Putting Polyhedral Loop Transformations to Work

Unified Model and Compiler Interface

Albert Cohen

with Cédric Bastoul, Sylvain Girbal, Saurabh Sharma, Olivier Temam

A3 Group
Whole program optimization for peak performance

- Uniprocessor
- OpenMP

Iterative, feedback-directed optimization

1. Implement the *useful* transformations
2. Choose the transformation *sequence and parameters*
Whole program optimization for peak performance
- Uniprocessor
- OpenMP

Iterative, feedback-directed optimization
1. Implement the *useful* transformations
2. Choose the transformation *sequence and parameters*

This talk: *promote the polytope model as a viable representation and transformation framework for semi-automatic program optimization and parallelization*
Example: Matrix Multiplication

Alpha EV67, dynamic analysis with Compaq’s Alpha simulator
95% of peak performance (Parello and Temam, SuperComputing’02)

<table>
<thead>
<tr>
<th>Transformations</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original ikl j loop nest, Compaq f90 -O2 -unroll 1 -nopipeline</td>
<td>1.00</td>
</tr>
<tr>
<td>Compaq f90 -O5 + KAP</td>
<td>3.37</td>
</tr>
<tr>
<td>3D blocking for L1 and TLB</td>
<td>2.62</td>
</tr>
<tr>
<td>3D + interchange for store queue + unrolling for ILP</td>
<td>3.71</td>
</tr>
<tr>
<td>3D + int + unroll + register blocking</td>
<td>9.90</td>
</tr>
<tr>
<td>3D + int + unroll + reg block + prefetch</td>
<td>10.37</td>
</tr>
<tr>
<td>3D for L1 and L2 + copy for TLB + int + unroll + reg block + prefetch</td>
<td>12.75</td>
</tr>
<tr>
<td>3D + copy + int + unroll + reg block + prefetch + low level opt</td>
<td>13.56</td>
</tr>
</tbody>
</table>
Example of Composition of Transformations

for (i=0; i<1000; i++)
    for (j=0; j<m; j++)
        B[j] = A[i][j] + ...
    for (j=0; j<n; j++)
        ... = B[j] + ...

fuse

for (i=0; i<1000; i++)
    for (j=0; j<max(m, n); j++)
        if (j<m)
            B[j] = A[i][j] + ...
        if (j<n)
            ... = B[j] + ...
Example of Composition of Transformations

\[
\text{for } (i=0; i<1000; i++) \\
\quad \text{for } (j=0; j<\text{max}(m,n); j++) \\
\quad \quad \text{if } (j<m) \\
\quad \quad \quad B[j] = A[i][j] + \ldots \\
\quad \quad \text{if } (j<n) \\
\quad \quad \quad \ldots = B[j] + \ldots \\
\]

\[\rightarrow\text{shift}\]

\[
\text{for } (i=0; i<1000; i++) \\
\quad \text{for } (j=0; j<\text{max}(m,n+1); j++) \\
\quad \quad \text{if } (j<m) \\
\quad \quad \quad B[j] = A[i][j] + \ldots \\
\quad \quad \text{if } (j>0 \land j<=n) \\
\quad \quad \quad \ldots = B[j-1] + \ldots \\
\]
Example of Composition of Transformations

for (i=0; i<1000; i++)
  for (j=0; j<max(m,n+1); j++)
    if (j<m)
      B[j] = A[i][j] + ...
    if (j>0 && j<=n)
      ... = B[j-1] + ...

strip-mine

for (ii=0; ii<1000; ii+=10)
  for (i=ii; i<ii+10; i++)
    for (j=0; j<max(m,n+1); j++)
      if (j<m)
        B[j] = A[i][j] + ...
      if (j>0 && j<=n)
        ... = B[j-1] + ...
Example of Composition of Transformations

for (ii=0; i<1000; i+=10)
  for (i=ii; i<ii+10; i++)
    for (j=0; j<max(m,n+1); j++)
      if (j<m)
        B[j] = A[i][j] + ... 
      if (j>0 && j<=n)
        ... = B[j-1] + ...

prefetch

for (ii=0; i<1000; i+=10)
  for (i=ii; i<ii+10; i++)
    for (j=0; j<max(m,n+1); j++)
      if (j<m)
        if (j%4==0)
          prefetch A[i+1][j]
        B[j] = A[i][j] + ...
      if (j>0 && j<=n)
        ... = B[j-1] + ...

Some Problems With Syntax-Based Approaches

- Control overhead
- Regenerate control structures after each transformation
- Fixed transformation sequence
- Non-local transformations

```c
if (m<n && m%4==0)
    for (ii=0; i<1000; i+=10)
        for (i=ii; i<ii+10; i++)
            for (j=0; j+3<m; j+=4)
                prefetch A[i+1][j]
                B[0] = A[i][j] + ...
                ... = B[j-1] + ...
                B[j] = A[i][j+1] + ...
                ... = B[j] + ...
                B[j] = A[i][j+2] + ...
                ... = B[j+1] + ...
                B[j] = A[i][j+3] + ...
                ... = B[j+2] + ...
            for (j=m; j<n; j++)
                ... = B[j-1] + ...
else if (m<n && m%4==1)
    ...
```
1. Polyhedral Representation
Unified Loop Nest Transformation Framework

- Operated by optimization *and* architecture *experts*
Unified Loop Nest Transformation Framework

⭐ Operated by optimization and architecture experts

⭐ Express any composition of analyses and transformations
Unified Loop Nest Transformation Framework

★ Operated by optimization *and* architecture *experts*

★ Express any *composition* of analyses and transformations

★ Domain-specific, representation of *loop nests*
Operated by optimization *and* architecture *experts*

Express any *composition* of analyses and transformations

Domain-specific, representation of *loop nests*

No *intermediate* translation to syntax-tree
**Static Control Parts (SCoPs)**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{for (i=1; i&lt;3; i++)}</td>
<td></td>
</tr>
<tr>
<td>\textbf{S}_1</td>
<td>...</td>
</tr>
<tr>
<td>\textbf{while (A[j]!=0)}</td>
<td></td>
</tr>
<tr>
<td>\textbf{S}_2</td>
<td>...</td>
</tr>
<tr>
<td>\textbf{for (k=0; k&lt;j; k++)}</td>
<td>\textbf{if (j&gt;=2)}</td>
</tr>
<tr>
<td>\textbf{S}_3</td>
<td>...</td>
</tr>
<tr>
<td>\textbf{S}_4</td>
<td>...</td>
</tr>
<tr>
<td>\textbf{for (p=0; p&lt;6; p++)}</td>
<td></td>
</tr>
<tr>
<td>\textbf{S}_5</td>
<td>...</td>
</tr>
<tr>
<td>\textbf{S}_6</td>
<td>...</td>
</tr>
</tbody>
</table>
## SCoP Coverage (SpecFP)

<table>
<thead>
<tr>
<th></th>
<th>SCoPs</th>
<th>Statements</th>
<th>Array References</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>Param.</td>
<td>ifs</td>
</tr>
<tr>
<td>applu</td>
<td>19</td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>apsi</td>
<td>80</td>
<td>80</td>
<td>25</td>
</tr>
<tr>
<td>art</td>
<td>28</td>
<td>27</td>
<td>4</td>
</tr>
<tr>
<td>lucas</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>mgrid</td>
<td>12</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>quake</td>
<td>20</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>swim</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>
SCoP Size (SpecFP)

SpecFP: Statement Distribution

Statement Range

SCoPs

0 - 1  2 - 3  4 - 7  8 - 15  16 - 31  32 - 63  64 - 127  128 - 255  256 - 511  512 - 1023  1024 - 2047  > 2048

Legend:
- applu
- api
- art
- lucas
- mgrid
- quake
- swim
SCoP Depth (SpecFP)

SpecFP: Statement Depth

Statements

Statement Depth

0 1 2 3 4 5

applu  apsi  art  lucas  mgrid  quake  swim
SCoP Polyhedral Representation

- Describes each statement *separately*
- Captures *control* and *array access* semantics
  Through *parameterized* affine (in)equalities

1. A *domain*
   The bounds of the enclosing loops

2. A *schedule*
   An affine function assigning logical dates to iterations

3. A list of *access functions*
   To describe array references
A few facts

1. Polytopes are very expressive
   \[ AX \geq 0 \] suffices to characterize all executions of a statement
   (schedule, domain and memory accesses)

2. Affine schedules emerged in automatic parallelization
3. Affine schedules are not popular for optimization
   (too expensive, too restrictive, too general, non intuitive...)

Existing Polyhedral Representations
Existing Polyhedral Representations

A few facts

1. Polytopes are very expressive
   \[ AX \geq 0 \] suffices to characterize all executions of a statement
   (schedule, domain and memory accesses)

2. Affine schedules emerged in automatic parallelization

3. Affine schedules are not popular for optimization
   (too expensive, too restrictive, too general, non intuitive...)

Some common biases

- Schedules are only meant to describe parallelism
- Only one-dimensional schedules are useful
- Schedule and domains can be merged in one matrix
Existing Polyhedral Representations

A few facts

1. Polytopes are very expressive
   \[ AX \geq 0 \] suffices to characterize all executions of a statement
   (schedule, domain and memory accesses)

2. Affine schedules emerged in automatic parallelization

3. Affine schedules are not popular for optimization
   (too expensive, too restrictive, too general, non intuitive...)

Some common biases

- Schedules are only meant to describe parallelism
- Only one-dimensional schedules are useful
- Schedule and domains can be merged in one matrix

We use \textit{separate} matrices and \textit{full}-dimensional \textit{sequential} schedules
Affine Schedule

- Dense, totally ordered (sequential) schedule
- Unimodular matrix for iteration ordering
- Matrix for parameterization and iteration shifting
- Vector for instruction scattering
Affine Schedule

- Dense, totally ordered (sequential) schedule
- Unimodular matrix for iteration ordering ($A$)
- Matrix for parameterization and iteration shifting ($\Gamma$)
- Vector for instruction scattering ($\beta$)

$$\theta(\vec{i}, \vec{q}) = \begin{bmatrix} 0 & \ldots & 0 & 0 & \ldots & 0 & \beta_0 \\ A_{1,1} & \ldots & A_{1,d} & \Gamma_{1,1} & \ldots & \Gamma_{1,g} & \Gamma_{1,g+1} \\ 0 & \ldots & 0 & 0 & \ldots & 0 & \beta_1 \\ A_{2,1} & \ldots & A_{2,d} & \Gamma_{2,1} & \ldots & \Gamma_{2,g} & \Gamma_{2,g+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ A_{d,1} & \ldots & A_{d,d} & \Gamma_{d,1} & \ldots & \Gamma_{d,g} & \Gamma_{d,g+1} \\ 0 & \ldots & 0 & 0 & \ldots & 0 & \beta_d \end{bmatrix} \begin{bmatrix} i_1 \\ \vdots \\ i_d \\ q_1 \\ \vdots \\ q_g \\ 1 \end{bmatrix}$$
Domain and Access Functions

- **Domain matrix**
  - Exact characterization of the valid iteration vectors
  - Parameterized by symbolic constants
Domain and Access Functions

- **Domain matrix**
  Exact characterization of the valid iteration vectors
  Parameterized by symbolic constants

- **Access function**
  Iteration Vector $\mapsto (\text{Array Name}, \text{Vector})$
  Parameterized by symbolic constants
Polyhedral Representation Example

**Access function**

for \( A[2*i][j+1] \)

\[
\begin{bmatrix}
i & j & m & n & 1 \\
2 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 \\
\end{bmatrix}
\]

**2-dimensional domain of \( S_3 \)**

(with parameters \( m \) and \( n \))

\[
\begin{bmatrix}
i & j & m & n & 1 \\
1 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 0 & -1 \\
0 & 1 & 0 & 0 & -5 \\
0 & -1 & 0 & 1 & -1 \\
\end{bmatrix} \geq 0
\]

**5-dimensional schedule for \( S_3 \)**

\((i, j) \mapsto (p_0, p_1, p_2, p_3, p_4)\)

\[
\begin{bmatrix}
p_0 & p_1 & p_2 & p_3 & p_4 & i & j & m & n & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 \\
\end{bmatrix} = 0
\]
2. **Unified Transformation Model**
## Primitives

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Prerequisites</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIGHTU$(S, U)$</td>
<td>$</td>
<td>\det(U)</td>
</tr>
<tr>
<td>SHIFT$(S, M)$</td>
<td></td>
<td>$\Gamma^S \leftarrow \Gamma^S + M$</td>
</tr>
<tr>
<td>FUSE$(P, o)$</td>
<td></td>
<td>$b = \max{\beta^S_{\dim(P)+1} \mid (P, o) \sqsubseteq \beta^S} + 1$; Move($(P, o + 1), (P, o + 1), b)$; Move$(P, (P, o + 1), -1)$</td>
</tr>
</tbody>
</table>
## Composition of Primitives

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Prerequisites</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIGHTU(S, U)</td>
<td></td>
<td>(</td>
</tr>
<tr>
<td>SHIFT(S, M)</td>
<td></td>
<td>(\Gamma^S \leftarrow \Gamma^S + M)</td>
</tr>
<tr>
<td>FUSE(P, o)</td>
<td></td>
<td>(b = \max{\beta_{\dim(P)+1}^S \mid (P, o) \subseteq \beta^S} + 1); (\text{Move}((P, o + 1), (P, o + 1), b);) (\text{Move}(P, (P, o + 1), -1))</td>
</tr>
<tr>
<td>TILE(S, o, k)</td>
<td>(S \in S) (\land o &lt; d^S) (\land k &gt; 0)</td>
<td>(S \leftarrow \text{STRIPMINE}(S, o, k);) (S \leftarrow \text{STRIPMINE}(S, o + 2, k);) (S \leftarrow \text{INTERCHANGE}(S, o + 1))</td>
</tr>
</tbody>
</table>

\(Tiling\)
Transformation Language

- Script “generative” language
  - To produce the implementation of primitives
  - To compose primitives

Benefits:
- Regenerate the syntax tree after the last transformation
- Few ordering constraints
- Complex optimizations, e.g., forward array substitution
- Combined transformations to reduce search space
  Example, “smart” register tiling: strip-mining + privatization for permutability + interchange + array contraction + register promotion
Transformation Language

- Script “generative” language
  - To produce the implementation of primitives
  - To compose primitives

- Benefits
  - Regenerate the syntax tree \textit{after the last transformation}
  - Few ordering constraints
  - Complex optimizations, e.g., forward array substitution
  - Combined transformations to reduce search space

  Example, “smart” register tiling: strip-mining + privatization for permutability + interchange + array contraction + register promotion
3. SOFTWARE TOOLS
Code Generation with CLooG

- Robust version of Quilleré and Rajopadhye’s algorithm
- Parameterized unions of linearly bounded lattices
- Depth recursion with direct optimization of conditionals
- Tradeoff between code expansion and control overhead

```
for (i=1; i<=6; i+=2)
    for (j=1; j<=7-i; j++)
        S1; S2
    for (j=8-i; j<=n; j++)
        S1
for (i=7; i<=n; i+=2)
    for (j=1; j<=n; j++)
        S1
```
Implementation Within Open64/ORC

- **WRaP**: WHIRL Represented as Polyhedra
  - Syntax tree of *static control parts* → tree of polyhedral representations
  - Mapping polytopes to the syntax tree
    - From matrix columns to symbol table entries
    - From abstract arrays to symbol table entries
    - From abstract statements to statement nodes
Implementation Within Open64/ORC

- **WRaP**: WHIRL Represented as Polyhedra
  - Syntax tree of *static control parts* → tree of polyhedral representations
  - Mapping polytopes to the syntax tree
    - From matrix columns to symbol table entries
    - From abstract arrays to symbol table entries
    - From abstract statements to statement nodes

- Enables *whole program* optimization
  - Combined transformations of loops and syntactic expressions may be applied to the whole WRaP
  - Array regions, interprocedural analysis
  - Correctness and compatibility with non-affine sections
1. *Suspend* the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations.
1. **Suspend** the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations

2. **W2P**: recognition of Static Control Parts (SCoPs)
   - Affine loop bounds, conditionals and array subscripts
   - Build polyhedral domains, sequential schedules and array accesses
   - Graceful degradation when all conditions are not met
WRaP-IT: an Open64/ORC Interface-Tool

1. *Suspend* the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations

2. *W2P*: recognition of Static Control Parts (SCoPs)
   - Affine loop bounds, conditionals and array subscripts
   - Build polyhedral domains, sequential schedules and array accesses
   - Graceful degradation when all conditions are not met

3. *URUK*: apply WRaP analyses and transformations
1. **Suspend** the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations

2. **W2P**: recognition of Static Control Parts (SCoPs)
   - Affine loop bounds, conditionals and array subscripts
   - Build polyhedral domains, sequential schedules and array accesses
   - Graceful degradation when all conditions are not met

3. **URUK**: apply WRaP analyses and transformations

4. **WLooG**: code generator (CLooG) with WHIRL output
   - Generate new loops, conditionals and variables
   - Move/duplicate the original statement nodes
1. **Suspend** the WHIRL compilation flow after loop normalization, induction variables, and scalar optimizations

2. **W2P**: recognition of Static Control Parts (SCoPs)
   - Affine loop bounds, conditionals and array subscripts
   - Build polyhedral domains, sequential schedules and array accesses
   - Graceful degradation when all conditions are not met

3. **URUK**: apply WRaP analyses and transformations

4. **WLooG**: code generator (CLooG) with WHIRL output
   - Generate new loops, conditionals and variables
   - Move/duplicate the original statement nodes

5. **Resume** the compilation flow, redoing scalar optimization
Some Related Works

- Codesign and synthesis of specialized coprocessors
  - MMAAlpha
  - PICO

- Analysis and transformation frameworks
  - Omega/Petit
  - PIPS, Polaris, SUIF
  - Stratego

- Generative programming
  - ATLAS (BLAS library generator)
  - FFTW (FFT algorithm customization and optimization)
  - SPIRAL (signal-processing language, customization and optimization)
4. Thank You

http://www-rocq.inria.fr/a3/wrap-it