CA3404: Winter 2018
Distributed and Parallel Algorithms

Week - 1 : Introduction, architectures and languages for parallel and distributed processing

Course email-id: dispalgo2018@gmail.com
Dt: 04-01-2018, 11-01-2018, 12-01-2018, 12-02-18

Do you know ?

How cloud storage works ?

For e.g. Google drive - how it works ?

Cloud storage is
->a model of data storage in which the
->digital data is stored in logical pools,
->the physical storage spans multiple servers (and often locations), and
->the physical environment is typically owned and managed by a hosting company.
->These cloud storage providers are responsible for keeping the data available and accessible, and the physical environment protected and running.

People and organizations buy or lease storage capacity from the providers to store user, organization, or application data

A brief overview- how the Google drive works!

Think if you wish to develop cloud storage like Google Drive - how you'll approach to solve this problem?
You need to break the problem into multiple components.

First, you need a base distributed storage system. This is where you store files in a distributed manner. This helps with scalability and resiliency. You can do this using distributed file systems like ceph, glusterfs or NFS. You can even use object databases.

Second, you need a mechanism to integrate this file system to your front end. This depends on the file system in use. Some like Ceph and distributed databases already have APIs. Other require you build an application to handle this.

A brief overview- how the Google drive works contd..

Third, you need a mechanism to store Metadata. In some cases, you can store application specific Metadata in the file system of choice. So you need to provide a mechanism for it. A simple distributed database of sorts is sufficient here.

Lastly, you need to write a scalable front end. This provides the interface to the customer and wires the Metadata and access to the file system.

There are many way of handling de-duplication. One way it generating file signatures to track and eliminate duplicates. If a user tries to upload a file that is already present according to the signature then it's new record simply points to the existing file

And

Do you know about the processor you use in your typical desktop or laptop ?

Dual core, Quad Core, in mobile you would have heard about Octa-core processors.

So what are these multi-core processors ?

A **multi-core processor** is a
->single computing component with
->two or more independent processing units called cores, which read and execute program instructions.
->The instructions are ordinary CPU instructions(such as add, move data, and branch) but
“*the single processor can run multiple instructions on separate cores at the same time, increasing overall speed for programs amenable to parallel computing*.”

Transition to parallel from sequential processing

Classically, algorithm designers assume a computer with only

- ➔ one processing element;
- ➔ the algorithms they design are said to be sequential,
- ➔ since the algorithms' steps must be performed in a particular sequence, one after another.
- ➔ But today's computers often have multiple processors, each of which performs its own sequence of steps.
- ➔ Even basic desktop computers often have multicore processors that include a handful of processing elements.
- ➔ But also significant are vast clusters of computers, such as those used by the Department of Defense to simulate nuclear explosions and by Google to process search engine queries.

Let’s consider a case-study

Suppose we're working with such a real-world multiprocessor system.
Inevitably, we'll run into some problem that we want it to solve.
Letting *T* represent the amount of time that a single processor takes to solve this problem,
we would hope to develop an algorithm that takes *T / p* time on a p-processor system.
In reality, this will rarely be possible to achieve entirely: The processors will almost always need to spend some time coordinating their work, which will lead to a penalty.

Contd..

Sometimes such a near-optimal speedup is easy to achieve.

Suppose, for instance, that

we have an array of integers, and we want to display all the negative integers in the array.
How will you approach this problem if I tell you that you got multiple processors, now how you'll take help of parallelism to solve this problem ?

We can approach in this manner:

We can divide the array into equal-sized segments,
one segment for each processor,
and
each processor can display all the negative integers in its segment.

The sequential algorithm would take O(n) time.

In our multiprocessor algorithm each processor handles an array segment with at most $\lceil n / p \rceil$ elements,
so processing the whole array takes O(n / p) time.

Inherent Sequential Problem:

There are some problems where it is hard to imagine how to use multiple processors to speed up the processing time.

An example of this is determining the depth-first search order of a graph: It seems that any algorithm will be "forced" to process a child only after its parent, so if the graph's height is something like n / 2, it will necessarily take O(n) time. This seems like an inherently sequential problem.

Administrivia

->Course email-id : dispalgo2018@gmail.com, all communications regarding course will be take through this email-id only and the relevant FB group.

->20% Mid-sem, 60% End-sem.

->**20% Internals** : Assignments of total 10%.
"Multiple theoretical and programming assignments will be given."
Evaluation of each assignment will be done before giving next assignment!
"*Weightage of each assignment will be equal which will be decided depending upon the number of assignments at the end of the course."*

Administrivia..

-> 10% comprising of multiple Quizzes/class-tests (in-class)

->No marks on attendance. It will be maintained, and
“75% attendance will still be required to sit in exams - mid-sem, and end-sem.”

-> **Office hours**: Officially: *3:00 pm to 4:00pm, Thursday after your Cryptography class.*
You can come at any scheduled time by taking appointment in advance, about the topic you wish to discuss.

Few Imp Points to Remember

->Eat my head and mind both but clear your concepts(during/after the class whatever suitable as per situation),
->if you're not understanding anything in the class, and you hesitate to ask in public, clear your things during the office hours. These office hours will be for yours and yours only!
->There were some not-so-good experience in the last-sem during the learning phase, for which I'll be willing to make up this time but for-god-sake you need to come forward.
Warning:. "***I will not give a single mark if you'll not deserve it, whether next sem is placements or whatever - so START STUDYING properly from DAY-1*** "

Start Studying the subjects

From DAY - 1

“Remember”

Next Sem Placements - Learn and score A+(85+/100)

Plagiarism Policy of this course

There will be **grade deduction** if you caught into plagiarism.

Maximum Acceptable plagiarism percentage is **20%** in any assignment.

Moss will be used throughout the course for programming assignments.
For more info: https://theory.stanford.edu/~aiken/moss/

I might take the interview for theoretical assignments to check whether you understood the assignments and did on your own or not!

Syllabus we'll be covering before mid-sem

Week 1:
Introduction, architectures and languages for parallel and distributed processing

Week 2:
Abstract models of parallel computing, PRAM (Parallel Random Access Machine).

Week 3:
Distributed and parallel algorithms and their complexity.

Week 4:
Interaction between processes, communication, synchronization.

contd..

Week 5:
Synchronization b/w processes. Topologies, synchronous algorithms.

Week 6:
Asynchronous algorithms. Algorithms for parallel sorting. Algorithms for parallel searching.

Relevant Books for the course:

Textbooks:

1. Internet
2. http://www.thi.informatik.uni-frankfurt.de/lehre/pda/ws0910/pda_ws0910_skript.pdf
3. Parallel and distributed computing by S.K.Basu, PHI publisher
4. Parallel and distributed systems by Arun Kulkarni, Wiley India

Reference books:(as given in your curriculum)

1. Parallel Computation, Model and Methods by Akl,
2. An Introduction to Parallel Algorithms, by J'aJ'a, J
3. Synthesis of Parallel Algorithms by J. H. Rief,
4. Introduction to Distributed Algorithms by Gerard Tel,

Parallel & distributed models

Multiprocessor systems come in many different flavors, but there are two basic categories:

➔ parallel computers and
➔ distributed computers.

These two terms are used with some overlap, but usually:
a parallel system is one in which the processors are closely connected, while
a *distributed system* has processors that are more independent of each other.

Parallel and distributed systems

What is a parallel computer?

- ➔ A collection of processing elements that communicate and cooperate to solve large problems fast.

What is a distributed system?

- ➔ A collection of independent computers that appear to its users as a single coherent system.

- ➔ A parallel computer is implicitly a distributed system.

Contd..

- ➔ Parallel computing makes use of concurrency to reduce the runtime, increase the throughput.

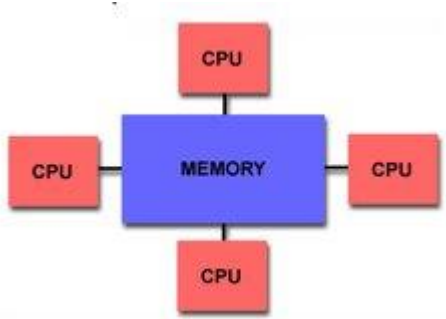
- ➔ Distributed computing may also exploit concurrency, although the performance and energy constraints generally differ from those of a tightly coupled parallel computer in one location.

Contd...

The use and organization of multiple processors to solve a problem

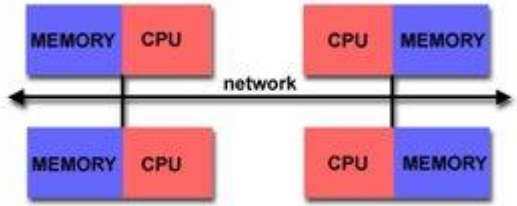
Parallel

- ➔ Processor share clock and memory
- ➔ Same OS
- ➔ Frequent communication



Distributed

- ➔ Memory not shared
- ➔ Different clocks
- ➔ Different OS
- ➔ Infrequent communication



Parallel Computing - an example

As an example of code for a shared-memory computer, below is a
Java fragment intended to “find the sum of all the elements in a long array.”

Variables whose name begin with
my_ are specific to each processor; this might be implemented by storing these variables in individual processors' registers.
The code fragment assumes that a variable array has already been set up with the numbers we want to add and that there is a variable procs that indicates how many processors our system has.
In addition, we assume each register has its own my_pid variable, which stores that processor's own processor ID, a unique number between 0 and procs − 1.

```
// 1. Determine where processor's segment is and add up numbers in segment.
count = array.length / procs;
my_start = my_pid * count;
my_total = array[my_start];
for(my_i = 1; my_i < count; my_i++) my_total += array[my_start + my_i];

// 2. Store subtotal into shared array, then add up the subtotals.
subtotals[my_pid] = my_total;                    // line A in remarks below
my_total = subtotals[0];                         // line B in remarks below
for(my_i = 1; my_i < procs; my_i++) {
    my_total += subtotals[my_i];                 // line C in remarks below
}

// 3. If array.length isn't a multiple of procs, then total will exclude some
// elements at the array's end. Add these last elements in now.
for(my_i = procs * count; my_i < array.length; my_i++) my_total += array[my_i];
```

Here, we first divide the array into segments of length count, and each processor adds up the elements within its segment, placing that into its variable my_total. We write this variable into shared memory in line A so that all processors can read it; then we go through this shared array of subtotals to find the total of the subtotals. The last step is to take care of any numbers that may have been excluded by trying to divide the array into p equally-sized segments.

To be contd...

Recap from the last lecture:

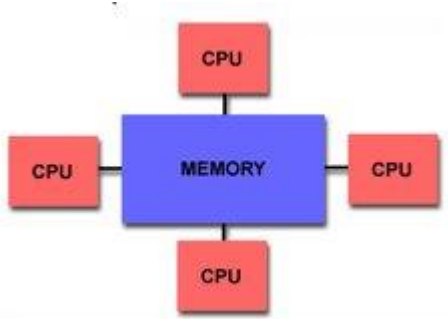
We covered basic definition of parallel and distributed systems and an example of problem solving using parallel computing.

Recap: (Parallel and Distributed Systems)

The use and organization of multiple processors to solve a problem

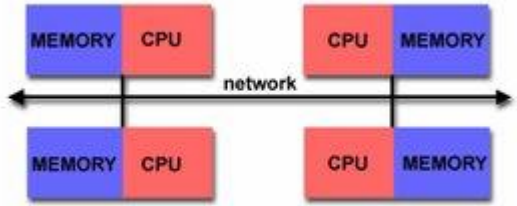
Parallel

- ➔ Processor share clock and memory
- ➔ Same OS
- ➔ Frequent communication



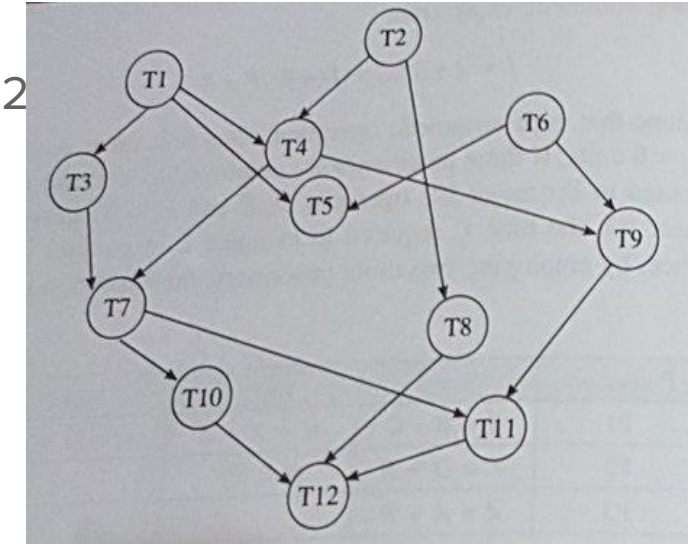
Distributed

- ➔ Memory not shared
- ➔ Different clocks
- ➔ Different OS
- ➔ Infrequent communication



Parallel Computing - 2nd Example

Here the program to solve a problem is represented by a task graph.
A task graph is a graph where the nodes represent the subtasks and the directed arrows represent the inter-task dependency.
Suppose a task T consists of 12 subtasks T1, T2,..,T12 as shown in the figure right:
Assumption here is suppose that all the subtasks require unit time.
Q1: If **3** processors are available
and the subtasks are scheduled on the processors then how many units of time will be required to finish the task?



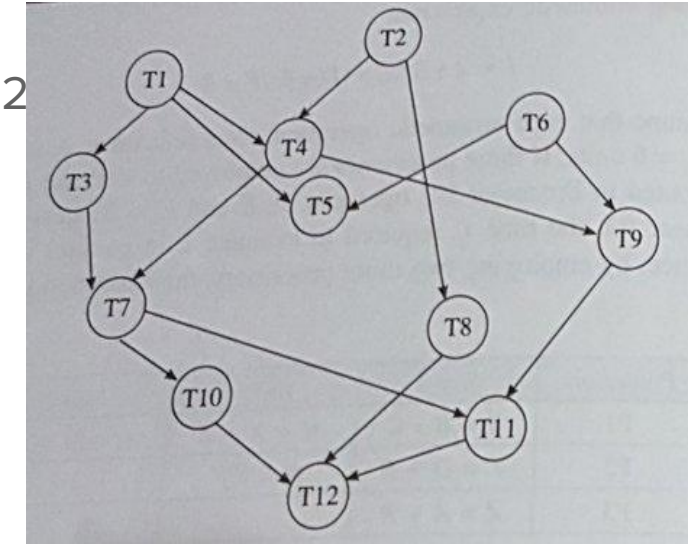
Solution:

If 3 processors are available and the subtasks are scheduled on the processors as shown below, 5 units of time will be required to finish the task.

Processors	Step 1	Step 2	Step 3	Step 4	Step 5
P1	T1	T3	T7	T10	T12
P2	T2	T4	T8	T11	
P3	T6	T5	T9		

Parallel Computing - 2nd Example

Here the program to solve a problem is represented by a task graph.
A task graph is a graph where the nodes represent the subtasks and the directed arrows represent the inter-task dependency.
Suppose a task T consists of 12 subtasks T1, T2,..,T12 as shown in the figure right:
Assumption here is suppose that all the subtasks require unit time.
Q2: If **2** processors are available
and the subtasks are scheduled on the processors then how many units of time will be required to finish the task?



Solution:

If 2 processors are available and the subtasks are scheduled on the processors as shown below, 7 units of time will be required to finish the task.

Processors	Step1	Step2	Step 3	Step 4	Step 5	Step 6	Step 7
P1	T1	T3	T4	T7	T11	T10	T12
P2	T2	T6	T5	T9	T8		

Observations from previous examples:

There is a tradeoff b/w
the no. of processors used & the time required in solving the problem.

Note:
1:) It is not always possible to reduce the overall time. Inter-processor communication is a sizeable fraction of the total time needed to solve a problem. It can be reduced by dividing the computation in such a way that the two processors needed to communicate to each other should be as close to each other as possible and by choosing the right topology of the processor interconnection network.

2:) Communication delays can be divided into 4 parts:

i) communication-processing time ii) queuing time

iii)transmission time iv) propagation time

3:) Factors that influence communication delays are:

i) algorithms used to control the communication network

ii) communication network topology, i.e., the number, nature and location of the communication links

iii)structure of the problem to be solved and the design of the algorithm to match the structure.

4:) If the subtasks of a task graph represent functions or procedures in high-level, Then parallel processing of this task graph represents coarse grain parallelism. In this, processors communicate among themselves less frequently.

5:) This division into independently executable tasks is constrained by inter-task data dependency.
In parallel processing, the main objective is to reduce the execution time of a problem

Some questions that must pop-up in your mind

i:) How should the processing elements be interconnected ?

ii) How should the problem be partitioned into subtasks and mapped on the processors?

iii) How should data routing and synchronization among subtasks executing on different processors or processing elements be made?

iv) How much time is wasted in inter-processor communication ?

All of these questions will be covered in this course.

Let’s Consider a Situation

Suppose, there is a situation where the same operation is to be repeated on a large set of data like the elements of large matrices and vectors.
It is not always possible to employ so many processors!

SO what’s the solution ? THINK !

Solution:

“Pipelining”

We can process the data in parallel by using pipelined vector computer. In this, a number of highly pipelined functional units are employed to expedite computation involving array data.

It will be covered in detailed in upcoming slides/lectures.

Advantages of Parallel Computing

1) Reduces time for task completion
2) Solving large and complex real-world applications
3) Provides concurrency
4) Better resource utilization
5) Fault tolerance

Disadvantages of Parallel Computing

1) Requires complex hardware
2) Expensive than serial computing.
3) No task can be perfectly serializable, so shared resources have to be used serially,
4) Task interdependencies must be considered before design.
5) Communication overhead exists as multiple processors are involved in computing

Applications of Parallel Computing

1) Weather forecasting and oceanographic processing.
2) Seismic data processing.
3) Remote sensing, image processing and soft computing.
4) Advanced graphics and virtual reality, particularly in the entertainment industry.
5) Financial processing.
6) Drug design and gene sequencing.
7) Big data, databases, data mining, oil exploration.
8) Web search engines, web-based business services.
9) Medical Imaging and diagnosis, Pharmaceutical design.
10) Microelectronics in chip design.

Classification of Parallel Computing

1) **Distributed computing:** It is a computing concept that refers to multiple computer systems working on a single problem. In this, a single problem is divided into many parts, and each part is executed by different computers. As long as computers are networked, they can communicate with each other to solve the problem. If it is done properly, the computers perform like a single entitity. The ultimate goal of distributed computing is to maximize performance by connecting users and IT resources in a cost-effective, transparent and reliable manner. This type of computing is highly scalable.

Classification of Parallel Computing

2. **Cluster computing**: A cluster is a group of loosely coupled homogeneous computers that work together closely, so that in some aspects they can be regarded as a single computer. Clusters are composed of multiple standalone machines connected by a network. Modern clusters are typically designed to handle more difficult problems that require nodes to share intermediate results with each other more often. This requires a high bandwidth and more importantly, a low -latency interconnection network.

Classification of Parallel Computing

3:) **Grid Computing**: It is a model for allowing companies to use a large number of computing resources on demand, no matter where they are located. It can also be considered as a service for sharing computer power and data storage capacity over the Internet. A true grid comprises multiple distinct distributed processing environments. Grid computing employs not only the resources but whole systems from various locations while crossing geographic and political boundaries. The SETI@home and Folding@home are the best-known examples of grid computing.

Issues in Parallel Computing

1. Design of parallel computers
2. Design of efficient parallel algorithms
3. Parallel programming models
4. Parallel computer language
5. Methods for evaluating parallel algorithms
6. Parallel programming tools.

Architecture of Parallel processing/computing

Following are the basic parallel architecture components:

1) **Processors**: No. of processors used in the system, with variation in cores, the cache memories of processors, e.t.c
2) **Memory** : Shared or distributed
3) **Communication:** The communication can be synchronous or asynchronous. The interconnection may be through buses with different topologies.
4) **Control**: The control may be centralized or distributed.

Classification of parallel architectures: (ASSIGNMENT 1 - Due Date: 24th Jan, Wed, 11pm class, Discuss it with the relevant diagrams)

1. Classification based on specific type of parallel architectures:

a) Pipeline computers
b) Array processors
c) Multiprocessor architecture
d) Systolic Architecture
e) Dataflow architecture.

2. Classification based on architectural schemes:

a) Flyn's Classification: SISD, SIMD, MISD, MIMD
b) Shore's Classification: Machine 1, Machine 2, Machine 3, Machine 4, Machine 5, Machine 6
c) Feng's Classifcation:WSBS, WPBS, WSBP, WPBP

3. Classification based on memory access:

a) Shared Memory Architecture: UMA, NUMA
b) Distributed Memory Architecture
c) Hybrid Distributed-Shared Memory

4. Classification based on interconnection among processing elements and memory modules.

5. Classification based on characteristic nature of processing element:

a) CISC Processors
b) RISC Processors
c) DSP and Vector processors

Fine-grained Parallelism

- ➔ In fine-grained parallelism, a program is broken down to a large number of small tasks. These tasks are assigned individually to many processors. The amount of work associated with a parallel task is low and the work is evenly distributed among the processors. Hence, fine-grained parallelism facilitates load balancing*.
- ➔ As each task processes less data, the number of processors required to perform the complete processing is high. This in turn, increases the communication and synchronization overhead.
- ➔ Fine-grained parallelism is best exploited in architectures which support fast communication. Shared memory architecture which has a low communication overhead is most suitable for fine-grained parallelism

Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any single resource.

Coarse-grain Parallelism

In coarse-grained parallelism, a program is split into large tasks. Due to this, a large amount of computation takes place in processors.
This might result in load imbalance, wherein certain tasks process bulk of the data while others might be idle.
Further, coarse-grained parallelism fails to exploit the parallelism in the program as most of the computation is performed sequentially on a processor.

Message-passing architecture takes a long time to communicate data among processes which makes it suitable for coarse-grained parallelism.

Topics we covered in last class

Classification of parallel computing

- Distributed computing
- Grid computing
- Cluster computing

Issues in parallel computing:

-Design of parallel computers, efficient parallel algorithms, parallel programming models, parallel computer language, methods for evaluating parallel algorithms, parallel programming tools.

Architecture: Processors, Memory, communication, control

Coarse Grain parallelism, Fine grain parallelism

Topics of today’s class

Some General Parallel Terminology:Supercomputing / High Performance Computing (HPC), Node, CPU / Processor / Core, Task, Pipelining, Shared Memory,

Performance Metrics: Speedup, Efficiency, Redundancy, Scalability, Utilization, Amdahl’s Law

Parallel programming models: Shared Memory Model (without threads), Shared Memory Model(with threads), Distributed Memory / Message Passing Model, Data Parallel Model, Hybrid Model, SPMD and MPMD

Some General Parallel Terminology

Supercomputing / High Performance Computing (HPC)

- Using the world's fastest and largest computers to solve large problems.

Node

-A standalone "computer in a box". Usually comprised of multiple CPUs/processors/cores, memory, network interfaces, etc. Nodes are networked together to comprise a supercomputer.

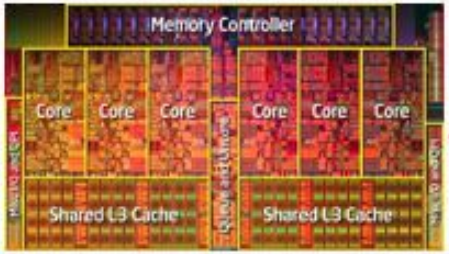
CPU / Processor / Core:

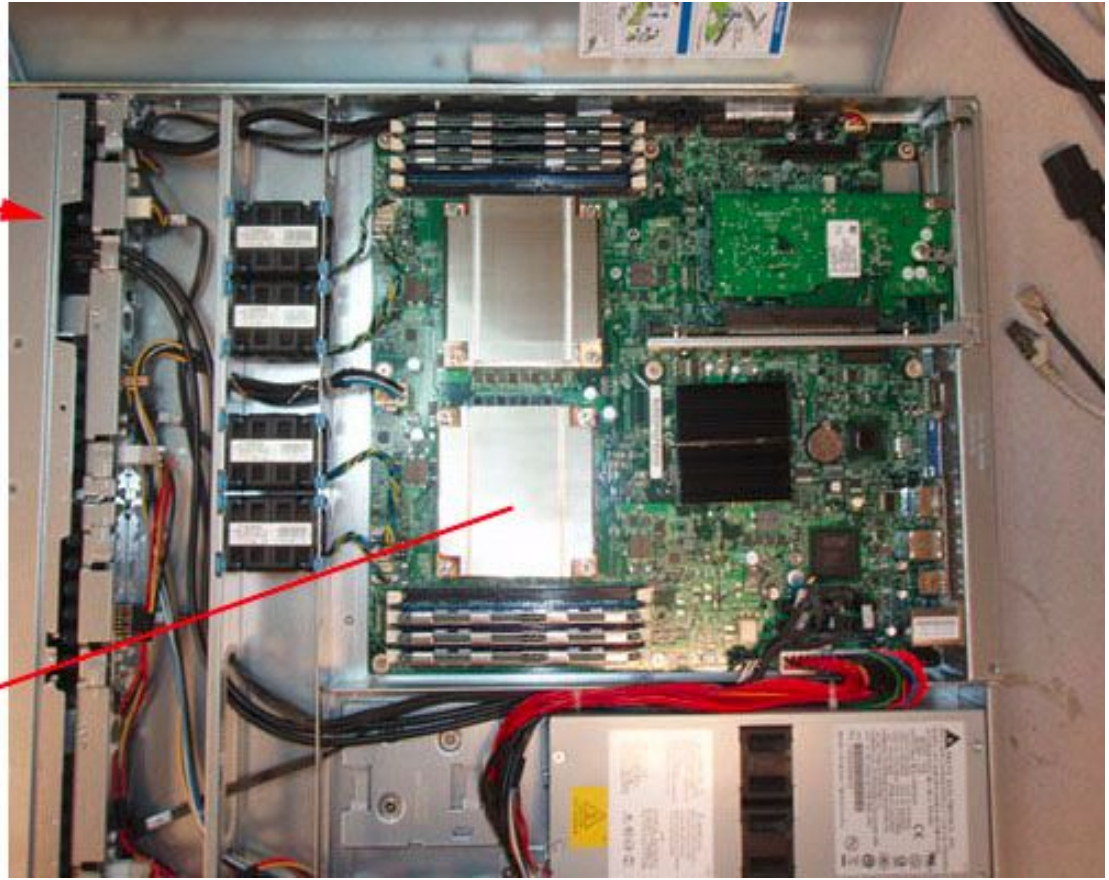
Then, multiple CPUs were incorporated into a node. Then, individual CPUs were subdivided into multiple "cores", each being a unique execution unit.

Supercomputer - each blue light is a node

Node - standalone Von Neumann computer

CPU / Processor / Socket - each has multiple cores / processors.





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. A parallel program consists of multiple tasks running on multiple processors.

Pipelining

Breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing.

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.

Performance Metrics

The performance metrics used for parallel system can be measured in terms of two aspects - performance metrics for processors and performance metrics for parallel computers.

Performance Metrics for Parallel Computers:

1) **Speedup**: Defined as the ratio b/w sequential execution time on a single processor (T(s))and parallel execution time on multiple processors (T(p)).

   Speedup, S(p) = T(s)/ T(p)

2) **Efficiency**: It defines the measure related to computational resources in terms of usage. Thus can define efficiency E(p) as the ratio b/w the performance(speedup) and number of processors used to achieve that performance. E(p) <=1

$$E(p) = S(p)/p = T(s)/ (p \times T(p))$$

3) **Redundancy**: The redundancy can be defined as the ratio b/w the total number of operations performed by p processors O(p) and the total number of operations performed by single processors O(s). So

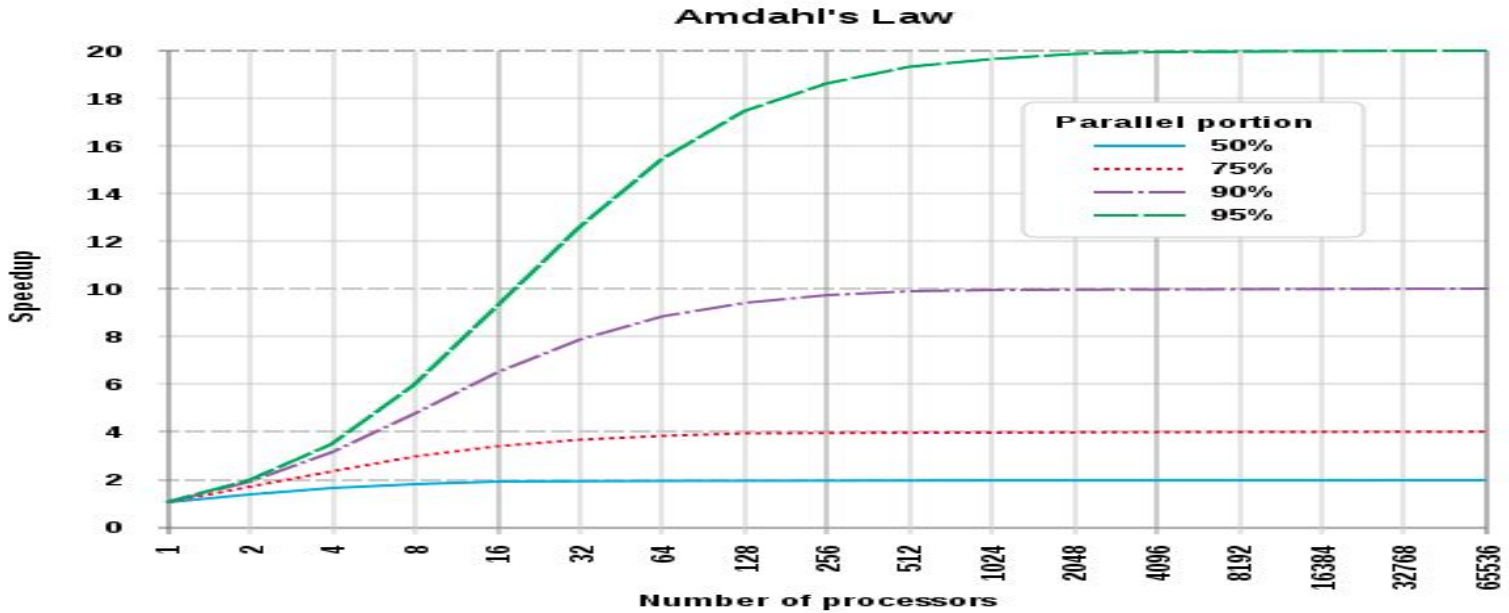
$$R(p) = O(p)/O(s)$$

4.) **Utilization**: The utilization measures the execution capacity of the resources. Thus, we can define utilization U(p) as the ratio b/w resources utilized during the execution and the actual capacity for the execution. Hence

$$U(p) = R(p) \times E(p)$$

5) **Scalability**: The scalability is the performance metric used to measure the capacity of utilization by increasing the number of processors. By observation we can say that if the number of processors increases, then the efficiency and speedup will automatically drop down because of assigned overheads.

6) **Amdahl’s Law**: is a formula which gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved.
Amdahl's law is often used in parallel computing to predict the theoretical speedup when using multiple processors.
For example, if a program needs 20 hours using a single processor core,
and a particular part of the program which takes one hour to execute cannot be parallelized,
while the remaining 19 hours ($p = 0.95$) of execution time can be parallelized,
then regardless of how many processors are devoted to a parallelized execution of this program,
the minimum execution time cannot be less than that critical one hour.
Hence, the theoretical speedup is limited to at most 20 times ($1/(1 - p) = 20$).



Evolution according to Amdahl's law of the theoretical speedup in latency of the execution of a program in function of the number of processors executing it, for different values of p. The speedup is limited by the serial part of the program. For example, if 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 20 times.

For this reason,
---*parallel computing with many processors is useful only for highly parallelizable programs*-----

Amdahl's law can be formulated in the following way:

$$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

where

- S_{latency} is the theoretical speedup of the execution of the whole task;
- s is the speedup of the part of the task that benefits from improved system resources;
- p is the proportion of execution time that the part benefiting from improved resources originally occupied.

Note: Amdahl's law applies only to the cases where the problem size is fixed. In practice, as more computing resources become available, they tend to get used on larger problems (larger datasets), and the time spent in the parallelizable part often grows much faster than the inherently serial work. In this case, **Gustafson's law** gives a less pessimistic and more realistic assessment of the parallel performance.

Derivation of Amdahl’s Law

A task executed by a system whose resources are improved compared to an initial similar system can be split up into two parts:

- a part that does not benefit from the improvement of the resources of the system
- a part that benefits from the improvement of the resources of the system.

An example is a computer program that processes files from disk.
A part of that program may scan the directory of the disk and create a list of files internally in memory. After that, another part of the program passes each file to a separate thread for processing. The part that scans the directory and creates the file list cannot be sped up on a parallel computer, but the part that processes the files can.

The execution time of the whole task before the improvement of the resources of the system is denoted as T .
It includes the execution time of the part that would not benefit from the improvement of the resources and the execution time of the one that would benefit from it.
The fraction of the execution time of the task that would benefit from the improvement of the resources is denoted by p. The one concerning the part that would not benefit from it is therefore 1 - p. Then:

$$T = (1 - p)\,T + p.T$$

It is the execution of the part that benefits from the improvement of the resources that is accelerated by the factor “**s**” after the improvement of the resources.
Consequently, the execution time of the part that does not benefit from it remains the same, while the part that benefits from it becomes:
p/s (T)

The theoretical execution time T (s) of the whole task after the improvement of the resources is then:

$$T(s) = (1 - p)T + \frac{p}{s}T.$$

Amdahl's law gives the theoretical speedup in latency of the execution of the whole task at fixed workload W, which yields

$$S_{\text{latency}}(s) = \frac{TW}{T(s)W} = \frac{T}{T(s)} = \frac{1}{1 - p + \frac{p}{s}}.$$

Examples: Parallel Programs

If 30% of the execution time may be the subject of a speedup, *p* will be 0.3; if the improvement makes the affected part twice as fast, *s* will be 2. Amdahl's law states that the overall speedup of applying the improvement will be:

$$S_{\text{latency}} = \frac{1}{1 - p + \frac{p}{s}} = \frac{1}{1 - 0.3 + \frac{0.3}{2}} = 1.18.$$

Assume that we are given a serial task which is split into four consecutive parts, whose percentages of execution time are $p1 = 0.11$, $p2 = 0.18$, $p3 = 0.23$, and $p4 = 0.48$ respectively. Then we are told that the 1st part is not sped up, so $s1 = 1$, while the 2nd part is sped up 5 times, so $s2 = 5$, the 3rd part is sped up 20 times, so $s3 = 20$, and the 4th part is sped up 1.6 times, so $s4 = 1.6$. By using Amdahl's law, the overall speedup is

$$S_{\text{latency}} = \frac{1}{\frac{p1}{s1} + \frac{p2}{s2} + \frac{p3}{s3} + \frac{p4}{s4}} = \frac{1}{\frac{0.11}{1} + \frac{0.18}{5} + \frac{0.23}{20} + \frac{0.48}{1.6}} = 2.19.$$

Notice how the 20 times and 5 times speedup on the 2nd and 3rd parts respectively don't have much effect on the overall speedup when the 4th part (48% of the execution time) is accelerated by only 1.6 times.

Examples: Serial Programs

For example, with a serial program in two parts *A* and *B* for which T_A = 3 s and T_B = 1 s,

- if part *B* is made to run 5 times faster, that is $s = 5$ and $p = T_B/(T_A + T_B) = 0.25$, then

$$S_{\text{latency}} = \frac{1}{1 - 0.25 + \frac{0.25}{5}} = 1.25;$$

if part *A* is made to run 2 times faster, that is $s = 2$ and $p = T_A/(T_A + T_B) = 0.75$, then

$$S_{\text{latency}} = \frac{1}{1 - 0.75 + \frac{0.75}{2}} = 1.60.$$

Therefore, making part *A* to run 2 times faster is better than making part *B* to run 5 times faster.

Solve it.

We are considering an enhancement to the processor of a server. The new CPU is 10X faster. IO bound server, so 60% time waiting for IO. What is the speed of the system.

Solution

- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$