Chapter 1: Parallel Computer Model

Introduction

From an application point of view, the mainstream of usage of computer is experiencing a trend of four ascending levels of sophistication:

- Data processing
- Information processing
- Knowledge processing
- Intelligence processing

With more and more data structures developed, many users are shifting to computer roles from pure data processing to information processing. A high degree of parallelism has been found at these levels. As the accumulated knowledge bases expanded rapidly in recent years, there grew a strong demand to use computers for knowledge processing. Intelligence is very difficult to create; its processing even more so. Todays computers are very fast and obedient and have many reliable memory cells to be qualified for data-information-knowledge processing.

Parallel processing is emerging as one of the key technology in area of modern computers. Parallel appears in various forms such as lookahead, vectorization concurrency, simultaneity, data parallelism, interleaving, overlapping, multiplicity, replication, multiprogramming, multithreading and distributed computing at different processing level.

1.1 The state of computing

Modern computers are equipped with powerful hardware technology at the same time loaded with sophisticated software packages. To access the art of computing we firstly review the history of computers then study the attributes used for analysis of performance of computers.

Computer Development Milestones

How it all started…

- 500 BC: Abacus (China) – The earliest mechanical computer/calculating device.
    - Operated to perform decimal arithmetic with carry propagation digit by digit

- 1642: Mechanical Adder/Subtractor (Blaise Pascal)
- 1827: Difference Engine (Charles Babbage)
- 1941: First binary mechanical computer (Konrad Zuse; Germany)
- 1944: Harvard Mark I (IBM)
  - The very first electromechanical decimal computer as proposed by Howard Aiken

Computer Generations

- 1st 2nd 3rd 4th 5th
- Division into generations marked primarily by changes in hardware and software technologies

Evolution of computer system

Presently the technology involved in designing of its hardware components of computers and its overall architecture is changing very rapidly for example: processor clock rate increase about 20% a year, its logic capacity improve at about 30% in a year; memory speed at increase about 10% in a year and memory capacity at about 60% increase a year also the disk capacity increase at a 60% a year and so overall cost per bit improves about 25% a year.

But before we go further with design and organization issues of parallel computer architecture it is necessary to understand how computers had evolved. Initially, man used simple mechanical devices – abacus (about 500 BC) , knotted string, and the slide rule for computation. Early computing was entirely mechanical like : mechanical adder/subtracter (Pascal, 1642) difference engine design (Babbage, 1827) binary mechanical computer (Zuse, 1941) electromechanical decimal machine (Aiken, 1944). Some of these machines used the idea of a stored program a famous example of it is the Jacquard Loom and Babbage's Analytical Engine which is also often considered as the first real computer. Mechanical and electromechanical machines have limited speed and reliability because of the many moving parts. Modern machines use electronics for most information transmission.

Computing is normally thought of as being divided into generations. Each successive generation is marked by sharp changes in hardware and software technologies. With some exceptions, most of the advances introduced in one generation are carried through to later generations. We are currently in the fifth generation.

First generation of computers (1945- 54)

- Technology & Architecture:
  - Vacuum Tubes
  - Relay Memories
  - CPU driven by PC and accumulator
  - Fixed Point Arithmetic
- Software and Applications:
  - Machine/Assembly Languages
  - Single user
  - No subroutine linkage
  - Programmed I/O using CPU
- Representative Systems:
  - ENIAC, Princeton IAS, IBM 701

Second generation of computers (1954 – 64)

- Technology & Architecture:
  - Discrete Transistors
  - Core Memories
  - Floating Point Arithmetic
  - I/O Processors
  - Multiplexed memory access
- Software and Applications:
  - High level languages used with compilers
  - Subroutine libraries
  - Processing Monitor
- Representative Systems:
  - IBM 7090, CDC 1604, Univac LARC

Third Generation computers *(1965 to 1974)*

- Technology & Architecture:
  - IC Chips (SSI/MSI)
  - Microprogramming
  - Pipelining
  - Cache
  - Look-ahead processors
- Software and Applications:
  - Multiprogramming and Timesharing OS
  - Multiuser applications
- Representative Systems:
  - IBM 360/370, CDC 6600, T1-ASC, PDP-8

Fourth Generation computer (*(1975 to 1990)*

- Technology & Architecture:

    - LSI/VLSI
    - Semiconductor memories
    - Multiprocessors
    - Multi-computers
    - Vector supercomputers
- Software and Applications:
    - Multiprocessor OS
    - Languages, Compilers and environment for parallel processing
- Representative Systems:
    - VAX 9000, Cray X-MP, IBM 3090

Fifth Generation computers(1991 - Present)

- Technology & Architecture:
    - Advanced VLSI processors
    - Scalable Architectures
    - Superscalar processors
- Software and Applications:
    - Systems on a chip
    - Massively parallel processing
    - Grand challenge applications
    - Heterogeneous processing
- Representative Systems:
    - S-81, IBM ES/9000, Intel Paragon, nCUBE 6480, MPP, VPP500
    - Fujitsu VPP500, Cray MPP, TMC CM-5, Intel Paragon

Elements of Modern Computers

The hardware, software, and programming elements of modern computer systems can be characterized by looking at a variety of factors in context of parallel computing these factors are:

- Computing problems
- Algorithms and data structures
- Hardware resources
- Operating systems
- System software support
- Compiler support

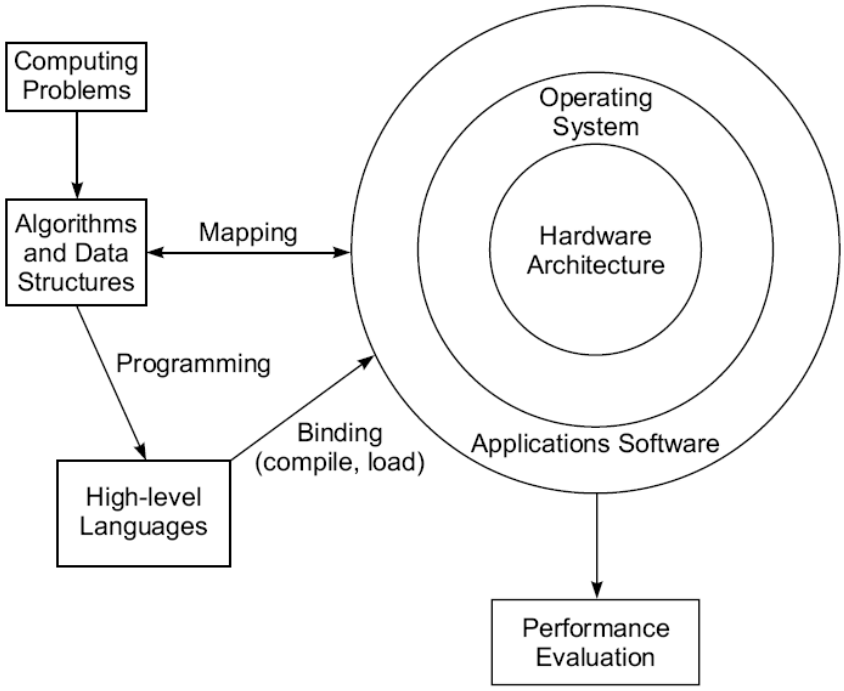


Fig. 1.1 Elements of a modern computer system

Computing Problems

- Numerical computing complex mathematical formulations tedious integer or floating -point computation
- Transaction processing accurate transactions large database management information retrieval
- Logical Reasoning logic inferences symbolic manipulations

Algorithms and Data Structures

- Traditional algorithms and data structures are designed for sequential machines.
- New, specialized algorithms and data structures are needed to exploit the capabilities of parallel architectures.
- These often require interdisciplinary interactions among theoreticians, experimentalists, and programmers.

Hardware Resources

- The architecture of a system is shaped only partly by the hardware resources.
- The operating system and applications also significantly influence the overall architecture.
- Not only must the processor and memory architectures be considered, but also the

architecture of the device interfaces (which often include their advanced processors).

Operating System

- Operating systems manage the allocation and deallocation of resources during user program execution.
- UNIX, Mach, and OSF/1 provide support for multiprocessors and multicomputers
- multithreaded kernel functions virtual memory management file subsystems network communication services
- An OS plays a significant role in mapping hardware resources to algorithmic and data structures.

System Software Support

- Compilers, assemblers, and loaders are traditional tools for developing programs in high-level languages. With the operating system, these tools determine the bind of resources to applications, and the effectiveness of this determines the efficiency of hardware utilization and the system's programmability.
- Most programmers still employ a sequential mind set, abetted by a lack of popular parallel software support.
- Parallel software can be developed using entirely new languages designed specifically with parallel support as its goal, or by using extensions to existing sequential languages.
- New languages have obvious advantages (like new constructs specifically for parallelism), but require additional programmer education and system software.
- The most common approach is to extend an existing language.

Compiler Support

- Preprocessors use existing sequential compilers and specialized libraries to implement parallel constructs.
- Precompilers perform some program flow analysis, dependence checking, and limited parallel optimizations.
- Parallelizing Compilers requires full detection of parallelism in source code, and transformation of sequential code into parallel constructs.
- Compiler directives are often inserted into source code to aid compiler parallelizing efforts.

Evolution of Computer Architecture

In last four decades, computer architecture has gone through revolutionary changes. We started with Von Neumann architecture and now we have multicomputers and multiprocessors.

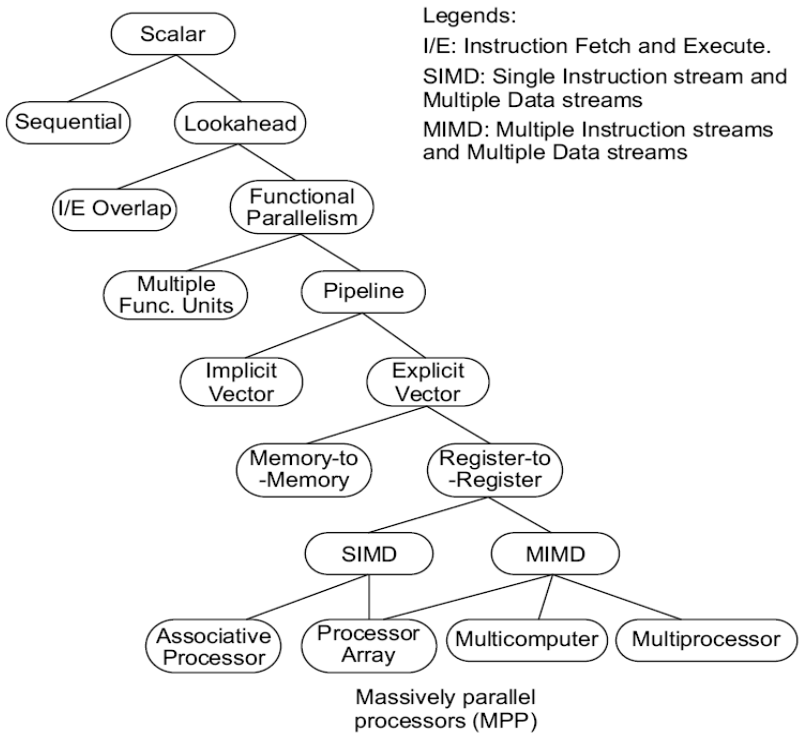


Fig. 1.2 Tree showing architectural evolution from sequential scalar computers to vector processors and parallel computers

- The study of computer architecture involves both the following:
  - Hardware organization
  - Programming/software requirements
- The evolution of computer architecture is believed to have started with von Neumann architecture
  - Built as a sequential machine
  - Executing scalar data
- Major leaps in this context came as…
  - Look-ahead, parallelism and pipelining
  - Flynn's classification
  - Parallel/Vector Computers
  - Development Layers

Flynn's Classification

Michael Flynn (1972) introduced a classification of various computer architectures based on notions of instruction and data streams as shown in the fig 1.3.

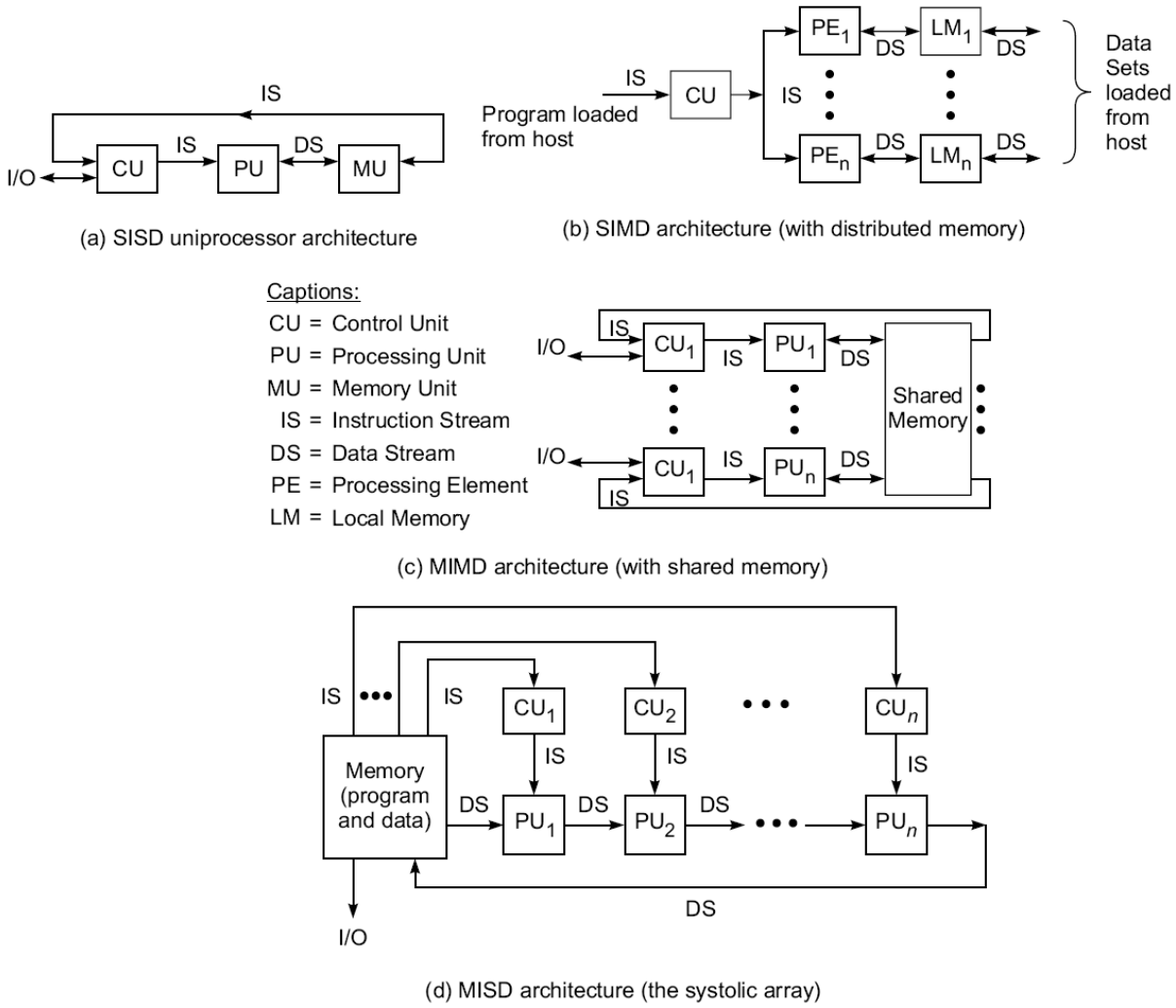


Fig. 1.3 Flynn's classification of computer architectures (Derived from Michael Flynn, 1972)

An instruction stream is sequence of instructions executed by machine. And a data stream is a sequence of data including input, partial or temporary results used by instruction stream. Each of these dimensions can have only one of two possible states: *Single* or *Multiple*. Flynn's classification depends on the distinction between the performance of control unit and the data processing unit rather than its operational and structural interconnections. Following are the four category of Flynn classification and characteristic feature of each of them.

1. Single instruction stream, single data stream (SISD)

The figure 1.3 (a) is represents an organization of simple SISD computer having one control unit, one processor unit and single memory unit.

- They are also called scalar processor i.e., one instruction at a time and each instruction have only one set of operands.

- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- Instructions are executed sequentially.
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes

b) Single instruction stream, multiple data stream (SIMD) processors

- A type of parallel computer
- Single instruction: All processing units execute the same instruction issued by the control unit at any given clock cycle where there are multiple processor executing instruction given by one control unit.
- Multiple data: Each processing unit can operate on a different data element as shown if figure below the processor are connected to shared memory or interconnection network providing multiple data to processing unit

SIMD processor organization

- This type of machine typically has an instruction dispatcher, a very high-bandwidth internal network, and a very large array of very small-capacity instruction units.
- Thus single instruction is executed by different processing unit on different set of data as shown in figure 1.3.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing and vector computation.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays e.g., Connection Machine CM-2, Maspar MP-1, MP-2 and Vector Pipelines processor e.g., IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

c) Multiple instruction stream, multiple data stream (MIMD)

- Multiple Instruction: every processor may be executing a different instruction stream

- Multiple Data: every processor may be working with a different data stream as shown in figure 1.3 multiple data stream is provided by shared memory.
- Can be categorized as loosely coupled or tightly coupled depending on sharing of data and control
- Execution can be synchronous or asynchronous, deterministic or non-deterministic

MIMD processor organizations

- As shown in figure 1.3 there are different processor each processing different task.
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.

d) Multiple instruction stream, single data stream (MISD)

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams as shown in figure 1.3 a single data stream is forwarded to different processing unit which are connected to different control unit and execute instruction given to it by control unit to which it is attached.

MISD processor organization

- Thus in these computers same data flow through a linear array of processors executing different instruction streams as shown in figure 1.3.
- This architecture is also known as systolic arrays for pipelined execution of specific instructions.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).
- Some conceivable uses might be:

1. Multiple frequency filters operating on a single signal stream
2. Multiple cryptography algorithms attempting to crack a single coded message.

Here the some popular computer architecture and there types

SISD IBM 701, IBM 1620, IBM 7090, PDP VAX11/ 780

SISD (With multiple functional units) IBM360/91 (3); IBM 370/168 UP

SIMD (Word Slice Processing) Illiac – IV ; PEPE

SIMD (Bit Slice processing) STARAN; MPP; DAP

MIMD (Loosely Coupled) IBM 370/168 MP; Univac 1100/80

MIMD(Tightly Coupled) Burroughs- D – 825

Parallel / Vector Computers

Intrinsic parallel computers are those that execute program in MIMD mode. There are two major classes:

(1) Shared memory multiprocessor and
(2) Message passing multicomputers.

The processor in multiprocessor communicates with each other through shared variables in a common memory. Each computer node in a multicomputer system has a local memory, unshared with other nodes and interprocessor communication is done through message passing among the nodes.
Explicit vector instructions were introduced with the appearance of vector processors and equipped with multiple vector pipelines that can be concurrently used. Two families of pipelined vector processors:

(1)Memory-to-memory architecture supports the pipelined flow of vector operands directly from the memory to pipeline and then back to memory.

(2) Register-to-register architecture uses vector registers to interface between the memory and functional pipelines.

Development Layers

A layered development of parallel computer is illustrated in Fig 1.4. based on a classification by Lionel Ni(1990)

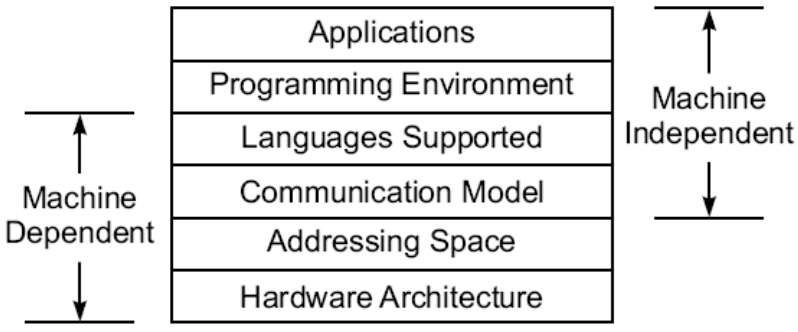


Fig. 1.4 Six layers for computer system development (Courtesy of Lionel Ni, 1990)

New Challenges

The technology of parallel processing is the outgrowth of several decades of research and industrial advances in microelectronics, printed circuits, high density packaging, advanced processors, memory systems, peripheral devices, communication channels, language evaluation, compiler sophistication, operating systems, programming environments, and application challenges.

PERFORMANCE ATTRIBUTES

Performance of a system depends on

- Hardware technology
- Architectural features
- Efficient resource management
- Algorithm design
- Data structures
- Language efficiency
- Programmer skill
- Compiler technology

When we talk about performance of computer system we would describe how quickly a given system can execute a program or programs. Thus we are interested in knowing the turnaround time. Turnaround time depends on:

- Disk and memory accesses
- Input and output
- Compilation time
- Operating system overhead
- CPU time

An ideal performance of a computer system means a perfect match between the machine capability and program behavior. The machine capability can be improved by using better hardware technology and efficient resource management. But as far as program behavior is concerned it depends on code used, compiler used and other run time conditions. Also a machine performance may vary from program to program. Because there are too many programs and it is impractical to test a CPU's speed on all of them, benchmarks were developed. Computer architects have come up with a variety of metrics to describe the computer performance.

Clock rate and CPI / IPC

Since I/O and system overhead frequently overlaps processing by other programs, it is fair to consider only the CPU time used by a program, and the user CPU time is the most important factor. CPU is driven by a clock with a constant cycle time (usually measured in nanoseconds, which controls the rate of internal operations in the CPU. The clock mostly has the constant cycle time (t in nanoseconds). The inverse of the cycle time is the clock rate ($f = 1/\tau$, measured in megahertz). A shorter clock cycle time, or equivalently a larger number of cycles per second, implies more operations can be performed per unit time. The size of the program is determined by the instruction count (Ic). The size of a program is determined by its instruction count, I*c*, the number of machine instructions to be executed by the program. Different machine instructions require different numbers of clock cycles to execute. CPI (cycles per instruction) is thus an important parameter.

Average CPI

It is easy to determine the average number of cycles per instruction for a particular processor if we know the frequency of occurrence of each instruction type.

Of course, any estimate is valid only for a specific set of programs (which defines the instruction mix), and then only if there are sufficiently large number of instructions.

In general, the term CPI is used with respect to a particular instruction set and a given program mix. The time required to execute a program containing I*c* instructions is just

$T = Ic * CPI * \tau$.

Each instruction must be fetched from memory, decoded, then operands fetched from memory, the instruction executed, and the results stored.

The time required to access memory is called the memory cycle time, which is usually k times the processor cycle time τ. The value of k depends on the memory technology and the processor-memory interconnection scheme. The processor cycles required for each instruction (CPI) can be attributed to cycles needed for instruction decode and execution (*p*), and cycles needed for memory references (*m* k*).

The total time needed to execute a program can then be rewritten as

$T = Ic * (p + m*k)*\tau$.

System Attributes

The five attributes (I*c* ,p, m, k, τ) are influenced by four system attributes:

Instruction set architecture, Compiler technology, CPU implementation and control and cache and memory hierarchy as shown below

FACTORS	I_C	p	m	k	t
Instruction set architecture.	X	X			
Compiler technology.	X	X	X		
CPU implementation & control		X			X
Cache & memory hierarchy				X	X

- **The instruction set architecture affects program length and p.**
- **Compiler design affects the values of I_C, p & m.**
- **The CPU implementation & control determine the total processor time= p*t**
- **The memory technology & hierarchy design affect the memory access time= k*t**

MIPS: The *millions of instructions per second*, this is calculated by dividing the number of instructions executed in a running program by time required to run the program. The MIPS rate is directly proportional to the clock rate and inversely proportion to the CPI. All four systems attributes (instruction set, compiler, processor, and memory technologies) affect the MIPS rate, which varies also from program to program. MIPS does not proved to be effective as it does not account for the fact that different systems often require different number of instruction to implement the program. It does not inform about how many instructions are required to perform a given task. With the variation in instruction styles, internal organization, and number of processors per system it is almost meaningless for comparing two systems.

MFLOPS (pronounced ``megaflops'') stands for ``millions of floating point operations per second.'' This is often used as a ``bottom-line'' figure. If one know ahead of time how many operations a program needs to perform, one can divide the number of operations by the execution time to come up with a MFLOPS rating. For example, the standard algorithm for multiplying **n*n** matrices requires $\mathbf{2n^3 - n}$ operations (n^2 inner products, with **n** multiplications and **n-1**additions in each product). Suppose you compute the product of two 100 *100 matrices in 0.35 seconds. Then the computer achieves

$(2(100)^3 - 100)/0.35 = 5{,}714{,}000$ ops/sec = 5.714 MFLOPS

The term ``theoretical peak MFLOPS'' refers to how many operations per second would be possible if the machine did nothing but numerical operations. It is obtained by calculating the time it takes to perform one operation and then computing how many of them could be done in one second. For example, if it takes 8 cycles to do one floating point multiplication, the cycle time on the machine is 20 nanoseconds, and arithmetic operations are not overlapped with one another, it takes 160ns for one multiplication, and (1,000,000,000 nanosecond/1sec)*(1 multiplication / 160 nanosecond) = $6.25*10^6$ multiplication /sec so the theoretical peak performance is 6.25 MFLOPS. Of course, programs are not just long sequences of multiply and add instructions, so a machine rarely comes close to this level of performance on any real program. Most machines will achieve less than 10% of their peak rating, but vector processors or other machines with internal pipelines that have an effective CPI near 1.0 can often achieve 70% or more of their theoretical peak on small programs.

Throughput rate : Another important factor on which system's performance is measured is throughput of the system which is basically how many programs a system can execute per unit time Ws. In multiprogramming the system throughput is often lower than the CPU throughput Wp which is defined as

Wp = f/(Ic * CPI)

Unit of Wp is programs/second.

Ws <Wp as in multiprogramming environment there is always additional overheads like timesharing operating system etc. An Ideal behavior is not achieved in parallel computers because while executing a parallel algorithm, the processing elements cannot devote 100% of their time to the computations of the algorithm. Efficiency is a measure of the fraction of time for which a PE is usefully employed. In an ideal parallel system efficiency is equal to one. In practice, efficiency is between zero and ones of overhead associated with parallel execution

Speed or Throughput (W/Tn) - the execution rate on an n processor system, measured in FLOPs/unit-time or instructions/unit-time.

Speedup (Sn = T1/Tn) - how much faster in an actual machine, n processors compared to 1 will perform the workload. The ratio T1/T∞is called the *asymptotic speedup*.

Efficiency (En = Sn/n) - fraction of the theoretical maximum speedup achieved by n processors.

Degree of Parallelism (DOP) - for a given piece of the workload, the number of processors that can be kept busy sharing that piece of computation equally. Neglecting overhead, we assume that if k processors work together on any workload, the workload gets done k times as fast as a sequential execution.

Scalability - The attributes of a computer system which allow it to be gracefully and linearly scaled up or down in size, to handle smaller or larger workloads, or to obtain proportional decreases or increase in speed on a given application. The applications run on a scalable machine may not scale well. Good scalability requires the algorithm *and* the machine to have the right properties

Thus in general there are five performance factors (I*c*, p, m, k, t) which are influenced by four system attributes:

- instruction-set architecture (affects *Ic* and *p*)
- compiler technology (affects *Ic* and *p* and *m*)
- CPU implementation and control (affects *p *t*) cache and memory hierarchy (affects memory access latency, *k ´t*)
- Total CPU time can be used as a basis in estimating the execution rate of a processor.

Programming Environments

Programmability depends on the programming environment provided to the users.

Conventional computers are used in a sequential programming environment with tools developed for a uniprocessor computer. Parallel computers need parallel tools that allow specification or easy detection of parallelism and operating systems that can perform parallel scheduling of concurrent events, shared memory allocation, and shared peripheral and communication links.

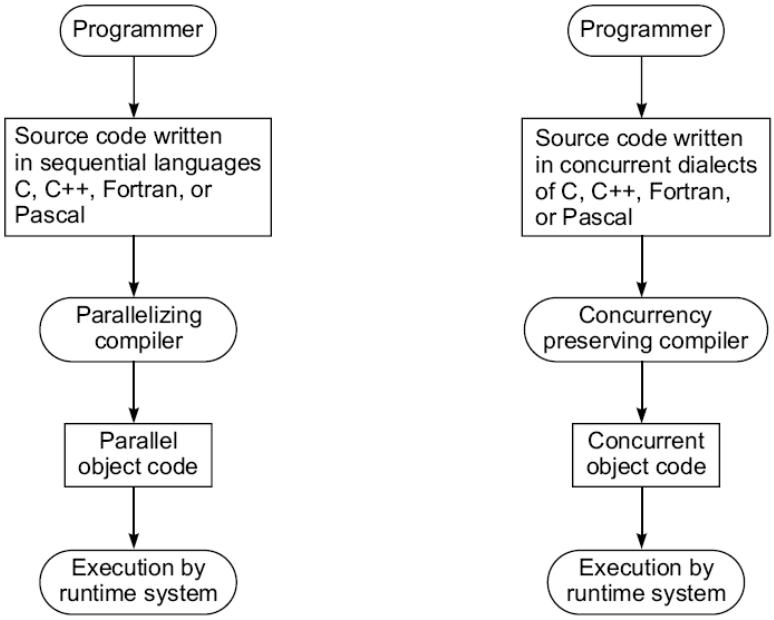


Fig. 1.5 Two approaches to parallel programming (Courtesy of Charles Seitz; adapted with permission from "Concurrent Architectures", p. 51 and p. 53, *VLSI and Parallel Computation*, edited by Suaya and Birtwistle, Morgan Kaufmann Publishers, 1990)

Implicit Parallelism

Use a conventional language (like C, Fortran, Lisp, or Pascal) to write the program.

Use a parallelizing compiler to translate the source code into parallel code.

The compiler must detect parallelism and assign target machine resources.

Success relies heavily on the quality of the compiler.

Explicit Parallelism

Programmer writes explicit parallel code using parallel dialects of common languages.

Compiler has reduced need to detect parallelism, but must still preserve existing parallelism and assign target machine resources.

Needed Software Tools

Parallel extensions of conventional high-level languages.

Integrated environments to provide different levels of program abstraction validation, testing and debugging performance prediction and monitoring visualization support to aid program development, performance measurement graphics display and animation of computational results

1.2 MULTIPROCESSOR AND MULTICOMPUTERS

Two categories of parallel computers are discussed below namely shared common memory or unshared distributed memory.

Shared memory multiprocessors

- Shared memory parallel computers vary widely, but generally have in common the ability for all processors to access all memory as global address space.
- Multiple processors can operate independently but share the same memory resources.
- Changes in a memory location effected by one processor are visible to all other processors.
- Shared memory machines can be divided into two main classes based upon memory access times: ***UMA*** , ***NUMA and COMA***.

Uniform Memory Access (UMA):

- Most commonly represented today by Symmetric Multiprocessor (SMP) machines
- Identical processors
- Equal access and access times to memory
- Sometimes called CC-UMA - Cache Coherent UMA. Cache coherent means if one processor updates a location in shared memory, all the other processors know about the update. Cache coherency is accomplished at the hardware level.

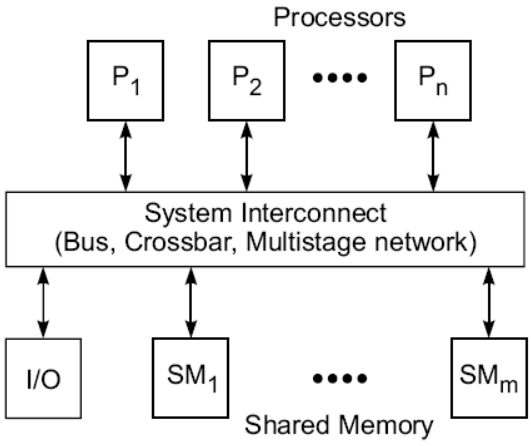


Fig. 1.6 The UMA multiprocessor model

Non-Uniform Memory Access (NUMA):

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower

If cache coherency is maintained, then may also be called CC-NUMA - Cache Coherent NUMA

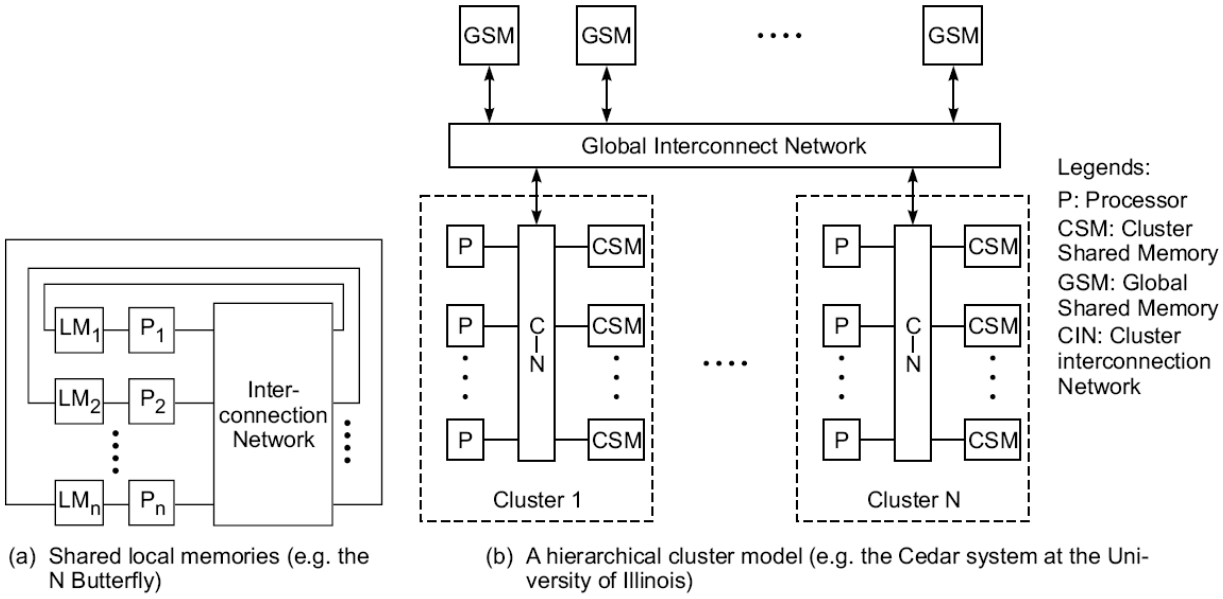


Fig. 1.7 Two NUMA models for multiprocessor systems

The COMA model : The COMA model is a special case of NUMA machine in which the distributed main memories are converted to caches. All caches form a global address space and there is no memory hierarchy at each processor node.

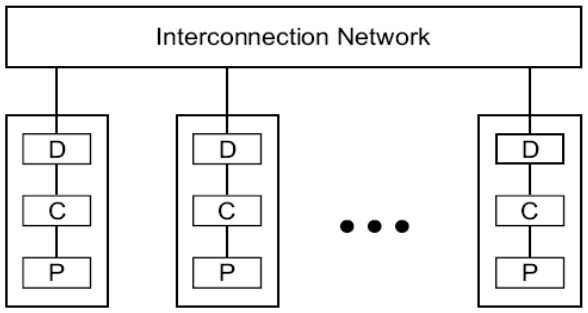


Fig. 1.8 The COMA model of a multiprocessor (P: Processor, C: Cache, D: Directory; e.g. the KSR-1)

Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
- Programmer responsibility for synchronization constructs that insure "correct" access of global memory.
- Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors.

Distributed Memory

- Like shared memory systems, distributed memory systems vary widely but share a common characteristic. Distributed memory systems require a communication network to connect inter-processor memory.

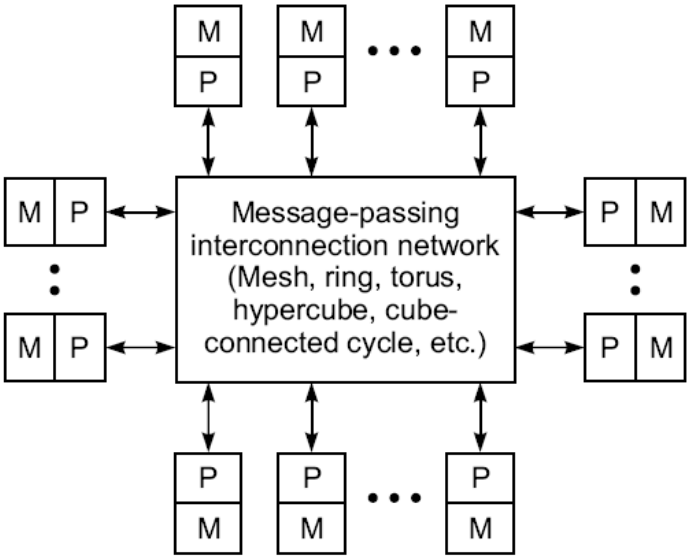


Fig. 1.9 Generic model of a message-passing multicomputer

- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Because each processor has its own local memory, it operates independently. Changes it makes to its local memory have no effect on the memory of other processors. Hence, the concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated. Synchronization between tasks is likewise the programmer's responsibility.
- Modern multicomputer use hardware routers to pass message. Based on the interconnection and routers and channel used the multicomputers are divided into generation
  - 1st generation : based on board technology using hypercube architecture and software controlled message switching.
  - 2nd Generation: implemented with mesh connected architecture, hardware message routing and software environment for medium distributed – grained computing.
  - 3rd Generation : fine grained multicomputer like MIT J-Machine.
- The network "fabric" used for data transfer varies widely, though it can be as simple as Ethernet.

Advantages:

- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

Disadvantages:

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to

this memory organization.

- Non-uniform memory access (NUMA) times

Taxonomy of MIMD Computers

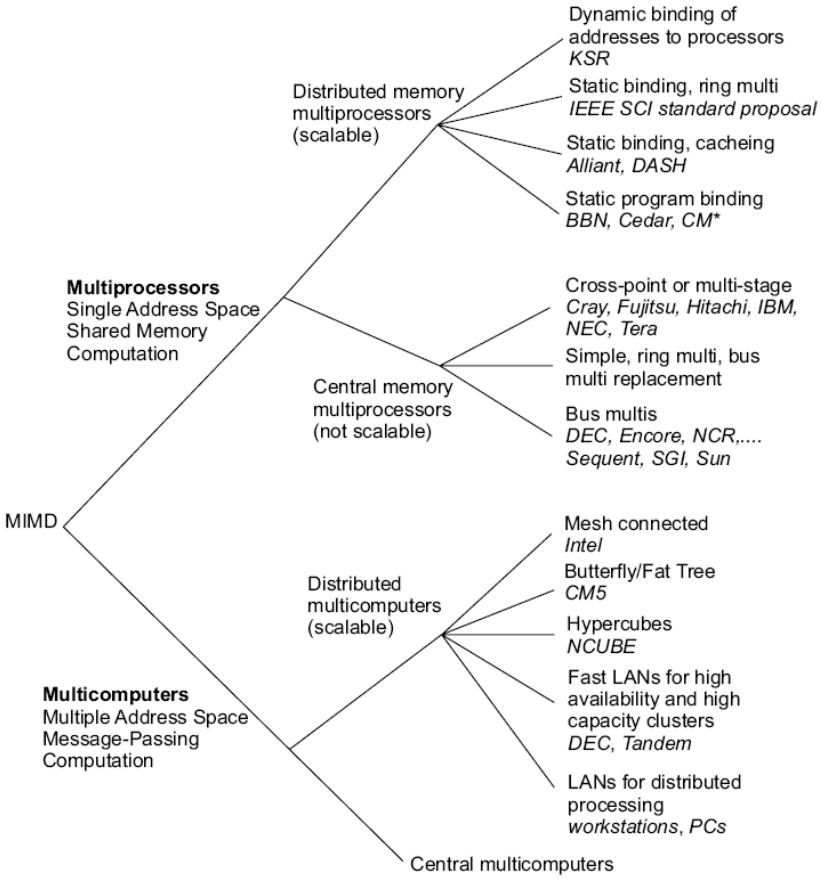


Fig. 1.10 Bell's taxonomy of MIMD computers (Courtesy of Gordon Bell; reprinted with permission from the *Communications of ACM*, August 1992)

1.3 MULTIVECTOR AND SIMD COMPUTERS

In this section, we introduce supercomputers and parallel processors for vector processing and data parallelism. We classify supercomputers either as pipelined vector machines using a few powerful processors equipped with vector hardware, or as SIMD computers emphasizing massive data parallelism.

Vector Supercomputers

A vector computer is often built on top of a scalar processor. As shown in Fig. 1.11, the vector processor is attached to the scalar processor as an optional feature. Program and data are first loaded into the main memory through a host computer. All instructions are first decoded by the scalar control unit. If the decoded instruction is a scalar operation or a program control operation, it will be directly executed by the scalar processor using the scalar functional pipelines.

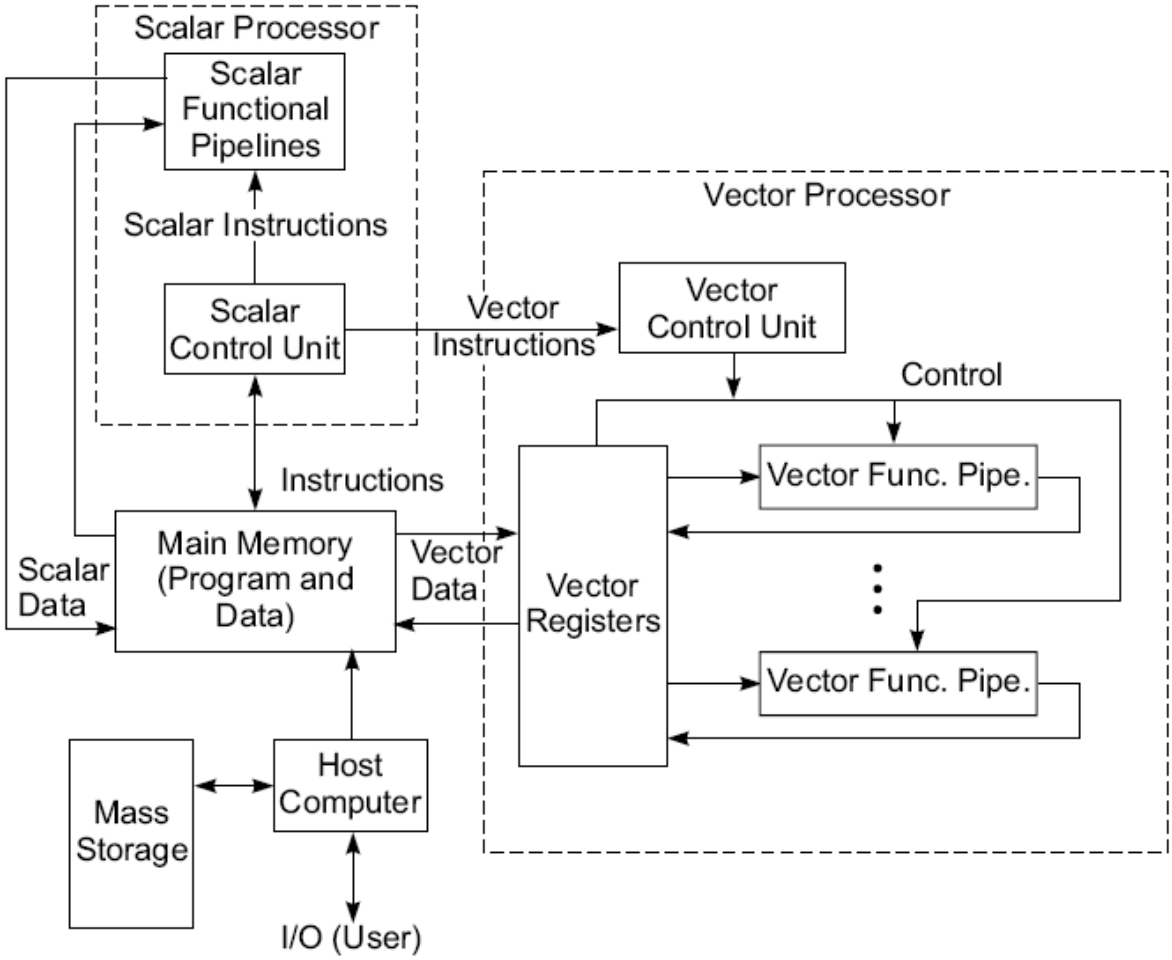


Fig. 1.11 The architecture of a vector supercomputer

Vector Processor Models:

Figure l .l l shows a register-to-register architecture. Vector registers are used to hold the vector operands, intermediate and final vector results. The vector functional pipelines retrieve operands from and put results into the vector registers. All vector registers are programmable in user instructions. Each vector register is equipped with a component counter which keeps track of the component registers used in successive pipeline cycles.

Representative Supercomputer:

Over a dozen pipelined vector computers have been manufactured, ranging from workstations to mini- and supercomputers. Notable early examples include the Stardent 3000 multiprocessor equipped with vector pipelines, the Convex C3 Series, the DEC VAX 9000, the IBM 390/VF, the Cray Research Y-MP family, the NEC SX Series, the Fujitsu VP2000, and the Hitachi S-810120.

SIMD Supercomputers

In Fig. 1.3b, we have shown an abstract model of SlMD computers having a single instruction stream over multiple data streams. An operational model of SIMD computers

is presented below (Fig. 1. 13) based on the work of H. J. Siegel (19791)

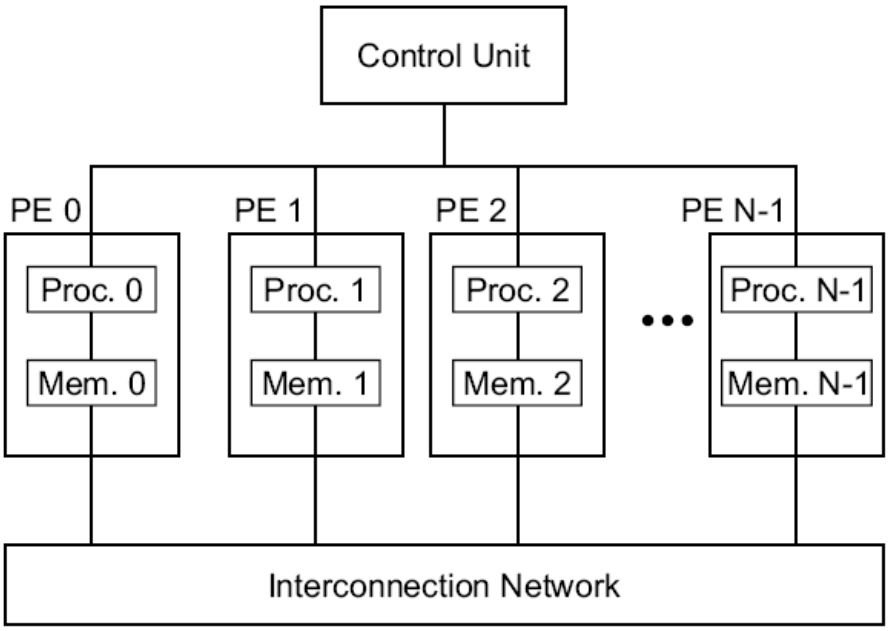


Fig. 1.12 Operational model of SIMD computers

SIMD Machine Model
An operational model of an SIMD computer is specified by a 5-tuple:
M=(N, CJ, M, R)

where –
(1 N is the number of processing elements (PEs) in the machine. For example, the llliac IV had 64 PEs and the Connection Machine CM-2 had 65,536 PEs.
(2) C is the set of instructions directly executed by the control unit (CU), including scalar and program flow control instructions.
(3) I is the set of instructions broadcast by the CU to all PEs for parallel execution. These include arithmetic, logic, data routing, masking, and other local operations executed by each active PE over data within that PE.
(4) M is the set of masking schemes, where each mask partitions the set of PEs into enabled and disabled subsets.
(5) R is the set of data-routing functions, specifying various patterns to be set up in the interconnection network for inter-PE communications.

One can describe a particular SIMD machine architecture by specifying the S-tuple. An example SIMD machine is partially specified below.

1.4 PRAM AND VLSI MODELS

Theoretical models of parallel computers are abstracted from the physical models studied in previous sections. These models are often used by algorithm designers and VLSI device/chip developers. The ideal models provide a convenient framework for developing

parallel algorithms without worry about the implementation details or physical constraints.

PRAM model (Parallel Random Access Machine):

PRAM Parallel random access machine; a theoretical model of parallel computation in which an arbitrary but finite number of processors can access any value in an arbitrarily large *shared memory* in a single time step. Processors may execute different instruction streams, but work *synchronously*. This model assumes a shared memory, multiprocessor machine as shown:

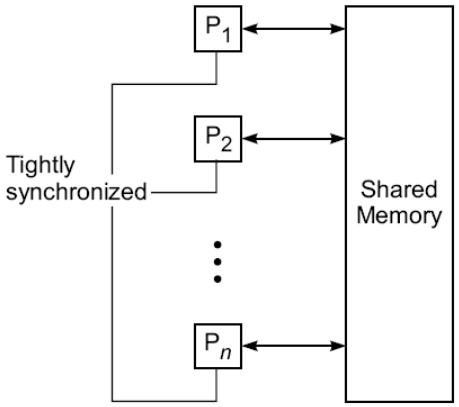


Fig. 1.14 PRAM model of a multiprocessor system with shared memory, on which all *n* processors operate in lockstep in memory access and program execution operations. Each processor can access any memory location in unit time

Exclusive read (ER)—This allows at most one processor to read from any memory location in each cycle, a rather restrictive policy.

Exclusive write (EW)—This allows at most one processor to write into a memory location at time.

Concurrent read (CR)—This allows multiple processors to read the same information from the same memory cell in the same cycle.

Concurrent write (CW)—This allows simultaneous writes to the same memory location. In order to avoid confusion_ some policy must be set up to resolve the write conflicts.

The four most important variations of the PRAM are:

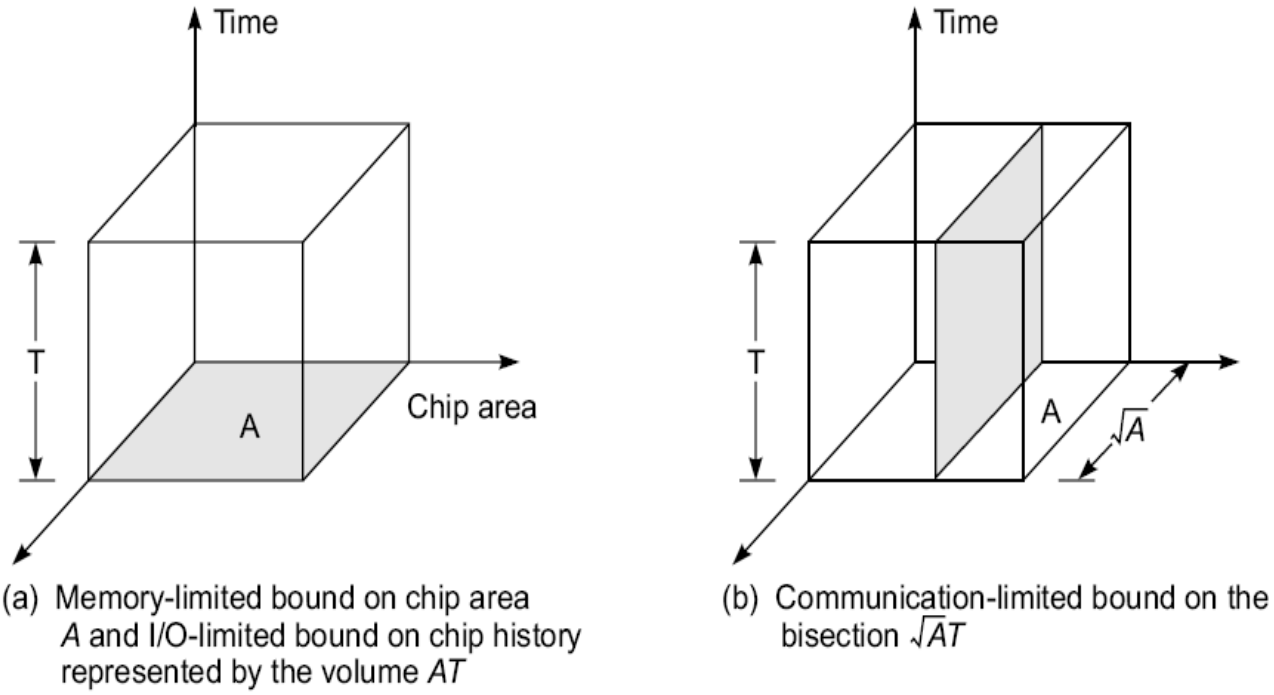
- **EREW** - Exclusive read, exclusive write; any memory location may only be accessed once in any one step. Thus forbids more than one processor from reading or writing the same memory cell simultaneously.
- **CREW** - Concurrent read, exclusive write; any memory location may be read any

number of times during a single step, but only written to once, with the write taking place after the reads.

- **ERCW** – This allows exclusive read or concurrent writes to the same memory location.
- **CRCW** - Concurrent read, concurrent write; any memory location may be written to or read from any number of times during a single step. A CRCW PRAM model must define some rule for resolving multiple writes, such as giving priority to the lowest-numbered processor or choosing amongst processors randomly. The PRAM is popular because it is theoretically tractable and because it gives algorithm designers a common target. However, PRAMs cannot be emulated *optimally* on all *architectures*.

VLSI Model:

Parallel computers rely on the use of VLSI chips to fabricate the major components such as processor arrays memory arrays and large scale switching networks. The rapid advent of very large scale intergrated (VSLI) technology now computer architects are trying to implement parallel algorithms directly in hardware. An AT^2 model is an example for two dimension VLSI chips.



(a) Memory-limited bound on chip area A and I/O-limited bound on chip history represented by the volume AT

(b) Communication-limited bound on the bisection $\sqrt{A}T$

Fig. 1.15 The AT^2 complexity model of two-dimensional VLSI chips

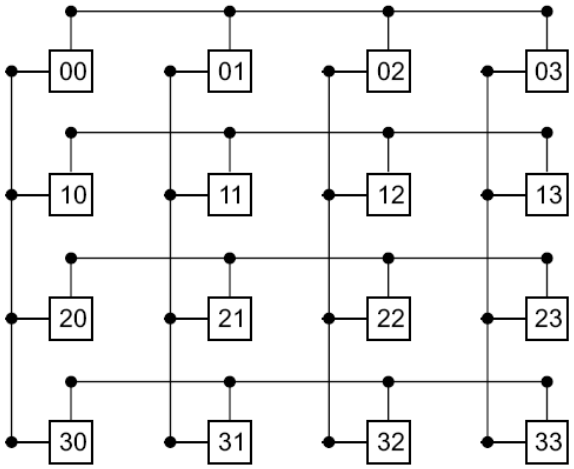


Fig. 1.16 A 4 × 4 mesh of processing elements (PEs) with broadcast buses on each row and on each column (Courtesy of Prasanna Kumar and Raghavendra; reprinted from *Journal of Parallel and Distributed Computing*, April 1987)