Lecture 1

Course Introduction and Overview

Dr. Soner Onder

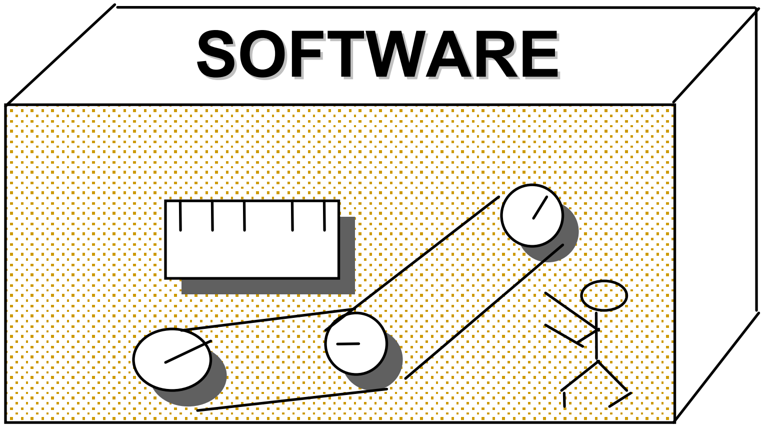
CS 4431

Michigan Technological University

Computer Architecture Is …

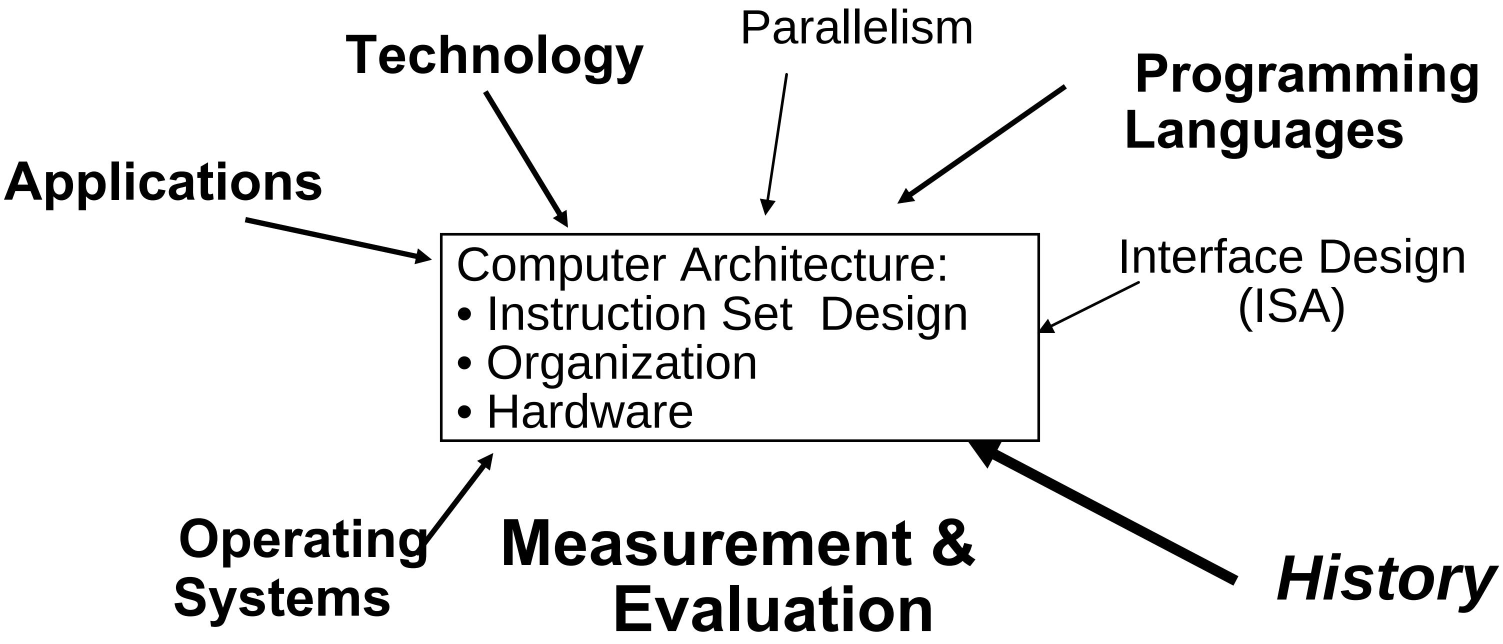
the attributes of a [computing] system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

Amdahl, Blaaw, and Brooks, 1964



CS 4431 Course Focus

Understanding the design techniques, machine structures, technology factors, evaluation methods that will determine the form of computers in 21st Century



Technology

Parallelism

Programming Languages

Applications

Computer Architecture:
- Instruction Set Design
- Organization
- Hardware

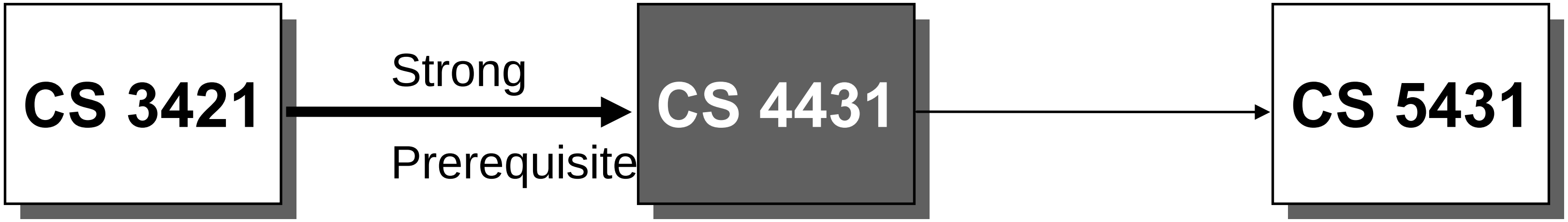
Interface Design (ISA)

Operating Systems

Measurement & Evaluation

History

Related Courses



CS 3421

Strong

Prerequisite

CS 4431

CS 5431

How to build it
Implementation details

Why, Analysis,
Evaluation

Parallel Architectures,
Languages, Systems

Grading

- 30 % Project
  - To be done individually
  - Consists of multiple phases
- 10 % Homework assignments
- 30 % 2 Midterm Exams
- 30 % Final Exam

- For success in this course you must turn in the project and/or homework assignments on time.
- Attendance is important.

Topic Coverage

Textbook: Michel Dubois, Murali Annavaram, Per Stenstrom, ***Parallel Computer Organization and Design*.**

1. **Introduction :** Components of a parallel architecture, parallelism in architectures, performance, technological challenges.
2. **Impact of technology :** Basic laws of electricity, technology scaling, power and energy, reliability.
3. **Processor microarchitecture :** Instruction set architecture, statically scheduled pipelines, dynamically scheduled pipelines, VLIW and Vector microarchitectures.
4. **Memory hierarchy :** Caches and virtual memory.
5. **Multiprocessor systems :** Message passing architectures, bus-based architectures, scalable systems, cache only parallel machines.
7. **Coherence and synchronization :** Store atomicity, sequential consistency, synchronization.
8. **Chip multiprocessors :** Core multi-threading, chip multi-processor architectures.

Introduction

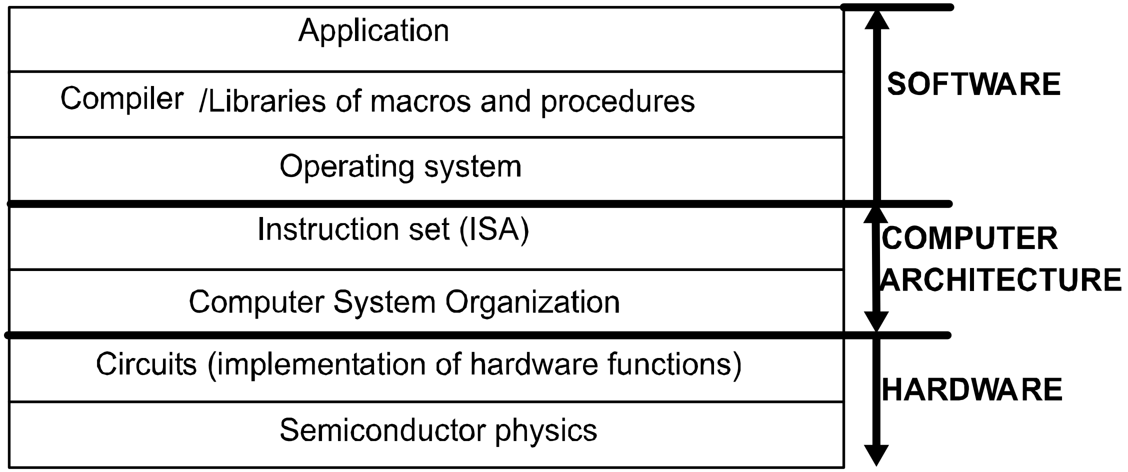
- Computer architecture: definition
- System components
- Technological factors and trends
- Performance metrics and evaluation
- Quantitative principles of computer design

What is Computer Architecture?

Old definition: Instruction Set Architecture (ISA)

Today's definition is much broader: hardware organization of computers (how to build computer)--includes ISA.

Layered view of computer systems:



Application

Compiler /Libraries of macros and procedures

Operating system

Instruction set (ISA)

Computer System Organization

Circuits (implementation of hardware functions)

Semiconductor physics

SOFTWARE

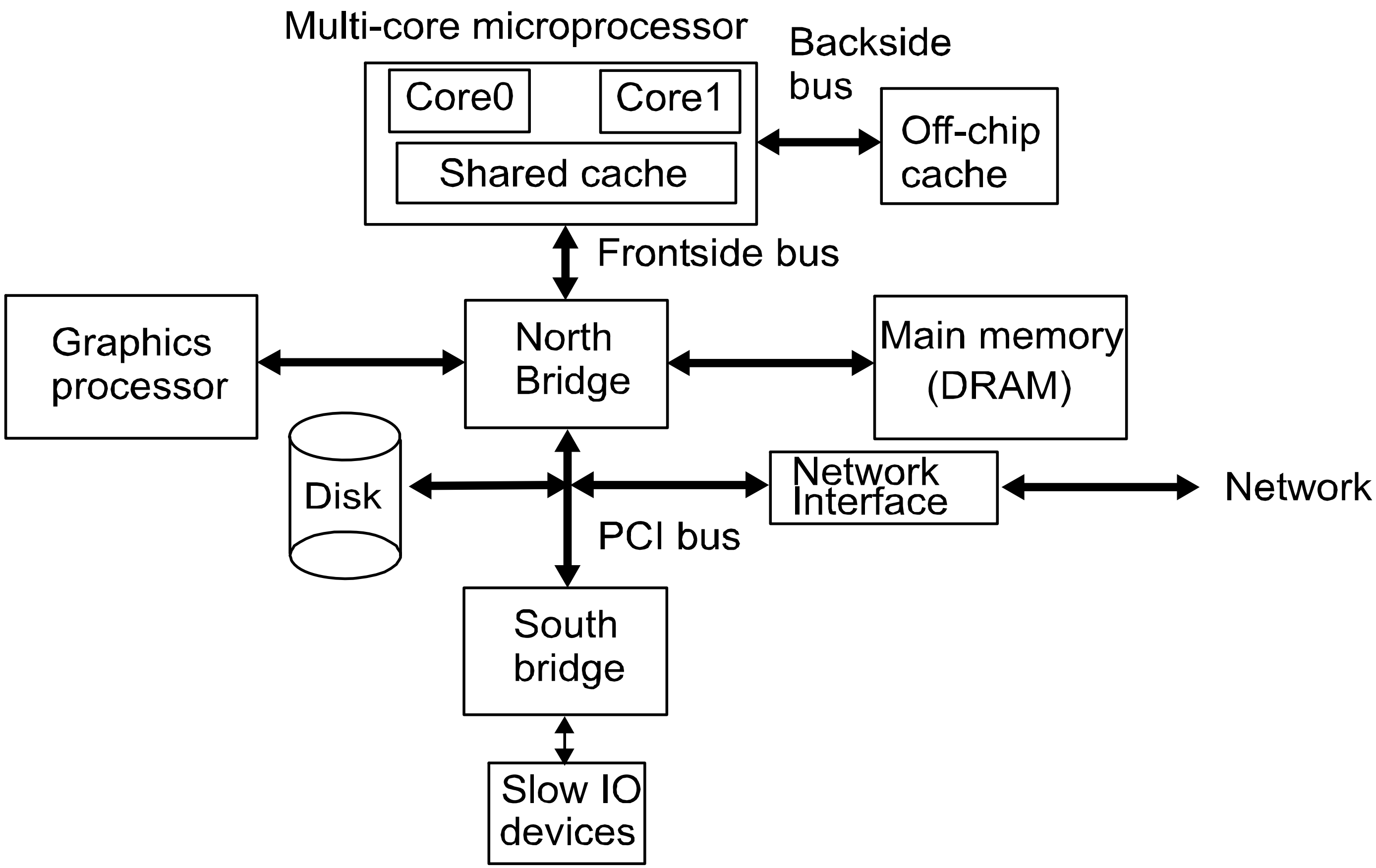
COMPUTER ARCHITECTURE

HARDWARE

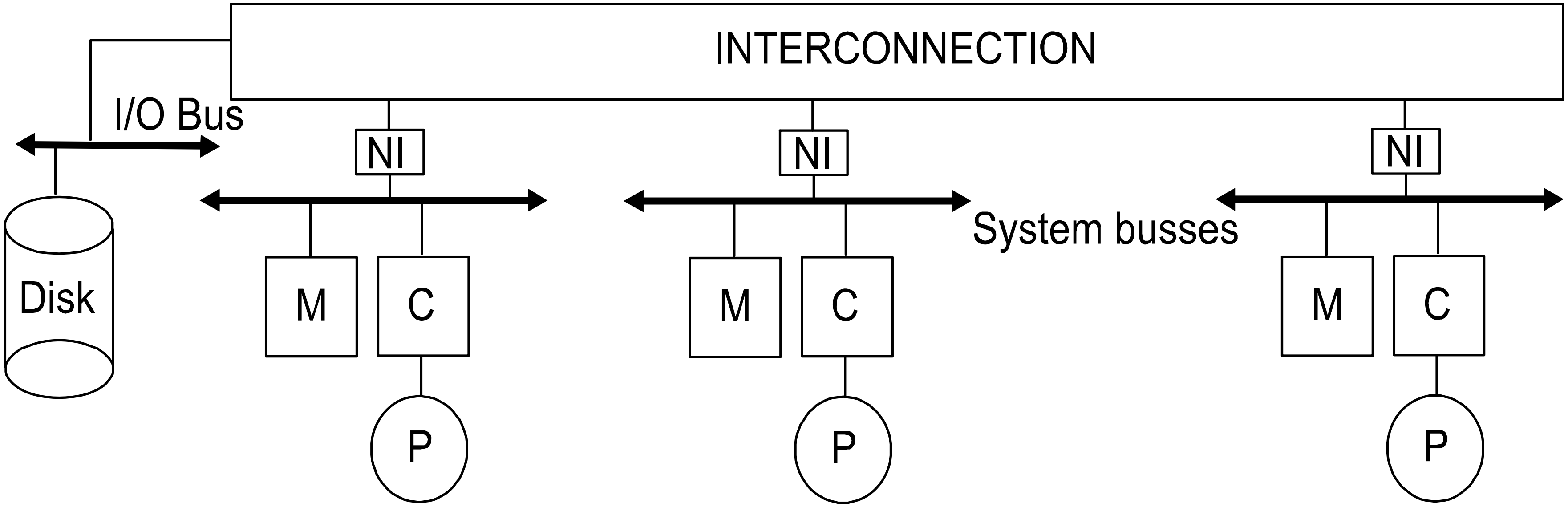
Role of the computer architect:

To make design trade-offs across the hw/sw interface to meet functional, performance and cost requirements.

Organization of a modern PC architecture



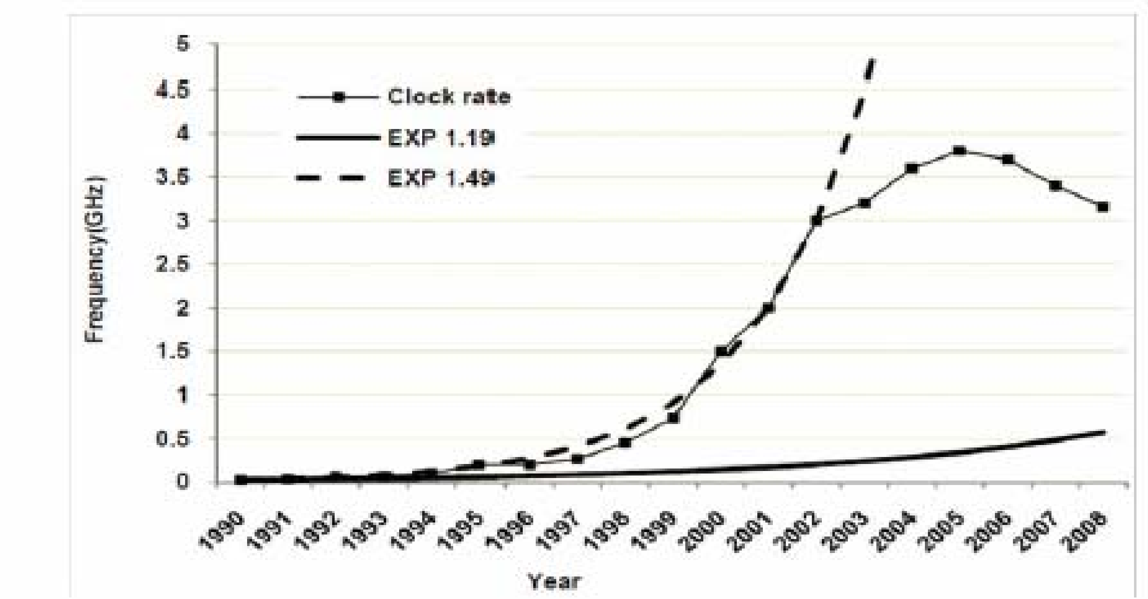
Organization of a generic high-end parallel system



Processor architecture

Historically the clock rates of microprocessors have increased exponentially due to process improvements, deeper pipelines and circuit design techniques.

Highest clock rate of intel processors in every year from 1990 to 2008 :



This historical trend has subsided over the past 10 years. **IF IT HAD KEPT UP, TODAY'S CLOCK RATES WOULD BE MORE THAN 30ghz!!!!**

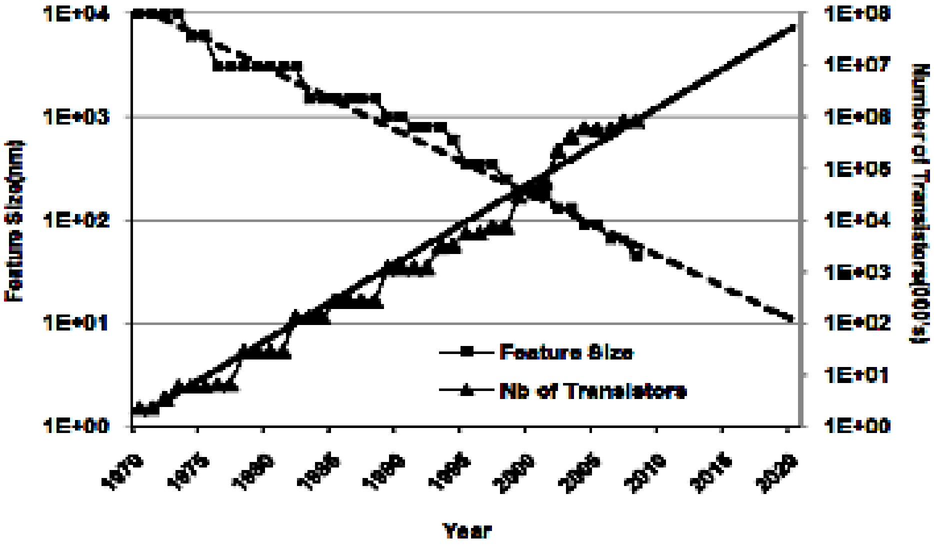
Processor architecture

Pipelining (i.e., architecture) and circuit techniques have greatly contributed to the dramatic rise of the clock rate

The 1.19 curve corresponds to process improvements alone,

rest is due to architecture and circuits.

Additionally computer architects take advantage of the growing number of circuits.



New process every 2 year

feature size reduced by 30% every process

or halved every 5 years

Number(b) of transistor doubles every 2 years (Moore's law)

1B transistors reached in 2008

100B in 2021

A sandbox to play in so to speak. How do we use 100b transistors????

Can this trend continue?

MEMORY SYSTEMS

Main memory speed is not growing as fast as processors' speed. Growing gap between processor and memory speed (the so-called "memory wall")

One wants to design a memory system that's big, fast and cheap

- The approach is to use a multi-level hierarchy of memories
- Memory hierarchies rely on principle of locality
- Efficient management of the memory hierarchy is key

Cost and size of memories in a basic pc (2008):

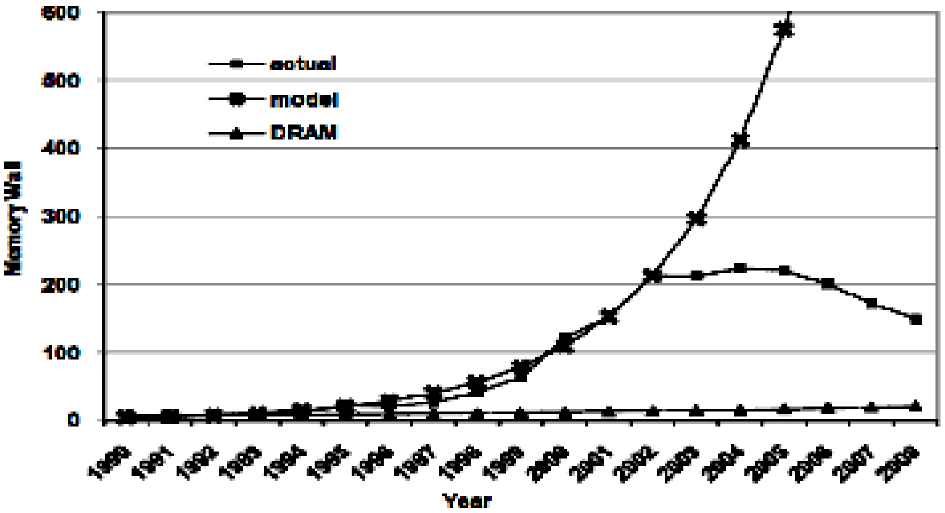
Memory	Size	Marginal Cost	Cost per MB	Access Time
L2 Cache (on chip)	**1MB**	**$20/MB**	**$20**	**5 nsec**
Main Memory	**1GB**	**$50/GB**	**5c**	**200 nsec**
Disk	**500GB**	**$100/500GB**	**0.02c**	**5msec**

Memory wall?? Which memory wall??

Historically, microprocessor speed has increased by 50% a year while dram performance improved by 7% a year.

Although dram density keeps increasing by 4x every 3 years, this created the perception that this problem would last forever.

Trends have changed dramatically in the past 6 years. The “memory wall” (relative performance of processors vs dram).



DRAM: 1.07 CGR

Memory wall =
memory_cycle/processor_cycle

In 1990, it was about 4 (25MHz,150ns).
Grew to 200 exponentially until 2002
Has tappered off since then

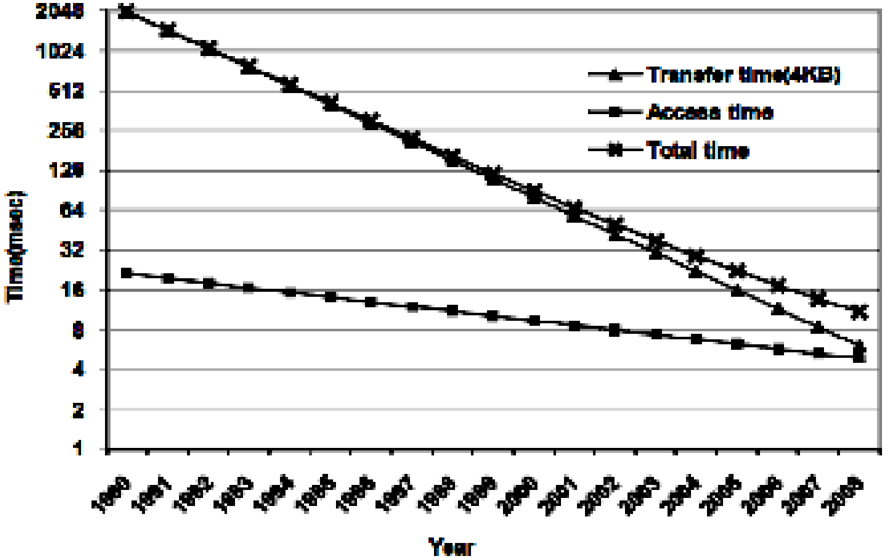
Although still a big problem, the memory wall stopped growing around 2002.

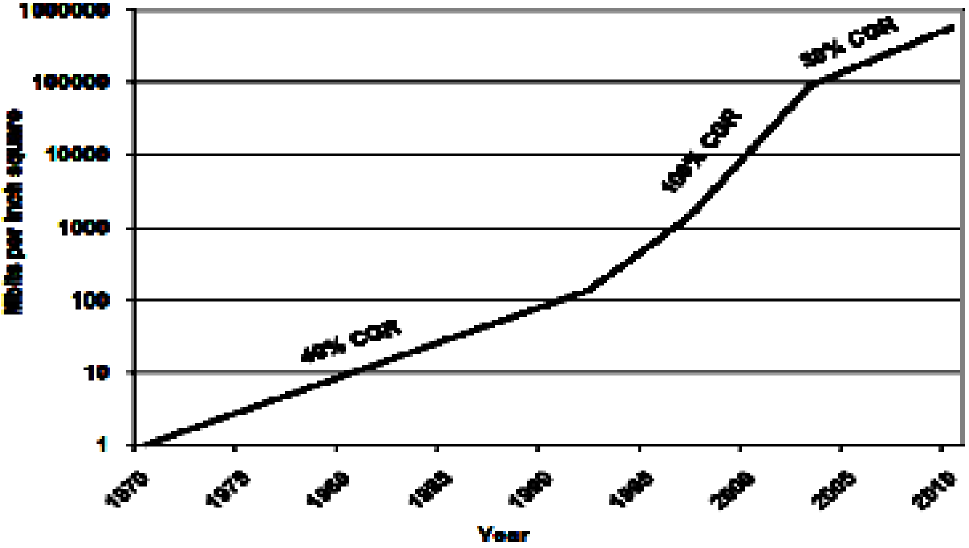
With the advent of multicore microarchitectures the memory problem has shifted from latency to bandwidth

DISK

Historically disk performance & density improved by 40%

Disk time = access time + transfer time





Historically transfer times have dominated

But today transfer and access times are of the same order

In future access time will dominate (much slower curve)

Note: all these times are still in the order of milliseconds :➔must switch context.

Networks

Networks are present at many levels:

1. ***On-chip interconnects*** forward values from and to different stages of a pipeline and among execution units AND connect cores to shared cache banks.
2. ***System interconnects*** connect processors (CMPs) to memory & i/o.
3. ***I/o interconnects*** (usually a bus such as e.g., PCI) connect various I/O devices to the system bus.
4. ***Inter-system interconnects*** connect separate systems (separate chassis or box) and includes
    1. Sans (system-area networks --connecting systems at very short distances),
    2. Lan (local area networks --connecting systems within an organization or a building),
    3. Wan (wide area networks --connecting multiple LANs at long distances).

5. ***Internet***. Most computing systems are connected to the internet, which is a global, worldwide interconnect.

Parallelism in architectures

The most successful microarchitecture has been the scalar processor.

1. A typical scalar instruction operates on scalar operands :
    1. Add o1,o2,o3 o1 <=o2+o3

2. Execute multiple scalar instructions at a time. Takes advantage of **ILP**,i.e., Instruction-level parallelism, the parallelism exposed in single thread or single process execution :
    1. *Pipelining.*
    2. *Superscalar.*
    3. *Superpipelining.*

3. CMPs (chip multiprocessors) exploits parallelism exposed by different threads running in parallel :Thread level parallelism or TLP. Can be seen as multiple scalar processors running in parallel.

Parallelism in architectures

Vector and array processors

A typical vector instruction executes directly on vector operands:

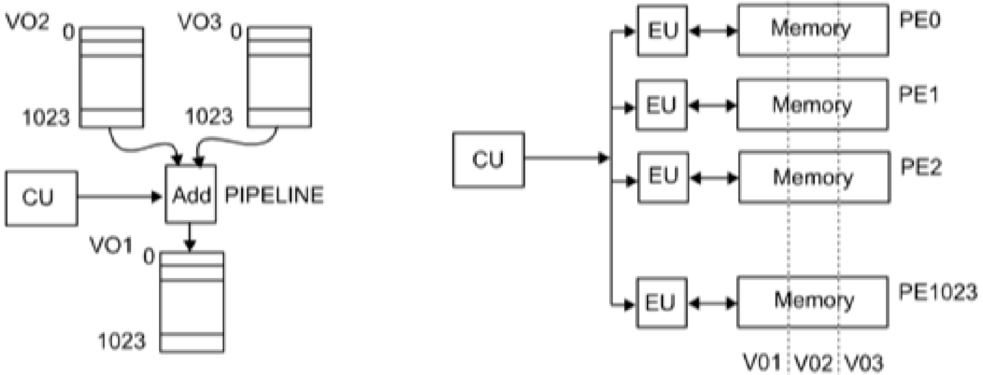
Vadd vo1,vo2,vo3   vo1 <= vo2+vo3

Vo_k is a vector of scalar components

Equivalent to computing

Vo1[i] <= Vo2[i]+vo3[i], i=0,..,n

Vector instructions are executed by pipelines or parallel arrays :



(a) Vector

(b) Array

Power

Total power is dynamic + static (leakage)

$$P_{dynamic} = \alpha c v^2 f$$

$$P_{static} = v i_{sub} \approx v e^{-kvt/t}$$

Dynamic power : favors parallel processing over higher clock rate.

Dynamic power roughly proportional to f^3.

Take a U.P. And replicate it 4 times: 4x speedup & 4x power.

Take a U.P. And clock it 4 times faster: 4x speedup but 64x dynamic power!

Static power : because circuits leak whatever the frequency is.

Power/energy are critical problems :

Power (immediate energy dissipation) must be dissipated.

Otherwise temperature goes up (affects performance, correctness and may possibly destroy the circuit, short term or long term).

Effect on the supply of power to the chip.

Energy (depends on power and speed):

Costly; global problem.

Battery operated devices.

Reliability

Transient failures (or soft errors):

- Charge q = c x v ← If c and v decrease then it is easier to flip a bit.
- Sources are cosmic rays and alpha particles radiating from the packaging material.
- Device is still operational but value has been corrupted. Should detect/correct and continue execution.
- Also: electrical noise causes similar failures.

Intermittent/temporary failures:

- Last longer and is due to :
  1. Temporary: environmental variations (eg, temperature).
  2. Intermittent: aging.
- Should try to continue execution.

Permanent failures:

- Means that the device will never function again.
- Must be isolated and replaced by spare.

Process variations increase the probability of failures

Wire delays

Wire delays don’t scale like logic delays.

Processor structures must expand to support more instructions.

Thus wire delays dominate the cycle time; slow wires must be local.

Design complexity

Processors are becoming so complex that a large fraction of the development of a processor or system is dedicated to verification.

Chip density is increasing much faster than the productivity of verification. engineers (new tools, speed of systems).

CMOS endpoint

CMOS is rapidly reaching the limits of miniaturization.

- Feature sizes will reach atomic dimensions in less than 15 years.
- Options?
  - Quantum computing?
  - Nanotechnology?
  - Analog computing?

Performance remains a critical design factor

Performance metrics (measure)

Metric #1 : Time to complete a task (t_{exe}): Execution time, Response time, Latency.

"X is N times faster than y" means : $t_{exe}(y)/t_{exe}(x) = N$

The major metric used in this course.

Metric #2 : number of tasks per day, hour, sec, ns

The throughput for x is n times higher than y if throughput(x)/throughput(y) = n

Not the same as latency (example of multiprocessors).

Examples of unreliable metrics:

Mips: millions of instructions per second.

Mflops: millions of floating point operations per second.

Execution time of a program is the ultimate measure of performance.

Benchmarking.

Which program to choose?

1. *Real programs*: Porting problem; complexity; not easy to understand the cause of results.
2. *Kernels*: Computationally intense piece of real program.
3. *Toy benchmarks* : e.g. Quicksort, matrix multiply.
4. *Synthetic benchmarks* : not real.
5. *Benchmark suites*:

Spec: standard performance evaluation corporation

Scientific/engineeing/general purpose

Integer and floating point

New set every so many years (95,98,2000,2006)

Tpc benchmarks:

For commercial systems.

Tpc-b, tpc-c, tpc-h, and tpc-w

Embedded benchmarks.

Media benchmarks.

Reporting performance for a set of programs

Let t_i be the execution time of program i:

1. (Weighted) arithmetic mean of execution times:

$$\sum_i T_i / N \qquad \text{Or} \qquad \sum_i T_i \times W_i$$

The problem here is that the programs with longest execution times dominate the result.

2. Dealing with speedups

Speedup measures the advantage of a machine over a reference machine for a program i

$$S_i = \frac{T_{R,\,i}}{T_i}$$

Arithmetic mean of speedups.

Harmonic mean :

$$\bar{S} = \frac{N}{\sum_i^N \frac{1}{S_i}} = \frac{N}{\sum_i^N \frac{T_i}{T_{R,\,i}}}$$

Reporting performance for a set of programs

Geometric means of speedups:

$$\bar{S} = \sqrt[N]{\prod_{i=1}^{N} S_i}$$

Mean speedup comparions between two machines are independent of the reference machine.

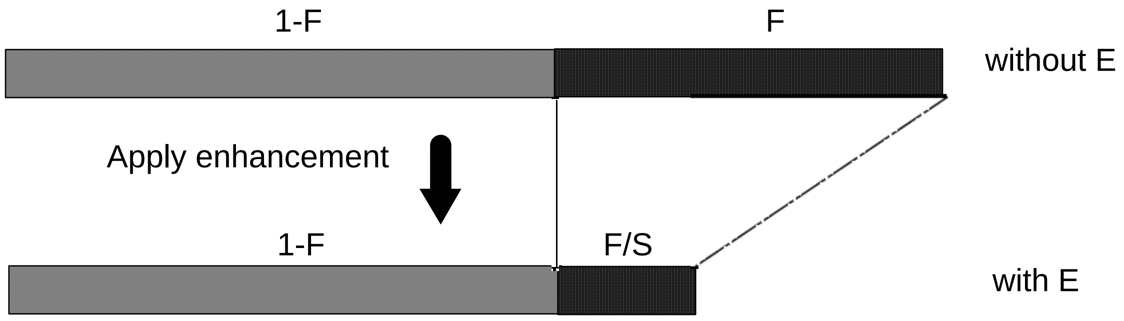
Easily composable.

Used to report spec numbers for integer and floating point.

	Program A	Program B	Arithmetic Mean	Speedup (ref 1)	Speedup (ref 2)
Machine 1	**10 sec**	**100 sec**	**55 sec**	**91.8**	**10**
Machine 2	**1 sec**	**200 sec**	**100.5 sec**	**50.2**	**5.5**
Reference 1	**100 sec**	**10000 sec**	**5050 sec**		
Reference 2	**100 sec**	**1000 sec**	**550 sec**		

		Program A	Program B	Arithmetic	Harmonic	Geometric
Wrt Reference 1	**Machine 1**	**10**	**100**	**55**	**18.2**	**31.6**
	Machine 2	**100**	**50**	**75**	**66.7**	**70.7**
Wrt Reference 2	**Machine 1**	**10**	**10**	**10**	**10**	**10**
	Machine 2	**100**	**5**	**52.5**	**9.5**	**22.4**

Amdahl's law



Enhancement E accelerates a fraction F of the task by a factor S

$$T_{exe}(\text{withE}) = T_{exe}(\text{withoutE}) X \left[(1-F) + \frac{F}{S}\right]$$

$$\text{Speedup}(E) = \frac{T_{exe}(\text{withoutE})}{T_{exe}(\text{withE})} = \frac{1}{(1-F) + \frac{F}{S}}$$

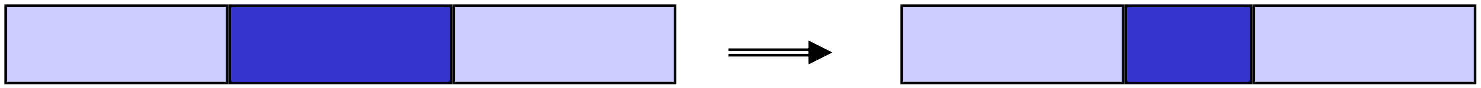
Amdahl's Law

$$\mathrm{ExTime}_{\mathrm{new}} = \mathrm{ExTime}_{\mathrm{old}} \times \left[\left(1 - \mathrm{Fraction}_{\mathrm{enhanced}}\right) + \frac{\mathrm{Fraction}_{\mathrm{enhanced}}}{\mathrm{Speedup}_{\mathrm{enhanced}}} \right]$$

$$\mathrm{Speedup}_{\mathrm{overall}} = \frac{\mathrm{ExTime}_{\mathrm{old}}}{\mathrm{ExTime}_{\mathrm{new}}} = \frac{1}{\left(1 - \mathrm{Fraction}_{\mathrm{enhanced}}\right) + \frac{\mathrm{Fraction}_{\mathrm{enhanced}}}{\mathrm{Speedup}_{\mathrm{enhanced}}}}$$

Best you could ever hope to do:

$$\mathrm{Speedup}_{\mathrm{maximum}} = \frac{1}{\left(1 - \mathrm{Fraction}_{\mathrm{enhanced}}\right)}$$



Amdahl’s Law example

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

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

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

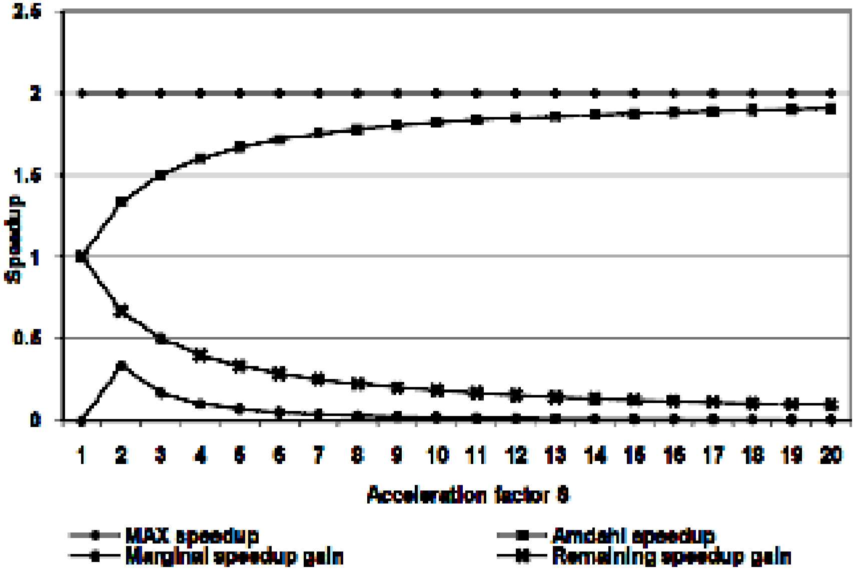
- Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster

Lessons from amdahl’s law

Improvement is limited by the fraction of the execution time that cannot be enhanced :

$$\mathrm{SPEEDUP}(E) < \frac{1}{1-F}$$

Law of diminishing returns :



F=0.5

Optimize the common case

Execute the rare case in software (e.G. Exceptions)

Fundamental performance Equations for CPUs

$$T_{exe} = IC \times CPI \times T_C$$

IC: Depends on program, compiler and ISA.

CPI: Depends on instruction mix, ISA, and implementation.

T_c: Depends on implementation complexity and technology.

CPI (Cycles per instruction) is often used instead of execution time.

When processor executes more than one instruction per clock use IPC (instructions per clock)

$$T_{exe} = (IC \times T_c) / IPC$$

Processor Performance Equation

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization	X		X
Technology			X

Revisiting Performance

Performance is in units of things per sec bigger is better

If we are primarily concerned with response time

performance(x) = 1 / execution_time(x)

" X is n times faster than Y" means

$$n = \frac{\text{Performance(X)}}{\text{Performance(Y)}} = \frac{\text{Execution\_time(Y)}}{\text{Execution\_time(X)}}$$

Relative Performance Metrics

Percent Improvement in Performance

“X is *n*% faster than Y” means:

$$n \equiv 100 \left[\frac{Perf_X - Perf_Y}{Perf_Y} \right] = 100 \left[\frac{Perf_X}{Perf_Y} - \frac{Perf_Y}{Perf_Y} \right]$$

$$n = 100 \left[S_{X/Y} - 1 \right]$$

Example:

Y takes 15 seconds to complete a task,

X takes 10 seconds to complete the same task.

Revisiting Amdahl's Law

Example: An enhancement (E) improves the speed of Floating Point (FLP) instructions by a factor of 2.

S_E = 2

$$S_{E/0} = \frac{1}{(1 - F_E) + \left(\frac{F_E}{2}\right)}$$

Where: F_E = the fraction of FLP instructions in Program

e.g. General Pgm: If F_E = 0.1, Then $S_{E/0}$ =

e.g. Scientific Pgm: If F_E = 0.9, Then $S_{E/0}$ =

Cycles Per Instruction

Calculating CPI:

- F_k $\equiv$ *Fraction of instructions of type* k, k $\in$ {*1*…m}
- CPI_k $\equiv$ *CPI for instruction type k*
- CPI $\equiv$ *Mean CPI for entire program*

$$CPI = \sum_{k=1}^{m} F_k \times CPI_k$$

Conclusion: Invest Resources where time is Spent!

- Focus on instruction types for which (F_k x CPI_k) is largest

Example: Calculating CPI

Typical Instruction Mix

Type (k)	F_k	CPI_k	F_k*CPI_k	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
CPI =			**1.5**	

Example

Suppose we have the following measurements:

Frequency of FP operations: 25 %

Average CPI of FP operations: 4.0

Average CPI of other instructions: 1.33

Frequency of FPSRQ: 2 %

CPI of FPSQR: 20

Assume two design alternatives: Decrease the CPI of FPSQR to 2, or decrease the average CPI of all FP operations to 2.5. Use processor performance equation to calculate.

Observe that the clock speed and instruction count remain identical.

Find original CPI first:

$\text{CPI} = 4 \text{ x } 0.25 + 1.33 \text{ x } 0.75 = 2.0$

$\text{CPI}_{\text{new sqrt}} = \text{CPi}_{\text{original}} - 0.02 \text{ x } (\text{CPI}_{\text{old FPSQRT}} - \text{CPI}_{\text{of new FPSQRT}})$

$= 2.0 - 0.02 * (20 - 2) = 1.64$

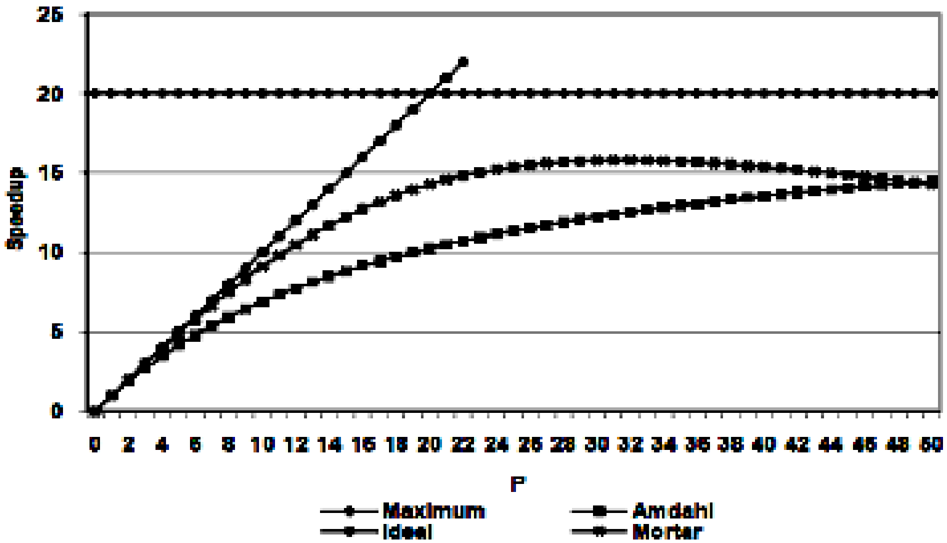
$\text{CPI}_{\text{new FP}} = (0.75 \text{ x } 1.33) + (0.25 \text{ x } 2.5) = 1.62$

$\text{Speed-up} = \text{CPI}_{\text{original}} / \text{CPI}_{\text{new FP}}$

$_{=}\ 2.0 / 1.62 = 1.23 = 1.23$

Parallel speedup

$$S_P = \frac{T_1}{T_P} = \frac{1}{1 - F + F \S P} = \frac{P}{F + P(1 - F)} < \frac{1}{1 - F}$$



F=0.95

Note: Speedup can be superlinear. How can that be??

Overall not very hopeful.

Gustafson’s law

Redefine speedup:

The *rationale* is that, as more and more cores are integrated on chip over time, the workloads are also growing.

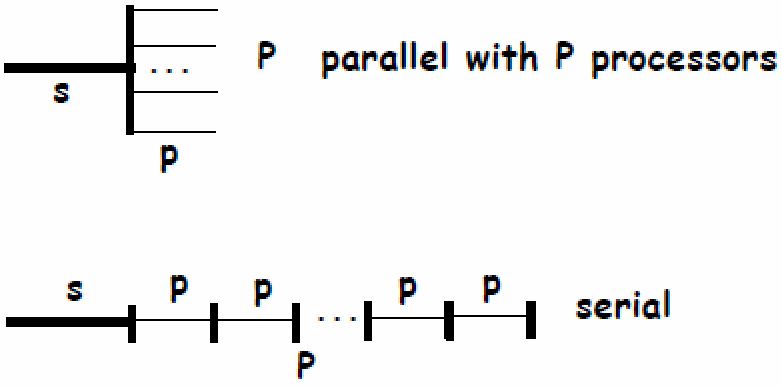
Starts with the execution time on the parallel machine with p processors:

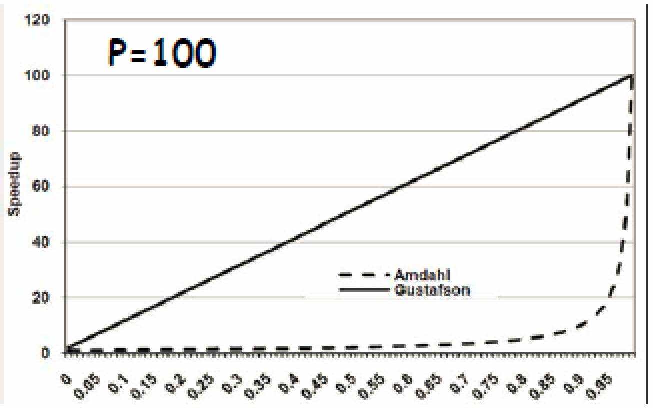
$$T_P = s + p$$

s is the time taken by the serial code and *p* is the time taken by the parallel code.

Execution time on one processor is : $T_1 = s + pP$

Let f=p/(s+p). Then S_P = (s+pP)/(s+p) = 1-F+FP = **1+F(P-1)**


s
. . .
p
P
parallel with P processors
s
p
p
. . .
p
p
P
serial


P=100
Speedup
Amdahl
Gustafson