Iterative Optimization in the Polyhedral Model: Part I, One-Dimensional Time

Louis-Noël Pouchet, Cédric Bastoul, Albert Cohen and Nicolas Vasilache

ALCHEMY, INRIA Futurs / University of Paris-Sud XI

March 12, 2007

Fifth International Symposium on Code Generation and Optimization
San Jose, California
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 ⇔ 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

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

\[
\begin{align*}
\text{for (i=0; i<n; ++i) } & \begin{cases} 
\theta_{S_1} = i \\
\theta_{S_2} = j + n
\end{cases} \\
& \text{for (i=0; i<n; ++i) } \\
& \begin{cases} 
\theta_{S_1} = i \\
\text{for (j=n; j<2\times n; ++j) } \\
\text{for (i=0; i<n; ++i) } \\
\text{. . S2(i,j-n); }
\end{cases}
\end{align*}
\]

- Specify the outer-most loop only
- The outer-most loop for S2 becomes \( j \)
One-Dimensional Scheduling

Distribute loops + Interchange loops for S2

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

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

for (i=0; i<n; ++i) {
  . S1(i);
  . for (j=n; j<2*n; ++j)
    . . S2(i,j-n);
}

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reversal</td>
<td>Changes the direction in which a loop traverses its iteration range</td>
</tr>
<tr>
<td>skewing</td>
<td>Makes the bounds of a given loop depend on an outer loop counter</td>
</tr>
<tr>
<td>interchange</td>
<td>Exchanges two loops in a perfectly nested loop, a.k.a. permutation</td>
</tr>
<tr>
<td>peeling</td>
<td>Extracts one iteration of a given loop</td>
</tr>
<tr>
<td>shifting</td>
<td>Allows to reorder loops</td>
</tr>
<tr>
<td>fusion</td>
<td>Fuses two loops, a.k.a. jamming</td>
</tr>
<tr>
<td>distribution</td>
<td>Splits a single loop nest into many, a.k.a. fission or splitting</td>
</tr>
</tbody>
</table>
One-Dimensional Scheduling

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

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

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

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

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

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

\[
\theta_{S_1}(\vec{x}_{S_1}) = t_{1_{S_1}}i_{S_1} + t_{2_{S_1}}n + t_{3_{S_1}}1
\]

\[
\theta_{S_2}(\vec{x}_{S_2}) = t_{1_{S_2}}i_{S_2} + t_{2_{S_2}}j_{S_2} + t_{3_{S_2}}n + t_{4_{S_2}}1
\]

▶ 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>(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>
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>
<tr>
<td>#Legal</td>
<td>6561</td>
<td>912</td>
<td>792</td>
<td>360</td>
<td>798</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>( 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>

\[\downarrow\]

| #Legal | 6561 | 912 | 792 | 360 | 798 |

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

   \[\downarrow\]

   | #Legal | 6561 | 912 | 792 | 360 | 798 |

   - 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
     
     |         | matmult | locality | fir   | h264  | crout |
     |---------|---------|----------|-------|-------|-------|
     | $i$-Bounds | $-1,1$ | $-1,1$  | $0,1$ | $-1,1$ | $-3,3$ |
     | $c$-Bounds | $-1,1$ | $-1,1$  | $0,3$ | $0,4$  | $-3,3$ |
     | #Sched.   | $1.9 \times 10^4$ | $5.9 \times 10^4$ | $1.2 \times 10^7$ | $1.8 \times 10^8$ | $2.6 \times 10^{15}$ |

     - 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

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

\[ \mathcal{P}_{S1} = \begin{bmatrix}
1 & 0 & 0 & -1 \\
-1 & 0 & 1 & 0 \\
0 & 1 & 0 & -1 \\
-1 & 0 & 1 & 0 \\
-1 & -1 & 1 & 2
\end{bmatrix} \cdot \begin{bmatrix} i \\ j \\ n \\ 1 \end{bmatrix} \geq 0 \]
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 $\mathbf{x}_S$ and $\mathbf{p}$

```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];
}
```

\[
f_s(\mathbf{x}_{S2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} x_{S2} \n \end{pmatrix}
\]

\[
f_a(\mathbf{x}_{S2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} x_{S2} \n \end{pmatrix}
\]

\[
f_x(\mathbf{x}_{S2}) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{pmatrix} x_{S2} \n \end{pmatrix}
\]
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 $S_1$ and $S_2$: a subset of the Cartesian product of $D_{S_1}$ and $D_{S_2}$ (exact analysis)

```c
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 $\vec{x}_S$ and $\vec{p}$
- Data dependence between $S_1$ and $S_2$: a subset of the Cartesian product of $D_{S_1}$ and $D_{S_2}$ (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
- Valid Farkas Multipliers
  - Causality condition
  - Farkas Lemma
- Legal Distinct Schedules
  - Many to one
Space Construction

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

\[
\begin{cases}
D_{R\delta S} \quad \text{i}_R : \\
\quad \text{i}_S : \\
\quad \text{j}_S : \\
\quad \text{n} : \\
\quad \text{1} : \\
\end{cases}
\begin{align*}
\lambda_{D,1,1} - \lambda_{D,1,2} + \lambda_{D,1,7} \\
\lambda_{D,1,3} - \lambda_{D,1,4} - \lambda_{D,1,7} \\
\lambda_{D,1,5} - \lambda_{D,1,6} \\
\lambda_{D,1,2} + \lambda_{D,1,4} + \lambda_{D,1,6} \\
\lambda_{D,1,0}
\end{align*}
\]
**Space Construction**

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

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

- Causality condition
- Farkas Lemma
- Identification
- Projection

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

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

- Affine Schedules
  - Causality condition
  - Farkas Lemma

- Valid Farkas Multipliers
  - Identification
  - Projection

- Valid Transformation Coefficients

- Legal Distinct Schedules

▶ One point in the space ⇔ 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>$\bar{i}$-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>$1.2 \times 10^4$</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>$2.6 \times 10^{13}$</td>
<td>798</td>
<td>0.046</td>
</tr>
</tbody>
</table>
Workflow

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

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

Figure: The effect of the compiler

(a) GCC -O3
(b) ICC -fast
Performance Comparison

**Figure:** Best Version vs Original
Heuristics Scan

Propose a decoupling heuristic:

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

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

Propose a decoupling heuristic:

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

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

- 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 \( \bar{t} \) part)
2. replace an exhaustive enumeration of the \( \bar{t} \) combinations by a limited set of random draws in the \( \bar{t} \) 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

<table>
<thead>
<tr>
<th>S1: B[j] = A[j]</th>
<th>for (c1=-N;c1&lt;=min(-2,M-N);c1++)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2: C[j] = A[j + N]</td>
<td>for (j=0;j&lt;=M;j++)</td>
</tr>
<tr>
<td></td>
<td>S1(c1+N,j);</td>
</tr>
<tr>
<td></td>
<td>for (c1=-1;c1&lt;=M-N;c1++) {</td>
</tr>
<tr>
<td></td>
<td>for (j=0;j&lt;=M;j++)</td>
</tr>
<tr>
<td></td>
<td>S2(c1+1,j);</td>
</tr>
<tr>
<td>for (i=0;i&lt;=M;i++) {</td>
<td>for (c1=max(M-N+1,-1);c1&lt;=M-1;c1++)</td>
</tr>
<tr>
<td>for (j=0;j&lt;=M;j++) {</td>
<td>for (j=0;j&lt;=M-1;j++)</td>
</tr>
<tr>
<td>S1(i,j);</td>
<td>S2(c1+1,j);</td>
</tr>
<tr>
<td>S2(i,j);</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td></td>
</tr>
</tbody>
</table>

→ 19.4% speedup, without vectorization