Performance

CSE 378 Spring 2009

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, how big the transistors are)
  - 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

\[ \text{Execution\_time} = (\#\text{insts executed}) \times \text{CPI} \times (\text{time/cycle}) \]

Moore’s Law

Courtesy Intel Corp.
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
  \[ \text{Performance}_x = \frac{1}{\text{Execution time}_x} \]

- “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)
  \[ \frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution time}_y}{\text{Execution time}_x} \]
Definition of CPU execution time

\[
\text{CPU execution time} = (\#\text{cycles}) \times (\text{time per cycle})
\]

- (\#\text{cycles}) depends on program, compiler, and input
- (\text{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

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

- (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 = (#insts executed) * CPI * (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, multi-core

---

Benchmarks

- Benchmark: workload representative of what a system will be used for
- Industry benchmarks
  - SPECint and SPECfp industry benchmarks updated every few years,
  - 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
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:
    - \((\sum \text{exec} \_ \text{time}) / n\)
  - harmonic mean of rates:
    - \(n / (\sum 1/\text{rate})\)
  - geometric mean of rates:
    - \((\prod \text{rate})^{1/n}\)
Power

\[ \text{EPI} = \text{Joules} / \text{instruction} = \text{Watts} / \text{IPS} \]

Dynamic Power Dissipation

CMOS Inverter
Moore’s Law

![Moore's Law Graph]

Courtesy Intel Corp.

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

<table>
<thead>
<tr>
<th>Product</th>
<th>Name</th>
<th>Core</th>
<th>Frequency</th>
<th>Performance</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium 4</td>
<td>Proliant DL380</td>
<td>5</td>
<td>3.0 GHz</td>
<td>1.64</td>
<td>60</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>Proliant DL360</td>
<td>6</td>
<td>2.8 GHz</td>
<td>1.54</td>
<td>60</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>Proliant DL350</td>
<td>7</td>
<td>2.6 GHz</td>
<td>1.45</td>
<td>60</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>Proliant DL330</td>
<td>8</td>
<td>2.4 GHz</td>
<td>1.36</td>
<td>60</td>
</tr>
<tr>
<td>Pentium 4</td>
<td>Proliant DL310</td>
<td>9</td>
<td>2.2 GHz</td>
<td>1.27</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 3: EPI of Intel Microprocessors


05/06/09  CSE378 Performance.  19
Figure 2: Normalized Power versus Normalized Scalar Performance for Multiple Generations of Intel Microprocessors

Parallelism

\[ S(\infty) = \frac{1}{f} \quad \text{(Amdahl's Law)} \]

Amdahl's Law (Parallel Processor Speedup)

- \( S(P) \) is parallel speedup using \( P \) processors
  - (sequential execution time) / (parallel exec time using \( P \) processors)
- Assume:
  - fraction \( f \) of the application's execution is "inherently sequential"
  - fraction \((1-f)\) can be perfectly parallelized
- \( S(P) = \frac{1}{f + \frac{(1-f)}{P}} \)
- \( S(\infty) = \frac{1}{f} \)
  - For example, if 20% of your program is inherently sequential, the maximum possible parallel speedup is at 5

- What fraction of rendering a web page is inherently sequential?
- What fraction of Google's workload is inherently sequential?
What Next?

- We'll look at parallelism
- First, pipelining – a simple approach to speeding up the data path
- Then, “instruction level parallelism”: more aggressive techniques for to allow execution of more than one instruction at a time
- Both of the above preserve the sequential semantics of the ISA
- Multi-core changes the ISA (and makes exploiting parallelism the software's problem...)