Outline

- Performance Factors
- Power Wall
- Amdahl's Law
Performance Metrics

- **Purchasing perspective**
  - Given a collection of machines, which has the best performance?
  - Best cost/performance ratio?

- **Design perspective**
  - Faced with a set of design options, which has the
    - Best performance?
    - Best performance improvement?

<table>
<thead>
<tr>
<th>Airplane</th>
<th>Capacity</th>
<th>Range</th>
<th>Speed</th>
<th>Passenger Throughput (Speed x Capacity)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 777</td>
<td>375</td>
<td>4630</td>
<td>610</td>
<td>228750</td>
</tr>
<tr>
<td>Boeing 747</td>
<td>470</td>
<td>4150</td>
<td>610</td>
<td>286700</td>
</tr>
<tr>
<td>BAC/Sud Concorde</td>
<td>132</td>
<td>4000</td>
<td>1350</td>
<td>178200</td>
</tr>
<tr>
<td>Douglas DC-8-50</td>
<td>146</td>
<td>8750</td>
<td>544</td>
<td>79424</td>
</tr>
</tbody>
</table>

Computer Performance: TIME

- **Response Time (Latency)**
  - How long does it take for my job to run?
  - How long does it take to execute a job?
  - How long should I wait for the database query?

- **Throughput**
  - How many jobs can run at once on the machine?
  - What is the average execution time?
  - How much work is getting done?

- If we upgrade a processor with a faster processor, what do we decrease?
- If we add a new machine, what do we improve?
CPU Clock Cycles

- Wall clock time, Elapsed time, or Response time
  - Counts everything (disk, memory, I/O)
  - A useful number, but often not good for comparison
- CPU Time
  - Doesn’t count I/O or time spent by other programs
  - Can be broken up into system and user time
- Our Focus: **User CPU Time**
  - Time spent executing the lines of codes that are “in” our program

Defining Performance

- Reducing response time
  - To maximize performance, need to minimize response time
    \[
    \text{Performance (x)} = \frac{1}{\text{Execution Time (x)}}
    \]
- If performance of X is greater than Y by a factor of “n”
  \[
  \frac{\text{Performance (X)}}{\text{Performance (Y)}} = \frac{\text{Execution Time (Y)}}{\text{Execution Time (X)}} = n
  \]
- Decreasing the response time mostly increases the throughput
Performance Factors

- Another way of reporting the execution time is to use cycles

\[
\text{CPU execution time for a program} = \frac{\# \text{ CPU clock cycles for a program}}{\text{clock rate}} \times \text{clock cycle time}
\]

- Clock “ticks” indicate when to start an activity
- Cycle time = time between ticks = seconds per cycle
- Clock rate (frequency) = cycles per second

Improving Performance

- Said another way,

\[
\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}
\]

- So we can improve performance (everything being equal) in the following way:
  - \( \frac{\text{cycles}}{\text{program}} \) # of required cycles for a program
  - \( \frac{\text{seconds}}{\text{cycle}} \) the clock cycle time,
  - \( \frac{\text{cycles}}{\text{program}} \) the clock rate

- Improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program
Improving Performance

• Said another way,

\[
\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}
\]

• So we can improve performance (everything being equal) in the following way:
  - Decreasing # of required cycles for a program
  - Decreasing the clock cycle time,
  - Increasing the clock rate

• Improve performance by reducing either the length of the clock cycle or the number of clock cycles required for a program

Cycles Required for a Program

• Can we assume number of cycles = number of instructions?
  - This assumption is incorrect, different instructions take different amounts of time on different machines, why?
    - Remember that these are machine instructions, not lines of C code
    - Multiplication takes more time than addition
    - Floating point operations take more time than integers
    - Accessing memory takes more time than accessing registers
    - NOTE: Changing the cycle time often changes the number of cycles required for various instructions
Example Problem

Our favorite program runs in 10 seconds on computer A, which has a 4 GHz clock. We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program. What clock rate should we tell the designer to target?

Solution

\[
\text{CPU execution time for a program} = \frac{\text{# CPU clock cycles for a program}}{\text{clock rate}}
\]

\[
\text{CPU execution time for a program}_A = \frac{\text{# CPU clock cycles for a program}_A}{\text{clock rate}_A}
\]

10 secs = \( \frac{\text{CPU Clock Cycles}_A}{4 \text{ GHz}} \)

6 secs = \( \frac{1.2 \times \text{CPU Clock Cycles}_A}{\text{Clock Rate}_B} \) = \( \frac{1.2 \times \text{CPU Clock Cycles}_A}{\text{Clock Rate}_B} \)

Substituting, \( \frac{\text{Clock Rate}_B}{6} = 8 \text{ GHz} \)
Cycles Per Instruction (CPI)

- A given program will require
  - Some number of instructions (machine instructions)
  - Some number of cycles
  - Some number of seconds to execute

\[
\text{# CPU clock cycles} = \frac{\text{# Instructions}}{\text{Average clock cycles per instruction}}
\]

- Cycles per Instruction (CPI) : the average number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

Effective CPI

- Computing the overall effective CPI is done by summing the different instructions and their individual clock cycles

\[
\text{Overall Effective CPI} = \frac{\sum_{i=1}^{n} CPI_i \times IC_i}{IC}
\]

- Where \( IC_i \) is the count of the number of instructions of class \( i \) executed
- \( CPI_i \) is the (average) number of clock cycles per instruction for that instruction class
- \( n \) is the number of instruction classes
- \( IC \) is the total instruction count

- The overall effective CPI varies by instruction mix – a measure of dynamic frequency of instructions across many programs
### CPI Example

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code Sequence</th>
<th>Instruction count for instruction class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A 2 B 1 C 2</td>
</tr>
<tr>
<td>2</td>
<td>A 4 B 1 C 1</td>
</tr>
</tbody>
</table>

- Which code sequence executes the most instructions? Which will be faster?

### CPI Example

<table>
<thead>
<tr>
<th>Instruction class</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code Sequence</th>
<th>Instruction count for instruction class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A 2 B 1 C 2</td>
</tr>
<tr>
<td>2</td>
<td>A 4 B 1 C 1</td>
</tr>
</tbody>
</table>

- Which code sequence executes the most instructions? Which will be faster?
  - Code Sequence 1: CPI = \((2 \times 1 + 1 \times 2 + 2 \times 3)/5 = 2.0\)
  - Code Sequence 2: CPI = \((4 \times 1 + 1 \times 2 + 1 \times 3)/6 = 1.5\)
Performance Equation

- Our basic performance equation becomes

\[ \text{CPU Execution Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle} \]

\[ \text{CPU Execution Time} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}} \]

- These factors separate three factors that affect performance
  - Can measure CPU execution time (run the program)
  - Clock rate is given
  - Can measure the overall instruction count using profilers/simulators
  - CPI varies by the instruction type and implementation details

Determines CPU Performance

\[ \text{CPU time} = \text{Instruction_count} \times \text{CPI} \times \text{clock_cycle} \]

<table>
<thead>
<tr>
<th></th>
<th>Instruction_count</th>
<th>CPI</th>
<th>clock_cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming language</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Processor organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Determines CPU Performance

CPU time = Instruction_count x CPI x clock_cycle

<table>
<thead>
<tr>
<th></th>
<th>Instruction_count</th>
<th>CPI</th>
<th>clock_cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Programming language</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISA</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Processor organization</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

The Power Wall

FIGURE 1.16 Clock rate and Power for Intel x86 microprocessors over eight generations and 25 years. The Pentium 4 made a dramatic jump in clock rate and power but less so in performance. The Prescott thermal problems led to the abandonment of the Pentium 4 line. The Core 2 line reverts to a simpler pipeline with lower clock rates and multiple processors per chip. The Core i5 pipelines follow in its footsteps.
Quantify Power

- For CMOS chips, traditional dominant energy consumption has been in switching transistors, called *dynamic power*

$$\text{Power}_{\text{Dynamic}} = \frac{1}{2} \times \text{Capacitive Load} \times \text{Voltage}^2 \times \text{Frequency}$$

$$\text{Energy}_{\text{Dynamic}} = \frac{1}{2} \times \text{Capacitive Load} \times \text{Voltage}^2$$

- For fixed task, slowing clock rate (frequency switched) reduces power, but not energy

Quantify Power

- Capacitive load is a function of the number of transistors connected to the output and technology
  - Determines capacitance of wires and transistors
- Dropping voltage helps both, so went from 5 V to 1 V
- To save energy and dynamic power, most CPUs now turn off the clock of inactive modules (Eg: Floating Point Unit)

- **Problem:** Suppose 15% reduction in voltage results in a 15% reduction in frequency and the new processor has the capacitance load of 85% of the more complex old processor, then what is the impact on dynamic power?
**Quantify Power**

- Capacitive load is a function of the number of transistors connected to the output and technology
  - Determines capacitance of wires and transistors
- Dropping voltage helps both, so went from 5 V to 1 V
- To save energy and dynamic power, most CPUs now turn off the clock of inactive modules (Eg: Floating Point Unit)

- **Problem:** Suppose 15% reduction in voltage results in a 15% reduction in frequency and the new processor has the capacitance load of 85% of the more complex old processor, then what is the impact on dynamic power?
  - **Solution:** \[ P = CVdd^2F = 0.85(C) \times (0.85Vdd)^2 \times 0.85(F) = 0.85^4(CVdd^2F) \]
    
    => Dynamic power is reduced by 52% compared to original

**Growth in Microprocessor Performance**
Multi-Cores, Clock Rate and Power

• **The Sea Change:** From uniprocessor designs to multicores!

<table>
<thead>
<tr>
<th>Product</th>
<th>AMD Opteron X4 (Bar celon a)</th>
<th>Intel Nehalem</th>
<th>IBM Power 6</th>
<th>Sun Ultra SP ARC T2 (Ni aga ra 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores per chip</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Clock rate</td>
<td>2.5 GHz</td>
<td>~2.5 GHz</td>
<td>4.7 GHz</td>
<td>1.4 GHz</td>
</tr>
<tr>
<td>Microprocessor power</td>
<td>120 W</td>
<td>~100 W</td>
<td>~100 W</td>
<td>94 W</td>
</tr>
</tbody>
</table>

Amdahl’s Law
(Law of Diminishing Returns)

• Every improvement typically affects a fraction of the program
• Execution time is generally divided into two classes: affected class and unaffected class

\[
T_{\text{after improvement}} = T_{\text{unaffected}} + \frac{T_{\text{affected}}}{\text{Improvement Factor}}
\]

\[
\text{Speedup} = \frac{T_{\text{before improvement}}}{T_{\text{after improvement}}}
\]

• Example: Suppose a program runs in 100 seconds and multiply is responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?
• How about making it 5 times faster?
Solution

\[ T(\text{after improvement}) = T(\text{unaffected}) + \frac{T(\text{affected})}{\text{Improvement Factor}} \]

\[ \frac{100}{4} = (100 - 80) + \frac{80}{\text{Improvement Factor}} \]

\[ 25 = 20 + \frac{80}{\text{Improvement Factor}} \]

\[ \text{Improvement Factor} = \frac{80}{(25 - 20)} = 16 \]

To make it 5 times faster, the equation will change to \( \frac{100}{5} = (100 - 80) + \frac{80}{\text{Improvement Factor}} \)

\[ \Rightarrow 20 = 20 + \frac{80}{\text{Improvement Factor}} \]

\[ \Rightarrow \text{It is impossible to make the program run 5 times faster} \]

MIPS (Million Instructions Per Second)

\[ \text{MIPS} = \frac{\text{Instruction Count}}{\text{Execution Time} \times 10^6} \]

- MIPS rating misleading
- Execution time is the only valid and unimpeachable measure of performance

\[ \text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{clock cycle} \]

\[ \frac{\text{seconds}}{\text{program}} = \frac{\text{Instructions}}{\text{program}} \times \frac{\text{Clock cycle}}{\text{Instruction}} \times \frac{\text{seconds}}{\text{clock cycle}} \]
Summary

- Performance is specific to a particular program
  - Total execution time is a consistent summary of performance
- For a given architecture, performance comes from
  - Increases the clock rate (without adversely affecting CPI)
  - Improvements in processor organization that lower the CPI
  - Compiler enhancements that lower CPI and/or instruction count
- Pitfall
  - Expecting improvement in one aspect of a machine's performance to affect the total performance