Parellel

Christian Winwood
A02074021

February 1, 2021

1 Implementation

My implementation was fairly simple. First I decided to set each column equal to a process and have process '0' represent the master. The second thing I decided was what to pass in the message. The message I am passing is going to represent the current row that the ball is on.

1.1 Master

I start with the master process. It passes however many balls there are to the middle process to start. After that it waits until it receives every ball back from the process. Once every ball is received it finally sends out a kill process command and exits itself.

1.2 Columns

Each column is going to wait until they receive a message, or ball, from someone else. They then randomly generate a direction and pass the ball in that direction. When a ball they receive is at the last row, they increase the amount of balls they received and send a message back to master saying one ball is completed. When they receive the kill command, they finish the program and end.

1.3 Passing

Passing is a little more complicated then just passing left and right. This is because the pegs are actually offset on every other row. To navigate this, I first generate a random number to see which direction to pass. Then, depending on if I am on an even or odd row, I either pass that direction or pass to myself.

2 Running

The .cpp file is provided. In order to run you must first use the command

```
mpic++ main.cpp
```

which will create a new file called 'a.out'. We then need to run this with 8 processes (7 columns + 1 master). To do this use the command

```
mpirun --oversubscribe -np 8 a.out
```

The process will then run with 1500 balls dropped.

2.1 Desired Output

After running the application, you will get an output of each column and the amount of balls that ended up in that column. It should be normally distributed or fairly close to it.