CS4100: Computer System Design

Fundamentals of Quantitative Design and Analysis

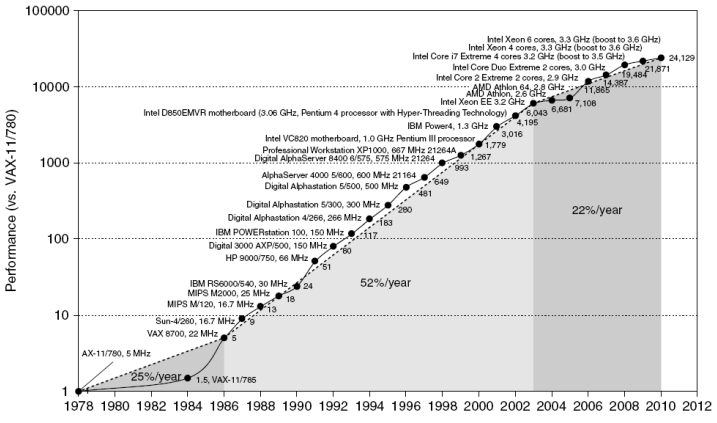
Madhu Mutyam
PACE Laboratory
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Aug 6-10, 2015

Layered Computer Design

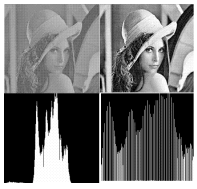
	Application Layer
Algorithm Designer	⇓
	Algorithm Layer
Programmer	⇓
	High-Level Language Layer
System Personnel	⇓
	Operating System Layer
Computer Architect	⇓
	Architecture Layer
Computer Designer	⇓
	Microarchitecture Layer
Logic Designer	⇓
	Logic Layer
Chip Designer	⇓
	Transistor Layer

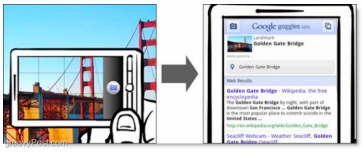
Processor Performance

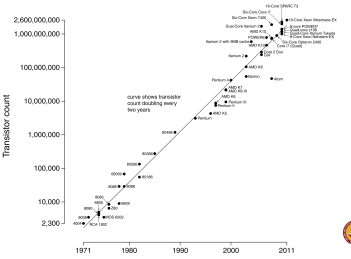


- Advances in IC technology
- Advances in computer architecture

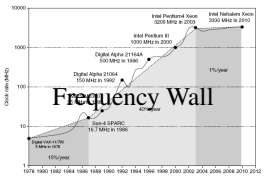
Applications Demanding High Performance

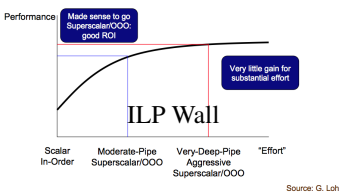


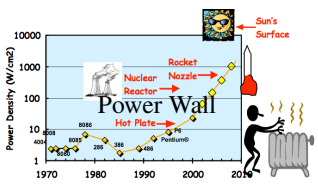


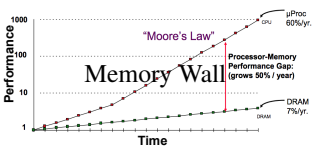


Roadblocks for Single-Core Performance



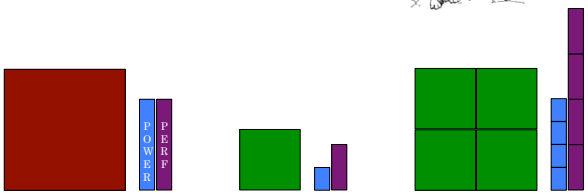






Break the Wall

Paradigm Shift

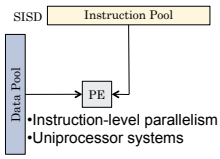


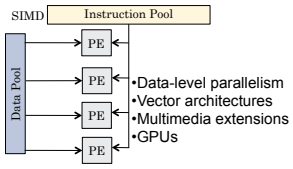
- Multiple slower/simpler cores instead of a faster/powerful processor

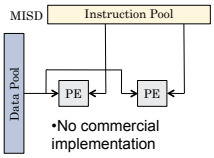
Classes of Parallelism and Parallel Architectures

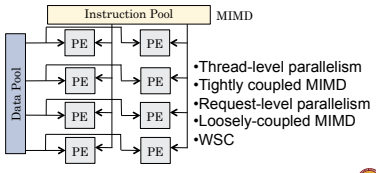
- Parallelism in applications
  - Data-level parallelism
  - Task-level parallelism
- Architectural parallelism
  - Instruction-level parallelism (ILP)
  - Vector architectures and Graphics processor units (GPUs)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)

Flynn's Taxonomy









Aspects of Computer Design

- Instruction set design
- Implementation – micro-architecture and hardware design
- Optimising the design
  - Personal mobile devices – *Energy efficiency, responsiveness*
  - Desktop computing – *Price-performance*
  - Servers – *Availability, scalability, throughput*
  - Warehouse-scale computers – *Price-performance, power, availability*
  - Embedded computers – *Price*

Course Objectives

- Understanding the fundamentals of computer architecture
- Understanding the design of superscalar processors
- Providing an overview on multi-core processors
- Understanding the internals of DRAM-based memory controllers

Syllabus

- Fundamentals of Quantitative Design and Analysis (1 week)
- Instruction Set Principles (1 week)
- Memory Hierarchy Design (3 weeks)
- Fundamentals of Pipelining (1 week)
- Exploiting Instruction-Level Parallelism (4 weeks)
- Exploiting Thread-Level Parallelism (3 weeks)
- Computer Arithmetic (1 week)

Resources and Evaluation

- Reference Books
  - J.L. Hennessy and D.A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann. 5th Edition, 2012.
  - J.P. Shen and M.H. Lipasti. Modern Processor Design. Fundamentals of Superscalar Processors. McGraw-Hill International, 2005.
  - B. Jacob, N.G. Spencer, and D. Wang. Memory Systems: Cache, DRAM, and Disk. Morgan Kaufmann, 2006.
  - C. Hahacher, Z. Vranesic, and S. Zaky. Computer Organization. McGraw Hill, 2002.
- Evaluation Mechanism
  - 2 Quizzes (40%)
  - Surprise Tests (10%)
  - End Semester Examination (50%)
- **Institute attendance policy is strictly followed**

CS4110: Computer System Design Lab

- 2-member groups are allowed
- Take-home programming assignments
- Assignment evaluations are scheduled in P slot
- Evaluation:
  - Programming assignments (85%)
  - Viva-voce (15%)
- **Any form of unfair means is treated as a serious academic offence**

Principles of Computer Design

- Computer Architecture = ISA + Microarchitecture + Implementation
- A single ISA can be realized using multiple microarchitectures
  - Both Intel Xeon and AMD Opteron processors use 64-bit x86 ISA
- Same microarchitecture can be implemented differently
  - Xeon 7560 and Core i7 processors from Intel have almost the same microarchitecture but provide different clock rates and memory systems
- Take advantage of parallelism
  - Bit-level
  - Instruction-level
  - Data-level
  - Thread-level
- Principle of locality
  - A program spends 90% of its execution time in only 10% of the code
  - Spatial and temporal locality
- Focus on the common case

Amdahl's Law

$$Speedup = \frac{Performance_{entire\ task\ using\ the\ enhancement\ when\ possible}}{Performance_{entire\ task\ without\ using\ the\ enhancement}} = \frac{ExecutionTime_{entire\ task\ without\ using\ the\ enhancement}}{ExecutionTime_{entire\ task\ using\ the\ enhancement\ when\ possible}}$$

- *Performance improvement to be gained from using some enhancement* is limited by *the fraction of the time the enhancement can be used*

$$ExecTime_{new} = ExecTime_{old} \times [(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}$$

$$Speedup_{overall} = \frac{ExecTime_{old}}{ExecTime_{new}} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Example #1

- Consider two processors A_{old} and A_{new}. A_{old} spends 30% time in computation and 70% time waiting for I/O. Some enhancements are incorporated in A_{new} so that it achieves 15× improvement in the computation time. What is the overall speedup gained by incorporating the enhancement?
  According to Amdahl's law:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$Fraction_{enhanced} = 0.3$$

$$Speedup_{enhanced} = 15$$

$$Speedup_{overall} = \frac{1}{(1 - 0.3) + \frac{0.3}{15}} = 1.389$$

Example #2

- Consider an application where floating-point (FP) instructions are responsible for 50% of the execution time for the application while FP square root (FPSQR) is responsible for 20% of the execution time. Compare the following design alternatives, assuming that both alternatives require the same effort.
  - Design #1: Make all FP instructions in the processor run faster by 1.6×
  - Design #2: Speedup the FPSQR operation by 10×

  According to Amdahl's law:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{enhanced}) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$Speedup_{overall}^{Design\ \#1} = \frac{1}{(1 - 0.5) + \frac{0.5}{1.6}} = \frac{1}{0.8125} = 1.23$$

$$Speedup_{overall}^{Design\ \#2} = \frac{1}{(1 - 0.2) + \frac{0.2}{10}} = \frac{1}{0.82} = 1.22$$

Design #1 is better than Design #2

Example #3

- When parallelizing an application, the ideal speedup achieved is equal to the number of processors. What is the speedup with 100 processors if 80% of the application is parallelizable, ignoring the cost of communication?
  According to Amdahl's law:

$$Speedup_{overall} = \frac{1}{(1 - Fraction_{parallel}) + \frac{Fraction_{parallel}}{Speedup_{100\_processors}}}$$

$$Fraction_{parallel} = \frac{80}{100} = 0.8$$

$$Speedup_{100\_processors} = \frac{ExecTime_{1\_processor}}{ExecTime_{100\_processors}} = \frac{T(1)}{(\frac{T(1)}{100})} = 100$$

$$Speedup_{overall} = \frac{1}{(1 - 0.8) + \frac{0.8}{100}} = \frac{1}{0.208} = 4.808$$

The Processor Performance Equation

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

$$\text{Cycles per instruction (CPI)} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count (IC)}}$$

$$\text{CPU time} = \text{IC} \times \text{CPI} \times \text{Clock cycle time}$$

- To optimize CPU time:
  - *IC* – ISA and compiler technology
  - *CPI* – ISA and microarchitecture
  - *Clock cycle time* – Hardware technology and microarchitecture

The Processor Performance Equation (Contd)

$$\begin{aligned}\text{CPU time} &= \text{CPU clock cycles for a program} \times \text{Clock cycle time} \\ &= (\Sigma_{i=1}^{n} IC_i \times CPI_i) \times \text{Clock cycle time}\end{aligned}$$

$$\begin{aligned}CPI &= \frac{\Sigma_{i=1}^{n} IC_i \times CPI_i}{\text{IC}} \\ &= \Sigma_{i=1}^{n} \frac{IC_i}{\text{IC}} \times CPI_i\end{aligned}$$

Example #4

Consider the following measurements of a program:

Frequency of FP operations	=	25%
Average CPI of FP operations	=	4
Average CPI of other operations	=	1.5
Frequency of FPSQR	=	2%
CPI of FPSQR	=	20

Which one of the following design alternatives is better?

- Design #1: Decrease the CPI of FPSQR to 2
- Design #2: Decrease the average CPI of all FP operations to 2.5

Example #4 (Contd)

$$\begin{aligned}CPI_{\text{original}} &= \Sigma_{i=1}^{n} \frac{IC_i}{\text{IC}} \times CPI_i \\ &= (25\% \times 4) + (75\% \times 1.5) = 2.125\end{aligned}$$

$$\begin{aligned}CPI_{\text{Design \#1}} &= CPI_{\text{original}} - 2\% \times (CPI_{FPSQR_{old}} - CPI_{FPSQR_{new}}) \\ &= 2.125 - 2\% \times (20 - 2) = 1.765\end{aligned}$$

$$CPI_{\text{Design \#2}} = (25\% \times 2.5) + (75\% \times 1.5) = 1.75$$

Design #2 is better than Design #1

Thank You