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 year 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 → New high-performance libraries needed

- New architecture → New optimization flow needed

- Architecture complexity/diversity increases faster than optimization progress

- Traditional approaches are not oriented towards performance portability...
Important Issues

- New architecture → New high-performance libraries needed

- New architecture → 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

Architectural characteristics
ALU, SIMD, Caches, ...

Compiler optimization interaction
GCC has 205 passes...

Domain knowledge
Linear algebra, FFT, ...

Optimizing compilation process

Code for architecture 1
Code for architecture 2

Code for architecture N

..........
The Optimization Problem

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

Optimizing compilation process

Code for architecture 1
Code for architecture 2
Code for architecture N

locality improvement,
= vectorization,
parallelization, etc...
The Optimization Problem

Architectural characteristics
ALU, SIMD, Caches, ...

Compiler optimization interaction
GCC has 205 passes...

Domain knowledge
Linear algebra, FFT, ...

Optimizing compilation process

Code for architecture 1

Code for architecture 2

........

Code for architecture N

parameter tuning,
= phase ordering,
etc...
The Optimization Problem

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

Code for architecture 1
Code for architecture 2
........
Code for architecture N

pattern recognition,
= hand-tuned kernel codes, etc...
The Optimization Problem

- Architectural characteristics
  - ALU, SIMD, Caches, ...

- Compiler optimization interaction
  - GCC has 205 passes...

- Domain knowledge
  - Linear algebra, FFT, ...

= Auto-tuning libraries

- Code for architecture 1
- Code for architecture 2
- Code for architecture N
The Optimization Problem

**Architectural characteristics**
ALU, SIMD, Caches, ...

**Compiler optimization interaction**
GCC has 205 passes...

**Domain knowledge**
Linear algebra, FFT, ...

In reality, there is a complex interplay between all components

**Optimizing compilation process**

Our approach: build an expressive set of program versions

Code for architecture 1
Code for architecture 2

........

Code for architecture N
Iterative Optimization Flow

Input code → Optimization 1 → Optimization 2 → …… → Optimization N → Compiler

High-level transformations

Program version = result of a sequence of loop transformation
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
Outline:

1. The Polyhedral Model
2. Search Space Construction and Evaluation
3. Search Space Traversal
4. Interleaving Selection
5. Conclusions and Future Work

ALCHEMY group

ALCHEMY, INRIA Saclay
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
   → 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
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
   → 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

3 Code generation: from model to code
   → "Apply" the transformation in the model
   → 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

\[
D_{S1} = \begin{bmatrix}
1 & 0 & 0 & -1 \\
-1 & 0 & 1 & 0 \\
0 & 1 & 0 & -1 \\
-1 & 0 & 1 & 0 \\
-1 & -1 & 1 & 2
\end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix} \geq \vec{0}
\]

\[\text{Iteration domain of } S_1\]
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 $\vec{p}$

```latex
\begin{align*}
  f_s(x_S^2) &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} x_S^2 \\ n \\ 1 \end{pmatrix} \\
  f_a(x_S^2) &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} x_S^2 \\ n \\ 1 \end{pmatrix} \\
  f_x(x_S^2) &= \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{pmatrix} x_S^2 \\ n \\ 1 \end{pmatrix}
\end{align*}
```
The Polyhedral Model:

Alchemy group

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 $D_{S1}$ and $D_{S2}$ (**exact analysis**)

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

\[ D_{S1} \delta S2 : \begin{bmatrix}
1 & -1 & 0 & 0 \\
1 & 0 & 0 & -1 \\
-1 & 0 & 0 & 3 \\
0 & 1 & 0 & -1 \\
0 & -1 & 0 & 3 \\
0 & 0 & 1 & -1 \\
0 & 0 & -1 & 3
\end{bmatrix} \begin{pmatrix}
i_{S1} \\
i_{S2} \\
i_{S1}
\end{pmatrix} \geq 0 \]

S1 iterations

S2 iterations
Program Transformations

Original Schedule

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

S2: C[i][j] += A[i][k] * B[k][j];

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

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

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

\[ \Theta^{S_1} \vec{x}_{S_1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \end{pmatrix} \]

```
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];
  }
```

\[ \Theta^{S_2} \vec{x}_{S_2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \end{pmatrix} \]

- 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

\[
\Theta_{S1} \vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}
\]

\[
\Theta_{S2} \vec{x}_{S2} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}
\]

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

\[
\begin{align*}
\Theta^{S_1} \vec{x}_{S_1} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix} \\
\Theta^{S_2} \vec{x}_{S_2} &= \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \end{pmatrix}
\end{align*}
\]

\[
\begin{align*}
\text{for } (i = 0; i < n; ++i) \\
\text{for } (j = 0; j < n; ++j) \\
\text{C}[i][j] &= 0; \\
\text{for } (k = 0; k < n; ++k) \\
\text{C}[i][j] &= \text{A}[i][k] \ast \text{B}[k][j];
\end{align*}
\]

\[
\begin{align*}
\text{for } (i = 0; i < n; ++i) \\
\text{for } (j = 0; j < n; ++j) \\
\text{C}[i][j] &= 0; \\
\text{for } (k = 0; k < n; ++k) \\
\text{C}[i-n][j] &= \text{A}[i-n][k] \ast \text{B}[k][j];
\end{align*}
\]

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

Distribute loops + Interchange loops for S2

```c
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];
    }
```

\[
\Theta^{S1}_x S1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}
\]

\[
\Theta^{S2}_x S2 = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}
\]

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];
}

\[ \Theta^{S_1} \vec{x}_{S_1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \end{pmatrix} \]

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;

\[ \Theta^{S_2} \vec{x}_{S_2} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \end{pmatrix} \]

► All instances of S1 are executed after the last S2 instance
Program Transformations

A legal schedule

\[
\begin{align*}
\text{for (i = 0; i < n; ++i) for (j = 0; j < n; ++j) }
\Theta^{S_1} \vec{x}_{S_1} &= \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix} \\
\Theta^{S_2} \vec{x}_{S_2} &= \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \end{pmatrix} \\
\text{for (i = n; i < 2*n; ++i) for (j = 0; j < n; ++j) C[i][j] = 0;}
\end{align*}
\]

Delay the S2 instances

Constraints must be expressed between \(\Theta^{S_1}\) and \(\Theta^{S_2}\)
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];

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

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

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];

\begin{itemize}
  \item Number of rows of \( \Theta \) \leftrightarrow \text{number of outer-most sequential loops}
\end{itemize}
Program Transformations

Representing a schedule

\[
\begin{align*}
\text{for } (i = 0; i < n; ++i) & \\
& \text{for } (j = 0; j < n; ++j) \\
S1: & \quad C[i][j] = 0; \\
& \quad \text{for } (k = 0; k < n; ++k) \\
S2: & \quad C[i][j] += A[i][k] \ast B[k][j];
\end{align*}
\]

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

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

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

\text{for } (i = n; i < 2 \ast n; ++i) \\
& \text{for } (j = 0; j < n; ++j) \\
C[i][j] = 0; \\
& \text{for } (k = n+1; k <= 2 \ast n; ++k) \\
& \text{for } (j = 0; j < n; ++j) \\
& \text{for } (i = 0; i < n; ++i) \\
C[i][j] += A[i][k-n-1] \ast B[k-n-1][j];
\]
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] \times B[k][j];
    }

ΘS1. \vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ 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)
            C[i][j] += A[i][k-n-1] \times B[k-n-1][j];

ΘS2. \vec{x}_{S2} = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \\ 1 \end{pmatrix}

\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} \cdot \begin{pmatrix} i \\ j \\ i \\ j \\ k \\ n \\ n \\ 1 \\ 1 \end{pmatrix}^T

Θ. \vec{x} = \begin{pmatrix} i \\ j \\ i \\ j \\ k \\ n \\ n \\ 1 \\ 1 \end{pmatrix} \cdot \vec{p} \quad c
## Program Transformations

### Representing a schedule

```plaintext
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];
}

Θ_{S1}.\vec{x}_{S1} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix}
```

```plaintext
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

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \vec{i} )</td>
<td>reversal</td>
</tr>
<tr>
<td>( \vec{i} )</td>
<td>skewing</td>
</tr>
<tr>
<td>( \vec{i} )</td>
<td>interchange</td>
</tr>
<tr>
<td>( \vec{p} )</td>
<td>fusion</td>
</tr>
<tr>
<td>( \vec{p} )</td>
<td>distribution</td>
</tr>
<tr>
<td>( c )</td>
<td>peeling</td>
</tr>
<tr>
<td>( c )</td>
<td>shifting</td>
</tr>
</tbody>
</table>
Example: Semantics Preservation (1-D)
Example: Semantics Preservation (1-D)

Property (Causality condition for schedules)

Given $R \subseteq S$, $\theta_R$ and $\theta_S$ are legal iff for each pair of instances in dependence:

$$\theta_R(x_R) < \theta_S(x_S)$$

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

Lemma (Affine form of Farkas lemma)

Let \( D \) 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 \( D \) iff it is a positive affine combination:

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

\( \lambda_0 \) and \( \vec{\lambda}^T \) are called the Farkas multipliers.
Example: Semantics Preservation (1-D)
Example: Semantics Preservation (1-D)

Affine Schedules \rightarrow \text{Valid Farkas Multipliers} \rightarrow \text{Legal Distinct Schedules}

- Causality condition
- Farkas Lemma
Example: Semantics Preservation (1-D)

\[ \theta_S(\vec{x}_S) - \theta_R(\vec{x}_R) - 1 = \lambda_0 + \vec{\lambda}^T \left( D_{R,S} \left( \vec{x}_R \right) + \vec{d}_{R,S} \right) \geq 0 \]

\[
\begin{align*}
D_{R\delta S} & : \\
i_R & : \\
i_S & : \\
\lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\
-\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\
\lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\
\lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\
\lambda_{D_{1,0}} 
\end{align*}
\]

- Causality condition
- Farkas Lemma
- Identification
Example: Semantics Preservation (1-D)

\[ \theta_S(\vec{x}_S) - \theta_R(\vec{x}_R) - 1 = \lambda_0 + \lambda^T \left( D_{R,S} \left( \begin{array}{c} \vec{x}_R \\ \vec{x}_S \end{array} \right) + \vec{d}_{R,S} \right) \geq 0 \]

\[
\begin{align*}
D_{R\delta S} & \quad i_R : \quad -t_{1R} = \lambda_{D,1,1} - \lambda_{D,1,2} + \lambda_{D,1,3} - \lambda_{D,1,4} \\
i_S : \quad t_{1S} &= -\lambda_{D,1,1} + \lambda_{D,1,2} + \lambda_{D,1,5} - \lambda_{D,1,6} \\
j_S : \quad t_{2S} &= \lambda_{D,1,7} - \lambda_{D,1,8} \\
n : \quad t_{3S} - t_{2R} &= \lambda_{D,1,4} + \lambda_{D,1,6} + \lambda_{D,1,8} \\
l : \quad t_{4S} - t_{3R} - 1 &= \lambda_{D,1,0}
\end{align*}
\]
Example: Semantics Preservation (1-D)

- Causality condition
- Farkas Lemma
- Identification
- Projection

- Solve the constraint system
- Use (purpose-optimized) Fourier-Motzkin projection algorithm
  - Reduce redundancy
  - Detect implicit equalities
Example: Semantics Preservation (1-D)

- Causality condition
- Farkas Lemma
- Identification
- Projection

Valid Transformation Coefficients
Legal Distinct Schedules
Example: Semantics Preservation (1-D)

- Affine Schedules
  - Causality condition
  - Farkas Lemma

- Valid Farkas Multipliers
  - Identification
  - Projection

- Bijection

- Valid Transformation Coefficients

- Legal Distinct Schedules

▶ 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]
▶ 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")
→ Combinatorial problem: lexicopositivity of dependence satisfaction

A solution:

- Encode dependence satisfaction with decision variables [Feautrier,92]
  \[
  \Theta^S_k(\vec{x}_S) - \Theta^R_k(\vec{x}_R) \geq \delta, \quad \delta \in \{0, 1\}
  \]
- 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 $\Theta^R, \Theta^S \ldots$ of dimension $m$, the program semantics is preserved if the three following conditions hold:

(i) $\forall D_R, S, \delta^D_{p} \in \{0, 1\}$

(ii) $\forall D_R, S, \sum_{p=1}^{m} \delta^D_{p} = 1$ (1)

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

$$\Theta^S_p(\vec{x}_S) - \Theta^R_p(\vec{x}_R) \geq - \sum_{k=1}^{p-1} \delta^D_{k} \cdot (K.\vec{n} + K) + \delta^D_{p}$$

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

$\rightarrow$ 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 \(\mathcal{D}_{R,S}:\)
   - build \(S_{\mathcal{D}_{R,S}}\) the set of schedules strongly satisfying \(\mathcal{D}_{R,S}\) (i.e., set \(\delta = 1\))
   - \(\mathcal{T}_k' = \mathcal{T}_k \cap S_{\mathcal{D}_{R,S}}\)
   - 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

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

**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]

(a) GCC -O3

(b) ICC -fast

Figure: The effect of the compiler
Quantitative Analysis: The Hypothesis

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

→ 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 $\Theta$ are more performance impacting than the last ones
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

- 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

Performance distribution - 8x8 DCT

- Best
- Average
- Worst

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

Large performance variation
Observations on the Performance Distribution

- 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: \( \overline{t} > \overline{p} > c \)

- Non-uniform distribution of performance

- No clear analytical property of the optimization function

→ 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 \( \vec{t} \) coefficients of upper dimensions
- Refinement: tune \( \vec{p} \) and \( \vec{c} \) coefficients

About the space of schedules:

- Highly constrained: small change in \( \vec{t} \) 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
  \[
  \begin{pmatrix}
  \text{blue}
  \\
  \text{cyan}
  \\
  \end{pmatrix}
  +
  \begin{pmatrix}
  \text{red}
  \\
  \text{brown}
  \\
  \end{pmatrix}
  =
  \begin{pmatrix}
  \text{blue}
  \\
  \text{brown}
  \\
  \end{pmatrix}
  \]

- Column cross-over
  \[
  \begin{pmatrix}
  \text{blue}
  \\
  \text{red}
  \\
  \text{yellow}
  \\
  \end{pmatrix}
  +
  \begin{pmatrix}
  \text{green}
  \\
  \text{orange}
  \\
  \text{gray}
  \\
  \end{pmatrix}
  =
  \begin{pmatrix}
  \text{green}
  \\
  \text{orange}
  \\
  \text{gray}
  \\
  \end{pmatrix}
  \]

- Both preserve legality
Dedicated GA Results

- GA converges towards the maximal space speedup
Experimental Results [1/2]

Performance improvement for AMD Athlon64

baseline: gcc -O3 -ftree-vectorize -msse2
Experimental Results [2/2]

Performance improvement for ST231

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**

→ **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

→ 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
  - 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 (not portable)
- 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
   - Equivalence 1-d interleaving with total preorders
   - **Affine encoding of total preorders**
   - Generalization to multidimensional interleavings
   - 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

$$\Theta^R_k(\vec{x}_R) \leq \Theta^S_k(\vec{x}_S) \land \Theta^S_k(\vec{x}'_S) \leq \Theta^R_k(\vec{x}'_R)$$

- 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 fusible at level $p$ if, $\forall k \in \{1, \ldots, p\}$, there exists two semantics-preserving schedules $\Theta^R_k$ and $\Theta^S_k$ such that:

1. $\forall k \in \{1, \ldots, p\}$, $-c < \Theta^R_k(\vec{0}) - \Theta^S_k(\vec{0}) < c$
2. $\sum_{i=1}^{d^R} \theta^R_{k,i} > 0$, $\sum_{i=1}^{d^S} \theta^S_{k,i} > 0$

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
  \[ p_{i,j} : i < j \quad s_{i,j} : i > j \quad e_{i,j} : i = j \]
- Enforce totality and mutual exclusion
- Enforce all cases of transitivity through affine inequalities connecting some variables. Ex: \( e_{i,j} = 1 \land e_{j,k} = 1 \Rightarrow e_{i,k} = 1 \)

\[
O = \begin{cases} 
0 \leq p_{i,j} \leq 1 \\
0 \leq e_{i,j} \leq 1 \\
0 \leq s_{i,j} \leq 1 
\end{cases}
\]

constrained to:

\[
O = \begin{cases} 
0 \leq p_{i,j} \leq 1 \\
0 \leq e_{i,j} \leq 1 \\
p_{i,j} + e_{i,j} \leq 1 \\
\forall k \in ]j,n] \quad e_{i,j} + e_{i,k} \leq 1 + e_{j,k} \\
\forall k \in ]i,j[ \quad p_{i,k} + p_{k,j} \leq 1 + p_{i,j} \\
\forall k \in ]j,n] \quad e_{i,j} + p_{i,k} \leq 1 + p_{j,k} \\
\forall k \in ]i,j[ \quad e_{i,j} + p_{j,k} \leq 1 + p_{i,k} \\
\forall k \in ]j,n] \quad e_{k,j} + p_{i,k} \leq 1 + p_{i,j} \\
\forall k \in ]i,j[ \quad e_{k,j} + p_{j,k} \leq 1 + p_{i,j} \\
\forall k \in ]j,n] \quad e_{i,j} + p_{i,j} + p_{j,k} \leq 1 + p_{i,k} + e_{i,k} \\
\end{cases}
\]

Variables are binary
Relaxed mutual exclusion
Basic transitivity on \( e \)
Basic transitivity on \( p \)
Complex transitivity on \( p \) and \( e \)
Complex transitivity on \( s \) and \( p \)
Search Space Statistics

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

- Start from all total preorders ($O$)
- 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 $O$ from non-fusible ones

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

**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

Performance Improvement - Intel Xeon 7450 (24 threads)

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?
Conclusions and Future Work:

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 $\vec{t}$ coefficients, the heuristic looks for the smallest absolute value for the $\vec{p}$ and $c$ coefficients.
Performance Results for AMD Opteron 16-cores

Performance Improvement - AMD Opteron 8380 (16 threads)

- icc-par (baseline)
- maxfuse-icc
- iter-icc

Baseline: ICC 11.0 -fast -parallel -fopenmp
Variability for GEMVER

![Variability for GEMVER](image)

Performance Improvement / icc-par

Version Index

Xeon 7450

Opteron 8380
Future Work: Knowledge Transfer

Current approach:

- Training: 1 program $\rightarrow$ 1 effective transformation
- On-line: Compute similarities with existing program, apply the same transformation
  $\rightarrow$ Does not work well for fine-grain optimization
Future Work: Knowledge Transfer

Current approach:
- Training: 1 program → 1 effective transformation
- On-line: Compute similarities with existing program, apply the same transformation
  → 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)?