Performance of computer systems

- Many different factors among which:
  - Technology
    - Raw speed of the circuits (clock, switching time)
    - Process technology (how many transistors on a chip)
  - Organization
    - What type of processor (e.g., RISC vs. CISC)
    - What type of memory hierarchy
    - What types of I/O devices
  - How many processors in the system
  - Software
    - O.S., compilers, database drivers etc

What are some possible metrics?

Traditional measures:

- Raw speed (peak performance = clock rate)
- Execution time (or response time): time to execute a program from beginning to end.
  - Need benchmarks for integer dominated programs, scientific, graphical interfaces, multimedia tasks, desktop apps, utilities etc.
- Throughput (total amount of work in a given time)
  - measures utilization of resources (good metric when many users: e.g., large data base queries, Web servers)
  - Improving (decreasing) execution time will improve (increase) throughput.
  - Most of the time, improving throughput will decrease execution time
What are some possible metrics?

Recently:

• Measures that concern power
  – Watts = joules / second
  – Energy per instruction = joules / instruction executed

• Why be concerned about power?
  – Battery life in portable devices
  – Heat dissipation issues
  – Server rooms are most constrained by their cooling capacity
  – Dense clusters can be constrained by the ability to route enough power into the installation and/or to the individual processors

CPU Execution Time
Moore’s Law

Processor-Memory Performance Gap

- x Memory latency decrease (10x over 8 years but densities have increased 100x over the same period)
- o x86 CPU speed (100x over 10 years)
Comparing Processors Isn’t Straightforward

- Different architectures have different instruction sets
  - Can’t run the same set of (machine) instructions on both

- Even different models in the same architecture may have a complicated relationship
  - Model A’s multiply is 6 times faster than model B’s
  - Model A’s add is 3 times faster than model B’s
  - Model A’s memory system is 8 times faster than model B’s

- But, we really want to compare performance across processors…

Comparing Performance

- The “right measure” is execution time
  - Take some C program, compile, link and run on both processors
  - Measure the time it takes from start to end of the execution

- Notice that this means we are evaluating the compilers as well as the processors
  - Is that reasonable?

- If we’re not careful, we might be measuring other things as well
  - E.g., speed of IO devices
Execution time Metric

- **Execution time:** inverse of performance
  \[ \text{Performance}_A = \frac{1}{\text{Execution time}_A} \]

- “Processor A is faster than Processor B”
  \[ \text{Execution time}_A < \text{Execution time}_B \]
  \[ \text{Performance}_A > \text{Performance}_B \]

- **Relative performance** (a computer is “n times faster” than another one)
  \[ \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} \]

Definition of CPU execution time

**CPU execution time** = (#cycles) * (time per cycle)

- (#cycles) depends on program, compiler, and input

- (time per cycle) is the inverse of clock rate
  - Depends on the processor’s implementation
  - Clock rate measured in MHz or GHz
Another form of the equation

CPU execution\_time = 
\((#\text{insts executed}) \times (\text{cycles} / \text{instruction}) \times (\text{time/cycle})\)

- (cycles / instruction) is called CPI
- CPI depends on processor’s implementation:
  - CPI = 1  “Single cycle”
  - CPI > 1  Some instructions require more than one cycle
  - CPI < 1  Some form of parallel execution

How to Improve Performance?

CPU execution\_time = 
\((#\text{insts executed}) \times (\text{cycles} / \text{instruction}) \times (\text{time/cycle})\)

- Reduce (#insts executed) : better compilers
- Reduce (time/cycle) : higher clock rates or better processor implementations
- Reduce CPI : more internal parallelism in processor implementation
  - Pipelining, Superscalar, multi-threaded
Benchmarks

- Benchmark: workload representative of what a system will be used for
- Industry benchmarks
  - SPECint and SPECfp industry benchmarks updated every few years, Currently SPEC CPU2006
  - Linpack (Lapack), NASA kernel: scientific benchmarks
  - TPC-A, TPC-B, TPC-C and TPC-D used for databases and data mining
  - Other specialized benchmarks (Olden for list processing, Specweb, SPEC JVM98 etc...)
- Benchmarks for desktop applications, web applications are not as standard
- Beware! Compilers (command lines) are super optimized for the benchmarks

CINT2006 (Integer Component of SPEC CPU2006):

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Language</th>
<th>Application Area</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>401 fma772</td>
<td>C</td>
<td>Compression</td>
<td>Julii (Staders) fma72 version 1.6.3, modified to do most work in memory, rather than doing I/O.</td>
</tr>
<tr>
<td>403-gcc</td>
<td>C</td>
<td>C Compiler</td>
<td>Based on gcc Version 3.3. generates code for Opteron.</td>
</tr>
<tr>
<td>429-ensf</td>
<td>C</td>
<td>Combinatorial Optimization</td>
<td>Vehicle scheduling. Uses an exact simplex algorithm which is also used in commercial products to schedule public transport.</td>
</tr>
<tr>
<td>445-gobsm</td>
<td>C</td>
<td>Artificial Intelligence: Go</td>
<td>Plays the game of Go, a simply described but deeply complex game.</td>
</tr>
<tr>
<td>456-trnsmr</td>
<td>C</td>
<td>Search Game Sequence</td>
<td>Protein sequence analysis using profile hidden Markov models (profile HMMs)</td>
</tr>
<tr>
<td>458-tengg</td>
<td>C</td>
<td>Artificial Intelligence: Chess</td>
<td>A highly-ranked chess program that also plays several chess variants.</td>
</tr>
<tr>
<td>452-ilquant</td>
<td>C</td>
<td>Physics / Quantum Computing</td>
<td>Simulates a quantum computer, running Shor's polynomial-time factorization algorithm.</td>
</tr>
<tr>
<td>454-z2vec</td>
<td>C</td>
<td>Video Compression</td>
<td>A reference implementation of Z2VEC, encodes a video frame using 2 parameter sets. The H.264/AVC standard is expected to replace SPE2.</td>
</tr>
<tr>
<td>471-mmapp</td>
<td>C++</td>
<td>Discrete Event Simulation</td>
<td>Uses the OpenGL++ discrete event simulator to model a large Ethernet campus network.</td>
</tr>
<tr>
<td>473-astar</td>
<td>C++</td>
<td>Path-finding Algorithms</td>
<td>Pathfinding for 3D maps, including the well known A* algorithm.</td>
</tr>
<tr>
<td>463-salomon</td>
<td>C++</td>
<td>SQL Processing</td>
<td>A modified version of salomon C++, which transforms SQL documents to other document types.</td>
</tr>
</tbody>
</table>
How to summarize benchmark performance

• n programs in the benchmark suite. What is the relative performance “overall”?

• A number of alternatives:
  – arithmetic mean of execution times:
    • \( \frac{\sum \text{exec.time}}{n} \)
  – harmonic mean of rates:
    • \( \frac{n}{\sum \frac{1}{\text{rate}}} \)
  – geometric mean of rates:
    • \( \left( \prod \text{rate} \right)^{1/n} \)

Power
Table 2: Performance and Power of Intel Microprocessors, 130 nm to 65 nm

<table>
<thead>
<tr>
<th>Product</th>
<th>Normalized Performance</th>
<th>Normalized Power</th>
<th>EPI on 65 nm at 1.33 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>E6487</td>
<td>0.8</td>
<td>1.0</td>
<td>16</td>
</tr>
<tr>
<td>Pentium</td>
<td>0.2</td>
<td>0.7</td>
<td>14</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>0.4</td>
<td>0.5</td>
<td>24</td>
</tr>
<tr>
<td>Pentium 4 (Willamette)</td>
<td>0.3</td>
<td>0.3</td>
<td>26</td>
</tr>
<tr>
<td>Pentium 4 (Cedar Mill)</td>
<td>0.9</td>
<td>0.9</td>
<td>36</td>
</tr>
<tr>
<td>Pentium M (Willamette)</td>
<td>0.4</td>
<td>0.4</td>
<td>25</td>
</tr>
<tr>
<td>Pentium M (Cedar Mill)</td>
<td>0.4</td>
<td>0.4</td>
<td>15</td>
</tr>
<tr>
<td>Core Duo</td>
<td>0.7</td>
<td>0.7</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 3: EPI of Intel Microprocessors


Figure 2: Normalized Power versus Normalized Scalar Performance for Multiple Generations of Intel Microprocessors
Computer design: Make the common case fast

- Amdahl’s law (speedup)
- Speedup = (performance with enhancement)/(performance base case)
  
  Or equivalently,
  
  Speedup = (exec.time base case)/(exec.time with enhancement)

- For example, application to parallel processing
  - s fraction of program that is sequential
  - Speedup S is at most 1/s
  - That is if 20% of your program is sequential the maximum speedup with an infinite number of processors is at most 5