A Note on the Performance Distribution of Affine Schedules

Louis-Noël Pouchet¹, Cédric Bastoul¹, John Cavazos² and Albert Cohen¹

¹ALCHEMY, INRIA Futurs / University of Paris-Sud XI, France
²Computer and Information Sciences, University of Delaware, USA

January 27, 2008

2nd Workshop on Statistical and Machine learning approaches to ARchitectures and compilaTion
Göteborg, Sweden
Outline

Motivation

- Automatic performance portability: iterative compilation
- Search space expressiveness \implies bring the iterative optimization problem into the polyhedral model

- Tradeoff expressiveness / traversal easiness
  - Improve static characterization of the search space
  - Highlight dynamic properties
  - Validate a dedicated heuristic to traverse the space
The Model

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];
    }

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

θ^{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} 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 = 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)
The Model

Original Schedule

```
for (i = 0; i < n; ++i) {
    for (j = 0; j < n; ++j) {
        for (k = 0; k < n; ++k) {
            C[i][j] += A[i][k] * B[k][j];
        }
        S1: C[i][j] = 0;
    }
    S2: C[i][j] += A[i][k] * B[k][j];
}
```

\[
\begin{align*}
\Theta^{S1} \cdot \bar{x}_{S1} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ n \\ 1 \end{pmatrix} \\
\Theta^{S2} \cdot \bar{x}_{S2} &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ k \\ n \end{pmatrix}
\end{align*}
\]

- 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)
The Model

Original Schedule

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

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

- 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)
The Model

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];

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

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

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

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

▶ All instances of S1 are executed before the first S2 instance
The Model

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];

The outer-most loop for S2 becomes \( k \)
The Model

Illegal schedule

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

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 after the last S2 instance
The Model

A legal schedule

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

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

for (i = 0; i < n; ++i)
for (j = 0; j < n; ++j)
for (k = 0; k < n; ++k)
\[S_1: C[i][j] = 0;\]
\[S_2: C[i][j] += A[i][k] \ast B[k][j];\]

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

Delay the S2 instances

Constraints must be expressed between \(\Theta^{S_1}\) and \(\Theta^{S_2}\)
The Model

Implicit fine-grain parallelism

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

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

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];

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

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

Representing a schedule

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

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

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

Representing a schedule

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

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

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

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];
The Model

Representing a schedule

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

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

for \( i = 0; i < n; ++i \)
for \( j = 0; j < n; ++j \)
\( S1: C[i][j] = 0; \)
for \( k = 0; k < n; ++k \)
\( S2: C[i][j] += A[i][k] \cdot B[k][j]; \)

for \( i = n; i < 2 \cdot n; ++i \)
for \( j = 0; j < n; ++j \)
\( C[i][j] = 0; \)
for \( k = n + 1; k <= 2 \cdot n; ++k \)
for \( j = 0; j < n; ++j \)
\( C[i][j] += A[i][k-n-1] \cdot B[k-n-1][j]; \)

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \bar{i} )</td>
<td>reversal</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>( \bar{p} )</td>
<td>fusion</td>
</tr>
<tr>
<td>distribution</td>
<td>Splits a single loop nest into many, a.k.a. fission or splitting</td>
</tr>
<tr>
<td>( c )</td>
<td>peeling</td>
</tr>
<tr>
<td>shifting</td>
<td>Allows to reorder loops</td>
</tr>
</tbody>
</table>
The Search Space

Challenges
- Completeness (combinatorial problem)
- Scalability (large integer polyhedra computation)

Proposed solution
- Philosophically close to Feautrier’s maximal fine-grain parallelism
- One point in the space $\iff$ one distinct legal program version
- Bound schedule coefficients in $[-1, 1]$ to limit control overhead
- No completeness, but decent scalability
- Deliver a mechanism to automatically complete / correct schedules
The Hypothesis

Extremely large generated spaces: $> 10^{30}$ points

→ we must leverage static characteristics to build traversal mechanisms

Hypothesis:

- 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

- The more a schedule dimension impacts a performance distribution, the more it is constrained
**DCT benchmark**

▶ 32x32 Discrete Cosine Transform, 5 statements, 35 dependences
▶ 2 imperfectly nested loops
▶ 3 sequential schedule dimensions outputted

<table>
<thead>
<tr>
<th>Schedule dimension</th>
<th>$\vec{i}$</th>
<th>$\vec{i} + \vec{p}$</th>
<th>$\vec{i} + \vec{p} + c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension 1</td>
<td>39</td>
<td>66</td>
<td>471</td>
</tr>
<tr>
<td>Dimension 2</td>
<td>729</td>
<td>19683</td>
<td>531441</td>
</tr>
<tr>
<td>Dimension 3</td>
<td>60750</td>
<td>1006020</td>
<td>64855485</td>
</tr>
<tr>
<td>Total combined</td>
<td>$1.7 \times 10^9$</td>
<td>$1.3 \times 10^{12}$</td>
<td>$1.6 \times 10^{16}$</td>
</tr>
</tbody>
</table>

**Figure:** Search Space Statistics for *dct*
**DCT benchmark**

- 32x32 Discrete Cosine Transform, 5 statements, 35 dependences
- 2 imperfectly nested loops
- 3 sequential schedule dimensions outputted

<table>
<thead>
<tr>
<th>Schedule dimension</th>
<th>$\vec{i}$</th>
<th>$\vec{i} + \vec{p}$</th>
<th>$\vec{i} + \vec{p} + c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension 1</td>
<td>39</td>
<td>66</td>
<td>471</td>
</tr>
<tr>
<td>Dimension 2</td>
<td>729</td>
<td>19683</td>
<td>531441</td>
</tr>
<tr>
<td>Dimension 3</td>
<td>60750</td>
<td>1006020</td>
<td>64855485</td>
</tr>
<tr>
<td>Total combined</td>
<td>$1.7 \times 10^9$</td>
<td>$1.3 \times 10^{12}$</td>
<td>$1.6 \times 10^{16}$</td>
</tr>
</tbody>
</table>

**Figure:** Search Space Statistics for dct

- Search space analyzed: $66 \times 19683 = 1.29 \times 10^6$ different legal program versions (arbitrary compositions of skewing, reversal, interchange, fusion, distribution)
**Performance Distribution [1/2]**

(a) Representatives for each point of $\Theta_1$  
(b) Raw performance of each point of $\Theta_2$, for the best value for $\Theta_1$

**Figure:** Performance Distribution for DCT

- Only 0.14% of analyzed points achieve at least 80% of the speedup
- $\Theta_1$ is a good discriminant for performance
- Variance analysis shows $\bar{i} > \bar{p} > \bar{c}$
Performance Distribution [2/2]

- **L1 Accesses** captures the performance distribution shape
- **Branch count** shows control overhead introduced
- **Origin of performance improvement is opaque most of the time**
  - Interaction with the compiler (trigger optimizations)
  - Better use of processor features

**Figure**: Hardware Counters Distribution for DCT
Search Space Statistics

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># St.</th>
<th># Deps.</th>
<th># Dim.</th>
<th>$\bar{i}$</th>
<th>$\bar{i} + \bar{p}$</th>
<th>$\bar{i} + \bar{p} + c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>latnrm</td>
<td>11</td>
<td>75</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>fir</td>
<td>4</td>
<td>36</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>lmsfir</td>
<td>9</td>
<td>112</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>iir</td>
<td>8</td>
<td>66</td>
<td>3</td>
<td>1</td>
<td>9</td>
<td>18</td>
</tr>
</tbody>
</table>

Figure: Search Space Statistics

- Only one sequence of interchange + skewing + reversal possible for the outer-most loop
- Highly constrained benchmark: side effect of the search space construction algorithm
- Search space must be computed to detect the pattern
Performance Distribution

Performance distribution - IIR
Performance distribution - LMSFIR
Performance distribution - LATNRM

(a) iir
(b) lmsfir
(c) latnrm

Figure: Performance Distribution for 3 UTDSP benchmarks

- Significant speedup to discover
- Performance distribution is almost flat
- Final variance analysis confirm the base hypothesis
Results of the Decoupling Heuristic

- Capitalize on the performance distribution ordering: propose a decoupling heuristic mechanism
- Principle: Iterate first on the most performance impacting coefficients, use a completion algorithm for the non-explored coefficients

<table>
<thead>
<tr>
<th></th>
<th>dct</th>
<th>matmult</th>
<th>lpc</th>
<th>edge-c2d</th>
<th>iir</th>
<th>fir</th>
<th>lmsfir</th>
<th>latnrm</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Inst.</td>
<td>5</td>
<td>2</td>
<td>12</td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>#Loops</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>i</td>
<td>39</td>
<td>76</td>
<td>243</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Space</td>
<td>$1.6 \times 10^{16}$</td>
<td>912</td>
<td>$&gt;10^{25}$</td>
<td>$5.6 \times 10^{15}$</td>
<td>$&gt;10^{19}$</td>
<td>$9.5 \times 10^{7}$</td>
<td>$2.8 \times 10^{8}$</td>
<td>$&gt;10^{22}$</td>
</tr>
<tr>
<td>Id Best</td>
<td>46</td>
<td>16</td>
<td>489</td>
<td>11</td>
<td>34</td>
<td>33</td>
<td>51</td>
<td>6</td>
</tr>
<tr>
<td>Speedup</td>
<td>57.1%</td>
<td>42.87%</td>
<td>31.15%</td>
<td>5.58%</td>
<td>37.50%</td>
<td>40.24%</td>
<td>30.98%</td>
<td>15.11%</td>
</tr>
</tbody>
</table>

Figure: Heuristic Performance for AMD Athlon

- Near space optimal speedup discovered in at most 51 runs for SCoPs of less than 10 statements
Conclusion

Properties of the search space

- "Classical" transformations usually associated to specific schedule coefficients
- Classes of schedule coefficients \((\vec{i}, \vec{p}, c)\) map into subspaces ordered w.r.t performance variation
- Schedule rows map into subspaces ordered w.r.t. performance
- Very low density of the best transformations (0.xx%)

Application

- **Partition the optimization space to narrow the search**
- Motivate a heuristic traversal leveraging these characteristics
- Validated on Intel x86_32, AMD x86_64, embedded MIPS32 (Au1500), embedded VLIW (ST231)
Ongoing Work

- **Scalability** Use genetic algorithm traversal for the larger SCoPs
  - Legality preserving operators

- **Expressiveness** Integrate tiling by means of permutability constraints
  - New (static/dynamic) properties of the search space

- **Parallelism** Express coarse-grain parallelism thanks to tiling
  - New search algorithm