CSCI7551 Homework 4

Luke(Tuguluke Abulitibu)

October 14, 2014

1 Problem 1

1.1 (a)

See attachment(together with hand-written algorithm for different processor.)

1.2 (b)

since *produce* $A[i,j] := V$, then *consume* $A[i, me]$ *into* T, from the algorithm in part (a):

For process 0. The returned value of V is $\max(A[2,3], A[2,0])$

For process 1. The returned value of V is $\max(A[2,2], A[2,1])$

For process 2. The returned value of V is $\max(A[2,1], A[2,2])$

For process 3. The returned value of V is $\max(A[2,0], A[2,3])$

For process 4. The returned value of V is $\max(A[2,7], A[2,4])$

For process 5. The returned value of V is $\max(A[2,6], A[2,5])$

For process 6. The returned value of V is $\max(A[2,5], A[2,6])$

For process 7. The returned value of V is $\max(A[2,4], A[2,7])$

$A[i,j]$ denotes diffrent V inputs.

1.3 (c)

Since we have to 'wait' for other process to finish the job, this is the **barrier synchronization.**

2 Exercise 4.17

2.1 (b)

From Homework 2 Excercise 1 (In-class corrected version:) We can pretty much do the same process for

Program 1 MIMD pseudo code for tree summation

```
/* Tree summation of n elementd of a one dimensional array for n a power pf two */
private k;
shared S[N], N, j, k;
S = {x_0, x_1, ..., x_{N-1}}                                   /* Data */
j := log_2 N;                                                  /* Depth*/
K := N;                                                        /* Number of elements */
for i := 0 step 1 to j fork SUMMATION
    m := m/2;
    SUMMATION :
    for m := 0 step 1 to k                                     /* Executed by m processors */
        S[m] := S[m] + S[m + k], (0 ≤ m ≤ k);                  /* Loop */
        BARRIER
        join m;
```

MIMD pseudocode for histogram calculation, the KEY here is the critical section for $h[j] := h[j] + 1$.
For this pseudocode, we can only paralalize the calculation on j an keeping the critical on h apart, i.e. we synchronize $h[j]$.
So, we can not have all of the processes 'waiting' on one process to finish the critical section.

Program 2 MIMD pseudo code for histogram calculation

```
/* Histogram calculation, extending sequential pseudocode with fork/join */
private i, j;
shared x[1 : N], h[0 : M − 1], scale, M, x1, x0;
scale := M/(x1 − x0);
for i := 1 step 1 to N fork HISTOGRAM
    HISTOGRAM :
    j[i] := int(scale ∗ (x[i] − x0));
    critical unnamed;
    h[j[i]] := h[j[i]] + 1;
    end critical;                                              /* Synchronization */
    join N;
```

2.2 (c)

The answer is yes. From textbook page 121. Quote:"The most important synchronized access operations are p**roduce**, which waits for the variable to be empty, writes its value, and sets it full, and **consume**, which waits for the variable to be full, reads its value, and set to empty".
So, if we use produce/comsume on $h[j]$, we don't have to let all the processes to 'wait' on one process, but only let processes that had the same j wait on each other and the rest can continue, which will improve the parallelism.

3 Excercise 4.21

From textbook page 36 equation (2.20). we know speedup with p processors as

$$S_p = \frac{T_1}{T_p}$$

where T_p is the time to perform the computation using p processors or arithmatic units.

3.1 (a)

For single processor to run the program, it need to go through all line by it self, so

$$T_1 = t_0 + t_1 + N \times \frac{t_2}{N} = t_0 + t_1 + t_2.$$

For multiprocessors: Since the critical section(they are runing first) can only fit one at a time, so the time it need for critical is Pt_1. The schedule can be improved bu multithreads, for that we need $\lceil \frac{N}{P} \rceil \times \frac{t_2}{N}$, if N is devidable by P. We get:

$$T_p = t_0 + Pt_1 + \lceil \frac{N}{P} \rceil \frac{t_2}{N}$$

plug in $N = 1000$, $P = 50$, $t_0 = 1$, $t_1 = 4$, $t_2 = 495$. we can get

$$T_1 = 1 + 4 + 495 = 500$$

$$T_p = 1 + 50 \times 4 + \lceil \frac{1000}{5000} \rceil \frac{495}{1000} = 210.9$$

So that speed up is

$$S_p = \frac{T_1}{T_p} = \frac{500}{210.9} = 2.3708.$$

3.2 (b)

For this one, since the critical is right after the schedule, so any processes who finished calculating can go to critical, so that critical saves time.
For single processor to run the program, it need to go through all line by it self, so

$$T_1 = t_0 + N\frac{t_2}{N} + N\frac{t_1}{N} = t_0 + t_1 + t_2$$

For multiprocessors: just like part (a), to run the schedule, we need $\lceil \frac{N}{P} \rceil \times \frac{t_2}{N}$, if N is devidable by P.
While critical section can use the 'empty' space, that is, to save. without it the time would be just $\frac{t_1}{N} \times N$. But remember the 'empty' thread in the schedule, they can be calculated by

$$\lfloor \frac{\frac{t_2}{N}}{\frac{t_1}{N}} \rfloor = \lfloor \frac{t_2}{t_1} \rfloor$$

so overall, we save $N - \lfloor \frac{t_2}{t_1} \rfloor$ in the whole process, so that we can get the final result:

$$T_p = t_0 + \lceil \frac{N}{P} \rceil (\frac{t_2}{N}) + \frac{t_1}{N}(N - \lfloor \frac{t_2}{t_1} \rfloor)$$

Now, plug in $N = 1000$, $P = 50$, $t_0 = 1$, $t_1 = 4$, $t_2 = 495$. we can get

$$T_1 = 1 + 4 + 495 = 500$$

$$T_p = 1 + \lceil \frac{1000}{5000} \rceil \frac{495}{1000} + \frac{4}{1000}(1000 - \lfloor \frac{495}{4} \rfloor) = 14.408$$

So that speed up is

$$S_p = \frac{T_1}{T_p} = \frac{500}{14.408} = 34.703.$$

Which is faster than (a).