The Polyhedral Compilation Framework

Louis-Noël Pouchet

Dept. of Computer Science and Engineering
Ohio State University
pouchet@cse.ohio-state.edu

November 18, 2011

Overview of Today's Lecture

Topic: automatic optimization of applications

What this lecture is about:

- The main ideas behind polyhedral high-level loop transformations
- A (partial) review of the state-of-the-art in polyhedral compilation
  - Optimization algorithms
  - Available software

And what is is NOT about:

- In-depth compilation algorithms
- Low-level optimizations (e.g., machine-specific code)

Compilers

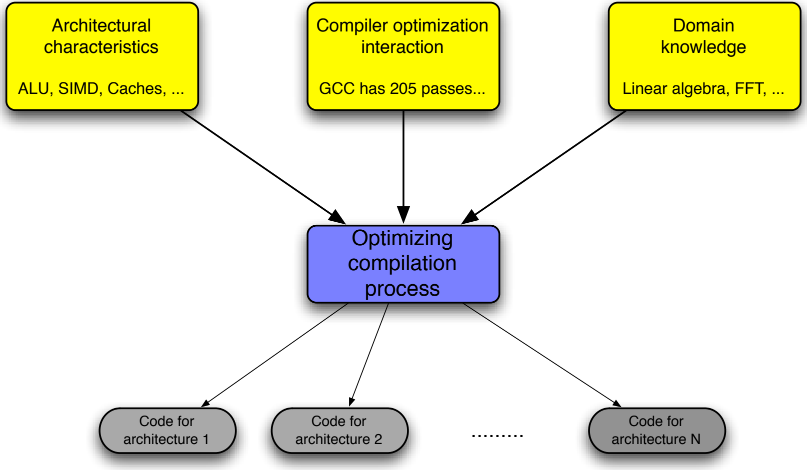
Compilers translate a human-readable program into machine code

- Numerous input languages and paradigm (from ASM to Java)
  - Abstraction: a single high-level intermediate representation for programs
  - The compiler front-end translates any language to this IR
  - Adding a new language requires only extending the front-end
- Make the "most" of the available hardware
  - The compiler back-end translates into machine code
  - Specific optimizations for each supported architecture
  - Adding a new target requires only extending the back-end
- Be reusable: avoid redesign a new compiler for each new chip
  - Many optimizations are useful for numerous targets (parallelization, vectorization, data cache, ...)
  - The compiler middle-end maps a program to an (abstract) model of computation

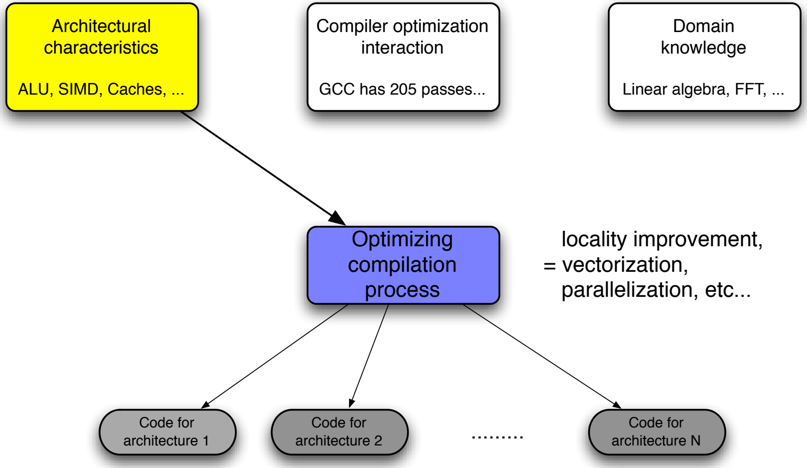
Compiler Middle-end

- Responsible for transforming the program in a form suitable for a better execution
  - Typical: remove dead code (DCE), mutualize common expressions computation (CSE)
  - More advanced: create SIMD-friendly loops, extract task parallelism
  - Experimental: algorithm recognition, equivalent program substitution, ...
- Composed of numerous passes, each responsible of a specific optimization

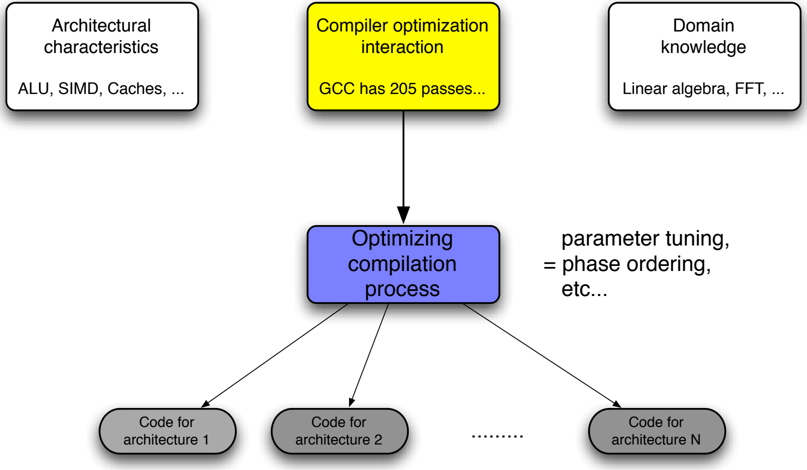
The Optimization Problem



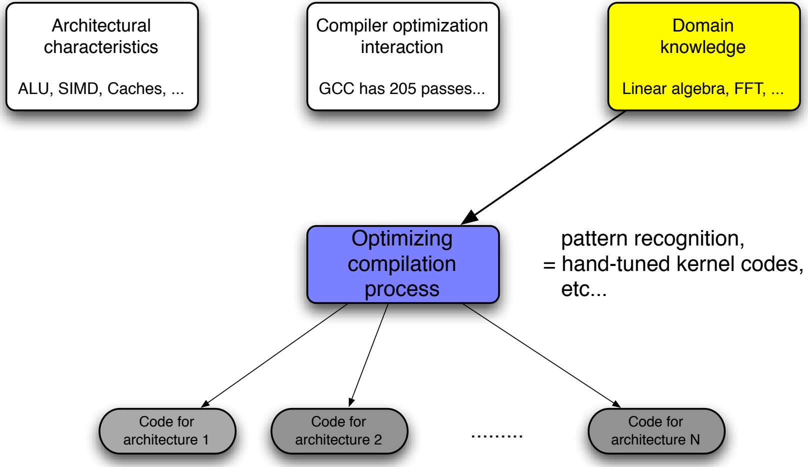
The Optimization Problem



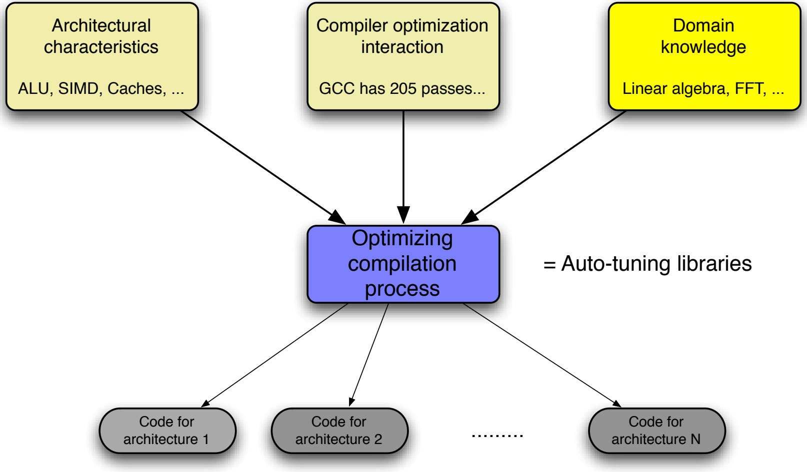
The Optimization Problem



The Optimization Problem



The Optimization Problem



Outline

High-Level Transformations

Running Example: matmult

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

Loop Transformations in Production Compilers

Limitations of standard syntactic frameworks:

- Composition of transformations may be tedious
  - composability rules / applicability
- Parametric loop bounds, impectly 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
  - Any 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

Compilation Flow

Affine transformation framework:

- Data dependence analysis
- Optimization
- Code generation

The Polyhedral Model

Polyhedral Program Optimization: a Three-Stage Process

1 Analysis: from code to model

$\longrightarrow$ Existing prototype tools

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

$\longrightarrow$ GCC GRAPHITE (now in mainstream), LLVM Polly (prototype)

$\longrightarrow$ Reservoir Labs R-Stream, IBM XL/Poly

Polyhedral Program Optimization: a Three-Stage Process

1 Analysis: from code to model

$\longrightarrow$ Existing prototype tools

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

$\longrightarrow$ GCC GRAPHITE (now in mainstream), LLVM Polly (prototype)

$\longrightarrow$ Reservoir Labs R-Stream, IBM XL/Poly

2 Transformation in the model

$\longrightarrow$ Build and select a program transformation

Polyhedral Program Optimization: a Three-Stage Process

1 Analysis: from code to model

→ Existing prototype tools

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

→ GCC GRAPHITE (now in mainstream), LLVM Polly (prototype)

→ 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 <= 2; ++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;
7: A[2][0] = 2 * 0;
8: A[2][1] = 2 * 1;
9: A[2][2] = 2 * 2;
```

Motivating Example [2/2]

A few observations:

- Statement is executed 9 times
- There is a different values for i, j associated to these 9 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 Program Representation

- Find a compact representation (critical)
- 1 point in the set $\leftrightarrow$ 1 executed instance (to allow optimization operations, such as counting points)
- Can retrieve when the instance is executed (total order on the set)
- Easy manipulation: scanning code must be re-generated
- Deal with parametric and infinite domains
- Non-unit loop strides
- Generalized affine conditionals (union of polyhedra)
- Data-dependent conditionals

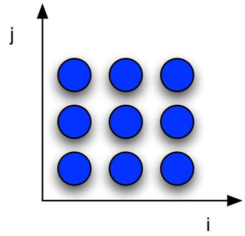
Returning to the Example

Example

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

Modeling the iteration domain:

- Polytope dimension: set by the number of surrounding loops
- Constraints: set by the loop bounds



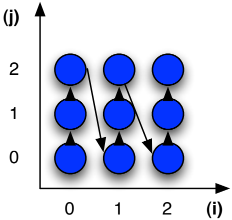
Returning to the Example

Example

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

Modeling the iteration domain:

- Polytope dimension: set by the number of surrounding loops
- Constraints: set by the loop bounds



Returning to the Example

Example

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

Modeling the iteration domain:

- Polytope dimension: set by the number of surrounding loops
- Constraints: set by the loop bounds

$$\mathcal{D}_R : \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} . \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & -1 & 2 \end{bmatrix} . \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \geq \vec{0}$$

$$0 \leq i \leq 2, \quad 0 \leq j \leq 2$$

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** [Clauss/Verdoolaege]
- Scan a polytope (code generation): **CLooG** [Quillere/Bastoul]
- Find the lexicographic minimum: **PIP** [Feautrier]

Polyhedral Representation of Programs

Static Control Parts

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

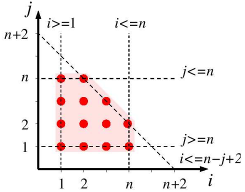
Polyhedral Representation of Programs

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] = ...
```

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



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];
}
```

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

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

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

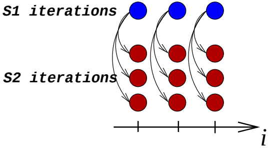
Polyhedral Representation of Programs

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;
}
```

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

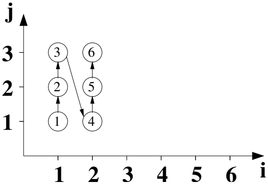


Program Transformations

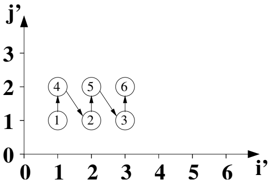
Scheduling Statement Instances

Interchange Transformation

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



$\Longrightarrow$



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

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

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

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

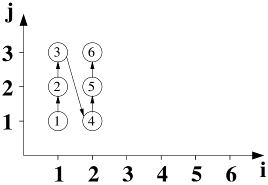
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

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

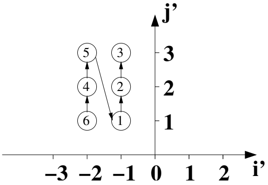
Scheduling Statement Instances

Reversal Transformation

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



$\Longrightarrow$



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

(a) original polyhedron

$A\vec{x} + \vec{a} \geq \vec{0}$

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

(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

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

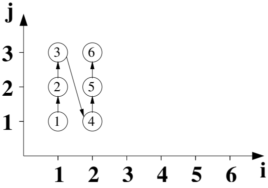
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

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

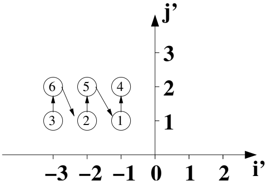
Scheduling Statement Instances

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



$\Longrightarrow$



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

(a) original polyhedron

$A\vec{x} + \vec{a} \geq \vec{0}$

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

(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

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

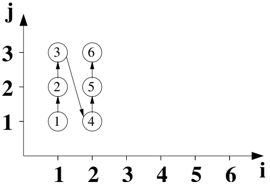
```
do i = 1, 2
  do j = 1, 3
     S(i,j)
```

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

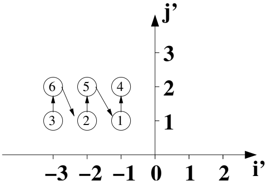
Scheduling Statement Instances

Coumpound Transformation

The transformation matrix is the composition of an interchange and reversal



$\Longrightarrow$



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

(a) original polyhedron

$A\vec{x} + \vec{a} \geq \vec{0}$

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

(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 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

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];

  }
```

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

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

```
for (i = 0; i < n; ++i)
 for (j = 0; j < n; ++j)
  C[i][j] = 0;
for (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];

  }
```

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

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

```
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];

  }
```

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

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

```
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];

  }
```

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

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

```
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];
```

- Number of rows of $\Theta \leftrightarrow$ number of outer-most sequential loops

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];
```

$$\Theta.\vec{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} . (\underbrace{\mathbf{i}\ \mathbf{j}\ \mathbf{i}\ \mathbf{j}\ \mathbf{k}}_{\vec{\imath}}\ \underbrace{\mathbf{n}\ \mathbf{n}}_{\vec{p}}\ \underbrace{\mathbf{1}\ \mathbf{1}}_{\mathbf{c}})^T$$

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];

  }
```

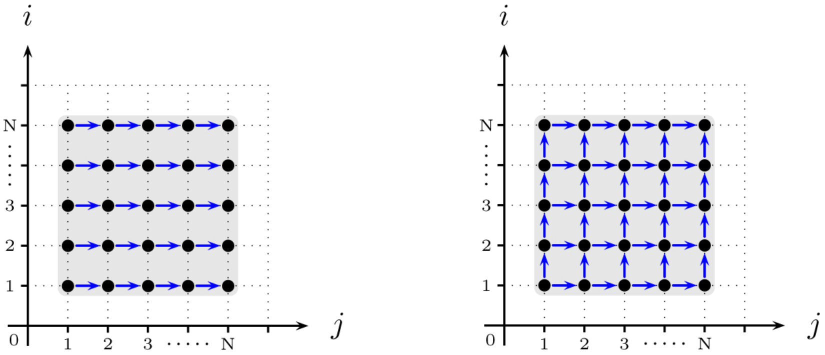
$$\Theta^{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} . \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}$$

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

```
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];
```

	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

Pictured Example



Example of 2 extended dependence graphs

Legal Program Transformation

A few properties:

- A transformation is illegal if a dependence crosses the hyperplane backwards
- A dependence going forward between 2 hyperplanes indicates sequentiality
- No dependence between any point of the hyperplane indicates parallelism

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 $\forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}$:

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

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?
- For example, 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, aka a legal transformation
- Embed some properties in this legal schedule
  - latency: minimize the time between the first and last iteration
  - parallelism (for placement)
  - permutability (for tiling)
  - ...

A 2-step approach:

- Find the solution set of all legal affine schedules
- Find an ILP formulation for the objective function

Selecting a Good Schedule

Build a cost function to select a (good) schedule:

- Minimize latency: bound the execution time

  Bound the program execution / find bounded delay [Feautrier]
  Given $L = w_0 + \vec{u}.\vec{w}$, compute $min(\Theta(\vec{x}) - L)$ s.t. Θ is legal

- Exhibit coarse-grain parallelism

  Placement constraints [Lim/Lam]
  $\Theta^R(\vec{x}_R) = \Theta^S(\vec{x}_S)$ for all instances s.t. Θ is legal

- Improve data locality (spatial/temporal reuse)

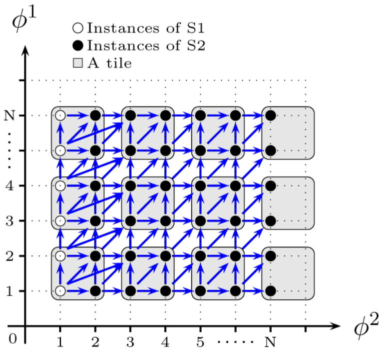
- Many more possible...

Tiling

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

Motivating Example

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]);
}
```

Motivating Example

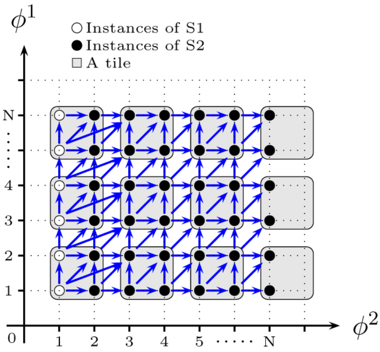
Example (FDTD-2D tiled)

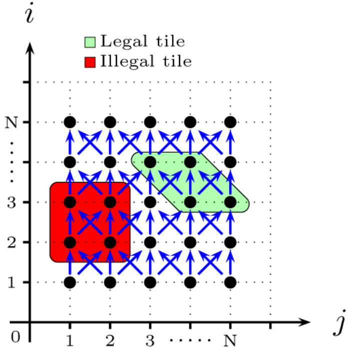
```
for (c0 = 0; c0 <= (((ny + 2 * tmax + -3) * 32 < 0?((32 < 0?-((-(ny + 2 * tmax + -3) + 32 + 1) / 32) : -((-(ny + 2 *
tmax + -3) + 32 - 1) / 32))) : (ny + 2 * tmax + -3) / 32)); ++c0) {
  #pragma omp parallel for private(c3, c4, c2, c5)
   for (c1 = (((c0 * 2 < 0?-(-c0 / 2) : ((2 < 0?(-c0 + -2 - 1) / -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

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)

Example: 1D-Jacobi

1-D Jacobi (imperfectly nested)

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

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

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

Example: 1D-Jacobi

1-D Jacobi (imperfectly nested)

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

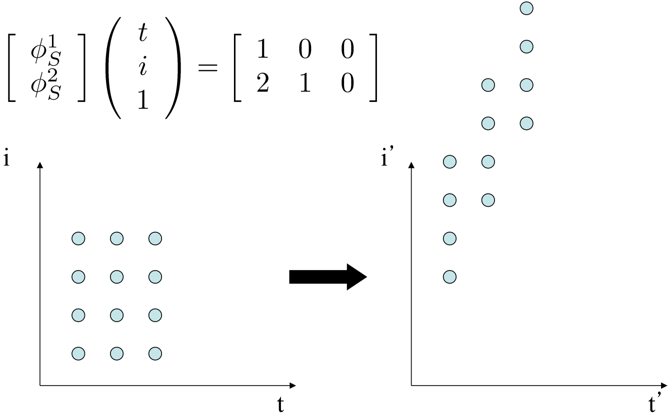
$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

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

Example: 1D-Jacobi

Transforming S

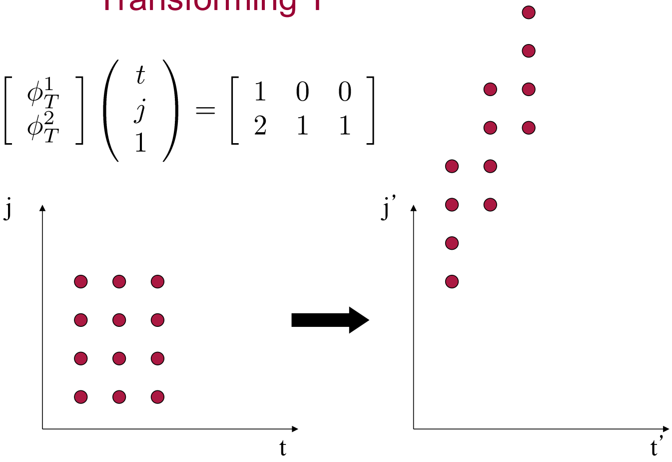
$$\begin{bmatrix} \phi_S^1 \\ \phi_S^2 \end{bmatrix} \begin{pmatrix} t \\ i \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$



Example: 1D-Jacobi

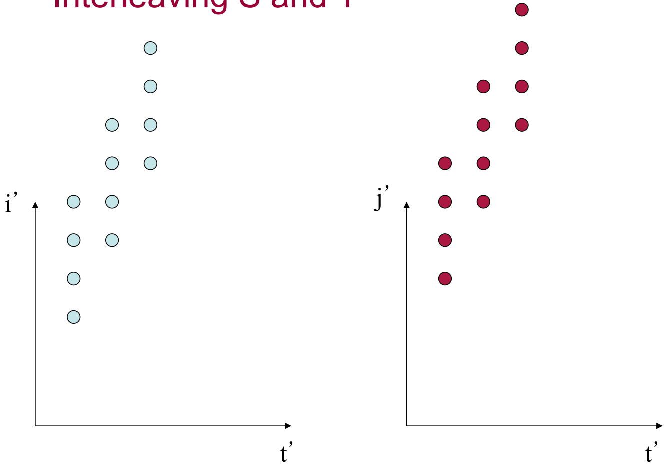
Transforming T

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



Example: 1D-Jacobi

Interleaving S and T

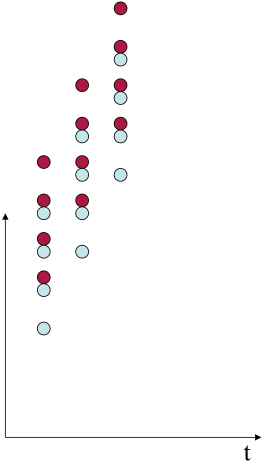


Example: 1D-Jacobi

Interleaving S and T

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

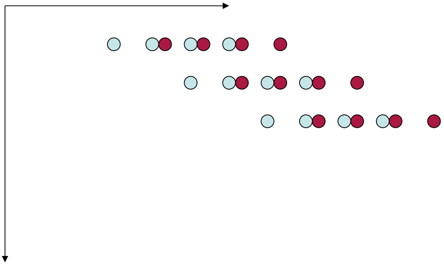
$$\begin{bmatrix} \phi_T^1 \\ \phi_T^2 \end{bmatrix} \begin{pmatrix} t \\ j \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$



Example: 1D-Jacobi

1-D Jacobi (imperfectly nested) – transformed code

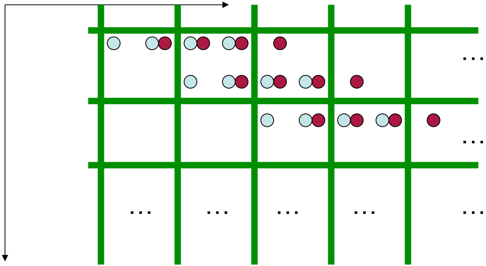
```
    for (t0=0;t0<=M-1;t0++) {
S':   b[2]=0.333*(a[2-1]+a[2]+a[2+1]);
      for (t1=2*t0+3;t1<=2*t0+N-2;t1++) {
S:      b[-2*t0+t1]=0.333*(a[-2*t0+t1-1]+a[-2*t0+t1]
                               +a[-2*t0+t1+1]);
T:      a[-2*t0+t1-1]=b[-2*t0+t1-1]; }
T':   a[N-2]=b[N-2]; }
```



Example: 1D-Jacobi

1-D Jacobi (imperfectly nested) – transformed code

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



Fusion-driven Optimization

Overview

Problem: How to improve program execution time?

- Focus on shared-memory computation
  - OpenMP parallelization
  - SIMD Vectorization
  - Efficient usage of the intra-node memory hierarchy
- Challenges to address:
  - Different machines require different compilation strategies
  - One-size-fits-all scheme hinders optimization opportunities

Question: how to restructure the code for performance?

Objectives for a Successful Optimization

During the program execution, interplay between the hardware ressources:

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

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

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

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

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

Running Example

Original code

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

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

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

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

Running Example

Cost model: maximal fusion, minimal synchronization

[Bondhugula et al., PLDI'08]

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

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

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

Running Example

Maximal distribution: best for Intel Xeon 7450

Poor data reuse, best vectorization

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

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

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

Running Example

Balanced distribution/fusion: best for AMD Opteron 8380

Poor data reuse, best vectorization

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

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

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

Running Example

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

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

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

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

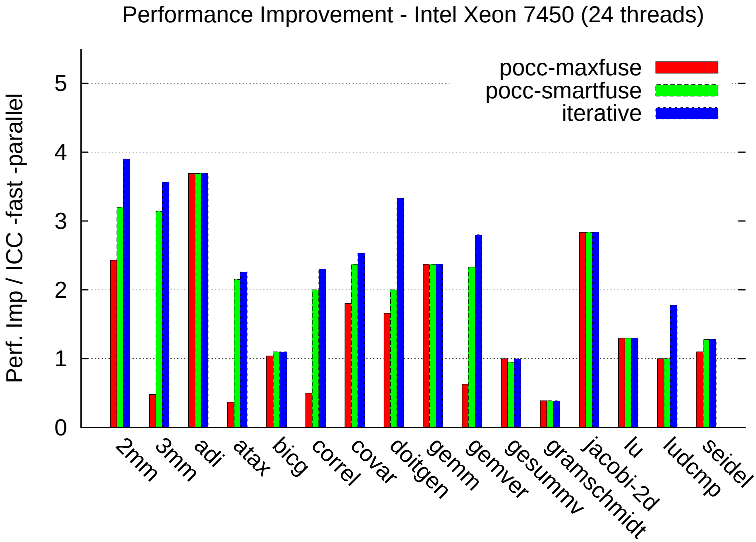
Table: Summary of the optimization process

Experimental Setup

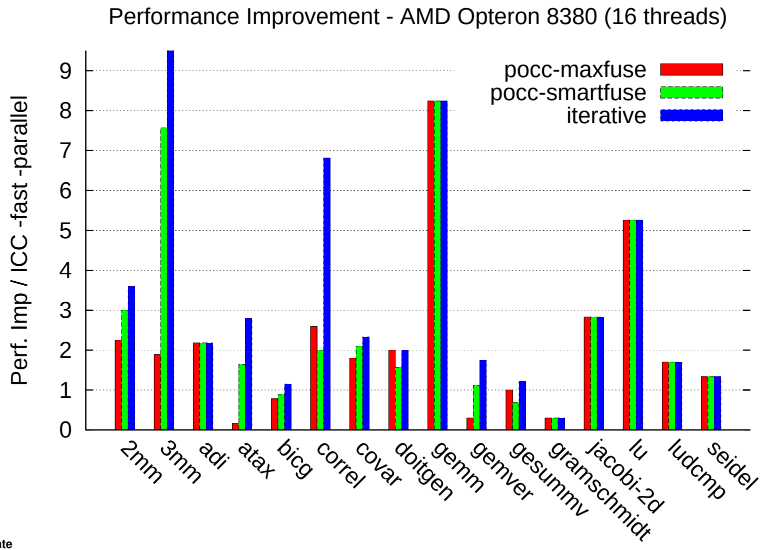
We compare three schemes:

- **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

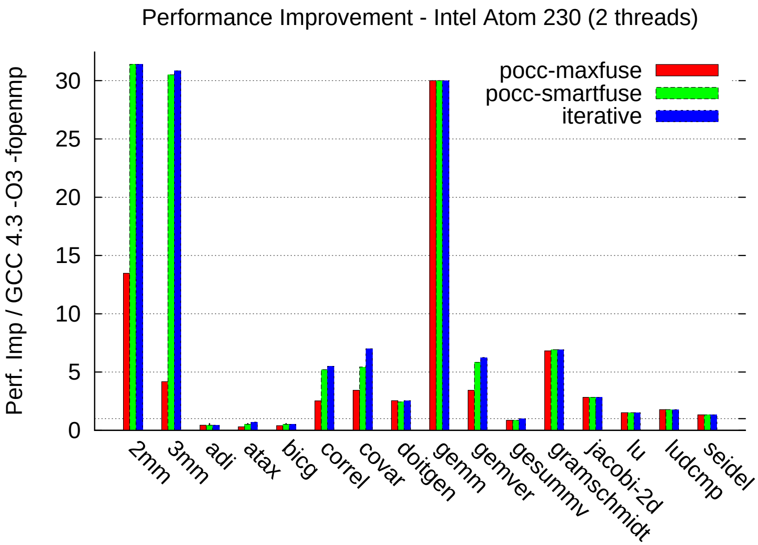
Performance Results - Intel Xeon 7450 - ICC 11



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 $2\times$)

Conclusions

Take-home message:

$\Rightarrow$ **Fusion / Distribution / Code motion highly program- and machine-specific**

$\Rightarrow$ **Minimum empirical tuning + polyhedral framework gives very good performance on several applications**

$\Rightarrow$ **Complete, end-to-end framework implemented and effectiveness demonstrated**

Future work:

- Further pruning of the search space (additional static cost models)
- Statistical search techniques

Polyhedral Toolbox

Polyhedral Software Toolbox

- Analysis:
  - Extracting the polyhedral representation of a program: Clan, PolyOpt
  - Computing the dependence polyhedra: Candl
- Mathematical operations:
  - Doing polyhedral operations on $\mathbb{Q}$-, $\mathbb{Z}$- and $\mathcal{Z}$-polyhedral: PolyLib, ISL
  - Solving ILP/PIP problems: PIPLib
  - Computing the number of points in a (parametric) polyhedron: Barvinok
  - Projection on $\mathbb{Q}$-polyhedra: FM, the Fourier-Motzkin Library
- Scheduling:
  - Tiling hyperplane method: PLuTo
  - Iterative selection of affine schedules: LetSee
- Code generation:
  - Generating C code from a polyhedral representation: CLooG
  - Parametric tiling from a polyhedral representation: PrimeTile, DynTile, PTile

Polyhedral Compilers

Available polyhedral compilers:

- Non-Free:
  - IBM XL/Poly
  - Reservoir Labs R-Stream
- Free:
  - GCC (see the GRAPHITE effort)
  - LLVM (see the Polly effort)
  - PIPS/Par4All (C-to-GPU support)
- Prototypes (non exhaustive list!):
  - **PolyOpt** from OSU, a polyhedral compiler using parts of PoCC and the Rose infrastructure
  - **PoCC, the POlyhedral Compiler Collection**
    http://pocc.sourceforge.net
    Contains Clan, Candl, Pluto, LetSee, PIPLib, PolyLib, FM, ISL, Barvinok, CLooG, ...
  - SUIF, Loopo, Clang+ISL, ...

Polyhedral Methodology Toolbox

- Semantics-preserving schedules:
  - Dependence relation finely characterized with dependence polyhedra
  - Algorithms should harness the power of this representation (ex: legality testing, parallelism testing, etc.)
- Scheduling:
  - Scheduling algorithm can be greedy (level-by-level) or global
  - Beware of scalability
  - Special properties can be embedded in the schedule via an ILP (ex: fusion, tiling, parallelism)
- Mathematics:
  - Beware of the distinction between $\mathbb{Q}$-, $\mathbb{Z}$- and $\mathcal{Z}$-polyhedra: always choose the most relaxed one that fits the problem
  - Farkas Lemma is useful to characterize a solution set
  - Farkas Lemma is also useful to linearize constraints

(Partial) State-of-the-art in Polyhedral Compilation

(...In my humble opinion)

- Analysis
  - Array Dataflow Analysis [Feautrier,IJPP91]
  - Dependence polyhedra [Feautrier,IJPP91] (Candl)
  - Non-static control flow support [Benabderrahmane,CC10]
- Program transformations:
  - Tiling hyperplane method [Bondhugula,CC08/PLDI08]
  - Convex space of all affine schedules [Vasilache,07]
  - Iterative search of schedules [Pouchet,CGO07/PLDI08]
  - Vectorization [Trifunovic,PACT09]
- Code generation
  - Arbitrary affine scheduling functions [Bastoul,PACT04]
  - Scalable code generation [Vasilache,CC06/PhD07]
  - Parametric Tiling [Hartono et al,ICS09/CGO10]

Some Ongoing Research [1/2]

- Scalability: provide more scalable algorithms, operating on hundreds of statements
  - Trade-off between optimality and scalability
  - Redesigning the framework: introducing approximations
- Vectorization: pre- and post- transformations for vectorization
  - Select the appropriate transformations for vectorization
  - Generate efficient SIMD code
- Scheduling: get (very) good performance on a wide variety of machines
  - Using machine learning to characterize the machine/compiler/program
  - Using more complex scheduling heuristics

Some Ongoing Research [2/2]

- GPU code generation
  - Specific parallelism pattern desired
  - Generate explicit communications
- Infrastructure development
  - Robustification and dissemination of tools
  - Fast prototyping vs. evaluating on large applications
- Polyhedral model extensions
  - Go beyond affine programs (using approximations?)
  - Support data layout transformations natively

Extra: Scheduling in the Polyhedral Model

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)

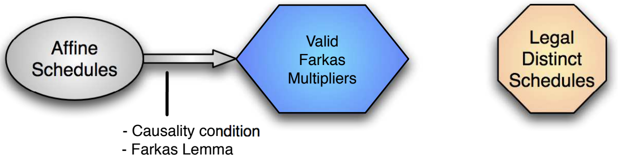
Lemma (Affine form of Farkas lemma)

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:

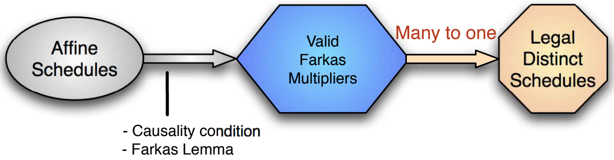
$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T(A\vec{x}+\vec{b}), \textit{ with } \lambda_0 \geq 0 \textit{ and } \vec{\lambda} \geq \vec{0}.$$

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

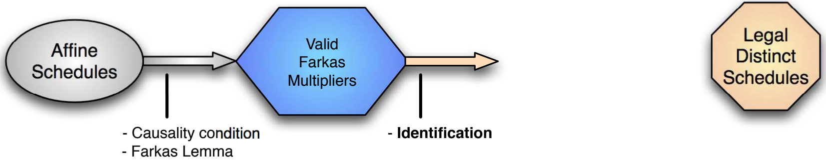
Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



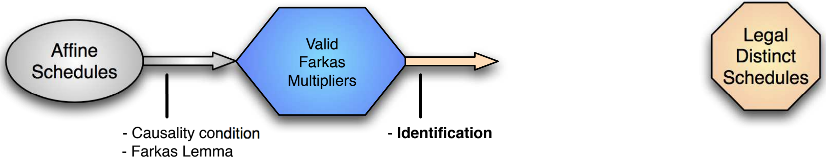
Example: Semantics Preservation (1-D)



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

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

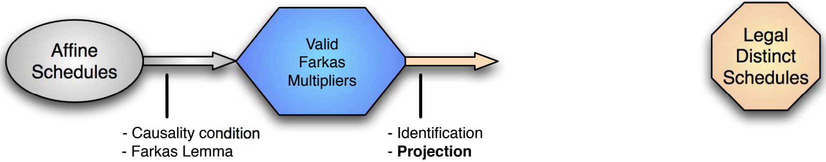
Example: Semantics Preservation (1-D)



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

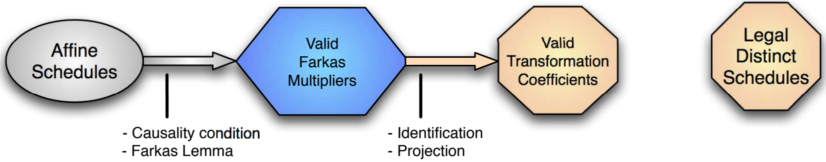
$$\begin{cases} D_{R\delta S} & \mathbf{i_R} & : & -t_{1_R} & = & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ & \mathbf{i_S} & : & t_{1_S} & = & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ & \mathbf{j_S} & : & t_{2_S} & = & \lambda_{D_{1,7}}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

Example: Semantics Preservation (1-D)

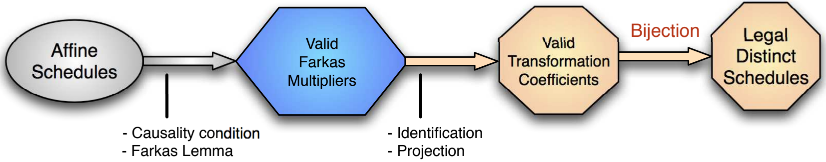


- Solve the constraint system
- Use (purpose-optimized) Fourier-Motzkin projection algorithm
  - Reduce redundancy
  - Detect implicit equalities

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
- These conditions for semantics preservation are not new! [Feautrier,92]

Generalization to Multidimensional Schedules

p-dimensional schedule **is not** $p \times 1$-dimensional schedule:

- Once a dependence is strongly satisfied ("loop"-carried), must be discarded in subsequent dimensions
- Until it is strongly satisfied, must be respected ("non-negative")

$\rightarrow$ Combinatorial problem: lexicopositivity of dependence satisfaction

A solution:

- Encode dependence satisfaction with decision variables [Feautrier,92]

$$\Theta_k^S(\vec{x}_S) - \Theta_k^R(\vec{x}_R) \geq \delta, \quad \delta \in \{0,1\}$$

- Bound schedule coefficients, and nullify the precedence constraint when needed [Vasilache,07]

Legality as an Affine Constraint

Lemma (Convex form of semantics-preserving affine schedules)

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

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

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

$$\text{(iii)} \quad \forall \mathcal{D}_{R,S},\ \forall p \in \{1,\dots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}, \tag{2}$$

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

$\longrightarrow$ Note: schedule coefficients must be bounded for Lemma to hold

$\longrightarrow$ Scalability challenge for large programs

Extra 2: Results on Loop Fusion/Distribution

Compiler Optimizations for Performance

- **High-level loop transformations are critical for performance...**
  - Coarse-grain parallelism (OpenMP)
  - Fine-grain parallelism (SIMD)
  - Data locality (reduce cache misses)

Compiler Optimizations for Performance

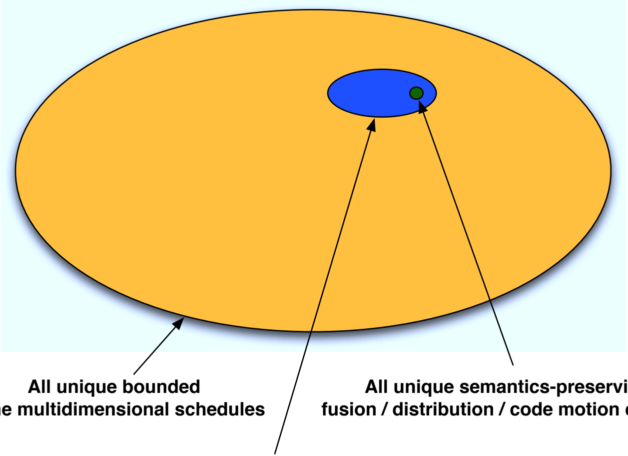
- **High-level loop transformations are critical for performance...**
  - Coarse-grain parallelism (OpenMP)
  - Fine-grain parallelism (SIMD)
  - Data locality (reduce cache misses)
- **... But deciding the best sequence of transformations is hard!**
  - Conflicting objectives: more SIMD implies less locality, etc.
  - It is machine-dependent and of course program-dependent
  - Expressive search spaces are required, but challenge the search!

Compiler Optimizations for Performance

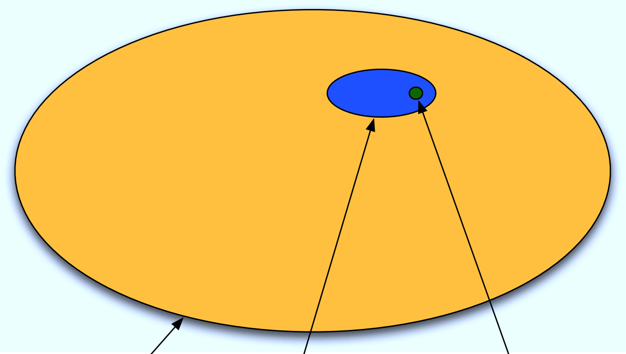
- **High-level loop transformations are critical for performance...**
  - Coarse-grain parallelism (OpenMP)
  - Fine-grain parallelism (SIMD)
  - Data locality (reduce cache misses)
- **... But deciding the best sequence of transformations is hard!**
  - Conflicting objectives: more SIMD implies less locality, etc.
  - It is machine-dependent and of course program-dependent
  - Expressive search spaces are required, but challenge the search!

- **Our approach:**
  - **Convexity:** model optimization spaces as convex set (ILP, scan, project, etc.)
  - **Pruning:** make our spaces contain all and only semantically equivalent programs in our framework
  - **Optimization:** decompose in two more tractable sub-problems without any loss of expressiveness, empirical search + ILP models

Spaces of Affine Loop transformations



Spaces of Affine Loop transformations



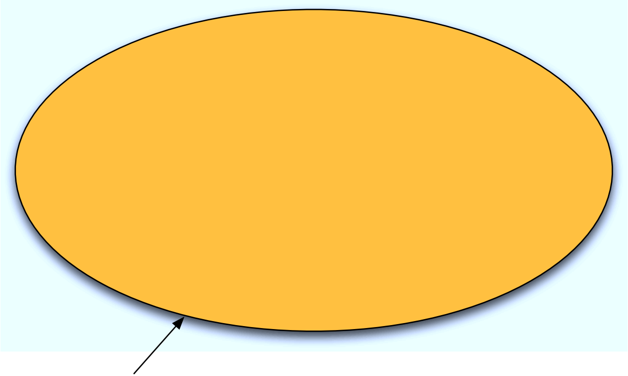
All unique bounded
affine multidimensional schedules

All unique semantics-preserving
fusion / distribution / code motion choices

All unique semantics-preserving
affine multidimensional schedules

Bounded: 10^{200} Legal: 10^{50} Empirical search: 10

Spaces of Affine Loop transformations



All unique bounded affine multidimensional schedules

1 point $\leftrightarrow$ 1 unique transformed program

Affine Schedule

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:

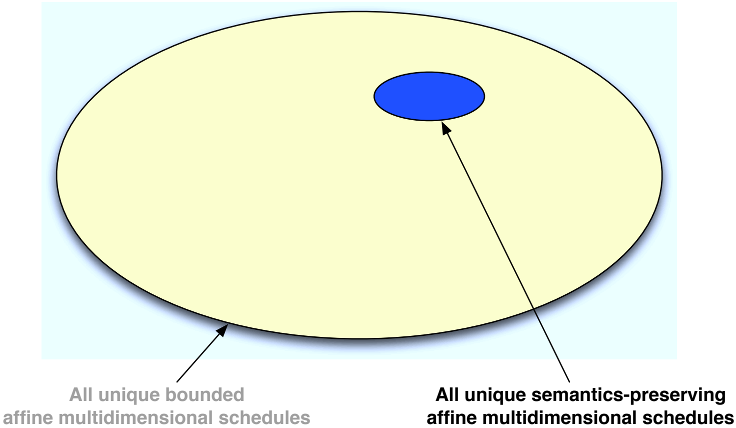
$$\Theta^S(\vec{x}_S) = \begin{pmatrix} \theta_{1,1} & \dots & \theta_{1,d+p+1} \\ \vdots & & \vdots \\ \theta_{m,1} & \dots & \theta_{m,d+p+1} \end{pmatrix} . \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix}$$

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 $x,y \in \mathbb{Z}$

Space of Semantics-Preserving Affine Schedules



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

Semantics Preservation

Definition (Causality condition)

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

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

$\prec$ denotes the *lexicographic ordering*.

$(a_1,\ldots,a_n) \prec (b_1,\ldots,b_m)$ iff $\exists i,\ 1 \le i \le min(n,m)$ 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:

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

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

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

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

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

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:

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

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

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

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

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

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

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

Convex Form of All Bounded Affine Schedules

Lemma (Convex form of semantics-preserving affine schedules)

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

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

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

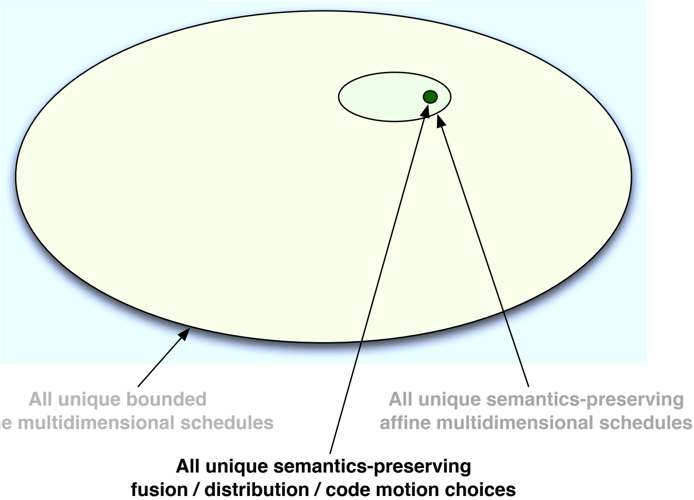
(iii) $\forall \mathcal{D}_{R,S},\ \forall p \in \{1,\ldots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S},$

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

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

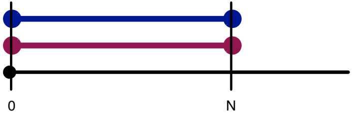
→ Bounded coefficients required [Vasilache,07]

Space of Semantics-Preserving Fusion Choices



1 point $\leftrightarrow$ 1 unique semantically equivalent program
(up to "partial" statement reordering)

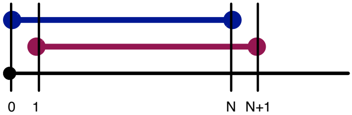
Fusion in the Polyhedral Model




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

Perfectly aligned fusion

Fusion in the Polyhedral Model

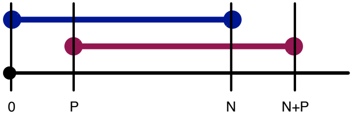



```
Blue(0);
for (i = 1; i <= N; ++i) {
  Blue(i);
  Red(i-1);
}
Red(N);
```

Fusion with shift of 1

Not all instances are fused

Fusion in the Polyhedral Model

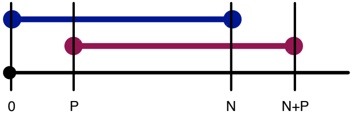



```
for (i = 0; i < P; ++i)
  Blue(i);
for (i = P; i <= N; ++i) {
  Blue(i);
  Red(i-P);
}
for (i = N+1; i <= N+P; ++i)
  Red(i-P);
```

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.

Affine Constraints for Fusibility

- **Two statements can be fused if their timestamp can overlap**

Definition (Generalized fusibility check)

Given v_R (resp. v_S) the set of vertices of $\mathcal{D}_R$ (resp. $\mathcal{D}_S$). R and S are fusible at level p if, $\forall k \in \{1 \dots p\}$, there exist two semantics-preserving schedules Θ_k^R and Θ_k^S such that

$$\exists (\vec{x}_1, \vec{x}_2, \vec{x}_3) \in v_R \times v_S \times v_R, \qquad \Theta_k^R(\vec{x}_1) \leq \Theta_k^S(\vec{x}_2) \leq \Theta_k^R(\vec{x}_3)$$

- Intersect $\mathcal{L}$ with fusibility and distribution constraints
- **Completeness:** if the test fails, then there is no sequence of affine transformations that can implement this fusion structure

Fusion / Distribution / Code Motion

Our strategy:

1. Build a set containing all unique fusion / distribution / code motion combinations
2. Prune all combinations that do not preserve the semantics

Given two statements R and S, three choices:

1. R is *fully before* S $\rightarrow$ distribution + code motion
2. R is *fully after* S $\rightarrow$ distribution + code motion
3. otherwise $\rightarrow$ fusion

$\Rightarrow$ It corresponds to all total preorders of R and S

Affine Encoding of Total Preorders

Principle:

- Model a total preorder with 3 binary variables

  $p_{i,j}: \ i < j \qquad s_{i,j}: \ i > j \qquad e_{i,j}: \ i = j$

- Enforce totality and mutual exclusion
- Enforce all cases of transitivity through affine inequalities connecting some variables. Ex: $e_{i,j} = 1 \wedge e_{j,k} = 1 \Rightarrow e_{i,k} = 1$

Affine Encoding of Total Preorders

Principle:

- Model a total preorder with 3 binary variables

  $p_{i,j} : \; i < j \qquad s_{i,j} : \; i > j \qquad e_{i,j} : \; i = j$
- Enforce totality and mutual exclusion
- Enforce all cases of transitivity through affine inequalities connecting some variables. Ex: $e_{i,j} = 1 \wedge e_{j,k} = 1 \Rightarrow e_{i,k} = 1$

- **This set contains one and only one point per distinct total preorder of n elements**

Affine Encoding of Total Preorders

Principle:

- Model a total preorder with 3 binary variables

  $p_{i,j} : i < j \qquad s_{i,j} : i > j \qquad e_{i,j} : i = j$
- Enforce totality and mutual exclusion
- Enforce all cases of transitivity through affine inequalities connecting some variables. Ex: $e_{i,j} = 1 \wedge e_{j,k} = 1 \Rightarrow e_{i,k} = 1$

- **This set contains one and only one point per distinct total preorder of n elements**
- Easy pruning: just bound the sum of some variables

  e.g., $e_{1,2} + e_{4,5} + e_{8,12} < 3$
- Automatic removal of supersets of unfusible sets

Convex set of All Unique Total Preorders

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

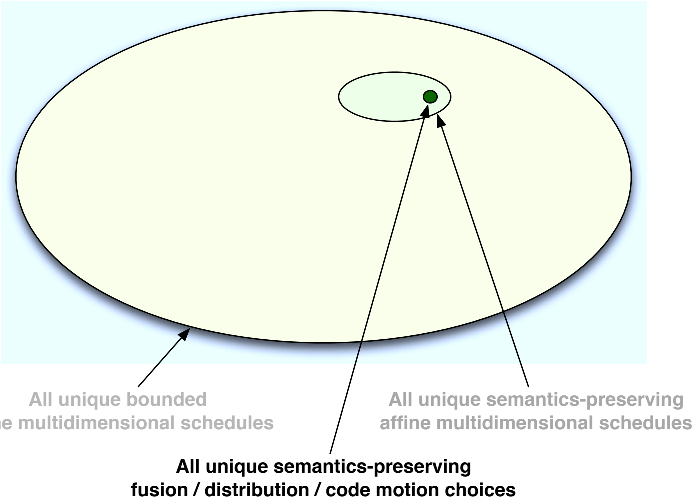
- Systematic construction for a given n, needs n^2 Boolean variables
- **Enable ILP modeling, enumeration, etc.**
- Extension to multidimensional total preorders (i.e., multi-level fusion)

Pruning for Semantics Preservation

Intuition: enumerate the smallest sets of unfusible statements

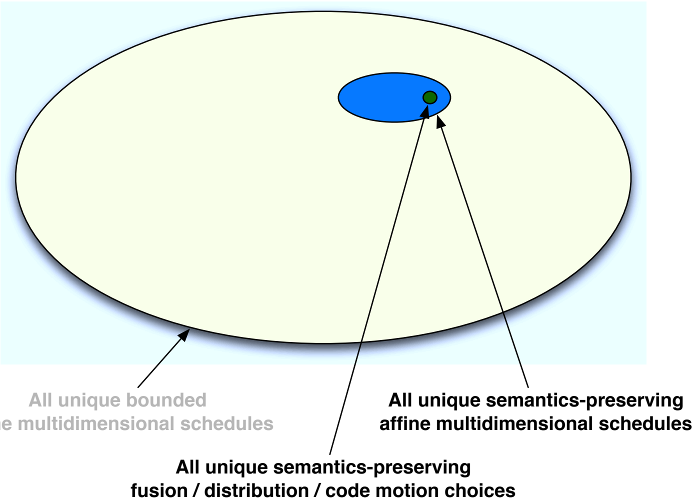
- Use an intermediate structure to represent sets of statements
  - Graph representation of maybe-unfusible sets (1 node per statement)
  - Enumerate sets from the smallest to the largest
- Leverage dependence graph + properties of fusion / distribution
- Compute properties by intersecting $\mathcal{L}$ with additional fusion / distribution / code motion affine constraints
- Any individual point can be removed from O

Space of Semantics-Preserving Fusion Choices



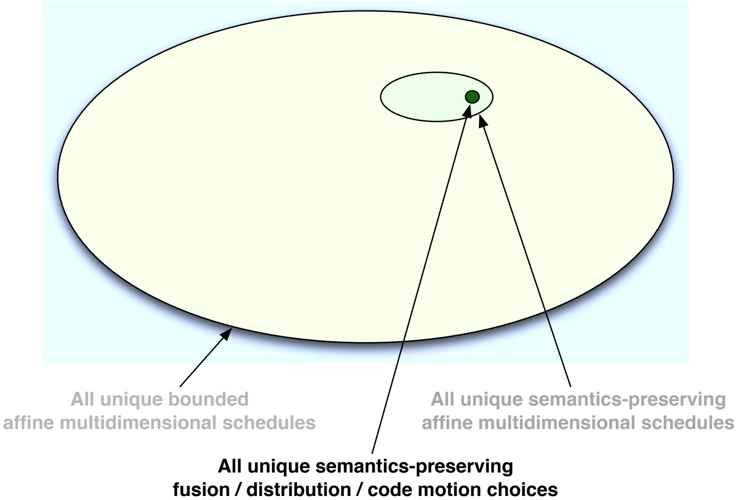
1 point $\leftrightarrow$ 1 unique semantically equivalent program (up to statement reordering)

Space of Semantics-Preserving Fusion Choices



1 point $\leftrightarrow$ **many** unique semantically equivalent programs
(up to iteration reordering)

Space of Semantics-Preserving Fusion Choices



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

Objectives for Effective Optimization

Objectives:

- Achieve efficient coarse-grain parallelization
- Combine iterative search of profitable transformations for tiling
  - $\rightarrow$ loop fusion and loop distribution

Tiling Hyperplane method [Bondhugula,08]

- Model-driven approach for automatic parallelization + locality improvement
- Tiling-oriented
- Poor model-driven heuristic for the selection of loop fusion (<u>not portable</u>)
- Overly relaxed definition of fused statements

Fusibility Restricted to Non-negative Schedules

- Fusibility is not a transitive relation!
  - Example: sequence of matrix-by-vector products $x = Ab,\ y = Bx,\ z = Cy$
  - $x = Ab,\ y = Bx$ can be fused, also $y = Bx,\ z = Cy$
  - They cannot be fused all together

- **Determining the Fusibility of a group of statements is reducible to exhibiting compatible pairwise loop permutations**
  - Extremely easy to compute all possible loop permutations that lead to fuse a pair of statements
  - Never check $\mathcal{L}$ on more than two statements!

- **Stronger definition of fusion**
  - Guarantee **at most c instances are not fused**

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

  - No combinatorial choice

The Optimization Algorithm in a Nutshell

Proceeds from the outer-most loop level to the inner-most:

1. Compute the space of valid fusion/distribution/code motion choices
2. **Select a fusion/distribution/code motion scheme** in this space
3. Compute an affine schedule **that implements this scheme**
   - Static cost model to select the schedule
   - Compound of skewing, shifting, fusion, distribution, interchange, tiling and parallelization (OpenMP)
   - **Maximize locality** for each set of statements to be fused

Experimental Results

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

Table: Search space statistics and performance improvement

- **Performance portability:** empirical search on the target machine of the optimal fusion structure
- Outperforms state-of-the-art cost models
- Full implementation in the source-to-source polyhedral compiler PoCC

Conclusion

Take-home message:

$\Rightarrow$ **Clear formalization of loop fusion** in the polyhedral model

$\Rightarrow$ **Formal definition of all semantically equivalent programs** up to:

- statement reordering
- limited affine iteration reordering
- arbitrary affine iteration reordering

$\Rightarrow$ **Effective and portable hybrid empirical optimization algorithm** (parallelization + data locality)

Future work:

- Develop static cost models for fusion / distribution / code motion
- Use statistical techniques to learn optimization algorithms