CSCI 322
Principles of Concurrent Programming

Filip Jagodzinski
Announcements

HW 1

• Due tomorrow, 6 July, 11:59pm
• Late submissions are NOT accepted

• For the programming portion, why the following won’t work:

```cpp
for (int x=0; x<n; x++){
    // create and start thread i_x
    // join
}
```

Midterm Exam

• Will be made available on Thursday
• Will be due Monday, 11 July, at 11:59pm
• During the last half of tomorrow’s (Wednesday) lecture I’ll go over the sample midterm exam posted on the course website
Q: How do we estimate the performance gain of using a computer with cache versus one without cache. Assume that the cache is a factor of $\tau$ “faster” than memory?
Q: How do we estimate the performance gain of using a computer with cache versus one without cache. Assume that the cache is a factor of $\tau$ “faster” than memory?

$$\beta =$$

$$T_m =$$

$$T_c =$$

$$T_{av} =$$

$$G(\tau, \beta) =$$
Q: How do we estimate the performance gain of using a computer with cache versus one without cache. Assume that the cache is a factor of $\tau$ “faster” than memory?

\[ \beta = \text{cache reuse ratio; the fraction of loads or reads that can be reused from cache} \]
\[ T_m = \text{access time to main memory} \]
\[ T_c = T_m / \tau = \text{access time for cache} \]
\[ T_{av} = \beta T_c + (1-\beta) T_m \]
\[ G(\tau, \beta) = T_m / T_{av} = \text{access performance gain} \]
\[ = \tau T_c / (\beta T_c + (1-\beta) \tau T_c) \]
\[ = \tau / (\beta + \tau (1-\beta)) \]
From last time …

Q: How do we estimate the performance gain of using a computer with cache versus one without cache. Assume that the cache is a factor of \( \tau \) “faster” than memory?

\[ \beta = \text{cache reuse ratio; the fraction of loads or reads that can be reused from cache} \]

\[ T_m = \text{access time to main memory} \]

\[ T_c = T_m / \tau = \text{access time for cache} \]

\[ T_{av} = \beta T_c + (1-\beta)T_m \]

\[ G(\tau, \beta) = T_m / T_{av} = \text{access performance gain} \]

\[ = \tau T_c / (\beta T_c + (1-\beta)\tau T_c) \]

\[ = \tau / (\beta + \tau(1-\beta)) \]

Using the above formula, a plot of different combinations of \( \beta \) and \( \tau \) produces the following performance gain curves.

Be able to interpret this plot and answer, Q: For a specific \( \beta, \tau \) combination, what change(s) must be made to \( \beta \) or \( \tau \) so that performance gain doubles?
From last time ...

Assume a square matrix \( m = n \)

Q: Do these two methods output the same result?
Q: Do these two methods consume the same amount of time?

```c
int sumArray1(int a[m][n]){
    int i, j, sum=0;
    for (i=0; i<m, i++){
        for (j=0; j<n; j++){
            sum += a[i][j];
        }
    }
    return sum;
}
```

```c
int sumArray2(int a[m][n]){
    int i, j, sum=0;
    for (j=0; j<m, j++){
        for (i=0; i<n; i++){
            sum += a[i][j];
        }
    }
    return sum;
}
```
From last time ...

Assume a square matrix \(( m = n )\)

```c
int sumArray1(int a[m][n]){
    int i, j, sum=0;
    for (i=0; i<m, i++){
        for (j=0; j<n; j++){
            sum += a[i][j];
        }
    }
    return sum;
}
```

```c
int sumArray2(int a[m][n]){
    int i, j, sum=0;
    for (j=0; j<m, j++){
        for (i=0; i<n; i++){
            sum += a[i][j];
        }
    }
    return sum;
}
```
From last time ...

Assume a square matrix ( \( m = n \) )

```
int sumArray1(int a[m][n]){
    int i, j, sum=0;
    for (i=0; i<m, i++){
        for (j=0; j<n; j++){
            sum += a[i][j];
        }
    } 
    return sum;
}
```

```
int sumArray2(int a[m][n]){
    int i, j, sum=0;
    for (j=0; j<m, j++){
        for (i=0; i<n; i++){
            sum += a[i][j];
        }
    } 
    return sum;
}
```

```
\[
\begin{array}{ccc}
  & 0 & 1 & 2 \\
 0 & 1 & 3 & 6 \\
1 & 8 & 9 & 0 \\
2 & 6 & 7 & 2 \\
\end{array}
\]

sumArray1 : sum = ?
sumArray2 : sum = ?
From last time …

Assume a square matrix (m = n)

```
int sumArray1(int a[m][n]){
    int i, j, sum=0;
    for (i=0; i<m, i++){
        for (j=0; j<n; j++){
            sum += a[i][j];
        }
    }
    return sum;
}
```

```
int sumArray2(int a[m][n]){
    int i, j, sum=0;
    for (i=0; i<n; i++){
        for (j=0; j<m, j++){
            sum += a[i][j];
        }
    }
    return sum;
}
```

```
\[
\begin{array}{ccc}
  & 0 & 1 & 2 \\
0 & 1 & 3 & 6 \\
1 & 8 & 9 & 0 \\
2 & 6 & 7 & 2 \\
\end{array}
\]

sumArray1 : sum = 1 + 3 + 6 + 8 + 9 + 0 + 6 + 7 + 2 = 42
sumArray2 : sum = 1 + 8 + 6 + 3 + 9 + 7 + 6 + 0 + 2 = 42

Same “sum”? : yes
Same “run time”? : for small m,n, probably yes, for large m and n, most likely not … Why?
From last time ...

Q: If cache is loaded row order, then how can you ensure that the code runs as efficiently as possible?
From last time ...

Q: If cache is loaded row order, then how can you ensure that the code runs as efficiently as possible?
From last time ...

If summation “order” is 1, 8, 6, -3, 3, 9, 7, -7, 4, 0, 2, and 10

Q: How many entries can be summed before an eviction needs to occur?

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>8</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>6</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>-3</td>
<td>-7</td>
<td>10</td>
</tr>
</tbody>
</table>
From last time ...

If summation “order” is 1, 3, 4, 8, 9, 0, 6, 7, 2, -3, -7, 10

Q: How many entries can be summed before an eviction needs to occur?
If we can parallelize code (threads) should we?  
Why or why not?
From last time ...

\[
\begin{array}{cc}
A & B \\
\begin{array}{cc}
2 & 3 \\
4 & 5 \\
\end{array} & \begin{array}{cc}
6 & 7 \\
8 & 9 \\
\end{array}
\end{array}
\]

\[
\begin{array}{cc}
AB \\
\begin{array}{cc}
36 & 41 \\
64 & 73 \\
\end{array}
\end{array}
\]
From last time ...

\[
\begin{bmatrix}
2 & 3 \\
4 & 5 \\
\end{bmatrix}
\times
\begin{bmatrix}
6 & 7 \\
8 & 9 \\
\end{bmatrix}
= 
\begin{bmatrix}
36 & 41 \\
64 & 73 \\
\end{bmatrix}
\]

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];
    }
}

\[(AB)_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj}\]
From last time ...

\[
\begin{array}{cc}
A & B \\
\begin{array}{cc}
2 & 3 \\
4 & 5 \\
\end{array} & \begin{array}{cc}
6 & 7 \\
8 & 9 \\
\end{array}
\end{array}
\Rightarrow AB
\begin{array}{cc}
36 & 41 \\
64 & 73 \\
\end{array}
\]

```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];
    }
}
```

\[
(AB)_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj}
\]
From last time ...

\[
\begin{align*}
\begin{array}{ccc}
A & B & AB \\
\hline
2 & 3 & \times & 6 & 7 & = & 36 & 41 \\
4 & 5 & & 8 & 9 & & 64 & 73 \\
\end{array}
\end{align*}
\]

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];
    }
  }
}

\[(AB)_{ij} = \sum_{k=1}^{m} A_{ik}B_{kj}\]
Q: How can the below program be parallelized?

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];
        }
    }
From last time …

Q: How can the below program be parallelized?

```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];
    }
}
```

```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];
}
```

Q: How should different for loop iterations be doled out to the available processors/cores if x=n, x=n^2, x=n^3
From last time ...

**Read set**: set of variables read by a process

**Write set**: set of variables written to by a process

Two processes are *independent* if ________________________
From last time ...

**Read set**: set of variables read by a process

**Write set**: set of variables written to by a process

Two processes are **independent** if the write set of each is **disjoint** from both the read and write sets of the other.
Independent sets
When can code be parallelized

Scenario 2: Process $P_a$ reads from $d$, and writes to $d$ and $b$, and process $P_b$ reads from $b$

Q: Can we safely run both processes concurrently?
When can code be parallelized

Scenario 2: Process $P_a$ reads from $d$, and writes to $d$ and $b$, and process $P_b$ reads from $b$

Q: Are $P_a$ and $P_b$ independent?

Q: Can we safely run both processes concurrently?
When can code be parallelized

Scenario 2: Process $P_a$ reads from $d$, and writes to $d$ and $b$, and process $P_b$ reads from $b$

Q: Are $P_a$ and $P_b$ independent?
Scenario 2: Process $P_a$ reads from d, and writes to d and b, and process $P_b$ reads from b

Q: Are $P_a$ and $P_b$ independent?

Q: Is $P_a$’s write set disjoint from the read and write sets of $P_b$?
Q: Is $P_b$’s write set disjoint from the read and write sets of $P_a$?
When can code be parallelized

Scenario 2: Process $P_a$ reads from $d$, and writes to $d$ and $b$, and process $P_b$ reads from $b$

Q: Are $P_a$ and $P_b$ independent?  No

Q: Is $P_a$’s write set disjoint from the read and write sets of $P_b$?  No

Q: Is $P_b$’s write set disjoint from the read and write sets of $P_a$?  Yes
When can code be parallelized

Scenario 2: Process $P_a$ reads from $d$, and writes to $d$ and $b$, and process $P_b$ reads from $b$

**Q: Are $P_a$ and $P_b$ independent?**

No

**Task:** Give an example (code, threads) that “shows” why processes that are NOT independent may produce “wrong” results when run concurrently.
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

```
b = 3
d = 4
c0
i1 : b = d * 2
i2 : d = b
i3 : print(b)
oc
```
When can code be parallelized

Two processes are independent if the write set of each is disjoint from both the read and write sets of the other.

Q: What are the possible histories for i1, i2, and i3?

(assume an infinite number of CPUs)

```
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

- b = 3
- d = 4
- co
- i1 : b = d * 2
- i2 : d = b
- i3 : print(b)
- oc
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other

Q: What are the possible histories for \(i_1, i_2, \) and \(i_3?\)

(assume an infinite number of CPUs)

\[
\begin{array}{cccccc}
 a & b & c & d & e & f \\
 \text{3} & \text{4} \\
\end{array}
\]

\[
\begin{align*}
 b &= 3 \\
 d &= 4 \\
 \text{co} \\
 i_1 &: b = d \times 2 \\
 i_2 &: d = b \\
 i_3 &: \text{print}(b) \\
 \text{oc} \\
\end{align*}
\]

\[
\begin{align*}
 i_1 < i_2 < i_3 \\
 i_1 < i_3 < i_2 \\
 i_2 < i_1 < i_3 \\
 i_3 < i_2 < i_1 \\
\end{align*}
\]
When can code be parallelized

Two processes are independent if the write set of each is disjoint from both the read and write sets of the other.

Q: What are the possible histories for i1, i2, and i3?

(assume only 2 CPUs)

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
3 & 4 \\
\end{array}
\]

\[
\begin{align*}
b &= 3 \\
d &= 4 \\
\text{co} \\
i1 & : b = d * 2 \\
i2 & : d = b \\
i3 & : \text{print}(b) \\
o &
\end{align*}
\]
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: What are the possible histories for i1, i2, and i3?

(assume only 2 CPUs)

Q: Which instructions are executed by \( P_a \), and which by \( P_b \)?

\[
\begin{array}{cccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} \\
3 & 4 & & & & \\
\end{array}
\]

b = 3

d = 4

c

i1 : b = d * 2

i2 : d = b

i3 : print(b)

oc
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Start with the architecture diagram. Label the reads and writes, and identify which of the instructions (i1, i2, i3) each process executes.

```
b = 3
d = 4
co
i1 : b = d * 2
i2 : d = b
i3 : print(b)
oc
```
Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: In this diagram, $P_a$ performs which instruction(s), and $P_b$?
Two processes are independent if the write set of each is disjoint from both the read and write sets of the other.

Q: Are \( P_a \) and \( P_b \) independent?

No

Q: How many unique outputs are possible?

In class exercise
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: What are the possible execution histories for i1, i2, and i3?
When can code be parallelized

Two processes are independent if the write set of each is disjoint from both the read and write sets of the other.

Instructions are executed sequentially in a thread, thus the only constraint is that $i_1 < i_2$.
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: Are $P_a$ and $P_b$ independent?
No

In the previous slides, we showed that using the concept of disjoint sets, threads $a$ and $b$ cannot be run concurrently ... here's the proof ...
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: Are \( P_a \) and \( P_b \) independent?

No
Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

**Q:** Are P\(_a\) and P\(_b\) independent?

**No**
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: Are $P_a$ and $P_b$ independent?

No
Two processes are independent if the write set of each is disjoint from both the read and write sets of the other.

**Q:** Are $P_a$ and $P_b$ independent?

**No**
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: Are \( P_a \) and \( P_b \) independent?

No

\[
\begin{align*}
P_a & : i1 : b = d \times 2 \\
P_b & : i2 : d = b \\
& : i3 : \text{print}(b)
\end{align*}
\]

\[
\begin{align*}
P_a & : b = 3 \\
P_b & : d = 4 \\
& : \text{co}
\end{align*}
\]

\[
\begin{align*}
P_a & : i1 : b = \text{co} \\
P_b & : i2 : d = \text{co} \\
& : i3 : \text{print}(\text{co})
\end{align*}
\]
When can code be parallelized

Two processes are independent if the write set of each is disjoint from both the read and write sets of the other.

Q: Are $P_a$ and $P_b$ independent?

No

$P_a$

i1 : b = d * 2
i2 : d = b
i3 : print(b)

$P_b$

i1 : b = 8
i3 : print(b)

$i_1 < i_2 < i_3$

$i_1 < i_3 < i_2$

$i_3 < i_1 < i_2$
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

**Q:** Are $P_a$ and $P_b$ independent?

**No**

- $P_a$: $i1 : b = d \times 2$
- $i2 : d = b$
- $i3 : \text{print}(b)$

- $P_b$: $b = 8$
- $d = 8$
- $\text{print} : 8$

**Correct order:** $i1 < i2 < i3$

**Incorrect order:** $i3 < i1 < i2$
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

**Q:** Are \( P_a \) and \( P_b \) independent?

No

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

\[ b = 3 \]
\[ d = 4 \]

\[ i_1 : b = d \times 2 \]
\[ i_2 : d = b \]
\[ i_3 : \text{print}(b) \]

\[ P_a \]

\[ P_b \]

\[ i_1 < i_2 < i_3 \]
\[ b = 8 \]
\[ d = 8 \]
\[ \text{print} : 8 \]

\[ i_1 < i_3 < i_2 \]
\[ b = 8 \]
\[ d = 8 \]
\[ \text{print} : 8 \]

\[ i_3 < i_1 < i_2 \]
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: Are \( P_a \) and \( P_b \) independent?

No

\[ a \quad b \quad c \quad d \quad e \quad f \]

\[ 3 \quad 4 \]

\[ b = 3 \]

\[ d = 4 \]

\[ \text{co} \]

\[ P_a \]

\[ \text{i1 : } b = d \times 2 \]

\[ \text{i2 : } d = b \]

\[ \text{i3 : print(b)} \]

\[ \text{oc} \]

\[ b = 8 \]

\[ d = 8 \]

\[ \text{print : 8} \]

\[ i1 < i2 < i3 \]

\[ i1 < i3 < i2 \]

\[ i3 < i1 < i2 \]
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: Are $P_a$ and $P_b$ independent?

No

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
num = 3 # Initial value num = 4 # Initial value

# Process 1
num *= 2
print(num)

# Process 2
num = 8
num = 8
print(num)
```

$i_1 < i_2 < i_3$
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: Are \( P_a \) and \( P_b \) independent?

No

---

\( P_a \)

\[ i_1 : b = d \times 2 \]

\[ i_2 : d = b \]

\[ i_3 : \text{print}(b) \]

\( P_b \)

\[ b = 8 \]

\[ d = 8 \]

\[ \text{print : 8} \]
When can code be parallelized

Two processes are **independent** if the write set of each is disjoint from both the read and write sets of the other.

Q: Are \( P_a \) and \( P_b \) independent?

No

\[ \begin{align*}
\text{P_a} & : i1 : b = d \times 2 \\
\text{P_b} & : i3 : \text{print}(b)
\end{align*} \]
Q: If you have a 4 processor computer, how would you dole out programs $p_1$, $p_2$, $p_3$ and $p_4$ which rely on variables a-j? You have NOT been told which of those variables the 4 programs read from or write to.

- $P_1$: b, d, f
- $P_2$: j, e, c
- $P_3$: h, d, g, f
- $P_4$: a, g, e, j, b, c, i

In class exercise
Q: If you have a 4 processor computer, how would you dole out programs $p_1$, $p_2$, $p_3$ and $p_4$ which rely on variables a-j? You have NOT been told which of those variables the 4 programs read from or write to.

On the board explanation
Q: Should we parallelize this for loop?

```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];
}
```
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

```
\begin{array}{ccc}
A & B & C \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
```

Q: What are the entries of the product matrix?
Shared Variable Programming

Q: Should we parallelize this for loop?

Assume you have 12 processors ... how do you dole out “chunks” of the matrix product calculation?
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
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
```

\[ P_0 = 1 \times 4 + 2 \times 4 + 3 \times 4 = 4 + 8 + 12 = 24 \]
**Shared Variable Programming**

```c
for (i = 0 to n-1, j = 0 to n-1) {
    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?

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>

\[ A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \times \begin{bmatrix} 4 & 5 & 6 \\ 4 & 5 & 6 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 24 & 30 & 36 \\ 24 & 30 & 36 \\ 24 & 30 & 36 \end{bmatrix} = C \]

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

\[ P1 = 1 \times 5 + 2 \times 5 + 3 \times 5 = 5 + 10 + 15 = 30 \]
Shared Variable Programming

Q: Should we parallelize this for loop?

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

\[P_0 = 1\times4 + 2\times4 + 3\times4 = 4 + 8 + 12 = 24\]
\[P_1 = 1\times5 + 2\times5 + 3\times5 = 5 + 10 + 15 = 30\]
\[P_2 = 1\times6 + 2\times6 + 3\times6 = 6 + 12 + 18 = 36\]
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];
}
```

A

B

C

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

```
\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}
```
**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**

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

**X**

```
B
4 5 6
4 5 6
4 5 6
```

**C**

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

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

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

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

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

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

![Parallelization Diagram]

A

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</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>
</tbody>
</table>

B

<table>
<thead>
<tr>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

C

<table>
<thead>
<tr>
<th>24</th>
<th>30</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>30</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>30</td>
<td>36</td>
</tr>
</tbody>
</table>

The “last” parallelization refers to the summation of all the little “parts”

P0 = 1*4 + 2*4 + 3*4 = 4 + 8 + 12 = 24
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?

\[
\begin{array}{ccc}
A & \times & B \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

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

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

Q: Should we parallelize this for loop?

\[
\frac{1}{23\quad \frac{23}{31}}
\]

\[a, b, c\]

\[\text{Q: Why is } c[i,j]\text{ not “inside” the } k \text{ loop?}\]

\[
\begin{align*}
\text{for } (k=0 \text{ to } n-1) \{ &  \\
& c[i,j] = 0.0; \\
& c[i,j] = c[i,j] + a[i,k]*b[k,j]; \\
& \}
\end{align*}
\]
**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?

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

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

\[
\text{P0} = 1 \times 4 + 2 \times 4 + 3 \times 4 = 4 + 8 + 12 = 24
\]

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

Q: Should we parallelize this for loop?

```
c[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];
}
```

A

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>

B

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

C

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>30</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>30</td>
<td>36</td>
</tr>
<tr>
<td>24</td>
<td>30</td>
<td>36</td>
</tr>
</tbody>
</table>

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

Q: Should we parallelize this for loop?

```c
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];
}
```

| i   | j   | c[i,j]|  | i   | j   | c[i,j]|  | i   | j   | c[i,j]|
|-----|-----|-------|  |-----|-----|-------|  |-----|-----|-------|
| 1   | 2   | 3     |  | 1   | 2   | 3     |  | 1   | 2   | 3     |
| 1   | 2   | 3     |  | 1   | 2   | 3     |  | 1   | 2   | 3     |
| 1   | 2   | 3     |  | 1   | 2   | 3     |  | 1   | 2   | 3     |

A \times B = C

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

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

Q: Should we parallelize this for loop?

A

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

C

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

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

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

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

B

C

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

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

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

\[
\begin{array}{c|c|c}
\text{A} & \text{B} & \text{C} \\
1 & 2 & 3 & 4 & 5 & 6 & 24 & 30 & 36 \\
1 & 2 & 3 & 4 & 5 & 6 & 24 & 30 & 36 \\
1 & 2 & 3 & 4 & 5 & 6 & 24 & 30 & 36 \\
\end{array}
\]

**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

```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?

To parallelize, we can divide the loop into two parts:

A

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

B

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C

<table>
<thead>
<tr>
<th></th>
<th>24</th>
<th>30</th>
<th>36</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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?

P0

i=0
j=0
k=0
c[i,j] = ...

P1

i=0
j=0
k=1
c[i,j] = ...

CSCS 322
Concurrent Programming
Concurrent programs

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**: 

```c
for [i = 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**:
- **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**: Action which indivisibly examines or changes a state
- **History, interleaving, trace**: Particular sequence of atomic actions
- **Critical Section**:

```c
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

```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**: 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
  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];
}
```

- **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
  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];
}
```

- **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**: 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

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**: 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

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

```c
for [k = 0 to n-1]
    c[i,j] = c[i,j] + a[i,k]*b[k,j];
```
Q: How do we demonstrate properties of a concurrent program?
Concurrent programs

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

Test and debug  “run the program” multiple times and “test”
Concurrent programs

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"

$m$   $n$   possible histories
3    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
Concurrent programs

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 1 6
Concurrent programs

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”

\[ m \times n \]
possible histories

<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>
</tbody>
</table>

A

B

C

We “could” test this manually

ABC

ACB

BAC

BCA

CAB

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

Test and debug

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

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?

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

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?

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

A1  First jot down
A2  ALL of the
B1  possible
B2  combinations
Q: How do we demonstrate properties of a concurrent program?

Test and debug

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  First jot down
A2  ALL of the possible combinations
B1
B2  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

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?

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

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

Then, identify those that meet the criteria:


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:

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

<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>B1</td>
<td>A1</td>
<td>B2</td>
<td>A1</td>
</tr>
<tr>
<td>B2</td>
<td>A2</td>
<td>B1</td>
<td>A1</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?

<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>
</tbody>
</table>

Then, identify those that meet the criteria:

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

A1

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

A2

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

B1

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

B2

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

- A1 B2 B1 A2
- A2 B2 B1 A1
- B1 B2 A1 A2
- B2 A1 B1 A2
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>6</td>
</tr>
</tbody>
</table>

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

$m$ $n$ possible histories

<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>6</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
Concurrent programs

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

Test and debug

$m$ threads executing $n$ atomic actions

$$
\begin{array}{c|c|c}
m & n & possible histories \\
\hline
3 & 1 & 6 \\
2 & 2 & 6 \\
3 & 2 & \\
\end{array}
$$

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

Enumerate all possible combinations
Select those that meet the dependence criteria

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

\[
\begin{array}{ccc}
m & n & possible \ histories \\
3 & 1 & 6 \\
2 & 2 & 6 \\
3 & 2 & 90 \\
\end{array}
\]

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

\[ m \] 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
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
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>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: \[ \frac{(mn)!}{(n!)^m} \]
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”

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

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**
Arms

await