. Our method would allow to calculate the number of integer points exactly.

Memory Size Calculation and Related Optimizations In the area of embedded systems, quite a few authors have proposed to estimate/calculate the minimum memory size needed to perform a computation [1, 2, 29, 42] for a given loop order. The basic idea is that an array element that has been written to in a statement only needs to be stored in memory until the last time it is read. By performing loop transformations, the order of computation changes and similarly the time between writing a value to an array element and its last use changes. For each given loop schedule, the memory size needed is the maximum of the number of live array elements throughout the program run. The live array elements are described by linear inequalities with existential variables [1, 2, 42]. To enumerate them, [1, 42] use heuristics, and none of the proposed methods handle parametric programs, e.g., with symbolic loop bounds. Using our algorithm, these methods can be extended to handle symbolic loops bounds. An example taken from Balasa et al. [2] is discussed in section 5.3.

Automatic Construction of Parallel Hardware from Software Loops

In [5, 22, 26], automatic methods are proposed which map computations in loop nests to parallel processing elements in FPGAs and ASICs. The iterations of the loop nest are projected onto a lower-dimensional processor array. Changing the direction of projection affects both the number of processors and the latency. They search for optimal projection directions that produce the minimal number of processors for a given latency. The number of processors equals the number of integer points in the projection of the iteration space onto the processor space. The kinds of projections produced by this compiler phase is somewhat similar to the projection in Section 1.1. Parametric loop nests, i.e., with symbolic loop bounds, were left for future work [26, Sect. 5], and requires enumerating integer projections of parametric polytopes.

Analytical Computation of Communication Volume in Parallel Systems

In [9], [27] and [39], the communication requirements for given schedules of parallel HPF (High Performance Fortran) programs are computed using polyhedral methods. Computing the iterations that are executed by a particular processor in cyclic and block distributions in HPF leads to constraints with modulo and floor expressions. These are transformed into linear constraints by introducing extra existential variables. Not only the computations performed by each processor, but also the array elements that need to be communicated between parallel processors are represented by parametric polytopes with existential variables. In [27], the resulting quasi-polynomials are used to improve the data distribution and result in a speed-up of 3.5 for a parallel multi-grid application.

5 Experiments

5.1 Test Environment

All computations were performed in exact long integer arithmetic using GMP [24]. We used PolyLib version 5.20.0 with an additional fix for a bug in the implementation of the chamber decomposition which we uncovered during our

experiments. Our fix will be available in the next release and is required to reproduce some of the results we discuss. Many routines in `PolyLib` require a parameter that defines the maximum size of a table used in the double description computation. In earlier versions, this parameter had to be conservatively set to a high number to ensure successful operation, resulting in significant memory overhead. We changed `PolyLib` to dynamically grow this table and all our experiments, both using our own enumeration implementation as well as the implementation of Clauss’s method included in `PolyLib`, are performed with this parameter set to zero. As to our own library, available from <http://freshmeat.net/projects/barvinok/>, we used version 0.13 configured with the `--use-fractional` option. This version contains an optimization which dramatically reduces the memory requirements, eliminating some cases where previous versions would run out memory, at the expense of slightly longer running times for some individual cases. This optimization is also used in `LatTE` [17] and was communicated to us by Ruriko Yoshida.

5.2 Reuse Distances

We performed extensive experiments calculating reuse distances of a set of relatively small but representative test programs including matrix-matrix multiplication and Cholesky factorization. The second column of Table 1 shows the number of times a particular rule from Section 3.1 was used. The remaining columns are explained in Section 5.3. The row “Fixed” refers to the special case of a unique existential variable determined by an equality; “Change” refers to a change of basis. In most of the tests we assume a cache line size of four words. Most of the tests also have array dimensions that are a multiple of the cache line size. For these sets, the resulting enumerator was experimentally verified through a cache simulation. Note that for the sets we considered, we never had to use PIP. We did need to split the sets, which means that simply ignoring the existential quantifiers would have produced the wrong result. Interestingly, some sets contained redundant existential variables, even though they were created by Omega which should have removed them.

type	RD	Chatterjee	Balasa	Boulet
Sets	19177	8+13	4	1
Fixed	3470	0+2	14	5
Change+Fixed	0	0+0	0	2
Unique	4890	8+9	0	0
Change+Unique	18	0+0	0	0
Redundant	684	0+0	2	1
Split	286	0+0	0	0
PIP	0	0+0	0	0

Table 1. Rule application distribution for polytopes originating from reuse distance equations (RD), cache miss analysis (Chatterjee), memory size estimation (Balasa) and communication volume computation (Boulet).

To investigate the impact of the input size, we calculated the reuse distances for matrix-matrix multiplication for varying sizes of the matrices, ranging from 20×20 to 640×640 . There was no measurable increase in computation time for both variants of our method. We also considered matrix sizes that are not multiples of the cache line size. Omega fails to simplify the resulting Presburger formulas, however, and produces formulas containing `Unknowns`, making them

inexact. We were therefore forced to devise a less straightforward way of computing reusing distances, avoiding Omega as much as possible. This modification increases the number of sets to enumerate, e.g., for 20×20 the number of sets increases to 1176. Some of the resulting sets for matrices of size 19×19 and 41×41 proved too difficult to handle. For both sizes, we found at least one set where we had to abort `PolyLib` after one hour while it was calculating quasi-polynomials. PIP did not produce a result for these sets either, as well as for two other sets that we were able to handle using our reduction rules.

Our next experiment was a comparison of the relative performance of PIP and our rules when combined with our polytope enumeration technique. A priori, we would expect that the method with PIP would perform worse since PIP itself is worst-case exponential and the use of PIP may significantly increase the dimension of the problem. Table 2 shows that this increase did not occur for our set of examples. Ignoring the 4 sets that failed to produce an answer (column “?”) as well as the 11355 sets without existential variables (not shown in the table), of the 7952 resulting polytopes, almost 90% have the same dimension as the original set. Furthermore, except for 8 polytopes which experience an increase in dimension, all others have a dimension that is smaller than that of the original set. There are even 35 polytopes with a decrease in dimension that is *larger* than the number of existential variables. The explanation for this phenomenon is that some of the sets allow a range of rational values in one of the dimension, but only a single integer value, e.g., $4 \leq 5i \leq 7$. Again, this is surprising since Omega should have discovered the corresponding equality. For the sets that PIP was able to handle, Figure 5 shows the relative execution time on the left, for sets with an execution time larger than 0.1s, and the relative size of the resulting enumerator on the right, for sets where this relative size is not exactly one. We conclude that for our set of examples, neither method has a clear performance gain over the other.

#EV	Dimension Decrease						
	?	-1	0	1	2	3	4
1			6186	527	25		
2	6		779	102	41	10	
3	2	2	122	66	11	6	
4			6	38	5		7
5			3	1	5		1
6	2						3

Table 2. Dimension decrease induced by PIP in terms of the number of existential variables (*#EV*).

We previously reported [41] that our method for enumerating parametric polytopes is faster, sometimes significantly, than Clauss’s method. Figure 6 provides further evidence of this improvement. As input we use the parametric polytopes generated by PIP on the reuse distance sets. From a total of 18951 polytopes, 907 had a computation time of more than 0.1s. The implementation of Clauss’s method failed to produce a complete result for 190 of these polytopes, due to “degenerate domains”. The ratio of the execution times for the remaining polytopes is shown for the “raw” polytopes on the left and for the polytope with equalities removed on the right. The horizontal axis identifies the individual polytopes. For 17 polytopes on the left and 8 polytopes on the right, the computation with Clauss’s method exceeds 10 minutes. The “ratio” for these polytopes is fixed to 100000 on the figures.

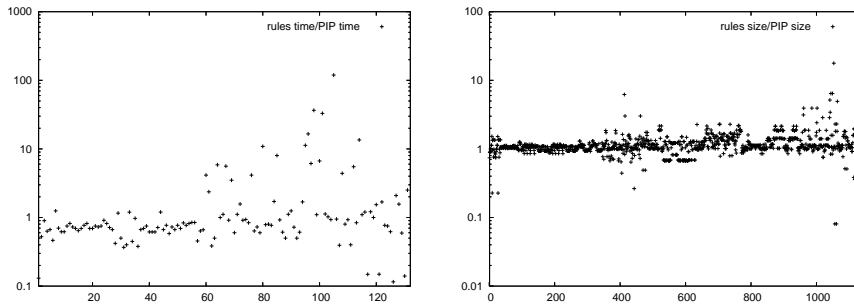


Fig. 5. Comparison between PIP and our rules

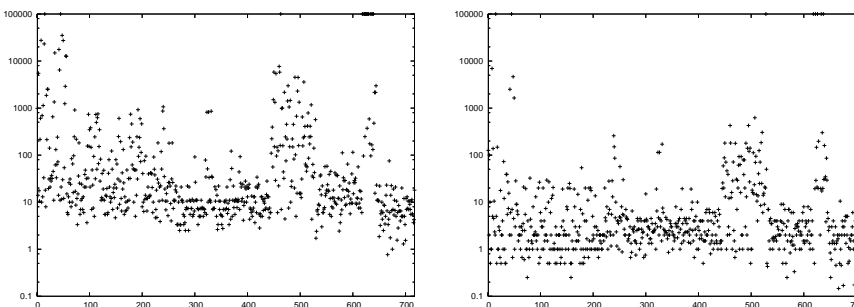


Fig. 6. Execution time ratio for Clauss's method compared to ours for the original polytopes on the left and preprocessed polytopes on the right.

5.3 Other Applications

In this section, we mainly compare the combination of PIP and Clauss's method to the combination of PIP and our method [41]. Boulet and Redon [10] report that manual intervention is needed to efficiently calculate the enumerator of the communication volume of a particular program on an 8×8 processor array. Indeed, directly applying Clauss's method on the output from PIP, as they apparently propose, leads to a computation time of 713s. If we consider the same problem on a 64×64 array, the computation time even increases to 6855s. The output from PIP contains a few equalities, however, and the implementation of Clauss's method apparently does not exploit those. Removing these equalities in a preprocessing step, we obtain the more reasonable times of 0.04s (8×8) and 1.43s (64×64). Using our own method, which removes equalities automatically, instead of Clauss's we obtain a result in 0.01s for both sizes. Our heuristics also work for this example, as shown in Table 1.

Balasa et al. [2] propose a method for computing the number of array elements accessed by a reference in a loop nest execution. As an example, they count the number of array elements accessed by 4 references in a motion estimation loop kernel, for a number of different values of the symbolic loop bounds. We considered the same loop kernel, but handled the symbolic loop bounds parametrically, thereby obtaining a single solution for all possible values of the symbolic loop bounds. The execution times for the four references using Clauss's method (after removing equalities) were respectively 1.38s, 0.01s, 1.41s and 1.41s. Using our method, we obtained the symbolic results in respectively 0.06s, 0.01s, 0.07s and 0.04s.

Finally, we considered Chatterjee's large formula [12]. Table 3 shows the computation times for the 8 disjuncts in this formula each considered as a separate set. The computation time for these sets is larger than for trivial examples, but

still reasonable. Using Clauss’s method to enumerate one of the resulting parametric polytopes did not produce a result even after 15 hours. To count the number of solutions to the whole formula, we first compute the disjoint union using Omega, which only takes a fraction of a second. The resulting 13 sets exhibit computation times comparable to those in Table 3. The rules that are applied are shown in Table 1, with the original disjuncts on the left and the disjoint sets on the right.

PIP	3.16s	53.34s	82.95s	74.43s	4.20s	56.07s	87.62s	87.62s
Heuristics	3.67s	53.52s	80.72s	68.02s	4.14s	56.23s	88.03s	80.14s

Table 3. Computation time for Chatterjee’s sets

6 Conclusions

Many compiler analyses and optimizations require the enumeration of the integer projection of a parametric polytope. We have shown that this can be achieved by reducing the problem to a problem of enumerating parametric polytopes, either by using PIP or by applying a number of rewriting rules. Our experiments show that this reduction, together with our polynomial method for enumerating parametric polytopes [41], yields a method that works well in practice and can solve many problems that were previously considered very difficult or even unsolvable. There is in general no clear performance difference between directly using PIP or applying our rewrite rules for the reduction step. However, since there are some examples where PIP did not produce a result in a reasonable amount of time and since the applicability of the rules can easily be checked, it seems appropriate to try this set of rules first and only use PIP when no complete reduction is achieved.

References

- [1] S. Anantharaman and S. Pande. Compiler optimizations for real time execution of loops on limited memory embedded systems. In *The 19th IEEE Systems Symposium (RTSS98)*, 1998.
- [2] F. Balasa, F. Catthoor, and H. De Man. Background memory area estimation for multidimensional signal processing systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(2):157–172, June 1995.
- [3] A. Barvinok and K. Woods. Short rational generating functions for lattice point problems. *J. Amer. Math. Soc.*, 16:957–979, Apr. 2003.
- [4] A. I. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. In *34th Annual Symposium on Foundations of Computer Science*, pages 566–572. IEEE, Nov. 1993.
- [5] M. Bednara, F. Hannig, and J. Teich. Generation of distributed loop control. In *In Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation (SAMOS)*, volume 2268 of *Lecture Notes in Computer Science*, pages 154–170, 2002.
- [6] K. Beyls. *Software Methods to Improve Data Locality and Cache Behavior*. PhD thesis, Ghent University, 2004.
- [7] A. J. C. Bik. *Compiler Support for Sparse Matrix Computations*. PhD thesis, University of Leiden, The Netherlands, 1996.
- [8] B. Boigelot and L. Latour. Counting the solutions of Presburger equations without enumerating them. *Theoretical Computer Science*, 313(1):17–29, Feb. 2004.

- [9] P. Boulet and X. Redon. Communication pre-evaluation in HPF. In *EU-ROPAR'98*, volume 1470 of *LNCS*, pages 263–272. Springer Verlag, 1998.
- [10] P. Boulet and X. Redon. Communication pre-evaluation in HPF. Technical report, Université des sciences et technologies de Lille, 1998. AS-182.
- [11] V. Braberman, D. Garbervetsky, and S. Yovine. On synthesizing parametric specifications of dynamic memory utilization. Technical report, Oct. 2003.
- [12] S. Chatterjee, E. Parker, P. J. Hanlon, and A. R. Lebeck. Exact analysis of the cache behavior of nested loops. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, pages 286–297. ACM Press, 2001.
- [13] P. Clauss. Counting solutions to linear and nonlinear constraints through Ehrhart polynomials: Applications to analyze and transform scientific programs. In *International Conference on Supercomputing*, pages 278–285, 1996.
- [14] P. Clauss. Handling memory cache policy with integer points counting. In *European Conference on Parallel Processing*, pages 285–293, 1997.
- [15] P. Clauss and V. Loechner. Parametric analysis of polyhedral iteration spaces. *Journal of VLSI Signal Processing*, 19(2):179–194, July 1998.
- [16] P. D’Alberto, A. Veidembbaum, A. Nicolau, and R. Gupta. Static analysis of parameterized loop nests for energy efficient use of data caches. In *Workshop on Compilers and Operating Systems for Low Power (COLP01)*, Sept. 2001.
- [17] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes, Mar. 2003. <http://www.math.ucdavis.edu/~latte/theory.html>.
- [18] S. Derrien, A. Turjan, C. Zissulescu, B. Kienhuis, and E. Deprettere. Deriving efficient control in Kahn process network. In *Proc. of the "Int. Workshop on Systems, Architectures, Modeling, and Simulation, (SAMOS 2003)"*, 2003.
- [19] E. Ehrhart. Polynômes arithmétiques et méthode des polyèdres en combinatoire. *International Series of Numerical Mathematics*, 35, 1977.
- [20] P. Feautrier. Parametric integer programming. *Operationnelle/Operations Research*, 22(3):243–268, 1988.
- [21] J. Ferrante, V. Sarkar, and W. Thrash. On estimating and enhancing cache effectiveness. In U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Proceedings of the Fourth International Workshop on Languages and Compilers for Parallel Computing*, volume 589 of *Lecture Notes in Computer Science*, pages 328–343. Springer-Verlag, Aug. 1991.
- [22] D. Fimmel and R. Merker. Design of processor arrays for real-time applications. In *Proc. Int. Conf. Euro-Par '98*, Lecture Notes in Computer Science, pages 1018–1028, 1998.
- [23] B. Franke and M. O’Boyle. Array recovery and high-level transformations for DSP applications. *ACM Transactions on Embedded Computing Systems*, 2(2):132–162, May 2003.
- [24] Free Software Foundation, Inc. GMP. Available from <ftp://ftp.gnu.org/gnu/gmp>.
- [25] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: a compiler framework for analyzing and tuning memory behavior. *ACM Transactions on Programming Languages and Systems*, 21(4):703–746, 1999.
- [26] F. Hannig and J. Teich. Design space exploration for massively parallel processor arrays. In *Proceedings of the Sixth International Conference on Parallel Computing Technologies*, volume 2127 of *Lecture Notes in Computer Science*, pages 51–65, 2001.
- [27] F. Heine and A. Slowik. Volume driven data distribution for NUMA-machines. In *Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 415–424, 2000.
- [28] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. The Omega library. Technical report, University of Maryland, Nov. 1996.
- [29] P. G. Kjeldsberg, F. Catthoor, and E. J. Aas. Data dependency size estimation for use in memory optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(7), July 2003.
- [30] B. Lisper. Fully automatic, parametric worst-case execution time analysis. In J. Gustafsson, editor, *Proc. Third International Workshop on Worst-Case Execution Time (WCET) Analysis*, pages 77–80, Porto, July 2003.

- [31] V. Loechner, B. Meister, and P. Clauss. Precise data locality optimization of nested loops. *J. Supercomput.*, 21(1):37–76, 2002.
- [32] B. Meister. Projecting periodic polyhedra for loop nest analysis. In M. Gerndt and E. Kereku, editors, *Proceedings of the 11th Workshop on Compilers for Parallel Computers (CPC 04)*, pages 13–24, July 2004.
- [33] E. Parker and S. Chatterjee. An automata-theoretic algorithm for counting solutions to Presburger formulas. In *Compiler Construction 2004*, volume 2985 of *Lecture Notes in Computer Science*, pages 104–119, Apr. 2004.
- [34] W. Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. In *Proceedings of the 1991 ACM/IEEE conference on Supercomputing*, pages 4–13. ACM Press, 1991.
- [35] W. Pugh. Counting solutions to Presburger formulas: How and why. In *SIGPLAN Conference on Programming Language Design and Implementation (PLDI'94)*, pages 121–134, 1994.
- [36] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986.
- [37] R. Seghir. Une nouvelle approche pour le calcul des polynômes d’Ehrhart d’un polyèdre paramétré, 2003.
- [38] B. Sturmfels. On vector partition functions. *J. Comb. Theory Ser. A*, 72(2):302–309, 1995.
- [39] E. Su, A. Lain, S. Ramaswamy, D. J. Palermo, E. W. Hodges IV, and P. Banerjee. Advanced compilation techniques in the PARADIGM compiler for distributed-memory multicomputers. In *International Conference on Supercomputing*, pages 424–433, 1995.
- [40] A. Turjan, B. Kienhuis, and E. Deprettere. A compile time based approach for solving out-of-order communication in Kahn process networks. In *IEEE 13th International Conference on Application-specific Systems, Architectures and Processors (ASAP'2002)*, July 2002.
- [41] S. Verdoolaege, R. Seghir, K. Beyls, V. Loechner, and M. Bruynooghe. Analytical computation of Ehrhart polynomials: Enabling more compiler analyses and optimizations. In *Proceedings of International Conference on Compilers, Architectures, and Synthesis for Embedded Systems, Washington D.C.*, pages 248–258, Sept. 2004.
- [42] Y. Zhao and S. Malik. Exact memory size estimation for array computations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(5):517–521, October 2000.