**Code Generation Using Fourier Motzkin**

**Previously**
- Data dependences and loops
- Loop transformations
- Unimodular transformation framework
- Kelly and Pugh transformation framework (affine transformations per statement)

**Today**
- Code generation
- use Fourier Motzkin to calculate new loop bounds
- review array access transformations
- review loop bounds transformations

**Code Generation**

**Goals**
- express outermost loop bounds in terms of symbolic constants and constants
- express inner loop bounds in terms of any enclosing loop variables, symbolic constants, and constants

**Approach**
- Project out inner loop iteration variables to determine loop bounds for outer loops
- Fourier Motzkin elimination is the algorithm that projects a variable out of a polyhedron

**Fourier-Motzkin Elimination: The Idea**

**Polyhedron**
- convex intersection of a set of inequalities
- model for iteration spaces

**Problem**
- given a polyhedron how do we generate loop bounds that scan all of its points?
- example: two possible loop orders
  - \((i, j)\)
  - \((j, i)\)

**Fourier-Motzkin Elimination: The Algorithm**

\[ \text{FM}(P, i_k) \Rightarrow P' \]

**Input:**
- \(P = \{(i_1, i_2, ..., i_d) \mid Q_i^T \geq (q + B p)\} \)
- \(i_k\) such that \(1 \leq k \leq d\)

**Output:**
- \(P' = \{(i_1, ..., i_{k-1}, i_{k+1}, ..., i_d) \mid Q_i'^T \geq (q' + B' p)\} \)

**Algorithm:**
- for each lower bound of \(i_k\), \(L \leq c_1 i_k\)
- for each upper bound of \(i_k\), \((c_2 i_k \leq U)\)
- \(P = P - \{L \leq c_1 i_k\}\)
- \(P = P - \{c_2 i_k \leq U\}\)
- \(P' = P' \cup \{c_2 L \leq c_1 U\}\)
Distinguishing Upper and Lower Bounds

Simple Algorithm
- given that the polyhedron is represented as follows:
  \[ P = \{(i_1, i_2, \ldots, i_d) \mid Q\hat{i} \geq (\bar{q} + B\bar{p})\} \]
- any constraint with a positive coefficient for \( i_k \) is a lower bound
- any constraint with a negative coefficient for \( i_k \) is an upper bound

\[ j \leq 5 \quad i \leq j \quad 1 \geq i \]

Triangular Iteration Space Example

(\( i, j \)) for target iteration space

(\( j, i \)) for target iteration space

General Algorithm for Generating Loop Bounds

Input: \( P = \{(i_1, i_2, \ldots, i_d) \mid Q\hat{i} \geq (\bar{q} + B\bar{p})\} \)
where the i vector is the desired loop order

Output:
\( L_{i_k}, U_{i_k}, \ldots, L_{i_k} \) such that \( L_{i_k} = f(t_1, \ldots, i_{k-1}) \)
\( U_{i_k}, U_{i_k}, \ldots, L_{i_k} \) such that \( U_{i_k} = g(t_1, \ldots, i_{k-1}) \)

Algorithm:
\[ P_n = P \]
for \( k = d \) to \( 1 \) by \(-1\)
\[ L_{i_k} = \text{all lower bounds for } i_k \text{ in } P_k \]
\[ U_{i_k} = \text{all upper bounds for } i_k \text{ in } P_k \]
\[ P_{k-1} = FM(P_k, i_k) \]

Loop Skewing and Permutation

Original code
\[
\begin{align*}
do & \ i = 1,6 \\
do & \ j = 1,5 \\
A(i,j) & = A(i-1,j+1)+1 \\
& \text{endo} \\
& \text{endo}
\end{align*}
\]
Distance vector: \((1, -1)\)

Skewing followed by Permutation:
\[
\begin{bmatrix}
1 & 1 \\
1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
i \\
j
\end{bmatrix}
= 
\begin{bmatrix}
i' \\
j'
\end{bmatrix}
\]
Transforming the Dependences and Array Accesses

Original code

\[
\begin{align*}
do & \ i = 1,6 \\
do & \ j = 1,5 \\
A(i,j) &= A(i-1,j+1)+1 \\
\end{align*}
\]

Dependence vector:
\[
\begin{bmatrix}
1 & 1 \\
1 & 0
\end{bmatrix}
\begin{bmatrix}
1 \\
-1
\end{bmatrix}
= \begin{bmatrix}
0 \\
1
\end{bmatrix}
\]

New Array Accesses:
\[
\begin{align*}
A\left(i\right) & = A(i,j) \\
A\left(i\right) & = A(i',j') \\
A\left(i\right) & = A(i-1,j+1) \\
A\left(i\right) & = A(j'-1,i'-1) + A(j',i'-1) + A(j',i'+1)
\end{align*}
\]

Transforming the Loop Bounds

Original code

\[
\begin{align*}
do & \ i = 1,6 \\
do & \ j = 1,5 \\
A(i,j) &= A(i-1,j+1)+1 \\
\end{align*}
\]

Bounds:
\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
i \\
j
\end{bmatrix}
\geq
\begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix}
\begin{bmatrix}
j' \\
i'
\end{bmatrix}
\geq
\begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix}
\begin{bmatrix}
i \\
j
\end{bmatrix}
\]

Transformed code (use general loop bound alg)

\[
\begin{align*}
do & \ i' = 2,6 \\
do & \ j' = \max(i'-5,1), \min(6,i'-1) \\
A(j',i') & = A(j'-1,i'-1) + A(j',i'-1) + A(j',i'+1) \\
\end{align*}
\]

Wavefront Parallelism Example

Example

\[
\begin{align*}
do & \ i = 1,6 \\
do & \ j = 1,\min(5,7-i) \\
A(i,j) &= A(i-1,j-1) + A(1,j-1) \\
\end{align*}
\]

Goal

\begin{itemize}
\item Determine a unimodular transformation that enables indicating that the inner loop is fully parallel. (with an OpenMP directive for example)
\end{itemize}

\[
T = \begin{bmatrix}
0 & 1 \\
1 & 0
\end{bmatrix}
\begin{align*}
do & \ i' = 1,5 \\
do & \ j' = 1,7-i' \ (parallel) \\
A(j',i') & = A(j'-1,i'-1) + A(j',i'-1) \\
\end{align*}
\]

Concepts

Fourier-Motzkin Elimination
\begin{itemize}
\item algorithm
\item using for code generation
\end{itemize}

Loop bounds
\begin{itemize}
\item how to determine upper and lower bounds for a variable when bounds are in matrix format
\end{itemize}

Examples
\begin{itemize}
\item triangular matrix
\item skew and permute example
\item wavefront example
\end{itemize}
Next Time

Lecture
- Parallelization with no synchronization or communication

Suggested Exercises
- 11.3.2, 11.3.3, 11.3.4