Parallel Computing: Problem Set 1 Document

Due on October 18th at 23:59

Mailbox: yuzhh1@shanghaitech.edu.cn
Student ID: 2020533156
Student Name: Zhenghong Yu

Contents

1 Problem 1

1.1 Question 1

Consider the problem of adding a list of n numbers. Assume that one person can add two numbers in time t_c. How long will one person take to add n numbers?
Solution: (n-1)t_c

1.2 Question 2

Now assume eight people are available for adding these n numbers and that the list has already been divided into eight equal parts. Each person can add two numbers in time t_c. Furthermore, a person can pass on the result of an addition (in the form of a single number) to the person sitting next to him or her in time t_w. How long will it take to add n numbers if

a) All eight people sat in a circle.
b) The eight people are sitting in two rows of four people each.
Solution:
a) $(\frac{n}{8}+2)t_c+4t_w$
b) $(\frac{n}{8}+2)t_c+3t_w$

1.3 Question 3

If you are free to seat the eight people in any configuration, how would you seat them as to minimize the time taken to add the list of numbers? Is there a best configuration independent of t_c and t_w?
Solution:
I'll let the eight people sit in a cube, which will take $(\frac{n}{8}+2)t_c+3t_w$. There is not a best configuration independent of t_c and t_w.

2 Problem 2

Compare the execution of the following code to calculate the greatest common divisor of x and y as performed on a SIMD versus an MIMD architecture.

```
gcd (x,y) {
while (x != y) {
     if (x > y) x = x-y;
     else y = y-x;
     }
}
```

Suppose two arrays $x=(x_0,\cdots,x_{n-1})$ and $y=(y_0,\cdots,y_{n-1})$ are each distributed across n processors with x_i and y_i on processor i. We wish to find $z=(z_0,\cdots,z_{n-1})$, where $z_i=gcd(x_i,y_i)$. Assuming that a compare or a subtract instruction each takes one time step, how many time steps would it take to calculate the result when $x=(52,24)$ and $y=(12,64)$ using
a) A two processor SIMD architecture.
b) A two processor MIMD architecture.
In each case, write a trace for the two processors, indicating clearly which instruction is executed at which time step.

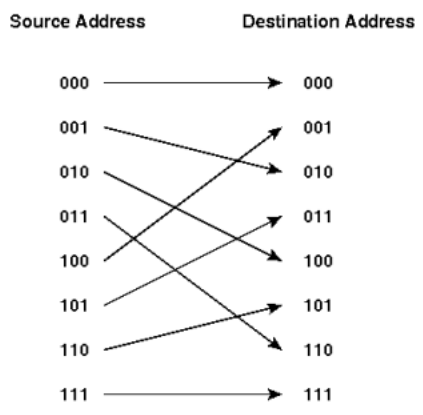
Solution:

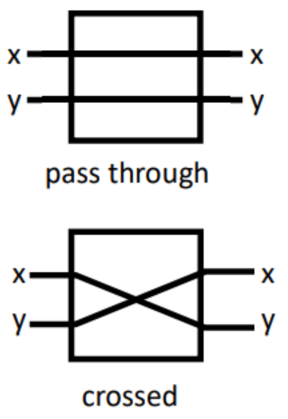
	SIMD		MIMD	
	Processor A	Processor B	Processor A	Processor B
step 1	x != y is true (52,12)	x != y is true (24,64)	x != y is true (52,12)	x != y is true (24,64)
step 2	x > y is true (52,12)	x > y is false (24,64)	x > y is true (52,12)	x > y is false (24,64)
step 3		y = y-x (24, 40)	x = x-y (40,12)	y = y-x (24, 40)
step 4	x = x-y (40,12)		x != y is true (40,12)	x != y is true (24,40)
step 5	x != y is true (40,12)	x != y is true (24,40)	x > y true (40,12)	x > y is false (24,40)
step 6	x > y true (40,12)	x > y is false (24,40)	x = x-y (28,12)	y = y-x (24, 16)
step 7	x = x-y (28,12)		x != y is true (28,12)	x != y is true (24,16)
step 8		y = y-x (24, 16)	x > y true (28,12)	x > y true (24,16)
step 9	x != y is true (28,12)	x != y is true (24,16)	x = x-y (16,12)	x = x-y (8,16)
step 10	x > y true (28,12)	x > y true (24,16)	x != y is true (16,12)	x != y is true (8,16)
step 11	x = x-y (16,12)	x = x-y (8,16)	x > y true (16,12)	x > y false (8,16)
step 12	x != y is true (16,12)	x != y is true (8,16)	x = x-y (4,12)	y = y-x (8,8)
step 13	x > y true (16,12)	x > y false (8,16)	x != y is true (4,12)	x != y is false (8,8)
step 14	x = x-y (4,12)		x > y false (4,12)	
step 15		y = y-x (8,8)	y = y-x (4,8)	
step 16	x != y is true (4,12)	x != y is false (8,8)	x != y is true (4,8)	
step 17	x > y false (4,12)		x > y false (4,8)	
step 18	y = y-x (4,8)		y = y-x (4,4)	
step 19	x != y is true (4,8)		x != y is false (4,4)	
step 20	x > y false (4,8)		End	
step 21	y = y-x (4,4)			
step 22	x != y is false (4,4)			
	End			

3 Problem 3

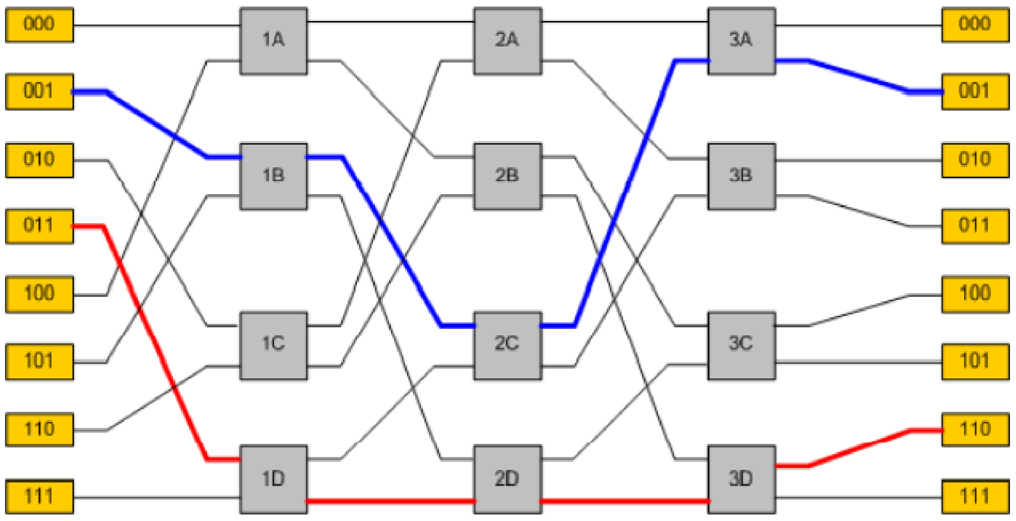
A *perfect shuffle* for a set of 2^n processors consists of connecting each processor i to another processor j whose ID in binary is a left rotation of i's ID in binary. A perfect shuffle on 8 processors is shown below. For example, the left rotation of 100 is 001 (the 1 bit wraps around to the right side), so there is an edge between 100 and 001.

A switch consists of two inputs x and y and two outputs. If the inputs are passed through the switch, then the outputs are also x and y. Otherwise, the inputs are crossed, and the outputs are y and x. A switch is shown below.





An *Omega network* for a set of 2^n processors is an indirect network consisting of n layers of perfect shuffles of the processors. Switches in the network can be set to either pass through or crossed in order to perform routing. For example, in the Omega network below, switches 1B, 2C and 3A are set to pass through to route from processor 001 to 001, while switch 2D is set to pass through and switches 1D and 3D are set to crossed to route from 011 to 110.



For this problem, devise an algorithm to route a message between two processors in an Omega network. Explain why your algorithm is correct.

Solution:

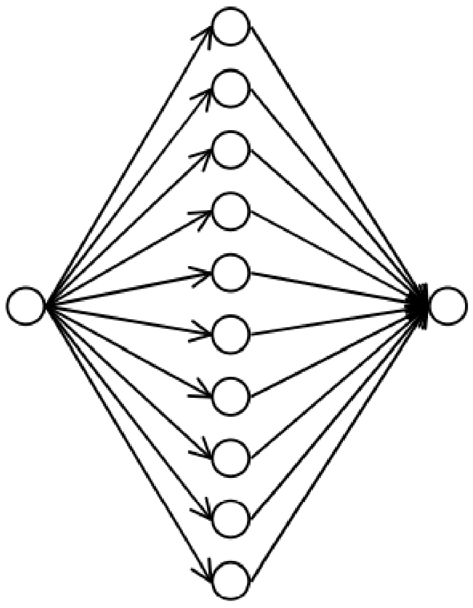
For each pair of processors:

- First determine the digits of the source processor which differ to to the destination processor.
- Then, for the ith layer of Omega network, set the switch as pass cross if the i digit of the source processor is differ to the destination processor, otherwise set it as pass though.

We can know that each layer help to change the corresponding bit of processor ID, the algorithm is like a XOR gate to find the correct destination.

4 Problem 4

The task graph shown in the figure below represents a parallel application. Each circle represents an indivisible task. There are 12 tasks: an initialization task, 10 computation tasks, and a finalization task. Each of the 12 tasks takes exactly one unit of time on one processor. The initialization task must complete before any of the computation tasks begin. Similarly, all 10 computation tasks must complete before the finalization task begins. The computation tasks can be executed in any order.



4.1 Question 1

What is the maximum speedup that can be achieved if this problem is solved on a parallel computer with 2 processors? Give a diagram showing how the tasks would be allocated to processors.

Solution:

Processor A	Processor B
Initialization Task	
Task 1	Task 6
Task 2	Task 7
Task 3	Task 8
Task 4	Task 9
Task 5	Task 10
Finalization Task	

The time speedup is $\frac{12}{7}$

4.2 Question 2

b) What is the maximum speedup that can be achieved if this problem is solved on any parallel computer?

Based on Amdahl's law, the maximum speedup is:

$$S_p = \frac{12}{2 + \frac{10}{p}}$$

Since the minimum value of $\frac{10}{p} = 1$ as one single task must finished by one processor, we can get

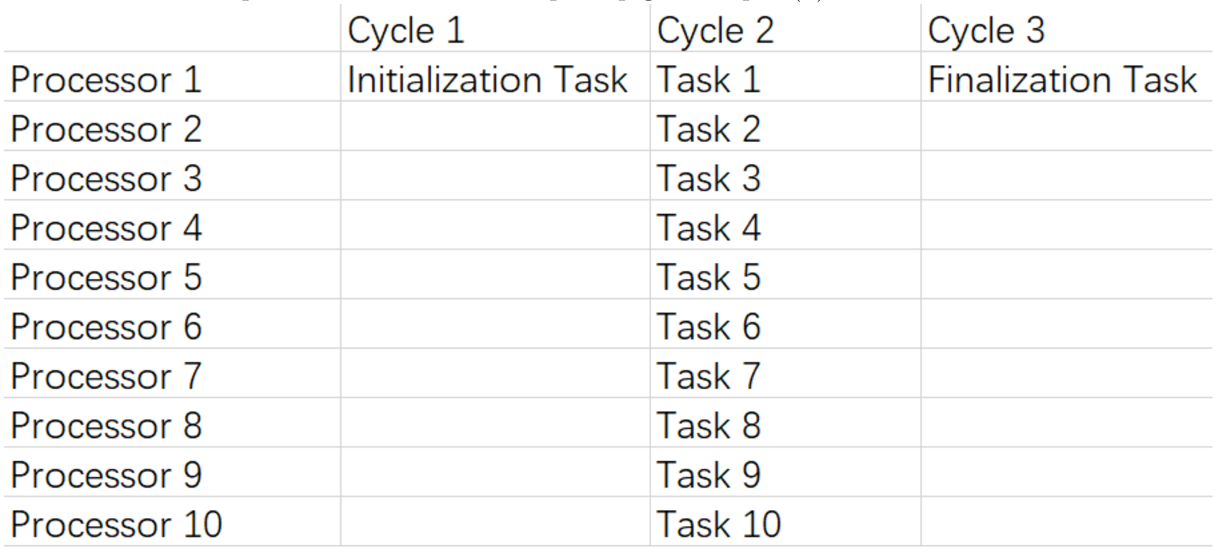
$$max\{S_p\} = \frac{12}{2+1} = 4$$

4.3 Question 3

What is the minimum number of processors required to achieve the speedup given in part (b)? Give a diagram showing how the tasks would be allocated to processors.

Solution:

We need at least 10 processors to achieve the speedup given in part(b)



	Cycle 1	Cycle 2	Cycle 3
Processor 1	Initialization Task	Task 1	Finalization Task
Processor 2		Task 2	
Processor 3		Task 3	
Processor 4		Task 4	
Processor 5		Task 5	
Processor 6		Task 6	
Processor 7		Task 7	
Processor 8		Task 8	
Processor 9		Task 9	
Processor 10		Task 10	

5 Problem 5

Three students taking the Parallel Computing course have written parallel programs for their lab assignments.

5.1 Question 1

Amanda's parallel program achieves a speedup of 9 on 10 processors. What is the maximum fraction f of the computation that can be inherently sequential in her program?

Solution:

Based on Amdahl's law, we have

$$S_p = \frac{1}{f + (1-f)/p}$$

where $p = 10$, $S_p = 9$, we can get $f = \frac{1}{81}$. So the maximum fraction f of the computation that can be inherently sequential in her program is $\frac{1}{81}$.

5.2 Question 2

Brian's parallel program executes in 225 seconds on 16 processors. By timing parts of his program, he determines that 9 seconds is spent performing initialization and finalization on one processor, and for the remaining 216 seconds all 16 processors are executing. What is the *scaled speedup* achieved by Brian's program?

Solution:

$$S_p = \frac{fT_1 + p(1-f)T_1}{T_1} = 15.4$$

5.3 Question 3

Cindy times her parallel program on 10 processors and finds that for 270 seconds, all processors are active, and for 30 seconds, one processor is executing inherently sequential code. Assuming that the size of the sequential code does not increase as the problem size increases, what is the scaled speedup she can expect for p processors, where p = 20, 30, 40?

Solution:

$$S_p = \frac{30 + 270 \cdot p}{30 + 270} = \frac{1 + 9p}{10}$$

Thus $S_p(20) = 18.1$, $S_p(30) = 27.1$, $S_9(40) = 36.1$.

6 Problem 6

6.1 Question 1

Suppose we have a problem for which there is a sequential algorithm running in time n on an input of size n. Now, consider a parallel algorithm for the problem which has parallel running time $n/p + 2logp$ when using p processors. Is it possible to maintain isoefficiency for this parallel algorithm? If so, give the necessary relationship between n and p.

Solution:

Sequential running time $t_s = n$

Parallel running time $t_p = \frac{n}{p} + 2\log p$

Parallel Overhead $\gamma(n,p) =$ parallelwork - sequential work

$= p * t_p$ - t_s

$= n + 2p\log p - n$

$= 2p\log p$

Isoefficiency when sequential work $= c * \gamma(n,p)$

So need $n = \Omega(p\log p)$

6.2 Question 2

Suppose now the sequential running time is n^2 and the parallel running time is $n^2/p + n^3/\sqrt{p}$. Can this algorithm algorithm maintain isoefficiency, and if so, what is the necessary relationship between n and p?

Solution:

Sequential running time $t_s = n^2$

Parallel running time $t_p = \frac{n^2}{p} + \frac{n^3}{\sqrt{p}}$

Parallel Overhead $\gamma(n,p) =$ parallelwork - sequential work

$= p * t_p$ - t_s

$= n^3\sqrt{p}$
Isoefficiency when sequential work $= c * \gamma(n, p)$
So need $n = \Omega(n^3\sqrt{p})$, which is impossible
So can't maintian isoefficiency