Simplifying Dependent Reductions*

Sanjay Rajopadhye
Sanjay.Rajopadhye@colostate.edu
Computer Science Department
Ft. Collins, CO, USA

Abstract

Reductions combine multiple input values with an associative operator to produce (a single or multiple) result(s). When the *same* input value contributes to *multiple* outputs, there is an opportunity to *reuse* partial results, enabling *reduction simplification*. Gautam and Rajopadhye [6] showed how reductions in the polyhedral model could be simplified *automatically* and *optimally*. In this paper, we tackle the case when (some) input values *depend on* (some) outputs. This couples simplification with the classic scheduling problem. We show how to extend the Gautam-Rajopadhye algorithm to optimally simplify such dependent reductions.

ACM Reference Format:
Sanjay Rajopadhye. 2018. Simplifying Dependent Reductions. In *IMPACT '21: 11th International Workshop on Polyhedral Compilation Techniques*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/1122445.1122456

Keywords: Polyhedral model, Reduction, Scheduling

1 Introduction

Reductions, i.e., computational patterns that combine multiple input values with an associative operator to produce (a single or multiple) result(s) are ubiquitous, and important. A *commutative* reduction is one where the operator is also commutative. Mathematical equations involving reductions serve as clean succinct specifications of most dense linear algebra computations, of algorithms for signal/image processing, and of dynamic programming solutions to many optimization problems. Furthermore, the recent explosion of AI/ML, deep learning, and data science has highlighted higher order tensors as an important data structure. Tensor algorithms, notably tensor contractions are also naturally expressed using reductions.

*Supported in part by ARO, contract W911NF-19-1-0420.

IMPACT '21, January 20, 2021,

ACM ISBN 978-1-4503-XXXX-X/18/06…$00.05
https://doi.org/10.1145/1122445.1122456

Polyhedral reductions are those where the input and output data collections are (dense) multidimensional arrays (i.e., *tensors*), accessed with linear/affine functions of the indices.

When the *same* input value contributes to multiple outputs of a reduction, there is an opportunity for reusing partial results, thereby reducing (pun intended) the asymptotic complexity. This is called *simplification*.

Consider the following equation.

$$P[i] = \begin{cases} i = 0 & : \quad 0 \\ i > 0 & : \quad \sum_{j=0}^{i-1} Q[j] \end{cases} \tag{1}$$

The equation specifies that each element of a one dimensional array P is obtained by *reducing* (using addition as the operator) a subset of values of an input array Q. The arrays have size $O(N)$, and N is viewed as a *parameter* of the program/equation. We seek to optimize the asymptotic execution time of our program as a function of N.

Since the i-th element of the output involves the reduction of i values, the nominal complexity of each equation is $O(N^2)$. However, this can be improved if we can recognize that the summation is a prefix-sum: all the values (except the last one) 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 equation below.

$$P[i] = \begin{cases} i = 0 & : \quad Q[i] \\ i > 0 & : \quad P[i-1] + Q[i-1] \end{cases} \tag{2}$$

Eqn. 2 is not the only possible simplification. We could choose to compute the i-th output from the $(i+1)$-th one, and get

$$P[i] = \begin{cases} i = N & : \quad \sum_{j=0}^{i-1} Q[j] \\ i > 0 & : \quad P[i+1] - Q[i] \end{cases} \tag{3}$$

Although the first branch of this equation still performs a reduction operation taking $O(N)$ time, it is evaluated for only *one* point of the result (at $i = N$) and the other branch specifies that each of the remaining $\Theta(N)$ points take $O(1)$ time, so the overall complexity is $O(N)$, albeit two-fold slower that the first solution.

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). They assume, as in the example above, that the reductions are *independent* in the sense that the variable(s) *read* in the reduction are distinct from the variable *written* or *accumulated* by the equation.

This paper addresses the case when this last assumption does not hold, i.e., *dependent* reductions. Consider a variation of the above example, where X is both an input and an output (appears on both the left and right hand side) of the reduction equation as in the example below.

$$P[i] = \begin{cases} i = 0 & : \quad foo(0) \\ i > 0 & : \quad foo\left(\sum_{j=0}^{i-1} P[j]\right) \end{cases} \tag{4}$$

This equation is equivalent to a system with Eqn. 1 computing just the reduction, and Eqn. 5 below applying the function *foo* to its result.

$$Q[i] = foo(P[i]) \tag{5}$$

But now, Eqn. 3 while still a legal *simplification* does not yield a legal program, because of a cyclic dependence: $P[N]$ depends on the entire prefix $0 \ldots N-1$ of elements of Q, but $Q[i]$ depends on its successor. The new system of equations is not computable, i.e., does not admit a legal schedule.

This is the problem we tackle. We modify the Gautam-Rajopadhye algorithm to simplify dependent reductions while ensuring that the new system remains computable, while retaining optimality of the Gautam-Rajopadhye algorithm.

2 Background

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 $cz + \gamma \geq 0 \geq cz + \gamma'$). When $\mathcal{D}$ has such thick equalities, we say that it *effectively saturates* the constraint $\langle c, \gamma \rangle$, or, by abuse of notation, simply saturates c. Along any vector, ρ 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 (thick) linear subspace that contains $\mathcal{D}$.

Parameters, volume and complexity: A polyhedron may have one or more designated indices, like N above, called its size parameter(s).[1] 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.[2] 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.

A *facet*, $\mathcal{F}$ of a polyhedron $\mathcal{D}$ is its intersection with the *equality* $az + \alpha = 0$ associated with exactly one constraint. We say that the facet *saturates* the constraint $\langle a, \alpha \rangle$. More than one constraint may be saturated, and this yields *faces*. The concept of "thickness" can be extended to faces too: a thick face is the set of points in the polyhedron burt within a constant distance of the face. 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 *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."[3]

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

$$\begin{aligned} \forall z \in \mathcal{D} : Y[Bz] &= \bigoplus e(z) \\ &= \bigoplus X[Az] \end{aligned} \tag{6}$$

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, 17, 19] 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 combination of the *basis vectors* of the null space of A. The *reuse space* of our expression e, which we denote as $\mathcal{R}(e)$ is just the null-space of A, When an expression is to be evaluated

[1] Although not explicitly stated, Gautam and Rajopadhye [6] assumed a single size parameter.

[2] We can also simplify parameterized programs/equations where each instance has an unbounded computation.

[3] 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 uniquely identify the constraint being saturated.

only at points in some domain D, its *share space* $\mathcal{S}(\mathcal{D}, e)$, is defined to be $\mathcal{L}(\mathcal{D}) \cap \mathcal{R}(e)$. This is a linear space.[4]

3 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. 6 recursively, going down the thick face lattice, starting with $\mathcal{D}$, and at each step we simplify Eqn. 6, but restricted to $\mathcal{F}$.

The key idea is that exploiting reuse along $\rho \in \mathcal{S}(\mathcal{F}, e)$ avoids evaluating e at most points in $\mathcal{F}$. Specifically, let $\mathcal{F}'$ is the translation of $\mathcal{F}$ along ρ, and $\mathcal{F}'\backslash\mathcal{F}$ and $\mathcal{F}\backslash\mathcal{F}'$ be their differences. The union of these two is also the union of the thick facets of $\mathcal{F}$. Exploiting reuse along ρ 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 $\mathcal{F}$ 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 $\mathcal{D}_Y$, i.e., it contributes to a "boundary" of the result. This happens if $\ker(B) \subseteq \ker(c)$. Second, and with respect to any reuse vector ρ, we define a face to be *inward* (respectively, *outward* and *invariant*) if $c\rho > 0$ (respectively, $c\rho < 0$ and $c\rho = 0$).

A preprocessing step ensures that there are no invariant boundary faces. No residual computation is performed on any outward boundary, and on 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 $\oplus$.
- 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 ρ 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, $\mathcal{L}(\mathcal{D}) \cap \mathcal{R}(e)$, and all available reuse is fully exploited. This holds regardless of the choice of ρ 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., $c_i\rho \geq 0$ holds for all non-boundary facets.

There are two implications of this. First, this means that the feasible space of legal reuse vectors ρ is no longer the linear subspace $\in \mathcal{S}(\mathcal{D}, e)$, 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, consider Eqn. 7 below.

$$Y[i] = \max_{j=i}^{2i} X[i] \tag{7}$$

If 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 $O(N^2)$.

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 may have to backtrack and try alternative values of ρ, but the feasible space of legal reuse vectors may be infinite.

Gautam and Rajopadhye solve this by proving that the infinite feasible space can be partitioned into equivalence classes based on the labels they assign to the non-boundary facets. Thus, because there are finitely many faces, and each has finitely many possible labels, a backtracking search over the thick face lattice leads to an optimal choice of ρ's.

However, note that despite the optimality, it may not be possible exploit all available dimensions of reuse when the reduction operator does not admit an inverse, as we saw in the example of Eqn 7.

4 Simplifying dependent reductions

We now describe our main result. We show how to solve the problem of simplifying dependent reductions 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 [3, 4, 8, 9, 12–17].

We first define *compatibility* to capture the notion of the conditions under which a new 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 $\mathcal{T}$ 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** $\mathcal{T}$, or with the original program, if some feasible schedule Θ_Y, respects the constraint $\Theta_Y r \succ 0$, where $\succ$ denotes the lexicographic order, i.e., iff*

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

*Otherwise, we say that r **violates** $\mathcal{T}$.*

[4] When it is more than one dimensional, there are infinitely many choices for the basis vectors.

*We say that a reuse vector, ρ 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 pogram 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 scalar α, the point αx, is also in $\mathcal{P}$. The causality constraint states that for all pairs of iteration points, $z_X \in \mathcal{D}_X$ and $z_Y \in \mathcal{D}_Y$ 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 Θ satisfy causality iff the following constraint is satisfied.

$$\Theta_X \begin{bmatrix} z_X \\ p \\ 1 \end{bmatrix} - \Theta_Y \begin{bmatrix} z_Y \\ p \\ 1 \end{bmatrix} \succ 0 \tag{8}$$

Now, if Θ is a legal schedule vector[5] then it is easy to see that $\alpha\Theta$, for any positive α also satisfies 8. Indeed it is just a α-fold slowdown of Θ. It is also easy to show that the cone is *blunt*, i.e., it does not contain the origin (otherwise a schedule that maps all the instances of all variables to the single time step, **0** would be a legal schedule), and because the schedule coefficients are integers, it is *finitely generated.* □

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 C, and let C have m generators, $g_1 \ldots g_m$.

Theorem 4.1. *Simplifying the equation for Y using a reuse vector, ρ is legal iff, for some generator, g_i of C, $g_i B\rho \geq 0$.*

Proof. If $g_i B\rho < 0$ for all generators, then no feasible schedule $\Theta_Y \in C$ satisfies the self dependence $B\rho$, it satisfies the constraint that $\Theta_Y[B\rho, 0, 0]^T \succeq 0$ □

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.

5 Related work and discussion

Reductions as first class constructs in programming languages have a long history, going back to Iverson's APL [7]. In the context of the polyhedral model, Roychowdhury's Ph.D. dissertation [18] used Eqn. 6 with a linear access function on the lhs as high level a specification (he called them *weakly sigle assignment codes* or WSAC). His work was in the context of systolic array synthesis, and the hot topic *du jour* was uniformization/localization of broadcasts of many-to-one read via the properties of the read matrix A. He also showed that since a many-to-one write was the dual of a broadcast the same techniques could be used to serialize reductions. The dissertation is highly recommended reading with interesting results that haven't appeared in a journal or conference. Le Verge [10] introduced reductions in Alpha, a polyhedral domain specific (pun intended) equational language [11].

Yang, Atkinson and Carbin [20, 21] were the first to formulate and tackle the problem of simplifying dependent reductions, and our motivating example is due to them. Their paper makes a number of important contributions.

- They formulate the problem of simplifying dependent reductions as a bilinear programming problem.
- They also present a simple heuristic that works when (i) provided a sequential schedule, and (ii) the reduction operator admits an inverse.
- Fifteen years after Gautam and Rajopadhyes' work, they provide many pragmatic instances, drawn from probabilistic programming and Bayesian inference, where simplifying reductions is an important optimization. In the thirteen benchmarks they studied, it yielded at least one degree reduction of the asymptotic complexity.

Like us, they combine simplification with scheduling, but they do it more directly. We explain their approach and discuss its limitations. They first use the state of the art scheduler by Pouchet et al. [13–15] to formulate multidimensional scheduling as a *single* linear optimization problem. For each variable/equation X_i defined over a d_i-dimensional domain, there is an $m \times (1 + d_i)$ matrix of schedule coefficients[6] Θ_i. 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 directly extending these schedule constraints to incorporate simplification. The first is that, in the Gautam-Rajopadhye recursive algorithm, the ρ vectors are chosen one by one as the algorithm traverses the face lattice. Each face introduces $B\rho$ as a new dependence, not present in the original program, and scheduling constraints must now be augmented.

Yang et al. resolve this by first formulating the Gautam-Rajopadhye algorithm as a *single* linear programming problem, by setting up *simultaneous* constraints that must be

[5] Note that we usually think of a d-dimensional schedule as a matrix with $d + 1$ rows, but the set of all these coefficients constitute the unknown variables in the optimal scheduling linear program, and the *schedule vector* is a vector of all these coefficients, and this is this that we claim belongs to a cone.

[6] Here, m is the number of dimensions of the schedule, and it must be specified in the formulation. Clearly m need not be more than $1 + d'_i$, where the maximum number of dimensions of any variable in the program is d'_i.

satisfied for *all* the faces of $\mathcal{D}$, each one with a distinct reuse vector. Next, they add new scheduling constraints to include these (self) dependence ρ vectors in the causality constraints of the form, $\Theta_Y\ [B\rho, 0, 0]^T > 0$. They couple this with a linear objective function to minimize the asymptotic complexity.

However, since the schedule coefficients Θ_Y and the reuse vectors ρ are both unknowns, the formulation becomes a *bilinear* programming problem. Furthermore, because the bilinear program has an independent reuse vector ρ for *each face* of the face lattice, and the number of faces is usually exponential in the size of the input program,[7] their bilinear program is most likely intractable. The authors concede this and indeed, state that they consider it, "only as a specification instead of a complete solution."

Rather, they propose a simple heuristic that works really well for many (indeed, all) examples they encounter in statistical machine learning. They start with a sequential schedule that may be chosen by any standard algorithm. Next, they choose an arbitrary ρ for each face. If it violates the given schedule, then they replace it by its negation $-\rho$. This works if the operator admits an inverse since in this case, the share space $\mathcal{S}(\mathcal{D}, e)$ is a linear space. Moreover, they state that the initial sequential schedule is obtained from a "PLUTO-like scheduler built into ISL." To the best of our knowledge the ISL implementation of the PLUTO scheduler is parallel, which can be made sequential, but it is not easy to formulate the feasible space of all sequential schedules.

6 Conclusion

We tackled the problem of simplifying dependent polyhedral reductions, a strict generalization of the problem previously addressed by Gautam and Rajopadhye [6]. We show how the Gautam-Rajopadhye algorithm can be extended to resolve this new problem, while retaining the optimality of the complexity reduction. Our algorithm works for all reduction operators, not just those that admit an inverse, such as max, and min that occur in dynamic programming algorithms. We are still on the lookout for programs and algorithms that may benefit from reduction simplification.

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] 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] Kenneth A. Iverson. *A Programming Language*. Jonh Wiley & Sons, New York, 1962.

[8] 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.

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

[10] H. Le Verge. Reduction operators in alpha. In D. Etiemble and J-C. Syre, editors, *Parallel Algorithms and Architectures, Europe*, LNCS, pages 397–411, Paris, June 1992. Springer Verlag. See also, Le Verge Thesis (in French).

[11] H. Le Verge, C. Mauras, and P. Quinton. The ALPHA language and its use for the design of systolic arrays. *Journal of VLSI Signal Processing*, 3(3):173–182, September 1991.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] V. P. Roychowdhury. *Derivation, Extensions and Parallel Implementation of Regular Iterative Algorithms*. PhD thesis, Stanford University, Department of Electrical Engineering, Stanford, CA, December 1988.

[19] 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.

[20] Cambridge Yang, Eric Atkinson, and Michael Carbin. Simplifying multiple-statement reductions with the polyhedral model. Technical Report 2007.11203, arXiv, July 2020.

[21] Cambridge Yang, Eric Atkinson, and Michael Carbin. Simplifying dependent reductions in the polyhedral model. In *POPL 21: Proceedings of the 48th annual ACM Symposium on Principles of Programming Languages*, pages xx–yy, Copenhagen, Denmark, January 2021. ACM. The title of the paper in the POPL 2021 program/web site is "Simplifying Multiple-Statement Reductions with the Polyhedral Model".

[7] For example, a simple d-dimensional loop nest has $2d$ constraints—the lower and upper bounds of the loops—and 2^d vertices.