What is Parallel Computing?

Parallel Computing is several processing elements working simultaneously to solve a problem faster.

What is Parallel Computing?

Parallel Computing is several processing elements working simultaneously to solve a problem faster. The primary consideration is *elapsed time*, not throughput or sharing of resources at remote locations.

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook
- Databases

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook
- Databases
- Weather prediction

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook
- Databases
- Weather prediction
- Picture generation (movies!)

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook
- Databases
- Weather prediction
- Picture generation (movies!)
- Economics Modeling (Wall Street :-()

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook
- Databases
- Weather prediction
- Picture generation (movies!)
- Economics Modeling (Wall Street :-()
- Simulations in a wide range of areas ranging from science to arts

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook
- Databases
- Weather prediction
- Picture generation (movies!)
- Economics Modeling (Wall Street :-()
- Simulations in a wide range of areas ranging from science to arts
- Air traffic control
- Airline Scheduling

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook
- Databases
- Weather prediction
- Picture generation (movies!)
- Economics Modeling (Wall Street :-()
- Simulations in a wide range of areas ranging from science to arts
- Air traffic control
- Airline Scheduling
- Testing (Micron...)

Who Needs Parallel Computing?

- Googling (one of the largest parallel cluster installations)
- Facebook
- Databases
- Weather prediction
- Picture generation (movies!)
- Economics Modeling (Wall Street :-()
- Simulations in a wide range of areas ranging from science to arts
- Air traffic control
- Airline Scheduling
- Testing (Micron...)
- You!

Serial Computer Model: The von Neumann Model

A von Neumann computer consists of a random-access memory (RAM), a read only tape, a write only output tape, and a central processing unit (CPU) that stores a program that cannot modify itself. A RAM has instructions like load, store, read, write, add, subtract, test, jump, and halt. There is a uniform cost criterion in that each instruction takes only one unit of time. The CPU executes instructions of the program sequentially.

How to classify Parallel Computers?

Parallel computers can be classified by:

- type and number of processors
- interconnection scheme of the processors
- communication scheme
- input/output operations

Models of Parallel Computers

There is no single accepted model primarily because the performance of parallel programs depends on a set of interconnected factors in a complex fashion that is machine dependent.

- Flynn's classification
- Shared memory model
- Distributed memory model (a.k.a. Network model)

Flynn's classification

- SISD: Single Instruction Single Data

Flynn's classification

- SISD: Single Instruction Single Data
- SIMD: Single Instruction Multiple Data

Flynn's classification

- SISD: Single Instruction Single Data
- SIMD: Single Instruction Multiple Data
- MIMD: Multiple Instruction Multiple Data

Flynn's classification

- SISD: Single Instruction Single Data
- SIMD: Single Instruction Multiple Data
- MIMD: Multiple Instruction Multiple Data
- MISD: Multiple Instruction Single Data

Shared Memory Model

- Each processor has its own local memory and can execute its own local program.

Shared Memory Model

- Each processor has its own local memory and can execute its own local program.
- The processors can communicate through shared memory.

Shared Memory Model

- Each processor has its own local memory and can execute its own local program.
- The processors can communicate through shared memory.
- Each processor is uniquely identified by an index called a processor number or processor ID, which is available locally.

Shared Memory Model

- Each processor has its own local memory and can execute its own local program.
- The processors can communicate through shared memory.
- Each processor is uniquely identified by an index called a processor number or processor ID, which is available locally.
- Two models of operation:
  - Synchronous $\rightarrow$ All processors operate synchronously under the control of a common clock. (Parallel Random Access Machine, PRAM)

Shared Memory Model

- Each processor has its own local memory and can execute its own local program.
- The processors can communicate through shared memory.
- Each processor is uniquely identified by an index called a processor number or processor ID, which is available locally.
- Two models of operation:
  - Synchronous $\rightarrow$ All processors operate synchronously under the control of a common clock. (Parallel Random Access Machine, PRAM)
  - Asynchronous $\rightarrow$ Each processor operates under a separate clock. In this mode its the programmers responsibility to synchronize, or set the appropriate synchronization points.

Shared Memory Model

- Each processor has its own local memory and can execute its own local program.
- The processors can communicate through shared memory.
- Each processor is uniquely identified by an index called a processor number or processor ID, which is available locally.
- Two models of operation:
  - Synchronous $\rightarrow$ All processors operate synchronously under the control of a common clock. (Parallel Random Access Machine, PRAM)
  - Asynchronous $\rightarrow$ Each processor operates under a separate clock. In this mode its the programmers responsibility to synchronize, or set the appropriate synchronization points.
- This model is MIMD

Variations of the Shared Memory Model

- Algorithms for the PRAM model are usually of SIMD type, that is all processors are executing the same program such that during each time unit all processors execute the same instruction but with different data in general.
- Variations of the PRAM:

Variations of the Shared Memory Model

- Algorithms for the PRAM model are usually of SIMD type, that is all processors are executing the same program such that during each time unit all processors execute the same instruction but with different data in general.
- Variations of the PRAM:
  - EREW(exclusive read, exclusive write)

Variations of the Shared Memory Model

- Algorithms for the PRAM model are usually of SIMD type, that is all processors are executing the same program such that during each time unit all processors execute the same instruction but with different data in general.
- Variations of the PRAM:
  - EREW(exclusive read, exclusive write)
  - CREW(concurrent read, exclusive write)

Variations of the Shared Memory Model

- Algorithms for the PRAM model are usually of SIMD type, that is all processors are executing the same program such that during each time unit all processors execute the same instruction but with different data in general.
- Variations of the PRAM:
  - EREW(exclusive read, exclusive write)
  - CREW(concurrent read, exclusive write)
  - CRCW(concurrent read, concurrent write)

Variations of the Shared Memory Model

- Algorithms for the PRAM model are usually of SIMD type, that is all processors are executing the same program such that during each time unit all processors execute the same instruction but with different data in general.
- Variations of the PRAM:
  - EREW(exclusive read, exclusive write)
  - CREW(concurrent read, exclusive write)
  - CRCW(concurrent read, concurrent write)
    - Common CRCW: All processors in a machine using CRCW writing to a certain location must write the same data or there is an error.
    - Priority CRCW: The processor, among all processors attempting to write to the same location at the same time, with the highest priority succeeds in writing.
    - Arbitrary CRCW: Arbitrary processor succeeds in writing.

Distributed Memory Model (a.k.a Network Model)

A **network** is a graph $G = (N, E)$, where N and E are sets, with each node $i \in N$ and each edge (i,j) $\in$ E represents a two way communication link between processors i and j. There is no shared memory, though each processor has local memory. The operation of the network may be synchronous or asynchronous.

Distributed Memory Model (a.k.a Network Model)

A **network** is a graph $G = (N, E)$, where N and E are sets, with each node $i \in N$ and each edge (i,j) $\in$ E represents a two way communication link between processors i and j. There is no shared memory, though each processor has local memory. The operation of the network may be synchronous or asynchronous.

Message passing model: Processors coordinate activity by `sending` and `receiving` messages. A pair of communicating processors do not have to be adjacent. Each processor "acts" as a network node. The process of delivering messages is called *routing*. The network model incorporates the topology of the underlying network.

Characteristics of Network Model

Diameter Maximum distance between any two processors in the network. Distance is the number of links (hops) in between the two processors along the shortest path.

Characteristics of Network Model

Diameter: Maximum distance between any two processors in the network. Distance is the number of links (hops) in between the two processors along the shortest path.

Connectivity: Minimum number of links that must be cut before a processor becomes isolated. (same as minimum degree)

Characteristics of Network Model

Diameter Maximum distance between any two processors in the network. Distance is the number of links (hops) in between the two processors along the shortest path.

Connectivity Minimum number of links that must be cut before a processor becomes isolated. (same as minimum degree)

Maximum degree - Maximum number of links that any processor has. This would also be the number of ports that a processor must have.

Characteristics of Network Model

Diameter Maximum distance between any two processors in the network. Distance is the number of links (hops) in between the two processors along the shortest path.

Connectivity Minimum number of links that must be cut before a processor becomes isolated. (same as minimum degree)

Maximum degree - Maximum number of links that any processor has. This would also be the number of ports that a processor must have.

Cost Total number of links in the network. This is more important than number of processors because connections are what make up the bulk of the cost in a network. (Can be found by summing the degrees of all nodes, then dividing by two.)

Characteristics of Network Model

Diameter Maximum distance between any two processors in the network. Distance is the number of links (hops) in between the two processors along the shortest path.

Connectivity Minimum number of links that must be cut before a processor becomes isolated. (same as minimum degree)

Maximum degree - Maximum number of links that any processor has. This would also be the number of ports that a processor must have.

Cost Total number of links in the network. This is more important than number of processors because connections are what make up the bulk of the cost in a network. (Can be found by summing the degrees of all nodes, then dividing by two.)

Bisection width - Minimum number of links that have to be cut to bisect the network, that is, into two components, one with $\lfloor n/2 \rfloor$ nodes and the other with $\lceil n/2 \rceil$ nodes.

Characteristics of Network Model

Diameter Maximum distance between any two processors in the network. Distance is the number of links (hops) in between the two processors along the shortest path.

Connectivity Minimum number of links that must be cut before a processor becomes isolated. (same as minimum degree)

Maximum degree - Maximum number of links that any processor has. This would also be the number of ports that a processor must have.

Cost Total number of links in the network. This is more important than number of processors because connections are what make up the bulk of the cost in a network. (Can be found by summing the degrees of all nodes, then dividing by two.)

Bisection width - Minimum number of links that have to be cut to bisect the network, that is, into two components, one with $\lfloor n/2 \rfloor$ nodes and the other with $\lceil n/2 \rceil$ nodes.

Symmetry Does the graph look the same at ALL nodes? Helpful in reliability and code writing. A qualitative rather than a quantitative measure.

Characteristics of Network Model

Diameter Maximum distance between any two processors in the network. Distance is the number of links (hops) in between the two processors along the shortest path.

Connectivity Minimum number of links that must be cut before a processor becomes isolated. (same as minimum degree)

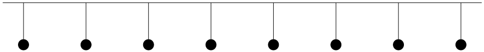
Maximum degree - Maximum number of links that any processor has. This would also be the number of ports that a processor must have.

Cost Total number of links in the network. This is more important than number of processors because connections are what make up the bulk of the cost in a network. (Can be found by summing the degrees of all nodes, then dividing by two.)

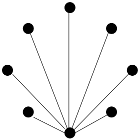
Bisection width - Minimum number of links that have to be cut to bisect the network, that is, into two components, one with $\lfloor n/2 \rfloor$ nodes and the other with $\lceil n/2 \rceil$ nodes.

Symmetry Does the graph look the same at ALL nodes? Helpful in reliability and code writing. A qualitative rather than a quantitative measure.

I/O bandwidth Number of processors with I/O ports. Usually "outer" processors are the only ones with I/O capabilities to reduce cost.

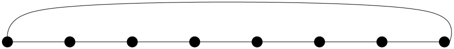


bus−based

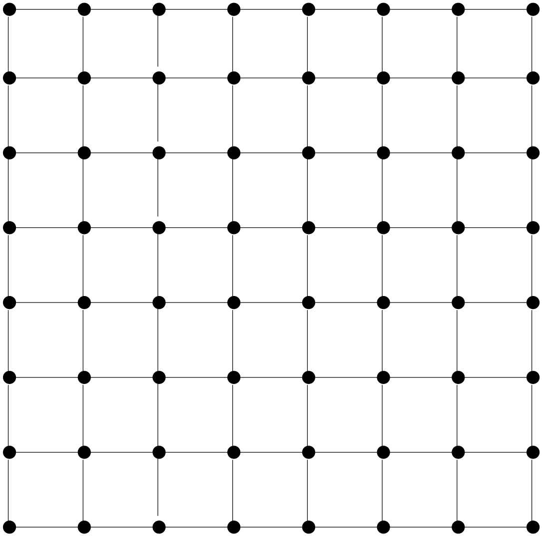


star

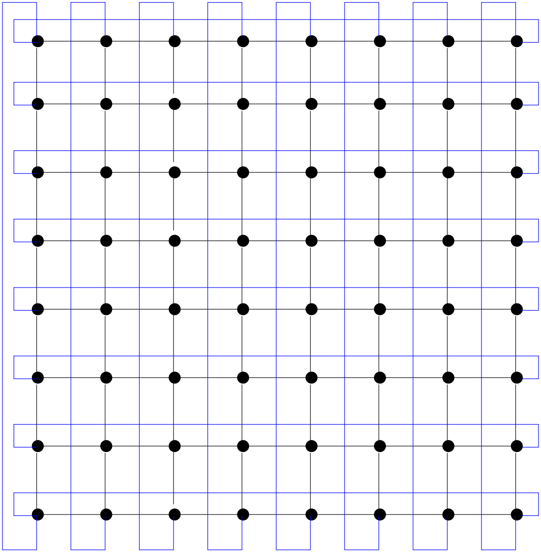
linear array



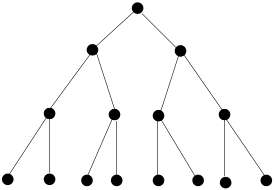
ring



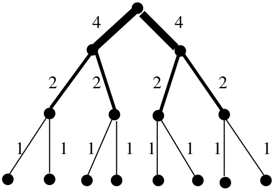
An 8 x 8 mesh



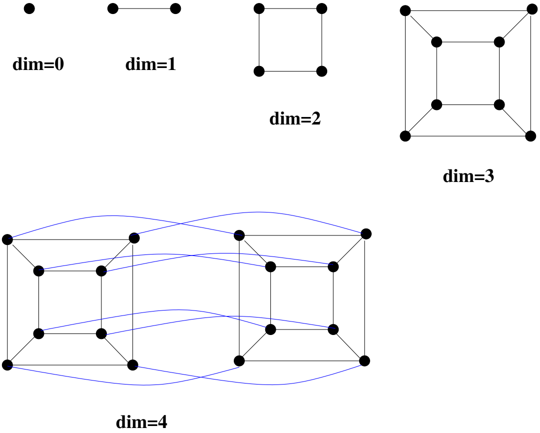
An 8 x 8 torus (a.k.a. wrap–around mesh)



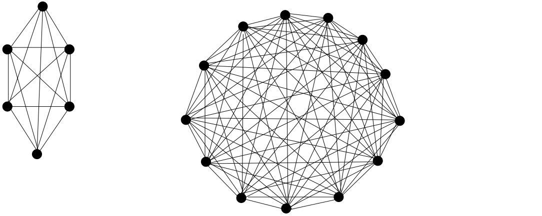
a 15−node complete binary tree



a 15−node fat tree



Hypercubes with dimensions 0 through 4



Crossbars (or completely−connected networks)

How to define a good network parallel computer?

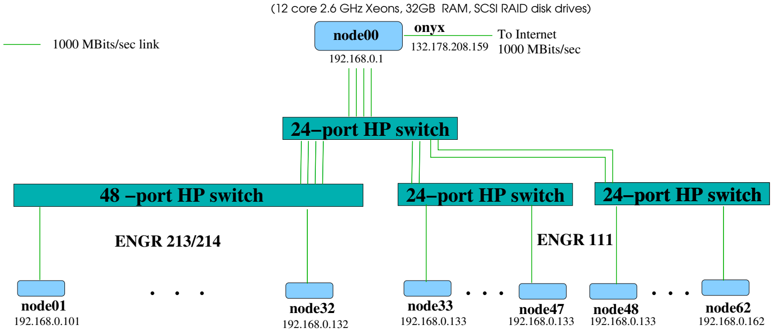
For a "good" parallel machine the diameter should be low, connectivity should be high, max. degree low, bisection width high, symmetry should be present and cost low!

Comparison of network models

Type	Diameter	Min. deg.	Max. deg.	Bisection Width	Cost	Sym.
bus	1	1	1	1	n-1 (?)	yes
star	2	1	n-1	$\lfloor n/2 \rfloor$	n-1	no
linear array	n-1	1	2	1	n-1	no
ring	$\lfloor n/2 \rfloor$	2	2	2	n	yes
mesh	$2\sqrt{n}-2$	2	4	$\sqrt{n}+(\sqrt{n} \bmod 2)$	$2n-2\sqrt{n}$	no
torus	$\sqrt{n}-(\sqrt{n} \bmod 2)$	4	4	$2\sqrt{n}+2(\sqrt{n} \bmod 2)$	2n	yes
binary tree	$2\lg(n+1)-2$	1	3	1	$n-1$	no
fat tree	$2\lg(n+1)-2$	1	$(n+1)/2$	$(n+1)/4$	$(\lg n-1)(n+1)/2$	no
hypercube	$\lg n$	$\lg n$	$\lg n$	n/2	$(n \lg n)/2$	yes
crossbar	1	n-1	n-1	$\lfloor n/2 \rfloor \lceil n/2 \rceil$	$n(n-1)/2$	yes

Notes:

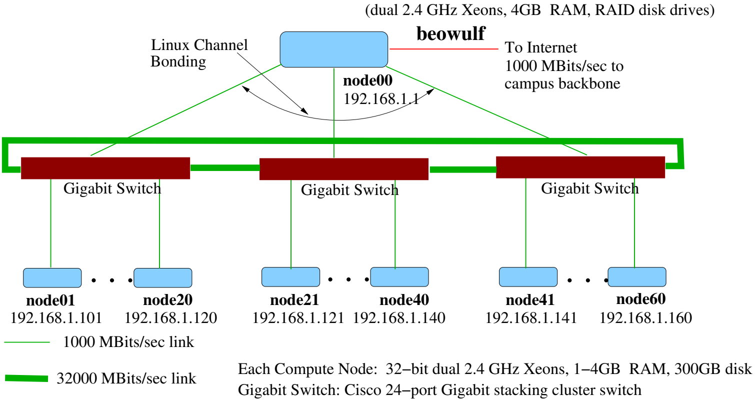
- Assume that there are a total of n processors in each network model.
- The mesh and the torus each have $n = m \times m$ nodes, that is, n is a square.



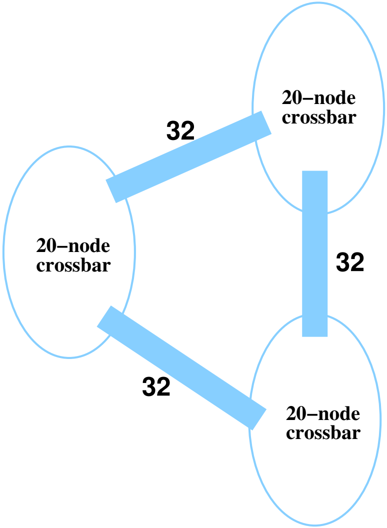
Per Node:
64–bit Intel Xeon quad–core 3.1–3.2 GHz
8GB RAM
250 GB disk
NVIDIA Qudaro 600: 96 cores, 1 GB memory

Linux Cluster Lab

Total:
260 cores
528 GB memory
~19TB raw disk space



Beowulf Cluster Architecture



Beowulf Cluster Network

Local Research Clusters

- **Genesis Cluster**.
  - 64 cores: 16 quad core Intel i7 CPUs in 16 nodes. Configured as 8 compute and 8 I/O nodes
  - 192GB memory: 12 GB memory per node
  - 100TB of raw disk space
  - 3 GigE channel bonded network on each node (4 GigE channel on head node)
  - Infiniband Mellanox Technologies MT25208: 10 Gig/s network

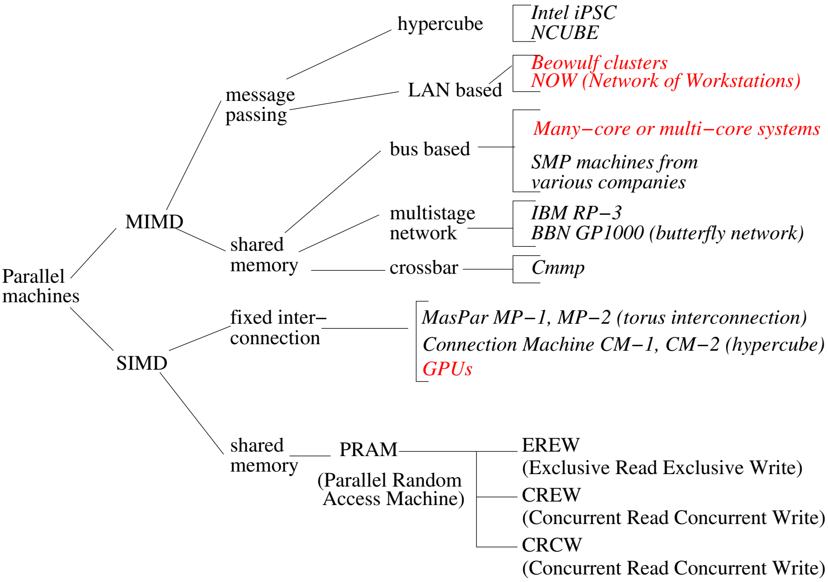
Local Research Clusters

- **Genesis Cluster**.
  - 64 cores: 16 quad core Intel i7 CPUs in 16 nodes. Configured as 8 compute and 8 I/O nodes
  - 192GB memory: 12 GB memory per node
  - 100TB of raw disk space
  - 3 GigE channel bonded network on each node (4 GigE channel on head node)
  - Infiniband Mellanox Technologies MT25208: 10 Gig/s network
- **Boise State R1 Cluster**
  - 272 cores: 1 Head node + 16 compute nodes with Dual AMD Opteron 6128 8 core 2.0 Ghz
  - 576GB memory: Head node: 64GB memory, Compute nodes: 32GB memory
  - Eight nodes have dual NVIDIA Tesla 2070 (or GTX 680) cards with 448 cores each
  - Around 50TB of raw disk space
  - InfiniBand: Mellanox MT26428 [ConnectX VPI PCIe 2.0 5GT/s – IB QDR / 10GigE]

Local Research Clusters

- **Genesis Cluster**.
  - 64 cores: 16 quad core Intel i7 CPUs in 16 nodes. Configured as 8 compute and 8 I/O nodes
  - 192GB memory: 12 GB memory per node
  - 100TB of raw disk space
  - 3 GigE channel bonded network on each node (4 GigE channel on head node)
  - Infiniband Mellanox Technologies MT25208: 10 Gig/s network
- **Boise State R1 Cluster**
  - 272 cores: 1 Head node + 16 compute nodes with Dual AMD Opteron 6128 8 core 2.0 Ghz
  - 576GB memory: Head node: 64GB memory, Compute nodes: 32GB memory
  - Eight nodes have dual NVIDIA Tesla 2070 (or GTX 680) cards with 448 cores each
  - Around 50TB of raw disk space
  - InfiniBand: Mellanox MT26428 [ConnectX VPI PCIe 2.0 5GT/s – IB QDR / 10GigE]
- **Kestrel Cluster**.
  - 1024 cores: 32 nodes with 2 Intel Xeon E5-2600 series processors – 16 cores
  - 2048GB memory: 32 GB memory per node
  - 64TB Panasas Parallel File Storage
  - Infiniband Mellanox Technologies ConnectX-3FDR

- Check the list of top 500 supercomputers on the website: http://www.top500.org/
- Check out various statistics on the top 500 supercomputers? For example, what operating system is used the most? What is the most number of cores per CPU? Which vendors are the dominant players?



Types of parallel machines (with historical machines)

Evaluating Parallel Programs

Parallelizing problems involves dividing the computation into tasks or processes that can be executed simultaneously.

- **speedup** $= S_p(n) = \frac{T^*(n)}{T_p(n)} = \frac{\text{best sequential time}}{\text{time with } p \text{ processors}}$

Evaluating Parallel Programs

Parallelizing problems involves dividing the computation into tasks or processes that can be executed simultaneously.

- **speedup** $= S_p(n) = \frac{T^*(n)}{T_p(n)} = \frac{\text{best sequential time}}{\text{time with } p \text{ processors}}$
  - linear speedup: a speedup of p with p processors.

Evaluating Parallel Programs

Parallelizing problems involves dividing the computation into tasks or processes that can be executed simultaneously.

- **speedup** $= S_p(n) = \frac{T^*(n)}{T_p(n)} = \frac{\text{best sequential time}}{\text{time with } p \text{ processors}}$
  - linear speedup: a speedup of p with p processors.
  - superlinear speedup: An anomaly that usually happens due to
    - using a sub-optimal sequential algorithm,
    - unique feature of the parallel system,
    - or only on some instances of a problem.

Evaluating Parallel Programs

Parallelizing problems involves dividing the computation into tasks or processes that can be executed simultaneously.

- **speedup** $= S_p(n) = \frac{T^*(n)}{T_p(n)} = \frac{\text{best sequential time}}{\text{time with } p \text{ processors}}$
  - linear speedup: a speedup of p with p processors.
  - superlinear speedup: An anomaly that usually happens due to
    - using a sub-optimal sequential algorithm,
    - unique feature of the parallel system,
    - or only on some instances of a problem.
      (e.g. searching in an unordered database.)

Evaluating Parallel Programs

- **cost** $= C_p(n) = T_p(n) \times p =$ parallel runtime $\times$ number of processors.

Evaluating Parallel Programs

- **cost** $= C_p(n) = T_p(n) \times p =$ parallel runtime $\times$ number of processors.
- **efficiency** $= E_p(n) = \frac{T^*(n)}{C_p(n)}, 0 < E_p(n) \leq 1$
  - A **cost-optimal** parallel algorithm has efficiency equal to $\Theta(1)$, which implies linear speedup. What causes efficiency to be less than 1?

Evaluating Parallel Programs

- **cost** $= C_p(n) = T_p(n) \times p =$ parallel runtime $\times$ number of processors.
- **efficiency** $= E_p(n) = \frac{T^*(n)}{C_p(n)}, 0 < E_p(n) \leq 1$
  - A **cost-optimal** parallel algorithm has efficiency equal to $\Theta(1)$, which implies linear speedup. What causes efficiency to be less than 1?
    - all processes may not be performing useful work (load balancing problem)

Evaluating Parallel Programs

- **cost** $= C_p(n) = T_p(n) \times p =$ parallel runtime $\times$ number of processors.
- **efficiency** $= E_p(n) = \frac{T^*(n)}{C_p(n)}, 0 < E_p(n) \leq 1$
  - A **cost-optimal** parallel algorithm has efficiency equal to $\Theta(1)$, which implies linear speedup. What causes efficiency to be less than 1?
    - all processes may not be performing useful work (load balancing problem)
    - extra computations are present in the parallel program that were not there in the best serial program

Evaluating Parallel Programs

- **cost** $= C_p(n) = T_p(n) \times p =$ parallel runtime $\times$ number of processors.
- **efficiency** $= E_p(n) = \frac{T^*(n)}{C_p(n)}, 0 < E_p(n) \leq 1$
  - A **cost-optimal** parallel algorithm has efficiency equal to $\Theta(1)$, which implies linear speedup. What causes efficiency to be less than 1?
    - all processes may not be performing useful work (load balancing problem)
    - extra computations are present in the parallel program that were not there in the best serial program
    - communication time for sending messages (communication overhead)

Granularity of parallel programs

- Granularity can be described as the size of the computation between communication or synchronization points.

Granularity of parallel programs

- Granularity can be described as the size of the computation between communication or synchronization points.
- The granularity can be measured by looking at the ratio of computation time to communication time in a parallel program. A coarse-grained program has high granularity, leading to lower communication overhead. A fine-grained program has a low ratio, which implies a lot of communication overhead.

Scalability

- **Scalability** is the effect of scaling up hardware on the performance of a parallel program

Scalability

- **Scalability** is the effect of scaling up hardware on the performance of a parallel program
- Scalability can be architectural or algorithmic. Here we are concerned with more with algorithmic scalability.

Scalability

- **Scalability** is the effect of scaling up hardware on the performance of a parallel program
- Scalability can be architectural or algorithmic. Here we are concerned with more with algorithmic scalability.
- The idea is that the efficiency of a parallel program should not deteriorate as the number of processors being used increases.

Scalability

- **Scalability** is the effect of scaling up hardware on the performance of a parallel program
- Scalability can be architectural or algorithmic. Here we are concerned with more with algorithmic scalability.
- The idea is that the efficiency of a parallel program should not deteriorate as the number of processors being used increases. Scalability can be measured for either:
  - a fixed total problem size, or

Scalability

- **Scalability** is the effect of scaling up hardware on the performance of a parallel program
- Scalability can be architectural or algorithmic. Here we are concerned with more with algorithmic scalability.
- The idea is that the efficiency of a parallel program should not deteriorate as the number of processors being used increases. Scalability can be measured for either:
  - a fixed total problem size, or
    - In this case efficiency always tends to zero as number of processors gets larger.

Scalability

- **Scalability** is the effect of scaling up hardware on the performance of a parallel program
- Scalability can be architectural or algorithmic. Here we are concerned with more with algorithmic scalability.
- The idea is that the efficiency of a parallel program should not deteriorate as the number of processors being used increases. Scalability can be measured for either:
  - a fixed total problem size, or
    - In this case efficiency always tends to zero as number of processors gets larger.
  - a fixed problem size per processor.

Scalability

- **Scalability** is the effect of scaling up hardware on the performance of a parallel program
- Scalability can be architectural or algorithmic. Here we are concerned with more with algorithmic scalability.
- The idea is that the efficiency of a parallel program should not deteriorate as the number of processors being used increases. Scalability can be measured for either:
  - a fixed total problem size, or
    - In this case efficiency always tends to zero as number of processors gets larger.
  - a fixed problem size per processor. :
    - This is more relevant, especially for the network model. This is saying that if we use twice as many processors to solve a problem of double the size, then it should take the same amount of time as the original problem.

Parallel Summation on a Shared memory machine

Assume that we have an input array $A[0..n-1]$ of n numbers in shared memory. Assume that we have p processors indexed from 0 to $p-1$.

The simple algorithm partitions the numbers among the p processors such that the ith processor gets roughly n/p numbers.

- ith process add its share of n/p numbers an stores the partial sum in shared memory.
- process 0 adds the p partial sums to get the total sum.

Parallel Summation on a Shared memory machine

Assume that we have an input array $A[0..n-1]$ of n numbers in shared memory. Assume that we have p processors indexed from 0 to $p-1$.
The simple algorithm partitions the numbers among the p processors such that the ith processor gets roughly n/p numbers.

- ith process add its share of n/p numbers an stores the partial sum in shared memory.
- process 0 adds the p partial sums to get the total sum.

$T_p(n) = \Theta(n/p + p)$, $T^*(n) = \Theta(n)$
$S_p(n) = \Theta\left(\frac{p}{1+p^2/n}\right)$
$E_p(n) = \Theta\left(\frac{1}{1+p^2/n}\right)$
$\lim_{n\to\infty} E_p(n) = \Theta(1)$

Parallel Summation on a Shared memory machine

Assume that we have an input array $A[0..n-1]$ of n numbers in shared memory. Assume that we have p processors indexed from 0 to $p-1$.
The simple algorithm partitions the numbers among the p processors such that the ith processor gets roughly n/p numbers.

- ith process add its share of n/p numbers an stores the partial sum in shared memory.
- process 0 adds the p partial sums to get the total sum.

$T_p(n) = \Theta(n/p + p)$, $T^*(n) = \Theta(n)$
$S_p(n) = \Theta\left(\frac{p}{1+p^2/n}\right)$
$E_p(n) = \Theta\left(\frac{1}{1+p^2/n}\right)$
$\lim_{n\to\infty} E_p(n) = \Theta(1)$
Good speedup is possible if $p << n$, that is, the problem is very large and the number of processors is relatively few.

Parallel Summation on a Distributed memory machine

- Assume that process 0 initially has the n numbers.
  1. Process 0 sends n/p numbers to each processor.
  2. Each processor adds up its share and sends partial sum back to process 0.
  3. Process 0 adds the p partial sums to get the total sum.

  No speedup :-(Step 1 itself takes as much time as a sequential algorithm!

Parallel Summation on a Distributed memory machine

- Assume that process 0 initially has the n numbers.
  1. Process 0 sends n/p numbers to each processor.
  2. Each processor adds up its share and sends partial sum back to process 0.
  3. Process 0 adds the p partial sums to get the total sum.

  No speedup :-(Step 1 itself takes as much time as a sequential algorithm!
- Assume that each processor has its share (that is n/p) numbers to begin with. Then we only Step 2 and 3 above. The speedup and efficiency then is the same as in the shared memory case.

Limitations of Parallel Computing?

Amdahl's Law. Assume that the problem size stays fixed as we scale up the number of processors. Let t_s be the fraction of the sequential time that cannot be parallelized and let t_p be the fraction that can be be run fully in parallel. Let $t_s + t_p = 1$ be a constant. Then, the speedup is:

Limitations of Parallel Computing?

Amdahl's Law. Assume that the problem size stays fixed as we scale up the number of processors. Let t_s be the fraction of the sequential time that cannot be parallelized and let t_p be the fraction that can be be run fully in parallel. Let $t_s + t_p = 1$ be a constant. Then, the speedup is:

$$S_p(n) = \frac{t_s + t_p}{t_s + \frac{t_p}{p}} = \frac{p}{t_s(p-1)+1}$$

$$E_p(n) = \frac{t_s + t_p}{p(t_s + \frac{t_p}{p})} = \frac{1}{t_s(p-1)+1}$$

Limitations of Parallel Computing?

Amdahl's Law. Assume that the problem size stays fixed as we scale up the number of processors. Let t_s be the fraction of the sequential time that cannot be parallelized and let t_p be the fraction that can be be run fully in parallel. Let $t_s + t_p = 1$ be a constant. Then, the speedup is:

$$S_p(n) = \frac{t_s + t_p}{t_s + \frac{t_p}{p}} = \frac{p}{t_s(p-1)+1}$$

$$E_p(n) = \frac{t_s + t_p}{p(t_s + \frac{t_p}{p})} = \frac{1}{t_s(p-1)+1}$$

$$\lim_{p \to \infty} S_p(n) = \frac{1}{t_s}$$

$$\lim_{p \to \infty} E_p(n) = 0$$

Limitations of Parallel Computing?

Gustafson's Law. Assume that the problem size grows in proportion to the scaling of the number of processors. The idea is to keep the running time constant and increase the problem size to use up the time available.

Limitations of Parallel Computing?

Gustafson's Law. Assume that the problem size grows in proportion to the scaling of the number of processors. The idea is to keep the running time constant and increase the problem size to use up the time available. Let t_s be the fraction of the sequential time that cannot be parallelized and let t_p be the fraction that can be be run fully in parallel. Let $t_s + t_p = 1$ be a constant.

Limitations of Parallel Computing?

Gustafson's Law. Assume that the problem size grows in proportion to the scaling of the number of processors. The idea is to keep the running time constant and increase the problem size to use up the time available. Let t_s be the fraction of the sequential time that cannot be parallelized and let t_p be the fraction that can be be run fully in parallel. Let $t_s + t_p = 1$ be a constant.

If we keep the run time constant, then the problem needs to be scaled enough such that the parallel run time is still $t_s + t_p$, which implies that the serial time would be $t_s + pt_p$. Thus the speedup would be:

Limitations of Parallel Computing?

Gustafson's Law. Assume that the problem size grows in proportion to the scaling of the number of processors. The idea is to keep the running time constant and increase the problem size to use up the time available. Let t_s be the fraction of the sequential time that cannot be parallelized and let t_p be the fraction that can be be run fully in parallel. Let $t_s + t_p = 1$ be a constant.

If we keep the run time constant, then the problem needs to be scaled enough such that the parallel run time is still $t_s + t_p$, which implies that the serial time would be $t_s + pt_p$. Thus the speedup would be:

$$\begin{aligned} S_p(n) &= \frac{t_s + pt_p}{t_s + t_p} = t_s + pt_p \\ &= t_s + (1 - t_s)p \\ &= p + (1 - p)t_s \end{aligned}$$

$$\lim_{p \to \infty} S_p(n) = \infty$$

Exercises

1. Read Wikipedia entries for “Multi-core", “Computer Cluster" and “Supercomputer"
2. What is the diameter, connectivity, cost and bisection width of a 3-dimensional torus with $n = m \times m \times m$ nodes?
3. Draw the layout of a 6-dimensional hypercube with 64 processors. The goal is to make the wiring pattern systematic and neat.
4. A parallel computer consists of 10000 processors, each capable of a peak execution rate of 10 GFLOPs. What is the performance of the system in GFLOPS when 20% of the code is sequential and 80% is parallelizable. (Assume that the problem is not scaled up on the parallel computer)