Recap: von Neumann Architecture

By loading different programs into memory, your computer can perform different functions.
Recap: How my “C code” becomes a “program”
Recap: How my “Java code” becomes a “program”
Recap: How my “Python code” becomes a “program” 

Interpreter (e.g., python)
Outline

• Definition of “Performance”
• What affects each factor in “Performance Equation”
• Instruction Set Architecture & Performance
Definition of “Performance”
What do you want for a computer?

- Latency/Execution time
- Frame rate
- Responsiveness
- Real-time
- Throughput
- Cost
- Volume
- Weight
- Battery life
- Low power/low temperature
- Reliability

The most direct measurement of performance
Execution Time

- The simplest kind of performance
- Shorter execution time means better performance
- Usually measured in seconds

```
120007a30: 0f00bb27 ldah gp,15(t12)
120007a34: 509cbd23 lda gp,-25520(gp)
120007a38: 00005d24 ldah t1,0(gp)
120007a3c: 0000bd24 ldah t4,0(gp)
120007a40: 2ca422a0 ldl t0,-23508(t1)
120007a44: 130020e4 beq t0,120007a94
120007a48: 00003d24 ldah t0,0(gp)
120007a4c: 2ca4e2b3 stl zero,-23508(t1)
120007a50: 0004ff47 clr v0
120007a54: 28a4e5b3 stl zero,-23512(t4)
120007a58: 20a421a4 ldq t0,-23520(t0)
120007a5c: 0e0020e4 beq t0,120007a98
120007a60: 0204e147 mov t0,t1
120007a64: 0304ff47 clr t2
120007a68: 0500e0c3 br 120007a80
```

How many of these?

How long is it take to execution each of these?

Cycles

Seconds

Instruction Cycle
CPU Performance Equation

**Performance** = \( \frac{1}{\text{Execution Time}} \)

**Execution Time** = \( \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}} \)

\( ET = IC \times CPI \times CT \)

1 GHz = \( 10^9 \text{Hz} = \frac{1}{10^9} \text{sec per cycle} = 1 \text{ ns per cycle} \)

\( 1 \text{GHz} = 10^9 \text{Hz} = \frac{1}{10^9} \text{sec per cycle} = 1 \text{ ns per cycle} \)

**Frequency** (i.e., clock rate)
Speedup

- The relative performance between two machines, X and Y. Y is $n$ times faster than X

\[ n = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y} \]

- The speedup of Y over X

\[ \text{Speedup} = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y} \]
What’s the limiting factor?
What Affects Each Factor in Performance Equation
Use “performance counters” to figure out!

- Modern processors provide performance counters
  - Instruction counts
  - Cache accesses/misses
  - Branch instructions/mispredictions
- How to get their values?
  - You may use “perf stat” in Linux
  - You may use Instruments -> Time Profiler on a Mac
  - Intel’s vtune — only works on Windows w/ Intel processors
  - You can also create your own functions to obtain counter values
Programmers can also set the cycle time

https://software.intel.com/sites/default/files/comment/1716807/how-to-change-frequency-on-linux-pub.txt

---

Subject: setting CPU speed on running linux system

If the OS is Linux, you can manually control the CPU speed by reading and writing some virtual files in the "/proc"

1.) Is the system capable of software CPU speed control? If the "directory" /sys/devices/system/cpu/cpu0/cpufreq exists, speed is controllable. If it does not exist, you may need to go to the BIOS and turn ON KINI and any other C and P state control and via...

2.) What speed is the box set to now? Do the following:
   $ cd /sys/devices/system/cpu
   $ cat /cpu0/cpufreq/cpuinfo_max_freq
   3143000
   $ cat /cpu0/cpufreq/cpuinfo_min_freq
   1256000

3.) What speeds can I set to? Do:
   $ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
   It will list highest settable to lowest; example from my i5-2400 "SNACKOVER" DX850 HEDT board, I see:
   3153000 3150000 3143000 2526000 2193000 2660000 2527300 2394300 2201000 2128000 1938000 1852600 1729600 159660
   You can choose from among those numbers to set the "high water" mark and "low water" mark for speed. If you set "h...

4.) Show me how to set all to highest settable speed!
   Use the following little.sh/bash script:
   $ cd /sys/devices/system/cpu # a virtual directory made visible by device drivers
   $ newSpeedTop=$(awk '{print $1}' /cpu0/cpufreq/scaling_available_frequencies)
   $ newSpeedLow=newSpeedTop # make them the same in this example
   $ for c in /cpu(0-9) ; do
     > echo $newSpeedTop >$c/cpufreq/scaling_max_freq
     > echo $newSpeedLow >$c/cpufreq/scaling_min_freq
     > done
   $ 

5.) How do I return to the default - i.e. allow machine to vary from highest to lowest?
   Edit line # 3 of the script above, and re-run it. Change the line:
   $ newSpeedLow=newSpeedTop # make them the same in this example
Recap: How my “C code” becomes a “program”

Source Code

Compiler (e.g., gcc)

Objects, Libraries

Program

Memory

Storage

Processor

Data

Instructions

One Time Cost!
Recap: How my "Java code" becomes a "program"

Instructions: One Time Cost!
Everytime when we run it!
Recap: How my “Python code” becomes a “program”

Everytime when we run it!
• gcc has different optimization levels.
  • -O0 — no optimizations
  • -O3 — typically the best-performing optimization

```c
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

```c
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```
How about “computational complexity”

• Algorithm complexity provides a good estimate on the performance if —
  • Every instruction takes exactly the same amount of time
  • Every operation takes exactly the same amount of instructions

These are unlikely to be true
Summary of CPU Performance Equation

\[
\text{Performance} = \frac{1}{\text{Execution Time}}
\]

\[
\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

\[
ET = IC \times CPI \times CT
\]

- IC (Instruction Count)
  - ISA, Compiler, algorithm, programming language, \text{programmer}
- CPI (Cycles Per Instruction)
  - Machine Implementation, microarchitecture, compiler, application, algorithm, programming language, \text{programmer}
- Cycle Time (Seconds Per Cycle)
  - Process Technology, microarchitecture, \text{programmer}
Instruction Set Architecture (ISA) & Performance
Recap: ISA — the interface b/w processor/software

• Operations
  • Arithmetic/Logical, memory access, control-flow (e.g., branch, function calls)
• Operands
  • Types of operands — register, constant, memory addresses
  • Sizes of operands — byte, 16-bit, 32-bit, 64-bit
• Memory space
  • The size of memory that programs can use
  • The addressing of each memory locations
  • The modes to represent those addresses
Popular ISAs
## MIPS v.s. x86

<table>
<thead>
<tr>
<th></th>
<th>RISC-V</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA type</td>
<td>Reduced Instruction Set Computers (RISC)</td>
<td>Complex Instruction Set Computers (CISC)</td>
</tr>
<tr>
<td>instruction width</td>
<td>32 bits</td>
<td>1 ~ 17 bytes</td>
</tr>
<tr>
<td>code size</td>
<td>larger</td>
<td>smaller</td>
</tr>
<tr>
<td>registers</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>addressing modes</td>
<td>reg+offset</td>
<td>base+offset</td>
</tr>
<tr>
<td></td>
<td></td>
<td>base+index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>scaled+index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>scaled+index+offset</td>
</tr>
<tr>
<td>hardware</td>
<td>simple</td>
<td>complex</td>
</tr>
</tbody>
</table>
Amdahl’s Law
Amdahl’s Law

\[
\text{Speedup}_{\text{enhanced}}(f, s) = \frac{1}{(1 - f) + \frac{f}{s}}
\]

- \(f\) — The fraction of time in the original program
- \(s\) — The speedup we can achieve on \(f\)

\[
\text{Speedup}_{\text{enhanced}} = \frac{\text{Execution Time}_{\text{baseline}}}{\text{Execution Time}_{\text{enhanced}}}
\]
Amdahl’s Law

\[ Speedup_{enhanced}(f, s) = \frac{1}{(1 - f) + \frac{f}{s}} \]

Execution Time_{baseline} = 1

Execution Time_{enhanced} = (1-f) + f/s

\[ Speedup_{enhanced} = \frac{Execution Time_{baseline}}{Execution Time_{enhanced}} = \frac{1}{(1 - f) + \frac{f}{s}} \]
We can apply Amdahl’s law for multiple optimizations

- These optimizations must be dis-joint!
  - If optimization #1 and optimization #2 are dis-joint:

  \[
  \text{Speedup}_{\text{enhanced}}(f_{\text{Opt1}}, f_{\text{Opt2}}, s_{\text{Opt1}}, s_{\text{Opt2}}) = \frac{1}{(1 - f_{\text{Opt1}} - f_{\text{Opt2}}) + \frac{f_{\text{Opt1}}}{s_{\text{Opt1}}} + \frac{f_{\text{Opt2}}}{s_{\text{Opt2}}}}
  \]

  - If optimization #1 and optimization #2 are not dis-joint:

  \[
  \text{Speedup}_{\text{enhanced}}(f_{\text{OnlyOpt1}}, f_{\text{OnlyOpt2}}, f_{\text{BothOpt1Opt2}}, s_{\text{OnlyOpt1}}, s_{\text{OnlyOpt2}}, s_{\text{BothOpt1Opt2}}) = \frac{1}{(1 - f_{\text{OnlyOpt1}} - f_{\text{OnlyOpt2}} - f_{\text{BothOpt1Opt2}}) + \frac{f_{\text{BothOpt1Opt2}}}{s_{\text{BothOpt1Opt2}}} + \frac{f_{\text{OnlyOpt1}}}{s_{\text{OnlyOpt1}}} + \frac{f_{\text{BothOpt1Opt2}}}{s_{\text{OnlyOpt2}}} + \frac{f_{\text{OnlyOpt2}}}{s_{\text{OnlyOpt2}}}}
  \]
Amdahl’s Law Corollary #1

• The maximum speedup is bounded by

\[
\text{Speedup}_{\text{max}}(f, \infty) = \frac{1}{(1 - f) + \frac{f}{\infty}}
\]

\[
\text{Speedup}_{\text{max}}(f, \infty) = \frac{1}{(1 - f)}
\]
Corollary #1 on Multiple Optimizations

- If we can pick just one thing to work on/optimize

\[
\begin{align*}
\text{Speedup}_{\text{max}}(f_1, \infty) &= \frac{1}{1 - f_1} \\
\text{Speedup}_{\text{max}}(f_2, \infty) &= \frac{1}{1 - f_2} \\
\text{Speedup}_{\text{max}}(f_3, \infty) &= \frac{1}{1 - f_3} \\
\text{Speedup}_{\text{max}}(f_4, \infty) &= \frac{1}{1 - f_4}
\end{align*}
\]

The biggest \( f_x \) would lead to the largest \( \text{Speedup}_{\text{max}} \)!
Corollary #2 — make the common case fast!

• When f is small, optimizations will have little effect.
• Common == **most time consuming** not necessarily the most frequent
• The uncommon case doesn’t make much difference
• The common case can change based on inputs, compiler options, optimizations you’ve applied, etc.
Identify the most time consuming part

• Compile your program with -pg flag
• Run the program
  • It will generate a gmon.out
  • gprof your_program gmon.out > your_program.prof
• It will give you the profiled result in your_program.prof
If we repeatedly optimizing our design based on Amdahl’s law...

- With optimization, the common becomes uncommon.
- An uncommon case will (hopefully) become the new common case.
- Now you have a new target for optimization.
- — You have to revisit “Amdahl’s Law” every time you applied some optimization.
Don’t hurt non-common part too mach

• If the program spend 90% in A, 10% in B. Assume that an optimization can accelerate A by 9x, by hurts B by 10x...

• Assume the original execution time is $T$. The new execution time

$$ET_{\text{new}} = \frac{ET_{\text{old}} \times 90\%}{9} + ET_{\text{old}} \times 10\% \times 10$$

$$ET_{\text{new}} = 1.1 \times ET_{\text{old}}$$

$$Speedup = \frac{ET_{\text{old}}}{ET_{\text{new}}} = \frac{ET_{\text{old}}}{1.1 \times ET_{\text{old}}} = 0.91 \times \ldots \text{slowdown!}$$

You may not use Amdahl’s Law for this case as Amdahl’s Law does NOT
(1) consider overhead
(2) bound to slowdown
Corollary #3, Corollary #4 & Corollary #5

\[
\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, \infty) = \frac{1}{(1 - f_{\text{parallelizable}}) + \frac{f_{\text{parallelizable}}}{\infty}}
\]

\[
\text{Speedup}_{\text{parallel}}(f_{\text{parallelizable}}, \infty) = \frac{1}{(1 - f_{\text{parallelizable}})}
\]

- Single-core performance still matters — it will eventually dominate the performance
- Finding more “parallelizable” parts is also important
- If we can build a processor with unlimited parallelism — the complexity doesn’t matter as long as the algorithm can utilize all parallelism — that’s why bitonic sort works!
“Fair” Comparisons
TFLOPS (Tera FLoating-point Operations Per Second)
TFLOPS (Tera FLoating-point Operations Per Second)

- TFLOPS does not include instruction count!
  - Cannot compare different ISA/compiler
  - Different CPI of applications, for example, I/O bound or computation bound
  - If new architecture has more IC but also lower CPI?

<table>
<thead>
<tr>
<th></th>
<th>TFLOPS</th>
<th>clock rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>XBOX One X</td>
<td>6</td>
<td>1.75 GHz</td>
</tr>
<tr>
<td>PS4 Pro</td>
<td>4</td>
<td>1.6 GHz</td>
</tr>
<tr>
<td>GeForce GTX 1080</td>
<td>8.228</td>
<td>3.5 GHz</td>
</tr>
</tbody>
</table>
Is TFLOPS (Tera FLoating-point Operations Per Second) a good metric?

\[
\text{TFLOPS} = \frac{\text{# of floating point instructions} \times 10^{-12}}{\text{Execution Time}}
\]

\[
= \frac{\text{IC} \times \% \text{ of floating point instructions} \times 10^{-12}}{\text{IC} \times \text{CPI} \times \text{CT}}
\]

\[
= \frac{\% \text{ of floating point instructions} \times 10^{-12}}{\text{CPI} \times \text{CT}}
\]

IC is gone!

- Cannot compare different ISA/compiler
  - What if the compiler can generate code with fewer instructions?
  - What if new architecture has more IC but also lower CPI?
- Does not make sense if the application is not floating point intensive
They try to tell it’s the better AI hardware


<table>
<thead>
<tr>
<th></th>
<th>K80 2012</th>
<th>TPU 2015</th>
<th>P40 2016</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inferences/Sec &lt;10ms latency</td>
<td>$\frac{1}{13}X$</td>
<td>1X</td>
<td>2X</td>
</tr>
<tr>
<td>Training TOPS</td>
<td>6 FP32</td>
<td>NA</td>
<td>12 FP32</td>
</tr>
<tr>
<td>Inference TOPS</td>
<td>6 FP32</td>
<td>90 INT8</td>
<td>48 INT8</td>
</tr>
<tr>
<td>On-chip Memory</td>
<td>16 MB</td>
<td>24 MB</td>
<td>11 MB</td>
</tr>
<tr>
<td>Power</td>
<td>300W</td>
<td>75W</td>
<td>250W</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>320 GB/S</td>
<td>34 GB/S</td>
<td>350 GB/S</td>
</tr>
</tbody>
</table>
Extreme Multitasking Performance

- Dual 4K external monitors
- 1080p device display
- 7 applications
What’s missing in this video clip?

- The ISA of the “competitor”
- Clock rate, CPU architecture, cache size, how many cores
- How big the RAM?
- How fast the disk?
Choose the right metric — Latency v.s. Throughput/Bandwidth
Latency v.s. Bandwidth/Throughput

- Latency — the amount of time to finish an operation
  - access time
  - response time
- Throughput — the amount of work can be done within a given period of time
  - bandwidth (MB/Sec, GB/Sec, Mbps, Gbps)
  - IOPs
  - MFLOPs
RAID — Improving throughput

Access time: 10 ms
Bandwidth: 125 MB/sec

Aggregated Bandwidth: 500 MB/sec
## Latency/Delay v.s. Throughput

<table>
<thead>
<tr>
<th>Toyota Prius</th>
<th>100 Gb Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>● 100 miles (161 km) from UCSD</td>
<td>● 100 miles (161 km) from UCSD</td>
</tr>
<tr>
<td>● 75 MPH on highway!</td>
<td>● Lightspeed! — $3 \times 10^8$ m/sec</td>
</tr>
<tr>
<td>● Max load: 374 kg = 2,770 hard drives (2TB per drive)</td>
<td>● Max load: 4 lanes operating at 25GHz</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>bandwidth</strong></td>
<td><strong>100 Gb/s or</strong></td>
</tr>
<tr>
<td></td>
<td><strong>12.5 GB/sec</strong></td>
</tr>
<tr>
<td><strong>290 GB/sec</strong></td>
<td></td>
</tr>
<tr>
<td><strong>latency</strong></td>
<td><strong>2 Peta-byte over 167772 seconds</strong></td>
</tr>
<tr>
<td></td>
<td>$= 1.94$ Days</td>
</tr>
<tr>
<td><strong>3.5 hours</strong></td>
<td><strong>You can start watching the movie as soon as you get a frame!</strong></td>
</tr>
<tr>
<td><strong>response time</strong></td>
<td></td>
</tr>
<tr>
<td>You see nothing in the first 3.5 hours</td>
<td></td>
</tr>
</tbody>
</table>
Announcements

• Login piazza, Canvas
• Check our website
• Reading quiz due tomorrow before class
• Assignment #1 due next Monday