BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems

Session 2: Parallel and Distributed Systems

Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

Context

Big Data Systems use basic principles of Parallel and Distributed Systems.

We need to understand these principles

Topics for today

- Road-blocks of processor scaling
- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing

Topics for today

- Road-blocks of processor scaling
- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing

Roadblocks of Processor / Vertical Scaling

- The Frequency Wall
  - ✓Not much headroom
- The Power Wall
  - ✓Dynamic and static power dissipation
- The Memory Wall
  - ✓Gap between compute bandwidth and memory bandwidth

Frequency wall

- More CPU Power -> More CPU hungry applications
- More Disk space -> New requirements to fill it up
- Applications enjoyed free performance benefits from latest processors
- 500 MHz -> 1 GHz -> 2 GHz -> 3.4 GHz -> 4 GHz ?

(No need to rewrite the S/W or even new release. Sometimes rebuilding is required)

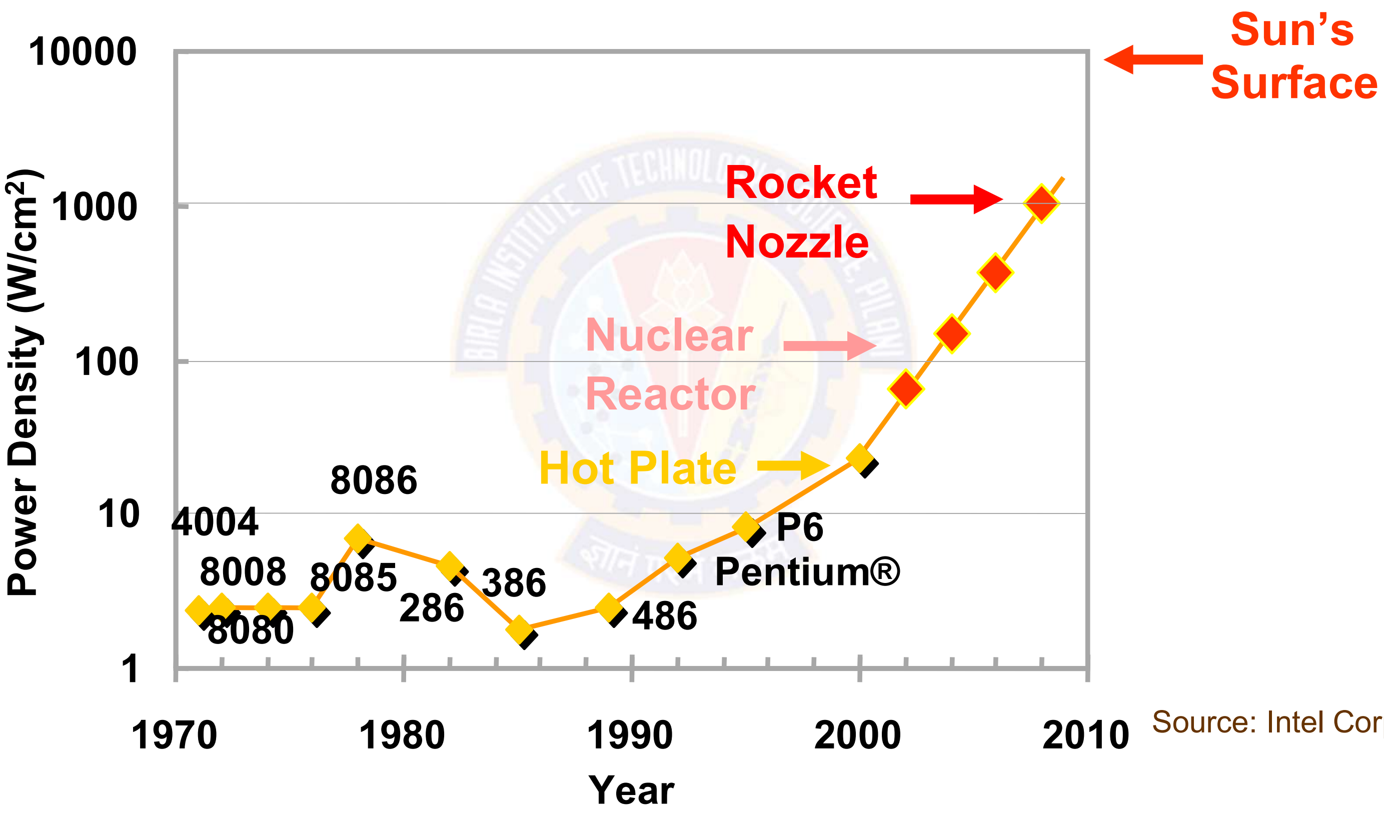
Why we do not have a 10GHz processor now?

Chip designers are under pressure to deliver faster CPUs that will not risk changing the structure of your program, and possibly break it, in order to make it run faster

Revolutions in computing:

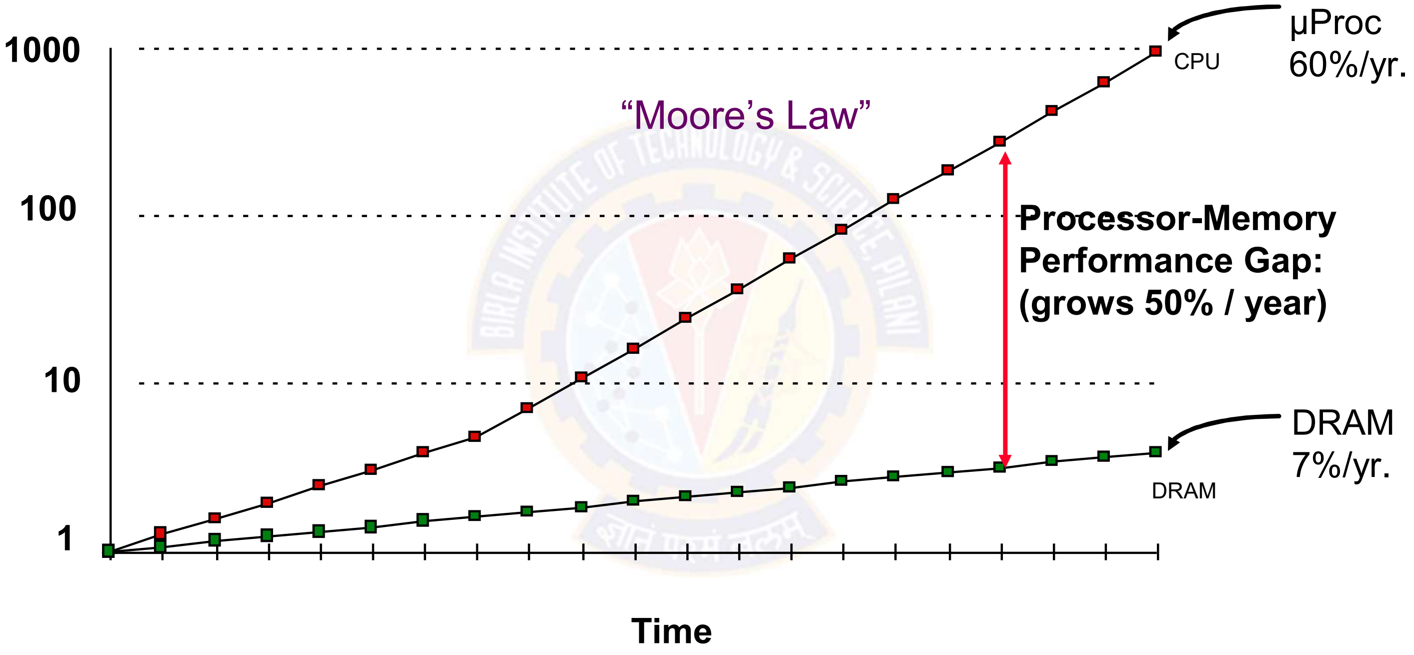
- 1990 - First Revolution in SW development – Object Oriented Programming
- Applications will increasingly need to be concurrently running if they want to fully exploit continuing exponential multi-core CPU throughput gains
- Next Revolution in SW development – Parallel (Concurrent) Programming

Thermal wall (CPU Power consumption)



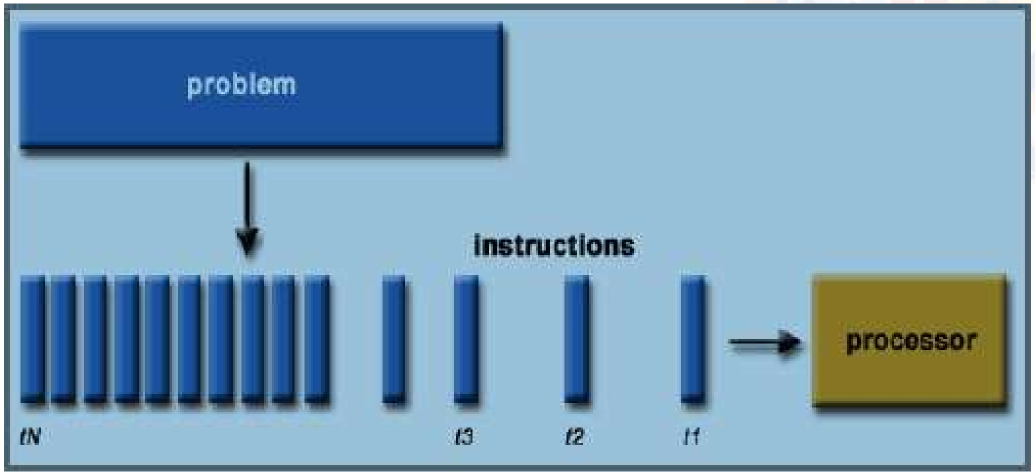
Source: Intel Corp.

Memory Wall


1000
100
10
1
Time
"Moore's Law"
CPU
µProc
60%/yr.
Processor-Memory
Performance Gap:
(grows 50% / year)
DRAM
DRAM
7%/yr.

Serial Computing

- Software written for serial computation:
  - ✓ A problem is broken into a discrete series of instructions
  - ✓ Instructions are executed sequentially one after another
  - ✓ Executed on a single processor
  - ✓ Only one instruction may execute at any moment in time
  - ✓ Single data stores - memory and disk

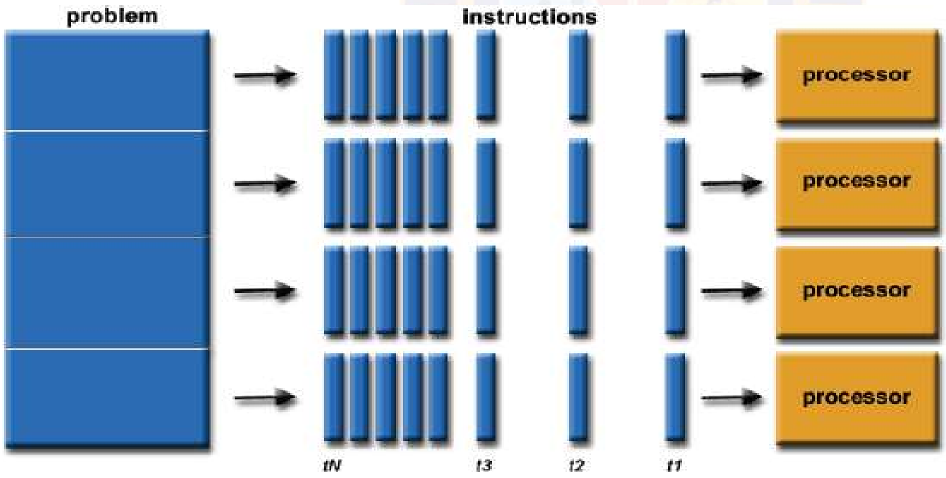


Extra info:

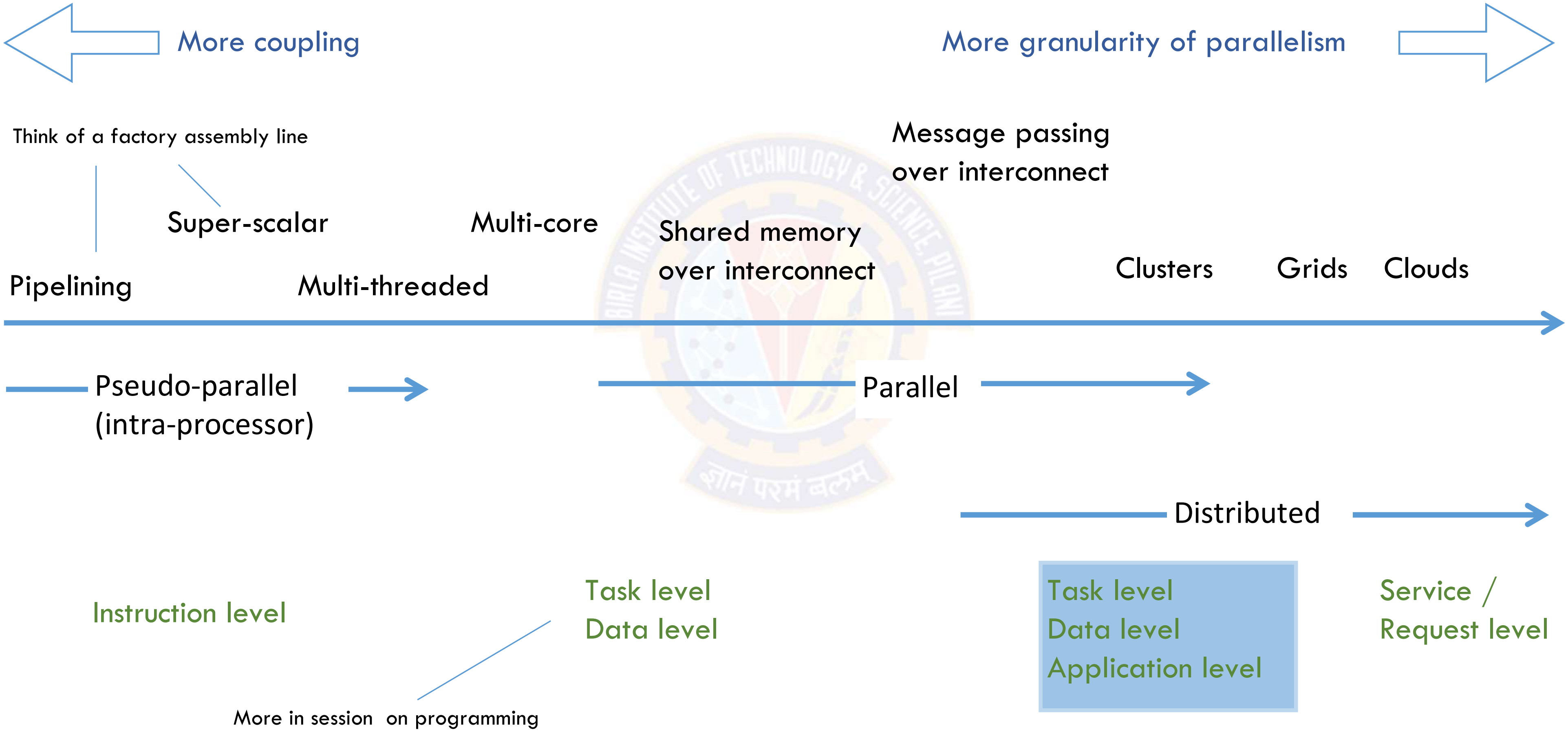
- Von Neumann architecture : common memory store and pathways between instructions and data - causes Von Neumann bottleneck
- Harvard architecture separates them to reduce bottleneck.
- Modern architectures use separate caches for instruction and data.

Parallel Computing

- Simultaneous use of multiple compute resources to solve a computational problem
  - ✓ A problem is broken into discrete parts **that can be solved concurrently**
  - ✓ Each part is further broken down to a series of instructions
  - ✓ Instructions from each part execute simultaneously on different processors
  - ✓ Different processors can work with independent memory and storage
  - ✓ An overall control/coordination mechanism is employed



Spectrum of Parallelism



More coupling

More granularity of parallelism

Think of a factory assembly line

Pipelining

Super-scalar

Multi-threaded

Multi-core

Shared memory
over interconnect

Message passing
over interconnect

Clusters

Grids

Clouds

Pseudo-parallel
(intra-processor)

Parallel

Distributed

Instruction level

Task level
Data level

More in session on programming

Task level
Data level
Application level

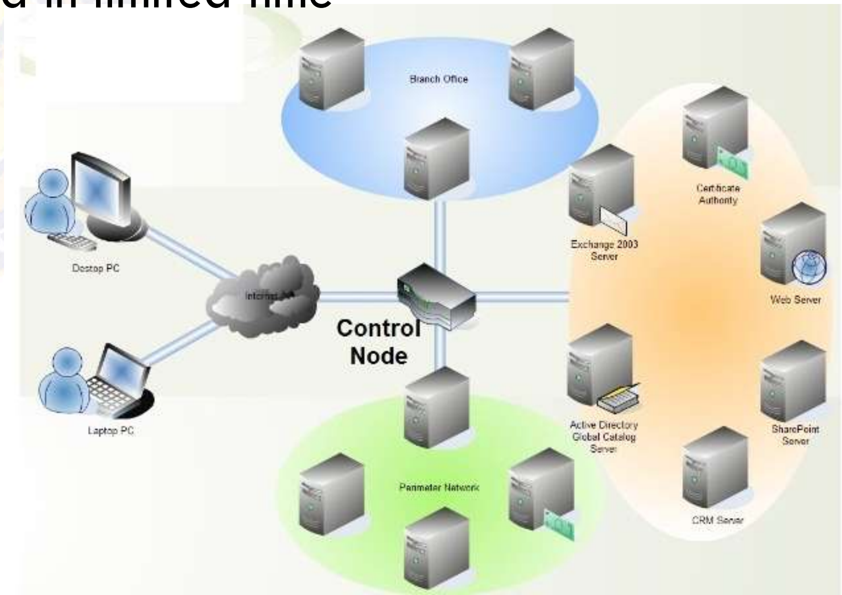
Service /
Request level

Comparing Parallel and Distributed Systems

Parallel System	Distributed System
Computer system with several processing units attached to it	Independent, autonomous systems connected in a network accomplishing specific tasks
A common shared memory can be directly accessed by every processing unit in a network	Coordination is possible between connected computers with own memory and CPU
Tight coupling of processing resources that are used for solving single, complex problem	Loose coupling of computers connected in network, providing access to data and remotely located resources
Programs may demand fine grain parallelism	Programs have coarse grain parallelism

Distributed Computing

- In distributed computing,
  - ✓Multiple computing resources are connected in a network and computing tasks are distributed across these resources
  - ✓Results in increase in speed and efficiency of system
  - ✓Faster and more efficient than traditional methods of computing
  - ✓More suitable to process huge amounts of data in limited time



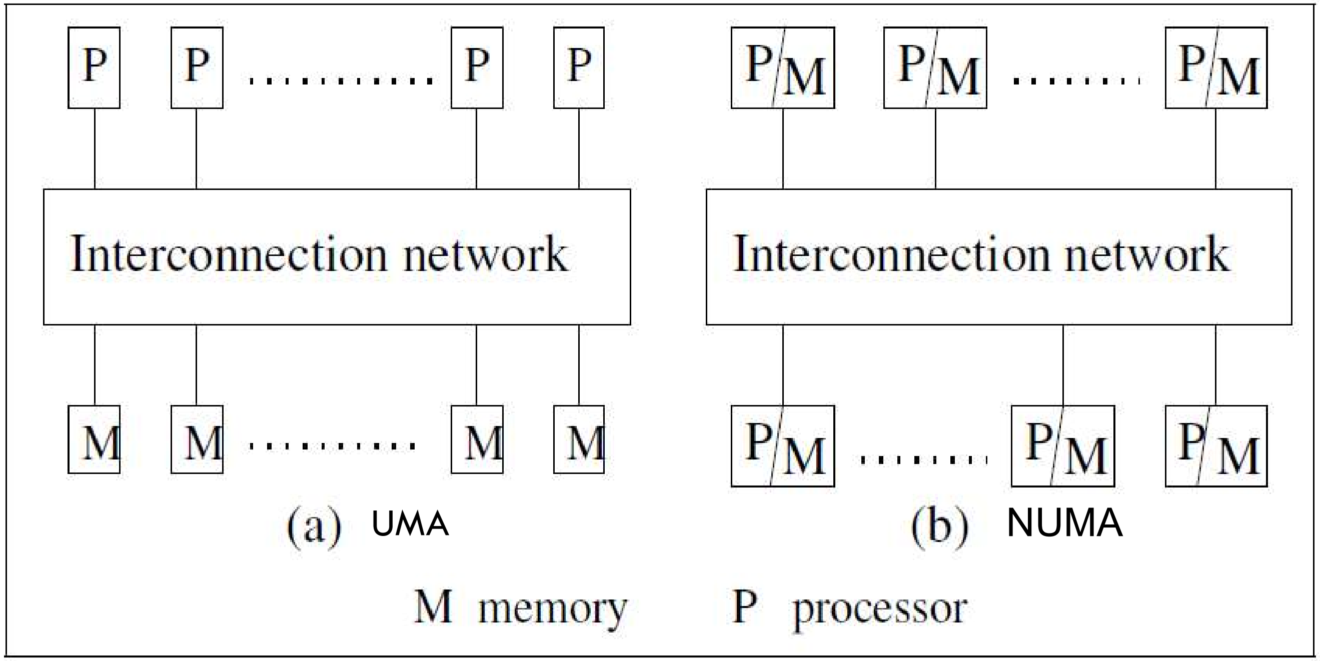
Multi-processor Vs Multi-computer systems

UMA

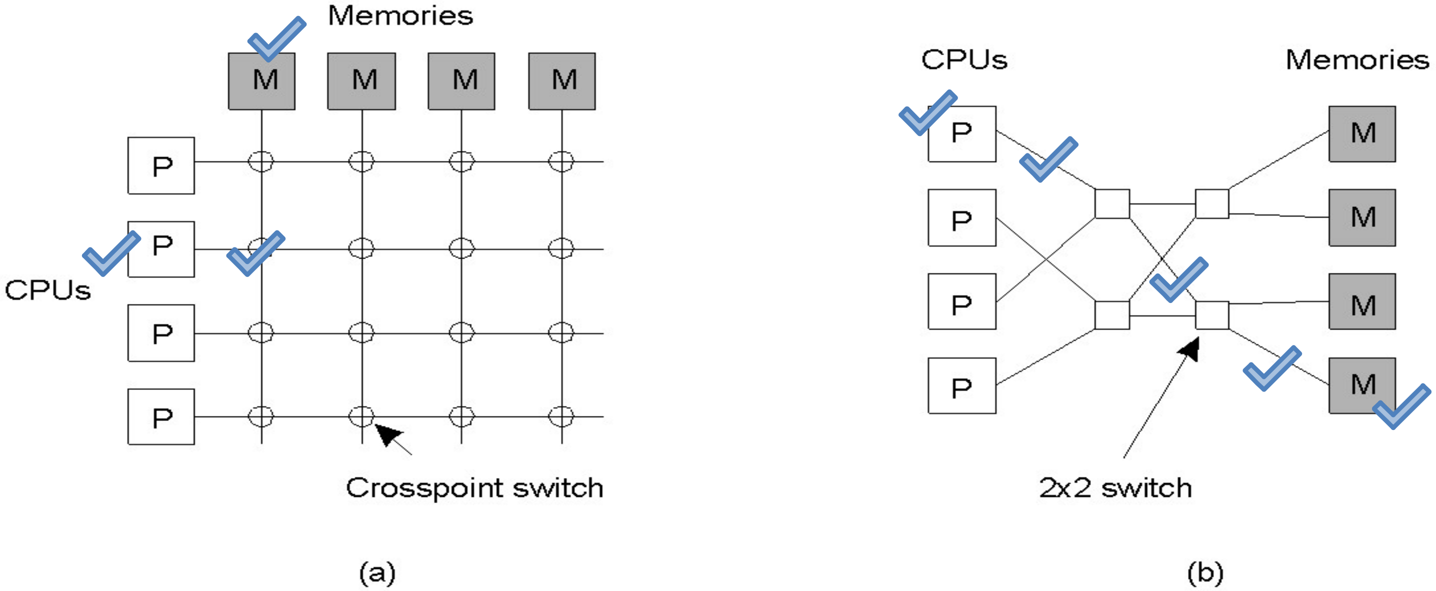
- Uniform Memory Access Multiprocessor
- Shared memory address space
- No common clock
- Fast interconnect

NUMA

- Non Uniform Memory Access Multicomputer
- May have shared address spaces
- Typically message passing
- No common clock



Interconnection Networks



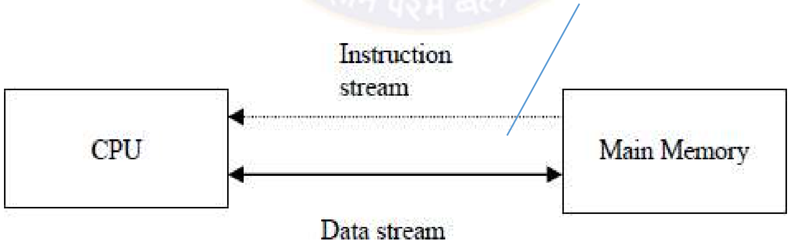
a) A crossbar switch - faster
b) An omega switching network - cheaper

Classification based on Instruction and Data parallelism

Instruction Stream and Data Stream

- The term ‘stream’ refers to a sequence or flow of either instructions or data operated on by the CPU.
- In the complete cycle of instruction execution, a flow of instructions from main memory to the CPU is established. This flow of instructions is called **instruction stream.**
- Similarly, there is a flow of operands between processor and memory bi-directionally. This flow of operands is called **data stream.**

Reduce **Von Neumann bottleneck** with separate caches



Flynn’s Taxonomy

Data Streams		Instruction Streams: Single	Instruction Streams: Multiple
	Single	SISD Uniprocessors Pipelining	MISD Uncommon Fault tolerance
	Multiple	SIMD Scientific computing Matrix manipulations	MIMD Multi-computers Distributed Systems

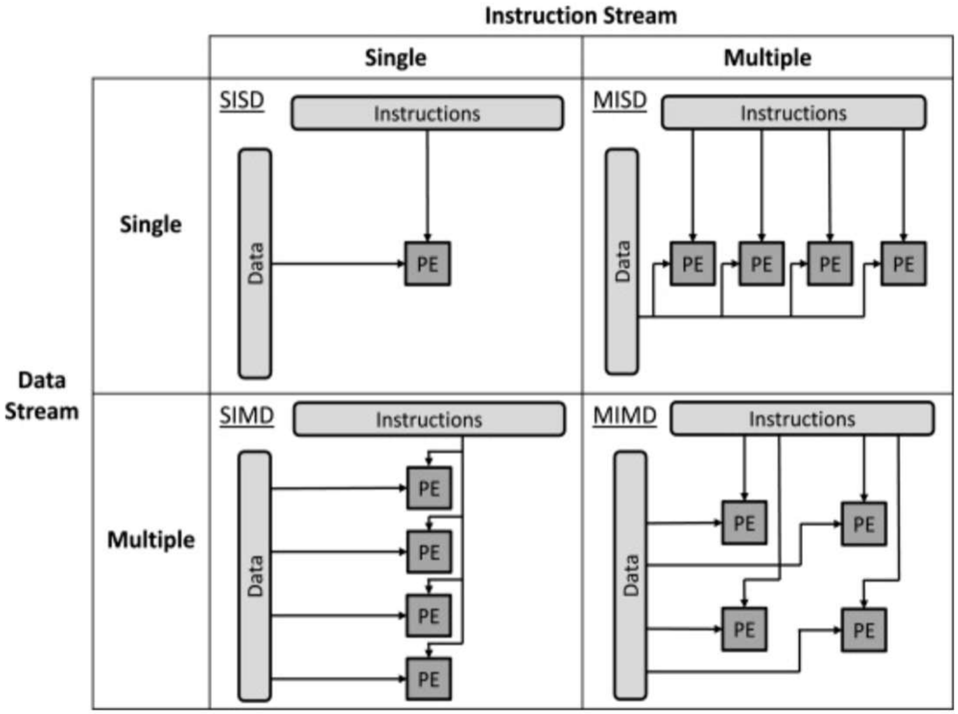


Image from sciencedirect.com

Understanding Flynn’s Taxonomy in Computer Architecture | Baeldung on Computer Science

Some basic concepts (esp. for programming in Big Data Systems)

- Coupling
  - Tight - SIMD, MISD shared memory systems
  - Loose - NOW, distributed systems, no shared memory
- Speedup
  - how much faster can a program run when given N processors as opposed to 1 processor — T(1) / T(N)
  - We will study Amdahl's Law, Gustafson's Law, Universal scalability law
- Parallelism of a program
  - Compare time spent in computations to time spent for communication via shared memory or message passing
- Granularity
  - Average number of compute instructions before communication is needed across processors
- Note:
  - If coarse granularity, use distributed systems else use tightly coupled multi-processors/computers
  - **Potentially high parallelism doesn't lead to high speedup if granularity is too small leading to high overheads**

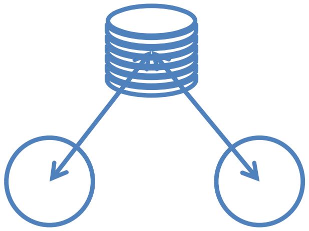
Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing

Motivation for parallel / distributed systems (1)

- Inherently distributed applications
  - e.g. financial tx involving 2 or more parties
- Better scale in creating multiple smaller parallel tasks instead of a complex task
  - e.g. evaluate an aggregate over 6 months data
- Processors getting cheaper and networks faster
  - e.g. Processor speed 2x / 1.5 years, network traffic 2x/year, processors limited by energy consumption
- Better scale using replication or partitioning of storage
  - e.g. replicated media servers for faster access or shards in search engines
- Access to shared remote resources
  - e.g. remote central DB
- Increased performance/cost ratio compared to special parallel systems
  - e.g. search engine runs on a Network-of-Workstations

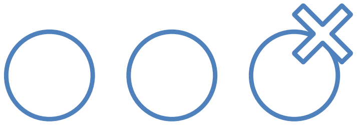
replicated / partitioned storage



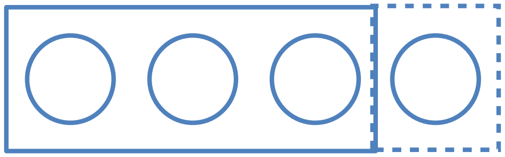
remote shared resource

Motivation for parallel / distributed systems (2)

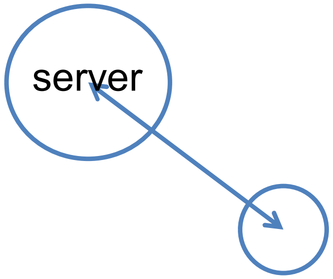
- Better reliability because less chance of multiple failures
  - Be careful about Integrity : Consistent state of a resource across concurrent access
- Incremental scalability
  - Add more nodes in a cluster to scale up
  - e.g. Clusters in Cloud services, autoscaling in AWS
- Offload computing closer to user for scalability and better resource usage
  - Edge computing



cluster nodes



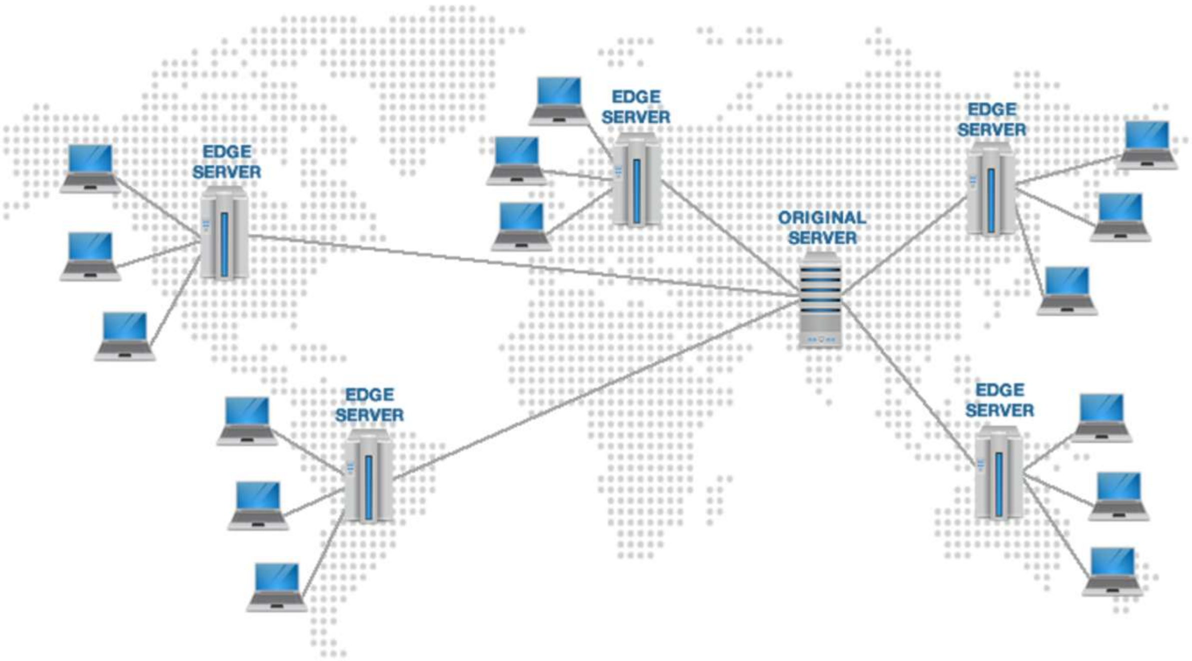
resize cluster



offload model inference to edge

Machine Learning at the edge:
https://www.datasciencecentral.com/machine-learning-at-the-edge/

Distributed network of content caching servers



This would be a P2P network if you were using bit torrent for free

Techniques for High Volume Data Processing

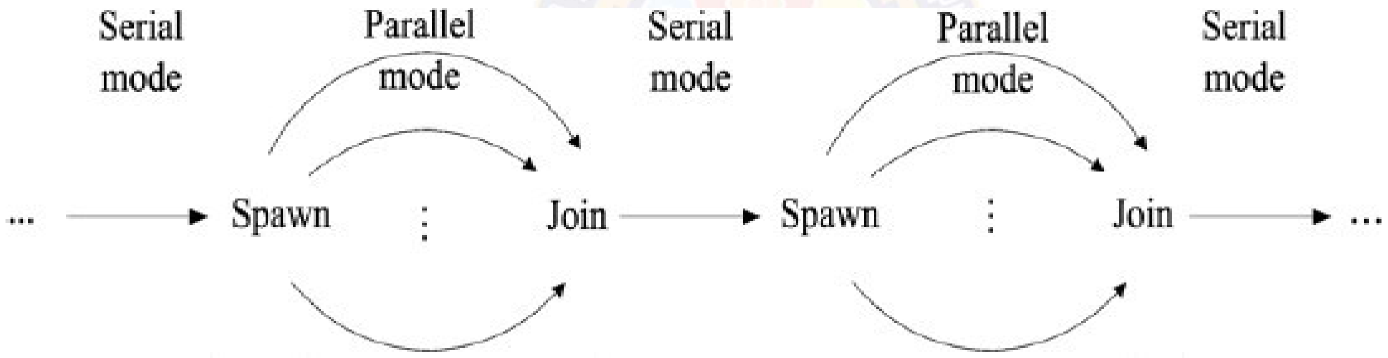
Method	Description	Usage
Cluster computing	A collection of computers, homogenous or heterogenous, using commodity components running open source or proprietary software, communicating via message passing	Commonly used in Big Data Systems, such as Hadoop
Massively Parallel Processing (MPP)	Typically, proprietary Distributed Shared Memory machines with integrated storage (Terra Data ?)	May be used in traditional Data Warehouses, Data processing appliances, e.g. EMC Greenplum (postgreSQL on an MPP)
High-Performance Computing (HPC)	Known to offer high performance and scalability by using in-memory computing	Used to develop specialty and custom scientific applications for research where results is more valuable than cost

Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing

Limits of Parallelism

- A parallel program has some sequential / serial code and significant parallelized code



Amdahl's Law – (1) - 1967

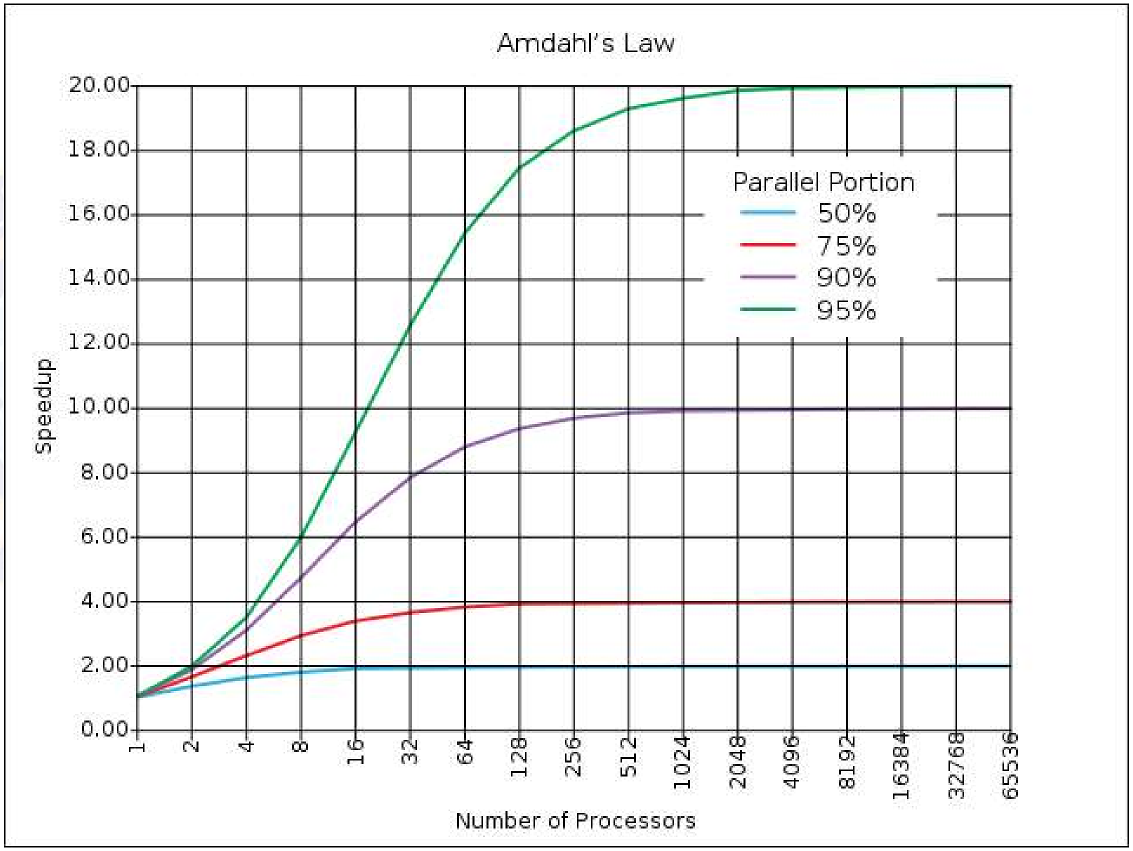
$$\text{Speed Up} = \frac{1}{(1-P) + P/N}$$

P = Parallel part (%) of the program
N = No. of Processors (Workers)

P = 0; Completely sequential – Speedup = 1
P = 1; Completely parallel – Speedup = N
P = 0.5; N=2; Partial Parallel – Speedup = 1.333

2007

The 40th Anniversary of Amdahl's Law



Amdahl's Law – (2)

- T(1) : Time taken for a job with 1 processor
- T(N) : Time taken for <u>same</u> job with N processors
- Speedup S(N) = T(1) / T(N)
- S(N) is ideally N when it is a perfectly parallelizable program, i.e. data parallel with no sequential component
- Assume fraction of a program that cannot be parallelised (serial) is f and 1-f is parallel
  - ✓ T(N) >= f * T(1) + (1-f) * T(1) / N

Only parallel portion is faster by N

- S(N) = T(1) / (f * T(1) + (1-f) * T(1) / N)
- S(N) = 1 / (f + (1-f) / N)
- <u>Implication</u> :
  - ✓ If N=>inf, S(N) => 1/f
  - ✓ The effective speedup is limited by the sequential fraction of the code

$$\text{Speed Up} = \frac{1}{(1-P) + P/N}$$

Amdahl’s Law – Example calculation

10% of a program is sequential (f) and there are 100 processors.

What is the effective speedup ?

S(N) = 1 / (f + (1-f) / N)

S(100) = 1 / (0.1 + (1-0.1) / 100)

= 1 / 0.109

= 9.17 (approx)

Super-linear speedup

Memory hierarchy / caches may increase the speedup

Single processor will have cache size D

But N processors will have total cache size N * D

So, in a parallel environment the effect of cache may go up to N fold

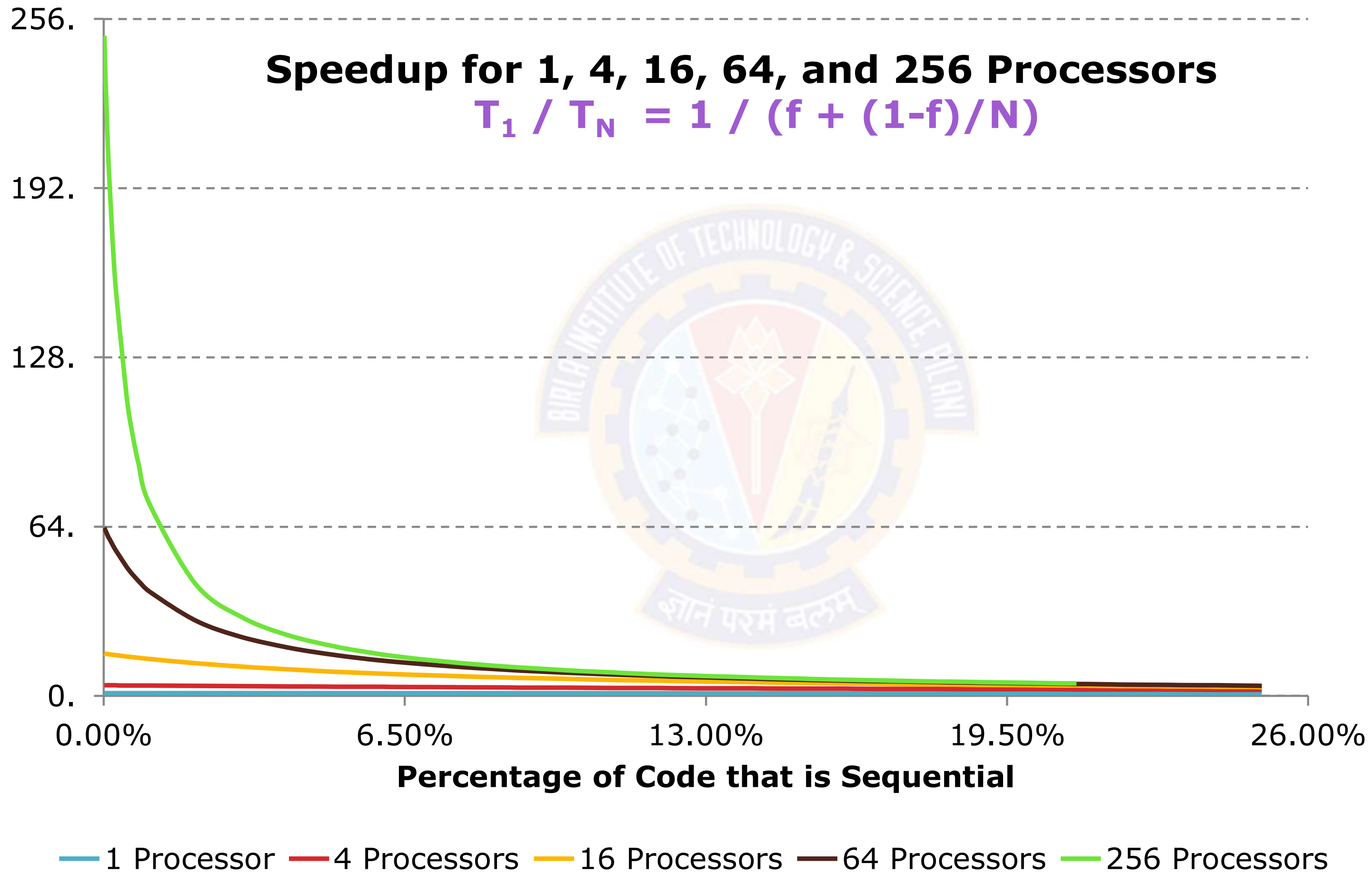
Partitioning a data parallel program to run across multiple processors may lead to a better cache hit.*

Possible in some workloads with not much of other overheads, esp in embarrassingly parallel applications.

* There could be also be cache coherency overheads.

Hadoop Superlinear Scalability - ACM Queue

Speedup plot



Why Amdahl's Law is such bad news

$S(N) = T_1 / T_N = 1 / (f + (1-f)/N)$ (f = percentage of the program that is sequential)

$S(N) \sim 1/f$, for large N

Suppose 33% of a program is sequential

- Then even a billion processors won't give a speedup over 3

- For the 256 cores to gain ≥100x speedup, we need
  $100 \leq 1 / (\mathbf{f} + (1-\mathbf{f})/256)$
  Which means **f** ≤ .0061 or 99.4% of the algorithm must be perfectly parallelizable !!

But wait - may be we are missing something

- The key assumption in Amdahl's Law is **total workload is fixed** as no of processors is increased (Sequential part doesn't increase with resources)
- This need not be happening in practice
- Additional processors can be used for more complex workload and new age larger parallel problems
- So, Amdahl's law under-estimates Speedup
  - What if we assume **fixed workload** per processor - (Gustafson-Barsis Law)
  - What if we factor resource contention and coherence also instead of serial fraction alone - (Universal Scalability Law)

Gustafson-Barsis Law - (1988)

Let W be the execution workload of the program before adding resources

f is the sequential part of the workload

So $W = f * W + (1-f) * W$

Let W(N) be larger execution workload after adding N processors

So $W(N) = f * W + \underline{N * (1-f) * W}$ **Parallelizable work can increase N times**

The theoretical speedup in latency of the whole task at a fixed interval time T

$$S(N) = T * W(N) / T * W$$

$$= W(N) / W = (f * W + N * (1-f) * W) / W$$

$$S(N) = f + (1-f) * N$$

S(N) is not limited by f as N scales

So solve larger problems when you have more processors

Usage Scenarios

- **Amdahl’s** law can be used when the workload is fixed, as it calculates the potential speedup with the assumption of a fixed workload. Moreover, it can be utilized when the non-parallelizable portion of the task is relatively large, highlighting the diminishing returns of parallelization.

- **Gustafson’s** law is applicable when the workload or problem size can be scaled proportionally with the available resources. It also addresses problems requiring larger problem sizes or workloads, promoting the development of systems capable of handling such realistic computations.

Actual Limitations in speedup

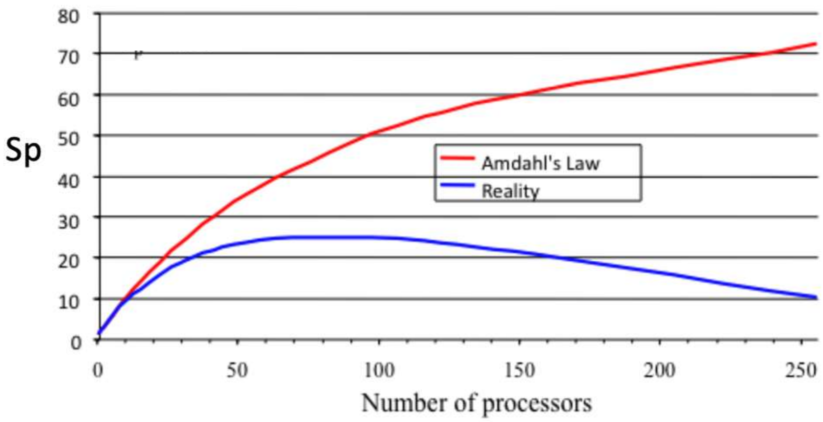
Besides the sequential component of the program, communication delays also result in reduction of speedup

A and B exchange messages in blocking mode

Say processor speed is 0.5ns / instruction

Say network delay one way is 10 us

For one message delay, A and B would have each executec 10us/0.5ns = 20000 instructions



+ context switching, scheduling, load balancing, I/O …

Universal Scalability Law

Universal Scalability Law (USL) extends Amdahl's Law by explicitly accounting for communication overhead between processors / nodes within a system

It is a mathematical model that helps predict how well a system will scale by considering factors like

- ✓ concurrency,
- ✓ contention, and
- ✓ coherency delay

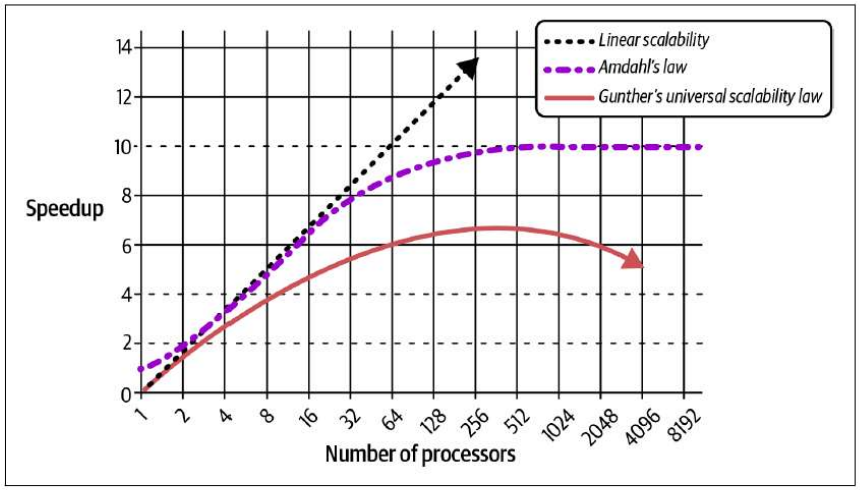
The Universal Scalability Law (Dr. Neil J. Gunther) is typically represented by an equation that relates relative capacity (C(N)) to the number of nodes (N), and parameters that capture the effects of contention and coherency delay:

$$C(N) = N / (1 + \alpha(N-1) + \beta N(N-1))$$

Where:

- **N:** The number of nodes (processors, cores, etc.) in the system.
- **α:** A parameter reflecting the level of contention within the system.
- **β:** A parameter capturing the coherency delay between nodes.

[$0 <= \alpha, \beta <= 1$]



http://www.perfdynamics.com/Manifesto/USLscalability.html

USL – Example Numerical problem

Let's say you have a system with $\alpha = 0.1$ and $\beta = 0.01$, and you want to determine the capacity (C(N)) when you have 10 resources (N=10). Using the USL formula $C(N) = N / (1 + \alpha(N-1) + \beta N(N-1))$:

$$C(10) = 10 / (1 + 0.1(10-1) + 0.01 * 10 * (10-1))$$
$$C(10) = 10 / (1 + 0.9 + 0.09)$$
$$C(10) = 10 / 2.99$$
$$C(10) \approx 3.34$$

This means that with 10 resources and the given contention and bottleneck parameters, you would expect a capacity of approximately 3.34, indicating that you are not getting a linear increase in performance with the number of resources.

Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing

Data Access Strategies: Partition

- Strategy:
  - ✓Partition data – typically, equally – to the nodes of the (distributed) system
- Cost:
  - ✓ Network access and merge cost when query needs to go across partitions
- Advantage(s):
  - ✓ Works well if task/algorithm is (mostly) data parallel
  - ✓ Works well when there is Locality of Reference within a partition
- Concerns
  - ✓ Merge across data fetched from multiple partitions
  - ✓ Partition balancing
  - ✓ Row vs Columnar layouts - what improves locality of reference ?
  - ✓ Will study shards and partition in Hadoop, **MongoDB**, and Cassandra

Data Access Strategies: Replication

- Strategy:
  - ✓ Replicate all data across nodes of the (distributed) system
- Cost:
  - ✓ Higher storage cost
- Advantage(s):
  - ✓ All data accessed from local disk: no (runtime) communication on the network
  - ✓ High performance with parallel access
  - ✓ Fail over across replicas
- Concerns
  - ✓ Keep replicas in sync — various consistency models between readers and writers
  - ✓ Will study in depth for MongoDB, Cassandra

Data Access Strategies – Networked Storage

- Common Storage on the Network:
  - ✓ Storage Area Network (for raw access – i.e. disk block access)
  - ✓ Network Attached Storage (for file access)

- Common Storage on the Cloud:
  - ✓ Use Storage as a Service
  - ✓ e.g. Amazon S3

Topics for today

- What are parallel / distributed systems
- Motivation for parallel / distributed systems
- Limits of parallelism
- Data access strategies - Replication, Partitioning, Messaging
- Cluster computing

Cluster - Goals

- Continuous availability
- Data integrity
- Linear scalability
- Open access
- Parallelism in processing
- Distributed systems management

Computer Cluster - Definition

- A cluster is a type of distributed processing system
  - ✓ consisting of a collection of inter-connected stand-alone computers
  - ✓ working together as a single, integrated computing resource
  - ✓Examples:
    - High Availability Clusters

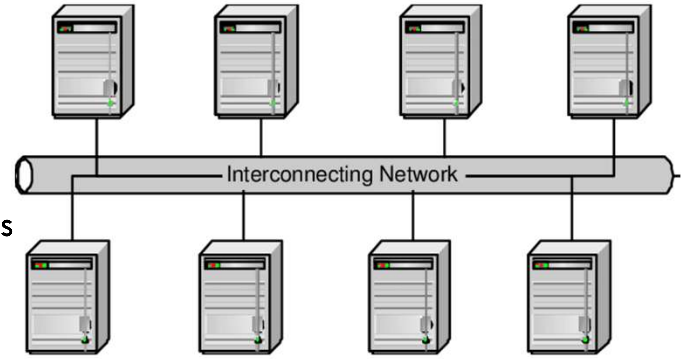
      ServiceGuard, Lifekeeper, Failsafe, heartbeat, HACMP, failover clusters
    - High Performance Clusters

      Beowulf; 1000 nodes; parallel programs; MPI
    - Database Clusters

      Oracle Parallel Server (OPS) - RAC
    - Storage Clusters

      Cluster filesystems; same view of data from each node.

      HDFS



Cluster - Objectives

- A computer cluster is typically built for one of the following two reasons:
  - ✓ High Performance - referred to as compute-clusters
  - ✓ High Availability - achieved via redundancy

  An off-the-shelf or custom load balancer, reverse proxy can be configured to serve the use case

- Question: How is this relevant for Big Data?

  Hadoop nodes form a cluster for **performance** (independent Map/Reduce jobs are started on multiple nodes) and **availability** (data is replicated on multiple nodes for fault tolerance)

Most Big Data systems run on a cluster configured for performance and availability

Motivation for using Clusters (1)

- Rate of obsolescence of computers is high
  - ✓ Even for mainframes and supercomputers
  - ✓ Servers (used for high performance computing) are to be replaced every 3 to 5 years.
- Solution: Build a cluster of commodity workstations
  - ✓ Incrementally add nodes to the cluster to meet increasing workload
  - ✓ Add nodes instead of replacing (i.e. let older nodes operate at a lower speed)
  - ✓ This model is referred to as a scale-out cluster

Motivation for using Clusters (2)

- Scale-out clusters with commodity workstations as nodes are suitable for software environments that are resilient:
  - ✓ i.e. individual nodes may fail, but
  - ✓ middleware and software will enable computations to keep running (and keep services available) for end users
  - ✓for instance, back-ends of Google and Facebook use this model.

- On the other hand, (public) cloud infrastructure is typically built as clusters of servers
  - ✓ due to higher reliability of individual servers – used as nodes – (compared to that of workstations as nodes).

Cluster Middleware - Some Functions

Single System Image (SSI) infrastructure

- ✓ Glues together OSs on all nodes to offer unified access to system resources
  - ✓ Single process space
  - ✓ Cluster IP or single entry point
  - ✓ Single auth
  - ✓ Single memory and IO space
  - ✓ Process checkpointing and migration
  - ✓ Single IPC space
  - ✓ Single fs root
  - ✓ Single virtual networking
  - ✓ Single management GUI

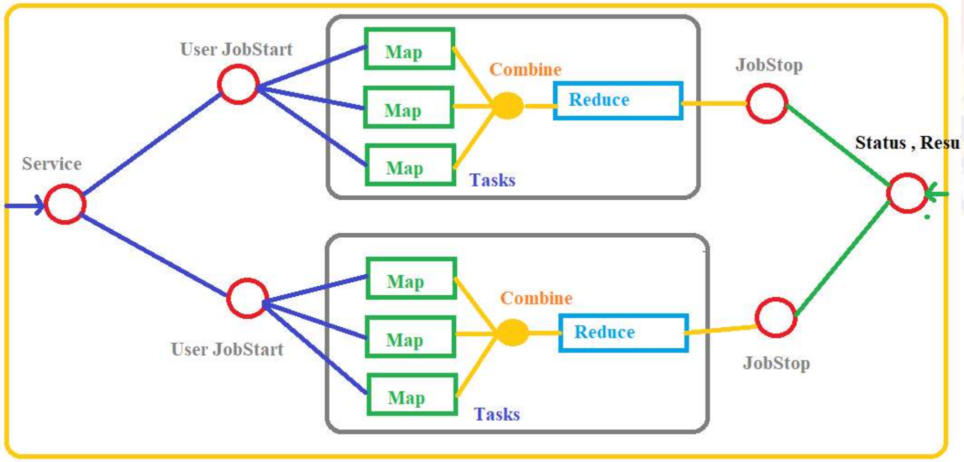
High Availability (HA) Infrastructure

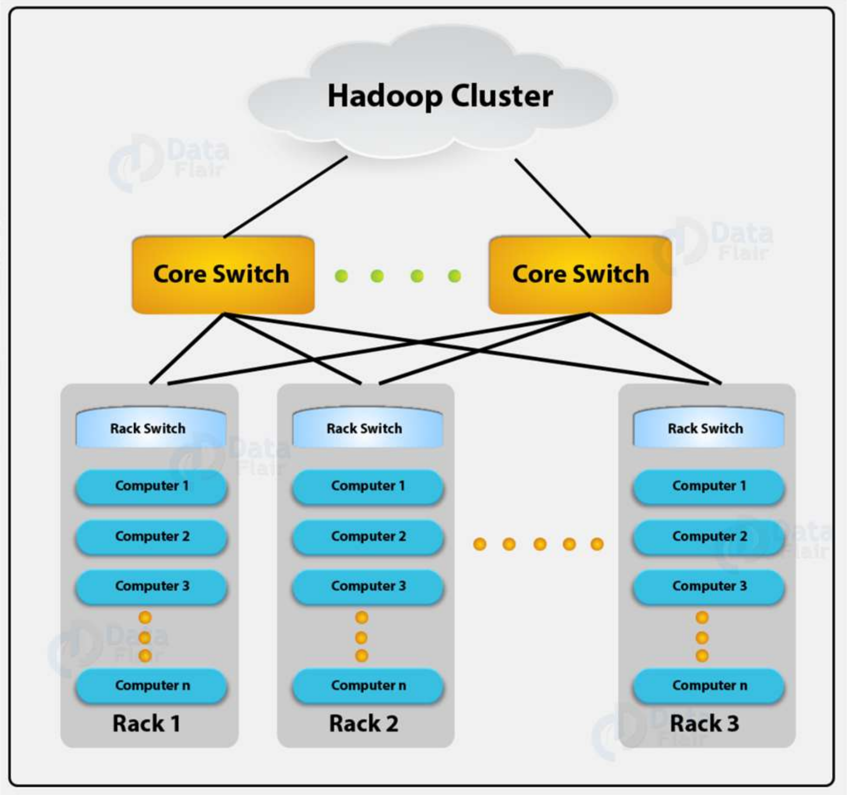
- ✓ Cluster services for
  - ✓ Availability
  - ✓ Redundancy
  - ✓ Fault-tolerance
  - ✓ Recovery from failures

http://www.cloudbus.org/papers/SSI-CCWhitePaper.pdf

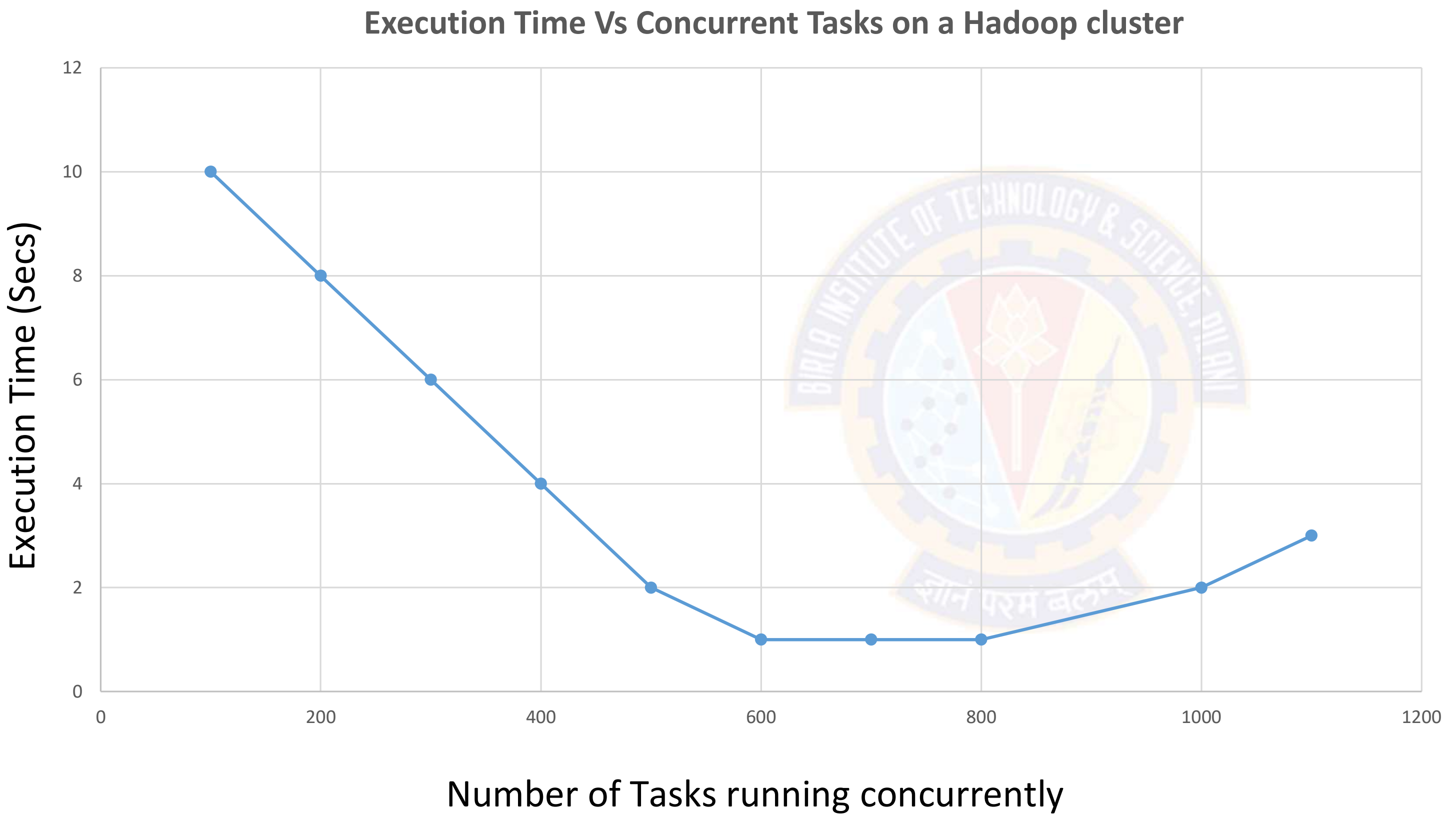
Example cluster: Hadoop

- A job divided into tasks
- Considers every task either as a Map or a Reduce
- Tasks assigned to a set of nodes (cluster)
- Special control nodes manage the nodes for resource management, setup, monitoring, data transfer, failover etc.
- Hadoop clients work with these control nodes to get the job done





Limits of Parallelism in distributed computing



Overheads for

✓ Distributed Scheduling

✓ Local on node scheduling

✓ Communication

✓ Synchronization, etc.

Summary

- Motivation and classification of parallel systems
- Computing limits of speedup
- How replication, partitioning helps in Big Data storage and access
- Cluster computing basics

Next Session:

Fault Tolerance, Big Data Analytics and Systems