Homework 5

• Will be posted tonight
• Due next Monday
• No programming task
Homework 5

- Will be posted tonight
- Due next Monday
- No programming task

Lab 7

- The sequential, and threaded version with locally scoped variables, sum the entries of the array **6 times**.
Announcement

Homework 5

- Will be posted tonight
- Due next Monday
- No programming task

Lab 7

- The sequential, and threaded version with locally scoped variables, sum the entries of the array **6 times**.
- The threaded version that uses a globally scoped variable does NOT work as intended (due to race conditions, threads writing over other thread’s results, etc.). Your goal is to fix the threaded version that uses a globally scoped variable ... that version MUST rely on the globally scoped variable.
Butterfly Barrier: Combines multiple 2-process barriers. At stage $s$, synchronize with a process $2^{s-1}$ away. The number of processes must be a power of 2.
From last time

S1

S2

Dissemination barrier
(Last stage not shown)
Today

Dissemination Barrier
Data Parallel Algorithms
Monitors
Dissemination Barrier

Q: Why use a dissemination barrier versus a butterfly barrier?
Q: What is/are the difference(s) between a butterfly and dissemination barrier?
Dissemination Barrier

Just as a butterfly barrier, in the dissemination barrier, at stage $s$, a process synchronizes with a process $2^{s-1}$ away.
Dissemination Barrier

At stage $s=1$, $2^{s-1} = 2^0 = 1$
Dissemination Barrier

S1
Dissemination Barrier

At stage $s=2$, $2^{s-1} = 2^1 = 2$
Dissemination Barrier

S1

S2

worker  worker  worker  worker  worker  worker
Dissemination Barrier

At stage $s=3$, $2^{s-1} = 2^2 = 4$
Dissemination Barrier

It seems that S2 and S3 are the same.
Dissemination Barrier

S1

S2

Q: Using a 2-process barrier, how do processes “communicate”

Dissemination barrier
(Last stage not shown)
Dissemination Barrier

- Set arrival flag of worker “on the right”
- Wait on own flag
- Clear own flag
Dissemination Barrier

Q: Why use a dissemination barrier versus a butterfly barrier?
Q: What is/are the difference(s) between a butterfly and dissemination barrier?
Q: Why use a dissemination barrier versus a butterfly barrier?
Q: What is/are the difference(s) between a butterfly and dissemination barrier?
Q: Where/how are barriers used?
Several processes execute the “same” code and work on different parts of shared data.

\[
\begin{align*}
a &= \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array} \\
b &= \begin{array}{cccccc}
1 & 3 & 6 & 10 & 15 & 21 \\
\end{array}
\end{align*}
\]

Q: What is the relationship between arrays \( a \) and \( b \)?
Data Parallel Algorithms

Several processes execute the “same” code and work on different parts of shared data.

\[
a = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

\[
\text{sum} = \begin{array}{cccccc}
1 & 3 & 6 & 10 & 15 & 21 \\
\end{array}
\]

Q: What is a sequential algorithm for computing the partial sum?
Several processes execute the “same” code and work on different parts of shared data.

\[ a = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \]

\[ \text{sum} = \begin{bmatrix} 1 & 3 & 6 & 10 & 15 & 21 \end{bmatrix} \]

Q: What is a sequential algorithm for computing the partial sum?

```java
sum[0] = 0;
for (int i=1; i<n; i++){
    sum[i] = sum[i-1] + a[i];
}
```

Q: How many computation steps (rounds) were required to calculate the partial sums when \( n = 6 \)?

Q: Can we do better?
Data Parallel Algorithms

If we wanted to find the sum of all elements of $a$, we could ...

$$a = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}$$
If we wanted to find the **sum** of all elements of \( a \), we could ...

\[
a = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

3 7 11

**Q: What’s next?**
If we wanted to find the **sum** of all elements of $a$, we could ...

$$a = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
3 & 7 & 11 & & & \\
& & & & 10 \\
\end{array}$$
If we wanted to find the **sum** of all elements of \( a \), we could ...

\[
a = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

\[
\begin{array}{ccc}
3 & 7 & 11 \\
\end{array}
\]

\[
10
\]

\[
21
\]

Q: How many computation rounds were needed assuming parallelization?
Data Parallel Algorithms

If we wanted to find the **sum** of all elements of \( a \), we could ...

<table>
<thead>
<tr>
<th>( a = )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round 1</td>
<td></td>
<td></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>
<tr>
<td></td>
<td>3</td>
<td>7</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Round 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Round 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21</td>
</tr>
</tbody>
</table>

There is a clear need for barriers. **Q: How does this help us in calculating partial sums?**

**Q:** How many computation rounds were needed assuming parallelization?

Note that round 2 cannot proceed before round 1 has completed, and round 3 cannot proceed before round 2 has completed.
If we wanted to find the **partial sum** of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
a = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6
\end{bmatrix}
\]

Goal

\[
\text{sum} = \begin{bmatrix}
1 & 3 & 6 & 10 & 15 & 21
\end{bmatrix}
\]
If we wanted to find the **partial sum** of all elements of $a$, we could ...

Use barriers to “double the distance” at which elements are added.

$\begin{align*}
    a &= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \\
    \text{sum} &= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}
\end{align*}$

Set $\text{sum}[i] = a[i]$

Goal

$\begin{bmatrix} 1 & 3 & 6 & 10 & 15 & 21 \end{bmatrix}$
If we wanted to find the **partial sum** of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
a = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

\[
\text{sum} = \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array}
\]

\[
\text{sum} = \begin{array}{cccccc}
1 & 3 & 5 & 7 & 9 & 11 \\
\end{array}
\]

Q: Are we “closer” to the having computed the partial sum?

Goal

\[
\text{sum} = \begin{array}{cccccc}
1 & 3 & 6 & 10 & 15 & 21 \\
\end{array}
\]
If we wanted to find the **partial sum** of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{array}{ccccccc}
\text{a} &=& 1 & 2 & 3 & 4 & 5 & 6 \\
\text{sum} &=& 1 & 2 & 3 & 4 & 5 & 6 \\
\text{sum} &=& 1 & 3 & 5 & 7 & 9 & 11 \\
\end{array}
\]

Q: A portion of the partial sum array has been correctly computed.
Q: What is the next step?

| Goal | \text{sum} = | 1 | 3 | 6 | 10 | 15 | 21 |
If we wanted to find the **partial sum** of all elements of $a$, we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{align*}
a &= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \\
\text{sum} &= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \\
\text{sum} &= \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 \end{bmatrix} \\
\text{sum} &= \begin{bmatrix} 1 & 3 & 6 & 10 & 14 & 18 \end{bmatrix}
\end{align*}
\]

**Goal**

\[
\begin{bmatrix} 1 & 3 & 6 & 10 & 15 & 21 \end{bmatrix}
\]
Data Parallel Algorithms

If we wanted to find the *partial sum* of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{align*}
\text{a} &= \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6
\end{array} \\
\text{sum} &= \begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6
\end{array} \\
\text{sum} &= \begin{array}{cccccc}
1 & 3 & 5 & 7 & 9 & 11
\end{array} \\
\text{sum} &= \begin{array}{cccccc}
1 & 3 & 6 & 10 & 14 & 18
\end{array}
\end{align*}
\]

| Goal | sum = | 1 | 3 | 6 | 10 | 15 | 21 |
Data Parallel Algorithms

If we wanted to find the **partial sum** of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{align*}
  a &= \begin{array}{cccccc}
     1 & 2 & 3 & 4 & 5 & 6 \\
  \end{array} \\
  \text{sum} &= \begin{array}{cccccc}
     1 & 2 & 3 & 4 & 5 & 6 \\
  \end{array} \\
  \text{sum} &= \begin{array}{cccccc}
     1 & 3 & 5 & 7 & 9 & 11 \\
  \end{array} \\
  \text{sum} &= \begin{array}{cccccc}
     1 & 3 & 6 & 10 & 14 & 18 \\
  \end{array} \\
\end{align*}
\]

Goal

\[
\begin{array}{cccccc}
  \text{sum} &= & 1 & 3 & 6 & 10 \quad & 15 \quad & 21 \\
\end{array}
\]
If we wanted to find the **partial sum** of all elements of \(a\), we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{array}{c}
a = \\
\text{sum} = \\
\text{sum} = \\
\text{sum} = \\
\end{array}
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 \\
1 & 3 & 5 & 7 & 9 & 11 \\
1 & 3 & 6 & 10 & 14 & 18 \\
\end{array}
\]

Goal
\[
\begin{array}{cccccc}
1 & 3 & 6 & 10 & 15 & 21 \\
\end{array}
\]
If we wanted to find the **partial sum** of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{align*}
a &= \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\
\text{sum} &= \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\
\text{sum} &= \begin{array}{cccccc} 1 & 3 & 5 & 7 & 9 & 11 \\
\text{sum} &= \begin{array}{cccccc} 1 & 3 & 6 & 10 & 14 & 18 \\
\end{array}
\end{array}
\end{array}
\end{align*}
\]

Q: Are we “closer” to the having computed the partial sum?

<table>
<thead>
<tr>
<th>Goal</th>
<th>( \text{sum} = )</th>
</tr>
</thead>
</table>
|      | \begin{array}{cccccc} 1 & 3 & 6 & 10 & 15 & 21 \\
\end{array}  |
If we wanted to find the **partial sum** of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{align*}
a &= 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \\
\text{sum} &= 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \\
\text{sum} &= 1 \quad 3 \quad 5 \quad 7 \quad 9 \quad 11 \\
\text{sum} &= 1 \quad 3 \quad 6 \quad 10 \quad 14 \quad 18 \\
\end{align*}
\]

All of these can be done concurrently

**Q:** What is the next step?

**Goal**

\[
\begin{align*}
\text{sum} &= 1 \quad 3 \quad 6 \quad 10 \quad 15 \quad 21 \\
\end{align*}
\]
If we wanted to find the **partial sum** of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{array}{cccccc}
    \text{a} & = & 1 & 2 & 3 & 4 \\
    \text{sum} & = & 1 & 2 & 3 & 4 \\
    \text{sum} & = & 1 & 3 & 5 & 7 \\
    \text{sum} & = & 1 & 3 & 6 & 10 \\
    \text{sum} & = & 1 & 3 & 6 & 10 & 15 \\
\end{array}
\]

**Goal**

\[
\begin{array}{cccccc}
    \text{sum} & = & 1 & 3 & 6 & 10 & 15 & 21 \\
\end{array}
\]
If we wanted to find the **partial sum** of all elements of \( a \), we could ...

Use barriers to “double the distance” at which elements are added.

\[
\begin{array}{cccccc}
\text{a} = & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{sum} = & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{sum} = & 1 & 3 & 5 & 7 & 9 & 11 \\
\text{sum} = & 1 & 3 & 6 & 10 & 14 & 18 \\
\text{sum} = & 1 & 3 & 6 & 10 & 15 & 21 \\
\end{array}
\]

Goal: \[
\begin{array}{cccccc}
\text{sum} = & 1 & 3 & 6 & 10 & 15 & 21 \\
\end{array}
\]

Q: Are we done?
Use barriers to “double the distance” at which elements are added.

Q: Advantage/speedup? : After $\log_2 n$ rounds all partial sums have been computed
Data Parallel Algorithms

\begin{verbatim}
int a[n], sum[n], old[n];
process Sum[i = 0 to n-1] {
    int d = 1;
    sum[i] = a[i];  /* initialize elements of sum */
    barrier(i);
    # SUM: sum[i] = (a[i-d+1] + ... + a[i])
    while (d < n) {
        old[i] = sum[i];  /* save old value */
        barrier(i);
        if ((i-d) >= 0)
            sum[i] = old[i-d] + sum[i];
        barrier(i);
        d = d+d;  /* double the distance */
    }
}
\end{verbatim}

Copy from “last” step

Perform calculation

\texttt{barrier(i)} implements a barrier for process \texttt{i}, and it returns after all \texttt{n} processes have called \texttt{barrier}.
Monitors

By this point we’ve talked about semaphores, locks, and barriers. Each of these allow us to impose mutex and process synchronization among multiple concurrent processes.
Monitors

By this point we’ve talked about semaphores, locks, and barriers. Each of these allow us to impose mutex and process synchronization among multiple concurrent processes.

Q: Think back to semaphores. What were some of their shortcomings?

Think back to the midterm exam semaphore question. Easy? Difficult?

on the board explanation
Monitors

By this point we’ve talked about semaphores, locks, and barriers. Each of these allow us to impose mutex and process synchronization among multiple concurrent processes.

Q: Think back to semaphores. What were some of their shortcomings?

Semaphores are a low-level mechanism – it is easy to make errors when using them.

• You (a programmer) must be careful NOT to omit an increment or decrement somewhere in your code, or to use the wrong semaphore (in case there are multiple ones being used)
Monitors

By this point we’ve talked about semaphores, locks, and barriers. Each of these allow us to impose mutex and process synchronization among multiple concurrent processes.

Q: Think back to semaphores. What were some of their shortcomings?

Semaphores are a low-level mechanism – it is easy to make errors when using them.

- You (a programmer) must be careful NOT to omit an increment or decrement somewhere in your code, or to use the wrong semaphore (in case there are multiple ones being used)
- Semaphores are also global, thus you must examine the entire program to know how they are used
By this point we’ve talked about semaphores, locks, and barriers. Each of these allow us to impose mutex and process synchronization among multiple concurrent processes.

**Q: Think back to semaphores. What were some of their shortcomings?**

Semaphores are a low-level mechanism – it is easy to make errors when using them.

- You (a programmer) must be careful NOT to omit an increment or decrement somewhere in your code, or to use the wrong semaphore (in case there are multiple ones being used)
- Semaphores are also **global**, thus you must examine the entire program to know how they are used
- Semaphores provide BOTH mutual exclusion and synchronization techniques, but what if we want to use these concepts independently?
Monitors

• Provide a more object-oriented style approach to mutex and synchronization
• More structure than a semaphore
• A data abstraction mechanism
• Can be implemented easily
• Access to monitor variables is only through interface
• Mutual exclusion of all monitor procedures is implicit (procedures in the same monitor cannot be executed concurrently)
• Condition synchronization is via condition variables
Monitors

- Provide a more object-oriented style approach to mutex and synchronization
- More structure than a semaphore
- A data abstraction mechanism
- Can be implemented easily
- Access to monitor variables is only through interface
- Mutual exclusion of all monitor procedures is implicit (procedures in the same monitor cannot be executed concurrently)
- Condition synchronization is via condition variables

```plaintext
Monitor

The object’s state ... **not** accessible from the “outside”

The “methods” that are accessible from the outside
```

// condition variables
// procedures
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as **passive**.

<table>
<thead>
<tr>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>The object’s state <strong>not</strong> accessible from the “outside”</td>
</tr>
<tr>
<td>// condition variables</td>
</tr>
<tr>
<td>The “methods” that are accessible from the outside</td>
</tr>
<tr>
<td>// procedures</td>
</tr>
</tbody>
</table>
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as **passive**.
- A monitor is used to group together the representation and implementation of a shared resource.

The object’s state ... **not** accessible from the “outside”

<table>
<thead>
<tr>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>// condition variables</td>
</tr>
<tr>
<td>// procedures</td>
</tr>
</tbody>
</table>

The “methods” that are accessible from the outside
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as **passive**.
- A monitor is used to group together the representation and implementation of a shared resource.
- In different languages, they are created in different ways.

- The condition variables are most often static – why?

<table>
<thead>
<tr>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>The object’s state <strong>not</strong> accessible from the “outside”</td>
</tr>
<tr>
<td>// condition variables</td>
</tr>
<tr>
<td>The “methods” that are accessible from the outside</td>
</tr>
<tr>
<td>// procedures</td>
</tr>
</tbody>
</table>
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as *passive*.
- A monitor is used to group together the representation and implementation of a shared resource.
- In different languages, they are created in different ways.

Monitor

This is the abstract monitor “type”, a class that has functionality common to ALL monitors (such as signal).
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as **passive**.
- A monitor is used to group together the representation and implementation of a shared resource.
- In different languages, they are created in different ways.

![Diagram of Monitor and mName]

This specific monitor is the representation of the abstract Monitor object. It provides functionality and variables for representing and implementing access to a shared resource.
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as **passive**.
- A monitor is used to group together the representation and implementation of a shared resource.
- In different languages, they are created in different ways.

This specific monitor is the representation of the abstract Monitor object. It provides functionality and variables for representing and implementing access to a shared resource.

The variable(s) `con_var` is/are static.

Q: What is a consequence of this?
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as **passive**.
- A monitor is used to group together the representation and implementation of a shared resource.
- In different languages, they are created in different ways.

Multiple threads/workers may be using **mName** to coordinate sharing a shared resource.
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as **passive**.
- A monitor is used to group together the representation and implementation of a shared resource.
- In different languages, they are created in different ways.

Because of the static designation of `con_var`, if there are two or more threads/processes using `mName`, there is only 1 copy of `con_var`.
Monitors

- **Active processes** (threads running concurrently) interact by calling procedures in the same monitor.
- The monitors are referred to as **passive**.
- A monitor is used to group together the representation and implementation of a shared resource.
- In different languages, they are created in different ways.

- Only procedure names are visible to "outside" of the monitor.
- Monitors may not access variables declared outside of the monitor.
- Permanent variables are initialized before any procedures are called.

```
con_var
procedure 1
procedure 2
```
Monitors

```java
monitor mName{
    // declare permanent values
    // initialization statements
    // procedures
}
```

To alter the internal state:

```java
call mName.opName(argument(s))
```
Monitors State Transitions