Code Generation in the Polyhedral Model Is Easier Than You Think (PACT ‘04)

Cédric Bastoul

Laboratoire PRiSM, Université de Versailles Saint Quentin
Introduction

• Polyhedral model has achieved advances in automatic parallelization and optimization

• Complexity of code generation deterrent for using polyhedral model in optimizing compilers
  • Coping generated code size and control overhead
  • Time consuming
  • Current algorithm only cover limited possible transformation
Background and Notations

• Polyhedral Model: correspond to static control programs
  • Control statements are **do** loop with affine bounds and **if** conditional with affine conditions (can be more complex)
  • Affine bounds and conditions depend only on outer loop counters and constant parameters

• SCoP: a minimum set of consecutive statements with static control

```
\begin{align*}
\text{do } i=1, n \\
S1: & \quad x = a(i,i) \\
& \quad \text{do } j=1, i-1 \\
& \quad x = x - a(i,j)^2 \\
S2: & \quad p(i) = 1.0/\sqrt{x} \\
& \quad \text{do } j=i+1, n \\
S3: & \quad x = a(i,j) \\
& \quad \text{do } k=1, i-1 \\
S4: & \quad x = x - a(j,k)*a(i,k) \\
S5: & \quad a(j,i) = x*p(i)
\end{align*}
```

Figure 1. A Cholesky factorization kernel
Background and Notations

• Iteration domain can always be specified by a polyhedron

• Polyhedron: A convex set of points in a lattice, also called $\mathbb{Z}$-polyhedron or lattice-polyhedron

• Scattering function $\theta(x) = Tx + t$: An affine function specifying for each integral point in the iteration domain a new coordinate for the corresponding statement instance
Background and Notations

- Capture sequential execution order with scheduling function using abstract syntax tree (AST)

```
<table>
<thead>
<tr>
<th>do i=1, n</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1: x = a(i,i)</td>
</tr>
<tr>
<td>do j=1, i-1</td>
</tr>
<tr>
<td>S2: x = x - a(i,j)**2</td>
</tr>
<tr>
<td>S3: p(i) = 1.0/sqrt(x)</td>
</tr>
<tr>
<td>do j=i+1, n</td>
</tr>
<tr>
<td>S4: x = a(i,j)</td>
</tr>
<tr>
<td>do k=1, i-1</td>
</tr>
<tr>
<td>S5: x = x - a(j,k)*a(i,k)</td>
</tr>
<tr>
<td>S6: a(j,i) = x*p(i)</td>
</tr>
</tbody>
</table>
```

Figure 1. A Cholesky factorization kernel

Figure 2. AST of the program in Figure 1
Program Transformation

• Central part of polyhedral framework
• Using scattering function to modify source polyhedral into target polyhedral
• New polyhedral containing same points but in a new coordinate system and lexicographic order
Affine Transformation

• Previous work has several limitation on scattering function
  • Unimodular (T matrix must be square and determinant $\pm 1$
  • Or invertible
• Their solution

$$T = \left\{ \left( \begin{array}{c} \frac{\vec{y}}{\vec{x}} \\ \frac{\vec{z}}{\vec{x}} \end{array} \right) \mid \left[ \begin{array}{cc} Id & -T \\ 0 & A \end{array} \right] \left( \begin{array}{c} \vec{y} \\ \vec{z} \end{array} \right) \geq \left( \begin{array}{c} \vec{t} \\ \vec{c} \end{array} \right) \right\}.$$
Rational Transformation

• General shape: \( \theta(\vec{x}) = (T \vec{x} + \vec{t})/d, \)
  - \( d \) is a constant vector

• Their solution: intro auxiliary variable standing for quotient of the division

![Diagram](image)

(a) original polyhedron \( A \vec{x} \geq \vec{c} \)

Figure 4. Rational reordering \( \theta(i) = i/3 + 1 \)
Non-uniform Transformation

• Several transformation per statement

\[ \theta(\bar{x}) = \begin{cases} 
  T_1 \bar{x} + t_1 & \text{if } \bar{x} \in D_1 \\
  T_2 \bar{x} + t_2 & \text{if } \bar{x} \in D_2 \\
  \vdots & \\
  T_n \bar{x} + t_n & \text{if } \bar{x} \in D_n 
\end{cases} \]

• Build partition is trivial when iteration domain is split using affine conditions

• Partition with non-affine criteria is possible
Scanning Polyhedra

• Critical step in the framework, can spoil performance if bad control management occurs
  • Producing redundant conditions
  • Complex loop bounds
  • Unused iterations
  • Code explosion (instruction cache)

• Quilleré et al. method gives best result
  • Guaranteed to avoid redundant control
  • Suffers more limitations
Extended Quilleré et al. Algorithm

• Basic mechanism

List of transformed polyhedral to be scanned \((TS_1, ..., TS_n)\);

\[\text{Context, i.e. set of constraints on the global parameters}\]

\[\text{First dimension } d = 1. \text{ Generating code from AST: the constraint system labelling each node can be directly translated as loop bound and as surrounding conditional respectively}\]

• Their extension: reducing code size without degrading performance and reducing code generation processing time
(a) Initial domains to scan

\[ T_{S1} : \begin{cases} 1 \leq i \leq n \\ j = i \end{cases} \]

\[ T_{S2} : \begin{cases} 1 \leq i \leq n \\ i \leq j \leq n \end{cases} \]

\[ T_{S3} : \begin{cases} 1 \leq i \leq m \\ j = n \end{cases} \]

\[ T_{S3} : \begin{cases} n + 1 \leq i \leq m \\ j = n \end{cases} \]

(b) Projection and separation onto the first dimension

\[ \text{do } i=1, n \]

\[ \text{if } (i=n) \text{ then} \]

\[ S1(j=i) \]

\[ S2(j=i) \]

\[ S3(j=i) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

(c) Recursion on next dimension

\[ \text{do } i=1, n-2 \]

\[ S1(j=i) \]

\[ S2(j=i) \]

\[ S3(j=i) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]

\[ \text{if } (i=n-1) \text{ then} \]

\[ S1(j=i-1) \]

\[ S2(j=i-1) \]

\[ S3(j=i-1) \]

\[ \text{do } j=i+1, n-1 \]

\[ \text{do } i=n+1, m \]
Reducing Code Size

• Cause: Separating polyhedra often results in isolating some points, while is not always necessary

• Solution:
  • Scan node in depth first order and build statements
  • check if a point directly precedes or follows the node in lexicographic ordering
  • if yes, merge using a polyhedral union
  • 464B -> 176B
Complexity Issues

- Main computing: separation into disjoint polyhedral
  - Given a list of $n$ polyhedra, worst case is $O(3^n)$
- Memory usage: allocate memory for each separated domain
- Solution:
  - Using pattern matching for computing
    - At a given depth, domains are often the same (17%) or disjoint (36%)
    - Quick check before polyhedra operation
      - Comparing elements of constraints system, find 75% of equalities
      - Comparing unknowns having fixed value, find 94% of disjoints
    - Improve factor near to 2
  - Prefer more naïve algorithm when high memory consumption
    - Merge polyhedral when intersections are not empty instead of separating
      - Usually far less memory, but less efficient
Experimental Results

<table>
<thead>
<tr>
<th>SCoPs</th>
<th>Code Generation</th>
<th>Robustness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Domains</td>
</tr>
<tr>
<td>applu</td>
<td>25</td>
<td>757</td>
</tr>
<tr>
<td>apsi</td>
<td>109</td>
<td>2192</td>
</tr>
<tr>
<td>art</td>
<td>62</td>
<td>499</td>
</tr>
<tr>
<td>equake</td>
<td>40</td>
<td>639</td>
</tr>
<tr>
<td>lucas</td>
<td>11</td>
<td>2070</td>
</tr>
<tr>
<td>mgrid</td>
<td>12</td>
<td>369</td>
</tr>
<tr>
<td>swim</td>
<td>6</td>
<td>123</td>
</tr>
<tr>
<td>adm</td>
<td>109</td>
<td>2260</td>
</tr>
<tr>
<td>dyfesm</td>
<td>112</td>
<td>1497</td>
</tr>
<tr>
<td>mdg</td>
<td>33</td>
<td>530</td>
</tr>
<tr>
<td>mg3d</td>
<td>63</td>
<td>1442</td>
</tr>
<tr>
<td>qcd</td>
<td>74</td>
<td>819</td>
</tr>
</tbody>
</table>

Not able to deal with all real-life code generations

Biggest SCoP, using 22 minutes and 1GB RAM to be optimally generated
Experimental Results

- CLooG: same performance as the original code
Demo
Conclusion

• Giving a scattering function is another way of specifying a reordering and has several advantages over others, like tile or fuse or skew

• Tools like CLooG have remove the difficulties of using transformation framework

• Ongoing work:
  • Point out most compute intensive parts in the source program
  • Find affine constraints on and between evey SCoP paremeters