HY425 Lecture 01: Introduction

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

October 13, 2011

People

Instructor
Dimitris Nikolopoulos
e-mail: dsn
office: G-104 (UoC), G-115 (FORTH-ICS)
office hours: by appointment

Personnel
Vassilis Papaefstathiou
e-mail: papaef
office: G-113 (FORTH-ICS)
office hours: by appointment

Lectures
Monday–Wednesday, 11:00–13:00, Rho Alpha 203
Friday 11:00–13:00, Rho Alpha 203 on a need basis

Course topics

Advanced Computer Architecture

- Principles of computer systems design (0.5 week)
- Basic pipelining (1 week)
- Instruction-level parallelism in HW (3 weeks)
- Instruction-level parallelism in SW (2 weeks)
- Memory hierarchies (2 weeks)
- Multiprocessors (2 weeks)
- Storage systems (1 week, but rarely make it)
- Interconnection networks (1 week, but rarely make it)
- Reserved for future use (0.5 week)

Assignments

Paper assignments

- Homework problems on paper
- Nominal load per homework: 3 hours
- Expect 3–5 paper assignments

Machine assignments

- Alternating with homework assignments
- Nominal load per assignment: 20 hours
- Expect 2–3 machine assignments

Assignments

Lab assignments

- Simulation of essential processor components
- Dynamic binary instrumentation (PIN)
- Projects are individual (no exceptions)
- Potential topics:
  1. Event counting
  2. Branch prediction
  3. Cache design, prefetching
  4. Multiprocessor coherence and consistency (optional)

Grading

Exams

- Midterm (Friday 29.10.10): 20%
- Final: 20%
- No threshold on exams, just get a 5.0+ average

Assignments

- Programming assignments 35%
- Homework assignments on paper 25%
- Do not cheat, we use moss

Other important course information

- Web page: http://www.csd.uoc.gr/˜hy425
- Mailing list: hy425-list@csd.uoc.gr

Renaissance in computer design ending?

Microprocessors

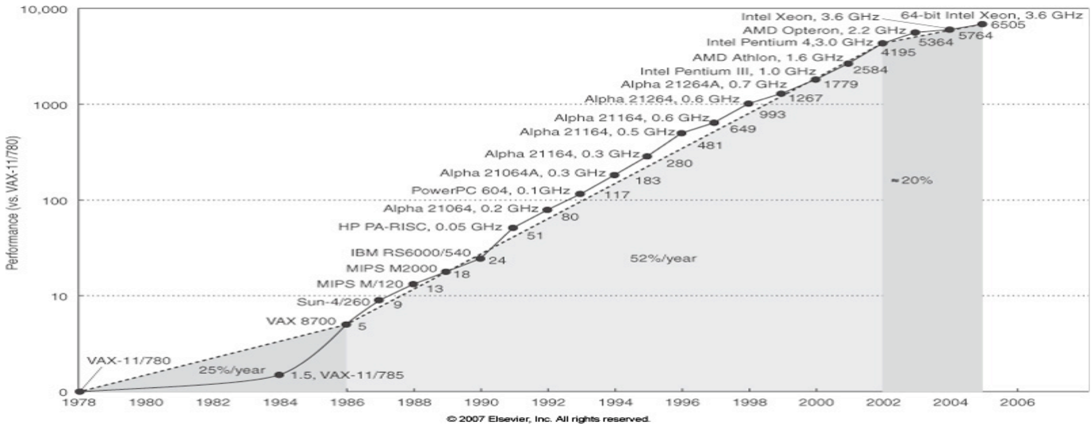
- RISC processors shifted focus to ILP, caches
- "Free" performance scaling with new technology
- Sustainable performance improvement until about 2002

New challenges

- Performance wall (lack of ILP, faster clocks, long latencies)
- Power wall (Performance per Watt drops)
- Reliability wall (more components, higher failure rates)

Renaissance in computer design ending?

Processor performance trends



Defining computer architecture

Instruction Set Architecture

- ISAs converged to a common RISC paradigm
  - CISC ISAs implemented on RISC pipelines
- Load-store architectures, general-purpose registers
- Aligned memory addressing, simple addressing modes
- Byte, word, double-word operands
- Arithmetic, logic, control operations
- Fixed-length encoding

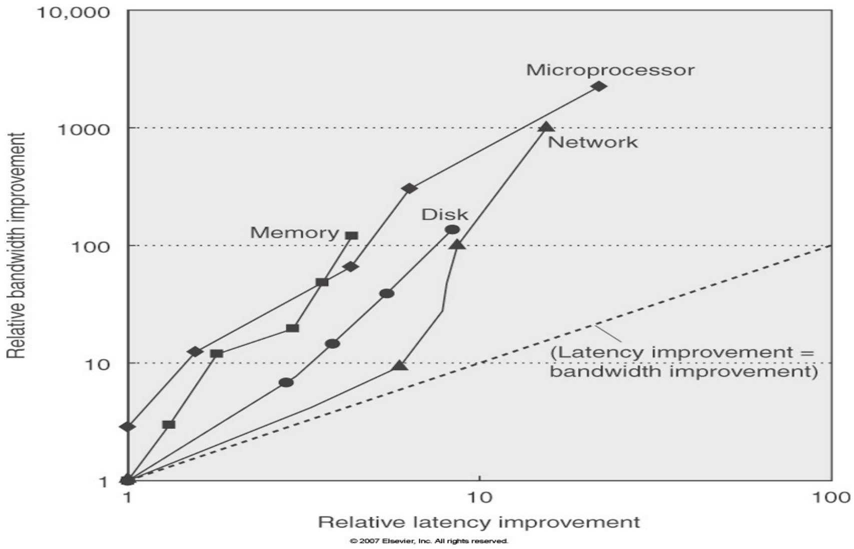
Defining computer architecture

Hardware Organization

- Processor architecture
  - Pipelining, hazards, ILP, HW/SW interface
- Memory hierarchies
- Interconnects
- I/O systems
- Hardware technology used (e.g. component size)
- Computer architecture focuses on organization and quantitative principles of design

What drives CA research and innovation?

Latency lags bandwidth



Transistor size trends and questions

Feature sizes, higher performance?

- Transistor size went town from 10 micros to 45 nanometers
- Quadratic increase in density, linear drop in feature size
- Linear increase in transistor performance

Where is the catch?

- Lower voltage to maintain safe operation
- Higher resistance and capacitance per unit of length
- Shorter wires but with higher resistance/capacitance
- Wire delays improving poorly compared to transistors

Power

The power equation

$$Power_{dynamic} = \frac{1}{2} \times \textit{Capacitive load} \times Voltage^2 \times frequency$$

$$Energy_{dynamic} = \textit{Capacitive load} \times Voltage^2$$

$$Power_{static} = Current_{static} \times Voltage$$

- Power due to switching more transistors increases
- Static power due to leakage current increasing

Measuring reliability

Reliability equations

$$MTTF = Mean\ Time\ To\ Failure$$

$$FIT = Failures\ In\ Time\ (per\ billion\ hours) = \frac{1}{MTTF}$$

$$MTTR = Mean\ Time\ to\ Repair$$

$$Module\ availability = \frac{MTTF}{MTTF + MTTR}$$

$$FIT_{system} = \sum_{i=1}^{\#components} FIT_i$$

Comparing design alternatives

Design X is n times faster than design Y

$$\frac{Execution\ time_Y}{Execution\ time_X} = n$$

- Wall-clock time: time to complete a task
- CPU time: time CPU is busy
- Workload: Mixture of programs (including OS) on a system
- Kernels: Common, important functions in applications
- Microbenchmarks: Synthetic programs trying to:
  - Isolate components and measure performance
  - Imitate workloads of real world in a controlled setting

Benchmarks

Desktop

- SPECCPU (revised every few years)
- Real programs measuring processor-memory activity

Multi-core desktop/server

- SPECOMP, SPECMPI (scientific), SPECapc (graphics)
- Focus on parallelism, synchronization, communication

Benchmarks

Client/Server

- SPECjbb, SPECjms, SPECjvm, SPECsfs, SPECmail, . . .
- Measuring throughput (how many tasks per unit of time)
- Measuring latency (how quickly does client get response)

Embedded systems

- EEMBC, MiBench
- Measuring performance, throughput, latency

Summarizing performance

Arithmetic mean of wall-clock time

- Biased by long-running programs
- May rank designs in non-intuitive ways:
  - Machine A: Program $P_1 \rightarrow 1000$ secs., $P_2 \rightarrow 1$ secs.
  - Machine B: Program $P_1 \rightarrow 800$ secs., $P_2 \rightarrow 100$ secs.
  - What if machine runs P_2 most of the time?

Measuring against a reference computer

$$n = \frac{SPEC_{ratio_A}}{SPEC_{ratio_B}} = \frac{\frac{\text{Execution time}_{reference}}{\text{Execution time}_A}}{\frac{\text{Execution time}_{reference}}{\text{Execution time}_B}} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = \frac{\text{Performance}_A}{\text{Performance}_B}$$

Summarizing performance (cont.)

Example

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40

Means

- Total time ignores program contribution to total workload
- Arithmetic mean biased by long programs
- Weighted arithmetic mean a better choice?
- How do we calculate weights?

Summarizing performance (cont.)

Weighted arithmetic mean

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

Example, W(1) = W(2) = 50

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40
Weighted mean	500.50	55.00	20.00

Summarizing performance (cont.)

Weighted arithmetic mean

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

Example, W(1) = 0.909 W(2) = 0.091

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40
Weighted mean	91.91	18.19	20.00

Summarizing performance (cont.)

Weighted arithmetic mean

$$\sum_{i=1}^{n} Weight_i \times Time_i$$

Example, W(1) = 0.999 W(2) = 0.001

	Computer A	Computer B	Computer C
Program P1 (secs)	1	10	20
Program P2 (secs)	1000	100	20
Total time (secs)	1001	110	40
Weighted mean	2.00	10.09	20.00

Summarizing performance (cont.)

Using ratios

- Ratios against reference machine are independent of mixture of programs

Geometric mean

$$\sqrt[n]{\prod_{i=1}^{n} Execution\ time\ ratio_i}$$

$$\frac{Geometric\ mean_A}{Geometric\ mean_B} = Geometric\ mean(\frac{A}{B}) \quad (1)$$

Pros and cons of geometric means

Pros

- Consistent rankings, independent of program frequencies
- Not influenced by peculiarities of any single machine

Cons

- Geometric mean does not predict execution time
  - Sensitivity to benchmark vs. machine remains
  - Encourages machine tuning for specific benchmarks
  - Benchmarks can not be touched, but compilers can!
- Any “averaging” metric loses information

Qualitative principles of design

Taking advantage of parallelism

- Use pipelining to overlap instructions
- Use multiple execution units
- Use multiple cores
- Use multiple processors to increase throughput

Locality

- Programs reuse instructions and data
- 90-10 rule
  - 90% of execution time spent running 10% of instructions
- Programs access data in nearby addresses

Qualitative principles of design (cont.)

Make the common case fast

- Trade-off's in design (e.g. performance vs. power/area)
- Provide efficient design for the common case
- Amdahl's Law

Using performance equations

- Processor performance equation = $f(\textit{cycle time}, \textit{instruction count}, \textit{stalls}, \textit{instruction mix}, \ldots)$

Amdahl's Law

$$\textit{speedup} = \frac{\textit{performance with enhancement}}{\textit{performance without enhancement}}$$

$$\textit{execution time}_{new} = \textit{execution time}_{old} \times \left((1 - \textit{fraction}_{enhanced}) + \frac{\textit{fraction}_{enhanced}}{\textit{speedup}_{enhanced}}\right)$$

$$\textit{speedup}_{overall} = \frac{\textit{execution time}_{old}}{\textit{execution time}_{new}} = \frac{1}{(1 - \textit{fraction}_{enhanced}) + \frac{\textit{fraction}_{enhanced}}{\textit{speedup}_{enhanced}}} \Rightarrow$$

$$\textit{speedup}_{overall} \to \frac{1}{1 - \textit{fraction}_{enhanced}}$$

Processor performance equation

CPU time

$$CPU\ time = CPI \times cycle\ time$$

$$CPI = \frac{CPU\ clock\ cycles}{instruction\ count} \Rightarrow$$

$$CPU\ time = instruction\ count \times CPI \times cycle\ time \Rightarrow$$

$$CPU\ time = \frac{instructions}{program} \times \frac{clock\ cycles}{instruction} \times \frac{seconds}{clock\ cycle}$$

Processor performance equation

How can CA help?

- Technology has been providing faster clock speeds
  - Main performance factor for almost 20 years
  - Trend seems to reverse
  - Limitations due to power consumption, reliability
- Architecture can pack more computing power in same area
- Architecture can improve CPI
- Algorithms and compilers can reduce instruction count

Processor performance equation

Instruction mixes

$$CPU\ time = CPI \times cycle\ time$$

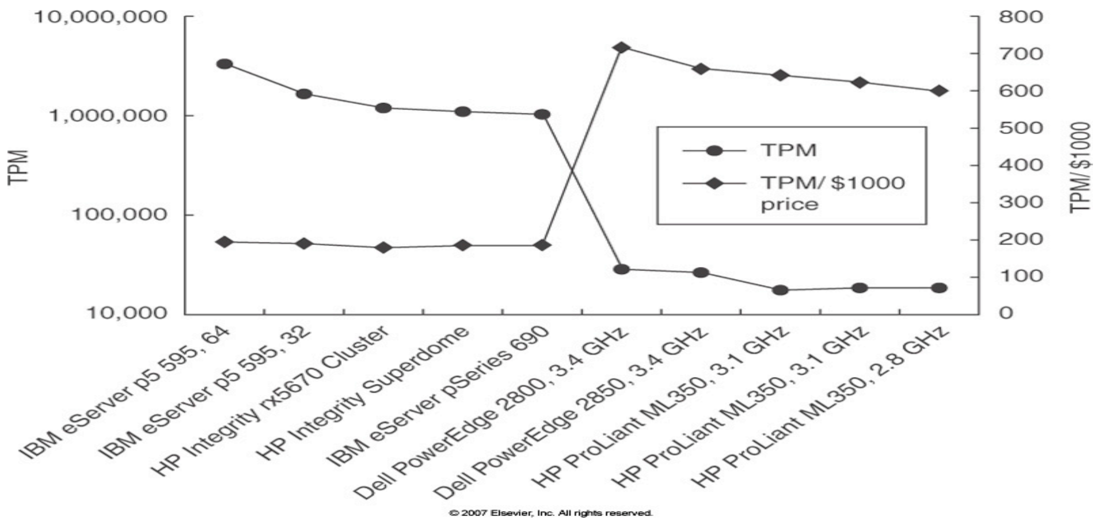
$$CPI = \frac{CPU\ clock\ cycles}{instruction\ count} \Rightarrow$$

$$CPU\ time = instruction\ count \times CPI \times cycle\ time \Rightarrow$$

$$cpu\ time = \frac{instructions}{program} \times \frac{clock\ cycles}{instruction} \times \frac{seconds}{clock\ cycle} \Rightarrow$$

$$CPU\ time = \left(\sum_{i=1}^{n} IC_i \times CPI_i\right) \times cycle\ time$$

Price/performance



Concluding remarks

Fallacies and pitfalls

- Ignoring Amdahl's law
- Reliability is as good as that of the most faulty component
- Cost of processor dominates system cost?
  - Currently, on servers and laptops storage dominates cost!
- Benchmarks remain valid for long
  - Workloads evolve (Internet, laptops, handheld computers, sensors, controllers, actuators, . . .)
  - Tuning for depreciated benchmarks undesirable

Concluding remarks (cont.)

Fallacies and pitfalls

- Reliability metrics ignoring lifetime of component
- Peak performance is expected performance
- Detecting but not correcting faults
  - Many components in the architecture non-critical for correct operation
  - Important to protect, check and duplicate critical components