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 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 $n$

M. Lam

**CS243: Loop Transformations**
Pipelineable Parallelism

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

\[
\begin{align*}
\text{for } i &= 1 \TO m \\
\text{for } j &= 1 \text{ to } n \\
\end{align*}
\]

\[
\begin{align*}
\text{for } j &= 1 \TO n \\
\text{for } i &= 1 \text{ to } m \\
\end{align*}
\]
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 \((p_{r-1}, p_2, ..., p_1)\), \((p_1, p_{r-2}, ..., p_{r-1})\), ..., \((p_1, p_2, ..., p_{r-1})\).
Recall: Blocking for Matrix Multiplication

\[
\begin{align*}
1000 & \times 1000 = 1000 \\
32 & \times 1000 = 32
\end{align*}
\]

Data Accessed

1002000

65024
Experimental Results

- With Blocking
- Without Blocking

![Graph showing speedup with and without blocking vs processors](image)
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++) {
          for (k = 0; k < n; k++) {
            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];
          }
        }
      }
    }
  }
  ```
How to Block Loops?

- Fully permutable loop nests can be blocked
  1. Stripmine to create more fully permutable loops
     
```plaintext
for (i = 0; i < n; i++) {
  <code>
}
```

  =>

```plaintext
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); j++) {
              Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
          }
      }
      ```
    }
2. How to Make Loops Fully Permutable? (example)

Example:

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

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

\( i' = i \quad 0 \leq i' \leq m \)
\( j' = i + j \quad 0 \leq j'-i' \leq n \)
\( j = j' - i' \)

\[
\begin{bmatrix}
i' \\
j'
\end{bmatrix} =
\begin{bmatrix}
1 & 0 \\
1 & 1
\end{bmatrix}
\begin{bmatrix}
i \\
j
\end{bmatrix}
\]
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
  - $O(n^{r-1})$ 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 $C_1$, $c_1$, $C_2$, $c_2$:

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

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

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

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

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

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

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

Compare:

Loops

\[ F_1 i_1 + f_1 \]
\[ F_2 i_2 + f_2 \]
\[ C_1 i_1 + c_1 \]
\[ C_2 i_2 + c_2 \]

Processor ID

Array

\[ i_1 \leq i_2 \]

Loops

\[ F_2 i_2 + f_2 \]
\[ F_1 i_1 + f_1 \]
\[ C_1 i_1 + c_1 \]
\[ C_2 i_2 + c_2 \]

Time Stage

Array
Comparing the Two Problems

Communication-Free Parallelism:
C: Space partitioning of Computation to Processor ID
For every pair of data dependent accesses $F_{1i1}+f_1$ and $F_{2i2}+f_2$

Find $C_1$, $c_1$, $C_2$, $c_2$:
\[\forall i_1, i_2 \quad F_{1i1} + f_1 = F_{2i2} + f_2 \rightarrow C_{1i1} + c_1 = C_{2i2} + c_2\]
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 $F_{1i1}+f_1$ and $F_{2i2}+f_2$
Let $B_{1i1} + b_1 \geq 0$, $B_{2i2} + b_2 \geq 0$ be the corresponding loop bound constraints,

Find $C_1$, $c_1$, $C_2$, $c_2$:
\[\forall i_1, i_2 \quad B_{1i1} + b_1 \geq 0, \quad B_{2i2} + b_2 \geq 0\]
\[\quad (i_1 \leq i_2) \land (F_{1i1} + f_1 = F_{2i2} + f_2) \rightarrow C_{1i1} + c_1 \leq C_{2i2} + c_2\]
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
$Ax \geq 0$, $c^T x < 0$
has a real-valued solution $x$
or, the dual system
$A^T y = c$, $y \geq 0$
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)
**Example: Cholesky Decomposition**

```c
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]);
}
```

```c
for (i = 1; i <= N; i++) {
    for (j = 1; j <= i; j++) {
        for (k = 1; k <= j; 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]);
    }
}
```

Transformed Space

![Diagram showing transformed space with indices i, j, and k]
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**

```plaintext
for (i=1; i<=n; i++) {
    X[i] = Y[i] + Z[i];     \text{(s1)}
    W[A[i]] = X[i];        \text{(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,x-2]*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
        T[i,j] = exp(D[i,j]);
        E[i] += T[i,j];
        for j = 0 to Sj-1
            F[i,j] = exp([T[i,j]]/E[i])
// 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
for j = 0 to Sj-1
    T[i,j] = exp(D[i,j]);
    E[i] += T[i,j];
for j = 0 to Sj-1   /* Parallel loop */
    F[i,j] = exp([T[i,j]]/E[i]
Summary: Two Key Algorithms

Loops

Array

Processor ID

Time Stage

i_1 \leq i_2

Loops

Array