Anjuman-i-islam's

M.H. Saboo Siddik College of Engineering

8, Saboo Siddik Polytechnic Road, Byculla, Mumbai 400 008.

Department of Computer Engineering

(Affiliated to University of Mumbai)

Seat No.	7281020		
Subject Name	High Performance Computing.	**Date of Exam**	08/06/2021
Subject Code	52753	**Semester**	VIII
Invigilator Signature		**Total No. of Pages**	08
		Student Signature	

Q2 Solve any Two questions out of three.

A. Explain different mapping techniques that are used in load balancing.

→ Mapping Techniques are used to assign task to processes.
- They are used to solve major sources of overhead.
- load imbalance, IPC
- The techniques for load balancing are broadly divided into mapping 2 types.

(i) Static Mapping.

- Before executing algorithm, the tasks are distributed among processes.
- If task sizes are unknown a static mapping can lead to serious load imbalance.
- Static mapping is applicable for tasks that are generated statically and advance uniform computational requirements are known in advance.

(ii) Dynamic Mapping:

- Also referred to as dynamic load balancing.
- Tasks are distributed at runtime.

Seat No.	7281020	Semester	VIII
Subject name & Code	HPC - 52753	Student Signature	

- Applicable for tasks that are.
  a. generated dynamically.
  b. have non-uniform computational requirements.

- Two different classes of dynamic mapping.

(i) Centralized dynamic Mapping.

- All executable tasks are maintained in a central common datastructure.
- They are maintained by special processes.
  - Master: Manage the group of available tasks.
  - Slave: Depend on master to obtain work.

- When a slave process has no task, it take a portion of available work from master; and a new task is generated, it is added to the pool of tasks in the master process.
- Main issue is when many processes are used, the master becomes bottleneck.
- Soln: Chunk scheduling, where when a process runs out of work it gets tasks (group) from master. In other words chunk of work will be assigned at once.

(ii) Distributed Dynamic mapping

- Tasks are distributed among processes.
- They exchange task while runtime to distribute load.
- Each process can send or receive work from other processes.
- Main issue is now sending & recieving processes are paired together and who will initiate the work transfer.

Seat No.	7281020	Semester	VIII
Subject name & Code	HPC-52753	Student Signature	

B. Discuss in detail pipeline hazards with its type

→ Pipeline Hazards occur when instructions read or write registers that are used by other instructions.

- The type of conflicts that arise are divided into 3 main categories.

(i) Structural Hazards (Resource conflicts).

- These hazards are caused by access to memory by two instructions at a same time. These conflicts can be solved by using seperate instructions in the pipeline
- It occurs when the processor is not capable of executing all instructions simultaneously.
- Not so prevalent in modern processors, as the instruction set architecuture is designed to support pipelining.

(ii) Data Hazards (Data Dependency)

- It occurs when the current instructions depends on the results of the previous instructions, but the result is not yet available.
- Divided in 4 Categories:

(a) RAR Hazard.

- Occurs when 2 instructions read from same register.
- Doesnot cause a major problem as reading doesnot means changing value in register.
- Therefore, two instructions that have RAR Hazard can execute in successive cycles.

ADD r1 r2 r3 ← Both instructions read
SUB r4 r5 r3 ← r3. creating RAR

Seat No.	7281020	Semester	VIII
Subject name & Code	HPC-52753	Student Signature	

b. RAW Hazard.

- Occurs when an instruction reads register that was written by previous instruction.
- Also called data dependencies or true dependencies.

eg: ADD r1 r2 r3 ← Subtract reads the o/p of ADD
SUB r4 r5 r1 ← creating RAW Hazard.

c. WAR & WAW.

- Also called as Name dependencies.
- These occurs when a o/p of register of an instruction has been read or written by previous instructions.
- If processor executes instructions in the sequence they occur and uses the same pipeline for all instructions WAR & WAW hazards donot cause any problem in execution processes.

eg:

ADD r1 r2 r3 ←
SUB r2 r3 r6 ← WAR Hazard.

ADD r1 r2 r3
SUB r1 r5 r6
↑ WAW Hazard.

(iii) Branch Hazards.

- Branch instructions, particularly conditional branch inst^n create data dependencies b/w branch inst^n & previous inst^n fetch stage of the pipeline.

- Since branch computes address of next instructions. that the instruction should be fetched from.
- It consumes time & also time is required to flush the pipeline & fetch instruction from the calculated target address.
- A lot of time is wasted in it & is called as Branch penalty.

Q3 Solve any ~~two~~ out of three.

B. State & explain the performance metrics.

a. Speedup.

→ - The speed increase because of the parallel system. compared to the uniprocessor system is called as speedup.
- It is the ratio of the speed of parallel system to that of sequential system.
- It can also be given as the ratio of the time taken to execute a program on sequential system to that on a parallel system.
- Represented as $S(n)$ & is given as.

$$S(n) = \frac{T(1)}{T(n)}.$$

b. Efficiency.

→ - Efficiency in parallel system is the ratio of the actual speed-up obtained by a system to the ideal speedup that should be achieved according to the no of processors used in a parallel system.
- Ideal time required to execute a program using n processor should be $T(1)/n$

Thus, efficiency (η) can be given as.

$$\eta \text{ or } E(n) = \frac{\text{Actual Speed-up}}{\text{Ideal Speed-up}} = \frac{T(1)}{n\,T(n)}.$$

where,

actual speedup = $1/T(n)$ & Ideal speedup = $n/T(1)$.

c. Throughput

→

- Throughput of a system is defined as the number of programs executed per unit time. This is represented as Ws and is given as:

$$Ws = \frac{\text{Number of programs}}{\text{Time in seconds.}}$$

d. Scalability.

- A parallel system is said to be scalable if the same efficiency is obtained by increasing the number of processor.
- Since efficiency is dependent upon the number of processors & it keeps on decreasing as we increase the no of processor.
- We have $\eta = \frac{T(1)}{n(T(n))}$

if we increase n then η will decrease, so to have a stable or same efficiency $T(n)$ should be reduced in the same proportion as that of increase in 'n'.

Seat No.	7281020	Semester	VIII
Subject name & Code	HPC-52753	Student Signature	

A. Write a parallel MPI program to broadcast a data from a root process to 4 other processes.

→

The following program is written in python programming language using the native MPI library called mpi4py

Code:

```
from mpi4py import MPI.

comm~~unicator~~ = MPI.comm_world.
#using comm_world as communicator.
rank = comm.Get_rank()
# To get rank.
size = comm.Get_size()
# To get size.

data = 0.

def print_message(data__, rank)
        print(f' Data : {data_} Recieved at
                process id: {rank_}')

if rank != 0: # checking for slave process.
    data = comm.recv(source = rank-1, tag = 11)
    print_message(data, rank)
    data += 1
comm.send(data, dest = (rank+1)%size, tag = 11)
```

Seat No.	7281020	Semester	VIII
Subject name & Code	HPC-52753	Student Signature	

```
if rank == 0 : # Master process.
    data = comm.recv (source = size -1, tag = 11)
    if data != 0 :
        print_message(data, rank).
    comm.send ( data, dest = (rank+1) % size, tag=11)
```

Alternatively, we can also use MPI_Bcast ~~command~~ as primitive as follows:

```
from mpi4py import MPI

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

if rank == 0:
    data = [i for i in range(size)]
else :
    data = None.

data = comm.bcast (data, root=0)
print(f' Data : {data} on the process {rank} ').
```