ParameTrick: Coefficient Generalization for Faster Polyhedral Scheduling
Contents

1. Polyhedral Optimization
2. ParameTrick
3. Results
4. Conclusion and Future Work
**Polyhedral Optimization: Introduction**

**Objective:** Loop based optimization

---

**Interchange**

**Initial Code**
```c
for(i=0; i<N; i++){
    for(j=0; j<i; j++){
        A[i][j] = 0;
    }
}
```

**Final Code**
```c
#pragma omp parallel for
for(i=0; i<N; i++){
    for(j=i; j<N; j++){
        A[i][j] = 0;
    }
}
```

**Schedule** $\Theta = [i \ j]$

---

**Fusion/Distribution**

**Initial Code**
```c
for(i=0; i<N; i++){
    for(j=0; j<i; j++){
        A[i][j] = A[i+1][j];
        B[i][j] = 0;
    }
}
```

**Final Code**
```c
#pragma omp parallel for
for(i=0; i<N; i++){
    for(j=0; j<i; j++){
        A[i][j] = A[i+1][j];
        B[i][j] = 0;
    }
}
```

**Schedule** $\Theta = [i \ j]$

---

**Skewing**

**Initial Code**
```c
for(i=2; i<N; i++){
    for(j=2; j<i; j++){
    }
}
```

**Final Code**
```c
#pragma omp parallel for
for(i=2; i<N; i++){
    for(j=2; j<i; j++){
        A[i][j] = A[i-1][j+1];
        B[i][j] = 0;
    }
}
```

**Schedule** $\Theta = [i \ j]$

---

**And much more...**
Polyhedral Optimization: Automatic Optimization Pipeline

1. **Algebraic Abstraction**: from for loop based kernels, we extract the semantically important information

   ```c
   for(i=0; i<10; i++){
       for(j=0; j<20; j++){
           A[j][i] = 0;
       }
   }
   ```

   Domain
   \[
   \begin{bmatrix}
   1 & 0 & 0 \\
   1 & 0 & 1 \\
   0 & -1 & 0 \\
   \end{bmatrix}
   \geq 0
   \]

   Initial Schedule
   \( \Theta = [i \ j] \)

   Dependencies
   \( \emptyset \)

   2. **Polyhedral Scheduler**: automatically finds a scheduling transformation (complete order of the iterations)

   Output Schedule
   \( \Theta = [j // i/] \)

   3. **Code generation**: generating the code corresponding to the scheduling transformation

   ```c
   #pragma omp parallel for
   for(i=0; i<10; i++){
       for(j=0; j<20; j++){
           A[j][i] = 0;
       }
   }
   ```
**Polyhedral Scheduler:** Scheduling Space

**Objective:** finding an optimal (execution time) scheduling function

Scheduling function for the statement $S$:

$$\Theta(S) = [\varphi_0(S) \ldots \varphi_k(S)]$$

where

$$\varphi_i(S) = T_i(S) \cdot \begin{bmatrix} i \cdot t^2 \\ N \\ 1 \end{bmatrix}$$

Example:

```
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        A[i][j][1] = 0; //S
    }
}
```

Schedule: $\Theta(S) = \begin{bmatrix} I \\ J \end{bmatrix}$

- $\varphi_0(S) = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$
- $\varphi_1(S) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}$

The scheduler looks for the optimal $T_i(S)$ vector.

It builds 1 ILP (Integer Linear Programming) problem for each scheduling dimension $[T_i(S_0) \ldots T_i(S_m)]$. 
Polyhedral Scheduler: Constraints

Legality Constraints: allows only transformations preserving the semantics

For each data-dependency $\delta_{S \rightarrow R}$

Legality: $\varphi_i(R) - \varphi_i(S) \geq 0$

We apply Farkas Lemma + Fourier Motzkin Elimination to obtain constraints only on the $[\bar{T}_i(R) \bar{T}_i(S)]$ space
**ParameTrick**: Idea

**Objective**: simplifying constraints’ numerical complexity

Input Code:

```c
for(i=0; i<537; i++)
{
    c[i] = b;  // S
}
for(j=0; j<537; j++)
{
    d[i][j] = c[i];  // R
}
```

Original Domains:

\[
D_S = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 537 \\
-1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & -1 & 537 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & -1 & 537 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
\]

\[
D_R = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -1 & 537 \\
-1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & -1 & 537 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & -1 & 537 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
\]

Original Dependency:

\[
\delta_{S \rightarrow R} = \begin{pmatrix} i_S \\
0 & 1 & 0 & 0 & 0 & 0 & i_R \\
-1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & -1 & 537 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 537 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix} \geq 0
\]

Legality Constraint:

\[
\begin{pmatrix} t_i^S \\
0 & 0 & 537 & -1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
-537 & 537 & 537 & -1 & 0 \\
-537 & 537 & 0 & -1 & 0 \\
-1 & 1 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 \\
\end{pmatrix} \geq 0
\]

**ParameTrick**

*Substitute big coefficients with parametric constants*

Let’s substitute 537 by N:

Simplified Dependency:

\[
\delta_{S \rightarrow R} = \begin{pmatrix} i_S \\
0 & 1 & 0 & 0 & 0 & 0 & i_R \\
-1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & -1 & 537 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 537 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix} \geq 0
\]

Legality Constraint:

\[
\begin{pmatrix} t_i^S \\
0 & 0 & 1 & -1 & 1 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 \\
-1 & 1 & -1 & 1 & 0 & 0 \\
-1 & 1 & -1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 10 \\
0 & 0 & 0 & 0 & -1 & 10 \\
\end{pmatrix} \geq 0
\]
**ParameTrick: Idea**

Not convinced?

```c
for (i=0; i<3200; i++)
for (j=0; j<3600; j++) {
    tmp[i][j] = 0;
    for (k=0; k<4400; ++k)
        tmp[i][j] += alpha*A[i][k]*B[k][j]; // S
}

for (i=0; i<3200; i++)
for (j=0; j<4800; j++) {
    D[i][j] *= beta;
    for (k=0; k<3600; ++k)
        D[i][j] += tmp[i][k]*C[k][j]; // R
}
```

Original

Legality(dep$_{S\rightarrow R}$)

After ParameTrick

Legality(dep$_{S\rightarrow R}$)

N1=4800
N2=4400
N3=3600
N4=3200
Results: PolyBench (using PolyTOPS scheduler)

We compared \textit{Compilation time} between 2 variants of ParameTrick:

- **p-trick**: Using ParameTrick + Positivity Constraints (N > 0, M > 0)
- **p-trick-extra**: Using ParameTrick + Relation Constraints (N > 0, M > 0, N > M)

We also compared the \textit{Execution time} of the final scheduling transformation:

- Speedup = 1 means that the same transformation was found

We used our scheduler PolyTOPS for the experiments, using isl-0.25 as ILP solver

HW Specification: Intel Xeon E5-2683 CPU(x86 64), 2 sockets, 16 cores per socket

\textit{G. Consolaro et al, PolyTOPS: Reconfigurable and Flexible Polyhedral Scheduler, CGO’’24[Accepted]}
## Results: PolyBench (using PolyTOPS scheduler)

### Compilation:
- **Speedups** (GMP? Fourier Motzkin?)
- **Slowdowns** (extra constraints and variables)

### Execution:
- Identical transformations
- **Speedups**
- **Slowdowns**

<table>
<thead>
<tr>
<th>Case</th>
<th>Original Time (ms)</th>
<th>Speedup (original / p-trick)</th>
<th>Speedup (original / p-trick-extra)</th>
<th>Speedup (original / p-trick)</th>
<th>Speedup (original / p-trick-extra)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3mm</td>
<td>&gt;600000</td>
<td>3636.32</td>
<td>&gt;176.67</td>
<td>25.12</td>
<td>n.a.</td>
</tr>
<tr>
<td>correlation</td>
<td>2021.4</td>
<td>2.31</td>
<td>0.09</td>
<td>1</td>
<td>n.a.</td>
</tr>
<tr>
<td>cholesky</td>
<td>9032.07</td>
<td>151.43</td>
<td>147.72</td>
<td>2.05</td>
<td>2.23</td>
</tr>
<tr>
<td>lu</td>
<td>14774.03</td>
<td>263.84</td>
<td>291.1</td>
<td>1.61</td>
<td>1.61</td>
</tr>
<tr>
<td>floyd-warshall</td>
<td>11820.16</td>
<td>391.19</td>
<td>386.67</td>
<td>1.27</td>
<td>1.27</td>
</tr>
<tr>
<td>bicc</td>
<td>19.94</td>
<td>0.91</td>
<td>0.73</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>covariance</td>
<td>887.65</td>
<td>5.63</td>
<td>2.41</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>fftd-2d</td>
<td>650.69</td>
<td>6.86</td>
<td>5.47</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>gemm</td>
<td>22.49</td>
<td>1.01</td>
<td>0.52</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>gemver</td>
<td>3282.38</td>
<td>107.28</td>
<td>106.12</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>gesumnnv</td>
<td>29.96</td>
<td>0.93</td>
<td>0.93</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>heat-3d</td>
<td>751.84</td>
<td>4.41</td>
<td>4.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>jacob-1d</td>
<td>30.97</td>
<td>1.23</td>
<td>1.22</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>jacob-2d</td>
<td>351.62</td>
<td>3.53</td>
<td>4.82</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>mtv</td>
<td>11.35</td>
<td>0.91</td>
<td>0.94</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>seidel-2d</td>
<td>51.89</td>
<td>0.97</td>
<td>0.9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>syrkz</td>
<td>22.25</td>
<td>0.99</td>
<td>0.72</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>syrk</td>
<td>20.93</td>
<td>0.92</td>
<td>0.73</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>trisolv</td>
<td>23.15</td>
<td>1.09</td>
<td>1.08</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>trmm</td>
<td>31.14</td>
<td>1.02</td>
<td>0.61</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2mm</td>
<td>2523.95</td>
<td>35.93</td>
<td>5.53</td>
<td>0.8</td>
<td>1</td>
</tr>
<tr>
<td>gramschmidt</td>
<td>274.84</td>
<td>1.53</td>
<td>0.32</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td>symm</td>
<td>322.79</td>
<td>4.55</td>
<td>3.5</td>
<td>0.78</td>
<td>0.78</td>
</tr>
<tr>
<td>atax</td>
<td>59.94</td>
<td>2.12</td>
<td>1.28</td>
<td>0.76</td>
<td>1</td>
</tr>
<tr>
<td>doigten</td>
<td>3984.24</td>
<td>34.59</td>
<td>7.6</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>durbin</td>
<td>195.04</td>
<td>2.79</td>
<td>2.67</td>
<td>0.00002</td>
<td>0.00002</td>
</tr>
</tbody>
</table>
Results: PolyBench

What happens if we increase the loop bounds? (isl-solver)

The compilation time is completely invariant to the loop bound values when using ParameTrick.

Without ParameTrick, depending on the case, we can notice an important growth in terms of compilation time.
Results: MindSpore (using PolyTOPS and FPL solver)

MindSpore is an AI Framework that implements a polyhedral pipeline (AKG) using PolyTOPS scheduler.

We applied ParameTrick to some AI operators from MindSpore.

The scheduling transformation found is the same for these cases. The execution time is identical.

<table>
<thead>
<tr>
<th>Case</th>
<th>Original Time (ms)</th>
<th>Time (ms) (p-trick)</th>
<th>Speedup (p-trick)</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch_norm</td>
<td>&gt;600000</td>
<td>9764</td>
<td>&gt;61.45</td>
</tr>
<tr>
<td>two2fractal_v1</td>
<td>96519</td>
<td>78</td>
<td>1237.42</td>
</tr>
<tr>
<td>two2fractal_v2</td>
<td>105</td>
<td>51</td>
<td>2.05</td>
</tr>
<tr>
<td>two2fractal_v3</td>
<td>344</td>
<td>28</td>
<td>12.29</td>
</tr>
<tr>
<td>maxpool_grad_v1</td>
<td>5583</td>
<td>2333</td>
<td>2.39</td>
</tr>
<tr>
<td>force_grad</td>
<td>381</td>
<td>180</td>
<td>2.12</td>
</tr>
<tr>
<td>max_pool_grad_v2</td>
<td>&gt;600000</td>
<td>529</td>
<td>&gt;1134</td>
</tr>
<tr>
<td>hpl_cholesky</td>
<td>9291</td>
<td>121</td>
<td>76.78</td>
</tr>
<tr>
<td>hpl_lu</td>
<td>29396</td>
<td>97</td>
<td>303.05</td>
</tr>
</tbody>
</table>
Conclusion and Future Work

• We showed how ParameTrick can decrease tremendously the compilation time while losing only few optimization opportunities in practice

• This simple technique can be used to schedule cases that would be untreatable otherwise

• We indirectly showed that in some cases, Pluto cost function is definitely not enough. What is missing?

• Is there a way to understand if a kernel could benefit (or not) from ParameTrick?

• Further analysis about GMP impact would help explaining the compilation time reduction
Thank you.

See you at the Poster session :)

Bring digital to every person, home and organization for a fully connected, intelligent world.

Copyright©2024 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.