CS 314 Principles of Programming Languages

Lecture 23

Zheng Zhang

Department of Computer Science
Rutgers University

Friday 2\textsuperscript{nd} December, 2016
Class Information

- Project 2 deadline extended to 12/7 Wednesday, 11:55pm EST.
- Project 3 will be posted 12/8 Thursday.
- For those who are taking CS 214 at the same time, the CS 214 instructors will arrange a makeup exam due to the exact time conflict.
Review: Dependence and Parallelization

Programming with Concurrency:

- A PROCESS or THREAD is an independent execution context.
- Classic von Neumann model of computing has single thread control, however, parallel programs have more than one.

Question:

How to decompose a sequential task into multiple independent subtasks?
Review: Dependence and Parallelization

- Dependence analysis is key to task decomposition.
- A task decomposition problem can be modeled as a directed graph such that if $A \rightarrow B$ the result of task $A$ is required for the processing of task $B$.

Example:

\[
\begin{align*}
C &= 4; \\
D &= 2; \\
B &= 5 + D; \\
A &= B + C;
\end{align*}
\]
Statement Level Parallelism

Dependence relation: all statement–to–statement execution orderings for a sequential program that must be preserved if the meaning of the program is to remain the same.

<table>
<thead>
<tr>
<th>S1</th>
<th>pi = 3.14</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>r = 5.0</td>
</tr>
<tr>
<td>S3</td>
<td>area = pi * r**2</td>
</tr>
</tbody>
</table>

Parallelization is a type of program reordering transformation.

For any type of program reordering:

**Semantics is preserved if dependence is preserved.**
Dependence — Overview

Bernstein’s Condition: — There is a data dependence from statement \( S_1 \) to statement \( S_2 \) \( (S_1 \delta S_2) \) if

1. Both statements access the same memory location, and
2. One of them is a write.
3. There is a run–time execution path from \( S_1 \) to \( S_2 \).
Data dependence classification

“$S_2$ depends on $S_1$” — $S_1 \delta S_2$

**True dependence** occurs when $S_1$ writes a memory location that $S_2$ later reads.

**Anti dependence** occurs when $S_1$ reads a memory location that $S_2$ later writes.

**Output dependence** occurs when $S_1$ writes a memory location that $S_2$ later writes.

**Input dependence** occurs when $S_1$ reads a memory location that $S_2$ later reads.

Note: Input dependences do not restrict statement (load/store) order!
Loop-level Parallelism

We will use loop analysis as example to describe dependence analysis and parallelization.

Typically, scientific codes:

- Have loops that comprise most of the computation in the program. Tens of lines of code might comprise up to 90% computation.
- Use arrays as their main data structures.
- Amenable to automatic parallelization.

As a result, many optimizing transformations concentrate on loop level optimizations. Most loop level optimizations are source–to–source, i.e., reshape loops at the source level.
Exercise: Loop Dependence

Assume we only have scalar and subscripted variables (no pointers and no control dependence) for data dependence analysis.

Which loop is parallelizable?

\[
\begin{align*}
&\text{do } I = 1, 100 \\
&\quad \text{do } J = 1, 100 \\
&\quad \quad A(I,J) = A(I,J) + 1 \\
&\quad \quad \text{enddo} \\
&\quad \text{enddo} \\
&\text{do } I = 1, 99 \\
&\quad \text{do } J = 1, 100 \\
&\quad \quad A(I,J) = A(I+1,J) + 1 \\
&\quad \quad \text{enddo} \\
&\quad \text{enddo} \\
\end{align*}
\]
Exercise: Loop Dependence

Assume we only have scalar and subscripted variables (no pointers and no control dependence) for data dependence analysis.

Which loop is parallelizable?

\[
\begin{align*}
&\text{do I = 1, 100} \\
&\quad \text{do J = 1, 100} \\
&\quad \quad A(I,J) = A(I,J) + 1 \\
&\quad \text{enddo} \\
&\text{enddo} \\
\end{align*}
\]

\[
\begin{align*}
&\text{do I = 1, 99} \\
&\quad \text{do J = 1, 100} \\
&\quad \quad A(I,J) = A(I+1,J) + 1 \\
&\quad \text{enddo} \\
&\text{enddo} \\
\end{align*}
\]

parallelization

\[
\begin{align*}
&\text{doall I = 1, 100} \\
&\quad \text{doall J = 1, 100} \\
&\quad \quad A(I,J) = A(I,J) + 1 \\
&\quad \text{enddo} \\
&\text{implicit barrier sync.} \\
&\text{enddo} \\
\end{align*}
\]

\[
\begin{align*}
&\text{doall I = 1, 99} \\
&\quad \text{doall J = 1, 100} \\
&\quad \quad A(I,J) = A(I+1,J) + 1 \\
&\quad \text{enddo} \\
&\text{implicit barrier sync.} \\
&\text{enddo} \\
\end{align*}
\]
Loop Iteration Space

- Lexicographical (sequential) order for the above iteration space:
  
  \[(1,1), (1,2), \ldots, (1,6)\]
  
  \[(2,2), (2,3), \ldots (2,6)\]
  
  \[
  \ldots
  
  \[(5,5), (5,6)\]

  
  do \(I = 1, 5\)
  
  do \(J = I, 6\)
  
  \[
  \ldots
  
  \]
  
  enddo

  enddo

  \(1 \leq I \leq 5\)

  \(I \leq J \leq 6\)

- Given \(I = (i_1, \ldots, i_n)\) and \(I' = (i'_1, \ldots, i'_n)\),

  \(I < I'\) iff \((i_1, i_2, \ldots i_k) = (i'_1, i'_2, \ldots i'_k)\) \& \(i_{k+1} < i'_{k+1}\)
Dependence Testing

Given

\[
\begin{align*}
\text{do } & i_1 = L_1, U_1 \\
& \quad \cdots \\
\text{do } & i_n = L_n, U_n \\
S_1 & \quad A(f_1(i_1, \ldots, i_n), \ldots, f_m(i_1, \ldots, i_n)) = \ldots \\
S_2 & \quad \ldots = A(g_1(i_1, \ldots, i_n), \ldots, g_m(i_1, \ldots, i_n))
\end{align*}
\]

A dependence between statement \( S_1 \) and \( S_2 \), denoted \( S_1 \delta S_2 \), indicates that \( S_1 \), the source, must be executed before \( S_2 \), the sink on some iteration of the nest.

Let \( \alpha \& \beta \) be a vector of \( n \) integers within the ranges of the lower and upper bounds of the \( n \) loops.

Does \( \exists \alpha \leq \beta \), s.t.

\[
f_k(\alpha) = g_k(\beta) \quad \forall k, 1 \leq k \leq m
\]
Dependence Testing

SIV - Single Induction Variable Test

1. Single loop nest with constant lower (LB) and upper (UB) bounds, and step 1

   for i = LB, UB, 1
   ...
   endfor

   The loop bounds define the iteration space for loop induction variable i.

2. Two array references with array subscript (index) expressions of the form (true dependence)

   for i = LB, UB, 1
   R1: X(a*i + c1) = ... \write
   R2: ... = X(a*i + c2) ... \read
   endfor

   where a, c1, and c2 are integer constants, R1 and R2 are references to the same array, i is the loop induction variable, and a \neq 0.

Question: Is there a true dependence between R1 and R2?
Dependence Testing

There is a dependence between R1 and R2 iff

\[ \exists i, i' : i \leq i' \ and \ (a \ast i + c_1) = (a \ast i' + c_2) \]

where \( i \) and \( i' \) are two iterations in the iteration space of the loop. This means that in both iterations, the same element of array \( X \) would be accessed.

So let’s just solve the equation:

\[ (a \ast i + c_1) = (a \ast i' + c_2) \equiv \]

\[ \frac{c_1 - c_2}{a} = i' - i = \Delta d \]

There is a dependence with distance \( \Delta d \) iff

1. \( \Delta d \) is an integer value and
2. \( \text{UB} - \text{LB} \geq \Delta d \geq 0 \)
Dependence Testing Examples

1. for i = LB, UB, 1
   R1: X(i) = ...  \ write
   R2: ... = X(i - 2) ...  \ read
   endif

   a=1, c₁=0, c₂=-2 \( \Rightarrow \) \( \Delta d = 2 \) (dependence)

2. for i = LB, UB, 1
   R1: X(2*i) = ...  \ write
   R2: ... = X(2*i - 1) ...  \ read
   endif

   a=2, c₁=0, c₂=-1 \( \Rightarrow \) \( \Delta d = \frac{1}{2} \) (no dependence)

Assume R1 executes before R2.

Classification of dependences:

- R1 is write, R2 is read \( \Rightarrow \) \textbf{true} dependence
- R1 is read, R2 is write \( \Rightarrow \) \textbf{anti} dependence
- R1 is write, R2 is write \( \Rightarrow \) \textbf{output} dependence
Next time:

- Matrix Matrix Multiplication and Locality Optimization