Iterative Optimization in the Polyhedral Model

Louis-Noël Pouchet

ALCHEMY group, INRIA Saclay / University of Paris-Sud 11, France

January 18th, 2010

Ph.D Defense

A Brief History...

- A Quick look backward:
  - 20 years ago: 80486 (1.2 M trans., 25 MHz, 8 kB cache)
  - **10 years ago:** Pentium 4 (42 M trans., **1.4 GHz**, 256 kB cache, SSE)
  - **7 years ago:** Pentium 4EE (169 M trans., **3.8 GHz**, 2 Mo cache, SSE2)
  - **4 years ago:** Core 2 Duo (**291 M** trans., **3.2 GHz**, 4 Mo cache, SSE3)
  - **1 years ago:** Core i7 Quad (**781 M** trans., **3.2 GHz**, 8 Mo cache, SSE4)

- Memory Wall: 400 MHz FSB speed vs 3+ GHz processor speed
- Power Wall: going multi-core, "slowing" processor speed
- Heterogeneous: CPU(s) + accelerators (GPUs, FPGA, etc.)

A Brief History...

- A Quick look backward:
  - 20 years ago: 80486 (1.2 M trans., 25 MHz, 8 kB cache)
  - **10 years ago:** Pentium 4 (42 M trans., **1.4 GHz**, 256 kB cache, SSE)
  - **7 years ago:** Pentium 4EE (169 M trans., **3.8 GHz**, 2 Mo cache, SSE2)
  - **4 years ago:** Core 2 Duo (**291 M** trans., **3.2 GHz**, 4 Mo cache, SSE3)
  - **1 years ago:** Core i7 Quad (**781 M** trans., **3.2 GHz**, 8 Mo cache, SSE4)
- Memory Wall: 400 MHz FSB speed vs 3+ GHz processor speed
- Power Wall: going multi-core, "slowing" processor speed
- Heterogeneous: CPU(s) + accelerators (GPUs, FPGA, etc.)

Compilers are facing a much harder challenge

Important Issues

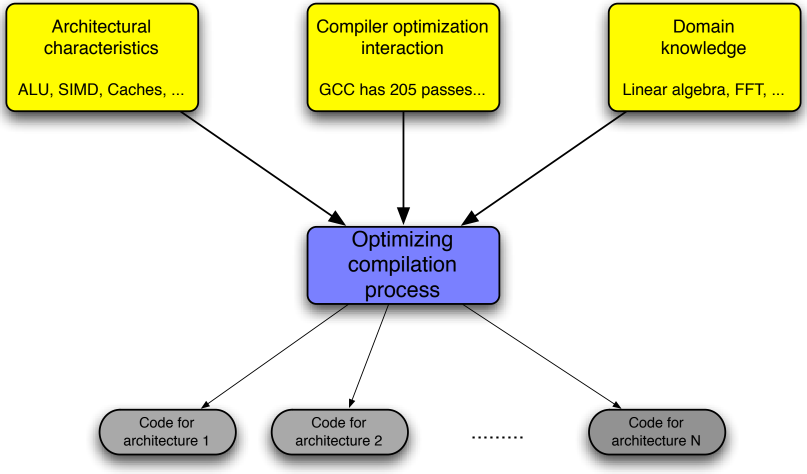
- New architecture $\longrightarrow$ New high-performance libraries needed
- **New architecture $\longrightarrow$ New optimization flow needed**
- Architecture complexity/diversity increases faster than optimization progress
- **Traditional approaches are not oriented towards performance portability...**

Important Issues

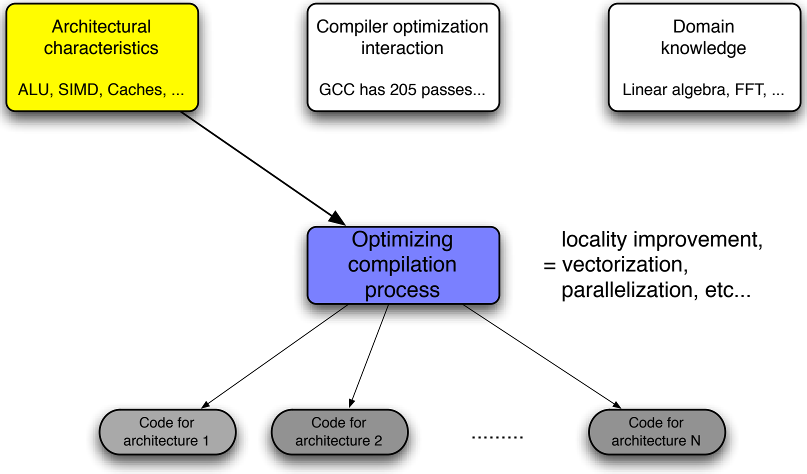
- New architecture $\longrightarrow$ New high-performance libraries needed
- **New architecture $\longrightarrow$ New optimization flow needed**
- Architecture complexity/diversity increases faster than optimization progress
- **Traditional approaches are not oriented towards performance portability...**

We need a portable optimization process

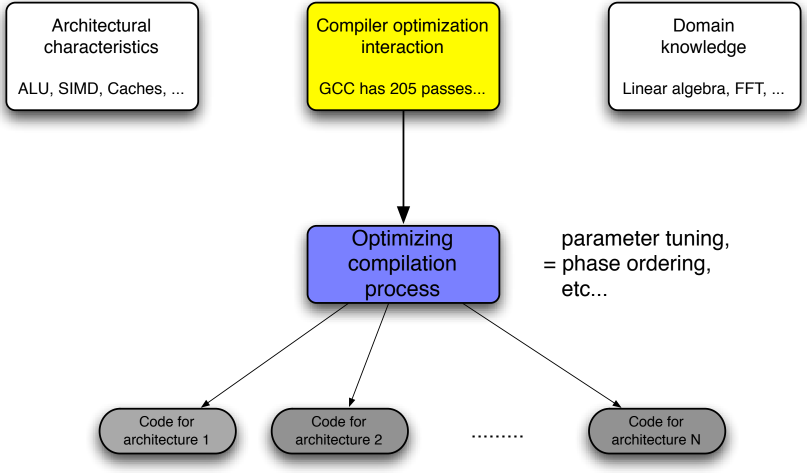
The Optimization Problem



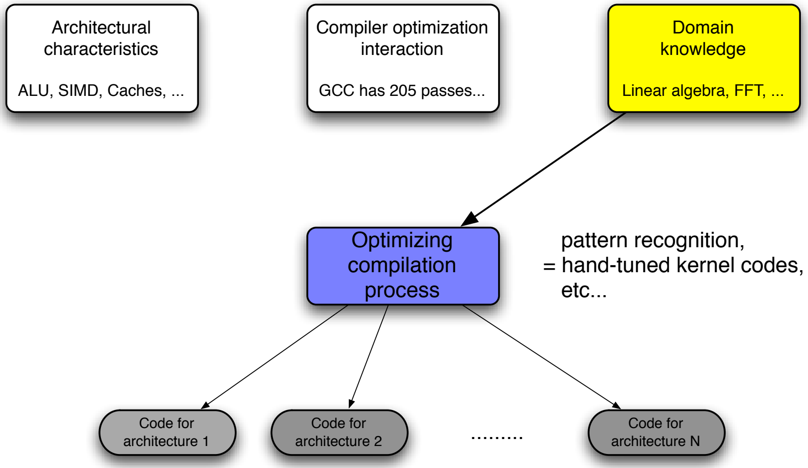
The Optimization Problem



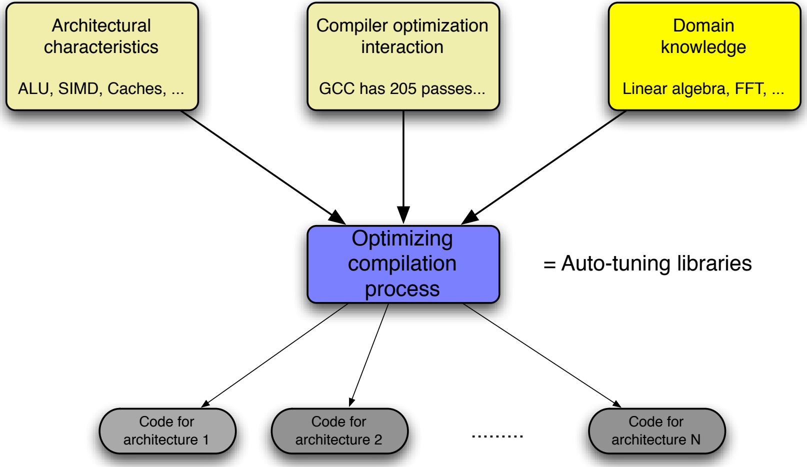
The Optimization Problem



The Optimization Problem



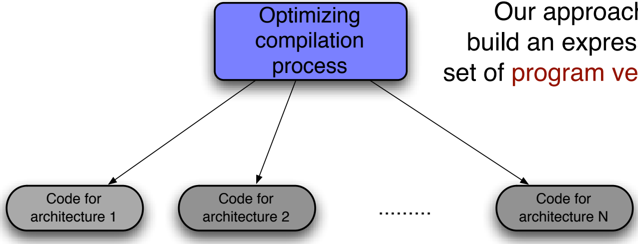
The Optimization Problem



The Optimization Problem

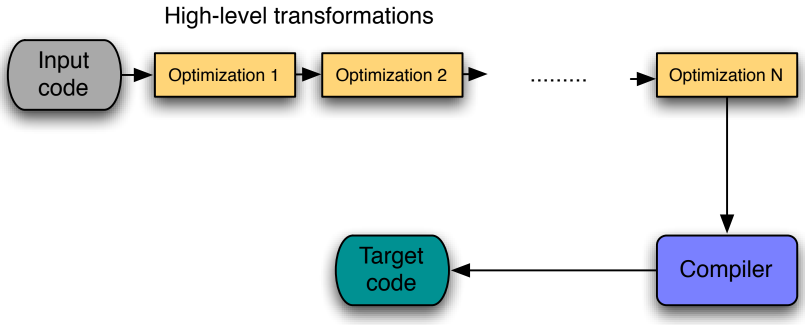
Architectural characteristics	Compiler optimization interaction	Domain knowledge
ALU, SIMD, Caches, ...	GCC has 205 passes...	Linear algebra, FFT, ...

In reality, there is a complex interplay between all components

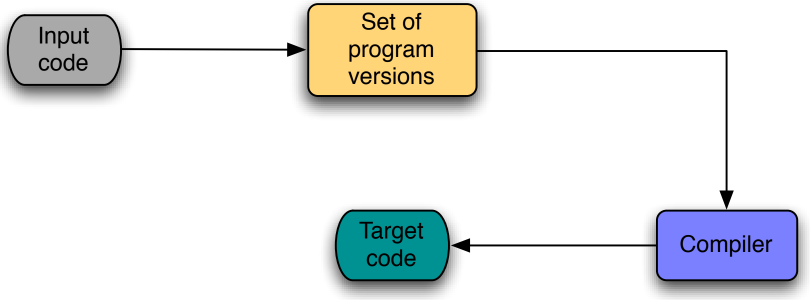


Our approach:
build an expressive
set of program versions

Iterative Optimization Flow

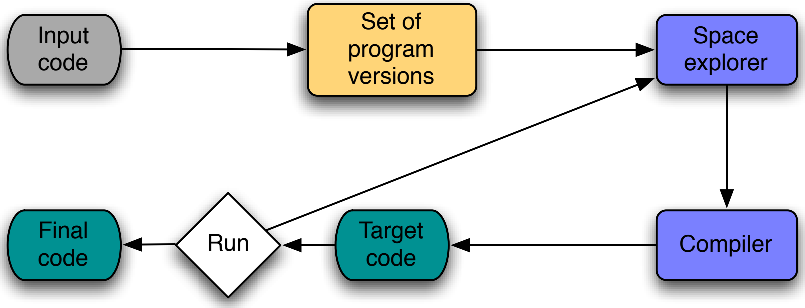


Iterative Optimization Flow



Program version = result of a sequence of loop transformation

Iterative Optimization Flow



Program version = result of a sequence of loop transformation

Other Iterative Frameworks

- Focus usually on composing existing compiler flags/passes
  - Optimization flags [Bodin et al.,PFDC98] [Fursin et al.,CGO06]
  - Phase ordering [Kulkarni et al.,TACO05]
  - Auto-tuning libraries (ATLAS, FFTW, ...)

- Others attempt to select a transformation sequence
  - SPIRAL [Püschel et al.,HPEC00]
  - Within UTF [Long and Fursin,ICPPW05], GAPS [Nisbet,HPCN98]
  - CHiLL [Hall et al.,USCRR08], POET [Yi et al.,LCPC07], etc.
  - URUK [Girbal et al.,IJPP06]

Other Iterative Frameworks

- Focus usually on composing existing compiler flags/passes
  - Optimization flags [Bodin et al.,PFDC98] [Fursin et al.,CGO06]
  - Phase ordering [Kulkarni et al.,TACO05]
  - Auto-tuning libraries (ATLAS, FFTW, ...)

- Others attempt to select a transformation sequence
  - SPIRAL [Püschel et al.,HPEC00]
  - Within UTF [Long and Fursin,ICPPW05], GAPS [Nisbet,HPCN98]
  - CHiLL [Hall et al.,USCRR08], POET [Yi et al.,LCPC07], etc.
  - URUK [Girbal et al.,IJPP06]

- Capability proven for efficient optimization
- Limited in applicability (legality)
- Limited in expressiveness (mostly simple sequences)
- Traversal efficiency compromised (uniqueness)

Our Approach: Set of Polyhedral Optimizations

What matters is the **result of the application of optimizations**, not the optimization sequence

All-in-one approach: [Pouchet et al.,CGO07/PLDI08]

- **Legality:** semantics is always preserved
- **Uniqueness:** all versions of the set are distinct
- **Expressiveness:** a version is the result of an arbitrarily complex sequence of loop transformation

- **Completion algorithm** to instantiate a legal version from a partially specified one
- **Dedicated traversal heuristics** to focus the search

The Polyhedral Model

The Polyhedral Model vs Syntactic Frameworks

Limitations of standard syntactic frameworks:

- Composition of transformations may be tedious
- Approximate dependence analysis
  - Miss optimization opportunities
  - Scalable optimization algorithms

The polyhedral model:

- Works on executed statement instances, finest granularity
- Model arbitrary compositions of transformations
- Requires computationally expensive algorithms

A Three-Stage Process

1 Analysis: from code to model

$\longrightarrow$ Existing prototype tools (some developed during this thesis)

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

$\longrightarrow$ GCC GRAPHITE (now in mainstream)

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

A Three-Stage Process

1 Analysis: from code to model

→ Existing prototype tools (some developed during this thesis)

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

→ GCC GRAPHITE (now in mainstream)

→ Reservoir Labs R-Stream, IBM XL/Poly

2 Transformation in the model

→ Build and select a program transformation

A Three-Stage Process

1 Analysis: from code to model

$\longrightarrow$ Existing prototype tools (some developed during this thesis)

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

$\longrightarrow$ GCC GRAPHITE (now in mainstream)

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

2 Transformation in the model

$\longrightarrow$ Build and select a program transformation

3 Code generation: from model to code

$\longrightarrow$ "Apply" the transformation in the model

$\longrightarrow$ Regenerate syntactic (AST-based) code

Polyhedral Representation of Programs

Static Control Parts

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

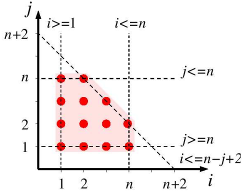
Polyhedral Representation of Programs

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];
}
```

$$f_{\mathbf{s}}(\vec{x_{S2}}) = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \end{array}\right] . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathbf{a}}(\vec{x_{S2}}) = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right] . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

$$f_{\mathbf{x}}(\vec{x_{S2}}) = \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \end{array}\right] . \begin{pmatrix} \vec{x_{S2}} \\ n \\ 1 \end{pmatrix}$$

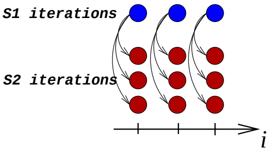
Polyhedral Representation of Programs

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

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];
  }
```

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

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

```
for (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];

  }
```

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

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

```
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 <u>sequential</u> 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];

  }
```

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

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

```
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];
```

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

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];

 }
```

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

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

```
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];
```

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

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
$\vec{i}$	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

Example: Semantics Preservation (1-D)

Example: Semantics Preservation (1-D)

Property (Causality condition for schedules)

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

$$\theta_R(\vec{x_R}) < \theta_S(\vec{x_S})$$

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

Example: Semantics Preservation (1-D)

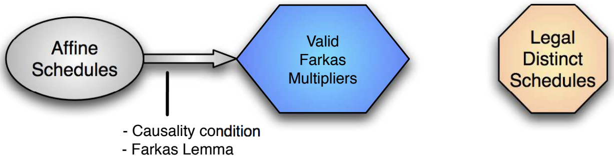
Lemma (Affine form of Farkas lemma)

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

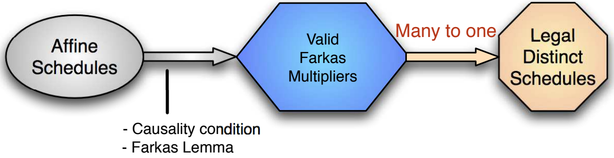
$$f(\vec{x}) = \lambda_0 + \vec{\lambda}^T(A\vec{x}+\vec{b}), \text{ with } \lambda_0 \geq 0 \text{ and } \vec{\lambda} \geq \vec{0}.$$

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

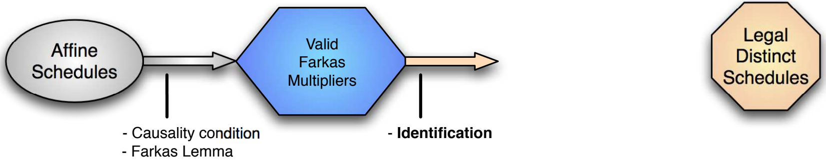
Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



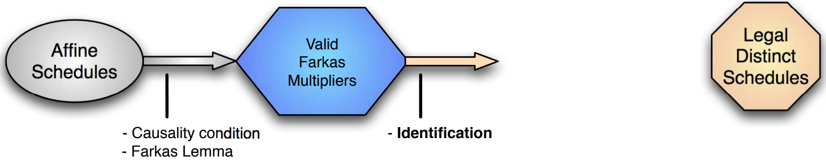
Example: Semantics Preservation (1-D)



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

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

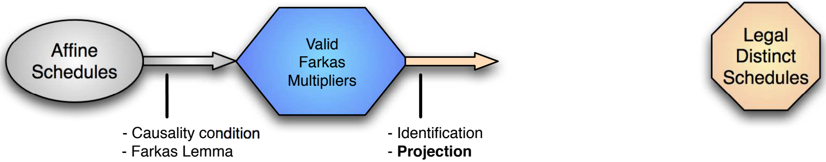
Example: Semantics Preservation (1-D)



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

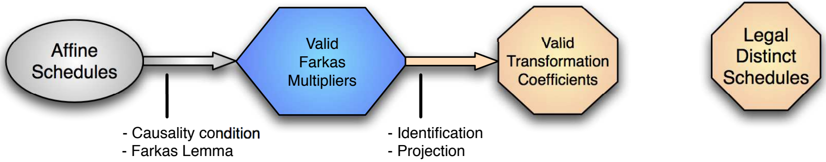
$$\left\{ \begin{array}{llcrcl} D_{R\delta S} & \mathbf{i_R} & : & -t_{1_R} & = & \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\ & \mathbf{i_S} & : & t_{1_S} & = & -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\ & \mathbf{j_S} & : &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

Example: Semantics Preservation (1-D)

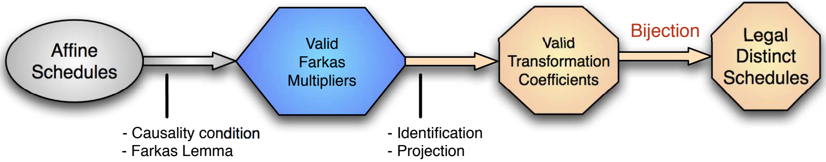


- Solve the constraint system
- Use (purpose-optimized) Fourier-Motzkin projection algorithm
  - Reduce redundancy
  - Detect implicit equalities

Example: Semantics Preservation (1-D)



Example: Semantics Preservation (1-D)



- One point in the space $\Leftrightarrow$ one set of legal schedules w.r.t. the dependences
- These conditions for semantics preservation are not new! [Feautrier,92]
- But never coupled with iterative search before

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")

$\longrightarrow$ 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:

$$\begin{aligned}
&(i) && \forall \mathcal{D}_{R,S},\ \delta_p^{\mathcal{D}_{R,S}} \in \{0,1\} \\
&(ii) && \forall \mathcal{D}_{R,S},\ \sum_{p=1}^{m} \delta_p^{\mathcal{D}_{R,S}} = 1 && (1) \\
&(iii) && \forall \mathcal{D}_{R,S},\ \forall p \in \{1,\dots,m\},\ \forall \langle \vec{x}_R, \vec{x}_S \rangle \in \mathcal{D}_{R,S}, && (2) \\
& && \Theta_p^S(\vec{x}_S) - \Theta_p^R(\vec{x}_R) \geq -\sum_{k=1}^{p-1} \delta_k^{\mathcal{D}_{R,S}}.(K.\vec{n}+K) + \delta_p^{\mathcal{D}_{R,S}}
\end{aligned}$$

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

$\longrightarrow$ Severe scalability challenge for large programs

Search Space Construction and Evaluation

Objectives for the Search Space Construction

- Provide **scalable** techniques to construct the search space
- **Adapt** the space construction to the machine specifics (esp. parallelism)
- Search space is infinite: requires appropriate **bounding**
- **Expressiveness:** allow for a rich set of transformations sequences
- Compiler optimization heuristics are fragile, manage it!

Overview of the Proposed Approach

1. Build a convex set of candidate program versions
   - Affine set of schedule coefficients
   - Enforce legality and uniqueness as affine constraints
2. Shape this set to a form which allows an efficient traversal
   - Redundancy-less Fourier-Motzkin elimination algorithm
   - Force FM-property by applying Fourier-Motzkin elim. on the set
3. Traverse the set
   - Exhaustively, for performance analysis
   - Heuristically, for scalability

Search Space Construction

Principle: Feautrier's + coefficient bounding

Output: 1 **independent polytope per schedule dimension**

Algorithm

Init: Set all dependencies as unresolved

1. $k = 1$
2. *Set $\mathcal{T}_k$ as the* ***polytope*** *of valid schedules with all unresolved dependencies weakly satisfied (i.e., set $\delta = 0$)*
3. *For each unresolved dependence $\mathcal{D}_{R,S}$:*
   1. *build $\mathcal{S}_{\mathcal{D}_{R,S}}$ the set of schedules strongly satisfying $\mathcal{D}_{R,S}$ (i.e., set $\delta = 1$)*
   2. $\mathcal{T}_k^{'} = \mathcal{T}_k \bigcap \mathcal{S}_{\mathcal{D}_{R,S}}$
   3. *if $\mathcal{T}_k^{'} \neq \emptyset$, $\mathcal{T}_k = \mathcal{T}_k^{'}$. Mark $\mathcal{D}_{R,S}$ as resolved*
4. *If unresolved dependence remains, increment k and go to 1*

Some Properties of the Algorithm

- Without bounding, equivalent to Feautrier's genuine scheduling algorithm
- With bounding, **sensitive to the dependence traversal order**
  - Heuristics to select the dependence order: pairwise interference, traffic ranking, etc.
  - May also search for different orders
- May not minimize the schedule dimensionality
- **Outer dimensions** (i.e., outer loops) **are more constrained**
- Inner dimensions tend to be parallel, if possible (SIMD friendly)

Search Space Size

- Bound each coefficient between $[-1, 1]$ to avoid complex control overhead and drive the search

Benchmark	#Inst.	#Dep.	#Dim.	dim 1	dim 2	dim 3	dim 4	Total
compress	6	56	3	20	136	10857025	n/a	2.9×10^{10}
edge	3	30	4	27	54	90534	43046721	5.6×10^{15}
iir	8	66	3	18	6984	$> 10^{15}$	n/a	$> 10^{19}$
fir	4	36	2	18	52953	n/a	n/a	9.5×10^{7}
lmsfir	9	112	2	27	10534223	n/a	n/a	2.8×10^{8}
mult	3	27	3	9	27	3295	n/a	8.0×10^{5}
latnrm	11	75	3	9	1896502	$> 10^{15}$	n/a	$> 10^{22}$
lpc-LPC_analysis	12	85	2	63594	$> 10^{20}$	n/a	n/a	$> 10^{25}$
ludcmp	14	187	3	36	$> 10^{20}$	$> 10^{25}$	n/a	$> 10^{46}$
radar	17	153	3	400	$> 10^{20}$	$> 10^{25}$	n/a	$> 10^{48}$

Figure: Search Space Statistics

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

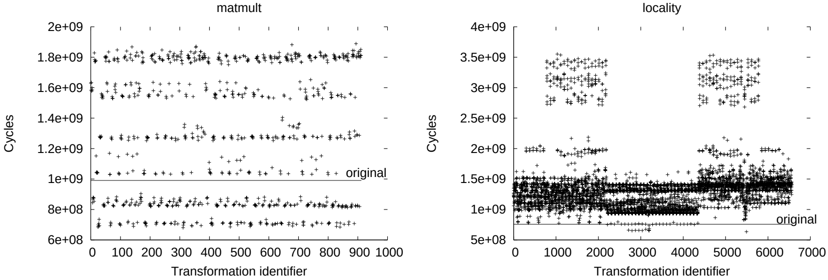


Figure: Performance distribution for `matmult` and `locality`

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

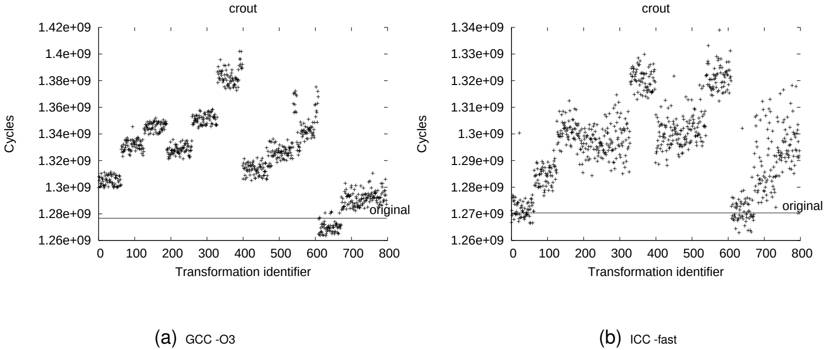


Figure: The effect of the compiler

Quantitative Analysis: The Hypothesis

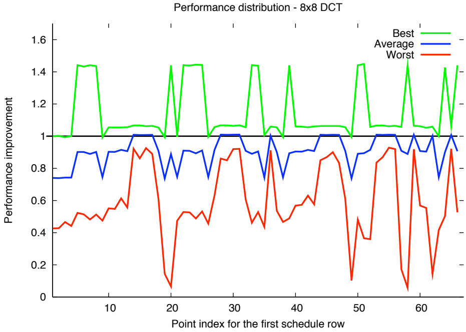
Extremely large generated spaces: $> 10^{50}$ points

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

Hypothesis: [Pouchet et al,SMART08]

- **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**

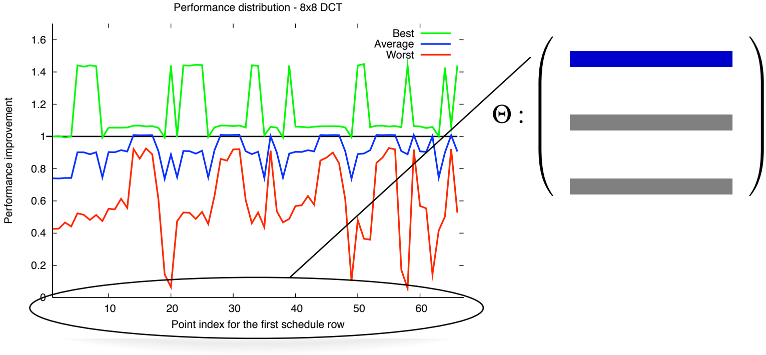
Observations on the Performance Distribution




```
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);
 }
```

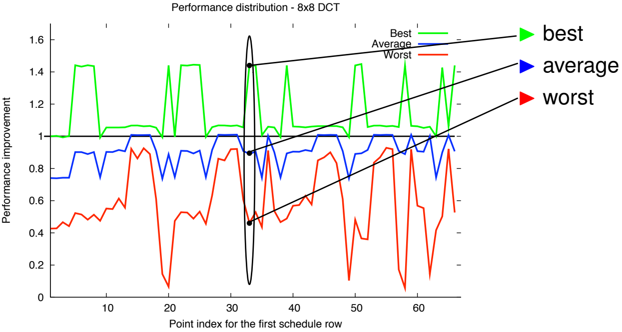
- Extensive study of 8x8 Discrete Cosine Transform (UTDSP)
- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions

Observations on the Performance Distribution



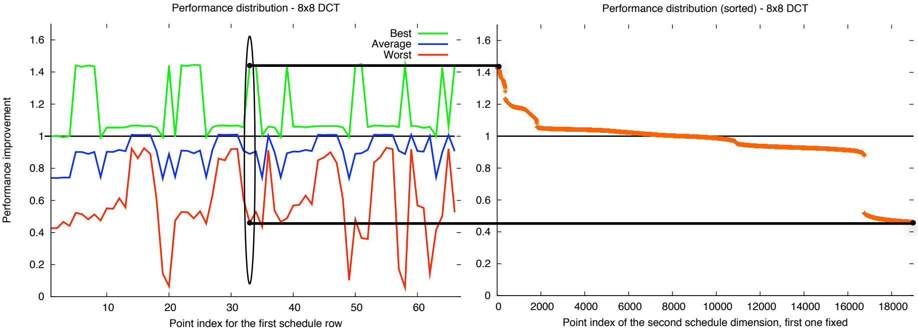
- Extensive study of 8x8 Discrete Cosine Transform (UTDSP)
- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions

Observations on the Performance Distribution



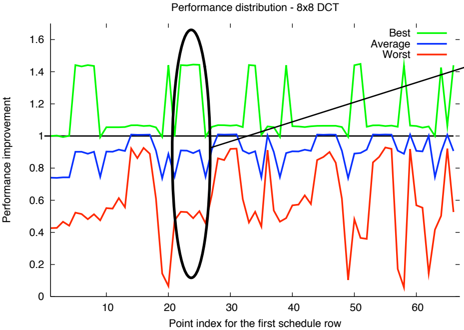
- Take one specific value for the first row
- Try the 19863 possible values for the second row

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: $< 0.02\%$

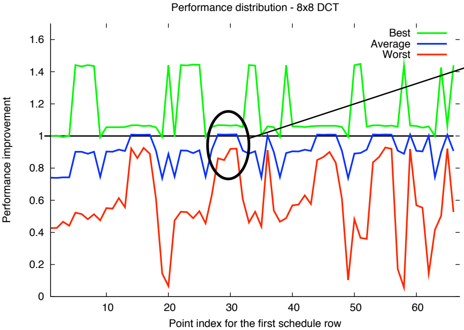
Observations on the Performance Distribution



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

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

Search Space Traversal

Objectives for Efficient Traversal

Main goals:

- Enable feedback-directed search
- **Focus the search on interesting subspaces**

Provide mechanisms to decouple the traversal:

- Leverage our knowledge on the performance distribution
- Leverage static properties of the search space
- Completion mechanism, to instantiate a full schedule from a partial one
- Traversal heuristics adapted to the problem complexity
  - **Decoupling heuristic:** explore first iterator coefficients (deterministic)
  - **Genetic algorithm:** improve further scalability (non-deterministic)

Some Results for 1-D Schedules

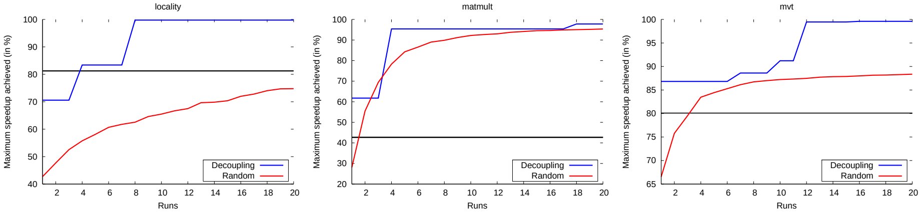
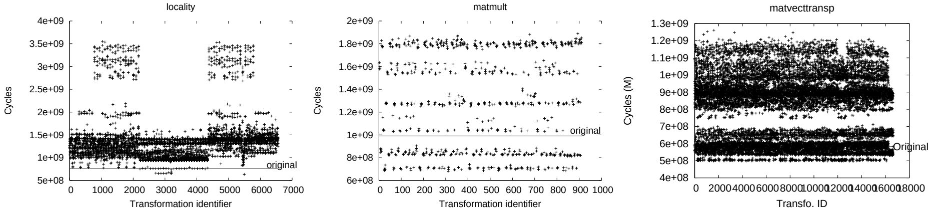


Figure: Comparison between random and decoupling heuristics



Inserting Randomness in the Search

About the performance distribution:

- The performance distribution is not uniform
- Wild jump in the space: tune $\vec{\imath}$ coefficients of upper dimensions
- Refinement: tune $\vec{p}$ and $\vec{c}$ coefficients

About the space of schedules:

- Highly constrained: **small change in $\vec{\imath}$ may alter many other coefficients**
- Rows are independent: no inter-dimension constraint
- Some transformations (e.g., interchange) must **operate between rows**

Genetic Operators

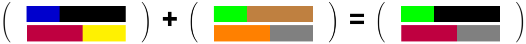
Mutation

- Probability varies along with evolution
- Tailored to focus on the most promising subspaces
- Preserves legality (closed under affine constraints)

Cross-over

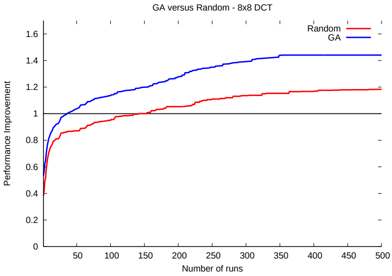
- Row cross-over

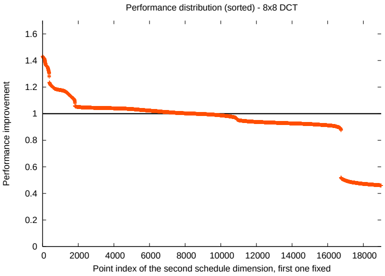
- Column cross-over



- Both preserve legality

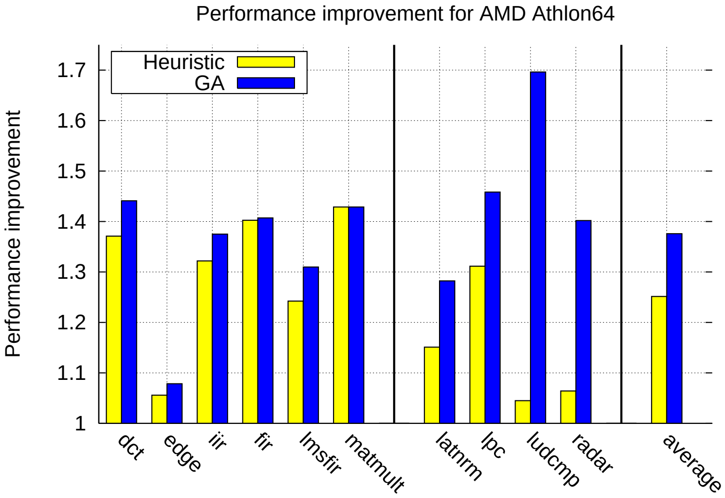
Dedicated GA Results





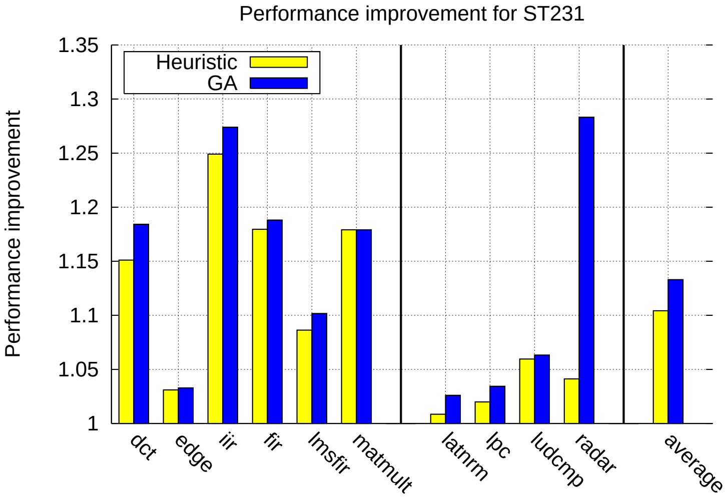
- GA converges towards the maximal space speedup

Experimental Results [1/2]



baseline: gcc -O3 -ftree-vectorize -msse2

Experimental Results [2/2]



baseline: st200cc -O3 -OPT:alias=restrict -mauto-prefetch

Assessments from Experimental Results

Looking into details (hardware counters+compilation trace):

- **Better activity** of the processing units
- Best version may **vary significantly for different architectures**
- Different source code may **trigger different compiler optimizations**

$\longrightarrow$ **Portability of the optimization process validated w.r.t. architecture/compiler**

Assessments from Experimental Results

Looking into details (hardware counters+compilation trace):

- **Better activity** of the processing units
- Best version may **vary significantly for different architectures**
- Different source code may **trigger different compiler optimizations**

$\longrightarrow$ **Portability of the optimization process validated w.r.t. architecture/compiler**

- Limitation: poor compatibility with coarse-grain parallelism

  Can we reconcile tiling, parallelization, SIMD and iterative search?

Multidimensional Interleaving Selection

Overview of the Problem

Objectives:

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

Existing framework: tiling hyperplane [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

Our Strategy in a Nutshell...

1. Introduce the concept of **fusability**
2. Introduce a modeling for arbitrary loop fusion/distribution combinations
   1. Equivalence 1-d interleaving with total preorders
   2. **Affine encoding of total preorders**
   3. Generalization to multidimensional interleavings
   4. Pruning technique to keep only semantics-preserving ones
3. Design a **mixed iterative and model-driven algorithm** to build optimizing transformations

Fusability of Statements

- Fusion $\Leftrightarrow$ interleaving of statement **instances**
- Two statements are fused if their timestamp overlap

$$\Theta_k^R(\vec{x_R}) \leq \Theta_k^S(\vec{x_S}) \wedge \Theta_k^S(\vec{x_S}') \leq \Theta_k^R(\vec{x_R}')$$

- Better approach: at most c instances are not fused (approximation)

Definition (Fusability restricted to non-negative schedule coefficients)

Given two statements R, S such that R is surrounded by d^R loops, and S by d^S loops. They are fusable at level p if, $\forall k \in \{1 \ldots p\}$, there exists two semantics-preserving schedules Θ_k^R and Θ_k^S such that:

$$\begin{array}{ll} (i) & \forall k \in \{1, \ldots, p\}, \quad -c < \Theta_k^R(\vec{0}) - \Theta_k^S(\vec{0}) < c \\ (ii) & \displaystyle\sum_{i=1}^{d^R} \theta_{k,i}^R > 0, \ \sum_{i=1}^{d^S} \theta_{k,i}^S > 0 \end{array}$$

Exact solution is hard: may require Ehrart polynomials for general case

Affine Encoding of Total Preorders

Principle: [Pouchet,PhD10]

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

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

Search Space Statistics

Pruning for semantics preservation ($\mathcal{F}$):

- Start from all total preorders ($\mathcal{O}$)
- Prove when fusability is a transitive relation: equivalent to checking the existence of **pairwise compatible loop permutations**
- Check graph of compatible permutations to determine fusable sets, prune $\mathcal{O}$ from non-fusable ones

			$\mathcal{O}$			$\mathcal{F}^1$				
Benchmark	#loops	#refs	#dim	#cst	#points	#dim	#cst	#points	#Tested	Time
advect3d	12	32	12	58	75	9	43	26	52	0.82s
atax	4	10	12	58	75	6	25	16	32	0.06s
bicg	3	10	12	58	75	10	52	26	52	0.05s
gemver	7	19	12	58	75	6	28	8	16	0.06s
ludcmp	9	**35**	**182**	**3003**	$\approx 10^{12}$	**40**	**443**	**8**	16	**0.54s**
doitgen	5	7	6	22	13	3	10	4	8	0.08s
varcovar	7	26	42	350	**47293**	22	193	**96**	192	**0.09s**
correl	5	12	30	215	4683	21	162	176	352	0.09s

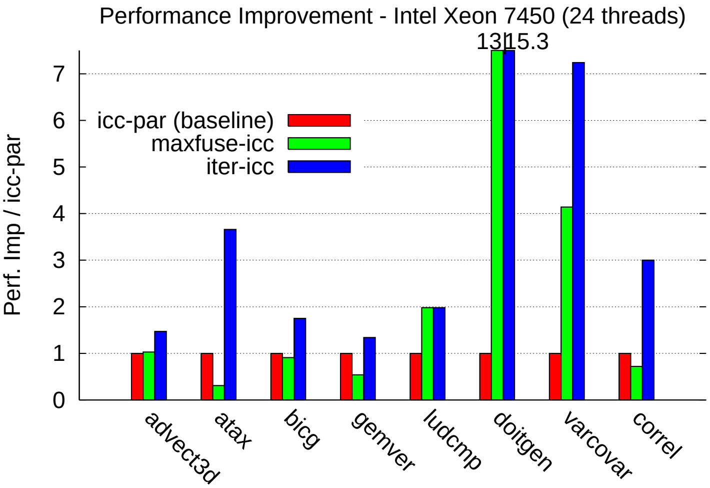
Figure: Search space statistics

Optimization Algorithm

- Proceeds **level-by-level**
- Starting from the outer-most level, **iteratively select an interleaving**
- For this interleaving, compute an optimization which respects it
  - Compound of skewing, shifting, fusion, distribution, interchange, tiling and parallelization (OpenMP)
  - **Maximize locality** for each partition of statements

- **Automatically adapt to the target architecture**
- Solid improvement over existing model-driven approach
- Up to 150$\times$ speedup on 24 cores, 15$\times$ speedup over autopll compiler

Performance Results for Intel Xeon 24-cores



baseline: ICC 11.0 -fast -parallel -fopenmp

Conclusions and Future Work

Summary of Contributions

We have designed, built and experimented **all required blocks to perform an efficient iterative selection of fine-grain loop transformations** in the polyhedral model.

- Theoretically sound and practical iterative optimization algorithms
  - Significant increase in expressiveness of iterative techniques
  - Well-designed (but complex) problems
  - Extensive experimental analysis of the performance distribution
  - Subspace-driven traversal techniques for polytopes
- Theoretical framework for generalized fusion
- Practical solution for machine-dependent parallelization + vectorization + locality
- Implementation in publicly available tools: PoCC, LetSee, FM, etc.

Future Work: Machine Learning

Machine Learning could improve the scalability:

- Currently, no reuse from previous compilation / space traversal
- Efficiency proved on (simpler) compilation problems

Main issues:

- Fine-grain vs. coarse-grain optimization
- **Knowledge representation**
- Features for similarity computation

Take-Home Message

Iterative Optimization: the last hope, or a new hope?

- **Efficient, more expressive and portable mechanisms can be built**
- **The polyhedral representation is adaptable to iterative compilation**
- Performance-demanding programmers can afford long compilation time
- Still require to execute different codes: not always possible
- Downside of polyhedral expressiveness: algorithmic complexity

Questions:

- Can we increase the accuracy of static models, given the complexity of modern compilers and chips?
- Can we systematically reach the performance of hand-tuned code with an automatic approach?

Take-Home Message

Iterative Optimization: the last hope, or a new hope?

- **Efficient, more expressive and portable mechanisms can be built**
- **The polyhedral representation is adaptable to iterative compilation**
- Performance-demanding programmers can afford long compilation time
- Still require to execute different codes: not always possible
- Downside of polyhedral expressiveness: algorithmic complexity

Questions:

- Can we increase the accuracy of static models, given the complexity of modern compilers and chips?
- Can we systematically reach the performance of hand-tuned code with an automatic approach?

Thank you!

Supplementary Slides

Yet Another Completion Algorithm

Principle: [Pouchet et al,PLDI08]

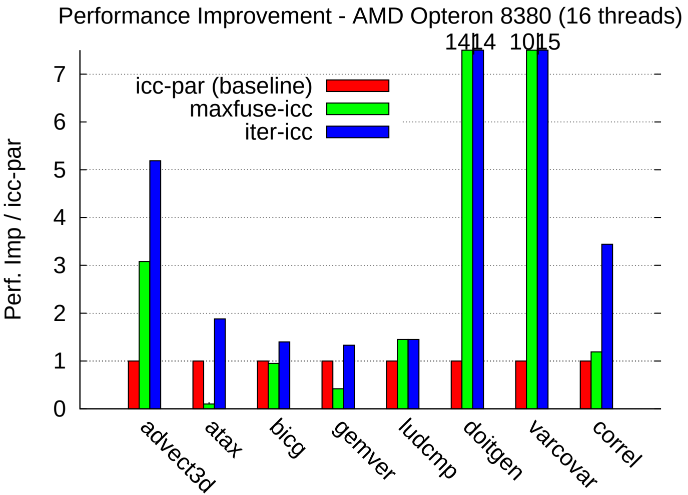
- Rely on a pre-pass to normalize the space (improved full polytope projection)
- Works in polynomial time w.r.t. the number of constraints in the normalized space

See also [Li et al,IJPP94] [Griebl,PACT98] [Vasilache,PACT07]...

Three fundamental properties:

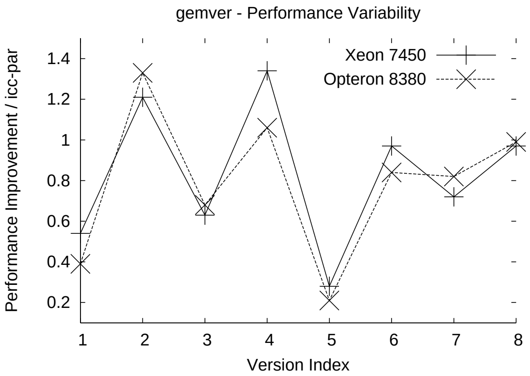
1. If $v_1, \ldots, v_k$ is a prefix of a legal point v, a completion is always found
2. This completion will only update $v_{k+1}, \ldots, v_{d_{\text{max}}}$, if needed;
3. When $v_1, \ldots, v_k$ are the $\vec{\imath}$ coefficients, the heuristic looks for the smallest absolute value for the $\vec{p}$ and c coefficients

Performance Results for AMD Opteron 16-cores



baseline: ICC 11.0 -fast -parallel -fopenmp

Variability for GEMVER



Future Work: Knowledge Transfer

Current approach:

- Training: 1 program $\longrightarrow$ 1 effective transformation
- On-line: Compute similarities with existing program, apply the same transformation

$\longrightarrow$ Does not work well for fine-grain optimization

Future Work: Knowledge Transfer

Current approach:

- Training: 1 program $\longrightarrow$ 1 effective transformation
- On-line: Compute similarities with existing program, apply the same transformation

$\longrightarrow$ Does not work well for fine-grain optimization

Proposed approach:

- Don't care about the sequence, only about properties of the schedule (parallelism degree, locality, etc.)
- Learn how to prioritize **performance anomaly** solving instead
- Rely on the polyhedral model to compute a matching optimization
- Some open problems:
  - How to compute (polyhedral) features? They are parametric
  - How to compute the optimization (combinatorial decision problem)?