Scheduling in the $\mathcal{Z}$-Polyhedral Model*

Gautam[1,2], DaeGon Kim[1] and S. Rajopadhye[1]
{gautam|kim}@cs.colostate.edu, Sanjay.Rajopadhye@colostate.edu

[1]Colorado State University
Computer Science Department
Fort Collins, CO 80523, USA

[2]IRISA
Université de Rennes I
Rennes, France

Abstract

The polyhedral model is extensively used for analyses and transformations of regular loop programs, one of the most important being automatic parallelization. The model, however, is limited in expressivity and the need for the generalization to more general class of programs has been widely known. Analyses and transformations in the polyhedral model rely on certain closure properties. Recently, these closure properties were extended to programs where variables may be defined over unions of $\mathcal{Z}$-polyhedra which are the intersection of polyhedra and lattices.

We present the scheduling analysis for the automatic parallelization of programs in the $\mathcal{Z}$-polyhedral model, and obtain multidimensional schedules through an ILP *formulation that minimizes latency. The resultant schedule can then be used to construct a space-time transformation to obtain an equivalent program in the $\mathcal{Z}$-polyhedral model.*

1. Introduction

The polyhedral model is a well developed formalism providing sophisticated analysis and transformations of the kernels of many compute- and data- intensive applications. Programs in the polyhedral model essentially comprise of (i) variables representing collections of values defined over polyhedral domains[1], and (ii) affine dependences between computations. Feautrier [6] showed that an important class of conventional imperative loop programs called *affine control loops* (ACLs) can be transformed to programs in the polyhedral model. Significant parts of the SpecFP and PerfectClub benchmarks are ACLs [2].

Many computations can be expressed in the polyhedral model, *e.g.*, matrix multiplication, LU-decomposition, Cholesky factorization, Kalman filtering, as well as algorithms arising in RNA secondary structure prediction [18]. Nevertheless, the polyhedral model suffers from certain limitations. Loop programs with a non-unit stride fall outside the scope of the model. This is an important class of programs [25, 16, 30, 9] arising in situations such as the red-black SOR computation for solving partial differential equations. As a consequence, *non-unimodular* transformations are also disallowed in the polyhedral model. Non-unimodular transformations are required for the derivation of parallel architectures with *periodic* processor activity, such as *multi-rate arrays* [15] and bidirectional systolic arrays.

It had long been conjectured that these limitations can be resolved through the extension of variable domains to unions of $\mathcal{Z}$-polyhedra which are the intersection of polyhedra and affine lattices. However, a key representation and interpretation for $\mathcal{Z}$-polyhedra and the associated family of dependences was only recently provided [10] along with the proofs of the required closure properties for $\mathcal{Z}$-polyhedral domains in this representation.

The $\mathcal{Z}$-polyhedral model allows specifications with a more general dependence pattern than the specifications in the polyhedral model.

Example 1 *Consider the following loop program*

```
for i = 1 to N
  A[i] = (i%2==0?A[i/2]:0);
```

This program exhibits a dependence pattern that is richer than the affine dependences of the polyhedral model. In other words, it is impossible to write an equivalent program in the polyhedral model, i.e., without the use of the mod operator or non-unit stride loops, that can perform the required computation. One may consider replacing the variable A by two variables X and Y corresponding to the even and odd points of A such that $A[2i] = X[i]$ and $A[2i-1] = Y[i]$. However, the definition of X now requires the mod operator, because $X[2i] = X[i]$ and $X[2i-1] = Y[i]$.

*This research was supported in part, by the National Science Foundation, under the grant EI-030614: HiPHiPECS: High Level Programing of High Performance Embedded Computing Systems

[1]These may be viewed as generalized multi-dimensional arrays, the bounds of which are given by arbitrary affine inequalities.

In addition to expressibility, the $\mathcal{Z}$-polyhedral model also enables more sophisticated analyses and transformations by providing greater information in the specifications *viz.*, pertaining to lattices.

Example 2 *Consider the following loop that elaborates the advantages of manipulating $\mathcal{Z}$-polyhedral domains.*

```
for i = 1 ... n
  if ((i%2==0)||(i%3==0))
    X[i] = X[i-1];
```

Note that the equation is not executed for every iteration of the form $6j+1$ and $6j+5$ where j is an integer. Below is a constant-time parallelization of this $\Theta(n)$ loop.

```
forall i = 2 ... n step 6
  X[i] = X[i-1];

forall i = 6 ... n step 6
  X[i] = X[i-1];

forall i = 3 ... n step 6
  X[i] = X[i-1];

forall i = 4 ... n step 6
  X[i] = X[i-1];
```

This parallelization, however, requires (i) value-based dependence analysis, (ii) scheduling analysis and (iii) code generation of equations on variables defined over $\mathcal{Z}$-polyhedral domains. A polyhedral approximation of the domain of the equation `X[i]=X[i-1]` would not yield a constant-time parallelization.

Automatic parallelization is one of the most important and widely studied analysis in the polyhedral model [12, 13, 7, 8, 4, 24, 23, 3, 26, 27, 17, 20]. This paper presents an algorithm for scheduling the more general programs of the $\mathcal{Z}$-polyhedral model. Our key contributions are (i) deriving precedence (causality) constraints for programs written in the $\mathcal{Z}$-polyhedral model, (ii) formulation of an *integer linear program* to obtain a schedule which is based on Farkas method and minimizes latency, and (iii) the generalization of the scheduling problem to multi-dimensional schedules (i.e., schedules where the "time instants" are multidimensional vectors under lexicographic order, corresponding naturally to nested loops). An important feature of our formulation is that it seeks schedules that can be used to construct a program transformation to obtain an equivalent specification in the $\mathcal{Z}$-polyhedral model (This has been a major drawback of previous methods using rational schedules).

The remainder of this paper is organized as follows. In the following section, we give an example to motivate the scheduling problem for programs written in the $\mathcal{Z}$-polyhedral model. The mathematical background on lattices, polyhedra, $\mathcal{Z}$-polyhedra and generalized affine functions is described in section 3. In section 4, we describe an equational language for high level specifications in the $\mathcal{Z}$-polyhedral model and present reduced dependence graphs as the required abstraction for our analysis. In section 5, we derive precedence constraints and then formulate an ILP to obtain valid (multidimensional) schedules. Finally, we discuss future and related work and present our conclusions.

2. Motivating Example

Consider the following loop program.

```
for i = 0 to N
  for j = 0 to N
    X[i,j] = (i%2==0?X[i,j-1]:Y[i-1,j]);
  for j = N downto 0
    Y[i,j] = (i%2==0?X[i,j]:Y[i,j+1]);
```

In order to parallelize this affine loop program extracting data dependence relations is a first and critical step. Such dependence relations should be respected in any parallelized program. Otherwise, the semantics of a program will not be preserved.

In order to know dependence relations, we now apply value-based dependence analysis [6], a well-known technique for extracting dependence relations. The obtained data dependence information are shown in Figure 1. A node in data dependence graph represents an iteration point of a statement, and the arrow between nodes specifies data-flow between two operations.

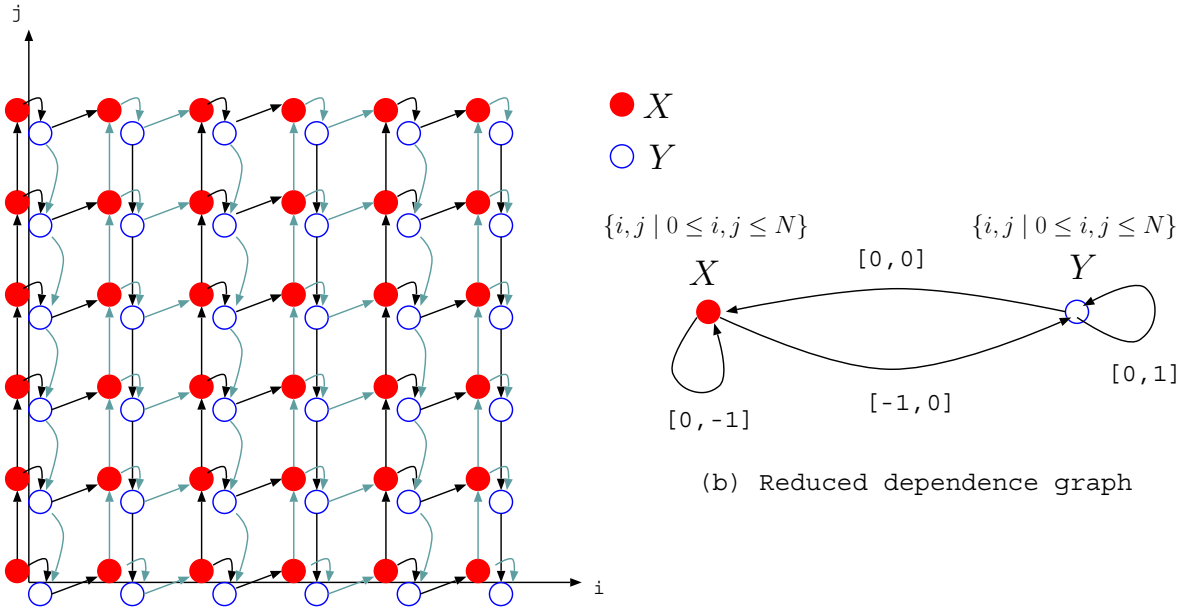
Now, we want to parallelize this computation. Because of the vertical dependence on X, the execution order of X will be increasing order of j index. Similarly, the execution order of Y will be decreasing order of j. Together with dependences between X to Y, this computation must be done sequentially exactly like the original loops, that is, for each i, first computing X in the increasing of j and then computing Y in the decreasing order of j, repeating this for $i+1$ until the whole computation finishes.

One may find one dimensional parallelism in the program by distinguishing true dependence from false dependence as shown in Figure 1. In the figure the true dependence are denoted with black arrows. The precise execution order can be given by the following schedules

$$\lambda_X(i,j) = \begin{cases} i \text{ even} & : \quad j \\ i \text{ odd} & : \quad j+2 \end{cases}$$

$$\lambda_Y(i,j) = \begin{cases} i \text{ even} & : \quad j+1 \\ i \text{ odd} & : \quad N-j \end{cases}$$

Here, $\lambda_X(i,j)$ (respectively, $\lambda_Y(i,j)$) represents the time instant at which the computation $X[i,j]$ (respectively, $Y[i,j]$) is executed. This execution order is realized by the parallelized loop program given below.



(a) Data dependence graph

(b) Reduced dependence graph

Figure 1. Motivating Example: Data dependence graph for $N = 5$ and Reduced Dependence Graph (RDG) in the Polyhedral model

```
for j = 0 to N
  forall i to (N/2) {
    Y[2i+1,N-j]=Y[2i+1,N-j+1];
    X[2i,j]=X[2i,j-1];
    Y[2i,j]=Y[2i,j];
    X[2i+1,j]=Y[2i,j];
  }
```

Note that the loop program is correct only when N is odd. For the sake of simplicity, we assume that N is odd. We also assume that the statements within an iteration of the `forall` loop are executed sequentially, i.e., there is synchronization between statements.

The key idea of this detection is based on precise information on dependence relations by separating a rectangle, called polyhedron, into two disjoint rectangle with holes, called $\mathcal{Z}$-polyhedra. Note that the actual representation of data dependence graph in Figure 1 is the RDG. Since N is not known at compile time, the data dependence graph is not possible to construct. The precise data dependence relations are shown in Figure 2.

In this paper, we address the problem of scheduling together with constructing RDG in $\mathcal{Z}$-polyhedral model. One may argue that the example has an equivalent loop program where parallelism can be detected even in Polyhedral model. As we argue in the introduction, every program in $\mathcal{Z}$-polyhedral model does not have an equivalent program in Polyhedral model. Also, it is not always obvious to write programs so that parallelism can be detected in Polyhedral model.

3. Mathematical Background

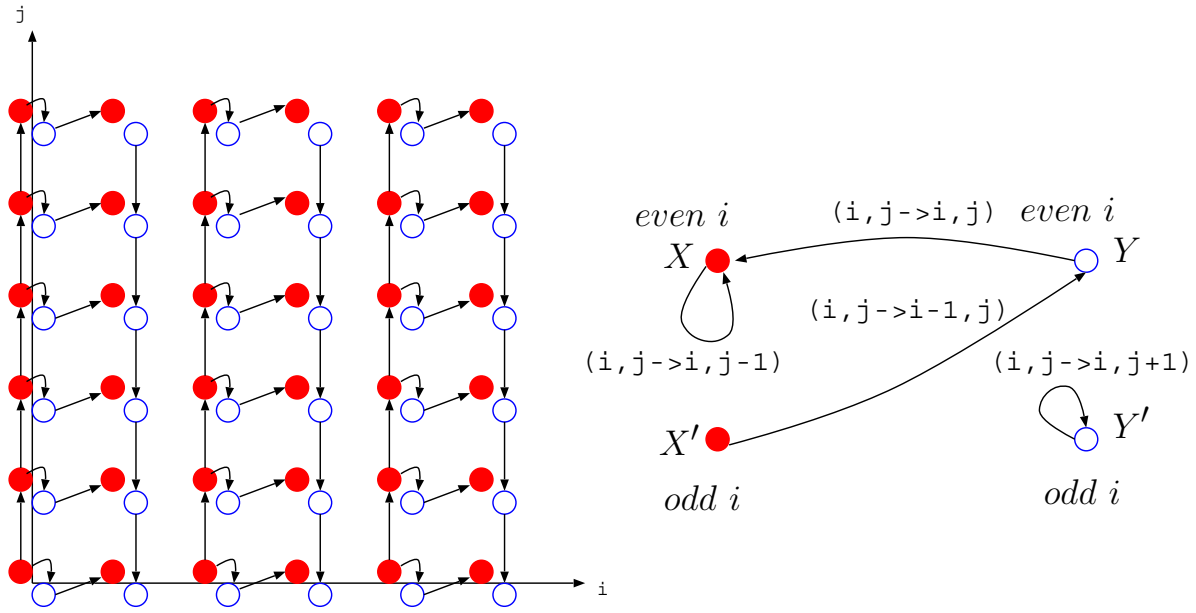
In this section, we will first provide the required mathematical background on linear algebra over integers. Then, we will define the mathematical objects required in our analysis.

3.1. Matrices

As a convention, we will denote matrices with the upper-case letters and vectors with the lower-case. Unless specifically mentioned, all matrices and vectors have integer elements. We will denote the identity matrix by I. Syntactically, the different elements of a vector v will be written as a list.

We will use the following concepts and properties of matrices

- The kernel of a matrix T, written as $\ker(T)$ is the set of all vectors z such that $Tz = 0$.
- A matrix is unimodular if it is square and its determinant is either 1 or -1.
- Two matrices L and L' are said to be *column equivalent* or *right equivalent* if there exists a unimodular matrix U such that $L = L'U$.
- A unique representative element in each set of matrices that are column equivalent is the one in *Hermite normal form* [11].



(a) Data dependence graph

Figure 2. Motivating Example: Data dependence graph for $N = 5$ and RDG in the $\mathcal{Z}$-polyhedral model; In RDG, two nodes above (respectively below) are associated with above (respectively below) $\mathcal{Z}$-polyhedron

Definition 1 *An $n \times m$ matrix H with column rank d is in* Hermite Normal Form*, if*

1. *For columns $2, \ldots, d$, the first positive element is below the first positive element for the previous column.*
   $\forall 1 \leq j \leq d, \exists i_j, 1 \leq i_1 < \ldots < i_d \leq n : H_{i_j,j} > 0$
2. *In the first d columns, all elements above the first positive element are zero.*
   $\forall 1 \leq j \leq d, 1 \leq i < i_j : H_{i,j} = 0$
3. *The first positive entry in columns $1, \ldots, d$ is the maximal entry on its row. All elements are non-negative in this row.*
   $\forall 1 \leq l < j \leq d : 0 \leq H_{i_j,l} < H_{i_j,j}$
4. *columns $d+1, \ldots m$ are zero-columns*
   $\forall d+1 \leq j \leq m, 1 \leq i \leq n : H_{i,j} = 0$

$$\begin{pmatrix} \blacktriangledown & 0 & 0 & 0 \\ \bigstar & 0 & 0 & 0 \\ \blacktriangle & \blacktriangledown & 0 & 0 \\ \blacktriangle & \blacktriangle & \blacktriangledown & 0 \\ \bigstar & \bigstar & \bigstar & 0 \\ \bigstar & \bigstar & \bigstar & 0 \end{pmatrix}$$

On the left is a template of a matrix in Hermite normal form. In the given template, ▼ denotes the maximal (non-negative) element in the row, ▲ denotes non-negative elements that are not maximal in the row and ★ denotes any integer.

For every matrix A, there exists a unique matrix H that is in Hermite normal form and column equivalent to A *i.e.*, there exists a unimodular matrix U such that $A = HU$. Note that the provided definition of the Hermite normal form does not require the matrix A to have full row rank.

3.2. Affine Lattices

The lattice generated by a matrix L is the set of all integer linear combinations of the columns of L. If the columns of a matrix are linearly independent, they constitute a *basis* of the generated lattice. The lattices generated by two matrices are equal *iff* the submatrices corresponding to the non-zero columns in their Hermite normal forms are equal. As a special case, the lattices generated by two $n \times m$ matrices are equal *iff* the matrices are column equivalent.

We use a generalization of the lattices generated by a matrix, additionally allowing offsets by constant vectors. These are called *affine lattices*. An affine lattice is a subset of $\mathbb{Z}^n$ and can be represented as $\{Lz + l | z \in \mathbb{Z}^m\}$ where L and l are an $n \times m$ matrix and n-vector respectively. We call z the coordinates of the affine lattice.

The affine lattices $\{Lz + l | z \in \mathbb{Z}^m\}$ and $\{L'z' + l' | z' \in \mathbb{Z}^{m'}\}$ are equal *iff* the lattices generated by L and L' are equal and $l' = Lz_0 + l$ for some constant vector $z_0 \in \mathbb{Z}^m$.

3.3. Integer Polyhedra

An *integer polyhedron*, $\mathcal{P}$ is a subset of $\mathbb{Z}^n$ that can be defined by a finite number of affine inequalities (also called affine constraints or just constraints when there is no ambiguity) with integer coefficients. We follow the convention that the affine constraint c_i is given as $(a_i^T z + \alpha_i \geq 0)$ where $z, a_i \in \mathbb{Z}^n, \alpha_i \in \mathbb{Z}$. The integer polyhedron, $\mathcal{P}$, satisfying the set of constraints $\mathcal{C} = \{c_1, \ldots, c_b\}$ is often written as

$\{z \in \mathbb{Z}^n | Qz+q \geq 0\}$ where $Q = (a_1 \ldots a_b)^T$ is an $b \times n$ matrix and $q = (\alpha_1 \ldots \alpha_b)^T$ is an b-vector eg. $\{i, j | 0 \leq i, 0 \leq j\}$ is the polyhedron corresponding to the first orthant.

We shall use the following properties and notation.

- The constraint $c \equiv (a^T z + \alpha \geq 0)$ of $\mathcal{P}$ is said to be *saturated iff* $(a^T z + \alpha = 0) \cap \mathcal{P} = \mathcal{P}$.
- The *lineality space* of $\mathcal{P}$ is defined as the linear part of the largest affine subspace contained in $\mathcal{P}$. It is given by $\ker(Q)$.
- The *context* of $\mathcal{P}$ is defined as the linear part of the smallest affine subspace that contains $\mathcal{P}$. If the saturated constraints in $\mathcal{C}$, are the rows of $\{Q_0 z + q_0 \geq 0\}$, then it is $\ker(Q_0)$.

3.4. Parameterized Integer Polyhedra

A parameterized integer polyhedron is an integer polyhedron where some indices are interpreted as size parameters. An equivalence relation can be defined on the set of iteration points in a parameterized integer polyhedron such that two iteration points are equivalent if they have identical values of size parameters. By this relation, a parameterized integer polyhedron may be partitioned into a set of equivalence classes, each of which is identified by the vector of size parameters. Equivalence classes correspond to program instances and are called instances of the parameterized integer polyhedron.

3.5. $\mathcal{Z}$-Polyhedra

A $\mathcal{Z}$-polyhedron is the intersection of an integer polyhedron and an affine lattice. It is also an integer polyhedron when the affine lattice is the canonical lattice, $\mathbb{Z}^n$. The required closure properties on unions of $\mathcal{Z}$-polyhedra were based on the following representation for $\mathcal{Z}$-polyhedra.

$$\{Lz + l | Qz + q \geq 0, z \in \mathbb{Z}^m\} \tag{1}$$

where L has full column rank and the polyhedron $\mathcal{P}^c = \{z | Qz + q \geq 0, z \in \mathbb{Z}^m\}$ has a context that is the universe, $\mathbb{Z}^m$. $\mathcal{P}^c$ is called the coordinate polyhedron of the $\mathcal{Z}$-polyhedron. The $\mathcal{Z}$-polyhedron for which L has no columns has a coordinate polyhedron in $\mathbb{Z}^0$. The empty $\mathcal{Z}$-polyhedron is denoted by $\{|\}$. The interpretation is that the $\mathcal{Z}$-polyhedral representation is said to be *based on* the affine lattice given by $\{Lz+l | z \in \mathbb{Z}^m\}$. Iteration points of the $\mathcal{Z}$-polyhedral domain are points of the affine lattice corresponding to valid coordinates. The set of valid coordinates is given by the coordinate polyhedron.

3.6. Parameterized $\mathcal{Z}$-Polyhedra

A parameterized $\mathcal{Z}$-polyhedron is a $\mathcal{Z}$-polyhedron where some rows of its corresponding affine lattice are interpreted as size parameters. Similar to parameterized integer polyhedra, an equivalence relation may be defined on the set of iteration points in a parameterized $\mathcal{Z}$-polyhedron such that two iteration points are equivalent if they have identical values of size parameters. Thus, a parameterized $\mathcal{Z}$-polyhedron may be partitioned into a set of equivalence classes, each of which is called an instance and identified by the vector of size parameters.

For the sake of explanation, and without loss of generality, we may impose that the rows that denote size parameters are before all non-parameter rows. The equivalent $\mathcal{Z}$-polyhedron based on the Hermite normal form of such a lattice has an important property that will used in our analysis; all points of the coordinate polyhedron with identical values of the first few indices belong to the same instance of the parameterized $\mathcal{Z}$-polyhedron.

Example 3 *Consider the $\mathcal{Z}$-polyhedron given by the intersection of the polyhedron $\{p, i | 0 \leq i \leq p\}$ and the lattice $\{j + k, j - k\}$[2]. It may be written as*

$$\{j+k, j-k | 0 \leq j-k \leq j+k\} = \{j+k, j-k | 0 \leq k \leq j\}$$

Now, suppose the first index, p, in the polyhedron is the size parameter. Similarly, the first row in the lattice $\{j + k, j - k\}$ corresponding to the $\mathcal{Z}$-polyhedron is the size parameter. The Hermite normal form of this lattice is $\{j', j' + 2k'\}$. The equivalent $\mathcal{Z}$-polyhedron is

$$\{j', j' + 2k' | k' \leq 0 \leq j' + 2k'\}$$

The iterations of this $\mathcal{Z}$-polyhedron belong to the same program instance iff *they have the same coordinate index j'. In this example, valid values of the parameter row trivially have a one-to-one correspondence with values of j'; identity being the required bijection.*

It is not always possible to obtain identity as the required bijection. For an example, study the following $\mathcal{Z}$-polyhedron with the first two rows considered as size parameters.

$$\{m, m + 2n, i + m, j + n | 0 \leq i \leq m; 0 \leq j \leq n\}$$

Here too, valid values of the parameter rows have a one-to-one correspondence with the values of m and n but it is impossible to obtain identity as the required bijection.

3.7. Affine Lattice Functions

An (standard) affine function is of the form $(z \rightarrow Tz + t)$ where T is an $n \times m$ matrix and t is an n vector. *Affine lattice functions* are of the form $(Kz + k \rightarrow Rz + r)$, where K has full column rank. Such functions provide a mapping from the iteration $Kz + k$ to the iteration $Rz + r$. We have imposed that K have full column rank to guarantee that $(Kz + k \rightarrow$

[2]For both the polyhedron and the affine lattice, the specification of the space $\mathbb{Z}^2$ is redundant. It can be derived from the number of indices and is therefore dropped for the sake of brevity.

$Rz + r)$ be a function and not a relation, mapping any point in its domain to a unique point in its range. All standard affine functions are also affine lattice functions. For any function f, f^{-1} will denote its relational inverse.

4. Equational Specification

An intuitive and general way of specifying programs in the $\mathcal{Z}$-polyhedral model is through a list of high level (mutually recursive) equations.

Consider the red-black SOR [29] for the iterative computation of partial differential equations. Iterations in the (i, j)-plane are divided into "red" points and "black" points, similar to the layout of squares in a chess board. First, black points (at even $i + j$) are computed using the four neighbouring red points (at odd $i + j$), then the red points are computed using its four neighbouring black points. These two phases are repeated until convergence. Introducing an additional dimension, k to denote the iterative application of the two phases, we get the following equation

$$C_{i,j,k} = \begin{cases} i+j \text{ even} : \frac{1}{4}(C_{i-1,j,k-1} + C_{i+1,j,k-1} \\ \qquad\qquad +C_{i,j-1,k-1} + C_{i,j+1,k-1}) & \text{// black} \\ i+j \text{ odd} : \frac{1}{4}(C_{i-1,j,k} + C_{i+1,j,k} \\ \qquad\qquad +C_{i,j-1,k} + C_{i,j+1,k}) & \text{// red} \end{cases}$$

This equation, with appropriate syntactic sugaring, is our preferred program in the $\mathcal{Z}$-polyhedral model. Here, C and the two branches of the equation are defined over $\mathcal{Z}$-polyhedral domains. A $\mathcal{Z}$-polyhedral domain is a union of $\mathcal{Z}$-polyhedra.

In general, the input for our scheduling analysis is a finite list of equations of the form

$$V = \begin{cases} \ldots & \ldots \\ \mathcal{D}_{V,i} : & \mathtt{op}(\ldots, U.(Lz + l \to Rz + r), \ldots) \\ \ldots & \ldots \end{cases} \quad (2)$$

where V and U are variables declared over the $\mathcal{Z}$-polyhedral domains $\mathcal{D}_V$ and $\mathcal{D}_U$ respectively, $\mathcal{D}_{V,i}$ is the $\mathcal{Z}$-polyhedral domain of the corresponding branch of the equation and $\mathtt{op}$ is an arbitrary, atomic, iteration-wise, single-valued function that takes a single time-step to evaluate. The affine lattice function $(Lz + l \to Rz + r)$ is a dependence such that the value of the (sub)expression, $U.(Lz + l \to Rz + r)$ at $Lz + l$ equals the value of U at $Rz + r$. The domains of different branches of an equation are disjoint and satisfy $\biguplus \mathcal{D}_{V,i} = \mathcal{D}_V$ to ensure that any variable is not under- or over-defined. Variables that are not defined by an equation are treated as input. These equations can be obtained from the more general equational language presented in [10] through a transformation based on the closure properties of $\mathcal{Z}$-polyhedral domains, called normalization.

Parameterized equational specifications in the $\mathcal{Z}$-polyhedral model are based on parameterized $\mathcal{Z}$-polyhedra. An instance of a parameterized specification is called a *program instance*. Every program instance in a parameterized specification is independent, thus, all dependences should map consumer iterations to producer iterations within the same program instance.

Our presentation of the equational specification is based on the ALPHA language [19, 14] and the MMALPHA framework for manipulating ALPHA programs, which relies on a library for manipulating polyhedra [28].

4.1. Basic Reduced Dependency Graph

A directed multi-graph called the *reduced dependence graph*, RDG, precisely describes the dependences between iterations of variables. It is defined as follows

- For every variable in the specification, there is a vertex in the RDG labeled by its name and annotated by its domain. We will refer to vertices and variables interchangeably.
- For every dependence of the variable V on U, there is an edge from V to U. It is annotated by the corresponding dependence function. We will refer to edges and dependences interchangeably.

At a finer granularity, every branch of an equation is associated to dependences between computations. Thus, a precise analysis would dictate that dependences be expressed separately for every branch. Again, for reasons of precision, we will express dependences of a variable separately for every element in the $\mathcal{Z}$-polyhedral domain of the corresponding branch of its equation. To enable these, we will replace a variable by a set of new variables as elaborated below.

In Equation (2), let $\mathcal{D}_{V,i}$ be written as a disjoint union of $\mathcal{Z}$-polyhedra given by $\biguplus_j \mathcal{Z}_j$. The variable V in the domain $\mathcal{Z}_j$ is replaced by a new variable, say X_j. Similarly, let U be replaced by new variables given as Y_k. The dependence of V in $\mathcal{D}_{V,i}$ on U is replaced by dependences from all X_j on all Y_k. An edge from X_j to Y_k may be omitted if there are no iterations in X_j that map to Y_k (mathematically, if the preimage of Y_k by the dependence function does not intersect with X_j).

A naive construction following these rules results in the *basic reduced dependence graph*. Figure 3a gives the basic RDG for example 1 which can be written as the following equation.

$$A = \begin{cases} \{2i | 1 \le 2i \le n\} & A.(2i \to i) \\ \{2i-1 | 1 \le 2i-1 \le n\} & 0.(i \to) \end{cases}$$

Let the two branches of the expression be denoted by X and Y respectively. Next, we will study a refinement on this RDG.

4.2. Refined Reduced Dependence Graph

In the RDG for the generic specification given in Equation (2), let X be a variable derived from V and defined on $\mathcal{Z}_X \in$

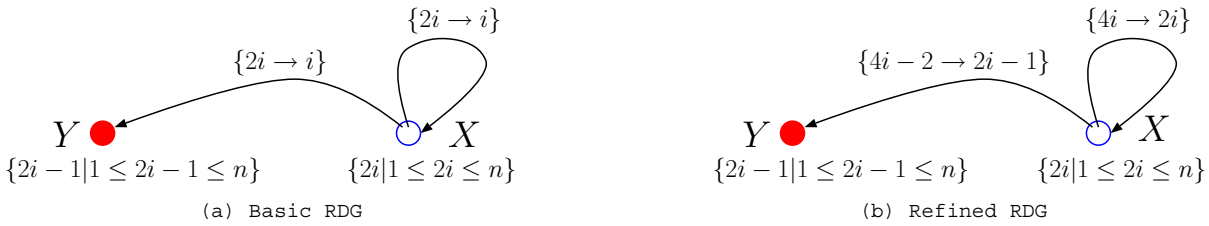


Figure 3. Basic and Refined Reduced Dependence Graphs for example 1.

$\mathcal{D}_{V,i}$, and let Y be a variable derived from U defined on $\mathcal{Z}_Y \in \mathcal{D}_U$ where $\mathcal{Z}_X$ and $\mathcal{Z}_Y$ are given as follows

$$\begin{aligned} \mathcal{Z}_X &= \{L_X z_X + l_X | z_X \in \mathcal{P}_X^c\} \\ \mathcal{Z}_Y &= \{L_Y z_Y + l_Y | z_Y \in \mathcal{P}_Y^c\} \end{aligned}$$

A dependence of the form $(Lz + l \to Rz + r)$ is directed form X to Y. X at $(Lz+l) \in \mathcal{Z}_X$ cannot be evaluated before Y at $(Rz+r) \in \mathcal{Z}_Y$. The affine lattice $\{Lz+l|z \in \mathbb{Z}^n\}$ may contain points that do not lie in the affine lattice $\{L_X z_X + l_X | z_X \in \mathbb{Z}^{n_X}\}$. Similarly, the affine lattice $\{Rz + r | z \in \mathbb{Z}^n\}$ may contain points that do not lie in the affine lattice $\{L_Y z_Y + l_Y | z_Y \in \mathbb{Z}^{n_Y}\}$. As a result, the dependence may be specified on a finer lattice than necessary and may safely be replaced by dependence of the form $(L'z' + l' \to R'z' + r')$ where

$$\begin{aligned} L' = L_X S, l' = L_X s + l_X \\ R' = L_Y S', r' = L_Y s' + l_Y \end{aligned} \tag{3}$$

where S and S' are matrices and s and s' are integer vectors. The refined RDG is a refinement of the basic RDG where every dependence has been replaced by a dependence satisfying Equation (3). Figure 3b gives the refined RDG for example 1.

5. The Scheduling Problem

Here, we will present the precedence imposed by dependences and then formulate an integer linear program to obtain valid schedules.

5.1. Causality Constraints

Dependences between the different iterations of variables impose an ordering on their evaluation. A valid schedule of the evaluation of these iterations is the assignment of an execution time to each computation so that precedence (causality) constraints are satisfied.

Let X and Y be two variables in the refined RDG defined on $\{L_X z_X + l_X | z_X \in \mathcal{P}_X^c\}$ and $\{L_Y z_Y + l_Y | z_Y \in \mathcal{P}_Y^c\}$ respectively. We seek to find schedules on X and Y of the following form

$$\begin{aligned} \lambda'_X(z_X) &= (L_X z_X + l_X \to \lambda_X(z_X)) \\ \lambda'_Y(z_Y) &= (L_Y z_Y + l_Y \to \lambda_Y(z_Y)) \end{aligned} \tag{4}$$

where λ_X and λ_Y are affine functions on z_X and z_Y respectively. Our motivation for such schedules is that all vectors and matrices comprise of integer scalars. If we seek schedules of the form $\lambda'(z')$ where λ' is an affine function and z' is an iteration in the domain of a variable, then we may potentially assign execution times to "holes" or computations that do not exist. In general, we seek multidimensional schedules in the form of affine functions on coordinate indices.

We will now formulate causality constraints using the refined RDG. Consider dependences from X to Y. All such dependences can be written as

$$(L_X(Sz + s) + l_X \to L_Y(S'z + s') + l_Y)$$

where S and S' are matrices and s and s' are vectors. The execution time for Y at $L_Y(S'z + s') + l_Y$ should precede the execution time for X at $L_X(Sz + s) + l_X$. With the nature of the schedules presented in Equation (4), our causality constraint becomes

$$\lambda_X(Sz + s) - \lambda_Y(S'z + s') \ge 1 \tag{5}$$

5.2. ILP Formulation

Here, we extract ILP (integer linear programming) constraints from causality constraints, as well as non-negativity of schedule. Then, we complete the ILP formulation with an objective function. Since our aim is using the PIP (Parameter Integer Programming) solver [5], the ILP formulation will be suitable form for the solver. We also show how a multidimensional schedule can be obtained in this model.

First, consider the non-negativity of schedule and therefore of the affine function λ. For each variable X, we want to impose the following condition:

$$\forall z \in \mathcal{P}_X^c, \lambda_X(z) \ge 0$$

By affine form of Farkas Lemma, this condition holds when $\lambda_X(z)$ is a non-negative affine combination of the constraints $\mathcal{C}$ of $\mathcal{P}_X^c$, i.e.

$$\lambda_X(z) \equiv \lambda_{X,0} + \sum_{k=1}^{b_X} \lambda_{X,k}(a_{X,k}^T z + \alpha_{X,k})$$

where $\lambda_{X,i} \ge 0$ for all $i = 0, \ldots, b_X$ and $a_{X,k}^T z + \alpha_{X,k} \ge 0$ is the k-th constraint of $\mathcal{C}$. From now on, this is a prototype of affine schedule functions.

Now, consider the causality constraint presented in Equation (5) for the dependence from X to Y.

$$\begin{aligned}\lambda_{X,0} + \left(\sum_{k=1}^{b_X} \lambda_{X,k}(a_{X,k}^T(Sz+s)+\alpha_{X,k})\right) - \lambda_{Y,0} \\ - \left(\sum_{k=1}^{b_Y} \lambda_{Y,k}(a_{Y,k}^T(S'z+s')+\alpha_{Y,k})\right) - 1 \quad \geq \quad 0\end{aligned}$$

where $Sz+s \in \mathcal{P}_X^c$ and $S'z+s' \in \mathcal{P}_Y^c$. Equivalently, we may say $z \in \mathcal{P}'_X \cap \mathcal{P}'_Y$ where $\mathcal{P}'_X$ and $\mathcal{P}'_Y$ are the preimage of $\mathcal{P}_X^c$ by $(z \to Sz+s)$ and $\mathcal{P}_Y^c$ by $(z \to S'z+s')$ respectively.

Now, first we will illustrate how to formulate the ILP constraints with the help of an example. Then, we will introduce an objective function for minimizing latency.

Consider Example 1 and its refined RDG given in Figure 3b. By the non-negativity constraints, the affine functions λ_X and λ_Y will be of the form

$$\lambda_X = \lambda_{X0} + \lambda_{X1}(2i-1) + \lambda_{X2}(N-2i)$$

$$\lambda_Y = \lambda_{Y0} + \lambda_{Y1}(2i-2) + \lambda_{Y2}(N-2i+1)$$

For the edge $(4i-2 \to 2i-1)$ from X to Y, the causality constraint is

$$\begin{aligned}\lambda_{X0} + \lambda_{X1}(4i-3) + \lambda_{X2}(N-4i+2) \\ -\lambda_{Y0} - \lambda_{Y1}(2i-1) - \lambda_{Y2}(N-2i-1) \quad \geq \quad 1 \end{aligned} \quad (6)$$

Similarly, an ILP constraint for the edge $(4i \to 2i)$ translates into the following condition

$$\begin{aligned}\lambda_{X0} + \lambda_{X1}2i + \lambda_{X2}(N-4i) \\ -\lambda_{X0} - \lambda_{X1}i - \lambda_{X2}(N-2i) \quad \geq \quad 1 \end{aligned} \quad (7)$$

where all the unknowns are non-negative. Since Y is assigned to a constant, one possibility is $\lambda_{Y0} = \lambda_{Y1} = \lambda_{Y2} = 0$. In this case, Equation (6) can be simplified to the following form

$$\begin{aligned}(\lambda_{X0} - \lambda_{X1} + \lambda_{X2} - 1) + (2\lambda_{X1} - \lambda_{X2})(2i-1) \\ +(\lambda_{X2})(N-2i) \quad \geq \quad 0\end{aligned}$$

Similarly, Equation (7) also simplifies to

$$(\lambda_{X1} - \lambda_{X2} - 1) + (\lambda_{X1} - \lambda_{X2})(2i-1) \geq 0$$

Now, we have the following inequalities

$$\begin{aligned}\lambda_{X0} - \lambda_{X1} + \lambda_{X2} - 1 &\geq 0 \\ 2\lambda_{X1} - \lambda_{X2} &\geq 0 \\ \lambda_{X0} &\geq 0 \\ \lambda_{X1} - \lambda_{X2} - 1 &\geq 0 \\ \lambda_{X1} - \lambda_{X2} &\geq 0\end{aligned} \quad (8)$$

All these steps for deriving ILP constraints can be performed automatically and these inequalities can be systematically solved by standard ILP solvers.

Now, we present an objective function when all domains are bounded. In this case, there exists an affine expression $\mathcal{L}$ of the program parameters such that $\mathcal{L} - \lambda_X(z) \geq 0$ for all $z \in \mathcal{P}_X^c$ if there exists an affine function λ_X. So, it does not restrict the space of valid affine function λ_X. We want to minimize the total latency by minimizing $\mathcal{L}$. In order to use the PIP solver, for each variable X we first add $\mathcal{L} - \lambda_X(z) \geq 0$ to the constraints, and the unknown variables in $\mathcal{L}$ are placed into the outermost dimensions because PIP solver gives the lexicographic minimum of a given parameterized polyhedron.

Let us continue Example 1. For the minimum latency affine schedule, we add the following constraints from $\mathcal{L}(= \mu_0 N + \mu_1) - \lambda_X(z)$. Here, $(\mu_0 N + \mu_1)$ is the prototype of latency and μ_0 and μ_1 are unknown constants. Solving, we get

$$\begin{aligned}\mu_0 N + \mu_1 - \lambda_{X0} - \lambda_{X1}(2i-1) - \lambda_{X1}(N-2i) \quad \geq \quad 0 \\ \text{or } (\mu_0 - \lambda_{X0})(N-2i) + (\mu_0 - \lambda_{X1})(2i-1) \\ +(\mu_0 + \mu_1 - \lambda_{X0}) \quad \geq \quad 0\end{aligned}$$

which implies the following

$$\begin{aligned}\mu_0 - \lambda_{X0} &\geq 0 \\ \mu_0 - \lambda_{X1} &\geq 0 \\ \mu_0 + \mu_1 - \lambda_{X0} &\geq 0\end{aligned} \quad (9)$$

Note that for the sake of simplicity, we do not consider the variable Y that is constant on all its iterations. In fact, Equations (8) and (9) define a polyhedron. Since we put μ_0 into the first dimension index, the lexicographic minimum of this polyhedron will provide minimum latency schedules for each variable. Finally, we will get the following schedules for X and Y

$$\begin{aligned}\lambda'_X \quad &: \quad (2i \to i) \\ \lambda'_Y \quad &: \quad (2i+1 \to 0)\end{aligned}$$

Now, consider multidimensional schedules in this model. The basic idea is the same as that in Polyhedral model. Thus, rather than presenting technical details, we just provide a precise and intuitive (maybe inefficient) formulation for multidimensional schedules. The basic idea of the following ILP formulation is satisfying as many dependences as possible and the theory justifying this formulation is given by Feautrier [8].

$$\begin{gathered}\max \sum_{e \in E} x_e \\ \text{subject to } 0 \leq x_e \leq 1 \\ \lambda_X(Sz+s) - \lambda_Y(S'z+s') - 1 \geq x_e \\ \lambda_X(z) \geq 0\end{gathered}$$

A new variable x_e for each $e \in E$ is introduced. When $x_e = 1$, the dependence associated to edge e will be satisfied. The objective function maximizes the number of the edges that are satisfied. The multidimensional time can be obtained in this model by the same way in the polyhedral model.

The purpose of this section was to present the transformation of causality constraints into ILP constraints precisely and

the illustration of this step with the help of an example. For technical details on Farkas scheduling algorithm, please refer to [7] by Feautrier.

6. Related Work

The scheduling problem is one of the most widely studied problems in the polyhedral model. For a detailed exposition of the various scheduling techniques, we would like to refer the interested reader to the text by Darte et al. [4].

The scheduling problem on recurrence equations with uniform (constant-sized) dependences was originally presented by Karp et. al. [12]. A similar problem was posed by Lamport [13] for programs with uniform dependences. Quinton showed how these techniques can be used for systolic synthesis [21, 22]. Shang and Fortes [27] and Lisper [17] presented optimal linear schedules for uniform dependence algorithms. Rao [26] first presented affine by variable schedules for uniform dependences and also proposed an improvement of the technique by Karp et al. [12] for multidimensional schedules. The first result of scheduling programs with affine dependences was solved by Rajopadhye and Fujimoto [24], and independently by Quinton and Van Dongen [23]. These results were generalized to variable dependent schedules by Mauras et. al. [20]. Feautrier presented the optimal solution to the affine scheduling problem (by variable) [7]. Feautrier also provided the extension to multidimensional time [8].

Darte and Robert [3] (as well as Feautrier [7] and Quinton [22]) formulate the schedule as the floor of a rational affine function, thus enabling LP rather than ILP to be used for efficient solutions. However, given a rational schedule, it is not known how to produce a space-time mapping and subsequently generate code. The schedules that we construct from the techniques presented in this paper can be used to perform appropriate program transformations to obtain an equivalent specification that can be input directly to the final code generation step, which has already been solved by Bastoul [1]. Also, we have extended the scheduling analysis to a class of programs that is strictly more general than those considered previously.

7. Conclusions and Future Work

The polyhedral model has been widely studied for the automatic parallelization of loop programs. However, it is limited in expressivity. It was recently extended to $\mathcal{Z}$-polyhedral domains [10]. In this paper, we presented the scheduling analysis for the $\mathcal{Z}$-polyhedral model, a strict generalization of the polyhedral model, to obtain multidimensional schedules with minimum latency. Our schedules are of the form $\lambda'(z) = (Lz + l \rightarrow \lambda(z))$ where $\{Lz + l | z \in \mathbb{Z}^n\}$ is the lattice corresponding to the $\mathcal{Z}$-polyhedron and $\lambda(z)$ is the affine function (comprising of integer scalars) on the coordinates of this lattice. As a result, an important property of our ILP formulation is that it searches only in the space of functions that can subsequently be used to construct a space-time transformation of the program.

Our schedules express unbounded parallelism and thus cannot directly be realized in any parallel hardware. However, such parallelism detection is the crucial precursor to any resource constrained analysis.

Future work involves the implementation of these transformations into a compiler optimization framework.

References

[1] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *IEEE PACT*, pages 7–16, 2004.

[2] C. Bastoul, A. Cohen, A. Girbal, S. Sharma, and O. Temam. Putting polyhedral loop transformations to work. In *Languages and Compilers for Parallel Computers*, pages 209–225, Oct 2003.

[3] A. Darte and Y. Robert. Affine-by-statement scheduling of uniform and affine loop nests over parametric domains. *J. Parallel Distrib. Comput.*, 29(1):43–59, 1995.

[4] A. Darte, Y. Robert, and F. Vivien. *Scheduling and Automatic Parallelization*. Birkhäuser, 2000.

[5] P. Feautrier. Parametric integer programming. *Operationnelle/Operations Research*, 22(3):243–268, 1988.

[6] P. Feautrier. Dataflow analysis of array and scalar references. *Int. Journal of Parallel Programming*, 20(1):23–53, 1991.

[7] P. Feautrier. Some efficient solutions to the affine scheduling problem. part I. one-dimensional time. *Int. Journal of Parallel Programming*, 21(5):313–348, Oct 1992.

[8] P. Feautrier. Some efficient solutions to the affine scheduling problem. part II. multidimensional time. *Int. Journal of Parallel Programming*, 21(6):389–420, Dec 1992.

[9] A. Fernández, J. M. Llabería, and M. Valero-García. Loop transformation using nonunimodular matrices. *IEEE Trans. Parallel Distrib. Syst.*, 6(8):832–840, 1995.

[10] Gautam and S. Rajopadhye. The $\mathcal{Z}$-polyhedral model. In *PPoPP'07: Symposium on Principles and Practice of Parallel Programming (accepted)*, 2007.

[11] C. Hermite. Sur l'introduction des variables continues dans la theorie des nombres. *J. Reine Angew. Math.*, 41:191–216, 1851.

[12] R. M. Karp, R. E. Miller, and S. V. Winograd. The organization of computations for uniform recurrence equations. *JACM*, 14(3):563–590, July 1967.

[13] L. Lamport. The parallel execution of DO loops. *Communications of the ACM*, pages 83–93, February 1974.

[14] H. Le Verge. *Un environnement de transformations de programmmes pour la synthèse d'architectures régulières*. PhD thesis, L'Université de Rennes I, IRISA, Campus de Beaulieu, Rennes, France, Oct 1992.

[15] P. Lenders and S. V. Rajopadhye. Multirate VLSI arrays and their synthesis. Technical Report 94-70-01, Oregon State University, December 1994.

[16] W. Li and K. Pingali. A singular loop transformation framework based on non-singular matrices. *Int. J. Parallel Program.*, 22(2):183–205, 1994.

[17] B. Lisper. Linear programming methods for minimizing execution time of indexed computations. In *Int. Workshop on Compilers for Parallel Computers*, 1990.

[18] R. B. Lyngsø, M. Zuker, and C. N. S. Pedersen. Fast evaluation of internal loops in rna secondary structure prediction. *Bioinformatics*, 15(6):440–445, 1999.

[19] C. Mauras. *ALPHA: un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones*. PhD thesis, L'Université de Rennes I, Rennes, France, December 1989.

[20] C. Mauras, P. Quinton, S. Rajopadhye, and Y. Saouter. Scheduling affine parametrized recurrences by means of variable dependent timing functions. In *International Conference on Application Specific Array Processing*, pages 100–110, 1990.

[21] P. Quinton. Automatic synthesis of systolic arrays from uniform recurrent equations. In *ISCA '84: Proceedings of the 11th annual international symposium on Computer architecture*, pages 208–214, 1984.

[22] P. Quinton. The systematic design of systolic arrays. In F. Fogelman Soulie, Y. Robert, and M. Tchuente, editors, *Automata Networks in Computer Science*, chapter 9, pages 229–260. Princeton University Press, 1987. Preliminary versions appear as IRISA Tech Reports 193 and 216, 1983, and in the proceedings of the IEEE Symposium on Computer Architecture, 1984.

[23] P. Quinton and V. Van Dongen. The mapping of linear recurrence equations on regular arrays. *Journal of VLSI Signal Processing*, 1(2):95–113, 1989.

[24] S. V. Rajopadhye, S. Purushothaman, and R. M. Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In *Foundations of Software Technology and Theoretical Computer Science*, pages 488–503, 1986.

[25] J. Ramanujam. Beyond unimodular transformations. *J. Supercomput.*, 9(4):365–389, 1995.

[26] S. K. Rao. *Regular Iterative Algorithms and their Implementations on Processor Arrays*. PhD thesis, Stanford University, Information Systems Lab., Stanford, Ca, October 1985.

[27] W. Shang and J. Fortes. Time optimal linear schedules for algorithms with uniform dependencies. *IEEE Transactions on Computers*, 40(6):723–742, June 1991.

[28] D. Wilde. A library for doing polyhedral operations. Technical Report PI 785, IRISA, Rennes, France, Dec 1993.

[29] B. Wilkinson and M. Allen. *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, Inc., 1999.

[30] J. Xue. Automating non-unimodular loop transformations for massive parallelism. *Parallel Computing*, 20(5):711–728, 1994.