Recover Polyhedral Transformations From Polyhedral Schedule

Nelson Lossing, Walid Astaoui, Gianpietro Consolaro, Harenome Razanajato, Zhen Zhang, Denis Barthou
Huawei Technologies France

IMPACT 2024, 14th International Workshop on Polyhedral Compilation Techniques
January 17th, 2024
Munich, Germany
Polyhedral Representation

2D+1 Representation

Example with polyhedral schedule 2D+1 representation of seidel-2d kernel:

```c
static void kernel_seidel_2d(int tsteps,
                           int n,
                           DATA_TYPE POLYBENCH_2D(A,N,N,n,n))
{
    int t, i, j;

    #pragma scop
    for (t = 0; t <= PB_TSTEPS - 1; t++)
        for (i = 1; i <= PB_N - 2; i++)
            for (j = 1; j <= PB_N - 2; j++)
    #pragma endscop
}
```

D corresponds to the number of iterators

$$\phi_{s_0} = [0 \ t \ 0 \ i \ 0 \ j \ 0]$$

$\alpha$ dimensions $\beta$ dimensions
Motivating example

bicg

- Initial schedule
  \[ \phi_{S_0} = [0 \ i \ 0] \]
  \[ \phi_{S_1} = [1 \ i \ 0] \]
  \[ \phi_{S_2} = [1 \ i \ 1 \ j \ 0] \]
  \[ \phi_{S_3} = [1 \ i \ 1 \ j \ 1] \]

Chlore transformation:
- reorder([1], [1,0]);
- split([1,0], 1);
- reorder([], [0,2,1]);
- fuse([0]);
- reorder([1,1,0], [1,0]);
- split([1,0,0], 2);
- split([1,0], 1);
- reorder([], [0,2,1]);
- fuse([0]);
- fuse([0]);
- fuse([0,2]);
- interchange([0,2,0], 1, 2, 0);
- embed([0,1]);
- embed([0,0]);

Our recovery transformation:
- Split([1, 1], 0);
- Fuse([0]);
- Fuse([0]);
- Fuse([0, 0]);
- Fuse([0, 0]);
- Interchange([2], 1, 3);

- Compute schedule
  \[ \phi_{S_0} = [i \ 0 \ 0] \]
  \[ \phi_{S_1} = [i \ 0 \ 1] \]
  \[ \phi_{S_2} = [j \ i \ 2] \]
  \[ \phi_{S_3} = [i \ j \ 3] \]

L. Bagnères et al, Opening polyhedral compiler’s black box, CGO’16

*with ClooG, context: _PB_M==_PB_N*
Transformation Primitives

- Transformation primitives either modify the $\alpha$ dimension, either modify the $\beta$ dimensions.
- All transformation primitives are invertible.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Reorder}(\beta, \bar{p})$</td>
<td>$\beta$-primitive</td>
<td>Reorder inner-loops or statements directly beneath the given outer-loop.</td>
</tr>
<tr>
<td>$\text{Split}(\beta)$</td>
<td>$\beta$-primitive</td>
<td>Split outer-loop just before given inner-loop or statement.</td>
</tr>
<tr>
<td>$\text{Fuse}(\beta)$</td>
<td>$\beta$-primitive</td>
<td>Fuse given loop with the next one on the same depth.</td>
</tr>
<tr>
<td>$\text{Embed}(\bar{L})$</td>
<td>$\alpha$-primitive</td>
<td>Embed given statements beneath an innermost one-iteration extra loop.</td>
</tr>
<tr>
<td>$\text{Unembed}(\bar{L})$</td>
<td>$\alpha$-primitive</td>
<td>Unembed removes the added innermost extra loop.</td>
</tr>
<tr>
<td>$\text{Reverse}(\bar{L}, d)$</td>
<td>$\alpha$-primitive</td>
<td>Reverse given output dimension for given statements.</td>
</tr>
<tr>
<td>$\text{Grain}(\bar{L}, d, c)$</td>
<td>$\alpha$-primitive</td>
<td>Add some pad between consecutive iterations in given output dimension for given statements.</td>
</tr>
<tr>
<td>$\text{Densify}(\bar{L}, d, c)$</td>
<td>$\alpha$-primitive</td>
<td>Densify removes some/all of the pad.</td>
</tr>
<tr>
<td>$\text{Shift}(\bar{L}, d, c, \bar{C})$</td>
<td>$\alpha$-primitive</td>
<td>Shift given output dimension by some (parametric) coefficient(s) for given statements.</td>
</tr>
<tr>
<td>$\text{Interchange}(\bar{L}, d_1, d_2)$</td>
<td>$\alpha$-primitive</td>
<td>Interchange two given output dimensions for given statements.</td>
</tr>
<tr>
<td>$\text{Skew}(\bar{L}, d_1, d_2, c)$</td>
<td>$\alpha$-primitive</td>
<td>Skew first output dimension by a coefficient of the second output dimension.</td>
</tr>
<tr>
<td>$\text{Reshape}(\bar{L}, d, d_{input}, c)$</td>
<td>$\alpha$-primitive</td>
<td>Skew given output dimension by a coefficient of the given input dimension.</td>
</tr>
</tbody>
</table>

- $\beta$ : beta-vector targeting an entity
- $\bar{p}$ : permutation vector
- $\bar{L}$ : list of schedule IDs
- $\bar{C}$ : list of parametric shift coefficients
- $d, d_1, d_2, d_{input}$ : an output/input dimension
- $c$ : a scalar coefficient
Recovery Algorithm

Source schedule

Normalized Source schedule

Normalization (section 4.1)

Target schedule

Normalized Target schedule
Recovery Algorithm

Normalization

\[ \phi_{S_0} : \begin{bmatrix} 5 & i & j & 1 & 0 & 0 & k \end{bmatrix} \quad \phi_{S_1} : \begin{bmatrix} 5 & i & j & 1 & 1 & 0 & k \end{bmatrix} \quad \phi_{S_2} : \begin{bmatrix} 5 & -i & j & 1 & 1 & 0 & k \end{bmatrix} \quad \phi_{S_3} : \begin{bmatrix} 5 & i & j & 0 & 0 & 0 & k \end{bmatrix} \quad \phi_{S_4} : \begin{bmatrix} 5 & i & 0 & 0 & 0 & 0 & k \end{bmatrix} \]

(a) Initial

(b) \( \beta \)-collapsing

(c) \( 2d + 1 \) format

(d) \( \beta \)-normalization
Recovery Algorithm

Source schedule

Normalized Source schedule

Alpha Minimal form

Normalized Target schedule

Target schedule

Normalization (section 4.1)

Alpha recovery (section 4.2)

Inverse Alpha recovery (section 4.2)
Recovery Algorithm

\(\alpha\)-recovery

- The \(\alpha\)-recovery algorithm search which \(\alpha\) transformations are required to achieve Alpha Minimal form
- Alpha Minimal form correspond to the identity between the input domain and the output schedule without the \(\beta\) dimensions:

\[
\begin{align*}
\phi_{S_0} &= [0, i, 0, j, 1, k, 0] \\
\phi_{S_1} &= [0, i, 0, j, 2, k, 0] \\
\phi_{S_2} &= [0, -i, 0, j, 2, k, 1] \\
\phi_{S_3} &= [0, i, 0, j, 0, k, 0] \\
\phi_{S_4} &= [0, i, 0, 0, 0, k, 1]
\end{align*}
\]

Initial (Normalize) Schedule

\[
\begin{align*}
\phi_{S_0} &= [i, j, k] \\
\phi_{S_1} &= [i, j, k] \\
\phi_{S_2} &= [i, j, k] \\
\phi_{S_3} &= [i, j, k] \\
\phi_{S_4} &= [i, k]
\end{align*}
\]

Alpha Minimal form

- The \(\alpha\) transformations are always considers in the same order:
  \{Embed, Unembed\} – \{Densify, Reverse\} – Shift – \{Skew, Interchange\} – Reshape
  - \{Embed, Unembed\}: modify number of schedule elements/dimensions
  - \{Densify, Reverse\}: modify the coefficient of an output dimension
  - Shift: add scalar coefficient of an output dimension
  - \{Skew, Interchange\} - Reshape: reorder of an output dimension, combine output dimensions together
- NB: The normalize source schedule is often already in the Alpha Minimal form
Recovery Algorithm

Source schedule → Normalized Source schedule → Alpha Minimal form → Normalized Target schedule → Target schedule

- **Normalization** (section 4.1)
- **Alpha recovery** (section 4.2)
- **Inverse Alpha recovery** (section 4.2)
- **Beta recovery** (section 4.3)
Recovery Algorithm

$\beta$-recovery

- The $\beta$-recovery algorithm search which $\beta$ transformations are required to transform source schedule to target schedule only considering the $\beta$ dimensions
- The $\beta$ transformations are repeatly consider in the same order:
  Reorder – Split – Fuse
  > Reorder: modify the order of the statements on the same depth level
  > Split: separate a statement from the current group of fused statements
  > Fuse: fuse the next statement in the same depth level
Recovery Algorithm

1. Source schedule
2. Normalized Source schedule
3. Target schedule

1. Alpha Minimal form
2. Inverse Alpha recovery (section 4.2)
3. Normalization (section 4.1)

- Normalization (section 4.1)
- Alpha recovery (section 4.2)
- Inverse Alpha recovery (section 4.2)
- Beta recovery (section 4.3)
Experiments

seidel-2d (example)

• Source schedule: \( \phi_{S_0}^S = [0 \ t \ 0 \ i \ 0 \ j \ 0] \)

• Target schedule: \( \phi_{S_0}^T = [t \ t + i \ 2 * t + i + j] \)

• Target Normalized schedule: \( \phi_{S_0}^{NT} = [0 \ t \ 0 \ t + i \ 0 \ 2 * t + i + j \ 0] \)

• Chlore (12 transformations)

  reshape([0,0,0,0], 2, 3, 1);
  grain([0,0,0,0], 1, 2);
  reshape([0,0,0,0], 1, 3, -1);
  interchange([0,0,0,0], 2, 3, 0);
  skew([0,0,0,0], 2, 3, -1);
  reverse([0,0,0,0], 2);
  interchange([0,0,0,0], 1, 3, 0);
  skew([0,0,0,0], 1, 3, -1);
  densify([0,0,0,0], 1);
  interchange([0,0,0,0], 1, 2, 0);
  skew([0,0,0,0], 1, 2, -1);

• Chlore Result schedule:

  \( \phi_{S_0}^{\text{chlore}} = [0 \ \phi_{S_0}^{\text{chlore}}[4] - i \ 0 \ t + i \ 0 \ 2 * t + i + j \ 0] \)

• Our recovery (3 transformations)

  Reshape([0], 5, 1, 1);
  Reshape([0], 5, 0, 2);
  Reshape([0], 3, 0, 1);

  • Reshape([0], 5, 1, 1): \( \phi_{S_0}^0 = [0 \ t \ 0 \ i \ 0 \ i + j \ 0] \)
  • Reshape([0], 5, 0, 2): \( \phi_{S_0}^1 = [0 \ t \ 0 \ i \ 0 \ 2 * t + i + j \ 0] \)
  • Reshape([0], 3, 0, 1): \( \phi_{S_0}^2 = [0 \ t \ 0 \ t + i \ 0 \ 2 * t + i + j \ 0] \)
Experiments

Result: Number of transformations recover by Chlore and our tool

- Polybench cases
- Source schedule = initial code
- Target schedule = schedule found with PolyTOPS schedule

<table>
<thead>
<tr>
<th>Name</th>
<th>Chlore</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>correlation</td>
<td>NA</td>
<td>45</td>
</tr>
<tr>
<td>covariance</td>
<td>NA</td>
<td>21</td>
</tr>
<tr>
<td>2mm</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>3mm</td>
<td>22</td>
<td>13</td>
</tr>
<tr>
<td>bicc</td>
<td>14</td>
<td>10</td>
</tr>
<tr>
<td>cholesky</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>doitgen</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>durbins</td>
<td>40</td>
<td>28</td>
</tr>
<tr>
<td>gemm</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>gemver</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>gesummv</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>gramschmidt*</td>
<td>29</td>
<td>13</td>
</tr>
<tr>
<td>lu</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Chlore</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvt</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>symm</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>syr2k</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>syrk</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>trisolv</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>trmm</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>floyd-warshall</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fdtd-2d</td>
<td>41</td>
<td>15</td>
</tr>
<tr>
<td>heat-3d</td>
<td>38</td>
<td>12</td>
</tr>
<tr>
<td>jacobi-1d</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>jacobi-2d</td>
<td>26</td>
<td>8</td>
</tr>
<tr>
<td>seidel-2d</td>
<td>12</td>
<td>3</td>
</tr>
</tbody>
</table>

G. Consolaro et al, PolyTOPS: Reconfigurable and Flexible Polyhedral Scheduler, CGO’24 [Accepted]
Conclusion and Future Work

• Describe an algorithm to recover transformation primitive that are applied between two polyhedral schedules

• Show that a restricted numbers of primitives is required for recovery

• Show that the set of recover primitive can be heavily reduce comparing with existing tool

• Extend $\alpha$ transformation primitive and $\alpha$ recovery with *stripmine* transformation

• Convert our transformation primitives to primitives for others tools
Thank you.