Selected Topics in Concurrency
COL 869

Relaxed Memory Concurrency
Concurrent Computing

Applications

Programming Languages

Compilers

OS

Architectures
Use of Concurrency

Pros.

• Concurrency improves performance

Cons.

• Concurrent programming is hard
• May result in tricky bugs in programs
• Requires careful analysis
Concurrent Programming

• Program with multiple threads

• Shared memory access
Concurrent Programming

• Program with multiple threads

• Shared memory access

Given a concurrent program and an outcome, is it a correct outcome?
Concurrent Programming

- Program with multiple threads
- Shared memory access

Given a concurrent program and an outcome, is it a correct outcome?

How does the program execute?
Question

Given a program and an outcome, is it a correct outcome?

Depends on

*Program execution*
Example

X = 0

X = X + 1;
X = X + 1;

What is the final value of X?
Example

X = 0

X = X + 1;

What is the final value of X?

Desired: X = 2
Execution

\[
X=0
\]

\[
a=X;  //  0
a = a+1; \quad \quad b=X;
X = a;  //  1
X = b;
\]
Execution

X=0

a=X;  // 0       b=X;  // 1
a = a+1;
X = a;  // 1      b = b+1;
X = b;
Execution

X=0

a=X; // 0     b=X; // 1
a = a+1;     b = b+1;
X = a; // 1    X = b; // 2

Output: X = 2
Another Execution

X=0

a=X;  // 0
a = a+1;
X = a;

b=X;
b = b+1;
X = b;
Another Execution

```
X=0

a = X;    // 0
a = a + 1;
X = a;

b = X;    // 0
b = b + 1;
X = b;
```
Another Execution

X = 0

a = X;  // 0
a = a + 1;
X = a;  // 1

b = X;  // 0
b = b + 1;
X = b;
Another Execution

X=0

a=X;  // 0
a = a+1;
X = a;  // 1

b=X;  // 0
b = b+1;
X = b;  // 1

Output:  X = 1
Example

X=0

X=X+1;       X=X+1;

What is the final value of X?

Desired : X=2

Reality : X = 1 / 2
What about This Execution?

<table>
<thead>
<tr>
<th>X=0</th>
<th>X=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=X; // 0</td>
<td>b=X;</td>
</tr>
<tr>
<td>a = a+1;</td>
<td>b = b+1;</td>
</tr>
<tr>
<td>X = a;</td>
<td>X = b;</td>
</tr>
<tr>
<td>a=X; // 0</td>
<td>b=X; // 0</td>
</tr>
<tr>
<td>a = a+1;</td>
<td>b = b+1;</td>
</tr>
<tr>
<td>X = a; // 1</td>
<td>X = b;</td>
</tr>
<tr>
<td>a=X; // 0</td>
<td>b=X; // 0</td>
</tr>
<tr>
<td>a = a+1;</td>
<td>b = b+1;</td>
</tr>
<tr>
<td>X = a; // 1</td>
<td>X = b; // 1</td>
</tr>
</tbody>
</table>
Understanding Concurrency

Key Question:

*How does concurrent programs execute?*

*Given an execution of a program, is it correct?*

Answer may vary across platforms.
How does concurrent programs execute?

“...the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” ~ Leslie Lamport
How does concurrent programs execute?

“...the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” ~ Leslie Lamport

Sequential consistency (SC)
How does concurrent programs execute?

“...the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program.” ~ Leslie Lamport

Sequential consistency (SC) ==> Interleaving Execution
# Earlier Example ➔ Interleaving Execution

<table>
<thead>
<tr>
<th>X=0</th>
<th>X=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=X;</td>
<td>a=X;</td>
</tr>
<tr>
<td>a = a+1;</td>
<td>a = a+1;</td>
</tr>
<tr>
<td>X = a;</td>
<td>X = a;</td>
</tr>
<tr>
<td>b=X;</td>
<td>b=X;</td>
</tr>
<tr>
<td>b = b+1;</td>
<td>b = b+1;</td>
</tr>
<tr>
<td>X = b;</td>
<td>X = b;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X=0</th>
<th>X=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=X;</td>
<td>a=X;</td>
</tr>
<tr>
<td>a = a+1;</td>
<td>a = a+1;</td>
</tr>
<tr>
<td>X = a;</td>
<td>X = a;</td>
</tr>
<tr>
<td>b=X;</td>
<td>b=X;</td>
</tr>
<tr>
<td>b = b+1;</td>
<td>b = b+1;</td>
</tr>
<tr>
<td>X = b;</td>
<td>X = b;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X=0</th>
<th>X=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=X;</td>
<td>a=X;</td>
</tr>
<tr>
<td>a = a+1;</td>
<td>a = a+1;</td>
</tr>
<tr>
<td>X = a;</td>
<td>X = a;</td>
</tr>
<tr>
<td>b=X;</td>
<td>b=X;</td>
</tr>
<tr>
<td>b = b+1;</td>
<td>b = b+1;</td>
</tr>
<tr>
<td>X = b;</td>
<td>X = b;</td>
</tr>
</tbody>
</table>
Sequential Consistency

Foundation of many concurrent algorithms
Example: Dekker’s Algorithm

\[
\text{wants\_to\_enter}[0] \leftarrow \text{false}, \text{wants\_to\_enter}[1] \leftarrow \text{false}, \text{turn} \leftarrow 0 \quad // \text{or 1}
\]

\[
\begin{align*}
\text{wants\_to\_enter}[0] & \leftarrow \text{true} \\
\text{while} \ (\text{wants\_to\_enter}[1]) \{ \\
\quad \text{if} \ (\text{turn} \neq 0) \{ \\
\quad\quad \text{wants\_to\_enter}[0] \leftarrow \text{false} \\
\quad\quad \text{while} \ (\text{turn} \neq 0) \ { // \text{busy wait} } \\
\quad\quad \text{wants\_to\_enter}[0] \leftarrow \text{true} \\
\quad\} \\
\quad \} \\
\quad \// \text{critical section} \\
\quad ...
\end{align*}
\]

\[
\begin{align*}
\text{turn} & \leftarrow 1 \\
\text{wants\_to\_enter}[0] & \leftarrow \text{false} \\
\text{wants\_to\_enter}[1] & \leftarrow \text{true} \\
\text{while} \ (\text{wants\_to\_enter}[0]) \{ \\
\quad \text{if} \ (\text{turn} \neq 1) \{ \\
\quad\quad \text{wants\_to\_enter}[1] \leftarrow \text{false} \\
\quad\quad \text{while} \ (\text{turn} \neq 1) \ { // \text{busy wait} } \\
\quad\quad \text{wants\_to\_enter}[1] \leftarrow \text{true} \\
\quad\} \\
\quad \} \\
\quad \// \text{critical section} \\
\quad ...
\end{align*}
\]

\[
\begin{align*}
\quad \text{turn} & \leftarrow 0 \\
\quad \text{wants\_to\_enter}[1] & \leftarrow \text{false} \\
\quad \// \text{remainder section}
\end{align*}
\]
Example: Dekker’s Algorithm

\[
\text{wants\_to\_enter}[0] \leftarrow \text{false}, \text{wants\_to\_enter}[1] \leftarrow \text{false}, \text{turn} \leftarrow 0 \quad // \text{or} \ 1
\]

\[
\text{wants\_to\_enter}[0] \leftarrow \text{true} \quad // \text{critical section}
\]

\[
\text{wants\_to\_enter}[1] \leftarrow \text{true} \quad // \text{critical section}
\]

\[
\text{while (wants\_to\_enter}[1]) \{...\} \quad // \text{critical section}
\]

\[
\text{while (wants\_to\_enter}[0]) \{...\} \quad // \text{critical section}
\]

Ensures mutual exclusion
Representative Example: Store Buffer (SB)

\[ X = Y = 0 \]

\[ X = 1; \quad Y = 1; \]
\[ a = Y; \quad b = X; \]
Program: Store Buffer (SB)

\[
\begin{align*}
X &= Y = 0 \\
X &= 1; & Y &= 1; \\
a &= Y; & b &= X;
\end{align*}
\]

Outcome:

\[a=0, \ b=1\]
Program: Store Buffer (SB)

\[
\begin{align*}
X &= Y = 0 \\
X &= 1; & Y &= 1; \\
a &= Y; & b &= X;
\end{align*}
\]

Outcome:

\[
\begin{align*}
a &= 0, & b &= 1 \\
a &= 1, & b &= 1
\end{align*}
\]
Program: Store Buffer (SB)

\[ X=Y=0 \]
\[ X = 1; \quad Y = 1; \]
\[ a = Y; \quad b = X; \]

Outcome:

\[ a=0, \ b=1 \]
\[ a=1, \ b=1 \]
\[ a=1, \ b=0 \]
Program: Store Buffer (SB)

X = Y = 0

X = 1;  Y = 1;
a = Y;  b = X;

Let’s execute the program...
Program: Store Buffer (SB)

\[
X = Y = 0
\]

\[
X = 1; \quad \quad \quad \quad \quad \quad Y = 1;
\]

\[
a = Y; \quad \quad \quad \quad \quad \quad b = X;
\]

Outcome:

\[
a = 0, \ b = 1
\]

\[
a = 1, \ b = 1
\]

\[
a = 1, \ b = 0
\]

\[
a = 0, \ b = 0
\]
Program: Store Buffer (SB)

X = Y = 0

X = 1;  Y = 1;
a = Y;  b = X;

a = 0, b = 0

Cannot be explained by thread interleaving.

Relaxed memory concurrency
Relaxed Memory Concurrency

Traditionally: Concurrency = thread interleaving

Reality: more behaviors than thread interleaving
Relaxed Memory Concurrency

Traditionally: Concurrency = thread interleaving

Reality: more behaviors than thread interleaving

Reasons:

Compiler: reorder instructions

Hardware:
- Out of order execution
- Data movement/communication is not instantaneous
Is it a problem?

Programmability: *Sequential consistency assumption may not hold!*
Is it a problem?

Programmability: *Sequential consistency assumption may not hold!*

Tools: *Verification, testing, analysis tools assume interleaving*
Is it a problem?

Programmability: *Sequential consistency assumption may not hold!*

Tools: *Verification, testing, analysis tools assume interleaving*

Compilers: *Not sure about correct transformations*
  - *Suboptimal compilation*
  - *compiler bugs*
How to address the Problem?

Programmability: *Sequential consistency assumption may not hold!*

Tools: *Verification, testing, analysis tools assume interleaving*

Compilers: *Not sure about correct transformations*
  - *Suboptimal compilation*
  - *compiler bugs*

Study *relaxed memory concurrency*
Content

Semantics
- Programming languages (C/C++)
- Architectures (x86, ARM, PowerPC)

Compiler Correctness
- Compiler optimizations
- Mappings to architectures

Program Correctness
- Data-race-freedom properties
Course Logistics

Visit:
http://www.cse.iitd.ac.in/~soham/COL869/page.html