Announcements

Midterm exam

- 12 February (Friday)
- In-class
- Closed book, closed notes
- Sample midterm exam has been posted to the course website; solutions to which will be provided in-class on 10 February

Homework #2

- I am NOT asking you to decompose the matrix multiplication calculation into several smaller ones.
- Each thread is meant to perform the entire matrix multiplication calculation.
- The size of the matrix and the number of times that you perform the calculation are the variables that will “tax” the CPU architecture and allow you to see the effect of threading versus non-threading
Announcements – the Big picture

By now you’ve noticed that merely issuing a thread to be executed doesn’t necessarily mean that the OS scheduler will schedule that thread for immediate execution.

- Barriers
- Rendezvous
- Critical regions
Announcements – the Big picture

By now you’ve noticed that merely issuing a thread to be executed doesn’t necessarily mean that the OS scheduler will schedule that thread for immediate execution.

- Barriers
- Rendezvous
- Critical regions

Our progression:

- Issue threads
By now you’ve noticed that merely issuing a thread to be executed doesn’t necessarily mean that the OS scheduler will schedule that thread for immediate execution.

- Barriers
- Rendezvous
- Critical regions

Our progression:

- Issue threads
- Issue threads whose output is collected
By now you’ve noticed that merely issuing a thread to be executed doesn’t necessarily mean that the OS scheduler will schedule that thread for immediate execution.

- **Barriers**
- **Rendezvous**
- **Critical regions**

Our progression:

- **Issue threads**
- **Issue threads whose output is collected**
- **Issue threads that communicate with each other and synchronization of “results”**
From last time

If you have $\text{numCPU} < n^2$ processors for a problem that can be broken up into $n^2$ parts, how do you dole out processes?
From last time

If you have \( \text{numCPU} < n^2 \) processors for a problem that can be broken up into \( n^2 \) parts, how do you dole out processes?

Thread_0

\[
\begin{array}{c|c|c|c}
\text{Thread}_0 & \text{CPU}_0 & \text{CPU}_1 & \text{CPU}_2 & \text{CPU}_9 \\
\end{array}
\]

Thread_1

Thread_2

Thread_20
From last time

If you have \( \text{numCPU} < n^2 \) processors for a problem that can be broken up into \( n^2 \) parts, how do you dole out processes?

Thread_0

\[
\begin{array}{c}
\text{CPU}_0 \\
\text{Thread}_0
\end{array}
\]

Thread_1

\[
\begin{array}{c}
\text{CPU}_1 \\
\text{Thread}_1
\end{array}
\]

Thread_2

\[
\begin{array}{c}
\text{CPU}_2 \\
\text{Thread}_2
\end{array}
\]

Thread_20

\[
\begin{array}{c}
\text{CPU}_9 \\
\text{Thread}_9
\end{array}
\]
From last time

If you have \( \text{numCPU} < n^2 \) processors for a problem that can be broken up into \( n^2 \) parts, how do you dole out processes?

- Thread_0
- Thread_1
- Thread_2
- Thread_20

\[
\begin{array}{c|c|c|c}
\text{CPU_0} & \text{CPU_1} & \text{CPU_2} & \text{CPU_9} \\
\hline
\text{Thread_0} & \text{Thread_1} & \text{Thread_2} & \text{Thread_9} \\
\text{Thread_10} & \text{Thread_11} & \text{Thread_12} & \text{Thread_19} \\
\text{Thread_20} & & & \\
\end{array}
\]
From last time

If you have \(\text{numCPU} < n^2\) processors for a problem that can be broken up into \(n^2\) parts, how do you dole out processes?

Task: Be able to answer the following flavors of questions:

Q: What factors affect how many processes are assigned to a CPU?
Q: What is the max or min number of processes each CPU receives, assuming an even distribution of threads among the CPUs?
Q: Does adding more threads necessarily decrease the run-time of the entire program?
From last time

If you have numCPU < n² processors for a problem that can be broken up into n² parts, how do you dole out processes?

Thread_0
Thread_1
Thread_2
Thread_20

Overhead can play a big role, and be a limiting factor in overall time needed for a threaded program to run to completion. Even if threads do not communicate, resources are needed to create, schedule, and monitor threads.
From last time

```java
double a[n,n], b[n,n], c[n,n];

for [i = 0 to n-1] {
    for [j = 0 to n-1] {
        # compute inner product of a[i,*] and b[*,j]
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

Q: which portions of it can be parallelized (for the time being think of this as “separate threads”) and which portion(s) of it cannot?
From last time

```c
double a[n,n], b[n,n], c[n,n];

for [i = 0 to n-1] {
    for [j = 0 to n-1] {
        # compute inner product of a[i,*] and b[*,j]
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

Q: which portions of it can be parallelized (for the time being think of this as “separate threads”) and which portion(s) of it cannot?

If you have $n$ processors
From last time

```java
double a[n,n], b[n,n], c[n,n];
for [i = 0 to n-1] {
    for [j = 0 to n-1] {
        # compute inner product of a[i,*] and b[*,j]
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

```java
c [i = 0 to n-1] {  # compute rows in parallel
    for [j = 0 to n-1] {
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

Q: which portions of it can be parallelized (for the time being think of this as “separate threads”) and which portion(s) of it cannot?

If you have $n$ processors
From last time

```c
double a[n,n], b[n,n], c[n,n];

for [i = 0 to n-1] {
    for [j = 0 to n-1] {
        # compute inner product of a[i,*] and b[* ,j]
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

Q: which portions of it can be parallelized (for the time being think of this as “separate threads”) and which portion(s) of it cannot?

```c
co [i = 0 to n-1] {  # compute rows in parallel
    for [j = 0 to n-1] {
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

If you have \( n \) processors

If you have \( n^2 \) processors
From last time

```cpp
double a[n,n], b[n,n], c[n,n];

for [i = 0 to n-1] {
    for [j = 0 to n-1] {
        # compute inner product of a[i,*] and b[*,j]
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

```cpp
co [i = 0 to n-1] {   # compute rows in parallel
    for [j = 0 to n-1] {
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

```cpp
co [i = 0 to n-1, j = 0 to n-1] { # all rows and
c[i,j] = 0.0;                       # all columns
    for [k = 0 to n-1]
        c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```

Q: which portions of it can be parallelized (for the time being think of this as “separate threads”) and which portion(s) of it cannot?

If you have \( n \) processors

If you have \( n^2 \) processors
From last time

double a[n,n], b[n,n], c[n,n];

for [i = 0 to n-1] {
  for [j = 0 to n-1] {
    # compute inner product of a[i,*] and b[*,j]
    c[i,j] = 0.0;
    for [k = 0 to n-1]
      c[i,j] = c[i,j] + a[i,k]*b[k,j];
  }
}

c0 [i = 0 to n-1] {  # compute rows in parallel
  for [j = 0 to n-1] {
    c[i,j] = 0.0;
    for [k = 0 to n-1]
      c[i,j] = c[i,j] + a[i,k]*b[k,j];
  }
}

c0 [i = 0 to n-1, j = 0 to n-1] {  # all rows and
  c[i,j] = 0.0;  # all columns
  for [k = 0 to n-1]
    c[i,j] = c[i,j] + a[i,k]*b[k,j];
}

Q: Can we parallelize this code further?
From last time

```
double a[n,n], b[n,n], c[n,n];

for [i = 0 to n-1] {
    for [j = 0 to n-1] {
        # compute inner product of a[i,*] and b[*,j]
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

```
co [i = 0 to n-1] { # compute rows in parallel
    for [j = 0 to n-1] {
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

```
co [i = 0 to n-1, j = 0 to n-1] { # all rows and
    c[i,j] = 0.0; # all columns
    for [k = 0 to n-1]
        c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```

Q: Can we parallelize this code further?

For each “previous” parallelization, we modified the code and “changed” each for loop to a series of concurrent execution calls.
From last time

```java
double a[n,n], b[n,n], c[n,n];
for [i = 0 to n-1] {
    for [j = 0 to n-1] {
        # compute inner product of a[i,*] and b[*,j]
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}
```

Q: Can we parallelize this code further?

For each “previous” parallelization, we modified the code and “changed” each for loop to a series of concurrent execution calls.
From last time

double a[n,n], b[n,n], c[n,n];

for [i = 0 to n-1] {
    for [j = 0 to n-1] {
        # compute inner product of a[i,*] and b[*,j]
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}

co [i = 0 to n-1] {  # compute rows in parallel
    for [j = 0 to n-1] {
        c[i,j] = 0.0;
        for [k = 0 to n-1]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
    }
}

co [i = 0 to n-1, j = 0 to n-1] {  # all rows and
    # all columns
    for [k = 0 to n-1]
        c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
Today

Shared Variable Programming
Q: Should we parallelize this for loop?

c[i,j] = c[i,j] + a[i,k]*b[k,j];
**Shared Variable Programming**

```c
const [i = 0 to n-1, j = 0 to n-1] { /* all rows and columns */
    c[i,j] = 0.0;

    for [k = 0 to n-1]
        c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```

**Q: Should we parallelize this for loop?**

**A:**

![Parallelization Diagram]

**Q: What are the entries of the product matrix?**

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3 \\
\end{array} \times \begin{array}{ccc}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6 \\
\end{array} = \begin{array}{ccc}
\text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } \\
\end{array}
\]
Shared Variable Programming

Q: Should we parallelize this for loop?

```
c0 = [i = 0 to n-1, j = 0 to n-1] { # all rows and
c[i][j] = 0.0; # all columns
for [k = 0 to n-1]
c[i][j] = c[i][j] + a[i][k]*b[k][j];
}
```

Assume you have 12 processors … how do you dole out “chunks” of the matrix product calculation?
Shared Variable Programming

\[
\text{co } [i = 0 \text{ to } n-1, j = 0 \text{ to } n-1] \{ \# \text{ all rows and } \\
\qquad c[i,j] = 0.0; \quad \# \text{ all columns} \\
\quad \text{for } [k = 0 \text{ to } n-1] \\
\qquad c[i,j] = c[i,j] + a[i,k]*b[k,j]; \}
\]

Q: Should we parallelize this for loop?

A

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3 \\
\end{array}
\] \text{x} \quad \begin{array}{ccc}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}
= \begin{array}{ccc}
\text{24} & 30 & 36 \\
\text{24} & 30 & 36 \\
\text{24} & 30 & 36 \\
\end{array}
\]

P0 = 1X4 + 2x4 + 3x4 = 4 + 8 + 12 = 24
Shared Variable Programming

```c
for [k = 0 to n-1]
    c[i, j] = c[i, j] + a[i, k]*b[k, j];
}
```

**Q:** Should we parallelize this for loop?

**A:**

```
A
1 2 3 1 2 3 1 2 3
B
4 5 6 4 5 6 4 5 6
C
24 30 36 24 30 36 24 30 36
```

\[
P0 = 1 \times 4 + 2 \times 4 + 3 \times 4 = 4 + 8 + 12 = 24 \\
P1 = 1 \times 5 + 2 \times 5 + 3 \times 5 = 5 + 10 + 15 = 30
\]
Q: Should we parallelize this for loop?

\[\begin{array}{ccc}
1 & 2 & 3 \\
\hline
1 & 2 & 3 \\
\hline
1 & 2 & 3
\end{array}\] \times \begin{array}{ccc}
4 & 5 & 6 \\
\hline
4 & 5 & 6 \\
\hline
4 & 5 & 6
\end{array} = \begin{array}{ccc}
24 & 30 & 36 \\
\hline
24 & 30 & 36 \\
\hline
24 & 30 & 36
\end{array}

P0 = 1 \times 4 + 2 \times 4 + 3 \times 4 = 4 + 8 + 12 = 24

P1 = 1 \times 5 + 2 \times 5 + 3 \times 5 = 5 + 10 + 15 = 30

P2 = 1 \times 6 + 2 \times 6 + 3 \times 6 = 6 + 12 + 18 = 36
### Shared Variable Programming

```c
co [i = 0 to n-1, j = 0 to n-1] { // all rows and
  c[i, j] = 0.0; // all columns
  for [k = 0 to n-1]
    c[i, j] = c[i, j] + a[i, k]*b[k, j];
}
```

**Q:** Should we parallelize this for loop?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Q:** What does it mean to parallelize over all of the columns?
Shared Variable Programming

```c
co [i = 0 to n-1, j = 0 to n-1] { // all rows and
    c[i, j] = 0.0; // all columns
    for [k = 0 to n-1]
        c[i, j] = c[i, j] + a[i, k] * b[k, j];
}
```

Q: Should we parallelize this for loop?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td>4 5 6</td>
<td>24 30 36</td>
</tr>
<tr>
<td>1 2 3</td>
<td>4 5 6</td>
<td>24 30 36</td>
</tr>
<tr>
<td>1 2 3</td>
<td>4 5 6</td>
<td>24 30 36</td>
</tr>
</tbody>
</table>

P0 = 1\times4 + 2\times4 + 3\times4 = 4 + 8 + 12 = 24

Q: What does it mean to parallelize over all of the columns?
Q: Should we parallelize this for loop?

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3 \\
\end{array}
\times
\begin{array}{ccc}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}
=\begin{array}{ccc}
24 & 30 & 36 \\
24 & 30 & 36 \\
24 & 30 & 36 \\
\end{array}
\]

Q: What does it mean to parallelize over all of the columns?

\[
P_0 = 1 \times 4 + 2 \times 4 + 3 \times 4 = 4 + 8 + 12 = 24
\]

\[
P_4, P_5, P_6
\]
Q: Should we parallelize this for loop?

\[
\begin{align*}
\text{P}_0 &= 1x4 + 2x4 + 3x4 = 4 + 8 + 12 = 24
\end{align*}
\]
Shared Variable Programming

Q: Should we parallelize this for loop?

\[
\begin{align*}
&\text{co [i = 0 to n-1, j = 0 to n-1] } \{ \text{ # all rows and columns} \\
&\quad \text{c[i,j] = 0.0; } \\
&\quad \text{for [k = 0 to n-1] } \\
&\quad \quad \text{c[i,j] = c[i,j] + a[i,k]*b[k,j]; } \\
&\}\end{align*}
\]

A

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3 \\
\end{array}
\]

B

\[
\begin{array}{ccc}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}
\]

C

\[
\begin{array}{ccc}
24 & 30 & 36 \\
24 & 30 & 36 \\
24 & 30 & 36 \\
\end{array}
\]

Q: Why is \(c[i,j]\) not “inside” the \(k\) loop?
**Shared Variable Programming**

```c
co [i = 0 to n-1, j = 0 to n-1] { # all rows and
    c[i,j] = 0.0; # all columns
    for [k = 0 to n-1]
        c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```

**Q: Should we parallelize this for loop?**

```
   A         B                         C
   1 2 3  x   4 5 6                   24 30 36
   1 2 3     4 5 6                   24 30 36
   1 2 3     4 5 6                   24 30 36
```

**Q: Why is c[i,j] not “inside” the k loop?**

```c
for (k=0 to n-1){
    c[i,j] = 0.0;
    c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```
Q: Should we parallelize this for loop?

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3 \\
\end{array}\times
\begin{array}{ccc}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}=
\begin{array}{ccc}
24 & 30 & 36 \\
24 & 30 & 36 \\
24 & 30 & 36 \\
\end{array}
\]

Q: Why is \(c[i,j]\) not “inside” the \(k\) loop?

\[
\text{for } (k=0 \text{ to } n-1)\{ \\
\quad c[i,j] = 0.0; \\
\quad c[i,j] = c[i,j] + a[i,k]*b[k,j]; \\
\}
\]
Shared Variable Programming

```c
void co [i = 0 to n-1, j = 0 to n-1] {  // all rows and
    c[i, j] = 0.0;  // all columns
    for [k = 0 to n-1]
        c[i, j] = c[i, j] + a[i, k]*b[k, j];
}
```

Q: Should we parallelize this for loop?

A: 

```
A
1 2 3
1 2 3
1 2 3

B
4 5 6
4 5 6
4 5 6

C
24 30 36
24 30 36
24 30 36
```

Q: If we were to parallelize the k loop also, what errors could result?
Shared Variable Programming

Q: Should we parallelize this for loop?

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3 \\
\end{array} \quad \times \quad \begin{array}{ccc}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6 \\
\end{array} = \begin{array}{ccc}
24 & 30 & 36 \\
24 & 30 & 36 \\
24 & 30 & 36 \\
\end{array}
\]

Q: If we were to parallelize the k loop also, what errors could result?

P0

i=0
j=0
\begin{array}{ccc}
c[i,j] = 0 \\
\end{array}
for (k=0 to 2)
\begin{array}{ccc}
c[i,j] = ... \\
\end{array}

P1

i=0
j=1
\begin{array}{ccc}
c[i,j] = 0 \\
\end{array}
for (k=0 to 2)
\begin{array}{ccc}
c[i,j] = ... \\
\end{array}

P2

i=0
j=2
\begin{array}{ccc}
c[i,j] = 0 \\
\end{array}
for (k=0 to 2)
\begin{array}{ccc}
c[i,j] = ... \\
\end{array}

Q: What values of i and j does processor 9 use?
Shared Variable Programming

Q: Should we parallelize this for loop?

\[
\begin{align*}
&\text{co [i = 0 to n-1, j = 0 to n-1]} \{ \text{# all rows and} \nspace \text{c[i,j] = 0.0; } \text{# all columns} \\
&\text{for [k = 0 to n-1]} \\
&\text{\hspace{1cm} c[i,j] = c[i,j] + a[i,k]*b[k,j];} \\
&\}
\end{align*}
\]

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3 \\
\end{array}
\times
\begin{array}{ccc}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}
= 
\begin{array}{ccc}
24 & 30 & 36 \\
24 & 30 & 36 \\
24 & 30 & 36 \\
\end{array}
\]

Q: What are the read and write sets of each process?

Q: Where do the \( k \) values reside for each processor?
Shared Variable Programming

Q: Should we parallelize this for loop?

A

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
1 & 2 & 3 \\
\end{array}
\]

B

\[
\begin{array}{ccc}
4 & 5 & 6 \\
4 & 5 & 6 \\
4 & 5 & 6 \\
\end{array}
\]

C

\[
\begin{array}{ccc}
24 & 30 & 36 \\
24 & 30 & 36 \\
24 & 30 & 36 \\
\end{array}
\]

Q: If we were to “parallelize” the code one more step, and assuming an infinite number of available resources, how many processors are needed to fully parallelize?
Shared Variable Programming

Q: Should we parallelize this for loop?

\[
\begin{align*}
\text{co} & \{ i = 0 \text{ to } n-1, j = 0 \text{ to } n-1 \} \{ \# \text{ all rows and} \\
& \quad \text{c}[i,j] = 0.0; \quad \# \text{ all columns} \\
& \quad \text{for} \ [k = 0 \text{ to } n-1] \\
& \quad \quad \text{c}[i,j] = \text{c}[i,j] + a[i,k]*b[k,j]; \\
\}
\end{align*}
\]

Q: If we were to “parallelize” the code one more step, and assuming an infinite number of available resources, how many processors are needed to fully parallelize?
Q: Should we parallelize this for loop?

Take a close look at P0 and P1. Q: What are their possible instruction histories?

Q: Are the write sets of P0 and P1 disjoint from the read and write sets of the other?
We want to “formally” be able to dissect and understand what can and cannot be concurrently executed. How do we do that?
Concurrent programs

- **State:**
- **Atomic Action:**
- **History, interleaving, trace:**
- **Critical Section:**

```c
co [i = 0 to n-1, j = 0 to n-1] {  # all rows and
c[i,j] = 0.0;                  # all columns
    for [k = 0 to n-1]
        c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```
Concurrent programs

- **State**: the values of all the variables at a point in time
- **Atomic Action**: 
- **History, interleaving, trace**: 
- **Critical Section**: 

```plaintext
co [i = 0 to n-1, j = 0 to n-1] { # all rows and columns
    c[i,j] = 0.0;
    for [k = 0 to n-1]
        c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```
Concurrent programs

- **State**: the values of all the variables at a point in time
- **Atomic Action**: Action which indivisibly examines or changes a state
- **History, interleaving, trace**:
- **Critical Section**:

```cpp
co [i = 0 to n-1, j = 0 to n-1] {  # all rows and
  c[i,j] = 0.0;  # all columns
  for [k = 0 to n-1]
    c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```
Concurrent programs

- **State**: the values of all the variables at a point in time
- **Atomic Action**: Action which indivisibly examines or changes a state
- **History, interleaving, trace**: Particular sequence of atomic actions
- **Critical Section**:

```c
for [i = 0 to n-1, j = 0 to n-1] { # all rows and
c[i,j] = 0.0; # all columns
for [k = 0 to n-1]
c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```

\[ S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \ldots S_n \]
Concurrent programs

- **State**: the values of all the variables at a point in time
- **Atomic Action**: Action which indivisibly examines or changes a state
- **History, interleaving, trace**: Particular sequence of atomic actions
- **Critical Section**: Section that cannot be interleaved with other actions

```plaintext
co [i = 0 to n-1, j = 0 to n-1] { # all rows and
c[i,j] = 0.0; # all columns
for [k = 0 to n-1]
    c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```
Concurrent programs

- **State**: the values of all the variables at a point in time
- **Atomic Action**: Action which indivisibly examines or changes a state
- **History, interleaving, trace**: Particular sequence of atomic actions
- **Critical Section**: Section that cannot be interleaved with other actions

```c
for [k = 0 to n-1]
    c[i,j] = c[i,j] + a[i,k]*b[k,j];
```

- **Property**:
  - **Safety Property**:
  - **Liveness Property**:

- **Partial Correctness**:
- **Termination**:
- **Total Correctness**:
Concurrent programs

- **State**: the values of all the variables at a point in time
- **Atomic Action**: Action which indivisibly examines or changes a state
- **History, interleaving, trace**: Particular sequence of atomic actions
- **Critical Section**: Section that cannot be interleaved with other actions

```c
co [i = 0 to n-1, j = 0 to n-1] { # all rows and all columns
    c[i,j] = 0.0;
    for [k = 0 to n-1]
        c[i,j] = c[i,j] + a[i,k]*b[k,j];
}
```

- **Property**: Something that is true of EVERY possible history
  - **Safety Property**: Program never enters a bad state (mutual exclusion)
  - **Liveness Property**: Program eventually enters good state (enter critical section without deadlock)
- **Partial Correctness**: Final state is correct, assuming the program terminates
- **Termination**: Every loop and procedure call terminates
- **Total Correctness**: Partially correct and termination
Concurrent programs

- **State**: the values of all the variables at a point in time
- **Atomic Action**: An action indivisibly examines or changes a state
- **History, interleaving, trace**: A particular sequence of atomic actions
- **Critical Section**: A section that cannot be interleaved with other actions

This sort of terminology allows us to ask and logically reason about important questions such as:

- **Q**: Can my program be run concurrently?
- **Q**: What are the chances of deadlock?
- **Q**: Are the results of a concurrent program reproducible?
- **Q**: Will subsequent executions of a concurrent program give the same results?

- **Property**: Something that is true of EVERY possible history
  - **Safety Property**: A program never enters a bad state (mutual exclusion)
  - **Liveness Property**: A program eventually enters a good state (enter critical section without deadlock)
- **Partial Correctness**: Final state is correct, assuming the program terminates
- **Termination**: Every loop and procedure call terminates
- **Total Correctness**: Partially correct and termination

```java
// Example code
for (k = 0 to n-1)
    c[i,j] = c[i,j] + a[i,k]*b[k,j];
```
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

“run the program” multiple times and “test”
Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$ atomic actions

“run the program” multiple times and “test”

$m \quad n \quad \text{possible histories}$

$3 \quad 1$

For $m=3$ and $n=1$, how many histories are possible, considering that each of the three threads is independent of the others?

In-class exercise
Q: How do we demonstrate properties of a concurrent program?

Test and debug

"run the program" multiple times and "test"

$m$ threads executing $n$ atomic actions

$m$ $n$ possible histories

3 1 6

A
B
C
Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$ atomic actions

“run the program” multiple times and “test”

$m \quad n \quad possible \ histories$

3 \quad 1 \quad 6

A

ABC
ACB
BAC
BCA
CAB
CBA

B

We “could” test this manually

C

CBA

We “could” test this manually
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

\[ m \text{ threads executing } n \]
atomic actions

For \( m=2 \) and \( n=2 \), how many histories are possible, considering each of the atomic actions in threads A are independent from the atomic actions in B?

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>possible histories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>?</td>
</tr>
</tbody>
</table>

In-class exercise
Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$
atomic actions

For $m=2$ and $n=2$, how many histories are possible, considering each of the atomic actions in threads A are independent from the atomic actions in B?

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>possible histories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>?</td>
</tr>
</tbody>
</table>

First jot down ALL of the possible combinations Because actions 1 and 2 in each thread are NOT independent, order must be preserved (A1 < A2, and B1 < B2)
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$ atomic actions

For $m=2$ and $n=2$, how many histories are possible, considering each of the atomic actions in threads A are independent from the atomic actions in B?

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>possible histories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>?</td>
</tr>
</tbody>
</table>


Then, identify those that meet the criteria.
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$ atomic actions

For $m=2$ and $n=2$, how many histories are possible, considering each of the atomic actions in threads A are independent from the atomic actions in B?

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>possible histories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>?</td>
</tr>
</tbody>
</table>

A1 Then, identify those that meet the criteria

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>A2</td>
<td>B2</td>
<td>B1</td>
</tr>
<tr>
<td>A1</td>
<td>B1</td>
<td>B2</td>
<td>A2</td>
</tr>
<tr>
<td>A1</td>
<td>B1</td>
<td>A2</td>
<td>B2</td>
</tr>
</tbody>
</table>

A2

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>A2</td>
<td>B2</td>
<td>B1</td>
</tr>
<tr>
<td>A1</td>
<td>B1</td>
<td>B2</td>
<td>A1</td>
</tr>
<tr>
<td>A1</td>
<td>B1</td>
<td>A2</td>
<td>B1</td>
</tr>
</tbody>
</table>

B1

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B2</td>
<td>A2</td>
<td>B1</td>
</tr>
<tr>
<td>A1</td>
<td>B1</td>
<td>A2</td>
<td>B1</td>
</tr>
</tbody>
</table>

B2

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B2</td>
<td>B1</td>
<td>A2</td>
</tr>
<tr>
<td>A1</td>
<td>B2</td>
<td>A1</td>
<td>B2</td>
</tr>
</tbody>
</table>

B

<table>
<thead>
<tr>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>A1</td>
<td>B1</td>
<td>B2</td>
</tr>
<tr>
<td>A2</td>
<td>A1</td>
<td>B2</td>
<td>B1</td>
</tr>
<tr>
<td>A2</td>
<td>A1</td>
<td>B2</td>
<td>A1</td>
</tr>
<tr>
<td>A2</td>
<td>B1</td>
<td>A2</td>
<td>B1</td>
</tr>
<tr>
<td>A2</td>
<td>B1</td>
<td>A1</td>
<td>B2</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>A1</td>
<td>B2</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>B1</td>
<td>A2</td>
</tr>
</tbody>
</table>
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$ atomic actions

For $m=2$ and $n=2$, how many histories are possible, considering each of the atomic actions in threads $A$ are independent from the atomic actions in $B$?

$m$  $n$  possible histories
3  1  6
2  2  6

B2  A1  A2  B1  B1  A2  A2  B1  A2  B1  A2  B1  A2  B1  A2  B1

Then, identify those that meet the criteria:


B1 B2 A1 A2  B2 A1 B1 A2
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$
atomic actions

$m$     $n$     possible histories
3       1       6
2       2       6
3       2

For $m=3$ and $n=2$, how many histories are possible?

In-class exercise
Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$
atomic actions

$\begin{array}{ccc}
m & n & \text{possible histories} \\
3 & 1 & 6 \\
2 & 2 & 6 \\
3 & 2 & \\
\end{array}$

For $m=3$ and $n=2$, how many histories are possible?

Enumerate all possible combinations
Select those that meet the dependence criteria
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

\[ m \text{ threads executing } n \text{ atomic actions} \]

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>possible histories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

For \( m=3 \) and \( n=2 \), how many histories are possible?

Enumerate all possible combinations
Select those that meet the dependence criteria

Any guesses how many histories there are with \( m=3 \) and \( n=2 \)?
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$ atomic actions

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>possible histories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>90</td>
</tr>
</tbody>
</table>

For $m=3$ and $n=2$, how many histories are possible?

Enumerate all possible combinations
Select those that meet the dependence criteria
Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$ atomic actions

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>possible histories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

For $m=4$ and $n=2$, how many histories are possible?

Enumerate all possible combinations
Select those that meet the dependence criteria
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

\( m \) threads executing \( n \) atomic actions

\[
\begin{array}{ccc}
\text{\( m \)} & \text{\( n \)} & \text{possible histories} \\
3 & 1 & 6 \\
2 & 2 & 6 \\
3 & 2 & 90 \\
4 & 2 & 2520 \\
\end{array}
\]

Q: What is the formula for calculating the count of number of histories?

For \( m=4 \) and \( n=2 \), how many histories are possible?

Enumerate all possible combinations
Select those that meet the dependence criteria
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

$m$ threads executing $n$
atomic actions

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>possible histories</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2520</td>
</tr>
</tbody>
</table>

Q: What is the formula for calculating the count of number of histories?

For $m=4$ and $n=2$, how many histories are possible?

Enumerate all possible combinations
Select those that meet the dependence criteria

Formula: \( \frac{(mn)!}{(n!)^m} \)
Concurrent programs

Q: How do we demonstrate properties of a concurrent program?

Test and debug

\[ m \text{ threads executing } n \text{ atomic actions} \]

“run the program” multiple times and “test”

For \( m=4 \) and \( n=7 \), how many histories are possible?
Q: How do we demonstrate properties of a concurrent program?

- Test and debug
- “run the program” multiple times and “test”

$m$ threads executing $n$ atomic actions

For $m=4$ and $n=7$, how many histories are possible?

472518347558400

Thus we cannot brute-force test all of the possible histories by repeatedly running the program and waiting for the scheduler to eventually schedule EACH of the possible histories and use the states and outputs to test if the concurrency program runs as intended.
Concurrent programs

Assertional reasoning
Concurrent programs

Assertional reasoning

- Axiomatic semantics (logic)
- Use assertions to characterize sets of states
- Atomic actions are predicate transformers
  - Predicate logic, such as first-order logic, second-order logic, etc. that rely on quantifiers such as “there exists” and “for all”

Goal: Prove that bad state cannot happen

Difficult to do correctly, so usually a combination of testing and operational reasoning (predicate logic) is used
Up Next ...