Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model

Uday Bondhugula[1], Muthu Baskaran[1], Sriram Krishnamoorthy[1],
J. Ramanujam[2], Atanas Rountev[1], and P. Sadayappan[1]

[1] Dept. of Computer Science and Engineering, The Ohio State University, Columbus, OH, USA
{bondhugu,baskaran,krishnsr,rountev,saday}@cse.ohio-state.edu,
[2] Dept. of Electrical and Computer Engg., Louisiana State University, Baton Rouge, LA , USA
jxr@ece.lsu.edu

Abstract. The polyhedral model provides powerful abstractions to optimize loop nests with regular accesses. Affine transformations in this model capture a complex sequence of execution-reordering loop transformations that can improve performance by parallelization as well as locality enhancement. Although a significant body of research has addressed affine scheduling and partitioning, the problem of automatically finding good affine transforms for communication-optimized coarse-grained parallelization together with locality optimization for the general case of arbitrarily-nested loop sequences remains a challenging problem.

We propose an automatic transformation framework to optimize arbitrarily-nested loop sequences with affine dependences for parallelism and locality simultaneously. The approach finds good tiling hyperplanes by embedding a powerful and versatile cost function into an Integer Linear Programming formulation. These tiling hyperplanes are used for communication-minimized coarse-grained parallelization as well as for locality optimization. The approach enables the minimization of inter-tile communication volume in the processor space, and minimization of reuse distances for local execution at each node. Programs requiring one-dimensional versus multi-dimensional time schedules (with scheduling-based approaches) are all handled with the same algorithm. Synchronization-free parallelism, permutable loops or pipelined parallelism at various levels can be detected. Preliminary studies of the framework show promising results.

1 Introduction and Motivation

Current trends in architecture are increasingly towards larger number of processing elements on a chip. This has led to multi-core architectures becoming mainstream along with the emergence of several specialized parallel architectures or accelerators. The difficulty of programming these architectures to effectively tap the potential of multiple on-chip processing units is a well-known challenge. Among several approaches to addressing this issue, one that is very promising but simultaneously very challenging is automatic parallelization.

Many compute-intensive applications often spend most of their running time in nested loops. This is particularly common in scientific and engineering applications.

The polyhedral model [6, 10, 12] provides a powerful abstraction to reason about transformations on such loop nests by viewing a dynamic instance (iteration) of each statement as an integer point in a well-defined space which is the statement's *polyhedron*. With such a representation for each statement and a precise characterization of inter or intra-statement dependences, it is possible to reason about the correctness and goodness of a sequence of complex loop transformations using machinery from Linear Algebra and Integer Linear Programming. The polyhedral model is applicable to loop nests in which the data access functions and loop bounds are affine combinations (linear combination with a constant) of the enclosing loop variables and parameters. While a precise characterization of data dependences is feasible for programs with static control structure and affine references/loop-bounds, code with non-affine array access functions or dynamic control can also be handled, using conservative assumptions.

Early approaches to automatic parallelization applied only to perfectly nested loops and involved the application of a sequence of transformations to the program's attributed abstract syntax tree. The polyhedral model has enabled much more complex programs to be handled, and easy composition and application of more sophisticated transformations [6, 12]. The task of program optimization in the polyhedral model may be viewed in terms of three phases: (1) static dependence analysis of the input program, (2) transformations in the polyhedral abstraction, and (3) generation of efficient loop code. Despite the progress in these techniques, several scalability challenges limited applicability to small loop nests. Significant recent advances in dependence analysis [28] and code generation [2, 23, 27] have demonstrated the applicability of the polyhedral techniques to real applications. However, current state-of-the-art polyhedral implementations still apply transformations manually and significant time is spent by an expert to determine the best set of transformations [6, 12]. An important open issue is the choice of transformations from the huge space of valid transforms. Our work addresses this problem, by formulating a way to obtain good transformations fully automatically.

Tiling is a key transformation and has been studied from two perspectives — data locality optimization and parallelization. Tiling for locality requires grouping points in an iteration space into smaller blocks (tiles) allowing reuse in multiple directions when the block fits in a faster memory (registers, L1, or L2 cache). Tiling for coarse-grained parallelism fundamentally involves partitioning the iteration space into tiles that may be concurrently executed on different processors with a reduced frequency and volume of inter-processor communication: a tile is atomically executed on a processor with communication required only before and after execution. Hence, one of the key aspects of an automatic transformation framework is to find good ways of performing tiling.

Existing automatic transformation frameworks [1, 13, 17–19] have one or more drawbacks or restrictions that do not allow them to effectively parallelize/optimize loop nests. All of them lack a realistic cost model that is suitable for coarse-grained parallel execution as is used in practice with manually developed parallel applications. With the exception of Griebl [13], previous work generally focuses on one or the other of the complementary aspects of parallelization and locality optimization. The approach we develop answers the following question: What is a good way to tile imperfectly nested loop sequences to minimize the volume of communication between tiles (in processor space) as well as improve data reuse at each processor?

The rest of this paper is organized as follows. Section 2 provides an overview of the polyhedral model. In Section 3 describes our automatic transformation framework. Section 4 shows step-by-step application of our approach through an example. Section 5 provides a summary of the implementation and initial results. Section 6 discusses related work and conclusions are presented in Section 7. Full details of the framework, transformations and optimized code for various examples, and experimental results are available in extended reports [3, 4].

2 Overview of the Polyhedral Framework

The set X of all vectors $x \in \mathbf{Z}^n$ such that $\mathbf{h}.\boldsymbol{x} = k$, for $k \in \mathbf{Z}$, forms an *affine hyperplane*. The set of parallel *hyperplane instances* corresponding to different values of k is characterized by the vector $\boldsymbol{h}$ which is normal to the hyperplane. Each instance of a hyperplane is an $n-1$ dimensional affine sub-space of the n-dimensional space. Two vectors $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ lie in the same hyperplane if $\mathbf{h}.\boldsymbol{x_1} = \mathbf{h}.\boldsymbol{x_2}$.

The set of all vectors $\boldsymbol{x} \in \mathbf{Z^n}$ such that $A\boldsymbol{x} + \boldsymbol{b} \geq 0$, where A is an integer matrix, defines a (convex) integer *polyhedron*. A *polytope* is a bounded polyhedron. Each runtime instance of a statement S is identified by its iteration vector $\boldsymbol{i}$, of dimensionality m_{S_k}, containing values for the indices of the loops surrounding it from outermost to innermost. Hence, a statement S is associated with a polytope characterized by a set of bounding hyperplanes or faces. This is true when the loop bounds are affine combinations of outer loop indices and program parameters (typically, symbolic constants representing the problem size). Let $\boldsymbol{p}$ be the vector of the program parameters.

A well-known known result useful for polyhedral analyses is the following [26]:

Lemma 1 (Affine form of Farkas Lemma). *Let $\mathcal{D}$ be a non-empty polyhedron defined by s affine inequalities or faces: $\mathbf{a_k}.\boldsymbol{x} + b_k \geq 0$, $1 \leq k \leq s$. An affine form $\psi(\boldsymbol{x})$ is non-negative everywhere in $\mathcal{D}$ iff it is a positive affine combination of the faces:*

$$\psi(\boldsymbol{x}) \equiv \lambda_0 + \sum_k \lambda_k(\mathbf{a_k}\boldsymbol{x} + b_k),\ \lambda_k \geq 0 \tag{1}$$

The non-negative constants λ_k are referred to as Farkas multipliers.

Polyhedral Dependences. Our dependence model is of exact affine dependences and same as the one used in [6, 18, 22, 28]. Dependences are determined precisely through array dataflow analysis [9], but the input need not be in single-assignment form. All dependences including anti (write-after-read), output (write-after-write) and input (read-after-read) dependences are considered. The Data Dependence Graph (DDG) is a directed multi-graph with each vertex representing a statement, and an edge, $e \in E$, from node S_i to S_j representing a polyhedral dependence from a dynamic instance of S_i to one of S_j: it is characterized by a polyhedron, $\mathcal{P}_e$, called the *dependence polyhedron* that captures the exact dependence information corresponding to edge, e (see Fig. 1(b) for an example). The dependence polyhedron is in the sum of the dimensionalities of the source and target statement's polyhedra (with dimensions for program parameters as well). Though the equalities in $\mathcal{P}_e$ typically represent the affine function mapping the target iteration vector $\boldsymbol{t}$ to the particular source $\boldsymbol{s}$ that is the last access to the conflicting

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    S1: A[i,j] = A[i,j]+u[i]*v[j];
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    S2: x[i] = x[i]+A[j,i]*y[j];
```

(a) original code

$$\mathcal{P}_{e_1}: \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & -1 \\ \hline 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \\ i' \\ j' \\ \hline N \\ 1 \end{bmatrix} \begin{matrix} \geq 0 \\ \geq 0 \\ \geq 0 \\ \geq 0 \\ \hline = 0 \\ = 0 \end{matrix}$$

(b) Dependence polyhedron for the inter-statement dependence on A

	$S1$			$S2$		
	i	j	$const$	i	j	$const$
c_1	0	1	0	1	0	0
c_2	1	0	0	0	1	0
c_3	0	0	0	0	0	1

(c) transformation

```
for (c1=0; c1<N; c1++)
  for (c2=0; c2<N; c2++)
    A[c2,c1] = A[c2,c1]+u[c2]*v[c1];
    x[c1] = x[c1]+A[c2,c1]*y[c1];
```

(d) transformed code

Fig. 1. Polyhedral transformation and dependences

memory location, also known as the *h-transformation* [10], the last access condition is not necessary; in general, the equalities can be used to eliminate variables from $\mathcal{P}_e$. In the rest of this section, we assume for convenience that $\boldsymbol{s}$ can be completely eliminated using h_e, being substituted by $h_e(\boldsymbol{t})$.
A one-dimensional affine transform for statement S_k is defined by:

$$\phi_{s_k} = \begin{bmatrix} f_1 & \dots & f_{m_{S_k}} \end{bmatrix} (\boldsymbol{i}) + f_0, \quad f_i \in \mathbf{Z}$$

A multi-dimensional affine transformation for a statement can now be represented by a matrix with each row being an affine hyperplane/transform. If such a transformation matrix has full column rank, it completely specifies when and where an iteration executes (one-to-one mapping from source to target). The total number of rows in the matrix may be much larger as some special rows, *splitters*, may represent unfused loops at a level. Fig. 1 shows application of a transformation. Such transformations capture the fusion structure as well as compositions of permutation, reversal, relative shifting, and skewing transformations. This representation for transformations has been used by many researchers [6, 11, 12, 15], and directly fits with scattering functions that a code generator like CLooG [2] supports. Our problem is thus to find the the transformation matrices that are best for parallelism and locality.

3 Finding good transformations

3.1 Legality of tiling imperfectly-nested loops

Theorem 1. *Let ϕ_{s_i} be a one-dimensional affine transform for statement S_i. For $\{\phi_{s_1}$, ϕ_{s_2}, $\dots$, $\phi_{s_k}\}$ to be a legal (statement-wise) tiling hyperplane, the following should hold for each edge e from S_i to S_j:*

$$\phi_{s_j}(\boldsymbol{t}) - \phi_{s_i}(\boldsymbol{s}) \geq 0, \quad \mathcal{P}_e \tag{2}$$

Proof. Tiling of a statement's iteration space defined by a set of tiling hyperplanes is said to be legal if each tile can be executed atomically and a valid total ordering of the tiles can be constructed. This implies that there exists no two tiles such that they both influence each other. Let $\{\phi^1_{s_1}, \phi^1_{s_2}, \ldots, \phi^1_{s_k}\}$, $\{\phi^2_{s_1}, \phi^2_{s_2}, \ldots, \phi^2_{s_k}\}$ be two statement-wise 1-d affine transforms that satisfy (2). Consider a tile formed by aggregating a group of hyperplane instances along $\phi^1_{s_i}$ and $\phi^2_{s_i}$. Due to (2), for any dynamic dependence, the target iteration is mapped to the same hyperplane or a greater hyperplane than the source, i.e., the set of all iterations that are outside of the tile and are influenced by it always lie in the forward direction along one of the independent tiling dimensions (ϕ^1 and ϕ^2 in this case). Similarly, all iterations outside of a tile influencing it are either in that tile or in the backward direction along one or more of the hyperplanes. The above argument holds true for both intra- and inter-statement dependences. For inter-statement dependences, this leads to an interleaved execution of tiles of iteration spaces of each statement when code is generated from these mappings. Hence, $\{\phi^1_{s_1}, \phi^1_{s_2}, \ldots, \phi^1_{s_k}\}$, $\{\phi^2_{s_1}, \phi^2_{s_2}, \ldots, \phi^2_{s_k}\}$ represent rectangularly tilable loops in the transformed space. If such a tile is executed on a processor, communication would be needed only before and after its execution. From locality point of view, if such a tile is executed with the associated data fitting in a faster memory, reuse is exploited in multiple directions.□

The above condition was well-known for the case of a single-statement perfectly nested loops from the work of Irigoin and Triolet [14] (as $h^T.R \geq \mathbf{0}$). We have generalized it above for multiple iteration spaces with exact affine dependences with possibly different dimensionalities and imperfect nestings for statements.

Tiling at an arbitrary depth. Note that the legality condition as written in (2) is imposed on all dependences. However, if it is imposed only on dependences that have not been carried up to a certain depth, the independent ϕ's that satisfy the condition represent tiling hyperplanes at that depth, i.e., rectangular blocking (stripmine/interchange) at that level in the transformed program is legal.

Consider the perfectly nested version of 1-d Jacobi shown in Fig. 2(a). The discussion that follows also applies to the imperfectly nested version, but for convenience we first consider the perfectly nested version. We first describe solutions obtained by existing state-of-the-art approaches: Lim and Lam's affine partitioning [18, 19] and Griebl's space and time tiling with Forward Communication-Only (FCO) placement [13].

Lim et al. [19] define legal time partitions which have the same property of tiling hyperplanes as described above. Their algorithm obtains affine partitions that minimize the *order* of communication while maximizing the *degree* of parallelism. Equation (2) gives legality constraints $c_t \geq 0$, $c_t + c_i \geq 0$, and $c_t - c_i \geq 0$ corresponding to dependences $(1, 0)$, $(1, 1)$, and $(1, -1)$. There are infinitely many valid solutions with the

```
for (t=1; t<T; t++)
  for (i=2; i<N−1; i++)
    a[t,i] = 0.33∗(a[t−1,i] +
                   a[t−1,i−1] + a[t−1,i+1]);
```

(a) 1-d Jacobi: perfectly nested

```
for (t=1; t<T; t++)
  for (i=2; i<N−1; i++)
    S1: b[i] = 0.33∗(a[i−1]+ a[i]+a[i+1]);
  for (i=2; i<N−1; i++)
    S2: a[i] = b[i];
```

(b) 1-d Jacobi: imperfectly nested

Fig. 2. 1-d Jacobi

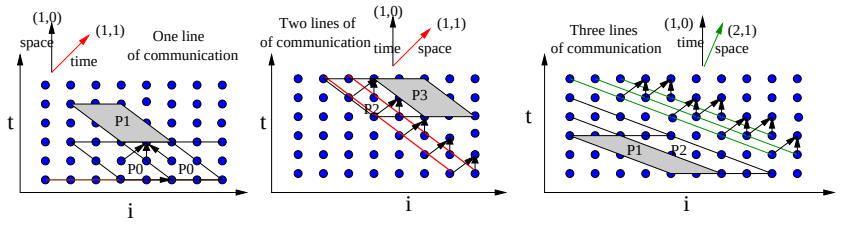


Fig. 3. Communication volume with different valid hyperplanes for perfectly nested 1-d Jacobi

same order complexity of synchronization, but with different communication volumes that may impact performance. Although it may seem that the volume may not affect performance, considering the fact that communication startup time on modern interconnects is significant, for higher dimensional problems such as n-d Jacobi, the ratio of communication to computation increases (proportional to tile size raised to $n-1$). Existing work on tiling [24, 25, 30] can find near communication-optimal tiles for perfectly nested loops with constant dependences, but cannot handle arbitrarily nested loops. For 1-d Jacobi, all solutions within the cone formed by vectors $(1,1)$ and $(1,-1)$ are valid tiling hyperplanes. For the imperfectly nested version of 1-d Jacobi, the valid cone is $(2,1)$ and $(2,-1)$ (discussed later). For imperfectly nested Jacobi, Lim's algorithm [19] finds two valid independent solutions without optimizing for any particular criterion. In particular, the solutions found by their algorithm (Algorithm A in [19]) are $(2,-1)$ and $(3,-1)$ which are clearly not the best tiling hyperplanes to minimize communication volume, though they do minimize the *order* of synchronization which is $O(N)$; in this case any valid hyperplane has $O(N)$ synchronization. Figure 3 shows that the required communication increases as the hyperplane gets more and more oblique. For a hyperplane with normal $(k,1)$, one would need $(k+1)T$ values from the neighboring tile.

Using Griebl's approach, we first find that only space tiling is enabled with Feautrier's schedule being $\theta(t,i)=t$, i.e., using $(1,0)$ as the scheduling hyperplane. With forward communication-only (FCO) placement, an allocation is found such that dependences have positive components along space dimensions thereby enabling tiling of the time dimension; this decreases the frequency of communication. In this case, time tiling is enabled with FCO placement along $(1,1)$. However, note that communication in the processor space occurs along $(1,1)$, i.e., two lines of the array are required. However, using $(1,0)$ and $(1,1)$ as tiling hyperplanes with $(1,0)$ as space and $(1,1)$ as inner time and a tile space schedule of $(2,1)$ leads to only one line of communication along $(1,0)$. Our algorithm finds such a solution. Below we develop a cost function for an affine transform that captures communication volume and reuse distance.

3.2 Cost Function

Consider the following affine form:

$$\delta_e(\boldsymbol{t}) = \phi_{s_i}(\boldsymbol{t}) - \phi_{s_j}(h_e(\boldsymbol{t})), \quad \boldsymbol{t} \in \mathcal{P}_e \tag{3}$$

The affine form $\delta_e(\boldsymbol{t})$ holds much significance. This function is the number of hyperplanes the dependence e traverses along the hyperplane normal. It gives us a measure

of the reuse distance if the hyperplane is used as time, i.e., if the hyperplanes are executed sequentially. Also, this function is a factor in the communication volume if the hyperplane is used to generate tiles for parallelization and used as a processor space dimension. An upper bound on this function means that the number of hyperplanes that would be communicated as a result of the dependence at the tile boundaries would not exceed this bound. We are particularly interested in whether this function can be reduced to a constant value or zero by choosing a suitable direction for ϕ: if possible, that particular dependence leads to constant or no communication for this hyperplane. Note that each δ_e is an affine function of the loop indices. The challenge is to use this function to obtain a suitable objective for optimization in the affine framework.

Challenges. The constraints obtained from (2) only guarantee legality of tiling (permutability). However, several problems are encountered when one tries to apply a performance factor to find a good tile shape out of the several possibilities. Farkas Lemma has been used by many approaches [10, 11, 13, 19] to eliminate loop variables from constraints by getting equivalent linear inequalities. The affine form in the loop variables is represented as a positive linear combination of the faces of the dependence polyhedron. When this is done, the coefficients of the loop variables on the left and right hand side are equated to eliminate the constraints of variables. This is done for each of the dependences, and the constraints obtained are aggregated. The resulting constraints are entirely in the coefficients of the tile mappings and Farkas multipliers. All Farkas multipliers can be eliminated, some by Gaussian elimination and the rest by Fourier-Motzkin elimination [19, 26]. However, an attempt to minimize communication volume ends up in an objective function involving both loop variables and hyperplane coefficients. For example, $\phi(\boldsymbol{t}) - \phi(h_e(\boldsymbol{t}))$ could be $c_1 i + (c_2 - c_3) j$, where $1 \leq i \leq N \wedge 1 \leq j \leq N \wedge i \leq j$. One ends up with such a form when a dependence is not uniform or for an inter-statement dependence, making it hard to construct an objective function involving only the unknown hyperplane coefficients.

3.3 Cost Function Bounding and Minimization

Theorem 2. *If all iteration spaces are bounded, there exists an affine form $v(\boldsymbol{p}) = \mathbf{u}.\boldsymbol{p} + w$ that bounds $\delta_e(\boldsymbol{t})$ for every dependence edge e:*

$$\begin{aligned} v(\boldsymbol{p}) \; - \; \left(\phi_{s_i}(\boldsymbol{t}) - \phi_{s_j}(h_e(\boldsymbol{t}))\right) \; &\geq \; 0, \;\; \boldsymbol{t} \in \mathcal{P}_e, \quad \forall e \in E \\ \textit{i.e.,} \qquad v(\boldsymbol{p}) \; - \; \delta_e(\boldsymbol{t}) \; &\geq \; 0, \;\; \boldsymbol{t} \in \mathcal{P}_e, \quad \forall e \in E \end{aligned} \tag{4}$$

Even if δ_e involves loop variables, one can find large enough constants in u that would be sufficient to bound $\delta_e(\boldsymbol{s})$. Note that the loop variables themselves are bounded by affine functions of the parameters, and hence the maximum value taken by $\delta_e(\boldsymbol{s})$ will be bounded by such an affine form. Also, since $v(\boldsymbol{p}) \geq \delta_e(\boldsymbol{s}) \geq 0$, v should increase with an increase in the structural parameters, i.e., the coordinates of u are positive. The reuse distance or communication volume for each dependence is bounded in this fashion by the same affine form. Such a bounding function was used by Feautrier [10] to find minimum latency schedules.

Now we apply Farkas Lemma to (4):

$$v(\boldsymbol{p}) - \delta_e(\boldsymbol{t}) \equiv \lambda_{e0} + \sum_{k=1}^{m_e} \lambda_{ek} \mathcal{P}_e^k, \qquad \lambda_{ek} \geq 0 \tag{5}$$

where $\mathcal{P}_e^k$ is a face of $\mathcal{P}_e$. The above is an identity and the coefficients of each of the loop indices in $\boldsymbol{i}$ and parameters in $\boldsymbol{p}$ on the left and right hand side can be gathered and equated. We now get linear inequalities entirely in coefficients of the affine mappings for all statements, components of row vector $\mathbf{u}$, and w. The above inequalities can at once be solved by finding a lexicographic minimal solution with $\boldsymbol{u}$ and w in the leading position, and the other variables following in any order. Let $\mathbf{u} = (u_1, u_2, \ldots u_k)$.

$$\text{minimize}_{\prec} \{u_1, u_2, \ldots, u_k, w, \ldots, c_i's, \ldots\} \tag{6}$$

Finding the lexicographic minimal solution for a system of linear inequalities is within the reach of the simplex algorithm and can be handled by the Parametric Integer Programming (PIP) software [8]. Since the structural parameters are quite large, we first want to minimize their coefficients. We do not lose the optimal solution since an optimal solution would have the smallest possible values for u's.

The solution gives a hyperplane for each statement. Note that the application of the Farkas Lemma to (4) is not required when a dependence is uniform, since the corresponding δ_e is independent of any loop variables. In such cases, we just have $w \geq \delta_e$.

3.4 Iteratively Finding Independent Solutions

Solving the ILP formulation in the previous section gives us a single solution to the coefficients of the best mappings for each statement. We need at least as many independent solutions as the dimensionality of the polytope associated with each statement. Hence, once a solution is found, we augment the ILP formulation with new constraints and obtain the next solution; the new constraints ensure linear independence with solutions already found. Let the rows of H_S represent the solutions found so far for a statement S. Then, the sub-space orthogonal to H_S [16, 21] is given by:

$$H_S^{\perp} = I - H_S^T \left(H_S H_S^T\right)^{-1} H_S \tag{7}$$

Note that $H_S^{\perp} . {H_S}^T = \mathbf{0}$, i.e., the rows of H_S are orthogonal to those of $H_S^{\perp}$. Let h_S^* be the next row (linear portion of the hyperplane) to be found for statement S. Let ${H^i}_S^{\perp}$ be a row of $H_S^{\perp}$. Then, any *one* of the inequalities given by $\forall i,\ {H^i}_S^{\perp} . \boldsymbol{h_S^*} > 0, {H^i}_S^{\perp} . \boldsymbol{h_S^*} < 0$ gives the necessary constraint to be added for statement S to ensure that h_S^* has a non-zero component in the sub-space orthogonal to H_S. This leads to a non-convex space, and ideally, all cases have to be tried and the best among those kept. When the number of statements is large, this leads to a combinatorial explosion. In such cases, we restrict ourselves to the sub-space of the orthogonal space where all the constraints are positive, i.e., the following constraints are added to the ILP formulation for linear independence:

$$\forall i, {H^i}_S^{\perp} . {h^*}_S \geq 0 \quad \wedge \quad \sum_i {H^i}_S^{\perp} h_S^* \geq 1 \tag{8}$$

By just considering a particular convex portion of the orthogonal sub-space, we discard solutions that usually involve loop reversals or combination of reversals with other transformations; however, we believe this does not make a difference in practice. The mappings found are independent on a per-statement basis. When there are statements with different dimensionalities, the number of such independent mappings found for each statement is equal to the number of outer loops it has. Hence, no more orthogonality constraints need be added for statements for which enough independent solutions have been found (the rest of the rows get automatically filled with zeros or linearly dependent rows). The number of rows in the transformation matrix is the same for each statement, and the depth of the deepest loop nest in the target code is the same as that of the source loop nest. Overall, a hierarchy of fully permutable loop nest sets is found, and a lower level in the hierarchy will not be obtained unless constraints corresponding to dependences that have been carried by the parent permutable set have been removed.

3.5 Communication and Locality Optimization Unified

The above algorithm finds both synchronization-free and pipelined parallelism. The best possible solution to (6) is with ($u = 0, w = 0$) and this happens when we find a hyperplane that has no dependence components along its normal, which is a fully parallel loop requiring no synchronization if it is at the outer level (*outer parallel*); it could be an inner parallel loop if some dependences were removed previously and so a synchronization is required after the loop is executed in parallel. Thus, in each of the steps where we find a new independent hyperplane, we end up first finding all synchronization-free hyperplanes; these are followed by a set of fully permutable hyperplanes that are tilable and pipelined parallel requiring constant boundary communication ($u = 0, w > 0$) w.r.t. the tile sizes. In the worst case, a hyperplane with $u > 0, w \geq 0$ results in long communication from non-constant dependences. It is important to note that the latter are pushed to the innermost level. By considering communication volume and its minimization, all degrees of parallelism are found in the order of their preference.

From the point of view of data locality, the hyperplanes that are used to scan the tile space are the same as the ones that scan points in a tile. Hence, data locality is optimized from two angles: (1) cache misses at tile boundaries are minimized for local execution (as cache misses at local tile boundaries are equivalent to communication along processor tile boundaries); (2) by reducing reuse distances, we increase the local cache tile sizes. The former is due to selection of good tile shapes and the latter by the right permutation of hyperplanes (implicit in the order in which we find them).

3.6 Space and Time in Transformed Iteration Space

By minimizing $\phi(\boldsymbol{t}) - \phi(\boldsymbol{s})$ as we find hyperplanes from outermost to innermost, we push dependence carrying to inner loops and also ensure that loops do not have negative dependence components (to the extent possible) so that target loops can be tiled. Once this is done, if the outer loops are used as space (any number desired, say k), and the rest are used as time (at least one time loop is required unless all loops are synchronization-free parallel), communication in the processor space is optimized as the outer space

Input Generalized dependence graph $G = (V, E)$ (includes dependence polyhedra $\mathcal{P}_e$, $e \in E$)

1: S_{max}: statement with maximum domain dimensionality
2: **for** each dependence $e \in E$ **do**
3: Build legality constraints: apply Farkas Lemma on $\phi(\boldsymbol{t}) - \phi(h_e(\boldsymbol{t})) \geq 0$ under $\boldsymbol{t} \in \mathcal{P}_e$, and eliminate all Farkas multipliers
4: Build communication volume/reuse distance bounding constraints: apply Farkas Lemma to $v(\boldsymbol{p}) - (\phi(\boldsymbol{t}) - \phi(h_e(\boldsymbol{t}))) \geq 0$ under $\mathcal{P}_e$, and eliminate all Farkas multipliers
5: Aggregate constraints from both into $C_e(i)$
6: **end for**
7: **repeat**
8: $C = \emptyset$
9: **for** each dependence edge $e \in E$ **do**
10: $C \leftarrow C \cup C_e(i)$
11: **end for**
12: Compute lexicographic minimal solution with $u's$ coefficients in the leading position followed by w to iteratively find independent solutions to C (orthogonality constraints are added as each solution is found)
13: **if** no solutions were found **then**
14: Cut dependences between two strongly-connected components in the GDG and insert the appropriate *splitter* in the transformation matrices of the statements
15: **end if**
16: Compute E_c: dependences carried by solutions of Step 12/14
17: $E \leftarrow E - E_c$; update the GDG (V, E)
18: **until** $H^{\perp}_{S_{max}} = \mathbf{0}$ and $E = \emptyset$

Output A transformation matrix for each statement (with the same number of rows)

Fig. 4. Overall algorithm

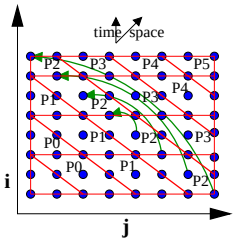
loops are the k best ones. Whenever the loops can be tiled, they result in coarse-grained parallelism as well as better reuse within a tile.

3.7 Fusion

The algorithm described in the previous section can also enable fusion across multiple iteration spaces that are weakly connected, as in sequences of producer-consumer loops. Solving for hyperplanes for multiple statements leads to a schedule for each statement such that all statements in question are *finely* interleaved: this is indeed fusion. This generalization of fusion is same as the one proposed in [6, 12], and naturally integrates into our algorithm. A detailed description can be found in an extended report [3].

Summary. The overall algorithm is summarized in Fig. 4. It can be viewed as transforming to a tree of permutable loop nest sets/bands — each node of the tree is a good permutable loop nest set. Step 12 finds such a band of permutable loops. If all loops are tilable, there is just one node containing all the loops that are permutable. On the other extreme, if no loops are tilable, each node of the tree has just one loop and no tiling is

```
for (i=0; i<N; i++)
  for (j=2; j<N; j++)
    a[i,j] = a[j,i]+a[i,j-1];
```



$$a[i',j'] \rightarrow a[i,j-1]$$
$$h_1 : i' = i, j' = j-1;$$
$$\mathcal{P}_1 : 2 \leq j \leq N, 1 \leq i \leq N$$
$$a[i',j'] \rightarrow a[j,i]$$
$$h_2 : i' = j, j' = i;$$
$$\mathcal{P}_2 : 2 \leq j \leq N,\ 1 \leq i \leq N, i-j \geq 1$$
$$a[j',i'] \rightarrow a[i,j]$$
$$h_3 : j' = i, i' = j$$
$$\mathcal{P}_3 : 2 \leq j \leq N,\ 1 \leq i \leq N,\ i-j \geq 1$$

Fig. 5. An example with non-uniform dependences

possible. At least two hyperplanes should be found at any level (without dependence removal/cutting) to enable tiling. Dependences from previously found solutions are thus not removed unless they have to be (step 17) to allow the next permutable band to be found, and so on. Hence, partially tilable or untilable input is handled. Loops in each node of the target tree can be stripmined/interchanged when there are at least two of them; however, it is illegal to move a stripmined loop across different levels in the tree.

4 Example

Figure 5 shows an example from the literature [7] with affine non-uniform dependences, together with the corresponding dependence polyhedra (the source iteration vector has been eliminated). For the first dependence, the tiling legality constraint is

$$c_i i + c_j j - c_i i - c_j (j-1) \geq 0 \quad \Rightarrow \quad c_j \geq 0$$

Since this is a constant dependence, the volume bounding constraint gives $w - c_j \geq 0$. For the second dependence, the tiling legality constraint is

$$(c_i i + c_j j) - (c_i j + c_j i) \ \geq \ 0$$

Applying Farkas Lemma (with $\mathcal{P}_2$), we have:

$$\begin{aligned}(c_i - c_j)i + (c_j - c_i)j \ \equiv \ & \lambda_0 + \lambda_1(N-i) + \lambda_2(N-j) \\ & + \lambda_3(i-j-1) + \lambda_4(i-1) + \lambda_5(j-1) \\ & \lambda_0, \lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5 \geq 0\end{aligned} \tag{9}$$

Equating LHS and RHS coefficients for i, j, N and the constants in (9), and eliminating Farkas multipliers through Fourier-Motzkin elimination, we obtain $c_i - c_j \geq 0$.
The volume bounding constraint is

$$u_1 N + w - (c_i j + c_j i - c_i i - c_j j) \geq 0$$

A similar application of Farkas Lemma, and elimination of the multipliers yields $u_1 \geq 0$, $u_1 - c_i + c_j \geq 0$, and $3u_1 + w - c_i + c_j \geq 0$. Due to the symmetry with respect to i

and j, the third dependence does not lead to any new constraints. Aggregating legality and volume bounding constraints for all dependences, we get the formulation:

$$\text{minimize}_{\prec}\ (u_1, w, c_i, c_j)$$

$$\begin{array}{rll} \text{subject to:} & c_j \geq 0 & w - c_j \geq 0 \\ & c_i - c_j \geq 0 & u_1 \geq 0 \\ & u_1 - c_i + c_j \geq 0 & 3u_1 + w - c_i + c_j \geq 0 \end{array}$$

The lexicographic minimal solution for the vector (u_1, w, c_i, c_j) is $(0, 1, 1, 1)$ (the zero vector is a trivial solution and is avoided). Hence, we get $c_i = c_j = 1$. Note that $c_i = 1$ and $c_j = 0$ is not obtained even though it is a valid tiling hyperplane as it involves more communication: it requires u_1 to be positive.

The next solution is forced to have a positive component in the subspace orthogonal to $(1, 1)$ given by (7) as (1,-1). This leads to the addition of the constraint $c_i - c_j \geq 1$ or $c_i - c_j \leq -1$ to the existing formulation. Adding $c_i - c_j \geq 1$ to (10), the lexicographic minimal solution is (1, 0, 1, 0), i.e., $u_1 = 1, w = 0, c_i = 1, c_j = 0$ ($u_1 = 0$ is no longer valid). Hence, $(1, 1)$ and $(1, 0)$ are the tiling hyperplanes obtained. (1,1) is used as space with one line of communication between processors, and the hyperplane (1,0) is used as time in a tile. The outer tile schedule is (2,1) (= (1,1) + (1,0)).

This transformation is in contrast to other approaches based on schedules which obtain a schedule and then the rest of the transformation matrix. Feautrier's greedy heuristic gives the schedule $\theta(i, j) = 2i+j-3$ which carries all dependences. However, using this as either space or time does not lead to communication or locality optimization. The (2,1) hyperplane has non-constant communication along it. In fact, the only hyperplane that has constant communication along it is (1,1). This is the best hyperplane to be used as a space loop if the nest is to be parallelized, and is the first solution that our algorithm finds. The (1,0) hyperplane is used as time leading to a solution with one degree of pipelined parallelism with one line per tile of near-neighbor communication (along (1,1)) as shown in Fig. 4. Hence, a good schedule that tries to carry all dependences (or as many as possible) is not necessarily a good loop for the transformed iteration space.

5 Implementation and Preliminary Results

We have implemented our transformation framework using PipLib 1.3.3 [8]. Our tool takes as input dependence information (dependence polyhedra and h-transformations) from LooPo's [20] dependence tester and generates statement-wise affine transformations. Though in theory the approach, relying on integer programming, has worst-case exponential time complexity, we observe that it runs extremely fast in practice. The transformations generated are provided to CLooG [2] as scattering functions. The goal is to obtain tiled shared memory parallel code, for example, OpenMP code for multi-core architectures. Table 1 summarizes the performance of transformed codes. The state-of-the-art from the research community is represented by [13, 17–19], while ICC 10.1 with '-fast -parallel' was used as the native compiler. The results were obtained on an Intel Core 2 Quad (Q6600 2.4 GHz). Due to space constraints, detailed experimental evaluation can be found elsewhere [4].

Table 1. Initial results: speedup over state-of-the-art research

Benchmark	Single core speedup		Multi-core speedup (4 cores)	
	over native compiler	over state-of-the-art research	over native compiler	over state-of-the-art research
1-d Jacobi (imperfect nest)	5.23x	2.1x	20x	1.7x
2-d FDTD	3.7x	3.1x	7.4x	2.5x
3-d Gauss-Seidel	1.6x	1.1x	4.5x	1.5x
LU decomposition	5.6x	5.7x	14x	3.8x
Matrix Vec Transpose	9.3x	5.5x	13x	7x

6 Related Work

Iteration space tiling [14, 24, 29] is a standard approach for aggregating a set of loop iterations into tiles, with each tile being executed atomically. In addition, researchers have considered the problem of selecting tile shape and size to minimize communication, improve locality or minimize finish time [24, 30]. These works are restricted to a single perfectly nested loop nest with uniform dependences.

Loop parallelization has been studied extensively. The reader is referred to the survey of Boulet et al. [5] for a detailed summary of older parallelization algorithms which accepted restricted input and/or are based on weaker dependence abstractions than exact polyhedral dependences. Scheduling with affine functions using faces of the polytope by application of the Farkas algorithm was first proposed by Feautrier [10]. Feautrier explored various possible approaches to obtain good affine schedules that minimize latency. The schedules carry all dependences and so all the inner loops can be parallel. However, transforming to permutable loops that are amenable to tiling was not addressed. Though schedules yield inner parallel loops, the time loops cannot be tiled unless communication in the space loops is in the forward direction (dependences have positive components along all dimensions). Several works [6, 13, 22] make use of such schedules. Overall, Feautrier's classic works [10, 11] are geared towards finding maximum fine-grained parallelism as opposed to tilability for coarse-grained parallelization with minimized communication and better locality.

Lim and Lam [18, 19] proposed an affine partitioning framework that identifies outer parallel loops (communication-free space partitions) and pipelined parallel permutable loops to maximize the degree of parallelism and minimize the order of synchronization. They employ the same machinery for blocking [17]. Several (infinitely many) solutions equivalent in terms of the criterion they optimize for result from their algorithm, and these significantly differ in communication cost and locality; no metric is provided to differentiate between these solutions. As seen in Sec. 3, without a cost function, the solutions obtained even for the simplest input are not satisfactory.

Ahmed et al. [1] proposed a framework for locality optimization of imperfectly nested loops for sequential execution. The approach embeds each statement into a product space, which is then transformed for locality. Their heuristic sets reuse distances in the target space for some dependences to zero (or a constant) to obtain coefficients of the embedding/transformation matrix. However, there is no concrete procedure to determine choice of the dependences and the number.

Griebl [13] presents an integrated framework for optimizing locality and parallelism with space and time tiling. Griebl's approach enables time tiling by using a forward communication-only placement with an existing schedule. As described in Sec. 3, using schedules as time loops may not lead to communication or locality-optimized solutions.

Cohen et al. [6] and Girbal et al. [12] developed a framework to compose sequences of transformations semi-automatically. Transformations are applied automatically, but specified manually by an expert. Pouchet et al. [22] searches the space of transformations (one-dimensional schedules) to find good ones through iterative optimization by employing performance counters. On the other hand, our approach is fully automatic. However, some empirical and iterative optimization is required to choose transforms that work best in practice. This is true when several fusion choices exist, or optimal tile sizes and unroll factors have to be determined. A combination of our algorithm and empirical search in a smaller space is an interesting approach to pursue.

7 Conclusions

We present a single framework that addresses automatic parallelization and data locality optimization in the polyhedral model. The proposed algorithm finds communication-minimized tiling hyperplanes for parallelization of a sequence of arbitrarily nested loops. The same hyperplanes also minimize reuse distances and improve data locality. The approach also enables fusion in the presence of producing-consuming loops. To the best of our knowledge, this work is the first to propose a practical cost model to drive automatic transformation in the polyhedral model. The framework has been implemented in a fully-automatic tool for transforming C/Fortran code using the LooPo infrastructure and CLooG. Preliminary experiments show very promising results.

Acknowledgments

We would like to thank Martin Griebl and his team (FMI, Universität Passau, Germany) for the LooPo infrastructure. We would also like to thank Cédric Bastoul (Paris-Sud XI University) and all other contributors of CLooG and PipLib. This work was supported in part by a State of Ohio Development Fund and the National Science Foundation through grants 0121676, 0121706, 0403342, 0508245, 0509442, 0509467 and 0541409.

References

1. N. Ahmed, N. Mateev, and K. Pingali. Synthesizing transformations for locality enhancement of imperfectly-nested loop nests. *IJPP*, 29(5), October 2001.
2. C. Bastoul. Code generation in the polyhedral model is easier than you think. In *IEEE PACT*, pages 7–16, Sept. 2004.
3. U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan. Affine transformations for communication minimal parallelization and locality optimization for arbitrarily-nested loop sequences. Technical Report OSU-CISRC-5/07-TR43, The Ohio State University, May 2007.

4. U. Bondhugula, J. Ramanujam, and P. Sadayappan. PLuTo: A practical and fully automatic polyhedral parallelizer and locality optimizer. Technical Report OSU-CISRC-5/07-TR70, The Ohio State University, Oct. 2007.
5. P. Boulet, A. Darte, G.-A. Silber, and F. Vivien. Loop parallelization algorithms: From parallelism extraction to code generation. *Parallel Computing*, 24(3–4):421–444, 1998.
6. A. Cohen, S. Girbal, D. Parello, M. Sigler, O. Temam, and N. Vasilache. Facilitating the search for compositions of program transformations. In *ICS*, pages 151–160, June 2005.
7. A. Darte and F. Vivien. Optimal fine and medium grain parallelism detection in polyhedral reduced dependence graphs. *IJPP*, 25(6):447–496, Dec. 1997.
8. P. Feautrier. Parametric integer programming. *Operationnelle/Operations Research*, 22(3):243–268, 1988.
9. P. Feautrier. Dataflow analysis of array and scalar references. *IJPP*, 20(1):23–53, 1991.
10. P. Feautrier. Some efficient solutions to the affine scheduling problem: I. one-dimensional time. *IJPP*, 21(5):313–348, 1992.
11. P. Feautrier. Some efficient solutions to the affine scheduling problem. part II. multidimensional time. *IJPP*, 21(6):389–420, 1992.
12. S. Girbal, N. Vasilache, C. Bastoul, A. Cohen, D. Parello, M. Sigler, and O. Temam. Semi-automatic composition of loop transformations for deep parallelism and memory hierarchies. *IJPP*, 34(3):261–317, June 2006.
13. M. Griebl. *Automatic Parallelization of Loop Programs for Distributed Memory Architectures*. FMI, University of Passau, 2004. Habilitation Thesis.
14. F. Irigoin and R. Triolet. Supernode partitioning. In *POPL*, pages 319–329, 1988.
15. W. Kelly and W. Pugh. A unifying framework for iteration reordering transformations. Technical Report CS-TR-3430, University of Maryland, College Park, 1995.
16. W. Li and K. Pingali. A singular loop transformation framework based on non-singular matrices. *IJPP*, 22(2):183–205, 1994.
17. A. Lim, S. Liao, and M. Lam. Blocking and array contraction across arbitrarily nested loops using affine partitioning. In *ACM SIGPLAN PPoPP*, pages 103–112, 2001.
18. A. W. Lim, G. I. Cheong, and M. S. Lam. An affine partitioning algorithm to maximize parallelism and minimize communication. In *ACM ICS*, pages 228–237, 1999.
19. A. W. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine partitions. *Parallel Computing*, 24(3-4):445–475, 1998.
20. LooPo - Loop parallelization in the polytope model. http://www.fmi.uni-passau.de/loopo.
21. R. Penrose. A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, 51:406–413, 1955.
22. L.-N. Pouchet, C. Bastoul, A. Cohen, and N. Vasilache. Iterative optimization in the polyhedral model: Part I, one-dimensional time. In *ACM CGO*, Mar. 2007.
23. F. Quilleré, S. V. Rajopadhye, and D. Wilde. Generation of efficient nested loops from polyhedra. *IJPP*, 28(5):469–498, 2000.
24. J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for multicomputers. *Journal of Parallel and Distributed Computing*, 16(2):108–230, 1992.
25. R. Schreiber and J. Dongarra. Automatic blocking of nested loops. Technical report, University of Tennessee, Knoxville, TN, Aug. 1990.
26. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1987.
27. N. Vasilache, C. Bastoul, and A. Cohen. Polyhedral code generation in the real world. In *CC*, pages 185–201, Mar. 2006.
28. N. Vasilache, C. Bastoul, S. Girbal, and A. Cohen. Violated dependence analysis. In *ACM ICS*, June 2006.
29. M. Wolf and M. S. Lam. A data locality optimizing algorithm. In *PLDI*, pages 30–44, 1991.
30. J. Xue. Communication-minimal tiling of uniform dependence loops. *JPDC*, 42(1):42–59, 1997.