Performance (III)
Power/Energy

Hung-Wei Tseng
Summary: Performance Equation

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

- ET = IC \times CPI \times \text{Cycle Time}
- 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
Programming languages

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

<table>
<thead>
<tr>
<th>Instruction count</th>
<th>LOC</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>480k</td>
<td>6</td>
</tr>
<tr>
<td>C++</td>
<td>2.8M</td>
<td>6</td>
</tr>
<tr>
<td>Java</td>
<td>166M</td>
<td>8</td>
</tr>
<tr>
<td>Perl</td>
<td>9M</td>
<td>4</td>
</tr>
<tr>
<td>Python</td>
<td>30M</td>
<td>1</td>
</tr>
</tbody>
</table>
**dynamic v.s. static instructions**

- Static instructions — number of instructions in the “compiled” code
- Dynamic instruction — number of instances of executing instructions when running the program

If the loop is executed 100 times, the dynamic instruction count will be $10+100\times10+10$

**static instructions: 30**
Amdahl’s Law

\[
\text{Speedup} = \frac{1}{\left(\frac{x}{S}\right) + (1-x)}
\]

- \(x\): the fraction of “execution time” that we can speed up in the target application
- \(S\): by how many times we can speedup \(x\)

\[
\text{total execution time} = 1
\]

\[
\text{total execution time} = \left(\frac{x}{S}\right) + (1-x)
\]
Amdahl’s Corollary #1

- Maximum possible speedup $S_{\text{max}}$, if we are targeting $x$ of the program.

\[
S = \text{infinity}
\]

\[
S_{\text{max}} = \frac{1}{0 \left( \frac{x}{\infty} + (1-x) \right)}
\]

\[
S_{\text{max}} = \frac{1}{(1-x)}
\]
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.

Common case

- 7x => 1.4x
- 4x => 1.3x
- 1.3x => 1.1x

Total = $\frac{20}{10} = 2x$
Don’t hurt non-common part too much

- If the program spends 90% of its time in A, 10% in B. Assume that an optimization can accelerate A by 9x, but hurts B by 10x...
- Assume the original execution time is $T$. The new execution time is:

$$T_{\text{new}} = \frac{Tx0.9}{9} + Tx0.1 \times 10$$

$$T_{\text{new}} = 1.1T$$

$$\text{Speedup} = \frac{T}{1.1T} = 0.91$$
Outline

- Amdahl’s Law (cont.)
- Power/Energy
- Other performance metrics
- Basic microprocessor design
Amdahl’s Corollary #3

- Assume that we have an application, in which $x$ of the execution time in this application can be fully parallelized with $S$ processors. What’s the speedup if we use a $S$-core processor instead of a single-core processor?

$$S_{par} = \frac{1}{\frac{x}{S} + (1-x)}$$
Recent advances in process technology have quadrupled the number of transistors you can fit on your die.

Currently, your key customer can use up to 4 processors for 40% of their application.

Which will you choose?

A. Increase the number of processors from 1 to 4

B. Use 2 cores, but add features that will allow the application to use two cores for 80% of execution.

\[
S_{\text{quad-core}} = \frac{1}{\frac{40\%}{4} + (1 - 40\%)} = 1.43
\]

\[
S_{\text{dual-core}} = \frac{1}{\frac{80\%}{2} + (1 - 80\%)} = 1.67
\]
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:
    \[
    \text{Speedup} = \frac{1}{(1 - X_{\text{Opt1}} - X_{\text{Opt2}}) + \frac{X_{\text{Opt1}}}{S_{\text{Opt1}}} + \frac{X_{\text{Opt2}}}{S_{\text{Opt2}}}}
    \]
  - If optimization #1 and optimization #2 are not dis-joint:
    \[
    S = \frac{1}{(1 - X_{\text{Opt1Only}} - X_{\text{Opt2Only}} - X_{\text{Opt1&Opt2}}) + \frac{X_{\text{Opt1}}}{S_{\text{Opt1Only}}} + \frac{X_{\text{Opt2}}}{S_{\text{Opt2Only}}} + \frac{X_{\text{Opt1&Opt2}}}{S_{\text{Opt1&Opt2}}}}
    \]

total execution time = 1
Assume that we have an application, in which 50% of the application can be fully parallelized with 2 processors. Assuming 80% of the parallelized part can be further parallelized with 4 processors, what’s the speed up of the application running on a 4-core processor?

Code can be optimized for 2-core = 50%*(1-80%) = 10%

Code can be optimized for 4-core = 50%*80% = 40%

\[
\text{Speedup}_{\text{quad}} = \frac{1}{(1-0.5) + \frac{0.10}{2} + \frac{0.40}{4}} = 1.54
\]
Amdahl’s Law for multiple optimizations

- Assume that memory access takes 30% of execution time.
  - Cache can speedup 80% of memory operation by a factor of 4
  - L2 cache can speedup 50% of the remaining 20% by a factor of 2
- What’s the total speedup?
  A. 1.22
  B. 1.23
  C. 1.24
  D. 2.63
  E. 2.86

Execution time can be optimized by L1 only = 30%*80% = 24%
Execution time can be optimized by L2 only = 30%*50%*20% = 3%

\[
\text{Speedup} = \frac{1}{(1- 0.27) + \frac{0.24}{4} + \frac{0.03}{2}} = 1.24
\]
Case study: more cores?

- If you cannot make your mobile Apps multithreaded, Apple A7 is the best
Case study: LOL

- Corollary #2
- The CPU is not the main performance bottleneck
- CPU parallelism doesn’t help, either
- You might consider
  - GPU
  - network
  - storage (loading maps)
Corollaries of Amdahl’s Law

- Maximum possible speedup $S_{\text{max}}$
  \[
  S_{\text{max}} = \frac{1}{(1-x)}
  \]

- Make the common case fast (i.e., $x$ should be large)
  - Common == most time consuming not necessarily the most frequent
  - Use profiling tools to figure out

- Estimate the potential of parallel processing
  \[
  S_{\text{par}} = \frac{x}{S} + (1-x)
  \]

- Estimate the effect of multiple optimizations
  \[
  S = \frac{1}{(1 - X_{\text{Opt1Only}} - X_{\text{Opt2Only}} - X_{\text{Opt1&Opt2}}) + \frac{X_{\text{Opt1}}}{S_{\text{Opt1Only}}} + \frac{X_{\text{Opt2}}}{S_{\text{Opt2Only}}} + \frac{X_{\text{Opt1&Opt2}}}{S_{\text{Opt1&Opt2}}}}
  \]

Amdahl’s Law can help you in making the right decision!
Power & Energy
Power & Energy

- Regarding power and energy, how many of the following statements are correct?
  1. Lowering the power consumption helps extending the battery life
  2. Lowering the power consumption helps reducing the heat generation
  3. Lowering the energy consumption helps reducing the electricity bill
  4. A CPU with 10% utilization can still consume 33% of the peak power

A. 0
B. 1
C. 2
D. 3
E. 4
Power

- Power is the direct contributor of “heat”
  - Packaging of the chip
  - Heat dissipation cost
- Two sources of power consumption
  - Dynamic power
  - Static power
Dynamic Power

- The power consumption due to the switching of transistor states
- Dynamic power per transistor
  \[ P_{\text{dynamic}} \sim a \cdot C \cdot V^2 \cdot f \cdot N \]
  - \( a \): average switches per cycle
  - \( C \): capacitance
  - \( V \): voltage
  - \( f \): frequency, usually linear with \( V \)
  - \( N \): the number of transistors
Doubling clock rate v.s. doubling cores

Assume the the power consumption of original core is $P$

$$\text{Power}_{2\text{-core}} = 2^1P$$

$$\text{Power}_{2\times\text{Clock}} = 2^3P = 8P$$
Static Power

- The power consumption due to leakage — transistors do not turn all the way off during no operation.
- Becomes the dominant factor in the most advanced process technologies.
- \[ P_{\text{Leakage}} \sim N \cdot V \cdot e^{-Vt} \]
  - \( N \): number of transistors
  - \( V \): voltage
  - \( Vt \): threshold voltage where transistor conducts (begins to switch)
Dynamic voltage/frequency scaling

- Dynamically trade-off power for performance
  - Change the voltage and frequency at runtime
  - Under control of operating system — that’s why updating iOS may slow down an old iPhone
- Recall: $P_{\text{dynamic}} \sim a \cdot C \cdot V^2 \cdot f \cdot N$
  - Because frequency $\sim$ to $V$
  - $P_{\text{dynamic}} \sim$ to $V^3$
- Reduce both $V$ and $f$ linearly
  - Cubic decrease in dynamic power
  - Linear decrease in performance (actually sub-linear)
    - Thus, only about quadratic in energy
  - Linear decrease in static power
    - Thus, only modest static energy improvement
- Newer chips can do this on a per-core basis
  - cat /proc/cpuinfo in Linux
Energy

- Energy = P * ET
- The electricity bill and battery life is related to energy!
- Lower power does not necessarily mean better battery life if the processor slows down the application too much
**Double Clock Rate or Double the # of Processors?**

- Assume 60% of the application can be fully parallelized with 2-core or speedup linearly with clock rate. Should we double the clock rate or duplicate a core?

\[
\text{Speedup}_{\text{2-core}} = \frac{1}{(1 - 0.6) + \frac{0.6}{2}} = 1.43 \\
\text{Power}_{\text{2-core}} = 2x \\
\text{Energy}_{\text{2-core}} = 2 \times \left[ 1/(1.43) \right] = 1.39 \\
\text{Speedup}_{2\times\text{Clock}} = 2 \\
\text{Power}_{2\times\text{Clock}} = 8x \\
\text{Energy}_{2\times\text{Clock}} = 8 / 2 = 4
\]
What happens if power doesn’t scale with process technologies?

If we are able to cram more transistors within the same chip area (Moore’s law continues), but the power consumption per transistor remains the same. Right now, if we power the chip with the same power consumption but put more transistors in the same area because the technology allows us to. How many of the following statements are true?

① The power consumption per chip will increase
② The power density of the chip will increase
③ Given the same power budget, we may not able to power on all chip area if we maintain the same clock rate
④ Given the same power budget, we may have to lower the clock rate of circuits to power on all chip area

A. 0
B. 1
C. 2
D. 3
E. 4
Power density
Dark silicon

- \( P_{\text{Leakage}} \sim N \times V \times e^{-Vt} \)
  - \( N \): number of transistors
  - \( V \): voltage
  - \( Vt \): threshold voltage where transistor conducts (begins to switch)
- Your power consumption goes up as the number of transistors goes up
  - You have to turn off/slow down some transistors completely to reduce leakage power
  - Intel TurboBoost: dynamically turn off/slow down some cores to allow a single core achieve the maximum frequency
  - big.LITTLE cores: Qualcomm Snapdragon 835 has 4 cores can achieve more than 2GHz but 4 other cores can only achieve up to 1.9GHz
How many of the following comparisons are fair?

① Comparing the frame rates of Halo 5 on AMD RyZen 1600X and civilization on Intel Core i7 7700X
② Using bit torrent to compare the network throughput on two machines
③ Comparing the frame rates of Halo 5 using medium settings on AMD RyZen 1600X and low settings on Intel Core i7 7700X
④ Using the peak floating point performance to judge the gaming performance of machines using AMD RyZen 1600X and Intel Core i7 7700X

A. 0
B. 1
C. 2
D. 3
E. 4
Benchmark
Benchmark suites

• A benchmark suite is a set of programs that are representative of a class of problems.
  • Desktop computing (many available online)
  • Server computing (SPECINT)
  • Scientific computing (SPECFP)
  • Embedded systems (EEMBC)
• There is no “best” benchmark suite.
  • Unless you are interested only in the applications in the suite, they are flawed
  • The applications in a suite can be selected for all kinds of reasons.
• To make broad comparisons possible, benchmarks usually are;
  • “Easy” to set up
  • Portable
  • Well-understood
  • Stand-alone
  • Run under standardized conditions
• Real software is none of these things.
Classes of benchmarks

- Microbenchmarks measure one feature of system
  - e.g. memory accesses or communication speed
- Kernels – most compute-intensive part of applications
  - Amdahl’s Law tells us that this is fine for some applications.
  - e.g. Linpack and NAS kernel benchmarks
- Full application:
  - SpecInt / SpecFP (for servers)
  - Other suites for databases, web servers, graphics,...
## SPECInt2006

<table>
<thead>
<tr>
<th>Application</th>
<th>Language</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400.perlbench</td>
<td>C</td>
<td>PERL Programming Language</td>
</tr>
<tr>
<td>401.bzip2</td>
<td>C</td>
<td>Compression</td>
</tr>
<tr>
<td>403.gcc</td>
<td>C</td>
<td>C Compiler</td>
</tr>
<tr>
<td>429.mcf</td>
<td>C</td>
<td>Combinatorial Optimization</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>C</td>
<td>AI: go</td>
</tr>
<tr>
<td>456.hmmer</td>
<td>C</td>
<td>Search Gene Sequence</td>
</tr>
<tr>
<td>458.sjeng</td>
<td>C</td>
<td>AI: chess</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>C</td>
<td>Quantum Computing</td>
</tr>
<tr>
<td>464.h264ref</td>
<td>C</td>
<td>Video Compression</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>C++</td>
<td>Discrete Event Simulation</td>
</tr>
<tr>
<td>473.astar</td>
<td>C++</td>
<td>Path-finding Algorithms</td>
</tr>
<tr>
<td>483.xalancbmk</td>
<td>C++</td>
<td>XML Processing</td>
</tr>
</tbody>
</table>
Other important metrics
Bandwidth

• The amount of work (or data) during a period of time
  • Network/Disks: MB/sec, GB/sec, Gbps, Mbps
  • Game/Video: Frames per second
• Also called “throughput”
• “Work done” / “execution time”
Response time and BW trade-off

- Increase bandwidth can hurt the response time of a single task
  - If you want to transfer a 2 Peta-Byte video from UCLA
    - 125 miles (201.25 km) from UCSD
    - Assume that you have a 100Gbps ethernet
      - 2 Peta-byte over 167772 seconds = 1.94 Days
      - 22.5TB in 30 minutes
      - Bandwidth: 100 Gbps
Or ...

<table>
<thead>
<tr>
<th>Toyota Prius</th>
<th>10Gb Ethernet</th>
</tr>
</thead>
<tbody>
<tr>
<td>• 25 miles from UNC</td>
<td>100 Gb/s or 12.5GB/sec</td>
</tr>
<tr>
<td>• 75 MPH on highway!</td>
<td></td>
</tr>
<tr>
<td>• 50 MPG</td>
<td></td>
</tr>
<tr>
<td>• Max load: 374 kg = 2,770 hard drives (2TB per drive)</td>
<td>2 Peta-byte over 167772 seconds = 1.94 Days</td>
</tr>
<tr>
<td>bandwidth</td>
<td>response time</td>
</tr>
<tr>
<td>290GB/sec</td>
<td>You see nothing in the first 4 hours</td>
</tr>
<tr>
<td>latency</td>
<td>You can start watching the movie as soon as you get a frame!</td>
</tr>
<tr>
<td>4 hours</td>
<td></td>
</tr>
</tbody>
</table>
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</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 (Giga FLoating-point Operations Per Second) a good metric?

- 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

\[
TFLOPS = \frac{\# \text{ of floating point instructions} / 10^9}{\text{Execution Time}} = \frac{\text{IC} \times \% \text{ of floating point instructions}}{\text{IC} \times \text{CPI} \times \text{CycleTime} \times 10^9} = \frac{\text{Clock Rate} \times \% \text{ FP ins.}}{\text{CPI} \times 10^9}
\]
Reliability

- Mean time to failure (MTTF)
  - Average time before a system stops working
  - Very complicated to calculate for complex systems
- Hardware can fail because of
  - Electromigration
  - Temperature
  - High-energy particle strikes