Performance & Technology
Todd C. Mowry
740
Sept 11, 2012

Topics:
- Performance measures
- Relating performance measures
- Memory Technology
  - SRAM, DRAM
- Disk Technology
- Recent Processor Trends

Performance expressed as a time

Absolute time measures
- difference between start and finish of an operation
- synonyms: running time, elapsed time, response time, latency, completion time, execution time
- most straightforward performance measure

Relative (normalized) time measures
- running time normalized to some reference time
  - e.g. time/reference time

Guiding principle: Choose performance measures that track running time.

Performance expressed as a rate

Rates are performance measures expressed in units of work per unit time.

Examples:
- millions of instructions / sec (MIPS)
- millions of floating point instructions / sec (MFLOPS)
- millions of bytes / sec (MBytes/sec)
- millions of bits / sec (Mbits/sec)
- images / sec
- samples / sec
- transactions / sec (TPS)

Key idea: Report rates that track execution time.

Example: Suppose we are measuring a program that convolves a stream of images from a video camera.

Bad performance measure: MFLOPS
- number of floating point operations depends on the particular convolution algorithm: \( n^2 \) matrix-vector product vs \( n \log n \) fast Fourier transform. An FFT with a bad MFLOPS rate may run faster than a matrix-vector product with a good MFLOPS rate.

Good performance measure: images/sec
- a program that runs faster will convolve more images per second.
Performance expressed as a rate (cont)

Fallacy: Peak rates track running time.

Example: the i860 was advertised as having a peak rate of 80 MFLOPS
• (40 MHz with 2 flops per cycle.)

However, the measured performance of some compiled linear algebra kernels (icc -O2) told a different story:

<table>
<thead>
<tr>
<th>Kernel</th>
<th>id fft</th>
<th>sasum</th>
<th>saxpy</th>
<th>sdot</th>
<th>sgemm</th>
<th>sgenv</th>
<th>spvma</th>
</tr>
</thead>
<tbody>
<tr>
<td>MFLOPS</td>
<td>8.5</td>
<td>3.2</td>
<td>6.1</td>
<td>10.3</td>
<td>6.2</td>
<td>15.0</td>
<td>8.1</td>
</tr>
<tr>
<td>%peak</td>
<td>11%</td>
<td>4%</td>
<td>7%</td>
<td>13%</td>
<td>8%</td>
<td>19%</td>
<td>10%</td>
</tr>
</tbody>
</table>

Relating time to system measures

Suppose that for some program we have:
• $T$ seconds running time (the ultimate performance measure)
• $C$ clock ticks, $I$ instructions, $P$ seconds/tick (performance measures of interest to the system designer)

\[
T \text{ secs} = C \text{ ticks} \times P \text{ secs/tick} = \left( \frac{I \text{ inst}}{I \text{ inst}} \right) \times C \text{ ticks} \times P \text{ secs/tick}
\]

Pipeline latency and throughput

Latency ($L$): time to process an individual image.
Throughput ($R$): images processed per unit time
One image can be processed by the system at any point in time

Video system performance

Latency ($L$): 3 secs/image.
Throughput ($R$): 1/L = 1/3 images/sec.

\[
T = L + (N-1)/R = 3N
\]

Stage 1

<table>
<thead>
<tr>
<th>Stage</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

1 out

2 out
Pipelining the video system

<table>
<thead>
<tr>
<th>video pipeline</th>
<th>stage 1 (buffer)</th>
<th>stage 2 (CPU)</th>
<th>stage 3 (display)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( I_1, \ldots, I_N )</td>
<td>( (L_1, R_1) )</td>
<td>( (L_2, R_2) )</td>
<td>( (L_3, R_3) )</td>
</tr>
<tr>
<td>( O_1, \ldots, O_N )</td>
<td>( (L_1, R_1) )</td>
<td>( (L_2, R_2) )</td>
<td>( (L_3, R_3) )</td>
</tr>
</tbody>
</table>

One image can be in each stage at any point in time.

- \( L_i \) = latency of stage \( i \)
- \( R_i \) = throughput of stage \( i \)

\[ L = L_1 + L_2 + L_3 \]
\[ R = \min(R_1, R_2, R_3) \]

Pipelined video system performance

Suppose:

\[ L_1 = L_2 = L_3 = 1 \]

Then:

\[ L = 3 \text{ secs/image} \]
\[ R = 1 \text{ image/sec} \]
\[ T = L + (N-1)/R = N + 2 \]

Relating time to latency & throughput

In general:

- \( T = L + (N-1)/R \)

The impact of latency and throughput on running time depends on \( N \):

- \( N = 1 \rightarrow T = L \)
- \( N >> 1 \rightarrow T = N/R \)

To maximize throughput, we should try to maximize the minimum throughput over all stages (i.e., we strive for all stages to have equal throughput).

Amdahl’s law

You plan to visit a friend in Normandy France and must decide whether it is worth it to take the Concorde SST ($3,100) or a 747 ($1,021) from NY to Paris, assuming it will take 4 hours Pgh to NY and 4 hours Paris to Normandy.

<table>
<thead>
<tr>
<th></th>
<th>NY to Paris Time</th>
<th>Total Time</th>
<th>Speedup over 747</th>
</tr>
</thead>
<tbody>
<tr>
<td>747:</td>
<td>8.5 hours</td>
<td>16.5 hours</td>
<td>1.0</td>
</tr>
<tr>
<td>SST:</td>
<td>3.75 hours</td>
<td>11.75 hours</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Taking the SST (which is 2.2 times faster) speeds up the overall trip by only a factor of 1.4!
Amdahl’s law (cont)

Old program (unenhanced)

\[ T_1 \] = time that cannot be enhanced.

\[ T_2 \] = time that can be enhanced.

Old time: \[ T = T_1 + T_2 \]

New program (enhanced)

\[ T'_1 = T_1 \]

\[ T'_2 = T_2' \]

New time: \[ T' = T'_1 + T'_2 \]

Speedup: \[ S_{\text{overall}} = \frac{T}{T'} \]

Amdahl’s law (cont)

Two key parameters:

\[ F_{\text{enhanced}} = \frac{T_2}{T} \quad \text{fract. of original time that can be improved} \]

\[ S_{\text{enhanced}} = \frac{T_2'}{T'_2} \quad \text{speedup of enhanced part} \]

\[ T' = T'_1 + T'_2 = T_1 + T_2' = T(1-F_{\text{enhanced}}) + T_2' \]

\[ = T(1-F_{\text{enhanced}}) + T(F_{\text{enhanced}}/S_{\text{enhanced}}) \quad \text{[by def. of } S_{\text{enhanced}}] \]

\[ = T(1-F_{\text{enhanced}}) + T(F_{\text{enhanced}}/S_{\text{enhanced}}) \quad \text{[by def. of } F_{\text{enhanced}}] \]

\[ \text{Amdahl’s Law: } \quad S_{\text{overall}} = \frac{T}{T'} = \frac{1}{(1-F_{\text{enhanced}}) + F_{\text{enhanced}}/S_{\text{enhanced}}} \]

Key idea: Amdahl’s law quantifies the general notion of diminishing returns. It applies to any activity, not just computer programs.

Amdahl’s law (cont)

Trip example: Suppose that for the New York to Paris leg, we now consider the possibility of taking a rocket ship (15 minutes) or a handy rip in the fabric of space-time (0 minutes):

<table>
<thead>
<tr>
<th></th>
<th>NY → Paris Time</th>
<th>Total Time</th>
<th>Speedup over 747</th>
</tr>
</thead>
<tbody>
<tr>
<td>747:</td>
<td>8.5 hours</td>
<td>16.5 hours</td>
<td>1.0</td>
</tr>
<tr>
<td>SST:</td>
<td>3.75 hours</td>
<td>11.75 hours</td>
<td>1.4</td>
</tr>
<tr>
<td>Rocket:</td>
<td>0.25 hours</td>
<td>8.25 hours</td>
<td>2.0</td>
</tr>
<tr>
<td>Rip:</td>
<td>0.0 hours</td>
<td>8.0 hours</td>
<td>2.1</td>
</tr>
</tbody>
</table>

\[ F_{\text{enhanced}} \quad \text{Max } S_{\text{overall}} \quad \text{Max } S_{\text{enhanced}} \]

Moral: It is difficult to speed up a program.

Moral++: It is easy to make premature optimizations.
**Computer System**

- Processor
- Cache
- Memory-I/O bus
- Memory controller
- Disk
- Display
- Network

**Levels in Memory Hierarchy**

<table>
<thead>
<tr>
<th></th>
<th>cache</th>
<th>virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>8 B</td>
<td>32 B</td>
</tr>
<tr>
<td>Cache</td>
<td>32 B</td>
<td>32 KB - 4 MB</td>
</tr>
<tr>
<td>Memory</td>
<td>8 KB</td>
<td>8 GB</td>
</tr>
<tr>
<td>Disk Memory</td>
<td>1 TB</td>
<td>1 TB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>size</th>
<th>speed</th>
<th>$./Mbyte</th>
<th>Block Size</th>
<th>speed</th>
<th>$./Mbyte</th>
<th>Block Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>200 B</td>
<td>1 ns</td>
<td>$60/MB</td>
<td>32 B</td>
<td>1.5 ns</td>
<td>$0.06/MB</td>
<td>32 B</td>
</tr>
<tr>
<td>Cache</td>
<td>32 B</td>
<td>10 ns</td>
<td>$0.06/MB</td>
<td>8 KB</td>
<td>40 ns</td>
<td>$0.0003/MB</td>
<td>8 KB</td>
</tr>
<tr>
<td>Memory</td>
<td>8 KB</td>
<td>300 ns</td>
<td>$0.0003/MB</td>
<td>1 TB</td>
<td>3,000,000 ns</td>
<td>$0.0003/MB</td>
<td>3,000,000 ns</td>
</tr>
<tr>
<td>Disk Memory</td>
<td>1 TB</td>
<td>10,000ns</td>
<td>$0.0003/MB</td>
<td>1 TB</td>
<td>3,000,000 ns</td>
<td>$0.0003/MB</td>
<td>3,000,000 ns</td>
</tr>
</tbody>
</table>

- Larger, slower, cheaper

**Static RAM (SRAM)**

- **Fast**
  - ~1.5 nsec access time

- **Persistent**
  - as long as power is supplied!
  - no refresh required

- **Stable**
  - High immunity to noise and environmental disturbances

- **Expensive**
  - ~$60/MByte
  - 6 transistors/bit

- Technology for caches

**Anatomy of an SRAM Cell**

- **Stable Configurations**

- **Terminology:**
  - bit line: carries data
  - word line: used for addressing

- **Write:**
  1. set bit lines to new data value
  2. raise word line to “high”
  3. set cell to new state (may involve flipping relative to old state)

- **Read:**
  1. set bit lines high
  2. set word line high
  3. see which bit line goes low
**SRAM Cell Principle**

**Inverter Amplifies**
- Negative gain
- Slope < -1 in middle
- Saturates at ends

**Inverter Pair Amplifies**
- Positive gain
- Slope > 1 in middle
- Saturates at ends

Vin → V1
V2 → Vin

**Bistable Element**

**Stability**
- Require Vin = V2
- Stable at endpoints
- Recover from perturbation
- Metastable in middle
- Fall out when perturbed

**Ball on Ramp Analogy**

**Example SRAM Configuration (16 x 8)**

**Dynamic RAM (DRAM)**

**Slower than SRAM**
- Access time ~40 nsec

**Nonpersistent**
- Every row must be accessed every ~1 ms (refreshed)

**Fragile**
- Electrical noise, light, radiation

**Cheaper than SRAM**
- ~$0.06 / MByte
- 1 transistor/bit

**Workhorse memory technology**
Anatomy of a DRAM Cell

Word Line
Bit Line
Storage Node

Writing

Reading

ΔV ~ C_{node} / C_{BL}

Addressing Arrays with Bits

Array Size
- R rows, R = 2^r
- C columns, C = 2^c
- N = R * C bits of memory

Addressing
- Addresses are n bits, where N = 2^n
- row(address) = address / C
  - leftmost r bits of address
- col(address) = address % C
  - rightmost bits of address

Example
- R = 2
- C = 4
- address = 6

Example 2-Level Decode DRAM (64Kx1)

DRAM Operation

Row Address
- Set Row address on address lines & strobe RAS
- Entire row read & stored in column latches
- Contents of row of memory cells destroyed

Column Address
- Set Column address on address lines & strobe CAS
- Access selected bit
  - READ: transfer from selected column latch to Dout
  - WRITE: Set selected column latch to Din

Rewrite
- Write back entire row
**DRAM Driving Forces**

**Capacity**
- 4X per generation
  - Square array of cells
- Typical scaling
  - Lithography dimensions 0.7X
  - Areal density 2X
  - Cell function packing 1.5X
  - Chip area 1.33X
- Scaling challenge
  - Typically $C_{\text{node}} / C_{\text{BL}} = 0.1-0.2$
  - Must keep $C_{\text{node}}$ high as shrink cell size

**Retention Time**
- Typically 16-256 ms
- Want higher for low-power applications

---

**DRAM Storage Capacitor**

**Planar Capacitor**
- Up to 1Mb
- $C$ decreases linearly with feature size

**Trench Capacitor**
- 4-256 Mb
- Lining of hole in substrate

** stacked Cell**
- > 16b
- On top of substrate
- Use high $\varepsilon$ dielectric

---

**Trench Capacitor**

**Process**
- Etch deep hole in substrate
  - Becomes reference plate
- Grow oxide on walls
  - Dielectric
- Fill with polysilicon plug
  - Tied to storage node

---

**IBM DRAM Evolution**

- IBM J. R&D, Jan/Mar ’95
- Evolution from 4 – 256 Mb
- 256 Mb uses cell with area 0.6 $\mu m^2$

**Cell Layouts**
- 4 Mb Cell Structure
- 16 Mb
- 64 Mb
- 256 Mb

---
**Mitsubishi Stacked Cell DRAM**

- IEDM '95
- Claim suitable for 1 – 4 Gb

**Technology**
- 0.14 µm process
  - Synchrotron X-ray source
- 8 nm gate oxide
- 0.29 µm² cell

**Storage Capacitor**
- Fabricated on top of everything else
- Rubidium electrodes
- High dielectric insulator
  - 50X higher than SiO₂
  - 25 nm thick
- Cell capacitance 25 femtofarads

---

**What's Inside A Disk Drive?**

- Spindle
- Arm
- Platters
- Actuator
- Electronics (including a processor and memory)
- SCSI connector

*Image courtesy of Seagate Technology*

---

**Disk Geometry**

- Disks consist of platters, each with two surfaces.
- Each surface consists of concentric rings called tracks.
- Each track consists of sectors separated by gaps.
Disk Geometry (Multiple-Platter View)

Aligned tracks form a cylinder.

Disk Capacity

<table>
<thead>
<tr>
<th>Parameter</th>
<th>18GB Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Platters</td>
<td>12</td>
</tr>
<tr>
<td>Surfaces / Platter</td>
<td>2</td>
</tr>
<tr>
<td>Number of tracks</td>
<td>6962</td>
</tr>
<tr>
<td>Number sectors / track</td>
<td>213</td>
</tr>
<tr>
<td>Bytes / sector</td>
<td>512</td>
</tr>
<tr>
<td>Total Bytes</td>
<td>18,221,948,928</td>
</tr>
</tbody>
</table>

Disk Operation

Operation
- Read or write complete sector

Seek
- Position head over proper track
- Typically 6-9ms

Rotational Latency
- Wait until desired sector passes under head
- Worst case: complete rotation
  10,025 RPM ⇒ 6 ms

Read or Write Bits
- Transfer rate depends on # bits per track and rotational speed
  - E.g., 213 * 512 bytes @10,025RPM = 18 MB/sec.
  - Modern disks have external transfer rates of up to 80 MB/sec
  - DRAM caches on disk help sustain these higher rates

Disk Performance

Getting First Byte
- Seek + Rotational latency = 7,000 – 19,000 µsec

Getting Successive Bytes
- ~ 0.06 µsec each
  - roughly 100,000 times faster than getting the first byte!

Optimizing Performance:
- Large block transfers are more efficient
- Try to do other things while waiting for first byte
  - switch context to other computing task
  - processor is interrupted when transfer completes
### Disk / System Interface

1. **Processor Signals Controller**
   - Read sector X and store starting at memory address Y
2. **Read Occurs**
   - "Direct Memory Access" (DMA) transfer
   - Under control of I/O controller
3. **I / O Controller Signals Completion**
   - Interrupts processor
   - Can resume suspended process

---

### Storage Trends

#### DRAM

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB access (ns)</td>
<td>375</td>
<td>200</td>
<td>100</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>9</td>
</tr>
<tr>
<td>typical size (MB)</td>
<td>0.064</td>
<td>0.256</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>2,000</td>
<td>8,000</td>
<td>125,000</td>
</tr>
</tbody>
</table>

#### SRAM

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB access (ns)</td>
<td>8,000</td>
<td>880</td>
<td>100</td>
<td>30</td>
<td>1</td>
<td>0.1</td>
<td>0.06</td>
<td>130,000</td>
</tr>
<tr>
<td>typical size (MB)</td>
<td>0.064</td>
<td>0.256</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>2,000</td>
<td>8,000</td>
<td>125,000</td>
</tr>
</tbody>
</table>

#### Disk

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$/MB access (ns)</td>
<td>500</td>
<td>100</td>
<td>8</td>
<td>0.3</td>
<td>0.01</td>
<td>0.005</td>
<td>0.0003</td>
<td>1,600,000</td>
</tr>
<tr>
<td>typical size (MB)</td>
<td>1</td>
<td>10</td>
<td>160</td>
<td>1,000</td>
<td>20,000</td>
<td>160,000</td>
<td>1,500,000</td>
<td>1,500,000</td>
</tr>
</tbody>
</table>

---

### CPU Clock Rates

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>8080</td>
<td>386</td>
<td>Pentium P-III</td>
<td>P-4</td>
<td>Core 2</td>
<td>Core i7</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Clock rate (MHz)</td>
<td>1</td>
<td>20</td>
<td>150</td>
<td>600</td>
<td>3300</td>
<td>2000</td>
<td>2500</td>
<td>2500</td>
</tr>
<tr>
<td>Cycle time (ns)</td>
<td>1000</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</td>
<td>0.50</td>
<td>0.4</td>
<td>2500</td>
</tr>
<tr>
<td>Cores</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Effective cycle time (ns)</td>
<td>1000</td>
<td>50</td>
<td>6</td>
<td>1.6</td>
<td>0.3</td>
<td>0.25</td>
<td>0.1</td>
<td>10,000</td>
</tr>
</tbody>
</table>

---

### The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.
The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.

- Disk seek time
- Flash SSD access time
- DRAM access time
- SRAM access time
- CPU cycle time
- Effective CPU cycle time

Year
- 1980
- 1985
- 1990
- 1995
- 2000
- 2005
- 2010

- CPU
- SRAM
- DRAM
- SSD
- Disk

Memory Technology Summary

- Cost and Density Improving at Enormous Rates
- Speed Lagging Processor Performance
- Memory Hierarchies Help Narrow the Gap:
  - Small fast SRAMS (cache) at upper levels
  - Large slow DRAMS (main memory) at lower levels
  - Incredibly large & slow disks to back it all up
- Locality of Reference Makes It All Work
  - Keep most frequently accessed data in fastest memory

The Rate of Single-Thread Performance Improvement has Decreased

Move to multi-core

Impact of Power Density on the Microprocessor Industry

- The future is not higher clock rates, but multiple cores per die.

Pat Gelsinger, ISSCC 2001

(Figure courtesy of Hennessy & Patterson, "Computer Architecture, A Quantitative Approach", VLS)
### Recent Intel Processors

#### Machine Evolution

- **386** 1985 0.3M
- **Pentium** 1993 3.1M
- **Pentium/MMX** 1997 4.5M
- **Pentium Pro** 1995 6.5M
- **Pentium III** 1999 8.2M
- **Pentium 4** 2001 42M
- **Core 2 Duo** 2006 291M
- **Core i7** 2008 731M

#### Added Features

- Instructions to support multimedia operations
  - Parallel operations on 1, 2, and 4-byte data, both integer & FP
- Instructions to enable more efficient conditional operations