Lecture 10

Midterm Review
Announcements

• Abe accounts
• Midterm on Thursday
• Sign up sheet
• No Quiz
• Office hours cancelled on Thursday
Blocked Matrix Multiplication

for i = 0 to N-1
    for j = 0 to N-1
        // load each block C[i,j] into cache, once :
        // B= n/N = cache line size
        for k = 0 to N-1
            // load each block A[i,k] and B[k,j] N^3 times
            // = 2N^3 × (n/N)^2
            // write each block C[i,j] once :
        Total:
        (2*N+2)*n^2
The results

<table>
<thead>
<tr>
<th>N,B</th>
<th>Unblocked Time (sec) (grind time in microsec)</th>
<th>Blocked Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>256, 64</td>
<td>0.55 (0.033)</td>
<td>0.048 (.0029)</td>
</tr>
<tr>
<td>512,128</td>
<td>15 (0.11)</td>
<td>0.37 (.0028)</td>
</tr>
</tbody>
</table>
Optimizing the block size

N=512
Comparison
Log Y scale improves readability
Crossover

![Graph showing Crossover]

- **Unblocked**
- **Blocked**

The graph illustrates the relationship between $T_{\text{grind}}$ and $N$, with $T_{\text{grind}}$ plotted on the y-axis and $N$ on the x-axis. The data points show a trend that increases with $N$. The blue line represents the Unblocked scenario, while the green line represents the Blocked scenario.
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
Terms and concepts

- PRAMs, CRCW
- SPMD, MIMD, SIMD
- Multiprocessors and multicomputers
- Cache coherence and consistency
- False sharing
- Snooping, directories
- NUMAs and SMPs
- Processor geometry
Implementation techniques

• Threads
• pthreads
• OpenMP
• Mutexes, semaphores, and barriers
• Critical sections, race conditions
• Red-Black/ Odd-even
False sharing in higher dimension arrays

- Compare with distributed memory solution
Practice problems

The following code performs barrier synchronization, where $n_{proc} = \text{number of processors}$.

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

Void barrier( . . . ){
    P(arrival);
    Count++;
    If (Count < n$proc) V(arrival);
    Else V(departure);
    P(departure);
    Count--;
    If (count > 0) V(departure);
    Else V(arrival);
    Return;
}

Why are two binary semaphores needed: arrival and departure

How must they be initialized to ensure correctness?

How must Count be initialized?
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

#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;
Consistency

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

Shared int X=0, Y=0;

<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>