Program transformations and optimizations in the polyhedral framework

Louis-Noël Pouchet

Dept. of Computer Science
UCLA

May 14, 2013

1st Polyhedral Spring School
St-Germain au Mont d'or, France

Overview

Agenda for the Lecture

Disclaimer: this lecture is (mainly) for new Ph.D students

1. Program Representation in the Polyhedral Model
2. Applying loop transformations in the model
3. Exact data dependence representation
4. Scheduling using Farkas-based methods
5. Scheduling using approximate data dependences
   - If time permits...

Running Example: DGEMM

Example (dgemm)

```
   /* C := alpha*A*B + beta*C */
   for (i = 0; i < ni; i++)
    for (j = 0; j < nj; j++)
S1:      C[i][j] *= beta;
   for (i = 0; i < ni; i++)
    for (j = 0; j < nj; j++)
      for (k = 0; k < nk; ++k)
S2:        C[i][j] += alpha * A[i][k] * B[k][j];
```

- Loop transformation: *permute(i,k,S2)*

Execution time (in s) on this laptop, GCC 4.2, ni=nj=nk=512

version	-O0	-O1	-O2	-O3 -vec
original	1.81	0.78	0.78	0.78
permute	1.52	0.35	0.35	0.20

http://gcc.gnu.org/onlinedocs/gcc-4.2.1/gcc/Optimize-Options.html

Running Example: fdtd-2d

Example (fdtd-2d)

```
  for(t = 0; t < tmax; t++) {
    for (j = 0; j < ny; j++)
      ey[0][j] = _edge_[t];
    for (i = 1; i < nx; i++)
      for (j = 0; j < ny; j++)
        ey[i][j] = ey[i][j] - 0.5*(hz[i][j]-hz[i-1][j]);
    for (i = 0; i < nx; i++)
      for (j = 1; j < ny; j++)
        ex[i][j] = ex[i][j] - 0.5*(hz[i][j]-hz[i][j-1]);
    for (i = 0; i < nx - 1; i++)
      for (j = 0; j < ny - 1; j++)
        hz[i][j] = hz[i][j] - 0.7* (ex[i][j+1] - ex[i][j] +
                ey[i+1][j]-ey[i][j]);
}
```

- Loop transformation: *polyhedralOpt(fdtd-2d)*

Execution time (in s) on this laptop, GCC 4.2, 64x1024x1024

version	-O0	-O1	-O2	-O3 -vec
original	2.59	1.62	1.54	1.54
polyhedralOpt	2.05	0.41	0.41	0.41

Manual transformations?

Example (fdtd-2d)

```
  for(t = 0; t < tmax; t++) {
    for (j = 0; j < ny; j++)
      ey[0][j] = _edge_[t];
    for (i = 1; i < nx; i++)
      for (j = 0; j < ny; j++)
        ey[i][j] = ey[i][j] - 0.5*(hz[i][j]-hz[i-1][j]);
    for (i = 0; i < nx; i++)
      for (j = 1; j < ny; j++)
        ex[i][j] = ex[i][j] - 0.5*(hz[i][j]-hz[i][j-1]);
    for (i = 0; i < nx - 1; i++)
      for (j = 0; j < ny - 1; j++)
        hz[i][j] = hz[i][j] - 0.7* (ex[i][j+1] - ex[i][j] +
                ey[i+1][j]-ey[i][j]);
}
```

NO NO NO!!!

Example (fdtd-2d tiled)

```
for (c0 = 0; c0 <= (((ny + 2 * tmax + -3) * 32 < 0?((32 < 0?-((-(ny + 2 * tmax + -3) + 32 + 1) / 32) : -((-(ny + 2 *
tmax + -3) + 32 - 1) / 32))) : (ny + 2 * tmax + -3) / 32)); ++c0) {
  #pragma omp parallel for private(c3, c4, c2, c5)
   for (c1 = (((c0 * 2 < 0?-(-c0 / 2) : ((2 < 0?(-c0 + -2 - 1) / -2 : (c0 + 2 - 1) / 2)))) > (((32 * c0 + -tmax + 1) *
32 < 0?-(-(32 * c0 + -tmax + 1) / 32) : ((32 < 0?(-(32 * c0 + -tmax + 1) + -32 - 1) / -32 : (32 * c0 + -tmax + 1 + 32
- 1) / 32))))?((c0 * 2 < 0?-(-c0 / 2) : ((2 < 0?(-c0 + -2 - 1) / -2 : (c0 + 2 - 1) / 2)))) : (((32 * c0 + -tmax + 1) *
32 < 0?-(-(32 * c0 + -tmax + 1) / 32) : ((32 < 0?(-(32 * c0 + -tmax + 1) + -32 - 1) / -32 : (32 * c0 + -tmax + 1 + 32
- 1) / 32))))); c1 <= (((((((ny + tmax + -2) * 32 < 0?((32 < 0?-((-(ny + tmax + -2) + 32 + 1) / 32) : -((-(ny + tmax +
-2) + 32 - 1) / 32))) : (ny + tmax + -2) / 32)) < (((32 * c0 + ny + 30) * 64 < 0?((64 < 0?-((-(32 * c0 + ny + 30) + 64
+ 1) / 64) : -((-(32 * c0 + ny + 30) + 64 - 1) / 64))) : (32 * c0 + ny + 30) / 64))?(((ny + tmax + -2) * 32 < 0?((32 <
0?-((-(ny + tmax + -2) + 32 + 1) / 32) : -((-(ny + tmax + -2) + 32 - 1) / 32))) : (ny + tmax + -2) / 32)) : (((32 * c0 +
ny + 30) * 64 < 0?((64 < 0?-((-(32 * c0 + ny + 30) + 64 + 1) / 64) : -((-(32 * c0 + ny + 30) + 64 - 1) / 64))) : (32 *
c0 + ny + 30) / 64)))) < c0?(((((ny + tmax + -2) * 32 < 0?((32 < 0?-((-(ny + tmax + -2) + 32 + 1) / 32) : -((-(ny + tmax
+ -2) + 32 - 1) / 32))) : (ny + tmax + -2) / 32)) < (((32 * c0 + ny + 30) * 64 < 0?((64 < 0?-((-(32 * c0 + ny + 30) + 64
+ 1) / 64) : -((-(32 * c0 + ny + 30) + 64 - 1) / 64))) : (32 * c0 + ny + 30) / 64))?(((ny + tmax + -2) * 32 < 0?((32 <
0?-((-(ny + tmax + -2) + 32 + 1) / 32) : -((-(ny + tmax + -2) + 32 - 1) / 32))) : (ny + tmax + -2) / 32)) : (((32 * c0 +
ny + 30) * 64 < 0?((64 < 0?-((-(32 * c0 + ny + 30) + 64 + 1) / 64) : -((-(32 * c0 + ny + 30) + 64 - 1) / 64))) : (32 *
c0 + ny + 30) / 64)))) : c0)); ++c1) {
     for (c2 = c0 + -c1; c2 <= (((((tmax + nx + -2) * 32 < 0?((32 < 0?-((-(tmax + nx + -2) + 32 + 1) / 32) : -((-(tmax +
nx + -2) + 32 - 1) / 32))) : (tmax + nx + -2) / 32)) < (((32 * c0 + -32 * c1 + nx + 30) * 32 < 0?((32 < 0?-((-(32 * c0 +
-32 * c1 + nx + 30) + 32 + 1) / 32) : -((-(32 * c0 + -32 * c1 + nx + 30) + 32 - 1) / 32))) : (32 * c0 + -32 * c1 + nx +
30) / 32))?(((tmax + nx + -2) * 32 < 0?((32 < 0?-((-(tmax + nx + -2) + 32 + 1) / 32) : -((-(tmax + nx + -2) + 32 - 1) /
32))) : (tmax + nx + -2) / 32)) : (((32 * c0 + -32 * c1 + nx + 30) * 32 < 0?((32 < 0?-((-(32 * c0 + -32 * c1 + nx + 30)
+ 32 + 1) / 32) : -((-(32 * c0 + -32 * c1 + nx + 30) + 32 - 1) / 32))) : (32 * c0 + -32 * c1 + nx + 30) / 32)))); ++c2)
{
       if (c0 == 2 * c1 && c0 == 2 * c2) {
         for (c3 = 16 * c0; c3 <= ((tmax + -1 < 16 * c0 + 30?tmax + -1 : 16 * c0 + 30)); ++c3)
            if (c0 % 2 == 0)
               (ey[0])[0] = (_edge_[c3]);
........... (200 more lines!)
```

Performance gain: 2-6× on modern multicore platforms

Loop Transformations in Production Compilers

Limitations of standard syntactic frameworks:

- Composition of transformations may be tedious
  - composability rules / applicability
- Parametric loop bounds, imperfectly nested loops are challenging
  - Look at the examples!
- Approximate dependence analysis
  - Miss parallelization opportunities (among many others)
- (Very) conservative performance models

Achievements of Polyhedral Compilation

The polyhedral model:

- Model/apply seamlessly arbitrary compositions of transformations
  - Automatic handling of imperfectly nested, parametric loop structures
  - Many loop transformation can be modeled
- Exact dependence analysis on a class of programs
  - Unleash the power of automatic parallelization
  - Aggressive multi-objective program restructuring (parallelism, SIMD, cache, etc.)
- Requires computationally expensive algorithms
  - Usually NP-complete / exponential complexity
  - Requires careful problem statement/representation

Research around Polyhedral Compilation

"first generation"

- Provided the theoretical foundations
- Focused on high-level/abstract objectives
- Was questioned about how practical is this model

"second generation"

- Provided practical / "more scalable" algorithms
- Focused on concrete performance improvement
- Still often questioned about how practical is this model
  - Thanks for the legacy! ;-)

Perspective on 25 Years of Computing

"first generation"

- Parallel machines were... rare
- Era of increased program performance for free
- Intel 486 DX2: 54 MIPS, 66MHz

"second generation"

- Parallel machines are ubiquitous (multi-core, SIMD, GPUs)
- Very difficult to get good performance
- Intel Core i7: 177730 MIPS, 3.33 GHz

Perspective on 25 Years of Computing

"first generation"

- Parallel machines were... rare
- Era of increased program performance for free
- Intel 486 DX2: 54 MIPS, 66MHz

"second generation"

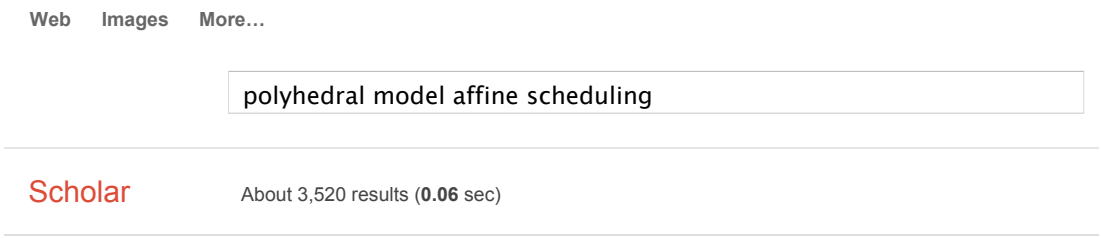
- Parallel machines are ubiquitous (multi-core, SIMD, GPUs)
- Very difficult to get good performance
- Intel Core i7: 177730 MIPS, 3.33 GHz

More complex problem can be solved, but more complex objectives are needed anyway

Disclaimer

This lecture is:

- Partial



- Biased (strongly!)
- Provocative at times (hopefully ;-))

Polyhedral Program Representation

Example: DGEMM

Example (dgemm)

```
   /* C := alpha*A*B + beta*C */
   for (i = 0; i < ni; i++)
    for (j = 0; j < nj; j++) {
S1:      C[i][j] *= beta;
      for (k = 0; k < nk; ++k)
S2:        C[i][j] += alpha * A[i][k] * B[k][j];
      }
```

This code has:

- imperfectly nested loops
- multiple statements
- parametric loop bounds

Granularity of Program Representation

DGEMM has:

- 3 loops
  - For loops in the code, while loops
  - Control-flow graph analysis
- 2 (syntactic) statements
  - Input source code?
  - Basic block?
  - ASM instructions?
- `S1` is executed `ni` $\times$ `nj` times
  - dynamic instances of the statement
  - Does not (necessarily) correspond to reality!

Some Observations

Reasoning at the loop/statement level:

- Some loop transformation may be very difficult to implement
  - How to fuse loops with different loop bounds?
  - How to permute triangular loops?
  - How to unroll-and-jam triangular loops?
  - How to apply time-tiling?
  - ...
- Statements may operate on the same array while being independent

Some Motivations for Polyhedral Transformations

- Known problem: scheduling of task graph
- Obvious limitations: task graph is not finite / size depends on problem / effective code generation almost impossible
- Alternative approach: use loop transformations
  - solve all above limitation
  - BUT the problem is to find a sequence that implements the order we want
  - AND also how to apply/compose them

- Desired features:
  - ability to reason at the instance level (as for task graph scheduling)
  - ability to easily apply/compose loop transformations

The Polyhedral Model

Polyhedral Program Optimization: a Three-Stage Process

1 Analysis: from code to model

→ Existing prototype tools

- URUK, Omega, ChiLL . . .
- ISL, Loopo
- PolyOpt/PoCC
  (Clan-Candl-LetSee-Pluto-Ponos-Cloog-Polylib-PIPLib-ISL-FM-...)

→ GCC GRAPHITE, LLVM Polly (now in mainstream)

→ Reservoir Labs R-Stream, IBM XL/Poly

Polyhedral Program Optimization: a Three-Stage Process

1 Analysis: from code to model

→ Existing prototype tools

- URUK, Omega, ChiLL . . .
- ISL, Loopo
- PolyOpt/PoCC
  (Clan-Candl-LetSee-Pluto-Ponos-Cloog-Polylib-PIPLib-ISL-FM-...)

→ GCC GRAPHITE, LLVM Polly (now in mainstream)

→ Reservoir Labs R-Stream, IBM XL/Poly

2 Transformation in the model

→ Build and select a program transformation

Polyhedral Program Optimization: a Three-Stage Process

1 Analysis: from code to model

→ Existing prototype tools

- URUK, Omega, ChiLL . . .
- ISL, Loopo
- PolyOpt/PoCC
  (Clan-Candl-LetSee-Pluto-Ponos-Cloog-Polylib-PIPLib-ISL-FM-...)

→ GCC GRAPHITE, LLVM Polly (now in mainstream)

→ Reservoir Labs R-Stream, IBM XL/Poly

2 Transformation in the model

→ Build and select a program transformation

3 Code generation: from model to code

→ "Apply" the transformation in the model

→ Regenerate syntactic (AST-based) code

Motivating Example [1/2]

Example

```
for (i = 0; i <= 1; ++i)
  for (j = 0; j <= 2; ++j)
    A[i][j] = i * j;
```

Program execution:

```
1: A[0][0] = 0 * 0;
2: A[0][1] = 0 * 1;
3: A[0][2] = 0 * 2;
4: A[1][0] = 1 * 0;
5: A[1][1] = 1 * 1;
6: A[1][2] = 1 * 2;
```

Motivating Example [2/2]

A few observations:

- Statement is executed 6 times
- There is a different values for i,j associated to these 6 instances
- There is an order on them (the execution order)

A rough analogy: polyhedral compilation is about (statically) scheduling tasks, where tasks are statement <u>instances</u>, or operations

Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only (over-approximation otherwise)

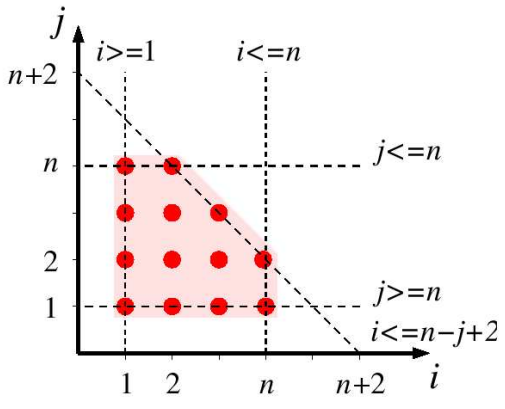
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only (over-approximation otherwise)
- Iteration domain: represented as integer polyhedra

```
for (i=1; i<=n; ++i)
. for (j=1; j<=n; ++j)
. . if (i<=n-j+2)
. . . s[i] = ...
```

$$\mathcal{D}_{S1} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 2 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix} \geq \vec{0}$$



Iteration domain of S_1

Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only (over-approximation otherwise)
- Iteration domain: represented as integer polyhedra
- Memory accesses: static references, represented as affine functions of $\vec{x_S}$ and $\vec{p}$

```
for (i=0; i<n; ++i) {
. s[i] = 0;
. for (j=0; j<n; ++j)
. . s[i] = s[i]+a[i][j]*x[j];
}
```

$$f_{\mathbf{s}}(\vec{x_{S2}}) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathbf{a}}(\vec{x_{S2}}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathbf{x}}(\vec{x_{S2}}) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

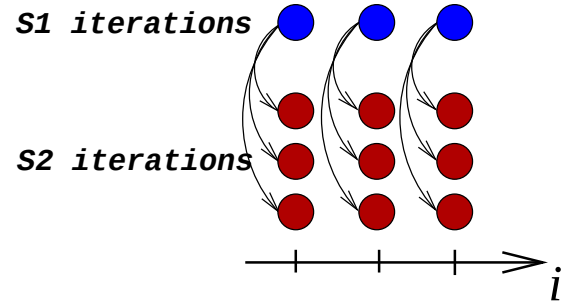
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only (over-approximation otherwise)
- Iteration domain: represented as integer polyhedra
- Memory accesses: static references, represented as affine functions of $\vec{x}_S$ and $\vec{p}$
- Data dependence between S1 and S2: a subset of the Cartesian product of $\mathcal{D}_{S1}$ and $\mathcal{D}_{S2}$ (**exact analysis**)

```
for (i=1; i<=3; ++i) {
. s[i] = 0;
. for (j=1; j<=3; ++j)
. . s[i] = s[i] + 1;
}
```

$$\mathcal{D}_{S1\delta S2} : \left[\begin{array}{cccc} 1 & -1 & 0 & 0 \\ \hline 1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 3 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 0 & 3 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 3 \end{array}\right] \cdot \begin{pmatrix} i_{S1} \\ i_{S2} \\ j_{S2} \\ 1 \end{pmatrix} \begin{array}{c} = 0 \\ \hline \geq \vec{0} \end{array}$$



Math Corner

The Affine Qualifier

Definition (Affine function)

A function $f : \mathbb{K}^m \to \mathbb{K}^n$ is affine if there exists a vector $\vec{b} \in \mathbb{K}^n$ and a matrix $A \in \mathbb{K}^{m \times n}$ such that:

$$\forall \vec{x} \in \mathbb{K}^m,\ f(\vec{x}) = A\vec{x} + \vec{b}$$

Definition (Affine half-space)

An affine half-space of $\mathbb{K}^m$ (affine constraint) is defined as the set of points:

$$\{\vec{x} \in \mathbb{K}^m \mid \vec{a}.\vec{x} \leq \vec{b}\}$$

Polyhedron (Implicit Representation)

Definition (Polyhedron)

A set $\mathcal{S} \in \mathbb{K}^m$ is a polyhedron if there exists a system of a finite number of inequalities $A\vec{x} \leq \vec{b}$ such that:

$$\mathcal{P} = \{\vec{x} \in \mathbb{K}^m \mid A\vec{x} \leq \vec{b}\}$$

Equivalently, it is the intersection of finitely many half-spaces.

Definition (Polytope)

A polytope is a bounded polyhedron.

Integer Polyhedron

Definition ($\mathbb{Z}$-polyhedron)

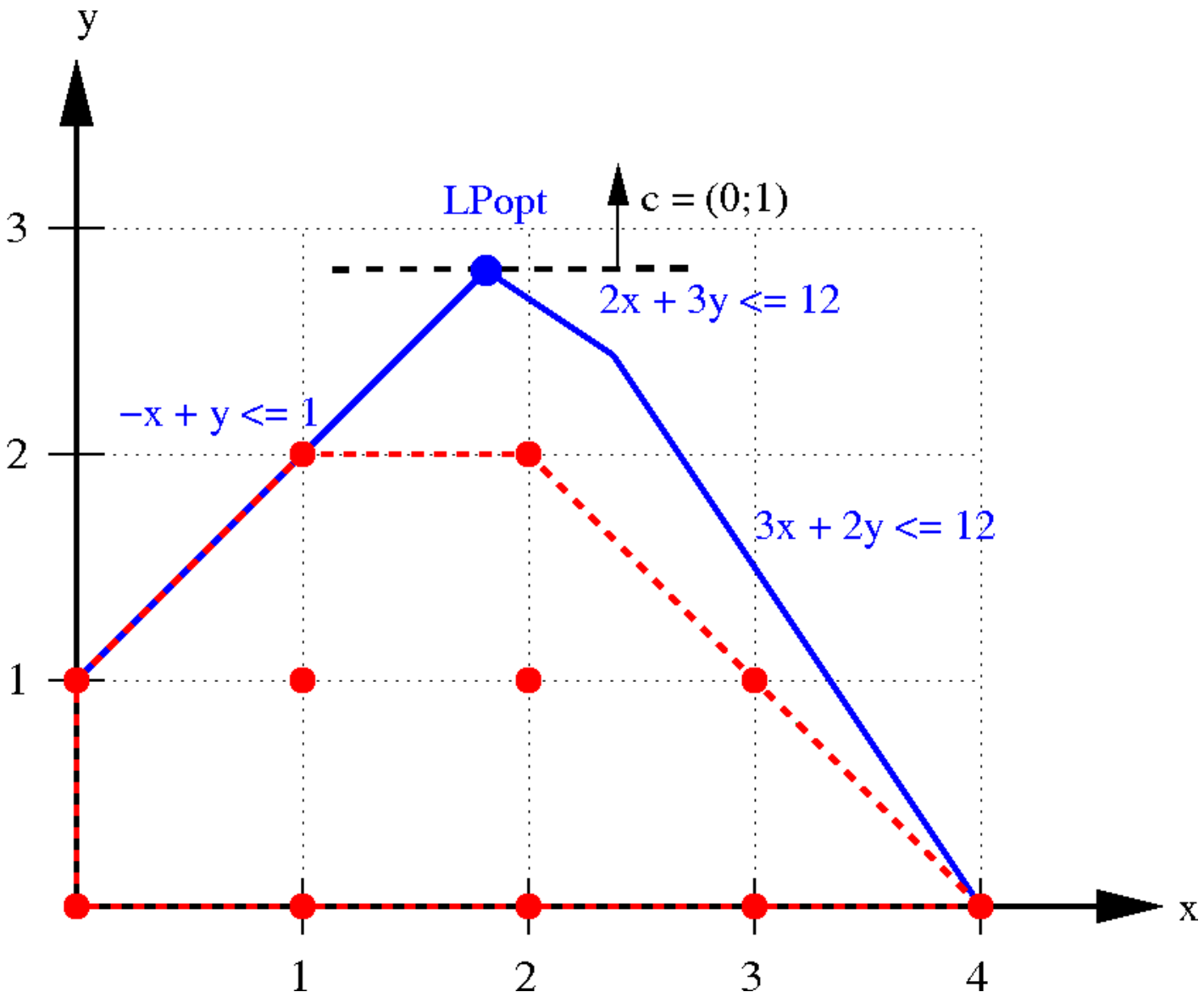
It is a polyhedron where all its extreme points are integer valued

Definition (Integer hull)

The integer hull of a rational polyhedron $\mathcal{P}$ is the largest set of integer points such that each of these points is in $\mathcal{P}$.

For the moment, we will "say" an integer polyhedron is a polyhedron of integer points (language abuse)

Rational and Integer Polytopes



Another View of Polyhedra

The dual representation models a polyhedron as a combination of lines L and rays R (forming the polyhedral cone) and vertices V (forming the polytope)

Definition (Dual representation)

$$\mathcal{P} : \{\vec{x} \in \mathbb{Q}^n \mid \vec{x} = L\vec{\lambda} + R\vec{\mu} + V\vec{\nu},\ \vec{\mu} \geq 0,\ \vec{\nu} \geq 0,\ \sum_i \nu_i = 1\}$$

Definition (Face)

A face $\mathcal{F}$ of $\mathcal{P}$ is the intersection of $\mathcal{P}$ with a supporting hyperplane of $\mathcal{P}$. We have:

$$dim(\mathcal{F}) \leq dim(\mathcal{P})$$

Definition (Facet)

A facet $\mathcal{F}$ of $\mathcal{P}$ is a face of $\mathcal{P}$ such that:

$$dim(\mathcal{F}) = dim(\mathcal{P}) - 1$$

Some Equivalence Properties

Theorem (Fundamental Theorem on Polyhedral Decomposition)

If $\mathcal{P}$ is a polyhedron, then it can be decomposed as a polytope $\mathcal{V}$ plus a polyhedral cone $\mathcal{L}$.

Theorem (Equivalence of Representations)

Every polyhedron has both an implicit and dual representation

- Chernikova's algorithm can compute the dual representation from the implicit one
- The Dual representation is heavily used in polyhedral compilation
- Some works operate on the constraint-based representation (Pluto)

Parametric Polyhedra

Definition (Paramteric polyhedron)

Given $\vec{n}$ the vector of symbolic parameters, $\mathcal{P}$ is a parametric polyhedron if it is defined by:

$$\mathcal{P} = \{\vec{x} \in \mathbb{K}^m \mid A\vec{x} \leq B\vec{n} + \vec{b}\}$$

- Requires to adapt theory and tools to parameters
- Can become nasty: case distinctions (QUAST)
- Reflects nicely the program **context**

Some Useful Algorithms

All extended to parametric polyhedra:

- Compute the facets of a polytope: **PolyLib** [Wilde et al]
- Compute the volume of a polytope (number of points): **Barvinok** [Claus/Verdoolaege]
- Scan a polytope (code generation): **CLooG** [Quillere/Bastoul]
- Find the lexicographic minimum: **PIP** [Feautrier]

Operations on Polyhedra

Definition (Intersection)

The intersection of two convex sets $\mathcal{P}_1$ and $\mathcal{P}_2$ is a convex set $\mathcal{P}$:

$$\mathcal{P} = \{\vec{x} \in \mathbb{K}^m \mid \vec{x} \in \mathcal{P}_1 \wedge \vec{x} \in \mathcal{P}_2\}$$

Definition (Union)

The union of two convex sets $\mathcal{P}_1$ and $\mathcal{P}_2$ is a set $\mathcal{P}$:

$$\mathcal{P} = \{\vec{x} \in \mathbb{K}^m \mid \vec{x} \in \mathcal{P}_1 \vee \vec{x} \in \mathcal{P}_2\}$$

The union of two convex sets may not be a convex set

Lattices

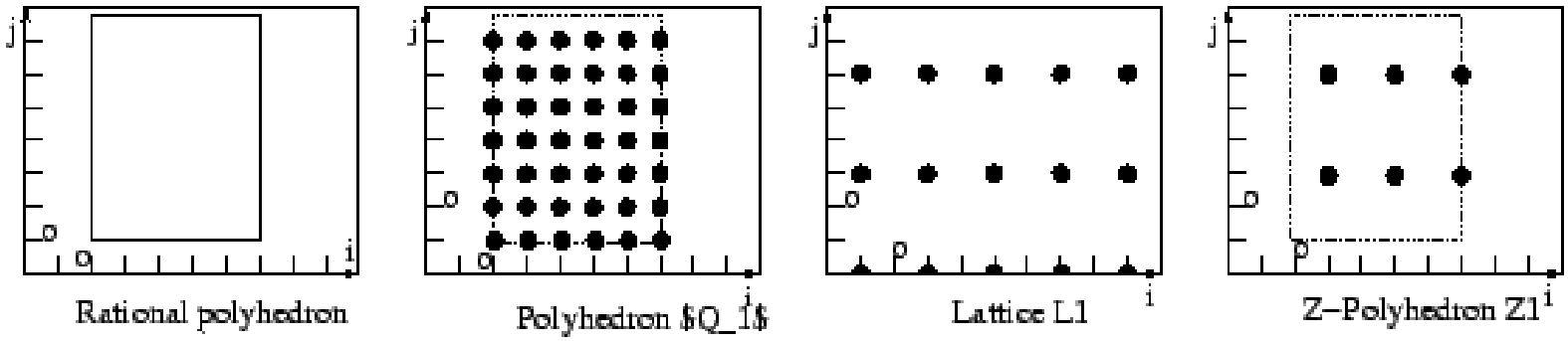
Definition (Lattice)

A subset L in $\mathbb{Q}^n$ is a lattice if is generated by integral combination of finitely many vectors: $a_1, a_2, \ldots, a_n$ ($a_i \in \mathbb{Q}^n$). If the a_i vectors have integral coordinates, L is an integer lattice.

Definition ($\mathcal{Z}$-polyhedron)

A $\mathcal{Z}$-polyhedron is the intersection of a polyhedron and an affine integral full dimensional lattice.

Pictured Example



Example of a $\mathcal{Z}$-polyhedron:

- $Q_1 = \{i,j \mid 0 \leq i \leq 5,\ 0 \leq 3j \leq 20\}$
- $L_1 = \{2i+1, 3j+5 \mid i,j \in \mathbb{Z}\}$
- $Z_1 = Q_1 \cap L_1$

Quick Facts on $\mathcal{Z}$-polyhedra

- **Iteration domains are in fact $\mathcal{Z}$-polyhedra with unit lattice**
- Intersection of $\mathcal{Z}$-polyhedra is not convex in general
- Union is complex to compute
- **Can count points, can optimize, can scan**

- **Implementation available for most operations in PolyLib**

Image and Preimage

Definition (Image)

The image of a polyhedron $\mathcal{P} \in \mathbb{Z}^n$ by an affine function $f : \mathbb{Z}^n \to \mathbb{Z}^m$ is a $\mathcal{Z}$-polyhedron $\mathcal{P}'$:

$$\mathcal{P}' = \{f(\vec{x}) \in \mathbb{Z}^m \mid \vec{x} \in \mathcal{P}\}$$

Definition (Preimage)

The preimage of a polyhedron $\mathcal{P} \in \mathbb{Z}^n$ by an affine function $f : \mathbb{Z}^n \to \mathbb{Z}^m$ is a $\mathcal{Z}$-polyhedron $\mathcal{P}'$:

$$\mathcal{P}' = \{\vec{x} \in \mathbb{Z}^n \mid f(\vec{x}) \in \mathcal{P}\}$$

We have $Image(f^{-1}, \mathcal{P}) = Preimage(f, \mathcal{P})$ if f is invertible.

Relation Between Image, Preimage and $\mathcal{Z}$-polyhedra

- The image of a $\mathbb{Z}$-polyhedron by an unimodular function is a $\mathbb{Z}$-polyhedron
- The preimage of a $\mathbb{Z}$-polyhedron by an affine function is a $\mathbb{Z}$-polyhedron
- The image of a polyhedron by an affine invertible function is a $\mathcal{Z}$-polyhedron
- The preimage of a $\mathcal{Z}$-polyhedron by an affine function is a $\mathcal{Z}$-polyhedron
- The image by a non-invertible function is **not** a $\mathcal{Z}$-polyhedron

Program Transformations

What Can Be Modeled?

Exact vs. approximate representation:

- Exact representation of iteration domains
  - Static control flow
  - Affine loop bounds (includes min/max/integer division)
  - Affine conditionals (conjunction/disjunction)
- Approximate representation of iteration domains
  - Use affine over-approximations, predicate statement executions
  - Full-function support

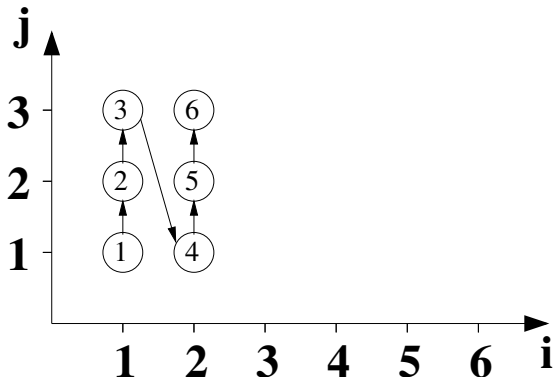
Key Intuition

- Programs are represented with geometric shapes
- Transforming a program is about modifying those shapes
  - rotation, skewing, stretching, ...
- But we need here to assume some order to scan points

Affine Transformations

Interchange Transformation

The transformation matrix is the identity with a permutation of two rows.



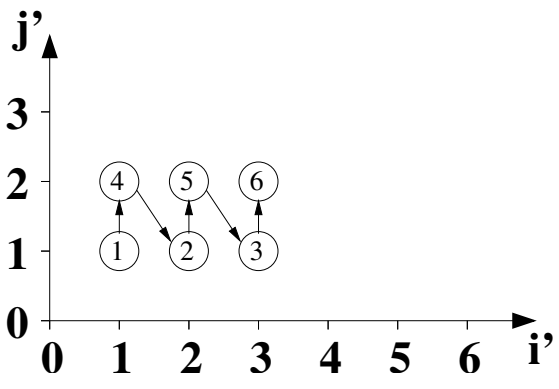
$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron
$A\vec{x} + \vec{a} \geq \vec{0}$

$\Longrightarrow$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function
$\vec{y} = T\vec{x}$



$$\begin{bmatrix} 0 & 1 \\ 0 & -1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron
$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

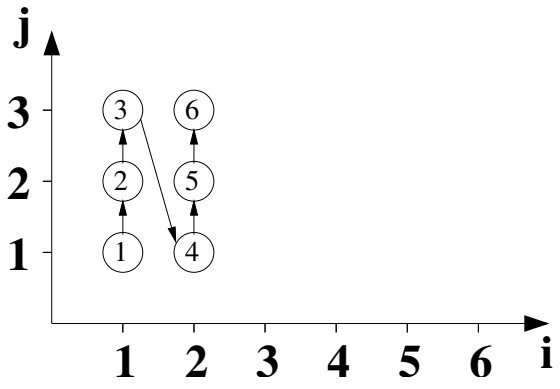
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do i' = 1, 3
  do j' = 1, 2
     S(i=j',j=i')
```

Affine Transformations

Reversal Transformation

The transformation matrix is the identity with one diagonal element replaced by -1.



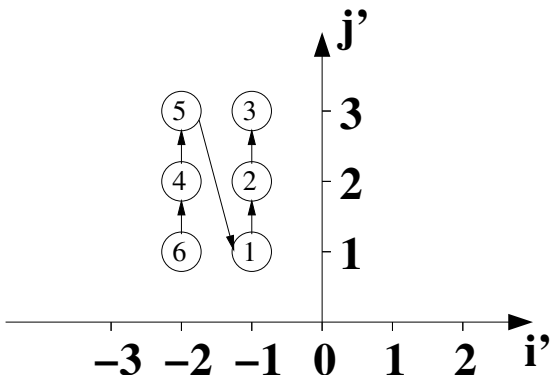
$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron
$A\vec{x} + \vec{a} \geq \vec{0}$

$\Longrightarrow$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function
$\vec{y} = T\vec{x}$



$$\begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron
$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

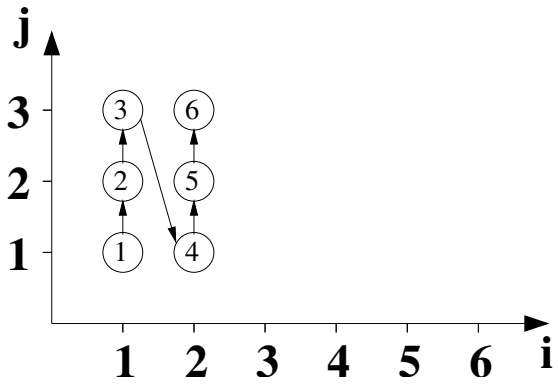
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do i' = -1, -2, -1
  do j' = 1, 3
     S(i=3-i',j=j')
```

Affine Transformations

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



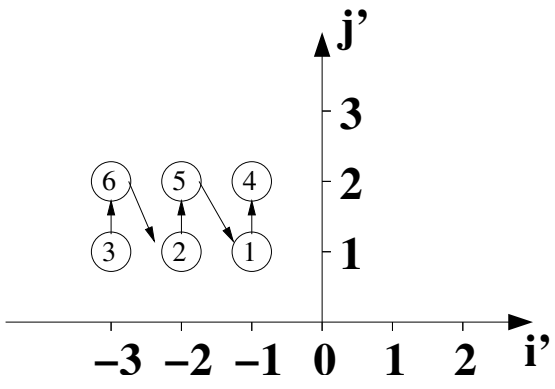
$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron
$A\vec{x} + \vec{a} \geq \vec{0}$

$\Longrightarrow$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function
$\vec{y} = T\vec{x}$



$$\begin{bmatrix} 0 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron
$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

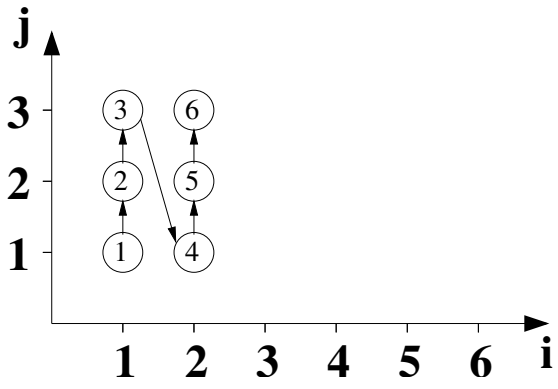
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do j' = -1, -3, -1
  do i' = 1, 2
     S(i=4-j',j=i')
```

Affine Transformations

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



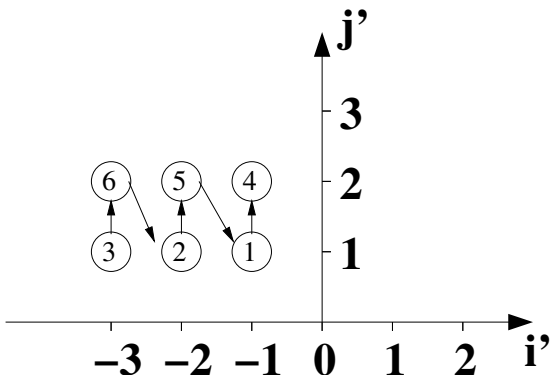
$$\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(a) original polyhedron
$A\vec{x} + \vec{a} \geq \vec{0}$

$\Longrightarrow$

$$\begin{pmatrix} i' \\ j' \end{pmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

(b) transformation function
$\vec{y} = T\vec{x}$



$$\begin{bmatrix} 0 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 0 \end{bmatrix} \begin{pmatrix} i' \\ j' \end{pmatrix} + \begin{pmatrix} -1 \\ 2 \\ -1 \\ 3 \end{pmatrix} \geq \vec{0}$$

(c) target polyhedron
$(AT^{-1})\vec{y} + \vec{a} \geq \vec{0}$

```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

```
do j' = -1, -3, -1
  do i' = 1, 2
     S(i=4-j',j=i')
```

So, What is This Matrix?

- We know how to generate code for arbitrary matrices with integer coefficients
  - Arbitrary number of rows (but fixed number of columns)
  - Arbitrary value for the coefficients
- Through code generation, the number of dynamic instances is preserved
- **But this is not true for the transformed polyhedra!**

Some classification:

- The matrix is unimodular
- The matrix is full rank and invertible
- The matrix is full rank
- The matrix has only integral coefficients
- The matrix has rational coefficients

A Reverse History Perspective

1. CLooG: arbitrary matrix
2. Affine Mappings
3. Unimodular framework
4. SARE
5. SURE

Program Transformations

Original Schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j){
  C[i][j] = 0;
  for (k = 0; k < n; ++k)
   C[i][j] += A[i][k]*
              B[k][j];
 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

Program Transformations

Original Schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j){
  C[i][j] = 0;
  for (k = 0; k < n; ++k)
   C[i][j] += A[i][k]*
              B[k][j];
 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

Program Transformations

Original Schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j){
  C[i][j] = 0;
  for (k = 0; k < n; ++k)
   C[i][j] += A[i][k]*
              B[k][j];
 }
```

- Represent Static Control Parts (control flow and dependences must be statically computable)
- Use code generator (e.g. CLooG) to generate C code from polyhedral representation (provided iteration domains + schedules)

Program Transformations

Distribute loops

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  for (k = 0; k < n; ++k)
   C[i-n][j] += A[i-n][k]*
                B[k][j];
```

- **All instances of S1 are executed before the first S2 instance**

Program Transformations

Distribute loops + Interchange loops for S2

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & \mathbf{1} & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \mathbf{1} & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k = n; k < 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n]*
              B[k-n][j];
```

- **The outer-most loop for S2 becomes k**

Program Transformations

Illegal schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (k = 0; k < n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k]*
              B[k][j];
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i-n][j] = 0;
```

- **All instances of S1 are executed <u>after</u> the last S2 instance**

Program Transformations

A legal schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

- **Delay the S2 instances**
- Constraints must be expressed between Θ^{S1} and Θ^{S2}

Program Transformations

Implicit fine-grain parallelism

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \end{pmatrix} . \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \\ \mathbf{n} \\ \mathbf{1} \end{pmatrix}$$

```
for (i = 0; i < n; ++i)
 pfor (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k = n; k < 2*n; ++k)
 pfor (j = 0; j < n; ++j)
  pfor (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n]*
              B[k-n][j];
```

- **Less (linear) rows than loop depth $\rightarrow$ remaining dimensions are parallel**

Program Transformations

Representing a schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

$$\Theta.\vec{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i & j & i & j & k & n & n & 1 & 1 \end{pmatrix}^T$$

Program Transformations

Representing a schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

$$\Theta.\vec{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i & j & i & j & k & n & n & 1 & 1 \end{pmatrix}^T$$

$$\vec{\imath} \qquad \vec{p} \qquad \mathbf{c}$$

Program Transformations

Representing a schedule

```
 for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j){
S1: C[i][j] = 0;
   for (k = 0; k < n; ++k)
S2:  C[i][j] += A[i][k]*
                B[k][j];
  }
```

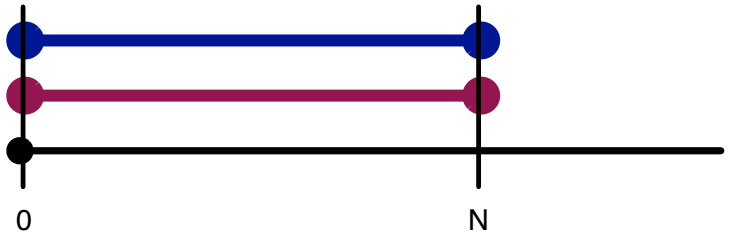
$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

$$\Theta^{S2}.\vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}$$

```
for (i = n; i < 2*n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (k= n+1; k<= 2*n; ++k)
 for (j = 0; j < n; ++j)
  for (i = 0; i < n; ++i)
   C[i][j] += A[i][k-n-1]*
              B[k-n-1][j];
```

	Transformation	Description
$\vec{\imath}$	`reversal`	Changes the direction in which a loop traverses its iteration range
	`skewing`	Makes the bounds of a given loop depend on an outer loop counter
	`interchange`	Exchanges two loops in a perfectly nested loop, a.k.a. `permutation`
$\vec{p}$	`fusion`	Fuses two loops, a.k.a. `jamming`
	`distribution`	Splits a single loop nest into many, a.k.a. `fission` or `splitting`
c	`peeling`	Extracts one iteration of a given loop
	`shifting`	Allows to reorder loops

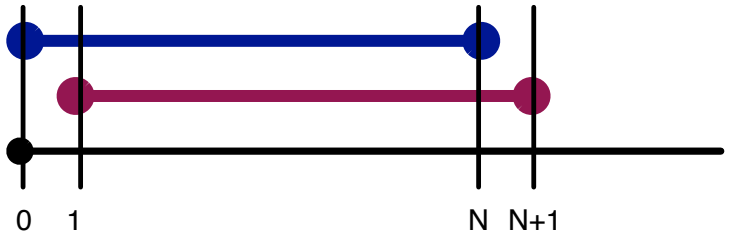
Fusion in the Polyhedral Model




```
for (i = 0; i <= N; ++i) {
  Blue(i);
  Red(i);
}
```

Perfectly aligned fusion

Fusion in the Polyhedral Model

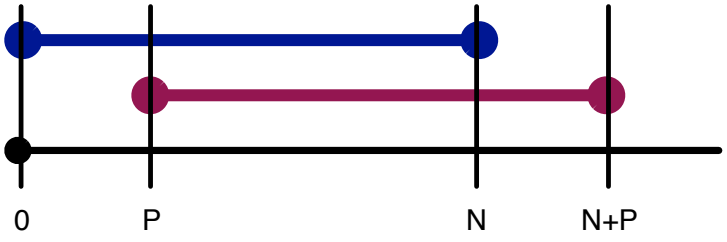



```
Blue(0);
for (i = 1; i <= N; ++i) {
  Blue(i);
  Red(i-1);
}
Red(N);
```

Fusion with shift of 1

Not all instances are fused

Fusion in the Polyhedral Model

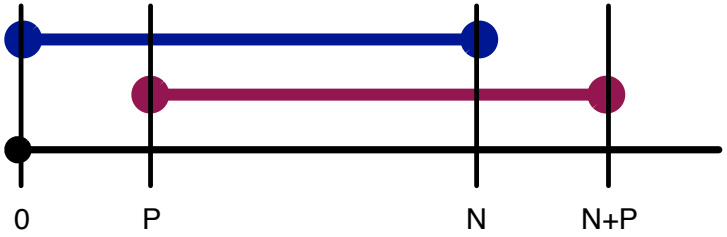



```
for (i = 0; i < P; ++i)
  Blue(i);
for (i = P; i <= N; ++i) {
  Blue(i);
  Red(i-P);
}
for (i = N+1; i <= N+P; ++i)
  Red(i-P);
```

Fusion with parametric shift of P

Automatic generation of prolog/epilog code

Fusion in the Polyhedral Model




```
for (i = 0; i < P; ++i)
  Blue(i);
for (i = P; i <= N; ++i) {
  Blue(i);
  Red(i-P);
}
for (i = N+1; i <= N+P; ++i)
  Red(i-P);
```

Many other transformations may be required to enable fusion: interchange, skewing, etc.

Scheduling Matrix

Definition (Affine multidimensional 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:

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

Θ^S_k denotes the kth row of Θ^S.

Definition (Bounded affine multidimensional schedule)

Θ^S is a bounded schedule if $\theta^S_{i,j} \in [x, y]$ with $x, y \in \mathbb{Z}$

Another Representation

One can separate coefficients of Θ into:

1. The iterator coefficients
2. The parameter coefficients
3. The constant coefficients

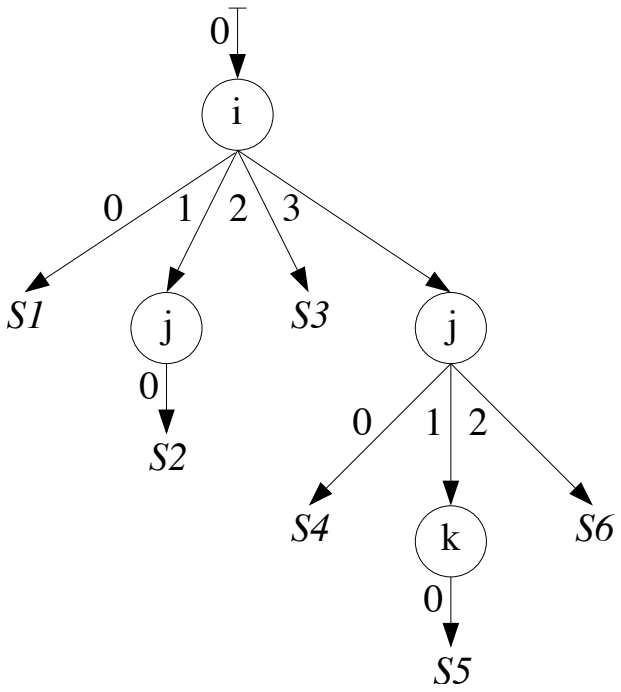
One can also enforce the schedule dimension to be 2d+1.

- A d-dimensional square matrix for the linear part
  - represents composition of interchange/skewing/slowing
- A $d \times n$ matrix for the parametric part
  - represents (parametric) shift
- A $d + 1$ vector for the scalar offset
  - represents statement interleaving

- See URUK for instance

Computing the 2d+1 Identity Schedule

```
    do i=1, n
S1    x = a(i,i)
      do j=1, i-1
S2      x = x - a(i,j)**2
S3    p(i) = 1.0/sqrt(x)
      do j=i+1, n
S4      x = a(i,j)
        do k=1, i-1
S5        x = x - a(j,k)*a(i,k)
S6      a(j,i) = x*p(i)
```



Transformation Catalogue [1/2]

Syntax	Effect
$\textsc{Unimodular}(P,\mathrm{U})$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S, \mathrm{A}^S \leftarrow \mathrm{U}.\mathrm{A}^S;\ \Gamma^S \leftarrow \mathrm{U}.\Gamma^S$
$\textsc{Shift}(P,\mathrm{M})$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S, \Gamma^S \leftarrow \Gamma^S + \mathrm{M}$
$\textsc{CutDom}(P,c)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S, \Lambda^S \leftarrow \mathsf{AddRow}\big(\Lambda^S, 0, c/\gcd(c_1,\ldots,c_{d^S+d^S_{\text{lv}}+d_{\text{gp}}+1})\big)$
$\textsc{Extend}(P,\ell,c)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S, \begin{cases} d^S \leftarrow d^S+1;\ \Lambda^S \leftarrow \mathsf{AddCol}(\Lambda^S,c,0); \\ \beta^S \leftarrow \mathsf{AddRow}(\beta^S,\ell,0); \\ \mathrm{A}^S \leftarrow \mathsf{AddRow}(\mathsf{AddCol}(\mathrm{A}^S,c,0),\ell,\mathbf{1}_\ell); \\ \Gamma^S \leftarrow \mathsf{AddRow}(\Gamma^S,\ell,0); \\ \forall(\mathtt{A},\mathrm{F}) \in \mathcal{L}^S_{\text{hs}} \cup \mathcal{R}^S_{\text{hs}}, \mathrm{F} \leftarrow \mathsf{AddRow}(\mathrm{F},\ell,0) \end{cases}$
$\textsc{AddLocalVar}(P)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S, d^S_{\text{lv}} \leftarrow d^S_{\text{lv}}+1;\ \Lambda^S \leftarrow \mathsf{AddCol}(\Lambda^S, d^S+1, 0);$ $\forall(\mathtt{A},\mathrm{F}) \in \mathcal{L}^S_{\text{hs}} \cup \mathcal{R}^S_{\text{hs}}, \mathrm{F} \leftarrow \mathsf{AddCol}(\mathrm{F}, d^S+1, 0)$
$\textsc{Privatize}(\mathtt{A},\ell)$	$\forall S \in \mathcal{S}_{\text{cop}}, \forall(\mathtt{A},\mathrm{F}) \in \mathcal{L}^S_{\text{hs}} \cup \mathcal{R}^S_{\text{hs}}, \mathrm{F} \leftarrow \mathsf{AddRow}(\mathrm{F},\ell,\mathbf{1}_\ell)$
$\textsc{Contract}(\mathtt{A},\ell)$	$\forall S \in \mathcal{S}_{\text{cop}}, \forall(\mathtt{A},\mathrm{F}) \in \mathcal{L}^S_{\text{hs}} \cup \mathcal{R}^S_{\text{hs}}, \mathrm{F} \leftarrow \mathsf{RemRow}(\mathrm{F},\ell)$
$\textsc{Fusion}(P,o)$	$b = \max\{\beta^S_{\dim(P)+1} \mid (P,o) \sqsubseteq \beta^S\}+1$ $\mathsf{Move}((P,o+1),(P,o+1),b);\ \mathsf{Move}(P,(P,o+1),-1)$
$\textsc{Fission}(P,o,b)$	$\mathsf{Move}(P,(P,o,b),1);\ \mathsf{Move}((P,o+1),(P,o+1),-b)$
$\textsc{Motion}(P,T)$	$\text{if } \dim(P)+1 = \dim(T) \text{ then } b = \max\{\beta^S_{\dim(P)} \mid P \sqsubseteq \beta^S\}+1 \text{ else } b=1$ $\mathsf{Move}(\mathrm{pfx}(T,\dim(T)-1),T,b)$ $\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S, \beta^S \leftarrow \beta^S + T - \mathrm{pfx}(P,\dim(T))$ $\mathsf{Move}(P,P,-1)$

Transformation Catalogue [2/2]

Syntax	Effect	Comments
$\textsc{Interchange}(P,o)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S$, $\begin{cases} \mathrm{U} = \mathrm{I}_{d^S} - \mathbf{1}_{o,o} - \mathbf{1}_{o+1,o+1} + \mathbf{1}_{o,o+1} + \mathbf{1}_{o+1,o}; \\ \textsc{Unimodular}(\beta^S, \mathrm{U}) \end{cases}$	swap rows o and $o+1$
$\textsc{Skew}(P,\ell,c,s)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S$, $\begin{cases} \mathrm{U} = \mathrm{I}_{d^S} + s \cdot \mathbf{1}_{\ell,c}; \\ \textsc{Unimodular}(\beta^S, \mathrm{U}) \end{cases}$	add the skew factor
$\textsc{Reverse}(P,o)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S$, $\begin{cases} \mathrm{U} = \mathrm{I}_{d^S} - 2 \cdot \mathbf{1}_{o,o}; \\ \textsc{Unimodular}(\beta^S, \mathrm{U}) \end{cases}$	put a -1 in (o,o)
$\textsc{StripMine}(P,k)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid P \sqsubseteq \beta^S$, $\begin{cases} c = \dim(P); \\ \textsc{Extend}(\beta^S, c, c); \\ u = d^S + d^S_{\text{lv}} + d_{\text{gp}} + 1; \\ \textsc{CutDom}(\beta^S, -k \cdot \mathbf{1}_c + (\mathrm{A}^S_{c+1}, \Gamma^S_{c+1})); \\ \textsc{CutDom}(\beta^S, k \cdot \mathbf{1}_c - (\mathrm{A}^S_{c+1}, \Gamma^S_{c+1}) + (k-1)\mathbf{1}_u) \end{cases}$	insert intermediate loop constant column $k \cdot \mathbf{i}_c \leq \mathbf{i}_{c+1}$ $\mathbf{i}_{c+1} \leq k \cdot \mathbf{i}_c + k - 1$
$\textsc{Tile}(P,o,k_1,k_2)$	$\forall S \in \mathcal{S}_{\text{cop}} \mid (P,o) \sqsubseteq \beta^S$, $\begin{cases} \textsc{StripMine}((P,o),k_2); \\ \textsc{StripMine}(P,k_1); \\ \textsc{Interchange}((P,0),\dim(P)) \end{cases}$	strip outer loop strip inner loop interchange

Some Final Observations

Some classical pitfalls

- The number of rows of Θ does not correspond to actual parallel levels
- Scalar rows vs. linear rows
- Linear independence
- Parametric shift for domains without parametric bound
- ...

Data Dependences

Purpose of Dependence Analysis

- Not all program transformations preserve the semantics
- Semantics is preserved if the dependence are preserved

- In standard frameworks, it usually means reordering statements/loops
  - Statements containing dependent references should not be executed in a different order
  - Granularity: usually a reference to an **array**

- In the polyhedral framework, it means reordering statement **instances**
  - Statement instances containing dependent references should not be executed in a different order
  - Granularity: a reference to an **array cell**

Example

Exercise: Compute the set of cells of A accessed

Example

```
for (i = 0; i < N; ++i)
  for (j = i; j < N; ++j)
    A[2i + 3][4j] = i * j;
```

- $\mathcal{D}_S$: $\{i,j \mid 0 \leq i < N,\ i \leq j < N\}$
- Function: $f_A : \{2i+3, 4j \mid i,j \in \mathbb{Z}\}$
- $Image(f_A, \mathcal{D}_S)$ is the set of cells of A accessed (a $\mathcal{Z}$-polyhedron):
  - Polyhedron: $Q : \{i,j \mid 3 \leq i < 2N+2,\ 0 \leq j < 4N\}$
  - Lattice: $L : \{2i+3, 4j \mid i,j \in \mathbb{Z}\}$

Data Dependence

Definition (Bernstein conditions)

Given two references, there exists a dependence between them if the three following conditions hold:

- they reference the same array (cell)
- one of this access is a write
- the two associated statements are executed

Three categories of dependences:

- RAW (Read-After-Write, aka flow): first reference writes, second reads
- WAR (Write-After-Read, aka anti): first reference reads, second writes
- WAW (Write-After-Write, aka output): both references writes

Another kind: RAR (Read-After-Read dependences), used for locality/reuse computations

Some Terminology on Dependence Relations

We categorize the dependence relation in three kinds:

- **Uniform dependences:** the distance between two dependent iterations is a constant
  - ex: $i \rightarrow i+1$
  - ex: $i,j \rightarrow i+1,j+1$
- **Non-uniform dependences:** the distance between two dependent iterations varies during the execution
  - ex: $i \rightarrow i+j$
  - ex: $i \rightarrow 2i$
- **Parametric dependences:** at least a parameter is involved in the dependence relation
  - ex: $i \rightarrow i+N$
  - ex: $i+N \rightarrow j+M$

Dependence Polyhedron [1/3]

Principle: model all **pairs** of instances in dependence

Definition (Dependence of statement instances)

A statement S depends on a statement R (written $R \rightarrow S$) if there exists an operation $S(\vec{x}_S)$ and $R(\vec{x}_R)$ and a memory location m such that:

1. $S(\vec{x}_S)$ and $R(\vec{x}_R)$ refer to the same memory location m, and at least one of them writes to that location,
2. x_S and x_R belongs to the iteration domain of R and S,
3. in the original sequential order, $S(\vec{x}_S)$ is executed before $R(\vec{x}_R)$.

Dependence Polyhedron [2/3]

1. *Same memory location*: equality of the subscript functions of a pair of references to the same array: $F_S\vec{x}_S + a_S = F_R\vec{x}_R + a_R$.

2. *Iteration domains*: both S and R iteration domains can be described using affine inequalities, respectively $A_S\vec{x}_S + c_S \geq 0$ and $A_R\vec{x}_R + c_R \geq 0$.

3. *Precedence order*: each case corresponds to a common loop depth, and is called a *dependence level*.

   For each dependence level l, the precedence constraints are the equality of the loop index variables at depth lesser to l: $x_{R,i} = x_{S,i}$ for $i < l$ and $x_{R,l} > x_{S,l}$ if l is less than the common nesting loop level. Otherwise, there is no additional constraint and the dependence only exists if S is textually before R.

   Such constraints can be written using linear inequalities: $P_{l,S}\vec{x}_S - P_{l,R}\vec{x}_R + b \geq 0$.

Dependence Polyhedron [3/3]

The dependence polyhedron for $R \rightarrow S$ at a given level l and for a given pair of references f_R, f_S is described as [Feautrier/Bastoul]:

$$\mathcal{D}_{R,S,f_R,f_S,l} : D\begin{pmatrix}\vec{x}_S \\ \vec{x}_R\end{pmatrix} + d = \left[\begin{array}{cc} F_S & -F_R \\ \hline A_S & 0 \\ 0 & A_R \\ PS & -P_R \end{array}\right]\begin{pmatrix}\vec{x}_S \\ \vec{x}_R\end{pmatrix} + \begin{pmatrix} a_S - a_R \\ c_S \\ c_R \\ b \end{pmatrix} \begin{array}{c} = 0 \\ \hline \geq \vec{0} \end{array}$$

A few properties:

- We can always build the dep polyhedron for a given pair of affine array accesses (it is convex)
- It is exact, if the iteration domain and the access functions are also exact
- it is over-approximated if the iteration domain or the access function is an approximation

Polyhedral Dependence Graph

Definition (Polyhedral Dependence Graph)

The Polyhedral Dependence Graph is a multi-graph with one vertex per syntactic program statement S_i, with edges $S_i \rightarrow S_j$ for each dependence polyhedron $\mathcal{D}_{S_i,S_j}$.

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

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

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:

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

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

The Farkas Lemma: Example

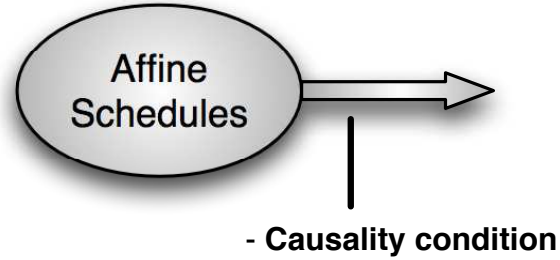
- Function: $f(x) = ax + b$
- Domain of x: $\{1 \leq x \leq 3\} \rightarrow x - 1 \geq 0, \quad -x + 3 \geq 0$
- Farkas lemma: $f(x) \geq 0 \Leftrightarrow f(x) = \lambda_0 + \lambda_1(x-1) + \lambda_2(-x+3)$

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)

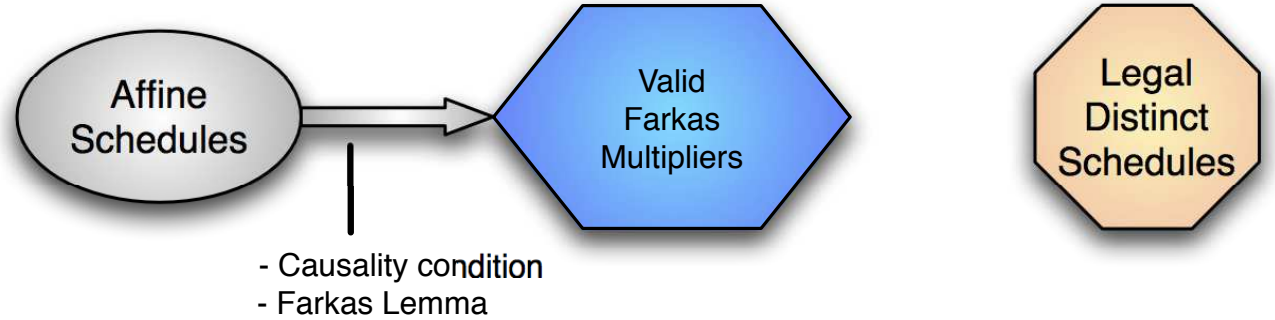
Lemma (Affine form of Farkas lemma)

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:

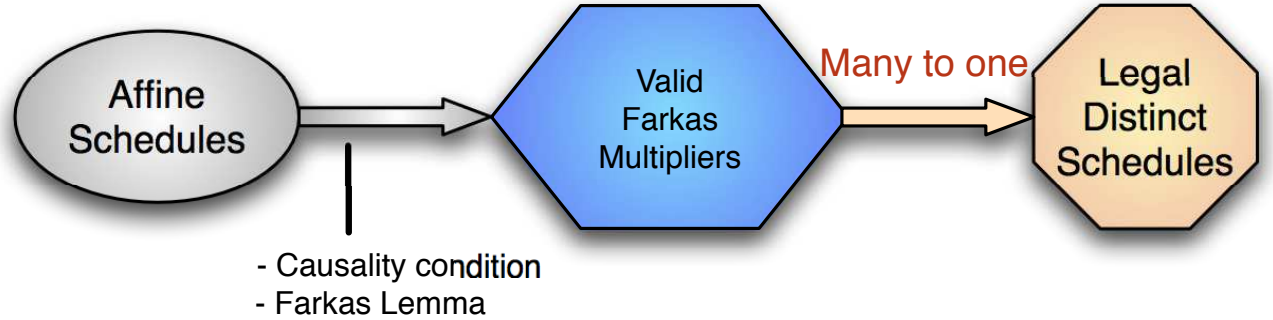
$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T(A\vec{x}+\vec{b}),\ \textit{with}\ \lambda_0 \geq 0\ \textit{and}\ \vec{\lambda} \geq \vec{0}.$$

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

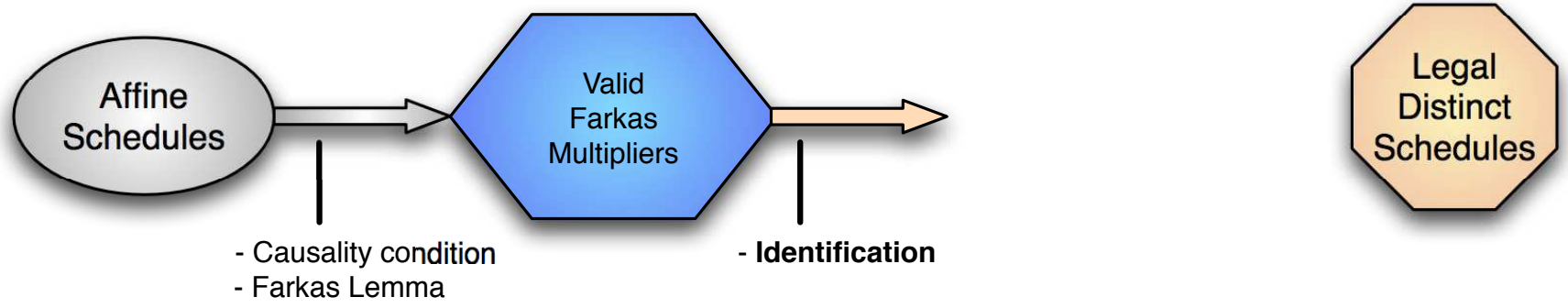
Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



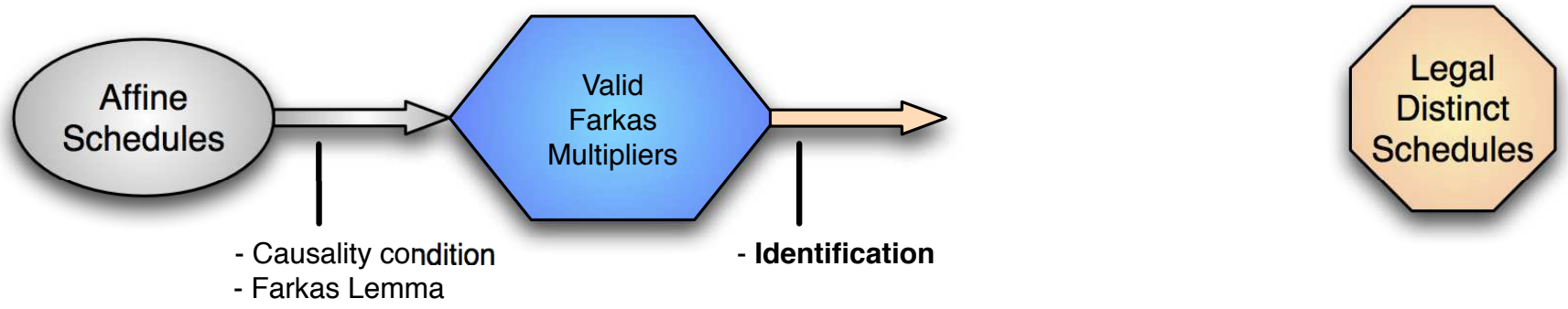
Example: Semantics Preservation (1-D)



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

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

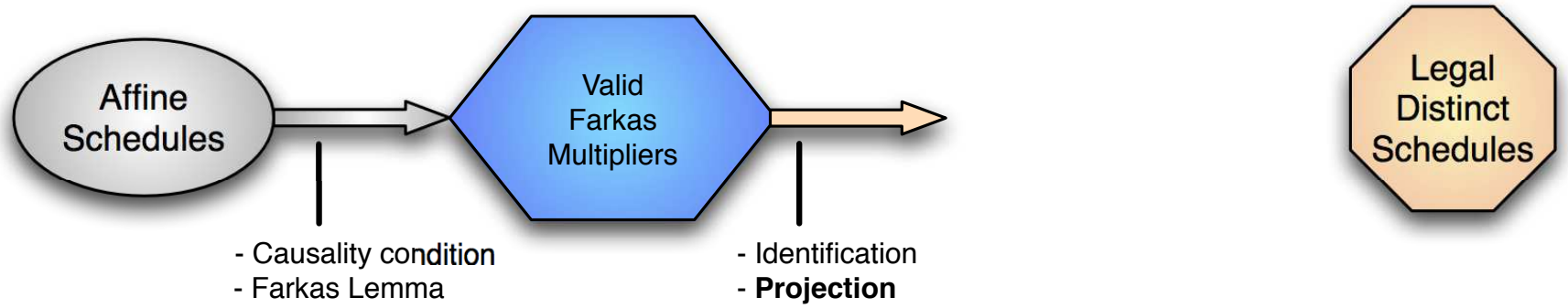
Example: Semantics Preservation (1-D)



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

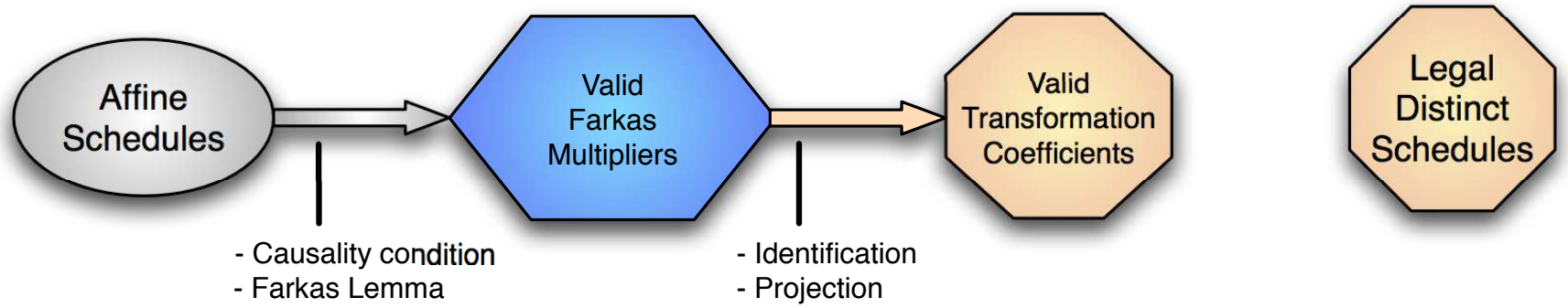
$$\left\{ \begin{array}{llrcl} D_{R\delta S} & \mathbf{i_R} & : & -t_{1_R} & = & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ & \mathbf{i_S} & : & t_{1_S} & = & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ & \mathbf{j_S} & : & t_{2_S}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

Example: Semantics Preservation (1-D)

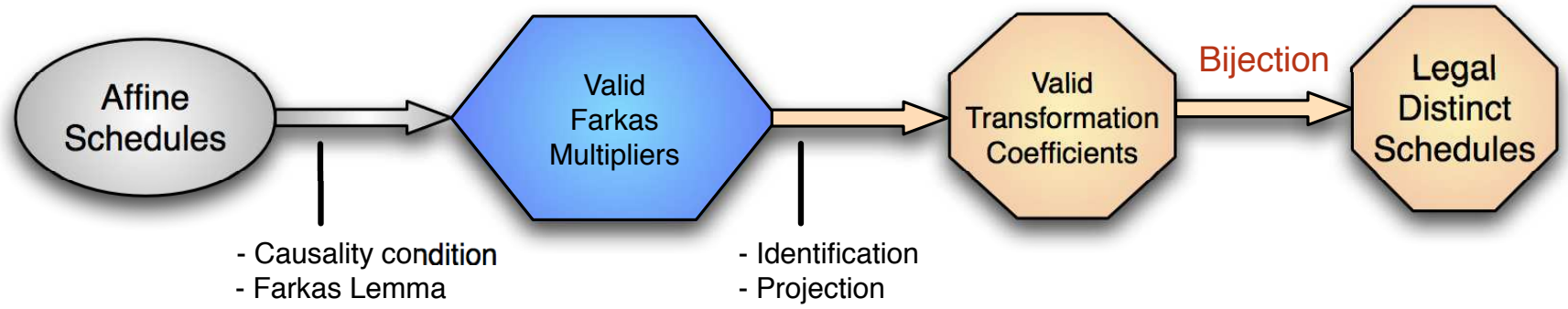


- Solve the constraint system
- Use (purpose-optimized) Fourier-Motzkin projection algorithm
  - Reduce redundancy
  - Detect implicit equalities

Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



- One point in the space ⇔ 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

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

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 $\Theta^R, \Theta^S \ldots$ of dimension m, the program semantics is preserved if:

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

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:

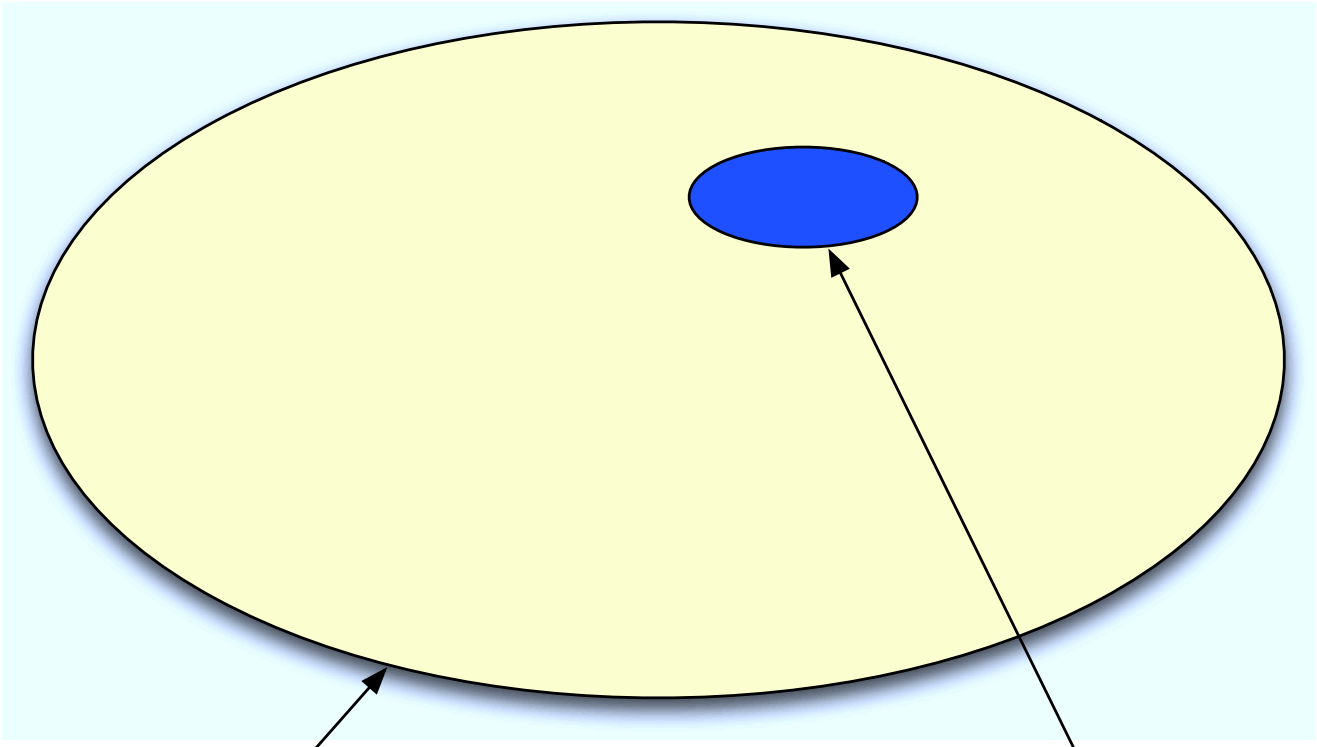
- Bound the schedule latency with a lower bound which does not prevent to find all solutions
- Intuitively:
  - $\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \geq \delta$ if the dependence has not been strongly satisfied
  - $\Theta^S(\vec{x}_S) - \Theta^R(\vec{x}_R) \geq -\infty$ if it has

Lemma (Schedule lower bound)

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

$$\max\left(\Theta_k^S(\vec{x}_S) - \Theta_k^R(\vec{x}_R)\right) > -K.\vec{n} - K$$

Space of Semantics-Preserving Affine Schedules



All unique bounded
affine multidimensional schedules

All unique semantics-preserving
affine multidimensional 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 $\Theta^R, \Theta^S \ldots$ 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) $\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\ldots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$

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

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

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

→ 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 <u>lexicographic</u> 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]

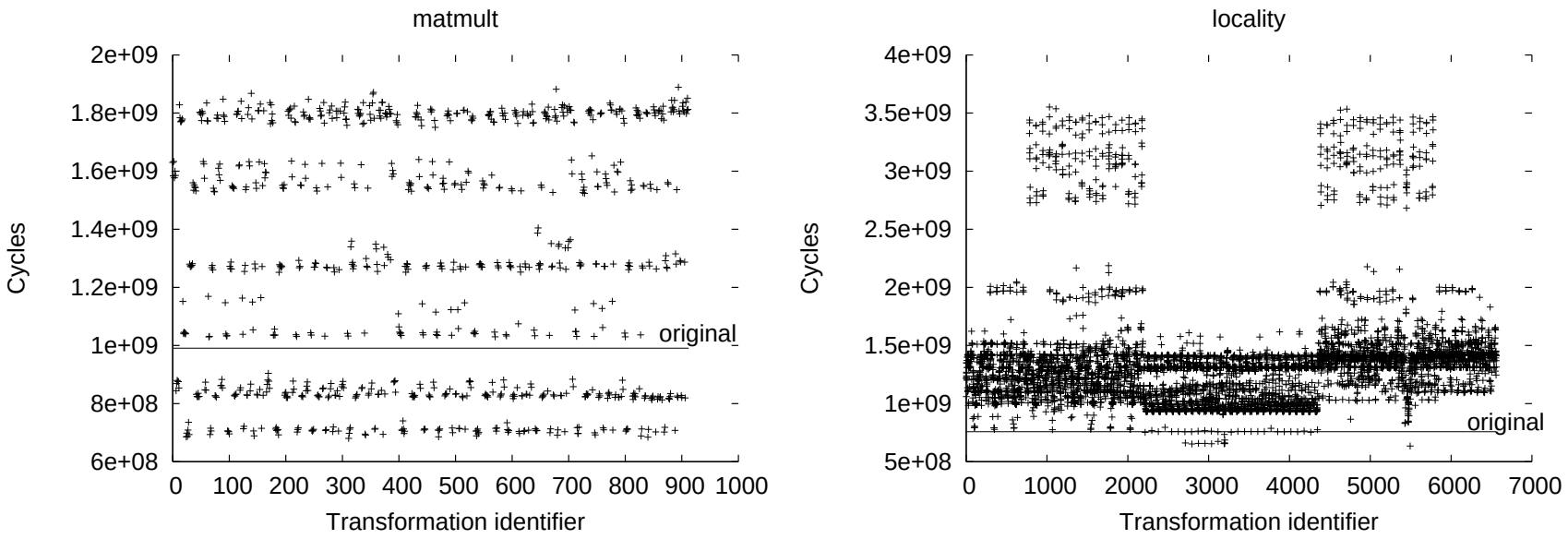
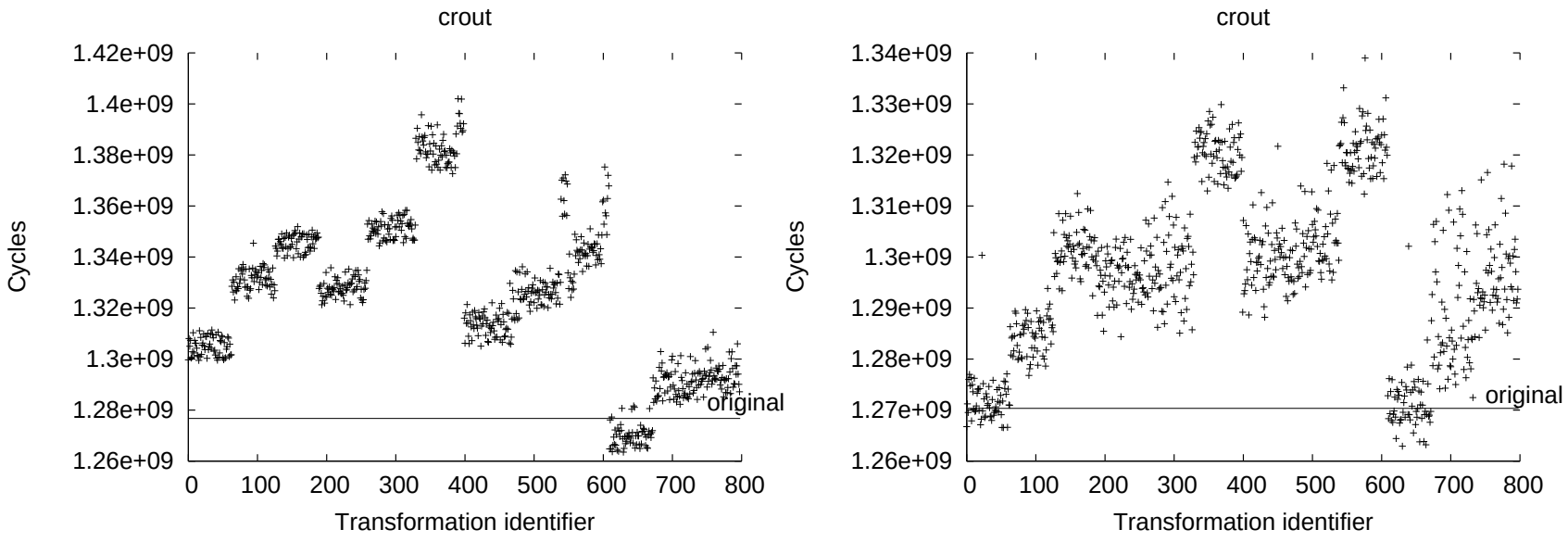


Figure : Performance distribution for `matmult` and `locality`

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



(a) GCC -O3

(b) ICC -fast

Figure : The effect of the compiler

Quantitative Analysis: The Hypothesis

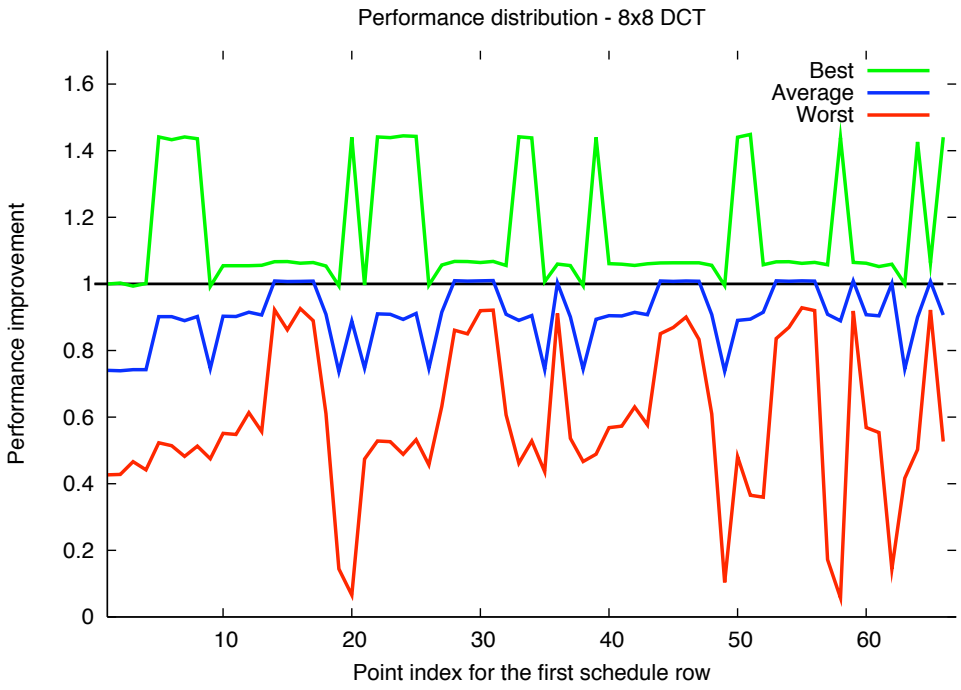
Extremely large generated spaces: $> 10^{50}$ points

$\rightarrow$ 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**

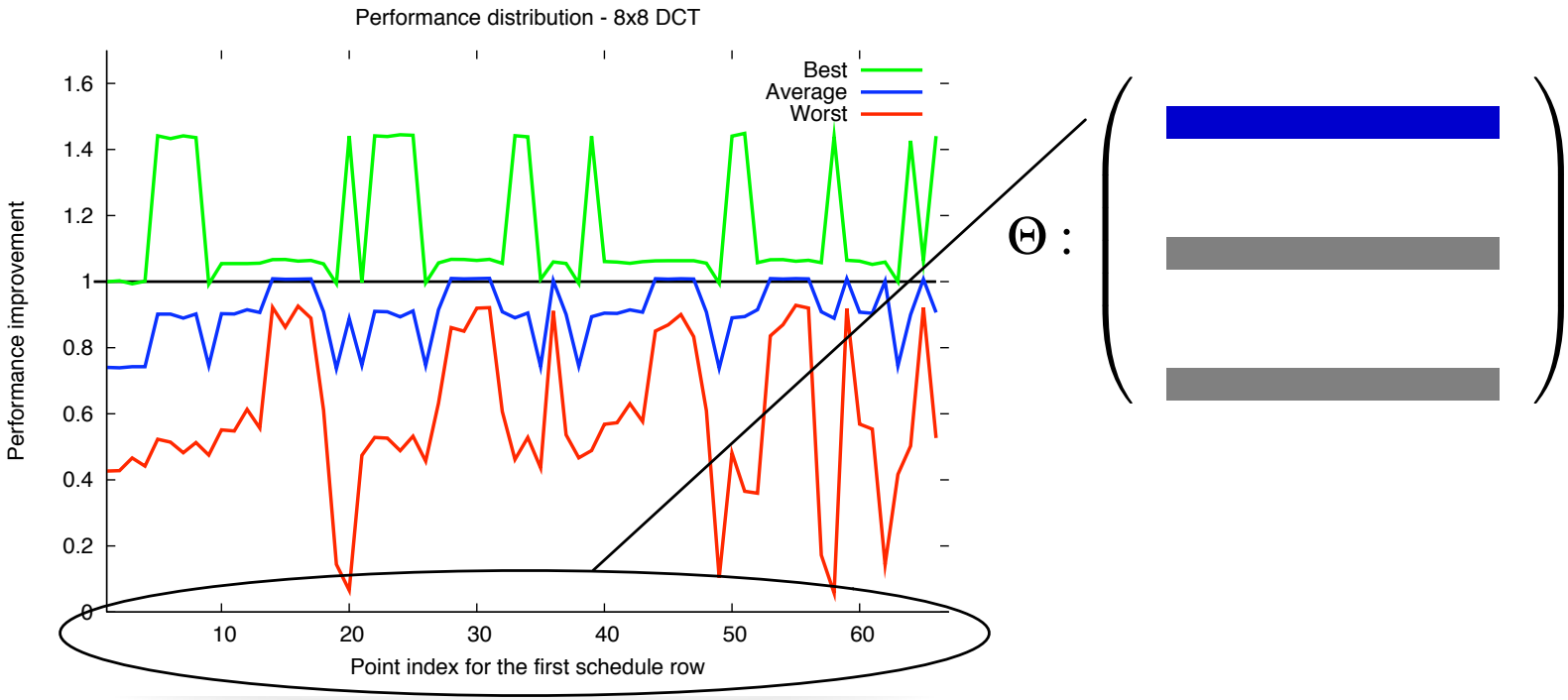
Observations on the Performance Distribution




```
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);
 }
```

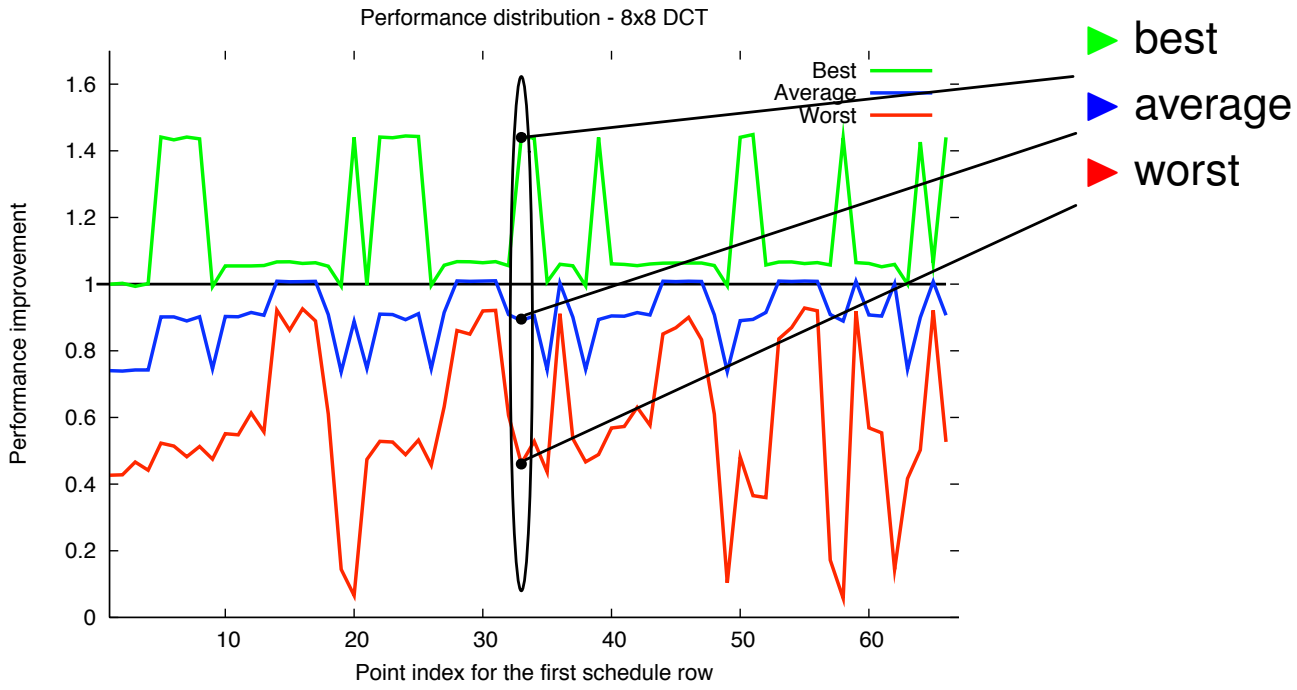
- Extensive study of 8x8 Discrete Cosine Transform (UTDSP)
- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions

Observations on the Performance Distribution



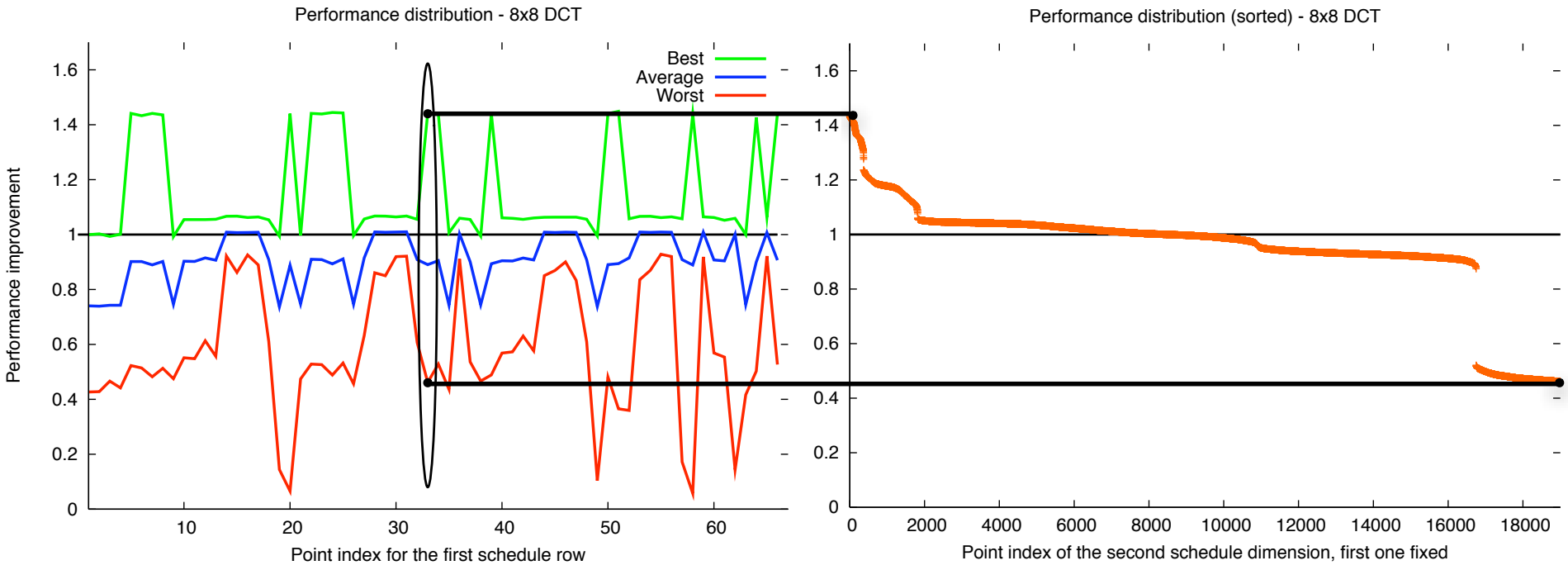
- Extensive study of 8x8 Discrete Cosine Transform (UTDSP)
- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions

Observations on the Performance Distribution



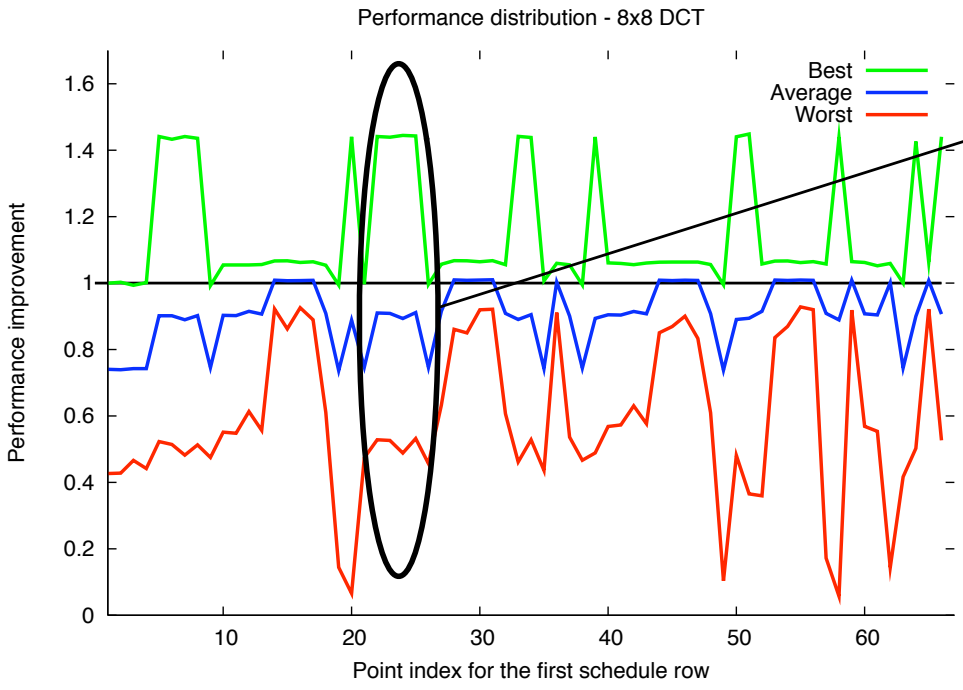
- Take one specific value for the first row
- Try the 19863 possible values for the second row

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: $< 0.02\%$

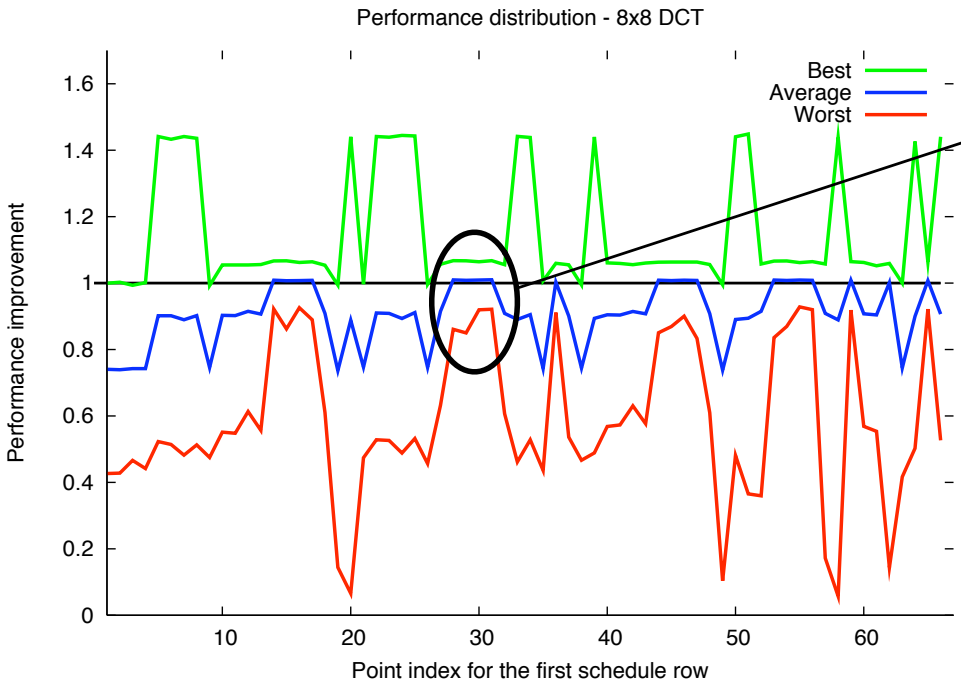
Observations on the Performance Distribution



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)

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

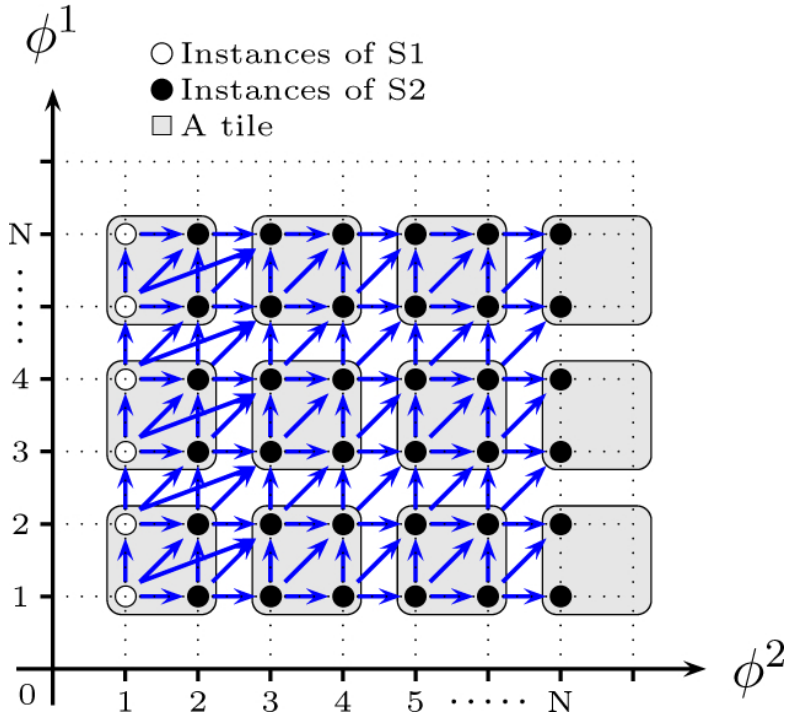
Key Observations

- 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

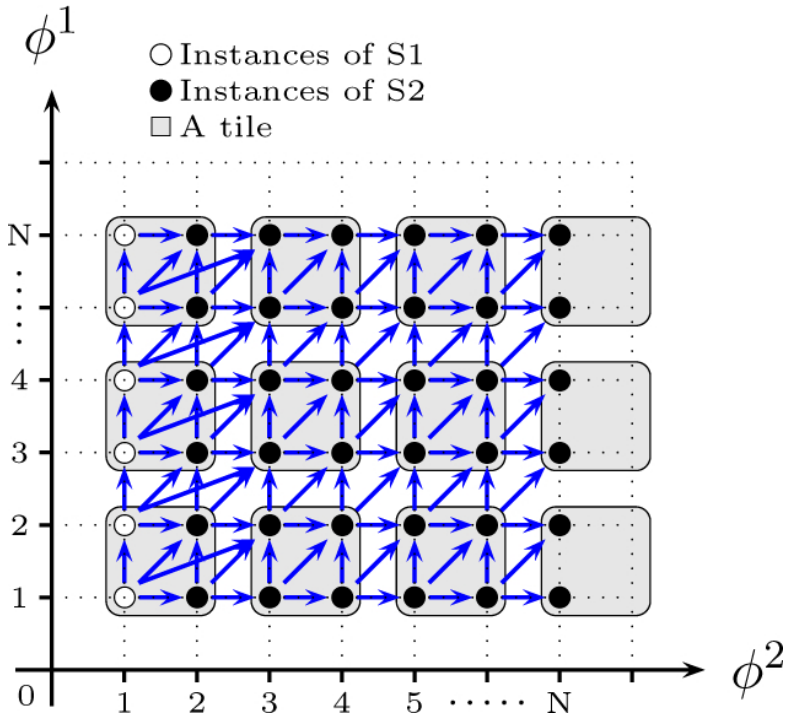


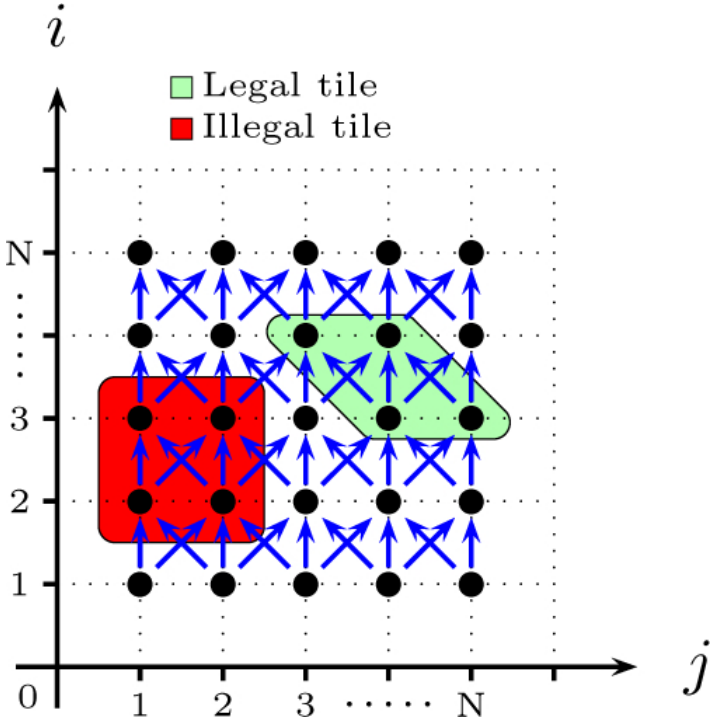
An Overview of Tiling

- 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_k^R, Θ_k^S two one-dimensional schedules. They are valid tiling hyperplanes if

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

- 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.

Special treatment when no permutable band can be found: splitting

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)

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

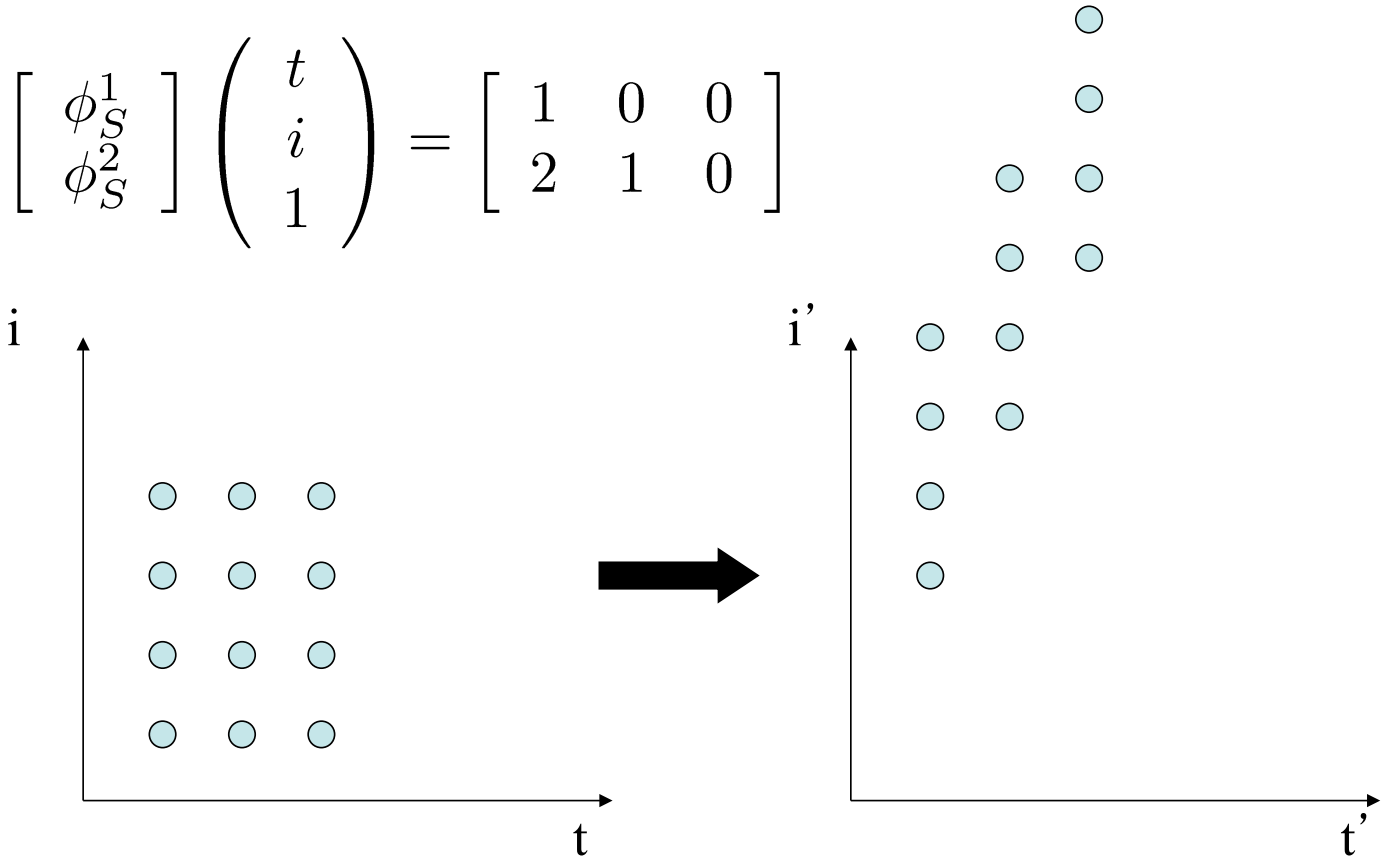
$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

- 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

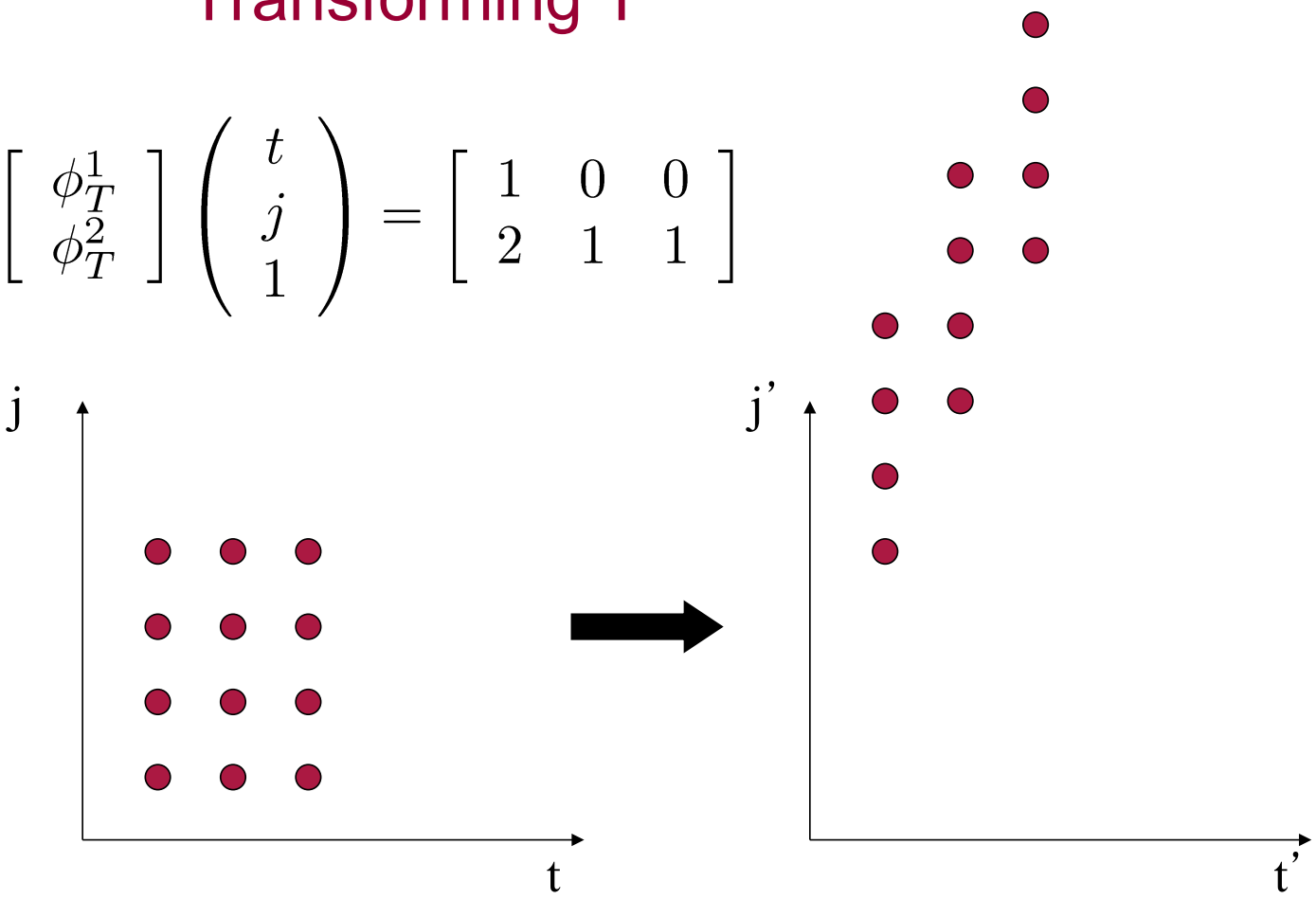
$$\begin{bmatrix} \phi_S^1 \\ \phi_S^2 \end{bmatrix} \begin{pmatrix} t \\ i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$



Example: 1D-Jacobi

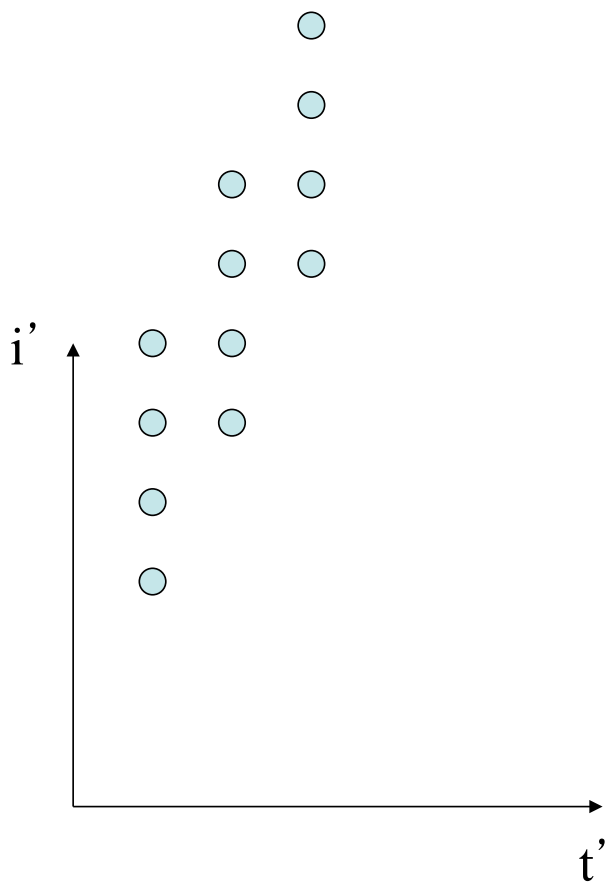
Transforming T

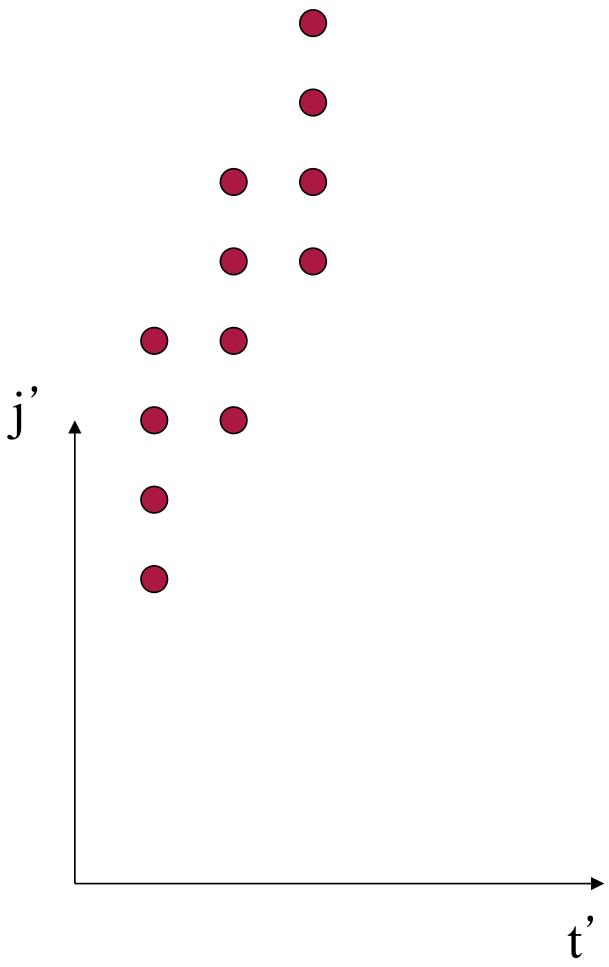
$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$



Example: 1D-Jacobi

Interleaving S and T



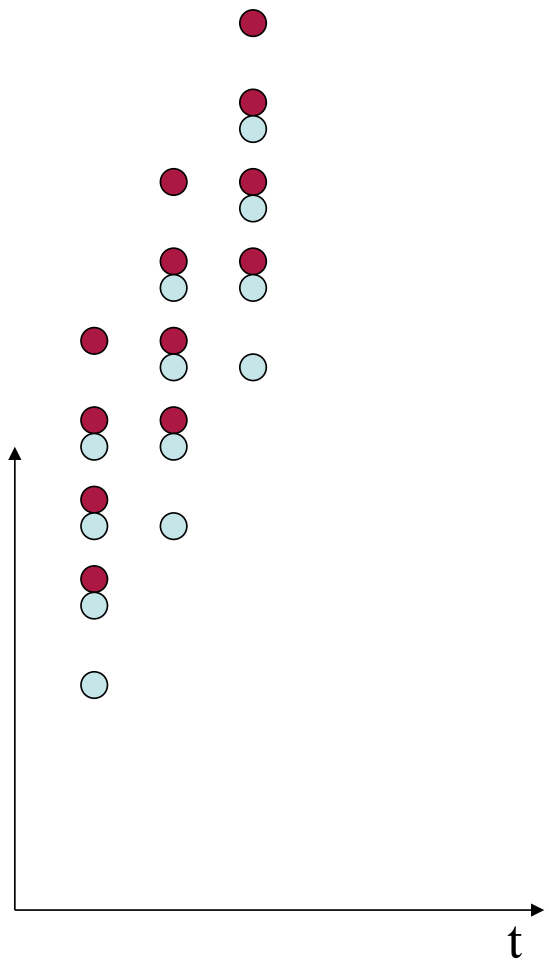


Example: 1D-Jacobi

Interleaving S and T

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

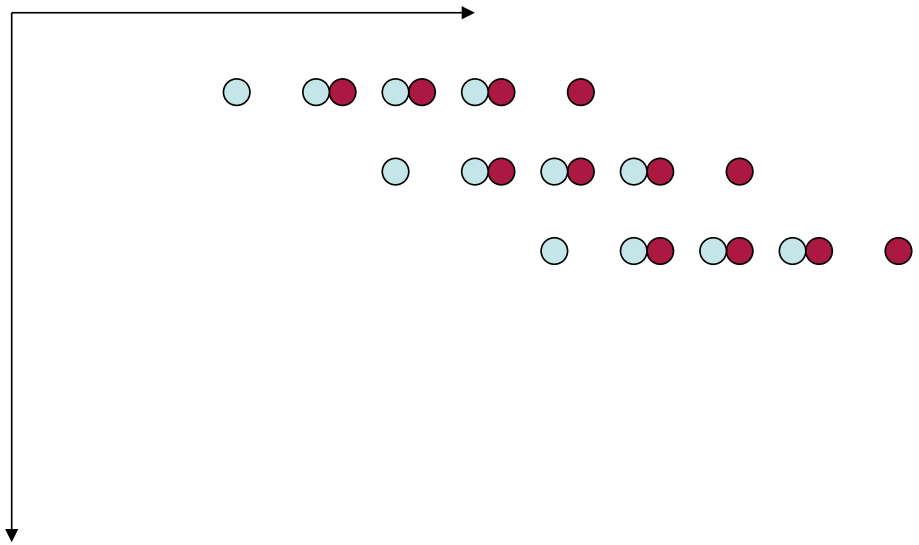
$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$



Example: 1D-Jacobi

1-D Jacobi (imperfectly nested) – transformed code

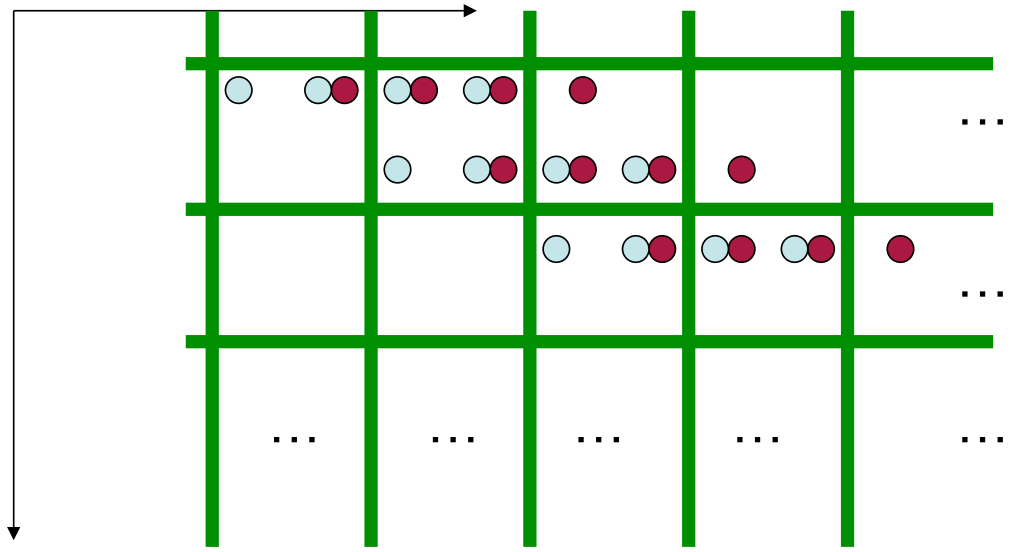
```
    for (t0=0;t0<=M-1;t0++) {
S':   b[2]=0.333*(a[2-1]+a[2]+a[2+1]);
      for (t1=2*t0+3;t1<=2*t0+N-2;t1++) {
S:      b[-2*t0+t1]=0.333*(a[-2*t0+t1-1]+a[-2*t0+t1]
                           +a[-2*t0+t1+1]);
T:      a[-2*t0+t1-1]=b[-2*t0+t1-1]; }
T':   a[N-2]=b[N-2]; }
```



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×			
4× Opteron 8380 / ICC 11	1×			

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×	2.4×		
4× Opteron 8380 / ICC 11	1×	2.2×		

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×	2.4×	3.9×	3.1×
4× Opteron 8380 / ICC 11	1×	2.2×	6.1×	8.3×

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

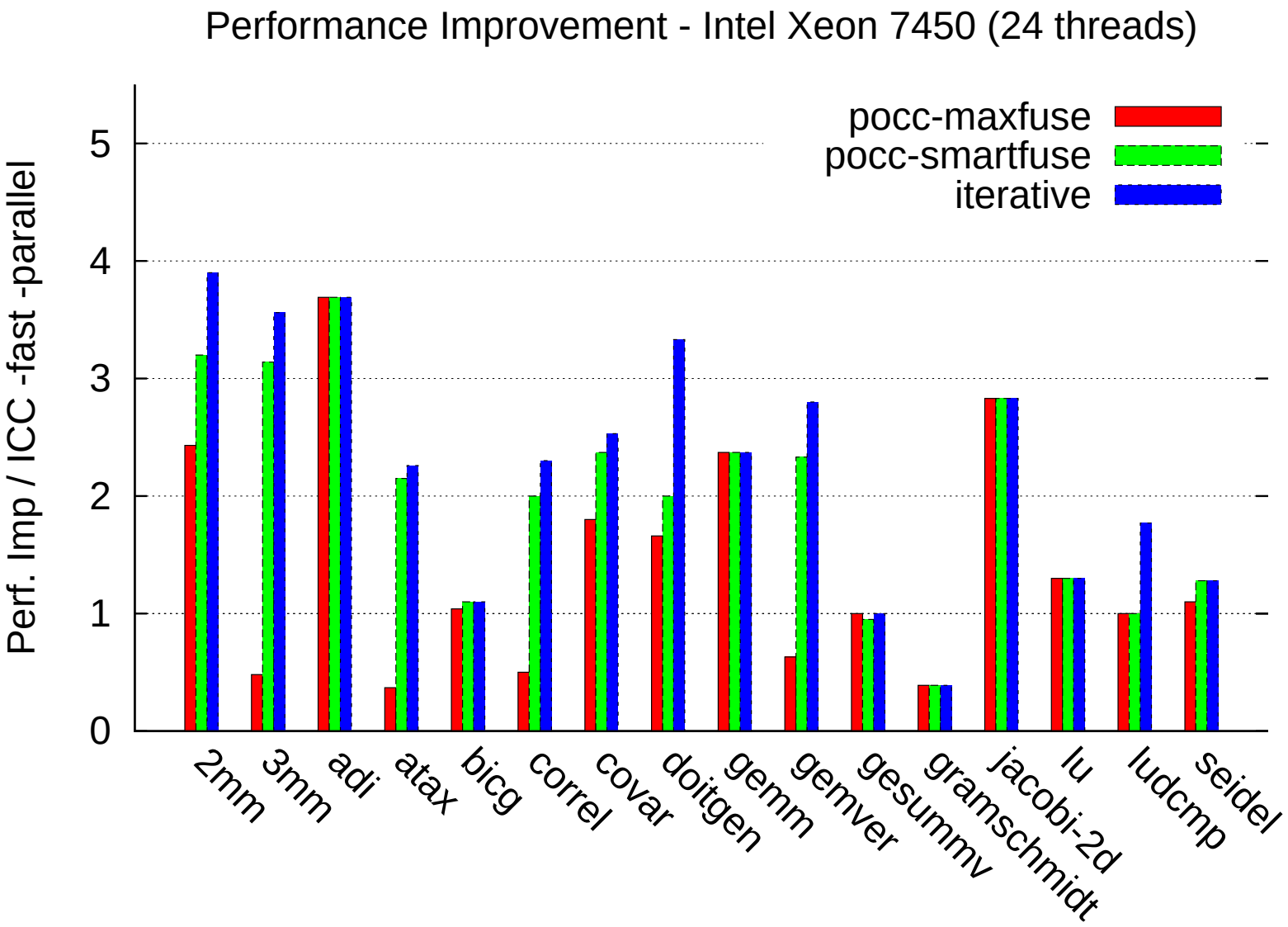
Table : Summary of the optimization process

Experimental Setup

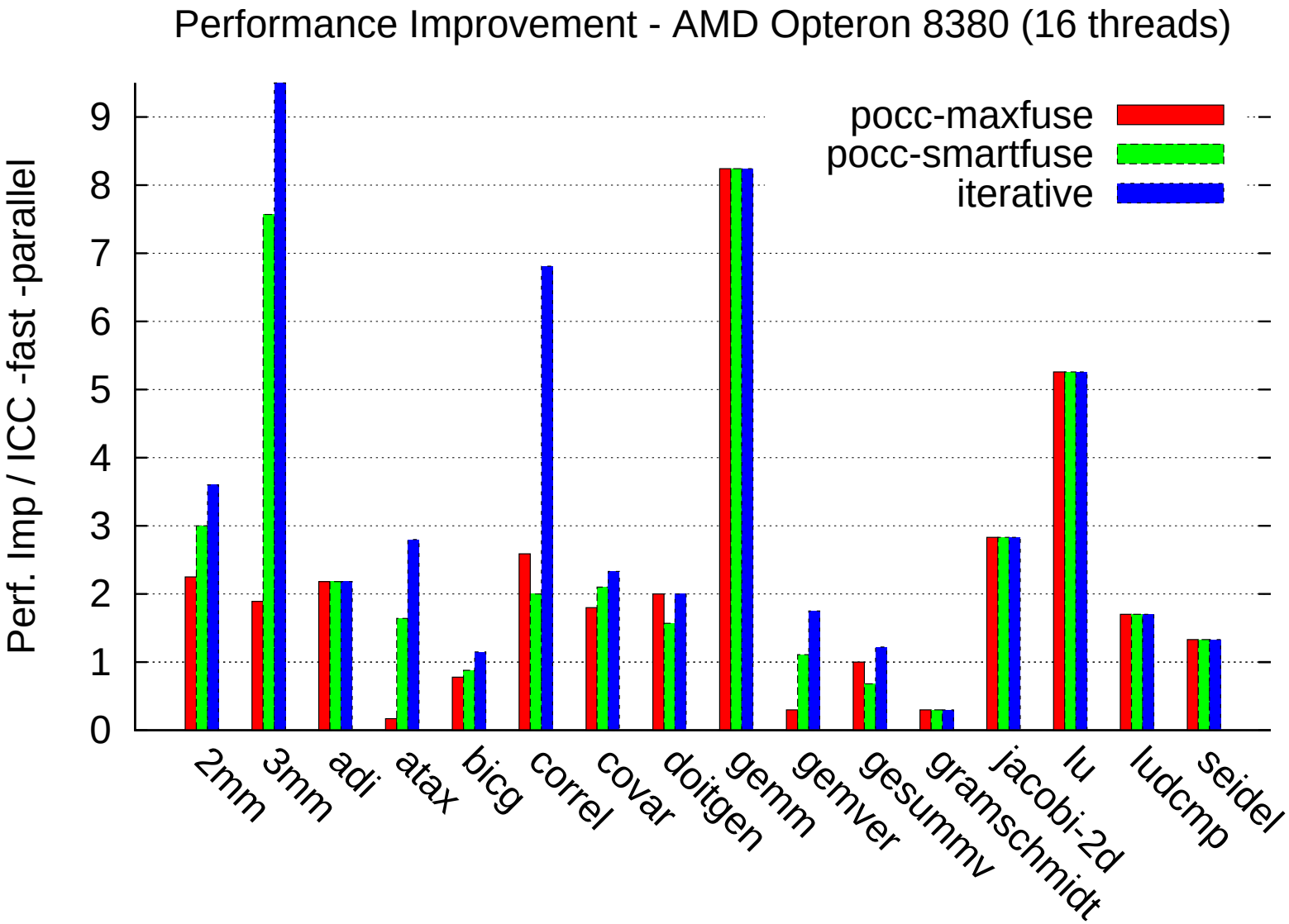
We compare three schemes:

- **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

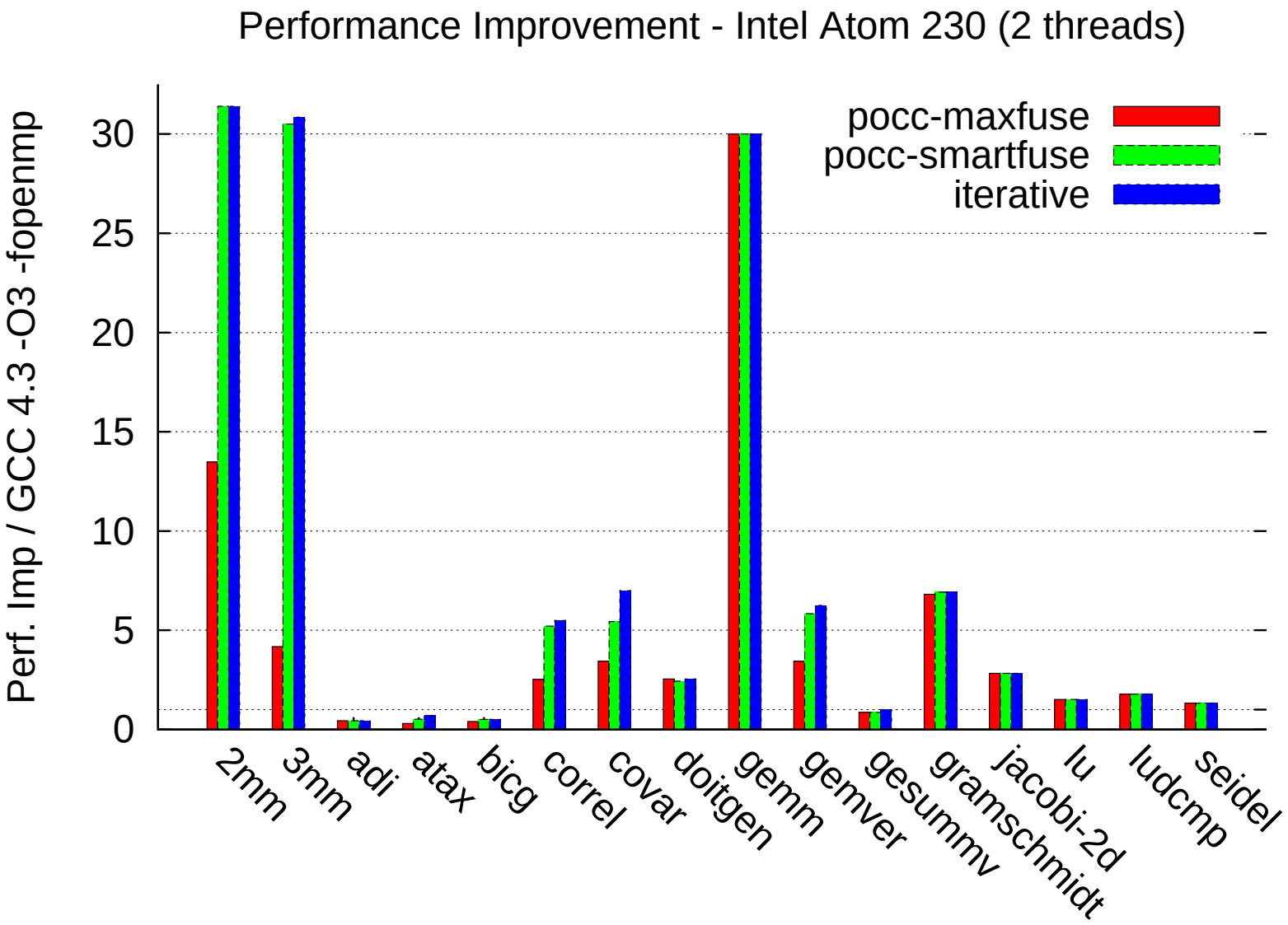
Performance Results - Intel Xeon 7450 - ICC 11



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 $2\times$)

Approximate Scheduling

Exact vs. Approximate Dependences

- So far, we used exact dependence representation
- But numerous other works use approximate representation:
  - dependence distance vector [Allen/Kenney,KMW]
  - dependence direction
  - DDV approximated as polyhedra [Darte-Vivien]

- Optimality may not always be lost when using a relaxed dependence representation!

Key Aspects of Dependence Representations

- Can use a graph to represent the dependences
  - Karp, Miller, Winograd: graph for SURE, using DDV
  - Darte-Vivien: "PDG"-like graph
- For certain kind dependences (i.e., uniform) "optimal" algorithm
  - Darte-Vivien
  - SURE

Some Observations

- Maximal parallelism (under some conditions) can be extracted from less precise representations
- Very fast scheduling algorithms can be derived
- Remaining question: how useful are those algorithms?

Conclusion

The End

Let's stop here for today!