Lecture 11
Pipelined Parallelism

1. Fully permutable loop nests & pipelining
2. Example: Transforming for full permutability
3. Time Affine Partitioning: Problem
4. Time Affine Partitioning Algorithm
5. $O(1)$ Synchronization problem

Readings: Chapter 11.8-11.9
1. Maximum Parallelism & No Communication

C: Space partitioning of Computation to Processor ID
For every pair of data dependent accesses $F_1i_1+f_1$ and $F_2i_2+f_2$

Find $C_1, c_1, C_2, c_2$:

$$\forall \ i_1, i_2 \quad F_1i_1+f_1 = F_2i_2+f_2 \rightarrow C_1i_1+c_1 = C_2i_2+c_2$$

with the objective of maximizing the rank of $C_1, C_2$
**SOR (Successive Over-Relaxation): An Example**

for $i = 1$ to $m$
  for $j = 1$ to $n$
Pipelineable Parallelism

for i = 1 TO m
    for j = 1 to n

Processor ID: p
Synchronization variable: t[p] initialized to 0

for j = 1 to n
    if (p==1) or (t[p-1]>=j)
    t[p]++;
A Fully Permutable Loop Nest: Example

for $i = 1$ TO $m$
  for $j = 1$ to $n$
  
for $j = 1$ TO $n$
  for $i = 1$ to $m$

\[
\begin{bmatrix}
  j' \\
  i'
\end{bmatrix} =
\begin{bmatrix}
  0 & 1 \\
  1 & 0
\end{bmatrix}
\begin{bmatrix}
  i \\
  j
\end{bmatrix}
\]
A Fully Permutable Loop Nest: Definition

- A loop nest is fully permutable if all the loops can be permuted arbitrarily without changing the semantics of the program.
r-Dimensional Pipelineable Parallelism

• r-deep fully permutable loop nest; r > 1
  – r choices of outermost loops
  – r-1 degrees of parallelism
  – \( O(n^{r-1}) \) parallelism
  – \( O(n) \) synchronization

• Code generation
  – r-1 outer loops: processor ID \((p_1, p_2, ..., p_{r-1})\)
  – Sequential rth loop: \(i_r\)
  – iteration \(i_r\) for processor \((p_1, p_2, ..., p_{r-1})\), waits for
    iteration \(i_r\) for processors \((p_{1-1}, p_2, ..., p_{r-1})\),
    \((p_1, p_{2-1}, ..., p_{r-1})\), ..., 
    \((p_1, p_2, ..., p_{r-1-1})\).
Sequential Code

- A sequential loop nest

- Degree of the outermost permutable loop nest?

- Intuition: Fully permutable loop nest:
  - Dependence does not point backwards along any axis
Pipelined Parallelism

1. Fully permutable loop nests & pipelining
2. Example: Transforming for full permutability
3. Time Affine Partitioning: Problem
4. Time Affine Partitioning Algorithm
5. $O(1)$ Synchronization problem
2. Example

for $i = 0$ TO $m$
    for $j = 0$ to $n$
        $X[i+1]=(X[j]+X[j+1]+X[j+2])$
Transforming for Fully Permutable Loop Nests

for i = 0 TO m
    for j = 0 to n
        X[i+1]=(X[j]+X[j+1]+X[j+2])
Transforming for Fully Permutable Loop Nests

for $i = 0$ TO $m$
for $j = 0$ to $n$
$X[j+1] = (X[j] + X[j+1] + X[j+2]) / 3$

for $i' = 0$ TO $m$
for $j' = i'$ to $i' + n$
$X[j'-i'+1] = (X[j'-i'] + X[j'-i'+1] + X[j'-i'+2]) / 3$

Loop bounds:

$i' = i$
$0 <= i' <= m$

$j' = i + j$
$0 <= j' - i' <= n$

$j = j' - i'$

$\begin{bmatrix}
  i' \\
  j'
\end{bmatrix} = \begin{bmatrix}
  1 & 0 \\
  1 & 1
\end{bmatrix} \begin{bmatrix}
  i \\
  j
\end{bmatrix}$

M. Lam
CS243: Loop Transformations 12
Fully Permutable?

\[
\begin{bmatrix}
i' \\
j'
\end{bmatrix} = \begin{bmatrix}
1 & 0 \\
1 & 1
\end{bmatrix} \begin{bmatrix}
i \\
j
\end{bmatrix}
\]
Pipelined Parallelism

1. Fully permutable loop nests & pipelining
2. Example: Transforming for full permutability
3. Time Affine Partitioning: Problem
4. Time Affine Partitioning Algorithm
5. $O(1)$ Synchronization problem
3. Recall: Pipelineable Parallelism

• r-deep fully permutable loop nest; \( r > 1 \)
  – \( r \) choices of outermost loops
  – \( r-1 \) degrees of parallelism
  – \( O(n^{r-1}) \) parallelism
  – \( O(n) \) synchronization

• Goal: Find transformation to maximize the degree of pipelining
• \( \rightarrow \) Find all the possible outermost loops
Finding the Maximum Degree of Pipelining

C: Time partitioning of Computation to Time

For every pair of data dependent accesses $F_1i_1+f_1$ and $F_2i_2+f_2$

Let $B_1i_1+b_1 \geq 0$, $B_2i_2+b_2 \geq 0$ be the corresponding loop bound constraints,

Find $C_1$, $c_1$, $C_2$, $c_2$:

$$\forall i_1, i_2 \quad B_1i_1 + b_1 \geq 0, \quad B_2i_2 + b_2 \geq 0$$

$$(i_1 \leq i_2) \land (F_1i_1+f_1 = F_2i_2+f_2) \rightarrow C_1i_1+c_1 \leq C_2i_2+c_2$$

with the objective of maximizing the rank of $C_1$, $C_2$
Solutions of Time Mapping $C$

\[
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}, \begin{bmatrix}
1 & 0 \\
1 & 1
\end{bmatrix}
\]
Solutions to Loop Transforms

\[
\begin{bmatrix}
1 & 0 \\
1 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
i' \\
j'
\end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\
j
\end{bmatrix}
\]

\[
\begin{bmatrix}
j' \\
i'
\end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} i \\
j
\end{bmatrix}
\]
Pipelined Parallelism

1. Fully permutable loop nests & pipelining
2. Example: Transforming for full permutability
3. Time Affine Partitioning: Problem
4. Time Affine Partitioning Algorithm
5. $O(1)$ Synchronization problem
4. Communication-Free vs Pipelining

Loops

\[ F_{1i_1} + f_1 \]

\[ F_{2i_2} + f_2 \]

\[ C_{1i_1} + c_1 \]

\[ C_{2i_2} + c_2 \]

Processor ID

Array

\[ i_1 \leq i_2 \]

Loops

\[ F_{2i_2} + f_2 \]

\[ F_{1i_1} + f_1 \]

\[ C_{1i_1} + c_1 \]

\[ C_{2i_2} + c_2 \]

Time Stage
Comparing the Two Problems

Communication-Free Parallelism:
C: Space partitioning of Computation to Processor ID
For every pair of data dependent accesses $F_1 i_1 + f_1$ and $F_2 i_2 + f_2$

Find $C_1, c_1, C_2, c_2$:

$$\forall i_1, i_2 \quad F_1 i_1 + f_1 = F_2 i_2 + f_2 \rightarrow C_1 i_1 + c_1 = C_2 i_2 + c_2$$

with the objective of maximizing the rank of $C_1, C_2$

Pipelining Parallelism:
C: Time mapping of Computation to Time
For every pair of data dependent accesses $F_1 i_1 + f_1$ and $F_2 i_2 + f_2$

Let $B_1 i_1 + b_1 \geq 0, B_2 i_2 + b_2 \geq 0$ be the corresponding loop bound constraints,

Find $C_1, c_1, C_2, c_2$:

$$\forall i_1, i_2 \quad B_1 i_1 + b_1 \geq 0, B_2 i_2 + b_2 \geq 0$$

$$(i_1 \leq i_2) \land (F_1 i_1 + f_1 = F_2 i_2 + f_2) \rightarrow C_1 i_1 + c_1 \leq C_2 i_2 + c_2$$

with the objective of maximizing the rank of $C_1, C_2$
Farkas Lemma

Finding the possible time dimensions $c$:
Given matrix $A$, find a vector $c$ such that
for all vectors $x$ such that $Ax \geq 0$,
\[ c^T x \geq 0 \]

Farkas Lemma, 1901 (real domain)
The primal system of inequalities
\[ Ax \geq 0, \quad c^T x < 0 \]
has a real-valued solution $x$
or, the dual system
\[ A^T y = c, \quad y \geq 0 \]
has a real-valued solution $y$, but never both.

Time partitioning: Find $c$ such that $A^T y = c, \ y \geq 0$

Note: Farkas Lemma: a theorem of the alternative
(no intuitive proof exists)
Cholesky Decomposition

for (i = 1; i ≤ N; i++) {
    for (j = 1; j ≤ i - 1; j++) {
        for (k = 1; k ≤ j - 1; k++)
            \( X[i,j] = X[i,j] - X[i,k]*X[j,k]; \)
        \( X[i,j] = X[i,j]/X[j,j]; \)
    }
    for (m=1; m=i-1; m++) {
        \( X[i,i]=X[i,i]-X[i,m]*X[i,m];\)
    }
    \( X[i,i] = \sqrt(X[i,i]); \)
}

for (i = 1; i ≤ N; i++) {
    for (j = 1; j ≤ i; j++) {
        for (k = 1; k ≤ i; k++)
            if (j<i && k<j)
                \( X[i,j] = X[i,j] - X[i,k]*X[j,k]; \)
        if (j==k && j<i)
            \( X[i,j] = X[i,j]/X[j,j]; \)
        if (i==j && k<i)
            \( X[i,i]=X[i,i]-X[i,k]*X[i,k];\)
        if (i==j && j==k)
            \( X[i,i] = \sqrt(X[i,i]); \)
    }
}
Blocking with Matrix Multiplication

- Original program
  ```
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      for (k = 0; k < n; k++) {
        Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
      }
    }
  }
  ```

- Stripmine 2 outer loops
  ```
  for (ii = 0; ii < n; ii = ii+B) {
    for (i = ii; i < min(n,ii+B); i++) {
      for (jj = 0; jj < n; jj = jj+B) {
        for (j = jj; j < min(n,jj+B); j++) {
          Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
        }
      }
    }
  }
  ```

- Permute loops
  ```
  for (ii = 0; ii < n; ii = ii+B) {
    for (jj = 0; jj < n; jj = jj+B) {
      for (k = 0; k < n; k++) {
        for (i = ii; i < min(n,ii+B); i++) {
          for (j = jj; j < min(n,jj+B); j++) {
            Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
          }
        }
      }
    }
  }
  ```
Blocking

• Fully permutable loop nests can be blocked
  – Stripmine to create more fully permutable loops
  – Permutable loops can be moved inside (by definition)
• Uses
  – Increase data locality
    • Block size can be chosen
      so data accessed in the block fits in memory
  – Reduce synchronization overhead
    • By a factor of the block size
    • Consideration: startup latency, load balance for triangular loops
Pipeli ned Parallelism

1. Fully permutable loop nests & pipelining
2. Example: Transforming for full permutability
3. Time Affine Partitioning: Problem
4. Time Affine Partitioning Algorithm
5. $O(1)$ Synchronization problem
5. Beyond Pipelined Parallelism

for (i=1; i<=n; i++) {
    X[i] = Y[i] + Z[i];    \text{(s1)}
    W[A[i]] = X[i];        \text{(s2)}
}

O(1) Synchronization

for (i=1; i<=n; i++) {
    X[i] = Y[i] + Z[i];   \(s1\)
    W[A[i]] = X[i];      \(s2\)
}

- Program dependence graph
  - Nodes: statements
  - Edges: data dependence

- Split the program into
  a sequence of strongly connected components
  separated by barriers

\[
\begin{align*}
\text{for (i}=1; \text{i}<\text{n; i}++ \}\{ \\
\text{X}[i] &\text{ = Y}[i] + Z[i]; \quad (s1) \\
\text{W}[A[i]] &\text{ = X}[i]; \quad (s2)
\}\end{align*}
\]
Algorithm

1. Find parallelism with minimum synchronization

   Find outermost communication-free parallelism
   Find outermost fully permutable loop nest
   If there are inner loops remaining
       Find program dependence graph
       Split the program into strongly connected components
       Repeat for each strongly connected component

2. Blocking can be applied based on machine characteristics
Summary: Two Key Algorithms

Loops

Array

Processor ID

\[
\begin{align*}
F_1 i_1 + f_1 & \\
F_2 i_2 + f_2 & \\
C_1 i_1 + c_1 & \\
C_2 i_2 + c_2 & \\
\end{align*}
\]

\[
\begin{align*}
i_1 & \leq i_2 \\
\text{Loops} & \\
\text{Array} & \\
\end{align*}
\]

\[
\begin{align*}
F_2 i_2 + f_2 & \\
F_1 i_1 + f_1 & \\
C_1 i_1 + c_1 & \\
C_2 i_2 + c_2 & \\
\end{align*}
\]

Time Stage
Example: Neural Network

// 2D 3x3 convolution (stride=1)
for i = 0 to channels-1
    for y = 2 to Sy-1
        for x = 2 to Sx-1
            B[i,y,x] += A[i,y-2,x-2]*W1[0,0] + A[i,y-2,x-1]*W1[0,1] + ...
            A[i,y-1,x-2]*W1[1,0] + ...
            A[i,y-2,x]*W1[2,0] + ...

// ReLU (Rectified Linear Unit)
for i = 0 to channels-1
    for y = 2 to Sy-1
        for x = 2 to Sx-1
            B[i,y,x] = max(B[i,y,x], 0)

// 2D 3x3 convolution (Stride = 2)
for i = 0 to channels-1
    for y = 2 to (Sy-1)/2
        for x = 2 to (Sx-1)/2
            C[i,y,x] += B[i,2*y-2,2*x-2]*W2[0,0] + ...
            B[i,2*y-1,2*x-2]*W2[1,0] + ...

// Dense neural network layer
for i = 0 to channels-1
    for j = 0 to Sj-1
        for y = 2 to (Sy-1)/2
            for x = 2 to (Sx-1)/2
                D[i,j] += C[i,y,x]*W3[j,y,x]

// Softmax:
for i = 0 to channels-1
    for j = 0 to Sj-1
        E[i] += exp(D[i,j])
    for j = 0 to Sj-1
        F[i,j] = D[i,j]/E[i]
Parallelization without Reduction Optimization

// 2D convolution (stride=1)
for i = 0 to channels-1  // Parallel loop
for y = 2 to Sy-1      // Permutable loop nest
  for x = 2 to Sx-1   // Permutable loop nest
    // 2D convolution
    B[i,y,x] += A[i,y-2,x-2]*W1[0,0] + A[i,y-2,x-1]*W1[0,1] + ...  
      A[i,y-1,x-2]*W1[1,0] + ...  
        A[i,y,x-2]*W1[2,0]  + ...
    // ReLU (Rectified Linear Unit)
    B[i,y,x] = max(B[i,y,x], 0)

// 2D convolution (Stride = 2)
if (y >= 4) && (x >= 4) && (y mod 2 == 0) && (x mod 2 == 0)
  C[i,y/2,x/2] += B[i,y-2,x-2]*W2[0,0] + ...  
    B[i,y-1,x-2]*W2[1,0] + ...

// Dense neural network layer
for j = 0 to Sj-1  /* Parallel loop */
  for y = 2 to (Sy-1)/2
    for x = 2 to (Sx-1)/2
      D[i,j] += C[i,y,x]*W3[j,y,x]

// Softmax
E[i] += exp(D[i,j])
for j = 0 to Sj-1  /* Parallel loop */
  F[i,j] = D[i,j]/E[i]