Lecture 11

Midterm Review
Announcements

• Midterm on Thursday
• No Quiz this week
• The assignment will be discussed in section
Midterm review
Terms and concepts

• Know the definition and significance of ….
• Parallel speedup and efficiency, super-linear speedup, strong scaling, weak scaling
• Amdahl's law, serial bottlenecks
• Granularity
• Data dependencies, loop carried dependence
• Load balancing, loop distribution
Terms and concepts

• SPMD, MIMD, SIMD
• Multiprocessors and multicomputers
• Cache coherence and consistency
• False sharing
• SMPs, snooping
Implementation techniques

- Threads
- OpenMP
- Mutexes, semaphores, and barriers
- Critical sections, race conditions
False sharing in higher dimension arrays

- Compare with distributed memory solution
Practice problems

The following code performs barrier synchronization, where $nt = \text{number of threads}$

Explain how the code works, demonstrating correct operation on 3 threads
Practice problems

Semaphore arrival(..), departure(..);
int Count;
Void barrier(. . . ){
    arrival.aquire();
    Count++;
    If (Count <$nt) arrival.release();
    Else departure.release()
    departure.aquire();
    Count--;
    If (count > 0) departure.release()
    Else arrival.release()
    Return;
}

Why are two binary semaphores needed: arrival and departure

How must they be initialized to ensure correctness?

How must Count be initialized?
OpenMP

List all possible outputs that result when the following OpenMP annotated C code is executed:

```c
#pragma omp parallel for shared(j,k)
for ( int i=0, j=0, k=0; i< 2; i++ )
    j = j + 10;
    k = j + 10;
}
cout << "k = " << k << endl;
```
OpenMP

Now with critical sections (see $pub/Midterm/openmp_ex.C)

```
#pragma omp parallel for shared(j,k)
for ( int i=0, j=0, k=0; i< 2; i++ )
    #pragma omp critical
    j = j + 10;
    #pragma omp critical
    k = j + 10;
}
cout << "k = " << k << endl;
```
Performance

- You observe the following running times for a parallel program running a fixed workload N
- What fraction of the total running time that runs on a single processor?
- What will the running time be on 4 processors?
- What is the speedup and efficiency on 8 processors?
- What is the maximum possible speedup on an infinite number of processors?

<table>
<thead>
<tr>
<th>NT</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10000</td>
</tr>
<tr>
<td>2</td>
<td>6000</td>
</tr>
<tr>
<td>8</td>
<td>3000</td>
</tr>
</tbody>
</table>
In the following code a thread increments a shared variable until reaching a given maximum value.

```java
class Q2 extends Thread {
    ... 
    public void run() {
        synchronized(sharedVar) {
            while (sharedVar < MAX_VAL) {
                sharedVar++;
            }
        }
    }
}
```

- If we create two threads running object Q2, both try to update the shared object. What will happen to the shared variable?
- If we create and run 100 Q2 objects, what amount thread parallelism would be achieved? Explain.
- Will all of the Q2 objects get to update the shared variable? If one of them does not, we call that effect "starvation." If starvation is a possibility, suggest a way to avoid such a situation. Explain in detail why our approach would ensure fairness.
Consistency

Assume that memory is sequentially consistent
How many times can procedure foo() be run?

Assume that both X and Y are shared int, and have been initialized to zero

<table>
<thead>
<tr>
<th>Thread 0</th>
<th>Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>X = 1;</td>
<td>Y = 1;</td>
</tr>
<tr>
<td>If (y==0)</td>
<td>If (x == 0)</td>
</tr>
<tr>
<td>foo();</td>
<td>foo();</td>
</tr>
</tbody>
</table>
Load Balancing

• We have an int array of length 1024, and share the computation evenly among the processors. The workload consists of updating the array.
• All the updates take the same amount of time, 1 second

a. What is the running time on 4 processors?
b. What is the running time on 5 processors?
c. What is the running time on 16 processors?
d. Describe the workload distribution scheme(s) you used to share the workload.
Load Balancing - Part II

• Now, we use a different algorithm to update the array.
• Some array elements take longer than others to update.
• In order to balance the workloads, we break each array into 32 contiguous chunks.
• 8 of these chunks complete in 10 seconds, and the rest 5 seconds.

a. What is the best possible running time we can expect on 4 processors, using any workload distribution scheme, and ignoring overhead costs?
b. What is the best possible running time we can expect on 5 processors?
c. What is the best possible running time on 16 processors?
d. Describe the workload distribution scheme(s) you used to balance the workload.