CSC2/455 Software Analysis and Improvement
Loop Transformations

Sreepathi Pai

URCS

April 3, 2019
Outline

Execution Order

Postscript
Outline

Execution Order

Postscript
for(i = 0; i <= 5; i++) {
    for(j = i; j <= 7; j++) {
        Z[j, i] = 0;
    }
}

Dependences?
Dependences

The statements in this loop do not have any dependence.
Assuming row-major ordering, what can you say about the locality of this execution order?
### Execution Order (New)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>
Changing the order

```c
for(i = 0; i <= 5; i++) {
    for(j = i; j <= 7; j++) {
        Z[j, i] = 0;
    }
}
```

What would the loop indices need to be if I wanted to execute `j` as the outermost loop?

```c
for(j = ?; j <= ?; j++) {
    for(i = ?; i <= ?; i++) {
        Z[j, i] = 0;
    }
}
```
New Loop Bounds

```c
for(j = 0; j <= 7; j++) {
    for(i = 0; i <= min(5, j); i++) {
        Z[j, i] = 0;
    }
}
```
The Problem

- **Given:**
  - a set of affine constraints (inequalities) defining the iteration space
  - an “preferred” execution order
- **Can we generate a set of loop bounds for each loop in the loop nest?**
Running Example

\[0 \leq i \leq 5\]
\[i \leq j \leq 7\]

- Order \(i\) (innermost loop) to \(j\) (outermost loop)
What are the loop bounds for $j$?

Let’s eliminate $i$:

$$0 \leq i$$
$$i \leq 5$$
$$i \leq j$$
$$j \leq 7$$
Loop bound for \( j \) (1)

Rearrange equations so that they are all in the form:

\[
\begin{align*}
L & \leq c_1 x_m \\
c_2 x_m & \leq U
\end{align*}
\]

- \( c_1, c_2, \ldots \) are constants, \( x_m \) is the index variable
- \( L \) and \( U \) are constraint expressions (possibly containing other variables)
- yields new constraint: \( c_2 L \leq c_1 U \) with \( x_m \) eliminated!
What are the loop bounds for $j$?

Let’s eliminate $i$:

\[
0 \leq 1i \\
1i \leq 5 \\
1i \leq j \\
j \leq 7
\]

yields:

\[
0 \leq 5 \\
0 \leq j \\
j \leq 7
\]
What are the loop bounds for \( i \)?

Let's eliminate \( j \):

\[
0 \leq i \\
i \leq 5 \\
i \leq 1j \\
1j \leq 7
\]

yields:

\[
0 \leq i \\
i \leq 5 \\
i \leq 7
\]
Results

- $0 \leq j \leq 7$ when $i$ eliminated
- $0 \leq i \leq 5$ when $j$ eliminated – but this is original loop bounds
  - not entirely unsurprising!
- This method is called Fourier–Motzkin elimination
  - Can *project* one dimension at a time
  - Now, need to iteratively construct projections
Fourier–Motzkin Elimination

- $S$ is the original set of iteration space constraints
- $C$ is the set of constraints involving $x_m$
- Form constraint $c_2 L \leq c_1 U$ with $x_m$ eliminated for each pair of $L$ and $U$ in $C$
  - Add to set $C_{new}$ if satisfiable
  - Else projection is not possible since $S$ is unsatisfiable (and hence contains 0 points)
- The projection is $S' = S - C + C_{new}$

Algorithm 11.11 in the Dragon Book.
Computing New Loop Bounds Iteratively

- Let $S_n$ be the original iteration space constraints
- Let ordering of variables be $v_1$ (outermost) to $v_n$ (innermost)
  - i.e. $v = [j, i]$
- In reverse order $i$ from $n$ to 1:
  - Let $L_{v_i}$ be lower bound constraints on $v_i$ in $S_i$
  - Let $U_{v_i}$ be upper bound constraints on $v_i$ in $S_i$
  - Let $S_{i-1}$ be the result of Fourier–Motzkin elimination of $v_i$ in $S_i$
- In order of $v_1$ to $v_n$:
  - Remove any redundant constraints in $L_{v_i}$ and $U_{v_i}$ implied by cumulative previous lower bound and upper bound constraints

Figure 11.15 in the Dragon Book.
For our example

- $S_2$ was original iteration space constraints
  - $L_i : 0 \leq i$
  - $U_i : i \leq 5, i \leq j$ implies $i \leq \min(5, j)$
- $S_1$ is $0 \leq j$ and $j \leq 7$ (i.e. $i$ was eliminated)
  - $L_j : 0 \leq j$
  - $U_j : j \leq 7$
More than permutations: Traversal Axis

- Original loop was iteration in 2-D space.
  - Say, $j$ was x-axis and $i$ was y-axis
- Original loop with $i$ outermost traversed “horizontally” (along x-axis) first
- Outermost $j$ traversed “vertically” first
- Now we want to traverse “diagonally”
How can we traverse diagonally?

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Add new constraints

- \( k = j - i \) is a constant in inner loop, increasing from 0 to 7 across outer loop
- Substitute \( i = j - k \) in the original constraints:
  - \( 0 \leq j - k \leq 5 \)
  - \( j - k \leq j \leq 7 \)
- Order loop in \( k, j \) order
  - \( L_j : k \leq j \)
  - \( U_j : j \leq 7, j \leq 5 + k \)
  - \( L_k : 0 \)
  - \( L_k : 7 \)
Result

```java
for(k = 0; k <= 7; k++) {
    for(j = k; j <= min(7, 5 + k); j++) {
        Z[j, j - k] = 0;
    }
}
```

If loop traversal order can be specified as an affine transformation, then the loop bounds can be generated as usual.

▷ Not all traversal orders are affine
▷ Deciding which axis to traverse is a harder problem
  ▷ For example, to improve locality or parallelism or both!
▷ Recall transformations from last class.
Affine Transformations Workflow

- Identify loops with affine iteration spaces
- Compute dependences
- Figure out transforms of affine spaces
  - must respect dependences
  - may optimize other metrics (e.g. locality, parallelism)
- Generate loops such that:
  - Dependence constraints are met
  - Transformed iteration space constraints are met
- Parallelize resulting loops
  - Vectorization
  - Software Pipelining
Software Pipelining

```c
for(i = 1; i <= m; i++)
    for(j = 1; j <= n; j++)
        X[i] = X[i] + Y[i, j];
```

- Inner loop is sequential
- Outer loop can be parallelized
  - Processor $i$ handles loop iteration $i$ of outer loop
Software Pipelined Loop

Each iteration of the inner loop is executed on a different processor, with data being passed from one processor to another.

<table>
<thead>
<tr>
<th>P0</th>
<th></th>
<th>P1</th>
<th></th>
<th>P3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What are the advantages of doing this?
Stuff we did not cover

- Loop tiling/blocking
  - Simple, see textbook
- Many other loop transformations
  - See the slides in the readings on polyhedral compilation posted on Blackboard
Outline

Execution Order

Postscript
References

- Chapter 11 of the Dragon Book
  - Section 11.3.2 of the Dragon Book
  - Section 11.9 of the Dragon Book