Introduction to Parallel/Distributed Computing

Types of Computing Systems

- There are several architectures which require a different OS:
  - Desktop PCs
  - Simple Batch System
  - Multiprogramming Batch System
  - Time sharing Systems
  - Parallel Systems
  - Distributed Systems
  - Clustered Systems
  - Real-time Systems
  - Embedded Systems
  - Handheld System

Desktop Systems

- Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs.
- PC operating systems therefore were neither **multiuser** nor **multitasking**.
- Instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These systems are called **Desktop Systems**
  - Example: PCs running Microsoft Windows and Apple Macintosh.
- Operating systems for these computers have benefited in several ways from the development of operating systems for **mainframes**.
- **Microcomputers** were immediately able to adopt some of the technology developed for larger operating systems.
- The hardware costs for microcomputers are sufficiently **low** that individuals have sole use of the computer, and CPU utilization is no longer a prime concern.

Simple Batch Systems

- In this type of system, there is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched together by type of languages and requirement.
- Then a special program, the monitor, manages the execution of each program in the batch.
- The monitor is always in the main memory and available for execution.
- Following are some disadvantages of this type of system:
  - No interaction between user and computer.
  - No mechanism to prioritize the processes.

Multiprogramming Batch Systems

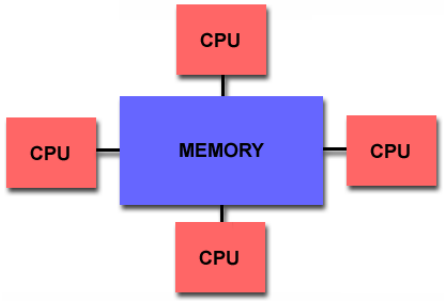
- In this the operating system picks up and begins to execute one of the jobs from memory.
- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.
- In non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

Time Sharing Systems

- **Time-Sharing Systems** are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.
- In time sharing systems the prime focus is on minimizing the response time, while in multiprogramming the prime focus is to maximize the CPU usage.

Multiprocessor Systems

- A multiprocessor system consists of several processors that share a common physical memory.
- Multiprocessor system provides higher computing power and speed.
- Here, all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.
- Following are some advantages of this type of system.
  - Enhanced performance
  - Increased the system's throughput
  - System can divide task into many subtasks which can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.



contd..

- Also called as Parallel Systems
- Due to their low message delay and high data transfer rate, they are also called tightly coupled systems
- *Tightly coupled system* – processors share memory and the internal clock; communication usually takes place through the shared memory.
- Advantages of parallel system:
  - Increased *throughput*
  - Economical
  - Increased reliability

Distributed Systems

- The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.
- Distributed systems comprises of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.
- Following are some advantages of this type of system.
  - As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
  - Fast processing
  - Less load on the Host Machine
- The two types of Distributed Operating Systems are:
  - Client /Server Systems
  - Peer to Peer Systems

contd..

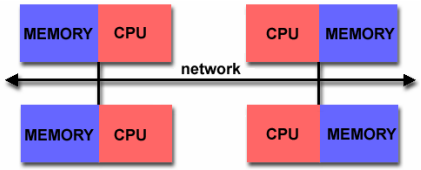
- Distribute the computation among several physical processors.
- Due to low data transfer rate and high message delay, these systems are also called loosely coupled systems
- *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various communications lines, such as high-speed buses or network communication.
- Advantages of distributed systems.
  - Resources Sharing
  - Computation speed up – load balancing

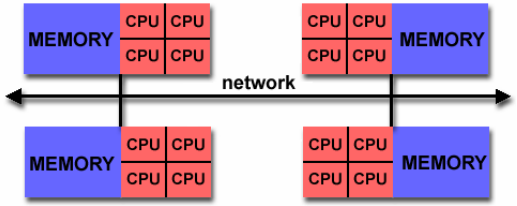
Distributed Systems

  - Scalability
  - Reliability
  - Fail-Safe
  - Communications
- OS has to cater for resource sharing.

Hybrid Systems

- Also called as Multi-Computing systems





contd..

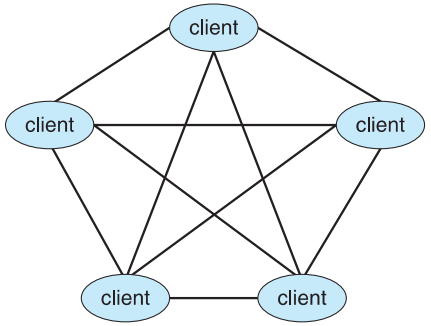
Client-Server Systems

- **Centralized systems** today act as **server systems** to satisfy requests generated by **client systems**.
- Server Systems can be broadly categorized as **compute servers** and **file servers**.
- **Compute-server systems** provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client (Ex: Remote Procedure Calls)
- **File-server systems** provide a file-system interface where clients can create, update, read, and delete files.

contd..

Peer-to-Peer Systems

- Systems with no central management, self organizing
- Peers in P2P are all equal and distributed
- Resource sharing
- Preferred when there are large number of peers in a network
- Heterogenous



Clustered Systems

- Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.
- Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
- The definition of the term clustered is **not concrete;** the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.
- Clustering is usually performed to provide **high availability**.

Real-time Systems

- It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.
- The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as **Hard Real-Time Operating Systems.**
- While the real-time operating systems that can only guarantee a maximum of the time, i.e. the critical task will get priority over other tasks, but no assurity of completing it in a defined time. These systems are referred to as **Soft Real-Time Operating Systems**.

Embedded Systems

- One of the most important and widely used categories of operating systems
- Hardware and software designed to perform a dedicated function
- Tightly coupled to their environment
- Issues:
  - Limited memory
  - Slow processors
  - Small display screens.
- Often, embedded systems are part of a larger system or product,
  - E.G. antilock braking system in a car.

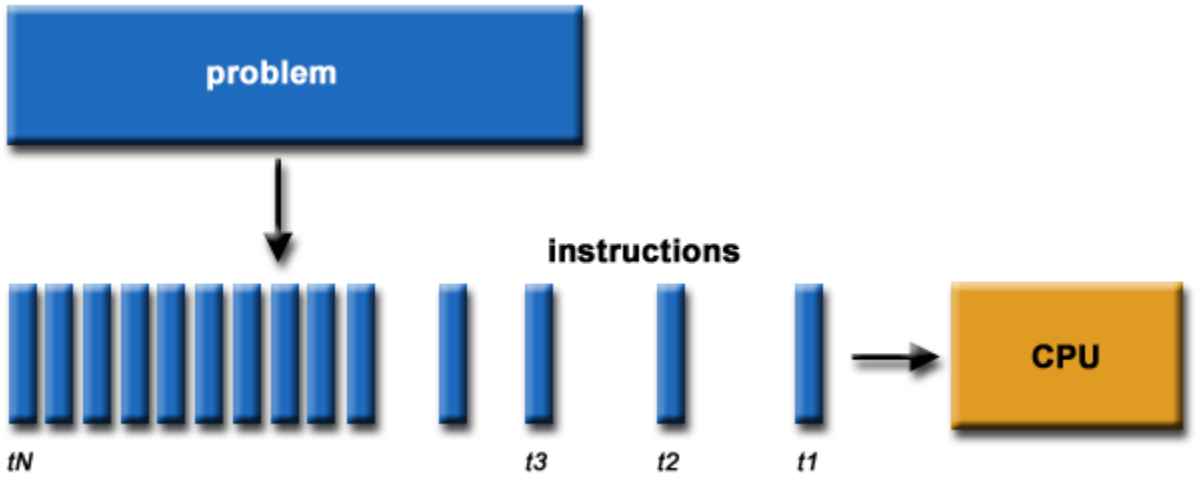
Handheld Systems

- Handheld systems include **Personal Digital Assistants(PDAs)**, such as Palmtops, Cellphones with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.
- Many handheld devices have between **512 KB** and **8 MB** of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.
- Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require **more power**. To include a faster processor in a handheld device would require a **larger battery** that would have to be replaced more frequently.
- The last issue confronting program designers for handheld devices is the small display screens typically available.
- Some handheld devices may use wireless technology such as **BlueTooth**, allowing remote access to e-mail and web browsing. **Cellular telephones** with connectivity to the Internet fall into this category.

Principles of Parallel/Distributed Computing

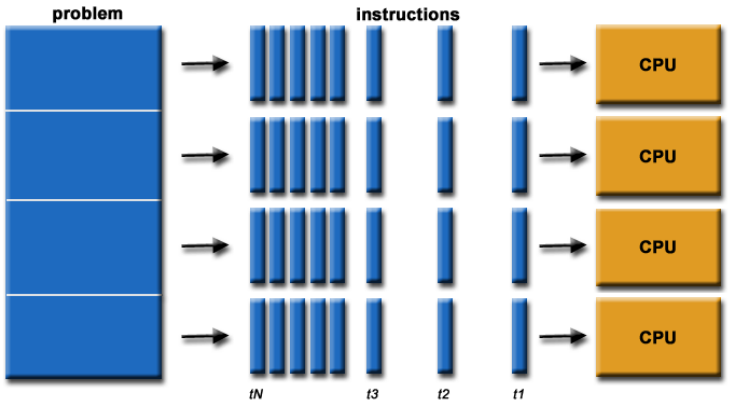
Serial Computation

- Traditionally, software has been written for ***serial*** computation:
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - Only one instruction may execute at any moment in time.



Parallel Computing

- In the simplest sense, ***parallel computing*** is the simultaneous use of multiple compute resources to solve a computational problem.
  - To be run using multiple CPUs
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
  - Instructions from each part execute simultaneously on different CPUs



Resource and Problem

- The compute resources can include:
  - A single computer with multiple processors;
  - An arbitrary number of computers connected by a network;
  - A combination of both.

- The computational problem usually demonstrates characteristics such as the ability to be:
  - Broken apart into discrete pieces of work that can be solved simultaneously;
  - Execute multiple program instructions at any moment in time;
  - Solved in less time with multiple compute resources than with a single compute resource.

Grand Challenge Problems

- Traditionally, parallel computing has been considered to be "the high end of computing" and has been motivated by numerical simulations of complex systems and "Grand Challenge Problems" such as:
  - weather and climate
  - chemical and nuclear reactions
  - biological, human genome
  - geological, seismic activity
  - mechanical devices - from prosthetics to spacecraft
  - electronic circuits
  - manufacturing processes

Applications

- Today, commercial applications are providing an equal or greater driving force in the development of faster computers. These applications require the processing of large amounts of data in sophisticated ways. Example applications include:
  - parallel databases, data mining
  - oil exploration
  - web search engines, web based business services
  - computer-aided diagnosis in medicine
  - management of national and multi-national corporations
  - advanced graphics and virtual reality, particularly in the entertainment industry
  - networked video and multi-media technologies
  - collaborative work environments

- Ultimately, parallel computing is an attempt to maximize the infinite but seemingly scarce commodity called **time**.

Why use parallel computing?

- The primary reasons for using parallel computing:
  - Save time - wall clock time
  - Solve larger problems
  - Provide concurrency (do multiple things at the same time)

- Other reasons might include:
  - Taking advantage of non-local resources - using available compute resources on a wide area network, or even the Internet when local compute resources are scarce.
  - Cost savings - using multiple "cheap" computing resources instead of paying for time on a supercomputer.
  - Overcoming memory constraints - single computers have very finite memory resources. For large problems, using the memories of multiple computers may overcome this obstacle.

Why use parallel computing?

- Limits to serial computing - both physical and practical reasons pose significant constraints to simply building ever faster serial computers:
  - Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
  - Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
  - Economic limitations - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

- The future: during the past 10 years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) suggest that ***parallelism is the future of computing***.

Concept and Terminology

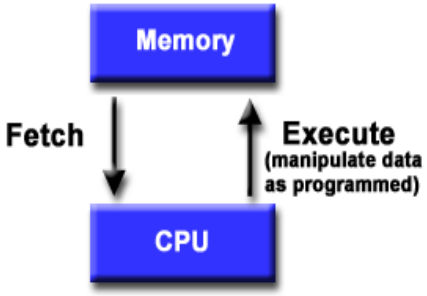
- Von Newmann Architecture
- Flynn's Classical Taxonomy

Von Neumann Architecture

- For over 40 years, virtually all computers have followed a common machine model known as the von Neumann computer. Named after the Hungarian mathematician John von Neumann.

- A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.

- Basic design:
  - Memory is used to store both program and data instructions
  - Program instructions are coded data which tell the computer to do something
  - Data is simply information to be used by the program
  - A central processing unit (CPU) gets instructions and/or data from memory, decodes the instructions and then ***sequentially*** performs them.



Flynn's Classical Taxonomy

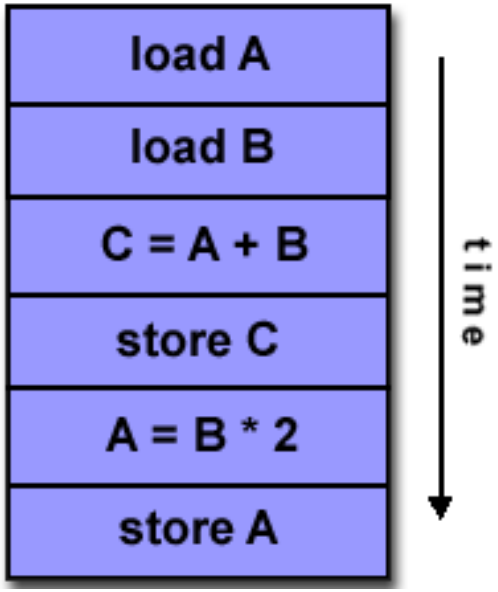
- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's Taxonomy.

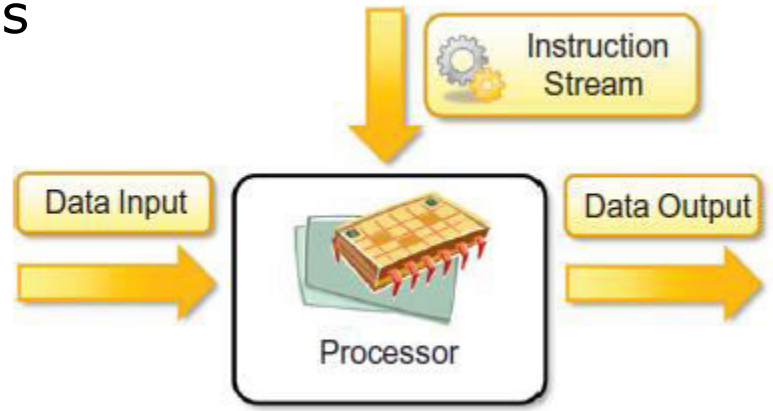
- Flynn's taxonomy distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction*** and ***Data***. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.

- There are 4 possible classifications according to Flynn.
  - **Single Instruction, Single Data (SISD)**
  - **Single Instruction, Multiple Data (SIMD)**
  - **Multiple Instruction, Single Data (MISD)**
  - **Multiple Instruction, Multiple Data (MIMD)**

Single Instruction Single Data

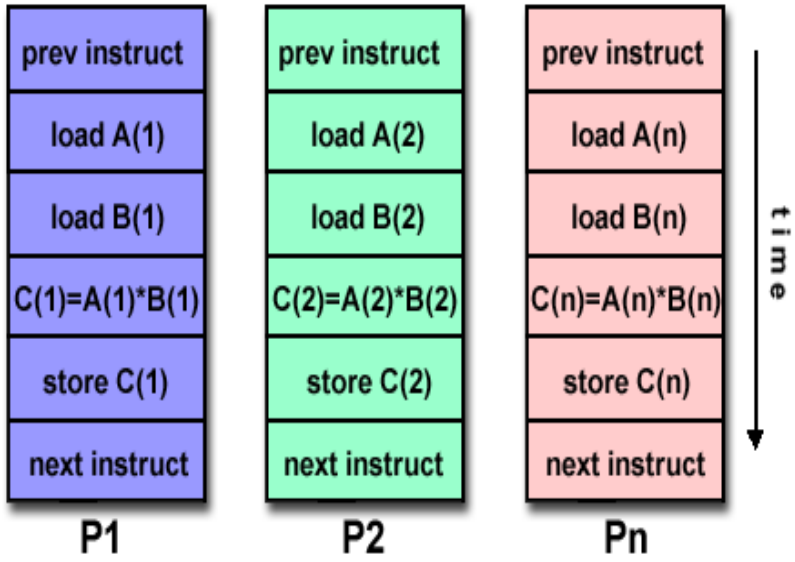
- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and until recently, the most prevalent form of computer
- Examples: most PCs, single CPU workstations and mainframes



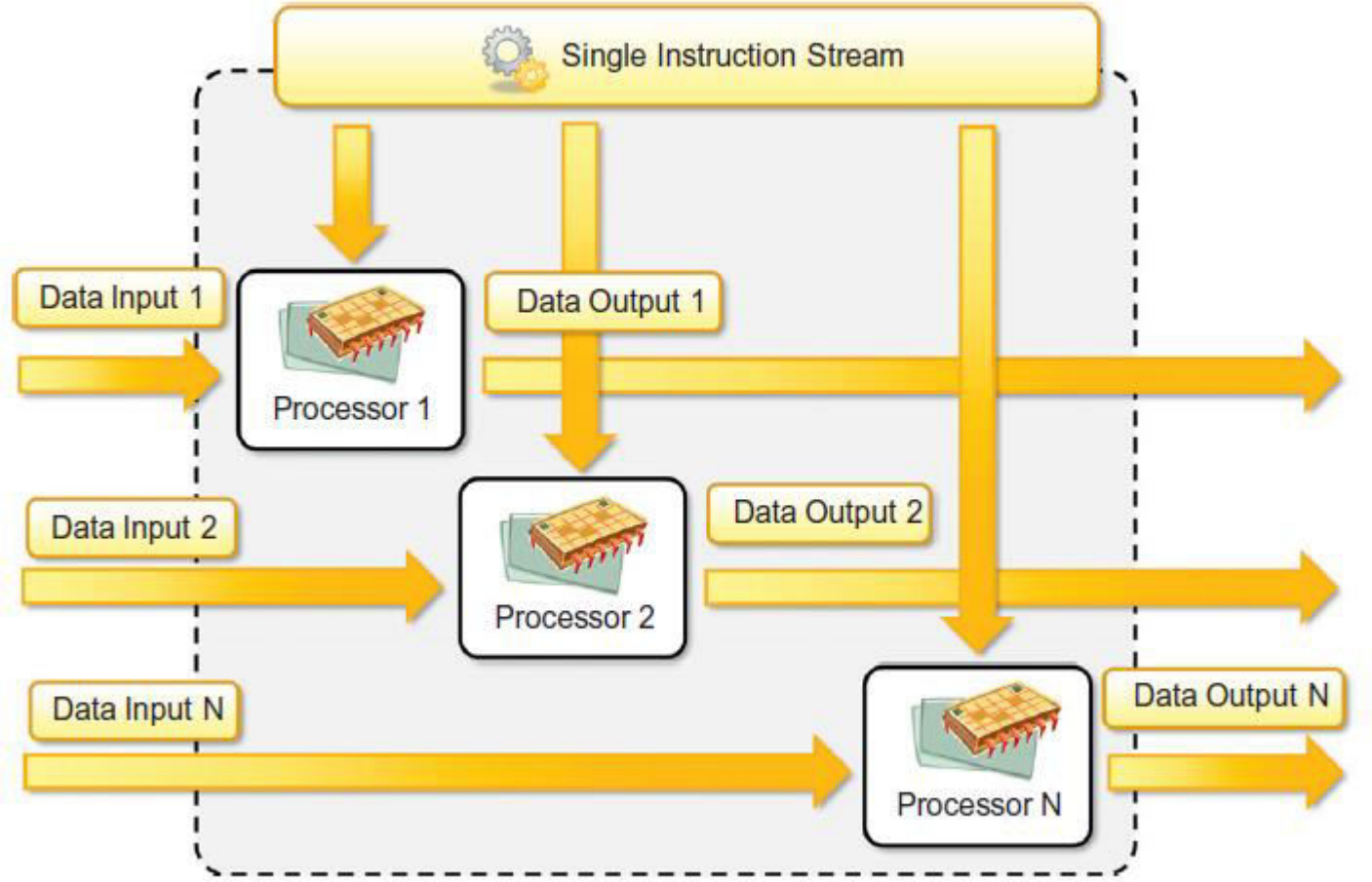


Single Instruction Multiple Data

- A type of parallel computer
- Single instruction: All processing units execute the same instruction at any given clock cycle
- Multiple data: Each processing unit can operate on a different data element
- 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.
- Best suited for specialized problems characterized by a high degree of regularity, such as image processing.
- Synchronous (lockstep) and deterministic execution
- Two varieties: Processor Arrays and Vector Pipelines
- Examples:
  - Processor Arrays: Connection Machine CM-2, Maspar MP-1, MP-2
  - Vector Pipelines: IBM 9000, Cray C90, Fujitsu VP, NEC SX-2, Hitachi S820

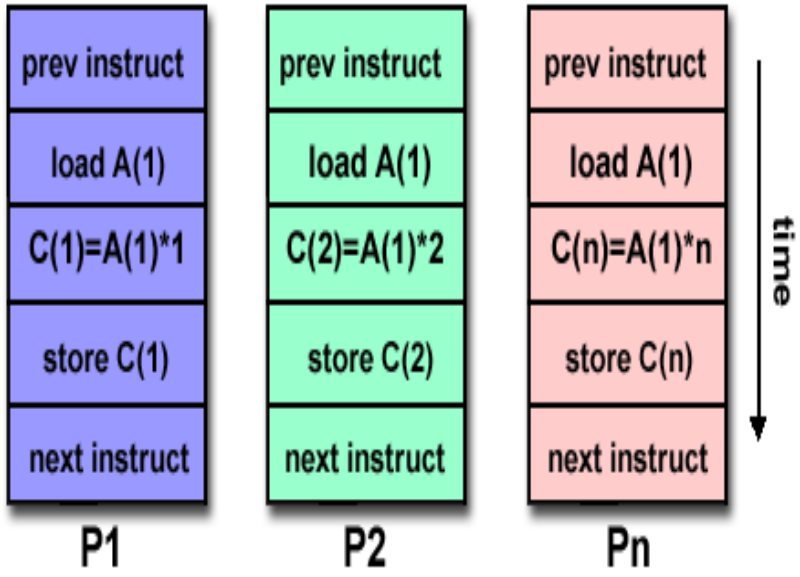


SIMD (contd..)

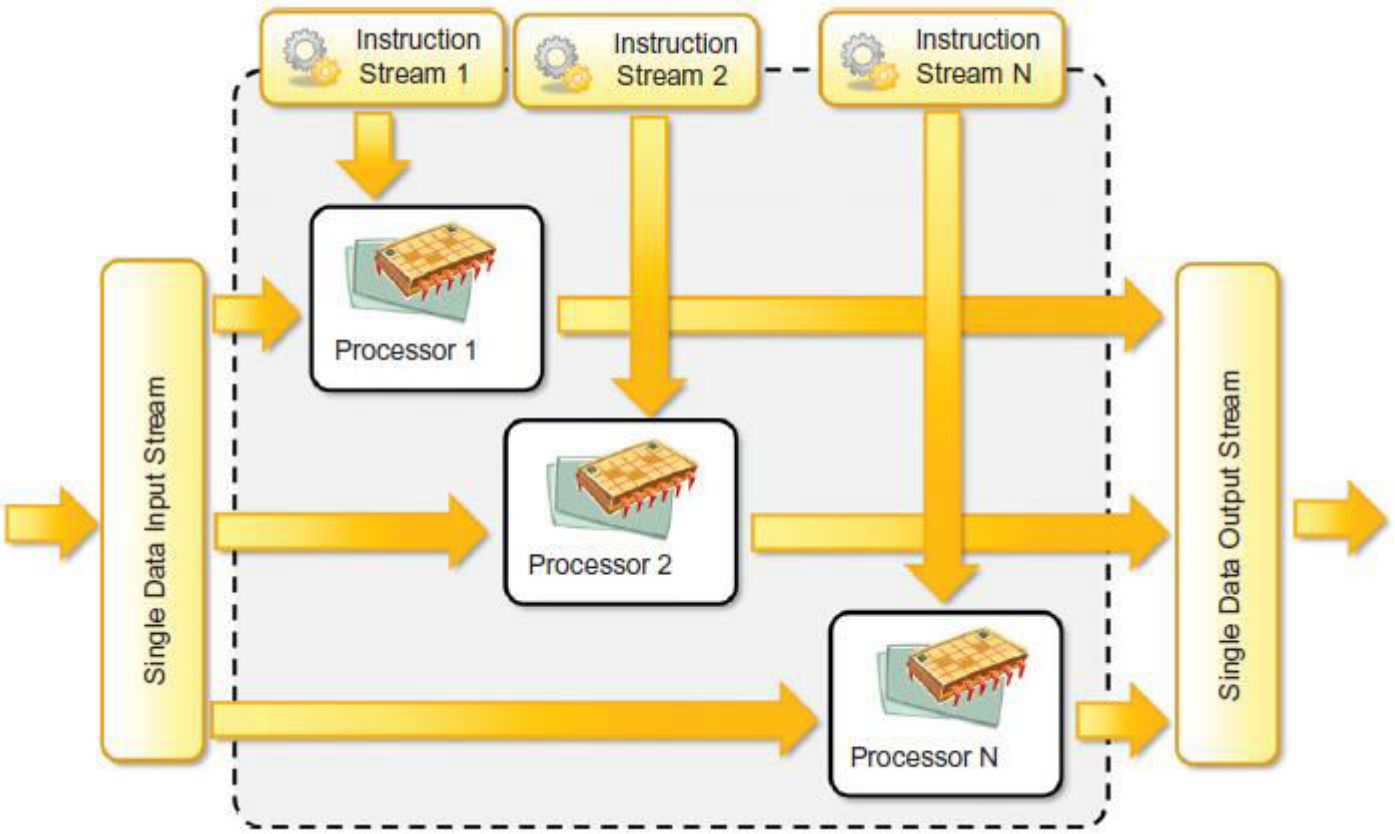

Single Instruction Stream
Data Input 1
Processor 1
Data Output 1
Data Input 2
Processor 2
Data Output 2
Data Input N
Processor N
Data Output N

Multiple Instruction Single Data

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- 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:
  - multiple frequency filters operating on a single signal stream
  - multiple cryptography algorithms attempting to crack a single coded message.

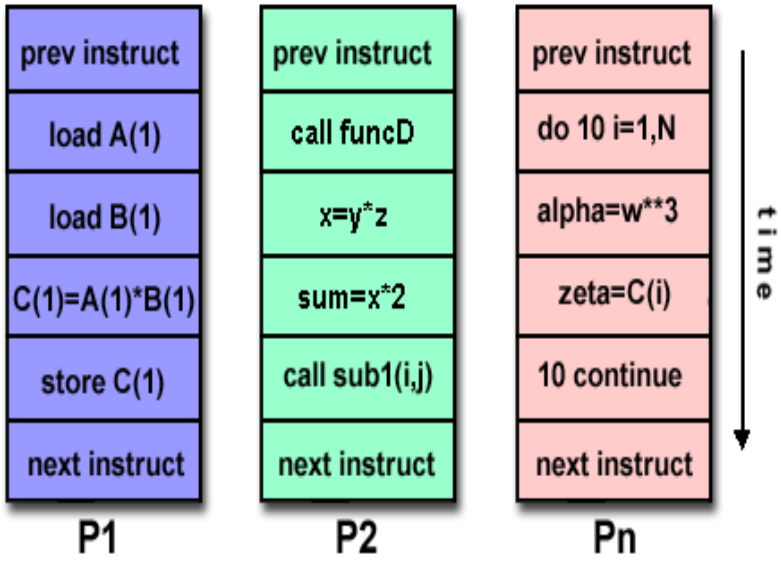


MISD (contd..)

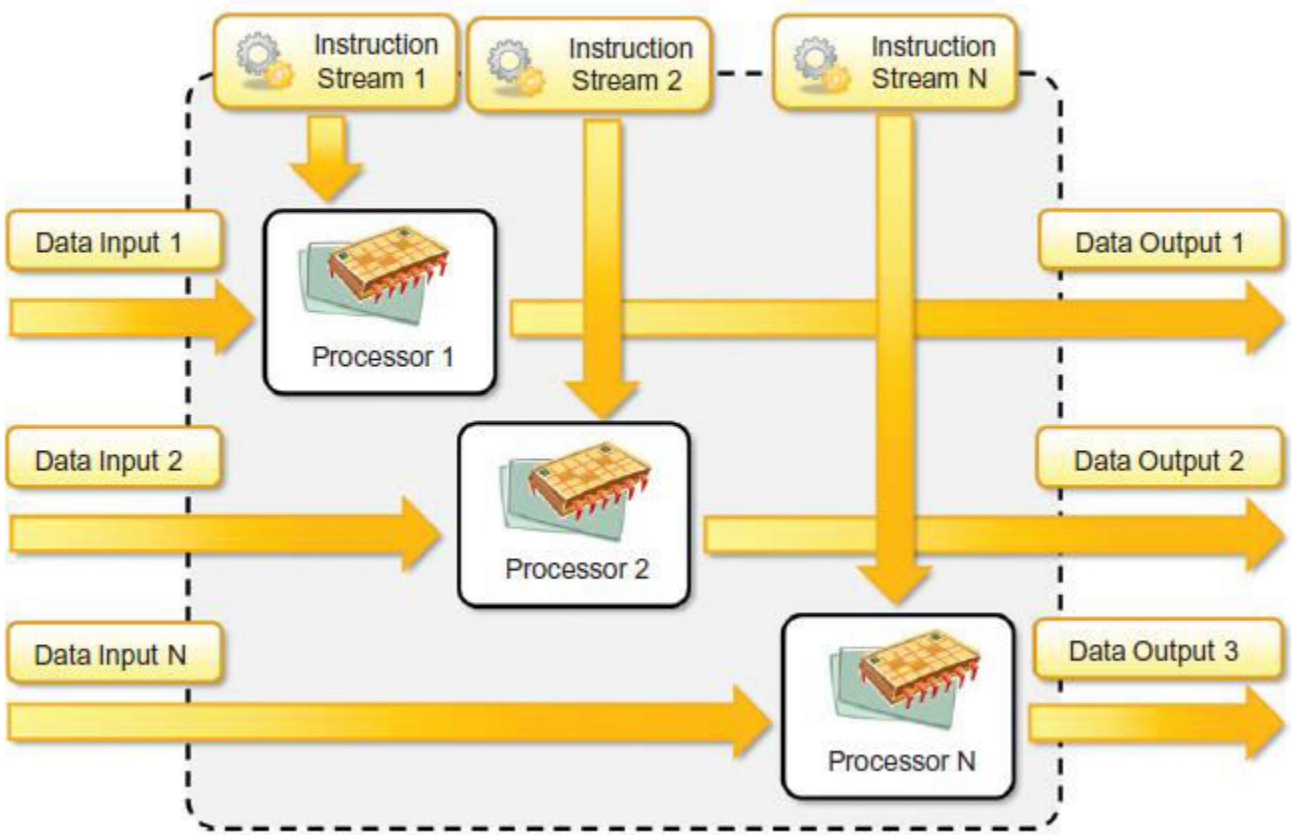

Instruction Stream 1
Instruction Stream 2
Instruction Stream N
Processor 1
Processor 2
Processor N
Single Data Input Stream
Single Data Output Stream

Multiple Instruction Multiple Data

- Currently, the most common type of parallel computer. Most modern computers fall into this category.
- Multiple Instruction: every processor may be executing a different instruction stream
- Multiple Data: every processor may be working with a different data stream
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer "grids" and multi-processor SMP computers - including some types of PCs.



MIMD (contd..)

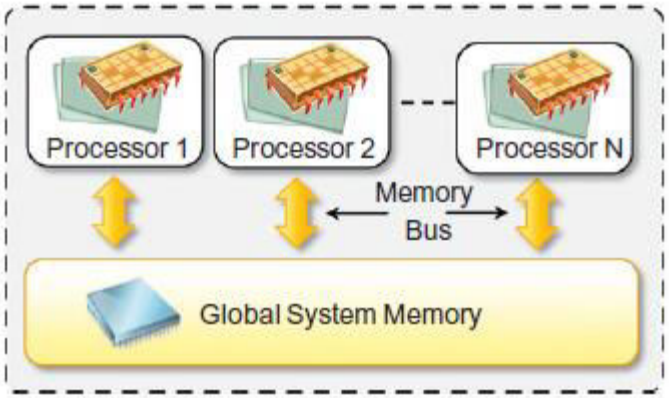

Instruction Stream 1
Instruction Stream 2
Instruction Stream N
Data Input 1
Data Input 2
Data Input N
Processor 1
Processor 2
Processor N
Data Output 1
Data Output 2
Data Output 3

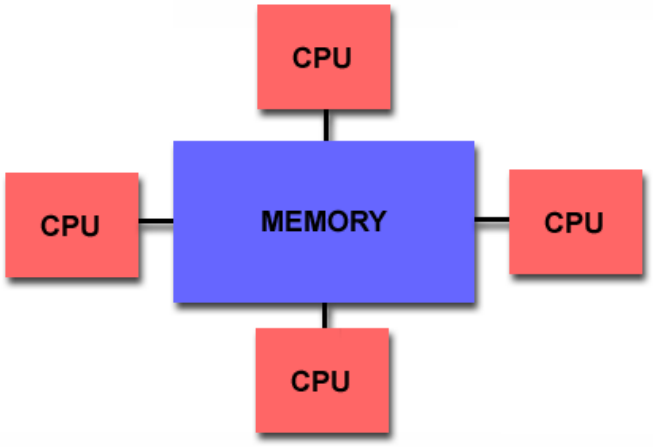
Parallel Computer Memory Architectures

- Shared Memory
- Distributed Memory
- Hybrid Distributed Shared Memory

Shared Memory

- 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*** and ***NUMA***.



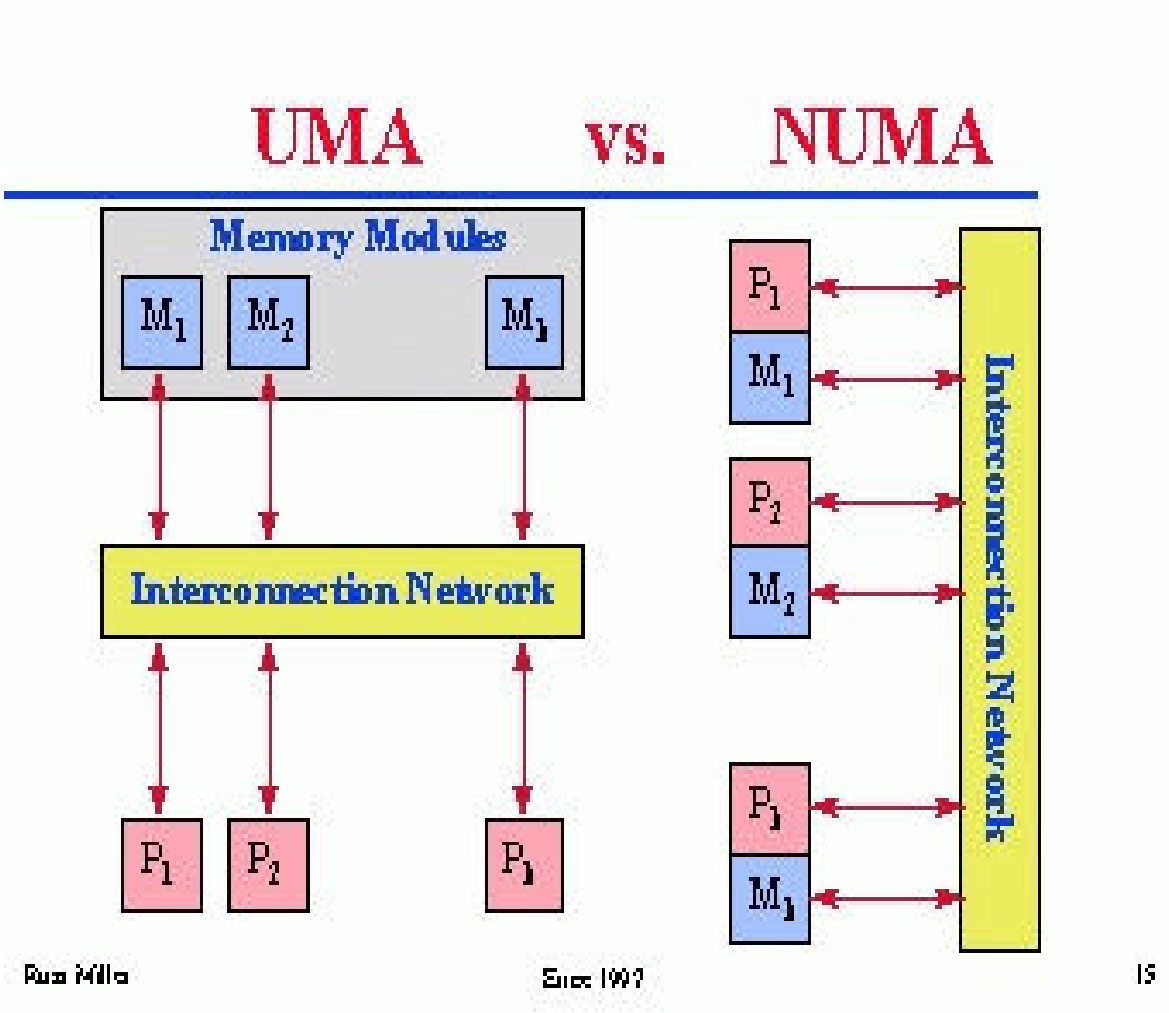


Shared Memory

- 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.

- 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

UMA and NUMA


UMA
vs.
NUMA
Memory Modules
M1
M2
M3
Interconnection Network
P1
P2
P3
P1
M1
P2
M2
P3
M3
Interconnection Network

Shared Memory

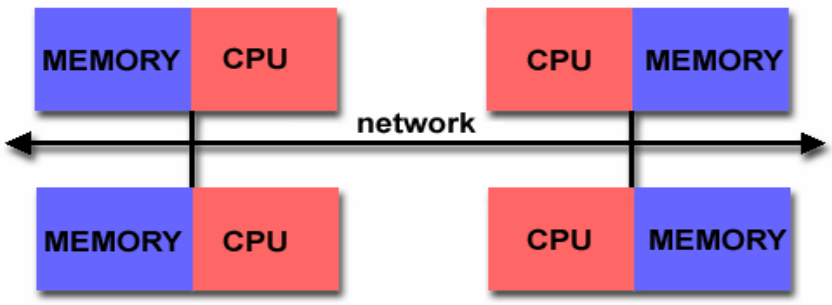
- Advantages:
  - Global address space provides a user-friendly programming perspective to memory
  - Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

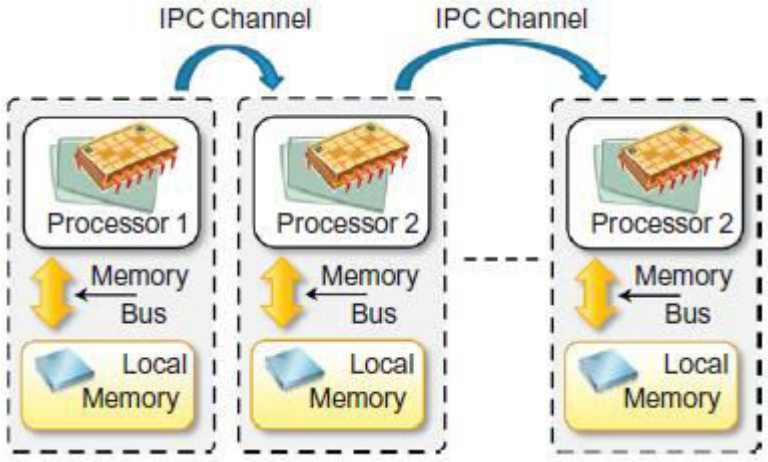
- Disadvantages:
  - Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
  - Programmer responsibility 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.
- 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.
- The network "fabric" used for data transfer varies widely, though it can can be as simple as Ethernet.

contd..


MEMORY
CPU
CPU
MEMORY
network
MEMORY
CPU
CPU
MEMORY


IPC Channel
IPC Channel
Processor 1
Processor 2
Processor 2
Memory Bus
Memory Bus
Memory Bus
Local Memory
Local Memory
Local Memory

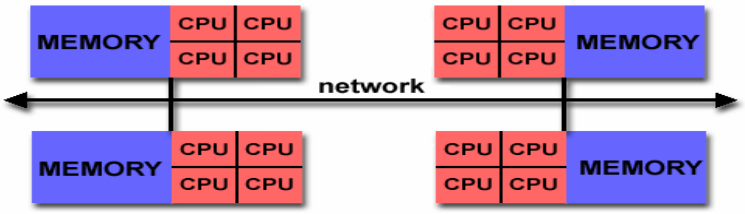
Distributed Memory

- 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

Distributed Shared Memory

- The largest and fastest computers in the world today employ both shared and distributed memory architectures.
- The shared memory component is usually a cache coherent SMP machine. Processors on a given SMP can address that machine's memory as global.
- The distributed memory component is the networking of multiple SMPs. SMPs know only about their own memory - not the memory on another SMP. Therefore, network communications are required to move data from one SMP to another.
- Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.
- Advantages and Disadvantages: whatever is common to both shared and distributed memory architectures.



Approaches of Parallel/Distributed Programming

- A sequential program is one that runs on a single processor and has a single line of control.
- To make many processors collectively work on a single program, the program must be divided into smaller independent chunks so that each processor can work on separate chunks of the problem.
- The program decomposed in this way is a parallel program.

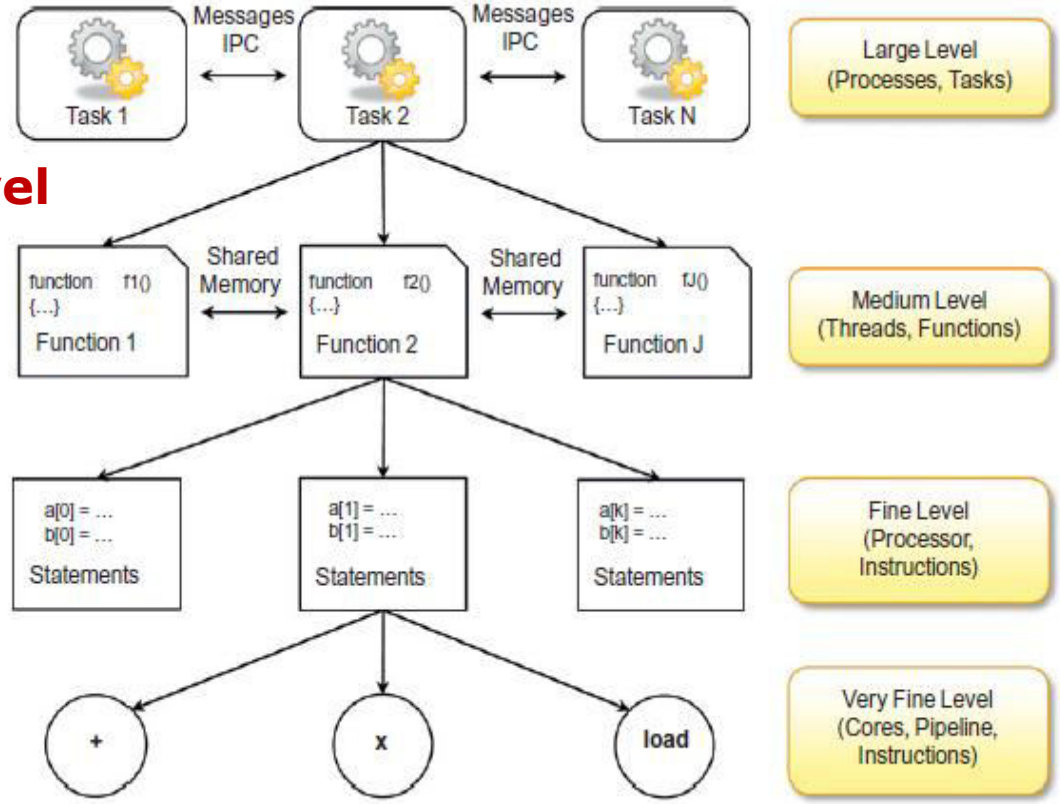
contd..

- A wide variety of parallel programming approaches are available.
- **Data parallelism** - the divide-and-conquer technique is used to split data into multiple sets, and each data set is processed on different PEs using the same instruction. This approach is highly suitable to processing on machines based on the SIMD model.
- **Process parallelism** - a given operation has multiple (but distinct) activities that can be processed on multiple processors.
- **Farmer-and-worker model** - a job distribution approach is used: one processor is configured as master and all other remaining PEs are designated as slaves; the master assigns jobs to slave PEs and, on completion, they inform the master, which in turn collects results.

Levels of Parallelism or Granularity

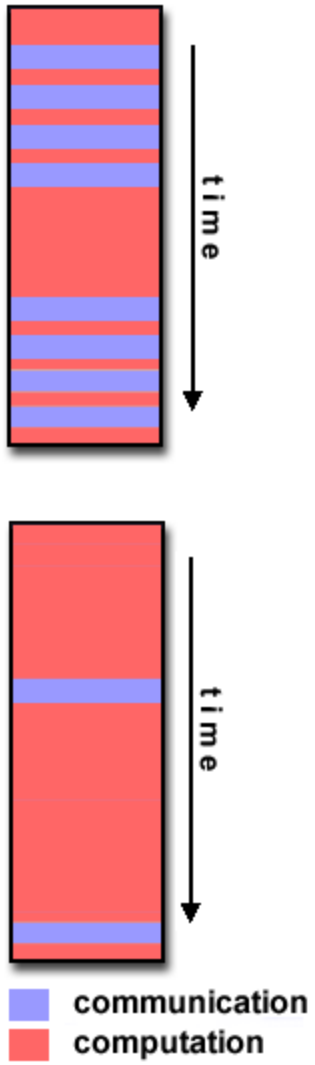
- Levels of parallelism are decided based on the lumps of code (grain size) that can be a potential candidate for parallelism.

- **Large or task level**
- **Medium or control level**
- **Fine or data level**
- **Very fine level**



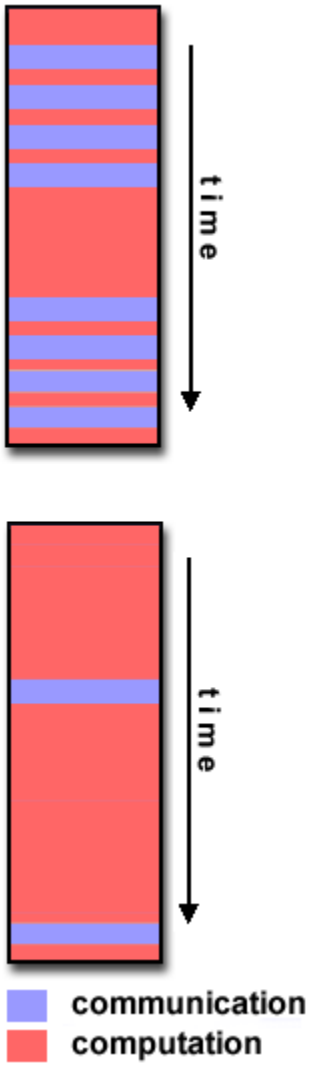
Granularity contd..

- In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
- Fine-grain Parallelism:
  - Relatively small amounts of computational work are done between communication events
  - Low computation to communication ratio
  - Facilitates load balancing
  - Implies high communication overhead and less opportunity for performance enhancement
  - If granularity is too fine it is possible that the overhead required for communications and synchronization between tasks takes longer than the computation.



Granularity contd..

- Coarse-grain Parallelism:
  - Relatively large amounts of computational work are done between communication/synchronization events
  - High computation to communication ratio
  - Implies more opportunity for performance increase
  - Harder to load balance efficiently
- Which is Best?
  - The most efficient granularity is dependent on the algorithm and the hardware environment in which it runs.
  - In most cases the overhead associated with communications and synchronization is high relative to execution speed so it is advantageous to have coarse granularity.
  - Fine-grain parallelism can help reduce overheads due to load imbalance.



Speedup Factor

- How much faster the multiprocessor solves the problem?

- We defined the speedup factor $S(p)$ which is a measure of relative performance

$$S(p) = \frac{\text{Execution time using single processor system}}{\text{Execution time using a multiprocessor with } p \text{ processors}} = \frac{t_s}{t_p}$$

- Maximum speedup (linear speedup)

$$S(p) \leq \frac{t_s}{t_s / p} = p$$

- Superlinear speedup $S(p) > p$

Efficiency

- If we want to know how long processors are being used on the computation. The efficiency *E* is defined as

$$E = \frac{\text{Execution time using one processor}}{\text{Execution time using a multiprocessor} \times \text{number of processors}}$$

$$= \frac{t_s}{t_p \times p}$$

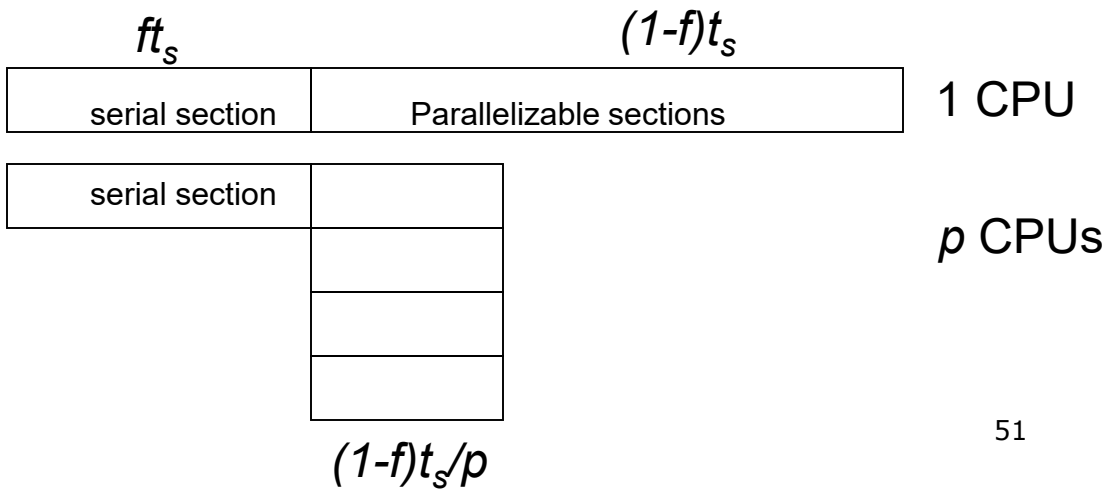
$$E = \frac{S(p)}{p} \times 100\%$$

while *E* is given as a percentage. If *E* is 50%, the processors are being used half the time on the actual computation, on average. If efficiency is 100% then the speedup is p.

Overheads

- Several factors will appear as overhead in the parallel computation
  - Periods when not all the processors can be performing useful work
  - Extra computations in the parallel version
  - Communication time between processors
- Assume the fraction of the computation that cannot be divided in to concurrent tasks is *f*.
- The time used to perform computation with *p* processors is

$$t_p = ft_s + (1-f)t_s / p$$

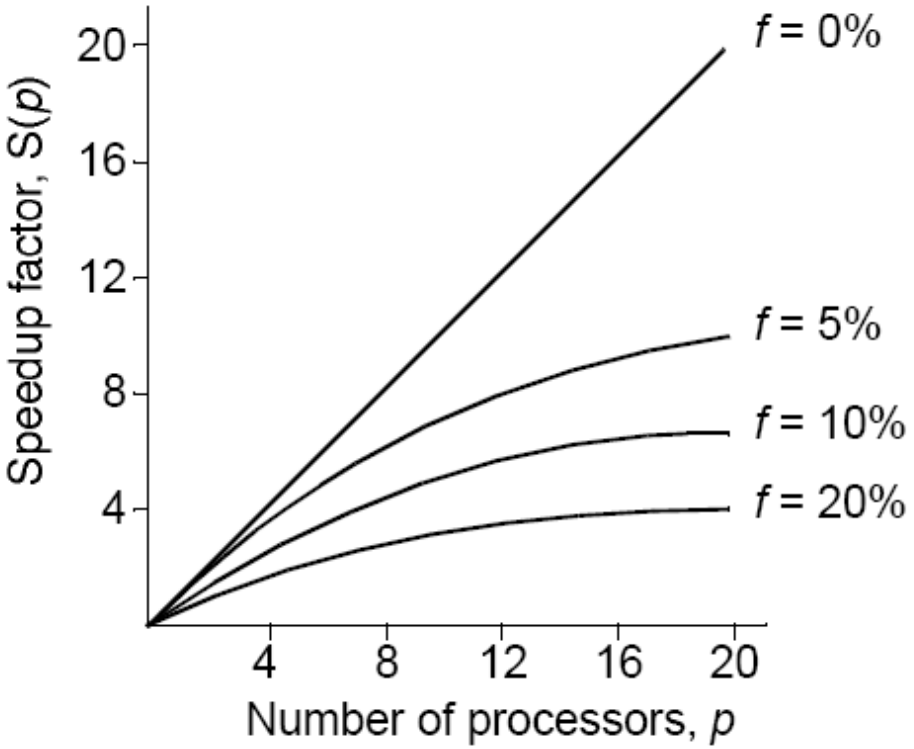


Amdahl's Law

- The speedup factor is given as

$$S(p) = \frac{t_s}{f t_s + (1-f)t_s / p} = \frac{p}{1 + (p-1)f}$$

$$\lim_{p \to \infty} S(p) = \frac{1}{f}$$



Complexity

- In general, parallel applications are much more complex than corresponding serial applications, perhaps an order of magnitude. Not only do you have multiple instruction streams executing at the same time, but you also have data flowing between them.
- The costs of complexity are measured in programmer time in virtually every aspect of the software development cycle:
  - Design
  - Coding
  - Debugging
  - Tuning
  - Maintenance
- Adhering to "good" software development practices is essential when working with parallel applications - especially if somebody besides you will have to work with the software.

Parallel Terminology

- **Task**
  - A logically discrete section of computational work. A task is typically a program or program-like set of instructions that is executed by a processor.
- **Parallel Task**
  - A task that can be executed by multiple processors safely (yields correct results)
- **Serial Execution**
  - Execution of a program sequentially, one statement at a time. In the simplest sense, this is what happens on a one processor machine. However, virtually all parallel tasks will have sections of a parallel program that must be executed serially.
- **Parallel Execution**
  - Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time.

Parallel Terminology

- **Shared Memory**
  - From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.

- **Distributed Memory**
  - In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

- **Communications**
  - Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.

Parallel Terminology

- **Synchronization**
  - The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point. Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.
- **Granularity**
  - In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
    - ***Coarse:*** relatively large amounts of computational work are done between communication events
    - ***Fine:*** relatively small amounts of computational work are done between communication events
- **Observed Speedup**
  - Observed speedup of a code which has been parallelized, defined as: wall-clock time of serial execution / wall-clock time of parallel execution
  - One of the simplest and most widely used indicators for a parallel program's performance.

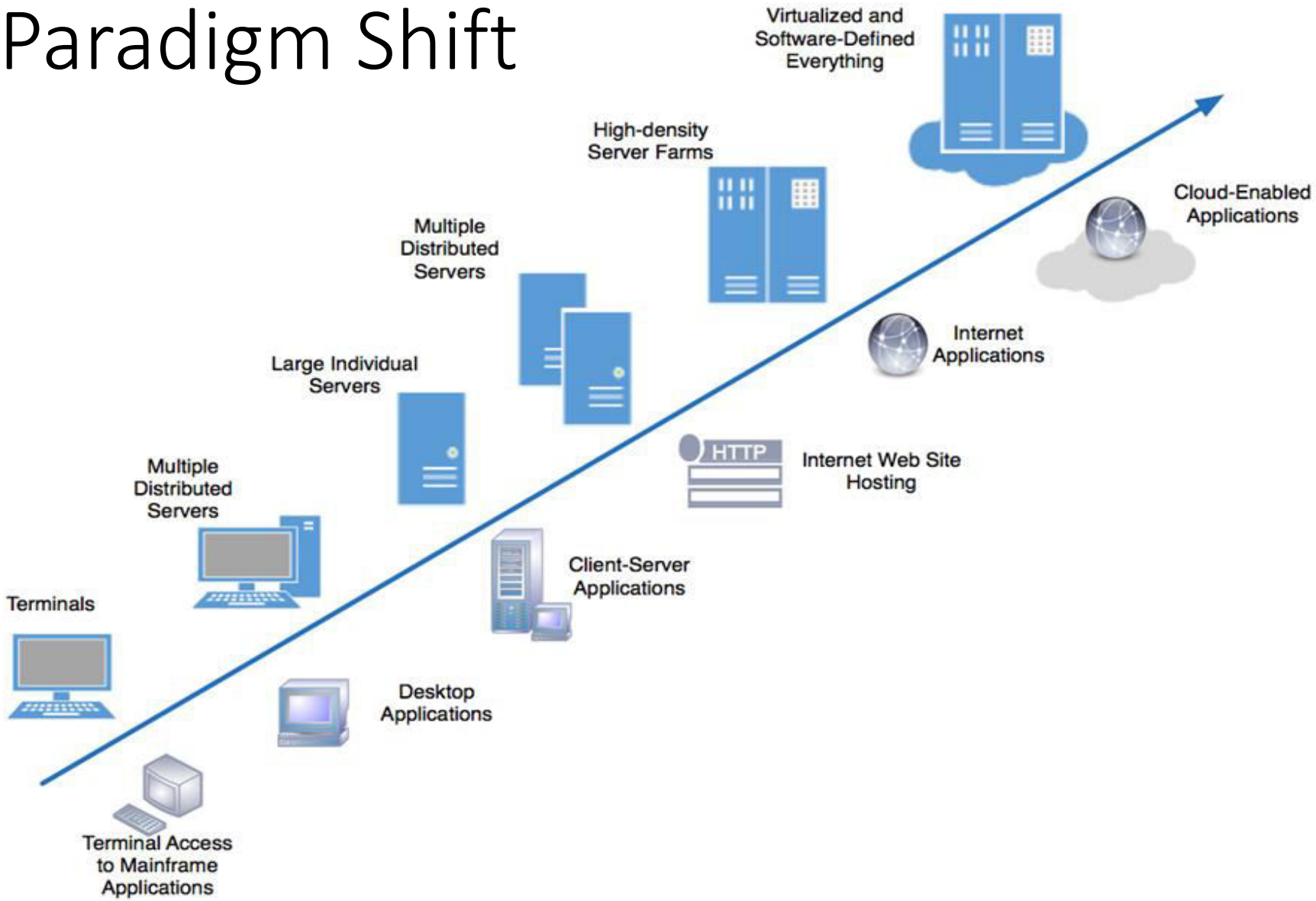
Parallel Terminology

- **Parallel Overhead**
  - The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as:
    - Task start-up time
    - Synchronizations
    - Data communications
    - Software overhead imposed by parallel compilers, libraries, tools, operating system, etc.
    - Task termination time
- **Massively Parallel**
  - Refers to the hardware that comprises a given parallel system - having many processors.

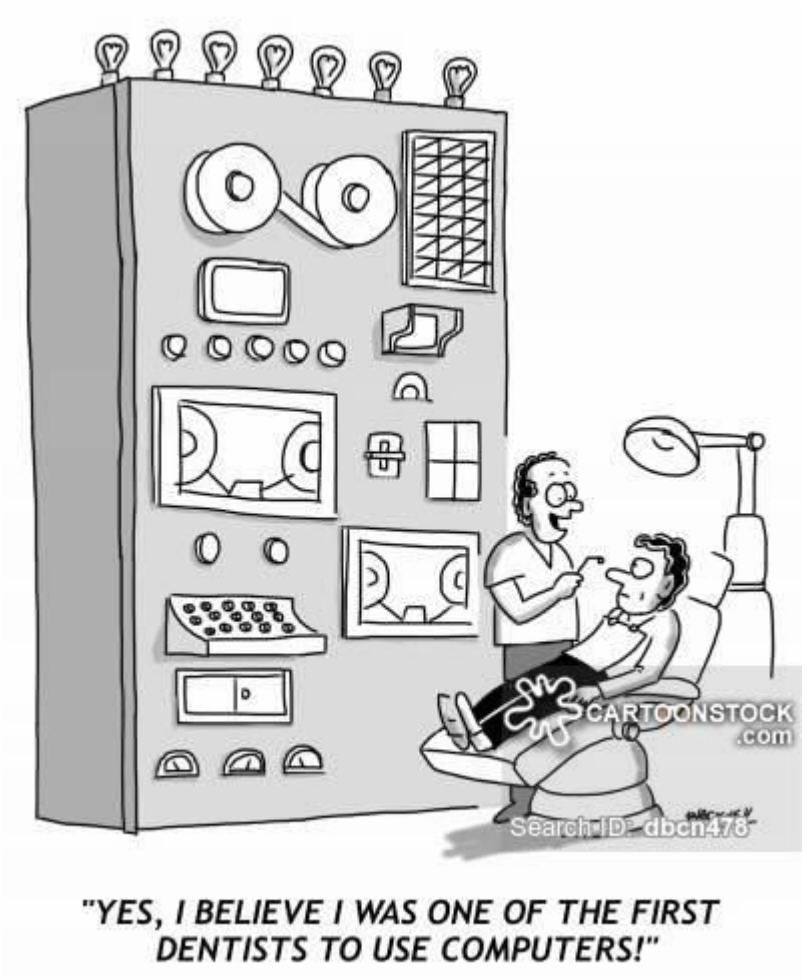
- **Scalability**
  - Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include:
    - Hardware - particularly memory-cpu bandwidths and network communications
    - Application algorithm
    - Parallel overhead related
    - Characteristics of your specific application and coding

Distributed Computing Models

Computing Paradigm Shift



Mainframe Computing



"YES, I BELIEVE I WAS ONE OF THE FIRST DENTISTS TO USE COMPUTERS!"

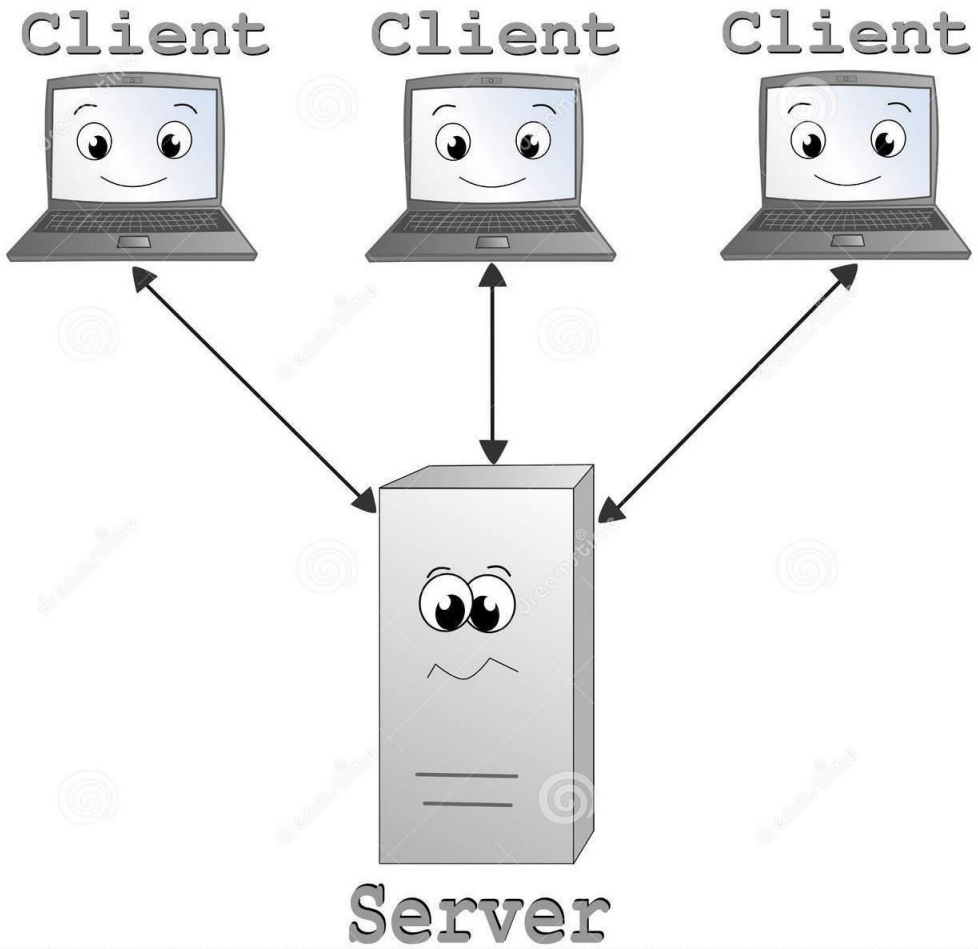
- Jobs
- Batches
- Processing
- To carry out some mundane and routine jobs such as payroll, accounts, inventory thus sparing employees from tedious jobs.
- It was available in one location, and anyone who needs it must go to computer center for availing it.

Personal Computing

"Tech support says the problem is located somewhere between the keyboard and my chair."

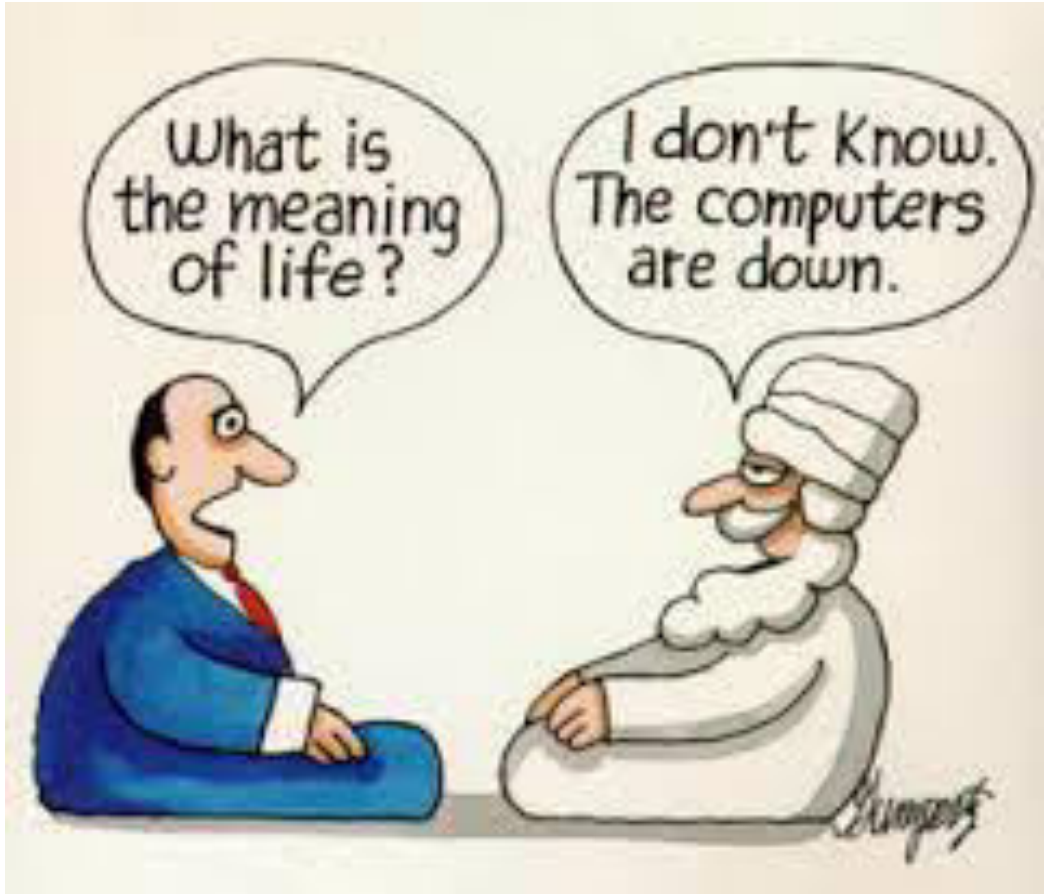
- Desktop computing - personal computer small enough to fit conveniently in an individual workspace.
- Providing computers to each employee on their desktop or workspace.
- Decentralized computing.
- Less expensive, easy to upgrade and less accessories needed.
- Information sharing with other users is a tedious process.

Network Computing



- Networked computers - Local Area network (LAN) achieved this.
- In the networked computing model- a relatively powerful computer- **server** is loaded with all software needed
- Each user is provided with a connected- **terminal** to access and work

Internet Computing



- Network computing such as LAN connected users within an office or institutions.
- Internet computing - connect organizations located in different geographical locations.

Utility Computing

- Conventional Internet hosting services have the capability to quickly arrange for the rental of individual servers, for example to provision a bank of web servers to accommodate a sudden surge in traffic to a web site.
- “Utility computing” usually envisions some form of virtualization so that the amount of storage or computing power available is considerably larger than that of a single time-sharing computer. Multiple servers are used on the “back end” to make this possible.
- These might be a dedicated computer cluster specifically built for the purpose of being rented out, or even an under-utilized supercomputer.
- The technique of running a single calculation on multiple computers is known as distributed computing.

Cluster Computing

- A computer cluster is a group of linked computers, working together closely thus in many respects forming a single computer.
- The components of a cluster are connected to each other through fast local area networks.
- Clusters are mainly used for load balancing and providing high availability.
- Requirements for such a computing increasing fast.
  - More data to process.
  - More compute intensive algorithms available.

Cluster Computing

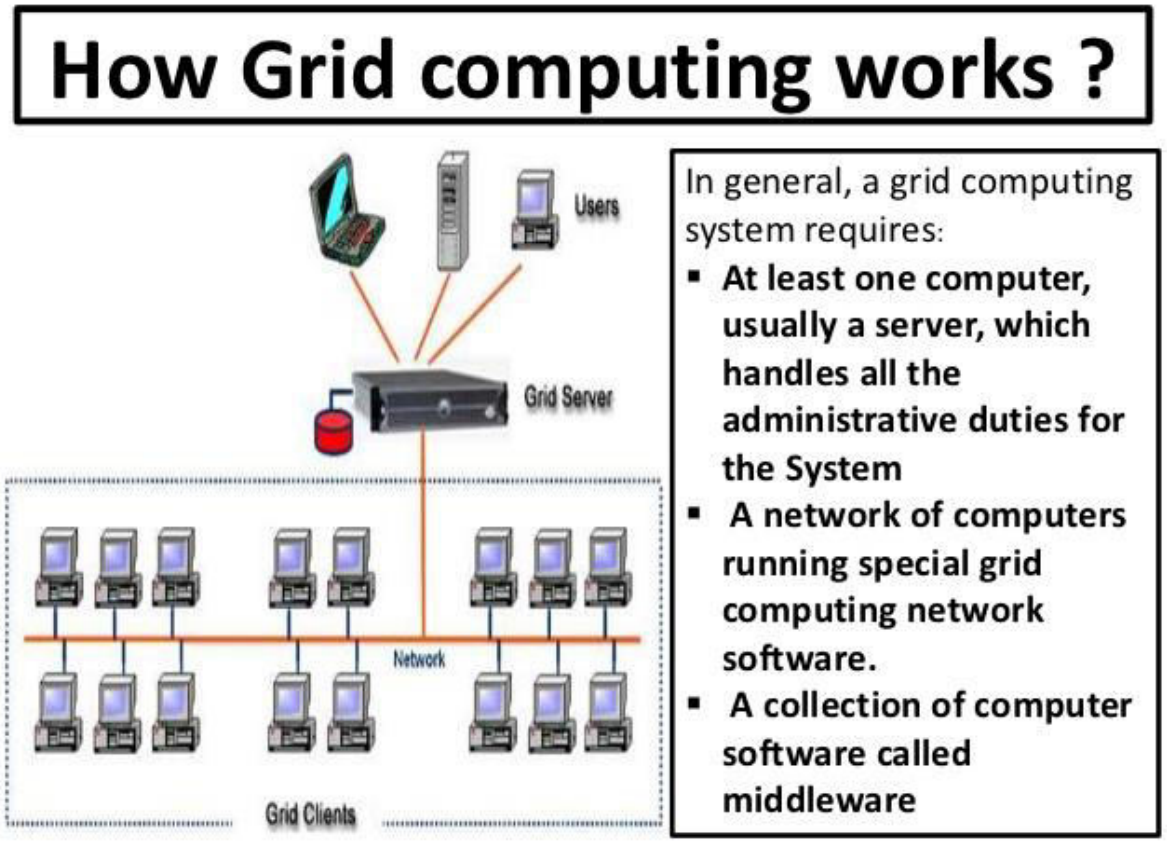
Benefits

- High Availability.
- Reducing cost.
- Manageability.

Drawbacks

- Problem in Finding Fault.
- The machines in a cluster are dedicated to work as a single unit.
- The computers in the cluster are normally contained in a single location.

Grid Computing



- Computing power available within an enterprise is not sufficient to carry out the computing task.
- Data required for the processing is generated at various geographical locations.
- GC requires the use of software that can divide and farm out pieces of a program as one large system image to several thousand computers.

Grid Computing

Benefits

- Enables applications to be easily scaled
- Better utilization of underused resources
- Parallelization of processing

Drawbacks

- Proprietary approach should be eliminated
- There is a single point of failure if one unit on the grid degrades
- No pay as you go

Grid Computing vs Cluster Computing

- Cluster is homogenous.
  - The cluster computers all have the same hardware and OS.
- The computers in the cluster are normally contained in a single location
- Grids are heterogeneous.
  - Run different operating systems and have different hardware.
- Grids are inherently distributed by its nature over a LAN, metropolitan or WAN.

Cloud Computing

- Cloud Computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services.

- The datacenter hardware and software is what we will call a Cloud.

Cloud Computing

Benefits

- Disaster recovery
- Increased Scalability
- Faster Deployment
- Metered Service
- Highly Automated

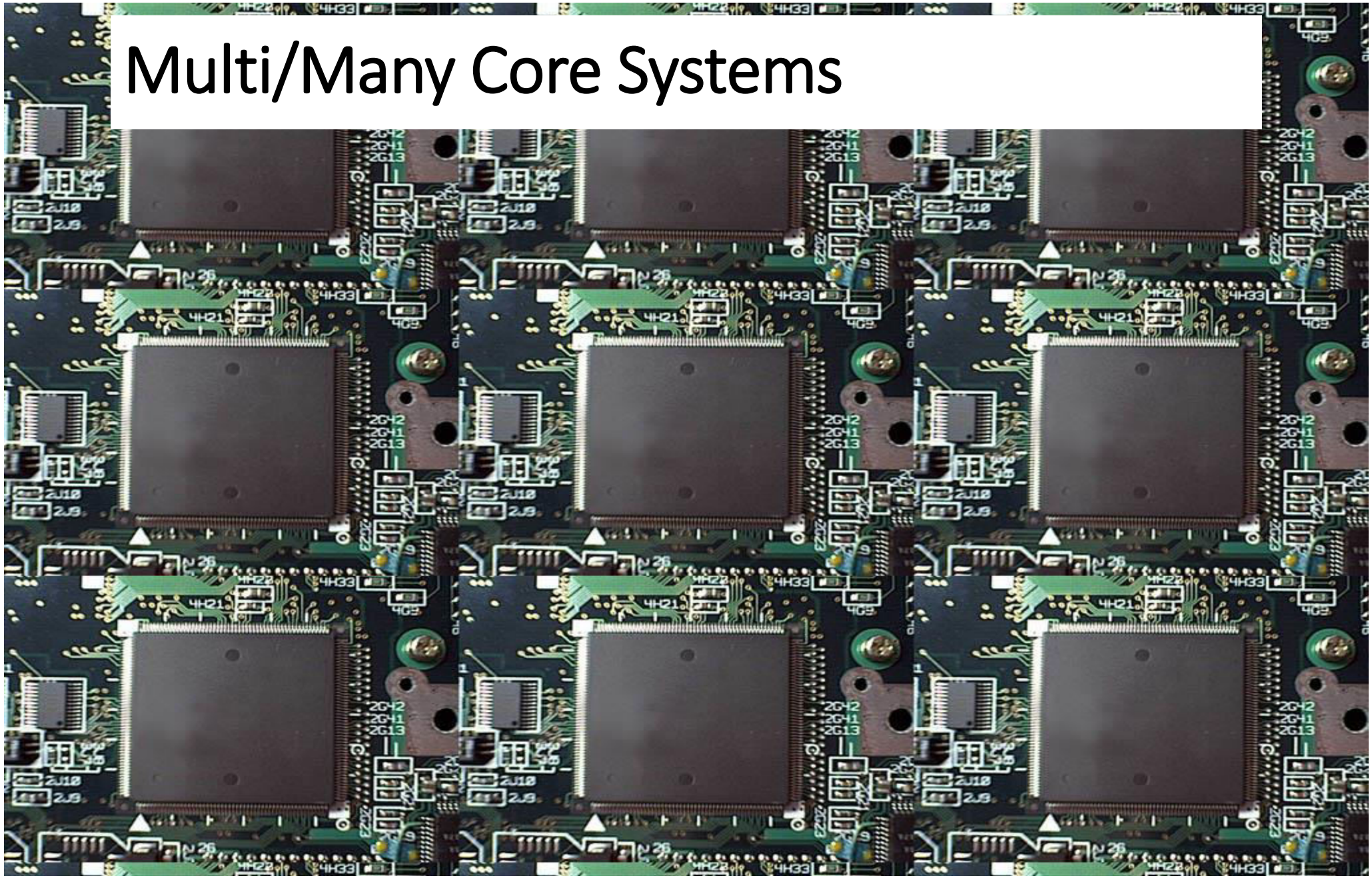
Drawbacks

- Constant Internet Connection
- High Speed Internet Required
- Data Stored is not secure

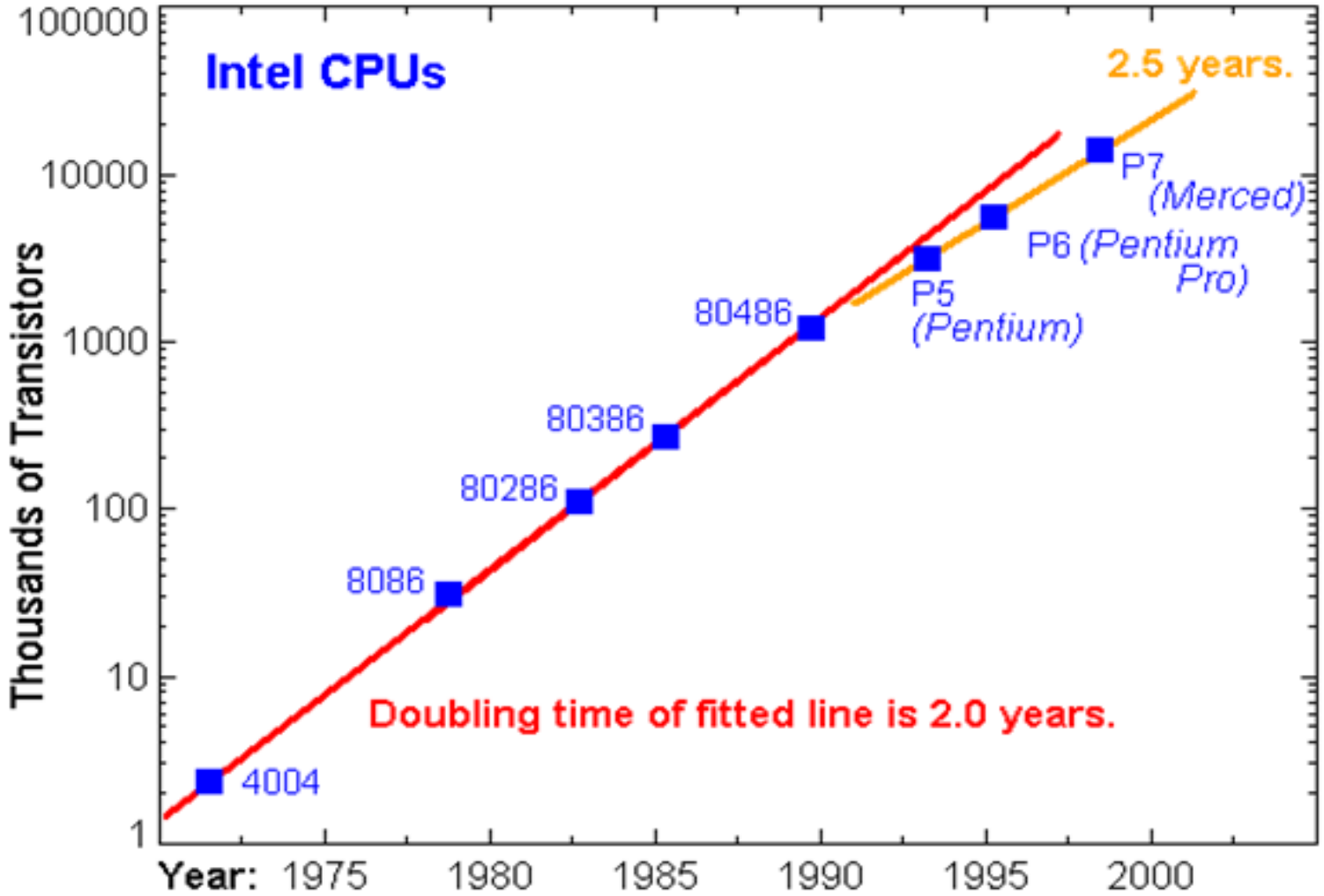
Grid Vs Cluster Vs Cloud Computing

Properties	Cloud	Cluster	Grid
On-demand self-Service			
Broad network access			
Resource pooling			
Rapid elasticity			
Measured service			

Multi/Many Core Systems



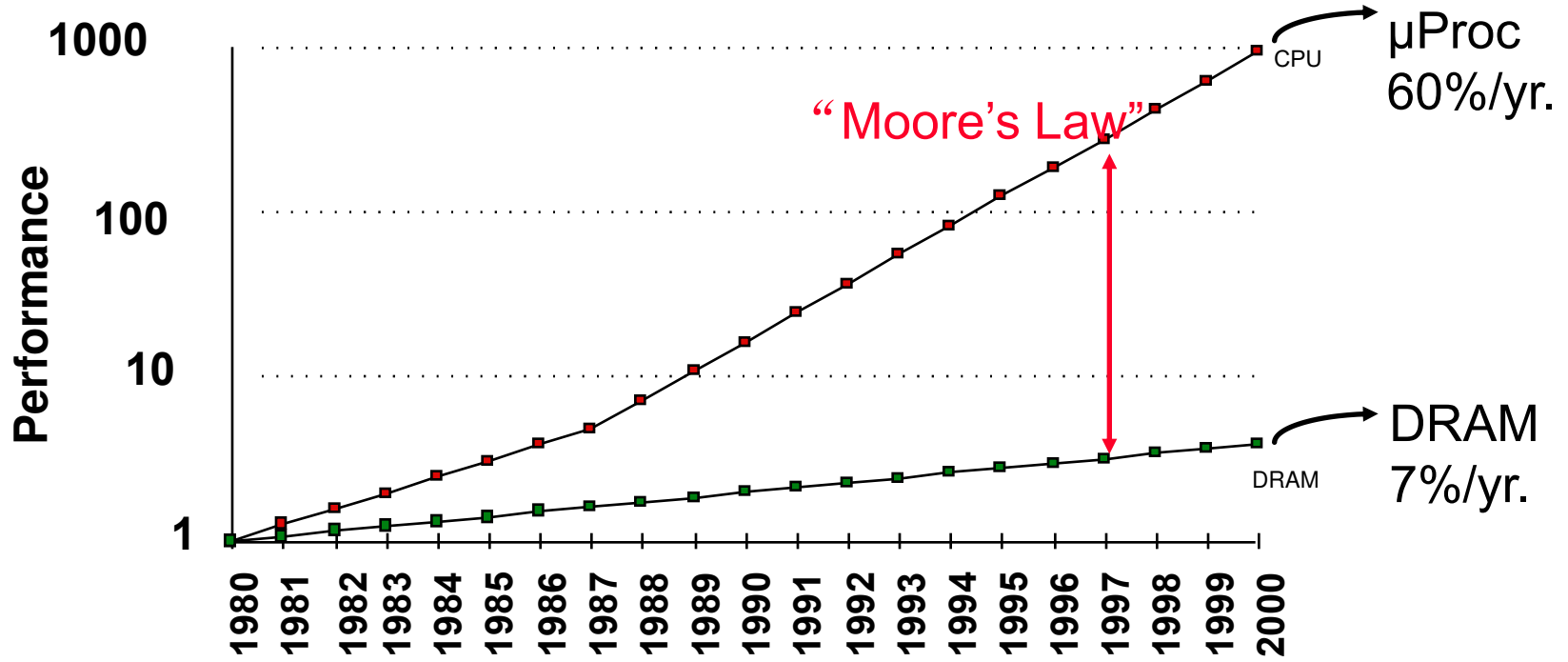
Moore's Law


Intel CPUs
2.5 years.
Thousands of Transistors
100000
10000
1000
100
10
1
Year: 1975
1980
1985
1990
1995
2000
4004
8086
80286
80386
80486
P5
(Pentium)
P6 (Pentium
Pro)
P7
(Merced)
Doubling time of fitted line is 2.0 years.

Processor Trends So Far..

- Smarter Brain
  - –(e.g. x386 → x486 → Pentium → P2 → P3 → P4)
- Larger Memory
  - –Larger caches, DRAM, Disk
- Smaller Head
  - –Fewer chips (integrate more things onto a chip)
- More Power Consumption
  - –few Watts →120+ Watts!
- More Complex
  - –1Billion Transistors; design + verification complexity

Processor-Memory Performance Gap



From D. Patterson, CS252, Spring 1998 ©UCB

Major Problem Today: End of the Road!

- Can’t increase Power Consumption (~100W!)
- Can’t increase Design Complexity
- Can’t increase Verification Requirements

No further improvements to a Processor?!

Obvious Answer: Use Multiple Brains

- If single brain can't be improved, use multiple brains!
- Put multiple simple CPUs on a single chip

Multi-Core!

The First Multi-Core: IBM Power 4 Processor

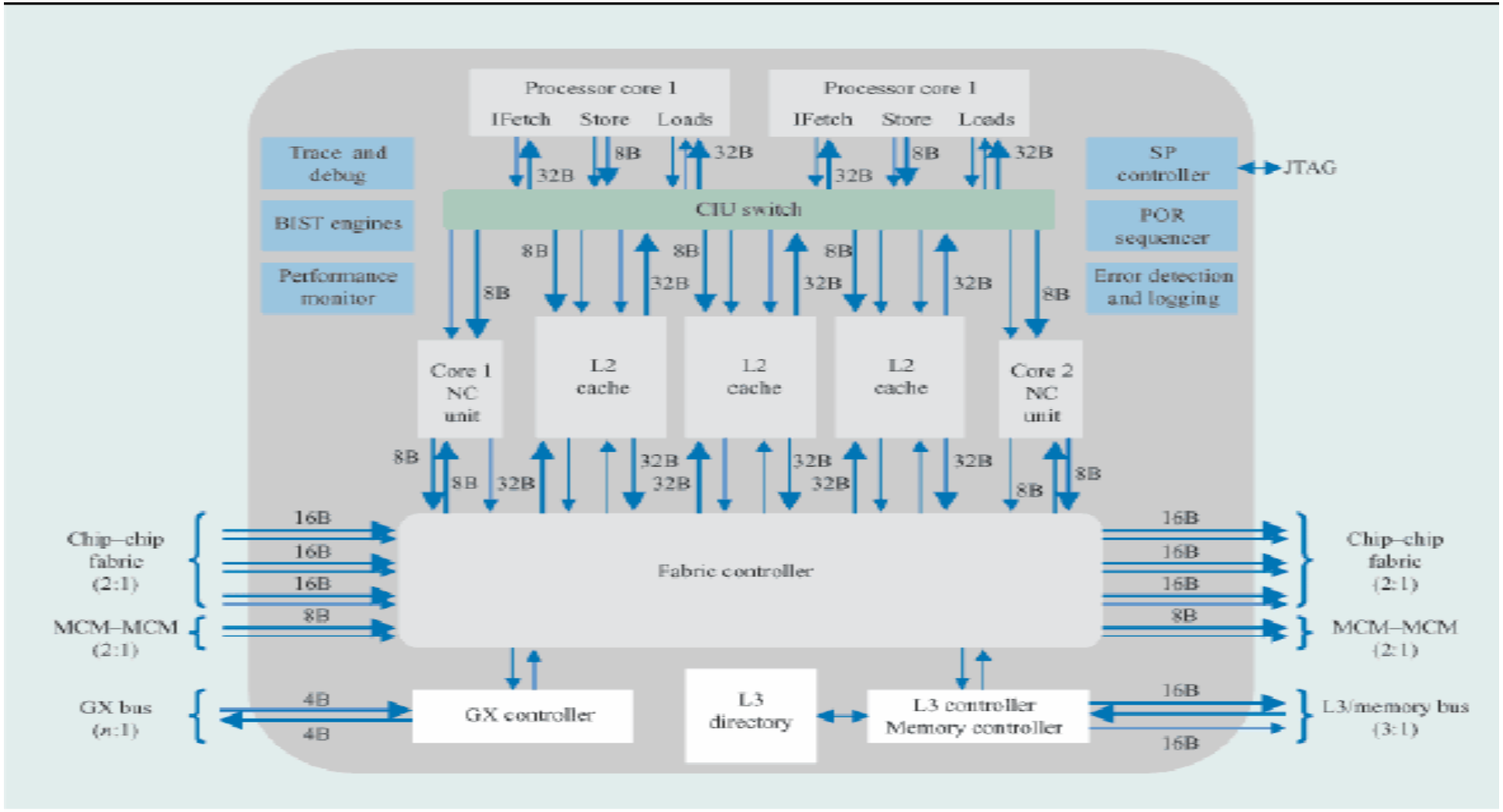


Figure 1

POWER4 chip logical view.

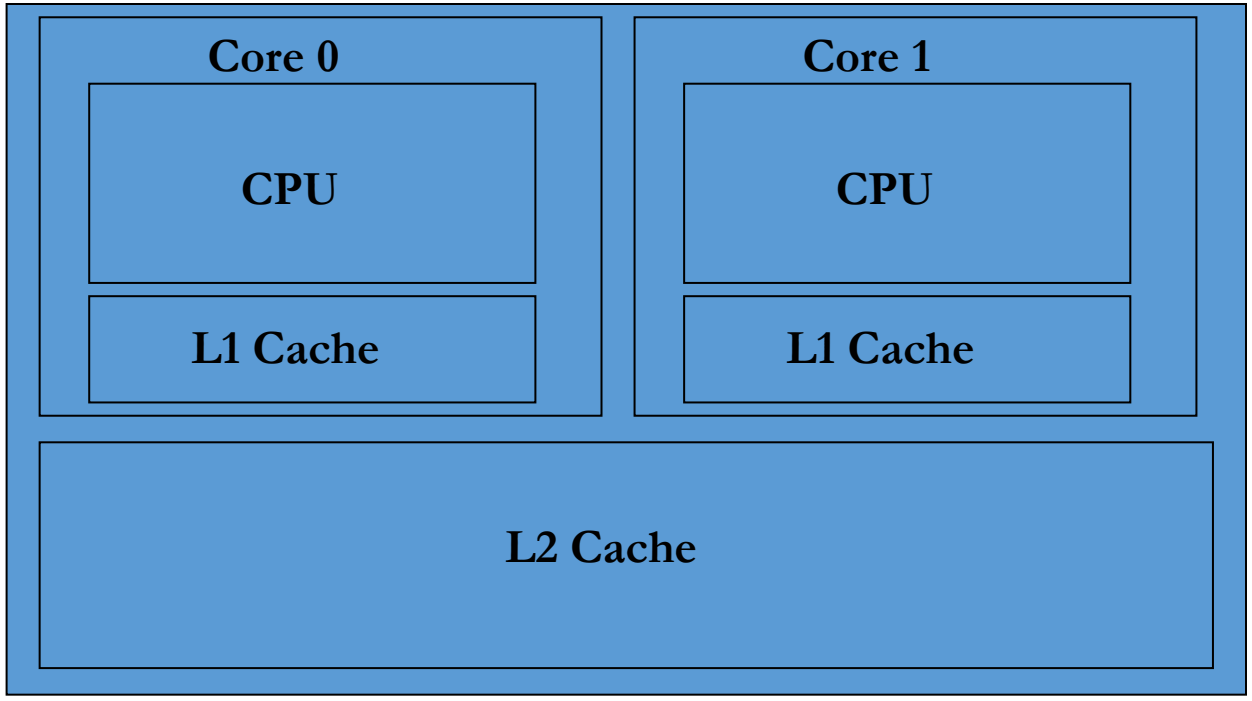
Definitions

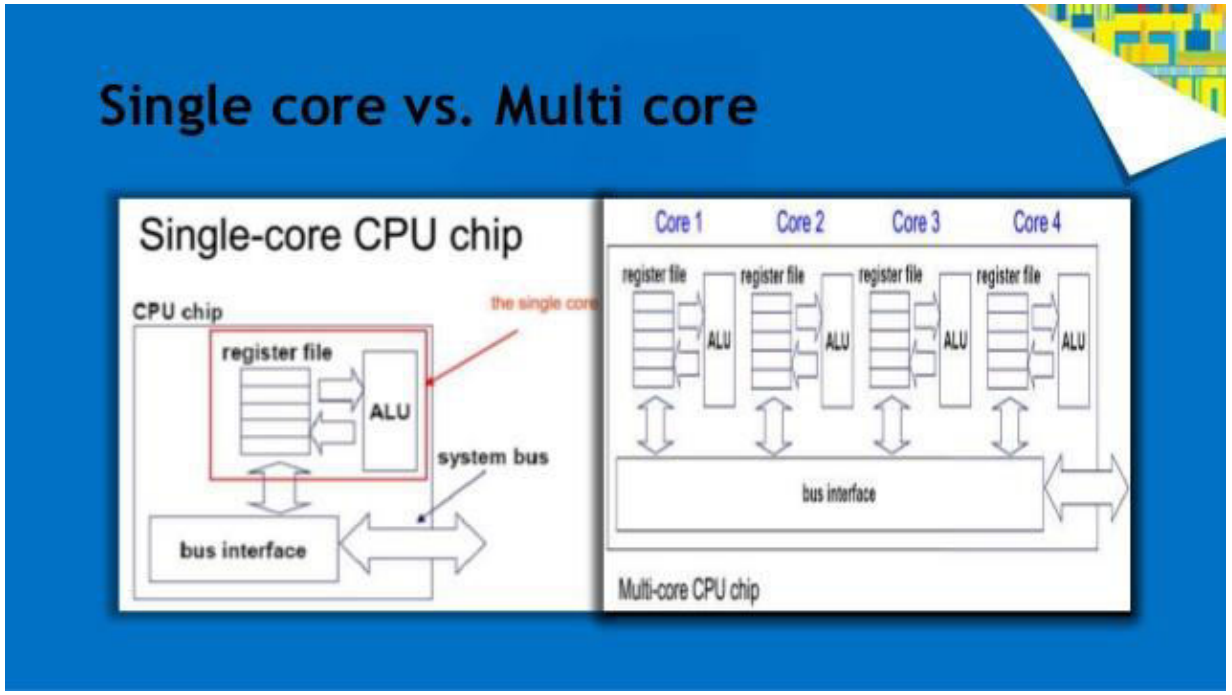
- A **Multi-Core** processor combines two or more independent **cores** into a single package composed of a single integrated circuit (IC), called a die, or more dies packaged together.

  A **dual-core** processor contains two cores, and a **quad-core** processor contains four cores.
- Typically the term **Many-Core** is sometimes used to describe multi-core architectures with an especially high number of cores (tens or hundreds).

Multi core

- Single Chip
- Multiple distinct processing Engine
- E.g.) Shared-cache Dual Core Architecture





Multicore..

Multi core

- Cores in a multi-core device may share a single coherent cache at the highest on- device cache level (e.g. L2 for the Intel Core 2) or may have separate caches (e.g. current AMD dual-core processors).
- The processors also share the same interconnect to the rest of the system.
- Each "core" independently implements optimizations such as superscalar execution, pipelining, and multithreading.
- A system with n cores is effective when it is presented with n or more threads concurrently.

Advantages of Multi-core

- The proximity of multiple CPU cores on the same die allows the cache coherency circuitry to operate at a much higher clock rate than is possible if the signals have to travel off-chip.
- Combining equivalent CPUs on a single die significantly improves the performance of cache snoop (alternative: Bus snooping) operations.
- This means that signals between different CPUs travel shorter distances, and therefore those signals degrade less. These higher quality signals allow more data to be sent in a given time period since individual signals can be shorter and do not need to be repeated as often.

Advantages of Multi-core(cont..)

- The largest boost in performance will likely be noticed in improved response time while running CPU-intensive processes, like antivirus scans, ripping/ burning media (requiring file conversion), or searching for folders.
- For example, if the automatic virus scan initiates while a movie is being watched, the application running the movie is far less likely to be starved of processor power, as the antivirus program will be assigned to a different processor core than the one running the movie playback.

Speed up and Amdahl’s Law

How to calculate parallel time (tp)?

Example 1

Example

Suppose we were to add n numbers on two computers, where each computer adds $n/2$ numbers together, and the numbers are initially all held by the first computer. The second computer submits its result to the first computer for adding the two partial sums together. This problem has several phases:

1. Computer 1 sends $n/2$ numbers to computer 2.
2. Both computers add $n/2$ numbers simultaneously.
3. Computer 2 sends its partial result back to computer 1.
4. Computer 1 adds the partial sums to produce the final result.

Example 1

Computation (for steps 2 and 4):

$$t_{comp} = n/2 + 1$$

Communication (for steps 1 and 3):

$$t_{comm} = (t_{startup} + n/2t_{data}) + (t_{startup} + t_{data}) = 2t_{startup} + (n/2 + 1)t_{data}$$

Example 2

- Suppose we are to add ‘n’ numbers on 3 processors where each processor adds n/3 numbers. The numbers are initially held by the first processor. The second and the third processors submit the results to the first for adding the partial sums. Assuming the $\mathbf{t_{start\text{-}up}}$ and $\mathbf{t_{data}}$ as 25 units and 32 units and ‘n’ as 6000, find the ***computation/communication ratio.*** Is this system setup, efficient? Why or why not?

Example 3

- Assume 0.1% of the runtime of a program is not parallelizable. This program is supposed to run on the Tianhe-2 supercomputer, which consists of 3,120,000 cores. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the parallel speedup on 30,000 and 3,000,000 cores using Amdahl's law?

  Input: f=0.1%, .001

- **For 30000, speed-up is 968**
- **For 3000000, speed-up is1000**

Example 4

- 95% of a program's execution time occurs inside a loop that can be executed in parallel. What is the maximum speedup we should expect from a parallel version of the program executing on 8 CPUs?

$$S \leq \frac{1}{0.05 + (1 - 0.05)/8} \cong 5.9$$

Example 5

- 5% of a parallel program's execution time is spent within inherently sequential code. The maximum speedup achievable by this program, regardless of how many PEs are used, is

$$\lim_{p \to \infty} \frac{1}{0.05 + (1 - 0.05)/p} = \frac{1}{0.05} = 20$$

Example 6

- An oceanographer gives you a serial program and asks you how much faster it might run on 8 processors. You can only find one function amenable to a parallel solution. Benchmarking on a single processor reveals 80% of the execution time is spent inside this function. What is the best speedup a parallel version is likely to achieve on 8 processors?

  Soln: Show that the answer is about 3.3

Example 7

Example 3 (p.40): Suppose that we want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/O 60% of the time, what is the overall speedup gained by incorporating the enhancement?

Answer

$$\text{Fraction}_{\text{enhanced}} = 0.4, \text{Speedup}_{\text{enhanced}} = 10$$

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1-0.4)+\frac{0.4}{10}} = \frac{1}{0.6+0.04} = \frac{1}{0.64} \approx 1.56$$