Scheduling Reductions on Realistic Machines

Gautam†
ggupta@cs.colostate.edu
Patrice Quinton‡
quinton@irisa.fr
Sanjay Rajopadhye‡
svr@cs.colostate.edu

ABSTRACT
Many computations can be modeled with systems of affine recurrence equations (SAREs) over polyhedral domains. We study the problem of scheduling individual computations of an SARE in the presence of reductions i.e., operations specifying the accumulation of a set of values to produce a single value. Reductions involve a commutative and associative operator and therefore, per se, do not impose any specific order. However, on realistic machines, operators have bounded fan-in and therefore an order of accumulation (serialization) is needed. Arbitrary serializations may adversely affect the running time of a program. We develop an algorithm to determine efficient serializations of all reductions. We illustrate our methods with two significant examples.

Categories and Subject Descriptors
F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—Sequencing and scheduling; G.2.2 [Discrete Mathematics]: Graph Theory—Graph algorithms

General Terms
Scheduling Theory, Algorithms, Languages

Keywords
Affine recurrence equations, polyhedral model, dependence

1. INTRODUCTION
Recurrence equations are widely used to specify many scientific and mathematical computations. They permit a highly parallel implementation. Systems of affine recurrence equations (SAREs) over polyhedral domains (the domain of a recurrence equation is the set of index points where it is defined) constitute the so-called polyhedral model—a formalism for reasoning about massively parallel computations (potentially even infinite) with a compact (finite) representation. The origins of the model go back to Systems of Uniform Recurrence Equations first proposed by Karp et al. [8]. This formalism was adapted and extended by researchers in the systolic synthesis community. Later, Feautrier [4] showed how SAREs may be used as an intermediate form for automatic parallelization of imperative loop programs, by giving an algorithm for exact dataflow analysis on a certain class of loops (called static control) in an imperative program. Although the work presented in this paper is based on SAREs over polyhedral domains, it can be easily adapted to the context of loop parallelization.

In this paper we address the scheduling problem in the presence of reductions, an even higher level representation of computations. A reduction is the application of a commutative and associative operator on a set of elements to produce a single value (for example, the summation operator on numbers). A first attempt to schedule computations in the presence of reduction operators is to assume that each reduction takes a single time step. However, this would require an operator with unbounded fan-in i.e., a concurrent write (CRCW) PRAM. This is unrealistic and it is therefore necessary to order the accumulation in reductions. This ordering is called serialization. The reduction operator is commutative and associative and therefore all serializations will yield the same result. We will see that arbitrary serializations may adversely affect the schedule (and therefore the running-time) of a program. In this paper, we propose an algorithm to determine the necessary and sufficient

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SPAA’02, August 10-13, 2002, Winnipeg, Manitoba, Canada.
Copyright 2002 ACM 1-58113-529-7/02/0008 ...$5.00.
conditions for a (piece-wise) linear/affine schedule that permits serialization of all reductions without “unacceptable” slowdown.

In our presentation, we start from the ideal case (concurrent write PRAM) which provides us with the lower bound of any valid schedule (the schedule for reductions as proposed by Redon and Feautrier [20]). Then, we will extend the scheduling problem to consider the restriction imposed by bounded fan-in operators.

A valid schedule is a solution of the constraints imposed by the causality of computations. These constraints may be formulated as a parametric integer program (PIP) [22], and a solution can be obtained using a PIP solver [3]. Thus, scheduling reductions (on realistic machines) is simply the derivation of additional constraints imposed by serialization and including them as input to the PIP solver. We prove that the derived constraints are both necessary and sufficient.

The remainder of this paper is organized as follows. Section 2 motivates the problem with two simple examples. Section 3 details the necessary background—the SARE formalism and ALPHA, a language based on it. Section 4 proposes our scheduling algorithm, and detailed examples are provided in section 5. Section 6 discusses related work and our scheduling algorithm, and detailed examples are provided in section 5. Section 7 concludes the paper. Proofs are given in the appendix.

2. MOTIVATIONS

2.1 Examples

Example 1: Convolution is the weighted sum of n consecutive values of a variable x to produce a value of y. The computation involved has the following form

\[ i \geq n : \quad y_i = \sum_{j=1}^{n} w_j x_{i-j} \]  

(1)

All values of x (and w) are input to the system and therefore are available at \( t = 0 \). In a CRCW PRAM (with an unbounded number of processors), once all arguments are known, reductions are evaluated in a single time step, and therefore all values of y are available at \( t = 1 \).

However, if only binary (or constant fan-in) accumulations were permitted, we would have to introduce a temporary variable to hold the partial results of accumulations. Equation (1) could be modified as:

\[ i \geq n : \quad c_{i,j} = \begin{cases} \text{ if } j = 1 & w_j x_{i-j} \\ \text{ if } 1 < j \leq n & c_{i,j-1} + w_j x_{i-j} \end{cases} 
\]

(2)

Given an unbounded number of binary operators, we could start the accumulation for all values of y simultaneously. Our program would terminate after \( n \) steps (linear time). Note that in any other order of accumulation for (example from \( n \) down to 1), \( n \) steps would still be needed.

Thus, this is an example of a computation involving reductions that cannot be serialized without a slowdown (The execution time changed from constant time to linear time).

Example 2: Recursive Convolution is very similar to (simple) convolution except that y is an input as well as an output (with \( y_i \) for \( -n \leq i \leq 0 \) given as input) defined as

\[ y_i = \sum_{j=1}^{n} w_j y_{i-j} \]  

(3)

Observe that the value of \( y \) at a certain index point depends on the values at the previous \( n \) indices, which in turn depend on the value of \( y \) at even earlier indices. By a simple induction, we can prove that \( y_i \) cannot be computed unless \( y_{i'} \) has been computed. Therefore, even on a CRCW PRAM, all values of \( y \) cannot be computed simultaneously.

They have to be computed in a certain order (\( y_i \) at time \( t_i = i \)). This is thus the lower bound of execution time on any machine.

Now we want to perform the same computations on a realistic machine. Similar to the previous example, through the use of a temporary variable (to hold partial results), we may serialize the reduction as follows

\[ i > 0 : \quad c_{i,j} = \begin{cases} \text{ if } j = 1 & w_j y_{i-j} \\ \text{ if } 1 < j \leq n & c_{i,j-1} + w_j y_{i-j} \end{cases} \]

(4)

Given an unbounded number of binary operators, we could start the accumulation for all values of \( y \) simultaneously. Our program would terminate after \( n \) steps (linear time). Note that in any other order of accumulation for (example from \( n \) down to 1), \( n \) steps would still be needed.

Thus, this is an example of a computation involving reductions that cannot be serialized without a slowdown (The execution time changed from constant time to linear time).

Example 2: Recursive Convolution is very similar to (simple) convolution except that \( y \) is an input as well as an output (with \( y_i \) for \( -n \leq i \leq 0 \) given as input) defined as

\[ y_i = \sum_{j=1}^{n} w_j y_{i-j} \]  

(3)

Observe that the value of \( y \) at a certain index point depends on the values at the previous \( n \) indices, which in turn depend on the value of \( y \) at even earlier indices. By a simple induction, we can prove that \( y_i \) cannot be computed unless \( y_{i'} \) has been computed. Therefore, even on a CRCW PRAM, all values of \( y \) cannot be computed simultaneously.

They have to be computed in a certain order (\( y_i \) at time \( t_i = i \)). This is thus the lower bound of execution time on any machine.

Now we want to perform the same computations on a realistic machine. Similar to the previous example, through the use of a temporary variable (to hold partial results), we may serialize the reduction as follows

\[ i > 0 : \quad c_{i,j} = \begin{cases} \text{ if } j = 1 & w_j y_{i-j} \\ \text{ if } 1 < j \leq n & c_{i,j-1} + w_j y_{i-j} \end{cases} \]

(4)

Given an unbounded number of binary operators, we could start the accumulation for all values of \( y \) simultaneously. Our program would terminate after \( n \) steps (linear time). Note that in any other order of accumulation for (example from \( n \) down to 1), \( n \) steps would still be needed.

Thus, this is an example of a computation involving reductions that cannot be serialized without a slowdown (The execution time changed from constant time to linear time).

Example 2: Recursive Convolution is very similar to (simple) convolution except that \( y \) is an input as well as an output (with \( y_i \) for \( -n \leq i \leq 0 \) given as input) defined as

\[ y_i = \sum_{j=1}^{n} w_j y_{i-j} \]  

(3)

Observe that the value of \( y \) at a certain index point depends on the values at the previous \( n \) indices, which in turn depend on the value of \( y \) at even earlier indices. By a simple induction, we can prove that \( y_i \) cannot be computed unless \( y_{i'} \) has been computed. Therefore, even on a CRCW PRAM, all values of \( y \) cannot be computed simultaneously.

They have to be computed in a certain order (\( y_i \) at time \( t_i = i \)). This is thus the lower bound of execution time on any machine.

Now we want to perform the same computations on a realistic machine. Similar to the previous example, through the use of a temporary variable (to hold partial results), we may serialize the reduction as follows

\[ i > 0 : \quad c_{i,j} = \begin{cases} \text{ if } j = 1 & w_j y_{i-j} \\ \text{ if } 1 < j \leq n & c_{i,j-1} + w_j y_{i-j} \end{cases} \]

(4)
partial result has definitely been computed when \( y_{i-n+2} \) is available. Continuing, we get \( t_i = t_{i-1} + 1 \). Solving this recurrence equation with our base case we get\(^1 \) \( t_i = i \).

Thus, this serialization (shown in figure 1) retained the complexity of execution time on a realistic machine. The goal of this paper is to develop a scheduling algorithm that discovers such “good” serializations automatically.

2.2 Linear vs. Logarithmic Serializations

We first recall the standard notions of bounds and order-relationships on functions. A function \( f(z) \) is said to have an upper bound \( u(z) \) within a domain \( D \) if for all \( z \in D \), \( f(z) < cu(z) \) for some constant \( c \). Similarly, \( f \) is said to have a lower bound \( l(z) \) if \( f(z) > cl(z) \) for all points in the domain. If \( f(z) \) is both an upper and a lower bound on \( f(z) \), we will say that they have the same order. Finally we say that \( f(z) \) contains \( l(z) \) if \( f(z) > l(z) \) at all points in the domain (i.e., \( c = 1 \)). We are interested in the case when \( f \), \( f' \), \( u \) and \( l \) are (piece-wise) linear functions.

It is well known that the fastest accumulation of \( n \) values (on \( \theta(n) \) processors) can be achieved in \( \log n \) time. However, this is overkill since it is work inefficient even on a PRAM. A work efficient implementation on \( p \) processors would partition the input data and take \( \frac{n}{p} + \log p \) steps. In the polyhedral model, a similar strategy would require \( \frac{n}{p} + p \) steps. For \( n \gg p \), the linear schedule is not significantly slower. Moreover, there is a large body of work on computing optimal linear schedules in the polyhedral model, and such schedules simplify the subsequent code generation problem.

In this paper, we will consider linear accumulations, i.e., accumulation strategies where combining all values along a line of index points takes time proportional to the number of index points on that line. Given a set of index points (and a value associated with each point), we define its size as the lower bound on the time to accumulate these values in this model. The following theorem (see appendix for proof) considers the problem of accumulating values in an \( n \)-dimensional “box” (rectangular parallelepiped).

**Theorem 1.** The size of a box \( B \) equals the 1-norm (i.e., the Manhattan distance) of its principal diagonal.

3. BACKGROUND

3.1 Systems of Affine Recurrence Equations

A parametrized system of affine recurrence equations (SARE) is a finite set of equations of the form

\[
\forall z, p \in D, \forall p \in P : U(z,p) = f(\ldots, V(dep(z,p), p), \ldots) \tag{6}
\]

where

- \( U \) and \( V \) are variables from a finite set \( V \). Each variable \( U \in V \) may be interpreted as a function \( U : D_U \rightarrow X \), from its domain \( D_U \) to a set \( X \) of values (eg. real, integers, etc.).

- In this paper, domains are integer polyhedra, i.e., \( D_U = \{(z, p) \in Z^{d_U} | A \begin{pmatrix} z \\ p \end{pmatrix} \geq b \} \)

\(^1\) or \( t_i = i + n - 1 \) if the first accumulation takes \( n \) steps to complete (\( y_i \) is computed at \( t = n \)).

- \( p \in P \subseteq Z^* \) is a vector of the size parameters of the system where \( P \) is also an integer polyhedron.

- \( D \) is an integer polyhedron in \( Z^{d_U} \) such that \( D \subseteq D_U \). It is called a domain of the equation. Note that the domain \( D \) of an equation such as (6) is in general a subset of \( D_U \) because a variable may be defined by more than one equation. The domains of all equations that have the same variable, \( U \) on the left hand side must form a partition of \( D_U \). This ensures that variables are not under- or over-defined.

- \( dep(z,p) = D \begin{pmatrix} z \\ p \end{pmatrix} + d \) is an affine function from \( Z^{d_U} \) to \( Z^{d_U} \) called a dependency function (dependency for short). Note that the SAREs where all variables have the same dimensions and all dependencies have the same linear part \( D = I \) (here \( I \) is the identity matrix), are called uniform (SURE), since \( dep(z,p) \) reduces to \( z + d \). Hence, SUREs are a proper subset of SAREs.

- \( f \) is an arbitrary, strict, single-valued function which is viewed as atomic for analysis purposes and executes in constant time (when we introduce the notion of reductions, this may no longer be the case).

Parameters \( p \) are almost identical to indices \( z \) except that within individual program instances they are constant (do not vary in the polyhedron \( P \subseteq Z^* \) of size parameters), and determine the size of computations in the program. Therefore, indices and parameters are sometimes treated homogeneously by setting \( z = (z,p) \). We will call this the set of extended indices of the variable.

It is possible to augment the above definition of SAREs to include reduction operations. Consider the following examples.

**Example:** Computing with SAREs: Forward Substitution

Let us consider solving, using forward substitution, a system of linear inequalities, \( Az = b \), where \( A \) is a lower triangular \( n \times n \) matrix with unit diagonal. The SARE-like specification (with \( n \) as the size parameter) of the computation would be

\[
n > 1 : \quad x_i = \begin{cases} b_i & \text{if } i = 1 \\ b_i - \sum_{j=1}^{i-1} A_{i,j} x_j & \text{if } 1 < i \leq n \end{cases} \tag{7}
\]

where the \( \sum \) is a reduction operation. We now introduce, via the ALPHA language, the precise syntax of reductions in SAREs.

3.2 The ALPHA language

ALPHA is a data parallel functional language based on the SARE formalism [11, 23], and has been used for automatic derivation of systolic and regular VLSI arrays, and for parallel code generation. Semantically, ‘ALPHA programs’ are identical to ‘SAREs (augmented with reductions) over polyhedral domains’ [24, 15], and shall use these terms interchangeably.

**Example:** Forward Substitution in Alpha

The ALPHA program for (7) is given in figure 2. Without delving into other details, let us study equations in ALPHA. The right hand side of an equation is an expression which is either (i) a variable or constant, (ii) a pointwise operator applied to one or more (typically two) expressions, (iii) the restriction of an expression to a domain, (iv) a case expression,
(v) the composition of an expression with a dependency, or

The \textit{case} construct has the usual meaning and allows us to define conditional expressions. We may therefore insist, without loss of generality, that there is only one equation for each variable (the rhs of multiple equations can be grouped into a \textit{case}). The \textit{restrict} construct \{(i = 1) \text{expr} immediately following the \textit{case} construct\} denotes \textit{expr} but restricted to a subset of index points as specified by a restriction domain. The \textit{reduce} construct corresponds to the summation in (7) and has three parts: an associative and commutative operator (+ here), a projection function ((i,j \rightarrow i)), and an expression or body (A \times X.(i,j \rightarrow j)).

The projection function maps the domain of the body to a lesser dimensional space. In the program, (i,j \rightarrow j) in the expression (body) of the reduce, is a dependency function and denotes the fact that to compute the body at [1,j], we need the value of X at [j] (the dependency on A is not explicitly written; it is the identity).

Dependences and projection functions have the syntax (idx, idx, ... \rightarrow i-expr, i-expr, ...), where every idx is an index name, and i-expr is an affine expression of the extended indices (system parameters and the idx’s). This syntax can be viewed as a special kind of lambda expression, restricted to affine mappings from $\mathbb{Z}^n$ to $\mathbb{Z}^m$. Such a function, f, may be equivalently represented by an m \times n matrix A, and an m-vector a, i.e., $f(z) = A \left( \begin{array}{c} z \\ p \end{array} \right) + a$.

All subexpressions in \texttt{ALPHA} (in addition to variables) also denote a function from indices to values in a polyhedral domain. These domains can be derived constructively from the domains of their subexpressions, (summarized in [15]).

3.2.1 Normalization

Domains in \texttt{ALPHA} obey certain closure properties, and this has an important consequence. Every expression can be simpliﬁed into a semantically equivalent unique normal form, called the case-restriction-dependency form. It consists of an (optional) outer case, each of whose branches is a (possibly restricted) simple expression. A simple expression consists of a unit composed with a single dependency function (if this is the identity, it may be omitted), or pointwise operators applied to such subexpressions\footnote{During scheduling we may specify the instantaneous execution of some expressions. Therefore breaking an expression does not slow down the program schedule.}.

\begin{itemize}
  \item \textbf{Expressions with (nested) reductions:} \\
    Normalization, as presented above, does not handle programs with reductions. We propose a straightforward extension to the normalization rules which introduces a new local variable for the body of every \textit{reduce} subexpression, and substitutes it by the corresponding local variable prior to normalization. The \textit{modified} unit (in the normal form) is either a variable, a constant or the reduction of a variable. Our analysis in the rest of this paper assumes normalized \texttt{ALPHA} programs.
\end{itemize}

3.2.2 Parametrized Reduction Domain

Although, our analysis can be performed on the normal form presented above, to aid explanation, we may assume, without loss of generality, that every equation containing a reduction has the form

$$X = \text{reduce} \left( \bigoplus_i \left( z \rightarrow A_P \left( \begin{array}{c} z \\ p \end{array} \right) + b_P \right), R \right)$$

where $A_P$ is a constant matrix, $b_P$ a constant vector and $R$ is some variable. Specifically, we may achieve this by assigning every \textit{reduce} subexpression to a local variable.

The accumulations of values in a subset of the domain of $R$ is with $\oplus$. For any $z_X$, a point in the domain of $X$, we define its \textit{parametrized reduction domain} $\mathcal{P}(z_X)$ as the set of points in the domain of $R$ that are mapped to $z_X$ by the function $z \rightarrow A_P \left( \begin{array}{c} z \\ p \end{array} \right) + b_P$

$$\mathcal{P}(z_X) = \{ z \in \text{Dom}(R) | A_P \left( \begin{array}{c} z \\ p \end{array} \right) + b_P = z_X \}$$

Note that $\mathcal{P}$ is parametrized both by the program parameter $p$ and the index $z_X$\footnote{Note that we allow $T_{eqX}$ to be a function of two points $z$ and $z'$ for full generality. In most practical cases, however, it is a constant.}.

4. Scheduling Reductions

To aid explanation, we will first present our algorithm assuming that the domain of $R$ is a single polyhedron denoted by $\text{Dom}(R)$ and has an affine schedule. Later we will show that the algorithm for the general case (domain of $R$ being a union of polyhedra with a piece-wise affine schedule) is a straightforward extension.

\textbf{Causality:} The scheduling function must respect \textit{causality} i.e., for any two variables $X$ and $Y$, whenever $X(z)$ depends on $Y(z')$ then $X(z)$ can only be computed after $Y(z')$ is evaluated. The schedule for every variable $(V)$ is a \textit{multidimensional affine function} ($\lambda_V$), such that $\lambda_V(z_V)$ is a vector that represents the multidimensional time instant when $V(z_V)$ is executed. Assuming that the computation of the rhs of the equation for $X(z)$ takes a single time unit once $Y(z')$ is computed, the causality constraint\footnote{Since program parameters are constant across an execution of the program, as a convention, we drop them from the parameter list of functions. However, it should be remembered that they parametrize the entire program.} on the schedule is

$$\lambda_Y(z') + 1 \leq \lambda_X(z)$$

where ‘\leq’ denotes lexicographic ‘lesser than or equal to’. In the general case, where computation of the RHS of the equation for $X(z)$ takes $T_{eqX}(z, z')$ time units after the availability of $Y(z')$, our causality constraint changes to

$$\lambda_Y(z') + T_{eqX}(z, z') \leq \lambda_X(z)$$

4.1 Scheduling on a CRCW PRAM

On a CRCW PRAM, accumulation of an unbounded number of values can be carried out in a single time unit.
Causality for the reduction in (8) imposes the following scheduling constraint
\[
z_X \in \text{Dom}(X), z_R \in \mathcal{P}(z_X) : \\
\lambda_X(z_X) \geq \lambda_R(z_R) + 1 \tag{12}
\]

Determining the fastest linear schedule of (8) may therefore be obtained by formulating (12) as a parametric integer program (PIP) to be solved using a PIP solver [3]. This is identical to the approach of Redon and Feautrier [20].

### 4.2 Partial Serialization

Observe that although there is no à priori order on the accumulation of values in a reduction, there exists, even in the CRCW PRAM schedule, a certain order in their availability as implied by the schedule \(\lambda_R\) of the variable \(R\). In particular, for any \(t\), the set of points \(z_X \in \text{Dom}(X), z_R \in \mathcal{P}(z_X) : \lambda_R(z_R) = t\) defines the “slices” of values that are available for accumulation at time instant \(t\). These values lie in a hyperplane, since \(\lambda_R\) is an affine function. We call them equitemporal hyperplanes (parametrized by \(t\)). Obviously a (partial) serialization of the reduction in the lexicographically increasing order of \(t\) will cause no slowdown of schedule.

This (partial) serialization may be employed to reduce the complexity of the operator (in terms of number of values that need to be accumulated in a single time step). Note that we may still have to accumulate an unbounded number of values (all the points over any equitemporal hyperplane) in a single time step, and therefore may still be unable to accumulate with bounded fan-in operators.

We may decompose our reduction (8) as follows

\[
\text{TempX} = \text{reduce}(\oplus, (z \rightarrow (A_{\lambda_R} \frac{z}{p}) + (b_{p_{\alpha_R}})), R) \tag{13}
\]

\[
X = \text{reduce}(\oplus, (z_X, t \rightarrow z_X), \text{TempX}) \tag{14}
\]

where the schedule for \(R\): \(\lambda_R(z_R)\) is written as \(A_R \frac{z}{p} + \alpha_R\). We have introduced a new temporary variable (TempX) defined over \((z_X, t)\) where \(z_X \in \text{Dom}(X)\) and \(f(z_X) \leq t \leq l(z_X)\) where \(f(z_X)\) and \(l(z_X)\) are the first and last time instants when values in \(\mathcal{P}(z_X)\) are available. TempX\(z_X, t)\) is the reduction of values in the \(t^{th}\) equitemporal hyperplane in \(\mathcal{P}(z_X)\).

The corresponding scheduling constraints are

\[
f(z_X) \leq t \leq l(z_X), z_X \in \text{Dom}(X) \tag{15}
\]

\[
\lambda_{\text{TempX}}(z_X) \geq t + T_{\text{eqTempX}}(z_X, t) \tag{16}
\]

On the CRCW PRAM model, we impose that \(T_{\text{eqTempX}}(z_X, t) = 1\) and \(T_{\text{eqX}}(z_X, t) = 0\) and we are assured that our decomposition does not cause any slowdown.

Observe that although this decomposition is based on the knowledge of \(\lambda_X\), the schedule that we seek, this is not really a problem—we are simply using the decomposition to explain and reason about the validity of our scheduler. If we are successful in finding a \(\lambda_X\) that satisfies the constraint explained later, then the decomposition of (13-14) is the one needed to implement the schedule.

### 4.3 Scheduling under bounded fan-in

We will now account for the additional constraints imposed by a realistic machine (operators with bounded fan-in). We decree that a constant-fold slowdown (viz à viz the CRCW schedule) is acceptable, but any increase in the number of dimensions of the schedule is unacceptable. We say reductions are efficiently serialized if the resulting program permits a schedule that has the same number of dimensions as the fastest possible linear schedule (on a CRCW model). Since constant-fold slowdown is acceptable we may assume binary operators without loss of generality.

To do this, we will allow ourselves the luxury of a CRCW machine to execute the reductions of (14) but impose stronger constraints than (12) for the equation (13). This does not impose any loss of generality because we can always take this schedule, slow it down by a factor of 2 (in its innermost time dimension) and obtain enough slack (extra time) to do the accumulation of (14) on a machine with binary operators. This is because the number of values in \(\text{TempX}\) that are reduced is bounded by the number of time instants between \(f(z_X)\) and \(l(z_X)\). Hence, our goal now is to simply develop constraints that ensure that equation (13) can be scheduled with binary operators.

To do so, we simply extend the notion of size defined in 2.2 to arbitrary polytopes. Essentially, the size of a polytope is the time needed (i.e., the lower bound on the time) to accumulate, using binary operators, the values at all points in the polytope, assuming that they are initially available. The following theorem relates the size of a polytope to that of boxes.

**Theorem 2.** The size of a polytope has the same order as the size of its bounding box.

The scheduling constraint (15) requires that the number of time steps between the availability of values in an equitemporal hyperplane, and the computation of the result at

*Note that in order to implement reduction with bounded fan-in operators, every parametrized reduction domain must be a polytope.*
$z_X$ should be enough to accumulate the values of the equi-
temporal hyperplane (which is nothing but its size, up to a
constant factor).

Our scheduling constraint thus reduces to

$$z_X \in \text{Dom}(X), f(z_X) \leq t \leq l(z_X)$$
$$sz(z_X, t) \leq \lambda_{X}(z_X) - t$$ \hspace{1cm} (17)

We find it convenient to define the rhs of the above con-
straint as the slack of an equitemporal hyperplane.

$$sl(z_X, t) = \lambda_{X}(z_X) - t.$$ Note that in (17), the lhs, $sz(z_X, t)$ is
one dimensional (i.e., a multidimensional vector with a non-
zero value only in its innermost dimension). If the slack is
truly multidimensional (i.e., some element other than its in-
nermost one has a non-zero value), the constraint (17) would
be trivially satisfied.

On the other hand, if the slack is not truly multidimen-
sional, let us denote its last (non-zero) dimension by $sl'$. Our scheduling constraint now reduces to

$$z_X \in \text{Dom}(X), f(z_X) \leq t \leq l(z_X) :$$
$$sl'(z_X, t) \geq sz(z_X, t)$$ \hspace{1cm} (18)
i.e., $sl'(z_X, t)$ contains $sz(z_X, t)$

**Theorem 3.** The scheduling constraint (18) are both nec-
essary and sufficient for accumulation with binary operators.

The desired schedule can be obtained by formulating (18) as a para-
metric integer program and solved using a PIP solver [3]. A failure would occur if an efficient serialization
doesn’t exist.

**Theorem 4.** If the scheduling constraint for accumula-
tion with bounded fan-in operators is satisfied for the equi-
temporal hyperplane with the tightest slack, then all equitem-
poral hyperplanes within the same parametrized reduction
domain will satisfy the scheduling constraint and the desired
schedule will be a constant-fold slower than the schedule ob-
tained by the reduced constraint.

From theorem 4, we see that for an existential analysis (or
for a schedule accurate upto a constant factor) we may limit
the above constraint to only the equitemporal hyperplane
with the tightest slack”.

### 4.4 Generalization and Extensions

We have studied the restricted case where the domain of
the reduced variable $R$ is a single polyhedron and has a lin-
ear schedule. In general, the domain of any ALPHA vari-
able may be the union of polyhedra, and then we simply need to
impose the scheduling constraints for every such subdomain
of $R$. Note that this general scheduling analysis does not
involve any increase in variables (in the linear program-
ning problem formulated and passed to PIP), only additional con-
straints may be imposed.

Another simple extension is to consider the use of piece-
wise linear schedules (PLS), where the domain of a variable
is “cut” into “pieces” by a finite number of hyperplanes,
and we seek a separate schedule in each of the subdomains.
It is well known [16] that (i) PLS’s can often yield better per-
formance, and (ii) the standard scheduling techniques can
be extended to compute optimal PLS’s, provided the pieces are
given.

Our algorithm for efficiently scheduling/serializing reduc-
tions can easily be extended to PLS’s (provided the pieces are
given), as we shall see through our examples below. Fur-
thermore, we use a simple strategy for determining the sub-
domains of each variable based on the subexpressions in the
rhs of its defining equation. Although it is known that in
general, this does not guarantee optimality [10], it suffices
for our examples. Note that automatically determining the
pieces optimally (with or without reductions) is however, an
open problem, and beyond the scope of this paper.

## 5. Examples

We illustrate our methods on two examples. The first one
is a classic and well known problem for which an ingenious
efficient serialization [7] was known since 1979. However,
determining this automatically with a scheduling tool has
remained an open problem till now. Our second example is
a somewhat contrived example that illustrates the subtleties
of the algorithm.

### 5.1 Optimal String Parenthesization

The dynamic programming algorithm for optimal string
parenthesization seeks to recursively determine $C(i, n)$
where $C(i, j)$ is defined as $C(i, j) = W(i, j) + \min_{k < j}(C(i, k) +
C(k, j))$ for $1 \leq i < j \leq n$. This may be written as the
following normalized ALPHA system

$$C = W(i, j) + \text{reduce}(\oplus, (i, j, k), R)$$
$$R = C(i, j, k \rightarrow i, j) + C(i, j, k \rightarrow k, j)$$ \hspace{1cm} (19)

To obtain the value of $C$ at $(i, j)$, we need the values of
$C$ at $(i, k)$ and at $(k, j)$ for $k$ between $i$ and $j$. $W(i, j)$ is
available as input (at $t = 0$). If $\lambda_{C}(i, j)$ is of the form $aj + bi$,
causality imposes $a > 0, b > 0$.

Our analysis requires that reductions are over variables
that are defined over polyhedral domains and have a linear
schedule. Causality for the schedule of $R$ imposes $\lambda_{R}(i, j, k) \geq
\max(\lambda_{C}(i, k), \lambda_{C}(k, j))$. We cut the reduction domain into
pieces according to the time of availability of its arguments
as follows

1. $\lambda_{C}(i, k) \geq \lambda_{C}(k, j)$ or $k \geq \frac{b + c}{a + c}$ where $c = \frac{b}{a}$
2. $\lambda_{C}(k, j) > \lambda_{C}(i, k)$ or $k < \frac{b + c}{a + c}$

3. We assign the equality arbitrarily to one subdomain
We will carry out our analysis in each of these pieces individually. However, for our desired solution we seek to simultaneously resolve the constraints posed by them.

Let us consider the subdomain $k < \frac{a_i+a_j}{c+1}$. The parametrized reduction domain for $z_C \in \text{Dom}(C)$ is

$$1 \leq i' < j' \leq N : P(i', j') = \left\{ (i, j, k) | i = i', j = j', i' < k < j' + \frac{c' + c}{c+1} \right\}$$

and its schedule $\lambda_R(i, j, k) = \lambda_C(k, j) = aj + bk$. From this, and the causality constraints $a > 0, b < 0^*$, we get the points $(i', j', k) = \lambda_C(k, j')$ as equitemporal hyperplanes in $P(i', j')$.

Since the size of equitemporal hyperplanes is 1, the scheduling constraint (18) for a machine with binary operators is trivially satisfied. From the schedule $\lambda_R$, we get partial serialization (imposed by the availability of arguments): from $k = \frac{i'+j' - c' + c}{c+1} - 1$ to $k = i' + 1$. This is also the final serialization in this subdomain. Similarly we may perform our analysis for the other subdomain of $R$ and see that once again the scheduling constraint is trivially satisfied and the final serialization is from $k = \frac{i'+j' - c' + c}{c+1}$ to $k = j' - 1$.

The simultaneous resolution of the scheduling constraints in the two subdomains doesn’t impose any additional constraints. Since the final schedule is slower than the solution of the constraints by a factor of 2. We have $\lambda_R(i, j, k) = 2 \times (a^* - \bar{k})$ in the first subdomain and $\lambda_R(i, j, k) = 2 \times (a''^* - \bar{k})$ in the second where $a^*, b^*, a'', b'' > 0$. If they all equal 1, we have the well known solution proposed by Guibas et al.

Note that a serialization of the reduction in either increasing $k$ or in decreasing $k$ results in a 2-dimensional schedule and thus causes a dimensional slowdown.

5.2 The Mirrored Reduction

We saw above that the optimal parenthesis problem was trivially solved by our analysis. We now present another example which is significantly more complex and therefore better expresses the strength of our analysis.

Let a computation be described by the following SARE defining $X$ over an $N \times M$ rectangle $\{i, j|1 \leq i < N, 1 \leq j < M\}$

$$X_{i,j} = X_{i-1,j} + X_{i,j-1} + \sum_{i' < i, j' < j} f(X(i', j'), X(i - i', j'), X(i', j - j'))$$

Its corresponding normalized ALPHA program is given by

$$X = X.(i, j \rightarrow i - 1, j) + X.(i, j \rightarrow i, j - 1) +$$

reduce($+, (i, j, i', j' \rightarrow i, j), R$)

$$R = f(X.(i, j, i', j' \rightarrow i - i', j'), X.(i, j, i', j' \rightarrow i - i', j'), X.(i, j, i', j' \rightarrow i - i', j')$$

where $R$ is defined over the (4-dimensional) domain

$\{i, j, i', j'|1 \leq i < N, 1 \leq j < M, 1 \leq i' < i, 1 \leq j' < j\}$

\footnote{Note that, although we seem to be using the solution of the causality constraints to find equitemporal hyperplanes (and thus seem to have a two-pass strategy), such is done only to aid presentation. In an actual problem solving process, the parameters $a$ and $b$ will also be input to the constraint solver.}

If $\lambda_X(i, j) \equiv \lambda_X(i - i', j')$ is of the form $aj + bi$, causality imposes $a > 0, b > 0$. Our analysis for efficient serialization requires that the variable $R$ is defined over a polyhedral domain and has a linear schedule. Causality for the schedule of $R$ imposes that $\lambda_R(i, j, i', j') \geq \max(\lambda_X(i, j'), \lambda_X(i - i', j'), \lambda_X(i', j - j'))$. Since the RHS of the equation defining $R$ has three subexpressions, we partition its domain into three regions based on which of the three is computed at the latest time instant. These regions are given by

\begin{align*}
\lambda_X(i', j') &\geq \lambda_X(i - i', j') \quad \text{or} \quad i \leq i' < i, \frac{j}{2} \leq j' < j \\
\lambda_X(i - i', j') &> \lambda_X(i', j) \quad \text{or} \quad 1 \leq i' < \frac{i}{2}, a j' \geq b i' + \frac{a j - b i}{2}
\end{align*}

\begin{align*}
\lambda_X(i', j - j') &> \lambda_X(i', j') \quad \text{or} \quad 1 \leq j' < \frac{j}{2}, a j' < b i' + \frac{a j - b i}{2}
\end{align*}

We carry out our analysis on each of these subdomains individually and then seek a solution to the three scheduling constraints posed simultaneously. Let us consider the subdomain $\frac{i}{2} \leq i' < i, \frac{j}{2} \leq j' < j$. The parametrized reduction domain for $z_X$ in $X$ is

$$P(i, j) = \{(i, j, i', j') | i = i_1, j = j_1 \frac{i}{2} \leq i' < i_1, \frac{j}{2} \leq j' < j_1 \}$$

and its schedule is $\lambda_R(i, j, i', j') = (aj' + bi')$. 

\footnote{We have chosen the inequalities to be strict/non-strict arbitrarily (but consistently to ensure that the three regions are disjoint)
Equitemporal hyperplanes in $P(i_1, j_1)$ are given by $a_j + b_i = t$. Let $sz(t)$ be the size of this equitemporal hyperplane

$$sz(t) \leq \frac{1}{c} \times \min \left( \frac{t - b(i_1 - 1)}{a}, \frac{t - a(j_1 - 1)}{b} \right) + 1 \quad (27)$$

$$\leq \frac{1}{c} \left( \frac{1}{a + b} \times (t - b(i_1 - 1) + t - a(j_1 - 1)) \right) + 1$$

where $c$ is a constant. The slack for this equitemporal hyperplane is $(a_j + b_i - t)$. The scheduling constraint (18) requires that

$$\frac{1}{c} \left( \frac{2t - (a_j + b_i)}{a + b} \right) + 2 \leq a_j + b_i - t \quad (28)$$

which can be satisfied by a constant-fold slowdown of the schedule. If we formulate the scheduling constraint for the other two subdomains, we get similar constraints which may all be resolved simultaneously. The desired schedule will be slower than the solution of the constraints by a factor of 2.

The accumulation of the partial results from different equitemporal hyperplanes has to be carried out in decreasing order of slack, from $t = \frac{a_j + b_i}{2}$ to $t = a(j_1 - 1) + b(i_1 - 1)$ (in the region C). The (partial) serialization for regions A and B is shown in figure 4.

Note that, most direct serializations of the reduction (e.g., the permutations of increasing/decreasing order of $f$ and/or $j'$) result in a 2-dimensional schedule and thus cause a dimensional slowdown.

6. RELATED WORK

Karp Miller and Winograd [8] resolved the scheduling problem for SUREs defined over the entire positive orthant. Quinton [13] and other researchers used some of these ideas for designing systolic arrays defined over a polyhedral index space. Rao [18, 19] and Roychowdhury [21] also investigated multidimensional schedules for SUREs. Delosme and Ipsen first studied the scheduling problems for affine recurrences (they defined the term ARE), but without considering the domains over which they were defined [2]. They showed that all (one-dimensional) affine schedules belong to a cone. Rajopadhye et al. addressed the problem of scheduling a single ARE defined over a polyhedral index domain [17]. Quinton and van Dongen also obtained a somewhat tighter result [14]. Yaacoby and Cappello [25] investigated a special class of AREs. Mauras et al. extended this result to SAREs, but with 1-dimensional variable dependent schedules [12]. Rajopadhye et al. further extended this and proposed piecewise affine schedules for SAREs [16]. Feautrier [5] gave an alternative formulation using Farkas’ lemma, for determining (one-dimensional, variable dependent) affine schedules for a SARE. He further extended the method to multidimensional schedules [6]. An excellent recent book by Darte, Robert and Vivien [1] provides a detailed description of scheduling SUREs, and SAREs. Redon and Feautrier described how to schedule reductions under the CRCW model. They did not describe how this could be efficiently serialized.

7. CONCLUSIONS

An algorithm to compute optimal linear schedules for reduction on machines with bounded fan-in operators. As a corollary, our algorithm enables us to determine situations where this schedule is no more than a constant fold slower than the CRCW optimal linear schedule on the CRCW model.

We have resolved the problem of determining efficient parallel affine schedules for programs specified at a very high level—mathematical equations that allow reductions (combination of potentially unbounded number of arguments with an associative, and often commutative operator). We gave necessary and sufficient conditions under which the schedule is no more than a constant factor slowdown over the best such (multidimensional affine) schedule, namely one that assumes a CRCW PRAM model. As a corollary, we prove that our algorithm finds the fastest linear schedule.

For ease of explanation, and to clearly mark our contribution from previous work, we presented our results in two steps: first the formulation of the scheduling constraints for a CRCW PRAM model, followed by the analysis and development of the constraints necessary to ensure efficient serialization. However the final scheduling algorithm does not use this two step approach. We simply make appropriate calls to PIP (i.e., resolve a integer linear programming problem) with increased scheduling dimensions. Therefore the complexity of our scheduling technique is the same as the complexity of Feautrier’s multidimensional scheduler [6]. We are currently implementing this scheduler into our automatic parallelization system based on the Alpha language.

For certain examples, it is necessary to exploit piecewise linear schedules in order be optimal. Determining the “pieces” automatically is currently an open problem.

8. REFERENCES

The size of a box $B$ equals the $1$-norm (i.e., the Manhattan distance) of its principal diagonal.

Proof We will first show that the size of a box is contained by the manhattan distance of its principal diagonal. Then, we will show that this manhattan distance is a lower bound on the number of steps required in any linear strategy of accumulation.

Let space be $n$-dimensional given by the basis $\beta = \{\beta_1, \beta_2, \ldots, \beta_n\}$. The box $B$ may be represented in this $n$-dimensional space by $n$ sets of lower and upper bounds denoted by $l_i$ and $u_i$ respectively (along the $i^{th}$ basis vector). Our proof for the first part is constructive. We will accumulate (reduce) dimension by dimension. When we reduce a $d$-dimensional box along a certain basis vector, say $\beta_i$, we obtain a $d-1$ dimensional box of partial results. The number of (binary) steps required for this accumulation is $u_i - l_i + 1$. After $n$ such reductions (along each $\beta_i$), we obtain the result of accumulating all the values in the box $B$.

The proposed accumulation can be expressed by the following equations.

$$P_1[j_1, j_2, \ldots, j_n] = \left\{ \begin{array}{ll}
  j_i = l_i, & j_h = u_h, h < i : P_{i-1} \\
  j_i > l_i, j_h = u_h, h < i : P_{i} (j_i \rightarrow j_i - 1) + P_{i-1}
\end{array} \right.$$

where $P(\cdot)$ are the values in the box and $P_1$ through $P_n$ are successive accumulations along the basis vectors $\beta_1, \ldots, \beta_n$. The $i^{th}$ accumulation is carried out on a $n-i$ dimensional cuboid and is initialized at $l_i$ and terminated at $u_i$. The final result of the reduction will be in $P_{n}[u_1, u_2, \ldots, u_n]$. The total number of steps required is $\sum_{i=1}^{n} (u_i - l_i + 1)$ which is the perimeter of the box or the manhattan distance of its principal diagonal. Since this value contains the size, we can say that the size is always a one-dimensional value, irrespective of the dimensionality of the box or space.

We now proceed to prove that the manhattan distance of the box (denoted by $M$) is also lower bound on the number of steps needed in any linear strategy of accumulation. Let the other linear strategy of accumulation be along the basis vectors given by $\alpha = \{\alpha_i\}$ and take $O$ number of steps. Let the projection of the box to the one-dimensional space spanned by $\alpha_i$ be of length $s_i$. Accumulation along different basis vectors may be overlapped. The number of steps needed for a linear accumulation is at least $\max(s_1, s_2, \ldots, s_n)$

$$O \geq \max(s_1, \ldots, s_n) \geq \frac{s_1 + \ldots + s_n}{n} = \frac{\sum_{i=1}^{n} s_i}{n} \quad (30)$$

Note that $s_i$’s are the sides of the bounding box of $B$ in the basis $\alpha$. Since $M$ is the perimeter of $B$, we have $\sum_{i=1}^{n} s_i \geq M$. Thus,

$$O \geq \frac{M}{n} \quad (31)$$

*Note that, in our analysis, we assume that we have as many operators as needed. We only constrain that they are binary.

1This fact also indicates that the worst case slowdown of the schedule (in the case of constant fan-in operators) is by a single dimension i.e., a $k$-dimensional schedule may become at worst a $k+1$ dimensional schedule after serialization.
Finally, since \( n \), the number of dimensions of space is constant, we have proved that \( M \) is bounded by the number of steps needed in any linear strategy of accumulation. We have already proved that size is contained by \( M \), therefore we can conclude that the size\(^ 1\) of a box is the Manhattan distance or the 1-norm of its principal diagonal.

**Theorem 2.** The size of a polytope has the same order as the size of its bounding box.

**Proof** Its is obvious that the size of a polytope is bounded from above by the size of its bounding box. We will proceed to show that this is also a lower bound. Let \( \beta = (\beta_1, \beta_2, \ldots, \beta_n) \) be the basis along which the fastest accumulation of values in the polytope is carried out. Let \( S_\beta \) denote the size of the polytope and \( S_\beta \) the size of its bounding box in basis \( \beta \). Let the length of this bounding box along \( \beta_i \) be \( s_i \). Thus its size \( S_\beta = \sum s_i \). Moreover, the length of the projection of the polytope onto the one-dimensional space spanned by \( \beta_i \) is \( s_i \).

Since our only constraint is that the accumulation of values in the polytope is linear in nature, we get

\[
S_\beta \geq \max(s_1, s_2, \ldots, s_n)
\geq \frac{s_1 + s_2 + \ldots + s_n}{n} = \frac{\sum s_i}{n}
\]

(32)

Since \( n \), the number of dimensions is a constant, we have \( S_\beta \) lower bounded by \( S_\beta \). As \( S_\beta \) is trivially upper bounded by \( S_\beta \), we conclude that \( S_\beta \) and \( S_\beta \) are of the same order.

**Theorem 3.** The scheduling constraints are both necessary and sufficient for accumulation with bounded fan-in operators. The schedule obtained is the fastest linear schedule for these machines.

**Proof** From our derivation of the scheduling constraint, it is obvious that it is sufficient. We now, prove by contradiction that it is also necessary. Assume that (18) is not necessary such that there exists an efficient serialization when (18) is unsatisfied.

The parametrized reduction domain is parametrized by points \((x, p)\) in the extended domain of \( X \). We have two sub cases, based on slack

**Case I** Slack is truly multidimensional. The scheduling constraint is trivially satisfied.

**Case II** Slack is not truly multidimensional. For a parametrized reduction domain \( P(x, p) \), let \( sz_{zm}(x, p) \) be the size of the equitemporal hyperplane with the tightest slack \((sl_{zm}(x, p))\). From theorem 4, we know that the scheduling constraint (18) holds iff the slack contains the size for the equitemporal hyperplane with the tightest slack. By the decomposition theorem for polyhedra, a polyhedron can be represented by \( V + C \), where \( V \) is a polytope and \( C \) is the characteristic cone of the polyhedron. The scheduling constraints (18) can be seen (with error upto a constant fold) as the bounded-ness of size by the slack along all vectors in the characteristic cone of the extended domain of \( X \).

Let us define a predicate \( I(vect) \) which returns true if \( sz_{zm} \) increases and \( sl_{zm} \) doesn’t increase along vect. We get two sub cases.

\(^1\)Note that the size is the order of the minimum number of steps needed to perform a linear accumulation of values in a box. It may be slower by a constant fold.