CSC2/455 Software Analysis and Improvement
Loop Transformations

Sreepathi Pai

URCS

April 3, 2019

Outline

Outline

Running Example

```
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.

Execution Order (Default)

```
[0, 0] [1, 0] [2, 0] [3, 0] [4, 0] [5, 0] [6, 0] [7, 0]
       [1, 1] [2, 1] [3, 1] [4, 1] [5, 1] [6, 1] [7, 1]
              [2, 2] [3, 2] [4, 2] [5, 2] [6, 2] [7, 2]
                     [3, 3] [4, 3] [5, 3] [6, 3] [7, 3]
                            [4, 4] [5, 4] [6, 4] [7, 4]
                                   [5, 5] [6, 5] [7, 5]
```

Assuming row-major ordering, what can you say about the locality of this execution order?

Execution Order (New)

```
[0, 0]
[1, 0] [1, 1]
[2, 0] [2, 1] [2, 2]
[3, 0] [3, 1] [3, 2] [3, 3]
[4, 0] [4, 1] [4, 2] [4, 3] [4, 4]
[5, 0] [5, 1] [5, 2] [5, 3] [5, 4] [5, 5]
[6, 0] [6, 1] [6, 2] [6, 3] [6, 4] [6, 5]
[7, 0] [7, 1] [7, 2] [7, 3] [7, 4] [7, 5]
```

Changing the order

```
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?

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

New Loop Bounds

```
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

$$0 \leq i$$
$$i \leq 5$$
$$i \leq j$$
$$j \leq 7$$

- 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:

$$\begin{aligned} L &\leq c_1 x_m \\ & \quad c_2 x_m \leq U \end{aligned}$$

- $c_1, c_2, \ldots$ 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:

$$\begin{aligned} 0 &\leq 1i \\ 1i &\leq 5 \\ 1i &\leq j \\ j &\leq 7 \end{aligned}$$

yields:

$$\begin{aligned} 0 &\leq 5 \\ 0 &\leq j \\ j &\leq 7 \end{aligned}$$

What are the loop bounds for i?

Let's eliminate j:

$$\begin{aligned} 0 &\leq i \\ i &\leq 5 \\ i &\leq 1j \\ & \quad 1j \leq 7 \end{aligned}$$

yields:

$$\begin{aligned} 0 &\leq i \\ i &\leq 5 \\ i &\leq 7 \end{aligned}$$

Results

- $0 <= j <= 7$ when i eliminated
- $0 <= i <= 5$ 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
  - $L_i : 0 <= i$
  - $U_i : i <= 5, i <= j$ implies $i <= min(5, j)$
- S_1 is $0 <= j$ and $j <= 7$ (i.e. i was eliminated)
  - $L_j : 0 <= j$
  - $U_j : j <= 7$

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?

```
[0, 0] [1, 1] [2, 2] [3, 3] [4, 4] [5, 5]
[1, 0] [2, 1] [3, 2] [4, 3] [5, 4] [6, 5]
[2, 0] [3, 1] [4, 2] [5, 3] [6, 4] [7, 5]
[3, 0] [4, 1] [5, 2] [6, 3] [7, 4]
[4, 0] [5, 1] [6, 2] [7, 3]
[5, 0] [6, 1] [7, 2]
[6, 0] [7, 1]
[7, 0]
```

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
  - $L_j : k <= j$
  - $U_j : j <= 7, j <= 5 + k$
  - $L_k : 0$
  - $L_k : 7$

Result

```
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

```
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.

```
      P0                P1                P3
X[1] += Y[1,1]
X[2] += Y[2,1]  X[1] += Y[1,2]
X[3] += Y[3,1]  X[2] += Y[2,2]  X[1] += Y[1,3]
X[4] += Y[4,1]  X[3] += Y[3,2]  X[2] += Y[2,3]
                X[4] += Y[4,2]  X[3] += Y[3,3]
                                X[4] += Y[4,3]
```

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

References

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