On Simplifying Dependent Polyhedral Reductions
Sanjay Rajopadhye, Colorado State University

Abstract
Reductions combine collections of input values with an associative (and usually also commutative) operator to produce either a single, or a collection of outputs. They are ubiquitous in computing, especially with big data and deep learning. When the same input value contributes to multiple output values, there is a tremendous opportunity for reducing (pun intended) the computational effort. This is called simplification. Polyhedral reductions are reductions where the input and output data collections are (dense) multidimensional arrays (i.e., tensors), accessed with linear/affine functions of the indices.

Gautam and Rajopadhye [6] showed how polyhedral reductions could be simplified automatically (through compile time analysis) and optimally (the resulting program had minimum asymptotic complexity). Yang, Atkinson and Carbin [16] extended this to the case when (some) input values depend on (some) outputs. Specifically, they showed how the optimal simplification problem could be formulated as a bilinear programming problem, and for the case when the reduction operator admits an inverse, they gave a heuristic solution that retained optimality.

In this note, we show that simplification of dependent reductions can be formulated as a simple extension of the Gautam-Rajopadhye backtracking search algorithm.

Keywords: Polyhedral model, Reduction, Scheduling

1 Background
This note uses the notation from the previous work [6, 16], that we encourage the reader to review. We briefly summarize and clarify the core ideas in these papers. Consider the following two equations.

\[ P[i] = \begin{cases} 
  i = 0 : & Q[i] \\
  i > 0 : & \sum_{j=i}^{2i-1} Q[j] 
\end{cases} \]  
\( (1) \)

\[ X[i] = \begin{cases} 
  i = 0 : & f(i) \\
  i > 0 : & f \left( \sum_{j=0}^{i-1} X[j] \right) 
\end{cases} \]  
\( (2) \)

These equations specify that each element of a one dimensional array (respectively, \( P \) and \( X \)) is obtained by reducing (the operator is addition) a subset of values of an input array (respectively, \( Q \) and \( X \)). The arrays have size \( N \), viewed as a parameter of the program/equation, and we seek to optimize the asymptotic execution time of our program as a function of \( N \). The first equation defines an independent reduction (\( P \) is output and \( Q \) is input), while the second equation is a dependent reduction: \( X \) is both input and output and appears on both the left and right hand side.
Since the $i$-th element of the output involves the reduction of $i$ values, the nominal complexity of each equation is $O(N^2)$. We can recognize that these summations are similar to prefix-sums: all the values (except the last/first ones) contributing to the $i$-th output, also contribute to the $(i-1)$-th output. Simplification exploits this fact to compute each result with a single operation (in $O(1)$ time), thereby reducing the asymptotic complexity to $O(N)$ as shown in the equations below.

\[
\begin{align*}
P[i] &= \begin{cases} 
  i = 0 & : Q[i] \\
  i > 0 & : A[i-1] + B[2i] - B[i]
\end{cases} \\
X[i] &= \begin{cases} 
  i = 0 & : f(i) \\
  i > 0 & : f(X[i-1])
\end{cases}
\end{align*}
\]

**Polyhedra:** A polyhedron (polytope) $\mathcal{D}$ is the intersection of a number of half-spaces or inequalities of the form $cz + \gamma \geq 0$ called constraints. Some constraints may either be equalities (i.e., the intersection of both $cz + \gamma \geq 0$ and $cz + \gamma \leq 0$), or “thick equalities,” (the intersection of $cz + \gamma \geq 0$ and $cz + \gamma' \leq 0$, for $\gamma' \neq \gamma$). When $\mathcal{D}$ has such thick equalities, we say that it saturates the constraint $\langle c, \gamma \rangle$, or simply saturates $c$. Along any vector, $\rho$ such that $c\rho \neq 0$, $\mathcal{D}$ has only a bounded number of points. The intersection of all such saturating constraints is denoted by $\mathcal{L}(\mathcal{D})$, and is the smallest linear subspace that contains $\mathcal{D}$.

A **facet**, $\mathcal{F}$ of a polyhedron $\mathcal{D}$ is its intersection with the equality $cz + \alpha = 0$ associated with exactly one constraint. We say that the facet saturates the constraint $\langle c, \alpha \rangle$. More than one constraint may be saturated, and this yields faces. The concept of “thickness” can be extended to faces too, and Gautam and Rajopadhye [6] define the **thick face lattice** of $\mathcal{D}$ which is a critical data structure during simplification. Zero-dimensional faces are called **vertices**, 1-dimensional faces are called **edges**, and $\mathcal{D}$ itself is the topmost face (it’s children are the facets). Faces are arranged level by level, and each face saturates exactly one constraint in addition to those saturated by its “parent.”

**Parameters, volume and complexity:** A polyhedron may have one or more designated indices, like $N$ above, called its **size parameter(s)**. There is no upper bound on parameters, and they allow us to define an unbounded family of polytopes, one for each value of the parameter. The **volume**, cardinality, or the number of integer points, in such a parametric polytope is known to be a polynomial function of the parameter, and this polynomial is the asymptotic complexity of a program that performs a contant time operation at every point in the polyhedron. The degree of this polynomial, also called the number of (free) dimensions of the polyhedron, is the number of indices in $\mathcal{D}$, less the number of linearly independent thick equalities.

**Equations, reductions, and their reuse and share space:** For the scope of this note, we seek to simplify equations of the form

\[
\forall z \in \mathcal{D} : Y[Bz] = \bigoplus_{z} e(z) = \bigoplus_{z} X[Az]
\]

Here, $e$ is some expression, called the reduction body, and there is no loss of generality in assuming that it simply reads an input array $X$. $A$ and $B$ are linear access functions matching the number of dimensions of $\mathcal{D}$, $X$ and $Y$, as appropriate. Reductions combine multiple values to produce multiple answers, and this is accomplished by a many-to-one (i.e., rank-deficient) linear **write access**, $B$, called the **projection** of the reduction. We say that the value of $e(z)$ **contributes** to the answer at $Y[Bz]$. The image of $\mathcal{D}$ by $B$, is the set of results produced by the reduction, and is the **domain** of $Y$, denoted by $\mathcal{D}_Y = B(\mathcal{D})$.

Simplification is possible only if the **same** input value is read at multiple points in $\mathcal{D}$. It is well known [5,14,15] that such **reuse** occurs when $A$ is rank-deficient: $z$ and $z'$ access the same value of $X$, iff $Az = Az'$, or $z - z'$ is a linear

\[\text{Admittedly, the notion of a parent is ambiguous in a lattice, but since the faces will be visited recursively in a top down manner, the call tree of this recursion will provide the necessary context to identify the constraint uniquely.}\]

\[\text{Although not explicitly stated, Gautam and Rajopadhye [6] assumed a single size parameter.}\]
combination of the basis vectors of the null space of \( A \). The reuse space of our expression \( e \), which we denote as \( \Re(e) \) is just the null-space of \( A \). When an expression is to be evaluated only at points in some domain \( D \), its share space \( S(D, e) \), is defined to be \( L(D) \cap \Re(e) \). This is a linear space.

2 Reduction Simplification

For clarity of explanation, we first assume that the reduction operator admits an inverse, \( \ominus \). Later, we consider noninvertible operators. We simplify Eqn. 5 recursively, going down the thick face lattice, starting with \( D \), and at each step we simplify Eqn. 5 but restricted to \( F \).

The key idea is that exploiting reuse along \( \rho \in S(\mathcal{F}, e) \) avoids evaluating \( e \) at most points in \( \mathcal{F} \). Specifically, let \( \mathcal{F}' \) is the translation of \( \mathcal{F} \) along \( \rho \), and \( \mathcal{F}' \setminus \mathcal{F} \) and \( \mathcal{F} \setminus \mathcal{F}' \) their differences. The union of these two is also the union of the thick facets of \( \mathcal{F} \). Exploiting reuse along \( \rho \) converts the original equation to a set of residual computations defined only on (a subset of) the facets of \( \mathcal{F} \). All the computation in \( \mathcal{F} \cap \mathcal{F}' \) is avoided. To understand the details, we first define two labels on faces (remember that in this recursive traversal, every face is associated with a unique constraint, \( \langle c, \gamma \rangle \)).

First, a face \( \mathcal{F} \), is said to be a boundary face if its image by \( B, B(\mathcal{F}) \) is also a face of \( D_{\mathcal{F}} \), i.e., it contributes to a “boundary” of the result. This happens if \( Bc = 0 \). Second, and with respect to any reuse vector \( \rho \), we define a face to be inward if \( c\rho > 0 \) (and respectively, outward if \( c\rho < 0 \) and invariant if \( c\rho = 0 \)).

A preprocessing step ensures that there are no invariant boundary faces. We do not perform any residual computations on the outward boundary, and on the invariant facets of \( \mathcal{F} \). The other residual computations are used in the following way.

- The inward boundary faces are used to initialize the final answer.
- The results of inward non-boundary faces are combined with \( Y[B(z - \rho)] \) using the operator \( \ominus \).
- The results of outward non-boundary faces are combined with \( Y[B(z - \rho)] \) using the operator \( \ominus \).

Optimality At each step of the recursion, the asymptotic complexity is reduced by exactly one polynomial degree, because facets of \( \mathcal{F} \) have one fewer free index. Furthermore, at each step, the faces saturated by the ancestors ensure that the new \( \rho \) is linearly independent of the previously chosen ones. Hence, the method is optimal—the reduction in asymptotic complexity is by a polynomial whose degree is the number of dimensions of the feasible reuse space of the original domain, \( L(D) \cap \Re(e) \), and all available reuse is fully exploited. This holds regardless of the choice of \( \rho \) at any level of the recursion (all roads lead to Rome) even though there are infinitely many choices in general.

Handling operators without inverses and impact on optimality Many algorithms, particularly in dynamic programming, perform reductions with operators like the min and the max, which do not admit an inverse. To handle such equations, we must ensure that the residual computation whose results are combined with \( \ominus \) must have an empty domain, i.e., the current face \( \mathcal{F} \) does not have any non-boundary outward facet, i.e., \( c\rho \geq 0 \) holds for all non-boundary facets.

There are two implications of this. First, this means that the feasible space of legal reuse vectors \( \rho \) is no longer the linear subspace \( S(D, e) \), but rather, must satisfy additional linear inequalities. Indeed, the feasible space may even be empty, and we may not be able to exploit all available reuse. For example, if the operator in Eqn. 1 is max, and we choose \( \rho = [1, 0] \), it makes the lower bound \( j \geq i \), an outward facet, while \( \rho = [-1, 0] \), makes the upper bound \( j \leq 2i \) outward. Hence, this equation cannot be simplified and its complexity will remain \( O(N^2) \).

\(^3\)When it is more than one dimensional, there are infinitely many choices for the basis vectors.
The second consequence is that, as the above algorithm recurses down the thick face lattice, the choice of the ρ at an earlier level may affect the feasible space of lower levels, and hence the recursive algorithm is not optimal. Gautam and Rajopadhye [6] solve this problem by observing that the infinite feasible space can be partitioned into equivalence classes based on the labels they assign to the non-boundary facets. There are because there are finitely many faces, and each has finitely many possible labels, and hence a backtracking search over the thick face lattice, formulated as a dynamic programming algorithm leads to an optimal choice of ρ’s.

3 Simplifying Multiple Statement Reductions

Recently, Yang, Atkinson and Carbin [16] extended this work to systems of dependent equations like the one in Eqn. 2. The main difficulty is that choosing a reuse vector ρ to exploit introduces a new dependence in the program: \( Y[Bz] \) depends on \( Y[B(z - \rho)] \), which corresponds to a dependence vector \( B\rho \). This may degrade the schedule of the program, or worse still, may even lead to a system of equations that do not admit a schedule, e.g., had we chosen to exploit reuse along \([-1, 0]\) in Eqn. 2

Yang et al. resolve this problem by combining the simplification and scheduling problem, building on a long history of research on scheduling in the polyhedral model. They use the state of the art formulation by Pouchet et al. [11, 10, 12] that formulates multidimensional scheduling as a single linear optimization problem. For each variable/equation \( X_i \) defined over a \( d_i \)-dimensional domain, there is an \((1+d_i) \times (1+d_i)\) matrix of schedule coefficients \( \Theta_i \), and the causality constraints are translated to linear inequalities defining a feasible space. Many objective functions are used in the literature, the most common being one that simultaneously optimizes for locality and parallelism [1].

However, there are two difficulties in doing this directly. The first is that, in the Gautam-Rajopadhye recursive algorithm, the ρ vectors are chosen one by one, and for each chosen introduces \( B\rho \) as a new dependence in the program that was not present in the original program.

Yang et al. resolve this by first formulating the Gautam-Rajopadhye algorithm as a single linear programming problem, by setting us simultaneous constraints that must be satisfied for each face of \( D \). Next, they modify the scheduling constraints so that the unknown ρ vectors appear in the causality constraints. Because of this, the formulation does not remain a linear programming problem, but becomes bilinear. They couple this with a linear objective function that minimizes the asymptotic complexity.

They concede that the solution of this problem may be difficult in the general case, but propose a very simple heuristic that works really well for many of the examples and algorithms they encounter in statistical machine learning. They do not provide data about the execution time of their (non-heuristic) implementation of the bilinear programming formulation.

4 Simplifying dependent reductions as a backtracking search

We now describe our main result. We show how to solve the problem of simplifying dependent reductions, like the ones tackled by Yang et al., by extending the Gautam-Rajopadhye backtracking search algorithm. It relies on the early work on polyhedral scheduling [2] and builds on a long history of scheduling [7, 8, 14, 9, 13, 3, 4].

We first define compatibility to capture the notion of the conditions under which an additional dependence (e.g., one that is introduced by simplification) does not introduce dependence cycles, and allows the program to admit a legal schedule.

**Definition 1.** Let \( T \) be the space of all legal schedules for a program. We say that a new uniform self dependence vector \( r \) on variable \( Y \) is compatible with \( T \), or with the original program, if some feasible schedule \( \Theta_Y \), respects...
the constraint $\Theta_Y r > 0$, where $\succ$ denotes the lexicographic order. This holds iff

$$\exists \Theta \in \mathcal{T} \text{ s.t. } \Theta r > 0$$

Otherwise, we say that $r$ violates $\mathcal{T}$. More specifically, a reuse vector $\rho$ for simplifying the reduction producing $Y$ is legal if the uniform (self) dependence vector, $B\rho$ on the variable $Y$ is compatible with the original program.

**Proposition 1.** The feasible space of all multidimensional schedules for polyhedral program is a blunt, finitely generated, rational cone. (see https://en.wikipedia.org/wiki/Convex_cone).

**Proof.** A polyhedral set $\mathcal{P}$ is a cone iff for any point $x \in \mathcal{P}$, and for a positive scalars, $\alpha$, the point $\alpha x$, is also in $\mathcal{P}$. The causality constraint states that for all pairs of iteration points, $z_X \in D_X$ and $z_Y \in D_Y$ such that $X[z_x]$ depends on $Y[z_y]$, the time-stamp of the producer is strictly before the time stamp of the consumer. Recall that a $d$-dimensional schedule $\Theta$ satisfies causality iff the following constraint is satisfied (here $\delta = [0 \ldots 0, 1]$ is a $d$-dimensional vector whose last element is 1).

$$\Theta_X \begin{bmatrix} z_X \\ p \\ 1 \end{bmatrix} - \Theta_Y \begin{bmatrix} z_Y \\ p \\ 1 \end{bmatrix} \succ \delta \quad (6)$$

Now if $\Theta$ is a legal schedule vector\(^4\) then it is easy to see that $\alpha \Theta$, for any positive $\alpha$ also satisfies\(^5\). Indeed it is just a $\alpha$-fold slowdown of $\Theta$. It is also easy to show that the cone is blunt, i.e., it does not contain the origin, and because the schedule coefficients are integers, it is finitely generated. \qed

We now formulate the additional legality conditions for reuse vectors used during simplification. Consider the cone defining the feasible schedule $\mathcal{T}$ of a program (system of equations) prior to simplification. Let its projection on the dimensions representing the variable $Y$ be $\mathcal{C}$, and let $\mathcal{C}$ have $m$ generators, $g_1 \ldots g_m$.

**Proposition 2.** Simplifying the equation for $Y$ using a reuse vector, $\rho$ is legal iff, for some generator, $g_i$ of $\mathcal{C}$, $g_i B\rho \geq 0$.

Thus, the legality conditions for reuse vectors used during the recursion in the Gautam-Rajopadhye algorithm now become the disjunction of $m$ convex constraints. There are possibly $m$-fold more choices to explore (but with the possibility of early termination), and once again, the optimality argument carries over.

**References**

[1] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. Pluto: A practical and fully automatic polyhedral program optimization system. In ACM Conference on Programming Language Design and Implementation, pages 101–113, Tuscon, AZ, June 2008. ACM SIGPLAN.

[2] Jean-Marc Delosme and Ilse C. F. Ipsen. Systolic array synthesis: Computability and time cones. In M. Cosnard, P. Quinton, Y. Robert, and M. Tchuente, editors, Int. Workshop on Parallel Algorithms and Architectures, pages 295–312. Elsevier Science, North Holland, April 1986.

[3] Paul Feautrier. Some efficient solutions to the affine scheduling problem. Part I. one-dimensional time. International Journal of Parallel Programming, 21(5):313–347, 1992.

\(^4\)Note that we call it a vectors even though it is a set of matrices, one for each variable in the program, since they constitute the unknown variables in the optimal scheduling linear program.
[4] Paul Feautrier. Some efficient solutions to the affine scheduling problem. Part II. multidimensional time. *International Journal of Parallel Programming*, 21(6):389–420, 1992.

[5] J. A. B. Fortes and D. Moldovan. Data broadcasting in linearly scheduled array processors. In *Proceedings, 11th Annual Symposium on Computer Architecture*, pages 224–231, 1984.

[6] Gautam Gupta and S. Rajopadhye. Simplifying reductions. In *POPL 2006: Proceedings, ACM Symposium on Principles of Programming Languages*, pages 30–41, Charleston SC., January 2006. ACM, ACM Press.

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

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

[9] C. Mauras, P. Quinton, S. Rajopadhye, and Y. Saouter. Scheduling affine parameterized recurrences by means of variable dependent timing functions. In S. Y. Kung and E. Swartzlander, editors, *International Conference on Application Specific Array Processing*, pages 100–110, Princeton, New Jersey, Sept 1990. IEEE Computer Society.

[10] L.-N. Pouchet, C. Bastoul, A. Cohen, and J. Cavazos. Iterative optimization in the polyhedral model: Part II, multidimensional time. In *PLDI*, pages 90–100, 2008.

[11] L.-N. Pouchet, C. Bastoul, A. Cohen, and N. Vasilache. Iterative optimization in the polyhedral model: Part I, one-dimensional time. In *ACM International Conference on Code Generation and Optimization (CGO’07)*, San Jose, California, March 2007. 144-156.

[12] Louis-Noël Pouchet, Uday Bondhugula, Cédric Bastoul, Albert Cohen, J. Ramanujam, and P. Sadayappan. Loop transformations: Convexity, pruning and optimization. In *POPL 11: Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 549–562, Copenhagen, Denmark, January 2011. ACM.

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

[14] S. V. Rajopadhye, S. Purushothaman, and R. M. Fujimoto. On synthesizing systolic arrays from recurrence equations with linear dependencies. In *Proceedings, Sixth Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 488–503, New Delhi, India, December 1986. Springer Verlag, LNCS 241.

[15] F. C. Wong and Jean-Marc Delosme. Broadcast removal in systolic algorithms. In *International Conference on Systolic Arrays*, pages 403–412, San Diego, CA, May 1988.

[16] Cambridge Yang, Eric Atkinson, and Michael Carbin. Simplifying multiple-statement reductions with the polyhedral model, 2020.