Lecture 8

Programming with Java Threads
Announcements

• Quiz
• No office hours today, by appointment on Friday
Programming with Java Threads
Java Threads

• A Java program begins with one thread running main()

• Continues to run until
  ▶ A call to Runtime::exit() has been made
  ▶ All threads have terminated

• We can create threads by extending java.lang.Thread
public class HelloT extends Thread {
    private int _tid;

    public HelloT(int tid) {_tid = tid; }

    public void run() {
        System.out.println ("Hello world from thread " + _tid + "!");
    }
}
Driver

```
int n = ...;
HelloT[] threads = new HelloT[n];

for (int i=0; i<n; i++)
    threads[i] = new HelloT(i);

for (int i=0; i<n; i++)
    threads[i].start();
```
Using the Runnable interface

• Our threaded class cannot extend any other classes
• Alternatively, we may implement the java.langRunnable interface
• See reference in the reading schedule
The Trapezoidal rule
The trapezoidal rule

• Use the trapezoidal rule to numerically approximate the definite integral

\[ \int_a^b f(x) \, dx \]
How the trapezoidal rule works

• Divide the interval \([a,b]\) into \(n\) segments of size \(h = 1/n\)

• Approximate the area under an interval using a trapezoid

• Area under the \(i^{\text{th}}\) trapezoid
  \[
  \frac{1}{2} (f(a+i\times h)+f(a+(i+1)\times h)) \times h
  \]

• Area under the entire curve
  \(\approx\) sum of all the trapezoids
Reference material

• For a discussion of the trapezoidal rule
  http://en.wikipedia.org/wiki/Trapezoidal_rule

• A applet to carry out integration

• Code on ieng6
  Serial and threaded codes
  $pub/examples/\text{trapz}$
Serial code (Following Pacheco)

main() {
    float f(float x) { return x*x; } // Function we're integrating

    float h = (b-a)/n; // h = trapezoid base width
        // a and b: endpoints
        // n = # of trapezoids
    float integral = (f(a) + f(b))/2.0;

    float x; int i;

    for (i = 1, x=a; i <= n-1; i++) {
        x += h;
        integral = integral + f(x);
    }
    integral = integral*h;
}
The parallel algorithm

• Decompose the integration interval into sub-intervals, one per thread
• Each thread computes the integral on its local subdomain
• Threads combine their local integrals into a global one
public class TrapzTh extends Thread{
    private int tid;
    private double integral;

    public TrapzTh(int tid){ this.tid = tid; }
    double getIntegral(){ return integral; }
    static double f(double x){ return x*x; }

    public void run() {
        get nt, a, b, n ....
        int local_n = n/nt;
        double local_a = a + (tid*local_n)*h, local_b = local_a + local_n*h;
        integral = (f(local_a) + f(local_b))/2.0 ;

        double x = local_a;
        for (int i = 1; i < local_n; i++) {
            x += h; integral += f(x); }
    }
}
long timing = System.currentTimeMillis();
TrapzTh []threads = new TrapzTh[nt];
for (int i=0; i<nt; i++)
    (threads[i] = new TrapzTh(i)).start();

double integral = 0;
for (int i=0; i<nt; i++){
    try {
        threads[i].join();
    } catch (InterruptedException e) {
    }
    integral += threads[i].getIntegral();
}

double h = (b-a)/n;       // Trapezoid base width
integral *= h;
timing += System.currentTimeMillis();
Performance

• Scaling
• What if we are running on 1000 cores?

<table>
<thead>
<tr>
<th># Threads</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (sec)</td>
<td>2.18</td>
<td>1.00</td>
<td>0.500</td>
<td>0.282</td>
</tr>
<tr>
<td>Speedup</td>
<td>1.00</td>
<td>x2.18</td>
<td>x4.36</td>
<td>x7.73</td>
</tr>
</tbody>
</table>
Effects of reordering

<table>
<thead>
<tr>
<th>Threads</th>
<th>Integral</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.3333333331321953</td>
</tr>
<tr>
<td>2</td>
<td>0.3333333281215109</td>
</tr>
<tr>
<td>4</td>
<td>0.333333312385049</td>
</tr>
<tr>
<td>8</td>
<td>0.3333333230529394</td>
</tr>
</tbody>
</table>
Machine arithmetic

double x = 0;
for(int i=0;i<10;i++) {
    x+=0.1;
    System.out.println(x);
}
Binary representation of decimals

\[
\begin{align*}
0.3333333333 \\
+ 0.3333333333 \\
= 0.6666666666 \\
+ 0.3333333333 \\
= 0.9999999999
\end{align*}
\]

mindprod.com/jgloss/floatingpoint.html