QUESTIONS ON CHAPTER 1

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

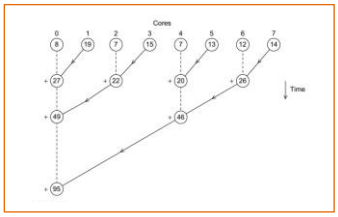
➢ ❶ Climate modeling ❷ Protein folding ❸ Drug discovery ❹ Energy research ❺ Data analysis

2. Why parallel systems are built ?

➢ Rather than building ever-faster, more complex, monolithic processors, the industry has decided to put multiple, relatively simple, complete processors on a single chip .

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

➢ We can pair the cores so that while core 0 adds in the result of core 1, core 2 can add in the result of core 3, core 4 can add in the result of core 5, and so on,
Then we can repeat the process with only the even-ranked cores :
0 adds in the result of 2, 4 , adds in the result of 6, and so on.
Now cores divisible by 4 repeats the process, and so on



4. Explain the difference between task and data parallelism?

• **Task parallelism** ➢ In task-parallelism, we partition the various tasks carried out in solving the problem among the cores.

• **Data parallelism** ➢ In data-parallelism, we partition the data used in solving the problem among the cores, and each core carries out more or less similar operations on its part of the data.

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

• **In a shared-memory system** ➢ the cores can share access to the computer's memory; in principle, each core can read and write each memory location.

• **In a distributed-memory system** ➢ each core has its own, private memory, and the cores must communicate explicitly by doing something like sending messages across a network .

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

• **Concurrent computing** ➢a program is one in which multiple tasks can be in progress at any instant

• **Parallel computing** ➢ a program is one in which multiple tasks cooperate closely to solve a problem

• **Distributed computing** ➢ a program may need to cooperate with other programs to solve a problem.

QUESTIONS ON CHAPTER 2

1. Draw and explain the von Neumann architecture ?

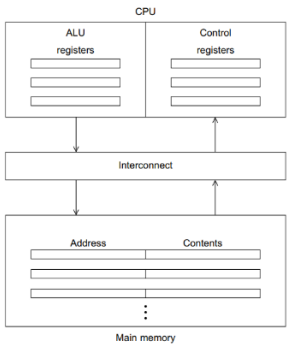


FIGURE 2.1
The von Neumann architecture

• **Consists of** ➢ (main memory- a central-processing unit (CPU) or processor or core - an interconnection between the memory and the CPU)

• **Main memory** ➢ consists of a collection of locations, each of which is capable of storing both instructions and data. Every location consists of an address, which is used to access the location and the contents of the location—the instructions or data stored in the location.
• **The central processing unit** ➢ is divided into a control unit and an arithmetic and logic unit (ALU). The control unit is responsible for deciding which instructions in a program should be executed, and the ALU is responsible for executing the actual instructions

2. Explain the concepts: processes, multitasking, and threads?
• **Processes** ➢ an instance of a computer program that is being executed
• **Multitasking** ➢ This means that the operating system provides support for the apparent simultaneous execution of multiple programs.
• **Threads** ➢ provides a mechanism for programmers to divide their programs into more or less independent tasks with the property that when one thread is blocked another thread can be run

3. How cache can speed up a program execution?
➢ Cache memory holds frequently used instructions / data that the processor may require next and it is faster to access memory than RAM since it is on the same chip as the processor, This reduces the need for frequent slower memory retrievals from main memory, which may otherwise keep the CPU waiting

4. What is the difference between cache hit and cache miss?
• **cache hit** ➢ When a cache is checked for information and the information is available
• **cache miss** ➢ If the information isn't available

5. 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 .
➢ In a fully associative cache , line 0 can be assigned to cache location 0, 1, 2, or 3.
In a direct mapped cache, we might assign lines by looking at their remainder after division by 4. So lines 0, 4, 8, and 12 would be mapped to cache index 0, lines 1 , 5 , 9 , and 13 would be mapped to cache index 1, and so on , In a two way set associative cache, 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 .

6. With an example show how matrix processing can be efficient by considering how cache works ?

```
double A[MAX][MAX], x[MAX], y[MAX];
. . .
/* Initialize A and x, assign y = 0 */
. . .
/* First pair of loops */
for (i = 0; i < MAX; i++)
for (j = 0; j < MAX; j++)
y[i] += A[i][j]*x[j];
. . .
/* Assign y = 0 */
. . .
/* Second pair of loops */
for (j = 0; j < MAX; j++)
for (i = 0; i < MAX; i++)
y[i] += A[i][j]*x[j];
```

Cache Line	Elements of A			
0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
3	A[3][0]	A[3][1]	A[3][2]	A[3][3]

Suppose that we have a cache with a maximum size of 8 Elements of 'A' (2 lines), and it's a direct-mapped, and MAX is 4, and 'A' is stored in memory like this **LOOK AT TABLE ABOVE**

First pair of loops: 4 Misses & 2 Lines are evicted due to the size of the cash (2 Lines only) .
Second pair of loops: 16 Misses & 14 Lines are evicted due to the size of the cash (2 Lines only). If MAX is 1000 The first pair of loops is much faster, approximately three times faster Lines only). If MAX is 1000 The first pair of loops is much faster, approximately three times faster than the second pair

7. What is the drawback of page table and how this drawback can be solved ?

➢ is that it can double the time needed to access a location in main memory.
In order to address this issue, processors have a special address translation cache called a translation-lookaside buffer, or TLB. It caches a small number of entries (typically 16–512) from the page table in very fast memory. Using the principle of spatial and temporal locality, we would expect that most of our memory references will be to pages whose physical address is stored in the TLB, and the number of memory references that require accesses to the page table in main memory will be substantially reduced

8. With examples, describe the pipelining and speculation concepts ?

• **Pipelining** ➢ The principle of pipelining is similar to a factory assembly line: while one team is bolting a car's engine to the chassis, another team can connect the transmission to the engine and the driveshaft of a car that's already been processed by the first team, and a third team can bolt the body to the chassis in a car that's been processed by the first two teams.
Example : suppose we want to add the floating point numbers 9.87 × 10^4 and 6.54 × 10^3 . Then we can use the following steps:

Time	Operation	Operand 1	Operand 2	Result
0	Fetch operands	9.87×10^4	6.54×10^3	
1	Compare exponents	9.87×10^4	6.54×10^3	
2	Shift one operand	9.87×10^4	0.654×10^4	
3	Add	9.87×10^4	0.654×10^4	10.524×10^4
4	Normalize result	9.87×10^4	0.654×10^4	1.0524×10^5
5	Round result	9.87×10^4	0.654×10^4	1.05×10^5
6	Store result	9.87×10^4	0.654×10^4	1.05×10^5

• **Speculation** ➢ In speculation, the compiler or the processor makes a guess about an instruction, and then executes the instruction on the basis of the guess
Example : in the following code, the system might predict that the outcome of z = x + y will give z a positive value, and, as a consequence, it will assign w = x.

```
z = x + y;
if (z > 0)
    w = x;
else
    w = y;
```

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

| coarser-grained parallelism than ILP, that is, the program units that are being simultaneously executed—threads—are larger or coarser than the **finer-grained** units—individual
• **coarse-grained multithreading** ➢ attempts to avoid this problem by only switching threads that are stalled waiting for a time-consuming operation to complete (e.g., a load from main memory)
• **fine-grained multithreading** ➢ the processor switches between threads after each instruction, skipping threads that are stalled.

10. Describe SIMD system.

➢ [Single instruction, multiple data] are parallel systems , operate on multiple data streams by applying the same instruction to multiple data items, so an abstract SIMD system can be thought of as having a single control unit and multiple ALUs .

11. 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]

12. Describe the MIMD systems .

➢ [Multiple instruction, multiple data] : systems support multiple simultaneous instruction streams operating on multiple data streams , consist of a collection of fully independent processing units or cores, each of which has its own control unit and its own ALU .

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

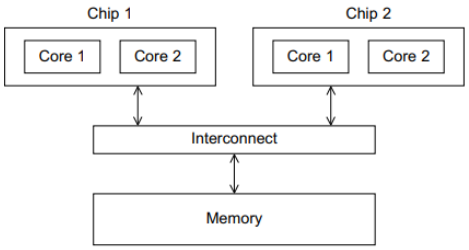


FIGURE 2.5

A UMA multicore system

- **UMA** (uniform memory access)systems are usually easier to program, since the programmer doesn't need to worry about different access times for different memory locations.

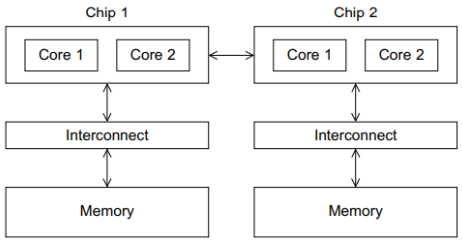
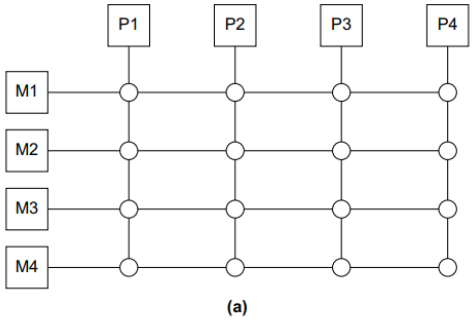


FIGURE 2.6

A NUMA multicore system

- **NUMA** (nonuniform memory access) systems have the potential to use larger amounts of memory than UMA systems.

14. Describe the crossbar as interconnect for shared memory system ?



(a)

- The lines are bidirectional communication links, the squares are cores or memory modules, and the circles are switches.

15. Draw ring, toroidal mesh, fully connected network, hypercube, crossbar, and omega network for eight processors and compute the bisection width for each interconnect.

is bisection width. To understand this measure, imagine that the parallel system is

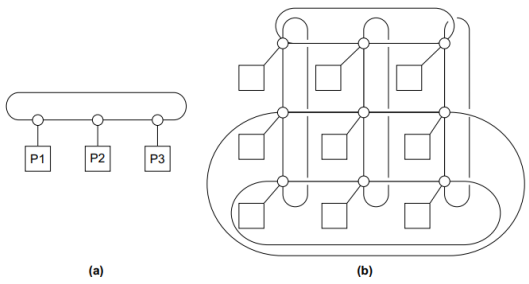


FIGURE 2.8
(a) A ring and (b) a toroidal mesh

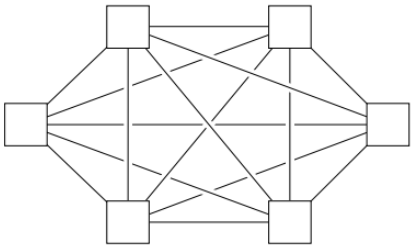


FIGURE 2.11
A fully connected network

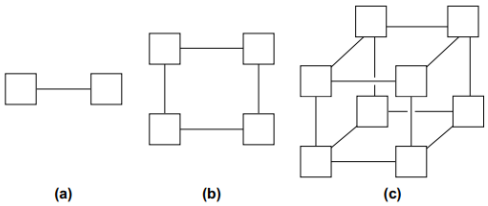


FIGURE 2.12
(a) One-, (b) two-, and (c) three-dimensional hypercubes

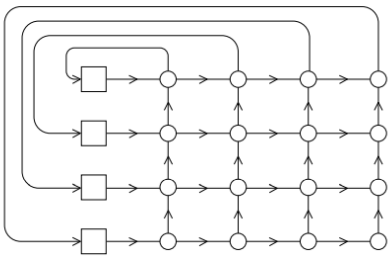


FIGURE 2.14
A crossbar interconnect for distributed-memory

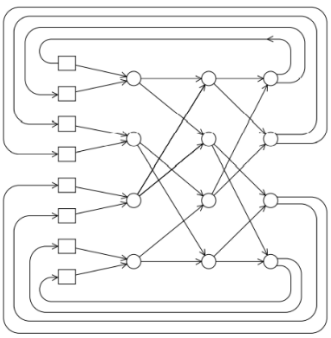


FIGURE 2.15
An omega network

- The bisection width of a p × p crossbar is p and the bisection width of an omega network is p/2

16. Show the difference between crossbar and omega network switches.

➢ ❶ **In the crossbar** as long as two processors don't attempt to communicate with the same processor, all the processors can simultaneously communicate with another processor. ❷ **in omega** network The switches are two-by-two crossbars, Observe that unlike the crossbar, there are communications that cannot occur simultaneously

17. Describe the latency concept.

➢ is the time that elapses between the source's beginning to transmit the data and the destination's starting to receive the first byte

18. Describe with example how cache coherence problem can happen?

➢ the caches we described for single processor systems provide no mechanism for insuring that when the caches of multiple processors store the same variable, an update by one processor to the cached variable is "seen" by the other processors. That is, that the cached value stored by the other processors is also updated

19. Explain the concepts: snooping cache coherence, directory-based cache coherence, and false sharing ?

• **Snooping cache coherence** ➢ The idea behind snooping comes from bus-based systems : When the cores share a bus, any signal transmitted on the bus can be "seen" by all the cores connected to the bus. snooping cache coherence isn't scalable.

snooping cache coherence is clearly a problem since a broadcast across the interconnect will be very slow relative to the speed of accessing local memory.

• **Directory-based cache coherence** ➢ Solve a problem of (broadcast across the interconnect will be very slow relative to the speed of accessing local memory) through the use data structure called a directory. The directory stores the status of each cache line. Typically, this data structure is distributed.

• **False sharing** ➢ does not cause incorrect results. However, it can ruin the performance of a program by causing many more accesses to memory than necessary

20. Which approach to use: distributed memory or shared memory ?

➢ distributed-memory interconnects such as the hypercube and the toroidal mesh are relatively inexpensive, and distributed-memory systems with thousands of processors that use these and other interconnects have been built. Thus, distributed-memory systems are often better suited for problems requiring vast amounts of data or computation.

21. Give the structure of SPMD programs.

➢ (single program multiple data) Instead of running a different program on each core, SPMD programs consist of a single executable that can behave as if it were multiple different programs through the use of conditional branches.

For example,

```
 if (I'm thread/process 0)
       do this;
else
      do that;
```

Observe that SPMD programs can readily implement data-parallelism.

For example ,

```
 if (I'm thread/process 0)
     operate on the first half of the array;
else /* I'm thread/process 1 */
     operate on the second half of the array;
```

22. Describe the difference between static and dynamic threads ?

• **Static threads** ➢ In this paradigm, all of the threads are forked after any needed setup by the master thread and the threads run until all the work is completed

• **Dynamic threads** ➢ In this paradigm, there is often a master thread and at any given instant a (possibly empty) collection of worker threads. The master thread typically waits for work requests

23. Give examples for nondeterminism when processors execute asynchronously.

➢ suppose we have two threads, one with id or rank 0 and the other with id or rank 1. Suppose also that each is storing a private variable my x, thread 0's value for my x is 7, and thread 1's is 19. Further, suppose both threads execute the following

CODE :

```
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

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

➢ **(mutual exclusion lock or mutex or lock)** is a special type of object that has support in the underlying hardware. 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 In busy-waiting, a thread enters a loop whose sole purpose is to test a condition.

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

➢ the 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 through the use of 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.

26. 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 process0");
Send(message, MSG CHAR, 100, 0);
} else if (my rank == 1) {
Receive(message, MSG CHAR, 100, 1);
printf("Process 1 > Received: %snn", message);
}
```

27. Give examples of functions for collective communication.

- MPI_Reduce() { Reduction (all to one) }
- MPI_Allreduce() { Reduction (all to all) }
- MPI_Bcast() { Broadcast (one to all) }
- MPI_Scatter() { Distribute data (one to all) }
- MPI_Gather() { Collect data (all to one) }
- MPI_Allgather() { Collect data (all to all) }

28. What one sided communication means ?

➢ **In one-sided communication, or remote memory access**, a single process calls a function, which updates either local memory with a value from another process or remote memory with a value from the calling process

29. How remote memory access affects the program performance ?

➢ This can simplify communication , since it only requires the active participation of a single process. Furthermore, it can significantly reduce the cost of communication by eliminating the overhead associated with synchronizing two processes . It can also reduce overhead by eliminating the overhead of one of the function calls (send or receive)

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

➢ ❶ In distributed-memory programs, only process 0 will access stdin. In shared memory programs, only the master thread or thread 0 will access stdin ❷ In both distributed-memory and shared-memory programs, all the processes/ threads can access std out and stderr ❸ Only a single process/thread will attempt to access any single file other than stdin, std out, or stderr

31. Explain the concepts speed up and efficiency.

• **speed up** ➢ is a metric that quantifies performance by comparing the execution of a serial algorithm and a parallelized version of the same algorithm

• **efficiency** ➢ is a metric that builds on top of speedup by adding awareness of the underlying hardware

32. What is meant by Amdahl's low?

➢ It says, roughly, that unless virtually all of a serial program is parallelized, the possible speedup is going to be very limited—regardless of the number of cores available.

33. When a program is weakly or strongly scalable ?

➢ If when we increase the number of processes/threads , we can keep the efficiency fixed without increasing the problem size, the program is said to be **strongly scalable .**

➢ If we can keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, then the program is said to be **weakly scalable**

34. How running time of serial and parallel programs can be computed ?

35. Outline the foster's methodology .

➢ **Partitioning** : Divide the computation to be performed and the data operated on by the computation into small tasks.

➢ **Communication** : Determine what communication needs to be carried out

➢ **Agglomeration** or aggregation : Combine tasks and communications identified in the first step into larger tasks

➢ **Mapping** : Assign the composite tasks identified in the previous step to processes/ threads

36. Apply the foster's methodology for making histogram out of data.

BIG Question Answer in pages (66,67,68,69,70) of chapter2 , with title(2.7.1 An example)