ILLINOIS TECH

College of Computing

CS 546 — Homework 1

Godha Pallavi Bhogadi (A20531496) — Section: 02

Due: Feb 16, 2026 (11:59 PM)

---

1. List three major problems requiring the use of supercomputing in the domain of Artificial Intelligence. (5 pts)

Answer:

Below are the three major problems requiring the use of supercomputing in the domain of Artificial Intelligence

(a) **In the training of Large Language Models (LLMs)**

In today's world data is increasing rapidly and there are massive amounts of data on which AI needs to be trained to predict the next outcome precisely. Using supercomputers for this problem can divide the work onto multiple processors and reduce the training time from years to weeks.

(b) **Weather Prediction**

Intially, to predict the weather many Partial Differential Equations (PDEs) are solved manually. With AI, PDEs can be solved easily. But data is massive in this case (can be decades of global satellite data) and also involves complex computing on this massive amounts of data. So, using supercomputers for weather prediction makes computing easy and prediction fast.

(c) **Generative Video (Graphics System)**

Videos are heavier than text and 1 second video could approximately contain thousands of images. Processing lots of images need massive computing power and having supercomputers for such graphics tasks would take less computing power as several parallel processors work together to generate a video.

2. What are the three basic programming paradigms for parallel processing? Explain. (5 pts)

Answer:

Below are the three basic programming paradigms categorized on how processors access memory and how processors communicate and pass information with each other.

(a) **Shared Memory Paradigm**

- In this model, all processors can access same global memory.

- When a program creates multiple threads for different tasks to run at the same time, they can communicate, almost instantly, through a shared space called global memory.
- Data communication is easy in this model.

(b) **Distributed - Memory(or Message Passing) Paradigm**

- In this model, each processor have its own private memory. For the communication of processors, each processor has to explicitly send a message and the other processor has to receive the message.
- Network Communication is required to share messages.
- Data communication is slow in this model as the message has to travel through a network before reaching the receiver.

(c) **Data Parallel Paradigm**

- In this model, the same instruction is executed simultaneously on many data elements
- This enables us to do thousands and millions of operations on many data points simultaneously.
- This can be used in matrix multiplication or when performing same operations on each element of large arrays.

3. What is the bottleneck in a von Neumann machine? (5 pts)

Answer:

- The bottleneck in Von Neumann Machine is due to separation of CPU and Memory Unit that effects overall speed and efficiency.
- Every piece of data and every instruction needs to travel in a same communication channel called System Bus.
- With years CPUs speed have increased and now have the ability to perform tasks at a higher rate but data/instruction fetch through system bus is very slow compared to CPUs speed.
- A computer cannot fetch both instruction and data at the same time. So, it must wait until the bus is clear.
- Due to this bottleneck, CPUs are not used to their full capacity and are constatly waiting for either data or instructions.
- This bottleneck is called **Memory-wall problem (or Memory Bandwidth Problem)** due to which performance of CPU becomes limited by how fast data moves not how fast CPU computes.

4. Discuss the difference between shared address space machines and distributed address space machines. Discuss the advantages and disadvantages of both architectures. (10 pts)

Answer:

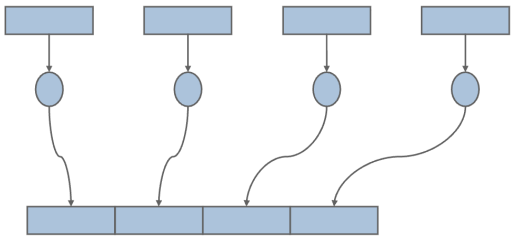
Shared Address Space Machines	**Distributed Address Space Machines**
(a) This is often called Shared Memory (b) In this machine, all processors have access to a single shared memory (c) Communication is easy as one processor has to just read data that another processor had written to the memory. (d) **Examples:** multi-core processors, on-chip multiprocessor (e) **Advantages** i. Fast Communication ii. Ease of Programming (f) **Disadvantages** i. Not easily reliable ii. Data Race Bugs iii. Synchronization overhead	(a) This is often called Private/Distributed Memory (b) Each processor have its own private memory and do not have access to another processors memory (c) Communication between processors is hard as one processor has to explicitly send a message and other has to receive it. (d) Examples" Supercomputers (e) **Advantages** i. Easily Scalable ii. Highly Reliable iii. No data race conditions (f) **Disadvantages** i. Communication overhead ii. Hard to Program

Table 1: Comparison of Machine Architectures

5. What is parallel I/O? Why do we need parallel I/O? (10 pts)

Answer:

Parallel I/O: Parallel I/O is the process of allowing multiple processes or threads of a parallel program to access data concurrently from a common file.



Why do we need parallel I/O?

- It is needed in modern applications to handle massive amounts of data.
  Accessing data parallelly will make the computation faster. Otherwise, it creates a bottleneck where CPUs/GPUs are not used in their full capacity and sits idle until they receive the data.
- It allows difference processors to write to different parts of the same file without corrupting the data.

6. What is Moore's Law? Does it still hold? Why or why not? (5 pts)

Answer:
Gordon Moore's Law: The number of transistors that can be fabricated on a single integrated circuit at a very reasonable cost doubles every year and the processor speeds also doubles at the very same rate.

Does it still hold? I think, No.

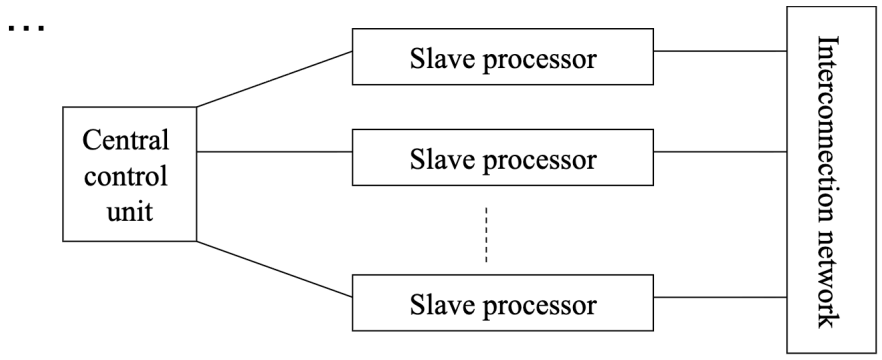
Why not?

- Transistors are already approaching atomic size level and cannot be shrink further.
- Smaller transistors are expensive and increasing the number of transistors could directly explode the cost of the circuit and the product.
- If number of transistors keeps increasing, it produces more heat and device gets heated up very quickly.

7. Compare and contrast SIMD vs MIMD and explain when you would prefer one over the other. (10 pts)

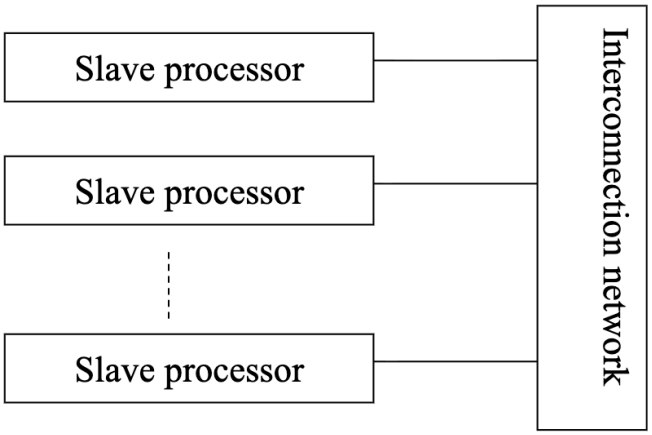
Answer:
SIMD (Single Instruction Multiple Datastream):

- The main idea of SIMD is that one control unit broadcasts on instruction (or a program) to multiple processors.
  And these processors execute the same instruction/program in lockstep on multiple data elements.
- Each processor may see different data.
- Individual processors can be turned on/off at each cycle using a mask (and the process of using a mask is called masking).
- All processors are expected to execute a step at the same time.



MIMD (Multiple instruction Multiple Datastream):

- The main idea of MIMD Architecture is that each processor executes an instruction (or a program) independent of other processors and each processor operate on separate data streams.
- Since processors in MIMD are independent, they can run at its own pace and are not forced to execute instrucions at the same time like in SIMD.
- SPMD (Single Program Multiple Data) is the common programming style in MIMD.
- Even though same program is run on different data, their execution paths and execution times differ.



Why do you prefer one over the other?

- We use SIMD when we have to do same operation on multiple data points at the same time. Like when performing same operation/ instruction on large arrays, matrices etc.
  Examples: Matrix Multiplication, Add one to each element in the array.
- We use MIMD when we have different tasks on different data and each task is independent of other tasks.
  It can be used when there's complex branching and each processor may have different execution path.

SIMD	MIMD
(a) Need custom hardware for processors. (b) Slave processors are simple and therefore low cost (c) Good for programs with lot of synchronization due to clock step operation	(a) Easy to build out of commodity parts (b) Processors are complex but availability of low-cost commodity microprocessors offsets the SIMD advantage. (c) It can handle a more general class of problems with reasonable efficiency.

Table 2: SIMD vs MIMD

8. Give two examples of anti-dependence and give the corresponding solutions to remove the dependence. (10 pts)

Answer:

Anti-dependence: A statement S2 has an anti-dependence on S1 iff S2 writes to a value that is read by S1.

Example 1:

$$S1: \quad a = b + c \qquad // \text{ reads b}$$
$$S2: \quad b = d + e \qquad // \text{ writes to b which is read by S1}$$

To remove the anti-dependence, S2 can write into a new variable (say, b_{new}) instead of b.

$$S1: \quad a = b + c \qquad // \text{ reads b}$$
$$S2: \quad b_{new} = d + e \qquad // \text{ writes to b which is read by S1}$$

Example 2:

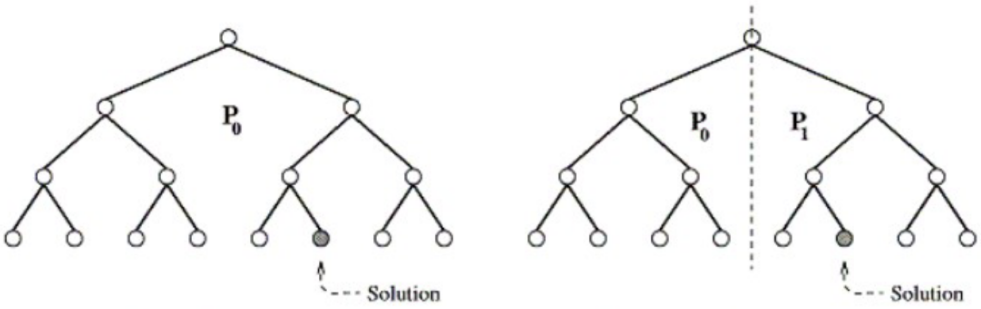
```
for (i = 0; i < n - 1; i++) {
    arr[i] = arr[i+1] + 1;
}
```

The above code has anti-dependence. For any iteration i, we read ar[i+1] and in the next iteration, we write to ar[i+1].
To remove the anti-dependence, we write to a new array first and then copy elemnts from new array to old array.

```
// compute and store values in a new array(arrNew)
for (i = 0; i < n - 1; i++) {
    arrNew[i] = arr[i+1] + 1;
}
```

```
//copy values from new array to the original array
for (i = 0; i < n - 1; i++) {
    arr[i] = arrNew[i];
}
```

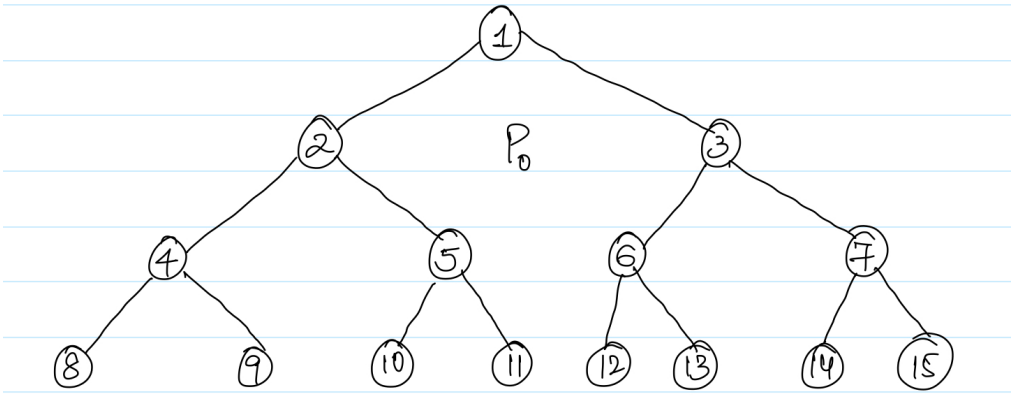
9. Consider the search tree shown in the figure, in which the dark node represents the solution. (10 pts)



(a) DFS with one processor (b) DFS with two processors

(a) If a sequential search of the tree is performed using the standard DFS algorithm, how much time does it take to find the solution if traversing each arc takes one unit of time?

Answer:



Below are the steps that the standard DFS algorithm would go through

i. Start at the root node 1.

ii. traverse $1 \rightarrow 2 \rightarrow 4 \rightarrow 8$

iii. Back track $8 \rightarrow 4$

iv. traverse $4 \rightarrow 9$

v. Back track $9 \rightarrow 4 \rightarrow 2$

vi. traverse $2 \rightarrow 5 \rightarrow 10$

vii. Back track $10 \rightarrow 5$

viii. traverse $5 \rightarrow 11$

ix. Back track $11 \rightarrow 5 \rightarrow 2 \rightarrow 1$

x. traverse $1 \rightarrow 3 \rightarrow 6 \rightarrow 12$

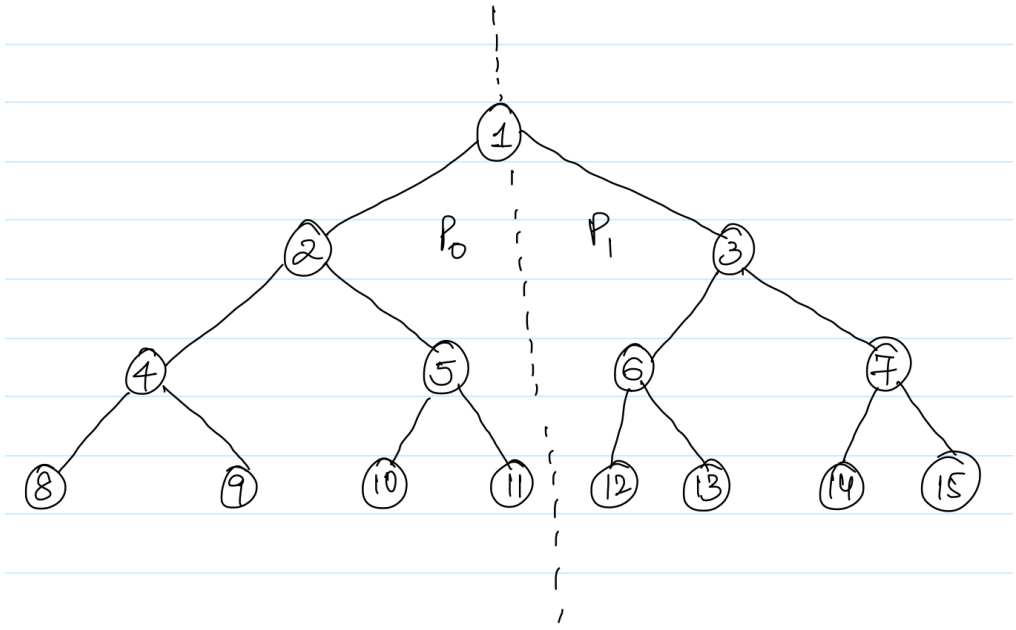
xi. Backtrack $12 \rightarrow 6$

xii. Traverse $6 \rightarrow 13$ (reached solution node)

Since traversing each arc takes one unit of time(including the cost for backtracking), total time needed to find the solution node = 19

(b) Assume the tree is partitioned between two processing elements as shown. If both perform DFS on their halves, how much time does it take to find the solution? What is the speedup? Is there a speedup anomaly? Explain.

Answer:



Below are the steps that takes place on each processor simultaneously.

Processor P0

i. Start at the root node 1

ii. traverse $1 \rightarrow 2 \rightarrow 4 \rightarrow 8$

iii. Back track $8 \rightarrow 4$

iv. traverse $4 \rightarrow 9$

Processor P1

i. Start at the root node 1

ii. traverse $1 \rightarrow 3 \rightarrow 6 \rightarrow 12$

iii. Back track $12 \rightarrow 6$

iv. traverse $6 \rightarrow 13$ (reached solution node)

As the second processor (P1) found the solution node, both processors come to an halt. Time taken to reach the solution node = 5

Speedup

$$\begin{aligned} \text{Speedup (S(p))} &= \frac{\text{Time taken by one processor}}{\text{Time taken by p processors}} \\ &= \frac{19}{5} \\ &= 3.8 \end{aligned}$$

Speedup Anomaly

Yes, there is a speedup anomaly. It is called superlinear speedup as the parallel program runs more than 2 times faster(2 - number of processors) when compared to the original program.

$S(p) > p$ **- Superlinear Speedup**

10. Consider a memory system with L1 cache of 32 KB and DRAM of 512 MB with the processor operating at 1 GHz. L1 latency is one cycle, DRAM latency is 100 cycles. In each memory cycle, the processor fetches four words (cache line size is four words). What is the peak achievable performance of a dot product of two vectors? Assume an optimal cache placement policy where necessary. (10 pts)

Given code:

```
for (i = 0; i < dim; i++)
    dot_prod += a[i] * b[i];
```

Answer:

Given

- Latency of L1 cache = 1 cycle
- Latency of DRAM = 100 cycles
- Processor frequency = 1 GHz = 10^9 cycles/second

$$\text{Time per cycle} = \frac{1}{10^9} = 10^{-9} \text{ seconds}$$

FLOPs per iteration

For each loop iteration:

$$\text{dot\_prod} += a[i] \times b[i]$$

$$= 2 \text{ FLOPs per iteration(one multiplication and one addition)}$$

Total FLOPs

We consider only four iterations as the processor fetches for every 4 iterations.

$$\text{Total FLOPs for 4 iterations} = 4 \times 2 = 8 \text{ FLOPs}$$

To process 4 iterations we must load:

$$a[i \ldots i+3] \quad \text{and} \quad b[i \ldots i+3]$$

Since cache line size is 4 words it has to fetch from the DRAM twice

$$\text{Total memory latency} = 2 \times 100 = 200 \text{ cycles}$$

FLOPs per cycle

$$\text{FLOPs per cycle} = \frac{8}{200} = 0.04$$

Peak Performance

$$\text{Performance} = (\text{FLOPs per cycle}) \times (\text{cycles per second})$$

$$= 0.04 \times 10^9 = 4 \times 10^7 \text{ FLOPs/sec} = 40 \text{ MFLOPs/sec}$$

11. Why is it difficult to construct a true shared-memory computer? What is the minimum number of switches for connecting p processors to a shared memory with b words (where each word can be accessed independently)? (10 pts)

Answer:

A true shared-memory computer is a computer that has same latency to any processor accessing any word of the shared memory.

But it is very difficult to construct such a shared memory computer due to following reasons

(a) If the number of processors increases and if many processors request for memory simultaneously, a single memory model cannot serve all the requests at once. This creates latency and performance degrades as the processors increases.

(b) To allow every processor to access memory location independently, we need to add a switch for every processor and memory location pair.
Handling large number of switches is hard and becomes impractical as number of switches increases.

(c) Signals travel at a finite speed. In large machines, memory located farther away takes longer to access than nearby memory. Therefore uniform memory access time cannot be maintained which contradicts the definition of true shared-memory computer.

We have p processors and b independently accessible memory words. For a true shared-memory computer, each processor must be connected with each memory location separately to access in parallel.
So, minimum number of switches for connecting p processors to a shared memory with b words = $p * b$

12. A cycle in a graph is defined as a path originating and terminating at the same node. The length of a cycle is the number of edges in the cycle. Show that there are no odd-length cycles in a d-dimensional hypercube. (10 pts)

Answer:

d-Dimensional Hypercube

A d-dimensional hypercube is a graph defined by the following rules.

(a) It has 2^d vertices.

(b) Each vertex is assigned a unique binary string of length d.

(c) There is an edge between two vertices if and only if their binary strings differ in exactly one position.

Bipartite

A graph is said to be bipartite if we can split its vertices into two graphs such that there are edges only between vertices of two groups and no edges between vertices of same group.

Hypercube is a bipartite

We can divide the vertices of hypercube into 2 groups.
Group A (V_{even}) has vertices with even number of 1's.
Group B (V_{odd}) has vertices with odd number of 1's.
As per the definition of hypercube, two adjancent vertices only differ in one bit.
So, vertices in V_{even} group are connected to V_{odd} group only and there's no edge within same group.
So, hypercube is a bipartite.

Bipartite has no-odd length cycles

(a) Start a vertex in V_{even}

(b) The first edge must lead to V_{odd}

(c) The second edge must lead to V_{even}

(d) So, any path of length k starting with V_{even} ends in V_{even} if k is even and ends in V_{odd} if k is odd.

(e) Similarly for any path of length k starting with V_{odd} ends in V_{odd} if k is even and ends in V_{even} if k is odd.

So, for a cycle to terminate at starting node, it requires even number of steps (or edges).
And therefore, there is no odd-length cycles in a d-dimensional hypercube.