Computer Science
Technical Report

Colorado State University

Simplifying Reductions

Gautam and S. Rajopadhye
[ggupta|svr]@cs.colostate.edu

September 20, 2004

Technical Report CS-04-107

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

Simplifying Reductions

Gautam and S. Rajopadhye
[ggupta|svr]@cs.colostate.edu

20th September 2004

Abstract

The polyhedral model enables very high level programming in the form of mathematical equations with reductions (associative and commutative operators applied to collections of values), and powerful static analysis techniques. In this model, the work complexity of an equation is typically a polynomial whose degree is the number of dimensions of its polyhedra. In equations with reductions, there is a unique opportunity to decrease the degree of this polynomial. Through careful reuse of partially computed values, one can often manually craft equivalent equations with lower complexity.

In this paper, we develop the foundations for complexity reduction through systematic transformations, and give necessary and sufficient conditions for their validity. Using these as constraints of a search, we also present an algorithm for optimally choosing the transformation parameters, yielding an equation with minimum complexity. We show how our algorithm can be easily extended to retain optimality while exploiting further simplification opportunities enabled by algebraic properties (distributivity, idempotence, invertible operators, etc.).

1 Introduction

Equations are often the most natural and succinct form of high level programs and algorithms in many scientific, mathematical and engineering applications. Many times, the equations use *reductions*: associative and commutative operators applied to collections of data values. For example, the product of two square matrices is specified by the equation, $C_{i,j} = \sum_{k=1}^{n} A_{i,k}B_{k,j}$; recursive convolution of a possibly infinite sequence of data samples is given by $y_i = \sum_{j=i}^{n} w_j x_{i-j}$, with appropriate boundary conditions for $i \leq n$; and the cost of optimal parenthesization of a string is given by $C_{i,j} = \min_{k=i}^{j-1} C_{i,k} + C_{k+1,j} + h(i,j,k)$, again with appropriate boundary conditions[1].

Such equations usually have a direct, not necessarily efficient, implementation as a loop program where the loop indices correspond to the index variables of the equation, and the data variables become arrays that are accessed through functions of the indices. The running time of such a program is proportional to the "volume" of the *index space* spanned by these loops, usually a polynomial whose degree is the maximum *number* of free index variables that are visible at any point in the equation. Thus, the matrix multiplication and optimal parenthesization equations

[1] In these simple examples, the "lower" (and "upper") bounds for the reductions are written with subscripts (and superscripts). They should not be viewed as implying an order, but merely as defining sets of indices. Later, when we formalize our notation, we will see how these sets are specified.

have cubic complexity, and the recursive convolution equations has quadratic complexity (or, if the number of samples is unbounded, linear complexity per sample).

However, such a direct implementation of an equation may be inefficient. Consider an equation, $Y_i = \sum_{j=1}^{i} \mathcal{F}(i,j)$ for $i = 1 \ldots n$. Since the function $\mathcal{F}$ is to be evaluated at all points in a triangular region ($i = 1 \ldots n$ and $j = 1 \ldots i$), the complexity of this is $\Theta(n^2)$. However, if there is potential reuse in the computation of $\mathcal{F}$, significant optimizations are possible. A classic, and possibly the simplest, example is a *scan* or *prefix* computation, specified by $\mathcal{F}(i,j) = X_j$. It is well known that the i-th answer, X_i can just be incrementally computed in *constant* time from the previous one, Y_{i-1}. This leads to a new, recursive equation, $Y_i = Y_{i-1} + X_i$, as always, with appropriate boundary conditions, which has $\Theta(n)$ complexity.

The keys to this optimization were the following observations: (i) a large number of values contributing to an instance of the answer variable are common to those contributing to an adjacent instance; and (ii) it is possible to impose an order among the answer instances such that the successive sets are monotonically increasing under inclusion.

Our main contribution in this paper is a systematic technique for detecting and exploiting such scan like computations in equational programs. For a particular class called *affine dependence equations* over *polyhedral domains*, we give necessary and sufficient conditions for exploiting the potential reuse of comon subexpressions. We present an algorithm that *optimally* exploits scans and produces an equivalent equation with minimal asymptotic complexity. We then adapt the algorithm so that it retains optimality in the presence of algebraic properties of distributivity, idempotency and the existence of an inverse. To the best of our knowliedge such "complexity reduction" has not been attempted previously as a systematic program transformation, let alone optimally.

The reminder of this paper is organized as follows. In the following section we motivate the problem through a number of seemingly simple examples. Next (Sections 3 and 4) we describe the necessary background: the language ALPHA that we use as a vehicle, and an essential data structure, the "thick" face lattice of polyhedral domains. Section 5 then describes the basic scan detection technique, and provides necessary and sufficient conditions for exploiting the reuse of common subexpressions. Section 6 proves the optimality of our algorithm and then shows how it can be extended without sacrificing optimality to account for algebraic properties of the operators in the program. In Section 7 present a detailed example: RNA secondary structure prediction through minimum energy folding. The original naive formulation of the algorithm as proposed by Zuker and his colleagues had $\Theta(n^4)$ complexity. An improved version with $\Theta(n^3)$ complexity was discovered by the same authors four years later. Our algorithm would enable a compiler to automatically produce the improved algorithm. Section 8 describes open problems, future work and concludes the paper. For clarity of presentation, all proofs have been moved to the Appendix.

2 Motivating Examples

The simple scan example that we used above was deliberately chosen for its simplicity. However, the general problem can be rather involved. As stated before, scan detection is a key component of our analysis. Often, algebraic properties such as distributivity, idempotency and/or the existence of an inverse operator, may come into play. To illustrate the difficulties, consider the following examples.

Ex1: **(a)** $Y_i = \sum_{j=0}^{i-1} X_{i-j}$;

(b) $Y_i = \bigoplus_{j=i}^{i+n} X_j$, for an arbitrary associative and commutative operator $\oplus$.

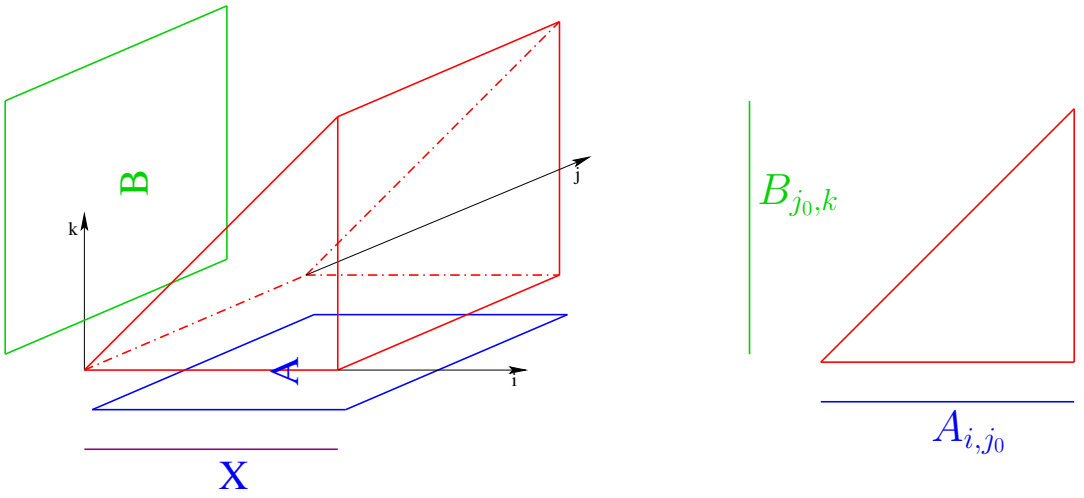

Figure 1: The geometry of the equation $X_i = \sum_{j,k=1}^{j=n;k=i} A_{i,j}B_{j,k}$. Each answer is the sum of the values in an $n \times i$ rectangle. The value at any point $[i,j,k]$ in this space is obtained by taking $A[i,j]$ and multiplying it with $B[j,k]$. In general, this value is distinct at all points, and so it seems that no sharing is possible. However, the subexpressions $A[i,j]$ and $B[j,k]$ are shared.

Ex2: $X_i = \sum_{j,k=1}^{j=n;\ k=i} A_{i,j}B_{j,k}$.

Ex3: As variants of Ex2 above, consider:

(a) $X_i = \sum_{j,k=1}^{j,k=n} A_{i,j}B_{j,k}$, where both the upper bounds are now n;

(b) $X_i = \sum_{j,k=1}^{j,k=i} A_{i,j}B_{j,k}$ where both the upper bounds are i; and

(c) $X_i = \sum_{k=1;j=k}^{k=i;j=k+N+1} A_{i,j-k+1}B_{j,k}$, where the bounds and the way A is accessed are changed.

Before proceeding further, we encourage the reader to study and simplify them as much as possible, using intuitive algebraic manipulation. We solve Ex2 here. The equation (Figure 1.a illustrates its geometry) yields a 1-dimensional result, indexed by i. The body of the summation is within the scope of three independent index variables, i, j and k, defining a 3-dimensional space, and the complexity of the equation seems to be $\Theta(n^3)$. Let AValue$[i,j,k]$ and BValue$[i,j,k]$ denote, respectively, the values of A and B that are used at any point $[i,j,k]$ in the index space. Similarly, let Body$[i,j,k]$ denote the value $A_{i,j}B_{j,k}$. Observe that AValue$[i,j,k]$ remains invariant as we move along k in the index space, and BValue$[i,j,k]$ is invariant along i. On the other hand, Body$[i,j,k]$ being the product of the two, is distinct at each index point. So it seems that no sharing is possible and the equation cannot be simplified.

However, we know that (i) the summation over j,k can be "broken up" into a double summation first over j and then over k, (ii) multiplication distributes over addition, (iii) and AValue is invariant

along k. This leads to the following algebraic simplification.

$$\begin{aligned} X_i &= \sum_{j,k=1}^{j=n;k=i} A_{i,j}B_{j,k} = \sum_{j=1}^{n}\sum_{k=1}^{i} A_{i,j}B_{j,k} \\ &= \sum_{j=1}^{n}\left(A_{i,j}\sum_{k=1}^{i} B_{j,k}\right) \end{aligned}$$

This is still $\Theta(n^3)$ because of the inner summation, a 3-dimensional subexpression, which we isolate by the following decomposition:

$$\begin{aligned} X_i &= \sum_{j=1}^{n} A_{i,j}B'_{i,j} \\ B'_{i,j} &= \sum_{k=1}^{i} B_{j,k} \end{aligned}$$

The first equation clearly has $\Theta(n^2)$ complexity. Moreover, a little bit of thought will show the second one is simply a set of n *independent* scans, one on each row of B (see Figure 1.b). It may be rewritten as the follows

$$B'[i,j] = B'[i-1,j] + B[i,j]$$

This is now $\Theta(n^2)$, thus achieving a reduction in the complexity of the original equation. The reader is encouraged to solve the variants in Ex3 above using similar reasoning (warning, Ex3.c may prove to be difficult but illuminating). This paper shows how to do such an analysis (i) *automatically* and (ii) *optimally* in the sense that a maximum reduction in the number of dimensions of the reduction is achieved whenever possible.

3 Alpha

In this section we describe ALPHA, the equational language for which we develop our analysis. There are two reasons for our choice of ALPHA. First, it is a simple language that succinctly captures exactly the abstractions needed for our analyses, while also being rich enough to express many algorithms encountered in computational science, engineering and mathematics. Second, it is well known that many programs in conventional imperative languages can be transformed through static analysis, to an intermediate form identical to an ALPHA program. The specific analyses required to achieve this are the exact data flow analysis of Feautrier [1] or equivalently, the value based dependence analysis in the Omega library [6], followed by the detection of reductions as described by Redon and Feautrier [7]. Our results may therefore be easily transported to other settings.

ALPHA is a strongly typed equational language, originally developed by Mauras [5] and later extended to include reductions by Le Verge [2]. An ALPHA program has a preamble consisting of declarations (system name, size parameters and input, output and local variables) and a body consisting of a set of equations. Variable declarations specify a domain, i.e., a set of constraints in index space where the variable is defined, and a value type (integer, real or boolean). For example, a (strictly) lower triangular, real valued matrix is specified as follows: "`A: {i,j| 0<j<i<=N} of real`." In the body there is one equation per local and output variable, each one of the form `Var = expression`.

To introduce the main features of ALPHA, Figure 2 shows a program to solve a triangular system of linear equations, $Ax = b$, where A is $n \times n$ lower triangular matrix with unit diagonal, using the

```
system ForwardSubstitution:                          -- this is a comment
    { N | N>1 }                                      -- N is a size parameter
    ( A : { i,j | 0<j<i<=N } of real;                -- A is a 2D input variable
      B : { i   | 0<i<=N } of real)                  -- B is a 1D input variable
  returns ( X : { i   | 0<i<=N } of real );          -- X is a 1D output variable
  let
      X = case                    -- a case expression, each of whose branches is
           {i | i=1} : B;         -- a restrict expression
           {i | i>1} : B - reduce(+, (i,j -> i), A * X.(i,j->j));
          esac;
  tel;
```

Figure 2: ALPHA program for forward substitution

following equation

$$\text{for } i = 1 \ldots n, x_i = \begin{cases} \text{if } i = 1 & b_j \\ \text{if } i > 1 & b_i - \sum_{j=1}^{i-1} A_{i,j} x_j. \end{cases}$$

The program is almost identical to the equation, except for syntactic sugar. The domain of `A` is triangular, while `B` and `X` are one-dimensional variables (vectors). The body is here, a single equation delineated by the `let` and `tel` keywords, and illustrates almost all the syntactic constructs for expressions. The `case` construct is used to define conditional expressions. The `restrict` construct has the syntax `<domain> : <expr>`. The `reduce` construct has three parts: an associative and commutative operator, a projection function, and an expression. Here the operator is `+`, and the projection is `(i,j->i)`, that projects the two-dimensional body expression to a one-dimensional result.

The expression, `A * X.(i,j->j)` is the body of the `reduce`. Here, `(i,j -> j)` is a *dependence function* and denotes the fact that to compute the body at `[i,j]`, we need the value of `X` at index point `[j]` (the dependence on `A` is not explicitly written—it is the identity). In ALPHA, dependences have the syntax `(idx, idx, ... -> i-expr, i-expr, ...)`, where each `idx` is an index name, and `i-expr` is an *affine* expression of the system parameters and the `idx`'s. ALPHA uses this syntax for specifying a multidimensional affine function in many different contexts (e.g., the projection function in a `reduce` expression). This syntax can be viewed as a special kind of lambda expression, restricted to affine mappings from $\mathcal{Z}^n$ to $\mathcal{Z}^m$. Such a function, f, may be equivalently represented by an $m \times n$ matrix A, and an m-vector a, i.e., $f(z) = Az + a$, and we will later use this form for analysis purposes. We will denote the function $(z \rightarrow f(z))$ as simply f.

Two aspects of the `reduce` are worth noting. First, the upper and lower bounds on j have not been specified: they are deduced automatically, from the sematic domains of the reduce body as explained later. Second, ALPHA reductions are more general than conventional mathematical notation because the projection function is not restricted to canonic projections, for example it could have been `(i,j -> i-j)`.

Semantics of Alpha Expressions

Table 1 formally describes the syntax and semantics of expressions. ALPHA is a data-parallel language, and hence expressions denote collections of values. ALPHA semantics [5] consist of two

Type of Expression	Syntax	Domain
Constant Expression	$\texttt{Expr} := \texttt{Const}$	$\mathcal{Z}^0$
Variables	$\texttt{Expr} := \texttt{Var}$	$\mathrm{DomDecl}(V)$
Pointwise Operation	$\texttt{Expr} := \texttt{Expr}_1\texttt{op}\texttt{Expr}_2 \mid \texttt{op}(\texttt{Expr}_1 \ldots \texttt{Expr}_n)$	$\bigcap_{i=1}^{n} \mathrm{Dom}(\texttt{Expr}_i)$
Case	$\texttt{case Expr}_1\texttt{; Expr}_2\texttt{; } \ldots \texttt{ Expr}_n \texttt{ esac}$	$\biguplus_{i=1}^{n} \mathrm{Dom}(\texttt{Expr}_i)$
Restriction	$\texttt{Domain : Expr}$	$\mathcal{D} \cap \mathrm{Dom}(\mathrm{Expr})$
Dependence	$\texttt{Expr.f}$ ($\texttt{f}$ is a dependence)	$\texttt{f}^{-1}(\mathrm{Dom}(\mathrm{Expr}))$
Reduce	$\texttt{reduce}(\oplus, \texttt{f}, \texttt{Expr})$ ($\texttt{f}$ is an affine function)	$\texttt{f}(\mathrm{Dom}(\texttt{Expr}))$

Table 1: Compositional rules specifying the Domains of ALPHA Expressions

parts: a semantic function and a domain. The semantic function is defined using classic methods and is fairly obvious, the only subtle point being dependences and reduce expressions, as explained below.

However, domains are a unique aspect of ALPHA. *Every* (sub) expression in a program has a domain, which can be determined from the domains of its subexpressions using the compositional rules given in Table 1. Due to the mathematical closure properties of polyhedra and affine functions [5] ALPHA domains are finite union of integral polyhedra, and may be computed automatically by a library for manipulating polyhedra [8].

Dependence expressions: First, note that the dependence $(z \rightarrow f(z))$, by itself, simply denotes the affine function, f. The dependence expression $E.(z \rightarrow f(z))$ denotes an expression whose value at z is the value of E at $f(z)$. Its domain must therefore be the set of points which are mapped by f to some point in the domain of E. This is nothing but the preimage of $\mathrm{Dom}(E)$ by f.

Reductions: $\texttt{reduce}(\oplus, (\texttt{z} \rightarrow f(\texttt{z})), \texttt{E})$ denotes an expression whose domain is the *image* of the domain of E by f. Its value at any point, $\texttt{z}$, in this domain is obtained by taking all points in the domain of E which are mapped to $\texttt{z}$ by f, and applying the associative and commutative operator $\oplus$ to the values of E at these points. We note that the image of a polyhedron by an arbitrary affine function is not in general, a polyhedron. A syntactic analysis, similar to type checking may be used to ensure that the expressions and the functions specified in a reduce are valid. We will assume that all ALPHA expressions that we encounder pass this test.

4 Properties of Polyhedral Domains

A *polyhedron*, $\mathcal{P}$ is defined as the set of points in $\mathcal{R}^n$ that satisfy a finite number of linear inequalities. In the polyhedral model, a *polyhedral domain* is the set of integer points in a polyhedron: $\mathcal{D} = \{z \in \mathcal{Z}^n | Qz \geq q, Q'z = q'\}$, where Q is an integer $m \times n$ matrix, Q' is an integer $m' \times n$ matrix, q is an m-vector and q' is an m'-vector. It is well known that polyhedra have a dual representation in terms of either a finite set of constraints or a finite set of generators (vertices, rays and lines). The lineality space of a polyhedral domain is defined as the dimensionally largest linear space along which the polyhedron extends indefinitely (i.e., the space spanned by its lines).

4.1 Effective Dimensions and Complexity

The number of dimensions of a polyhedral domain is the number of linearly independent vectors along which we can move from some iteration point in the domain and remain within the domain. It is equal to the number of indices in the declaration of the domain minus the number of linearly independent equalities in its constraint representation.

The number of *effective* dimensions of a polyhedral domain is the number of indices in its declaration minus the number of linearly independent *effective* equalities. Effective equalities are pairs of constraints of the form

$$\alpha_1 \leq a^T z \leq \alpha_2 \tag{1}$$

where α_1 and α_2 are scalar constants. Trivially, every equality $a^T z = \alpha$ is also an effective equality. To this effective equality, we associate an *effective linear subspace* given by $a^T z = 0$. We define the effective linear subspace of a polyhedral domain $\mathcal{D}$, denoted by $ELS(\mathcal{D})$, as the intersection of the effective linear subspaces of all its effective equalities. When no such pairs exist, $ELS(\mathcal{D})$ is the universe.

Theorem 1 *The number of effective dimensions of a polyhedral domain is equal to the number of dimensions of its effective linear subspace*

Theorem 2 *For any polyhedral domain $\mathcal{D}$, with k effective dimensions and $\mathcal{D}'$, the translation of $\mathcal{D}$ by a constant vector $r \in ELS(\mathcal{D})$, the domain $\mathcal{D} - \mathcal{D}'$ is a union of polyhedra, each of which*

- *can be written as $\beta \leq b^T z < \beta + b^T r, z \in \mathcal{D}$ for some the bounding constraint $b^T z \geq \beta$ of $\mathcal{D}$*
- *has $k-1$ effective dimensions*
- *has effective linear subspace equal to $ELS(\mathcal{D}) \cap \{b^T z = 0\}$ for the bounding constraint $b^T z \geq \beta$ of $\mathcal{D}$*
- *is that facet of $\mathcal{D}$ which saturates the effective equality corresponding to the above constraint.*

The *complexity* of a reduction is the number of *effective* dimensions of the expression to be reduced.

4.2 A (Thick) Face Lattice for Polyhedra

It is well known that a polyhedron can be represented by a face lattice. We propose a modification of the face lattice where faces are thick. We will represent a polyhedron by $\mathcal{C} = \{c_1, \ldots, c_m\}$, its bounding inequality constraints, and $\mathcal{C}'_e = \{c'_{m+1}, \ldots, c'_{m+m'}\}$, the set of its effective equalities. We assume that all redundant constraints have been removed. For $\mathcal{C} = \{c_1, \ldots, c_m\}$ we define $\mathcal{C}'_{ine} = \{c'_1, \ldots, c'_m\}$ as the set of corresponding effective equalities (with an arbitrary thickness). Let $\mathcal{C}' = \mathcal{C}'_e \cup \mathcal{C}'_{ine}$. Our face lattice is a lattice whose nodes are subsets of $\mathcal{C} \cup \mathcal{C}'$. Both c_i and c'_i cannot together belong to a node. The primed elements are said be be saturated in the node. We wish to emphasize that all the nodes of our face lattice represent some polyhedron given by the sublattice with this node as the root.

Each node has a level given by the number of linearly independent effective equalities that it saturates. At the top is the node that represents the polyhedron $\mathcal{C} \cup \mathcal{C}'_e$ (which may not be at level 0).

Each node in the lattice with l unsaturated elements has exactly l edges to nodes at the next lower level (we will abuse notation and call them its "children"). Every child shares the same

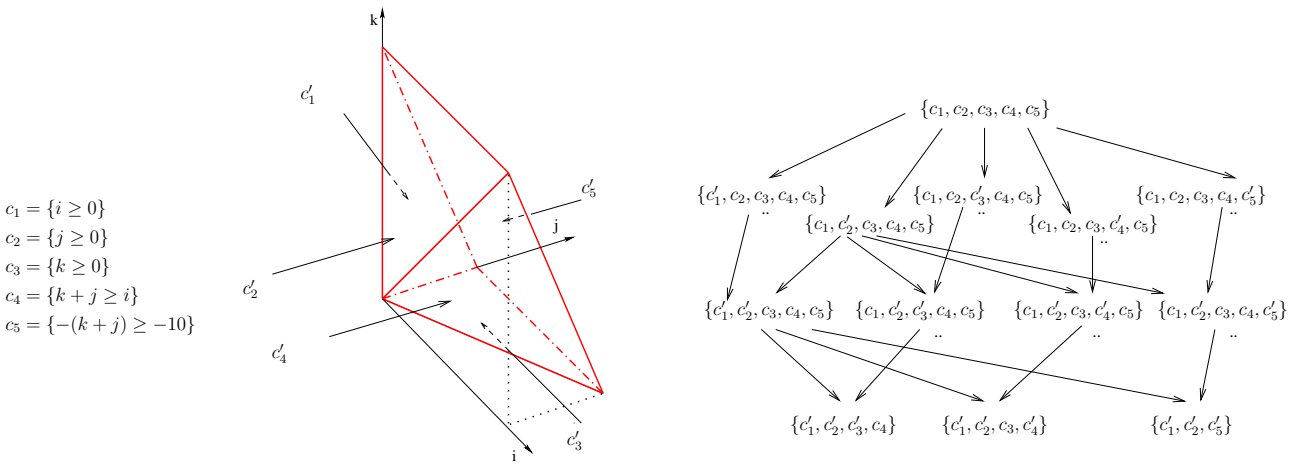

Figure 3: Face Lattice (Note that all edges are not shown to avoid clutter)

saturated elements as its parent, and in addition, saturates exactly one additional element from the set of unsaturated elements of its parent. The set of unsaturated elements of a child is a subset of the set of unsaturated elements of the parent as one of them has been saturated and some others rendered redundant[2]. Nodes that have an effective linear subspace of 0 do not have any children.

We conjecture that our face lattice can be constructed automatically from the standard face lattice of a polyhedron.

Our algorithm is based on a particular traversal of this face lattice. For an example please refer to figure 3.

5 Analysis

In this section we describe how we can achieve complexity reduction of a single equation of the form

$$X = \text{reduce}(\oplus, f_p, E) \tag{2}$$

We will focus on the case where no special algebraic properties of $\oplus$ are exploited. In particular, we suppose that it does not admit an inverse. In fact, we will see by the end of this section, that if $\oplus$ admits an inverse, then all available reuse in the expression can be trivially exploited.

5.1 Sharing

Our analysis works on the initial information that E is of the form $e.f_d$, where f_d is some dependence function. This information may be automatically derived through a variety of analyses, ranging in complexity from a simple traversal of the expression tree of E, to a sophisticated program analysis of the entire ALPHA system. In our simplification of reductions, we work on only the sharing rendered explicit by f_d. Since $E = e.f_d$, the semantics of a dependence expression imply that, for any point, z in the domain $\mathcal{D}_E$ of E, $E[z] = e[f_d(z)]$.

[2]For a precise analysis, those bounding constraints are tagged which are rendered redundant when any of the equality spaces are shifted by a constant. The children which additionally saturate the equality spaces corresponding to these constraints are at a level -2. In this paper, we will not concern with this behavior.

The null space $\mathcal{N}(f_d)$, which we call the *reuse subspace*, Rspace defines this sharing within E. This is because, for any $\rho \in$ Rspace, and any pair of points $z, z' \in \mathcal{D}_E$ such that $z' = z + \rho$, the semantics imply that $E[z+\rho] = e[f_d(z+\rho)] = e[f_d(z) + f_d(\rho)] = e[f_d(z)] = E[z]$.

For the sake of simplicity (and without loss of generality) we will assume that E is expressed in the basis $r_1, r_2, \ldots r_k, x_1, x_2, \ldots x_l$, where $r_1, \ldots r_k$ span Rspace. The *Available Reuse* of an expression is the subspace of Rspace such that exploiting reuse in that subspace achieves a reduction of the effective dimensions of the domain of the expression.

5.2 Exploiting of Shared Values

For the expression E, sharing can be exploited along $r_E = \sum c_i r_i, r \in AR(\mathcal{D}_E)$. Thus we may reuse the value $E[z_E - r_E]$ for $E[z]$ when $z_E - r_E \in \mathcal{D}_E$. If we shift E along r_E, we get precisely the computations of E that can be reused. These computations belong to the domain of the expression $E.(z_E \rightarrow z_E - r_E)$ denoted by $\mathcal{D}'_E$. As we move along r_E, we still need to perform those computations which has not been previously evaluated. These belong to the domain $\mathcal{D}_E - \mathcal{D}'_E$ denoted by $\mathcal{D}_{E_1}$. As can be seen from theorem 2, $\mathcal{D}_{E_1}$ is a union of polyhedra, each which is a facet of $\mathcal{D}_E$.

When there is a reduction of the expression E as specified by (2), the result X is defined on the domain $\mathcal{D}_X = f_p(\mathcal{D}_E)$. The value of X at an iteration point, z_X, is the combination (by $\oplus$) of those computations of E, the indices of which map to z_X by the projection function f_p. When there is sharing of values in E, a subset of the values that assign to the result at z_X may have been combined through the reduction to assign the result at another iteration point, say, $z_X - r_X$. The value $X[z_X]$ is then merely the combination of its "left-over" values with $X[z_X - r_X]$. These left-over values are those which belong to $\mathcal{D}_{E_1}$ and map to z_X by f_p. It can be seen that when r_X, the direction of propagation of partial evaluations in X, is a consequence of the sharing of values of E along r_E, then $r_X = f_p(r_E)$ or a multiple of it. Note that a dependence is introduced within the values of X along r_X. For this reason, we cannot choose $r_E \in \mathcal{N}(f_p)$, since then $r_X = 0$ and the dependence within X would be cyclic. The recurrence arising from a dependence along r_E is initialized at those computations which do not obtain any partial evaluation from another. This belong to the domain $\mathcal{D}_{XB} = f_p(\mathcal{D}_E) - f_p(\mathcal{D}'_E)$. The computations of X that need a residual reduction of left-over values of E lie in the domain $\mathcal{D}_{X_1} = f_p(\mathcal{D}_{E_1})$

We will now express mathematically, the reuse of E along r_E, for the reduction (2)

$$
\begin{aligned}
X &= \text{case} \\
&\quad \mathcal{D}_{XB} : X_1 \\
&\quad \mathcal{D}_{X_1} - \mathcal{D}_{XB} : X.(z \rightarrow z - r_X) \oplus X_1 \\
&\quad \mathcal{D}_X - \mathcal{D}_{X_1} - \mathcal{D}_{XB} : X.(z \rightarrow z - r_X) \\
&\quad \text{esac} \qquad (3) \\
X_1 &= \text{reduce}(\oplus, f_p, \mathcal{D}_{E_1} : E) \qquad (4)
\end{aligned}
$$

where these is a case expression for definition of X with a branch for the initialization of computations in X, and two for the recursion - one of which requires the evaluation of a *simpler* reduction (by theorem 2, the complexity of this reduction is an order of magnitude less than the previous).

In fact, our transformation should have been

$$
\begin{aligned}
X_1 &= \text{reduce}(\oplus, (z_X, i_r \rightarrow z_X), X'_1) \qquad (5) \\
X'_1 &= \text{reduce}(\oplus, (z \rightarrow f_p(z), r_E), \mathcal{D}_{E_1} : E)
\end{aligned}
$$

instead of (4) given above. Note that (5) is the combination of a collection of constant size for every value of the result X_1. However, this would be needed to safegaurd against any influence of the constant thickness[3] along r_E. All other properties of the two transformations are shared and thus for simplicity of the explanation, we have chosen (4).

Our equational transformation of the original reduction was under the assumption that a subset of the values that assign to the result at z_X may have already been combined for another result. Without a "containment", we would have needed to negate the contribution of those values that contributed to the result at $z_X - r_X$ and do not contribute to the result at z_X. Formally, the required negation would be the result of

$$\text{reduce}(\oplus, f_p, \{\mathcal{D}'_E - \mathcal{D}_E\} : E) \tag{6}$$

Note that this reduction is also simpler than the original by an order of magnitude. In fact, it is of the same form as (4) and thus shares all properties with it. However, its negation would require an inverse operator for the reduction operator $\oplus$, which may not always be present, e.g., there isn't any inverse for max or min. In the absence of the inverse operator, we need to verify conditions on the containment, as we move along r_E, of the set of values that combine to produce a result on the set for the next result. These conditions are both necessary and sufficient. In section 5.3, we will formalize these conditions.

However, remember that in the presence of an inverse operator these conditions are unnecessary. We will simply include (6) in our equational transformation. Since all properties for (4) are shared with (6), the associated simplification analysis will work naturally.

5.3 Containment

For the validity of the equational transformation proposed in (4) in the absence of an inverse operator, we need to ensure that there aren't any values that need to be negated from the previous result at $z_X - r_X$. Thus, there should not be a computation in $\mathcal{D}'_E - \mathcal{D}_E$ that maps to a valid iteration point in X by the projection function f_p.

Looking into the domain $\mathcal{D}'_E - \mathcal{D}_E$, from theorem 2, we know it is a union of polyhedra, each of which can be represented as

$$\beta_i \leq b_i^T z'_E < \beta_i - b_i^T r_E, z'_E \in \mathcal{D}'_E \tag{7}$$

where $b_i^T z'_E \geq \beta_i$ is a bounding constraint of $\mathcal{D}'_E$. These constraints in $\mathcal{D}'_E$ correspond to constraints $b_i^T z_E \geq \beta_i$ in $\mathcal{D}_E$ for $z_E = z'_E + r_E$. The above polyhedron is empty when $b_i^T r_E \geq 0$ in which case it cannot possibly contribute to a result of the reduction (6).

For $b_i^T r_E < 0$, the only way a negation can be avoided is when there isn't a computation in $\mathcal{D}'_E - \mathcal{D}_E$ that maps to a valid iteration point in X. Consider the boundary $b_i^T z_E = \beta_i$ of the constraint $b_i^T z_E \geq \beta_i$ for $\mathcal{D}_E$. Observe that, if there is a computation in this boundary that contributes to an "interior point"[4] in X, then there is the corresponding computation on the hyperplane $b_i^T z'_E = \beta_i = b_i^T (z_E - r_E)$ which would contribute to an iteration point in X.

This holds if all computations of the hyperplane $b_i^T z_E = \beta_i$ contribute to boundary points of X. Mathematically, this translates to

$$\mathcal{N}(f_p) \cap ELS(\mathcal{D}_E) \subseteq \{b_i^T z_E = 0\} \tag{8}$$

[3]For the same reason, we would have to perform a preprocessing of the initial reduction to remove the influence of the constant thickness of effective equalities.

[4]With interior, we mean that the iteration point z_X is such that $z_X + r_X \in \mathcal{D}_X$.

We define $\mathcal{A}$, the set of active constraints that do not satisfy (8) as the active constraints. We can summarize the condition for containment as, "For all active bounding constraints of $\mathcal{D}_E$, $b_i^T r$ should be greater than or equal to zero for a reuse direction r chosen from the available reuse space". This is embodied in the following theorem

Theorem 3 *To exploit sharing for the simplification of a reduction by an order of magnitude, in the absence of an inverse operator, the existence of a non trivial solution to*

$$b_i^T r \geq 0, \forall i \in \mathcal{A}, r \in AR(\mathcal{D}_E) \tag{9}$$

is both necessary and sufficient. Sharing may be exploited along any non trivial solution.

Note that, although in our presentation, we worked on a pre-decided r_E, our final condition is independent of any choice. In fact, from theorem 3, it can be seen that the condition is actually the basis for the selection of the reuse direction.

5.4 Recursion

The equational transformation proposed in (4), replaced the original reduction with a similar reduction but over the restricted domain $\mathcal{D}_{E_1} = \mathcal{D}_E - \mathcal{D}'_E$. From theorem 2, this is potentially a union of polyhedra. We can separate this reduction into a series of reductions, each over a polyhedral subdomain. The union of all these polyhedral subdomains equal $\mathcal{D}_{E_1}$ and their intersection is the empty domain. These domains are very similar to the polyhedral subdomains in theorem 2. The only difference is that these are disjoint. This can be easily incorporated with the following modification of the previous definition

$$\beta_i \leq b_i^T z < \beta_i + b_i^T r, z \in \mathcal{D}_i \tag{10}$$

where $\mathcal{D}_i = \mathcal{D}_{i-1} \cap \{b_{i-1}^T z \geq \beta_{i-1} + b_{i-1}^T r\}$ and $\mathcal{D}_0 = \mathcal{D}_E$. Note that such a polyhedral subdomain also corresponds to that facet of $\mathcal{D}$ which saturates the effective equality corresponding to $b_i^T z \geq \beta_i$. In fact, it is precisely that facet, with the bounding constraints $b_j^T z \geq \beta_j$ shifted by constant vector $b_j^T r$ for all $j < i$, with $\beta_i \leq b_i^T z < \beta_i + b_i^T r$ instantiated as the effective equality.

The polyhedral subdomain in (10) is non empty only when $b_i^T r > 0$. We will now demonstrate that the simpler reductions over these polyhedral subdomains are amenable to further reuse. The effective linear subspace of these is $ELS(\mathcal{D}_E) \cap \{b_i^T z = 0\}$. Their available reuse is therefore

$$ELS(\mathcal{D}_E) \cap \{b_i^T z = 0\} \cap \text{Rspace}$$

which is of one less dimension than $AR(\mathcal{D}_E)$ since $b_i^T r > 0$. Therefore, the maximum reduction of complexity of the initial reduction is the number of dimensions of available reuse of its domain. The above result shows that it is possible to exploit all available sharing, i.e., that exploiting a set of reuse vectors in Rspace does not preclude the use of any other linearly independent reuse vectors.

6 Optimality

We have seen that after reuse along one vector, further exploitation of sharing is possible on the residual reduction equation. We have also observed that the condition to be satisfied for a valid reuse is independent of any predecided choice. In this section, we will develop an algorithm to determine the reuse vector for the initial reduction and for all residual reductions at different recursion depths

to guarantee overall optimality in the complexity of the resulting equation(s). We will achieve this through a dynamic programming formulation.

Before we proceed, we consider the special case of a reduction, $\texttt{reduce}(\oplus, \texttt{f}_\texttt{p}, \texttt{E})$, where Dom(E) has a lineality space and the solution to the validity constraint embeds a non-trivial lineality (sub) space $\mathcal{L}$. We may project[5] the polyhedron along $\mathcal{L}$ to reduce the complexity of the reduction by the number of dimensions of $\mathcal{L}$. Values of X would be identical and thus need a broadcast along $f_p(\mathcal{L})$ of the values of the result of the modified reduction. Henceforth, we assume that if necessary we perform a preprocessing step and futher analysis is to be performed on the modified reduction. We would generate our face lattice only after this step.

Observe that for every face f in our face lattice, say representing the polyhedral domain $\mathcal{D}_f$, we can determine $ELS(\mathcal{D}_f)$ independently, since it only depends on the effective equalities that it saturates. Therefore, $AR(\mathcal{D}_f)$ can be determined independently as well. Now, we can claim that for every face in our face lattice, the validity constraints for containment are independent of the particular reuse that has been exploited at an ancestor face. This follows directly from their formulation:

Bounding constraints $b_i^T z_E \geq 0$ are active if they do not satisfy $\mathcal{N}(f_p) \cap ELS(\mathcal{D}_f) \subseteq \{b_i^T z_E = 0\}$ and the validity condition for a reuse vector r is $b_i^T r \geq 0, \forall i \in \mathcal{A}, r \in AR(\mathcal{D}_f)$ where $\mathcal{A}$ is the set of active bounding constraints. Both of the above conditions do not rely on any information obtained from an ancestor of a face in the face lattice and thus can be determined independently for every face.

If the validity constraint admits a non-trivial solution, the problem of evaluating a reduction of the expression at face f with the domain $\mathcal{D}_f$ can be transformed into a series of reductions, each of which reduces the expression on the domain represented by some of the children of face f. These children are those for whom $b_i^T r > 0$, those which are not saturated by the reuse vector. Thus, they are chosen through a choice of the reuse vector r. Other children will said to be masked out by r. The complexity of the reduction at face f is the maximum of the complexity of the reduction for its unmasked children, and the number of effective dimensions of the result (as that many values always need to be computed. It is the complexity of the supporting equation (3)). Given the independence of the validity analysis for all nodes, and the dependence of complexity of a parent on its children, we will traverse our face lattice bottom-up, assigning the complexity to each face as we proceed.

At a face, if sharing may be exploited, there might possibly exist an infinite number of reuse vectors. However, we can prove that only a finite number of these need to be considered for optimality. We will state through the following theorem

Theorem 4 *If the solution to the validity constraints $\mathcal{K} = b_i^T r \geq 0, \forall i \in \mathcal{A}, r \in AR(\mathcal{D}_f)$ for the polyhedral domain $\mathcal{D}_f$ is a pointed cone, one of the extremal rays of $\mathcal{K}$ is the reuse direction that gives the optimal complexity to the reduction of the expression on $\mathcal{D}_f$. If it has a non trivial lineality space, any ray in the space that satisfies at least one bounding constraint $\notin \mathcal{A}$ gives the optimal complexity to the reduction.*

Since we have restricted the our search space for the reuse vector r to a finite number of choices, we can always choose the optimal direction of reuse for every face as we proceed up our face lattice.

[5] We assume here that $\mathcal{N}(f_p) \cap \mathcal{L} = \phi$ since that would mean the accumulation of an infinite number of identical values for a result. Although this assumption seems absolutely reasonable, we can modify our model to handle that case as well. We choose not to present it here for clarity.

6.1 Size Calculations

In the previous section, we constructed an algorithm to choose reuse vectors optimally for every facet. We wish to remind there was another restriction that needed to be imposed on the finite set of choices. This was $r \notin \mathcal{N}(f_p)$, that reuse cannot be chosen in the null space of the projection which would introduce a cyclic dependence within X. Although this constraint doesn't affect any previously established result, we wish to exploit such reuse as well, if it allows a greater reduction in complexity.

When we have sharing of values within the set that contributes to the same computation in X, then we may employ a higher operator to "multiply" that value with the number of occurrences. The number of occurrences are given by the Ehrhardt polynomial [4] of the parameterized polyhedron (parameterized by the indices of the result and the basis completion of $\mathcal{N}(f_p) \cap \mathcal{N}(f_d)$ to $\mathcal{N}(f_p)$) for the collection of values that are identical and contribute to each computation in X. In some cases this higher order operator may be provided eg., multiplication for addition, exponentiation for multiplication[6]. The complexity of implementing this higher operator may be the same as the reduce operator, in which case we have improved the complexity of the reduction by the number of dimensions of sharing within the null space of f_p. In any case the complexity of this operator cannot be more than a logarithm of the Ehrhardt number, since even in the case when a higher operator is not provided, we may define one which can accumulate n identical values in a logarithmic number of steps.

As a side effect of this exploitation we have changed the expression inside the reduction, by "multiplying" it with the Ehrhardt polynomial. In general, we can assume that this polynomial has no shared values. Thus, this is a terminal step and does not permit further reuse. In our algorithm for optimal choice, we just need to introduce a choice, that which follows such exploitation.

6.2 Factorization of the Projection

For some problems, it may be beneficial to transform the reduction into a reduction of reductions by factorizing f_p into $f''_p.f'_p$ as

$$\text{reduce}(\oplus, (z' \rightarrow f''_p(z')), \text{reduce}(\oplus, (z \rightarrow f'_p(z)), E))$$

where z' are the indices of the inner reduction. This transformation can be seen as the combination of values for subsets and later combining their partial results to complete one computation of the original reduction. For complexity, we just need to concentrate on the inner reduction. Note that the following property holds

$$\mathcal{N}(f'_p) \subset \mathcal{N}(f_p)$$

The projection function has two effects on reuse. It removes some constraints from the set of active constraints $\mathcal{A}$ and thus expanding the scope of solutions to the validity condition. The other effects is on the effective dimensionality of the results which is a lower bound on the complexity.

There are infinitely many factorizations possible. We will use the above stated effects to restrict the choice to a finite set. Since, the only advantageous influence, of the projection function, is through expanding the scope of the validity condition, we just need to consider projection functions that make a subset of previously active constraints inactive. Thus the set of choices of projection functions to consider is simply the intersection of the original projection function with the finitely-many elements of the powerset of $\mathcal{A}$.

[6] max is idempotent and we may simply project out the dimensions.

Consider the interaction between the face lattices corresponding to the different projections. For face in these lattices, we may factorize the projection function such that it makes some more active bounding constraints inactive. However, we cannot remove inactive bounding constraints and make them active in any face at a lower level. Thus, in our algorithm, the optimal choice now includes the ability to choose among factorizations of the current projection at every face.

6.3 Influence of Algebraic Properties

We may employ algebraic properties of operators to change the expression that is reduced to one of its subexpressions. The advantages of such a transformation can be seen in the motivating examples given in section 2. In particular, we look at two properties

Same Operator Simplification If we have a reduction of the form reduce($\oplus, f_p, exp_1 \oplus exp_2$), then we may transform it to reduce($\oplus, f_p, exp_1$) $\oplus$ reduce($\oplus, f_p, exp_2$). Since we have simply separated the original reduction into two parallel reductions, possibly exposing a greater available reuse in both, we can always perform this step before we begin our analysis.

Distributivity If $\otimes$ distributes over $\oplus$ and exp_1 has available reuse such that

$$\mathcal{N}(f_p) \cap ELS(\mathcal{D}) \subseteq AR(\mathcal{D}_{exp_1})$$

a reduce expression of the form reduce($\oplus, f_p, exp_1 \otimes exp_2$) can be transformed to $exp_1 \otimes$reduce($\oplus, f_p, exp_2$). The transformation, here, may require a change of the projection function but may possibly expose greater sharing because it is the reduction of a subexpression. This case is very similar to the case discussed in section 6.2 where we need to only consider a finite number of decompositions possible as a result of projection factoring. Here, we would have to factorize the projection such that

$$\mathcal{N}(f'_p) \subset \mathcal{N}(f_p) \cap AR(\mathcal{D}_{exp_1})$$

Since exploitation of distributivity only results in finitely many possibilities, we get a set of lattices corresponding to the required projections. Again, in our algorithm, the optimality choice will now include the ability to choose among possible applications of the distributivity property of operators to reduce the complexity of the reduction at every face in the face lattice.

7 Real-World Application: RNA Secondary Structure Prediction

Now, we will show the applicability of our algorithm to a real world problem. RNA secondary structure prediction is a highly compute intensive computation. Extensive research in the bioinformatics community has been directed at the improvement of the running time of these algorithms. Here, we take a celebrated result [3] and show that it is an instance of the hand optimization of a specific problem that our algorithm would optimize automatically.

Instead of showing all the different face lattices of the domain corresponding to different projection functions and the optimal selection of the choice at every face, we simply show the choices our algorithm would make and the associated transformations.

RNA secondary structure prediction (of a specific kind) is expressed as

$$VBI \quad = \quad \text{reduce}(\min, f_p,$$

$$
\begin{aligned}
& sz.(i,j,i',j' \rightarrow i'-i+j-j'-2) + \\
& lop.(i,j,i',j' \rightarrow |i'-i-j+j'|) + \\
& sz'.(i,j,i',j' \rightarrow i'-i+j-j'-2) + \\
& st.(i,j,i',j' \rightarrow i',j') + \\
& st.(i,j,i',j' \rightarrow i,j) + \\
& V.(i,j,i',j' \rightarrow i',j'))
\end{aligned} \tag{11}
$$

where f_p is $(i,j,i',j' \rightarrow i,j)$ and $i < i' < j' < j$. We note that $(i,j,i',j' \rightarrow |i'-i-j+j'|)$ can be divided into two cases, one when $i'-i-j+j' \geq 0$ and the other $i'-i-j+j' < 0$. Let us assume this separation has been made in the preprocessing and in this demonstration we will work on the case when $i'-i-j+j' \geq 0$. The other case will be identical since the sharing of the subexpression is the same.

We note that, $st.(i,j,i',j' \rightarrow i,j)$ is such that $\mathcal{N}(f_p) \cap ELS(\mathcal{D}) \subseteq (AR_{st1})$ and $+$ distributes over min. Thus, our analysis may distribute $st.(i,j,i',j' \rightarrow i,j)$ out of the reduction.

$$
\begin{aligned}
VBI \;=\; & st + \text{reduce}(\min, f_p, \\
& sz.(i,j,i',j' \rightarrow i'-i+j-j'-2) + \\
& lop(i,j,i',j' \rightarrow i'-i-j+j') + \\
& sz'.(i,j,i',j' \rightarrow i'-i+j-j'-2) + \\
& st.(i,j,i',j' \rightarrow i',j') + \\
& V.(i,j,i',j' \rightarrow i',j'))
\end{aligned}
$$

Note that sz and sz' have the same dependency and thus will be amenable to sharing along the same space. For simplicity, we may replace these two subexpressions by $sz''.(i,j,i',j' \rightarrow i'-i+j-j'-2)$. Similarly, expressions of st and V can be replaced with an expression that has the same dependence on a local variable g, which is the pointwise addition of $st$and V.

$$
\begin{aligned}
VBI \;=\; & st + \text{reduce}(\min, f_p, \\
& sz''.(i,j,i',j' \rightarrow i'-i+j-j'-2) + \\
& lop(i,j,i',j' \rightarrow i'-i-j+j') + \\
& g.(i,j,i',j' \rightarrow i',j'))
\end{aligned}
$$

Let us now look at the sharing of the three subexpressions. sz'' is shared along the 3-dimensional space spanned by $(1,0,1,0)$, $(0,1,0,1)$ and $(1,1,0,0)$, lop shared along the 3- dimensional space spanned by $(1,0,1,0)$, $(0,1,0,1)$ and $(1,-1,0,0)$, and g shared along the 2- dimensional space spanned by $(1,0,0,0)$, $(0,1,0,0)$. The sharing of the entire expression is ϕ since the intersection of sharing of its subexpressions is trivial.

Our analysis would consider the use of distributivity again to change the expression inside the reduction and thus potentially expose more sharing. Any of the subexpressions may be distributed out. However, distributing g out would involve changing the projection function to $i,j,i',j' \rightarrow i,j,i',j'$ since $\mathcal{N}(f_p) \cap AR(\mathcal{D}_g) = \phi$. The distribution of either of the other two expressions out of the reduction achieves similar results. Let us look into distributing sz'' out.

The projection function will have to be factorized such that the following property holds

$$\mathcal{N}(f_{p_1}) = \mathcal{N}(f_p) \cap AR(D_{sz''})$$

where f_{p_1} is the projection of the inner reduction. $\mathcal{N}(f_{p_1}) = \{(0,0,1,1)\}$ and for this projection the sharing of the expression inside this reduce is the intersection of the sharing of *lop* and g which is the space spanned by $r_E = (1,-1,0,0)$.

Now that one dimension of sharing has been exposed for the expression inside the reduce, all that needs to be verified is whether containment is satisfied, and this sharing can be exploited. Considering the set of bounding constraints of the domain $\{i,j,i',j'|i < i' < j' < j, i'-i > j-j'\}$ which becomes $\{(-1,0,1,0),(0,0,-1,1),(0,1,0,-1),(-1,-1,1,1)\}$. The bounding constraint $(0,0,-1,1)$ is inactive as satisfies (8). For all the constraints, $(-1,0,1,0)$,$(0,1,0,-1)$ and $(-1,-1,1,1)$, the direction along $-r_E$ satisfies the validity constraints. In addition, it also saturates the boundary $(-1,-1,1,1)$. Thus, we may succesfully exploit the sharing and the dependence with the results is $r_X = f_{p_1}(-r_E) = (-1,1,0)$.

After factorization and distribution, the transformated equations are,

$$\begin{aligned} VBI &= st + \text{reduce}(\min, (i,j,k \rightarrow i,j), \\ &\quad sz''.(i,j,i',j' \rightarrow i'-i+j-j'-2) + X) \\ X &= \text{reduce}(\oplus, (i,j,i',j' \rightarrow i,j,i'-j'), \\ &\quad lop(i,j,i',j' \rightarrow i'-i-j+j') + \\ &\quad g.(i,j,i',j' \rightarrow i',j')) \end{aligned}$$

In the next step, when sharing is exploited, these equations are transformed to

$$\begin{aligned} VBI &= st + \text{reduce}(\min, (i,j,k \rightarrow i,j), \\ &\quad sz''.(i,j,i',j' \rightarrow i'-i+j-j'-2) + X) \\ X &= \text{case} \\ &\quad \{i,j,k|i = j-3\} : X' \\ &\quad \{i,j,k|i < j-3\} : X.(i,j,k \rightarrow i+1, j-1, k) + X' \\ &\quad \text{esac} \\ X' &= \text{reduce}(\oplus, (i,j,i',j' \rightarrow i,j,i'-j'), \mathcal{D}_{X'} : \\ &\quad lop(i,j,i',j' \rightarrow i'-i-j+j') + \\ &\quad g.(i,j,i',j' \rightarrow i',j'))) \end{aligned}$$

where $\mathcal{D}_X = \{i,j,i',j'|1 \leq i'-i \leq 3\}|\{i,j,i',j'|1 \leq j-j' \leq 3\}$, the "left-over" domains. Our final reduction has a polynomial complexity of order 3. The original program had a complexity of order 4.

8 Conclusions and Future Work

In paper, we have presented the optimal simplification of a single reduce equation provided that the reuse in the body of a reduction was specified. Detection of this reuse requires a preprocessing as indicated in section 5.1. The choice of exploitation of sharing within the expression E in one reduction might influence other reductions in a program and therefore we need to extend our analysis to a complete program rather than a single equation.

Another aspect that we have not covered in this paper is the impact of exploitation of sharing on other program analyses like scheduling and parallelization.

We have encountered some examples where a judicious *cutting* or index set splitting can enable complexity reductions that could not be simplified by our algorithm. It is an open question how

such "pre-processing" to our algorithm can be done automatically and optimally. We conjecture that it is undecidable in the most general case.

Our most immediate efforts will be on implementing these methods into the MMALPHA framework. We are also looking for application domains where the benefits of this analysis can have an impact.

References

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

[2] 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.

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

[4] I. G. Macdonald. The volume of a lattice polyhedron. *Proc. Camb. Phil. Soc.*, 59:719–726, 1963.

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

[6] W. Pugh. A practical algorithm for exact array dependence analysis. *Commun. ACM*, 35(8):102–114, 1992.

[7] Xavier Redon and Paul Feautrier. Detection of recurrences in sequential programs with loops. In *Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe*, pages 132–145. Springer-Verlag, 1993.

[8] D. Wilde. A library for doing polyhedral operations. Technical Report PI 785, IRISA, Rennes, France, Dec 1993. an extended version of the author's MS Thesis, Computer Science Dept, Oregon State University, Corvallis, OR. Dec 1993.

A Proofs

Theorem 1 *The number of effective dimensions of a polyhedral domain $\mathcal{D}$ is equal to the number of dimensions of its effective linear subspace $ELS(\mathcal{D})$.*

Proof The number of effective dimensions of a polyhedral domain is defined as the number of indices in its declaration minus the number of linearly independent effective equalities. The effective linear subspaces associated to these effective equalities is therefore also linearly independent. $ELS(\mathcal{D})$ is the intersection of the effective linear subspaces of all effective equalities. It is therefore the number of dimensions of space minus the number of linearly independent effective linear subspaces. The number of dimensions of space in which a domain is expressed is the number of indices in its declaration. Thus, $ELS(\mathcal{D})$ is precisely the effective dimensions of a polyhedral domain.

When there are no effective equalities, the effective dimensions of a polyhedral domain is the number of indices in its declaration which is the number of dimensions of space and therefore equal to its unconstrained $ELS(\mathcal{D})$. ∎

Theorem 2 *For any polyhedral domain $\mathcal{D}$, with k effective dimensions and $\mathcal{D}'$, the translation of $\mathcal{D}$ by a constant vector $r \in ELS(\mathcal{D})$, the domain $\mathcal{D} - \mathcal{D}'$ is a union of polyhedra, each of which*

- *can be written as $\beta \leq b^T z < \beta + b^T r, z \in \mathcal{D}$ for some bounding constraint $b^T z \geq \beta$ of $\mathcal{D}$*
- *has $k-1$ effective dimensions*
- *has effective linear subspace equal to $ELS(\mathcal{D}) \cap \{b^T z = 0\}$ for the bounding constraint $b^T z \geq \beta$ of $\mathcal{D}$*
- *is that facet of $\mathcal{D}$ which saturates the effective equality corresponding to the above constraint.*

Proof Let all the constraints of the domain $\mathcal{D}$ be expressed as

$$b_i^T z \geq \beta_i \tag{12}$$

For effective equalities of $\mathcal{D}$, r belongs to their effective linear subspace since $r \in ELS(\mathcal{D})$. Thus, for the constraints that pair in effective equalities $b_i^T r = 0$.

All corresponding constraints in $\mathcal{D}'$ can be written as $b_i^T (z - r) \geq \beta_i$ or

$$b_i^T z \geq \beta_i + b_i^T r \tag{13}$$

By definition all iteration points in $\mathcal{D} - \mathcal{D}'$ belong to the domain $\mathcal{D}$ and do not belong to $\mathcal{D}'$. Thus, all iteration points in $\mathcal{D} - \mathcal{D}'$ are those that belong to $\mathcal{D}$ and do not satisfy atleast one constraint of $\mathcal{D}'$. We will iterate over all the constraints in $\mathcal{D}'$. For some i the constraint (13) in $\mathcal{D}'$, corresponds to the constraint (12) in $\mathcal{D}$. The polyhedral domain that corresponds to the violation of this constraint can be written as

$$\beta_i \leq b_i^T z < \beta_i + b_i^T r, z \in \mathcal{D} \tag{14}$$

where the first inequality is satisfied by all points in $\mathcal{D}$. The above domain is non empty only when $b_i^T r > 0$. Remember that for contraints that pair in effective equalities, the dot product $b_i^T r$ equals zero. Therefore all constraints that give a non empty domain in (14) have to be bounding constraints. Also note that the above is the intersection of the original domain with an effective equality,

$$\beta_i \leq b_i^T z < \beta_i + b_i^T r \tag{15}$$

For non-empty domains, we can prove that effective equality is linearly independent to all previous effective equalities of $\mathcal{D}$ since $b_i^T r > 0$. Thus this polyhedral domain has $k-1$ effective dimensions.

The effective linear subspace corresponding to this effective equality is $b_i^T z = 0$. The effective linear subspace of this polyhedral domain is the intersection of the effective linear subspaces of all its effective equalities which is precisely

$$ELS(\mathcal{D}) \cap \{b_i^T z = 0\}$$

Finally, note that this polyhedral domain is that facet of $\mathcal{D}$ which additionally saturates the effective equality corresponding to $b_i^T z \geq \beta_i$

We have already seen that $\mathcal{D}'$ is the union of all these polyhedra. ∎

Theorem 3 *To exploit sharing for the simplification of a reduction by an order of magnitude, in the absence of an inverse operator, the existence of a non trivial solution to*

$$b_i^T r \geq 0, \forall i \in \mathcal{A}, r \in AR(\mathcal{D}_E)$$

is both necessary and sufficient. Sharing may be exploited along any non trivial solution.

Proof The proof to this follows from the construction of the condition. We will give it in the final version of the paper ∎

Theorem 4 *If the solution to the validity constraints $\mathcal{K} = b_i^T r \geq 0, \forall i \in \mathcal{A}, r \in AR(\mathcal{D}_f)$ for the polyhedral domain $\mathcal{D}_f$ is a pointed cone, one of the extremal rays of $\mathcal{K}$ is the reuse direction that gives the optimal complexity to the reduction of the expression on $\mathcal{D}_f$. If it has a non trivial lineality space, any ray in the space that satisfies at least one bounding constraint $\notin \mathcal{A}$ gives the optimal complexity to the reduction.*

Proof We will just present our intuition here. The complete proof will be given in the final paper.

Let us first consider the case when $\mathcal{K}$ is a pointed cone. If reuse can be exploited at a face, its complexity is the maximum complexity of its unmasked children, with the number of effective dimensions of the result. Therefore, if we have two masks, one which is a subset of the other, then the former is cannot result in a lesser complexity for the face than the latter. We can prove that all non-extremal rays, saturate a lesser number of bounding constraints than some extremal rays and thus result in extra facets (corresponding to the extra unsaturated bounding constraints).

When $\mathcal{K}$ is not a pointed cone, it is because some bounding constraints have been removed as they contribute to boundary values of the result. Considering all constraints, irrespective of whether they are in $\mathcal{A}$, will always result in a pointed cone, since as a pre-processing, we have projected the initial polyhedral domain along its lineality space. Any face of this polyhedral domain cannot have a lineality space since the redundant bounding constraints of a child are never the linearly independent with the other bounding constraints. Thus, when we have a non trivial lineality space of $\mathcal{K}$, we simply need to perform the reduction at one (or more) of its non-active constraints, those that do not belong to $\mathcal{A}$ and broadcast values in the result of this reduction. So, any ray that satisfies at least one bounding constraint $\notin \mathcal{A}$ gives the optimal complexity to the reduction.

Thus, one of the extremal rays of $\mathcal{K}$ is the reuse direction that gives the optimal complexity. ∎