6.189 IAP 2007

Lecture 11

Parallelizing Compilers
Outline

- Parallel Execution
- Parallelizing Compilers
- Dependence Analysis
- Increasing Parallelization Opportunities
- Generation of Parallel Loops
- Communication Code Generation
Types of Parallelism

- Instruction Level Parallelism (ILP) ➞ Scheduling and Hardware
- Task Level Parallelism (TLP) ➞ Mainly by hand
- Loop Level Parallelism (LLP) or Data Parallelism ➞ Hand or Compiler Generated
- Pipeline Parallelism ➞ Hardware or Streaming
- Divide and Conquer Parallelism ➞ Recursive functions
Why Loops?

- 90% of the execution time in 10% of the code
  - Mostly in loops

- If parallel, can get good performance
  - Load balancing

- Relatively easy to analyze
Programmer Defined Parallel Loop

- **FORALL**
  - No “loop carried dependences”
  - Fully parallel

- **FORACROSS**
  - Some “loop carried dependences”

Prof. Saman Amarasinghe, MIT.
Parallel Execution

- **Example**
  
  ```plaintext
  FORPAR I = 0 to N
  ```

- **Block Distribution: Program gets mapped into**
  
  ```plaintext
  Iters = ceiling(N/NUMPROC);
  FOR P = 0 to NUMPROC-1
    FOR I = P*Iters to MIN((P+1)*Iters, N)
  ```

- **SPMD (Single Program, Multiple Data) Code**
  
  ```plaintext
  If(myPid == 0) {
    ...
    Iters = ceiling(N/NUMPROC);
  }
  Barrier();
  FOR I = myPid*Iters to MIN((myPid+1)*Iters, N)
  Barrier();
  ```
Parallel Execution

- **Example**
  
  ```c
  FORPAR I = 0 to N
  ```

- **Block Distribution:** Program gets mapped into
  
  ```c
  Iters = ceiling(N/NUMPROC);
  FOR P = 0 to NUMPROC-1
      FOR I = P*Iters to MIN((P+1)*Iters, N)
  ```

- **Code that fork a function**
  
  ```c
  Iters = ceiling(N/NUMPROC);
  ParallelExecute(func1);
  ... 
  void func1(integer myPid)
  {
      FOR I = myPid*Iters to MIN((myPid+1)*Iters, N)
  }```
Outline

- Parallel Execution
- **Parallelizing Compilers**
- Dependence Analysis
- Increasing Parallelization Opportunities
- Generation of Parallel Loops
- Communication Code Generation
Parallelizing Compilers

● Finding FORALL Loops out of FOR loops

● Examples

    FOR I = 0 to 5

    FOR I = 0 to 5

    For I = 0 to 5
Iteration Space

- N deep loops → n-dimensional discrete cartesian space
  - Normalized loops: assume step size = 1

```plaintext
FOR I = 0 to 6
  FOR J = I to 7
```

- Iterations are represented as coordinates in iteration space
  - \( \overline{i} = [i_1, i_2, i_3, \ldots, i_n] \)
Iteration Space

- N deep loops $\rightarrow$ n-dimensional discrete cartesian space
  - Normalized loops: assume step size = 1

FOR $I = 0$ to 6
  FOR $J = I$ to 7

- Iterations are represented as coordinates in iteration space
- Sequential execution order of iterations $\Rightarrow$ Lexicographic order
  
  $[0,0], [0,1], [0,2], \ldots, [0,6], [0,7], [1,1], [1,2], \ldots, [1,6], [1,7], [2,2], \ldots, [2,6], [2,7], \ldots, [6,6], [6,7]$
Iteration Space

- N deep loops → n-dimensional discrete cartesian space
  - Normalized loops: assume step size = 1

```
FOR I = 0 to 6
  FOR J = I to 7
```

- Iterations are represented as coordinates in iteration space
- Sequential execution order of iterations
  - Lexicographic order
- Iteration $i$ is lexicographically less than $j$, $i < j$ iff there exists $c$ s.t. $i_1 = j_1$, $i_2 = j_2$, ..., $i_{c-1} = j_{c-1}$ and $i_c < j_c$
Iteration Space

- N deep loops $\rightarrow$ n-dimensional discrete cartesian space
  - Normalized loops: assume step size = 1

```
FOR I = 0 to 6
  FOR J = I to 7
```

- An affine loop nest
  - Loop bounds are integer linear functions of constants, loop constant variables and outer loop indexes
  - Array accesses are integer linear functions of constants, loop constant variables and loop indexes
Iteration Space

- N deep loops $\rightarrow$ n-dimensional discrete cartesian space
  - Normalized loops: assume step size = 1

FOR $I = 0$ to 6  
  FOR $J = I$ to 7

- Affine loop nest $\rightarrow$ Iteration space as a set of linear inequalities
  
  $0 \leq I$
  $I \leq 6$
  $I \leq J$
  $J \leq 7$
Data Space

- M dimensional arrays $\rightarrow$ m-dimensional discrete cartesian space
  - a hypercube

**Integer A(10)**

**Float B(5, 6)**
Dependences

- True dependence
  \[ a = a \]

- Anti dependence
  \[ a = a \]

- Output dependence
  \[ a = a \]

- Definition:
  Data dependence exists for a dynamic instance \( i \) and \( j \) iff
  - either \( i \) or \( j \) is a write operation
  - \( i \) and \( j \) refer to the same variable
  - \( i \) executes before \( j \)

- How about array accesses within loops?
Outline

- Parallel Execution
- Parallelizing Compilers
- **Dependence Analysis**
- Increasing Parallelization Opportunities
- Generation of Parallel Loops
- Communication Code Generation
Array Accesses in a loop

FOR I = 0 to 5

Iteration Space
0 1 2 3 4 5

Data Space
0 1 2 3 4 5 6 7 8 9 10 11 12
Array Accesses in a loop

FOR $I = 0$ to $5$
Array Accesses in a loop

FOR I = 0 to 5

iteration Space

0 1 2 3 4 5

Data Space

0 1 2 3 4 5 6 7 8 9 10 11 12
Array Accesses in a loop

FOR I = 0 to 5

Iteration Space

Data Space
Array Accesses in a loop

FOR I = 0 to 5


Iteration Space

Data Space
Recognizing FORALL Loops

- Find data dependences in loop
  - For every pair of array accesses to the same array
    - If the first access has at least one dynamic instance (an iteration) in which it refers to a location in the array that the second access also refers to in at least one of the later dynamic instances (iterations).
    - Then there is a data dependence between the statements
  - (Note that same array can refer to itself – output dependences)

- Definition
  - Loop-carried dependence: dependence that crosses a loop boundary

- If there are no loop carried dependences \( \Rightarrow \) parallelizable
Data Dependence Analysis

- **Example**
  
  ```
  FOR I = 0 to 5
  ```

- Is there a loop-carried dependence between A[I+1] and A[I]
  
  - Is there two distinct iterations $i_w$ and $i_r$ such that $A[i_w+1]$ is the same location as $A[i_r]$.
  
  $\exists$ integers $i_w, i_r$ where $0 \leq i_w, i_r \leq 5$ and $i_w \neq i_r$ with $i_w + 1 = i_r$.

- Is there a dependence between A[I+1] and A[I+1]
  
  - Is there two distinct iterations $i_1$ and $i_2$ such that $A[i_1+1]$ is the same location as $A[i_2+1]$.
  
  $\exists$ integers $i_1, i_2$ where $0 \leq i_1, i_2 \leq 5$ and $i_1 \neq i_2$ with $i_1 + 1 = i_2 + 1$. 

Integer Programming

● Formulation

■ \( \exists \) an integer vector \( \overline{i} \) such that \( \overline{A} \overline{i} \leq \overline{b} \) where
  \( \overline{A} \) is an integer matrix and \( \overline{b} \) is an integer vector

● Our problem formulation for \( A[i] \) and \( A[i+1] \)

■ \( \exists \) integers \( i_w, i_r \) \( 0 \leq i_w, i_r \leq 5 \) \( i_w \neq i_r \) \( i_w + 1 = i_r \)
■ \( i_w \neq i_r \) is not an affine function
  – divide into 2 problems
  – Problem 1 with \( i_w < i_r \) and problem 2 with \( i_r < i_w \)
  – If either problem has a solution \( \Rightarrow \) there exists a dependence

■ How about \( i_w + 1 = i_r \)
  – Add two inequalities to single problem
    \( i_w + 1 \leq i_r \) and \( i_r \leq i_w + 1 \)
Integer Programming Formulation

● Problem 1

\[ \begin{align*}
0 & \leq i_w \\
i_w & \leq 5 \\
0 & \leq i_r \\
i_r & \leq 5 \\
i_w & < i_r \\
i_w + 1 & \leq i_r \\
i_r & \leq i_w + 1
\end{align*} \]
Integer Programming Formulation

- **Problem 1**

  \[
  0 \leq i_w \quad \rightarrow \quad -i_w \leq 0 \\
  i_w \leq 5 \quad \rightarrow \quad i_w \leq 5 \\
  0 \leq i_r \quad \rightarrow \quad -i_r \leq 0 \\
  i_r \leq 5 \quad \rightarrow \quad i_r \leq 5 \\
  i_w < i_r \quad \rightarrow \quad i_w - i_r \leq -1 \\
  i_w + 1 \leq i_r \quad \rightarrow \quad i_w - i_r \leq -1 \\
  i_r \leq i_w + 1 \quad \rightarrow \quad -i_w + i_r \leq 1
  \]
Integer Programming Formulation

- Problem 1
  
  - $0 \leq i_w$ \implies -$i_w \leq 0$
  - $i_w \leq 5$ \implies $i_w \leq 5$
  - $0 \leq i_r$ \implies -$i_r \leq 0$
  - $i_r \leq 5$ \implies $i_r \leq 5$
  - $i_w < i_r$ \implies $i_w - i_r \leq -1$
  - $i_w + 1 \leq i_r$ \implies $i_w - i_r \leq -1$
  - $i_r \leq i_w + 1$ \implies -$i_w + i_r \leq 1$

- and problem 2 with $i_r < i_w$
Generalization

- An affine loop nest
  \[
  \text{FOR } i_1 = f_{11}(c_1...c_k) \text{ to } I_{u1}(c_1...c_k) \\
  \text{FOR } i_2 = f_{12}(i_1,c_1...c_k) \text{ to } I_{u2}(i_1,c_1...c_k) \\
  \text{......} \\
  \text{FOR } i_n = f_{1n}(i_1...i_{n-1},c_1...c_k) \text{ to } I_{un}(i_1...i_{n-1},c_1...c_k) \\
  A[f_{a1}(i_1...i_n,c_1...c_k), f_{a2}(i_1...i_n,c_1...c_k), ..., f_{am}(i_1...i_n,c_1...c_k)]
  \]

- Solve 2*n problems of the form
  - \(i_1 = j_1, i_2 = j_2, \ldots, i_{n-1} = j_{n-1}, i_n < j_n\)
  - \(i_1 = j_1, i_2 = j_2, \ldots, i_{n-1} = j_{n-1}, j_n < i_n\)
  - \(i_1 = j_1, i_2 = j_2, \ldots, i_{n-1} < j_{n-1}\)
  - \(i_1 = j_1, i_2 = j_2, \ldots, j_{n-1} < i_{n-1}\)
  - \(i_1 = j_1, i_2 < j_2\)
  - \(i_1 = j_1, j_2 < i_2\)
  - \(i_1 < j_1\)
  - \(j_1 < i_1\)
Multi-Dimensional Dependence

FOR I = 1 to n
    FOR J = 1 to n
Multi-Dimensional Dependence

FOR I = 1 to n
    FOR J = 1 to n

FOR I = 1 to n
    FOR J = 1 to n
What is the Dependence?

FOR $I = 1$ to $n$
    FOR $J = 1$ to $n$

FOR $I = 1$ to $n$
    FOR $J = 1$ to $n$
        $B[I] = B[I-1] + 1$
What is the Dependence?

FOR $I = 1$ to $n$
    FOR $J = 1$ to $n$

FOR $I = 1$ to $n$
    FOR $J = 1$ to $n$
What is the Dependence?

FOR \( I = 1 \) to \( n \)
FOR \( J = 1 \) to \( n \)

FOR \( I = 1 \) to \( n \)
FOR \( J = 1 \) to \( n \)
\[ B[I] = B[I-1] + 1 \]
Outline

- Parallel Execution
- Parallelizing Compilers
- Dependence Analysis
- Increasing Parallelization Opportunities
- Generation of Parallel Loops
- Communication Code Generation
Increasing Parallelization Opportunities

- Scalar Privatization
- Reduction Recognition
- Induction Variable Identification
- Array Privatization
- Interprocedural Parallelization
- Loop Transformations
- Granularity of Parallelism
Scalar Privatization

● Example

```
FOR i = 1 to n
    X = A[i] * 3;
    B[i] = X;
```

● Is there a loop carried dependence?
● What is the type of dependence?
Privatization

● **Analysis:**
  ■ Any anti- and output- loop-carried dependences

● **Eliminate by assigning in local context**

```c
FOR i = 1 to n
    integer Xtmp;
    Xtmp = A[i] * 3;
    B[i] = Xtmp;
```

● **Eliminate by expanding into an array**

```c
FOR i = 1 to n
    Xtmp[i] = A[i] * 3;
    B[i] = Xtmp[i];
```
Privatization

- Need a final assignment to maintain the correct value after the loop nest

- Eliminate by assigning in local context
  ```
  FOR i = 1 to n
    integer Xtmp;
    Xtmp = A[i] * 3;
    B[i] = Xtmp;
    if(i == n) X = Xtmp
  ```

- Eliminate by expanding into an array
  ```
  FOR i = 1 to n
    Xtmp[i] = A[i] * 3;
    B[i] = Xtmp[i];
    X = Xtmp[n];
  ```
Another Example

- How about loop-carried true dependences?
- Example
  
  ```
  FOR i = 1 to n
      X = X + A[i];
  ```

- Is this loop parallelizable?
Reduction Recognition

- **Reduction Analysis:**
  - Only associative operations
  - The result is never used within the loop

- **Transformation**
  
  ```c
  Integer Xtmp[NUMPROC];
  Barrier();
  FOR i = myPid*Iters to MIN((myPid+1)*Iters, n)
    Xtmp[myPid] = Xtmp[myPid] + A[i];
  Barrier();
  If(myPid == 0) {
    FOR p = 0 to NUMPROC-1
      X = X + Xtmp[p];
  ...
Induction Variables

● Example
  FOR i = 0 to N
  A[i] = 2^i;

● After strength reduction
  t = 1
  FOR i = 0 to N
  A[i] = t;
  t = t*2;

● What happened to loop carried dependences?

● Need to do opposite of this!
  ■ Perform induction variable analysis
  ■ Rewrite IVs as a function of the loop variable
Array Privatization

● Similar to scalar privatization

● However, analysis is more complex
  ■ Array Data Dependence Analysis: Checks if two iterations access the same location
  ■ Array Data Flow Analysis: Checks if two iterations access the same value

● Transformations
  ■ Similar to scalar privatization
  ■ Private copy for each processor or expand with an additional dimension
Interprocedural Parallelization

● Function calls will make a loop unparallelizable
  ■ Reduction of available parallelism
  ■ A lot of inner-loop parallelism

● Solutions
  ■ Interprocedural Analysis
  ■ Inlining
Interprocedural Parallelization

● Issues
  ■ Same function reused many times
  ■ Analyze a function on each trace → Possibly exponential
  ■ Analyze a function once → unrealizable path problem

● Interprocedural Analysis
  ■ Need to update all the analysis
  ■ Complex analysis
  ■ Can be expensive

● Inlining
  ■ Works with existing analysis
  ■ Large code bloat → can be very expensive
A loop may not be parallel as is

**Example**

```plaintext
FOR i = 1 to N-1
    FOR j = 1 to N-1
```
Loop Transformations

- A loop may not be parallel as is

Example

\[
\text{FOR } i = 1 \text{ to } N-1 \\
\text{ FOR } j = 1 \text{ to } N-1 \\
\]

- After loop Skewing

\[
\text{FOR } i = 1 \text{ to } 2*N-3 \\
\quad \text{FORPAR } j = \max(1,i-N+2) \text{ to } \min(i, N-1) \\
\qquad A[i-j+1,j] = A[i-j+1,j-1] + A[i-j,j];
\]
Granularity of Parallelism

- **Example**
  ```
  FOR i = 1 to N-1
    FOR j = 1 to N-1
  ```
- **Gets transformed into**
  ```
  FOR i = 1 to N-1
    Barrier();
    FOR j = 1+ myPid*Iters to MIN((myPid+1)*Iters, n-1)
    Barrier();
  ```
- **Inner loop parallelism can be expensive**
  - Startup and teardown overhead of parallel regions
  - Lot of synchronization
  - Can even lead to slowdowns
Granularity of Parallelism

- Inner loop parallelism can be expensive

- Solutions
  - Don’t parallelize if the amount of work within the loop is too small
  - Transform into outer-loop parallelism
Outer Loop Parallelism

- **Example**
  
  ```
  FOR i = 1 to N-1
    FOR j = 1 to N-1
  ```

- **After Loop Transpose**
  
  ```
  FOR j = 1 to N-1
    FOR i = 1 to N-1
  ```

- **Get mapped into**
  
  ```
  Barrier();
  FOR j = 1+ myPid*Iters to MIN((myPid+1)*Iters, n-1)
    FOR i = 1 to N-1
  Barrier();
  ```
Outline

- Parallel Execution
- Parallelizing Compilers
- Dependence Analysis
- Increasing Parallelization Opportunities
- **Generation of Parallel Loops**
- Communication Code Generation
Generating Transformed Loop Bounds

for $i = 1$ to $n$ do
  $X[i] = ...$
  for $j = 1$ to $i - 1$ do
    $... = X[j]$

- Assume we want to parallelize the $i$ loop

- What are the loop bounds?

- Use Projections of the Iteration Space
  - Fourier-Motzkin Elimination Algorithm

\[
\begin{align*}
(p, i, j) & : 1 \leq i \leq n \\
& \quad 1 \leq j \leq i - 1 \\
& \quad i = p
\end{align*}
\]
Space of Iterations

\[
\begin{align*}
\text{for } p = 2 \text{ to } n \text{ do } \\
i = p \\
\text{for } j = 1 \text{ to } i - 1 \text{ do }
\end{align*}
\]
for p = 2 to n do
i = p
for j = 1 to i - 1 do
Projections

for p = 2 to n do

    i = p

    for j = 1 to i - 1 do

        p = my_pid()
        if p >= 2 and p <= n then
            i = p
            for j = 1 to i - 1 do
Fourier Motzkin Elimination

1 ≤ i ≤ n
1 ≤ j ≤ i-1
i = p

- Project i → j → p

- Find the bounds of i
  1 ≤ i
  j+1 ≤ i
  p ≤ i
  i ≤ n
  i ≤ p

i: max(1, j+1, p) to min(n, p)
i: p

- Eliminate i
  1 ≤ n
  j+1 ≤ n
  p ≤ n
  1 ≤ p
  j+1 ≤ p
  p ≤ p
  1 ≤ j

- Eliminate redundant
  p ≤ n
  1 ≤ p
  j+1 ≤ p
  1 ≤ j

- Continue onto finding bounds of j
Fourier Motzkin Elimination

- $p \leq n$
- $1 \leq p$
- $j+1 \leq p$
- $1 \leq j$

- **Find the bounds of** $j$
  - $1 \leq j$
  - $j \leq p - 1$

  **j:** $1$ to $p - 1$

- **Eliminate** $j$
  - $1 \leq p - 1$
  - $p \leq n$
  - $1 \leq p$

- **Eliminate redundant**
  - $2 \leq p$
  - $p \leq n$

- **Find the bounds of** $p$
  - $2 \leq p$
  - $p \leq n$

  **p:** $2$ to $n$

- $p = \text{my\_pid}()$
- if $p \geq 2$ and $p \leq n$ then
  - for $j = 1$ to $p - 1$ do
    - $i = p$
Outline

● Parallel Execution
● Parallelizing Compilers
● Dependence Analysis
● Increasing Parallelization Opportunities
● Generation of Parallel Loops
● Communication Code Generation
Communication Code Generation

- **Cache Coherent Shared Memory Machine**
  - Generate code for the parallel loop nest

- **No Cache Coherent Shared Memory or Distributed Memory Machines**
  - Generate code for the parallel loop nest
  - Identify communication
  - Generate communication code
Identify Communication

● Location Centric
  ■ Which locations written by processor 1 is used by processor 2?
  ■ Multiple writes to the same location, which one is used?
  ■ Data Dependence Analysis

● Value Centric
  ■ Who did the last write on the location read?
    – Same processor → just read the local copy
    – Different processor → get the value from the writer
    – No one → Get the value from the original array
Last Write Trees (LWT)

- Input: Read access and write access(es)

```plaintext
for i = 1 to n do
    for j = 1 to n do
        A[j] = ...
        ... = X[j-1]
```

- Output: a function mapping each read iteration to a write creating that value

![Diagram of Last Write Trees (LWT)]
The Combined Space

- Receive iterations: \( 1 \leq i_{\text{recv}} \leq n \)
- Last-write relation: \( 0 \leq j_{\text{recv}} \leq i_{\text{recv}} - 1 \)
- Send iterations: \( i_{\text{send}} = i_{\text{recv}} \)
- Computation decomposition for: \( P_{\text{recv}} = i_{\text{recv}} \)
- Non-local communication: \( P_{\text{recv}} \neq P_{\text{send}} \)
for i = 1 to n do
  for j = 1 to n do
    A[j] = ...
    ... = X[j-1]

\begin{align*}
1 \leq i_{recv} &\leq n \\
0 \leq j_{recv} &\leq i_{recv} - 1 \\
i_{send} &= i_{recv} \\
P_{recv} &= i_{recv} \\
P_{send} &= i_{send} \\
P_{recv} &\neq P_{send}
\end{align*}
Communication Loop Nests

**Send Loop Nest**

\[
\text{for } p_{send} = 1 \text{ to } n - 1 \text{ do} \\
\quad i_{send} = p_{send} \\
\quad \text{for } p_{recv} = i_{send} + 1 \text{ to } n \text{ do} \\
\quad \quad i_{recv} = p_{recv} \\
\quad \quad j_{recv} = i_{send} \\
\quad \quad \text{send } X[i_{send}] \text{ to iteration } (i_{recv}, j_{recv}) \text{ in processor } p_{recv}
\]

**Receive Loop Nest**

\[
\text{for } p_{recv} = 2 \text{ to } n \text{ do} \\
\quad i_{recv} = p_{recv} \\
\quad \text{for } j_{recv} = 1 \text{ to } i_{recv} - 1 \text{ do} \\
\quad \quad p_{send} = j_{recv} \\
\quad \quad i_{send} = p_{send} \\
\quad \quad \text{receive } X[j_{recv}] \text{ from iteration } i_{send} \text{ in processor } p_{send}
\]

Prof. Saman Amarasinghe, MIT.
Merging Loops

- Computation
- Send
- Recv

Iterations

Prof. Saman Amarasinghe, MIT.
if \( p == 1 \) then
\[
X[p] = ... \\
\text{for } pr = p + 1 \text{ to } n \text{ do} \\
\quad \text{send } X[p] \text{ to iteration } (pr, p) \text{ in processor } pr
\]

if \( p >= 2 \) and \( p <= n - 1 \) then
\[
X[p] = ... \\
\text{for } pr = p + 1 \text{ to } n \text{ do} \\
\quad \text{send } X[p] \text{ to iteration } (pr, p) \text{ in processor } pr \\
\text{for } j = 1 \text{ to } p - 1 \text{ do} \\
\quad \text{receive } X[j] \text{ from iteration } (j) \text{ in processor } j \\
\quad ... = X[j]
\]

if \( p == n \) then
\[
X[p] = ... \\
\text{for } j = 1 \text{ to } p - 1 \text{ do} \\
\quad \text{receive } X[j] \text{ from iteration } (j) \text{ in processor } j \\
\quad ... = X[j]
\]
Communication Optimizations

- Eliminating redundant communication
- Communication aggregation
- Multi-cast identification
- Local memory management
Summary

- **Automatic parallelization of loops with arrays**
  - Requires Data Dependence Analysis
  - Iteration space & data space abstraction
  - An integer programming problem

- **Many optimizations that’ll increase parallelism**

- **Transforming loop nests and communication code generation**
  - Fourier-Motzkin Elimination provides a nice framework