DEPARTMENT OF COMPUTER SCIENCE ENGINEERING (DATA SCIENCE & AIML)

UNIT TEST-I SOLUTION

Class: TE **Semester: VI** **Subject: CSDLO6011 High Performance Computing**

Date: 22-02-2024 **Time: 2:00 TO 3:30 PM** **Max marks: 40**

Note the following instructions

1. **Attempt all questions.**
2. **Draw neat diagrams wherever necessary.**
3. **Write everything in Black ink (no pencil) only.**
4. **Assume data, if missing, with justification.**

Q. N.	Questions
Q.1.	**Attempt any two.**
1 **Solution:**	Write a short note on Demand Driven Computation. □ **Demand-driven Computers/ Reduction Machines/ Lazy Computers** □ In **demand driven machines**, if there is a *demand for the result then only the computation triggers.* □ **Data-driven machines** takes a **bottom-up approach** as it select instructions for execution based on the availability of their operands. □ **Demand-driven machines** takes a **top-down approach**, attempting to execute the instruction (a **demander**) that yields the final result. This triggers the execution of instructions that yield its operands, and so forth. □ The demand-driven approach matches naturally with functional programming languages (e.g. LISP and SCHEME).

- A ***demand-driven machines*** corresponds to **lazy machines** because operation are executed only when their result are required by another instruction. While ***data-driven machines*** are called **eager machines** because operation are carried out immediately after all their operands becomes available.
- **Reduction machines (*demand-driven machines*)**
- ***Advantages:***
  - **High parallelism potential,**
  - Easy manipulation of data structures, and
  - Only execute required instructions.
- **Disadvantage:**
  - They do not share objects with changing local state, and
  - Do require time to propagate demand tokens.
- Example Consider the following example of a arithmetic expression :

  a=[(b+1) * c – (d÷e)]
- The ***data-driven computation*** chooses a **bottom-up approach** , starting from **b+1** and **d ÷ e** then proceeding to the '*' operation finally to the outermost operation '-'.
- A ***demand-driven computation*** chooses a **top- down approach** by *first demanding the value of* **'a'** ,which triggers the demand for evaluating the next level expression **(b+1)*c** and **d ÷ e** ,which in turns triggers the demand for evaluating **b+1** at the inner most level. The results are returned to the nested ***demander*** in the reverse order before **'a'** is evaluated.

2

Solution:

Describe the fundamental limits of serial computing.

Power Wall (The Computational Power Argument – from Transistors to FLOPS):

Increasingly, microprocessor performance is limited by achievable power dissipation rather than by the number of available integrated-circuit resources (transistors and wires).

Thus, the only way to significantly increase the performance of microprocessors is to improve power efficiency at about the same rate as the performance increase.

Frequency Wall (The Data Communication Argument):

Conventional processors require increasingly deeper instruction pipelines to achieve higher operating frequencies.
This technique has reached a point of diminishing returns, and even negative returns if power is taken into account.

Memory Wall (The Memory/Disk Speed Argument):

On multi-gigahertz symmetric processors --- even those with integrated memory controllers --- latency to DRAM memory is currently approaching 1,000 cycles.

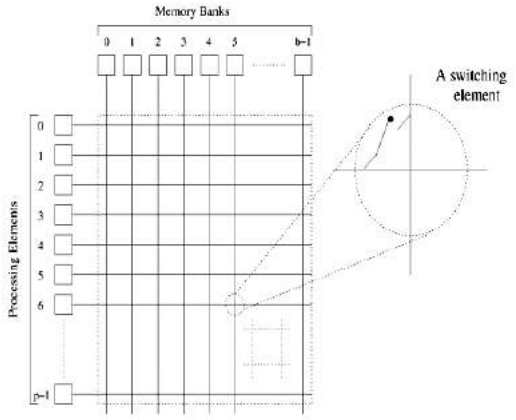
	As a result, program performance is dominated by the activity of moving data between main storage (the effective-address space that includes main memory) and the processor.
3 **Solution:**	Explain SPMD Model. **Single Program Multiple Data (SPMD)** is a special case of the Multiple Instruction Multiple Data model (MIMD) of Flynn's classification. In the SPMD model, a single program is executed simultaneously on multiple data elements. Here, each processing element (PEs) runs the same program but with different data elements, allowing for parallel processing and increased efficiency. In the SPMD model, the process id is used for branching. Each instance of the program works on its own data and may follow different conditional branches or execute loops differently. **Execution Process** In the SPMD model, the program is written once but replicated many times, with each PE executing the same program but with different data elements. The program is divided into tasks, which are assigned to different PEs for processing. The PEs operate independently of each other and can execute conditional branches or loops differently depending on their assigned data elements. **Synchronization Methods** Synchronization is a crucial aspect of the SPMD model to ensure that all PEs complete their assigned tasks before moving to the next phase of the program. One common method of synchronization in SPMD is Barrier Primitives, which allows multiple threads or processes to wait for each other to reach a specific point in the program before continuing to execute. Barrier Primitives make each PE wait at its primitive until all other PEs have completed their tasks. **Use Cases** The SPMD model is used extensively in high-performance computing (HPC) applications that require massive amounts of data processing. Some of the most common use cases of the SPMD model include weather forecasting, scientific simulations, and financial modeling. The SPMD model can significantly improve processing speed and efficiency, making it an ideal choice for large-scale computing tasks.
4	Compare and Contrast Data and Task Level Parallelism.

Solution:

Data Parallelisms	Task Parallelisms
1. Same task are performed on different subsets of same data.	1. Different task are performed on the same or different data.
2. Synchronous computation is performed.	2. Asynchronous computation is performed.
3. As there is only one execution thread operating on all sets of data, so the speedup is more.	3. As each processor will execute a different thread or process on the same or different set of data, so speedup is less.
4. Amount of parallelization is proportional to the input size.	4. Amount of parallelization is proportional to the number of independent tasks is performed.
5. It is designed for optimum load balance on multiprocessor system.	5. Here, load balancing depends upon on the e availability of the hardware and scheduling algorithms like static and dynamic scheduling.

Q.2. **Attempt any two**

1



1. **Identify** the above network topology in parallel platforms and explain it in detail.
2. **Make use of** 8 processors and 8 memory banks to calculate the following:
    a. Diameter
    a. Bisection Width
    a. Arc Connectivity
    a. Cost (No. of links)

Solution:

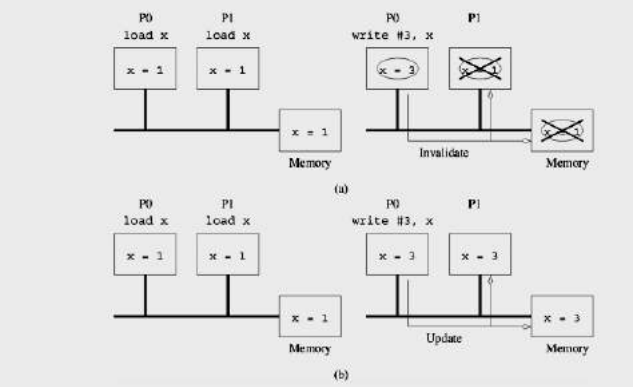
The above network topology in diagram is **Crossbars.**

The no. of processors p = 8 and Memory banks = 8
Diameter = 1
Bisection Width = p = 8
Arc Connectivity = 1
Cost (No. of links) = $p^2 = 64$

2 **Explain** the Invalidate Protocol for Cache Coherence in MultiProcessor Systems. **Develop** a state machine diagram for invalidate protocol of cache coherence.
Consider the following initial state of two processors P0 and P1.

If there is a write operation performed on P0 with a new value x=3. **Apply** Invalidate Protocol to construct the parallel program execution flow with the simple three-state coherence protocol (Shared, Invalid and Dirty).

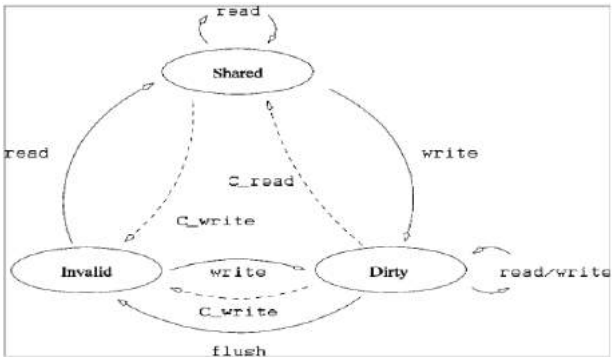
Solution:



Maintaining Coherence Using Invalidate Protocols

- Initially the variable x resides in the global memory.
- The first step executed by both processors is a load operation on this variable.At this point, the state of the variable is said to be shared, since it is shared by multiple processors.
- When processor P0 executes a store on this variable, it marks all other copies of this variable as invalid. It must also mark its own copy as modified or dirty.
- This is done to ensure that all subsequent accesses to this variable at other processors will be serviced by processor P0 and not from the memory.
- At this point, say, processor P1 executes another load operation on x .
- Processor P1 attempts to fetch this variable and, since the variable was marked dirty by processor P0, processor P0 services the request.
- Copies of this variable at processor P1 and the global memory are updated and the variable re-enters the shared state.

State machine diagram for invalidate protocol of cache coherence



Parallel Program Execution Flow

Time	Instruction at Processor 0	Instruction at Processor 1	Variables and their states at Processor 0	Variables and their states at Processor 1	Variables and their states in Global mem.
					x = 5, D y = 12, D
	read x		x = 5, S		x = 5, S
		read y		y = 12, S	y = 12, S
	x = x + 1		x = 6, D		x = 5, I
		y = y + 1		y = 13, D	y = 12, I
	read y		y = 13, S	y = 13, S	y = 13, S
		read x	x = 6, S	x = 6, S	x = 6, S
	x = x + y		x = 19, D	x = 6, I	x = 6, I
	x = x + 1		x = 20, D		x = 6, I

3 **Solution:**	**Explain** Communication Costs in Parallel machines with respect to Store and Forward Routing in detail. **Calculate** the communication cost using Store and Forward Routing mechanism for the following scenario: We need to send 20 words from Processor P0 to P7, the path has 14 communication links, the startup time required for the communication is 10s, per-hop time is 12s and time required for transfer of each word is 7s. **Store-and-Forward Routing** A message traversing multiple hops is completely received at an intermediate hop before being forwarded to the next hop. The total communication cost for a message of size m words to traverse l communication links is $t_{comm} = t_s + (mt_w + t_h)l.$ In most platforms, th is small and the above expression can be approximated by $t_{comm} = t_s + mlt_w.$ **As per given data in question,** m = 20 l = 14 t_s = 10s t_h = 12 t_w = 7 Communication Cost = 10 + (20* 7 + 12) * 14 = 2138
Q.3.	**Attempt any one.**

1

Consider the execution of the query:
MODEL = "CIVIC" AND YEAR = 2001 AND (COLOR = "GREEN" OR COLOR = "WHITE") on the following database:

ID#	Model	Year	Color	Dealer	Price
4523	Civic	2002	Blue	MN	$18,000
3476	Corolla	1999	White	IL	$15,000
7623	Camry	2001	Green	NY	$21,000
9834	Prius	2001	Green	CA	$18,000
6734	Civic	2001	White	OR	$17,000
5342	Altima	2001	Green	FL	$19,000
3845	Maxima	2001	Blue	NY	$22,000
8354	Accord	2000	Green	VT	$18,000
4395	Civic	2001	Red	CA	$17,000
7352	Civic	2002	Red	WA	$18,000

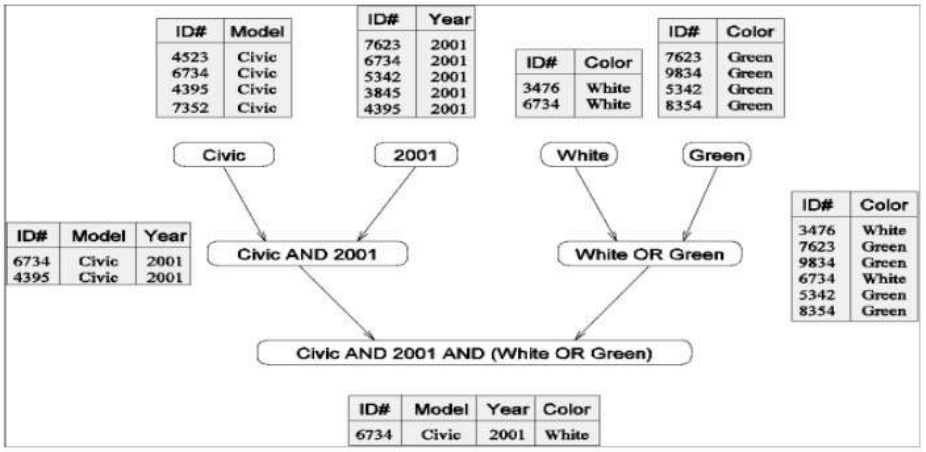
For the parallel execution of the above query perform the following:

a) **Decompose** the problem in multiple tasks
b) **Build** a task dependency graph for the tasks identified
c) **Select** appropriate mapping of tasks to different processors taking into account data and control dependency
d) Assume the costs associated with tasks at each level and **Calculate** the critical path length
e) **Calculate** the average and maximum degree of concurrency
f) **Create** a task interaction graph based on decomposition you have performed

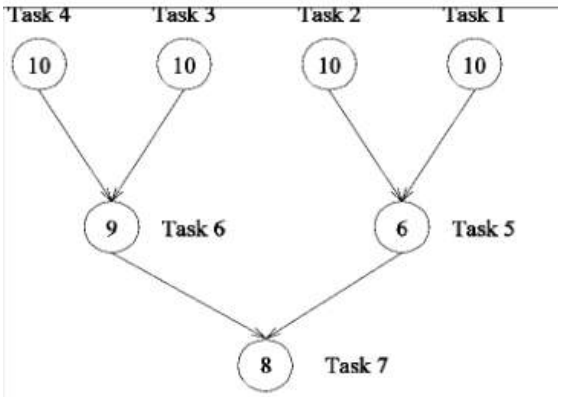
Comment on the shortest parallel execution time for the decomposition you have done for this problem.

Solution:

a) **Decompose** the problem in multiple tasks



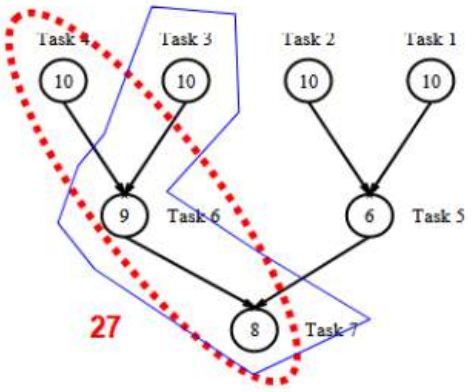
b) **Build** a task dependency graph for the tasks identified



c) **Select** appropriate mapping of tasks to different processors taking into account data and control dependency

Task 1 – Processor 0
Task 2 – Processor 1
Task 3 – Processor 2
Task 4 – Processor 3
Task 5 – Processor 0
Task 6 – Processor 2
Task 7 – Processor 0

d) Assume the costs associated with tasks at each level and **Calculate** the critical path length



e) **Calculate** the average and maximum degree of concurrency

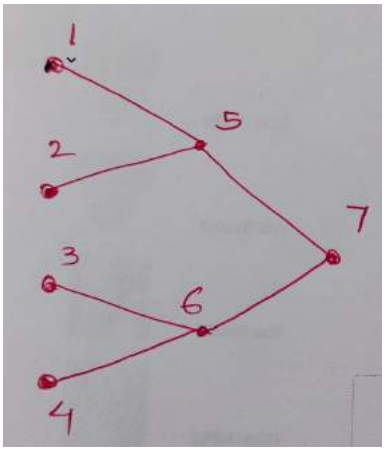
Phase 1: Task 1, Task 2, Task 3 and Task 4 can be executed parallelly (if you have more than 3 processors). So the total amount of work in this phase is the sum of the weights of those four nodes: 10+10+10+10 = 40.

Phase 2: Task 6 and Task 5 can be executed parallelly (if you have more than 1 processor). So the total amount of work in this phase is: 9+6 = 15.

Phase 3: You can execute only Task 7, so the total amount of work here is 8.

The maximum number of concurrency is max(40, 15, 8) = 40. The average degree of concurrency is (40+15+8)/(10+9+8) = 63/27 = 2.33

f) **Create** a task interaction graph based on decomposition you have performed



Comment on the shortest parallel execution time for the decomposition you have done for this problem.

Assuming each task takes 10ms for computation and 5ms for communication. The completion can ideally be completed in 50ms (10+5+5+10+5+5+10)

2

Consider the following array for a sorting problem using Quick Sort Algorithm.

5	17	4	18	25	63	0	8	7	9	1	23	15	3	45	27

For the parallel execution of the above sorting problem answer the following:

a) **Select** the most optimal decomposition technique for the above problem and justify the reason for your selection
b) **Decompose** the problem using quick sort
c) **Build** a task dependency graph for the tasks identified in step 2
d) **Select** appropriate mapping of tasks to different processors taking into account data and control dependency
e) Assume the costs associated with tasks at each level and **calculate** the critical path length
f) **Calculate** the average and maximum degree of concurrency
g) **Create** a task interaction graph based on decomposition you have performed

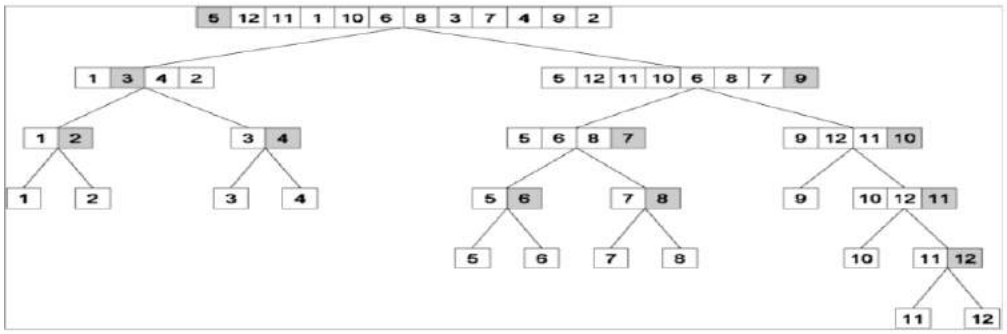
Comment on the shortest parallel execution time for the decomposition you have done for this problem.

Solution:

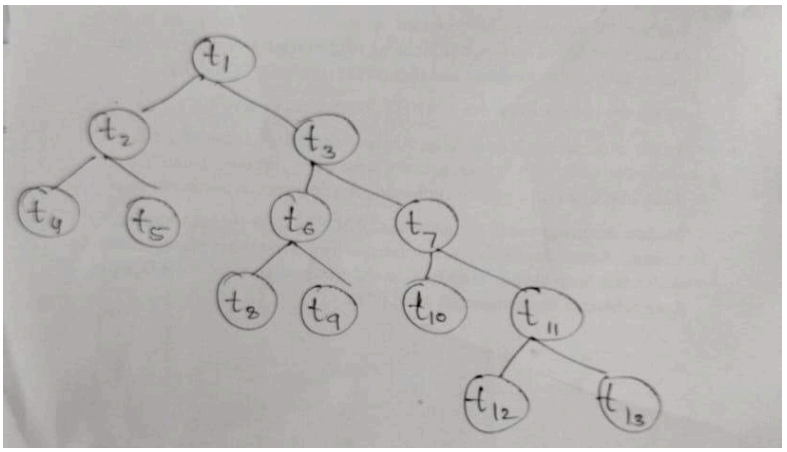
a) **Select** the most optimal decomposition technique for the above problem and justify the reason for your selection.

Recursive decomposition works well for divide and conquer kind of problems. Quick sort is a classical example of divide and conquer technique.

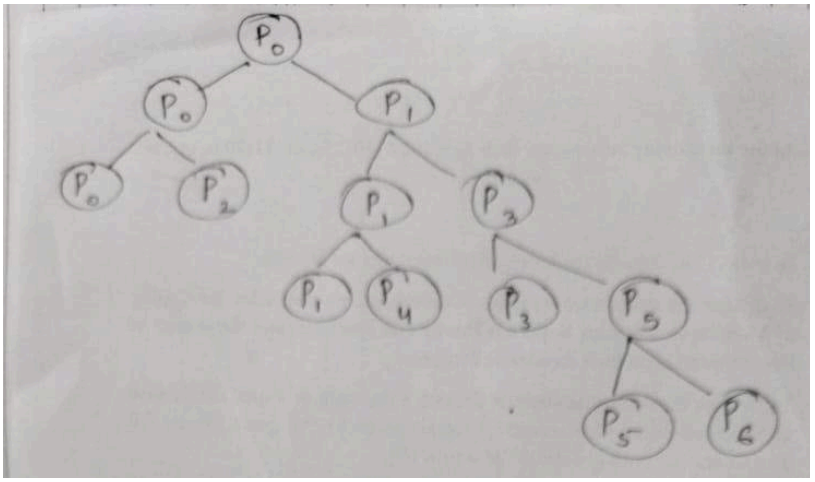
b) **Decompose** the problem using quick sort



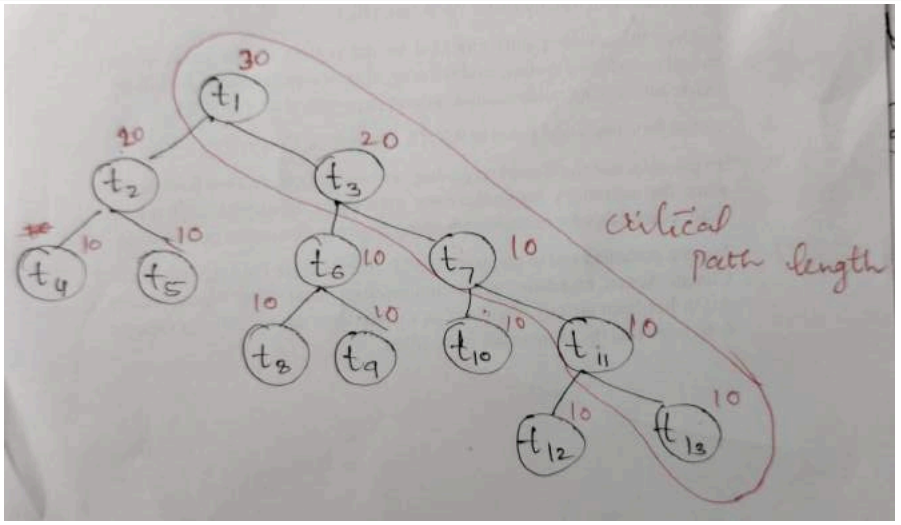
c) **Build** a task dependency graph for the tasks identified in step 2



d) **Select** appropriate mapping of tasks to different processors taking into account data and control dependency



e) Assume the costs associated with tasks at each level and **calculate** the critical path length

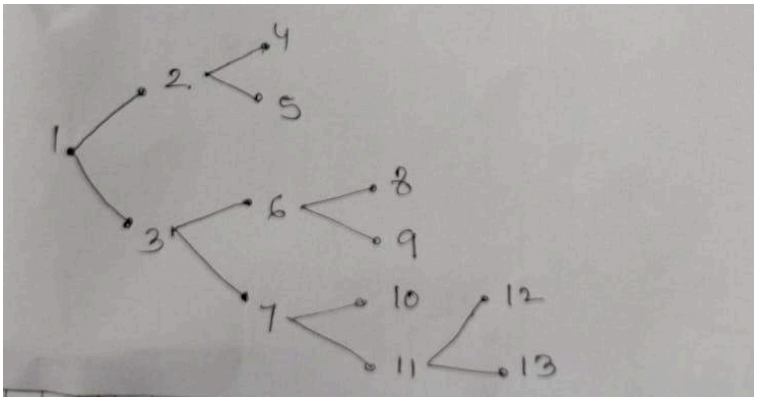


Critical Path length: **80**

f) **Calculate** the average and maximum degree of concurrency

Maximum degree of concurrency: **40**
Average Degree of Concurrency: **30+40+40+40+20/80 = 2.125**

g) **Create** a task interaction graph based on decomposition you have performed



Comment on the shortest parallel execution time for the decomposition you have done for this problem.

Assuming each task takes time as mentioned in point e) for computation and 5ms for communication. The completion can ideally be completed in 130ms
(30+5+5+20+5+5+5+5+5+10+5+5+5+5+10+5+5)