Introduction to High Performance Computing

Jun Ni, Ph.D. , Associate Professor
Department of Radiology
Carver College of Medicine

Information Technology Services

The University of Iowa, Iowa City

Parallel Computing

- Outline
  - Concept of parallel programming
  - Parallel computing environment and system

Need for Parallel Computing

- Need for computational speed
  - Numerical modeling and simulation of scientific and engineering problems
  - Require repetitive calculation on large amounts of data
  - Achieve computational results within desirable time
  - Integrated to CAD for effective and efficient product processing
  - Real time simulation is need.

Need for Parallel Computing

- Need for computational speed
  - Modeling of DNA structures
  - Forecasting weather
  - Prediction in missile defense system
  - Study in astronomy
  - Grand challenge problems
  - The current computer systems do not meet the need of today's computations.

Need for Parallel Computing

- Example:
  - Weather forecasting system
    - Atmosphere is modeled by mathematical governing equations and numerically divided into many cells in three dimensions
    - Each cell has many physical variable to be computed, such as temperature, pressure, humidity, wind speed and direction, etc.
    - 1mile width by 1 mile long and by 1 mile height, as a cell size, scan to total 10 miles high
    - Global region can be schemed as $5x10^8$ cells.

Need for Parallel Computing

- Example:
  - Weather forecasting system
    - Each cell calculation requires to have 200 floating-point operations, we need 10^{11} floating-point operations. That is just for one computation for an interval time.
    - If we choose time interval is 10 minutes and we predict 10 days , we will have $10x24x60 = 1.44x10^4$ time intervals. Therefore we need about $10^4x10^{11} = 10^{15}$ operations to accomplish the computational task.

Need for Parallel Computing

- Example:
  - Weather forecasting system
    - 1 GFlops/s machine would finish this job in 10 days.
    - In other words, if we want to finish this job in 10 minutes, we need to have a 2TFlops/s supercomputer!
    - In reality, 200 floating-point operations is away not enough to handle the iterative procedures during computation.
    - We need a PFlops/s machine!

Need for Parallel Computing

- Another example:
  - Astronomy study of bodies in space
  - Each body is attracted to each other body by gravity
  - The motion of body can be calculated based on the total forces acted on the body.
  - If there is N bodies, there should be N-1 forces need to be calculated, which requires to have N^2 floating-point operations. For example,
  - Galaxy has 10^{11} stars, we 10^{22} floating-point operations.

Need for Parallel Computing

- Another example:
  - Effective algorithm the number of operations to $N\log_2 N$ calculations. This is we still have $10^{11}\log_2 10^{11}$ floating-point operations.
  - It take 10^9 years to accomplish N^2-algorithm and one year accomplish $N\log^2 N$-algorithm.

The Need for More Computational Power

- Example: suppose that we wish to execute this code in one second:

```
/* x, y, and z are arrays of floats,        */
/* each containing a trillion entries       */
for (i = 0, i < ONE_TRILLION; i++)
        z[i] = x[i] + y[i];
```

The Need for More Computational Power

- A conventional computer would successively fetch `x[i]` and `y[i]` from memory into registers, add them, and store the result in `z[i]`.
- This would require 3×10^{12} copies between memory and registers each second.
- Given the size of the memory and assuming transfers at the speed of light, we would need to fit a word of memory into $\approx 10^{-10}$ m, the size of a relatively small atom.

The Need for More Computational Power

- Unless we figure out how to represent a word with an atom, it will be impossible to build our computer.
- Thus we must increase the number of processors, increasing the number of memory transfers and computations per second.
- Directions in hardware and software.

The Need for More Computational Power

- The system designers must concern themselves with:
  - The design and implementation of an interconnection network for the processors and memory modules.
  - The design and implementation of system software for the hardware.

The Need for More Computational Power

- The system users must concern themselves with:
  - The algorithms and data structures for solving their problem.
  - Dividing the algorithms and data structures into sub problems.
  - Identifying the communications needed among the sub problems.
  - Assignment of sub problems to processors and memory modules.

Need for Parallel Computing

- One way to increase computational speed is use multiple processors operating together on a single problem.
- From the hardware aspect, people can build multiple processor machines, traditionally called supercomputing, or utilize distributed computers to form a cluster.
- Recently such computing is immigrated to Internet by integrate hundreds to thousands of distributed computers together on a global scale “virtual supercomputer” to solve single computational problem. That is called “Grid” computing.

High Performance Computer

- Definition of a high performance Computer
  - a computer which can solve large problems in a reasonable amount of time
- Characterization of high performance computer
  - fast operation of instruction
  - large memory
  - high speed interconnect
  - high speed input/output

High Performance Computer

- How it works?
  - make sequential computation faster.
  - perform computation in parallel.

High Performance Computer

- Available commercial high performance computer?
  - SGI/Cray: Power Challenge, Origin-2000, T3D/T3E
  - HO/Convex: SPP-1200, 2000
  - IBM: SP
  - Tandem:

High Performance Computer

- In theory, we can obtain virtually unlimited computational power.
- But we have ignored how the processors will work together to solve the problem.
- Getting the collection of processors to work together is extremely complex and requires a huge amount of work.

Need for Parallel Computing

- No matter what computer system we put together, we need to split the problem into many parts, and each part is performed by a separate processor in parallel.
- Writing program for such form of computation is known as parallel programming.
- The objective of parallel computing is to significantly increase in performance

Need for Parallel Computing

- The idea is that n computers could provide up to n times the computational speed of a single computer. In other words, the computational job could be completed in1/nth of the time used by a single computer.
- In practical, people would not achieve that expectation, because there is a need of interaction between parts, both for extra data transfer and synchronization of computations.

Need for Parallel Computing

- Never the less, one can achieve substantial improvement, depending upon the problem and amount of parallelism (the way to parallelize the computational job).
- In addition, multiple computers often have more total main memory than a single computer, which enables problems that require larger amounts of main memory to be tackled.

Types of Parallel Computer Systems

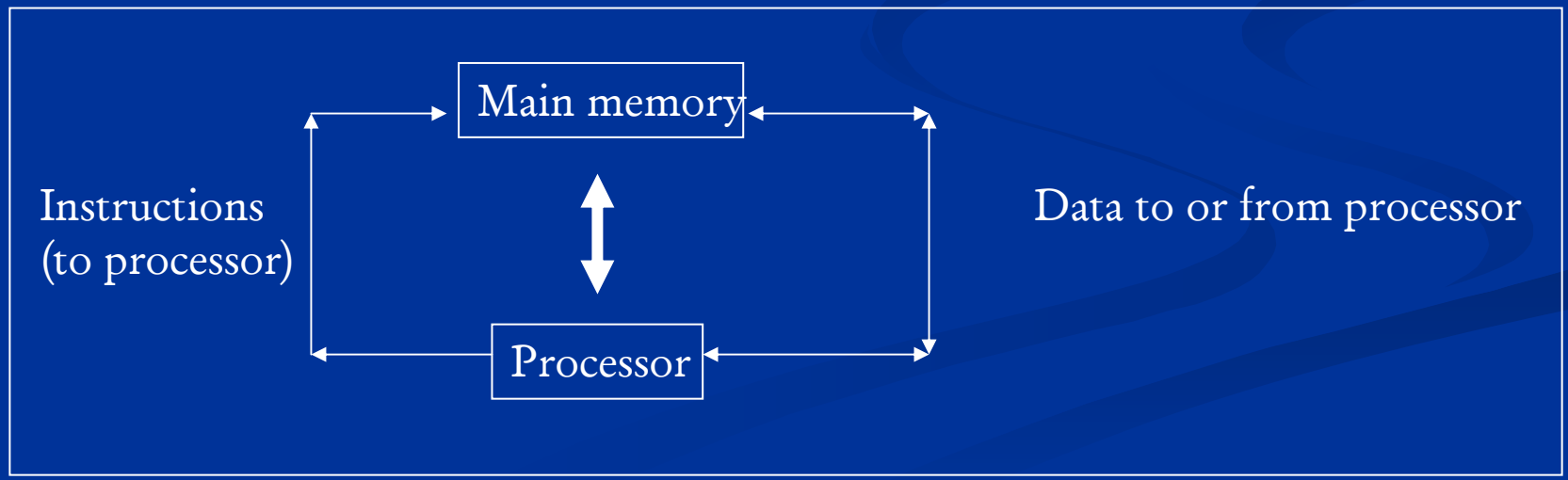
- Single computer with multiple internal processors
- Multiple interconnected computers (cluster system)
- Multiple Internet-connected computers (distributed systems)
- Multiple Internet-connected, heterogeneous, globally distributed systems, in “virtual” organization (grid computing system)

Types of Parallel Computer Systems

- Hardware architecture classification:
  - Single computer with multiple internal processors
  - Multiple interconnected computers (cluster system)
  - Multiple Internet-connected computers (distributed systems)
  - Multiple Internet-connected, heterogeneous, globally distributed systems, in “virtual” organization (grid computing system)

Types of Parallel Computer Systems

- Memory based classification:
  - Shared memory multiprocessor system
    - Supercomputing such as Cray, SGI Origin, ...
    - Conventional computer consists of a processor executing a program stored in a main memory



Shared Memory Systems

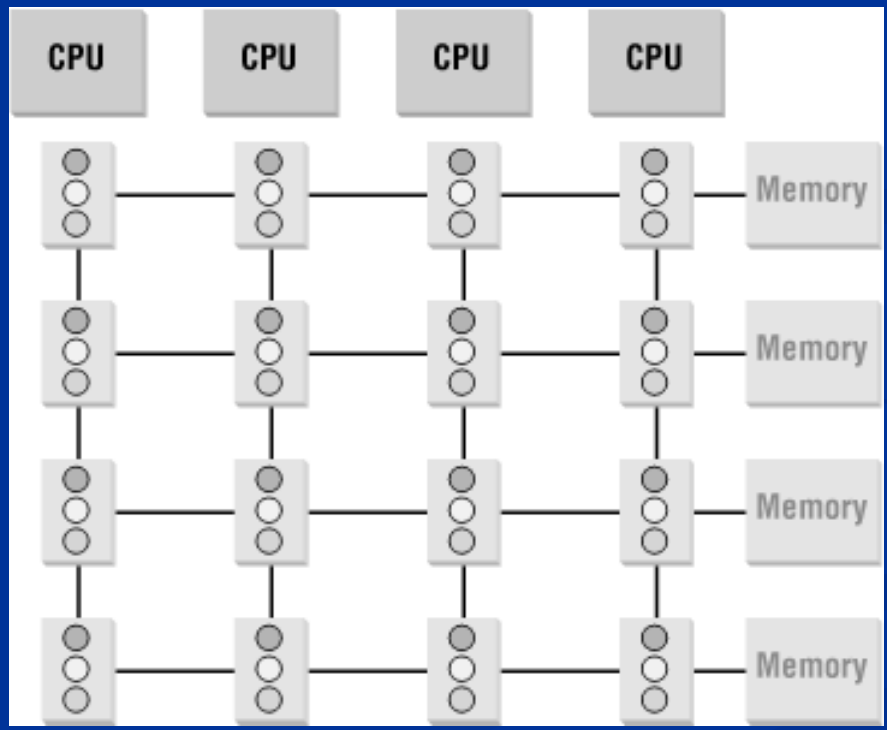
- The simplest shared memory architecture is bus based.
- All processors share a common bus to memory and other devices.
- The bus can become saturated resulting in large delays in the fulfillment of requests.
- Some of the contention is relieved by large caches, however the architecture still does not scale well.
- The SGI Challenge XL is bus based and has only 36 processors.

Shared Memory Systems

- Most shared memory architectures rely on a switch based interconnection network.
- The basic unit of the Convex SPP1200 is a 5 x 5 crossbar switch.
- A crossbar is a rectangular mesh of wires with switches at points of intersection.
- The switches can either allow signals to pass through in both vertical and horizontal simultaneously or they can redirect from horizontal to vertical.

Shared Memory Systems

- Example of 4 x 4 crossbar switch:

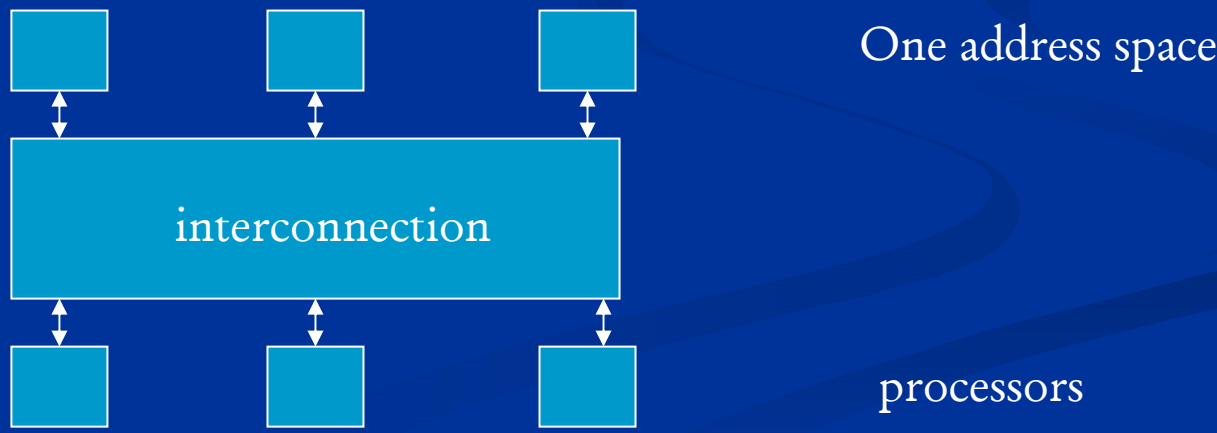


Shared Memory Systems

- With the crossbar switch, communication between two units will not interfere with communication between any other two units.
- Crossbar switches don't suffer from the problems of saturation as in busses.
- Crossbar switches are very expensive as they require mn switches for m processors and n memory units.

Types of Parallel Computer Systems

- Memory based classification:
  - Shared memory multiprocessor system
    - Multiple processors connected to a shared memory with single address space. Multiple processors are connected to the memory though interconnection network



Types of Parallel Computer Systems

- Programming a shared memory multiprocessor involves having executable code stored in the memory for each processor to execute.
- The data for each problem will also be stored in the shared memory.
- Each program could access all the data if need.
- It is desirable to have a parallel programming language which allows the shared variables and parallel code should be declared.

Types of Parallel Computer Systems

- In most of cases, people need to insert special parallel programming library into existing sequential programming codes to perform parallel computing.
- Alternatively, one can introduce threads for individual processor.

Types of Parallel Computer Systems

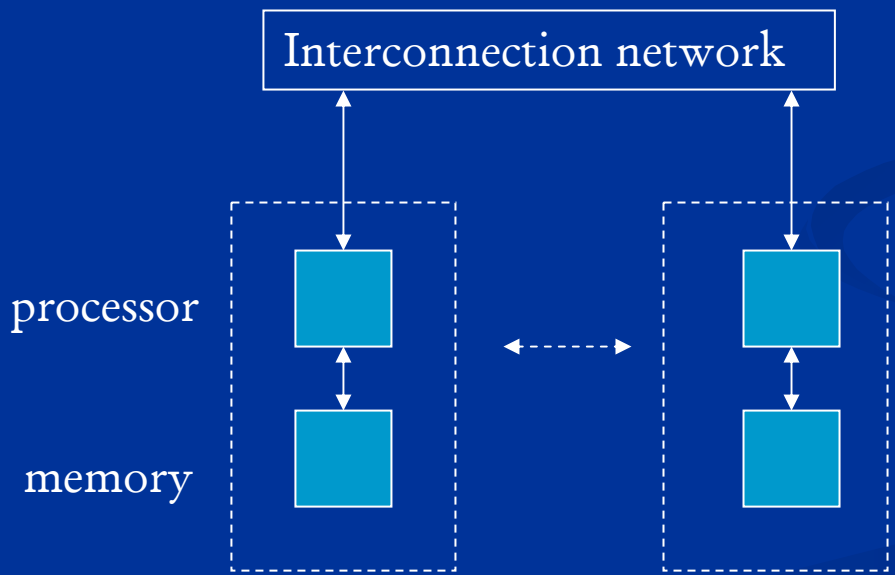
- Distributed memory system or message-passing multi-computer
  - The system is connected with multiple independent computers through an interconnection network.
  - Each computer consists of a processor and local memory that is not accessible by the other processors, since each computer has its own address space.
  - The interconnection network is used to pass messages among the processors.
  - Massages include commands and data that other processor may require for the computations.

Types of Parallel Computer Systems

- Distributed memory system or message-passing multi-computer
  - Such system can be built-in processors with memory, for example like IBM SP system
  - Or can be self-contained computer that could operate independently (PC-LINUX operated cluster) or distributed system through Internet.
  - The traditional way to do parallel programming is to introduce a message-passing library to the sections coded by a sequential-programming language.

Types of Parallel Computer Systems

- Distributed memory system or message-passing multi-computer



Types of Parallel Computer Systems

- Programming message-passing multicomputer still involves dividing the overall problem into parts that are intended to be executed simultaneously to solve the problem.
- The independent parallel subpart of the problem is defined as a process. Therefore, in parallel computing, one can divide a problem into a number of processes.
- One may have multiple processes executed on multiple processors.

Types of Parallel Computer Systems

- If the number of processes is the same as or less than the number of the processors, one can distribute each process to each processor for load balance.
- However, if there were more processes than processors, then more than one process would be executed on one processor, in a time-shared fashion.

Types of Parallel Computer Systems

- Shortcomings of message-passing based parallel programming
  - Require programmers to provide explicit message-passing calls
  - Data are not shared; it must be copied, which limits the applications that require multiple operations across amounts of data.

Types of Parallel Computer Systems

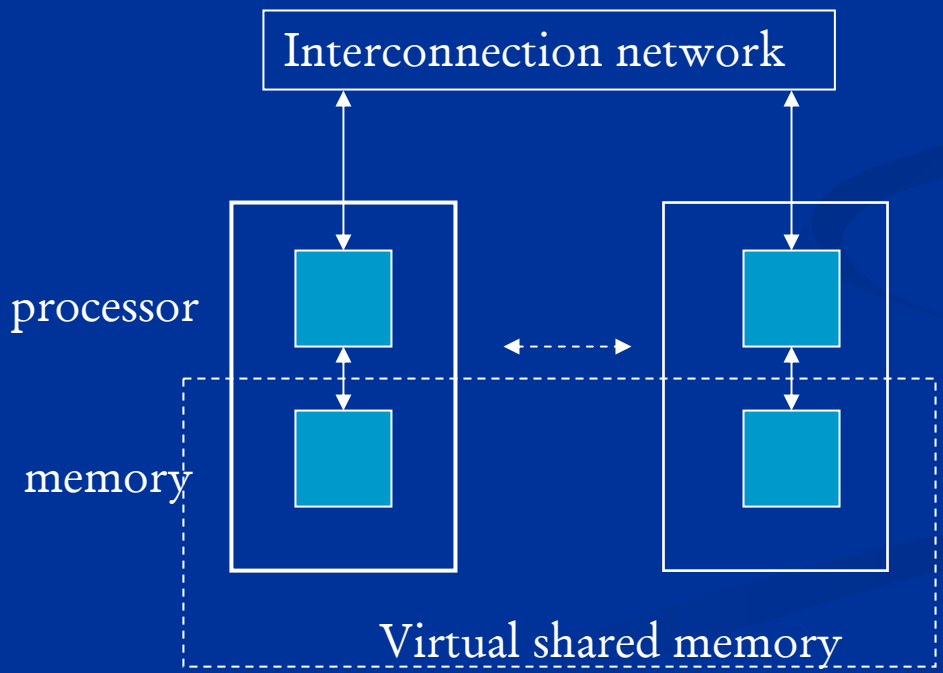
- Advantages
  - Scalable to large system
  - Applicability to computers connected on a network (either inter-networked or global networked)
  - Easy to replace
  - Easy to maintain
  - Cost much cheaper.

Distributed Shared Memory System

- Each processor has access to the whole memory using a single memory address space, although the memory is distributed.
- The technology is also called “virtual shared memory” or distributed memory system

Distributed Shared Memory System

- KSR1 multiprocessor system use such technique



Classification of Instruction stream and data stream

- MIMD and SIMD
  - Each single-instruction stream generated from program operates single data (SISD)
  - Each single-instruction stream generated from program operates multiple data (SIMD)
  - Multiple instruction stream generated form program operates single data (MISD) (not exits).
  - Multiple instruction stream generated form program operates multiple data (MISD)

Classification of Instruction Stream and Data Stream

- Within MIMD, one has
  - Multiple program multiple data structure
  - Single program multiple data structure

Computer Architectures

- The classical von Neumann machine consists of a CPU and main memory.
- The CPU consists of a control unit and an arithmetic-logic unit (ALU).
- The control unit is responsible for directing the execution of instructions.
- The ALU is responsible for carrying out the actual computations.

Computer Architectures

- The CPU contains very fast memory locations called registers.
- Both instructions and data are moved between the registers and memory along a bus.
- The bus is a bottleneck. No matter how fast the CPU is, the speed of execution is limited by the rate at which we can transfer instructions and data between memory and the CPU.

Computer Architectures

- An intermediate memory is introduced called cache.
- Cache is faster than main memory but slower than registers.
- Programs tend to access both instructions and data sequentially.
- Thus a small block of instructions and data in the cache will mean most memory accesses will be from the fast cache rather than the slower main memory.

Computer Architectures

- There are a variety of many different architectures (hardware designs).
- Flynn classified systems according to the number of instruction streams and the number of data streams.

Computer Architectures

- The simplest architecture (typically found in personal computers) is single-instruction single-data (SISD).
- On the opposite extreme is multiple-instruction multiple-data (MIMD) in which multiple autonomous processors operate on their own data.

Computer Architectures (SISD)

- The first extension to CPUs for speedup was pipelining.
- The various circuits of the CPU are split up into functional units which are arranged into a pipeline.
- Each functional unit operates on the result of the previous functional unit during a clock cycle.

Computer Architectures (SISD)

- Suppose that the addition operation was split into the following sequence of operations:
  1. Fetch the operands from memory.
  2. Compare exponents.
  3. Shift one operand.
  4. Add
  5. Normalize the result.
  6. Store Result in memory.

Computer Architectures (SISD)

- Consider the following code:

```
for (i = 0; i < 100; i++)
    z[i] = x[i] + y[i];
```

- While x[0] and y[0] are in stage 4,
  x[1] and y[1] will be in stage 3,
  x[2] and y[2] will be in stage 2,
  and x[3] and y[3] will be in stage 1.

Computer Architectures (SISD)

- Thus when the pipeline is full, we can produce a result every clock cycle, presumably six times faster than without pipelining.

Computer Architectures (SIMD)

- Vector processors perform the same operation on several inputs simultaneously.
- They are considered a variation (not pure) of the SIMD architecture.
- The basic instruction is only issued once for several operands.

Computer Architectures (SIMD)

Compare the Fortran 77 code (sequential):

```
      do 100 i = 1, 100
            z(i) = x(i) + y(i)
  100 continue
```

with the equivalent Fortran 90 code (vector):

```
z(1:100) = x(1:100) + y(1:100)
```

Computer Architectures (SIMD)

- Pure SIMD systems have a single CPU devoted to control and a large collection of subordinate processors each with its own registers.
- Each cycle the control CPU broadcasts an instruction to all of the subordinates.
- Each subordinate either executes the instruction or sits idle.

Computer Architectures (SIMD)

- Consider the following sequence of sequential instructions:

```
for (i = 0; i < 1000; i++)
    if (y[i] != 0.0)
        z[i] = x[i]/y[i];
    else
        z[i] = x[i];
```

Computer Architectures (SIMD)

- Then each subordinate processor would execute these sequence of operations:

Step 1 Test local `y != 0.0`.

Step 2 a. If local `y` was nonzero, `z[i] = x[i]/y[i]`.

b. If local `y` was zero, do nothing.

Step 3 a. If local `y` was nonzero, do nothing.

b. If local `y` was zero, `z[i] = x[i]`.

Computer Architectures (SIMD)

- Notice though that all of the processors are idle in either step two or step three.
- In programs with many conditional branches, it is possible some processors will remain idle for long periods of time.
- Examples of SIMD machines are the MP2 with 16,384 processors and the CM2 with 65,536 processors.

Computer Architectures (MIMD)

- All the processors in MIMD machines are autonomous, possessing a control unit and an ALU.
- Each processor operates on its own pace.
- There is often no global clock and no implicit synchronization.
- There are shared-memory systems and distributed-memory systems.

Distributed Memory Systems

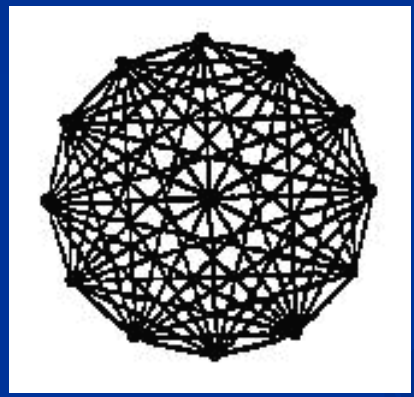
- Distributed memory systems are constructed from nodes in which each processor has its own private memory.
- There are two main types of distributed memory systems: static networks and dynamic networks.

Distributed Memory Systems

- Static networks are constructed so that each vertex corresponds to a node (processor/memory pair).
- There are no switches as vertices in static networks.
- If a there is no direct connection between two nodes, then intermediate nodes would have to forward communication between them.

Distributed Memory Systems

- For performance a fully connected network is desirable.



- But they are impractical to build for more than a few nodes.

Distributed Memory Systems

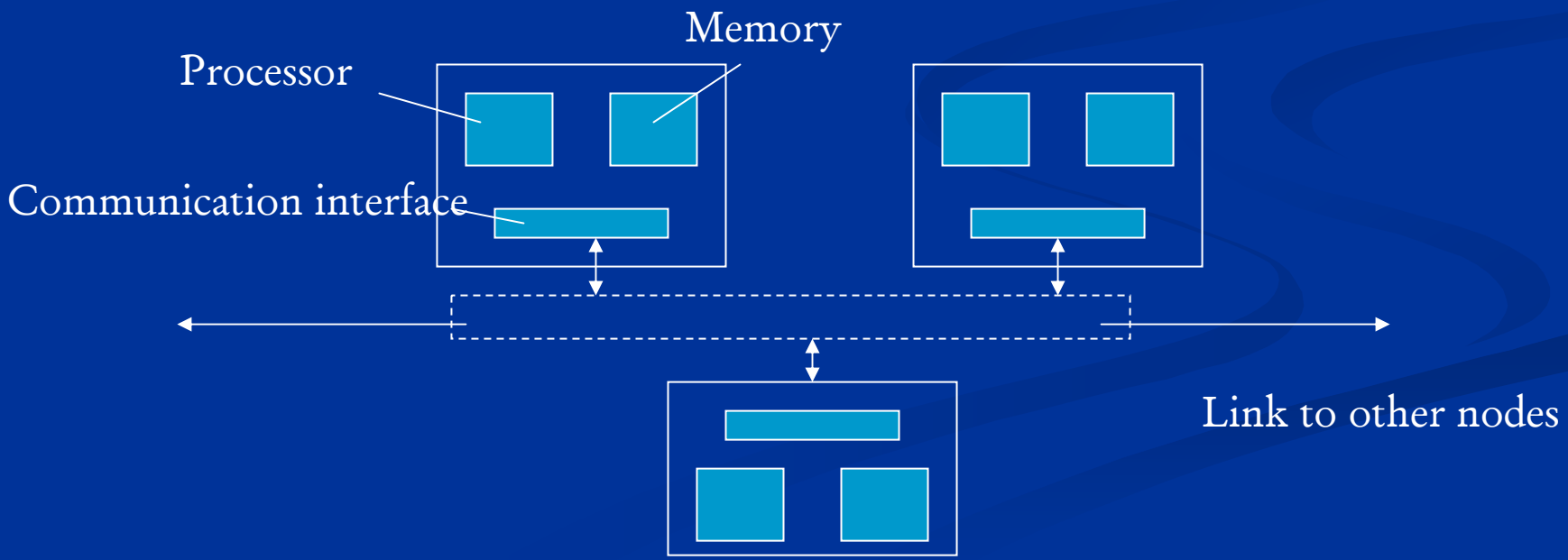
- Static networks can be arranged as a linear array, a ring, hypercube, 2d mesh, 3d mesh, and 2d torus, in increasing order of connectivity.
- The Intel Paragon is a 2D mesh and the Cray T3E is a 3d torus. Both scale to thousands of nodes.

Distributed Memory Systems

- Dynamic networks are constructed so that some vertices correspond to switches that route communications.
- A crossbar switch, as describe earlier, would be optimal but also very expensive.
- Most switches are multistage such that a communication that conflicts with another communication may be delayed.
- Examples are omega networks.

Architectural Features of Message-Passing Multi-computer

- Static network message-passing multi-computer system
  - Having direct fixed physical links between computers (nodes)



Architectural Features of Message-Passing Multi-computer

- Network Criteria (key issues in network design are network bandwidth, network latency, and cost)
  - Bandwidth: number of bits that can be transmitted in unit time (bits/s)
  - Network latency: time make a message transfer through the network
  - Communication latency: total time to send a message, including software overhead and interface delays
  - Message latency or startup time: time required for a zero-length message being sent

Architectural Features of Message-Passing Multi-computer

- Number of links in a path between nodes is also an important consideration as this will be a major factor in determining the delay of a message passing
- Diameter is the minimum number of links between two farther nodes in the network. It is used to determine the worst case delays.

Architectural Features of Message-Passing Multi-computer

- How efficiently a parallel problem can be solved using a multi-computer system within a specific network is extremely important.
- The diameter gives the maximum distance and can be used to find the communication lower bound of some parallel algorithm.
- Bisection width: number of links that must be cut to divide the network into two equal parts.

Architectural Features of Message-Passing Multi-computer

- Interconnection systems
  - Completely connected network: each node has a link to every other node.
    - N nodes could have n-1 links from each node to other n-1 nodes.
    - Therefore, there should be n(n-1)/2 links in all. It is applied to small n. not practical to large n

Architectural Features of Message-Passing Multi-computer

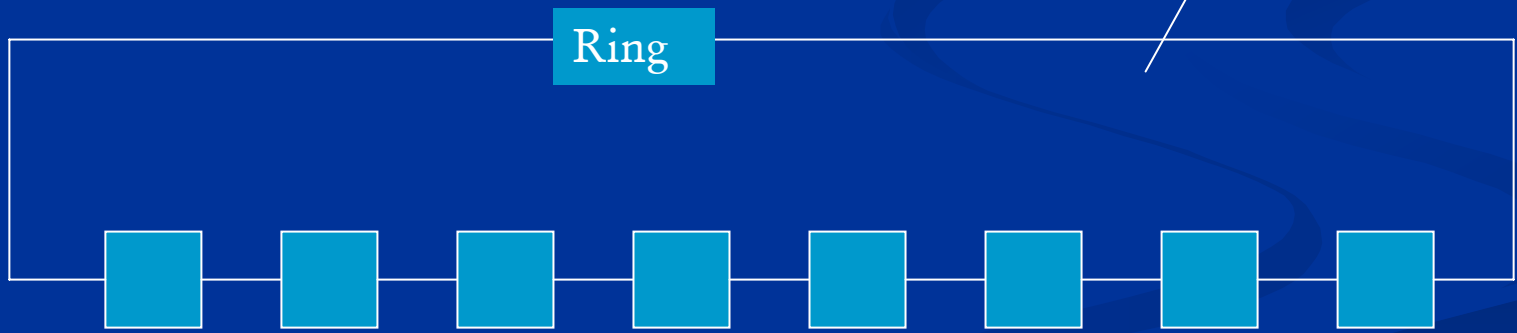
- Interconnection systems
  - Important static networks with restricted interconnection, mainly line/ring, mesh, hypercube, and tree network.
- Line/Ring: each node has two links and link only to neighboring node
  - N-node ring requires n links
  - Two end node are farthest away in a line and hence the diameter is n-1
  - Routing algorithm is necessary to find routes between nodes that re not directly connected, if the network do not provide complete interconnections.

Line

N = 8
Number of links 8
Diameter 7

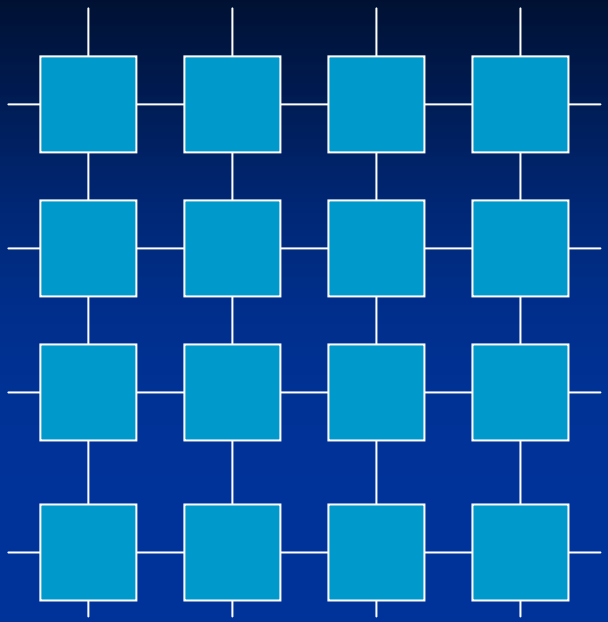
Ring

N = 8
Number of links 8
Diameter n/2 = 4



Architectural Features of Message-Passing Multi-computer

- Mesh: 2-dimensional mesh; each node connected to four nearest nodes the diameter of sqrt(n) by sqrt(n) is 2sqrt(n-1)
- Free-node can be linked to form a torus

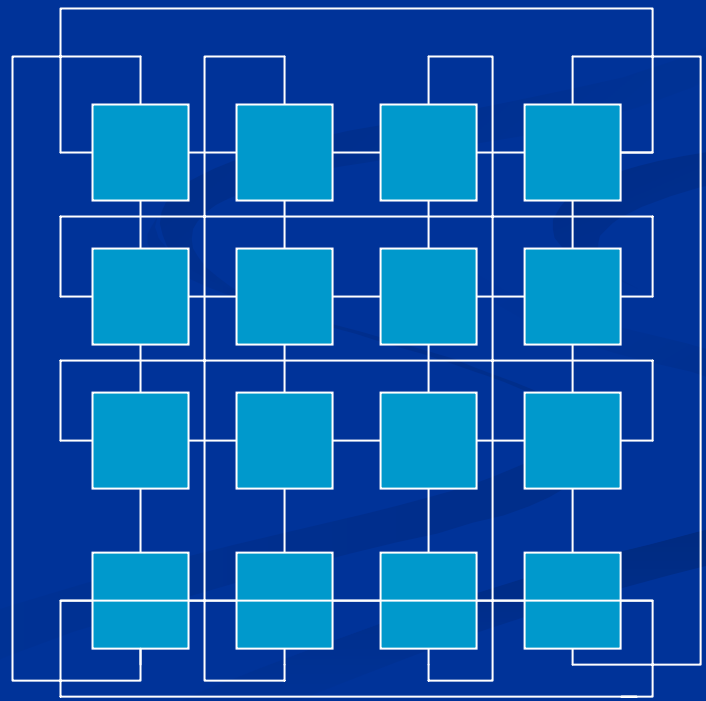


Mesh

N = 16
Links 21
Diameter 2*(sqrt(16)-1) = 6

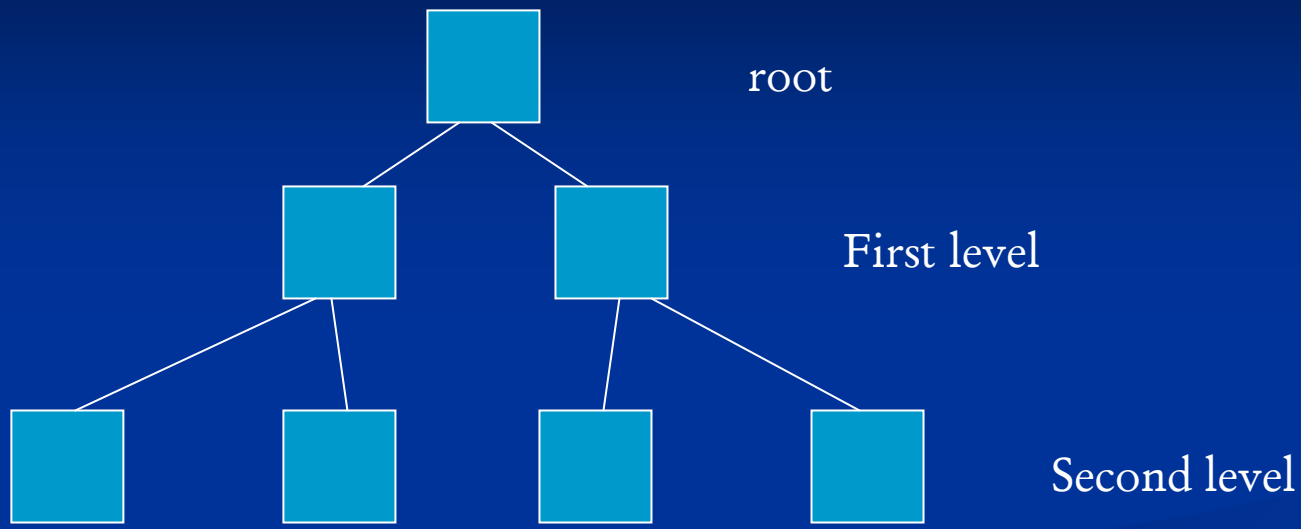
Torus

N = 16
Links 32
Diameter 4



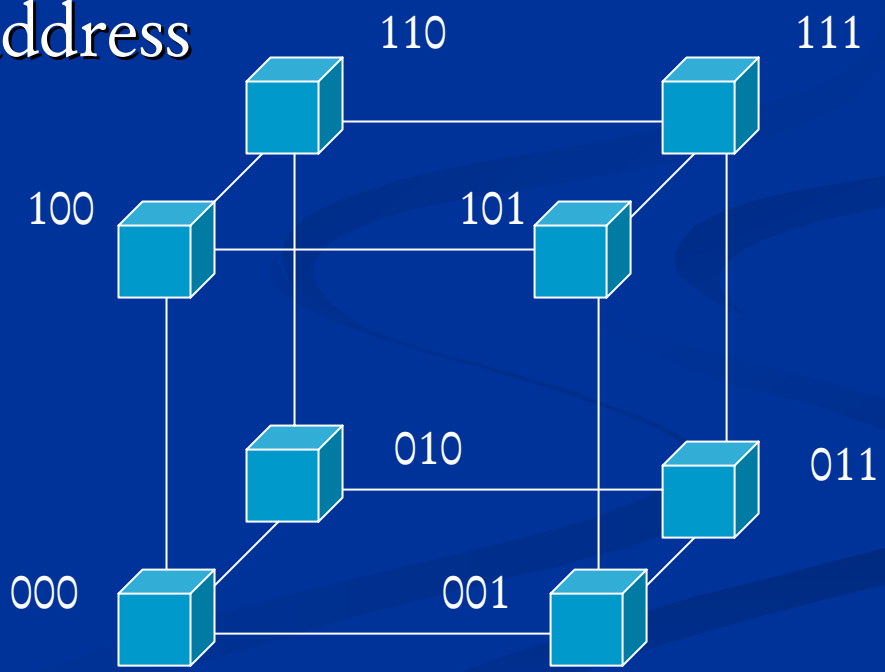
Architectural Features of Message-Passing Multi-computer

- Tree Network: binary network or hierarchy tree network; each node has two links to two nodes.
- root level: one node
- First level: two nodes
- Second level: four nodes
- ...
- jth level: 2^{j+1}-1 nodes
- CM5 system deploys such architecture


root
First level
Second level

Architectural Features of Message-Passing Multi-computer

- Hypercube Network (d-dimension)
  - Use d-bit binary address
  - Diameter is $\log_2 n$
  - Caltech's Cosmic Cube
  - Minimum distance deadlock free



Architectural Features of Message-Passing Multi-computer

- Embedding
  - Applied to static network
  - Describes mapping nodes of one network onto another network
  - Example: ring embedded into mesh; mesh can be embedded into a torus
  - Dilation is uded to indicate the quality of the embedding. Dilation is the maximum number of links in the embedding network corresponding to one link in embedding network

Architectural Features of Message-Passing Multi-computer

- Communication methods
  - In many cases, it is often to route a message through intermediate nodes from the source node to the destination node.
  - Two basic ways: circuit switching and packet switching

Architectural Features of Message-Passing Multi-computer

- Circuit switching system: establishing a path and maintain all the links in the path for the message to pass, uninterrupted, from source to destination, and links are reserved, until the message is complete.
- Packet switching, message is divided into packets of information, each includes the source and destination address for routing the packet through the interconnection network.

Architectural Features of Message-Passing Multi-computer

- Store-and-forward packet switching and its latency
- Wormwhole routing was introduced to reduce the size of the buffer and decrease the latency.

- The concept of deadlock and livelock
- Input and output

Networked Computers as a Multi-Computer Platform

- Cluster of workstation (COWs) and Network of workstations (NOWs) offers a very attractive alternative to expensive supercomputers and parallel computing system for HPC.
- Advantages:
  - Low cost
  - Portable to be incorporated with lately developed processor
  - Existing software can be used and modified

Networked Computers as a Multi-Computer Platform

- Ethernet packet transmission
- Point-to-point communication in high-performance parallel interface
- Commons with static network multi-computer
- Communication delay in networked multi-computer system will be much greater than the static networked multi-computer system.
- Strong requirement for job balance due to different speed of distributed platform.

Communication and Routing

- When two nodes can't communicate directly, they must communicate through other nodes.
- The nodes through which the communication occurs defines the route the messages take.
- Most systems use a deterministic shortest path routing algorithm.

Communication and Routing

- There are two methods nodes can use in relaying messages.
- Store-and-forward routing is used when an intermediate node reads in the entire message before forwarding it.
- Cut-through routing occurs when an intermediate node immediately forwards each identifiable piece of the message (packet).

Communication and Routing

- Cut-through routing requires less memory because only a packet at a time is stored.
- Cut-through routing is also faster because it does not wait on all the packets of the message before forwarding them.
- Therefore cut-through routing is preferred and most commonly used.

Communication and Routing

- A process is an instance of a program or subprogram executing autonomously on a processor.
- Processes can be considered running or blocked.
- A process is running when its instructions are currently being executed on a processor.
- A process is blocked when the operating system has not scheduled it to run on a processor, usually because it is waiting for something to be done (or message received).

Communication and Routing

- All processes have a parent, which is the process that created (spawned) it.
- Processes can have children, which are processes they created (spawned).
- Processes are typically spawned through a combination of the `fork()` and `exec()` UNIX system calls.

Potential for Increase Computational Speed

- Process: Divide computation into tasks or processes that are executed simultaneously.
- Size of process can be described by its granularity.
  - In coarse granularity, each process contains a large number of sequential instructions and takes a substantial time to execute.
  - In fine granularity, a process may have a few or one instruction

Potential for Increase Computational Speed

- Sometimes, granularity is defined as the size of the computation between communication or synchronization points
- In general, we want to increase granularity to reduce the costs of process creation and inter-process communication, which likely reduce the number of processes and parallelism
- For message passing, it is very important to reduce communication laency.

$$\text{Granularity} = \frac{\text{Computation time} \quad T_{comp}}{\text{Communication time} \quad T_{comm}}$$

In domain decomposition, we want to increase the size of data (sub-domain), hence, decrease process number and decrease communication loss.

Decrease Processors to be used.

Design a parallel algorithm which can easily vary the granularity, which we call "scalable design"

$$\text{Speedup, } S(n) = \frac{\text{Execution time used in single processor, } Ts}{\text{Execution time used in multi processors } Tm}$$

- A Measure of relative performance between a multiprocessor system and a single Processor system.
- Used to compare a parallel solution with a sequential solution.
- The algorithms for a parallel implementation and sequential implementation are usually different.
- In theoretical analysis, we use

$$\text{Speedup, } S(n) = \frac{\text{No. of computational steps using one processor}}{\text{No. of parallel computational steps with n processors}}$$

- Example: a parallel sorting algorithm requires 4n steps and a sequential algorithm requires nlogn steps. The speedup is (1/4)logn.

- The maximum speedup is n with n processors (linear speedup.

- If the parallel algorithm did not achieve better than n times the speedup over the current sequential algorithm, the parallel algorithm can certainly be emulated on a single processor.
- It suggests that the original sequential algorithm was not optimal.
- The maximum speedup would be achieved if the computation can be exactly divided into equal during processes. One process is mapped onto one processor (no overhead), i.e.,

  $S(n) = Ts/(Ts/n) = n$

  It is called supperlinear sppedup.

- $S(n) > n$ maybe seen on occasion, but usually this is due to using a suboptimal sequential algorithm or some unique feature of the architecture that favors the parallel formation.
- Reasons for superlinear speedup phenomena
  - Extra memory: total memory in multiprocessor computer is large than the single processor system and it can hold more of the problem data at any larger than that in the single processor system.

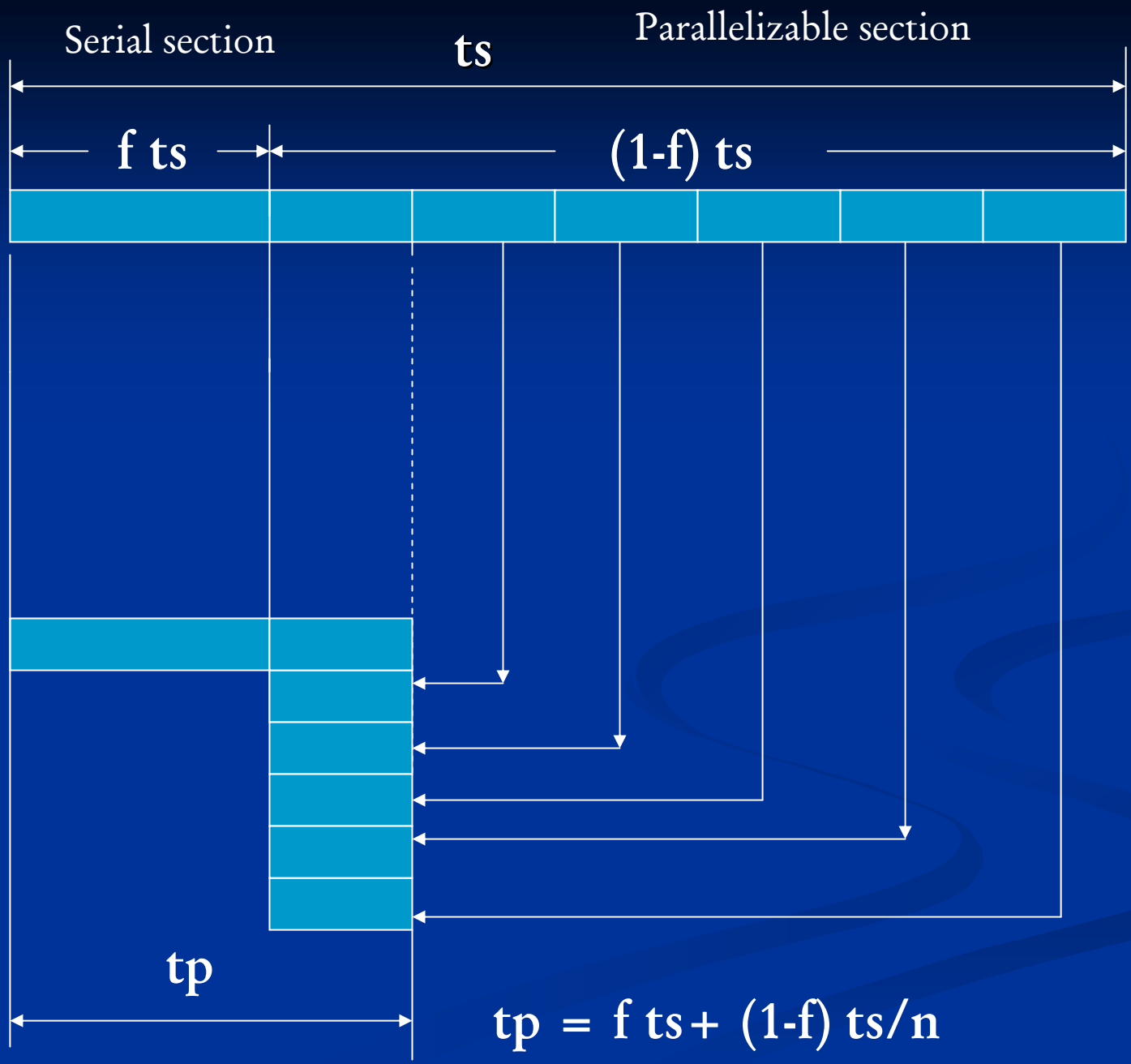
- Some part of a computation cannot be divided at all into concurrent processes and must be performed serially.
- Especially initialization period for data variables declaration or data value input.
- It is better just let one processor to do the initialization job, before submit to concurrent sub task.

- Overhead in parallel version which limit speedup:
  - Periods, when not all the processors can be performing useful work and are idle, including only one processor's activity for initialization and input/ouput.
  - Extra computations in the parallel version not appearing in the sequential version
  - Communication time for sending/receiving message

- Maximum speedup:
- If f is the part of computation that can not be divided into concurrent tasks and if there is no overhead incurs when computation is divided into concurrent parts, the time to perform the computation with n processors is

  $$\text{Time} = f\,t_s + (1-f)\,t_s/n$$

  where t_s is the execution time on single processor


Serial section
ts
Parallelizable section
f ts
(1-f) ts
tp
tp = f ts + (1-f) ts/n

$$tp = f\,ts + (1-f)\,ts/n$$

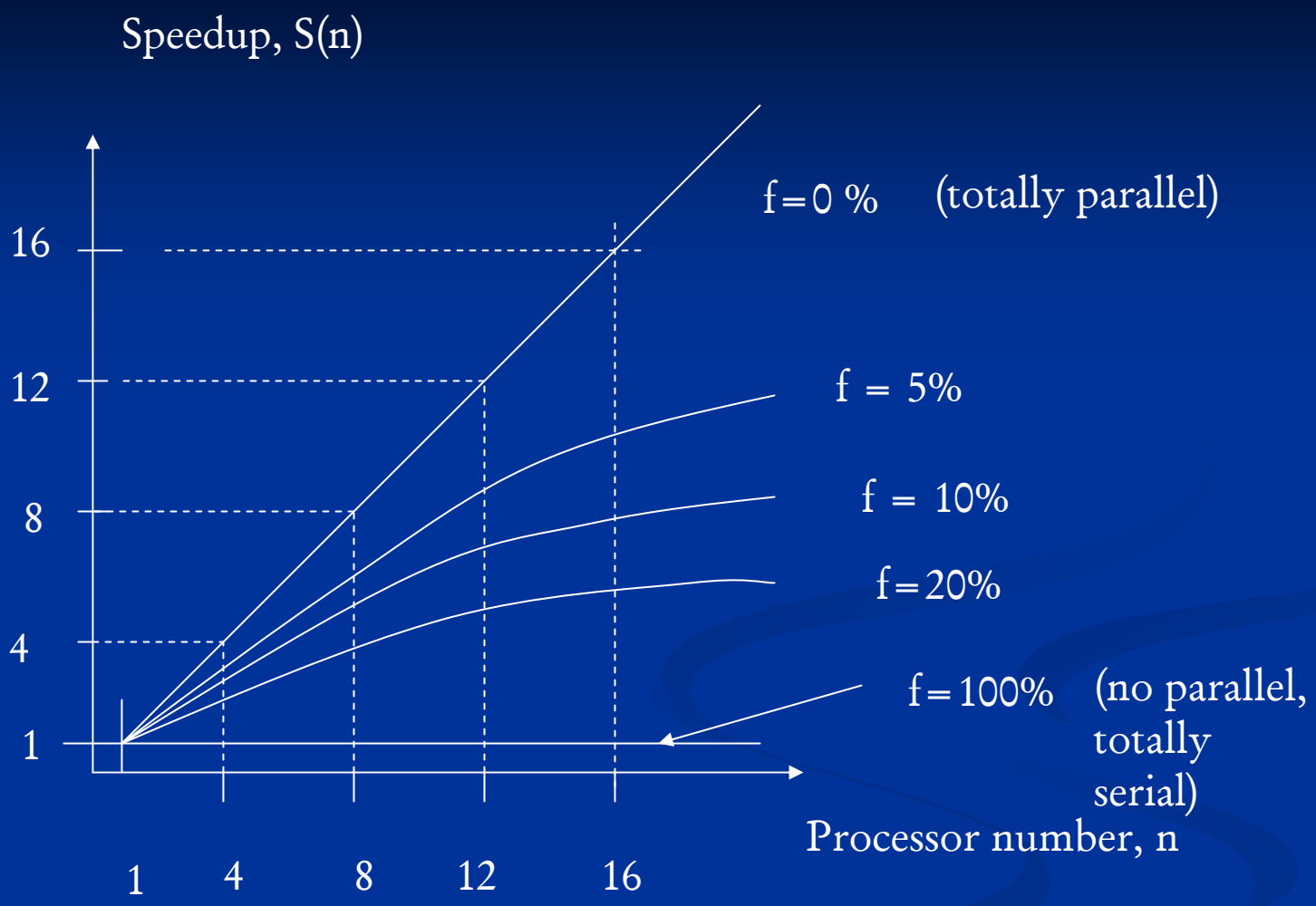
$$\text{Speedup} = \frac{ts}{tp} = \frac{ts}{f\,ts + (1-f)\,ts/n}$$

$$= \frac{n}{1 + (n-1)\,f}$$

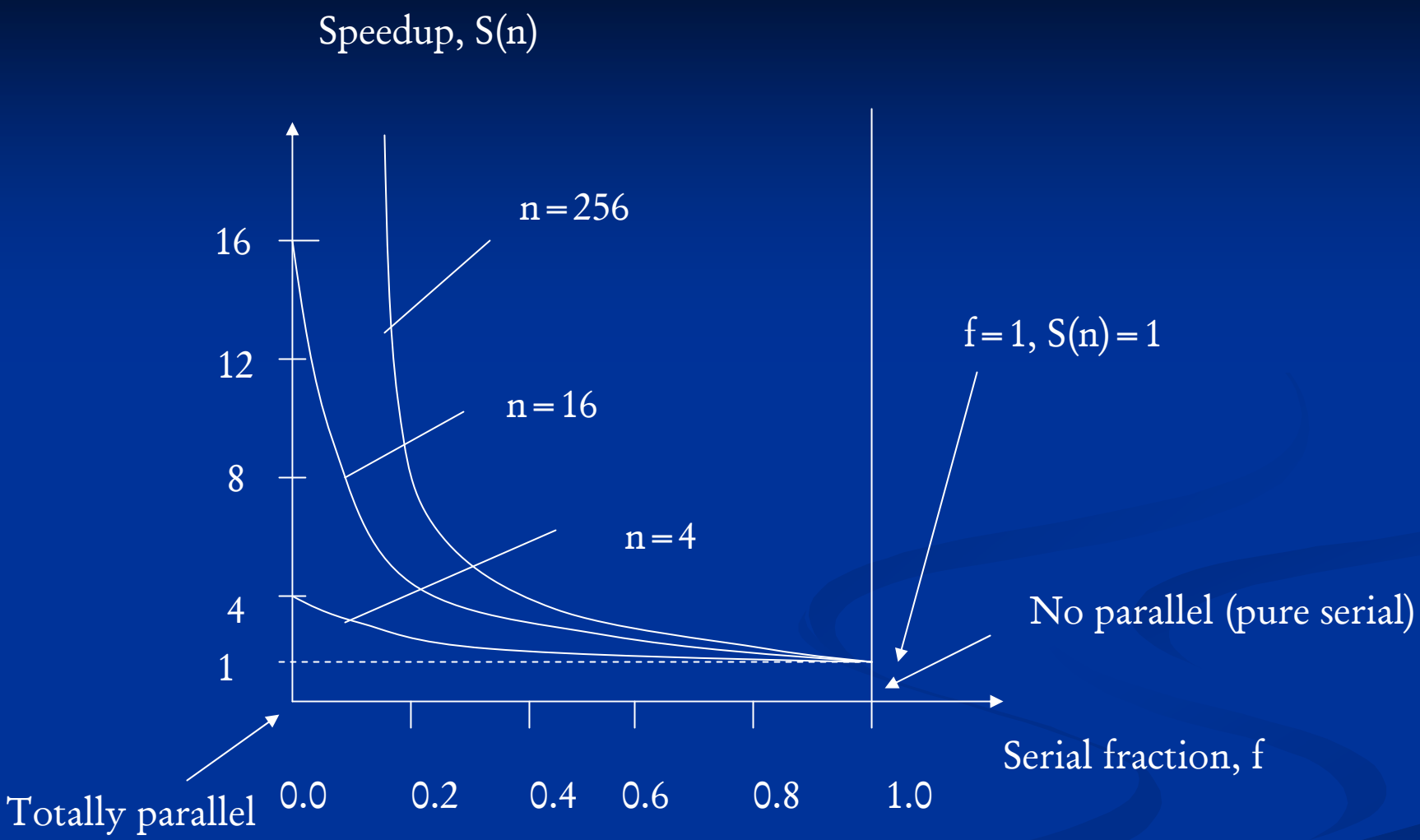
Amdahl's law

- The processor number is increased, one has $S(n) = 1/f$
- The speedup in only dependent on the fraction of series computation portion.

- From the law, we can also see that
  - Even the problem can be totally parallelized, that is $f = 0$, one has the speedup $S(n) = 1$



Speedup vs. number of processors



Speedup, S(n) vs. serial fraction, f

- Efficiency:
  - System efficiency, E is defined as

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

$$= \frac{ts}{tp \times n} = \frac{ts}{[\, f\, ts + (1-f)\, ts/n \,]\, n}$$

$$= \frac{S(n) \times 100\%}{n} = \frac{1}{f\, n + (1-f)} \times 100\%$$

- Cost:
  - is defined as

Cost =Execution time x (total number of processors used)

Cost of a sequential computation is simply its execution time ts.

Cost of parallel computation is tp x n

$= [f\ ts + (1-f)\ ts/n] \times n = f\ ts\ n + (1-f)\ ts$

$= (ts \times n)/S(n) = ts/E$

- Gustafson's Law:
  - IN practice a large multiprocessor usually allows a larger size of problem. Therefore, the problem size is not independent of the number of processors.
  - It is assume that serial section of the code does not increase as the problem size.
  - Introduce scalable speedup factor
  - s is the fractional time for executing the serial part of computation and p is the fractional time for executing the parallel part of the computation on a single processor
  - $S(n) = n + (1-n)\, s$