Lecture 10
Loop Transformations for Communication-Free Parallelism

1. Examples of Loop Transformations
2. Affine Partitioning
3. Code Generation

Readings: Chapter 11–11.3, 11.6–11.7.4, 11.9-11.9.6
Shared Memory Machines

Performance on Shared Address Space Multiprocessors: Parallelism & Locality

Interconnect
1. Parallelism and Locality

- Parallelism DOES NOT necessarily imply speed up!
- Key concept in manual or automatic parallelization:
  - Multiprocessor performance: Improve locality with loop transformations
    - Minimize communication
    - Operations using the same data are executed on the same processor
  - Sequential processor performance: Improve locality with loop transformations
    - Minimize cache misses
    - Operations using the same data are executed close in time
  - Shift focus from finding “independent operations” to ordering “dependent operations”
Iteration Space

FOR i = 0 to 5
  FOR j = i to 7
    ...

• n-deep loop nests: n-dimensional polytope
• Iterations: coordinates in the iteration space
• Assume: iteration index is incremented in the loop
• Sequential execution order: lexicographic order
  – [0,0], [0,1], …, [0,6], [0,7],
  [1,1], …, [1,6], [1,7], …

• A loop transform is legal
  – if all data dependences in the original loop nest are honored with sequential execution in the new loop.
Example 1

for $I = 1$ to $4$
  for $J = 1$ to $3$
    $Z[I,J] = Z[I-1,J]$

Quiz
1. Is the inner or outer loop a doall loop?
2. How to parallelize for a multiprocessor?
3. Data locality on a uniprocessor, or on a multiprocessor?
4. What is the expected performance?
Transform 1: Loop Permutation (Loop Interchange)

For $I' = 1$ to $4$
For $J = 1$ to $3$
$Z[I',J] = Z[I'-1,J]$

For $I = 1$ to $4$
For $J = 1$ to $3$
$Z[I,J] = Z[I-1,J]$

Quiz

1. Is the transform legal?
2. Is the inner loop or outer loop a doall loop in (b)?
3. Is (b) good for uniprocessor locality?
4. For multiprocessor parallelism & locality?

Key: Assign dependent operations on the same processor and close in time.
Transform 2: Loop Fusion

for I = 1 to 4
  T[I] = A[I] + B[I]  (s1)
for I’ = 1 to 4
  C[I’] = T[I’] x T[I’]  (s2)

for J = 1 to 4
  C[J] = T[J] x T[J]  (s2)
Loop Transformations

• Many loop transforms have been proposed for parallelism & locality

• \( n \)-deep loop nest transforms \( \rightarrow n \)-dimensional matrix transforms
  
  – Interchange: \[
  \begin{bmatrix}
  0 & 1 \\
  1 & 0 
  \end{bmatrix}
  \]
  
  – Skewing: \[
  \begin{bmatrix}
  1 & 0 \\
  1 & 1 
  \end{bmatrix}
  \]
  
  – Reversal: \[
  \begin{bmatrix}
  1 & 0 \\
  0 & -1 
  \end{bmatrix}
  \]
  
  – Matrix multiplication of the above

• Cross statement transforms
  
  – Loop fusion: combining loops
  
  – Loop fission: separating loops
  
  – Re-indexing: shifting operations to different iterations in a loop
Loop Transformations

• It is easy to check if a transform is legal

• But which combinations of transformations should we apply
  – Many heuristics for many years!

• Key idea:
  – Unify them with affine transforms, and solve for the unknowns!
2. Affine Partitioning: A Contrived but Illustrative Example

FOR j = 1 TO n
    FOR i = 1 TO n
        A[i,j] = A[i,j] + B[i-1,j]; \textbf{(S_1)}
        B[i,j] = A[i,j-1] * B[i,j]; \textbf{(S_2)}
2. Affine Partitioning: A Contrived but Illustrative Example

FOR $j = 1$ TO $n$
    FOR $i = 1$ TO $n$
        $A[i,j] = A[i,j] + B[i-1,j]$;  \hspace{1cm} (S_1)
        $B[i,j] = A[i,j-1] \times B[i,j]$;  \hspace{1cm} (S_2)

$p = [1 \ -1] \begin{bmatrix} i \\ j \end{bmatrix} + 0$

$p = [1 \ -1] \begin{bmatrix} i \\ j \end{bmatrix} + 1$
Generate: SPMD (Single-Program Multiple Data) Code

Let p be the processor's ID number

\[
\begin{align*}
\text{if } (1-n \leq p \leq n) \text{ then} \\
\quad \text{if } [1 \leq p) \text{ then} \\
\quad \quad B[p,1] &= A[p,0] \times B[p,1]; \quad (S_2) \\
\quad \text{for } i_1 = \text{max}[1,1+p) \text{ to } \text{min}[n,n-1+p) \text{ do} \\
\quad \quad A[i_1,i_1-p] &= A[i_1,i_1-p] + B[i_1-1,i_1-p]; \quad (S_1) \\
\quad \quad B[i_1,i_1-p+1] &= A[i_1,i_1-p] \times B[i_1,i_1-p+1]; \quad (S_2) \\
\text{if } (p \leq 0) \text{ then} \\
\quad A[n+p,n] &= A[n+p,N] + B[n+p-1,n]; \quad (S_1)
\end{align*}
\]
Communication-Free Parallelization Algorithm

• Key Constraint: Communication-free parallelism => Operations using the same data must be mapped to the same processor!

• Two-step Algorithm

1. Given affine array indices in programs with arbitrarily nested loops
   – Find an affine mapping for each instruction to a processor ID such that there is no communication between processors
   – Eg:
     \[ S_1 \; p = [1 \; -1] \begin{bmatrix} i \\ j \end{bmatrix} + 0 \]
     \[ S_2 \; p = [1 \; -1] \begin{bmatrix} i \\ j \end{bmatrix} + 1 \]

2. Generate the SPMD code
Maximum Parallelism & No Communication

Let $i_1$ and $i_2$ be loop indices of 2 (not necessarily distinct) loops

For every pair of *data dependent* accesses $F_{1i_1}+f_1$ and $F_{2i_2}+f_2$

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

$$\forall \ i_1, i_2 \quad F_{1i_1}+f_1 = F_{2i_2}+f_2 \rightarrow C_{1i_1}+c_1 = C_{2i_2}+c_2$$

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

**Quiz:** Can you always find such a mapping?
Rank of Partitioning = Degree of Parallelism

Affine Mapping

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

Rank

0
1
2

iterations

Mapped to same processor
Solving Equations Give the Desired Partitioning (in the book)

FOR j = 1 TO n
    FOR i = 1 TO n
        A[i,j] = A[i,j] + B[i-1,j];  \hfill (S_1)
        B[i,j] = A[i,j-1] * B[i,j];  \hfill (S_2)

\[ p = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + 0 \]

\[ p = \begin{bmatrix} 1 & -1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + 1 \]
3. Step 2: Code Generation

• Naive
  – Each processor visits all the iterations in the original execution order
  – Executes only if it owns that iteration
  – Quiz: why is it correct?

• Optimization
  – Removes unnecessary looping and condition evaluation
Example 1: Loop Permutation

1. Find affine partitioning: \( c_1, c_2, c_0 \) such that

\[
p = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + c_0
\]

2. Suppose iteration \( i, j \) & \( i', j' \) refer to same location

\[
i = i' - 1
\]
\[
j = j'
\]

3. No communication means:

\[
c_1 i + c_2 j + c_0 = c_1 i' + c_2 j' + c_0
\]
\[
c_1(i'-1) + c_2 j' + c_0 = c_1 i' + c_2 j' + c_0
\]

\[
c_1 = 0
\]
\[
p = c_2 j + c_0
\]

4. Pick simplest \( c_0 \) and non-zero \( c_2 \):

\[
c_2 = 1, c_0 = 0
\]
\[
p = j
\]
Example 1: Code Generation

**Initial Code:**

for I = 1 to 4
  for J = 1 to 3
    Z[I,J] = Z[I-1,J]

for I = 1 to 4
  for J = 1 to 3
    Z[I,J] = Z[I-1,J]

**Optimized Code:**

for P = 1 to 3
  for I = 1 to 4
    Z[I,P] = Z[I-1,P]

**SPMD (single program multiple data) code:**

for I = 1 to 4
  Z[I,P] = Z[I-1,P]
Equivalent to Loop Permutation (Loop Interchange)

Our algorithm performs the same effect in 2 steps (partitioning + code generation)
Useful for handling multiple statements in a program
Example 2: Loop Fusion

1. Find affine partitioning: $c_{1,1}, c_{1,0}, c_{2,1}, c_{1,0}$, such that

   $s1: \left[ p \right] = \left[ c_{1,1} \right] \left[ i \right] + c_{1,0}$
   $s2: \left[ p \right] = \left[ c_{2,1} \right] \left[ i' \right] + c_{2,0}$

2. Suppose iteration $i$ & $i'$ refer to the same location
   $i = i'$

3. No communication means:
   $c_{1,1} i + c_{1,0} = c_{2,1} i' + c_{2,0}$

   $c_{1,1} = c_{2,1}$
   $c_{1,0} = c_{2,0}$

4. Pick simplest, non-zero values for $c_{1,1}, c_{2,1}$:
   $c_{1,1} = c_{2,1} = 1, c_{1,0} = c_{2,0} = 0$
   $p = i; p = i'$

Quiz: How would you transform this code?
Loop Fusion

for I = 1 to 4
    T[I] = A[I] + B[I]  (s1)
for I' = 1 to 4
    C[I'] = T[I'] x T[I']  (s2)

\[ p = (I[I]) \]

s1: \[ p = [1][i] \]
s2: \[ p = [1][i'] \]
Example 3

Do you know how the algorithm works without simulating it?

for I = 1 to 4
T[I] = A[I] + B[I] \hspace{1cm} (s1)

for I' = 1 to 4
C[I'] = T[5-I'] \times T[5-I'] \hspace{1cm} (s2)

Homework: Derive the answer by simulating the affine partitioning algorithm.
Example 4: 2 Nested, DoAll Loops

1. Find affine partitioning: $c_1, c_2, c_0$ such that

\[ p = \begin{bmatrix} c_1 & c_2 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix} + c_0 \]

2. Suppose iteration $i,j$ & $i', j'$ refer to same location

   \[ i = i' \]
   \[ j = j' \]

3. No communication means:

   \[ c_1 i + c_2 j + c_0 = c_1 i' + c_2 j' + c_0 \]
   \[ c_1 i' + c_2 j' + c_0 = c_1 i' + c_2 j' + c_0 \]

4. No constraints

   Two basis vectors: $[c_1 \ c_2] = [1 \ 0]$, or $[c_1 \ c_2] = [0 \ 1]$

   Two answers for $p$: two degrees of parallelism

\[
\begin{bmatrix}
  p_1 \\
  p_2
\end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}
\]
Example 4: 2 Nested DoAll Loops

for I = 1 to 4
  for J = 1 to 3
    $Z[I,J] = Z[I,J] + 1$

for p1 = 1 to 4
  for p2 = 1 to 3
  for I = 1 to 4
    for J = 1 to 3
      if (I == p1 & J == p2)
        $Z[I,J] = Z[I,J] + 1$

Optimization

$\begin{bmatrix} p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} i \\ j \end{bmatrix}$

for p1 = 1 to 4
  for p2 = 1 to 3
    $Z[p1,p2] = Z[p1,p2] + 1$
Example 5: Imperfectly Nested Loops

for I = 1 to 4
    Z[I,1] = Y[I];
for J = 1 to 3
    Z[I,J] = Z[I,J]+1

\[
\begin{bmatrix}
p_1 \\
p_2
\end{bmatrix} = \begin{bmatrix}1 & 0 \end{bmatrix} i + \begin{bmatrix}0 & 1 \end{bmatrix}
\]

for p1 = 1 to 4
    for p2 = 1 to 3
        if (p1 == I & 1 == J)
            Z[p1,1] = Y[p1]
        if (p1 == I & J == p2)
            Z[p1,J] = Z[p1,J]+1

Optimization

for p1 = 1 to 4
    for p2 = 1 to 3
        if p2 == 1
            Z[p1,1] = Y[p1]
        if p2 == 1
            Z[p1,p2] = Z[p1,p2]+1

M. Lam
CS243: Loop Transformations
**Optimizing Arbitrary Loop Nesting Using Affine Partitions (chotst, NAS)**

```
DO 1 J = 0, N
  I0 = MAX ( -M, -J )
  DO 2 I = I0, -1
    DO 3 JJ = I0 - I, -1
      DO 3 L = 0, NMAT
    DO 2 L = 0, NMAT
  A(L,I,J) = A(L,I,J) * A(L,0,I+J)
  DO 4 L = 0, NMAT
    EPSS(L) = EPS * A(L,0,J)
  DO 5 JJ = I0, -1
    DO 5 L = 0, NMAT
      A(L,0,J) = A(L,0,J) - A(L,JJ,J) ** 2
    DO 4 L = 0, NMAT
  A(L,0,J) = 1. / SQRT ( ABS (EPSS(L) + A(L,0,J)) )
  DO 6 I = 0, NRHS
    DO 7 K = 0, N
      DO 8 L = 0, NMAT
        B(I,L,K) = B(I,L,K) * A(L,0,K)
      DO 7 JJ = 1, MIN (M, N-K)
        DO 7 L = 0, NMAT
          B(I,L,K+JJ) = B(I,L,K+JJ) - A(L,-JJ,K+JJ) * B(I,L,K)
        DO 6 K = N, 0, -1
          DO 9 L = 0, NMAT
            B(I,L,K) = B(I,L,K) * A(L,0,K)
          DO 6 JJ = 1, MIN (M, K)
            DO 6 L = 0, NMAT
              B(I,L,K-JJ) = B(I,L,K-JJ) - A(L,-JJ,K) * B(I,L,K)
```

*Result of affine transform: 1 outermost loop parallelism: index \( L \)*
Chotst: Results with Affine Partitioning + Blocking

(Unimodular: a subset of affine partitioning for perfect loop nests)

![Graph showing speedup vs. number of processors for Unimodular + Blocking and Affine Partitioning + Blocking]
Maximum Parallelism & No Communication

Let \( i_1 \) and \( i_2 \) be loop indices of 2 (not necessarily distinct) loops.

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 \( C_1, C_2 \).
General Lesson

• Parallelizing compiler research
  – Compiler research
    • Studying examples to inspire code transforms
    • Years of heuristic transforms at the source level
  – Parallel algorithm research (systolic arrays)
    • Geometric transforms to map computation onto VLSI while minimizing communication
    • Inspired matrix transformation approach
  – Combining two research threads
    • Generalize matrix approach to arbitrary loop nesting
    Most influential paper award, 2001

• Solving math equations is easy for compilers!
• But … how many programs have communication-free parallelism?
  – An important building block for parallelism with communication
  – The full story in the next class – that took months to invent!