Advanced Computer Architecture

MS (Computer Science)
Semester: Fall 2023

Lecture 01

Dr. Khurram Bhatti
Associate Professor

khurram.bhatti@itu.edu.pk

About the Instructors!

- Khurram Bhatti
  - Associate Professor @ ITU [7.5 Years]
  - Post-Doctorate (2013-2014)
    - KTH Royal Institute of technology, Stockholm, Sweden
    - High Performance Computing (HPC)/ Massively Parallel Systems
  - PhD & MS (2006-2011)
    - University of Nice-Sophia Antipolis, Nice, France
    - Real-time Embedded Systems
  - Bachelors (2000-2003)
    - NED UET, Karachi, Pakistan
    - Industrial Electronics

About the Instructors!

- Khurram Bhatti
  - Research Interests (Potential Theses for you)
    - Information/Hardware Security
    - Vulnerability, Side-Channel Leakage assessment in Blockchain (National Center of Excellence in Cyber Security, NCCS)
    - Side & Covert Channel Vulnerability Assessment, detection and mitigation
    - Tech Interventions for Climate Change-induced challenges
    - AI for Earth!

http://itu.edu.pk/faculty-itu/dr-khurram-bhatti/
http://itu.edu.pk/research/eclab/

About the Instructors!

- Teaching Assistant:

Muhammad Shoaib

BSEE –IIUI

MSEE –ITU

PhD Candidate –UET

Practical Information

Grading and Marks Distribution

- Grading is Relative
  - Assignments/Projects: 10%
  - Quizzes: 15%
  - Mid-term: 25%
  - Final Exam: 50%

Practical Information

- Academic Integrity
  - Cooperation is allowed, cheating is NOT!
  - ZERO tolerance on PLAGIARISM –can result into a failure of course entirely.
  - Assignments are individual (sometimes group) tasks and they are designed to help you better understand the course –Don't copy assignments!
  - Quizzes will be at random
  - Be respectful in terms of time and classroom decorum!
  - Interact – it helps

About You!

About the Course: ACA

- Course Objectives
  - How does computing hardware work
  - Where is computing hardware going in the future? – And learn how to contribute to this future…
  - How does this impact system software and applications?
  - Essential to understand OS/compilers
  - How are future technologies going to impact computing systems?
  - Research Methodology –Do's and Don't of research (on the sidelines)
- Prerequisite Knowledge Expected
  - Basic understanding of Digital Logic Design

About the Course: ACA

– Course Resources

- Books:
- Computer Organization & Design, the hardware/software interface, 5th Edition, Hennessy and Patterson
- Computer Architecture: A Quantitative Approach, 5th Edition, Hennessy and Patterson
- Research/semester Projects will be provided as the course evolves
- Assignments will help
- Tutorials on review and specific topics can be arranged
- Guest Lectures will be arranged
  - At least 01, at most 02

About the Course: ACA

- Broader Outline

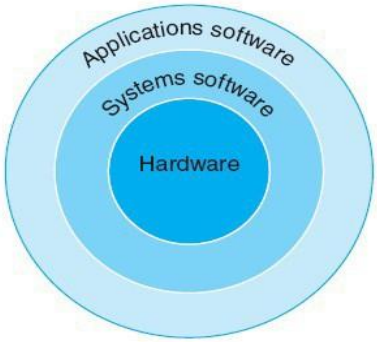
- Introduction and evolution of computers
- Operations and operands
- Arithmetic for computers, signed and unsigned numbers
- Assessing performance of computers
- Processor datapath and control, single cycle datapath, multicycle datapath
- Fundamentals of quantitative design and analysis
  - Introduction
  - Performance Metrics for Computer Architects
  - Basic ISA, Pipelining, RISC Architecture (review)
- Memory hierarchy design
- Instruction-level parallelism (ILP) and its exploitation
- Data-level parallelism
- Thread-level parallelism
- Warehouse scale computers
- Hardware Security

What will we learn in this course??

- How the programs written in high level language are translated into language of hardware
- Interface between software and hardware?
- What determines performance of a program and how can it be improved?
- What techniques are used by hardware designer to improve performance?

From high level language to hardware language

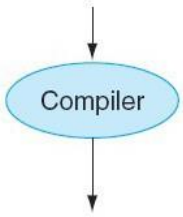
- At application-level: Applications are written using millions of lines of complex instructions
- Hardware can execute only simple instructions
- Several layers are needed to go from complex instruction to simple instruction



From high level language to hardware language

High-level language program (in C)

```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```



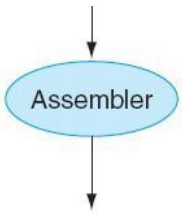
Assembly language program (for MIPS)

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```


From high level language to hardware language

Assembly language program (for MIPS)

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```



Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

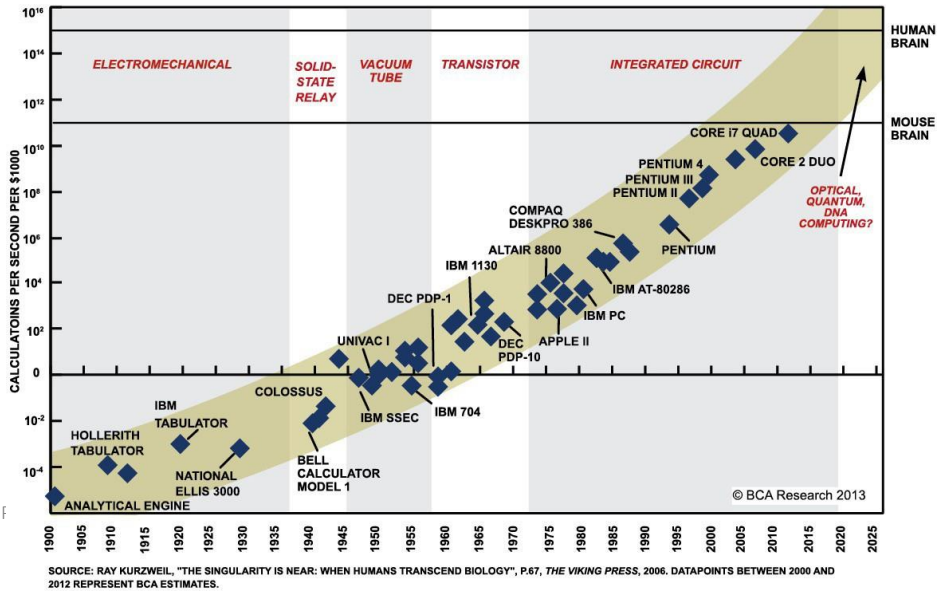
Background/History of Computing

Fundamentals of Quantitative Design and Analysis

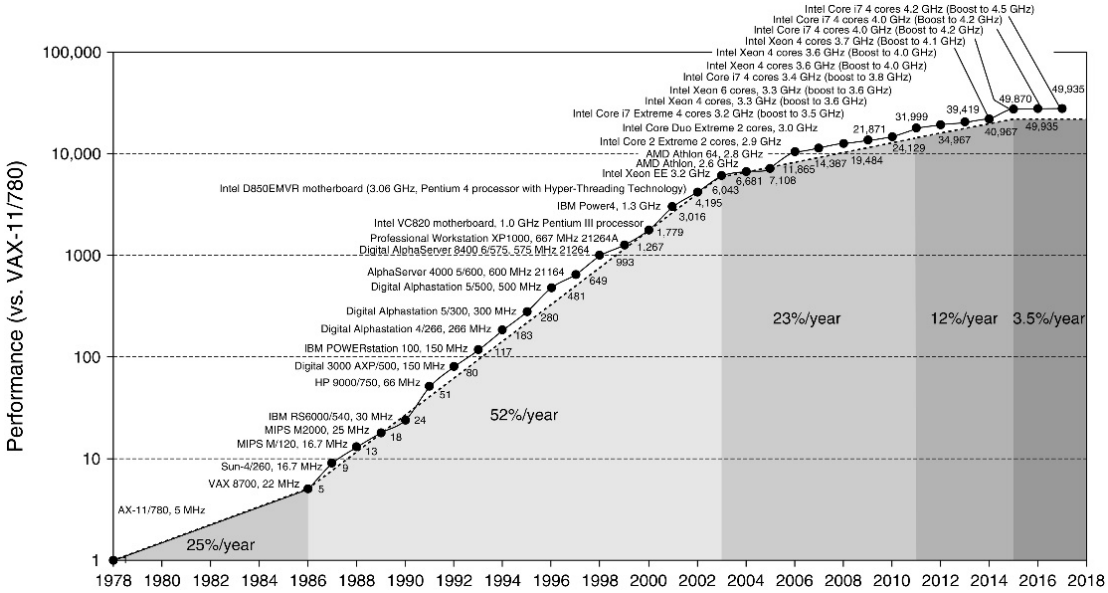
- Introduction – Moore's Law
  - Moore's law: the number of transistors on a chip double every 2 years (or so)
  - More Precisely:
    - Feature sizes shrink by 0.7x
    - Number of transistors per die increases by 2x
    - Speed of transistors increases by 1.4x

- Introduction – Moore's Law

- Moore's law: the number of transistors on a chip double every 2 years (or so)



Fundamentals of Quantitative Design and Analysis



- Prior to the mid-1980s, growth in processor performance was largely technology-driven and averaged about 22% per year, or doubling performance every 3.5 years.
- The increase in growth to about 52% starting in 1986, or doubling every 2 years, is attributable to more advanced architectural and organizational ideas typified in RISC architectures.
- In 2003 the limits of power due to the end of Dennard scaling and the available instruction-level parallelism slowed uniprocessor performance to 23% per year until 2011, or doubling every 3.5 years.
- From 2011 to 2015, the annual improvement was less than 12%, or doubling every 8 years in part due to the limits of parallelism of Amdahl's Law.
- Since 2015, with the end of Moore's Law, improvement has been just 3.5% per year, or doubling every 20 years!

Fundamentals of Quantitative Design and Analysis

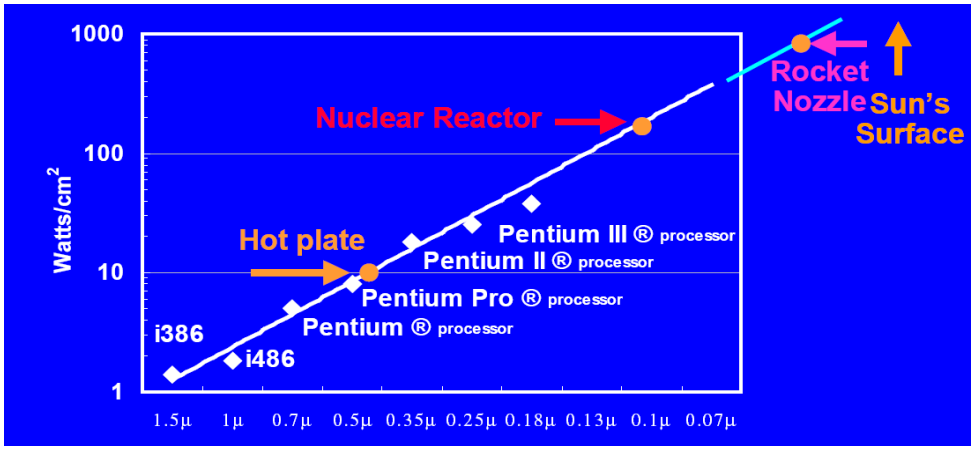
- An unprecedented sustained growth of over 50% per year for a period of straight 17 years (hardware renaissance era).

○ Effects

- This growth has enhanced the capability of common users
  - High performance personal computers of today outperform supercomputers of ten years before
- New classes of computers are introduced
  - More powerful PCs
  - Rise of smart phones (e.g., galaxy S-IV has a quad core processor)
  - Tablet computers, notebooks
  - Warehouse scale computers containing tens of thousands of servers

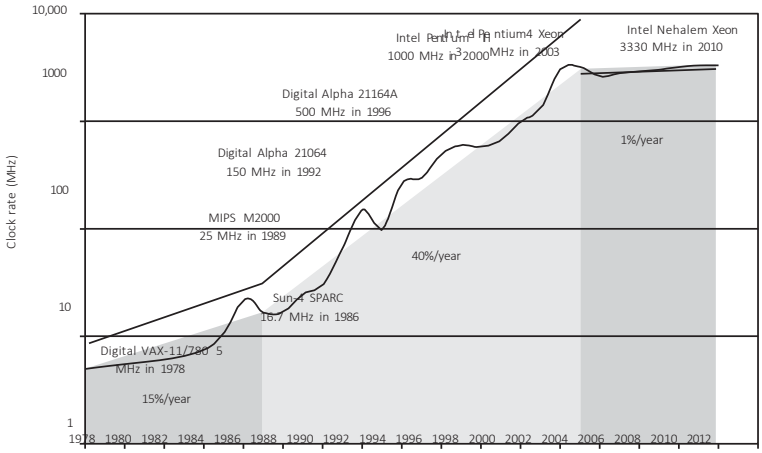
Fundamentals of Quantitative Design and Analysis

- Since 2003, single-processor performance improvement has dropped to 22% per year
  - Power dissipation
  - Lack of (more) extractable parallelism



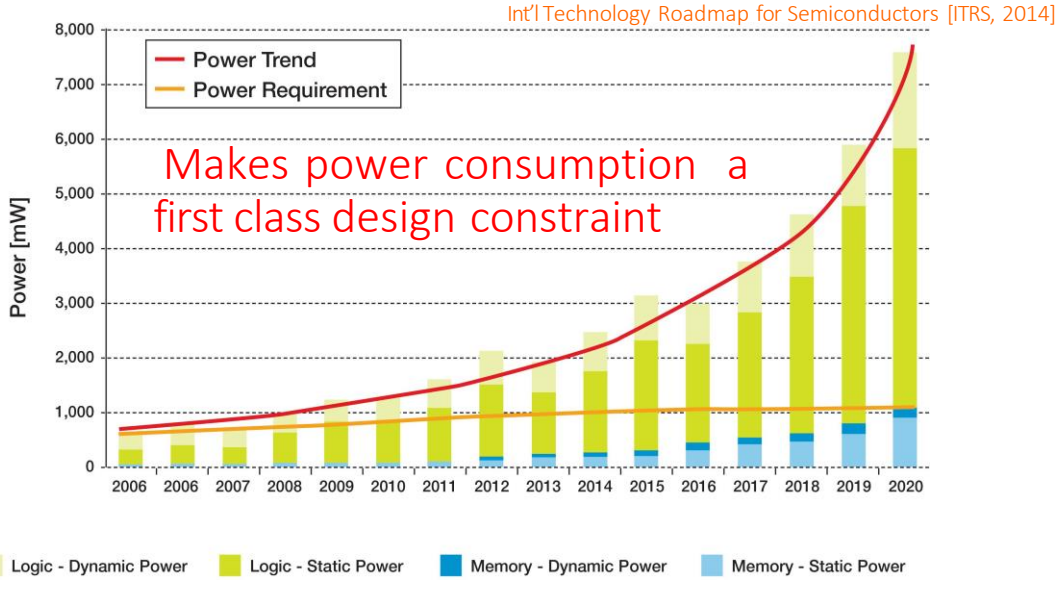
- Introduction – Evolution of Clock Speed

- First 32-bitmicroprocessor required only 2 watts
- Intel core-i7 requires 130 watts
- More clock rate,higher power and vice versa



Moore's Law Goes Multicore –Power & Energy

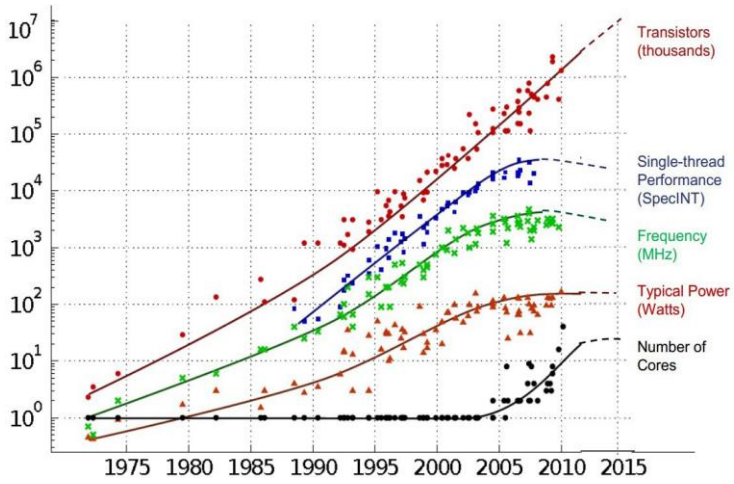
Int'l Technology Roadmap for Semiconductors [ITRS, 2014]



Fundamentals of Quantitative Design and Analysis

Moore's Law Goes Multicore –Parallelism

Parallelize more to take advantage of multi/many core architecture!

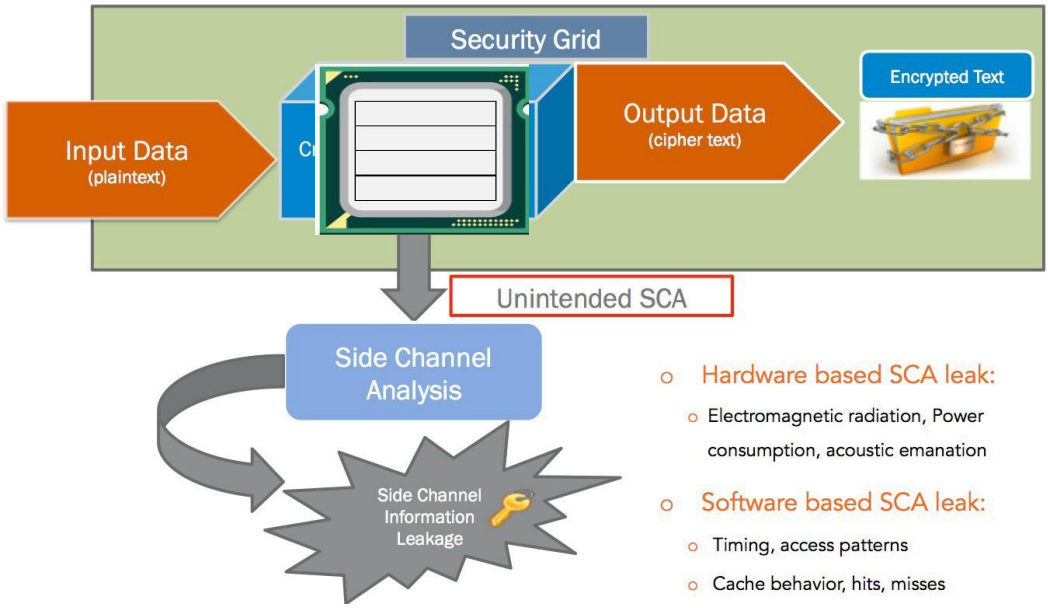


Int'l Technology Roadmap for Semiconductors [ITRS, 2014]

- Multicore scaling only works if applications are perfectly scalable
- They are not!
- Lack of extractable parallelism in applications is the dominant factor of dark silicon!

Fundamentals of Quantitative Design and Analysis

Moore's Law Goes Multicore –Security



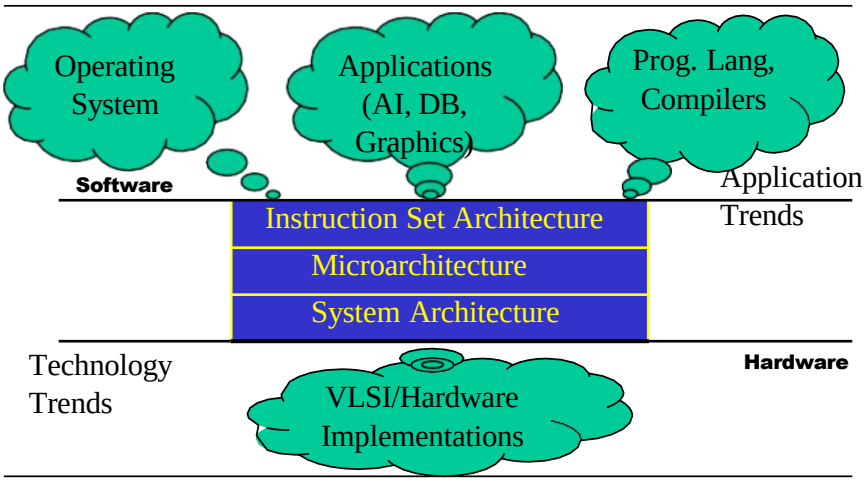
- Hardware based SCA leak:
  - Electromagnetic radiation, Power consumption, acoustic emanation
- Software based SCA leak:
  - Timing, access patterns
  - Cache behavior, hits, misses

Fundamentals of Quantitative Design and Analysis

- Introduction
  - Classification of computing systems
  - Defining computer architecture
    - oFlynn's Taxonomy
  - Trends in technology
  - Trends in power and energy in integrated circuits
  - Trends in cost
  - Dependability
  - Measuring, reporting, and summarizing performance
  - Quantitative principles of computer design

Fundamentals of Quantitative Design and Analysis

– Introduction



Classes of computers

- Personal mobile devices (PMDs)
- Desktop computers
- Servers
- Cluster/warehouse scale computers
- Embedded computers

Classes of computers

oPersonal Mobile Devices (PMDs)

- Wireless devices with multimedia interface such as smart phones and tablet computers
- Constraints
  - Cost
  - Energy efficiency
  - Size requirement
  - Responsiveness & Predictability

oDesktop computing

- Low end note books to high end workstations, battery operated lap tops etc..
- Constraints
  - Cost-performance trade-off mainly

Fundamentals of Quantitative Design and Analysis

- Servers
  - Large scale and more reliable file computing services
  - Key features: Availability, Scalability, Throughput e.g.
  - transactions per minute or results per minute, capability of handling number of requests per unit time

Application	Cost of downtime per hour	Annual losses with downtime of		
		1% (87.6 h/year)	0.5% (43.8 h/year)	0.1% (8.8 h/year)
Brokerage service	$4,000,000	$350,400,000	$175,200,000	$35,000,000
Energy	$1,750,000	$153,300,000	$76,700,000	$15,300,000
Telecom	$1,250,000	$109,500,000	$54,800,000	$11,000,000
Manufacturing	$1,000,000	$87,600,000	$43,800,000	$8,800,000
Retail	$650,000	$56,900,000	$28,500,000	$5,700,000
Health care	$400,000	$35,000,000	$17,500,000	$3,500,000
Media	$50,000	$4,400,000	$2,200,000	$400,000

Fundamentals of Quantitative Design and Analysis

Classes of computers

- Clusters/Warehouse Scale Computers
  - Software as a Service (SaaS) like social networking, search, multiplayer games, video sharing… has led to *clusters.*
  - Largest clusters are termed as warehouse scale computers
  - Critical factors
    - Price-performance
    - Power
    - Availability just like servers
    - Scalability through LANs

Classes of computers

- Embedded computers
  - Found in everyday machines; microwave ovens, washing machines, printers, ….
  - Also found in NOT so common machines; Aerospace applications, Nuclear, power plants,…
  - Constraints
    - Cost
    - Application-Specific Performance
    - Energy Efficiency
    - Reliability in remote operating conditions
    - Temporal Correctness (sometimes)

Classes of computers

oSummary

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Internet of things/ embedded
Price of system	$100–$1000	$300–$2500	$5000–$10,000,000	$100,000–$200,000,000	$10–$100,000
Price of microprocessor	$10–$100	$50–$500	$200–$2000	$50–$250	$0.01–$100
Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

Fundamentals of Quantitative Design and Analysis

- Introduction
  - Classes of computers
  - Defining computer architecture
    - Flynn's Taxonomy
  - Trends in technology
  - Trends in power and energy in integrated circuits
  - Trends in cost
  - Dependability
  - Measuring, reporting, and summarizing performance
  - Quantitative principles of computer design

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- Problem Statement: Determine what attributes are important for a new computer, then design a computer to maximize performance and energy efficiency while staying within cost, power, and availability constraints
- How to Design: Instruction set design, functional organization, logic design, and implementation
- How to Implement: Implementation encompasses IC design, packaging, power, cooling, and familiarity with a very wide range of technologies from compilers and operating systems to logic design and packaging.

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- Classes of Application Parallelism
  - Parallelism at multiple levels is the driving force behind computer design
  - Two kinds of parallelism in applications exist

- Data-Level Parallelism (DLP)
  - Arises because there are many data items that can be operated on at the same time

- Task-Level Parallelism (TLP)
  - Arises because tasks of work are created that can operate independently and largely in parallel

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- Architectural Parallelism
  - Computer hardware can exploit this application parallelism in FOUR different ways:

① Instruction-Level Parallelism
- Uses pipelining and speculative execution

② Vector Architectures and Graphic Processor Units (GPUs)
- Exploits data-level parallelism in applications
- Applies a single instruction to a collection of data in parallel

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- **Architectural Parallelism**
  - Computer hardware can exploit this application parallelism in FOUR different ways:

③ Thread-Level Parallelism
- Exploits either data-level or task-level parallelism in tightly coupled hardware that allows interaction between various threads

④ Request-Level Parallelism
- Exploits parallelism among largely decoupled tasks specified by the programmer or the operating system

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- **Flynn's Taxonomy**
  - Micheal Flynn in 1966 proposed a classification for exploitation techniques of Parallelism by the hardware.
  - He classified all computers into FOUR classes based on the instruction and data streams being used by them

① Single instruction stream, single data stream (SISD)
② Single instruction stream, multiple data streams (SIMD)
③ Multiple instruction streams, single data stream (MISD)
④ Multiple instruction streams, multiple data streams (MIMD)

Defining Computer Architecture

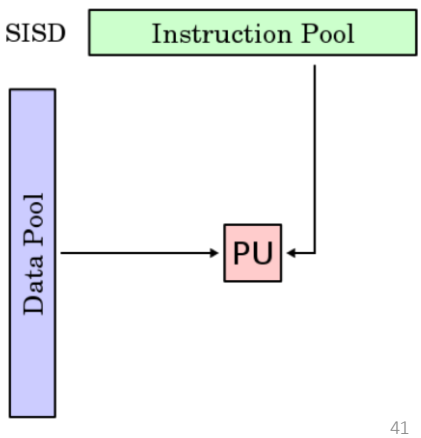
- Flynn's Taxonomy

1. Single instruction stream, single data stream (SISD)
   - Uniprocessor category
   - From programmer's perspective, it is the standard sequential computer
   - Can exploit instruction-level parallelism
   - Use ILP techniques such as superscalar and speculative execution

Defining Computer Architecture

oFlynn's Taxonomy

1. Single instruction stream, single data stream (SISD)



Defining Computer Architecture

- Flynn's Taxonomy

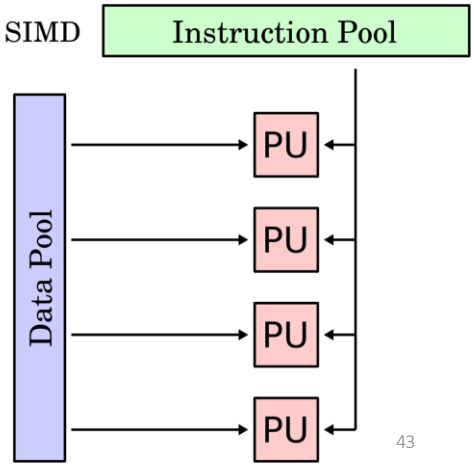
② Single instruction stream, multiple data streams (SIMD)

- Same instruction is executed by multiple processors
- Used data streams are different
- Exploits data-level parallelism by applying the same operations to multiple items of data in parallel
- Each processor has its own data memory (hence the MD of SIMD)
- Single instruction memory and control processor, which fetches and dispatches instructions.
- Vector architectures, multimedia extensions to standard instruction sets, and GPUs use Data-level parallelism like SIMD

Defining Computer Architecture

oFlynn's Taxonomy

② Single instruction stream, multiple data streams (SIMD)



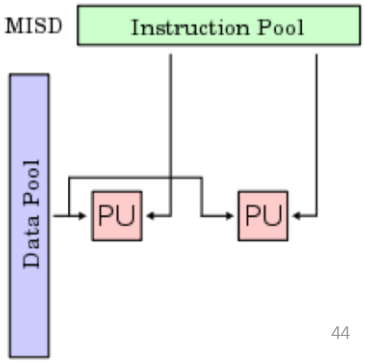
Defining Computer Architecture

oFlynn's Taxonomy

③ Multiple instruction streams, single data stream (MISD)

- No commercial multiprocessor of this type has been built to date, but it rounds out this simple classification

C= a+b C= A*b



Defining Computer Architecture

o Flynn's Taxonomy

④ Multiple instruction streams, multiple data streams (MIMD)

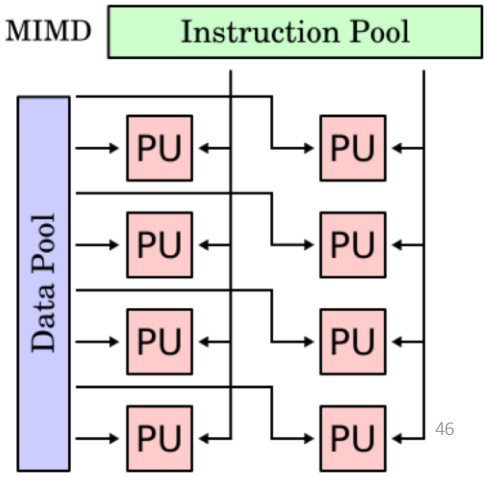
- Each processor fetches its own instructions and operates on its own data
- Targets task-level parallelism
- More flexible than SIMD and more generally applicable
- Inherently more expensive than SIMD
- Can also exploit data-level parallelism, although the overhead is likely to be higher
- Clusters and warehouse-scale computers that exploit request-level parallelism, where many independent tasks can proceed in parallel naturally with little need for communication or synchronization.

Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- Flynn's Taxonomy

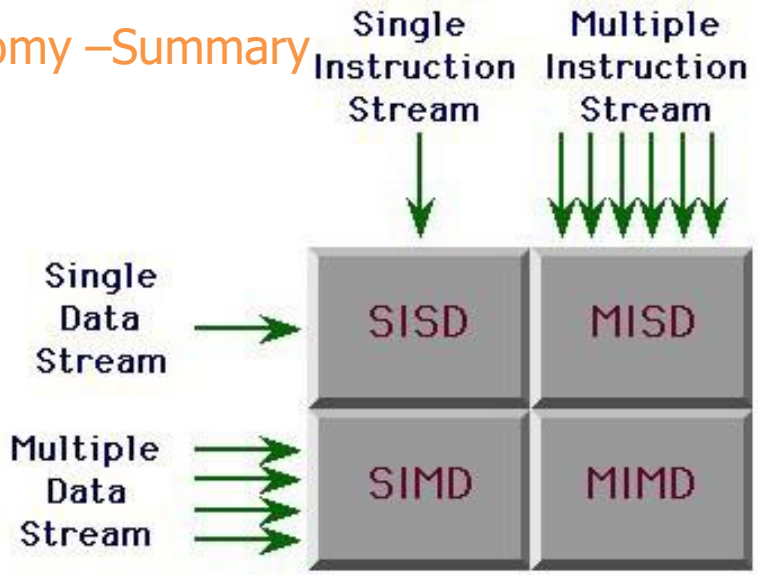
④ Multiple instruction streams, multiple data streams (MIMD)



Fundamentals of Quantitative Design and Analysis

Defining Computer Architecture

- Flynn's Taxonomy –Summary



Fundamentals of Quantitative Design and Analysis

- Introduction
  - Classes of computers
  - Defining computer architecture
    - Flynn's Taxonomy
  - Trends in technology
  - Trends in power and energy in integrated circuits
  - Trends in cost
  - Dependability
  - Measuring, reporting, and summarizing performance
  - Quantitative principles of computer design

Fundamentals of Quantitative Design and Analysis

Trends in Technology

- Computer Architecture must be designed to survive rapid changes in computer technology

- Five technologies are critical to modern implementations
  1. Integrated circuit logic technology
  2. Semiconductor DRAM (dynamic random-access memory)
  3. Semiconductor Flash (electrically erasable programmable read- only memory)
  4. Magnetic disk technology
  5. Network technology

Trends in Performance

- Bandwidth (or throughput) Vs Latency
  - Bandwidth or throughput
    - The total amount of work done in a given time, such as megabytes per second for a disk transfer.
  - Latency or response time
    - The time between the start and the completion of an event, such as milliseconds for a disk access

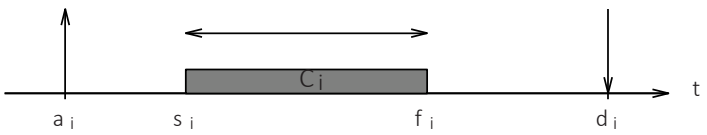
Trends in Performance

- Bandwidth (or throughput) Vs Latency

Performance Trends: Bandwidth over Latency

pp. 18, Edition 5, H&P

As we shall see in Section 1.8, *bandwidth* or *throughput* is the total amount of work done in a given time, such as megabytes per second for a disk transfer. In contrast, *latency* or *response time* is the time between the start and the completion of an event, such as milliseconds for a disk access. Figure 1.9 plots the relative



pp. 27, G.C. BUTTAZZO

Response time R_i is the difference between the finishing time and the request time: $R_i = f_i - r_i$;

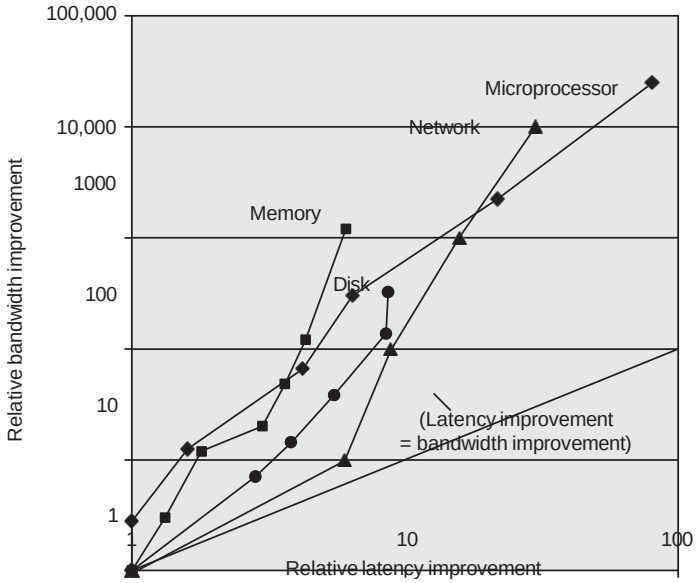
Fundamentals of Quantitative Design and Analysis

Trends in Performance

- Bandwidth (or throughput) Vs Latency

Latency improved 6X to 80X
Bandwidth improved about 300X to 25,000X

Courtesy: Patterson [2004]



Fundamentals of Quantitative Design and Analysis

- Introduction
  - Classes of computers
  - Defining computer architecture
    - Flynn's Taxonomy
  - Trends in technology
  - Trends in power and energy in integrated circuits
  - Trends in cost
  - Dependability
  - Measuring, reporting, and summarizing performance
  - Quantitative principles of computer design

Trends in Power & Energy

- System level perspective
  - From the viewpoint of a system designer, there are three primary concerns.
    - Maximum power a processor ever requires?
    - Sustained power consumption?
    - Energy efficiency of overall system?

Trends in Power & Energy

- System level perspective
  - Maximum power a processor ever requires
    - Meeting this demand can be important to ensuring correct operation
    - Excessive power requirement results in voltage drop, thus leading to device malfunction
  - Sustained power consumption
    - Called the *Thermal Design Power (TDP)* , determines the cooling requirements of a system
    - TDP is neither peak power nor the actual average power that will be consumed during a given computation
    - A typical power supply for a system is usually sized to exceed the TDP, and a cooling system is usually designed to match or exceed TDP

Trends in Power & Energy

- System level perspective
  - Energy Efficiency of the System
    - Power is Energy per unit time OR energy is the cumulative power consumed over a certain period of time
    - Which Metric is RIGHT to measure system's efficiency and why? POWER or ENERGY?

    - Example:

Processor A has 25% high power consumption than Processor B

Processor A take 30% less time to execute a task than Processor B

Energy Factor for A: 1.25 x 0.70 = 0.875 compared to B

POWER is rather a constraint than a design criteria!

Trends in Power & Energy

- Microprocessor level perspective
  - Energy & Power in a CMOS Processor
    - Traditional primary energy consumption has been in switching transistors, also called dynamic energy

$$\text{Energy}_{\text{dynamic}} \propto \text{Capacitive load} \times \text{Voltage}^2$$

    - Equation is for complete pulse of transition (1-0-1), for single transition

$$\text{Energy}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2$$

    - The power required per transistor is just the product of the energy of a transition multiplied by the frequency of transitions

$$\text{Power}_{\text{dynamic}} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

Slowing clock rate reduces power, but not energy. Why?

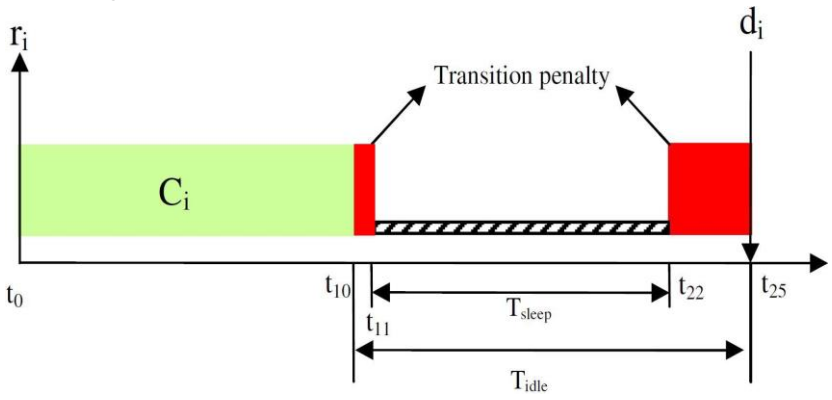
Trends in Power & Energy

- Microprocessor level perspective
  - Energy & Power –Software Techniques
    - Do nothing well: Turn off the clock of inactive modules
    - Dynamic Voltage-Frequency Scaling (DVFS): Devices have intervals of low activity where there is no need to operate at the highest clock frequency and voltages, scale them to lower operating points
    - Design for typical case: Based on the target application, design low power operating modes to be used at run-time
    - Overclocking: The chip decides that it is safe to run at a higher clock rate for a short time possibly on just a few cores until temperature starts to rise. The Turbo mode!

Trends in Power & Energy

- Microprocessor level perspective
  - Energy & Power – Software Techniques

Dynamic Power Management

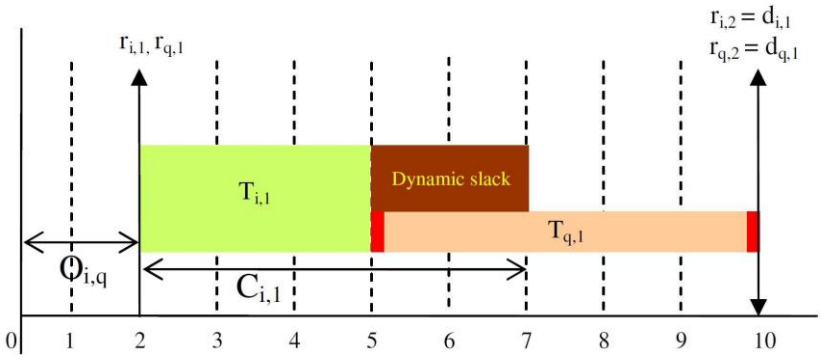


Trends in Power & Energy

oMicroprocessor level perspective

oEnergy & Power – Software Techniques

Dynamic Voltage and Frequency Scaling

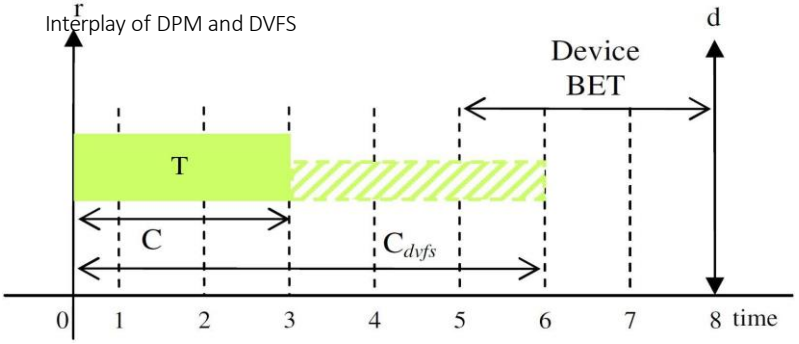


Trends in Power & Energy

oMicroprocessor level perspective

oEnergy & Power – Software Techniques

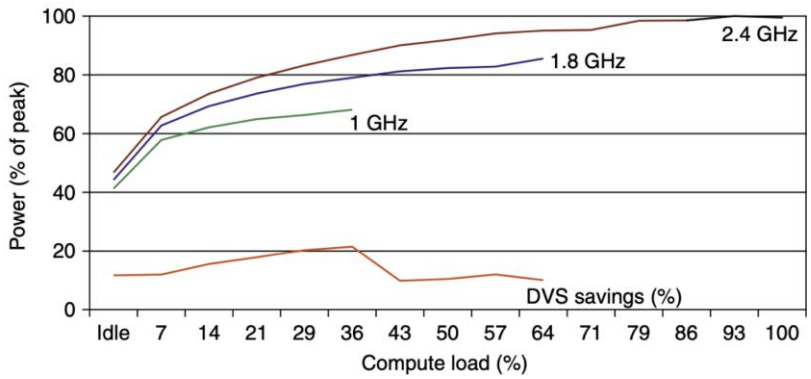
Interplay of DPM and DVFS



Fundamentals of Quantitative Design and Analysis

Trends in Power & Energy

- Microprocessor level perspective
  - Energy & Power –Workload vs Peak Power



Fundamentals of Quantitative Design and Analysis

Trends in Power & Energy

- Microprocessor level perspective
  - Energy & Power –Static Power
    - Static power is becoming an important issue because leakage current flows even when a transistor is off

$$\text{Power}_{\text{static}} \propto \text{Current}_{\text{static}} \times \text{Voltage}$$

    - Static power is proportional to number of devices
    - Increasing the number of transistors increases power even if they're idle
    - Leakage current increases in processors with smaller transistor sizes
    - Very low power systems are even turning off the power supply (power gating) to inactive modules to control loss due to leakage

Fundamentals of Quantitative Design and Analysis

- Introduction
  - Classes of computers
  - Defining computer architecture
    - Flynn's Taxonomy
  - Trends in technology
  - Trends in power and energy in integrated circuits
  - Dependability
  - Measuring, reporting, and summarizing performance
  - Quantitative principles of computer design

Fundamentals of Quantitative Design and Analysis

Dependability

- Rule: Design with NO SINGLE POINT FALURE
- Transient & permanent faults are becoming common
- Two measures of dependability
  - Module Reliability: A measure of continuous service accomplishment from a reference initial time
    - Measure in Mean Time To Failure (MTTF)
    - Reciprocal of MTTF is Failure In Time (FIT) or failure rate (measured: failures per billion hours)
    - Service interruption is measured in Mean Time To Repair (MTTR)
    - Mean Time Between Failures (MTBF) = MTTR + MTTF
  - Module Availability: A measure of the service accomplishment with respect to the alternation between the accomplishment and interruption

$$\text{Module availability} = \frac{\text{MTTF}}{(\text{MTTF} + \text{MTTR})}$$

Dependability

- Example: Calculate system's MTTF with a disk subsystem having following components and their respective MTTF:
  - 10 disks, each rated at 1,000,000-hour MTTF
  - 1 ATA controller, 500,000-hour MTTF
  - 1 power supply, 200,000-hour MTTF
  - 1 fan, 200,000-hour MTTF
  - 1 ATA cable, 1,000,000-hour MTTF
- Reciprocal of MTTF is Failure Rate
  - The sum of the failure rates is:

$$\text{Failure rate}_{\text{system}} = 10 \times \frac{1}{1{,}000{,}000} + \frac{1}{500{,}000} + \frac{1}{200{,}000} + \frac{1}{200{,}000} + \frac{1}{1{,}000{,}000}$$

$$= \frac{10+2+5+5+1}{1{,}000{,}000 \text{ hours}} = \frac{23}{1{,}000{,}000} = \frac{23{,}000}{1{,}000{,}000{,}000 \text{ hours}}$$

- The MTTF for the system is just the inverse of the failure rate:

$$\text{MTTF}_{\text{system}} = \frac{1}{\text{Failure rate}_{\text{system}}} = 43{,}500 \text{ hours}$$

- Introduction
  - Classes of computers
  - Defining computer architecture
    - Flynn's Taxonomy
  - Trends in technology
  - Trends in power and energy in integrated circuits
  - Trends in cost
  - Dependability
  - Measuring, reporting, and summarizing performance
  - Quantitative principles of computer design

Fundamentals of Quantitative Design and Analysis

- Introduction
  - Classes of computers
  - Defining computer architecture
    - oFlynn's Taxonomy
  - Trends in technology
  - Trends in power and energy in integrated circuits
  - Trends in cost
  - Dependability
  - Measuring, reporting, and summarizing performance
  - Quantitative principles of computer design

Fundamentals of Quantitative Design and Analysis

Quantitative Principles of Computer Design

- Take Advantage of Parallelism
  - At server level parallelism can be achieved through multiprocessors and multiple disks
  - At single processor level, its best example is pipelining
  - At memory level, set-associative cache is an example where multiple banks are searched in parallel
- Principle of Locality
  - 90-10 rule of thumb: Programs spend 90% of time in 10% of code
  - Temporal locality: Recently accessed data is likely to be accessed again in future
  - Spatial Locality: Items whose addresses are near one another tend to be referenced close together in time

Quantitative Principles of Computer Design

- Focus on the Common Case
  - In making a design trade-off, favor the frequent case over infrequent case
  - The impact of the improvement is higher if the occurrence is frequent
    - Example: The instruction fetch and decode unit of a processor may be used much more frequently than a multiplier
    - Optimize it first
  - Amdahl's Law quantifies common case performance improvement

Quantitative Principles of Computer Design

- Amdahl's Law –make the common case faster!
  - States that the *performance improvement* to be gained from using some *faster mode of execution* is limited by the fraction of time the faster mode can be used!
  - Speedup according to Amdahl's law: Speedup tells us how much faster a task will run using the computer with the *enhancement* as opposed to the original computer

$$\text{Speedup} = \frac{\text{Performance for entire task using the enhancement when possible}}{\text{Performance for entire task without using the enhancement}}$$

$$\text{Speedup} = \frac{\text{Execution time for entire task without using the enhancement}}{\text{Execution time for entire task using the enhancement when possible}}$$

Quantitative Principles of Computer Design

- Amdahl's Law
  - Two quick ways to find out speedups
    - ① The fraction of the computation time in the original computer that can be converted to take advantage of the enhancement
      - oExample:
        - For a program that takes 60 sec. in total, if 20 sec of the execution time can use an *enhancement*, the fraction is 20/60
        - Called $Fraction_{enhanced}$
        - Always less than or equal to 1

Quantitative Principles of Computer Design

- Amdahl's Law
  - TWO quick ways to find out speedups
    - ② The improvement gained by the enhanced execution mode, i.e.,how much faster the task would run if the enhanced mode were used for the entire program –the time of the original mode over the time of the enhanced mode
      - Example:
        - For a program that originally takes 10 sec and after using enhancement takes 2 sec, the improvement is 10/2
        - Called $Speedup_{enhanced}$
        - Always greater than 1

Quantitative Principles of Computer Design

- Amdahl's Law
  - Effect of *Enhancement* on Overall Speedup
    - Execution time using the original computer with the enhanced mode

$$\text{Execution time}_{\text{new}} = \text{Execution time}_{\text{old}} \times \left((1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)$$

    - Overall effect

$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Quantitative Principles of Computer Design

- Amdahl's Law -Examples
  - Example 01:
    - We want to enhance the processor used for Web serving. The new processor is 10 times faster on computation in the Web serving application than the original processor. Assuming that the original processor is busy with computation 40% of the time and is waiting for I/Os for 60% of the time, what is the overall speedup gained by incorporating the enhancement?
  - $\text{Fraction}_{\text{enhanced}}$?
  - $\text{Speedup}_{\text{enhanced}}$?

$$\text{Fraction}_{\text{enhanced}} = 0.4;\ \text{Speedup}_{\text{enhanced}} = 10;\ \text{Speedup}_{\text{overall}} = \frac{1}{0.6 + \frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

Quantitative Principles of Computer Design

Amdahl's Law -Examples

$$\frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Example 02:

- Suppose FP square root (FPSQR) is responsible for 20% of the execution time of a critical graphics benchmark. One proposal is to enhance the FPSQR hardware and speed up this operation by a factor of 10. The other alternative is just to try to make all FP instructions in the graphics processor run faster by a factor of 1.6.
- FP instructions are responsible for half of the execution time for the application. The design team believes that they can make all FP instructions run 1.6 times faster with the same effort as required for the fast square root. Which one is better solution?

$$\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1 - 0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1 - 0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

Quantitative Principles of Computer Design

The Processor Performance Equation

- Define: ticks, clock ticks, clock periods, clocks, cycles, clock cycles
- CPU time

$$\text{CPU time} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

- CPI (Cycles per Instruction)

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} = \frac{\text{Seconds}}{\text{Program}} = \text{CPU time}$$