ECSE 420 Assignment 1
Group 16

student1 : Erdong Luo
ID: 260778475

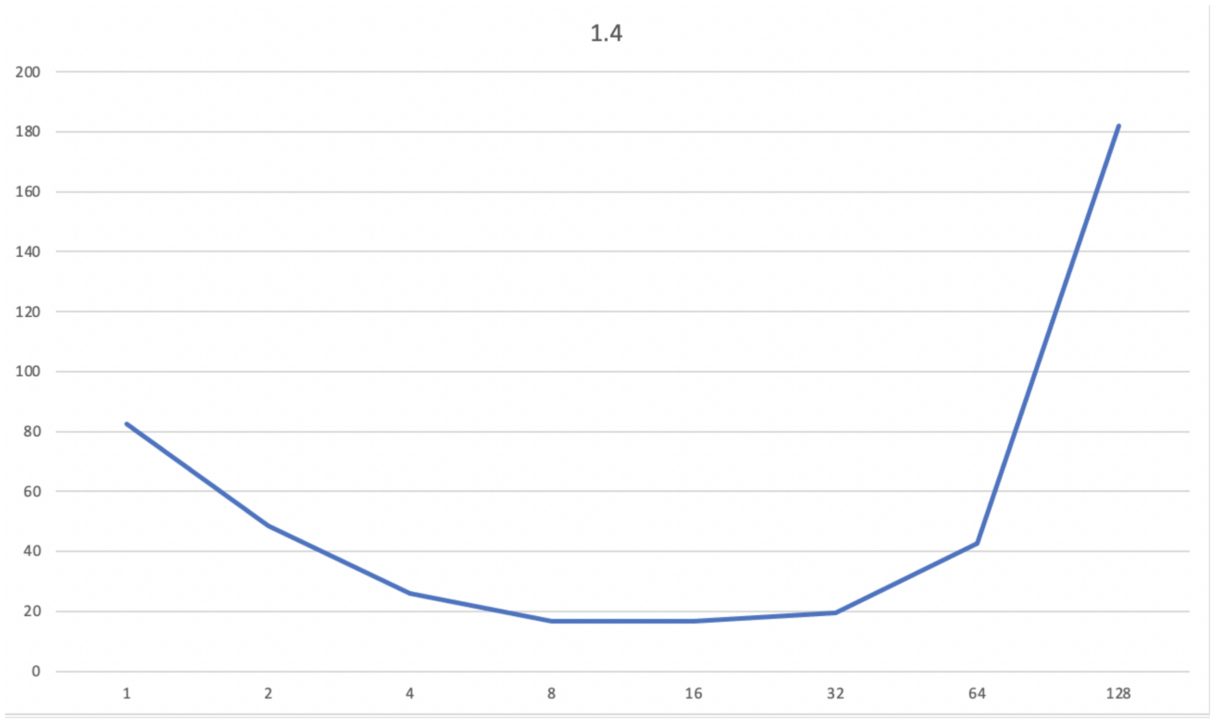
student2 : Sameen Mahtab
ID: 260737048

Appendix:

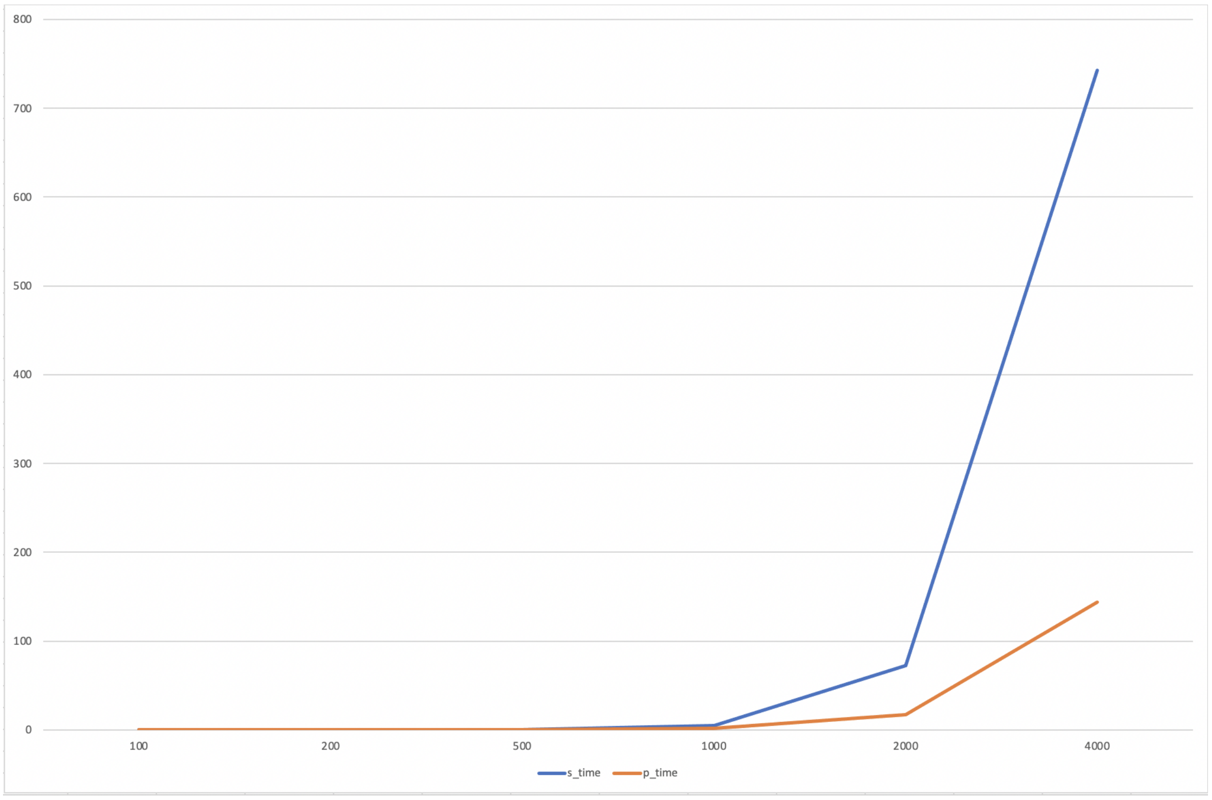
The following source codes are provided in the zip:
MatrixMultiplication.java
Deadlock.java
Deadlocking_dining_philosophers.java
No_deadlock.java
DiningPhilosophers.java

The explanations for the codes are in the same order below.

Question 1.4



Question 1.5



Question 1.6

Graph 1.4: Runtime decreases until a certain point. Afterwards, it continually increases. This makes sense, as after a certain point, the overhead generated from creating and switching threads causes diminishing returns, and eventually a decrease in performance (past the optimal thread count). This can be explained by the fact that there can only be so many actual physical threads, and past a certain point the threads being created and used are virtual, which creates more and more overhead.

Graph 1.5: When the matrix size is small, sequence multiplication is faster. When the matrix size is large, parallel multiplication is faster. Because for the sequential multiplication, the complexity is O(n^3). When the matrix size grows, the run time will increase dramatically. But for parallel multiplication, when the matrix is small, the overhead generated by creating and switching threads is large enough to cause it to run slower than sequential multiplication. But when the matrix is large, it is faster due to the low run time complexity.

Question 2.1

In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process. [1]

Question 2.2

1. We can set an order for resources. In the example, we have two tasks, task A and task B. We also have resource 1 and resource 2. In the normal condition, A might have 2 and B might have 1, thus causing deadlock. If we set an order for the resources, task A must have resource 1 to request for resource 2. We can avoid this. Then A can’t hold resource 2 while B is holding resource 1.
2. We can let a thread release all the resources after some time to let other threads to complete their tasks.

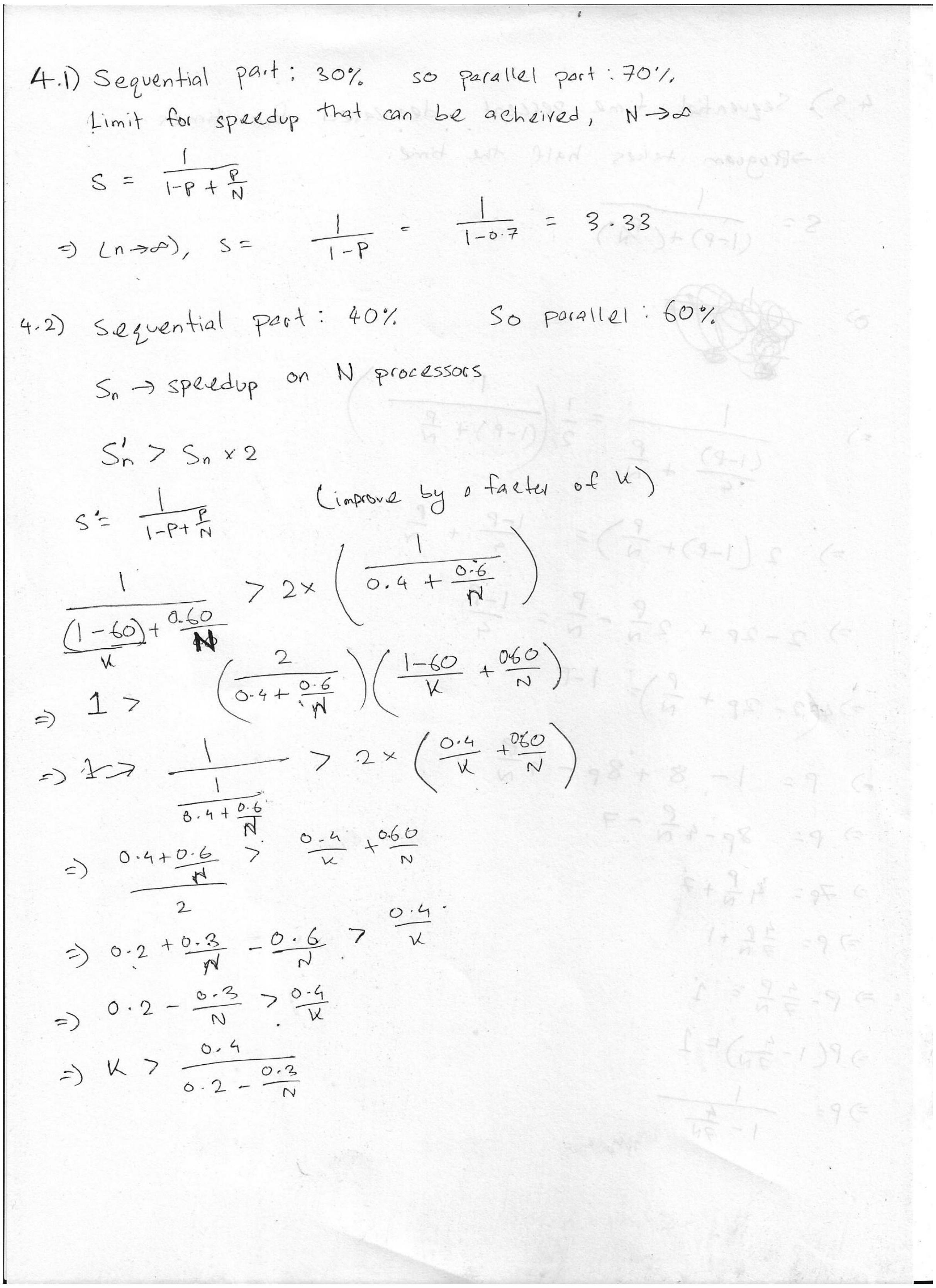
3. We can use Banker’s algorithm to detect deadlock. [2]

Question 3

In the first part of the dining philosophers problems, there are no measures to prevent deadlock or starvation. Deadlock occurs when a thread searches for resources held by another thread. After running the program Deadlocking_dining_philosophers.java the program runs into deadlocks as there is nothing preventing a philosopher from searching for a missing chopstick. All of the philosophers initially try to pick up a chopstick from the same side. For the solution for deadlock, we try to prevent circular wait. A philosopher is stopped from waiting if a chopstick is not available. He returns to thinking. The solution to prevent deadlock is in the code No_deadlock.java.

Reentrantlock synchronizes methods that try to access shared resources. trylock() is used to make sure the lock is not held by another thread. Starvation will occur if a number of philosophers don’t eat (get access to resources) for a very long time. For the third part, in order to eliminate starvation, the threads have been given priorities. The priority of a thread is reset to minimum after that thread has picked up two chopsticks, eaten and put the chopsticks back. When a thread has been in the thinking state for a long while, it’s priority is maximized. The DiningPhilosophers.java program is a deadlock and starvation free solution for the Dining Philosophers Problem.

Question 4



4.1) Sequential part : 30% so parallel part : 70%

Limit for speedup that can be acheived, $N \to \infty$

$$S = \frac{1}{1-P+\frac{P}{N}}$$

$$\Rightarrow (n \to \infty), \quad S = \frac{1}{1-P} = \frac{1}{1-0.7} = 3.33$$

4.2) Sequential part : 40% So parallel : 60%

$S_n \to$ speedup on N processors

$$S_n' > S_n \times 2$$

$$S' = \frac{1}{1-P+\frac{P}{N}} \quad \text{(improve by a factor of K)}$$

$$\frac{1}{\frac{(1-60)}{K} + \frac{0.60}{N}} > 2 \times \left(\frac{1}{0.4 + \frac{0.6}{N}} \right)$$

$$\Rightarrow 1 > \left(\frac{2}{0.4 + \frac{0.6}{N}} \right) \left(\frac{1-60}{K} + \frac{0.60}{N} \right)$$

$$\Rightarrow \cancel{1} > \frac{1}{\frac{1}{0.4+\frac{0.6}{N}}} > 2 \times \left(\frac{0.4}{K} + \frac{0.60}{N} \right)$$

$$\Rightarrow \frac{0.4 + \frac{0.6}{N}}{2} > \frac{0.4}{K} + \frac{0.60}{N}$$

$$\Rightarrow 0.2 + \frac{0.3}{N} - \frac{0.6}{N} > \frac{0.4}{K}$$

$$\Rightarrow 0.2 - \frac{0.3}{N} > \frac{0.4}{K}$$

$$\Rightarrow K > \frac{0.4}{0.2 - \frac{0.3}{N}}$$

4.3) Sequential time percent decreased four times.

→ Program takes half the time.

$$S = \frac{1}{(1-P) + (P/N)}$$

$\Rightarrow$ [illegible]

$$\Rightarrow \frac{1}{\frac{(1-P)}{4} + \frac{P}{N}} = \frac{1}{2}\left(\frac{1}{(1-P) + \frac{P}{N}}\right)$$

$$\Rightarrow 2\left((1-P) + \frac{P}{N}\right) = \frac{1-P}{4} + \frac{P}{N}$$

$$\Rightarrow 2 - 2P + 2\frac{P}{N} - \frac{P}{N} = \frac{1-P}{4}$$

$$\Rightarrow 4\left(2 - 2P + \frac{P}{N}\right) = 1 - P$$

$$\Rightarrow P = 1 - 8 + 8P - 4\frac{P}{N}$$

$$\Rightarrow P = 8P - 4\frac{P}{N} - 7$$

$$\Rightarrow 7P = 4\frac{P}{N} + 7$$

$$\Rightarrow P = \frac{4}{7}\frac{P}{N} + 1$$

$$\Rightarrow P - \frac{4}{7}\frac{P}{N} = 1$$

$$\Rightarrow P\left(1 - \frac{4}{7N}\right) = 1$$

$$\Rightarrow P = \frac{1}{1 - \frac{4}{7N}}$$

[1] https://en.wikipedia.org/wiki/Deadlock
[2]https://en.wikipedia.org/wiki/Banker%27s_algorithm