Loop Transformations: Convexity, Pruning and Optimization

Louis-Noël Pouchet, Uday Bondhugula, Cédric Bastoul, Albert Cohen, Jagannathan Ramanujam, Ponnuswamy Sadayappan, Nicolas Vasilache

Loop Transformations: Convexity, Pruning and Optimization

Louis-Noël Pouchet
The Ohio State University
pouchet@cse.ohio-state.edu

Uday Bondhugula
IBM T.J. Watson Research Center
ubondhug@us.ibm.com

Cédric Bastoul
University of Paris-Sud 11
cedric.bastoul@inria.fr

Albert Cohen
INRIA
albert.cohen@inria.fr

J. Ramanujam
Louisiana State University
jxr@ece.lsu.edu

P. Sadayappan
The Ohio State University
saday@cse.ohio-state.edu

Nicolas Vasilache
Reservoir Labs, Inc.
vasilache@reservoir.com

Abstract

High-level loop transformations are a key instrument in mapping computational kernels to effectively exploit resources in modern processor architectures. However, determining appropriate compositions of loop transformations to achieve this remains a significantly challenging task; current compilers may achieve significantly lower performance than hand-optimized programs. To address this fundamental challenge, we first present a convex characterization of all distinct, semantics-preserving, multidimensional affine transformations. We then bring together algebraic, algorithmic, and performance analysis results to design a tractable optimization algorithm over this highly expressive space. The framework has been implemented and validated experimentally on a representative set of benchmarks run on state-of-the-art multi-core platforms.

Categories and Subject Descriptors D 3.4 [*Programming languages*]: Processor — Compilers; Optimization

General Terms Algorithms; Performance

Keywords Compilation; Compiler Optimization; Parallelism; Loop Transformations; Affine Scheduling

1. Introduction

Loop nest optimization continues to drive much of the ongoing research in the fields of optimizing compilation [8, 27, 33], high-level hardware synthesis [22], and adaptive library generation [19, 47]. Loop nest optimization attempts to map the proper granularity of independent computation to a complex hierarchy of memory, computing, and interconnection resources. Despite four decades of research and development, it remains a challenging task for compiler designers and a frustrating experience for programmers — the performance gap, between expert-written code for a given machine and that optimized and parallelized automatically by a compiler, is *widening* with newer generations of hardware.

One reason for this widening gap is due to the way loop nest optimizers attempt to decompose the global optimization problem into simpler sub-problems. Recent results in feedback-directed optimization [33] indicate that oversimplification of this decomposition is largely responsible for the failure. The polyhedral compiler framework provides a convenient and powerful abstraction to apply composite transformations in one step [20, 24]. However, the space of affine transformations is extremely large. Linear optimization in such a high-dimensional space raises complexity issues and its effectiveness is limited by the lack of accurate profitability models.

This paper makes fundamental progresses in the understanding of polyhedral loop nest optimization. It is organized as follows. Section 2 formalizes the optimization problem and contributes a complete, convex characterization of all distinct, semantics-preserving, multidimensional affine transformations. Focusing on one critical slice of the transformation space, Section 3 proposes a general framework to reason about all distinct, semantics-preserving, multidimensional statement interleavings. Section 4 presents a complete and tractable optimization algorithm driven by the iterative evaluation of these interleavings. Section 5 presents experimental data. Related work is discussed in Section 6.

2. Problem Statement and Formalization

The formal semantics of a language allows the definition of program transformations and equivalence criteria to validate these transformations. Denotational semantics can be used as an abstraction to decide on the functional equivalence of two programs. But operational semantics is preferred to express program transformations themselves, especially in the case of transformations impacting the detailed interaction with particular hardware. Program optimization amounts to searching for a particular point in a space of semantics-preserving transformations. When considering optimizing compilation as a whole, the space has no particular property that makes it amenable to any formal characterization or optimization algorithm. As a consequence, compilers decompose the problem into sub-problems that can be formally defined, and for which complexity results and effective optimization algorithms can be derived.

The ability to encompass a large set of program transformations into a single, well understood optimization problem is the strongest asset of the polyhedral compilation model. In the polyhedral model, transformations are validated against dependence relations among dynamic instances of individual statements of a loop nest. The dependence relation is the denotational abstraction that defines functional equivalence. This dependence relation may not be statically computable, but (conservative) abstractions may be (finitely) represented through systems of affine inequalities, or unions of parametric polyhedra [2, 16, 22]. Two decades of work in polyhedral compilation has demonstrated that the main loop nest transformations can be modeled and effectively constructed as multidimensional affine schedules, mapping dynamic instances of individual statements to lexicographically sorted vectors of integers (or rational numbers) [6, 8, 17, 20, 24, 38]. Optimization in the polyhedral model amounts to selecting a good multidimensional affine schedule for the program.

In this paper, we make the following contributions:

- We take this well defined optimization problem, and provide a *complete, convex formalization* for it. This convex set of semantics-preserving, distinct, affine multidimensional schedules opens the door to more powerful linear optimization algorithms.
- We propose a decomposition of the general optimization problem over this convex polyhedron into sub-problems of much lower complexity, introducing the powerful *fusibility* concept. The fusibility relation generalizes the legality conditions for loop fusion, abstracting a large set of multidimensional, affine, fusion-enabling transformations.
- We demonstrate that exploring fusibility opportunities in a loop nest reduces to the enumeration of total preorders, and to the existence of pairwise compatible loop permutations. We also study the transitivity of the fusibility relation.
- Based on these results, we design a multidimensional affine scheduling algorithm targeted at achieving portable high performance across modern processor architectures.
- Our approach has been fully implemented in a source-to-source compiler and validated on representative benchmarks.

2.1 Loop optimization challenge

Let us illustrate the challenge of loop optimization from the standpoint of performance portability through a simple example, ThreeMatMat, shown in Figure 1. We seek the best combination of loop transformations—expressed as a multidimensional affine schedule—to optimize a sequence of three matrix-products for a given target platform.

```
  for (i1 = 0; i1 < N; ++i1)
    for (j1 = 0; j1 < N; ++j1)
      for (k1 = 0; k1 < N; ++k1)
R:      C[i1][j1] += A[i1][k1] * B[k1][j1];
  for (i2 = 0; i2 < N; ++i2)
    for (j2 = 0; j2 < N; ++j2)
      for (k2 = 0; k2 < N; ++k2)
S:      F[i2][j2] += D[i2][k2] * E[k2][j2];
  for (i3 = 0; i3 < N; ++i3)
    for (j3 = 0; j3 < N; ++j3)
      for (k3 = 0; k3 < N; ++k3)
T:      G[i3][j3] += C[i3][k3] * F[k3][j3];
```

Figure 1. ThreeMatMat: $C = AB, F = DE, G = CF$

We set $N = 512$ and computed 5 different versions of ThreeMatMat using combinations of loop transformations including tiling. We experimented on two platforms (quad-Xeon E7450 and quad-Opteron 8380), using the Intel ICC 11.0 compiler with aggressive optimization flags. The framework developed in this paper enables us to outperform ICC by a factor $2.28\times$ and $1.84\times$, respectively, for the Intel and AMD machines. We observe that the best version found depends on the target machine: for the Intel system, the best found loop fusion structure is shown in Figure 2(b), on which further polyhedral tiling and parallelization were applied (not shown in Figure 2). But on the AMD machine, distributing all statements and individually tiling them performs best, $1.23\times$ better than 2(b).

The efficient eecution of a computation kernel on a modern multi-core architecture requires the synergistic operation of many hardware resources, via the exploitation of thread-level parallelism; the memory hierarchy, including prefetch units, different cache levels, memory buses and interconnect; and all available computational units, including SIMD units. Because of the very complex interplay between all these components, it is currently infeasible to *guarantee at compile-time* which set of transformatios leads to maximal performance.

To maximize performance, one must carefully tune for the trade-off between the different levels of parallelism and the usage of the local memory components. This can be performed via loop fusion and distribution choices, that drive the success of subsequent optimizations such as vectorization, tiling or parallelization. It is *essential to adapt the fusion structure* to the program and target machine. However, the state-of-the-art provides only rough models of the impact of loop transformations on the actual execution. To achieve performance portability across a wide variety of architectures, empirical evaluation of several fusion choices is an alternative, but requires the construction and traversal of a search space of all possible fusion/distribution structures. To make the problem sound and tractable, we develop a complete framework to reason and optimize in the space of all semantics-preserving combinations of loop structures.

2.2 Background and notation

The *polyhedral model* is a flexible and expressive representation for loop nests. It captures the linear algebraic structure of statically predictable control flow, where loop bounds and conditionals are affine functions of the surrounding loop iterators and invariants (a.k.a. parameters, unknown at compilation time) [15, 20]; but it is not restricted to static control flow [2, 4].

The set of all executed instances of each statement is called an *iteration domain*. These sets are represented by affine inequalities involving the loop iterators and the global variables. Considering the ThreeMatMat example in Figure 1, the iteration domain of R is:

$$D_R = \{(i,j,k) \in \mathbb{Z}^3 \mid 0 \leq i < \mathrm{N} \wedge 0 \leq j < \mathrm{N} \wedge 0 \leq k < \mathrm{N}\}$$

D_R is a (parametric) integer polyhedron, that is a subset of $\mathbb{Z}^3$. The *iteration vector* $\vec{x}_R$ is the vector of the surrounding loop iterators, for R it is (i,j,k) and takes value in D_R.

In this work, a given loop nest optimization is defined by a multidimensional affine schedule, which is an affine form of the loop iterators and global parameters [18, 20].

DEFINITION 1 (Affine multi-dimensional schedule). *Given a statement S, an affine schedule Θ^S of dimension m is an affine form on the d outer loop iterators $\vec{x}_S$ and the p global parameters $\vec{n}$. $\Theta^S \in \mathbb{Z}^{m\times(d+p+1)}$ can be written as:*

$$\Theta^S(\vec{x}_S) = \begin{pmatrix} \theta_{1,1} & \dots & \theta_{1,d+p+1} \\ \vdots & & \vdots \\ \theta_{m,1} & \dots & \theta_{m,d+p+1} \end{pmatrix} \cdot \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix}$$

The scheduling function Θ^S maps each point in D_S with a multidimensional timestamp of dimension m, such that in the transformed code the instances of S defined in D_S will be executed following the lexicographic ordering of their associated timestamp. Multidimensional timestamps can be seen as logical clocks: the first dimension corresponds to days (most significant), next one is hours (less significant), the third to minutes, and so on. Note that every static control program has a multidimensional affine schedule [18], and that any composition of loop transformation can be represented in the polyhedral representation [20, 50]. The full program optimization is a collection of affine schedules, one for each syntactical program statement. Even seemingly non-linear transformations like loop tiling (a.k.a. blocking) and unrolling can be modeled [20, 38].

Finally, syntactic code is generated back from the polyhedral representation on which the optimization has been applied; CLOOG is the state-of-the art algorithm and code generator [3] to perform this task.

2.3 Semantics-preserving transformations

A program transformation must preserve the semantics of the program. The legality of affine loop nest transformations is defined at the level of dynamic statement instances. As a starting point for transformation space pruning, we build a *convex, polyhedral characterization of all semantics-preserving multidimensional affine schedules for a loop nest*.

Two statements instances are in *dependence relation* if they access the same memory cell and at least one of these accesses is a write. Given two statements R and S, a *dependence polyhedron* $D_{R,S}$ is a subset of the Cartesian product of D_R and D_S: $D_{R,S}$ contains all pairs of instances $\langle \vec{x}_R, \vec{x}_S \rangle$ such that $\vec{x}_S$ depends on $\vec{x}_R$, for a given array reference. Hence, for a transformation to preserve the program semantics, it must ensure that

$$\Theta^R(\vec{x}_R) \prec \Theta^S(\vec{x}_S),$$

where $\prec$ denotes the *lexicographic ordering*.[1]

To capture all program dependences we build a set of dependence polyhedra, one for each pair of array references accessing the same array cell (scalars being a particular case of array), thus possibly building several dependence polyhedra per pair of statements. The *polyhedral dependence graph* is a multi-graph with one node per statement, and an edge $e^{R \to S}$ is labeled with a dependence polyhedron $D_{R,S}$, for all dependence polyhedra.

The schedule constraints imposed by the precedence constraint are expressible as finding all non-negative functions over the dependence polyhedra [17]. It is possible to express the set of affine, non-negative functions over $D_{R,S}$ thanks to the affine form of the Farkas lemma [39].

LEMMA 1 (Affine form of Farkas Lemma). *Let D be a nonempty polyhedron defined by the inequalities $A\,\vec{x} + \vec{b} \geq \vec{0}$. Then any affine function $f(\vec{x})$ is non-negative everywhere in D iff it is a positive affine combination:*

$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T (A\vec{x} + \vec{b}), \text{ with } \lambda_0 \geq 0 \text{ and } \vec{\lambda}^T \geq \vec{0}.$$

λ_0 and $\vec{\lambda}^T$ are called Farkas multipliers.

Since it is a necessary and sufficient characterization, the Farkas Lemma offers a loss-less linearization of the constraints from the dependence polyhedron into direct constraints on the schedule coefficients. For instance considering the first time dimension (the first row of the scheduling matrices, that is, we set $p = 1$ in the following equation), to preserve the precedence relation we have:

$$\forall D_{R,S},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in D_{R,S}, \quad \Theta^S_p(\vec{x}_S) - \Theta^R_p(\vec{x}_R) \geq \delta_p^{D_{R,S}} \qquad (1)$$
$$\delta_p^{D_{R,S}} \in \{0,1\}$$

We resort to variable $\delta_1^{D_{R,S}}$ to model the dependence satisfaction. A dependence $D_{R,S}$ can be either *weakly satisfied* when $\delta_1^{D_{R,S}} = 0$, permitting $\Theta^S_1(\vec{x}_S) = \Theta^R_1(\vec{x}_R)$ for some instances in dependence, or *strongly satisfied* when $\delta_1^{D_{R,S}} = 1$, enforcing strict precedence (at the first time dimension) for all instances in dependence.

This model extends to multidimensional schedules, observing that once a dependence has been strongly satisfied, it does not contribute to the semantics preservation constraints for the subsequent time dimensions. Furthermore, for a schedule to preserve semantics, it is sufficient for every dependence to be strongly satisfied at least once. Following these observations, one may state a sufficient condition for semantics preservation (adapted from Feautrier's formalization [18]).

[1] $(a_1,\ldots,a_n) \prec (b_1,\ldots,b_m)$ iff there exists an integer $1 \leq i \leq min(n,m)$ s.t. $(a_1,\ldots,a_{i-1}) = (b_1,\ldots,b_{i-1})$ and $a_i < b_i$.

LEMMA 2 (Semantics-preserving affine schedules). *Given a set of affine schedules $\Theta^R, \Theta^S \ldots$ of dimension m, the program semantics is preserved if:*

$$\forall D_{R,S},\ \exists p \in \{1,\ldots,m\},\ \delta_p^{D_{R,S}} = 1 \ \wedge\ \left(\forall j < p,\ \delta_j^{D_{R,S}} = 0\right) \wedge$$
$$\left(\forall j \leq p, \forall \langle \vec{x}_R, \vec{x}_S \rangle \in D_{R,S},\ \Theta^S_j(\vec{x}_S) - \Theta^R_j(\vec{x}_R) \geq \delta_j^{D_{R,S}}\right)$$

The proof directly derives from the lexicopositivity of dependence satisfaction [18].

Regarding the schedule dimensionality m, it is sufficient to pick $m = d$ to guarantee the existence of a legal schedule (the maximum program loop depth is d). Obviously, any schedule implementing the original execution order is a valid solution [18].

This formalization involves an "oracle" to select the dimension p at which each dependence should be strongly satisfied. To avoid the combinatorial selection of this dimension, we conditionally nullify constraint (1) on the schedules *when the dependence was strongly satisfied at a previous dimension*. To nullify the constraint, we may always pick a lower bound lb for $\Theta^S_k(\vec{x}_S) - \Theta^R_k(\vec{x}_R)$. Without any loss of generality, we assume (parametric) loop bounds are non-negative. Vasilache [44] proposed a method to evaluate this lower bound when the coefficients of Θ are bounded: lb is greater or equal to the sum of the maximal coefficient values times the (possibly parametric) loop bounds. We can always select a large enough K such that [32]

$$\Theta^S_k(\vec{x}_S) - \Theta^R_k(\vec{x}_R) \geq -K.\vec{n} - K \qquad (2)$$

Note that this lower bound can also be linearized into constraints on Θ^R, Θ^S thanks to the Farkas Lemma. To obtain the schedule constraints we reinsert this lower bound in the previous formulation [44], such that either the dependence has not been previously strongly satisfied and then the lower bound is (1), or it has been and the lower bound is (2). We thus derive the convex form of semantics-preserving affine schedules of dimension m for a program as a corollary of Lemma 2.

LEMMA 3 (Convex form of semantics-preserving affine schedules). *Given a set of affine schedules $\Theta^R, \Theta^S \ldots$ of dimension m, the program semantics is preserved if the three following conditions hold:*

$$(i) \qquad \forall D_{R,S},\ \forall p,\ \delta_p^{D_{R,S}} \in \{0,1\}$$

$$(ii) \qquad \forall D_{R,S},\ \sum_{p=1}^{m} \delta_p^{D_{R,S}} = 1 \qquad (3)$$

$$(iii) \qquad \forall D_{R,S},\ \forall p \in \{1,\ldots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in D_{R,S}, \qquad (4)$$
$$\Theta^S_p(\vec{x}_S) - \Theta^R_p(\vec{x}_R) \geq -\sum_{k=1}^{p-1} \delta_k^{D_{R,S}}.(K.\vec{n} + K) + \delta_p^{D_{R,S}}$$

One can then build the set L_m of all (bounded) legal affine multidimensional schedules of dimension m by encoding the affine constraints given by Lemma 3 into a problem with m binary variables per dependence and $m \times (dim(\vec{x}_S) + dim(\vec{n}) + 1)$ variables per statement S. For an efficient construction of the constraints for semantics-preservation, one should proceed dependence by dependence. Building the constraints of the form of (4) is done for each dependence, then the Farkas multipliers are eliminated for instance with a scalable Fourier-Motzkin projection technique [33]. The set of constraints obtained for each schedules are then intersected, and the process is replicated for all dimensions.

2.4 Finding a schedule for the program

We achieved completeness: an Integer Linear Program operating on L_m can exhibit the best affine schedule according to a given objective function. Yet solving an arbitrary ILP on L_m is NP-complete [39]. Given the high dimensionality of this space on

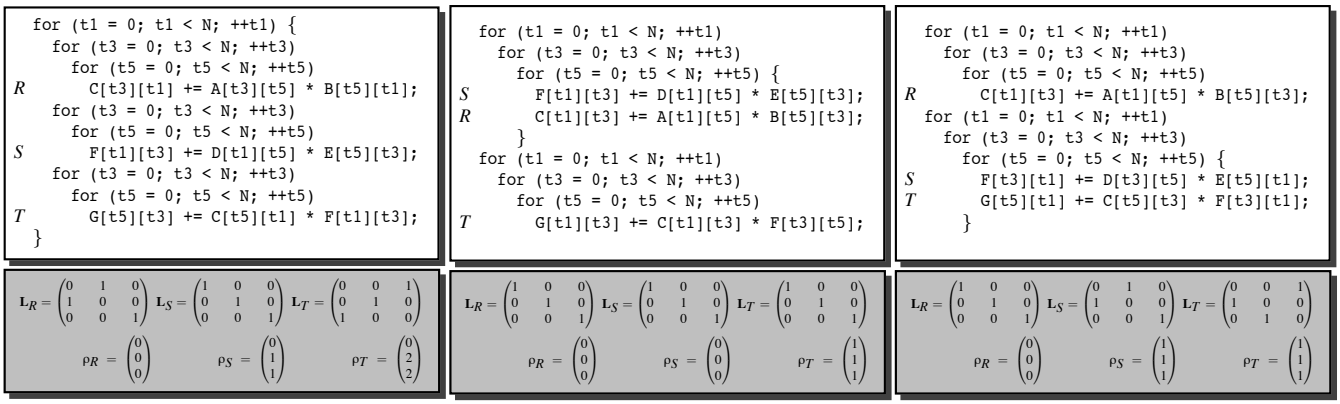

Figure 2. Three possible legal transformations for $C = AB, F = DE, G = CF$

the larger loop nests, our convex formalization induces a serious tractability challenge. To address this challenge and reduce the complexity effect of the problem, we propose to decouple the optimization into two sub-problems:

1. the problem of selecting how statements are interleaved — this is traditionally viewed as selecting a proper combination of loop fusion, loop distribution and code motion;
2. the problem of selecting an affine schedule compatible with the statement interleaving selected at the previous step.

Selecting a statement interleaving that maximizes a fusion criterion is still an NP-complete problem in general [12], but its complexity is dramatically reduced from the high-dimensionality linear optimization problem. We focus our efforts on this simplified problem.

2.5 Encoding statement interleaving

Fusion and fusibility of statements In the polyhedral model, loop fusion is characterized by the fine-grain interleaving of statement instances [6, 24]. Two statements are fully distributed if the range of the timestamps associated to their instances never overlap. Syntactically, this results in distinct loops to traverse the domains. One may define fusion as the negation of the distribution criterion. For such a case we say that two statements R,S are fused if there exists at least one pair of iterations for which R is scheduled before S, and another pair of iterations for which S is scheduled before R. This is stated in Definition 2.

DEFINITION 2 (Fusion of two statements). *Consider two statements R,S and their schedules Θ^R and Θ^S. R and S are said to be fused at level p if, $\forall k \in \{1 \dots p\}$, there exists at least three distinct executed instances $\vec{x_R}, \vec{x_R}'$ and $\vec{x_S}$ such that*

$$\Theta^R_k(\vec{x_R}) \leq \Theta^S_k(\vec{x_S}) \leq \Theta^R_k(\vec{x_R}'). \tag{5}$$

We now propose an encoding of Definition 2 such that we can determine, from the dependence graph, if it is possible to find a schedule leading to fuse the statements whatever additional transformation(s) may be required to enable fusion. This is called *fusibility*. The purpose is to exhibit sufficient affine constraints that can be added to L to keep in the solution space only the schedules which correspond to fusing the considered statements.

We rely on the fact that the first and last scheduled instances of R and S will conform to Definition 2 if the two statements are fused. These specific instances belong to the set of vertices of D_R and D_S. Thus, to determine whether R and S are fusable or not, it is sufficient to enumerate all vertices of D_R and D_S, and for each possible combinations of vertices plug their actual value into Eq (5) — leading to an affine constraint on the schedule coefficients — and test for the existence of a solution in the space of all semantics-preserving schedules L augmented with the new constraint. As soon as there is a solution to one of the problems formed, then the statements are fusable. This is stated in Definition 3.

DEFINITION 3 (Generalized fusibility check). *Given v_R (resp. v_S) the set of vertices of D_R (resp. D_S). R and S are fusable at level p if, $\forall k \in \{1 \dots p\}$, there exist two semantics-preserving schedules Θ^R_k and Θ^S_k such that*

$$\exists (\vec{x}_1, \vec{x}_2, \vec{x}_3) \in v_R \times v_S \times v_R, \quad \Theta^R_k(\vec{x}_1) \leq \Theta^S_k(\vec{x}_2) \leq \Theta^R_k(\vec{x}_3)$$

Note that in practice, the number of vertices of an iteration domain is often small (from 2 to 10), hence the number of cases to be checked is limited. We also present in Section 4.2 a specialization of this test to the case of non-negative schedule coefficients, which removes the need to enumerate the vertices.

Multidimensional statement interleaving Thanks to Lemma 3 and Definition 3, we can determine if a set of statements can be fused at a given level while preserving semantics. For our optimization problem, we are interested in building a space representing all ways to interleave the program statements.

To reason about statement interleaving, one may associate a vector ρ^S of dimension d (d being the maximal loop depth in the program) to each statement S, and order those vectors lexicographically. If some statement S is surrounded by less than d loops, ρ^S is post-padded with zeroes. Figure 2 gives an example of three possible optimizations for the ThreeMatMat example, as defined by different configurations of the ρ vectors. Note that to implement the interleaving specified by ρ several loop permutations may be required.

There exist an analogy between ρ^S and a standard encoding of multidimensional affine schedules where statement interleaving vectors are made explicit: for each statement S, one may constrain the rows of Θ^S to alternate between constant forms of a vector of dimension $d+1$ (usually denoted β [20]) and affine forms of the iteration and global parameter vectors. This $2d+1$-dimensional encoding does not incur any loss of expressiveness [11, 18, 20, 24]. However in this paper we explicitly give ρ important structural properties of the transformed loop nest:

1. if $\rho^R_k = \rho^S_k$, $\forall k \in \{1, \dots, p\}$ then the statements share (at least) p common loops;

2. if $\rho_p^R \neq \rho_p^S$, then the statements do not share any common loop starting at depth p. We thus have

$$\rho_p^R \neq \rho_p^S \Rightarrow \rho_k^R \neq \rho_k^S,\ \forall k \in \{p+1,\ldots,m\}$$

Our objective to model all possible interleavings is to build a space containing all ρ vectors for which there exists a semantics-preserving affine transformation that implements the specified interleaving. To guarantee that the solution space contains only useful points, we need to address the problem that several choices of ρ vectors represent the same multidimensional statement interleaving. Intuitively, the problem arises because there exists an infinite number of vectors which have the same lexicographic ordering. In particular, the order is invariant to translation of all coefficients of ρ at a given dimension, or by multiplication of all its coefficients by a non-negative constant. To abstract away these equivalences, we now formally define the concept of multidimensional statement interleaving.

DEFINITION 4 (Multidimensional statement interleaving). *Consider a set of statements S enclosed within at most d loops and their associated vectors $R = \{\rho^S\}_{S \in S}$. For a given $k \in \{1,\ldots,d\}$, the one-dimensional statement interleaving of S at dimension k defined by R is the partition of S according to the coefficients ρ_k^S. The multidimensional statement interleaving of S at dimension k defined by R is the list of partitions at dimension k.*

The structural properties of statement interleaving indicate that equivalence classes at dimension k correspond to statements that share common loops at depth k in the transformed loop nest.

DEFINITION 5 (Total preorder). *A total preorder on a set S is a relation $\lhd$ which is reflexive, transitive, and such that for any pair of elements $(S_1, S_2) \in S$, either $S_1 \lhd S_2$ or $S_2 \lhd S_1$ or both.*

An important result is that any preorder of a set S is isomorphic to a partial order of some equivalence classes of S. Applying this result to the structural properties of statement interleavings yields the following lemma.

LEMMA 4 (Structure of statement interleavings). *Each one-dimensional statement interleaving corresponds to a unique total preorder of the statements and reciprocally.*

3. Semantics-Preserving Statement Interleavings

We now propose a convex, complete characterization of multidimensional statement interleavings which serves as a basis to encode the space of valid ρ vectors.

3.1 Convex encoding of total preorders

We first present a solution for the problem of total preorders of scalar elements, that corresponds to one-dimensional interleavings.

One-dimensional case For a given set of n elements, we define O as the set of all and distinct total preorders of its n elements. The key problem is to model O as a convex polyhedron.[2]

To the best of our knowledge, uniqueness of orderings cannot be modeled in a convex fashion on a set with a linear number of variables. We propose to model the ordering of two elements i, j with three *binary* decision variables, defined as follows. $p_{i,j} = 1$ iff i precedes j, $e_{i,j} = 1$ iff i equals j and $s_{i,j} = 1$ iff i succeeds j. To model the entire set, we introduce three binary variables for each ordered pair of elements, i.e., all pairs (i, j) such that $1 \leq i < j \leq n$. This models a set with $3 \times n(n-1)/2$ variables.

[2] This problem is not a straight adaptation of standard order theory: we look for the *set* of *all distinct* total preorders of n elements, in contrast to classical work defining counting functions of this set [41].

$$\left\{ \begin{array}{l} 0 \leq p_{i,j} \leq 1 \\ 0 \leq e_{i,j} \leq 1 \\ 0 \leq s_{i,j} \leq 1 \end{array} \right\}$$

For instance, the outer-most interleaving $\rho_1^R = 0,\ \rho_1^S = 0,\ \rho_1^T = 1$ of Figure 2(b) is represented by:

$$e_{R,S} = 1,\ e_{R,T} = 0,\ e_{S,T} = 0$$
$$p_{R,S} = 0,\ p_{R,T} = 1,\ p_{S,T} = 1$$
$$s_{R,S} = 0,\ s_{R,T} = 0,\ s_{S,T} = 0$$

From there, one may easily recompute the corresponding total preorder $\{\rho_1^R = 0, \rho_1^S = 0, \rho_1^T = 1\}$, e.g., by computing the lexicographic minimum of a system W of 3 non-negative variables that replicate the ordering defined by all $p_{i,j}$, $e_{i,j}$ and $s_{i,j}$.

The first issue is the consistency of the model: an inconsistent preorder would make W infeasible, e.g., setting $e_{1,2} = 1$ and $p_{1,2} = 1$. The second issue is the totality of the relation. These two conditions can be merged into the the following equality, capturing both mutual exclusion and totality:

$$p_{i,j} + e_{i,j} + s_{i,j} = 1 \tag{6}$$

To simplify the system, we immediately get rid of the $s_{i,j}$ variables, thanks to (6). We also relax (6) to get:

$$p_{i,j} + e_{i,j} \leq 1$$

Mutually exclusive decision variables capture the consistency of the model for a single pair of elements. However, one needs to insert additional constraints to capture transitivity.

To enforce transitivity, the following rule must hold true for all triples of statements (i, j, k):

$$e_{i,j} = 1 \wedge e_{i,k} = 1 \Rightarrow e_{j,k} = 1.$$

Similarly, we have:

$$e_{i,j} = 1 \wedge e_{j,k} = 1 \Rightarrow e_{i,k} = 1.$$

These two rules set the basic transitivity of e variables. Since we are dealing with binary variables, the implications can be easily modeled as affine constraints:

$$\left\{ \begin{array}{ll} \forall k \in]j,n], & e_{i,j} + e_{i,k} \leq 1 + e_{j,k} \\ & e_{i,j} + e_{j,k} \leq 1 + e_{i,k} \end{array} \right\}$$

Slightly more complex transitivity patterns are also required. For instance, we also have transitivity conditions imposed by a connection between the value for some e coefficients and some p ones. For instance, $i < j$ and $j = k$ implies $i < k$. The general rules for those cases are:

$$e_{i,j} \wedge p_{i,k} \Rightarrow p_{j,k}$$
$$e_{i,j} \wedge p_{j,k} \Rightarrow p_{i,k}$$
$$e_{k,j} \wedge p_{i,k} \Rightarrow p_{i,j}$$

These three rules set the complex transitivity between the p and e variables. They can be modeled equivalently by following affine constraints:

$$\left\{ \begin{array}{ll} \forall k \in]j,n] & e_{i,j} + p_{i,k} \leq 1 + p_{j,k} \\ & e_{i,j} + p_{j,k} \leq 1 + p_{i,k} \\ \forall k \in]i,j[& e_{k,j} + p_{i,k} \leq 1 + p_{i,j} \end{array} \right\} \tag{7}$$

Generalizing this reasoning, we collect all constraints to enforce the transitivity of the total preorder relation. Those constraints are gathered in the following system, for $1 \leq i < j < n$, defining the convex set of all, distinct total preorders of n elements O:

$$\left\{\begin{array}{lll}
 & 0 \le p_{i,j} \le 1 \\ & 0 \le e_{i,j} \le 1 & \text{Variables are binary} \\
 & p_{i,j} + e_{i,j} \le 1 & \text{Relaxed mutual exclusion} \\
\forall k \in]j,n] & e_{i,j} + e_{i,k} \le 1 + e_{j,k} \\ & e_{i,j} + e_{j,k} \le 1 + e_{i,k} & \text{Basic transitivity on } e \\
\forall k \in]i,j[& p_{i,k} + p_{k,j} \le 1 + p_{i,j} & \text{Basic transitivity on } p \\
\forall k \in]j,n] & e_{i,j} + p_{i,k} \le 1 + p_{j,k} \\ & e_{i,j} + p_{j,k} \le 1 + p_{i,k} \\ \forall k \in]i,j[& e_{k,j} + p_{i,k} \le 1 + p_{i,j} & \text{Complex transitivity on } p \text{ and } e \\
\forall k \in]j,n] & e_{i,j} + p_{i,j} + p_{j,k} \le 1 + p_{i,k} + e_{i,k} & \text{Complex transitivity on } s \text{ and } p
\end{array}\right.$$

The following lemma is proved in [32, appendix A].

LEMMA 5 (Completeness and correctness of O). *The set O contains one and only one element per distinct total preorder of n elements.*

Multidimensional case O encodes all possible total preorders (or, statement interleaving) for a given loop level. To extend to the multidimensional interleaving case, we first replicate O for each dimension of the ρ vectors. However, further constraints are needed to also achieve consistency and uniqueness of the characterization across dimensions. Intuitively, if a statement is distributed at dimension k, then for all remaining dimensions it will also be distributed. This is modeled with the following equations:

$$\forall l > k,\ \left(p^k_{i,j} = 1 \Rightarrow p^l_{i,j} = 1\right) \wedge \left(s^k_{i,j} = 1 \Rightarrow s^l_{i,j} = 1\right)$$

The final expression of the set I of all, distinct d-dimensional statement interleavings is:

$$\left\{\begin{array}{ll}
\forall k \in \{1,\ldots,d\}, \quad \text{constraints on } O^k & \text{Total preorders at level } k \\
 & \\
p^k_{i,j} \le p^{k+1}_{i,j} \\ e^{k+1}_{i,j} + p^{k+1}_{i,j} \le p^k_{i,j} + e^k_{i,j} & \text{Statement interleaving uniqueness}
\end{array}\right.$$

It is worth noting that each O^k contains numerous variables and constraints; but when it is possible to determine — thanks to the dependence graph for instance — that a given ordering of two elements i and j is impossible, some variables and constraints are eliminated. Our experiments indicate that these simplifications are quite effective, improving the scalability of the approach significantly.

3.2 Pruning for semantics preservation

The set I contains all and only distinct multi-level statement interleavings. A pruning step is needed to remove all interleavings that does not preserve the semantics. The algorithm proceeds level-by-level, from the outermost to the innermost. Such a decoupling is possible because we have encoded multi-dimensional interleaving constraints in I, and that fusion at level k implies fusion at all preceding levels. In addition, leveraging Lemma 3 and Definition 3 we can determine the *fusibility* of a set of statements at a given level by exposing sufficient conditions for fusion on the schedules.

The general principle is to detect all the smallest possible sets of p unfusable statements at level k, and for each of them, to update I^k by adding an affine constraint of the form:

$$e^k_{S_1,S_2} + e^k_{S_2,S_3} + \ldots + e^k_{S_{p-1},S_p} < p - 1 \qquad (8)$$

thus preventing them (and any super-set of them) to be fused all together. We note F the final set with all pruning constraints for legality, $F \subseteq I$. A naive approach could be to enumerate all unordered subsets of the n statements of the program at level k, and check for fusibility, while avoiding to enumerate a super-set of an unfusable set. Instead, we leverage the polyhedral dependence graph to reduce the test to a much smaller set of structures.

The first step of our algorithm is to build a graph G to facilitate the enumeration of sets of statements to test for, with one node per statement. Sets of statements to test for fusibility will be represented as nodes connected by a path in that graph. Intuitively, if G is a complete graph then we can enumerate all unordered subsets of the n statements: enumerating all paths of length 1 gives all pairs of statements by retrieving the nodes connected by a given path, all paths of length 2 gives all triplets, etc. We aim at building a graph with less edges, so that we lower the number of sets of statements to test for fusibility. We first check the fusibility of all possible pairs of statements, and add an edge between two nodes only if (1) there is a dependence between them, and (2) either they must be fused together or they can be fused and distributed at that level. When two statements must be fused, they are merged to ensure all schedule constraints are considered when checking for fusibility.

The second step is to enumerate all paths of length ≥ 2 in the graph. Given a path p, the nodes in p represent a set of statements that has to be tested for fusibility. Each time they are detected to be not fusable, all paths with p as a sub-path are discarded from enumeration, and F^k is updated with an equation in the form of (8). The complete algorithm is shown in Figure 3.

```
PruneIllegalInterleavings: Compute F
Input:
  pdg: polyhedral dependence graph
  n: number of statements
  maxDepth: maximum loop depth
Output:
  F: the space of semantics-preserving distinct interleavings

1   F ← I
2   unfusable ← ∅
3   for d ← 1 to maxDepth do
4     G ← newGraph(n)
5     forall pairs of dependent statements R,S do
6       L_{R,S} ← buildLegalSchedules({R,S}, pdg)
7       if mustFuse(L_{R,S}, d) then
8         F^d ← F^d ∩ {e^d_{R,S} = 1}
9       elseif mustDistribute(L_{R,S}, d) then
10        F^d ← F^d ∩ {e^d_{R,S} = 0}
11      else
12        if ¬ canDistributeAndSwap(L_{R,S}, d) then
13          F^d ← F^d ∩ {e^d_{R,S} + p^d_{R,S} = 1}
14        end if
15        addEdge(G, R, S)
16      end if
17    end for
18    forall pairs of statements R,S such that e^d_{R,S} = 1 do
19      mergeNodes(G, R, S)
20    end for
21    for l ← 2 to n−1 do
22      forall paths p in G of length l such that
              there is no prefix of p in unfusable do
23        L_{nodes(p)} ← buildLegalSchedules({nodes(p)}, pdg)
24        if mustDistribute(L_{nodes(p)}, d) then
25          F^d ← F^d ∩ {Σ_p e^d_{pairs in p} < l−1}
26          unfusable ← unfusable ∪ p
27        end if
28      end for
29    end for
30  end for
```

Figure 3. Pruning algorithm

Procedure `buildLegalSchedules` computes the space of legal schedules L according to Lemma 3, for the set of statements given in argument. Procedure `mustDistribute` tests for the emptiness of L when augmented with fusion conditions from Definition 3 up to level d. If there is no solution in the augmented set of constraints, then the statements cannot be fused at that level and hence must be distributed. Procedure `mustFuse` checks if it is legal to distribute the statements R and S. The check is performed by inserting a *splitter* at level d. This splitter is a constant one-dimensional schedule at level d, to force the full distribution of statement instances at that level. If there is no solution in this set of constraints, then the statements cannot be distributed at that level and hence must be fused. Procedure `canDistributeAndSwap` tests if it is legal to distribute R and S at level d and to execute S before R. The latter is required to compute the legal values of the $s_{R,S}$ variables at that level. The check is performed in a similar fashion as with `mustFuse`, except the splitter is made to make R execute after S. Finally procedure `mergeNodes` modifies the graph edges after the merging of two nodes R and S such that: (1) if for T there was an edge $e^{T\to R}$ and not $e^{T\to S}$ or vice-versa, $e^{T\to R}$ is deleted, and $e_{S,T}=0$, this is to remove triplets of trivially unfusable sets; (2) if there are 2 edges between T and RS, one of them is deleted and its label is added to the remaining one existing label; (3) the label of the edge $e^{R\to S}$ is added to all remaining edges to/from RS, and $e^{R\to S}$ is deleted.

Applications The first motivation of building a separate search space of multidimensional statement interleavings is to decouple the selection of the interleaving from the selection of the transformation that enables this interleaving. One can then focus on building search heuristics for the fusion/distribution of statements only, and through the framework presented in this paper compute a schedule that respects this interleaving. Additional schedule properties such as parallelism and tilability can then be exploited without disturbing the outer level fusion scheme. We present in the following a complete technique to optimize programs based on iterative interleaving selection, leading to parallelized and tiled transformed programs. This technique is able to automatically adapt to the target framework, and successfully discovers the best performing fusion structure, whatever the specifics of the program, compiler and architecture.

Another motivation of building $\boldsymbol{I}$ is to enable the design of objective functions on fusion with the widest degree of applicability. For instance one can maximize fusion at outer level, by maximizing $\sum_{i,j} e^1_{i,j}$ or similarly distribution by minimizing the same sum. One can also assign a weight to the coefficients $e_{i,j}$ to favor fusion for statements carrying more reuse, for instance. This formulation allows to devise further pruning algorithms, offering to the optimization the most relevant choice of legal interleavings for a given target.

4. Optimizing for Locality and Parallelism

The previous sections define a general framework for multi-level statement interleaving. We address now the problem of *specializing* this framework to provide a complete optimization algorithm that integrates tiling and parallelization, along with the possibility to iteratively select different interleavings.

The optimization algorithm proceeds recursively, from the outermost level to the innermost. At each level of the recursion, we *select* the associated schedule dimension by instantiating its values. Then, we build the set of semantics-preserving interleavings at that level, pick one and proceed to the next level until a full schedule is instantiated. We present in Section 4.1 additional conditions on the schedules to improve the performance of the generated transformation, by integrating parallelism and tilability as criteria. Then we define in Section 4.2 a new fusibility criterion for a better performance impact, while restricting it to non-negative schedule coefficients. In Section 4.3, we show how to construct the set of legal interleaving without resorting on testing the set of legal schedules for sets larger than a pair of statements. Finally we present the complete optimization algorithm in Section 4.4.

4.1 Additional constraints on the schedules

Tiling (or blocking) is a crucial loop transformation for parallelism and locality. Bondhugula et al. developed a technique to compute an affine multidimensional schedule such that parallel loops are brought to the outer levels, and loops with dependences are pushed inside [6, 8]; at the same time, the number of dimensions that can be tiled are maximized. We extend and recast their technique into our framework.

Legality of tiling Tiling along a set of dimensions is legal if it is legal to proceed in fixed block sizes along those dimensions: this requires dependences not to be backward along those dimensions, thus avoiding a dependence path going out of and coming back into a tile; this makes it legal to execute the tile atomically. Irigoin and Triolet showed that a sufficient condition for a schedule Θ to be tilable [23], given R the dependence cone for the program, is that

$$\Theta.R \geq \vec{0}$$

In other words, this is equivalent to saying that all dependences must be weakly satisfied for each dimension Θ_k of the schedule. Such a property for the schedule is also known as Forward Communication Only property [21]. Returning to Lemma 3, it is possible to add an extra condition such that the p first dimensions of the schedules are permutable, a sufficient condition for the p first dimensions to be tilable. This translates into the following additional constraint on schedules, to enforce permutability of schedule dimensions.

DEFINITION 6 (Permutability condition). *Given two statements R,S. Given the conditions for semantics-preservation as stated by Lemma 3. Their schedule dimensions are permutable up to dimension k if in addition:*

$$\forall D_{R,S},\ \forall p \in \{1,\ldots,k\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in D_{R,S},$$
$$\Theta^S_p(\vec{x}_S) - \Theta^R_p(\vec{x}_R) \geq \delta_p^{D_{R,S}}$$

To translate k into actual number of permutable *loops*, the k associated schedule dimensions must express non-constant schedules.

Rectangular tiling Selecting schedules such that each dimension is independent with respect to all others enables a more efficient tiling. Rectangular or close to rectangular blocks are achieved when possible, avoiding complex loop bounds which may arise in the case of arbitrarily shaped tiles. We resort to augmenting the constraints, level-by-level, with independence constraints. Hence, to compute schedule dimension k, we need *instantiate* first a schedule for all previous dimensions 1 to $k-1$. This comes from the fact that orthogonality constraints are not linear or convex and cannot be modeled as affine constraints directly. In its complete form, adding orthogonality constraints 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 non-negative (that is, we restrict to have $\theta_{i,j} \in \mathbb{N}$). 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 strong difference in practice. For the rest of this paper, **we now assume** $\theta_{i,j} \in \mathbb{N}$.

Inner permutable loops If it is not possible to express permutable loops for the first level, Bondhugula proposed to split the statements into distinct blocks to increase the possibility to find outer permutable loops [8]. Since our technique already supports explicitly the selection of any semantics-preserving possibility to split statements into blocks via the statement interleaving, we propose instead to enable the construction of *inner* permutable loops, by choosing to maximize the number of dependences solved at the first levels until we (possibly) find permutable dimensions at the current level. Doing so increases the freedom for the schedule at inner dimensions when it is not possible to express permutable loops at the outer levels. Maximizing the number of dependences solved at a given level was introduced by Feautrier [18] and we use a similar form:

$$S^i = \max \sum_{D_{R,S}} \delta_k^{D_{R,S}} \tag{9}$$

This cost function replaces the permutability condition, when it is not possible to find a permutable schedule for a given dimension k.

Dependence distance minimization There are infinitely many schedules that may satisfy the permutability criterion from Definition 6 as well as (9). An approach that has proved to be simple, practical, and powerful has been to find those schedules that have the shortest dependence components along them [6]. For polyhedral code, the distance between dependent iterations can always be bounded by an affine function of the global parameters, represented as a p-dimensional vector $\vec{n}$.

$$\mathbf{u}.\vec{n} + w \geq \Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \quad \langle \vec{x}_R, \vec{x}_S \rangle \in D_{R,S} \quad \mathbf{u} \in \mathbb{N}^p, w \in \mathbb{N} \tag{10}$$

In order to maximize data locality, one may include read-after-read dependences in the set of dependences $D_{R,S}$ considered in the bounding function (10). We showed in (2) that there always exists a parametric form $-K.\vec{n} - K$ for the lower bound of the schedule latency. Reciprocally, $\mathbf{u}$ and w can always be selected large enough for $\mathbf{u}.\vec{n} + w$ to be an upper-bound on the schedule latency. So considering read-after-read dependences in the bounding function does not prevent from finding a legal schedule.

The permutability and bounding function constraints are recast through the affine form of the Farkas Lemma such that the only unknowns left are the coefficients of Θ and those of the bounding function, namely $\mathbf{u}$, w. Coordinates of the bounding function are then used as the minimization objective to obtain the unknown coefficients of Θ.

$$\text{minimize}_{\prec} \left(\mathbf{u}, w, \ldots, \theta_{i,1}, \ldots\right) \tag{11}$$

The resulting transformation is a complex composition of multi-dimensional loop fusion, distribution, interchange, skewing, shifting and peeling. Eventually multidimensional tiling is applied on all permutable bands, and resulting tiles can be executed in parallel or at worse with a pipeline-parallel schedule [8]. Tile sizes are computed such that data accessed by each tile roughly fits in the L1 cache.

4.2 Fusability check

We presented in Section 2.5 a generalized check for fusibility which guarantees at least one instance of the iteration domains are fused. But for fusion to have a stronger performance impact, a better criterion is preferred to guarantee that at most x instances are *not* finely interleaved. In general, computing the exact set of interleaved instances requires complex techniques. A typical example is schedules generating a non-unit stride in the supporting lattice of the transformed iteration domain. For such cases, computing the exact number of non-fused instances could be achieved using the Ehrhart quasi-polynomial of the intersection of the image of iteration domains by the schedules [10]. However, this refined precision is not required to determine if a schedule represents a potentially interesting fusion. We allow for a lack of precision to present an easily computable test for fusibility based on an *estimate* of the number of instances that are not finely interleaved.

For the sake of simplicity, we assume that loop bounds have been normalized such that $\vec{0}$ is the first iteration of the domain. When considering non-negative coefficients, the lowest timestamp assigned by a schedule Θ_k^R is simply $\Theta_k^R(\vec{0})$. One can recompute the corresponding timestamp by looking at the values of the coefficients attached to the parameters and the constant. Hence, the timestamp interval c between the two first scheduled instances by Θ_k^R and Θ_k^S is simply the difference of the (parametric) constant parts of the schedules. In addition, to avoid the case where Θ_k^R and/or Θ_k^S are (parametric) constant schedules, we force the linear part of the schedule to be non-null. This is formalized in Definition 7.

DEFINITION 7 (Fusability restricted to non-negative coefficients). *Given two statements R,S such that R is surrounded by d^R loops, and S by d^S loops. They are fusable at level p if, $\forall k \in \{1 \ldots p\}$, there exist two semantics-preserving schedules Θ_k^R and Θ_k^S such that:*

$$(i) \quad -c < \Theta_k^R(\vec{0}) - \Theta_k^S(\vec{0}) < c$$

$$(ii) \quad \sum_{i=1}^{d^R} \theta_{k,i}^R > 0, \ \sum_{i=1}^{d^S} \theta_{k,i}^S > 0$$

The constant c is an indicator on the timestamp difference between the first scheduled instance of R and the first scheduled instance of S. By tuning the allowed size for this interval, one can range from full, aligned fusion ($c = 0$) to full distribution (with c greater than the schedule latency of R or S). Note that Definition 7 is independent from any composition of affine transformations that may be needed to enable the fusion of the statements. The schedules that implement the fusion (hence modeling also the fusion-enabling sequence of transformations) are simply solutions in the space of semantics-preserving non-negative schedules augmented with the fusibility constraints.

4.3 Computation of the set of interleavings

We have now restricted $\theta_{i,j} \in \mathbb{N}$. A consequence is the design of a *specialized* version of our generic algorithm to compute the set of semantics-preserving interleavings. This specialized version leverages new results on the transitivity of fusibility for the case of non-negative schedule coefficients, as shown below. Furthermore, it does not require computing with L, which significantly improves the overall complexity of the procedure.

Fusability is the capability to exhibit a semantics-preserving schedule such that some of the instances are fused, according to Definition 7. First let us remark that fusibility is not a transitive relation. As an illustration, consider the sequence of matrix-by-vector products $x = Ab,\ y = Bx,\ z = Cy$. While it is possible to fuse them 2-by-2, it is not possible to fuse them all together. When considering fusing loops for $x = Ab,\ y = Bx$, one has to permute loops in $y = Bx$. When considering fusing loops for $y = Bx,\ z = Cy$, one has to keep loops in $y = Bx$ as is. We now present a sufficient condition to determine the transitivity of fusibility, based on the existence of compatible pairwise permutations.

First we introduce a decomposition of *one-dimensional schedules* in two sub-parts, with the objective of isolating loop permutation from the other transformations embedded in the schedule. One can decompose a one-dimensional schedule Θ_k^R with coefficients in

$\mathbb{N}$ into two sub-schedules μ^R and λ^R such that:

$$\Theta_k^R = \mu^R + \lambda^R,\ \ \mu_i^R \in \mathbb{N},\ \ \lambda_i^R \in \mathbb{N}$$

without any loss of expressiveness. Such a decomposition is always possible because of the distributivity of the matrix multiplication over the matrix addition. For our purpose, we are interested in modeling one-dimensional schedules which are *not* constant schedules. This is relevant as we do not want to consider building a schedule for fusion that would translate only into statement interleaving. On the contrary we aim at building a schedule that performs the interleaving of statements *instances*, hence the linear part of the schedule must be non-null. For R surrounded by d loops, we enforce μ to be a linear form of the d loop iterators:

$$\mu^R(\vec{x}_R) = \begin{pmatrix}\mu_1^R & \dots & \mu_d^R & \vec{0} & 0\end{pmatrix} \cdot \begin{pmatrix}i_1 & \dots & i_d & \vec{n} & 1\end{pmatrix}^t$$

To model non-constant schedules, we add the additional constraint $\sum_{i=1}^{d} \mu_i^R = 1$. Note that by constraining μ to have only one coefficient set to 1, this does not prevent to model any compositions of slowing or skewing: these would be embedded in the λ part of the schedule, as shown in the example below.

The μ part of the schedule models different cases of loop permutations. For instance for statement R surrounded by 3 loops in the ThreeMatMat example, μ^R can take only three values:

$$\mu^R(\vec{x}_R) = \begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix} \cdot \begin{pmatrix}i & j & k & N & 1\end{pmatrix}^t = (i)$$
$$\mu^R(\vec{x}_R) = \begin{pmatrix}0 & 1 & 0 & 0 & 0\end{pmatrix} \cdot \begin{pmatrix}i & j & k & N & 1\end{pmatrix}^t = (j)$$
$$\mu^R(\vec{x}_R) = \begin{pmatrix}0 & 0 & 1 & 0 & 0\end{pmatrix} \cdot \begin{pmatrix}i & j & k & N & 1\end{pmatrix}^t = (k)$$

while λ^R can take arbitrary values. For better illustration let us now build the decomposition of the schedule $\Theta_k^R = (2.j + k + 2)$. Θ_k^R is the composition of a permutation, a non-unit skewing and a shifting, and can be decomposed as follows:

$$\mu^R = \begin{pmatrix}0 & 1 & 0 & 0 & 0\end{pmatrix}$$
$$\lambda^R = \begin{pmatrix}0 & 1 & 1 & 0 & 2\end{pmatrix}$$
$$\Theta_k^R(\vec{x}_R) = (\mu^R + \lambda^R)(\vec{x}_R) = (2.j + k + 2)$$

One may note that another possible decomposition is $\mu^R(\vec{x}_R) = (k)$, $\lambda^R(\vec{x}_R) = (2j + 2)$. In general, when the schedule contains skewing it is possible to embed either of the skewing dimensions in the μ part of the schedule. For the sake of coherency we add an extra convention for the decomposition: μ matches the first non-null iterator coefficient of the schedule. Returning to the example, $\mu^R(\vec{x}_R) = (j)$, $\lambda^R(\vec{x}_R) = (j + k + 2)$ is thus the only valid decomposition of Θ_k^R.

Note that this decomposition prevents modeling of compositions of loop permutations in the λ part. For λ to represent a loop permutation, λ must have values in $\mathbb{Z}$, as shown in the following example:

$$\mu^R(\vec{x}_R) = \begin{pmatrix}1 & 0 & 0 & 0 & 0\end{pmatrix} \cdot \begin{pmatrix}i & j & k & N & 1\end{pmatrix}^t = \begin{pmatrix}i\end{pmatrix}$$
$$(\mu^R + \lambda)(\vec{x}_R) = \begin{pmatrix}j\end{pmatrix} \Rightarrow \lambda = \begin{pmatrix}-1 & 1 & 0 & 0 & 0\end{pmatrix}$$

which is not possible as we have constrained $\lambda_i \in \mathbb{N}$. Hence, when considering arbitrary compositions of permutation, (parametric) shifting, skewing and peeling, the $\mu + \lambda$ decomposition separates permutation (embedded in the μ part of the schedule) from the other transformations (embedded in the λ part of the schedule). We now show it is possible to determine if a set of statements are fusable only by looking at the possible values for the μ part of their schedules.

Considering three statements R, S, T that are fusable while preserving the semantics at level k. Then there exist $\Theta_k^R = \mu^R + \lambda^R,\ \Theta_k^S = \mu^S + \lambda^S,\ \Theta_k^T = \mu^T + \lambda^T$ leading to fusing those statements. Considering now the sub-problem of fusing only R and S, we build the set $M_{R,S}$ of all possible values of μ^R, μ^S for which there exist a λ^R, λ^S leading to fuse R and S. Obviously, the value of μ^R, μ^S leading to fusing R, S, T are in $M_{R,S}$, and μ^S, μ^T are also in $M_{S,T}$. Similarly μ^R, μ^T are in $M_{R,T}$. We derive a sufficient condition for fusibility based on pairwise loop permutations for fusion.

LEMMA 6 (Pairwise sufficient condition for fusibility). *Given three statements R, S, T such that they can be 2-by-2 fused and distributed. Given $M_{R,S}$ (resp. $M_{R,T}$, resp. $M_{S,T}$) the set of possible tuples μ^R, μ^S (resp. μ^R, μ^T, resp. μ^S, μ^T) leading to fusing R and S (resp. R and T, resp. S and T) such that the full program semantics is respected. R, S, T are fusable if there exists μ^R, μ^S, μ^T such that:*

$$\mu^R, \mu^S \in M_{R,S}$$
$$\mu^R, \mu^T \in M_{R,T}$$
$$\mu^S, \mu^T \in M_{S,T}$$

Proof. Given the schedule $\Theta_k^R = \mu^R + \lambda^R, \Theta_k^S = \mu^S + \lambda^S$ leading to fusing R and S, $\Theta_k^{'R} = \mu^R + \lambda^{'R}, \Theta_k^T = \mu^T + \lambda^T$ leading to fusing R and T, and $\Theta_k^{'S} = \mu^S + \lambda^{'S}, \Theta_k^{'T} = \mu^T + \lambda^{'T}$ leading to fusing S and T, such that they all preserve the full program semantics.

The schedule $\Theta_k^{*R} = \mu^R + \lambda^R + \lambda^{'R}, \Theta_k^{*S} = \mu^S + \lambda^S + \lambda^{'R}$ is legal, as adding $\lambda^{'R}$ consists in performing additional compositions of skewing and shifting, which cannot make the dependence vectors lexicographically negative. It cannot consist in performing a parametric shift (resulting in a loop distribution), Θ_k^R is a schedule fusing R and S and $\Theta_k^{'R}$ is a schedule fusing R and T. As Θ_k^{*R} is a non-constant schedule, it leads to fusing R and S. Generalizing this reasoning we can exhibit the following semantics-preserving schedule leading to the fusion of R, S, T:

$$\Theta_k^{*R} = \mu^R + \lambda^R + \lambda^{'R} + \lambda^S + \lambda^{'S} + \lambda^T + \lambda^{'T}$$
$$\Theta_k^{*S} = \mu^S + \lambda^R + \lambda^{'R} + \lambda^S + \lambda^{'S} + \lambda^T + \lambda^{'T}$$
$$\Theta_k^{*T} = \mu^T + \lambda^R + \lambda^{'R} + \lambda^S + \lambda^{'S} + \lambda^T + \lambda^{'T}$$

As all statements are fused 2-by-2, they are fused all together. As the three statements can be distributed 2-by-2, there is no dependence cycle. ∎

To stress the importance of Lemma 6, let us return to the ThreeMatMat example. We can compute the pairwise permutations for fusion sets at the outermost level:

$$M_{R,S} = \{(i,i);(i,j);(i,k);(j,i);(j,j);(j,k);(k,i);(k,j);(k,k)\}$$
$$M_{R,T} = \{(i,i);(j,k)\}$$
$$M_{S,T} = \{(i,k);(j,j)\}$$

These sets are computed by iteratively testing, against the set of constraints for semantics-preservation augmented with fusion and orthogonality constraints, for the existence of solutions with a non-null value for each of the coefficients associated with the statement iterators *for only the two considered statements*. This is in contrast to the general algorithm which requires to consider the whole set of candidate statements for fusion. Here we can decide that R, S, T are fusable, as the solution $\mu^R = j,\ \mu^S = i,\ \mu^T = k$ respects the conditions from Lemma 6. This solution is presented in Figure 2(a).

To improve further the tractability, we rely on two more standard properties on fusion. Given two statements R and S:

1. if R and S are not fusable, then any statement on which R transitively depends on is not fusable with S and any statement transitively depending on S;
2. reciprocally, if R and S must be fused, then any statement depending on R and on which S depends must also be fused with R and S.

These properties cut the number of tested sequences dramatically, in particular, in highly constrained programs such as loop-intensive kernels. They are used at each step of the optimization algorithm.

4.4 Optimization algorithm

We now present our optimization algorithm. The algorithm explores possible interleavings of dimension *maxExploreDepth*, and generates a collection of program schedules, each of them being a candidate for the optimization. We use iterative compilation to select the best performing one. For each candidate program schedule we generate back a syntactic C code, compile it and run it on the target machine.

The structure and principle of the optimization algorithm, shown in Figure 4, matches that of the pruning algorithm of Figure 3, as it also aims at computing a set of feasible interleavings at a given level. It is in essence a *specialization* of the pruning algorithm for our optimization problem instance. To decide the fusibility of a set of statements, we put the problem in a form matching the applicability conditions of Lemma 6. We merge nodes that must be 2-by-2 fused to guarantee that we are checking for the strictest set of program-wise valid μ values when considering fusibility.

```
OptimizeRec: Compute all optimizations
Input:
  Θ: partial program optimization
  pdg: polyhedral dependence graph
  d: current level for the interleaving exploration
  n: number of statements
  maxExploreDepth: maximum level to explore for interleaving
Output:
  Θ: complete program optimization

1   G ← newGraph(n)
2   F^d ← O
3   unfusable ← ∅
4   forall pairs of dependent statements R,S in pdg do
5     T_{R,S} ← buildLegalOptimizedSchedules({R,S}, Θ, d, pdg)
6     if mustDistribute(T_{R,S}, d) then
7       F^d ← F^d ∩ {e_{R,S} = 0}
8     else
9       if mustFuse(T_{R,S}, d) then
10        F^d ← F^d ∩ {e_{R,S} = 1}
11      end if
12      F^d ← F^d ∩ {s_{R,S} = 0}
13      M_{R,S} ← computeLegalPermutationsAtLevel(T_{R,S}, d)
14      addEdgeWithLabel(G, R, S, M_{R,S})
15    end if
16  end for
17  forall pairs of statements R,S such that e_{R,S} = 1 do
18    mergeNodes(G, R, S)
19  end for
20  for l ← 2 to n−1 do
21    forall paths p in G of length l such that
        there is no prefix of p in unfusable do
22      if ¬ existCompatiblePermutation(nodes(p)) then
23        F^d ← F^d ∩ {Σ_p e_{pairs in p} < l−1}
24        unfusable ← unfusable ∪ p
25      end if
26    end do
27  end for
28  forall i ∈ F^d do
29    Θ_{2d} ← embedInterleaving(Θ, d, i)
30    Θ_{2d+1} ← computeOptimizedSchedule(Θ, pdg, d, i)
31    if d < maxExploreDepth then
32      OptimizeRec(Θ, pdg, d+1, n, maxExploreDepth)
33    else
34      finalizeOptimizedSchedule(Θ, pdg, p)
35      output Θ
36    end if
37  end for
```

Figure 4. Optimization Algorithm

Procedure `buildLegalOptimizedSchedules` computes, for a given pair of statements R,S, the set $T_{R,S}$ of semantics-preserving non-negative schedules augmented with permutability and orthogonality constraints as defined in the previous Section, given the previously computed rows of Θ. Procedures `mustDistribute`, `mustFuse`, `mergeNodes` operate in a similar fashion as in the general case. Procedure `computeLegalPermutationsAtLevel` computes $M_{R,S}$ the set of all valid permutations μ^R, μ^S leading to fusion. To check if a given permutation μ^R, μ^S is valid and leading for fusion at level d, the set of constraints is tested for the existence of a solution where the schedule coefficients of row d corresponding to μ^R and μ^S is not 0. This is sufficient to determine the existence of the associated λ part. Procedure `existCompatiblePermutation` collects the sets M of the pairs of statements connected by the path p, and tests for the existence of μ values according to Lemma 6. If there is no compatible permutation, then an additional constraint is added to F^d such that it is not possible to fuse the statements in p all together. The constraint sets that the $e_{i,j}$ variables, for all pairs i,j in the path p, cannot be set to 1 all together. Procedure `embedInterleaving` instantiate a schedule row Θ_{2d} that implements the interleaving i, using only the scalar coefficients of the schedule. Procedure `computeOptimizedSchedule` instantiates a schedule row Θ_{2d+1} at the current interleaving dimension d, for all statements. The interleaving is given by i, and individually for each group of statements to be fused under a common loop, a schedule is computed to maximize fusion and to enforce permutability if possible. To select the coefficient values we resort to the objective function (11). Finally, procedure `finalizeOptimizedSchedule` computes the possibly remaining schedule dimensions, when *maxExploreDepth* is lower than the maximum program loop depth. Note that in this case, maximal fusion is used to select the interleaving, hence we do not need to build a set of interleavings for the remaining depths.

In practice this algorithm proved to be very fast, and for instance computing the set F^1 of all semantics-preserving interleavings at the first dimension takes less than 0.5 second for the benchmark ludcmp, pruning F^1 from about 10^{12} structures to 8, on an initial space with 182 binary variables to model all total preorders. Subsequently traversing F^1 and computing a valid transformation for all interleavings takes an additional 2.1 second.

5. Experimental Results

Studies performed on the performance impact of selecting schedules at various levels highlighted the higher impact of carefully selecting outer loops [32, 33]. The selection of the statement interleaving at the outermost level captures the most significant difference in terms of locality and communication. We can limit the recursive traversal of interleavings to the outer level only, while obtaining significant performance improvement and a wide range of transformed codes. Nevertheless, when the number of candidates in F^1 is very small, typically because of several loop-dependent dependences at the outer level, it is relevant to build F^2 and further. Note that in the experiments presented in this paper we traverse exhaustively only F^1.

Search space statistics Several $e_{i,j}$ and $p_{i,j}$ variables are set during the pruning of O, so several consistency constraints are made useless and are not built, significantly helping to reduce the size of the space to build. Table 1 illustrates this by highlighting, for our benchmarks considered, the properties of the polytope O in terms of the number of dimensions (#dim), constraints (#cst) and points (#points) when compared to F^1, the polytope of possible interleavings *for the first dimension only*. For each benchmark, we list #loops the number of loops, #stmts the number of statements, #refs the number of array references, as well as the time to build all candidate interleavings from the input source code (that is, in-

				O			F^1						
Benchmark	#loops	#stmts	#refs	#dim	#cst	#points	#dim	#cst	#points	Time	Pb. Size	perf-Intel	perf-AMD
advect3d	12	4	32	12	58	75	9	43	26	0.82s	300x300x300	1.47×	5.19×
atax	4	4	10	12	58	75	6	25	16	0.06s	8000x8000	3.66×	1.88×
bicg	3	4	10	12	58	75	10	52	26	0.05s	8000x8000	1.75×	1.40×
gemver	7	4	19	12	58	75	6	28	8	0.06s	8000x8000	1.34×	1.33×
ludcmp	9	14	35	182	3003	$\approx 10^{12}$	40	443	8	0.54s	1000x1000	1.98×	1.45×
doitgen	5	3	7	6	22	13	3	10	4	0.08s	50x50x50	15.35×	14.27×
varcovar	7	7	26	42	350	47293	22	193	96	0.09s	1000x1000	7.24×	14.83×
correl	5	6	12	30	215	4683	21	162	176	0.09s	1000x1000	3.00×	3.44×

Table 1. Search space statistics and performance improvement

cluding all analysis) on an Intel Xeon 2.4GHz. We also report the dataset size we used for the benchmarks (Pb. Size).

Performance evaluation The automatic optimization and parallelization approach has been implemented in PoCC, the *Polyhedral Compiler Collection*, a complete source-to-source polyhedral compiler based on available free software for polyhedral compilation.[3] The time to compute the space, select a candidate and compute a full transformation is negligible with respect to the compilation and execution time of the tested versions. In our experiments, the full process takes a few seconds for the smaller benchmarks, and up to about 2 minutes for correl on an Intel Xeon processor.

Extensive performance characterization is beyond the scope of this paper. The reader is referred to other publications [32, 34] for a more comprehensive analysis of the performance issues that are successfully addressed by our iterative framework on various high-end multi-core architectures. Nevertheless, to illustrate the benefit of our approach, we report in Table 1 the performance improvement obtained by our iterative process. We used Intel ICC 11.0 with options `-fast -parallel -openmp` as the baseline on the original code, and also to compile all our optimized programs. We report the performance improvement achieved over ICC with automatic parallelization enabled on the original code, under the column perf-Intel for a 4-socket Intel hex-core Xeon E7450 (Dunnington), running at 2.4GHz with 64GB of memory (24 cores, 24 hardware threads), and perf-AMD for a 4-socket AMD quad-core Opteron 8380 (Shanghai) running at 2.50GHz (16 cores, 16 hardware threads) with 64GB of memory.

Beyond absolute performance improvement, another motivating factor for iterative selection of fusion structures is performance portability. Because of significant differences in design, in particular in SIMD units' performance and cache behavior, a transformation has to be tuned for a specific machine. This leads to a significant variation in performance across tested frameworks. To illustrate this, we show in Figure 5 the relative performance normalized with respect to icc-par for gemver, for the Xeon and Opteron. The version index is plotted on the x axis; 1 represents maximal fusion while 8 corresponds to maximal distribution. For the Xeon, the best found version performs 10% better than the best fusion configuration for the Opteron. Optimizing for the Opteron, the best version performs 20% better than the best fusion structure for the Xeon.

Tuning the trade-off between fusion and distribution is a relevant approach to drive a complete high-level optimization framework. The main motivation is the difficulty to design a portable heuristic to select the best interleaving, as the best fusion structure is machine-dependent and even depends on the back-end compiler used. Also, as shown in Figure 2, the interleaving selection can dramatically impact the transformations required to implement the interleaving (e.g., loop interchange) and subsequently drives the optimization process.

[3] PoCC is available at `http://pocc.sourceforge.net`

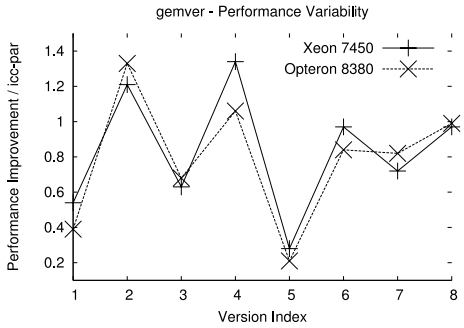

Figure 5. Performance variability for gemver

Our technique is able to automatically adapt to the target framework, and thanks to empirical search successfully discovers the best fusion structure, whatever the specifics of the program, compiler and architecture.

6. Related Work

Traditional approaches to loop fusion [25, 29, 40] are limited in their ability to handle compositions of loop transformations. This is mainly due to the lack of a powerful representation for dependences and transformations. Hence, non-polyhedral approaches typically study fusion in isolation from other transformations. Megiddo and Sarkar [30] presented a solution to the problem of loop fusion in parallel programs that performs fusion while preserving the parallelism in the input program. We believe that several interesting solutions are not obtained when fusion is decoupled from parallelization in those frameworks where legal fusion choices are left out of the framework. Darte et al. [13, 14] studied the combination of loop shifting and fusion for parallelization. In contrast to all of these works, the search space explored in this paper includes fusion in the presence of all polyhedral transformations.

Heuristics for loop fusion and tiling have been proposed in loop-nest optimizers [26, 35, 45, 48]. Those heuristics have also been revisited in the context of new architectures with non-uniform memory hierarchies and heterogeneous computing resources [37]. The polyhedral model is complementary to these efforts, opening many more opportunities for optimization in loop nest optimizers and parallelizing compilers. Note that the polyhedral model is currently being integrated into production compilers, including GCC[4], IBM XL and LLVM/Polly.

A heuristic that integrates loop fusion and tiling in the polyhedral model that subsumes a number of transformations such as interchange, skewing and loop shifting was presented in Bondhugula et al. [6, 8]. Their work builds complex sequences of transformations that enable communication-minimal tiling of im-

[4] Graphite framework: `http://gcc.gnu.org/wiki/Graphite`.

perfectly nested loops that generalizes the prior work on tiling perfectly-nested loops [21, 23, 36, 49] and is helpful in identifying parallelism-locality trade-offs. Bondhugula et al. [7] refined their static cost model for fusion by incorporating utilization of hardware prefetch stream buffers, loss of parallelism, and constraints imposed by privatization and code expansion. However, these techniques do not model a closed space of all and only distinct fusion structures, instead they operate on the entire space of valid fusion structures and select a specific one that minimizes a static cost function.

Across a broad range of machine architectures and compiler transformations, iterative compilation has proven to be effective in deriving significant performance benefits [1, 5, 19, 31, 33, 35, 37, 42, 46]. Nevertheless, none of iterative compilation approaches achieved the expressiveness and the ability to apply complex sequences of transformations presented in this paper, while focusing the search only on semantics-preserving transformation candidates.

Several powerful semi-automatic frameworks based on the polyhedral model [9, 11, 20, 24, 43] have been proposed; these frameworks are able to capture fusion structures, but do not construct profitable parallelization and tiling strategies using a model-based heuristic.

R-Stream is a source-to-source, auto-parallelizing compiler developed by Reservoir Labs, Inc. R-Stream is based on the polyhedral model and targets modern heterogeneous multi-core architectures, including multiprocessors with caches, systems with accelerators, and distributed memory architectures that require explicit memory management and data movement [28]. The affine scheduler in R-Stream builds on contributions from Feautrier [18], Megiddo and Sarkar [30] and Bondhugula et al. [8]. It introduced the concept of affine fusion to enable a single formulation for the joint optimization of cost functions representing tradeoffs between amount of parallelism and amount of locality at various depths in the loop nest hierarchy. Scheduling algorithms in R-Stream search an optimization space that is either constructed on a depth-by-depth basis as solutions are found or based on the convex space of all legal multi-dimensional schedules as had been illustrated by Vasilache [44]. R-Stream allows direct optimization of the tradeoff function using Integer Linear Programming (ILP) solvers as well as iterative exploration of the search space. Redundant solutions in the search space are implicitly pruned out by the combined tradeoff function as they exhibit the same overall cost. In practice, a solution to the optimization problem represents a whole class of equivalent scheduling functions with the same cost.

7. Conclusions

The selection of a profitable composition of loop transformations is a hard combinatorial problem. We proposed a complete, non-redundant characterization of multidimensional affine transformations as a convex space. We then pruned this polyhedron, focusing on multidimensional statement interleavings corresponding to a generalized combination of loop fusion, loop distribution and code motion. We proposed a practical optimization algorithm to explore the pruned polyhedron, while heuristically building a profitable, semantics-preserving, affine enabling transformation. This algorithm was applied to relevant benchmarks, demonstrating good scalability and strong performance improvements over state-of-the-art multi-core architectures and parallelizing compilers.

Acknowledgments We are grateful to Paul Feautrier for providing the foundations for multidimensional affine scheduling, and the simplification of Vasilache's original convex formulation of all semantics-preserving schedules that we presented in this paper.

This work was supported in part by the U.S. Defense Advanced Research Projects Agency through AFRL Contract FA8650-09-C-7915, the U.S. National Science Foundation through awards 0541409, 0811457, 0811781, 0926687 and 0926688, by the U.S. Army through contract W911NF-10-1-0004, and by the European Commission through the FP7 project TERAFLUX id. 249013.

References

[1] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. P. O'Boyle, J. Thomson, M. Toussaint, and C. K. I. Williams. Using machine learning to focus iterative optimization. In *Proc. of the Intl. Symposium on Code Generation and Optimization (CGO'06)*, pages 295–305, Washington, 2006.

[2] D. Barthou, J.-F. Collard, and P. Feautrier. Fuzzy array dataflow analysis. *Journal of Parallel and Distributed Computing*, 40:210–226, 1997.

[3] C. Bastoul. Code generation in the polyhedral model is easier than you think. In *IEEE Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT'04)*, pages 7–16, Juan-les-Pins, France, Sept. 2004.

[4] M.-W. Benabderrahmane, L.-N. Pouchet, A. Cohen, and C. Bastoul. The polyhedral model is more widely applicable than you think. In *Intl. Conf. on Compiler Construction (ETAPS CC'10)*, LNCS 6011, pages 283–303, Paphos, Cyprus, Mar. 2010.

[5] F. Bodin, T. Kisuki, P. M. W. Knijnenburg, M. F. P. O'Boyle, and E. Rohou. Iterative compilation in a non-linear optimisation space. In *W. on Profile and Feedback Directed Compilation*, Paris, Oct. 1998.

[6] 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)*, Apr. 2008.

[7] U. Bondhugula, O. Gunluk, S. Dash, and L. Renganarayanan. A model for fusion and code motion in an automatic parallelizing compiler. In *Proc. of the 19th Intl. conf. on Parallel Architectures and Compilation Techniques (PACT'10)*, pages 343–352, 2010.

[8] U. Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan. A practical automatic polyhedral program optimization system. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2008.

[9] C. Chen, J. Chame, and M. Hall. CHiLL: A framework for composing high-level loop transformations. Technical Report 08-897, U. of Southern California, 2008.

[10] P. Clauss. Counting solutions to linear and nonlinear constraints through ehrhart polynomials: applications to analyze and transform scientific programs. In *Proc. of the Intl. Conf. on Supercomputing*, pages 278–285. ACM, 1996.

[11] A. Cohen, S. Girbal, D. Parello, M. Sigler, O. Temam, and N. Vasilache. Facilitating the search for compositions of program transformations. In *ACM International conference on Supercomputing*, pages 151–160, June 2005.

[12] A. Darte. On the complexity of loop fusion. *Parallel Computing*, pages 149–157, 1999.

[13] A. Darte and G. Huard. Loop shifting for loop parallelization. Technical Report RR2000-22, ENS Lyon, May 2000.

[14] A. Darte, G.-A. Silber, and F. Vivien. Combining retiming and scheduling techniques for loop parallelization and loop tiling. *Parallel Proc. Letters*, 7(4):379–392, 1997.

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

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

[17] P. Feautrier. Some efficient solutions to the affine scheduling problem, part I: one dimensional time. *Intl. J. of Parallel Programming*, 21(5):313–348, Oct. 1992.

[18] P. Feautrier. Some efficient solutions to the affine scheduling problem, part II: multidimensional time. *Intl. J. of Parallel Programming*, 21(6):389–420, Dec. 1992.

[19] F. Franchetti, Y. Voronenko, and M. Püschel. Formal loop merging for signal transforms. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pages 315–326, 2005.

[20] S. Girbal, N. Vasilache, C. Bastoul, A. Cohen, D. Parello, M. Sigler, and O. Temam. Semi-automatic composition of loop transformations. *Intl. J. of Parallel Programming*, 34(3):261–317, June 2006.

[21] M. Griebl. Automatic parallelization of loop programs for distributed memory architectures. Habilitation thesis. Facultät für Mathematik und Informatik, Universität Passau, 2004.

[22] A.-C. Guillou, F. Quilleré, P. Quinton, S. Rajopadhye, and T. Risset. Hardware design methodology with the Alpha language. In *FDL'01*, Lyon, France, Sept. 2001.

[23] F. Irigoin and R. Triolet. Supernode partitioning. In *ACM SIGPLAN Principles of Programming Languages*, pages 319–329, 1988.

[24] W. Kelly. *Optimization within a Unified Transformation Framework*. PhD thesis, Univ. of Maryland, 1996.

[25] K. Kennedy and K. McKinley. Maximizing loop parallelism and improving data locality via loop fusion and distribution. In *Languages and Compilers for Parallel Computing*, pages 301–320, 1993.

[26] I. Kodukula, N. Ahmed, and K. Pingali. Data-centric multi-level blocking. In *ACM SIGPLAN'97 Conf. on Programming Language Design and Implementation*, pages 346–357, Las Vegas, June 1997.

[27] M. Kudlur and S. Mahlke. Orchestrating the execution of stream programs on multicore platforms. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI'08)*, pages 114–124. ACM Press, 2008.

[28] R. Lethin, A. Leung, B. Meister, N. Vasilache, D. Wohlford, M. Baskaran, A. Hartono, and K. Datta. R-stream compiler. In D. Padua, editor, *Encyclopedia of Parallel Computing*. Springer, 1st edition., 2011, 50 p. in 4 volumes, not available separately., hardcover edition, June 2011.

[29] K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Trans. Program. Lang. Syst.*, 18(4):424–453, 1996.

[30] N. Megiddo and V. Sarkar. Optimal weighted loop fusion for parallel programs. In *symposium on Parallel Algorithms and Architectures*, pages 282–291, 1997.

[31] A. Nisbet. GAPS: A compiler framework for genetic algorithm (GA) optimised parallelisation. In *HPCN Europe 1998: Proc. of the Intl. Conf. and Exhibition on High-Performance Computing and Networking*, pages 987–989, London, UK, 1998. Springer-Verlag.

[32] L.-N. Pouchet. *Interative Optimization in the Polyhedral Model*. PhD thesis, University of Paris-Sud 11, Orsay, France, Jan. 2010.

[33] L.-N. Pouchet, C. Bastoul, A. Cohen, and J. Cavazos. Iterative optimization in the polyhedral model: Part II, multidimensional time. In *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI'08)*, pages 90–100. ACM Press, 2008.

[34] L.-N. Pouchet, U. Bondhugula, C. Bastoul, A. Cohen, J. Ramanujam, and P. Sadayappan. Combined iterative and model-driven optimization in an automatic parallelization framework. In *Conf. on SuperComputing (SC'10)*, New Orleans, LA, Nov. 2010. To appear.

[35] A. Qasem and K. Kennedy. Profitable loop fusion and tiling using model-driven empirical search. In *Proc. of the 20th Intl. Conf. on Supercomputing (ICS'06)*, pages 249–258. ACM press, 2006.

[36] J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for multicomputers. *Journal of Parallel and Distributed Computing*, 16(2):108–230, 1992.

[37] M. Ren, J. Y. Park, M. Houston, A. Aiken, and W. J. Dally. A tuning framework for software-managed memory hierarchies. In *Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT'08)*, pages 280–291. ACM Press, 2008.

[38] L. Renganarayanan, D. Kim, S. Rajopadhye, and M. M. Strout. Parameterized tiled loops for free. *SIGPLAN Notices, Proc. of the 2007 PLDI Conf.*, 42(6):405–414, 2007.

[39] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1986.

[40] S. Singhai and K. McKinley. A Parameterized Loop Fusion Algorithm for Improving Parallelism and Cache Locality. *The Computer Journal*, 40(6):340–355, 1997.

[41] N. J. A. Sloane. Sequence a000670. The On-Line Encyclopedia of Integer Sequences.

[42] M. Stephenson, S. Amarasinghe, M. Martin, and U.-M. O'Reilly. Meta optimization: improving compiler heuristics with machine learning. *SIGPLAN Not.*, 38(5):77–90, 2003.

[43] A. Tiwari, C. Chen, J. Chame, M. Hall, and J. K. Hollingsworth. A scalable autotuning framework for computer optimization. In *IPDPS'09*, Rome, May 2009.

[44] N. Vasilache. *Scalable Program Optimization Techniques in the Polyhedra Model*. PhD thesis, University of Paris-Sud 11, 2007.

[45] S. Verdoolaege, F. Catthoor, M. Bruynooghe, and G. Janssens. Feasibility of incremental translation. Technical Report CW 348, Katholieke Universiteit Leuven Department of Computer Science, Oct. 2002.

[46] Y. Voronenko, F. de Mesmay, and M. Püschel. Computer generation of general size linear transform libraries. In *Intl. Symp. on Code Generation and Optimization (CGO'09)*, Mar. 2009.

[47] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimization of software and the atlas project. *Parallel Computing*, 27(1–2):3–35, 2001.

[48] M. Wolf, D. Maydan, and D.-K. Chen. Combining loop transformations considering caches and scheduling. In *MICRO 29: Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture*, pages 274–286, 1996.

[49] M. Wolfe. More iteration space tiling. In *Proceedings of Supercomputing '89*, pages 655–664, 1989.

[50] M. Wolfe. *High performance compilers for parallel computing*. Addison-Wesley Publishing Company, 1995.