CSE 4201
Computer Architecture

Prof. Mokhtar Aboelaze

Parts of these slides are taken from
Notes by Prof. David Patterson at UCB

CSE4201

- Office hours
- Grading Policy
  - HW
  - Quizzes
  - Midterm
  - Final
- Text: Computer Architecture: QA 4th edition

Syllabus

- Introduction and performance
- Review –MIPS and pipelines
- ILP (Instruction Level Parallelism) and limits of ILP
- Multiprocessors and thread level parallelism
- Memory
- Storage Systems
- Network processor or Multi-Core systems (depending on time)

Outline

- Historical prospective and new technology trends.
- Performance: How to measure it? What does it mean?
- MIPS 5 stage pipelining and IS

Historical Prospective

- Decade of 70's (Microprocessors)

  Programmable Controllers, Single Chip Microprocessors, Personal Computers
- Decade of 80's (RISC Architecture)

  Instruction Pipelining, Fast Cache Memories, Compiler Optimizations
- Decade of 90's (Instruction Level Parallelism)

  Superscalar Processors, Aggressive Code Scheduling, Low Cost Supercomputing, Out of Order Execution
- Decade of 00

  Thread level parallelism, Data level parallelism, multicore, SoC

Computer Architecture

- Computer architecture now is >> ISA
- What matters is how the complete system performs
- Time spent on IS this year is les than previous offering of the course More on ILP, TLP, multiprocessing, and **if** time permit Network processors and SoC.

Computer Architecture

"We are dedicating all of our future product development to multicore designs. … This is a sea change in computing"

Paul Otellini, President, Intel (2004)

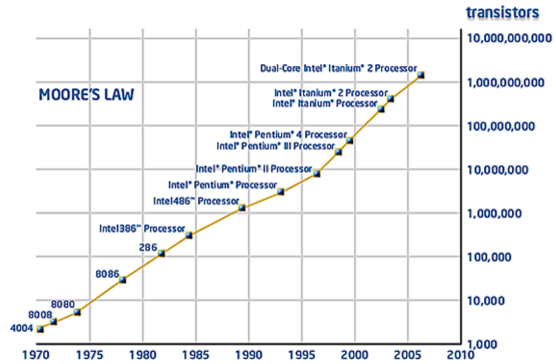
1000+ Level Parallelism, cannot be solved by just by computer architects and compiler writers alone, but also cannot be solved *without* participation of computer architects

David Patterson

Technology trends

- Used to be transistors are important, power is not a problem.
- Now, **Power is the problem** Transistors are almost free ?
- New challenges: Power and ILP (adding hardware helps, but finally the law of diminishing return kicks in).

Number of transistors



http://www.intel.com/technology/mooreslaw/index.htm

CPUs:

- 1982 Intel 80286
- 12.5 MHz
- 2 MIPS (peak)
- Latency 320 ns
- 134,000 xtors, 47 mm^2
- 16-bit data bus, 68 pins
- Microcode interpreter, separate FPU chip
- (no caches)

- 2001 Intel Pentium 4
- 1500 MHz (120X)
- 4500 MIPS (peak) (2250X)
- Latency 15 ns (20X)
- 42,000,000 xtors, 217 mm^2
- 64-bit data bus, 423 pins
- 3-way superscalar, Dynamic translate to RISC, Superpipelined (22 stage), Out-of-Order execution
- On-chip 8KB Data caches, 96KB Instr. Trace cache, 256KB L2 cache

Disks:

- CDC Wren I, 1983
- 3600 RPM
- 0.03 GBytes capacity
- Tracks/Inch: 800
- Bits/Inch: 9550
- Three 5.25" platters
- Bandwidth: 0.6 MBytes/sec
- Latency: 48.3 ms
- Cache: none

- Seagate 373453, 2003
- 15000 RPM (4X)
- 73.4 GBytes (2500X)
- Tracks/Inch: 64000 (80X)
- Bits/Inch: 533,000 (60X)
- Four 2.5" platters (in 3.5" form factor)
- Bandwidth: 86 MBytes/sec (140X)
- Latency: 5.7 ms (8X)
- Cache: 8 MBytes

Memory:

- 1980 DRAM (asynchronous)
- 0.06 Mbits/chip
- 64,000 xtors,
- 35 mm2
- 16-bit data bus per module, 16 pins/chip
- 13 Mbytes/sec
- Latency: 225 ns
- (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
- 256.00 Mbits/chip (4000X)
- 256,000,000 xtors, (64,000)
- 204 mm^2 (6x)
- 64-bit data bus per DIMM, 66 pins/chip (4X)
- 1600 Mbytes/sec (120X)
- Latency: 52 ns (4X)
- Block transfers (page mode)

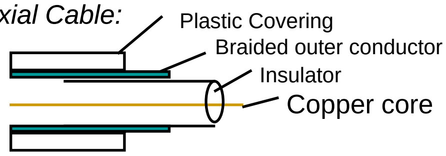
LANs

- Ethernet 802.3
- Year of Standard: 1978
- 10 Mbits/s link speed
- Latency: 3000 μsec
- Shared media
- Coaxial cable

- Ethernet 802.3ae
- Year of Standard: 2003
- 10,000 Mbits/s link speed (1000X)
- Latency: 190 μsec (15X)
- Switched media
- Category 5 copper wire

"Cat 5" is 4 twisted pairs in bundle

Coaxial Cable:



Twisted Pair:

Copper, 1mm thick,
twisted to avoid antenna effect

Two notions of "performance"

Plane	DC to Paris	Top Speed	Passen-gers	Throughput (p-mph)
Boeing 747	6.5 hours	610 mph	470	286,700
BAC/Sud Concorde	3 hours	1350 mph	132	178,200

- **Which has higher performance?**
- **Time to deliver 1 passenger?**
- **Time to deliver 400 passengers?**

Response Time v. Throughput

- **Time of Concorde vs. Boeing 747?**
  - **Concord is 6.5 hours / 3 hours = 2.2 times as fast**
- **Throughput of Boeing vs. Concorde?**
  - **Boeing 747: 286,700 p-mph / 178,200 p-mph = 1.6 times as fast**
- **Boeing is 1.6 times (160%) as fast in terms of throughput**
- **Concord is 2.2 times (220%) as fast in terms of flying time (response time)**

Computer Performance

- Response Time = Execution Time = Latency Time in a computer:
  - Time for 1 job (Interest to the user)
- Throughput = Bandwidth in a computer :
  - Jobs per unit time (interest to the system administrator)

Performance

- "X is n times as fast as Y" means
- Performance$_x$ = n X Performance$_y$

$$\frac{T_y}{T_x} = 1 + \frac{n}{100}$$

- Example, A completes job in 10 sec, B in 15, A is 50% faster than B

Making the Common Case Faster

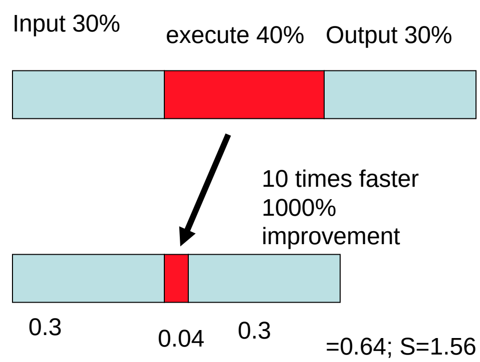
- Usually, we have a limited amount of resources, how to allocate them?
- Investing a lot of resources in improving a rare situation is not likely to improve the performance
- EX: Make division faster on the expense of overflow or divide by zero which is not likely to happen frequently anyway

Amdahl’s law

- If α is the fraction of the computations that could be enhanced by a factor of S, then

$$T_{new} = T_{old}\left((1-\alpha) + \frac{\alpha}{S}\right)$$

$$Speedup = \frac{1}{(1-\alpha) + \frac{\alpha}{s}}$$



Amdahl’s Law

- Consider a computer system that uses the CPU 50% of the time, and the I/O 50% of the time, consider two cases CPU costs 1/3 of system
- Improve the CPU by a factor of 5

$$speedup = \frac{1}{0.5 + \frac{0.5}{5}} = 1.667$$

- The cost is 2/3 + 1/3 * 5=2.33
- Improve CPU by a factor of 2

$$Speedup = \frac{1}{0.5 + \frac{0.5}{2}} = 1.33$$

- Cost = 2/3 + 1/3 * 2=1.33 (better investment)

Locality of Reference

- For programs, the rule of thumb is any program spends 90% of the time in 10% of the code
- Temporal Locality: Recently accessed items are likely to be accessed again in the near future
- Spatial Locality: Items whose addresses are near one another tends to be referenced close together in time

Locality of reference

- Calculate the speedup if we put a cache of 500KB in a system that runs a 5MB (10MB) program, assume 90%,10% rule of thumb and assume that cache is 5 times faster than RAM
- Case 1 $$speedup = \frac{1}{0.1 + \frac{0.9}{5}} = 3.57$$
- Case 2 $$speedup = \frac{1}{0.55 + \frac{0.45}{5}} = 1.5625$$

Performance

- $CPU_{time}=T_c$*IC * CPI
- Where, T_c is the clock cycle and IC is the instruction count
- CPI=(Number of clock cycles)/IC
- CPU_{time}=IC * CPI * T_c
- Clcok rate depends on organization and technology
- CPI depends on compiler, Inst. Set, and organization
- IC depends on program, compiler, and Inst. set
- The average CPI could be expressed as

$$CPI_{av} = \frac{\sum_{i=1}^{n} CPI_i * I_i}{IC}$$

Example

Question:

A program runs on a 400 MHz computer in 10 secs. We like the program to run in 6 secs by designing a faster CPU. Assume that increasing clock rate would mean the program needs 20% more clock cycles. What clock rate should the designer target?

Answer:

The number of clock cycles for the program on the present computer = 10 X 400 X 10^6 = 4000 X 10^6

With 20% increase, the new computer should take 1.2 X 4000 X 10^6 = 4800 X 10^6 cycles

Required execution time = 6 seconds

Then the required clock rate = 4800/6 X 10^6 cycles/sec = 800 MHz

Example

OP	Frequency	Cycles
ALU	43%	1
Load	21%	2
Store	12%	2
Branch	24%	2

- Increasing cycle by 15% → load = 1 cycle
- Old CPI = 0.43+2*0.21+2*0.12+2*0.24=1.57
- New CPI= 0.43+1*0.21+2*0.12+2*0.24=1.36
- Speedup=IC*1.57*TC/(IC*1.36*1.15TC)=1.003
- Barely, if more than 15%, NO WAY

Example

- Consider the previous example. If 25% ALU operations use a loaded operand that is not used again
- CPI=0.43+2*0.21+2*0.12+2*0.43=1.57
- Time = IC * 1.57 * T_c=1.57IC*T_c
- If we use another design that supports reg/mem instructions in 2 cycles and increases branching by 1 cycle.
- In this case, the 25% of loads are replaced by reg/mem instructions

Example 'cont.

ALU (LS)	43% — 10.75%	1
ALU reg/m	10.75%	2
Loads	10.25%	2
Store	12%	2
branches	24%	3

$$CPI = \frac{0.3225 + 2*(0.1075 + 0.1025 + 0.12 + 0.43) + 0.24*3}{1 - 0.25*0.43} = 1.908$$

$$CPU_{time} = 1.908*(1 - 0.25*0.43) IC*T_c = 1.703 IC*T_c$$

Bad Idea

Example

- Consider 2 CPU's
- The first, sets a condition code by a compare followed by a branch, 20% are branches (another 20% are compare)
- The seconds, compares in the branch instruction and 25% slower
- For A, CPI = 0.2*2 + 0.8 = 1.2
- Time = IC*1.2*T_c
- For B CPI = 0.25*2 + 0.75=1.25 (now branches are 20 out of 80)
- Time (for B) = 0.8IC*1.25*1.25T_c=1.25 IC*T_c Slower

EXAMPLE

Effect?

A program executed on machine A with a 1ns clock gives a CPI of 2.0. The same program with machine B having the same ISA and a 2ns clock gives a CPI of 1.2. Which machine is faster and by how much?

Answer: **Let I be the instruction count.**

CPU clock cycles for A = I x 2.0

Execution time on A = 2 x I ns

CPU clock cycles for B = I x 1.2

Execution time on B = I x 1.2 x 2 ns = 2.4 I ns

=> CPU A is faster by 1.2 times.

Example (RISC processor)

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%
			2.2	

Typical Mix

How much faster would the machine be if a better data cache reduced the average load time to 2 cycles? (1.6)

How does this compare with using branch prediction to shave a cycle off the branch time? (2.0)

What if two ALU instructions could be executed at once?

Example

Add register / memory operations

- One source operand in memory
- One source operand in register
- Cycle count of 2

Branch cycle count increased to 3

What fraction of the loads must be eliminated for this to pay off?

Base Machine (Reg / Reg)

Op	F_i	CPI_i
ALU	50%	1
Load	20%	2
Store	10%	2
Branch	20%	2

Example

Exec Time = Instr Cnt x CPI x Clock

Op	F_i	CPI_i		I_i	CPI_i	
ALU	.50	1	.5	.5 – X	1	.5 – X
Load	.20	2	.4	.2 – X	2	.4 – 2X
Store	.10	2	.2	.1	2	.2
Branch	.20	2	.4	.2	3	.6
Reg/Mem				X	2	2X

$\text{Instr Cnt}_{Old} \times \text{CPI}_{Old} \times \text{Clock}_{Old} = \text{Instr Cnt}_{New} \times \text{CPI}_{New} \times \text{Clock}_{New}$

$1.00 * 1.5 = (1 – X) * (1.7 – X)/(1 – X)$

$1.00 * 1.5 = (1 – X) * (1.7 – X)/(1 – X)$

$1.5 = 1.7 – X$

$0.2 = X$

ALL loads must be eliminated for this to be a win!

Power

- For CMOS chips, traditional dominant energy consumption has been in switching transistors, called *dynamic power*

$$P_{dynamic} = 0.5 * C_L * V^2 * f$$

- For mobile devices, energy better metric

$$P_{dynamic} = C_L * V^2$$

- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy
- Capacitive load is a function of number of transistors connected to output and technology, which determines capacitance of wires and transistors
- Dropping voltage helps both
- To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)

Power

- Suppose 15% reduction in voltage results in a 15% reduction in frequency. What is impact on dynamic power?

$$\begin{aligned} Power_{dynamic} &= 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched \\ &= 1/2 \times CapacitiveLoad \times (.85 \times Voltage)^2 \times 0.85 \times FrequencySwitched \\ &= (.85)^3 \times OldPower_{dynamic} \\ &\approx 0.6 \times OldPower_{dynamic} \end{aligned}$$

Power

- **Because leakage current flows even when a transistor is off. Now static power is important too**

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

- **Leakage current increases in processors with smaller transistor sizes**
- **Increasing the number of transistors increases power even if they are turned off**
- **In 2006, goal for leakage is 25% of total power consumption; high performance designs at 40%**
- **Very low power systems even gate voltage to inactive modules to control loss due to leakage**

Dependability

- How dependable the system is?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- Systems alternate between 2 states of service with respect to an SLA:

1. Service accomplishment, where the service is delivered as specified in SLA
2. Service interruption, where the delivered service is different from the SLA

- Failure = transition from state 1 to state 2
- Restoration = transition from state 2 to state 1

Dependability

- *Module reliability* = measure of continuous service accomplishment (or time to failure).
  2 metrics
1. *Mean Time To Failure* (*MTTF*) measures Reliability
2. *Failures In Time* (*FIT*) = 1/MTTF, the rate of failures
   - Traditionally reported as failures per billion hours of operation
- *Mean Time To Repair* (*MTTR*) measures Service Interruption
  - *Mean Time Between Failures* (*MTBF*) = MTTF+MTTR
- *Module availability* measures service as it alternates between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- *Module availability = MTTF / (MTTF + MTTR)*

Dependability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):

$$
\begin{aligned}
FailureRate &= 10 \times (1/1{,}000{,}000) + 1/500{,}000 + 1/200{,}000 \\
&= 10 + 2 + 5/1{,}000{,}000 \\
&= 17/1{,}000{,}000 \\
&= 17{,}000 FIT \quad 17 \times 10^{6} \\
MTTF &= 1{,}000{,}000{,}000/17{,}000 \\
&\approx 59{,}000 hours
\end{aligned}
$$

MIPS

- MIPS: Million Instructions Per Second
- $MIPS=IC/(T*10^6)=\#of\ cycles/(CPI*T*10^6)$
- $MIPS= (Clock\ rate)/(CPI*10^6)$
- The difficulty of choosing such a measure is it doesn't define what is an *instruction* (xor or div)
- Any optimizing compiler that tends to reduce the number of instructions resulting in saving execution time reduces the MIPS rating of the machine.

MIPS

- One way to overcome this difficulty is to use the *relative MIPS*
- Relative MIPS = $T_r/T * MIPS_r$
- Where
  - T_r is the execution time on a standard machine
  - T is the execution time of the machine to be rated
  - $MIPS_r$ is the MIPS rating of the standard machine
- The main problem is, what exactly is the reference machine

MFLOPS

- MFLOPS=Million Floating Point Operation per Second
- Still, what is a FP instruction
- Programs like compilers, almost has no FP operations at all
- Normalized MFLOPS gives weight to different FP operations (1 for add, 2 for multiply, 4 for divide, 8 for sqrt, …)

Measuring Performance

- Real Programs: we run the actual program and measure the time, the difficulty is which program?
- Kernels: Extract small pieces of real programs and use tem to evaluate performance (livermoore loops and linpack)
- Toy Benchmarks: small programs that produces results already known (quicksort, puzzle, …)
- Synthetic Benchmarks: Similar to kernels, specifically created to match the average frequency of different operations (whettstone and Dhrystone)

What Programs Measure for Comparison?

- User reality: CPI varies with program, workload mix, OS, compiler, etc.
- Ideally would run typical programs with typical input before purchase
- Called a "workload"; For example:
  - Engineers uses compilers, spreadsheets, CAD tools
  - Authors uses word processors, drawing programs, compression software
- In some situations its hard to do
  - Don't have access to machine to "benchmark" before purchase
  - Don't know workload in future

Basis of Evaluation

Pros		Cons
• representative	Actual Target Workload	• very specific • non-portable • difficult to run, or measure • hard to identify cause
• portable • widely used • improvements useful in reality	Full Application Benchmarks	•less representative
• easy to run, early in design cycle	Small "Kernel" Benchmarks	• easy to "fool"
• identify peak capability and potential bottlenecks	Microbenchmarks	• "peak" may be a long way from application performance

Examples

- Workstations: Standard Performance Evaluation Corporation (SPEC)
  - For example, for SPEC CPU2006: CINT has 12 benchmarks (9 use C, 3 use C++) (perlbench, bzip2, gcc, …..). While for CFP2006 has 17 benchmarks (C, C++, Fortran, and mix of C and Fortran) such as bwaves (fluid dynamics), dealII (Finite element anaysis), …
  - `http://www.spec.org`
  - Separate average for integer (CINT) and FP (CFP) relative to base machine
  - Benchmarks distributed in source code
  - Company representatives select workload
  - Compiler, machine designers target benchmarks so try to change every 3 years

Kernel Example

1. X=1.0
2. Y=1.0
3. Z=1.0
4. Do I=1,N8
   1. CALL P3(X,Y,Z)

SUBROUTINE P3(X,Y,Z)
X1=1
Y1=1
Z1=1
X1=T*(X1-Y1)
Y1=(T*(X1+Y1)
Z=(X1+Y1)/T

Reporting Performance

- If T_i is the time to run program I
- Arithmetic mean $\frac{1}{n_i}\sum_{i=1}^{n} T_i$
- Weighted arithmetic $\sum_{i=1}^{n} w_i T_i$
- Harmonic mean $\frac{n}{\sum_{i=1}^{n}\frac{1}{Rate_i}}, \quad Rate_i = f(1/T_i)$
- Weighted harmonic mean $\frac{n}{\sum_{i=1}^{n}\frac{w_i}{Rate_i}}, \quad Rate_i = f(1/T_i)$
- Geometric mean $\prod_{i=1}^{n} TR_i$

Where Tr_i is the execution time of program I normalized to a ref. machine

Example

•

P1	1	10	20	0.5	0.909	0.999
P2	1000	100	20	0.5	0.091	0.001
Arithmetic mean w1	500.5	55	20			
Arithmetic mean w2	91.8	18.18	20			
Arithmetic mean w3	2	10.09	20			
	A	B	C	w1	w2	w3

Example

•

	nroma	lized	To A			B		C	
	A	B	C	A	B	C	A	B	C
p1	1	10	20	0.1	1	2.0	0.05	0.5	1
P2	1	0.1	0.02	10	1	0.2	50	5	1
Arith	1	5.05	10.01	5.05	1	1.1	25.0 3	2.75	1
Geo	1	1	0.63	1	1	0.63	1.58	1.58	1

Example (cont)

- Notice that the geometric mean does not represent the relative execution times, for example in the first case, it said that A and B are equal
- The geometric mean is consistent independent of the normalization, A and B are the same and independent of C