Polyhedral Scheduling and Transformations

Louis-Noël Pouchet

CS & ECE
Colorado State University

February 23, 2020

PPoPP'20 Tutorial

Scheduling

Affine Scheduling

Definition (Affine schedule)

Given a statement S, a p-dimensional affine schedule Θ^R is an affine form on the outer loop iterators $\vec{x}_S$ and the global parameters $\vec{n}$. It is written:

$$\Theta^S(\vec{x}_S) = \mathbf{T}_S \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix}, \quad \mathbf{T}_S \in \mathbb{K}^{p \times dim(\vec{x}_S) + dim(\vec{n}) + 1}$$

- **A schedule assigns a timestamp to each executed instance of a statement**
- If T is a vector, then Θ is a one-dimensional schedule
- If T is a matrix, then Θ is a multidimensional schedule
  - Question: does it translate to sequential loops?

Legal Program Transformation

Definition (Precedence condition)

Given Θ^R a schedule for the instances of R, Θ^S a schedule for the instances of S. Θ^R and Θ^S are legal schedules if $\forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}$:

$$\Theta_R(\vec{x}_R) \prec \Theta_S(\vec{x}_S)$$

$\prec$ denotes the *lexicographic ordering*.

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

Scheduling in the Polyhedral Model

Constraints:

- The schedule must respect the precedence condition, for all dependent instances
- Dependence constraints can be turned into constraints on the solution set

Scheduling:

- Among all possibilities, one has to be picked
- Optimal solution requires to consider all legal possible schedules
  - Question: is this always true?

One-Dimensional Affine Schedules

For now, we focus on 1-d schedules

Example

```
for (i = 1; i < N; ++i)
  A[i] = A[i - 1] + A[i] + A[i + 1];
```

- Simple program: 1 loop, 1 polyhedral statement
- 2 dependences:
  - RAW: A[i] $\rightarrow$ A[i - 1]
  - WAR: A[i + 1] $\rightarrow$ A[i]

Checking the Legality of a Schedule

Exercise: given the dependence polyhedra, check if a schedule is legal

$$\mathcal{D}_1 : \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} eq \\ i_S \\ i'_S \\ n \\ 1 \end{pmatrix} \qquad \mathcal{D}_2 : \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} eq \\ i_S \\ i'_S \\ n \\ 1 \end{pmatrix}$$

1. $\Theta = i$
2. $\Theta = -i$

Checking the Legality of a Schedule

Exercise: given the dependence polyhedra, check if a schedule is legal

$$\mathcal{D}_1 : \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} eq \\ i_S \\ i'_S \\ n \\ 1 \end{pmatrix} \qquad \mathcal{D}_2 : \begin{bmatrix} 1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 1 & -1 \\ 0 & 1 & -1 & 0 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{pmatrix} eq \\ i_S \\ i'_S \\ n \\ 1 \end{pmatrix}$$

1. $\Theta = i$
2. $\Theta = -i$

Solution: check for the emptiness of the polyhedron

$$\mathcal{P} : \begin{bmatrix} \mathcal{D} \\ i_S \succ i'_S \end{bmatrix} \cdot \begin{pmatrix} i_S \\ i'_S \\ n \\ 1 \end{pmatrix}$$

where:

- $i_S \succ i'_S$ gets the consumer instances scheduled after the producer ones
- For $\Theta = -i$, it is $-i_S \succ -i'_S$, which is non-empty

A (Naive) Scheduling Approach

- Pick a schedule for the program statements
- Check if it respects all dependences

  This is called **filtering**

Limitations:

- How to use this in combination of an objective function?
- The density of legal 1-d affine schedules is low:

	`matmult`	`locality`	`fir`	`h264`	`crout`
$\vec{i}$-Bounds	$-1, 1$	$-1, 1$	$0, 1$	$-1, 1$	$-3, 3$
c-Bounds	$-1, 1$	$-1, 1$	$0, 3$	$0, 4$	$-3, 3$
#Sched.	1.9×10^4	5.9×10^4	1.2×10^7	1.8×10^8	2.6×10^{15}

$$\Downarrow$$

#Legal	6561	912	792	360	798

Objectives for a Good Scheduling Algorithm

- Build a legal schedule!
- Embed some properties in this legal schedule
  - latency: minimize the time of the last iteration
  - delay: minimize the time between the first and last iteration
  - parallelism / placement
  - permutability (for tiling)
  - ...

A possible "simple" two-step approach:

- Find the solution set of all legal affine schedules
- Find an ILP/PIP formulation for the objective function(s)

The Precedence Constraint (Again!)

Precedence constraint adapted to 1-d schedules:

Definition (Causality condition for schedules)

Given $\mathcal{D}_{R,S}$, Θ^R and Θ^S are legal iff for each pair of instances in dependence:

$$\Theta^R(\vec{x_R}) < \Theta^S(\vec{x_S})$$

Equivalently: $\Delta_{R,S} = \Theta^S(\vec{x_S}) - \Theta^R(\vec{x_R}) - 1 \geq 0$

- All functions $\Delta_{R,S}$ which are non-negative over the dependence polyhedron represent legal schedules
- For the instances which are not in dependence, we don't care
- First step: how to get all non-negative functions over a polyhedron?

Affine Form of the Farkas Lemma

Lemma (Affine form of Farkas lemma)

Let $\mathcal{D}$ be a nonempty polyhedron defined by $A\vec{x}+\vec{b} \geq \vec{0}$. Then any affine function $f(\vec{x})$ is non-negative everywhere in $\mathcal{D}$ iff it is a positive affine combination:

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

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

The Farkas Lemma: Example

- Function: $f(x) = ax + b$
- Domain of x: $\{1 \leq x \leq 3\} \rightarrow x - 1 \geq 0, \;\; -x + 3 \geq 0$
- Farkas lemma: $f(x) \geq 0 \Leftrightarrow f(x) = \lambda_0 + \lambda_1(x-1) + \lambda_2(-x+3)$

The system to solve:

$$\left\{\begin{array}{ccccccc} & & \lambda_1 & - & \lambda_2 & = & a \\ \lambda_0 & - & \lambda_1 & + & 3\lambda_2 & = & b \\ \lambda_0 & & & & & \geq & 0 \\ & & \lambda_1 & & & \geq & 0 \\ & & & & \lambda_2 & \geq & 0 \end{array}\right.$$

Example: Semantics Preservation (1-D)

Example: Semantics Preservation (1-D)

Property (Causality condition for schedules)

Given $R\delta S$, Θ^R *and* Θ^S *are legal iff for each pair of instances in dependence:*

$$\Theta^R(\vec{x_R}) < \Theta^S(\vec{x_S})$$

Equivalently: $\Delta_{R,S} = \Theta^S(\vec{x_S}) - \Theta^R(\vec{x_R}) - 1 \geq 0$

Example: Semantics Preservation (1-D)

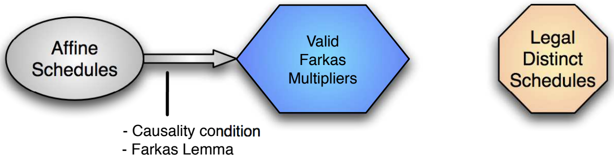
Lemma (Affine form of Farkas lemma)

Let $\mathcal{D}$ be a nonempty polyhedron defined by $A\vec{x}+\vec{b} \geq \vec{0}$. Then any affine function $f(\vec{x})$ is non-negative everywhere in $\mathcal{D}$ iff it is a positive affine combination:

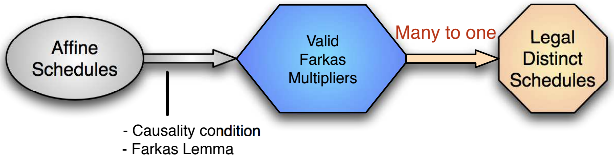
$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T(A\vec{x}+\vec{b}), \text{ with } \lambda_0 \geq 0 \text{ and } \vec{\lambda} \geq \vec{0}.$$

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

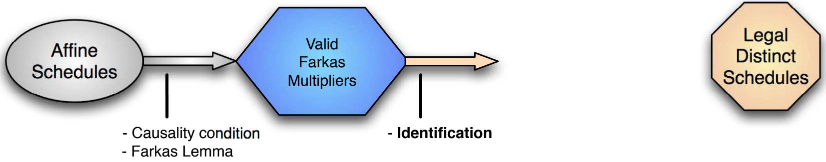
Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



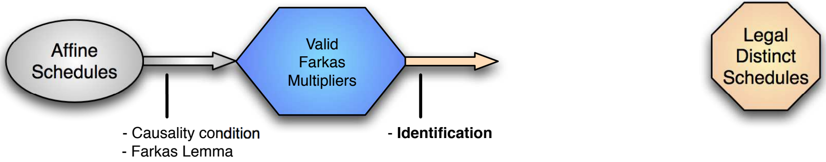
Example: Semantics Preservation (1-D)



$$\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) - 1 = \lambda_0 + \vec{\lambda}^T \left(D_{R,S} \begin{pmatrix} \vec{x}_R \\ \vec{x}_S \end{pmatrix} + \vec{d}_{R,S} \right) \geq 0$$

$$D_{R\delta S} \left\{ \begin{array}{rcl} \mathbf{i_R} & : & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ \mathbf{i_S} & : & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ \mathbf{j_S} & : & \lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\ \mathbf{n} & : & \lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\ \mathbf{1} & : & \lambda_{D_{1,0}} \end{array} \right.$$

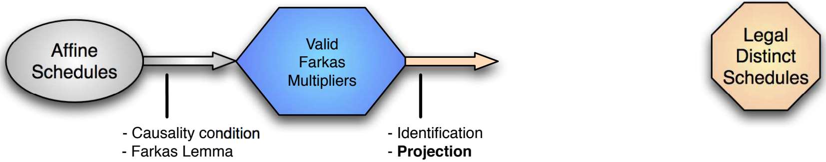
Example: Semantics Preservation (1-D)



$$\Theta^S(\vec{x_S}) - \Theta^R(\vec{x_R}) - 1 = \lambda_0 + \vec{\lambda}^T \left(D_{R,S} \begin{pmatrix} \vec{x_R} \\ \vec{x_S} \end{pmatrix} + \vec{d}_{R,S} \right) \geq 0$$

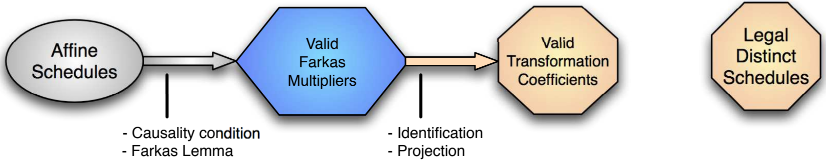
$$\left\{ \begin{array}{lllrcl} D_{R\delta S} & \mathbf{i_R} & : & -t_{1_R} & = & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ & \mathbf{i_S} & : & t_{1_S} & = & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ & \mathbf{j_S} & : & t_{2_S} & = & \lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\ & \mathbf{n} & : & t_{3_S} - t_{2_R} & = & \lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\ & \mathbf{1} & : & t_{4_S} - t_{3_R} - 1 & = & \lambda_{D_{1,0}} \end{array} \right.$$

Example: Semantics Preservation (1-D)

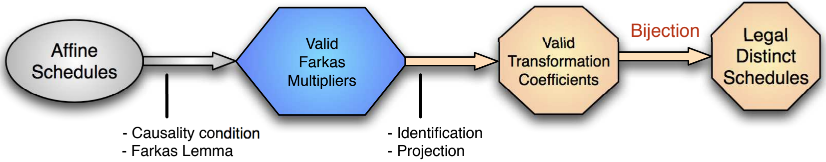


- Solve the constraint system
- Use (purpose-optimized) Fourier-Motzkin projection algorithm
  - Reduce redundancy
  - Detect implicit equalities

Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



- One point in the space $\Leftrightarrow$ one set of legal schedules w.r.t. the dependences

Scheduling Algorithm for Multiple Dependences

Algorithm

- Compute the schedule constraints for each dependence
- Intersect all sets of constraints
- Output is a convex solution set of all legal one-dimensional schedules

- Computation is fast, but requires eliminating variables in a system of inequalities: **projection**
- Can be computed as soon as the dependence polyhedra are known

Objective Function

Idea: bound the latency of the schedule and minimize this bound

Theorem (Schedule latency bound)

If all domains are bounded, and if there exists at least one 1-d schedule Θ, then there exists at least one affine form in the structure parameters:

$$L = \vec{u}.\vec{n} + w$$

such that:

$$\forall \vec{x}_R,\ L \geq \Theta_R(\vec{x}_R)$$

- Objective function: $\min\{\vec{u}, w \mid \vec{u}.\vec{n} + w - \Theta \geq 0\}$
- Subject to Θ is a legal schedule, and $\theta_i \geq 0$
- In many cases, it is equivalent to take the lexicosmallest point in the polytope of non-negative legal schedules

Example

$$\min\{\vec{u}, w \mid \vec{u}.\vec{n} + w - \Theta \geq 0\} : \Theta_R = 0,\ \Theta_S = k + 1$$

Example

```
parfor (i = 0; i < N; ++i)
  parfor (j = 0; j < N; ++j)
    C[i][j] = 0;
for (k = 1; k < N + 1; ++k)
  parfor (i = 0; i < N; ++i)
    parfor (j = 0; j < N; ++j)
      C[i][j] += A[i][k-1] + B[k-1][j];
```

Limitations of One-dimensional Schedules

- Not all programs have a legal one-dimensional schedule
- Question: does this program have a 1-d schedule?

Example

```
for (i = 1; i < N - 1; ++i)
  for (j = 1; j < N - 1; ++j)
    A[i][j] = A[i-1][j-1] + A[i+1][j] + A[i][j+1];
```

- Not all compositions of transformation are possible
  - Interchange in inner-loops
  - Fusion / distribution of inner-loops

Multidimensional Scheduling

Multidimensional Scheduling

- **Some program does not have a legal 1-d schedule**
- It means, it's not possible to enforce the precedence condition for all dependences

Example

```
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    s += s;
```

- Intuition: multidimensional time means nested time loops
- The precedence constraint needs to be adapted to multidimensional time

Dependence Satisfaction

Definition (Strong dependence satisfaction)

Given $\mathcal{D}_{R,S}$, the dependence is strongly satisfied at schedule level k if

$$\forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}, \quad \Theta_k^S(\vec{x}_S) - \Theta_k^R(\vec{x}_R) \geq 1$$

Definition (Weak dependence satisfaction)

Given $\mathcal{D}_{R,S}$, the dependence is weakly satisfied at dimension k if

$$\begin{array}{ll} \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}, & \Theta_k^S(\vec{x}_S) - \Theta_k^R(\vec{x}_R) \geq 0 \\ \exists \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}, & \Theta_k^S(\vec{x}_S) = \Theta_k^R(\vec{x}_R) \end{array}$$

Program Legality and Existence Results

- All dependence must be strongly satisfied for the program to be correct
- **Once a dependence is strongly satisfied at level k, it does not contribute to the constraints of level $k+i$**

- Unlike with 1-d schedules, it is always possible to build a legal multidimensional schedule for a SCoP [Feautrier]

Theorem (Existence of an affine schedule)

Every static control program has a multdidimensional affine schedule

Reformulation of the Precedence Condition

- We introduce variable $\delta_1^{\mathcal{D}_{R,S}}$ to model the dependence satisfaction
- Considering the first row of the scheduling matrices, to preserve the precedence relation we have:

$$\forall \mathcal{D}_{R,S},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}, \quad \Theta_1^S(\vec{x}_S) - \Theta_1^R(\vec{x}_R) \geq \delta_1^{\mathcal{D}_{R,S}}$$
$$\delta_1^{\mathcal{D}_{R,S}} \in \{0, 1\}$$

Lemma (Semantics-preserving affine schedules)

Given a set of affine schedules $\Theta^R, \Theta^S \dots$ of dimension m, the program semantics is preserved if:

$$\begin{array}{ll} & \forall \mathcal{D}_{R,S},\ \exists p \in \{1, \dots, m\},\ \delta_p^{\mathcal{D}_{R,S}} = 1 \\ \wedge & \forall j < p,\ \delta_j^{\mathcal{D}_{R,S}} = 0 \\ \wedge & \forall j \leq p, \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},\ \Theta_p^S(\vec{x}_S) - \Theta_p^R(\vec{x}_R) \geq \delta_j^{\mathcal{D}_{R,S}} \end{array}$$

Space of All Affine Schedules

Objective:

- Design an ILP which operates on all scheduling coefficients
- Easier optimality reasoning: the space contains all schedules (hence necesarily the optimal one)
- Examples: maximal fusion, maximal coarse-grain parallelism, best locality, etc.

idea:

- Combine all coefficients of all rows of the scheduling function into a single solution set
- Find a convex encoding for the lexicopositivity of dependence satisfaction
  - A dependence must be weakly satisfied until it is strongly satisfied
  - Once it is strongly satisfied, it must not constrain subsequent levels

Schedule Lower Bound

Idea:

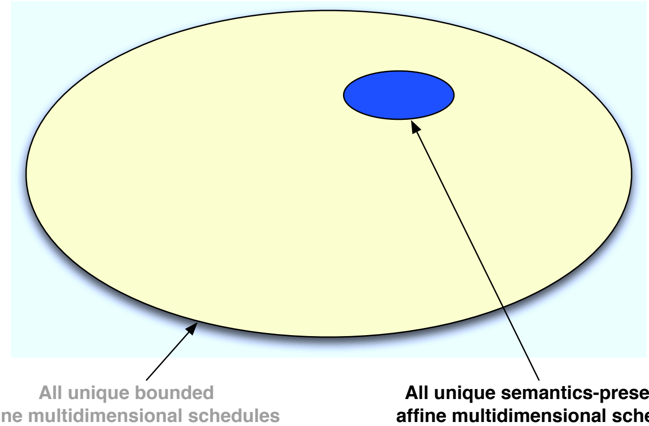
- Bound the schedule latency with a lower bound which does not prevent to find all solutions
- Intuitively:
  - $\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \geq \delta$ if the dependence has not been strongly satisfied
  - $\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \geq -\infty$ if it has

Lemma (Schedule lower bound)

Given Θ^R_k, Θ^S_k such that each coefficient value is bounded in $[x, y]$. Then there exists $K \in \mathbb{Z}$ such that:

$$\max\left(\Theta^S_k(\vec{x}_S) - \Theta^R_k(\vec{x}_R)\right) > -K.\vec{n} - K$$

Space of Semantics-Preserving Affine Schedules



1 point $\leftrightarrow$ 1 unique semantically equivalent program
(up to affine iteration reordering)

Semantics Preservation

Definition (Causality condition)

Given Θ^R a schedule for the instances of R, Θ^S a schedule for the instances of S. Θ^R and Θ^S preserve the dependence $\mathcal{D}_{R,S}$ if $\forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}$:

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

$\prec$ denotes the *lexicographic ordering*.

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

Lexico-positivity of Dependence Satisfaction

- $\Theta^R(\vec{x}_R) \prec \Theta^S(\vec{x}_S)$ is equivalently written $\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \succ \vec{0}$

Lexico-positivity of Dependence Satisfaction

- $\Theta^R(\vec{x}_R) \prec \Theta^S(\vec{x}_S)$ is equivalently written $\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \succ \vec{0}$
- Considering the row p of the scheduling matrices:

$$\Theta^S_p(\vec{x}_S) - \Theta^R_p(\vec{x}_R) \geq \delta_p$$

Lexico-positivity of Dependence Satisfaction

- $\Theta^R(\vec{x}_R) \prec \Theta^S(\vec{x}_S)$ is equivalently written $\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \succ \vec{0}$
- Considering the row p of the scheduling matrices:

$$\Theta^S_p(\vec{x}_S) - \Theta^R_p(\vec{x}_R) \geq \delta_p$$

  - $\delta_p \geq 1$ implies no constraints on δ_k, $k > p$
  - $\delta_p \geq 0$ is required if $\nexists k < p$, $\delta_k \geq 1$

Lexico-positivity of Dependence Satisfaction

- $\Theta^R(\vec{x}_R) \prec \Theta^S(\vec{x}_S)$ is equivalently written $\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \succ \vec{0}$
- Considering the row p of the scheduling matrices:

$$\Theta^S_p(\vec{x}_S) - \Theta^R_p(\vec{x}_R) \geq \delta_p$$

  - $\delta_p \geq 1$ implies no constraints on δ_k, $k > p$
  - $\delta_p \geq 0$ is required if $\nexists k < p$, $\delta_k \geq 1$

- Schedule lower bound:

Lemma (Schedule lower bound)

Given Θ^R_k, Θ^S_k such that each coefficient value is bounded in $[x, y]$. Then there exists $K \in \mathbb{Z}$ such that:

$$\forall \vec{x}_R, \vec{x}_S, \quad \Theta^S_k(\vec{x}_S) - \Theta^R_k(\vec{x}_R) > -K.\vec{n} - K$$

Convex Form of All Bounded Affine Schedules

Lemma (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) $\forall \mathcal{D}_{R,S},\ \delta_p^{\mathcal{D}_{R,S}} \in \{0,1\}$

(ii) $\forall \mathcal{D}_{R,S},\ \sum_{p=1}^{m} \delta_p^{\mathcal{D}_{R,S}} = 1$

(iii) $\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\ldots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$

Convex Form of All Bounded Affine Schedules

Lemma (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) $\forall \mathcal{D}_{R,S},\ \delta_p^{\mathcal{D}_{R,S}} \in \{0,1\}$

(ii) $\forall \mathcal{D}_{R,S},\ \sum_{p=1}^{m} \delta_p^{\mathcal{D}_{R,S}} = 1$

(iii) $\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\ldots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$

Convex Form of All Bounded Affine Schedules

Lemma (Convex form of semantics-preserving affine schedules)

Given a set of affine schedules $\Theta^R, \Theta^S \dots$ of dimension m, the program semantics is preserved if the three following conditions hold:

(i) $$\forall \mathcal{D}_{R,S},\ \delta_p^{\mathcal{D}_{R,S}} \in \{0,1\}$$

(ii) $$\forall \mathcal{D}_{R,S},\ \sum_{p=1}^{m} \delta_p^{\mathcal{D}_{R,S}} = 1$$

(iii) $$\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\dots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$$

$$\Theta_p^S(\vec{x}_S) - \Theta_p^R(\vec{x}_R) \geq \delta_p^{\mathcal{D}_{R,S}}$$

Convex Form of All Bounded Affine Schedules

Lemma (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) $\forall \mathcal{D}_{R,S},\ \delta_p^{\mathcal{D}_{R,S}} \in \{0,1\}$

(ii) $\forall \mathcal{D}_{R,S},\ \sum_{p=1}^{m} \delta_p^{\mathcal{D}_{R,S}} = 1$

(iii) $\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\ldots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$

$$\Theta_p^S(\vec{x}_S) - \Theta_p^R(\vec{x}_R) \geq \delta_p^{\mathcal{D}_{R,S}} - \sum_{k=1}^{p-1} \delta_k^{\mathcal{D}_{R,S}}.(K.\vec{n} + K)$$

Convex Form of All Bounded Affine Schedules

Lemma (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) $$\forall \mathcal{D}_{R,S},\ \delta_p^{\mathcal{D}_{R,S}} \in \{0,1\}$$

(ii) $$\forall \mathcal{D}_{R,S},\ \sum_{p=1}^{m} \delta_p^{\mathcal{D}_{R,S}} = 1$$

(iii) $$\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\ldots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$$
$$\Theta_p^S(\vec{x}_S) - \Theta_p^R(\vec{x}_R) - \delta_p^{\mathcal{D}_{R,S}} + \sum_{k=1}^{p-1} \delta_k^{\mathcal{D}_{R,S}}.(K.\vec{n} + K) \geq 0$$

$\longrightarrow$ Use **Farkas lemma to build all non-negative functions over a polyhedron** (here, the dependence polyhedra) [Feautrier,92]

Convex Form of All Bounded Affine Schedules

Lemma (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) $\forall \mathcal{D}_{R,S},\ \delta_p^{\mathcal{D}_{R,S}} \in \{0,1\}$

(ii) $\forall \mathcal{D}_{R,S},\ \sum_{p=1}^{m} \delta_p^{\mathcal{D}_{R,S}} = 1$

(iii) $\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\ldots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$

$$\Theta_p^S(\vec{x}_S) - \Theta_p^R(\vec{x}_R) - \delta_p^{\mathcal{D}_{R,S}} + \sum_{k=1}^{p-1} \delta_k^{\mathcal{D}_{R,S}}.(K.\vec{n} + K) \geq 0$$

→ Use **Farkas lemma to build all non-negative functions over a polyhedron** (here, the dependence polyhedra) [Feautrier,92]

→ Bounded coefficients required [Vasilache,07]

Maximal Fine-Grain Parallelism

Objectives:

- Have as few dimensions as possible carrying a dependence
- For dimension $k \in p..1$:

$$\min \sum_{\mathcal{D}_{R,S}} \delta_k^{\mathcal{D}_{R,S}}$$

- We use lexicographic optimization

Key Observations

Is all of this really necessary?

- We have encoded one objective per row of Θ
- Question: do we need to solve this large ILP/LP?

Feautrier's Greedy Algorithm

Main idea:

1. Start at row 1 of Θ
2. Build the set of legal one-dimensional schedules
3. Maximize the number of dependences strongly solved $(\max \delta_i)$
4. Remove strongly solved dependences from P
5. Goto 1

This is a <u>row-by-row decomposition of the scheduling problem</u>

Key Properties of Feautrier's Algorithm

- It terminates
- It finds "optimal" fine-grain parallelism

- Granularity of dependence satisfaction: all-or-nothing

Example

```
for (i = 0; i < 2 * N; ++i)      A[i] = A[2 * N - i];
```

Key Observations

Is all of this really necessary?

- Question: do we need to consider all statements at once?
- Insight: the PDG gives structural information about dependences
- Decomposition of the PDG into strongly-connected components

Feautrier's Scheduler

- **Schedule**(U, p):
- U is a set of edges in the GDG and p is an integer. Initially, $p = 1$ and U is the set of all edges in the GDG.

1. Compute the strongly connected components of U, $\{H_1, \ldots, H_n\}$, ranking them according to the reduced graph of U.
2. For each $i = 1, \ldots, n$, solve linear program (29).
   (a) If the solution is such that $\sigma = 0$, the algorithm fails. This never happens if the GDG comes from a sequential program.
   (b) If not, the schedules obtained at step 2 are the components of index p of the multidimensional schedule.
   (c) Build the set U' of unsatisfied edges, and, if $U' \neq \emptyset$, call recursively **Schedule**$(U', p + 1)$.

More Observations

- Some problems may be decomposed without loss of optimality
- The PDG gives extra information about further problem decomposition

Still, is all of this really necessary?

- Question: can we use additional knowledge about dependences?
- Uniform, non-uniform and parametric dependences

Cost Functions

Objectives for Good Scheduling

Fine-grain parallelism is nice, but...

- It has little connection with modern SIMD parallelism
- No information about the quality of the generated code
- Ignores all the important performance objectives:
  - Data locality / TLB / Cache consideration
  - Multi-core parallelism (sync-free, with barrier)
  - SIMD vectorization

Question: how to find a FAST schedule for a modern processor?

Performance Distribution for 1-D Schedules [1/2]

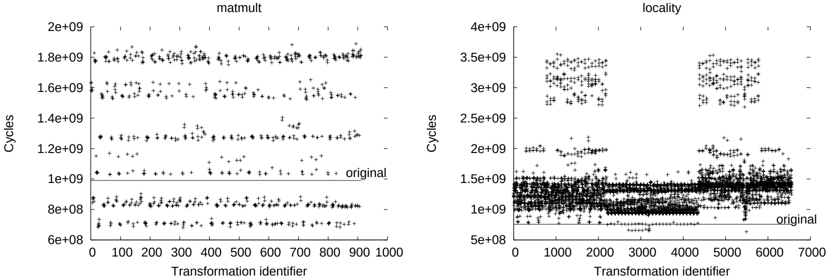


Figure: Performance distribution for `matmult` and `locality`

Performance Distribution for 1-D Schedules [2/2]

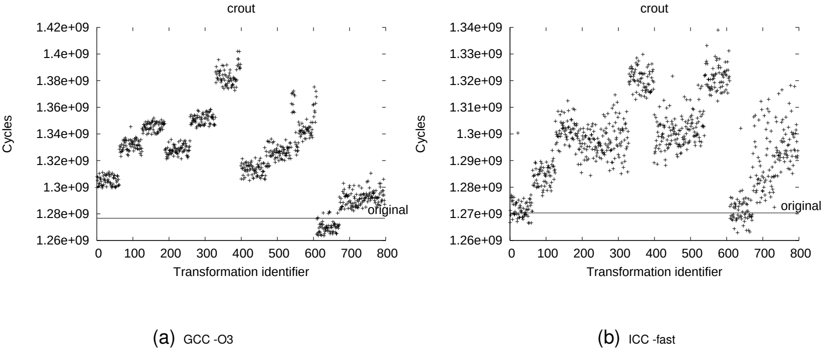


Figure: The effect of the compiler

Quantitative Analysis: The Hypothesis

Extremely large generated spaces: $> 10^{50}$ points

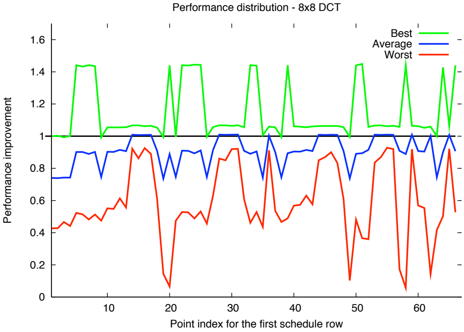
$\rightarrow$ we must leverage static and dynamic characteristics to build traversal mechanisms

Hypothesis:

- **It is possible to statically order the impact on performance of transformation coefficients, that is, decompose the search space in subspaces where the performance variation is maximal or reduced**

- **First rows of Θ are more performance impacting than the last ones**

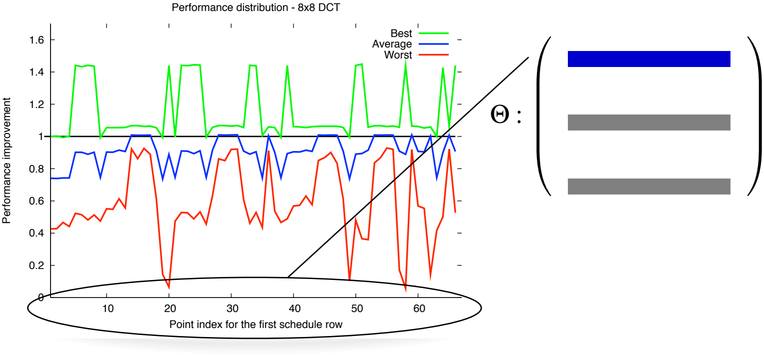
Observations on the Performance Distribution




```
for (i = 0; i < M; i++)
  for (j = 0; j < M; j++) {
    tmp[i][j] = 0.0;
    for (k = 0; k < M; k++)
      tmp[i][j] += block[i][k] *
                   cos1[j][k];
 }
for (i = 0; i < M; i++)
  for (j = 0; j < M; j++) {
    sum2 = 0.0;
    for (k = 0; k < M; k++)
      sum2 += cos1[i][k] * tmp[k][j];
      block[i][j] = ROUND(sum2);
 }
```

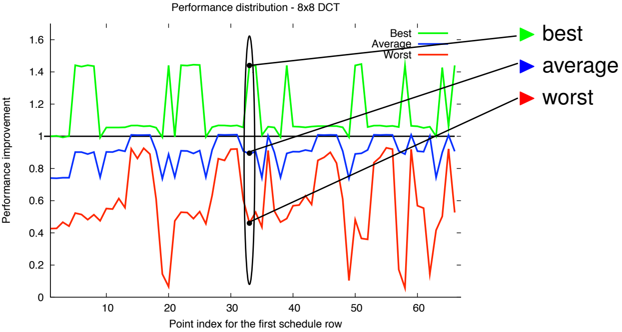
- Extensive study of 8x8 Discrete Cosine Transform (UTDSP)
- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions

Observations on the Performance Distribution



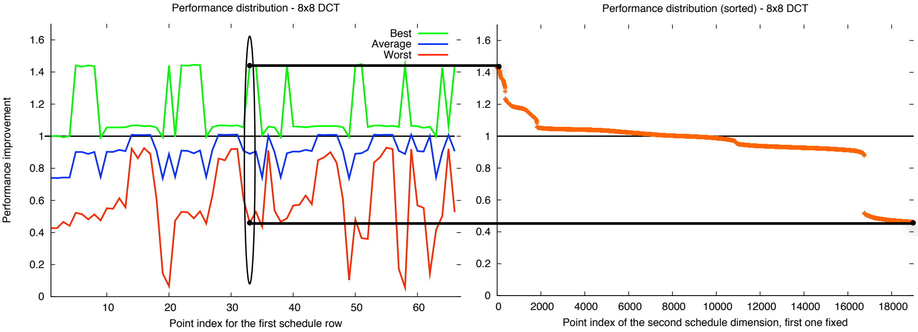
- Extensive study of 8x8 Discrete Cosine Transform (UTDSP)
- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions

Observations on the Performance Distribution



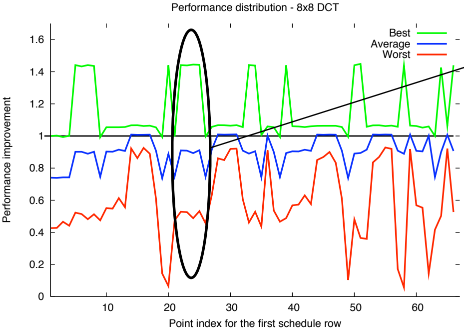
- Take one specific value for the first row
- Try the 19863 possible values for the second row

Observations on the Performance Distribution



- Take one specific value for the first row
- Try the 19863 possible values for the second row
- Very low proportion of best points: $< 0.02\%$

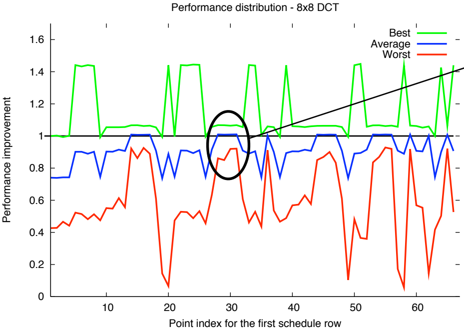
Observations on the Performance Distribution



Large **performance variation**

- Performance variation is large for good values of the first row

Observations on the Performance Distribution



Small **performance variation**

- Performance variation is large for good values of the first row
- It is usually reduced for bad values of the first row

Scanning The Space of Program Versions

The search space:

- Performance **variation** indicates to partition the space: $\vec{\imath} > \vec{p} > c$
- **Non-uniform distribution of performance**
- No clear analytical property of the optimization function

$\rightarrow$ Build dedicated **heuristic** and **genetic operators** aware of these **static and dynamic characteristics**

The Quest for Good Objective Functions

- For data locality, loop tiling is key
  - But what is the cost of tiling?
  - Is tiling the only criterion?
- For coarse-grain parallelism, doall parallelization is key
  - But what is the cost of parallelization?

Dependence Distance Minimization

- Idea: minimize the delay between instances accessing the same data
- Formulation in the polyhedral model:
  - Expression of the delay through parametric form
  - Use all dependences (including RAR)

Definition (Dependence distance minimization)

$$\mathbf{u}_k.\vec{n} + w_k \geq \Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \qquad \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S} \tag{1}$$

$$\mathbf{u}_k \in \mathbb{N}^p, w_k \in \mathbb{N}$$

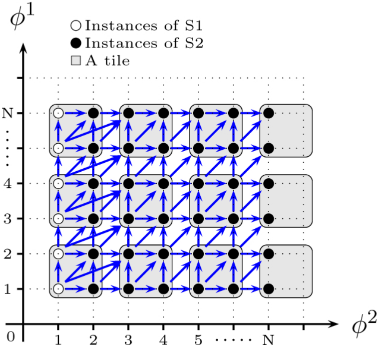
Key Observations

- Minimizing $d = \mathbf{u}_k.\vec{n} + w_k$ minimize the dependence distance
- When $d = 0$ then $\Theta_k^R(\vec{x}_R) = \Theta_k^S(\vec{x}_S)$
  - $0 \geq \Theta_k^R(\vec{x}_R) - \Theta_k^S(\vec{x}_S) \geq 0$
- d gives an indication of the communication volume between hyperplanes

An Overview of Tiling

Tiling: partition the computation into atomic blocs

- Early work in the late 80's
- Motivation: data locality improvement + parallelization

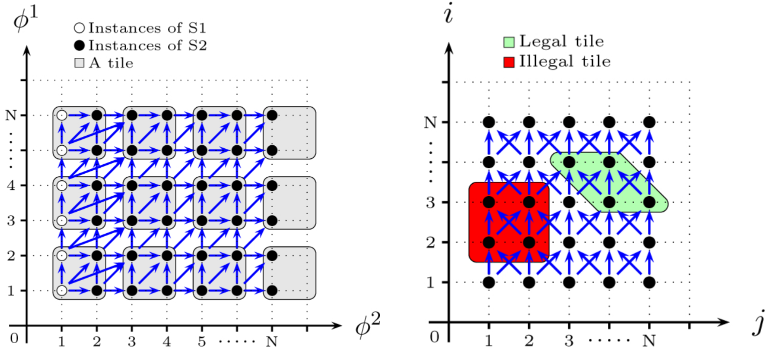


An Overview of Tiling

- Tiling the iteration space
  - It must be valid (dependence analysis required)
  - It may require pre-transformation
  - Unimodular transformation framework limitations
- Supported in current compilers, but limited applicability
- Challenges: imperfectly nested loops, parametric loops, pre-transformations, tile shape, ...
- Tile size selection
  - Critical for locality concerns: determines the footprint
  - Empirical search of the best size (problem + machine specific)
  - Parametric tiling makes the generated code valid for any tile size

Tiling in the Polyhedral Model

- Tiling partition the computation into blocks
- Note we consider only rectangular tiling here
- For tiling to be legal, such a partitioning must be legal



Key Ideas of the Tiling Hyperplane Algorithm

Affine transformations for communication minimal parallelization and locality optimization of arbitrarily nested loop sequences
[Bondhugula et al, CC'08 & PLDI'08]

- Compute a set of transformations to make loops tilable
  - Try to minimize synchronizations
  - Try to maximize locality (maximal fusion)
- Result is a set of *permutable* loops, if possible
  - Strip-mining / tiling can be applied
  - Tiles may be sync-free parallel or pipeline parallel
- Algorithm always terminates (possibly by splitting loops/statements)

Legality of Tiling

Theorem (Legality of Tiling)

Given Θ^R_k, Θ^S_k two one-dimensional schedules. They are valid tiling hyperplanes if

$$\forall \mathcal{D}_{R,S},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},\ \Theta^S_k(\vec{x}_S) - \Theta^R_k(\vec{x}_R) \geq 0$$

- For a schedule to be a legal tiling hyperplane, all communications must go forward: Forward Communication Only [Griebl]
- All dependences must be considered at each level, **including the previously strongly satisfied**
- **Equivalence between loop permutability and loop tilability**

Greedy Algorithm for Tiling Hyperplane Computation

1. Start from the outer-most level, find the set of FCO schedules
2. Select one which minimize the distance between dependent iterations
3. Mark dependences strongly satisfied by this schedule, but do not remove them
4. Formulate the problem for the next level (FCO), adding orthogonality constraints (linear independence)
5. solve again, etc.

<u>Special treatment when no permutable band can be found: splitting</u>

A few properties:

- Result is a set of permutable/tilable outer loops, when possible
- It exhibits coarse-grain parallelism
- Maximal fusion achieved to improve locality

Example: 1D-Jacobi

1-D Jacobi (imperfectly nested)

```
    for (t=1; t<M; t++) {
      for (i=2; i<N-1; i++) {
S:      b[i] = 0.333*(a[i-1]+a[i]+a[i+1]); }
      for (j=2; j<N-1; j++) {
T:      a[j] = b[j]; } }
```

$$\begin{bmatrix} \phi_S^1 \\ \phi_S^2 \end{bmatrix} \begin{pmatrix} t \\ i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

Example: 1D-Jacobi

1-D Jacobi (imperfectly nested)

$$\begin{bmatrix} \phi_S^1 \\ \phi_S^2 \end{bmatrix} \begin{pmatrix} t \\ i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$

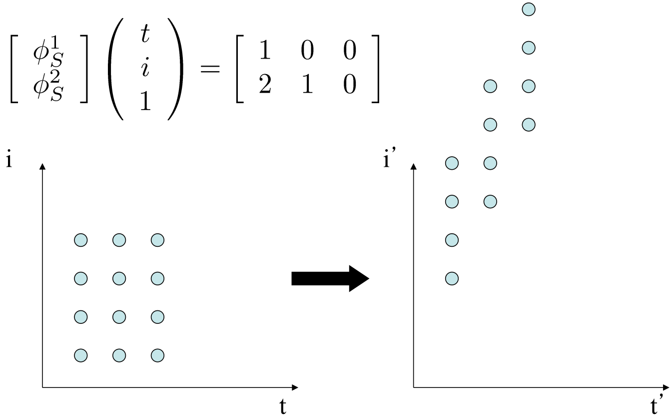
$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

- The resulting transformation is equivalent to a constant shift of one for T relative to S, fusion (j and i are named the same as a result), and skewing the fused i loop with respect to the t loop by a factor of two.
- The (1,0) hyperplane has the least communication: no dependence crosses more than one hyperplane instance along it.

Example: 1D-Jacobi

Transforming S

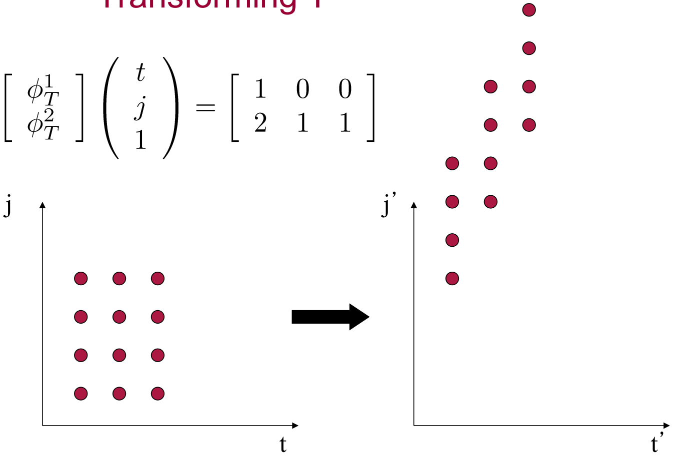
$$\begin{bmatrix} \phi_S^1 \\ \phi_S^2 \end{bmatrix} \begin{pmatrix} t \\ i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$



Example: 1D-Jacobi

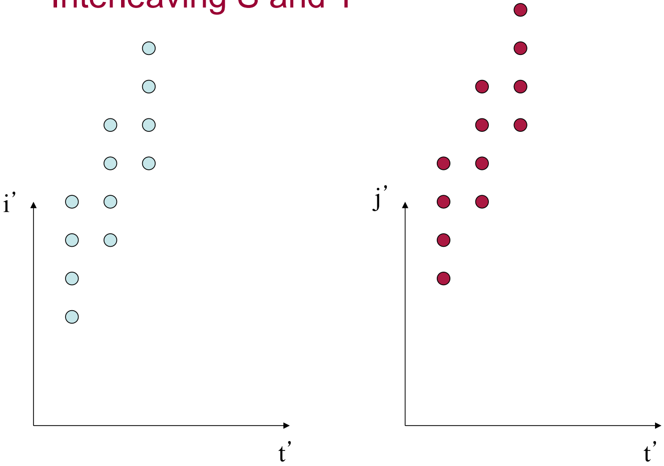
Transforming T

$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$



Example: 1D-Jacobi

Interleaving S and T

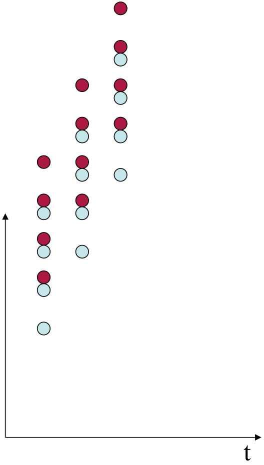


Example: 1D-Jacobi

Interleaving S and T

$$\begin{bmatrix} \phi_S^1 \\ \phi_S^2 \end{bmatrix} \begin{pmatrix} t \\ i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$

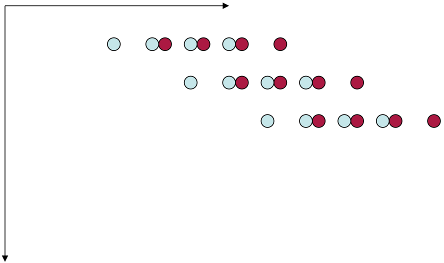
$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$



Example: 1D-Jacobi

1-D Jacobi (imperfectly nested) – transformed code

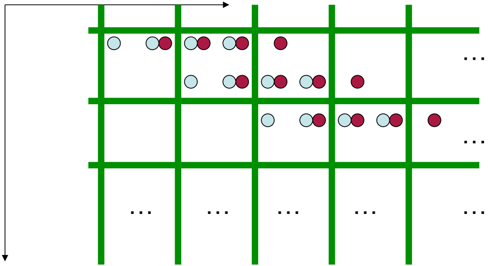
```
    for (t0=0;t0<=M-1;t0++) {
S':   b[2]=0.333*(a[2-1]+a[2]+a[2+1]);
      for (t1=2*t0+3;t1<=2*t0+N-2;t1++) {
S:      b[-2*t0+t1]=0.333*(a[-2*t0+t1-1]+a[-2*t0+t1]
                          +a[-2*t0+t1+1]);
T:      a[-2*t0+t1-1]=b[-2*t0+t1-1]; }
T':   a[N-2]=b[N-2]; }
```



Example: 1D-Jacobi

1-D Jacobi (imperfectly nested) – transformed code

```
    for (t0=0;t0<=M-1;t0++) {
S':   b[2]=0.333*(a[2-1]+a[2]+a[2+1]);
      for (t1=2*t0+3;t1<=2*t0+N-2;t1++) {
S:      b[-2*t0+t1]=0.333*(a[-2*t0+t1-1]+a[-2*t0+t1]
                              +a[-2*t0+t1+1]);
T:      a[-2*t0+t1-1]=b[-2*t0+t1-1]; }
T':   a[N-2]=b[N-2]; }
```



Fusion-driven Optimization

Overview

Problem: How to improve program execution time?

- Focus on shared-memory computation
  - OpenMP parallelization
  - SIMD Vectorization
  - Efficient usage of the intra-node memory hierarchy

- Challenges to address:
  - Different machines require different compilation strategies
  - One-size-fits-all scheme hinders optimization opportunities

Question: how to restructure the code for performance?

Objectives for a Successful Optimization

During the program execution, interplay between the hardware ressources:

- Thread-centric parallelism
- SIMD-centric parallelism
- Memory layout, inc. caches, prefetch units, buses, interconnects...

$\rightarrow$ **Tuning the trade-off between these is required**

A loop optimizer must be able to transform the program for:

- Thread-level parallelism extraction
- Loop tiling, for data locality
- Vectorization

Our approach: form a tractable search space of possible loop transformations

Running Example

Original code

Example ($tmp = A.B$, $D = tmp.C$)

```
  for (i1 = 0; i1 < N; ++i1)
    for (j1 = 0; j1 < N; ++j1) {
R:     tmp[i1][j1] = 0;
      for (k1 = 0; k1 < N; ++k1)
S:       tmp[i1][j1] += A[i1][k1] * B[k1][j1];
    }
  for (i2 = 0; i2 < N; ++i2)
    for (j2 = 0; j2 < N; ++j2) {
T:     D[i2][j2] = 0;
      for (k2 = 0; k2 < N; ++k2)
U:       D[i2][j2] += tmp[i2][k2] * C[k2][j2];
}
```

{R,S} fused, {T,U} fused

	Original	Max. fusion	Max. dist	Balanced
4× Xeon 7450 / ICC 11	$1\times$			
4× Opteron 8380 / ICC 11	$1\times$			

Running Example

Cost model: maximal fusion, minimal synchronization
[Bondhugula et al., PLDI'08]

Example ($tmp = A.B$, $D = tmp.C$)

```
  parfor (c0 = 0; c0 < N; c0++) {
    for (c1 = 0; c1 < N; c1++) {
R:      tmp[c0][c1]=0;
T:      D[c0][c1]=0;
       for (c6 = 0; c6 < N; c6++)
S:        tmp[c0][c1] += A[c0][c6] * B[c6][c1];
       parfor (c6 = 0;c6 <= c1; c6++)
U:        D[c0][c6] += tmp[c0][c1-c6] * C[c1-c6][c6];
       }                                          {R,S,T,U} fused
    for (c1 = N; c1 < 2*N - 1; c1++)
       parfor (c6 = c1-N+1; c6 < N; c6++)
U:        D[c0][c6] += tmp[c0][1-c6] * C[c1-c6][c6];
}
```

	Original	Max. fusion	Max. dist	Balanced
4× Xeon 7450 / ICC 11	$1\times$	$2.4\times$		
4× Opteron 8380 / ICC 11	$1\times$	$2.2\times$		

Running Example

Maximal distribution: best for Intel Xeon 7450

Poor data reuse, best vectorization

Example ($tmp = A.B$, $D = tmp.C$)

```
  parfor (i1 = 0; i1 < N; ++i1)
    parfor (j1 = 0; j1 < N; ++j1)
R:      tmp[i1][j1] = 0;
  parfor (i1 = 0; i1 < N; ++i1)
    for (k1 = 0; k1 < N; ++k1)
      parfor (j1 = 0; j1 < N; ++j1)
S:        tmp[i1][j1] += A[i1][k1] * B[k1][j1];
                                        {R} and {S} and {T} and {U} distributed
  parfor (i2 = 0; i2 < N; ++i2)
    parfor (j2 = 0; j2 < N; ++j2)
T:      D[i2][j2] = 0;
  parfor (i2 = 0; i2 < N; ++i2)
    for (k2 = 0; k2 < N; ++k2)
      parfor (j2 = 0; j2 < N; ++j2)
U:        D[i2][j2] += tmp[i2][k2] * C[k2][j2];
```

	Original	Max. fusion	Max. dist	Balanced
4× Xeon 7450 / ICC 11	1×	2.4×	3.9×	
4× Opteron 8380 / ICC 11	1×	2.2×	6.1×	

Running Example

Balanced distribution/fusion: best for AMD Opteron 8380

Poor data reuse, best vectorization

Example ($tmp = A.B$, $D = tmp.C$)

```
  parfor (c1 = 0; c1 < N; c1++)
    parfor (c2 = 0; c2 < N; c2++)
R:      C[c1][c2] = 0;
  parfor (c1 = 0; c1 < N; c1++)
    for (c3 = 0; c3 < N;c3++) {
T:      E[c1][c3] = 0;
      parfor (c2 = 0; c2 < N;c2++)
S:        C[c1][c2] += A[c1][c3] * B[c3][c2];
    }                          {S,T} fused, {R} and {U} distributed
  parfor (c1 = 0; c1 < N; c1++)
    for (c3 = 0; c3 < N; c3++)
      parfor (c2 = 0; c2 < N; c2++)
U:        E[c1][c2] += C[c1][c3] * D[c3][c2];
```

	Original	Max. fusion	Max. dist	Balanced
4× Xeon 7450 / ICC 11	$1\times$	$2.4\times$	$3.9\times$	$3.1\times$
4× Opteron 8380 / ICC 11	$1\times$	$2.2\times$	$6.1\times$	$8.3\times$

Running Example

Example ($tmp = A.B$, $D = tmp.C$)

```
  parfor (c1 = 0; c1 < N; c1++)
    parfor (c2 = 0; c2 < N; c2++)
R:     C[c1][c2] = 0;
  parfor (c1 = 0; c1 < N; c1++)
    for (c3 = 0; c3 < N;c3++) {
T:     E[c1][c3] = 0;
      parfor (c2 = 0; c2 < N;c2++)
S:       C[c1][c2] += A[c1][c3] * B[c3][c2];
    }                           {S,T} fused, {R} and {U} distributed
  parfor (c1 = 0; c1 < N; c1++)
    for (c3 = 0; c3 < N; c3++)
      parfor (c2 = 0; c2 < N; c2++)
U:       E[c1][c2] += C[c1][c3] * D[c3][c2];
```

	Original	Max. fusion	Max. dist	Balanced
4× Xeon 7450 / ICC 11	1×	2.4×	3.9×	3.1×
4× Opteron 8380 / ICC 11	1×	2.2×	6.1×	8.3×

The best **fusion/distribution choice** drives the quality of the optimization

Loop Structures

Possible grouping + ordering of statements

- **{**{R}, {S}, {T}, {U}**}**; **{**{R}, {S}, {U}, {T}**}**; ...
- **{**{R,S}, {T}, {U}**}**; **{**{R}, {S}, {T,U}**}**; **{**{R}, {T,U}, {S}**}**; **{**{T,U}, {R}, {S}**}**;...
- **{**{R,S,T}, {U}**}**; **{**{R}, {S,T,U}**}**; **{**{S}, {R,T,U}**}**;...
- **{**{R,S,T,U}**}**;

Number of possibilities: $>> n!$ (number of total preorders)

Loop Structures

Removing non-semantics preserving ones

- **{**{R}, {S}, {T}, {U}**}**; {{R}, {S}, {U}, {T}}; ...
- **{**{R,S}, {T}, {U}**}**; **{**{R}, {S}, {T,U}**}**; **{**{R}, {T,U}, {S}**}**; {{T,U}, {R}, {S}};...
- **{**{R,S,T}, {U}**}**; **{**{R}, {S,T,U}**}**; {{S}, {R,T,U}};...
- **{**{R,S,T,U}**}**

Number of possibilities: 1 to 200 for our test suite

Loop Structures

For each partitioning, many possible loop structures

- **{{R}, {S}, {T}, {U}}**
- For **S**: $\{i,j,k\}$; $\{i,k,j\}$; $\{k,i,j\}$; $\{k,j,i\}$; ...
- However, only $\{i,k,j\}$ has:
  - outer-parallel loop
  - inner-parallel loop
  - lowest striding access (efficient vectorization)

Possible Loop Structures for 2mm

- 4 statements, 75 possible partitionings
- 10 loops, up to 10! possible loop structures for a given partitioning

- **Two steps:**
  - Remove all partitionings which breaks the semantics: from 75 to 12
  - Use static cost models to select the loop structure for a partitioning: from $d!$ to 1

- Final search space: **12 possibilites**

Contributions and Overview of the Approach

- Empirical search on possible fusion/distribution schemes
- **Each structure drives the success of other optimizations**
  - Parallelization
  - Tiling
  - Vectorization

- Use static cost models to compute a complex loop transformation **for a specific fusion/distribution scheme**

- Iteratively test the different versions, retain the best
  - **Best performing loop structure is found**

Search Space of Loop Structures

- **Partition the set of statements into classes:**
  - This is deciding loop fusion / distribution
  - Statements in the same class will share at least one common loop in the target code
  - Classes are ordered, to reflect code motion
- **Locally on each partition, apply model-driven optimizations**

- Leverage the polyhedral framework:
  - Build the smallest yet most expressive space of possible partitionings [Pouchet et al., POPL'11]
  - Consider **semantics-preserving partitionings only**: orders of magnitude smaller space

Summary of the Optimization Process

	description	#loops	#stmts	#refs	#deps	#part.	#valid	Variability	Pb. Size
2mm	Linear algebra (BLAS3)	6	4	8	12	75	12	✓	1024x1024
3mm	Linear algebra (BLAS3)	9	6	12	19	4683	128	✓	1024x1024
adi	Stencil (2D)	11	8	36	188	545835	1		1024x1024
atax	Linear algebra (BLAS2)	4	4	10	12	75	16	✓	8000x8000
bicg	Linear algebra (BLAS2)	3	4	10	10	75	26	✓	8000x8000
correl	Correlation (PCA: StatLib)	5	6	12	14	4683	176	✓	500x500
covar	Covariance (PCA: StatLib)	7	7	13	26	47293	96	✓	500x500
doitgen	Linear algebra	5	3	7	8	13	4		128x128x128
gemm	Linear algebra (BLAS3)	3	2	6	6	3	2		1024x1024
gemver	Linear algebra (BLAS2)	7	4	19	13	75	8	✓	8000x8000
gesummv	Linear algebra (BLAS2)	2	5	15	17	541	44	✓	8000x8000
gramschmidt	Matrix normalization	6	7	17	34	47293	1		512x512
jacobi-2d	Stencil (2D)	5	2	8	14	3	1		20x1024x1024
lu	Matrix decomposition	4	2	7	10	3	1		1024x1024
ludcmp	Solver	9	15	40	188	10^{12}	20	✓	1024x1024
seidel	Stencil (2D)	3	1	10	27	1	1		20x1024x1024

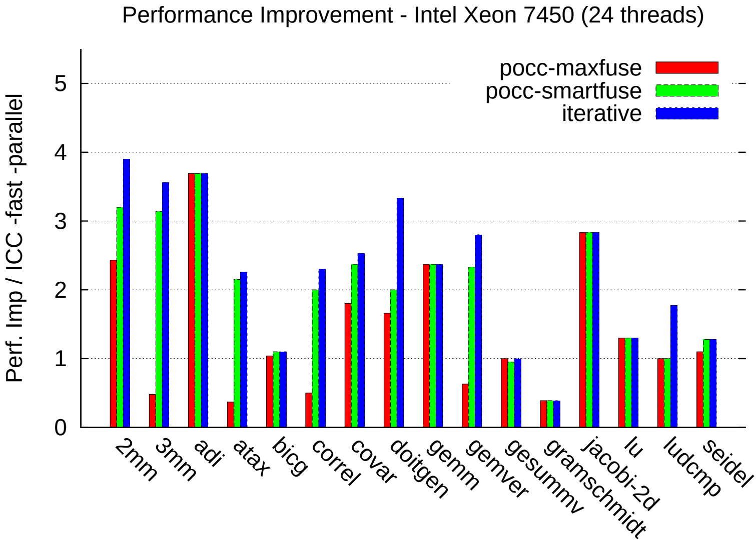
Table: Summary of the optimization process

Experimental Setup

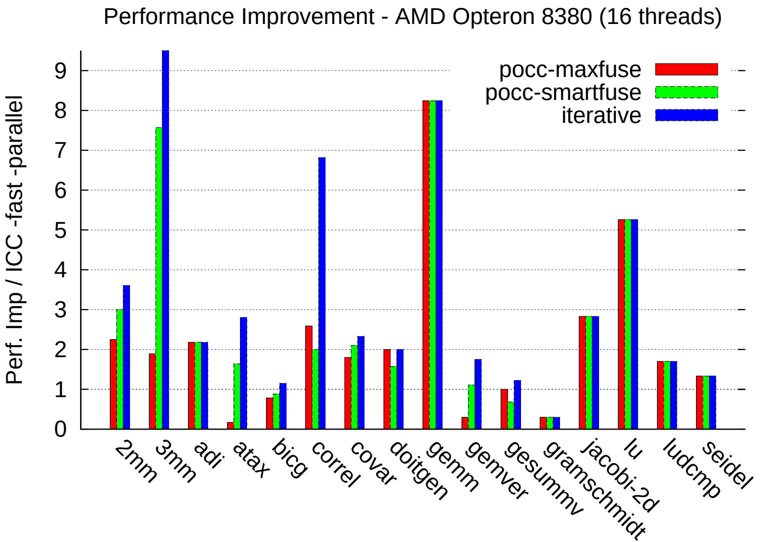
We compare three schemes:

- **maxfuse**: static cost model for fusion (maximal fusion)
- **smartfuse**: static cost model for fusion (fuse only if data reuse)
- **Iterative**: iterative compilation, output the best result

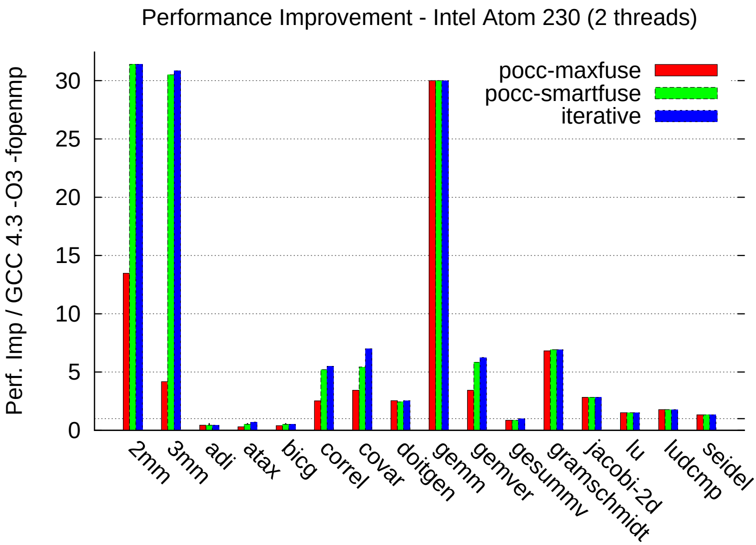
Performance Results - Intel Xeon 7450 - ICC 11



Performance Results - AMD Opteron 8380 - ICC 11



Performance Results - Intel Atom 330 - GCC 4.3



Assessment from Experimental Results

1. Empirical tuning required for **9 out of 16 benchmarks**
2. Strong performance improvements: $2.5\times$ - $3\times$ on average
3. Portability achieved:
   - Automatically **adapt** to the program and target architecture
   - No assumption made about the target
   - Exhaustive search finds the optimal structure (1-176 variants)
4. Substantial improvements over state-of-the-art (up to $2\times$)