1- Give some examples of applications that need high computational power.

- Video Games.
- Web Searches.
- Data Analysis.
- Climate Modeling.
- Protein Folding.
- Drug Discovery.
- Energy Research.

2- Why parallel systems are built?

- To get higher performance and since we cannot increase the speed of ICs more due to physical limitation as the power consumption and heat increases with speed, thus, we exploit increasing density to build parallel systems.

3- Describe how n processors can be used to compute and add p values in an efficient way.

- By Data-Parallelism, we can divide the data (values) to process among n processors (p / n).

4- Explain the difference between task and data parallelism.

- Task-Parallelism: partitioning the various tasks carried out in solving the problem among the cores.
- Data-Parallelism: partitioning the data used in solving the problem among the cores, and each carries out similar operations on its part of the data.

5- Describe the difference between shared and distributed memory systems.

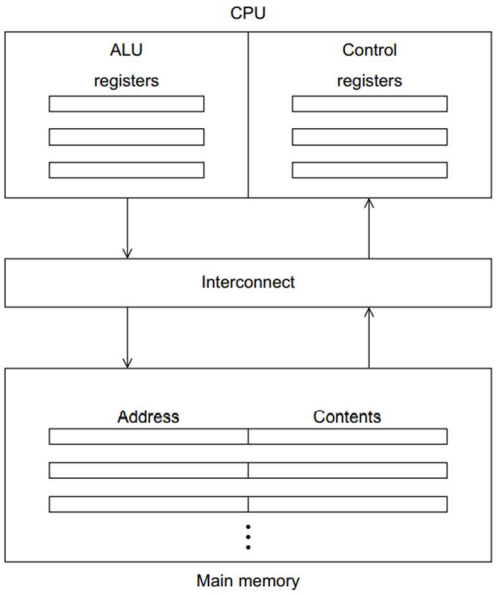
- Shared-Memory: cores can share access to the computer's memory.
- Distributed-Memory: each core has its own, private memory, and the cores must communicate explicitly by sending messages.

6- Explain the concurrent, parallel, and distributed concepts.

- Concurrent: multiple tasks can be in progress at any instant.
- Parallel: multiple tasks cooperate closely to solve a problem.
- Distributed: a program may need to cooperate with other programs to solve a problem.
- Parallel vs Distributed:
  - ❖ Parallel program: multiple tasks executed on cores that are physically close to each other.
  - ❖ Distributed programs: loosely coupled as tasks may be executed by multiple computers separated by big distance

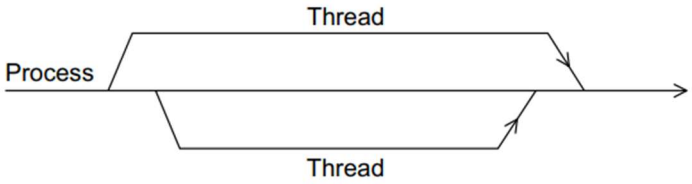
7- Draw and explain the von Neumann architecture.

- Consists of Memory & CPU & Interconnection.
- CPU consists of Control Unit & Arithmetic Logic Unit (ALU) & Registers.
- Interconnection between memory and CPU (bus) for sending instructions and data.



8- Explain the concepts: processes, multitasking, and threads.

- Processes: instance of a program that is being executed.
- **Multitasking**: simultaneous executing of multiple programs.
- **Threading**:
  - ❖ Dividing programs into independent tasks.
  - ❖ Threads are contained within processes.
  - ❖ When a thread is started, it forks off the process.
  - ❖ When a thread is terminated, it joins the process.



9- How can cache speed up a program execution?

- Collection of memory locations that can be accessed in less time than some other memory locations

10- What is the difference between cache hit and cache miss?

- **Cache Hit:** when CPU check for info and it's available.
- **Cache Miss**: when CPU check for info and it's NOT available.

11- Suppose the main memory consists of 16 lines with indexes 0–15, and the cache consists of 4 lines with indexes 0–3. Where lines should be stored using direct, fully associative, and 2-way mapping.

- **Fully Associative:** line 0 can be assigned to cache location 0, 1, 2, or 3.
- **Direct:** assigning lines by looking at their remainder after division by 4.
- **2-way:** we might group the cache into two sets: indexes 0 and 1 form one set , set 0 — and indexes 2 and 3 form another — set 1 , So we could use the remainder of the main memory index modulo 2, and cache line 0 would be mapped to either cache index 0 or cache index 1.

12- With an example show how matrix processing can be efficient by considering how cache works.

- By looping over rows instead of looping over columns.

13- What is the drawback of page table and how this drawback can be solved?

- **Drawback:** Doubling the time needed to access a location in main memory.
- **Solution:** Translation-lookaside Buffer (TLB).

14- With examples, describe the pipelining and speculation concepts.

- **Pipelining:** functional units are arranged in stages like a factory assembly line.
- **Speculation:** Executes the instruction based on the guess.

15- What is the difference between coarse-grained and fine-grained parallelism?

- **Fine-grained:** CPU switches between threads after each instruction skipping threads that are stalled.
- **Coarse-grained:** Only switch threads that are stalled.

16- Describe SIMD system (Single Instruction, Multiple Data).

- Parallel systems operating on multiple data streams by applying same instruction.

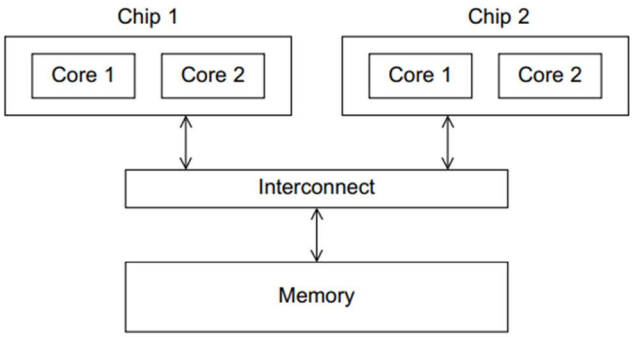
17- What is the characteristic of vector processers?

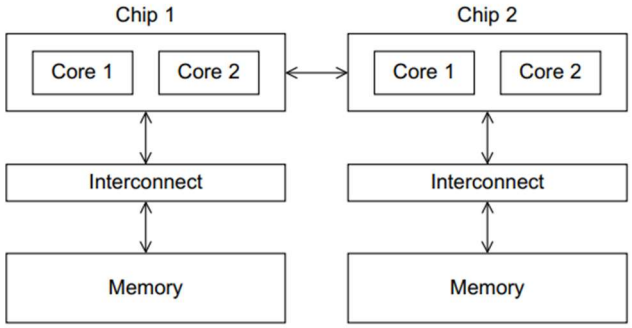
- Vector registers.
- Vectorized and pipelined functional units.
- Vector instructions.
- Interleaved memory.
- Strided memory access and hardware scatter/gather

18- Describe the MIMD systems. (Multiple Instructions, Multiple Date).

- Systems support multiple simultaneous instruction streams operating on multiple data streams.
- Collections of fully independent cores and each has its own control unit and ALU.

19- Draw the architecture for UMA and NUMA multicore systems and describe them.

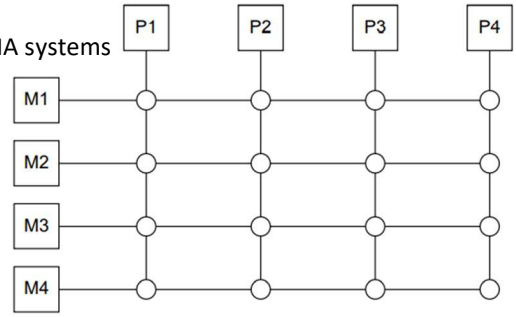




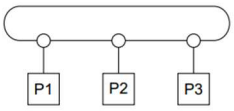
- UMA (uniform memory access):
  - ❖ Systems are usually easier to program
    as the programmer doesn't need to worry about different access times for different memory locations.
- NUMA (nonuniform memory access):
  - ❖ Systems have the potential to use larger amounts of memory than UMA systems

20- Describe the crossbar as interconnect for shared memory system.

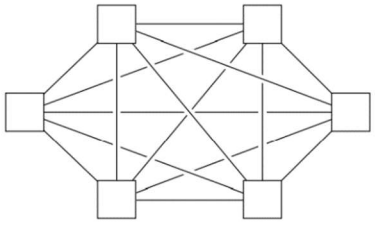
- **Lines** are bidirectional communication links.
- **Squares** are cores or memory modules.
- **Circles** are switches.



21- Draw ring

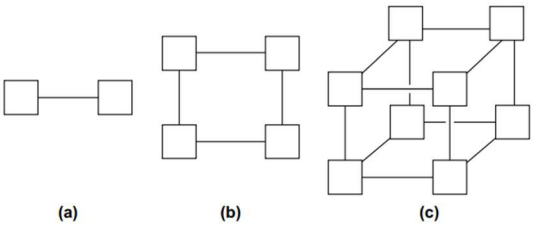


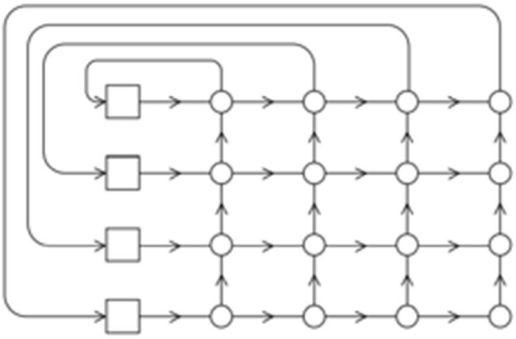
22- Draw Fully connected network



23- Draw Hypercube

Draw Crossbar (indirect)





24- Compute the bisection width for each interconnect.

- The bisection width of:
  - ❖ p × p crossbar is p.
  - ❖ Omega network is p/2
  - ❖ Two-dimensional toroidal mesh is 2√p

25- Show the difference between crossbar and omega network switches.

- **Crossbar:** if two processors don't attempt to communicate with the same processor,
  all the processors can simultaneously communicate with another.
- **Omega:** The switches are two-by-two crossbars, thus, some communications cannot occur simultaneously.

26- Describe the latency concept.

- The time that elapses between,
  the source's beginning to transmit the data and the destination's starting to receive the first byte.

27- Describe with example how cache coherence problem can happen.

- When storing same variable in caches of multiple CPU and it's updated by one CPU,
  but the update is not seen by other CPUs.

28- Explain the concepts: Snooping Cache Coherence, Directory-based Cache Coherence, and False Sharing.

- **Snooping Cache Coherence:** When the cores share a bus,
  any signal transmitted on the bus can be "seen" by all the cores connected to the bus.
- **Directory-based Cache Coherence:** solve snooping problem
  which is that broadcasting across the interconnect will be very slow
  using a data structure called a directory which stores the status of each cache line.
- **False Sharing:**
  - ❖ When system is behaving as if the elements of variable were being shared by the cores.
  - ❖ Doesn't cause incorrect results but ruin the performance.

29- Which approach to use: distributed memory or shared memory?

- Since Distributed-memory interconnects are quite cheap than in Shared-memory,
  thus, they are better suited for problems requiring vast amounts of data or computations

30- Give the structure of SPMD programs (Single program, Multiple data)

- Consist of a single executable behaving as if it were multiple different programs using conditional branches.

31- Describe the difference between static and dynamic threads.

- **Dynamic threads:**
  - ❖ There's a master thread and collection of worker threads.
  - ❖ The master thread typically waits for work requests.
- **Static threads:**
  - ❖ All threads are forked after any needed setup by the master thread and they run until all the work is completed.

32- Give examples for nondeterminism when processors execute asynchronously.

- Suppose we have two threads, **id 0 & id 1**.
- Each is storing a private variable x, thread 0's value for x is 7, and thread 1's is 19
- Both execute the following

```
printf("Thread %d > my val = %d\n", my rank, my x);
```

Then the output could be

```
- Thread 0 > my val = 7
- Thread 1 > my val = 19
```

but it could also be

```
- Thread 1 > my val = 19
- Thread 0 > my val = 7
```

33- How mutex and busy waiting can be used to ensure only one thread executes certain instructions at a time?

- **Mutex**: The basic idea is that each critical section is protected by a lock. Before a thread can execute the code in the critical section, it must "obtain" the mutex by calling a mutex function
- **Busy-waiting**: a thread enters a loop whose sole purpose is to test a condition

34- Calling functions designed for serial programs can be problematic in parallel program, give an example.

- C string library function *strtok* splits an input string into substrings.
- When it's first called, it's passed a string, and on subsequent calls it returns successive substrings.
- This can be arranged using a static char variable that refers to the string that was passed on the first call.
- Now suppose two threads are splitting strings into substrings.
- Clearly, if, for example, thread 0 makes its first call to *strtok*, and then thread 1 makes its first call to *strtok* before thread 0 has completed splitting its string, then thread 0's string will be lost or overwritten, and, on subsequent calls it may get substrings of thread 1's strings.

35- Write lines of code for sending a message from process 0 to process 1.

```
char message[100];
my_rank = get_rank();

if(my_rank == 0) {
    sprintf(message, "Greetings from process 0");
    send(message, MSG CHAR, 100, 1);
}
else if(my_rank == 1) {
    receive(message, MSG CHAR, 100, 1)
    printf("Process 1 > Received: %snn", message)
}
```

36- Give examples of functions for collective communication.

- *MPI_Reduce().*
- *MPI_Allreduce().*
- *MPI_Gather().*
- *MPI_Allgather().*
- *MPI_Bcast().*
- *MPI_Scatter().*

37- What one sided communication means?

- When a single process calls a function, to update either local/remote memory.

38- How remote memory access affects the program performance?

- Simplify communication since it requires a single process thus reducing cost of communication.

39- A lot of issues can appear with input and output in parallel system, give some rules to avoid these issues.

- **In distributed-memory**, only process 0 will access *stdin*.
- **In shared memory**, only the master thread/0 will access *stdin*.
- **In both**, all process/threads can access *stdout*.

40- Explain the concepts speed up and efficiency.

- **Speedup:** metric to quantify performance by comparing the execution of a serial algorithm and parallelized version of the same algorithm.
- **Efficiency:** metric to build on top of speedup by adding awareness of the underlying hardware.

41- What is meant by Amdahl's low?

- It says, unless all serial program are parallelized virtually, the possible speedup will be very limited.

42- When a program is weakly or strongly scalable?

- **Strong:** when increasing the # of pro/thr, the efficiency is fixed without increasing problem size.
- **Weakly:** keeping the efficiency fixed by increasing the problem size.

43- Outline the foster's methodology.

- Partitioning.
- Communication.
- Agglomeration.
- Mapping.

44- Write lines of code to time a block of MPI code and report a single elapsed time.

```
double local_start, local_finish, local_elapsed, elapsed;
MPI_Barrier(comm);

local_start = MPI_Wtime();
local_finish = MPI_Wtime();
local_elapsed = local_finish - local_start;

MPI_Reduce(&local_elapsed, &elapsed, 1, MPI_DOUBLE, MPI_MAX, 0, comm);
if(my_rank == 0)
    prinf("Elapsed time = %e seconds\n", elapsed);
```

45- Why at some point, the run-times of parallel program can start to get worse?

- Because of the relation between runtime of serial programs and parallel programs.

$$T_{\text{parallel}}(n,p) = T_{\text{serial}}(n)/p + T_{\text{overhead}}.$$

46- Sort 5, 9, 4, 3 using quick sort and using odd-even transposition sort.

- **Quick Sort:**
  - ❖ Select 5 as pivot to divide other numbers into 2 lists.(less and greater)
  - ❖ We repeat the same step for the 2 lists.
  - ❖ We combine the 3 lists together after sorting and put the first pivot back in middle.
- **Odd-even transposition sort:**

  Start: 5,9,4,3
  Even phase: Compare-swap (5,9) and (4,3), getting the list 5,9,3,4.
  Odd phase: Compare-swap (9,3), getting the list 5,3,9,4.
  Even phase: Compare-swap (5,3) and (9,4), getting the list 3,5,4,9.
  Odd phase: Compare-swap (5,4), getting the list 3,4,5,9.

47- Sort 15, 11, 9, 16, 3, 14, 8, 7, 4, 6, 12, 10, 5, 2, 13, 1 using parallel odd-even transposition sort (with and without lists' merge) on four processors.

	Process			
Time	0	1	2	3
Start	15, 11, 9, 16	3, 14, 8, 7	4, 6, 12, 10	5, 2, 13, 1
After Local Sort	9, 11, 15, 16	3, 7, 8, 14	4, 6, 10, 12	1, 2, 5, 13
After Phase 0	3, 7, 8, 9	11, 14, 15, 16	1, 2, 4, 5	6, 10, 12, 13
After Phase 1	3, 7, 8, 9	1, 2, 4, 5	11, 14, 15, 16	6, 10, 12, 13
After Phase 2	1, 2, 3, 4	5, 7, 8, 9	6, 10, 11, 12	13, 14, 15, 16
After Phase 3	1, 2, 3, 4	5, 6, 7, 8	9, 10, 11, 12	13, 14, 15, 16

48- Write lines of code for computing the partner rank in odd-even transposition sort?

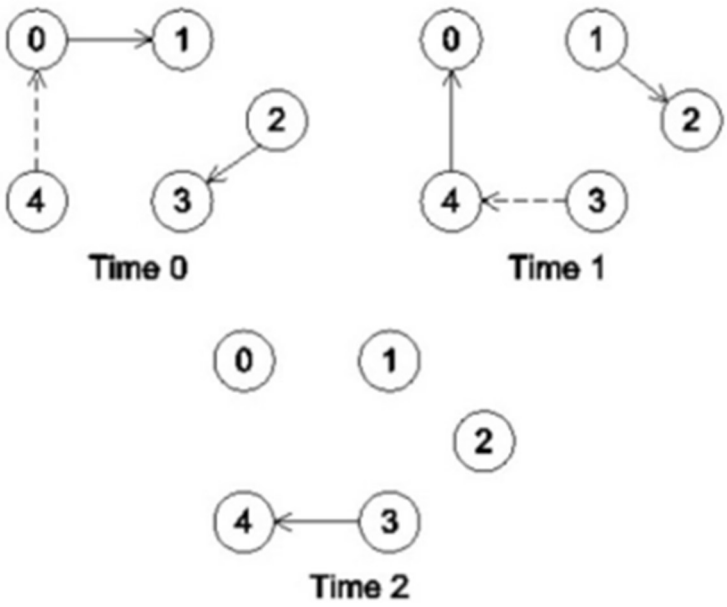
```
if(phase%2 == 0)
    if(my_rank%2 != 0)
        partner = my_rank-1;
    else partner = my_rank+1;

else if(phase%2 != 0)
    if(my_rank%2 != 0)
        partner = my_rank+1;
    else partner = my_rank-1;

if(partner==-1 || partner==comm.sz)
    partner = MPI_PROC_NULL;
```

49- How to make the communication safe in the parallel odd-even sort program and show how the communication is safe within five processes?

- To make a communication safe, it must be restructured.



50- Describe two applications of barriers.

- Timing some part of multi thread program.
- Using barriers in debugigng.

51- Reusing counter in barrier implementation using busy waiting is problematic, while it is not problematic when semaphores are used, explain why.

- **In busy waiting:** if the last thread tries to rest it, some other thread may hang in the *busy_wait* thread.
- **In semaphores:** as we reset it before releasing any of thread from barriers.

52- Executing concurrent multiple read and write operations on a linked list can cause problems. What are these problems and how they can be solved?

- **Problem**: Accessing invalid members
- **Solution**: Locking the list any time a thread attempts to access it.

53- Use insert function to add 2, 8, 7, 5 to a linked list. Then, use member function to find 5 and 9. Finally, delete 5 from the list.

```
int insert(int value, node head) {
    Node current = head;
    Node prev = null;
    Node temp;

    while(current!=null || current->data < value) {
        prev = current;
        current = current->next;
    }
    if(current==null || current->data > value) {
        temp->data = value;
        temp->next = current;

        if(prev == null) head = temp;
        else prev->next = temp;
    }
    else return 0;
}
```

```
int member(int value, node head) {
    Node current = head;

    while(current!=null && current->data < value)
        current = current->next;

    if(current==null || current->data > value)
        return 0;
    else return 1;
}
```

```
int delete(int value, node head) {
    Node current = head;
    Node prev = null;

    while(current!=null || current->data < value) {
        prev = current;
        current = current->next;
    }
    if(current!=null || current->data == value) {
        if(prev == null)
            head = current->next;
        else
            prev->next = current->next;

        free(current);
        return 1;
    }
    else return 0;
}
```