Performance (II)

Hung-Wei Tseng
Recap: von Neumann Architecture
Recap: 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 = \( 10^9 \text{Hz} = \frac{1}{10^9}\ \text{sec per cycle} = 1\ \text{ns per cycle} \)

\( Frequency \ (i.e., \ \text{clock rate}) \)

\[ 1 \]
Recap: Execution Time

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

Instructions Program

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

How many of these?

How long is it take to execution each of these?

Cycles/ Instruction × Seconds/ Cycle
Recap: 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

\[
\text{Speedup} = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y}
\]
Recap: Speedup of Y over X

Consider the same program on the following two machines, X and Y. By how much Y is faster than X?

<table>
<thead>
<tr>
<th>Machine</th>
<th>Clock Rate</th>
<th>Instructions</th>
<th>Percentage of Type-A Insts.</th>
<th>CPI of Type-A Insts.</th>
<th>Percentage of Type-B Insts.</th>
<th>CPI of Type-B Insts.</th>
<th>Percentage of Type-C Insts.</th>
<th>CPI of Type-C Insts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>3 GHz</td>
<td>50000000000</td>
<td>20%</td>
<td>8</td>
<td>20%</td>
<td>4</td>
<td>60%</td>
<td>1</td>
</tr>
<tr>
<td>Y</td>
<td>5 GHz</td>
<td>50000000000</td>
<td>20%</td>
<td>13</td>
<td>20%</td>
<td>4</td>
<td>60%</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
ET_Y = (5 \times 10^9) \times (20\% \times 13 + 20\% \times 4 + 60\% \times 1) \times \frac{1}{5 \times 10^{-9}} \text{ sec} = 4
\]

\[
\text{Speedup} = \frac{\text{Execution Time}_X}{\text{Execution Time}_Y} = \frac{5}{4} = 1.25
\]

- A. 0.2
- B. 0.25
- C. 0.8
- D. 1.25
- E. No changes
Outline

• What/who affects each factor in “Performance Equation”
• Amdahl’s Law and Performance
Recap: How programmer affects performance?

• Performance equation consists of the following three factors
  ① IC
  ② CPI
  ③ CT

How many can a *programmer* affect?

A. 0
B. 1
C. 2
D. 3
By adding the “sort” in the following code snippet, what the programmer changes in the performance equation to achieve better performance?

```cpp
std::sort(data, data + arraySize);

for (unsigned c = 0; c < arraySize*1000; ++c) {
    if (data[c % arraySize] >= INT_MAX/2)
        sum ++;
}
```

A. CPI
B. IC — we increased IC, suppose to make the performance worse
C. CT
D. IC & CPI
What Affects Each Factor in Performance Equation
for (i = 0; i < ARRAY_SIZE; i++)
{
    for (j = 0; j < ARRAY_SIZE; j++)
    {
        c[i][j] = a[i][j] + b[i][j];
    }
}

for (j = 0; j < ARRAY_SIZE; j++)
{
    for (i = 0; i < ARRAY_SIZE; i++)
    {
        c[i][j] = a[i][j] + b[i][j];
    }
}
How many of the following make(s) the performance of one better than the other between version A & version B?

A. 0  
B. 1  
C. 2  
D. 3

Demo — programmer & performance

```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 many of the following make(s) the performance of one better than the other between version A & version B?

A. 0
B. 1
C. 2
D. 3

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

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

\(O(n^2)\)

Complexity

Instruction Count?

Clock Rate

CPI

Same

Same

???
Use “performance counters” to figure out!

- Modern processors provides performance counters
  - instruction counts
  - cache accesses/misses
  - branch instructions/mis-predictions
- 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
Demo — programmer & performance

```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];
    }
}
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Instruction Count?</td>
<td>Same</td>
<td>Same</td>
</tr>
<tr>
<td>Clock Rate</td>
<td>Same</td>
<td>Same</td>
</tr>
<tr>
<td>CPI</td>
<td>Better</td>
<td>Worse</td>
</tr>
</tbody>
</table>
• How many of the following make(s) the performance of one better than the other between version A & version B?

   ① IC
   ② CPI
   ③ CT

A. 0
B. 1
C. 2
D. 3
Demo — Programmers can also set the cycle time

- Intel processors expose their “capabilities” to the system through files in /sys/devices/system/cpu/
- cpufreq/ contains the available frequencies
- Since it's a file...
- Different processors have different interfaces — manufacturer/architecture dependent
How programmer affects performance?

- Performance equation consists of the following three factors:
  - IC
  - CPI
  - CT

How many can a **programmer** affect?

A. 0
B. 1
C. 2
D. 3
How programming languages affect performance

• Performance equation consists of the following three factors
  ① IC
  ② CPI
  ③ CT

How many can the programming language affect?

A. 0
B. 1
C. 2
D. 3
How programming languages affect performance

• Performance equation consists of the following three factors
  ① IC
  ② CPI
  ③ CT

How many can the **programming language** affect?
A. 0
B. 1
C. 2
D. 3
Programming languages

• Which of the following programming language needs to highest instruction count to print “Hello, world!” on screen?
  A. C
  B. C++
  C. Java
  D. Perl
  E. Python
# Programming languages

- How many instructions are there in “Hello, world!”

<table>
<thead>
<tr>
<th></th>
<th>Instruction count</th>
<th>LOC</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>600k</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>C++</td>
<td>3M</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Java</td>
<td>~210M</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Perl</td>
<td>10M</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Python</td>
<td>~30M</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Which of the following programming language needs to highest instruction count to print “Hello, world!” on screen?

A. C  
B. C++  
C. Java  
D. Perl  
E. Python

Programming languages
Recap: How my “C code” becomes a “program”
Recap: How my “Java code” becomes a “program”

Source Code

Compiler (e.g., javac)

Java Virtual Machine (e.g., java)

Other (.class)

Jave Bytecode (.class)

00c2e800 00000008
00c2f000 00000008
00c2f800 00000008
00c30000 00000008
cafebabe 00000033
06000f09 00100011
0800120a 00130014
07001507

Instructions

Memory

Processor

0f00bb27 509cbd23 00005d24 0000d24 20e22a0 13e020e4 0003d24 20ca4e2b3
cafebabe 00c2e800 00c2f000 00c2f800 00c30000
00000008 00000008 00000008 00000008

Everytime when we run it!

One Time Cost!

00c2e800 00000008
001d0a00 06000f09 00100011
0800120a 00130014
07001507

cafebabe 00000033
00000008
00000008
00000008
00000008
00000008
00000008

Recap: How my “Python code” becomes a “program”

Everytime when we run it!
How programming languages affect performance

• Performance equation consists of the following three factors:
  1. IC
  2. CPI
  3. CT

How many can the **programming language** affect?

A. 0
B. 1
C. 2
D. 3
How compilers affect performance

- Performance equation consists of the following three factors
  1. IC
  2. CPI
  3. CT

How many can the compiler affect?

A. 0
B. 1
C. 2
D. 3
How compilers affect performance

- Performance equation consists of the following three factors
  1. IC
  2. CPI
  3. CT

How many can the **compiler** affect?

A. 0
B. 1
C. 2
D. 3
How compilers affect performance

• Performance equation consists of the following three factors
  ① IC
  ② CPI
  ③ CT

How many can the compiler affect?
A. 0
B. 1
C. 2
D. 3

C. 2
Revisited the demo with compiler optimizations!

- 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];
    }
}
```
Demo revisited — compiler optimization

- Compiler can reduce the instruction count, change CPI — with **limited capabilities**
- Compiler CANNOT help improving “crummy” source code

```cpp
if(option)
    std::sort(data, data + arraySize);

Compiler can never add this — only the programmer can!
```
```cpp
for (unsigned c = 0; c < arraySize*1000; ++c) {
    if (data[c%arraySize] >= INT_MAX/2)
        sum ++;
}
```
How about “computational complexity”

- Algorithm complexity provides a good estimate on the performance if —
  - Every line of pseudo code takes exactly the same amount of instructions (IC)
  - Every instruction takes exactly the same amount of time (CPI)

These are unlikely to be true
Instruction Set Architecture (ISA) & Performance
# 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>
Summary of CPU Performance Equation

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

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

\[
\text{ET} = IC \times CPI \times CT
\]

- IC (Instruction Count)
  - ISA, Compiler, algorithm, programming language, programmer
- CPI (Cycles Per Instruction)
  - Machine Implementation, microarchitecture, compiler, application, algorithm, programming language, programmer
- Cycle Time (Seconds Per Cycle)
  - Process Technology, microarchitecture, programmer
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

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

Execution Time_{\text{baseline}} = 1

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

\[ \text{Speedup}_{\text{enhanced}} = \frac{\text{Execution Time}_{\text{baseline}}}{\text{Execution Time}_{\text{enhanced}}} = \frac{1}{(1 - f) + \frac{f}{s}} \]
Assume that we have an application composed with a total of 500000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest instructions are integer instructions with average CPI of 1 cycle when using a 2GHz processor.

If we double the CPU clock rate to 4GHz that helps to accelerate all instructions by 2x except that load/store instruction cannot be improved — their CPI will become 12 cycles. What’s the performance improvement after this change?

A. No change
B. 1.25
C. 1.5
D. 2
E. None of the above
• Assume that we have an application composed with a total of 500000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest instructions are integer instructions with average CPI of 1 cycle when using a 2GHz processor.

• If we double the CPU clock rate to 4GHz that helps to accelerate all instructions by 2x except that load/store instruction cannot be improved — their CPI will become 12 cycles. What’s the performance improvement after this change?

A. No change
B. 1.25
C. 1.5
D. 2
E. None of the above
Assume that we have an application composed with a total of 500000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest instructions are integer instructions with average CPI of 1 cycle when using a 2GHz processor.

If we double the CPU clock rate to 4GHz that helps to accelerate all instructions by 2x except that load/store instruction cannot be improved — their CPI will become 12 cycles. What’s the performance improvement after this change?

A. No change
B. 1.25
C. 1.5
D. 2
E. None of the above

Recap: Speedup

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

\[ ET_{baseline} = (5 \times 10^5) \times (20\% \times 6 + 80\% \times 1) \times \frac{1}{2 \times 10^{-9}} \text{sec} = 5^{-3} \]

\[ ET_{enhanced} = (5 \times 10^5) \times (20\% \times 12 + 80\% \times 1) \times \frac{1}{4 \times 10^{-9}} \text{sec} = 4^{-3} \]

\[ \text{Speedup} = \frac{ET_{baseline}}{ET_{enhanced}} = \frac{5}{4} = 1.25 \]
Replay using Amdahl’s Law

• Assume that we have an application composed with a total of 500000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest instructions are integer instructions with average CPI of 1 cycle when using a 2GHz processor.

• If we double the CPU clock rate to 4GHz that helps to accelerate all instructions by 2x except that load/store instruction cannot be improved — their CPI will become 12 cycles. What’s the performance improvement after this change?

How much time in load/store?  
\[ 500000 \times (0.2 \times 6) \times 0.5 \text{ ns} = 300000 \text{ ns} \rightarrow 60\% \]

How much time in the rest?  
\[ 500000 \times (0.8 \times 1) \times 0.5 \text{ ns} = 200000 \text{ ns} \rightarrow 40\% \]

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

\[ \text{Speedup}_{\text{enhanced}}(40\%, 2) = \frac{1}{(1 - 40\%) + \frac{40\%}{2}} = 1.25 \times \]
Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time. By how much can we speed up the map loading process?

A. ~7x  
B. ~10x  
C. ~17x  
D. ~29x  
E. ~100x
Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time. By how much can we speed up the map loading process?

A. ~7x  
B. ~10x  
C. ~17x  
D. ~29x  
E. ~100x
Practicing Amdahl’s Law

Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time. By how much can we speed up the map loading process?

A. ~7x

B. ~10x

C. ~17x

D. ~29x

E. ~100x

\[ Speedup_{\text{enhanced}}(95\%, 100) = \frac{1}{(1 - 95\%) + \frac{95\%}{100}} = 16.81 \times \]
Amdahl’s Law on Multiple Optimizations

- 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:
Announcements

• Assignment #1 is up!
  • Due “tonight”
  • We’ve moved the TA’s office hours to Monday morning to help you!

• Reading quizzes
  • Due tomorrow

• Resources
  • Ask questions — piazza
  • Reading quizzes, turning in assignments — Canvas
  • Slides, schedule, assignment questions — Check our website
  • Video archive — Prof. Usagi’s Youtube channel
Computer Science & Engineering