Reasoning about Computer Performance

Jason Mars
What Do We Want in our Computers

- Frame rate
- Responsiveness
- Real-time
- Throughput
- Latency/Execution time
- Battery life
- Low power/low temperature
What Do We Want in our Computers

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

Performance!!!!
Performance grows at exponential rate
Performance Trends

Specint2000

Apple
Intel
Alpha
Sparc
Mips
HP PA
Power PC
AMD

Tuesday, January 22, 13
Performance Trends

But *what is performance?*
What is Performance

<table>
<thead>
<tr>
<th>Car</th>
<th>Time to Bay Area</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferrari</td>
<td>3.1 hours</td>
<td>160 mph</td>
<td>2</td>
<td>320</td>
</tr>
<tr>
<td>Greyhound</td>
<td>7.7 hours</td>
<td>65 mph</td>
<td>60</td>
<td>3900</td>
</tr>
</tbody>
</table>
What is Performance

- Time to do the task
  - execution time, response time, latency
- Tasks per day, hour, week, sec, ns...
  - throughput, bandwidth

<table>
<thead>
<tr>
<th>Car</th>
<th>Time to Bay Area</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ferrari</td>
<td>3.1 hours</td>
<td>160 mph</td>
<td>2</td>
<td>320</td>
</tr>
<tr>
<td>Greyhound</td>
<td>7.7 hours</td>
<td>65 mph</td>
<td>60</td>
<td>3900</td>
</tr>
</tbody>
</table>
How to Measure Execution Time?

```bash
% time program
... program results ...
90.7u 12.9s 2:39 65%
%
```
How to Measure Execution Time?

• Wall-clock time?
• user CPU time?
• user + kernel CPU time?

• **Answer:**

```
% time program
... program results ...
90.7u 12.9s 2:39 65%
```
How to Measure Execution Time?

• Wall-clock time?
• user CPU time?
• user + kernel CPU time?

• Answer: 103.6s
But What is Performance

\[
\text{Performance}_X = \frac{1}{\text{Execution Time}_X}, \text{ for program } X
\]
But What is Performance

\[ \text{Performance}_X = \frac{1}{\text{Execution Time}_X}, \text{ for program } X \]

- only has meaning in the context of a program or workload
But What is Performance

\[ \text{Performance}_X = \frac{1}{\text{Execution Time}_X} \text{, for program } X \]

- only has meaning in the context of a program or workload
- Not very intuitive as an absolute measure, but most of the time we’re more interested in relative performance.
Relative Performance

• Can be confusing
  
  • A runs in 12 seconds
  • B runs in 20 seconds
  
  • $A/B = 0.6$, so A is 40% faster, or 1.4X faster, or B is 40% slower
  • $B/A = 1.67$, so A is 67% faster, or 1.67X faster, or B is 67% slower

• For the future scientists
  
  • Must use GEOMEAN for performance!

• Needs a precise definition
Defining Speedup (Relative Performance)

\[
\text{Speedup } \ (X/Y) = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n
\]
Example

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?
Example

Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[ \text{Speedup } \left( \frac{X}{Y} \right) = \]

Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?
Example

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

  \[ \text{Speedup} = \frac{X}{Y} = \frac{9}{6} \]

  \[ X = \text{machine B} \]
  \[ Y = \text{machine A} \]

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?
Example

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[
\text{Speedup } \frac{X}{Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = \frac{9}{6} = 1.5x
\]

\(X = \text{machine B}\)
\(Y = \text{machine A}\)

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?
Example

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[ \text{Speedup } \left( \frac{X}{Y} \right) = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = \frac{9}{6} = 1.5x \]

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

\[ \text{Speedup } \left( \frac{X}{Y} \right) = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = \frac{3}{6} = 0.5x \]
Example

• Machine A runs program C in 9 seconds, Machine B runs the same program in 6 seconds. What is the speedup we see if we move to Machine B from Machine A?

\[
\text{Speedup } (X/Y) = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = \frac{9}{6} = 1.5x
\]

\[X = \text{machine B} \]
\[Y = \text{machine A}\]

• Machine B gets a new compiler, and can now run the program in 3 seconds. Speedup?

\[
\text{Speedup } (X/Y) = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = \frac{9}{3} = 3x
\]

\[X = \text{machine B} \]
\[Y = \text{machine A}\]
What is Time?

- CPU Execution Time = CPU clock cycles * Clock cycle time
  - Every conventional processor has a clock with an associated clock cycle time or clock rate
  - Every program runs in an integral number of clock cycles

- Cycle Time
  - MHz = millions of cycles/second, GHz = billions of cycles/second
  - X MHz = 1000/X nanoseconds cycle time
  - Y GHz = 1/Y nanoseconds cycle time
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
- Usually measured in seconds
- Shorter execution time means better performance
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
- Usually measured in seconds
- Shorter execution time means better performance

```
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: 28a4e5b3 stl zero,-23512(t4)
120007a54: 20a421a4 ldq t0,-23520(t0)
120007a58: 0e0020e4 beq t0,120007a98
120007a5c: 0204e147 mov t0,t1
120007a60: 0304ff47 clr t2
120007a64: 0500e0c3 br 120007a80
```
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
- Usually measured in seconds
- Shorter execution time means better performance

```
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: 28a4e5b3 stl zero,-23512(t4)
120007a54: 20a421a4 ldq t0,-23520(t0)
120007a58: 0e0020e4 beq t0,120007a98
120007a5c: 0204e147 mov t0,t1
120007a60: 0304ff47 clr t2
120007a64: 0500e0c3 br 120007a80
```

Tuesday, January 22, 13
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
- Usually measured in seconds
- Shorter execution time means better performance
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
- Usually measured in seconds
- Shorter execution time means better performance

```
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
```
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
- Usually measured in seconds
- Shorter execution time means better performance

```
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
```

```
processor
PC

How many of these?

instruction memory
```
Reasoning About Execution Time via Cycles

• Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
• Usually measured in seconds
• Shorter execution time means better performance

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 ld1 t0,-23508(t1)
120007a44: 130020e4 beq t0,120007a94
120007a48: 00003d24 ldah t0,0(gp)
120007a4c: 2ca4e2b3 stl zero,-23508(t1)
120007a50: 28a4e5b3 stl zero,-23512(t4)
120007a54: 20a421a4 ldq t0,-23520(t0)
120007a58: 0e0020e4 beq t0,120007a98
120007a5c: 0204e147 mov t0,t1
120007a60: 0304ff47 clr t2
120007a64: 0500e0c3 br 120007a80

How many of these?
How long is it take to execution each of these?
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
- Usually measured in seconds
- Shorter execution time means better performance

<table>
<thead>
<tr>
<th>Instruction Memory</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>120007a30: 0f00bb27</td>
<td>ldah gp,15(t12)</td>
</tr>
<tr>
<td>120007a34: 509cbd23</td>
<td>lda gp,-25520(gp)</td>
</tr>
<tr>
<td>120007a38: 00005d24</td>
<td>ldah t1,0(gp)</td>
</tr>
<tr>
<td>120007a3c: 0000bd24</td>
<td>ldah t4,0(gp)</td>
</tr>
<tr>
<td>120007a40: 2ca422a0</td>
<td>ld1 t0,-23508(t1)</td>
</tr>
<tr>
<td>120007a44: 130020e4</td>
<td>beq t0,120007a94</td>
</tr>
<tr>
<td>120007a48: 00003d24</td>
<td>ldah t0,0(gp)</td>
</tr>
<tr>
<td>120007a4c: 2ca4e2b3</td>
<td>stl zero,-23508(t1)</td>
</tr>
<tr>
<td>120007a50: 0004ff47</td>
<td>clr v0</td>
</tr>
<tr>
<td>120007a54: 28a4e5b3</td>
<td>stl zero,-23512(t4)</td>
</tr>
<tr>
<td>120007a58: 20a421a4</td>
<td>ldq t0,-23520(t0)</td>
</tr>
<tr>
<td>120007a5c: 0e0020e4</td>
<td>beq t0,120007a98</td>
</tr>
<tr>
<td>120007a60: 0204e147</td>
<td>mov t0,t1</td>
</tr>
<tr>
<td>120007a64: 0304ff47</td>
<td>clr t2</td>
</tr>
<tr>
<td>120007a68: 0500e0c3</td>
<td>br 120007a80</td>
</tr>
</tbody>
</table>
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
- Usually measured in seconds
- Shorter execution time means better performance

How long is it take to execution each of these?
Cycles per instruction * cycle time

How many of these?
Instruction Count!
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)
Reasoning About Execution Time via Cycles

• Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

• Computer A runs program C in 3.6 billion cycles. Program C consists of 2 billion dynamic instructions. What is the CPI?
Reasoning About Execution Time via Cycles

• Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

• Computer A runs program C in 3.6 billion cycles. Program C consists of 2 billion dynamic instructions. What is the CPI?

• $1.8 \text{ cyc}$
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

- Computer A runs program C in 3.6 billion cycles. Program C consists of 2 billion dynamic instructions. What is the CPI?
  - 1.8 cyc

- A computer is running a program with CPI = 2.0, and executes 24 million instructions, how long will it run?
Reasoning About Execution Time via Cycles

- Number of CPU cycles = Instructions executed * Average Clock Cycles per Instruction (CPI)

- Computer A runs program C in 3.6 billion cycles. Program C consists of 2 billion dynamic instructions. What is the CPI?
  - **1.8 cyc**

- A computer is running a program with CPI = 2.0, and executes 24 million instructions, how long will it run?
  - **48 mil. cyc**
Performance Equation

Execution Time =

How many instruction executed?

How long is it to execute each instruction
Performance Equation

Execution Time = \frac{\text{Instructions}}{\text{Program}}

- How many instructions executed?
- How long is it to execute each instruction?
Performance Equation

Execution Time = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}}

- How many instruction executed?
- How long is it to execute each instruction?
Performance Equation

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

How many instruction executed?

How long is it to execute each instruction?
Performance Equation

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

- How many instructions executed?
- How long is it to execute each instruction?

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

- How many instructions executed?
- How long is it to execute each instruction?

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

- \text{IC} (Instruction Count)
Performance Equation

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

- How many instruction executed?
- How long is it to execute each instruction

- ET = IC \times CPI \times CT
- IC (Instruction Count)
- CPI (Cycles Per Instruction)
Performance Equation

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

- **ET** = Execution Time
- **IC** (Instruction Count)
- **CPI** (Cycles Per Instruction)
- **CT** (Seconds Per Cycle)

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

- How many instructions executed?
- How long is it to execute each instruction?
Performance Equation

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

- How many instruction executed?
- How long is it to execute each instruction?

- \( ET = IC \times CPI \times CT \)
- IC (Instruction Count)
- CPI (Cycles Per Instruction)
- CT (Seconds Per Cycle)
  - 1 Hz = 1 second per cycle; 1 GHz = 1 ns per cycle
Computer Performance...
Performance Equation

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

- How many instruction executed?
- How long is it to execute each instruction?

- ET = IC * CPI * CT
- IC (Instruction Count)
- CPI (Cycles Per Instruction)
- CT (Seconds Per Cycle)
  - 1 Hz = 1 second per cycle; 1 GHz = 1 ns per cycle
Speedup

\[
\text{Speedup } (X/Y) = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n
\]
Speedup

\[
\text{Speedup} \ (X/Y) = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n
\]

\[
\text{Speedup} = \frac{\text{Execution time}_\text{baseline}}{\text{Execution time}_\text{improved system}}
\]
Speedup

\[ \text{Speedup (X/Y)} = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n \]

\[ \text{Speedup} = \frac{\text{Execution time baseline}}{\text{Execution time improved system}} \]

- Speedup when moving to the faster system (>1x)
  - Execution time of \textit{slower} system is in the \textbf{numerator}
- Speedup when moving to the slower system (<1x)
  - Execution time of \textit{faster} system is in the \textbf{numerator}
Performance Example

- Assume that we have an application composed with a total of 500,000 instructions, the average CPI of the application is 2 cycle. If the processor runs at 1GHz, how long is it take to execute the application?

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

• Assume that we have an application composed with a total of 500,000 instructions, the average CPI of the application is 2 cycle. If the processor runs at 1GHz, how long is it take to execute the application?

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

\[
500,000 \times *\]
Performance Example

• Assume that we have an application composed with a total of 500,000 instructions, the average CPI of the application is 2 cycle. If the processor runs at 1GHz, how long is it take to execute the application?

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

\[
500,000 \times 2 \times 1 \text{GHz}
\]
Performance Example

• Assume that we have an application composed with a total of 500,000 instructions, the average CPI of the application is 2 cycle. If the processor runs at 1GHz, how long is it take to execute the application?

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

\[
500,000 \times 2 \times 1 \text{ ns} =
\]
Performance Example

• Assume that we have an application composed with a total of 500,000 instructions, the average CPI of the application is 2 cycle. If the processor runs at 1GHz, how long is it take to execute the application?

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

\[
500,000 \times 2 \times 1 \text{ ns} = 1,000,000 \text{ ns}
\]
Slightly More Interesting Example...

- Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

\[
\text{Execution Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]
Slightly More Interesting Example...

• Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

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

\[
500,000 *
\]
Slightly More Interesting Example...

- Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

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

\[
500,000 \times (0.2 \times 6 + 0.8 \times 1)
\]
Slightly More Interesting Example...

- Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

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

\[
500,000 \times (.2 \times 6 + .8 \times 1) \times 1 \text{ ns} =
\]
Slightly More Interesting Example...

- Assume that we have an application composed with a total of 500,000 instructions, in which 20% of them are the load/store instructions with an average CPI of 6 cycles, and the rest of the instructions are integer instructions with average CPI of 1 cycle.

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

\[
500,000 \times (0.2 \times 6 + 0.8 \times 1) \times 1 \text{ ns} = 1,000,000 \text{ ns}
\]
Speedup example

• If we double the clock rate to be $2\text{GHz}$ without improve the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?
Speedup example

• If we double the clock rate to be 2GHz without improving the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?

500,000 *
Speedup example

- If we double the clock rate to be 2GHz without improve the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?

\[
500,000 \times (0.2 \times 12 + 0.8 \times 1)
\]
Speedup example

• If we double the clock rate to be 2GHz without improve the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?

\[
500,000 \times (0.2 \times 12 + 0.8 \times 1) \times 0.5 \text{ ns} = 
\]
Speedup example

• If we double the clock rate to be 2GHz without improve the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?

\[
500,000 \ast \left(0.2 \ast 12 + 0.8 \ast 1\right) \ast 0.5 \text{ ns} = 800,000 \text{ ns}
\]
Speedup example

- If we double the clock rate to be 2GHz without improve the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?

\[
\text{500,000} \times (0.2 \times 12 + 0.8 \times 1) \times 0.5 \text{ ns} = 800,000 \text{ ns}
\]

\[
\text{Speedup} = \frac{\text{Execution time}_{\text{baseline}}}{\text{Execution time}_{\text{improved system}}}
\]
Speedup example

- If we double the clock rate to be 2GHz without improving the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?

\[
500,000 \times (.2 \times 12 + .8 \times 1) \times 0.5 \text{ ns} = 800,000 \text{ ns}
\]

\[
\text{Speedup} = \frac{\text{Execution time}_{\text{baseline}}}{\text{Execution time}_{\text{improved system}}}
\]

new machine = 800,000 ns
Speedup example

• If we double the clock rate to be 2GHz without improve the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?

\[
500,000 \times (.2 \times 12 + .8 \times 1) \times 0.5 \text{ ns} = 800,000 \text{ ns}
\]

Speedup = \( \frac{\text{Execution time}_{\text{baseline}}}{\text{Execution time}_{\text{improved system}}} \)

new machine = 800,000 ns
old machine = 1,000,000 ns
Speedup example

- If we double the clock rate to be 2GHz without improving the memory latency, the average CPI for load/store instruction will also be doubled to 12 cycles. What’s the performance improvement after this change?

\[
\begin{align*}
500,000 & \times \\
(0.2 \times 12 + 0.8 \times 1) & \times \\
0.5 \text{ ns} & = \\
800,000 \text{ ns}
\end{align*}
\]

\[
\text{Speedup} = \frac{\text{Execution time (baseline)}}{\text{Execution time (improved system)}}
\]

old machine = 1,000,000 ns
new machine = 800,000 ns
Speedup example

• If we double the clock rate to be **2GHz** without improve the memory latency, the average CPI for load/store instruction will also be doubled to **12** cycles. What’s the performance improvement after this change?

\[
\begin{align*}
500,000 & \times \\
(0.2 \times 12 + 0.8 \times 1) & \times \\
0.5 \text{ ns} & = \\
800,000 \text{ ns}
\end{align*}
\]

\[
\text{Speedup} = \frac{\text{old machine} = 1,000,000 \text{ ns}}{\text{new machine} = 800,000 \text{ ns}} = 1.25x
\]
### Reasoning About Performance

- Why does an Intel Core i7 @ 2.8 GHz usually perform better than an Intel Core 2 Extreme @ 3.2 GHz or AMD Phenom II X4 @ 3.4GHz?
Reasoning About Performance

- Why does an Intel Core i7 @ 2.8 GHz usually perform better than an Intel Core 2 Extreme @ 3.2 GHz or AMD Phenom II X4 @ 3.4GHz?

\[
ET = IC \times CPI \times \text{Cycle Time}
\]
Reasoning About Performance

- Why does an Intel Core i7 @ 2.8 GHz usually perform better than an Intel Core 2 Extreme @ 3.2 GHz or AMD Phenom II X4 @ 3.4 GHz?

\[
ET = IC \times CPI \times \text{Cycle Time}
\]

<table>
<thead>
<tr>
<th>Processor Description</th>
<th>IC</th>
<th>CPI</th>
<th>Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i7-920 (Bloomfield 4c)</td>
<td>7835</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Core 2 Extreme QX9770 (Yorkfield 4c)</td>
<td>7412</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intel Core 2 Extreme QX9650 (Yorkfield 4c)</td>
<td>7346</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMD Phenom II X4 965 (Deneb 4c)</td>
<td>7139</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Reasoning About Performance

• Why does an Intel Core i7 @ 2.8 GHz usually perform better than an Intel Core 2 Extreme @ 3.2 GHz or AMD Phenom II X4 @ 3.4 GHz?

\[ \text{ET} = \times \quad \text{IC} \times \quad \text{CPI} \times \quad \text{Cycle Time} \]
Reasoning About Performance

- Why does an Intel Core i7 @ 2.8 GHz usually perform better than an Intel Core 2 Extreme @ 3.2 GHz or AMD Phenom II X4 @ 3.4GHz? 

\[ ET = \text{IC} \times \text{CPI} \times \text{Cycle Time} \]
What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same programs, different machines different ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same programs, different machines different ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same programs, different machines different ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same programs, different machines different ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td>✗</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same programs, different machines, different ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Same programs, different machines different ISA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Same programs, different machines different ISA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## What Affects Performance

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Same programs, different machines, different ISA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Same programs, different machines different ISA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
## What Affects Performance

<table>
<thead>
<tr>
<th></th>
<th>Number of instructions</th>
<th>CPI</th>
<th>Clock Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same machine different programs</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>same programs, different machines, same ISA</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Same programs, different machines different ISA</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
What Affects Performance?
What Affects Performance?

- Compiler
What Affects Performance?

• Compiler
  • IC, CPI
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
What Affects Performance?

• Compiler
  • IC, CPI
• Instruction Set Architecture
  • IC, CPI, CT
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
  - IC, CPI, CT
- Programmer
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
  - IC, CPI, CT
- Programmer
  - IC, CPI
What Affects Performance?

• Compiler
  • IC, CPI
• Instruction Set Architecture
  • IC, CPI, CT
• Programmer
  • IC, CPI
• Application
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
  - IC, CPI, CT
- Programmer
  - IC, CPI
- Application
  - IC
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
  - IC, CPI, CT
- Programmer
  - IC, CPI
- Application
  - IC

- Processor architecture
What Affects Performance?

• Compiler
  • IC, CPI
• Instruction Set Architecture
  • IC, CPI, CT
• Programmer
  • IC, CPI
• Application
  • IC

• Processor architecture
  • CPI, CT
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
  - IC, CPI, CT
- Programmer
  - IC, CPI
- Application
  - IC
- Processor architecture
  - CPI, CT
- Process technology
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
  - IC, CPI, CT
- Programmer
  - IC, CPI
- Application
  - IC
- Processor architecture
  - CPI, CT
- Process technology
  - CT
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
  - IC, CPI, CT
- Programmer
  - IC, CPI
- Application
  - IC

- Processor architecture
  - CPI, CT
- Process technology
  - CT
- Programming languages
What Affects Performance?

- Compiler
  - IC, CPI
- Instruction Set Architecture
  - IC, CPI, CT
- Programmer
  - IC, CPI
- Application
  - IC

- Processor architecture
  - CPI, CT
- Process technology
  - CT
- Programming languages
  - IC, CPI
The Practice of Measuring Performance
The Practice of Measuring Performance

• Peak throughput measures (simple programs)? (GFLOPS)
The Practice of Measuring Performance

- Peak throughput measures (simple programs)? (GFLOPS)
- Synthetic benchmarks (whetstone, dhrystone,...)?
The Practice of Measuring Performance

• Peak throughput measures (simple programs)? (GFLOPS)
• Synthetic benchmarks (whetstone, dhrystone,...)?
• Real applications
The Practice of Measuring Performance

- Peak throughput measures (simple programs)? (GFLOPS)
- Synthetic benchmarks (whetstone, dhrystone,...)?
- **Real applications**
- SPEC (best of both worlds, but with problems of their own)
  - *System Performance Evaluation Cooperative*
  - Provides a common set of real applications along with strict guidelines for how to run them.
  - provides a relatively unbiased means to compare machines
Danger in Benchmark Specific Measures

Measures *compiler* as much as *architecture*!
Amdahl’s Law
Amdahl’s Law

Execution time after improvement = \( \frac{\text{Execution Time Affected}}{\text{Amount of Improvement}} \) + Execution Time Unaffected

- The impact of a performance improvement is limited by the percent of execution time affected by the improvement.
Amdahl’s Law

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

- Not only application to CS, but can be used anywhere!
Amdahl’s Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]
Amdahl’s Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

\[
\text{Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

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

\[
\text{Speedup} = \frac{1}{\text{Speedup}_{\text{enhanced}}}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

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

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

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

\[
\text{Speedup} = \frac{1}{(1 - 0.25)}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

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

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + }
\]
Amdahl’s Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

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

\[ \text{Speedup} = \frac{1}{(1 - 0.25) + \frac{10}{}} \]

Tuesday, January 22, 13
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

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

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}
\]
Amdahl’s Law: The Super MPEG Decoder

- Assume that we have a game spending 25% of its time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

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

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}}
\]
Amdahl’s Law: The Super MPEG Decoder

• Assume that we have a game spending 25% of it’s time doing MPEG decoding. If we add a hardware MPEG decoder that can speed up the MPEG decoding by 10x. How much does the hardware MPEG decoder help?

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

\[
\text{Speedup} = \frac{1}{(1 - 0.25) + \frac{0.25}{10}} = 1.29
\]
Amdahl’s Law

• Make the common case fast!!!
Key Points
Key Points

• Be careful how you specify performance
Key Points

• Be careful how you specify performance

• Execution time = instructions * CPI * cycle time
Key Points

• Be careful how you specify performance

• Execution time = instructions * CPI * cycle time

• Use real applications
Key Points

• Be careful how you specify performance

• Execution time = instructions * CPI * cycle time

• Use real applications

• Use standards, if possible
Key Points

• Be careful how you specify performance

• Execution time = instructions * CPI * cycle time

• Use real applications

• Use standards, if possible

• Make the common case fast