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. Recall: Maximum Parallelism & No Communication

C: Space partitioning to Processor ID
For every pair of data dependent accesses \( F_{i1} + f_1 \) and \( F_{i2} + f_2 \)

Find \( C_1, c_1, C_2, c_2 \):

\[
\forall i_1, i_2 \quad F_{i1} + f_1 = F_{i2} + f_2 \rightarrow C_{i1} + c_1 = C_{i2} + c_2
\]

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 \( n \)
    \[ A[i,j] = c \times (A[i-1,j] + A[i,j-1]) \]

**Pipelineable Parallelism**

for \( i = 1 \) TO \( m \)
  for \( j = 1 \) TO \( n \)
    \[ A[i,j] = c \times (A[i-1,j] + A[i,j-1]) \]

Processor ID: \( p \)

Synchronization variable: \( t[p] \) initialized to 0

WAIT: thread waits until the condition becomes true

for \( j = 1 \) TO \( n \)
  if (\( p == 1 \)) or (WAIT(\( t[p-1] >= j \))
    \[ t[p]++ \]
Fully Permutable Loop Nests

- **Definition:**
  A loop nest is fully permutable if all the loops can be permuted arbitrarily without changing the semantics of the program.

- **Example:**

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

When is a Loop Fully Permutable?

- **Sequential execution order:**
  A loop nest is fully permutable if all the dependences are satisfied in the sequential execution order, under all loop permutations.

- **INTUITION:** A loop nest is fully permutable if
  - Its dependences do not point backwards along any axis.

- Relationship between communication-free parallelism & full permutability?
r-Dimensional Pipelineable Parallelism

- r-deep fully permutable loop nest, $r > 1$, with cross-iteration dependences
  - $r$ 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 $r$th loop: $i_r$
  - iteration $i_r$ for processor $(p_1, p_2, ..., p_{r-1})$, waits for iteration $i_r$ for processors $(p_{r-1}, p_2, ..., p_{r-1})$, $(p_1, p_{r-1}, ..., p_2), ... , (p_1, p_2, ..., p_{r-1})$.

Recall: Blocking for Matrix Multiplication

- Data Accessed
  - 1000 $\times$ 1000 = 1000000
  - 32 $\times$ 1000 = 32000
  - 1000 $\times$ 1000 = 1000000
  - 32 $\times$ 1000 = 32000
Experimental Results

blocking with matrix multiplication

- original program
  ```java
  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
  ```java
  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
  ```java
  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];
          }
        }
      }
    }
  }
  ```
**How to Block Loops?**

- Fully permutable loop nests can be blocked
  1. Stripmine to create more fully permutable loops
     
     ```
     for (i = 0; i < n; i++) {
         <code>
     }
     =>
     for (ii = 0; ii < n; ii = ii+B) {
         for (i = ii; i < min(n,ii+B); i++) {
             <code>
         }
     }
     
     2. Permute inner stripmined loop inside
     ```

**Uses of Blocking**

- Increase data locality
  - Block size can be chosen
  - so data accessed in the block fits in the faster hierarchy (virtual memory, cache, registers)
- Reduce synchronization overhead
  - By a factor of the block size
  - Consideration: startup latency, load balance for triangular loops
- SIMD instructions
  - To create contiguous vector access
    ```
    for (j = 0; j < n; j++) {
        for (k = 0; k < n; k++) {
            Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
        }
    }
    => for (jj = 0; jj < n; jj+=4) {
        for (k = 0; k < n; k++) {
            for (j = jj; jj < min(n, jj+4); jj++) {
                Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
            }
        }
    }
    ```
2. How to Make Loops Fully Permutable? (example)

Example:

```plaintext
for i = 0 TO m
    for j = 0 to n
        X[j+1] = (X[j] + X[j+1] + X[j+2])
```

Transforming for Full Permutability

```plaintext
for i = 0 TO m
    for j = 0 to n
        X[j+1] = (X[j] + X[j+1] + X[j+2])
```
Code Generation

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 $i'+n$
$X[j'-i'+1] = (X[j'-i'] + X[j'-i'+1] + X[j'-i'+2]) / 3$

Is the Result Fully Permutable?

$\begin{bmatrix} i' \\ j' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$
3. The Problem of Creating Fully Permutable Loops

- RECALL: r-deep fully permutable loop nest; r > 1
  - r choices of outermost loops
  - r-1 degrees of parallelism
  - O(n^r) parallelism
  - O(n) synchronization
- GOAL: Find transformation to maximize the degree of pipelining
  - 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_{i1}+f_1 and F_{i2}+f_2
Let B_{i1}+b_1 \geq 0, B_{i2}+b_2 \geq 0 be the corresponding loop bound constraints,
Find C_{i1}, C_{i2}, C_{j1}, C_{j2}:
\forall i_1, i_2 : B_{i1} + b_1 \geq 0, B_{i2} + b_2 \geq 0
(l_1 \leq l_2) \land (F_{i1}+f_1 = F_{i2}+f_2) \rightarrow C_{i1}+c_1 \leq C_{i2}+c_2
with the objective of maximizing the rank of C_{i1}, C_{i2}
Solutions of Time Mapping

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

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

Solutions to Loop Transforms

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

\[
\begin{bmatrix}
1 & 1 \\
1 & 0
\end{bmatrix}
\begin{bmatrix}
i \\
j'
\end{bmatrix}
\]
### 4. Time Partitioning Algorithm

**Loops**

\[ F_{i_1} + f_1 \quad F_{i_2} + f_2 \]

\[ C_{i_1} + c_1 \quad C_{i_2} + c_2 \]

**Array**

**Processor ID**

\[ i_1 \leq i_2 \]

**Time Stage**

---

**Comparing the Two Problems**

**Communication-Free Parallelism:**

C: Space partitioning of Computation to Processor ID

For every pair of data dependent accesses \( F_{i_1} + f_1 \) and \( F_{i_2} + f_2 \)

Find \( C_1, C_1, C_2, C_2 \):

\[ \forall i_1, i_2 \quad F_{i_1} + f_1 = F_{i_2} + f_2 \rightarrow C_{i_1} + c_1 = C_{i_2} + c_2 \]

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 \( F_{i_1} + f_1 \) and \( F_{i_2} + f_2 \)

Let \( B_{i_1} + b_1 \geq 0, B_{i_2} + b_2 \geq 0 \) be the corresponding loop bound constraints,

Find \( C_1, C_1, C_2, C_2 \):

\[ \forall i_1, i_2 \quad B_{i_1} + b_1 \geq 0, B_{i_2} + b_2 \geq 0 \]

\[ (i_1 \leq i_2) \land (F_{i_1} + f_1 = F_{i_2} + f_2) \rightarrow C_{i_1} + c_1 \leq C_{i_2} + c_2 \]

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 $A^T y = c, \quad y \geq 0$

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

Example: Cholesky Decomposition

for ($i = 1; i \leq N; i++$)
for ($j = 1; j \leq i-1; j++$)
for ($k = 1; k \leq j-1; k++$)
$X[i,j] = X[i,j] - X[i,k] * X[j,k];$
$X[i,j] = X[i,j] / X[j,j];$
for ($m=1; m<=i-1; m++$)
$X[i,i]=X[i,i]-X[i,m]*X[i,m];$
$X[i,i] = sqrt(X[i,i]);$

Transformed Space

for ($i = 1; i \leq N; i++$)
for ($j = 1; j \leq i; j++$)
for ($k = 1; k \leq j; k++$)
if ($j==k$)
$X[i,j] = X[i,j] / X[j,k];$
if ($i==j$)
$X[i,i]=X[i,i]-X[i,k]*X[i,k];$
if ($i==j$)
$X[i,i] = sqrt(X[i,i]);$
5. Beyond Pipelined Parallelism

What if there is only 1 fully permutable outermost loop?

Example:
for (i=1; i<=n; i++) {
    X[i] = Y[i] + Z[i];  (s1)
    W[A[i]] = X[i];     (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 O(1) barriers
**Algorithm**

1. Find parallelism with coarsest 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. Apply blocking to improve locality

---

**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-1,x-1]*W1[1,1] +
                       A[i,y,x-2]*W1[2,0] +
                       A[i,y,x-1]*W1[2,1] +
        // 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,2*y-2,2*x-2]*W3[j,y,x]

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

// 2D convolution (stride=1)
for i = 0 to channels-1 // Parallel loop
  for x = 2 to 5x-1 // Permutable loop next
    // 2D convolution
    B[i,y,x] = A[i,y-2,x-2]*W[0,0]+... A[i,y,x]*W[1,0]+...
    // 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]*W[0,0]+... B[i,y-1,x-2]*W[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]*W[0,j,y,x]

// Softmax
for j = 0 to Sj-1 /* Parallel loop */
  for y = 2 to (Sy-1)/2
    for x = 2 to (Sx-1)/2
      T[i,j] = exp(D[i,j])
      F[i,j] = exp(T[i,j])/E[i]