CSC352
Week #5 — Spring 2017
Making the Game of Life Parallel
https://www.youtube.com/watch?v=CgOcEZinQ2I
Serial Version

- Study it
- Run it on your laptop
- Use both dish and dish2 as the array of live cells, and see how they evolve

login to your 352b account

getchcopy GameOfLife.java
javac GameOfLife.java
java GameOfLife

Other option:

2-Thread Version

• As a group, discuss the different tissues associated with parallelizing the Game of Life and running it with two threads.

• List all the issues that must be addressed on the whiteboard

• How will you verify the correctness of the parallel version?

• Play-out (human play) the execution of the 2-thread program: two people or two groups play the roles of the two threads.
Group Work!

Could be Usefull...

- **What is a BlockingQueue?**
  
  `BlockingQueue` is a queue which is **thread safe** to insert or retrieve elements from it. Also, it provides a mechanism which blocks requests for inserting new elements when the queue is full or requests for removing elements when the queue is empty, with the additional option to stop waiting when a specific timeout passes. This functionality makes `BlockingQueue` a nice way of implementing the Producer-Consumer pattern, as the producing thread can insert elements until the upper limit of `BlockingQueue` while the consuming thread can retrieve elements until the lower limit is reached and of course with the support of the aforementioned blocking functionality.

  [https://examples.javacodegeeks.com/core-java/util/concurrent/java-blockingqueue-example/](https://examples.javacodegeeks.com/core-java/util/concurrent/java-blockingqueue-example/)
Thread safe: Implementation is guaranteed to be free of race conditions when accessed by multiple threads simultaneously.
Using a BlockingQueue

“A Queue that additionally supports operations that wait for the queue to become non-empty when retrieving an element, and wait for space to become available in the queue when storing an element”

- BlockingQueue is an interface in java.util.concurrent
- Need to use an implementation of it:
  - ArrayBlockingQueue
  - DelayQueue
  - LinkedBlockingQueue
  - PriorityBlockingQueue
  - SynchronousQueue

```java
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class UsingQueues {

    public static void main(String[] args) throws InterruptedException {
        BlockingQueue<Integer> toWorkerQ = new ArrayBlockingQueue<Integer>(2);
        BlockingQueue<Integer> fromWorkerQ = new ArrayBlockingQueue<Integer>(2);

        // create a worker and give it the two queues
        DemoThread t = new DemoThread(fromWorkerQ, toWorkerQ);

        // start thread
        t.start();

        // wait 1/2 second
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // send work to worker
        toWorkerQ.put(100);

        // wait for answer back from worker
        int x = fromWorkerQ.take();

        // display the result
        System.out.println("x = " + x);
    }
}
```

/**
 * DemoThread
 */

class DemoThread extends Thread {
    BlockingQueue<Integer> sendQ;
    BlockingQueue<Integer> receiveQ;

    DemoThread( BlockingQueue<Integer> sendQ,
                 BlockingQueue<Integer> receiveQ ) {
        this.sendQ = sendQ;
        this.receiveQ = receiveQ;
    }

    public void run(){
        int x=0;

        // block until there's something in the queue
        try {
            x = receiveQ.take();
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }

        // do some computation
        x = x*2;

        // send results back
        try {
            sendQ.put( x );
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
Implement the 2-Thread Game of Life in Java

Play out Serial Version

Play out Parallel Version
The following slides present Tips and Tricks...
Defining the Number of Threads at Execution Time
```java
public class UsingQueuesN {
    public static void main(String[] args) throws InterruptedException {
        if (args.length < 1) {
            System.out.println("Syntax: java UsingQueuesN n");
            System.out.println(" where n = # of threads");
            return;
        }
        int N = Integer.parseInt(args[0]);
        BlockingQueue<Integer> toWorkersQ = new ArrayBlockingQueue<Integer>(2*N);
        BlockingQueue<Integer> fromWorkersQ = new ArrayBlockingQueue<Integer>(2*N);
        // create a worker and give it the two queues
        DemoThreadN[] threads = new DemoThreadN[N];
        for (int i=0; i<N; i++) {
            DemoThreadN t = new DemoThreadN(i, fromWorkersQ, toWorkersQ);
            t.start();
            threads[i] = t;
        }
        // wait 1/2 second
        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        // send same amount of work to each worker
        for (int i=0; i<N; i++)
            toWorkersQ.put(100);
        // wait for answer back from worker
        for (int i=0; i<N; i++) {
            int x = fromWorkersQ.take();
            // display the result
            System.out.println("x = " + x);
        }
    }
}
```

class DemoThreadN extends Thread {
    private BlockingQueue<Integer> sendQ;
    private BlockingQueue<Integer> receiveQ;
    private int Id;

    DemoThreadN( int Id,
         BlockingQueue<Integer> sendQ,
         BlockingQueue<Integer> receiveQ ) {
        this.Id = Id;
        this.sendQ = sendQ;
        this.receiveQ = receiveQ;
    }

    public void run(){
        int x=0;

        // block until there's something in the queue
        try {
            x = receiveQ.take();
        } catch (InterruptedException e1) {
            e1.printStackTrace();
        }

        // do some computation
        x = x*( Id + 1 );

        // send results back
        try {
            sendQ.put( x );
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
Measuring Performance
• Pick setup that will not be slowed down by OS or non necessary IO operations

• Pick **best** serial algorithm available

• **Tune** the parallel version

• Keep the conditions **constant** (same grid size)

• Measure the **average** execution time of several runs for each case

• Use shell **scripts**! (See next slide)

• Pick several possible **measures of performance**
  • speedup
  • throughput
  • ?
Using Shell Scripts

http://www.science.smith.edu/dftwiki/index.php/CSC352:_Using_Bash,_an_example
Create a program that gets its degree of parallelism from the command line

```java
class PrintN {
    public static void main( String[] args ) {
        int N = Integer.parseInt( args[0] );
        System.out.println( "I got " + N );
    }
}
```

class PrintN {
    public static void main( String[] args ) {
        int N = Integer.parseInt( args[0] );
        System.out.println( "I got " + N );
    }
}

# at the Linux prompt:
bash
javac PrintN.java
for i in 1 2 3 4 5 6 7 8 9 10 ; do
    java PrintN $i
done

Run the program once in a loop from the command line…
```java
class PrintN {
    public static void main( String[] args ) {
        int N = Integer.parseInt( args[0] );
        System.out.println( "I got " + N );
    }
}
```

```bash
# at the Linux prompt:
bash
javac PrintN.java
for i in 1 2 3 4 5 6 7 8 9 10 ; do
    java PrintN $i
done
```

```bash
#!/bin/bash
# runPrintN.sh
#
javac PrintN.java
for i in 1 2 3 4 5 6 7 8 9 10 ; do
    java PrintN $i
done
```

Embed the commands just typed at the prompt into a Bash shell script
class PrintN {
    public static void main( String[] args ) {
        int N = Integer.parseInt( args[0] );
        System.out.println( "I got " + N );
    }
}

# at the Linux prompt:
bash
javac PrintN.java
for i in 1 2 3 4 5 6 7 8 9 10 ; do
    java PrintN $i
done

#! /bin/bash
# runPrintN.sh
#
javac PrintN.java
for i in 1 2 3 4 5 6 7 8 9 10 ; do
    for j in 1 2 3 ; do
        /usr/bin/time java PrintN $i
    done
done

Run each program a few times for the same level of parallelism, and measure execution time for each run…
Plotting the Resulting Timing Information With R
This R-Markdown illustrates how to quickly display a graph of the average execution times of an application running on 1 to 20 threads.

```{r}
noThreads <- c( 1, 2, 4, 8, 16, 20 )
execTimes <- c( 10, 8.5, 7.0, 6.0, 5.5, 7.3 )

jpeg( '/Users/thiebaut/Desktop/executionTimes.jpg' )
plot( noThreads, execTimes, type="b", col="blue",
     xlab="Number of Threads", ylab="Avg. Execution Time (s)"
)
dev.off()

plot( noThreads, execTimes, type="b", col="blue",
     xlab="Number of Threads", ylab="Avg. Execution Time (s)"
)
```
Make sure that the graph clearly shows **POINTS** and that the lines are understood to show the trend.