On Code-Generation in the Polyhedral Model

F. Quilleré & S. Rajopadhye
Irisa, Rennes, France

Abstract

Automatic parallelization in the polyhedral model is based on affine transformations from an original computation domain (iteration space) to a target space-time domain, often with a different transformation for each variable. Code generation is an often ignored step in this process that has a significant impact on the quality of the final code. Previous code generation methods are based on loop splitting, and have non-optimal behavior on parameterized programs.

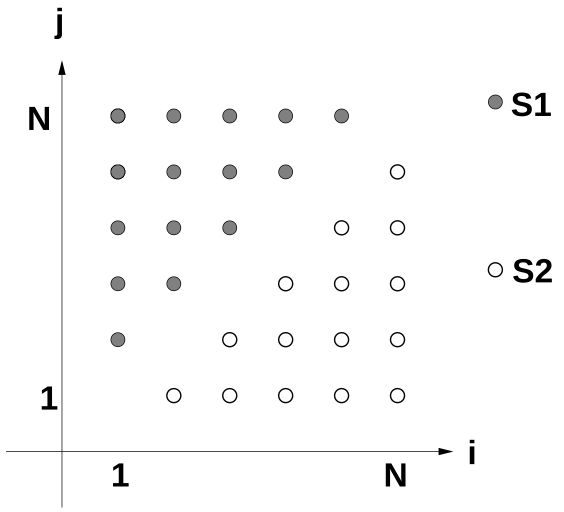
We have previously developed a general parameterized method for code generation based on dual representation of polyhedra. Here, we present a formal proof of correctness of the proposed method. We also discuss how our algorithm enables fine control over the tradeoff between code size and control overhead.

1 Introduction

The polyhedral model is a formalism for reasoning about parallel computations, using *Systems of Affine Recurrence Equations* defined over polyhedral shaped domains. Its use in automatic parallelization of nested loops goes back to the work of Kuck [Kuc78], who showed that the domain of nested loops with affine lower and upper bounds can be described in terms of a polyhedron, and the seminal work of Karp Miller and Winograd [KMW67] on scheduling systems of uniform recurrence equations. The model draws on a number of results from data flow analysis of static loop programs [Fea91], scheduling indexed computations using compact representations [Fea92, KMW67, RF90, QV89], and lifetime analysis and memory reuse [QR00]. Independently of the polyhedral model, many authors have developed linear-algebraic formulations for a large number of loop transformations [Ban93, Iri88, WL91], which consist of reindexing the source index domain using affine transformations and then scanning the new domain, generally in a different order. Though there has been considerable work on finding good transformations to optimize various measures of performance, code generation, the back-end of automatic parallelization, has received relatively less attention.

A key component of code generation is the (single) polyhedron scanning problem, namely producing a set of nested `do` loops which visit the points of a polyhedron in the lexicographic order of indices. This problem was resolved by Ancourt and Irigoin [AI91]. However, scanning multiple (possibly intersecting) domains is much more difficult (see Fig. 1). We cannot simply generate separate pieces of code that scan each polyhedron and "compose" them appropriately.

The problem of scanning unions of polyhedra is posed as follows. Given a set of "domain-qualified statements", i.e., a set of pairs D_i : `Si` where each D_i is a polyhedron and each `Si` is a



(a) Two triangular domains

```
for i = 1 .. N
  for j = i+1 .. N
    S1
```

(b) Scanning D_1 : `S1`

```
for i = 1 .. N
  for j = 1 .. i-1
    S1
  for j = i+1 .. N
    S2
```

Figure 1: Scanning unions of polyhedral domains is not straightforward. Let $D_1 = \{\mathtt{i},\mathtt{j} \mid \mathtt{1} \leq \mathtt{i} < \mathtt{j} \leq \mathtt{N}\}$, and $D_2 = \{\mathtt{i},\mathtt{j} \mid \mathtt{1} \leq \mathtt{j} < \mathtt{i} \leq \mathtt{N}\}$ (see a). Code to scan D_1 : `S1` can be produced easily (the perfect loop in (b) for example) as can code to scan D_2 : `S2`. But code to scan their union: the imperfectly nested loop of (c) is not a simple composition but a "merging" of the two loops.

statement, produce code that visits $\bigcup_i D_i$ in lexicographic order of the indices, and at each point visited, z executes each of the statements `Sk` for which $z \in D_k$.

2 Notation and Background

We briefly recall our background and notations. These are described more precisely in [Wil93] and in [QRW00]. A statement S defined over a polyhedral domain $\mathcal{D}$ is called a domain-qualified statement, and we denote it $\mathcal{D} :: S$.

In the polyhedral model, the symbolic program parameters are represented by dimensions added to the polyhedral domain of each statement. In this representation, the set of constraints that can be expressed on the parameters is the standard one of the polyhedral model, i.e., linear constraints. The constraints on the symbolic parameters can be specified as a polyhedron, whose dimension is the number of parameters.

A polyhedron is defined by $\mathcal{D} = \{\ x \mid Ax \geq b\ \}$ where A and b are a constant matrix and vector respectively. A parameterized polyhedron is a family $\mathcal{D}(p)$ of polyhedra, one per instance of the parameters p, and may be described by replacing vector b with an affine combination of the parameters:

$$\mathcal{D}(p) = \{\ x \mid Ax \geq Bp + b\ \}$$

where A and B are constant matrices and b is a constant vector. This parameterized polyhedron can be rewritten in the form of a non-parameterized polyhedron in the combined data and parameter index space as:

$$\begin{aligned}
\mathcal{D}(p) &= \left\{ x \mid \begin{pmatrix} A & -B \end{pmatrix} \begin{pmatrix} x \\ p \end{pmatrix} \geq b \right\} \\
\mathcal{D}' &= \left\{ \begin{pmatrix} x \\ p \end{pmatrix} \mid A' \begin{pmatrix} x \\ p \end{pmatrix} \geq b \right\}
\end{aligned}$$

A polyhedron may be equivalently specified either with a set of constraints (inequalities) or by its generators (vertices and rays). This description was proposed by Motzkin [MRTT53]. Polylib is based on this double description, and uses the Chernikova algorithm [Che65] to transform one to the other. Using this double descriptiom Le Verge and al [LVVW94] have proposed a method to scan a single polyhedron. They recursively decompose a polyhedron into a sequence of *contexts*.

3 The Multiple Polyhedra Problem

We introduce the problem of generating code for multiple statements, each defined over a different polyhedron (or a finite union of polyhedra). We assume that the definition domains are independent of each other, i.e., they may, or may not, be all disjoint.

In general, data dependencies between statements imply constraints on the scanning order. Thus, we consider the problem of scanning the union of all the domains under the constraint of a given scanning order, formulated as follows:

- a *schedule* assigns a logical execution time to each index vector of each domain. We assume that this time is represented by a k-dimensional time vector[1], and is an affine function Λ of the index vector;
- the scanning order must respect the lexicographic order of the logical execution times. We define the lexicographic order $\prec$ as follows: let x and y be two n-dimensional vectors; we define

$$x \prec y \equiv \exists p,\ 1 \leq p < n,\ (x_{(1..p)} = y_{(1..p)}) \wedge (x_{p+1} < y_{p+1})$$

It follows that domains can not be scanned independently. For example, consider the domains $\mathcal{D}_1$ = `{i| 1<=i<=10}` and $\mathcal{D}_2$ = `{i| 4<=i<=12}`, with respective schedules $\Lambda_1(i) = i + 1$ and $\Lambda_2(i) = 2i$. The statements associated with points 1 and 10 of $\mathcal{D}_1$ have to be executed at logical time 2 and 11, while the statements associated with points 4 and 8 of $\mathcal{D}_2$ have to be executed at times 8 and 16. Since $1 < 8 < 10 < 16$, we cannot first completely scan $\mathcal{D}_1$, and then scan $\mathcal{D}_2$, and neither can we scan $\mathcal{D}_2$ followed by $\mathcal{D}_1$.

Hence, it is not possible to generate separate loops to scan both $\mathcal{D}_1$ and $\mathcal{D}_2$. These loops must be partially merged, so that the statements are executed in the order given by the schedule.

The simplest solution uses a perfectly nested loop to scan a convex superset (such as the bounding box) of this union. However, since the domain scanned by this loop is also a superset of *each* statement's domain, we cannot unconditionally execute the statements within the loop

[1] k is the same for all domains

body. Rather, each statement S_i must be guarded by conditions which test that the current loop index vector belongs to $\mathcal{D}_i$.

The superset of the union scanned by the perfect loop may simply be the bounding box of the domain , as done by the LooPo code generator [GLW98]), or the convex closure of the domain as done by the Omega code generator [KPR95].

Solutions based on perfect loop nests yield compact but inefficient code because of the following limitations.

- Empty iterations: a perfectly nested loop scans a convex polyhedron. Since a domain that is a union of polyhedra may not be convex, some iterations of the loop may not execute any statements. It is often especially inefficient when a domain has an extreme aspect ratio.
- Control overhead: each loop iteration must test the guards of all guarded statements, causing a significant control overhead.
- Finally, finite unions of finite parameterized polyhedra do not always admit finite convex supersets. Consider for example the following parameterized domains : `{i| 1<=i<=N}` and `{i| 1<=i<=M}`; the smallest convex super-set of these domains is the infinite polyhedron `{i| 1<=i}` (note that `{i| 1<=i<=max(M,N)}` is not a convex polyhedron).

To avoid this overhead, we can separate a union of polyhedra into several distinct regions that can each be scanned using imperfectly nested loops. The resulting code is more efficient, since:

- Imperfectly nested loops can scan non-convex regions, avoiding empty iterations;
- It may be possible to choose these loops such that some statement guards become always true or always false; when a guard is always true, the guard may be removed; when a guard is always false, the entire statement can be removed.

However, the efficiency is obtained at the expense of code size for two reasons. First, a perfect loop is replaced by an imperfectly nested loop, which scans a disjoint union of polyhedra. Second, a statement domain may be divided into several disjoint polyhedra, where each polyhedron is scanned by a different loop. In this case, the statement code has to be duplicated in each of these loops.

4 Scanning Disjoint Polyhedra

In this section, we present a method for generating nested loops from polyhedra. We develop a formalism based on disjoint polyhedra. In next section, we describe how we use this formalism to generate code for statements defined over possibly non-disjoint polyhedra.

The following definitions and lemmas essentially state that, given two disjoint parametrerized polyhedra P and Q, there exists a loop nest scanning exactly the integer points of P and Q in the lexicographic order. In the rest of the section, we show that this also applies to sets of disjoint polyhedra.

Definition 1 *We note $\mathcal{C}_l(P)$ the projection of a polyhedron P on its l first dimensions in the lexicographic order. We define that $\mathcal{C}_0(P) = \mathcal{C}_0(\mathcal{Q})$.*

*We define that two disjoint n-dimensional polyhedra P and Q are l-**separable**, and we note PS_lQ, if and only if:*

$$\mathcal{C}_l(P) \cap \mathcal{C}_l(Q) = \emptyset \quad (1)$$
$$\mathcal{C}_{l-1}(P) \cap \mathcal{C}_{l-1}(Q) \neq \emptyset \quad (2)$$

Lemma 1 *Let P and Q be two disjoint n-dimensional polyhedra. There exists a unique l, $1 \leq l \leq n$, such that*

$$PS_lQ$$

Proof **Existence**: by definition, $\mathcal{C}_0(P) = \mathcal{C}_0(Q)$. Thus if $\mathcal{C}_1(P) \cap \mathcal{C}_1(Q) = \emptyset$, then PS_1Q, else, by recurrence, if there exists l such that $\mathcal{C}_l(P) \cap \mathcal{C}_l(Q) = \emptyset$, then PS_lQ, else PS_nQ since $\mathcal{C}_n(P) \cap \mathcal{C}_n(Q) = \emptyset$.

Uniqueness: Let PS_lQ. Then 1) $\forall l' < l, \mathcal{C}_{l'}(P) \cap \mathcal{C}_{l'}(Q) \neq \emptyset$, i.e., $P \not{S}_{l'} Q$, and 2) $\forall l' > l, \mathcal{C}_{l'-1}(P) \cap \mathcal{C}_{l'-1}(Q) = \emptyset$, i.e., $P \not{S}_{l'} Q$. $\square$

Definition 2 *Let P and Q be two disjoint n-dimensional polyhedra. We define that P **precedes** Q, and we note $P \lhd_l Q$, if and only if:*

- $\mathcal{C}_{l-1}(P_1) \cap \mathcal{C}_{l-1}(P_2) \neq \emptyset$, *et*
- $\forall p \in \mathbb{Z}^{l-1}$, $\forall x, y \in \mathbb{Z}^{n-l+1}$,

$$z_1 = \begin{pmatrix} p \\ x \end{pmatrix} \in P_1 \text{ and } z_2 = \begin{pmatrix} p \\ y \end{pmatrix} \in P_2 \Rightarrow z_1 \prec_l z_2$$

Lemma 2 *Let P and Q be two disjoint n-dimensional polyhedra. P and Q are l-separable if and only if $(P \lhd_l Q)$ or $(Q \lhd_l P)$*

Proof $\boxed{\Rightarrow}$ We proove that, if $\mathcal{P}_1$ and $\mathcal{P}_2$ are l-separable, then either $\mathcal{P}_1 \lhd_l \mathcal{P}_2$, or $\mathcal{P}_2 \lhd_l \mathcal{P}_1$.

By definition, $\mathcal{C}_{l-1}(\mathcal{P}_1) \cap \mathcal{C}_{l-1}(\mathcal{P}_2) \neq \emptyset$, thus there exists $p \in \mathbb{Z}^{l-1}$, and $x, y \in \mathbb{Z}^{n-l+1}$, suth that:

$$z_1 = \begin{pmatrix} p \\ x \end{pmatrix} \in \mathcal{P}_1 \text{ and } z_2 = \begin{pmatrix} p \\ y \end{pmatrix} \in \mathcal{P}_2$$

Moreover, either $z_1 \prec_l z_2$, or $z_2 \prec_l z_1$. We proove that in the first case, $\mathcal{P}_1 \lhd_l \mathcal{P}_2$. Let, if possible, $z'_1 \in \mathcal{P}_1$ such that $z_2 \prec z'_1$. Then there exists a et b such tthat

$$a + b = 1 \text{ et } (az_1 + bz'_1)^{(1..l)} = z_2^{(1..l)}$$

This is a contradiction with Definition 1. Thus $\forall p \in \mathbb{Z}^{l-1}$ et $\forall x, y \in \mathbb{Z}^{n-l+1}$ such that $z_1 = \begin{pmatrix} p \\ x \end{pmatrix} \in \mathcal{P}_1$ et $z_2 = \begin{pmatrix} p \\ y \end{pmatrix} \in \mathcal{P}_2$ et $z_1 \prec z_2$, then $\forall z'_1 = \begin{pmatrix} p \\ x' \end{pmatrix} \in \mathcal{P}_1$, $\forall z'_2 = \begin{pmatrix} p \\ y' \end{pmatrix} \in \mathcal{P}_2$, $z'_1 \prec z'_2$.

Now, let if possible, $p' \neq\in \mathbb{Z}^{n-l+1}$ and $z'_1 = \begin{pmatrix} p' \\ x' \end{pmatrix} \in \mathcal{P}_1$ et $z'_2 = \begin{pmatrix} p' \\ y' \end{pmatrix} \in \mathcal{P}_2$, such that $z'_2 \prec_l z'_1$. It follows that there exists a, a', b et b' such that

$$a + a' = 1, \ b + b' = 1, \ (az_1 + a'z'_1)^{(l)} = (bz_2 + b'z'_2)^{(l)}$$

which is a contradiction. This prooves that

$$z_1 \in \mathcal{P}_1,\ z_2 \in \mathcal{P}_2, z_1 \prec z_2 \Rightarrow \forall z_1 \in \mathcal{P}_1, \forall z_2 \in \mathcal{P}_2,\ z_1' \prec z_2'$$

$\boxed{\Leftarrow}$ We proove that, if $\mathcal{P}_1 \lhd_{l-1} \mathcal{P}_2$ ou $\mathcal{P}_2 \lhd_{l-1} \mathcal{P}_1$, then $\mathcal{P}_1$ and $\mathcal{P}_2$ are l-separable. It is sufficient to proove that $\mathcal{C}_l(\mathcal{P}_1) \cap \mathcal{C}_l(\mathcal{P}_2) = \emptyset$. Let us assume that $\mathcal{P}_1 \lhd_{l-1} \mathcal{P}_2$. Definition 2 states that $\forall p \in \mathbb{Z}^{l-1},\ \forall x, y \in \mathbb{Z}^{n-l+1}$,

$$z_1 = \begin{pmatrix} p \\ x \end{pmatrix} \in \mathcal{P}_1 \text{ and } z_2 = \begin{pmatrix} p \\ y \end{pmatrix} \in \mathcal{P}_2 \Rightarrow z_1 \prec_l z_2$$

i.e., that ${z_1}^{(l)} \neq {z_2}^{(l)}$. In the other case, we also proove that

$$\mathcal{P}_2 \lhd_{l-1} \mathcal{P}_1 \Rightarrow \mathcal{C}_l(\mathcal{P}_1) \cap \mathcal{C}_l(\mathcal{P}_2) = \emptyset$$

$\square$

4.1 Scanning Two Disjoint Polyhedra

Let P and Q be two disjoint n-dimensional polyhedra. With any loss of generality, we assume that $P \lhd_l Q$, where $1 \leq l \leq n$. We address the following problem: enumerate the integer points of P and Q in the lexicographic order. $P \lhd_l Q$ implies that PS_lQ, i.e., that $\mathcal{C}_{l-1}(P) \cap \mathcal{C}_{l-1}(Q) \neq \emptyset$. Our solution uses a perfect loop nest to scan the $(l-1)$ first dimensions, and two inner loop nests scanning the other dimensions of each polyhedron:

- We generate $\mathcal{L}_c$, $(l-1)$ nested loops scanning $C_{l-1}(P)$, i.e. the "common context" of P and Q. The body of this loop nest contains the following:
- $\mathcal{L}_P$, a loop nest scanning the last $n-l+1$ dimensions of P, in the context of $\mathcal{L}_c$.
- $\mathcal{L}_Q$, a loop nest scanning the last $n-l+1$ dimensions of Q, in the context of $\mathcal{L}_c$.

The textual placement of $\mathcal{L}_P$ and $\mathcal{L}_Q$ is implied by the relation $P \lhd_l Q$.

4.2 Generalization: Scanning a Set of Disjoint Polyhedra

Let $\mathcal{S}$ be a set of disjoint n-dimensional polyhedra, and $\mathcal{G} = (\mathcal{S}, \Gamma)$ the directed graph defined as follows: the vertices of $\mathcal{G}$ are the polyhedra of S, and there is an edge from P to Q if and only if there exists l such that $P \lhd_l Q$. We note $(P, Q) \in \Gamma_l \Leftrightarrow P \lhd_l Q$, and $\Gamma = \cup_{1 \leq l \leq n} \Gamma_l$.

Let $\mathcal{G}_l' = \cup_{l < i \leq n} \Gamma_i'$ be the non-directed graph such that $(P, Q) \in \Gamma_l'$ if and only if P and Q are l-separable. In other words, $(P, Q) \in \Gamma_l'$ if and only if $(P, Q) \in \Gamma_l$ or $(Q, P) \in \Gamma_l$ (Lemma 2).

We generate a loop nest scanning the union of polyhedra by recursively partitioning the graph. At each step l of the recursion, we generate one level of loop. Each loop scans one dimension of a subset of polyhedra.

To generate the first level of loops, we consider the graph $\mathcal{G}_1'$, which does not include any transition S_1. Let us consider a partition of $\mathcal{G}_1'$, defined its the connected components. Two polyhedra P and Q belong to the same part if and only if there is a path of relations $S_{l>1}$ from P to Q in $\mathcal{G}'$. For each part $\mathcal{E}_i$ of $\mathcal{G}_1'$, we do the following:

- For each polyhedron $P \in \mathcal{E}_i$, we consider $\mathcal{C}_1(P)$, the projection of P on the first dimension.
- Let $L_1(\mathcal{E}_i)$ be the convex union of $\mathcal{C}_1(P)$, for all P in $\mathcal{E}_i$.

- We generate a loop which scans $L_1(\mathcal{E}_i)$.

Note that, for each polyhedron P, the loop which scans the part $\mathcal{E}_i$ containing P scans a superset of the projection of P on the first dimension. In other words, some iterations of this loop will not scan any point of P, i.e., this loop structure is not the most efficient. In next section, we describe how we transform the initial set of polyhedra, in order to generate code optimized for speed.

The textual order of the loops must respect the order of the polyhedra, given by $\lhd_l$. In other words, two polyhedra P and Q scanned (in their first dimension) by two distinct loops must be 1-separable, and the textual order of the loops must respect the order of P and Q. We proove that, to order the loops, it is sufficient to compare the polyhedra $L_1(\mathcal{E}_i)$.

Proposition 1 *Let $P \in L_1(\mathcal{E}_i)$ and $Q \in L_1(\mathcal{E}_j)$. Then*

$$(L_1(\mathcal{E}_i) \lhd_1 L_1(\mathcal{E}_j)) \Leftrightarrow P \lhd_1 Q$$

Proof $\boxed{\Rightarrow}$ Implies that $\mathcal{C}_1(P) \lhd_1 \mathcal{C}_1(Q)$, i.e., that $P \lhd_1 Q$.
$\boxed{\Leftarrow}$ $(P \lhd_1 Q) \Leftarrow \mathcal{C}_1(P) \lhd_1 \mathcal{C}_1(Q)$, i.e., $L_1(\mathcal{E}_i) \lhd_1 L_1(\mathcal{E}_j)$. $\square$

We generate the next levels of loops recursively. Let us consider one part $\mathcal{E}_i$ of $\mathcal{G}'_1$. Generating loops for scanning the second dimension of the polyhedra in this part is the same problem than generating the first level of loops for all the polyhedra. We partition $\mathcal{E}_i$, such that each part of $\mathcal{E}_i$ is a connected component of $\mathcal{G}'_2$. Note that, in a connected component of $\mathcal{G}'_2$, some pairs of polyhedra can be 1-separable. In other words, all the polyhedra are not pairwise 2-separable, thus they are not all comparable by $\lhd_2$. Note also that, when two polyhedra are not comparable, the textual order of the corresponding loops can be arbitrary choosen. In other words, there exists a partial order on the polyhedra in a connected component. Given this partial order, a method to choose a textual order (out of possibly many solutions) just selects a traversal of this graph:

1. Let $i = 0$;
2. Find an unmarked polyhedron which has no predecessor and mark it with i;
3. Increment i and repeat step 2 until all polyhedra are marked;
4. Each polyhedron is now labeled with a position. The generated loops can be textually placed following the ascending order of these positions.

We recursively generate loops until level k, the dimension of the schedule. The remaining dimensions are parallel, i.e. do not carry any data dependencies, thus they can be scanned independently.

5 Code Generation

To generate code for statements defined over different polyhedra, possibly not disjoint, we use a recursive algorithm. At each step l of the recursion, we generate a level of loop by:

1. Projecting the polyhedra onto the outermost l dimensions;

2. Separating these projections into disjoint polyhedra;

3. Recursively generating loop nests that scan each of these;

4. Sorting these loops so that their textual order respects the lexicographic order.

In previous section, we have shown that separating the polyhedra into disjoint one is sufficient to ensure that we can generate code. However, we can always choose to not separate the polyhedra and simply generate loops with guards. If we choose to not separate the polyhedra at each step of the recursion, we finally generate a perfect lop nest that scan a convex union of the union of polyhedra. We implemented a simple heuristic which considers that the depth of a guard indicates its cost. A similar heuristic is also implemented in the Omega code generator [KPR95]. The heuristic kicks in for all loops whose depth is smaller than the chosen one. This corresponds to generating perfectly nested loops which scan the outermost dimensions, and then generating optimized loops for the inner dimensions.

However, the estimate is not realistic. The cost of a guard is the number of times it is potentially evaluated, i.e., the volume of its context polyhedron. Since there are well known techniques for estimating the number of integer points in a parameterized polyhedron using Ehrhart polynomials [CL96], we can use an exact function rather then a heuristic.

6 Related Work

The problem of loop synthesis was first addressed by Ancourt and Irigoin [AI91]. They use Fourier-Motzkin pairwise elimination to produce bounds at each level. The system of constraints is transformed into a triangular system by projecting out, for each loop, the inner loop indices. The main drawback of this method is that redundant bound have to be eliminated afterwards. Fourier-Motzkin based methods are used by a number of parallelizing compilers, including the Pandore II compiler [ALMP95], SUIF [Ama97] and the PIPS project [IJT91]. Collard, Risset and Feautrier [CRF95] show how the dual simplex method [Sch86, Fea88] can be used to generate loop bounds.

Our initial ideas on polyhedra separation and the intuition behind sorting were briefly presented by Quinton et al. [QRW95], and implemented in an early prototype ALPHA compiler. A similar method for loop separation was independently developed by Albiez [Alb98].

Collard [Col94] addresses the generation of loops from multiple polyhedra, with emphasis on the case of loops with incompatible strides. The generated code contains many guards. This paper assumes that there is a single time dimension, but the schedule may not admit a unimodular completion. The author does not therefore face the problem of polyhedron separation and/or sorting, but must confront non-unimodular transformations. Much of the paper is devoted to the (non-unit) strides due to the schedule, and to the removal of redundant guards.

The LOOPO project [GL96] at the University of Passau also explores the parallelization in the polyhedral model and its extension and addresses the problem of code generation. They also address the problem of imperfectly nested loop generation [GLW98]. Given a set of domain-qualified statements, they seek to recursively merge the polyhedra, in much the same way as we do. It seems however, that they do not exploit the *complete* context information, which leads to naive code.

The Omega system [KP93, KPR95], which is also freely available under the Gnu public license, produces code to scan unions of convex sets, in an extension of the polyhedral model. Although the generated code is very similar to ours, it uses a different, non recursive method. It starts from the naive code (a perfect nest of loops which scan the convex closure of the union of domains), and then iteratively tries to remove sources of overhead (guards and min/max operations in loop bounds).

7 Conclusion

We have addressed the problem of scanning the union of (not necessarily disjoint) polyhedra in lexicographic order their indices. The problem arises in the context of the code-generation phase of parallelizing compilers, especially when one permits different mappings to be applied to the individual statements in the body of a static control loop. Our specific context is that of compiling Alpha, a data-parallel functional language based on systems of affine recurrence equations defined over polyhedral domains. The MMAlpha system for manipulating Alpha programs (written using the Mathematica system) is available under the Gnu public license (see `http://www.irisa.fr/cosi/ALPHA`). The loop generation module in the Alpha code generator is written in C++ and is also available under the Gnu public license (at `http://www.irisa.fr/cosi/ALPHA/codegen`).

We use a unified representation for loops and guards, which ensures that bounds and remaining guards are not redundant, and our algorithm is based on a small set of polyhedral operations drawn from Polylib, a library of polyhedral operations [Wil93].

We have proposed a solution to the problem of sorting parameterized loops, and shown that any set of disjoint parameterized loops admit a textual order that respects a given lexicographic schedule. We have defined a partial order on parameterized polyhedra, and used this order to sort loops.

References

[AI91] C. Ancourt et F. Irigoin. – Scanning polyhedra with DO loops. *In : Third Symposium on Principles and Practice of Parallel Programming (PPoPP).* ACM SIGPLAN, pp. 39–50. – ACM Press.

[Alb98] O. Albiez. – Parcours d'une superposition de domaines. *In : RenPar'10.*

[ALMP95] F. Andre, M. Le Fur, Y. Maheo, et J.-L. Pazat. – The Pandore data-parallel compiler and its portable runtime. *Lecture Notes in Computer Science*, vol. 919, 1995, pp. 176–183.

[Ama97] S. P. Amarasinghe. – *Parallelizing Compiler Techniques Based on Linear Inequalities.* – Thèse, Stanford University, janvier 1997.

[Ban93] U. Banerjee. – *Loop Transformations for Restructuring Compilers: The Foundations.* – Norwell, Mass.: Kluwer Academic Publishers, 1993.

[Che65] N. Chernikova. – Algorithm for finding a general formula for the no n-negative solution of a system of linear inequalities. *U.S.S.R Computational Mathematics and Mathematical Physics*, vol. 5, n 2, 1965, pp. 228–233.

[CL96] P. Clauss et V. Loechner. – Parametric analysis of polyhedral iteration spaces. *In : IEEE Int. Conf. on Application Specific Array Processors, ASAP'96.* – IEEE Computer Society.

[Col94] J.-F. Collard. – Code generation in automatic parallelizers. *In : Proc. of the Int. Conf. on Applications in Parallel and Distributed Computing, IFIP W.G 10.3*, éd. par C. Girault. pp. 185–194. – Caracas, Venezuela, avril 1994.

[CRF95] J.-F. Collard, T. Risset, et P. Feautrier. – Construction of DO loops from systems of affine constraints. *Parallel Processing Letters*, vol. 5, n3, septembre 1995, pp. 421–436.

[Fea88] P. Feautrier. – Parametric integer programming. *Rairo Recherche Opérationnelle*, vol. 22, n3, septembre 1988, pp. 243–268.

[Fea91] P. Feautrier. – Dataflow analysis of array and scalar references. *International Journal of Parallel Programming*, vol. 20, n 1, Feb 1991, pp. 23–53.

[Fea92] P. Feautrier. – Some efficient solutions to the affine scheduling problem, part I, one dimensional time, and part II, multidimensional time. *International Journal of Parallel Programming*, vol. 21, n 5-6, octobre 1992, pp. 313–348, 389–420.

[GL96] M. Griebl et C. Lengauer. – The loop parallelizer LooPo. *In : Proc. Sixth Workshop on Compilers for Parallel Computers*, éd. par M. Gerndt, pp. 311–320. – Forschungszentrum Jülich, 1996.

[GLW98] M. Griebl, C. Lengauer, et S. Wetzel. – Code generation in the polytope model. *In : Proc. Int. Conf. on Parallel Architectures and Compilation Techniques (PACT'98)*. pp. 106–111. – IEEE Computer Society Press.

[IJT91] F. Irigoin, P. Jouvelot, et R. Triolet. – Semantical interprocedural parallelization: An overview of the PIPS project. *In : Conference Proceedings, 1991 International Conference on Supercomputing*. ACM SIGARCH, pp. 244–251. – Cologne, Germany, juin 17–21, 1991.

[Iri88] F. Irigoin. – *Code generation for the hyperplane method and for loop interchange*. – Rapport technique nENSMP-CAI-88-E102/CAI/I, Ecole Nationale Superieure des Mines de Paris, Oct 1988.

[KMW67] R. Karp, R. Miller, et S. Winograd. – The organization of computations for uniform recurrence equations. *Journal of the Association for Computing Machinery*, vol. 14, n3, juillet 1967, pp. 563–590.

[KP93] W. Kelly et W. Pugh. – *A Framework for Unifying Reordering Transformations*. – Rapport technique nCS-TR-3193, Dept. of Computer Science, University of Maryland, avril 93.

[KPR95] W. Kelly, W. Pugh, et E. Rosser. – Code generation for multiple mappings. *In : Frontiers '95: The 5th Symposium on the Frontiers of Massively Parallel Computation*. – McLean, VA, février 1995.

[Kuc78] D. J. Kuck. – *The Structure of Computers and Computations*. – NY, J. Wiley and Sons, 1978.

[LVVW94] H. Le Verge, V. Van Dongen, et D. K. Wilde. – *Loop nest synthesis using the polyhedral library*. – PI n830, Irisa, mai 1994.

[MRTT53] T. S. Motzkin, H. Raiffa, G. L. Thompson, et R. M. Thrall. – *Contribution to the theory of games*, chap. The Double Description Method. – Princeton University Press, 1953.

[QR00] F. Quilleré et S. Rajopadhye. – Optimizing memory usage in the polyhedral model. *Transactions on Programming Languages and Systems*, vol. 22, n 5, sep 2000, pp. 773–815.

[QRW95] P. Quinton, S. Rajopadhye, et D. Wilde. – Deriving imperative code from functional programs. *In : 7th Conference on Functional Programming Languages and Computer Architecture*. ACM, pp. 36–44. – La Jolla, CA, Jun 1995.

[QRW00] F. Quilleré, S. Rajopadhye, et D. Wilde. – Generation of efficient nested loops from polyhedra. *ijpp*, vol. 28, n 5, 2000, pp. 469,498.

[QV89] P. Quinton et V. Van Dongen. – The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing*, vol. 1, n 2, 1989, pp. 95–113.

[RF90] S. Rajopadhye et R. Fujimoto. – Synthesizing systolic arrays from recurrence equations. *Parallel computing*, vol. 14, juin 1990, pp. 163–189.

[Sch86] A. Schrijver. – *Theory of Linear and Integer Programming*. – Chichester, New York, Wiley, 1986, *A Wiley-Interscience publication.*, 471p.

[Wil93] D. Wilde. – *A Library for Doing Polyhedral Operations*. – PI n 785, Irisa, dec 1993.

[WL91] M. Wolf et M. Lam. – Loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, n 4, Oct 1991, pp. 452–471.