FACULTY OF SCIENCE

ACADEMY OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

MODULE	IT18X97/IT00197 PARALLEL PROGRAMMING
CAMPUS	AUCKLAND PARK CAMPUS (APK)
EXAM	EXAMINATION SSA OCTOBER 2021 **MEMO**

DATE: 28 October 2021	**SESSION:** 08:30 - 10:30
ASSESSOR(S):	MR R. MALULEKA
EXTERNAL MODERATOR:	DR A. EZUGWU (UKZN)
DURATION: 120 MINUTES	**MARKS:** 100

Please read the following instructions carefully:

1. An **additional 30 minutes** submission time will be allowed.
2. Answer all questions.
3. Answer each question in its entirety before moving on to the next.
4. **Submit** your answers in a **single PDF document**.
5. This paper consists of **8** pages, excluding the cover page.

QUESTION 1

Suppose you have joined Habitat for Humanity as a volunteer to help build house for families in need. You and the other volunteers will be separated into teams, which will be assigned to different tasks.

(a) Identify at least four (4) tasks involved in the construction of a house. (2)

Solution:
2 marks for identifying realistic tasks √√

(b) Which tasks from (a) exhibit task parallelism? (2)

Solution:
2 marks for identifying task-parallelism, i.e. tasks that can be done simultaneously, e.g. installing plumbing piping and electrical cabling. √√

(c) Which tasks from (a) exhibit data-dependency? (2)

Solution:
2 marks for identifying dependent tasks, e.g. pouring the foundation and putting up walls. √√

(d) Once construction is finished, the house will need to be painted. How can we use data-parallelism to partition the work of paining the house? (2)

Solution:
Assign different workers to paint different parts of the house at the same time. √√

(e) What are the three sources of overhead in parallel programs? (3)

Solution:
Inter-process Interaction, Idling, Excess Computations √√√

(f) How could the sources of overhead identified in (e) present themselves in the construction of a house? (6)

Solution:

- Inter-process Interaction: teams needing to communicate/share ideas/share items to achieve tasks. √√
- Idling: a team needing to wait for another team before it can start working, e.g. those installing windows having to wait for those putting up the walls. √√

- Excess Computations: any additional work involved in forming/-managing teams and assigning them tasks, e.g. separating volunteers into groups. $\surd\surd$

(g) Give the following metrics for ring network topology (aka linear array with wraparound link) with n nodes: (3)

i. diameter
ii. bisection width
iii. number of links

Solution:

i. floor(n/2), n/2 also acceptable $\surd$
ii. 2 $\surd$
iii. n $\surd$

Total: 20

QUESTION 2

(a) You have been hired by a company to acquire a multi-processor machine for their resource intensive application. You determine that 90% of the tasks in the application can run in parallel, and that the work can be uniformly divided. You also determine that application is able to handle an increasing amount of work as more cores are added. You can purchase a 4, 8, or 16 core machine. Which machine would give you the best value for money? Motivate your answer. (9)

Solution:
Since the problem size can grow as additional cores are added we should use Gustafson-Barsis's law to calculate the scaled speedup. We use the scaled speedup to determine the efficiency as a measure of where we are getting the most benefit. Scaled Speedup is given by: $\psi \leq np + (1 - np)s$.
Efficiency is given by: $\epsilon = \psi/np$

- 2 marks per calculation for each value of np. (6)
- 3 marks for choice with valid reasoning. (3)

(b) If an application running on a single processor has a fixed size problem, and spends 10% of its runtime on serial work and 90% on parallelizable work, what speedups can you expect to see on: (8)

(i) 6 processors?
(ii) an unlimited amount of processors?

Solution:
Since the problem size is fixed we use Amdalh's law:

$$S \leq \frac{1}{s + \frac{(1-s)}{np}}$$

- 4 marks for calculation on 6 processors (2 marks for correct function choice, 2 marks for correct calculation)
- Calculate speedup as p tends to infinity (4 marks)

(c) Suppose that MPI COMM WORLD consists of the three processes 0, 1 and 2, and suppose the following code is executed: (3)

```
int x, y, z;
switch(my_rank) {
case 0:
   x=1; y=2; z=3;
   MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Send(&y, 1, MPI_INT, 2, 12, MPI_COMM_WORLD);
   MPI_Bcast(&z, 1, MPI_INT, 1, MPI_COMM_WORLD);
   break;
case 1:
   x=4; y=5; z=6;
   MPI_Bcast(&x, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&y, 1, MPI_INT, 1, MPI_COMM_WORLD);
   break;
case 2:
   x=7; y=8; z=9;
   MPI_Bcast(&z, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Recv(&x, 1, MPI_INT, 0, 12, MPI_COMM_WORLD, &status);
   MPI_Bcast(&y, 1, MPI_INT, 1, MPI_COMM_WORLD);
   break;
}
```

What are the values of x, y, and z on each process after the code has been executed?

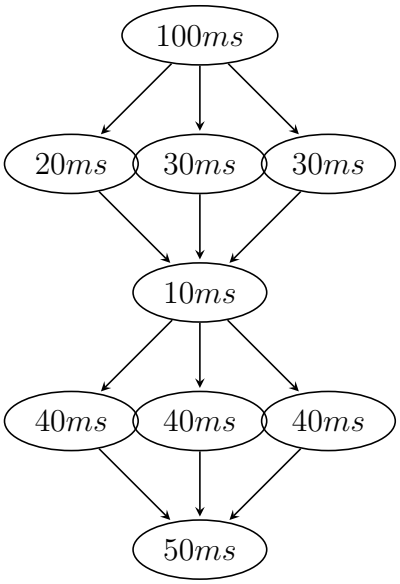
Solution:

	x	y	z
p0	1	2	5
p1	1	5	6
p2	2	5	1

Total: 20

QUESTION 3

Given the following task graph for a program running on a multiprocessor machine.



(a) Assuming a single (1) worker thread what is the runtime of this program? (2)

Solution:
Add all work: 360 $\surd\surd$

(b) What is the speedup when two (2) threads are used? (4)

Solution:
Serial Sections: $160ms$
Parallel Section $P_{1:::3}$: $50ms$
Parallel Section $P_{5:::7}$: $80ms$
Total: $290ms$
Speedup = sequential execution time/parallel execution time
Speedup $= 360/290 = 1.241$ $\surd\surd\surd\surd$

(c) What is the maximum degree of concurrency of the graph? (1)

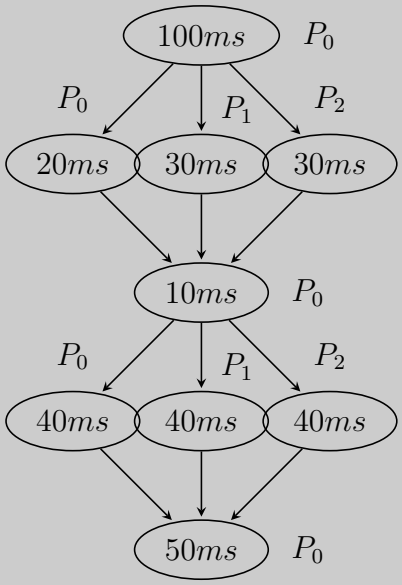
Solution:
3 $\surd$

(d) What is the average degree of concurrency of the graph? (3)

Solution:
avg. concurrency = total work / length of critical path
$= 360/230 = 1.5652$ $\surd\surd\surd$

(e) Show an efficient mapping of the tasks onto 3 processes. (5)

Solution:



(f) Discuss what data dependency is, how it may hinder parallelism, and how it can be resolved. (4)

Solution:

- Defn: The results of an instruction may be required for subsequent instructions. $\surd$
- Hinders parallelism as subsequent instructions cannot be executed until the after initial one. $\surd$
- Solution: Instructions may be reordered to execute independent instructions simultaneously. $\surd$

  May be done automatically at runtime if hardware supports this, or explicitly by the programmer. $\surd$

(g) The number of tasks into which a problem is decomposed determines its granularity. Discuss granularity of task decompositions, as well its effect on the performance of parallel programs. Use the example of dense matrix-vector multiplication to support your discussion. (6)

Solution:
max 6

- Decomposition into a large number of tasks results in fine-grained decomposition; (1)
- and that into a small number of tasks results in a coarse grained decomposition. (1)

- Often, using fewer processors improves performance of parallel systems. (1)
- Using fewer than the maximum possible number of processing elements to execute a parallel algorithm is called *scaling down* a parallel system. (1)
- A naive way of scaling down is to think of each processor in the original case as a virtual processor and to assign virtual processors equally to scaled down processors. (1)
- Since the number of processing elements decreases by a factor of n/p ,the computation at each processing element increases by a factor of n/p. (1)
- The communication cost should not increase by this factor since some of the virtual processors assigned to a physical processors might talk to each other. This is the basic reason for the improvement from building. (1)
- Use of dense matrix-vector multiplication example (3)

Total: 25

QUESTION 4

Rather than simply finding the sum of n values,

$$x_0 + x_1 + \ldots + x_{n-1},$$

prefix sums are the n partial sums

$$x_0, x_0 + x_1, \ldots, x_0 + x_1 + \ldots + x_{n-1}$$

(a) Devise a serial algorithm for computing the n prefix sums of an array with n elements. (4)

Solution:

```
1 prefix_sums[0] = x[0]              [1 mark]
2 for i in range(1,len(x)):            [1 mark]
3   prefix_sums[i] = prefix_sums[i-1] + x[i]  [2 marks]
```

(b) Parallelize your serial algorithm for a system with n processes, each of which stores one of the $x_i s$. Use only point-to-point communication. Assume that the array is referenced by a variable x at process zero and begin by having p_0 send the other processing their respective values. Each process should store its respective x_i in a variable x_i, and end by storing its corresponding prefix sum in a variable $prefix\_x\_i$. I.e. (9)

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Getrank()

x_i = ...
.
.
.
sys.stdout.write("Process %d calculated prefix_x_i = %d.\n" %
    (rank, prefix_x_i))
```

Solution:

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
size = comm.Get_size()
rank = comm.Getrank()

# array x already at p_0

if rank  = 0:
  x_i = x(0)
  for i in range(1,size):
    comm.Send(x(i), dest = i)          (1 mark)
else:
  comm.Recv(x_i, source = 0)             (1 mark)

if rank != 0:                        (1 mark)
  comm.Recv(preceding_sum, source = rank-1) (2 marks)
   prefix_x_i = preceding_sum + x_i          (1 marks)

if rank size-1:                        (1 mark)
  comm.Send(prefix_x_i, dest = rank+1)     (2 marks)

sys.stdout.write("Process %d calculated prefix_x_i = %d.\n" %
    (rank, prefix_x_i))
```

(c) Suppose that we are working with a communicator of size 4 and that X is a 4×4 matrix. (12)

$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix}$$

(i) How would the components of x be distributed among the processes in a program that used a block distribution?

(ii) How would the components of x be distributed among the processes in a program that used a cyclic distribution?

(iii) How would the components of x be distributed among the processes in a program that used a block-cyclic distribution with block-size $b = 2 \times 2$?

Solution:

(i) Process 0 : x_0, x_1, x_2, x_3 $\surd$
Process 1 : x_4, x_5, x_6, x_7 $\surd$
Process 2 : x_8, x_9, x_{10}, x_{11} $\surd$
Process 3 : $x_{12}, x_{13}, x_{14}, x_{15}$ $\surd$

(ii) Process 0 : x_0, x_4, x_8, x_{12} $\surd$
Process 1 : x_1, x_5, x_9, x_{13} $\surd$
Process 2 : x_2, x_6, x_{10}, x_{14} $\surd$
Process 3 : x_3, x_7, x_{11}, x_{15} $\surd$

(iii) Process 0 : x_0, x_1, x_4, x_5 $\surd$
Process 1 : x_2, x_3, x_6, x_7 $\surd$
Process 2 : x_8, x_9, x_{12}, x_{13} $\surd$
Process 3 : $x_{10}, x_{11}, x_{14}, x_{15}$ $\surd$

Total: 25

QUESTION 5

(a) Briefly reflect on your experiences working on a research project for this module, drawing on the theory you have learnt. Include the following in your discussion: (10)

- A brief background of your research topic;
- The results of your research;
- The parallel algorithm design model you used.

Solution:

- content (6)
- drawing on theory from module (2)
- structure (2)

Total: 10

— End of paper —