EE382V Multicore Computing: Assignment 1

Wenwen Zhang [wz3585], Chunheng Luo [cl38532]

September 11, 2016

Question 0

TACC User IDs:

- Wenwen Zhang: wenwen22
- Chunheng Luo: chluo

Question 1

Part (a)

- Assuming other parts of the program can be sped up by the factor of n, then the overall speedup is

$$Speedup = \frac{1}{0.4 + \frac{0.6}{n}}$$

- Assuming the method M accounts for x of the program's execution time on a single-core processor, M can be sped up by 2^3 and other parts of the program can be sped up by the factor of n, then the overall speedup is

$$Speedup = \frac{1}{\frac{x}{8} + \frac{(1-x)}{n}}$$

So in order to double the speedup, we require

$$\frac{1}{\frac{x}{8} + \frac{(1-x)}{n}} = 2 \times \frac{1}{0.4 + \frac{0.6}{n}}$$

which leads to

$$x = \frac{0.2n - 0.7}{0.125n - 1}$$

Therefore, M must account for $(0.2n - 0.7)/(0.125n - 1)$ of the total execution time **on a single-core processor** in order to double the overall speedup of the program.

Part (b)

Assuming the parts of the program that can be totally parallelized account for P of the total execution time on a single-core processor, and all of the other parts of the program, which accounts for $(1 - P)$ of the total execution time on a single-core processor, are not able to gain any speedup from the multicore architecture, then we have

$$S_2 = \frac{1}{(1-P) + \frac{P}{2}}$$

and

$$S_n = \frac{1}{(1-P) + \frac{P}{n}}$$

Solving the equations, we get

$$S_n = \frac{nS_2}{(2-n)S_2 + 2(n-1)}$$

Question 2

```
1 import java.util.Arrays;
2
3 class PetersonN implements Lock {
4     int N;
5     int[] gate;
6     int[] last;
7     public PetersonN(int numProc) {
8         N = numProc;
9         gate = new int[N];
10        Arrays.fill(gate, 0);
11        last = new int[N];
12        Arrays.fill(last, 0);
13    }
14    public void requestCS(int i) {
15      for (int k = 1; k < N; k++) {
16        gate[i] = k;
17        last[k] = i;
18        for (int j = 0; j < N; j++) {
19         while ((j != i) && (gate[j] >= k) && (last[k] == i))
20           no_op();// busy wait
21        }
22      }
23    }
24    public void releaseCS(int i) {
25        gate[i] = 0;
26    }
27 }
```

It is not possible to overtake a thread within a for loop iteration (within a level) since that would imply changing last[k], which immediately release the waiting thread from line 20.

However, between levels it is possible for some threads to overtake others an arbitrary number of times. In the situation described below, thread A is overtaken by thread B:
Suppose thread A has completed the first for iteration with i = 1 and is about to enter the next gate with i = 2, and thread B has just entered gate

1. If A is now paused by calling no_op(), thread B which has successfully completed level 1 may overtake A even if thread B is the later one to enter gate 1.

Question 3

In order make Filter Algorithm able to solve the l-exclusion problem, we can simply reduce the number gates from N to $(N - l)$.

```
1 const int  N;
2 int[N]     gate init 0;
3 int[N-l+1] last init 0;
4
5 /* For P_i */
6 request CS;
7 for (k = 1 : N - l) {
8   gate[i] = k;          // P_i is at gate k
9   last[k] = i;          // P_i updates last for that gate
10
11  int numAhead = l + 1;
12  while (numAhead > l && last[k] == i) {
13    for (j = 1 : N - 1) {
14          if (j != i && gate[j] >= k)
15            numAhead += 1;
16    }
17  }
18 }
19 CS;
20 release CS;
21 gate[i] = 0;
```

Question 4

* Code submitted on Canvas.

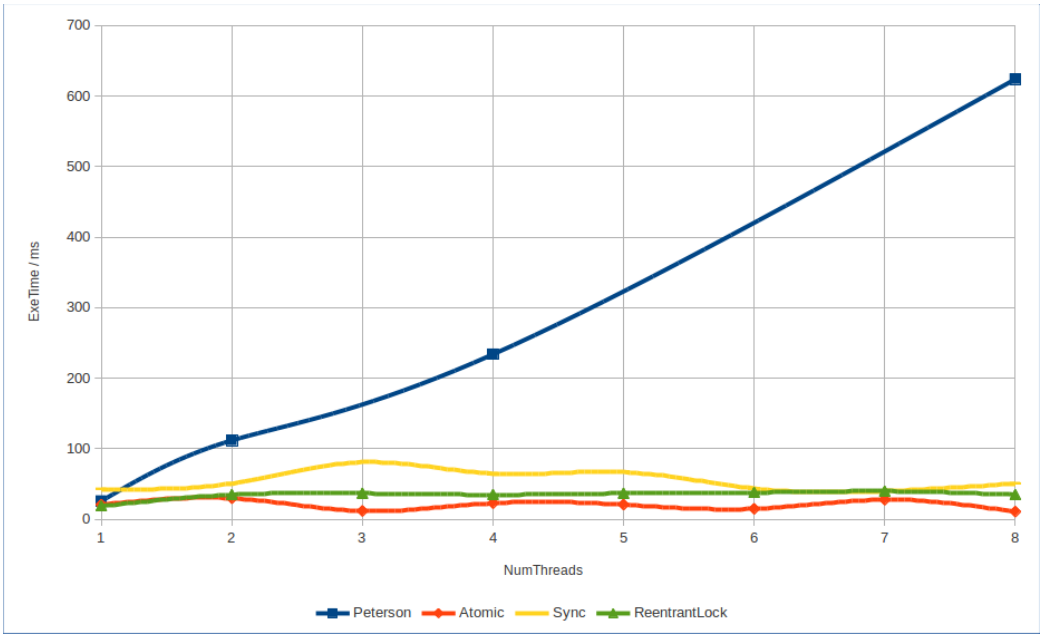

Figure 1: Execution time using the four methods

Question 5

* Code submitted on Canvas. The execution time plot is shown in Fig 1.