Outline

Context of this study:
- Focus on Loop Nest Optimization for regular loops
- Automatic method for parallelism extraction / loop transformation
- Combine iterative methods with the power of the polyhedral model
- Solution independent of the compiler and the target machine

Our contribution:
- Search space construction
  - 1 point in the space $\Leftrightarrow$ 1 distinct legal program version
  - suitable for various exploration methods
- Performance
  - 99% of the best speedup attained within 20 runs of a dedicated heuristic
  - wall clock optimal transformation discoverable on small kernels
One-Dimensional Scheduling

Original Schedule

```c
for (i=0; i<n; ++i) {
    . S1(i);
    . for (j=0; j<n; ++j)
    . . S2(i,j);
}
```

\[
\begin{align*}
\theta_{S1} &= i \\
\theta_{S2} &= i
\end{align*}
\]

- Specify the outer-most loop only
- **Initial outer-most loop is** \(i\)
One-Dimensional Scheduling

Distribute loops

\[
\begin{align*}
\theta_{S1} &= i \\
\theta_{S2} &= i + n
\end{align*}
\]

- Specify the outer-most loop only
- All instances of S1 are executed before the first S2 instance
One-Dimensional Scheduling

Distribute loops + Interchange loops for $S2$

\[
\begin{align*}
\theta_{S1} &= i \\
\theta_{S2} &= j + n
\end{align*}
\]

- Specify the outer-most loop only
- The outer-most loop for $S2$ becomes $j$
One-Dimensional Scheduling

Distribute loops + Interchange loops for S2

```c
for (i=0; i<n; ++i) {
  . S1(i);
  . for (j=0; j<n; ++j)
    . . S2(i,j);
}
```

```
\[
\begin{align*}
\theta_{S1} &= i \\
\theta_{S2} &= j + n
\end{align*}
\]

Transformation | Description
--- | ---
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
peeling | Extracts one iteration of a given loop
shifting | Allows to reorder loops
fusion | Fuses two loops, a.k.a. jamming
distribution | Splits a single loop nest into many, a.k.a. fission or splitting
One-Dimensional Scheduling

A schedule is an affine function of the iteration vector and the parameters

\[
\begin{align*}
\theta_{S1}(\vec{x}_{S1}) &= t_{1s1} \cdot i_{S1} + t_{2s1} \cdot n + t_{3s1} \cdot 1 \\
\theta_{S2}(\vec{x}_{S2}) &= t_{1s2} \cdot i_{S2} + t_{2s2} \cdot j_{S2} + t_{3s2} \cdot n + t_{4s2} \cdot 1
\end{align*}
\]
One-Dimensional Scheduling

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

A schedule is an affine function of the iteration vector and the parameters

\[
\theta_{S1}(\vec{x}_{S1}) = t_{1s1} \cdot i_{S1} + t_{2s1} \cdot n + t_{3s1} \cdot 1 \\
\theta_{S2}(\vec{x}_{S2}) = t_{1s2} \cdot i_{S2} + t_{2s2} \cdot j_{S2} + t_{3s2} \cdot n + t_{4s2} \cdot 1
\]

For \(-1 \leq t \leq 1\), there are \(3^7 = 2187\) possible schedules
One-Dimensional Scheduling

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

▶ A schedule is an affine function of the iteration vector and the parameters

\[ \theta_{S1}(\vec{x}_{S1}) = t_{1S1} \cdot i_{S1} + t_{2S1} \cdot n + t_{3S1} \cdot 1 \]
\[ \theta_{S2}(\vec{x}_{S2}) = t_{1S2} \cdot i_{S2} + t_{2S2} \cdot j_{S2} + t_{3S2} \cdot n + t_{4S2} \cdot 1 \]

▶ For \(-1 \leq t \leq 1\), there are \(3^7 = 2187\) possible schedules
▶ But only 129 legal distinct schedules
Our Objective

1. Search space construction
   - Efficiently construct a space of all legal, distinct affine schedules
Our Objective

Search space construction

- **Efficiently** construct a space of *all legal, distinct* affine schedules

<table>
<thead>
<tr>
<th></th>
<th>matmult</th>
<th>locality</th>
<th>fir</th>
<th>h264</th>
<th>crout</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$-Bounds</td>
<td>$[-1,1]$</td>
<td>$[-1,1]$</td>
<td>$[0,1]$</td>
<td>$[-1,1]$</td>
<td>$[-3,3]$</td>
</tr>
<tr>
<td>$c$-Bounds</td>
<td>$[-1,1]$</td>
<td>$[-1,1]$</td>
<td>$[0,3]$</td>
<td>$[0,4]$</td>
<td>$[-3,3]$</td>
</tr>
<tr>
<td>#Sched.</td>
<td>$1.9 \times 10^4$</td>
<td>$5.9 \times 10^4$</td>
<td>$1.2 \times 10^7$</td>
<td>$1.8 \times 10^8$</td>
<td>$2.6 \times 10^{15}$</td>
</tr>
</tbody>
</table>
Our Objective

1. Search space construction

   - Efficiently construct a space of all legal, distinct affine schedules

<table>
<thead>
<tr>
<th></th>
<th>matmult</th>
<th>locality</th>
<th>fir</th>
<th>h264</th>
<th>crout</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i</strong>-Bounds</td>
<td>−1,1</td>
<td>−1,1</td>
<td>0,1</td>
<td>−1,1</td>
<td>−3,3</td>
</tr>
<tr>
<td><strong>c</strong>-Bounds</td>
<td>−1,1</td>
<td>−1,1</td>
<td>0,3</td>
<td>0,4</td>
<td>−3,3</td>
</tr>
<tr>
<td><strong>#Sched.</strong></td>
<td>$1.9 \times 10^4$</td>
<td>$5.9 \times 10^4$</td>
<td>$1.2 \times 10^7$</td>
<td>$1.8 \times 10^8$</td>
<td>$2.6 \times 10^{15}$</td>
</tr>
</tbody>
</table>

| **#Legal** | 6561 | 912 | 792  | 360   | 798   |
Our Objective

Search space construction

- **Efficiently** construct a space of all legal, distinct affine schedules

<table>
<thead>
<tr>
<th></th>
<th>matmult</th>
<th>locality</th>
<th>fir</th>
<th>h264</th>
<th>crout</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$-Bounds</td>
<td>$-1,1$</td>
<td>$-1,1$</td>
<td>$0,1$</td>
<td>$-1,1$</td>
<td>$-3,3$</td>
</tr>
<tr>
<td>$c$-Bounds</td>
<td>$-1,1$</td>
<td>$-1,1$</td>
<td>$0,3$</td>
<td>$0,4$</td>
<td>$-3,3$</td>
</tr>
<tr>
<td>#Sched.</td>
<td>$1.9 \times 10^4$</td>
<td>$5.9 \times 10^4$</td>
<td>$1.2 \times 10^7$</td>
<td>$1.8 \times 10^8$</td>
<td>$2.6 \times 10^{15}$</td>
</tr>
</tbody>
</table>

- Rely on the **polyhedral model** and Integer Linear Programming to guarantee completeness and correctness of the space properties
Our Objective

Search space construction

- **Efficiently** construct a space of all legal, distinct affine schedules

<table>
<thead>
<tr>
<th></th>
<th>matmult</th>
<th>locality</th>
<th>fir</th>
<th>h264</th>
<th>crout</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$-Bounds</td>
<td>$[-1, 1]$</td>
<td>$[-1, 1]$</td>
<td>$[0, 1]$</td>
<td>$[-1, 1]$</td>
<td>$[-3, 3]$</td>
</tr>
<tr>
<td>$c$-Bounds</td>
<td>$[-1, 1]$</td>
<td>$[-1, 1]$</td>
<td>$[0, 3]$</td>
<td>$[0, 4]$</td>
<td>$[-3, 3]$</td>
</tr>
<tr>
<td>#Sched.</td>
<td>$1.9 \times 10^4$</td>
<td>$5.9 \times 10^4$</td>
<td>$1.2 \times 10^7$</td>
<td>$1.8 \times 10^8$</td>
<td>$2.6 \times 10^{15}$</td>
</tr>
</tbody>
</table>

- Rely on the polyhedral model and Integer Linear Programming to guarantee completeness and correctness of the space properties
- Search space will encompass unique, distinct compositions of reversal, skewing, interchange, fusion, peeling, shifting, distribution
Our Objective

1. Search space construction
   - **Efficiently** construct a space of all legal, distinct affine schedules

<table>
<thead>
<tr>
<th></th>
<th>matmult</th>
<th>locality</th>
<th>fir</th>
<th>h264</th>
<th>crout</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tilde{z})-Bounds</td>
<td>(-1,1)</td>
<td>(-1,1)</td>
<td>0,1</td>
<td>(-1,1)</td>
<td>(-3,3)</td>
</tr>
<tr>
<td>(c)-Bounds</td>
<td>(-1,1)</td>
<td>(-1,1)</td>
<td>0,3</td>
<td>0,4</td>
<td>(-3,3)</td>
</tr>
<tr>
<td>#Sched.</td>
<td>(1.9 \times 10^4)</td>
<td>(5.9 \times 10^4)</td>
<td>(1.2 \times 10^7)</td>
<td>(1.8 \times 10^8)</td>
<td>(2.6 \times 10^{15})</td>
</tr>
</tbody>
</table>

   - Rely on the **polyhedral model** and Integer Linear Programming to guarantee completeness and correctness of the space properties
   - Search space will encompass unique, distinct compositions of reversal, skewing, interchange, fusion, peeling, shifting, distribution

2. Search space exploration
   - Perform exhaustive scan to discover wall clock optimal schedule, and evidences of intricacy of the best transformation
   - Build an **efficient heuristic** to accelerate the space traversal
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only
- Iteration domain: represented as integer polyhedra

for (\texttt{i=1; i<=n; ++i})
  . for (\texttt{j=1; j<=n; ++j})
  . . if (\texttt{i<=n-j+2})
  . . . \texttt{s[i] = ...}

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

Static Control Parts

- Loops have affine control only
- Iteration domain: represented as integer polyhedra
- Memory accesses: static references, represented as affine functions of $\vec{x}_S$ and $\vec{p}$

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

\[
\begin{align*}
    f_s(\vec{x}_{S2}) &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_{S2} \\ n \\ 1 \end{bmatrix} \\
    f_a(\vec{x}_{S2}) &= \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_{S2} \\ n \\ 1 \end{bmatrix} \\
    f_x(\vec{x}_{S2}) &= \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_{S2} \\ n \\ 1 \end{bmatrix}
\end{align*}
\]
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only
- 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;
}
```
Polyhedral Representation of Programs

Static Control Parts

- Loops have affine control only
- Iteration domain: represented as integer polyhedra
- Memory accesses: static references, represented as affine functions of $x_s^i$ and $p$
- Data dependence between S1 and S2: a subset of the Cartesian product of $D_{S1}$ and $D_{S2}$ (exact analysis)
- Reduced dependence graph labeled by dependence polyhedra
Space Construction

Affine Schedules

Legal Distinct Schedules
Space Construction

Property (Causality condition for schedules)

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

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

Equivalently: $\Delta_{R,S} = \theta_S(x_S) - \theta_R(x_R) - 1 \geq 0$
Lemma (Affine form of Farkas lemma)

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

$$ f(\vec{x}) = \lambda_0 + \vec{\lambda}^T (A\vec{x} + \vec{b}) $$

with $\lambda_0 \geq 0$ and $\vec{\lambda} \geq \vec{0}$.

$\lambda_0$ and $\vec{\lambda}^T$ are called the Farkas multipliers.
Space Construction

- Causality condition
- Farkas Lemma
Space Construction

- Affine Schedules
  - Causality condition
  - Farkas Lemma
- Valid Farkas Multipliers
- Many to one
- Legal Distinct Schedules
Space Construction

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

\[
\begin{align*}
D_{R\delta S} & \quad i_R : \\
i_S & \quad : \\
j_S & \quad : \\
n & \quad : \\
1 & \quad :
\end{align*}
\]

\[ \begin{align*}
\lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\
-\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\
\lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\
\lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\
\lambda_{D_{1,0}}
\end{align*} \]
Space Construction

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

\[
\begin{align*}
D_{R\delta S} & : \quad -t_{1_R} = \lambda_{D_{1,1}} - \lambda_{D_{1,2}} + \lambda_{D_{1,3}} - \lambda_{D_{1,4}} \\
i_R & : \quad t_{1_S} = -\lambda_{D_{1,1}} + \lambda_{D_{1,2}} + \lambda_{D_{1,5}} - \lambda_{D_{1,6}} \\
i_S & : \quad t_{2_S} = \lambda_{D_{1,7}} - \lambda_{D_{1,8}} \\
j_S & : \quad t_{3_S} - t_{2_R} = \lambda_{D_{1,4}} + \lambda_{D_{1,6}} + \lambda_{D_{1,8}} \\
j & : \quad t_{4_S} - t_{3_R} - 1 = \lambda_{D_{1,0}}
\end{align*}
\]
Space Construction

- Affine Schedules
- Valid Farkas Multipliers
- Legal Distinct Schedules
- Causality condition
- Farkas Lemma
- Identification
- Projection

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

- Causality condition
- Farkas Lemma
- Identification
- Projection
Space Construction

- Causality condition
- Farkas Lemma
- Identification
- Projection

▶ One point in the space $\Leftrightarrow$ one set of legal schedules w.r.t. the dependence
Overview

Algorithm

- Add constraints obtained for each dependence
- Bound the space
- Search space: set of linear constraints on the schedule coefficients (i.e. \(\mathbb{Z}\)-polytope)

- To each integral point in the space corresponds a distinct program version where the semantics is preserved

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>(\tilde{i})-Bounds</th>
<th>#Sched</th>
<th>#Legal</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>matmult</td>
<td>(-1, 1)</td>
<td>(1.9 \times 10^4)</td>
<td>912</td>
<td>0.029</td>
</tr>
<tr>
<td>locality</td>
<td>(-1, 1)</td>
<td>(5.9 \times 10^4)</td>
<td>6561</td>
<td>0.022</td>
</tr>
<tr>
<td>fir</td>
<td>(0, 1)</td>
<td>(1.2 \times 10^7)</td>
<td>792</td>
<td>0.047</td>
</tr>
<tr>
<td>h264</td>
<td>(-1, 1)</td>
<td>(1.8 \times 10^8)</td>
<td>360</td>
<td>0.024</td>
</tr>
<tr>
<td>crout</td>
<td>(-3, 3)</td>
<td>(2.6 \times 10^{15})</td>
<td>798</td>
<td>0.046</td>
</tr>
</tbody>
</table>
**Workflow**

- **Polyhedral computing libraries**
  - PIPLib
  - PolyLib

- **Code generation**
  - CLooG

- **Iterative compilation and run of base source code with transformed SCoP**

- **SCoP representation**

- **Source Code**
  - Static Analysis
  - Space Construction
  - Space Exploration
  - Kernel Generation
  - Unit Generation
  - Compilation

- **Feedback from hardware counter(s)**

- **Target Code**

- **Polyhedral representation of SCoP**

- **Bounded search space**

- **PolyLib**: http://icps.u-strasbg.fr/polylib
- **CLooG**: http://www.cloog.org
- **PiPLib**: http://www.piplib.org
Performance Distribution [1/2]

Figure: Performance distribution for matmult and locality
Performance Distribution [2/2]

(a) GCC -O3

(b) ICC -fast

Figure: The effect of the compiler
Performance Comparison

Figure: Best Version vs Original
Heuristic Scan

Propose a decoupling heuristic:

- The general “form” of the schedule is embedded in the iterator coefficients

- Decouple the schedule: \( \theta_S(\vec{x}_S) = (\vec{i} \vec{p} \ c) \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix} \)

Parameters and constant coefficients can be seen as a refinement.

Addressing scalability to larger SCoPs:

1. Impose a static or dynamic limit to the number of runs (limit to the \( \vec{i} \) part).
2. Replace an exhaustive enumeration of the \( \vec{i} \) combinations by a limited set of random draws in the \( \vec{i} \) space.
Propose a decoupling heuristic:

- The general “form” of the schedule is embedded in the iterator coefficients

- Decouple the schedule: \( \theta_S(\vec{x}_S) = (\vec{i} \ \vec{p} \ c) \left( \begin{array}{c} \vec{x}_S \\ \vec{n} \\ 1 \end{array} \right) \)

- Parameters and constant coefficients can be seen as a refinement
Heuristic Scan

Propose a decoupling heuristic:

- The general “form” of the schedule is embedded in the iterator coefficients
- Decouple the schedule: \( \theta_S(\vec{x}_S) = (\vec{i} \vec{p} c) \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix} \)
- Parameters and constant coefficients can be seen as a refinement

Adressing scalability to larger SCoPs:

1. impose a static or dynamic limit to the number of runs (limit to the \( \vec{i} \) part)
2. replace an exhaustive enumeration of the \( \vec{i} \) combinations by a limited set of random draws in the \( \vec{i} \) space.
Results

Figure: Comparison between random and decoupling heuristics
Conclusion

- Optimizing and / or Enabling transformation framework on top of the compiler
- Encouraging speedups, fast heuristic convergence
- On small kernels, optimal transformation can be discovered
Conclusion

- Optimizing and / or Enabling transformation framework on top of the compiler
- Encouraging speedups, fast heuristic convergence
- On small kernels, optimal transformation can be discovered
Conclusion

- Optimizing and/or Enabling transformation framework on top of the compiler
- Encouraging speedups, fast heuristic convergence
- On small kernels, optimal transformation can be discovered
Conclusion

- Optimizing and/or Enabling transformation framework on top of the compiler
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Ongoing and future work:

- Couple with state-of-the-art feedback-directed iterative methods
  - Part II: multidimensional schedules
- Integrate into GCC GRAPHITE branch
Conclusion

- Optimizing and / or Enabling transformation framework on top of the compiler
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Ongoing and future work:

- Couple with state-of-the-art feedback-directed iterative methods
- Part II: multidimensional schedules
- Integrate into GCC GRAPHITE branch
Conclusion

- Optimizing and/or Enabling transformation framework on top of the compiler
- Encouraging speedups, fast heuristic convergence
- On small kernels, **optimal transformation** can be discovered

Ongoing and future work:
- Couple with state-of-the-art feedback-directed iterative methods
- Part II: multidimensional schedules
- Integrate into GCC GRAPHITE branch
# Intricacy of the Transformed Code

## Optimal Transformation for **locality**, GCC 4 -O3, P4 Xeon

| S1: B[j] = A[j] | for (c1=-N; c1<=min(-2, M-N); c1++)
|----------------|-----------------------------------
| S2: C[j] = A[j + N] | for (j=0; j<=M; j++)
|                | S1(c1+N, j);
|                | for (c1=-1; c1<=M-N; c1++) {
|                | for (j=0; j<=M; j++)
|                | S2(c1+1, j);
|                | for (j=0; j<=M; j++)
|                | S1(c1+N, j);
|                | }
|                | for (c1=max(M-N+1,-1); c1<=M-1; c1++)
|                | for (j=0; j<=M; j++)
|                | S2(c1+1, j); |

→ 19.4% speedup, without vectorization