CS 152
Computer Architecture and Engineering

Lecture 6 – Performance

2005-9-15

John Lazzaro
(www.cs.berkeley.edu/~lazzaro)

TAs: David Marquardt and Udam Saini

www-inst.eecs.berkeley.edu/~cs152/
Last Time: Processor Timing

T2 might be the critical (worst-case delay) path.

\[ x = g(a, b, c, d, e, f) \]

If \( d \) going 0-to-1 switches \( x \) 0-to-1, delay is \( T_1 \).
If \( a \) going 0-to-1 switches \( x \) 0-to-1, delay is \( T_2 \).

Would you be surprised if \( T_1 > T_2 \)? Why?
Today’s Lecture - Performance

- Measurement: what, why, how
- The performance equation
- Amdahl’s law
- How energy limits performance
Performance Measurement
(as seen by the customer)
Who (sensibly) upgrades CPUs often?

A professional who turns CPU cycles into money, and who is cycle-limited.

Artist tool: animation, video special effects.
How to decide to buy a new machine?

Measure After Effects “execution time” on a representative render “workload”

“Night flight”
City map and clouds computed “on the fly” with fractals
CPU intensive Trivial I/O

(still shot from the movie)
Interpreting Execution Time

**Power Book G4**

1.25 GHz

**Execution Time**: 1265 seconds

**After Effects - Render "NightFlight"**

Time in Seconds -- SHORTER bar means FASTER

PB 1.50 Rad9700
PB 1.33 Rad9700
PB 1.33 Rad9600
PB 1.25

Performance = \( \frac{1}{\text{Execution Time}} \) = 2.85 renders/hour

1.5 GHz PB (Y) is \( N \) times faster than 1.25 GHz PB (X). \( N \) is ?

\[
N = \frac{\text{Performance (Y)}}{\text{Performance (X)}} = \frac{\text{Execution Time (X)}}{\text{Execution Time (Y)}} = 1.19
\]

PB 1.5 Ghz : 3.4 renders/hour. PB 1.25 : 2.85 renders/hour.

Might make the difference in meeting a deadline ...
2 CPUs: Execution Time vs Throughput

Execution Time: Time for 1 job to complete

Throughput: # of parallel jobs/hour completed

Assume G5 MP execution time faster because AE does not use both Opteron CPUs. Could G5 and Opteron have similar Throughput? Why?
Q. Why do we care about After Effect’s performance?
A. We want the CPU we are designing to run it well!
Step 1: Analyze the right measurement!

**CPU Time:**
Time the CPU spends running program under measurement.

**Response Time:**
Total time: CPU Time + time spent waiting (for disk, I/O, ...).

**How to measure CPU time?**
\[ \% \text{ time } <\text{program name}> \]
\[ 25.77u \ 0.72s \ 0:29.17 \ 90.8\% \]
CPU time: Proportional to Instruction Count

Q. Once ISA is set, who can influence instruction count?
A. Compiler writer, application developer.

Q. Static count? (lines of program printout)
Q. Or dynamic count? (trace of execution)
A. Dynamic.

Rationale: Every additional instruction you execute takes time.

CPU time \( \alpha \) Program

Machine Instructions Program

Q. What type of computer architect influences the number of instructions a given program needs?
A. Instruction set architect.
CPU time: Proportional to Clock Period

Q. How can architects (not technologists) reduce clock period?
A. Shorten the machine critical path.

Q. What ultimately limits an architect’s ability to reduce clock period?
A. Clock-to-Q, setup times.

Rationale:
We measure each instruction’s execution time in “number of cycles”.
By shortening the period for each cycle, we shorten execution time.
Completing the performance equation

What factors make the CPI for a program differ from the underlying CPI of a CPU implementation?

- Cache behavior varies.
- Instruction mix varies.
- Branch prediction varies.

\[
\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}}
\]

We need all three terms, and only these terms, to compute CPU Time!

\[
\frac{\text{Cycles}}{\text{Instruction}} = \frac{\text{Seconds}}{\text{Cycle}}
\]

“CPI” -- The Average Number of Clock Cycles Per Instruction For the Program

When is it OK to compare clock rates?
An example for average CPI ...

A program’s load instructions “stride” through every memory address.

The cache never “hits”, so every load goes to DRAM (100x slower than loads that go to cache).

Thus, the average number of cycles for load instructions is higher for this program.

Thus, the average number of cycles for all instructions is higher for this program.

\[
\text{Seconds} = \frac{\text{Instructions}}{\text{Program}} \quad \frac{\text{Cycles}}{\text{Instruction}} \quad \frac{\text{Seconds}}{\text{Cycle}}
\]

Thus, program takes longer to run!
CPI as an analytical tool to guide design

Machine CPI:

\[
5 \times 30 + 1 \times 20 + 2 \times 20 + 2 \times 10 + 2 \times 20 = 100
\]

\[
= 2.7 \text{ cycles/instruction}
\]

Program Instruction Mix:

- Multiply 30%
- Load 20%
- Other ALU 20%
- Store 10%
- Branch 20%

Where program spends its time.
Amdahl’s Law (of Diminishing Returns)

If enhancement “E” makes multiply infinitely fast, but other instructions are unchanged, what is the maximum speedup $S$?

$$S_{\text{max}} = \frac{1}{\text{un-enhanced } \% / 100\%} = \frac{1}{48\% / 100\%} = 2.08$$

Attributed to Gene Amdahl -- “Amdahl’s Law”

What is the lesson of Amdahl’s Law?
Must enhance computers in a balanced way!
Invented the "one ISA, many implementations" business model.
Amdahl’s Law in Action

The program spends 30% of its time running code that can not be recoded to run in parallel.

Program We Wish To Run On N CPUs

Serial 30%

Parallel 70%

Compute speedup for $N = 2, 3, 4, 5$, and $\infty$.

<table>
<thead>
<tr>
<th>CPUs</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A law of diminishing returns ...

The program spends 30% of its time running code that can not be recoded to run in parallel.

\[ S = \frac{1}{(30\% + \frac{70\%}{N}) / 100\%} \]

<table>
<thead>
<tr>
<th>CPUs</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>∞</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>1.54</td>
<td>1.85</td>
<td>2.1</td>
<td>2.3</td>
<td>3.3</td>
</tr>
</tbody>
</table>
### Final thoughts: Performance Equation

<table>
<thead>
<tr>
<th>Seconds Program</th>
<th>Instructions Program</th>
<th>Cycles Instruction</th>
<th>Seconds Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal is to optimize execution time, not individual equation terms.</td>
<td>Machines are optimized with respect to program workloads.</td>
<td>The CPI of the program. Reflects the program’s instruction mix.</td>
<td>Clock period. Optimize jointly with machine CPI.</td>
</tr>
</tbody>
</table>

\[
\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} = \frac{\text{Cycles}}{\text{Instruction}} = \frac{\text{Seconds}}{\text{Cycle}}
\]
Administrivia: Upcoming deadlines ...

Friday 9/16: “ModelSim Checkoff”, in section, 125 Cory. For non-150s, 150 Lab Lecture 3”, 2-3 PM, 125 Cory.

Friday 9/23: “Xilinx Checkoff”, in section, 125 Cory. For non-150s, 150 Lab Lecture 4”, 2-3 PM, 125 Cory.

Monday 9/26: Lab 2 final report due via the submit program, 11:59 PM.

Mid-Term 1 Coming Up: Tuesday October 4th
Energy and Performance

1 Joule of energy is dissipated by a 1 Amp current flowing through a 1 Ohm resistor for 1 second. Also, 1 Watt for 1 second.

1 Watt: 1 Amp flowing through 1 Ohm.

1 Joule = 0.24 calories.
1 calorie raises 1 gram of water 1 °C
Snickers bar: 273,000 calories.

Sad fact: computers turn electrical energy into heat. Computation is a byproduct.

Air or water carries heat away, or chip melts.
IBM Power 4: How does die heat up?

4 dies on a multi-chip module

2 CPUs per die
IBM Power 4: Dissipating 115 Watts

66.8°C == 152°F
82°C == 179.6°F
A practical aside ...

If you build your own desktop machines ...

And you forget to put on the CPU heat sink before first boot ...

Prepare to buy a new CPU. Modern desktop CPUs “melt” after a few seconds running code without a heat sink.
Switching Energy: Fundamental Physics

Every logic transition dissipates energy.

\[ E_{0\rightarrow1} = \frac{1}{2} CV_{dd}^2 \quad E_{1\rightarrow0} = \frac{1}{2} CV_{dd}^2 \]

Strong result: Independent of technology.

State-of-the-art CPUs (90 nm): Switching energy is 70% of total energy.

Remainder: at 90nm, “switches” are “dimmers”!

“leakage” currents 65nm: 50/50!

How can we limit switching energy?
Cell:
The PS3 chip
Cell: Conventional CPU + 8 “SPUs”

- L2 Cache: 512 KB
- PowerPC
- Synergistic Processing Units (SPUs)
One Synergistic Processing Unit (SPU)

256 KB Local Store -- 128 128-bit Registers
SPU issues 2 inst/cycle (in order) to 7 execution units
SPU fills Local Store using DMA to DRAM and network
A “Schmoo” plot for a Cell SPU...

The lower $V_{dd}$, the less dynamic energy consumption.

$$E_{0\rightarrow1} = \frac{1}{2} C V_{dd}^2$$

$$E_{1\rightarrow0} = \frac{1}{2} C V_{dd}^2$$

The lower $V_{dd}$, the longer the maximum clock period, the slower the clock frequency.

<table>
<thead>
<tr>
<th>$V_{dd}$ (Volt)</th>
<th>0.9</th>
<th>1</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power (W)</td>
<td>26C</td>
<td>26C</td>
<td>26C</td>
<td>26C</td>
<td>32C</td>
</tr>
<tr>
<td></td>
<td>1W</td>
<td>1W</td>
<td>1W</td>
<td>2W</td>
<td>2W</td>
</tr>
<tr>
<td></td>
<td>26C</td>
<td>26C</td>
<td>26C</td>
<td>26C</td>
<td>28C</td>
</tr>
<tr>
<td></td>
<td>2W</td>
<td>2W</td>
<td>2W</td>
<td>2W</td>
<td>2W</td>
</tr>
<tr>
<td></td>
<td>25C</td>
<td>26C</td>
<td>26C</td>
<td>26C</td>
<td>28C</td>
</tr>
<tr>
<td></td>
<td>1W</td>
<td>1W</td>
<td>2W</td>
<td>2W</td>
<td>2W</td>
</tr>
</tbody>
</table>

- Failed

Freq (GHz)
Fewer transitions saves power ...

Lowering clock frequency while keeping voltage constant saves some power, because number of transitions go down. But we do less work too.

\[ E_{0\rightarrow1} = \frac{1}{2} \ C \ V_{dd}^2 \quad \ E_{1\rightarrow0} = \frac{1}{2} \ C \ V_{dd}^2 \]
Parallel programming saves power

1 W to get 2.2 GHz performance. 26°C die temp.

7 W to reliably get 4.4 GHz performance. 47°C die temp.

If a program that needs a 4.4 GHz CPU can be recoded to use two 2.2 GHz CPUs ... big win.
Timely example: iPod Nano

Lithium battery size (power) limited by small case ...

... but 14 hour battery life (4 for slide shows) a key selling point.

Also, CPU fans and heat sinks not an option!
Finding the iPod nano CPU ...

A close (?) relative

PP5020
digital media management system-on-chip

Dual ARM Processors
- Dual 32-bit ARM7TDMI processors
- Up to 80 MHz processor operation per core with independent clock-skipping feature on COP
- Efficient cross-bar implementation providing zero wait state access to internal RAM
- Integrated 96KB of SRAM
- 8KB of unified cache per processor
- Six DMA channels

Two 80 MHz CPUs. This chip is used in the full-sized iPods, with one CPU doing audio decoding, the other doing photos, etc.
Conclusions

Customers: measure to buy
Architects: measure for design

Tools: Performance Equation, CPI

Amdahl’s Law’s lesson: Balance

Energy: $E_{0\rightarrow1} = \frac{1}{2} C V_{dd}^2$ $E_{1\rightarrow0} = \frac{1}{2} C V_{dd}^2$
Lectures: What is next ....

<table>
<thead>
<tr>
<th>Day</th>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>9/20</td>
<td>Pipelining I</td>
</tr>
<tr>
<td>W</td>
<td>9/21</td>
<td></td>
</tr>
<tr>
<td>Th</td>
<td>9/22</td>
<td>Pipelining II</td>
</tr>
<tr>
<td>F</td>
<td>9/23</td>
<td></td>
</tr>
<tr>
<td>Sa</td>
<td>9/24</td>
<td></td>
</tr>
<tr>
<td>Su</td>
<td>9/25</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>9/26</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>9/27</td>
<td>Pipelining III</td>
</tr>
</tbody>
</table>

3 pipelining lectures