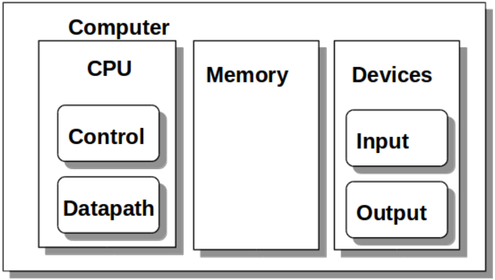
CS3350B Computer Architecture
CPU Performance and Profiling

Marc Moreno Maza

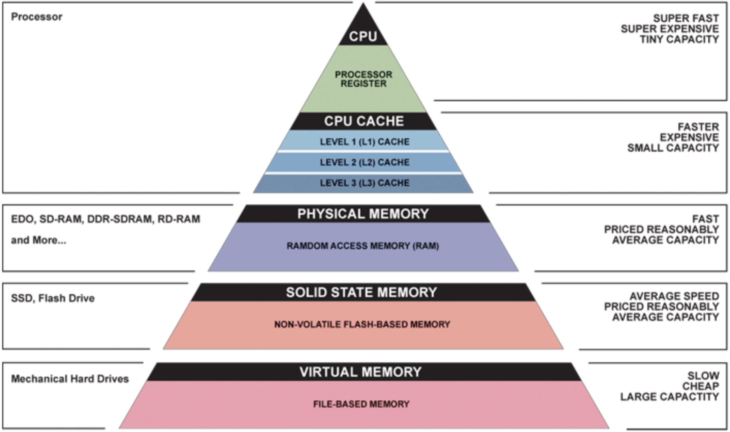
http://www.csd.uwo.ca/~moreno/cs3350_moreno/index.html
Department of Computer Science
University of Western Ontario, Canada

Tuesday January 10, 2017

Components of a computer

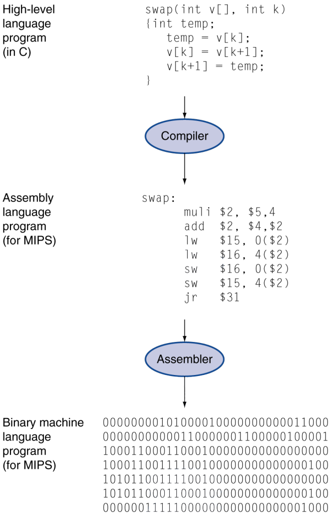


Memory hierarchy



Levels of program code

- **High-level language**
  - Level of abstraction closer to the problem domain
  - Designed for productivity and portability
- **Assembly language**
  - Textual representation of instructions
  - Many constructs of the HLL are translated into combinations of low-level constructs
- **Hardware representation**
  - Binary digits (bits)
  - Encoded instructions and data



Understanding Performance

- **Algorithm analysis:**
  Algorithm analysis:
  estimates the number of operations executed, the number of cache misses, etc.
- **Programming language, compiler, architecture**
  the compilation process determines the machine instructions executed per HLL operation
- **Processor and memory system**
  determine how fast instructions are executed
- **I/O system (including OS)**
  determines how fast I/O operations are executed

Performance Metrics

- Purchasing perspective:
  given a collection of machines, which one has the
  - best performance?
  - best cost?
  - best cost/performance?
- Design perspective:
  faced with design options, which one has the
  - best performance improvement?
  - best cost?
  - best cost/performance?
- Both require:
  - basis for comparison,
  - metrics for evaluation.
- **Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors**

CPU Performance

- We are normally interested in reducing
  - Response time (aka execution time) – the time between the start and the completion of a task
    - Important to individual users
  - Thus, to maximize performance, we need to minimize execution time

$$\text{performance}_X = 1/\text{execution\_time}_X$$

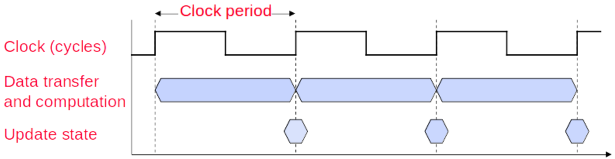
If X is n times faster than Y, then

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

- And we are interested in increasing
  - Throughput - the total amount of work done in a given unit of time
    - Important to data center managers
  - Decreasing response time usually improves throughput, but other factors are important (task scheduling, memory bandwidth, etc.)

CPU Clocking

- Almost all computers are constructed using a clock that determines when events take place in the hardware



- Clock period (cycle): duration of a clock cycle (CC)
  - determines the speed of a computer processor
  - e.g., 250ps $= 0.25$ns $= 250 \times 10^{-12} s$
- Clock frequency or rate (CR): cycles per second
  - the inverse of the clock period
  - e.g., 3.0GHz $=$ 3000MHz $= 3.0 \times 10^{9}$Hz
- CR $=$ 1 / CC.

Performance Factors

- It is important to distinguish *elapsed time* and the *time spent on our task*
- CPU execution time (CPU time) - time the CPU spends working on a task
  - Does not include time waiting for I/O or running other programs

$$\begin{array}{ccccc} \text{CPU execution time} & = & \#\text{CPU clock cycles} & \times & \text{clock} - \text{cycle} \\ \text{for a program} & & \text{for a program} & & \end{array}$$

or

$$\begin{array}{ccccc} \text{CPU execution time} & = & \#\text{CPU clock cycles} & / & \text{clock} - \text{rate} \\ \text{for a program} & & \text{for a program} & & \end{array}$$

- Thus, we can improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program.

Instruction Performance

$$\begin{array}{c}\#\text{CPU clock cycles}\\ \text{for a program}\end{array} = \begin{array}{c}\#\text{Instructions}\\ \text{for a program}\end{array} \times \begin{array}{c}\text{Average \# of clock cycles}\\ \text{per instruction}\end{array}$$

- **Clock cycles per instruction (CPI)** - the average number of clock cycles each instruction takes to execute:
  - different instructions may take different amounts of time depending on what they do;
  - a way to compare two different implementations of the same instruction set architecture (ISA).

The Classic Performance Equation

$$\begin{aligned}\text{CPU time} &= \text{Instruction\_count} \times \text{CPI} \times \text{clock\_cycle} \\ &\text{or} \\ \text{CPU time} &= \text{Instruction\_count} \times \text{CPI} / \text{clock\_rate}\end{aligned}$$

- always Keep in mind that the only complete and reliable measure of computer performance is time.
- For example, redesigning the hardware implementation of an instruction set to lower the instruction count may lead to an organization with
  - a slower clock cycle time or,
  - higher CPI,

  that offsets the improvement in instruction count.
- Similarly, because CPI depends on the type of instruction executed, the code that executes the fewest number of instructions may not be the fastest.

A Simple Example (1/2)

$$\text{Overall effective CPI} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{IC}_i)$$

Op	Freq	CPI_i	Freq $\times$ CPI_i	(1)
ALU	50%	1	.5	.5
Load	20%	5	1.0	.4
Store	10%	3	.3	.3
Branch	20%	2	.4	.4
			$\sum = 2.2$	1.6

(1) How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?
CPU time new $= 1.6 \times \text{IC} \times \text{CC}$; so 2.2 versus 1.6 which means 37.5% faster

A Simple Example (2/2)

$$\text{Overall effective CPI} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{IC}_i)$$

Op	Freq	CPI_i	Freq $\times$ CPI_i	(2)	(3)
ALU	50%	1	.5	.5	.25
Load	20%	5	1.0	1.0	1.0
Store	10%	3	.3	.3	.3
Branch	20%	2	.4	.2	.4
			$\sum = 2.2$	2.0	1.95

(2) How does this compare with using branch prediction to save a cycle off the branch time?
CPU time new = $2.0 \times \text{IC} \times \text{CC}$ so 2.2 versus 2.0 means 10% faster

(3) What if two ALU instructions could be executed at once?
CPU time new = $1.95 \times \text{IC} \times \text{CC}$ so 2.2 versus 1.95 means 12.8% faster

Understanding Program Performance

$$\text{CPU time} = \text{Instruction\_count} \times \text{CPI} \times \text{clock\_cycle}$$

- The performance of a program depends on the algorithm, the language, the compiler, the architecture, and the actual hardware.

	Instruction_count	CPI	clock_cycle
Algorithm	X	X	
Programming language	X	X	
Compiler	X	X	
ISA	X	X	X
Processor organization		X	X

Performance Summary

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

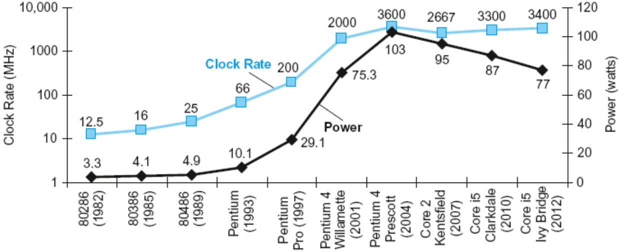
- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, T_c

Check Yourself

A given application written in Java runs 15 seconds on a desktop processor. A new Java compiler is released that requires only 0.6 as many instructions as the old compiler. Unfortunately, it increases the CPI by 1.1. How fast can we expect the application to run using this new compiler? Pick the right answer from the three choices below:

a. $\frac{15 \times 0.6}{1.1} = 8.2$ sec
b. $15 \times 0.6 \times 1.1 = 9.9$ sec
c. $\frac{15 \times 1.1}{0.6} = 27.5$ sec

Power Trends



- In complementary metal oxide semiconductor (CMOS) integrated circuit technology

$$\underset{(\times 30)}{\text{Power}} = \text{Capacitive load} \times \underset{(5V \rightarrow 1V)}{\text{Voltage}^2} \times \underset{(\times 1000)}{\text{Frequency switched}}$$

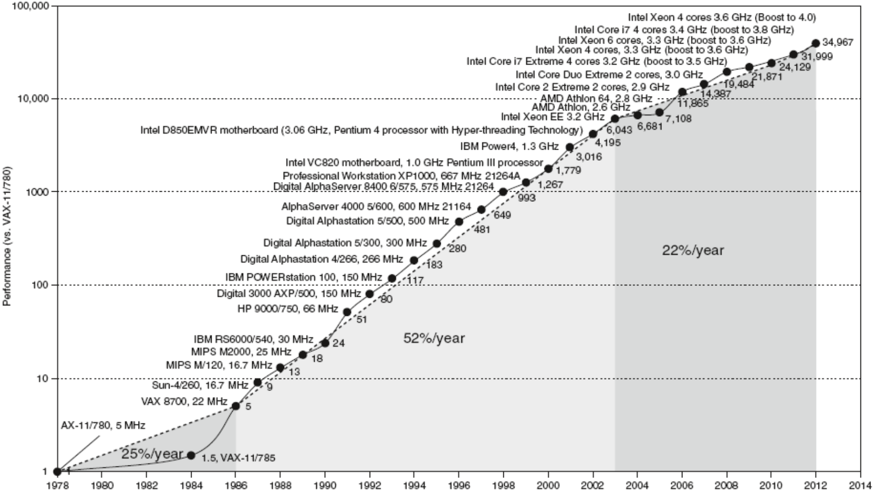
Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

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

- The power wall
  - 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

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
    - CINT2006 (integer) and CFP2006 (floating-point)

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

CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Profiling Tools

- Many profiling tools
  - `gprof` (static instrumentation)
  - `cachegrind`, `Dtrace` (dynamic instrumentation)
  - `perf` (performance counters)
- `perf` in `linux-tools`, based on event sampling
  - Keep a list of where "interesting events" (cycle, cache miss, etc) happen
  - CPU Feature: Counters for hundreds of events
    - Performance: Cache misses, branch misses, instructions per cycle, ...
  - Intel®64 and IA-32 Architectures Software Developer's Manual: Appendix A lists all counters `http://www.intel.com/products/processor/manuals/index.html`
  - `perf` user guide: `https://perf.wiki.kernel.org/index.php/Tutorial`

Exercise 1

```
void copymatrix1(int (*src)[n],
        int (*dst)[n], int n) {
  int i,j;
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      dst[i][j] = src[i][j];
}
```

```
void copymatrix2(int (*src)[n],
        int (*dst)[n], int n) {
  int i,j;
  for (j = 0; j < n; j++)
    for (i = 0; i < n; i++)
      dst[i][j] = src[i][j];
}
```

- `copymatrix1` vs `copymatrix2`
  - What do they do?
  - What is the difference?
  - Which one performs better? Why?
- `perf stat -e cycles -e cache-misses ./copymatrix1`
  `perf stat -e cycles -e cache-misses ./copymatrix2`
  - What does the output like?
  - How to interpret it?
  - Which program performs better?

Exercise 2

```
void lower1 (char* s) {
  int i;
  for (i = 0; i < strlen(s); i++)
    if (s[i]>='A' && s[i]<='Z')
      s[i] -= 'A'-'a';
}
```

```
void lower2 (char* s) {
  int i;
  int n = strlen(s);
  for (i = 0; i < n; i++)
    if (s[i]>='A' && s[i]<='Z')
      s[i] -= 'A'-'a';
}
```

- `lower1` vs `lower2`
  - What do they do?
  - What is the difference?
  - Which one performs better? Why?
- `perf stat -e cycles -e cache-misses ./lower1`
  `perf stat -e cycles -e cache-misses ./lower2`
  - What does the output like?
  - How to interpret it?
  - Which program performs better?