# PLUTO+: Near-Complete Modeling of Affine Transformations for Parallelism and Locality

Aravind Acharya    Uday Bondhugula

Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560012
INDIA
{aravind.acharya, uday}@csa.iisc.ernet.in

## Abstract

Affine transformations have proven to be very powerful for loop restructuring due to their ability to model a very wide range of transformations. A single multi-dimensional affine function can represent a long and complex sequence of simpler transformations. Existing affine transformation frameworks like the Pluto algorithm, that include a cost function for modern multicore architectures where coarse-grained parallelism and locality are crucial, consider only a sub-space of transformations to avoid a combinatorial explosion in finding the transformations. The ensuing practical trade-offs lead to the exclusion of certain useful transformations, in particular, transformation compositions involving loop reversals and loop skewing by negative factors. In this paper, we propose an approach to address this limitation by modeling a much larger space of affine transformations in conjunction with the Pluto algorithm's cost function. We perform an experimental evaluation of both, the effect on compilation time, and performance of generated codes. The evaluation shows that our new framework, Pluto+, provides no degradation in performance in any of the Polybench benchmarks. For Lattice Boltzmann Method (LBM) codes with periodic boundary conditions, it provides a mean speedup of 1.33× over Pluto. We also show that Pluto+ does not increase compile times significantly. Experimental results on Polybench show that Pluto+ increases overall polyhedral source-to-source optimization time only by 15%. In cases where it improves execution time significantly, it increased polyhedral optimization time only by 2.04×.

***Categories and Subject Descriptors*** D.3.4 [*Programming Languages*]: Processors—Compilers, Optimization, Code generation

***Keywords*** Affine transformations, polyhedral model, automatic parallelization, tiling, affine scheduling, stencil computations

## 1. Introduction

Affine transformation frameworks for loop optimization have known to be powerful due to their ability to model a wide variety of loop reordering transformations [1]. Affine transformations preserve the collinearity of points as well as the ratio of distances between points lying on a line. This makes the problem of generating code, after application of such transformations, tractable. The polyhedral compiler framework employs affine transformations for execution reordering. The transformations are typically applied on integer points in a union of convex polyhedra. A number of works have studied the problem of code generation under affine transformations of such integer sets [3, 7, 38] and code generators like Cloog [10], Omega+ [7], and ISL [38] exist.

Dependence analysis, automatic transformation, and code generation are the three key stages of an automatic polyhedral optimizer. A number of model-driven automatic transformation algorithms for parallelization exist in the literature. Among algorithms that work for the entire generality of the polyhedral framework, there are those of Feautrier [14, 15], Lim and Lam [24, 26], Griebl [17], and Bondhugula et al. [4, 5]. The Pluto algorithm based on [4, 5] is the most recent among these, and has been shown to be suitable for architectures where extracting coarse-grained parallelism and locality are crucial — prominently modern general-purpose multicore processors.

The Pluto algorithm employs an objective function based on minimization of dependence distances [4]. The objective function makes certain practical trade-offs to avoid a combinatorial explosion in determining the transformations. The trade-offs restrict the space of transformations modeled to a sub-set of all valid ones. Although this does not affect the correctness of the algorithm and a solution is guaranteed to be found (since the identity transformation corresponding to the original schedule still remains in its space), transformations crucial for certain dependence patterns and application domains are excluded. In particular, the excluded transformations are those that involve a negative coefficient in the affine transformations. Negative coefficients in affine functions capture loop reversal, loop skewing by negative factors, and more importantly, those transformations which include the former (reversal and negative skewing) as one or more components in a potentially long composed sequence of simpler transformations.

The domains that are affected include, but are not limited to, those that have symmetric dependence patterns, those of stencil computations, Lattice Boltzmann Method (LBM) simulations, and other computational techniques defined over periodic data domains. Although loop reversal or skewing by negative factors are just some among several well-known unimodular or affine transformations, they can be composed in numerous ways with other transformations in a sequence to form a compound transformation that may also enable loop tiling and fusion for example. All such compound transformations currently end up being excluded from Pluto's space. This paper addresses this limitation and pro-

*PPoPP'15*, February 7–11, 2015, San Francisco, CA, USA
ACM 978-1-4503-3205-7/15/02
http://dx.doi.org/10.1145/2688500.2688512

poses an approach to model a larger space of transformations than Pluto, while using the same objective function as the state-of-the-art. Through experimental evaluation, we demonstrate that (1) the enlarged transformation space includes transformations that provide a very significant improvement in parallel performance for a number of benchmarks, and (2) the framework does not pose a significant compilation time issue.

The rest of this paper is organized as follows. Section 2 provides detail on the problem domains and patterns that motivate our work. Section 3 describes our new framework in detail. Experimental evaluation is presented in Section 4. Related work and conclusions are presented in Section 5 and Section 6 respectively.

## 2. Motivation

There are a number of situations when a transformation that involves a reversal is desired. In this section, we discuss these domains and computation patterns. The examples presented here are representative and very simple for explanation.

### 2.1 Index Sets and Transformations

We first introduce some basic terminology for the purpose of this section. Complete notation will be introduced in the next section. In the polyhedral compiler framework, iteration spaces of statements surrounded by loops are modeled by integer sets called index sets. Let $I_S$ be the index set of a statement $S$. A simple example is given by:

$$\mathtt{I_S} = \{[\mathtt{i},\mathtt{j}] : 0 \leq \mathtt{i},\mathtt{j} \leq \mathtt{N}-1\}.$$

Here, $i$ and $j$ correspond to the original loop dimensions of $S$ from which a polyhedral representation was extracted, and $N$ is a program parameter. Then, an example of an affine transformation on $I_S$ is given by:

$$\mathtt{T_{I_S}}(\mathtt{i},\mathtt{j}) = (\mathtt{i}-\mathtt{j}+\mathtt{N}, \mathtt{i}+\mathtt{j}+1)$$

Both $i-j+N$ and $i+j+1$ are one-dimensional affine functions of $i$, $j$, and $N$. Formally, an affine transformation, $T$, for a $k$-dimensional vector, $\vec{i} \in I_S$ is one that can be expressed in the form:

$$T\left(\vec{i}\right) = M \cdot \vec{i} + \vec{m_0}$$

where $M$ is a $n \times k$ matrix and $m_0$ is an $n$-dimensional vector. For all purposes in this paper, all elements of these entities are integers, i.e., $M \in Z^{n\times k}$, $m_0 \in Z^n$, $\vec{i} \in Z^k$.

Consider the two statements in Figure 2a, $S_1$ and $S_2$, with index sets:

$$\mathtt{I_{S_1}} = \{[\mathtt{i}] : 0 \leq \mathtt{i} \leq \mathtt{N}-1\}, \quad \mathtt{I_{S_2}} = \{[\mathtt{i}] : 0 \leq \mathtt{i} \leq \mathtt{N}-1\}.$$

The schedule that corresponds to the original execution order is given by:

$$\mathtt{T_{S_1}}(\mathtt{i}) = (0,\mathtt{i}), \quad \mathtt{T_{S_2}}(\mathtt{i}) = (1,\mathtt{i}).$$

A transformation that exposes a parallel loop outside while fusing and exploiting locality inside involves a reversal for $S_2$, and is given by:

$$\mathtt{T_{S_1}}(\mathtt{i}) = (\mathtt{i},0), \quad \mathtt{T_{S_2}}(\mathtt{i}) = (\mathtt{N}-\mathtt{i}-1,1).$$

The transformed code is shown in Figure 2b.

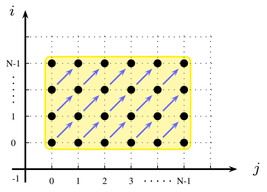
### 2.2 Skewing by Negative Factors for Communication-Free Parallelization

Consider the code in Figure 1. It has a single read-after-write (RAW) dependence represented by the distance vector (1,1).

The mapping $T(i,j) = (i-j,j)$ leads to a loop nest where the outer loop can be marked parallel while in the original code, the outer loop cannot be marked parallel. $\theta(i,j) = i$ represents a valid scheduling but is not useful in any way for performance by itself unless a suitable placement is found for it.

```
for (i=0; i<N; i++) {
  for (j=0; j<N; j++) {
    A[i+1][j+1] = f(A[i][j]);
  }
}
```

(a) Code



(b) Visualization

Figure 1: Example with dependence (1,1)

### 2.3 Symmetric Dependences

Figures 2 and 3 show two other patterns where the dependences are symmetric along a particular direction. Such examples were also studied in the synthesis of systolic arrays [9, 41] where researchers used folding or reflections to make dependences uniform or long wires shorter. A more compiler or code generation -centric approach is one where an index set splitting is applied to cut the domain vertically into two halves [6], and reverse and shift the halves in a way illustrated for periodic stencils in the next sub-section. Such cuts or index set splitting are automatically generated by existing work [6], and an implementation is available in Pluto.

```
for (i=0; i<N; i++){
   b[i] = f(a[i]);
}
for (i=0; i<N; i++){
   c[i] = f(b[N-1-i]);
}
```

(a) Code

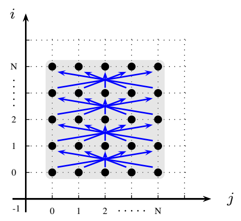
```
for (i=0; i<N; i++){
   b[i] = f(a[i]);
   c[N-1-i] = f(b[i]);
}
```

(b) Optimized

Figure 2: Symmetric consumer

```
for (i=0; i<=N-1; i++) {
  for (j=0; j<=N-1; j++) {
    a[i+1][j] = f(a[i][N-j-1]);
  }
}
```

(a) Code



(b) Visualization

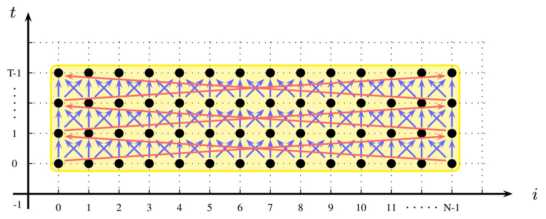
Figure 3: Symmetric dependences

### 2.4 Stencils on Periodic Domains

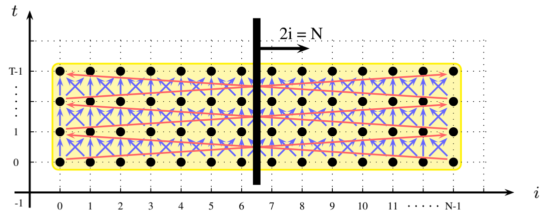
Stencils on periodic domains have long dependences in both directions along a dimension, as shown with red arrows in Figure 4b. The corresponding code is shown in Figure 4a. Such long dependences make tiling all loops invalid. Osheim et al [28] presented an approach to tile periodic stencils based on folding techniques [9, 41]. Bondhugula et al. [6] recently proposed an index set splitting technique that uses a hyperplane to cut close to the mid-points of all long dependences (Figure 4c). It opens the possibility of tiling through application of separate transformations on the split index sets to make dependences shorter. The sequence of transformations are shown in Figure 4g. However, these separate transformations involve reversals as one in the sequence (Figure 4d). The sequence of transformations enable existing time tiling

```
for (t=0; t <T−1; t++) {
  for (i=0; i<N; i++) {
    A[(t+1)%2][i] = (( i+1==N?A[t%2][0]:A[t%2][i+1])
        + 2.0∗A[t%2][i] + (i==0?A[t%2][N−1]:A[t%2][i−1]))/4.0;
  }
}
```

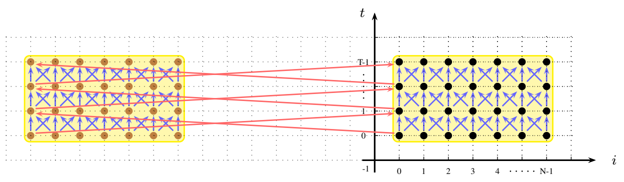
(a) A stencil on a periodic domain



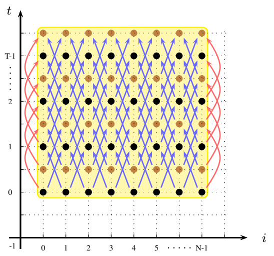
(b) Visualization for (a)



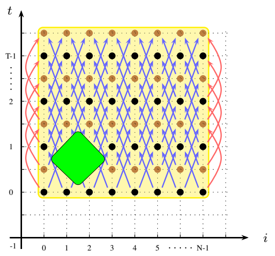
(c) Index Set Splitting



(d) Reversal of $S^+$: $T_{S^+}(t,i) \rightarrow (t,-i)$



(e) Parametric shift (all dependences have become short)



(f) Tiling: skewing to make dependences non-negative

(a) ISS: $I_{S^-} = I_S \cap \{2i \leq N-1\}, \quad S^+ = I_S \cap \{2i \geq N\}$
(b) Reversal: $T_{I_{S^-}}(t,i) = (t,i), \quad T_{I_{S^+}}(t,i) = (t,-i)$
(c) Parametric shift: $T_{I_{S^-}}(t,i) = (t,i), \quad T_{I_{S^+}}(t,i) = (t,-i+N)$
(d) Diamond tiling [2, 34]:

$$\begin{aligned} T_{I_S^-}(t,i) &= (t+i, t-i) \\ T_{I_S^+}(t,i) &= (t-i+N, t+i-N) \end{aligned}$$

(g) Transformations

Figure 4: A sequence of transformations that allows tiling for periodic stencils (with wrap-around boundary dependences)

techniques [2, 5, 34, 40] resulting in code that has been shown to provide dramatic improvement in performance including on a SPEC benchmark (*swim* from SPECFP2000) [6]. In addition, Lattice Boltzmann Method (LBM) codes share similarities with stencils, and are thus a very significant class of codes that benefit from the approach we will propose.

## 3. Expanding the Space of Transformations Modeled

In this section, we present our key contribution – the modeling of a larger, and nearly the complete space of affine transformations.

### 3.1 Notation

The data dependence graph for polyhedral optimization has a precise characterization of the dependence edges. Each dependence edge is a relation between source and target iterations. Although, in some works, it is represented as a conjunction of constraints called the dependence polyhedron, it can be viewed as a relation between source and target iterations involving affine constraints as well as existential quantifiers. Let $D_e$ be the relation associated with an edge $e$ in the data dependence graph. Then, an iteration $\vec{s}$ of statement $S_i$ depends on $\vec{t}$ of $S_j$ through edge $e$ when $\langle \vec{s} \rightarrow \vec{t} \rangle \in D_e$.

The set of iterations or statement instances to be executed for a statement $S$ is the domain or index set of $S$, and is denoted by $I_S$. Let $\vec{i_S}$ be an iteration vector of $S$, i.e., $\vec{i_S} \in I_S$, and let $m_S$ be its dimensionality. $\vec{i_S}$ has components corresponding to loops surrounding $S$ from outermost to innermost. Let $m_p$ be the number of global program parameters, i.e., symbols appearing in the program (typically representing problem sizes), and $\vec{p}$ be the vector of those program parameters. Then, a one-dimensional affine transformation for $S$ is given by the function:

$$\begin{aligned} \phi_S\left(\vec{i_S}\right) &= (c_1\ c_2\ \ldots\ c_{m_S}) \cdot \left(\vec{i_S}\right) \\ &\quad + \left(d_1\ d_2\ \ldots\ d_{m_p}\right) \cdot (\vec{p}) + c_0, \\ &\quad c_0, c_1, \ldots, c_{m_S}, d_1, d_2, \ldots, d_{m_p} \in \mathbb{Z} \end{aligned} \quad (1)$$

Each statement thus has its own set of $c_i$ and $d_i$ coefficients. The $c_i$'s, $1 \leq i \leq m_S$ correspond to dimensions of the index set of the statement and we will refer to them as the statement's *dimension coefficients*. The $d_i^S$, $1 \leq i \leq m_p$ correspond to parameters and thus represent parametric shifts. Finally, $c_0^S$ represents a constant shift. As an example, for the transformation for the first dimension of $S_2$ in Fig. 4e, the $c_i$'s, $d_i$'s, and $c_0$ would be:

$$\phi_{S_2}\begin{pmatrix} t \\ i \end{pmatrix} = (0,-1) \cdot \begin{pmatrix} t \\ i \end{pmatrix} + (0,1) \cdot \begin{pmatrix} T \\ N \end{pmatrix} + 0$$

The $\phi$ function can also be viewed as a hyperplane, and in the rest of this paper we refer to it as hyperplane or a transformation at a particular level. For convenience, at some places we will drop the superscript $S$ from $c_i$'s and $d_i$'s.

### 3.2 Background and Challenges

The Pluto algorithm [4] iteratively finds one-dimensional affine transformations, or hyperplanes, starting from outermost to the innermost while looking for tilable bands, i.e., $\phi$s satisfying the following constraint for all unsatisfied dependence relations $\langle \vec{s} \rightarrow \vec{t} \rangle \in D_e$:

$$\phi_{S_j}(\vec{t}) - \phi_{S_i}(\vec{s}) \geq 0. \tag{2}$$

The objective function it uses is that of reducing dependence distances using a bounding function. Intuitively, this reduces a factor in the reuse distances, and in cache misses after synchronization or communication volume in the case of distributed-memory. All dependence distances are bounded in the following way:

$$\phi_{s_j}(\vec{t}) - \phi_{s_i}(\vec{s}) \quad \leq \quad \vec{u}\cdot\vec{p} + w, \;\; \langle \vec{s}, \vec{t} \rangle \in D_e \tag{3}$$

The bounding function for the dependence distances is $\vec{u}.\vec{p} + w$, and the coefficients of $\vec{u}$ and the constant $w$ are then minimized, in order of decreasing priority, by finding the lexicographic minimum:

$$\text{lexmin}\left(\vec{u}, w, \ldots, c_i^S, d_i^S, \ldots\right). \tag{4}$$

The lexicographic minimum objective, *lexmin* for an ILP formulation was a special minimization objective first introduced by Feautrier [13] for purposes of program analysis. It is a slight variation of a typical linear objective function. The objective finds the lexicographically smallest vector with respect to the order of variables specified in the ILP. For (4), the best solution of $\vec{u} = 0, w = 0$ corresponds to a parallel loop.

Note that (2) and (3) both have a trivial zero vector solution, and getting rid of it is non-trivial if the dimension coefficients of $\phi^S$, the $c_i$'s, $i \geq 1$, are allowed to be negative. Alternatively, if we trade-off expressiveness for complexity and assume that all $c_i \geq 0$, the zero solution is avoided easily with:

$$\sum_{i=1}^{m_S} c_i \geq 1.$$

Pluto currently makes the above trade-off [5]. On the other hand, when negative coefficients are allowed, the removal of the zero solution from the full space, leads to a union of a large number of convex spaces. For example, for a 3-d statement, it leads to eight sub-spaces. For multiple statements, one would get a product of the number of these sub-spaces across all statements, leading to a combinatorial explosion. We now propose a scalable and compact approach to exclude the zero solution.

## 3.3 Excluding the Zero Solution

Intuitively, the challenge involved here is that removing a single point from the "middle" of a polyhedral space gives rise to a large disjunction of several polyhedral spaces, i.e., several different cases to consider. As explained earlier, the possibilities grow exponentially with the sum of the dimensionalities of all statements.

The approach we propose relies on the following observation. The magnitudes of $c_i$, $1 \leq i \leq m_S$ represent loop scaling and skewing factors, and in practice, we never need them to be very large. Let us assume $\forall i, -b \leq c_i \leq b$, for $b \geq 1$. Although the description that follows will work for any constant $b \geq 1$, we will use $b = 4$ to make resulting expressions more readable. It also turns out to be the value we chose for our implementation. Thus, $-4 \leq c_i^S \leq 4, 1 \leq i \leq m_S, \forall S$. When $c_i^S$'s are bounded this way, the scenario under which all $c_i$'s $1 \leq i \leq m_S$ are zero is captured as follows:

$$(c_1, c_2, \ldots, c_{m_S}) = \vec{0} \quad \Leftrightarrow \quad \sum_{i=1}^{i=m_S} 5^{i-1} c_i = 0.$$

The reasoning being: since $-4 \leq c_i \leq 4$, each $c_i$ can be treated as an integer in base 5. Since $5^i > \sum 5^{i-1} c_i$ for every $c_i$ in base 5, $c_i$, $1 \leq i \leq m_S$ will all be simultaneously zero only when $\sum_{i=1}^{i=m_S} 5^{i-1} c_i$ is zero. This in turn means:

$$(c_1, c_2, \ldots, c_{m_S}) \neq \vec{0} \quad \Leftrightarrow \quad \left| \sum_{i=1}^{i=m_S} 5^{i-1} c_i \right| \geq 1.$$

The absolute value operation can be eliminated by introducing a binary decision variable $\delta_S \in \{0, 1\}$, and enforcing the constraints:

$$\sum_{i=1}^{i=m_S} 5^{i-1} c_i \quad \geq \quad 1 - \delta_S 5^{m_S}, \tag{5}$$

$$-\sum_{i=1}^{i=m_S} 5^{i-1} c_i \quad \geq \quad 1 - (1 - \delta_S) 5^{m_S}. \tag{6}$$

***Explanation:*** Note that $5^{m_S} - 1$ and $1 - 5^{m_S}$ are the largest and the smallest values that the LHSs of (5) and (6) can take. Hence, when $\delta_S = 0$, constraint (5) enforces the desired outcome and (6) does not restrict the space in any way. Similarly, when $\delta_S = 1$, constraint (6) is enforcing and constraint (5) does not restrict the space. Thus, $\delta_S = 0$ covers the half-space given by

$$\sum_{i=1}^{i=m_S} 5^{i-1} c_i \geq 1,$$

and $\delta_S = 1$ covers the half-space given by

$$\sum_{i=1}^{i=m_S} 5^{i-1} c_i \leq -1.$$

## 3.4 Modeling the Complete Space of Linearly Independent Solutions

When searching for hyperplanes at a particular depth, the Pluto algorithm incrementally looks for linearly independent solutions until all dependences are satisfied and transformations are full column-ranked, i.e., one-to-one mappings for index sets. Determining constraints that enforce linear independence with respect to a set of previously found hyperplanes, introduces a challenge similar in nature to, but harder than that of zero solution avoidance.

Let $H_S$ be the matrix with rows representing a statement's dimension coefficients ($c_i^S, 1 \leq i \leq m_S$) already found from the outermost level to the current depth. Let $H_S^\perp$ be the sub-space orthogonal to $H_S$, i.e., each row of $H_S^\perp$ has a null component along the rows of $H_S$ ($H_S^\perp \cdot H_S^T = 0$). As an example, let $H_S$ be [1 0 0]. Then,

$$H_S^\perp = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

For a hyperplane to be linearly independent of previous ones, it should have a non-zero component in the orthogonal sub-space of previous ones, i.e., a non-zero component along at least one of the rows of $H_S^\perp$. We thus have two choices for every row of $H_S^\perp$, for the component being $\geq 1$ or $\leq -1$. The Pluto algorithm currently chooses $c_2 \geq 0, c_3 \geq 0, c_2 + c_3 \geq 1$, a portion of the linearly independent space that can be called the *non-negative orthant*. However, the complete linearly independent space is actually given by: $\mid c_2 \mid + \mid c_3 \mid \geq 1$. The restriction to the non-negative orthant among the four orthants may lead to the loss of interesting solutions, as the most profitable transformations may involve hyperplanes with negative coefficients. On the other hand, considering all orthants for each statement leads to a combinatorial explosion. For a 3-d statement, the space of linearly independent solutions has four orthants once the first hyperplane has been found. If there are multiple statements, the number of cases to be considered, if all of the orthants have to be explored, is the product of the number of orthants across all statements.

We address the above problem in a way similar to the way we avoid the zero solution in the previous section. We consider each of the rows of $H_S^\perp$, and compute the minimum and maximum values it could take given that the coefficients are bounded below and above by $-b$ and $b$. An expression is then constructed similar to (5) and (6) to avoid all of the components in the orthogonal sub-space being simultaneously zero. As an example, consider $H_S = [1\ 1\ 0]$. Then, $H_S^\perp = [[1\ -1\ 0], [0\ 0\ 1]]$. If $b = 4$, the maximum value taken by any row will be 8. To ensure linear independence, we require at least one of the rows to be non-zero, i.e.,

$$(c_1 - c_2, c_3) \neq \vec{0} \quad \Leftrightarrow \quad |c_1 - c_2| + |c_3| \geq 1.$$

The above can be captured through a single decision variable $\delta^l$ as follows:

$$\begin{aligned} 9(c_1 - c_2) + c_3 &\geq 1 - \delta_S^l 9^2, \\ -9(c_1 - c_2) - c_3 &\geq 1 - (1 - \delta_S^l)9^2. \end{aligned}$$

A strength of the above technique from a scalability viewpoint is that only a single decision variable per statement is required irrespective of the dimensionality of the statement or the level at which the transformation is being found. One may observe that linear independence constraints, whenever they are added, automatically imply zero solution avoidance. However, we have proposed a solution to both of these problems in order to maintain generality and flexibility for future customization of modeling, and to make the presentation clearer as the latter follows from the former.

### 3.5 Bounds on Coefficients

Recall that we bounded all $c_i$, $1 \leq i \leq m_S$ by $b$ and $-b$ where we chose $b = 4$ for clearer illustration in the previous sub-section. Besides these coefficients, the constant shifts $c_0$'s, and parametric shifts $d_i$'s are relative between statements, and thus there is no loss of generality in assuming that all of them are non-negative. In addition, the coefficients of $\vec{u}$ and $w$ are non-negative. Hence, all variables in our Integer Linear Programming (ILP) formulation are bounded from below while the statement dimension coefficients and the binary decision variables are bounded from both above and below.

### 3.6 Keeping the Coefficients Small

In the scenario where all or a number of the coefficient values are valid solutions with the same cost, for practical reasons, it is desirable to choose smaller ones, i.e., ones as close to zero as possible. A technique that is perfectly suited to achieve this goal is one of minimizing the sum of the absolute values of $c_i^S$ for each statement $S$, i.e., for each $S$, we define an additional variable, $c_{sum}^S$ such that:

$$c_{sum}^S = \sum_{i=1}^{i=m_S} |c_i^S|.$$

This in turn can be put in a linear form by expressing the RHS as a variable larger than the maximum of $2^{m_S}$ constraints, each generated using a particular sign choice for $c_i$, and then minimizing $c_{sum}$. For example, for a 2-d statement, we obtain

$$c_{sum} \geq \max(c_1 + c_2, -c_1 + c_2, c_1 - c_2, -c_1 - c_2).$$

In general, for each statement $S$, we have:

$$c_{sum}^S \geq \max\left(\pm c_1^S \pm c_2^S \cdots \pm c_{m_S}^S\right). \tag{7}$$

$c_{sum}^S$ is then included in the minimization objective at a particular position that we will see in Section 3.7 so that it is guaranteed to be minimized to be equal to the sum of the absolute values of $c_i^S$.

### 3.7 The Objective Function

Now, with constraints (5), (6), and similar ones for linear independence, the $\delta_S$, $\delta_S^l$s can be plugged into Pluto's lexicographic minimum objective at the right places with the following objective function:

$$\begin{aligned} &\text{lexmin}\,(\vec{u}, w, \\ &\qquad\vdots \\ &c_{sum}^{S_i}, c_1^{S_i}, c_2^{S_i}, \ldots, c_{m_S}^{S_i}, d_1^{S_i}, d_2^{S_i}, \ldots d_{m_p}^{S_i}, c_0^{S_i}, \delta_{S_i}, \delta_{S_i}^l, \\ &\qquad\vdots \\ &\qquad). \end{aligned} \tag{8}$$

Note that $c_{sum}$ coefficients appear ahead of the statement's dimension coefficients, and since (7) are the only constraints involving $c_{sum}$, minimizing $c_{sum}$ minimizes the sum of the absolute values of each statements' dimension coefficients, with $c_{sum}$ in the obtained solution becoming equal to that sum.

### 3.8 Complexity

In summary, to implement our approach on top of the Pluto algorithm, we used three more variables for each statement, $c_{sum}$, $\delta$, $\delta^l$, the last two of which are binary decision variables, i.e.,

1. $\delta_S$ to avoid the zero solution,
2. $\delta_S^l$ for linear independence,
3. and $c_{sum}^S$ to minimize the sum of absolute values of a statement's dimension coefficients.

Since the number of variables in the ILP formulation increases by three times the number of statements, this extension to the Pluto algorithm does not change its time complexity with respect to the number of statements and number of dependences. Note that our approach does not change the top-level iterative algorithm used in Pluto, but only the space of transformations modeled.

### 3.9 Completeness

Our formulation is still dimension-relative and characterizes the space of all valid schedules at a given level (nesting depth), on which an objective function is applied – it is thus not the complete space of all multi-dimensional schedules at once. When compared to the current Pluto algorithm, although transformations that involve negative coefficients are included in the space, due to the bounds we place on coefficients corresponding to statement's dimensions ($-b$ and $b$), we are also excluding transformations that originally existed in Pluto's space. To some extent, this bound ($b$ in Section 3) can be adjusted and increased to reasonable limits if desired – the caveat being larger numbers in the ILP formulation. Though this does not pose a problem for correctness, it can make the ILP more complex and longer to solve. However, we did not find the need to experiment with larger values of $b$ because, (1) coefficients larger than the value used were not needed for any of the kernels we came across, and (2) large values (of the order of loop bounds when the latter are constant) introduce spurious shifts and skews and should anyway be avoided. Note that no upper bound is placed on the shift coefficients of transformations. In summary, though we cannot theoretically claim a complete modeling of the transformation space, it is near-complete and apparently complete for all practical purposes.

## 4. Experiments

In this section, we study: (1) the scalability of our new approach, and (2) the impact of newly included transformations on performance.

We implemented the techniques proposed in Section 3 by modifying the automatic transformation component in Pluto version 0.11.2 (from the 'pet' branch of its public git repository) — we refer to our system as PLUTO+. In the rest of this section, *Pluto* refers to the existing Pluto version 0.11.2 (git branch 'pet'), while *Pluto+* refers to our new approach that models the larger space of transformations. Other than the component that automatically determines transformations, Pluto+ shares all remaining components with Pluto. With both Pluto and Pluto+, the Clang-based pet [39] frontend was used for polyhedral extraction from C, ISL [38] for dependence testing, Cloog-isl [3] 0.18.2 for code generation, and PIP [12] 1.4.0 as the lexmin ILP solver. For large ILPs with Pluto+ (100+ variables), GLPK [16] (version 4.45) was used. It was significantly faster than PIP or ISL's solver for such problems. With Pluto, GLPK did not make any significant difference as the ILP formulations were smaller. Typical options used for generating parallel code with Pluto were used for both Pluto and Pluto+; these were: '–isldep –lastwriter –parallel –tile'. Index set splitting (ISS) implementation made available by [6] was enabled ('–iss' flag). For periodic stencils, diamond tiling [2] was enabled ('–partlbtile').

All performance experiments were conducted on a 2-way SMP Intel Xeon E5 2680 system (Sandybridge microarchitecture) running at 2.7 GHz with 64 GB of DDR3-1600 RAM, and with Intel's C compiler (icc) 14.0.1. Details of the setup along with compiler flags are provided in Table 1. The default thread to core affinity of the Intel KMP runtime was used (KMP_AFFINITY granularity set to *fine, scatter*), i.e., threads are distributed as evenly as possible across the entire system. Hyperthreading was disabled, and the experiments were conducted with up to 16 threads running on 16 cores. Both Pluto and Pluto+ themselves were compiled using gcc 4.8.1 and run single-threaded.

Table 1: Architecture details

| | Intel Xeon E5-2680 |
|---|---|
| Clock | 2.7 GHz |
| Cores / socket | 8 |
| Total cores | 16 |
| L1 cache / core | 32 KB |
| L2 cache / core | 512 KB |
| L3 cache / socket | 20 MB |
| Peak GFLOPs | 172.8 |
| Compiler | Intel C compiler (icc) 14.0.1 |
| Compiler flags | -O3 -xHost -ipo -restrict -fno-alias -ansi-alias -fp-model precise -fast-transcendentals |
| Linux kernel | 3.8.0-44 |

***Benchmarks:*** Evaluation was done on 27 out of the 30 benchmarks/kernels from Polybench/C version 3.2 [30]; three of the benchmarks — namely, trmm, adi, and reg-detect were excluded as they were reported to be clearly not representative of computations intended [42]. Polybench's standard dataset sizes were used. In addition, we also evaluate Pluto+ on 1-d, 2-d, and 3-d heat equation applications with periodic boundary conditions. These can be found in the Pochoir suite [36]. Then, we also evaluate on Lattice Boltzmann Method (LBM) simulations that simulate lid-driven cavity flow (lbm-ldc) [8], flow past cylinder (lbm-fpc-d2q9), and Poiseuille flow [43] (lbm-poi-d2q9). An *mrt* version of LBM [11] involves multiple relaxation parameters for a single time step, and therefore a higher operational intensity. Periodic boundary conditions were employed for these. We consider two different configuration classes for lbm: d2q9 and d3q27. 'd3q27' signifies a 3-d data grid with a 27 neighbor interaction while 'd2q9', a 2-d one with 9 neighbor interactions. These are full-fledged real applications as opposed to kernels. And finally, we also evaluate on *171.swim* from

Table 2: Problem sizes for heat, swim, and LBM benchmarks

| Benchmark | Problem size |
|---|---|
| heat-1dp | $1.6 \cdot 10^6 \times 1000$ |
| heat-2dp | $16000^2 \times 500$ |
| heat-3dp | $300^3 \times 200$ |
| swim | $1335^2 \times 800$ |
| lbm-ldc-d2q9 | $1024^2 \times 50000$ |
| lbm-ldc-d2q9-mrt | $1024^2 \times 20000$ |
| lbm-fpc-d2q9 | $1024 \times 256 \times 40000$ |
| lbm-poiseuille-d2q9 | $1024 \times 256 \times 40000$ |
| lbm-ldc-d3q27 | $256^3 \times 300$ |

the SPECFP2000 suite. The *swim* benchmark [33, 35] solves shallow water equations on a 2-d grid with periodic boundary conditions.

## 4.1 Impact on Transformation and Compilation Times

Table 3 provides the time spent in automatic transformation as well as the total time spent in source-to-source polyhedral transformation. The automatic transformation time reported therein is the time taken to compute or determine the transformations as opposed to the total time taken to transform the source. The total time taken during polyhedral optimization to transform the source is reported as "Total polyhedral compilation time". Dependence analysis and code generation times are two other key components besides automatic transformation. The *Misc/other* component primarily includes time spent in post-transformation analyses such as computing hyperplane properties (parallel, sequential, tilable), and precise tilable band detection. This is potentially done multiple times for various late transformations like intra-tile optimization for better vectorization and spatial locality, and creating tile wavefronts. The lower part of Table 3 lists those benchmarks where Pluto+ actually finds a significantly different transformation involving a negative coefficient — to enable tiling and other optimization. All of these are partial differential equation solvers of some form where periodic boundary conditions are used. When there is no periodicity, both Pluto and Pluto+ find the same transformations.

Figure 5 shows the breakdown across various components of the polyhedral source-to-source parallelization. Note that code generation time as well as other post-transformation analysis times are affected by transformations obtained. We observe from Table 3 and Figure 5 that in a majority of cases, the auto-transformation time is at most few tens of milliseconds, and that the code generation time dominates for both Pluto and Pluto+ in many cases. Surprisingly, auto-transformation times are on average lower with Pluto+ — 11% lower on Polybench and 38% lower on the periodic stencil benchmarks. This is due to the fact that additional checks had been used in Pluto to ensure that the single orthogonal sub-space chosen to derive linear independence constraints was a feasible one. Such checks are obviously not needed with Pluto+. Swim presented the most challenging case with 219 variables in the Pluto+ ILP, and was the only benchmark for which Pluto+ used GLPK.

We find Pluto+ to be surprisingly scalable and practical on such a large set of kernels, benchmarks, and real applications from certain domains. None of these posed a serious scalability issue on the total polyhedral source-to-source transformation time. In a majority of the cases, the overall compilation time itself was low, from under a second to a few seconds. In nearly all cases where the total compilation time was significant and showed a significant increase with Pluto+, it was because code generation had taken much longer since a non-trivial transformation had been performed with Pluto+ while Pluto did not change the original

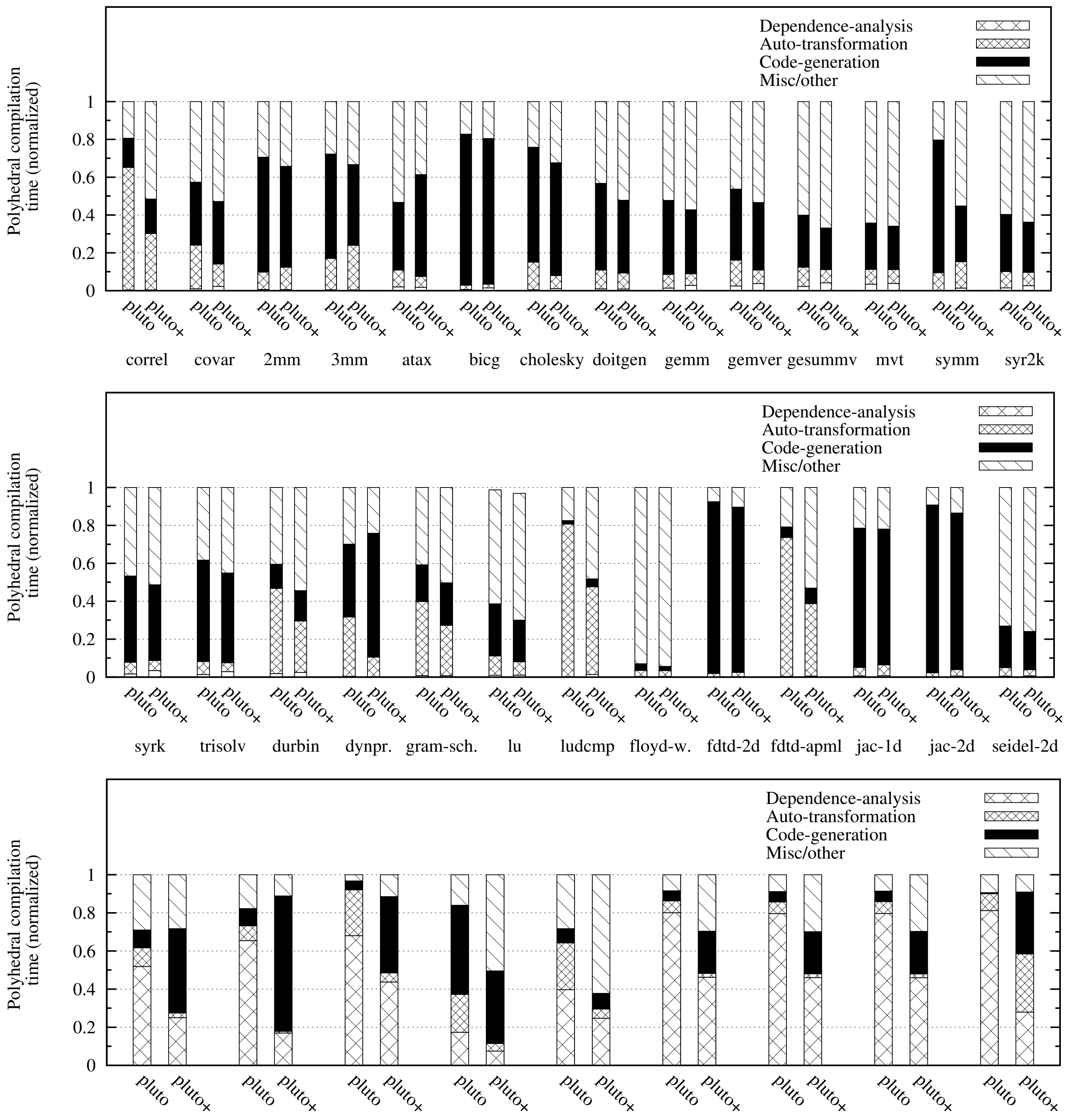


Figure 5: Polyhedral compilation time breakdown into dependence analysis, transformation, code generation, and miscellaneous (some benchmark names have been abbreviated; full names can be found in Table 3); both, Pluto and Pluto+ times are individually normalized to their respective total times for readability. Table 3 can be used in conjunction to determine the absolute time for any component.

Table 3: Impact on polyhedral compilation time (all times reported are in seconds)

| Benchmark | Automatic transformation time (s) | | Total polyhedral compilation time (s) | | Factor of increase due to Pluto+ | |
|---|---|---|---|---|---|---|
| | Pluto | Pluto+ | Pluto | Pluto+ | in transformation | overall |
| correlation | 0.470 | 0.214 | 0.726 | 0.719 | 0.45 | 0.99 |
| covariance | 0.043 | 0.027 | 0.188 | 0.227 | 0.62 | 1.21 |
| 2mm | 0.023 | 0.033 | 0.246 | 0.282 | 1.47 | 1.15 |
| 3mm | 0.070 | 0.126 | 0.423 | 0.536 | 1.81 | 1.27 |
| atax | 0.004 | 0.004 | 0.041 | 0.061 | 0.97 | 1.49 |
| bicg | 0.002 | 0.003 | 0.112 | 0.151 | 1.21 | 1.36 |
| cholesky | 0.048 | 0.025 | 0.328 | 0.366 | 0.52 | 1.12 |
| doitgen | 0.018 | 0.017 | 0.178 | 0.198 | 0.94 | 1.11 |
| gemm | 0.005 | 0.006 | 0.066 | 0.095 | 1.23 | 1.44 |
| gemver | 0.010 | 0.007 | 0.070 | 0.097 | 0.73 | 1.37 |
| gemvsumv | 0.004 | 0.004 | 0.039 | 0.064 | 1.12 | 1.63 |
| mvt | 0.002 | 0.003 | 0.026 | 0.036 | 1.28 | 1.38 |
| symm | 0.052 | 0.039 | 0.564 | 0.281 | 0.75 | 0.50 |
| syr2k | 0.009 | 0.010 | 0.100 | 0.147 | 1.20 | 1.48 |
| syrk | 0.004 | 0.005 | 0.066 | 0.095 | 1.24 | 1.44 |
| trisolv | 0.004 | 0.004 | 0.055 | 0.081 | 1.02 | 1.48 |
| durbin | 0.045 | 0.029 | 0.101 | 0.105 | 0.63 | 1.05 |
| dynprog | 0.171 | 0.103 | 0.545 | 1.007 | 0.60 | 1.85 |
| gramschmidt | 0.098 | 0.069 | 0.250 | 0.260 | 0.71 | 1.04 |
| lu | 0.006 | 0.005 | 0.117 | 0.164 | 0.88 | 1.40 |
| ludcmp | 0.973 | 0.263 | 1.209 | 0.569 | 0.27 | 0.47 |
| floyd-warshall | 0.009 | 0.012 | 0.298 | 0.400 | 1.29 | 1.34 |
| fdtd-2d | 0.043 | 0.057 | 2.424 | 2.527 | 1.31 | 1.04 |
| fdtd-apml | 1.581 | 0.563 | 2.158 | 1.475 | 0.36 | 0.68 |
| jacobi-1d-imper | 0.006 | 0.007 | 0.130 | 0.13 | 1.28 | 1.00 |
| jacobi-2d-imper | 0.015 | 0.026 | 0.705 | 0.751 | 1.71 | 1.07 |
| seidel-2d | 0.011 | 0.009 | 0.249 | 0.260 | 0.75 | 1.04 |
| Mean (geometric) | | | | | **0.89** | **1.15** |
| heat-1dp | 0.033 | 0.016 | 0.337 | 0.648 | 0.48 | 1.92 |
| heat-2dp | 0.190 | 0.078 | 2.430 | 9.282 | 0.41 | 3.82 |
| heat-3dp | 2.585 | 0.810 | 10.77 | 16.86 | 0.31 | 1.57 |
| lbm-ldc-d2q9 | 0.701 | 0.330 | 3.535 | 8.152 | 0.47 | 2.31 |
| lbm-ldc-d3q27 | 8.726 | 2.704 | 35.52 | 56.36 | 0.31 | 1.59 |
| lbm-mrt-d2q9 | 0.321 | 0.177 | 5.180 | 9.068 | 0.55 | 1.75 |
| lbm-fpc-d2q9 | 0.324 | 0.175 | 5.282 | 9.017 | 0.54 | 1.71 |
| lbm-poi-d2q9 | 0.323 | 0.176 | 5.226 | 9.001 | 0.55 | 1.72 |
| swim | 1.489 | 14.46 | 16.74 | 47.35 | 9.71 | 2.83 |
| Mean (geometric) | | | | | **0.62** | **2.04** |

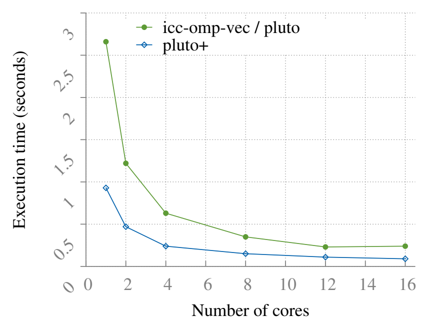
schedule (Table 3). We thus see this as an opportunity to further improve code generation techniques including integrating hybrid polyhedral and syntactic approaches like PrimeTile [18] where suitable. In conclusion, the increase in compilation time due to Pluto+ itself is quite acceptable.

## 4.2 Performance Impact on Periodic Heat Equation, LBM Simulations, and Swim

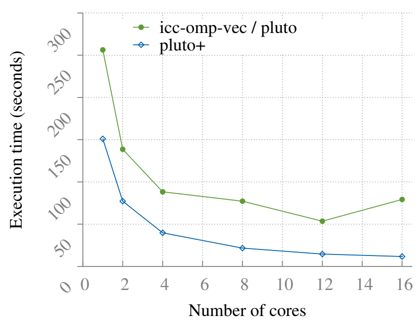
On all benchmarks from Polybench, Pluto+ obtained the same transformation (or equivalent ones) as Pluto. Polybench does not include any kernels where negative coefficients help, at least as per our cost function. We thus obtain the same performance on all of Polybench.

Figure 6 shows results where we compare Pluto, Pluto+, and ICC on the heat equation, swim, and LBM simulations. *icc-omp-vec* represents the original code optimized with little effort — by manually inserting an OpenMP pragma on the outermost space loop and '`#pragma ivdep`' for the innermost space loop to enable vectorization. The problem sizes used are shown in Table 2.

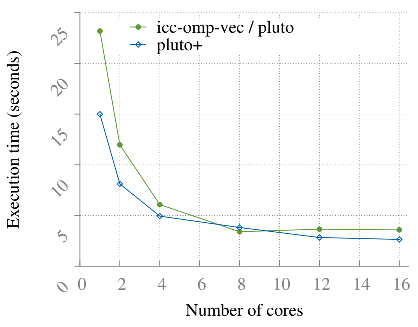
Pluto is unable to perform time tiling or any useful optimization on stencils with periodicity: this includes the LBM codes considered, swim, and heat equation codes. Its performance is thus the same as that achieved via *icc-omp-vec* through parallelization of the space loop immediately inside the time loop and vectorization of the innermost loop. On the other hand, Pluto+ performs time tiling for all periodic stencils using the diamond tiling technique [2]. We obtain speedups of 2.72, 6.73 and 1.4 over *icc-omp-vec* or Pluto for heat-1dp, heat-2dp, and heat-3dp respectively. For LBM, we also compare with Palabos [29], a well-known stable library package for LBM computations where a user manually provides the specification in C++. Palabos results are provided as a reference point as opposed to for a direct comparison since the approach falls into a different class. The performance of LBM benchmarks is usually reported in million lattice site updates per second (MLUPS). MLUPS is computed by dividing the total of grid updates (time steps times



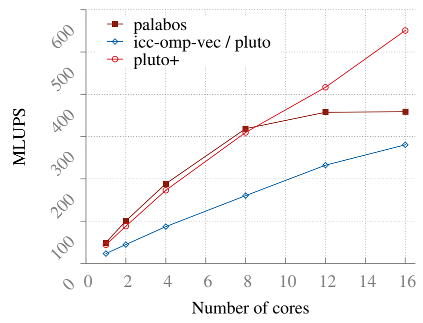
(a) `heat-1d`



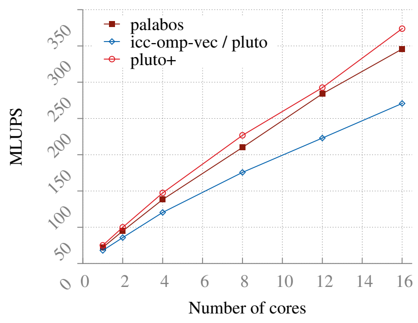
(b) `heat-2d`



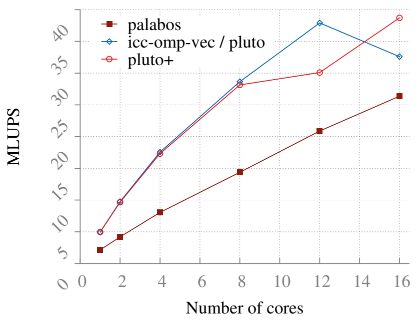
(c) `heat-3d`



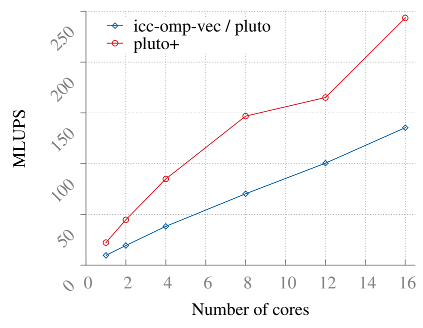
(d) `lbm-ldc-d2q9`



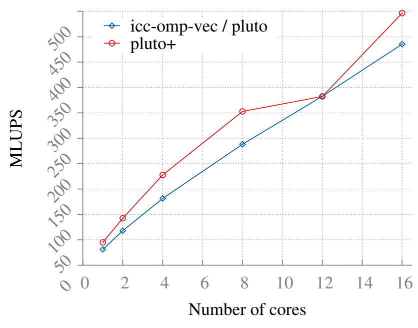
(e) `lbm-mrt-d2q9`



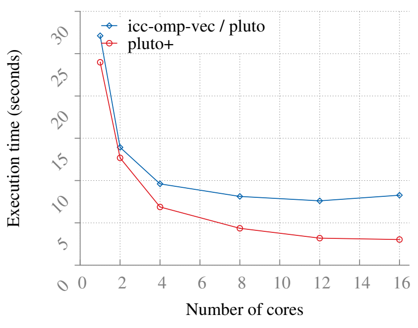
(f) `lbm-ldc-d3q27`



(g) `lbm-fpc-d2q9`



(h) `lbm-poi-d2q9`



(i) `swim`

Figure 6: Performance Impact

number of grid points) by the execution time. For flow-past cylinder and Poiseuille flow, we could not find a way to express them in Palabos, and hence those comparisons are not provided. For LBM, we obtain a mean speedup of 1.33× over *icc-omp-vec* and Pluto, and 1.62× over Palabos. All of these performance improvements are due to the improved locality as a result of time tiling enabled by Pluto+: due to reduced memory bandwidth utilization, besides improvement in single thread performance, it led to better scaling with the number of cores.

For the benchmarks on three-dimensional data grids (heat-3d and lbm-ldc-d3q27), we see little or no improvement in some cases. The 3-d benchmarks were particularly hard to optimize and we found that more work was necessary to automate for certain complementary aspects. For lbm-ldc-d3q27, the drop in *icc-omp-vec* or Pluto's performance was due to NUMA locality effects, an aspect that we do not model or optimize for. This was deduced from the observation that changing KMP's affinity setting from 'scatter' to 'compact' made the results follow a predictable pattern when running on 12 threads or more. However, we found the default 'scatter' setting to be delivering the best performance. Effective vectorization is also more difficult for 3-d stencils than for lower dimensional ones. Previous works that studied this problem in detail [19–21] demonstrated little or no improvement for 3-d cases.

For the swim benchmark, our tool takes as input a C translation of it (since the frontend only accepts C) with all three *calc* routines inlined. We verified that the Intel compiler provided exactly the same performance on this C version as on the original Fortran version. The inlining thus does not affect icc's optimization. Index set splitting application is fully automatic (from [6]) just like for the other periodic stencils. On swim, we obtain a speedup of 2.73× over icc's auto-parallelization.

## 5. Related Work

Affine schedules computed by Feautrier's scheduling algorithms [14, 15] allow for coefficients to be negative. However, the objectives used in finding such schedules are those of reducing the dimensionality of schedules and typically minimizing latency. Thus, they do not capture conditions for the validity of tiling, pipelined parallelism, communication-free parallelism, and are thus very different from the objective used in Pluto or Pluto+. The latter objective requires additional constraints for zero solution avoidance and linear independence which in turn necessitates techniques developed in this paper for modeling the space of affine transformations.

Griebl's approach [17] finds forward communication only (FCO) placements for schedules found by [14, 15]: placements determine where (processor) iterations execute while schedules determine when they execute. The FCO constraint is same as the tiling constraint in (2). FCO placements, when found, will allow tiling of both schedule dimensions and placement dimensions. Once the FCO constraint is enforced, the approach proposed to pick from the space of valid solutions relies on finding the vertices, rays, and lines (extremal solutions) of the space of valid transformations. From among these extremal solutions, the one that leads to a zero communication (zero dependence distance) for the maximum number of dependences is picked ([17], Section 7.4). Linear independence is not an issue since all extremal solutions are found, and those linearly dependent on scheduling dimensions can be excluded. Finding the vertices, rays, and lines of the space of valid transformations is very expensive given the number of dimensions (transformation coefficients) involved. The practicality and scalability of such an approach has not been demonstrated.

Lim and Lam's framework [24–26] was the first affine transformation framework to take a partitioning view, as opposed to a scheduling-cum-placement view, with the objective to reduce the frequency of synchronization and improve locality. The algorithm proposed to finally determine transformation coefficients, Algorithm A in [25], does not use a specific objective or cost function to choose among valid ones. It instead proposes an iterative approach to finding linearly independent solutions to the set of constraints representing the space of valid solutions. There is no bound on the number of steps it takes, or an assertion on its completeness or on the quality of the solution beyond maximizing the number of degrees of parallelism. The latter was a significant advance over prior art and provided better parallelization with tiling for a much more general class of codes. It has already been shown that only maximizing the number of degrees of parallelism is not sufficient, and solutions which exhibit the same number of degrees of parallelism can perform very differently [5].

Pouchet et al. [31]'s approach for iterative optimization is driven by search and empirical feedback as opposed to by a model, and is on the same space of schedules as those modeled in [14]. In spite of supporting negative coefficients in schedules, there is no notion of tiling or communication-free parallelization, whether or not they required negative coefficients. Its limitations are thus independent of our contributions here.

Vasilache [37] proposes a formulation that models the entire space of multi-dimensional schedules, i.e., all levels at a time. However, the modeling does not include an objective function. An objective function like that of Pluto and Pluto+ requires linear independence and trivial solution avoidance, and there is no known way to achieve this with the formulation in [37]. Constraints to ensure linear independence often necessitate a level by level approach [4, 23], and doing so without losing interesting valid solutions requires modeling such as the one we proposed in this paper. Pouchet et al. [32] developed a hybrid iterative cum analytical model-driven approach to explore the space of all valid loop fusion/distributions, while ensuring tilability and parallelization of the fused loop nests. The transformation coefficients were limited to non-negative ones to enable use of Pluto.

Kong et al. [21] presented a framework that finds schedules that enable vectorization and better spatial locality in addition to the dependence minimization objective of Pluto or Pluto+. Their work is thus complementary, and the limitations addressed by Pluto+ exist in their modeling of the space of transformations too.

The RSTREAM compiler [27] includes techniques that model negative coefficients [22]. The approach is different from ours and uses a larger number of decision variables. For each statement, one decision variable is used to model the direction choice (positive or negative) associated with each row of the orthogonal sub-space basis ($H_S^\perp$ in Section 3.4) — in effect, multiple decision variables for each statement capture the choice of the orthant. In contrast, our approach encodes linear independence for a statement using a single decision variable, and we achieved this by exploiting bounds on the components associated with each row of $H_S^\perp$. The RSTREAM approach is not available for a direct experimental comparison.

Bondhugula et al. [6] presented an automatic index set splitting technique to enable tiling for periodic stencils. The approach requires techniques developed in this work to be able to ultimately find the right transformations once such index set splitting has been performed. The same index set splitting technique is used in conjunction with Pluto+ to enable a richer variety of transformations, and we are able to reproduce performance improvements similar to those reported in [6] for 1-d, 2-d, and 3-d heat equations with periodicity, and the swim benchmark.

## 6. Conclusions

In this paper, we proposed Pluto+, an approach to model a significantly larger space of affine transformations than previous approaches in conjunction with a cost function. Experimental evaluation showed that Pluto+ provided no degradation in performance in any case. For Lattice Boltzmann Method (LBM) simulations with periodic boundary conditions, it provided a mean improvement of 1.33× over Pluto while running on 16 cores of a modern multicore SMP. On the swim benchmark from SPECFP2000, Pluto+ obtains a speedup of 2.73× over Intel C compiler's auto-parallelization. We also showed that Pluto+ does not pose a scalability issue and is often as scalable as Pluto. Experimental results show that Pluto+ increases overall polyhedral compilation time on Polybench by only 15% on average. In cases where it improved execution time significantly due to a different transformation, the total polyhedral compilation time was increased by 2.04×. As for the time component involved in computing transformations, Pluto+ is faster than Pluto on average. We believe our approach and these results present a significant advance in the field of static loop nest optimization.

## Acknowledgments

We gratefully acknowledge the anonymous reviewers of PPoPP'15, in particular, Reviewer-#2, for insightful comments that have improved the paper. We thank Albert Cohen for several discussions on key aspects of this work. We thank the Parallel Computing group at Intel Labs Bangalore for donating us Intel compiler software, loaning equipment that was used in part for the experimental evaluation, and for discussions related to optimization of LBM applications. We also thank Microsoft Research India for partly supporting Aravind Acharya's travel to PPoPP'15.

## References

[1] A. V. Aho, R. Sethi, J. D. Ullman, and M. S. Lam. *Compilers: Principles, Techniques, and Tools (second edition)*. Prentice Hall, 2006.

[2] V. Bandishti, I. Pananilath, and U. Bondhugula. Tiling stencil computations to maximize parallelism. In *Supercomputing*, pages 40:1–40:11, 2012.

[3] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 7–16, 2004. .

[4] U. Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan. Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model. In *International conference on Compiler Construction (ETAPS CC)*, 2008.

[5] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral parallelizer and locality optimizer. In *ACM SIGPLAN symposium on Programming Languages Design and Implementation (PLDI)*, pages 101–113, 2008.

[6] U. Bondhugula, V. Bandishti, A. Cohen, G. Potron, and N. Vasilache. Tiling and optimizing time-iterated computations on periodic domains. In *International conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 39–50, 2014.

[7] C. Chen. Polyhedra scanning revisited. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 499–508, 2012.

[8] S. Chen and G. D. Doolen. Lattice boltzmann method for fluid flows. *Annual review of fluid mechanics*, 30(1):329–364, 1998.

[9] C. Choffrut and K. Culik. Folding of the plane and the design of systolic arrays. *Information Processing Letters*, 17(3):149 – 153, 1983.

[10] Cloog. The Chunky Loop Generator. http://www.cloog.org.

[11] D. d'Humières. Multiple–relaxation–time lattice boltzmann models in three dimensions. *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 360(1792):437–451, 2002.

[12] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22(3):243–268, 1988.

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

[14] P. Feautrier. Some efficient solutions to the affine scheduling problem: Part II, multidimensional time. *International Journal of Parallel Programming*, 21(6):389–420, 1992.

[15] P. Feautrier. Some efficient solutions to the affine scheduling problem: Part I, one-dimensional time. *International Journal of Parallel Programming*, 21(5):313–348, 1992.

[16] GNU. GLPK (GNU Linear Programming Kit). https://www.gnu.org/software/glpk/.

[17] M. Griebl. *Automatic Parallelization of Loop Programs for Distributed Memory Architectures*. University of Passau, 2004. Habilitation thesis.

[18] A. Hartono, M. Baskaran, C. Bastoul, A. Cohen, S. Krishnamoorthy, B. Norris, and J. Ramanujam. A parametric multi-level tiler for imperfect loop nests. In *International conference on Supercomputing (ICS)*, 2009.

[19] T. Henretty, K. Stock, L.-N. Pouchet, F. Franchetti, J. Ramanujam, and P. Sadayappan. Data layout transformation for stencil computations on short simd architectures. In *ETAPS International conference on Compiler Construction (CC'11)*, pages 225–245, Mar. 2011.

[20] T. Henretty, R. Veras, F. Franchetti, L.-N. Pouchet, J. Ramanujam, and P. Sadayappan. A stencil compiler for short-vector SIMD architectures. In *ACM International Conference on Supercomputing*, 2013.

[21] M. Kong, R. Veras, K. Stock, F. Franchetti, L.-N. Pouchet, and P. Sadayappan. When polyhedral transformations meet simd code generation. In *ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, June 2013.

[22] A. Leung, N. Vasilache, B. Meister, and R. Lethin. Methods and apparatus for joint parallelism and locality optimization in source code compilation, June 3 2010. WO Patent App. PCT/US2009/057,194.

[23] W. Li and K. Pingali. A singular loop transformation framework based on non-singular matrices. *International Journal of Parallel Programming*, 22(2):183–205, 1994.

[24] A. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine transforms. In *ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pages 201–214, 1997.

[25] A. Lim and M. S. Lam. Maximizing parallelism and minimizing synchronization with affine partitions. *Parallel Computing*, 24(3-4): 445–475, 1998.

[26] A. Lim, G. I. Cheong, and M. S. Lam. An affine partitioning algorithm to maximize parallelism and minimize communication. In *ACM International Conference on Supercomputing (ICS)*, pages 228–237, 1999.

[27] B. Meister, N. Vasilache, D. Wohlford, M. Baskaran, A. Leung, and R. Lethin. R-Stream Compiler. In *Encyclopedia of Parallel Computing*, pages 1756–1765. 2011.

[28] N. Osheim, M. M. Strout, D. Rostron, and S. Rajopadhye. Smashing: Folding space to tile through time. In *Workshop on Languages and Compilers for Parallel Computing (LCPC)*, pages 80–93. Springer-Verlag, 2008.

[29] Palabos. Palabos. http://www.palabos.org/.

[30] Polybench. Polybench suite. http://polybench.sourceforge.net.

[31] L.-N. Pouchet, C. Bastoul, J. Cavazos, and A. Cohen. Iterative optimization in the polyhedral model: Part II, multidimensional time. In *ACM SIGPLAN symposium on Programming Languages Design and Implementation (PLDI)*, June 2008.

[32] L.-N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, P. Sadayappan, and N. Vasilache. Loop transformations: Convexity, pruning and optimization. In *ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'11)*, Jan. 2011.

[33] R. Sadourny. The dynamics of finite-difference models of the shallow-water equations. *J. Atm. Sciences*, 32(4), Apr. 1975.

[34] R. Strzodka, M. Shaheen, D. Pajak, and H.-P. Seidel. Cache accurate time skewing in iterative stencil computations. In *International conference on Parallel Processing (ICPP)*, pages 571–581, 2011.

[35] P. N. Swarztrauber. 171.swim spec cpu2000 benchmark description file. Standard Performance Evaluation Corporation. http://www.spec.org/cpu2000/CFP2000/171.swim/docs/171.swim.html, 2000.

[36] Y. Tang, R. A. Chowdhury, B. C. Kuszmaul, C.-K. Luk, and C. E. Leiserson. The Pochoir stencil compiler. In *ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 117–128, 2011.

[37] N. Vasilache. *Scalable Program Optimization Techniques in the Polyhedral Model*. PhD thesis, Université de Paris-Sud, INRIA Futurs, Sept. 2007.

[38] S. Verdoolaege. ISL: An Integer Set Library for the Polyhedral Model. In K. Fukuda, J. Hoeven, M. Joswig, and N. Takayama, editors, *Mathematical Software - ICMS 2010*, volume 6327, pages 299–302. Springer, 2010.

[39] S. Verdoolaege and T. Grosser. Polyhedral extraction tool. In *International workshop on Polyhedral Compilation Techniques (IMPACT)*, 2012.

[40] D. Wonnacott. Using time skewing to eliminate idle time due to memory bandwidth and network limitations. In *IPDPS*, pages 171 –180, 2000.

[41] Y. Yaacoby and P. R. Cappello. Converting affine recurrence equations to quasi-uniform recurrence equations. *VLSI Signal Processing*, 11(1-2):113–131, 1995.

[42] T. Yuki. Understanding PolyBench/C 3.2 kernels. In *International workshop on Polyhedral Compilation Techniques (IMPACT)*, Jan. 2014.

[43] Q. Zou and X. He. On pressure and velocity boundary conditions for the lattice Boltzmann BGK model. *Physics of Fluids (1994-present)*, 9(6):1591–1598, 1997.