Computer Architecture

Chap 1. Computer Abstractions and Technology

Hyuk-Jun Lee, PhD

Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea

Email: hyukjunl@sogang.ac.kr

The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Mobile Computing: Cell phones
  - Bioengineering: Human genome project
  - Data Mining: World Wide Web, Search Engines
  - Machine Learning: Deep learning
- Computers are pervasive

CPUs

Apple AP

Google TPU

Samsung AP

Nvidia GPU

Classes of Computers

- Desktop computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

The Processor Market

The PostPC Era

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

What You Will Learn

- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

Understanding Performance

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

Below Your Program

- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

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;
}
```

Compiler

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
```

Assembler

Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Components of a Computer

The BIG Picture

- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

Anatomy of a Computer

Anatomy of a Mouse

- Optical mouse
  - LED illuminates desktop
  - Small low-res camera
  - Basic image processor
    - Looks for x, y movement
  - Buttons & wheel
- Supersedes roller-ball mechanical mouse

Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory

Opening the Box (PC,Notebook)

Opening the Box (Smartphone)

Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, …
- Cache memory
  - Small fast SRAM memory for immediate access to data

Inside the Processor

- AMD Barcelona: 4 processor cores

Inside the Processor

- A12 processor (Apple)

Abstractions

The BIG Picture

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)

Networks

- Communication and resource sharing
- Local area network (LAN): Ethernet
  - Within a building
- Wide area network (WAN: the Internet)
- Wireless network: WiFi, Bluetooth

Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost

DRAM capacity

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Defining Performance

- Which airplane has the best performance?

Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time == execution time of application

  순수히 app가 돌아가는 시간을 측정
  OS의 스케쥴링, I/O, 대기 등등은 빼고
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

CPU Clocking

- Operation of digital hardware governed by a constant-rate clock

- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second 1 / clock period

  1 / (250 * 10^-12) = 4 * 10^9 hz(events per second)
  - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz 1초에 4 * 10^9 개의 명령어를 수행

한 머신 명령어는 한 클럭 사이클에서 실행한다(이상적)

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$
$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

cycle time

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

cycles per instruction

Instruction Count and CPI

dynamic

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

명령어 개수 명령어가 가지는 사이클 사이클 실행 시간

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

instruction count - static and dynamic
static - 컴파일 하고 나온 어셈블리 명령어의 명백히 보이는 개수
dynamic - 실제로 실행하는 명령어의 개수

The dynamic instruction count is the actual number of instructions executed by the CPU for a specific program execution, whereas the static instruction count is the number of instruction the program has.

- Instruction Count for a program

static IC는 명령어의 절대적인 크기를 나타내기 때문에 실제 실행되는 명령 개수와는 다르고 그냥 파일 크기를 나타내는데 사용

  - Determined by program, ISA and compiler

instruction set archeitecture

- Average cycles per instruction

ripple carry adder - 회로를 덜 쓰지만 carry를 기다려야해서 느림
carry look ahead adder - 회로를 더 쓰지만 carry를 미리 알아서 더 빠름

  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA

A는 클럭이 빠르고 B는 CPI가 낮다(더 적은 클럭을 사용해 명령어 수행 가능)
두 요소에서 A,B의 우위가 다르니 cpu time을 계산해서 성능을 비교한다

- Which is faster, and by how much?

1/clock rate

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…by this much

CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

  - Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left(\text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

CPI는 프로그램 마다 다르다

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

clock cycle = cpi * ic

- Sequence 1: IC = 5
  - Clock Cycles
    $= 2\times1 + 1\times2 + 2\times3$
    $= 10$
  - Avg. CPI = 10/5 = 2.0

2+1+2

- Sequence 2: IC = 6
  - Clock Cycles
    $= 4\times1 + 1\times2 + 1\times3$
    $= 9$
  - Avg. CPI = 9/6 = 1.5

4+1+1

Performance Summary

The BIG Picture

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

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Power Trends

클럭 속도가 여기서 멈추는 이유 - 전력 소모와 열이 심해서

0,1 비트 정보를 담는데 필요한 전력

- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

캐패시터에 담을 수 있는 전류량

×30

5V → 1V

×1000

Reducing Dynamic Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU

    전력을 줄이면 power frequency가 낮아짐
  - 15% voltage and 15% frequency reduction

    lower power supply voltage - slow -> need faster clock

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall

  캐패시터 전력을 너무 줄이면 트렌지스터가 작동을 안함
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

Uniprocessor Performance

Constrained by power, instruction-level parallelism, memory latency

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

Manufacturing ICs

cpu 불량률이 높다

- Yield: proportion of working dies per wafer

AMD Opteron X2 Wafer

- X2: 300mm wafer, 117 chips, 90nm technology
- X4: 45nm technology

Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

불량이 얼마나 나올지는 알 수 없지만 하나의 웨이퍼에 많은 die를 만들 수 있으면 성공률 높아짐

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios

    비율의 평균을 구할 때 기하 평균
    - CINT2006 (integer) and CFP2006 (floating-point)

여러개의 애플리케이션의 성능을 올리는 건 쉽지 않음

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

machine A 1 100 - arith mean 50.5 geo mean 10
machine B 10 10 - arith mean 10 geo mean 10
A는 두번째 앱의 성능만 좋아졌는데 그 좋아진 크기가 커서 절대적인 성능 증가를 보는 산술평균에서 좋게 보이지만
기하 평균적으로 보면 B랑 다를 게 없음
대신 B는 두 앱에서 성능 증가를 보임

SPECspeed 2017 Integer benchmarks on a 1.8 GHz Intel Xeon E5-2650L

Description	*Name*	*Instruction Count x 10^9*	*CPI*	*Clock cycle time (seconds x 10^–9)*	*Execution Time (seconds)*	*Reference Time (seconds)*	*SPECratio*
Perl interpreter	perlbench	2684	0.42	0.556	627	1774	2.83
GNU C compiler	gcc	2322	0.67	0.556	863	3976	4.61
Route planning	mcf	1786	1.22	0.556	1215	4721	3.89
Discrete Event simulation - computer network	omnetpp	1107	0.82	0.556	507	1630	3.21
XML to HTML conversion via XSLT	xalancbmk	1314	0.75	0.556	549	1417	2.58
Video compression	x264	4488	0.32	0.556	813	1763	2.17
Artificial Intelligence: alpha-beta tree search (Chess)	deepsjeng	2216	0.57	0.556	698	1432	2.05
Artificial Intelligence: Monte Carlo tree search (Go)	leela	2236	0.79	0.556	987	1703	1.73
Artificial Intelligence: recursive solution generator (Sudoku)	exchange2	6683	0.46	0.556	1718	2939	1.71
General data compression	xz	8533	1.32	0.556	6290	6182	0.98
Geometric mean							2.36

SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

$$\text{Overall ssj_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

SPECpower_ssj2008 for Xeon E5-2650L

Target Load %	Performance (ssj_ops)	Average Power (watts)
100%	4,864,136	347
90%	4,389,196	312
80%	3,905,724	278
70%	3,418,737	241
60%	2,925,811	212
50%	2,439,017	183
40%	1,951,394	160
30%	1,461,411	141
20%	974,045	128
10%	485,973	115
0%	0	48
Overall Sum	26,815,444	2,165
∑ssj_ops / ∑power =		12,385

아무것도 안해도 전력이 소모되는 이유
static power consumption
capacity에 있는 1의 정보를 담은 전력의 양이 줄어들지 않게 채워줘야 해서

성능은 부분적으로 나아지는 거기 때문에
얼마나 부분적으로 성능이 증가되는지 나태냄

Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

한 프로그램에서 A,B 두 부분이 있을 때
A만 실행 속도가 improve 됐다 하면 전체 프로그램이 빨라진 비율을 보기
성능된 A / 성능된 프로그램

$$T_{improved} = \frac{T_{affected}}{\text{improvement factor}} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!
- Corollary: make the common case fast

Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%) 90%를 안쓰고 있는데 100%를 쓰는 거의 반의 전력을 쓰고 있음
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

단순히 명령어를 많이 쓰는 걸로 성능이 좋은 건 아님
mul, add를 쓰는 cpu랑
madd를 쓰는 cpu랑 비교했을 때 mul, add를 쓰는게 mips가 더 클 수 있는데
이게 더 좋은 거냐는 다른 문제

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- CPI varies between programs on a given CPU

Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance