Evaluating Computers: Bigger, better, faster, more?
What do you want in a computer?
What do you want in a computer?

- Low latency -- one unit of work in minimum time
  - $1/\text{latency} = \text{responsiveness}$
- High throughput -- maximum work per time
  - High bandwidth (BW)
- Low cost
- Low power -- minimum jules per time
- Low energy -- minimum jules per work
- Reliability -- Mean time to failure (MTTF)
- Derived metrics
  - responsiveness/dollar
  - BW/\$ 
  - BW/Watt 
  - Work/Jule 
  - Energy * latency -- Energy delay product
  - MTTF/\$
Latency

• This is the simplest kind of performance
• How long does it take the computer to perform a task?
  • The task at hand depends on the situation.
• Usually measured in seconds
• Also measured in clock cycles
  • Caution: if you are comparing two different systems, you must ensure that the cycle times are the same.

\[ \text{Mhz} = \frac{\text{cycles/second}}{\text{seconds}} \]
\[ \text{Cycle time} = \frac{\text{seconds}}{\text{cycle}} \]
\[ \text{Latency} = \left( \frac{\text{seconds}}{\text{cycle}} \right) \times \text{cycles} = \text{seconds} \]
Where latency matters

• Application responsiveness
  • Any time a person is waiting.
  • GUIs
  • Games
  • Internet services (from the users perspective)

• “Real-time” applications
  • Tight constraints enforced by the real world
  • Anti-lock braking systems -- “hard” real time
  • Manufacturing control
  • Multi-media applications -- “soft” real time

• The cost of poor latency
  • If you are selling computer time, latency is money.
Limits on Speedup: Amdahl’s Law

• “The fundamental theorem of performance optimization”
• Coined by Gene Amdahl (one of the designers of the IBM 360)
• Optimizations do not (generally) uniformly affect the entire program
  – The more widely applicable a technique is, the more valuable it is
  – Conversely, limited applicability can (drastically) reduce the impact of an optimization.

Always heed Amdahl’s Law!!!
It is central to many many optimization problems
Amdahl’s Law in Action

• SuperJPEG-O-Rama2010 in the wild
• PictoBench spends 33% of it’s time doing JPEG decode
• How much does JOR2k help?

<table>
<thead>
<tr>
<th></th>
<th>w/o JOR2k</th>
<th>w/ JOR2k</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30s</td>
<td>21s</td>
</tr>
<tr>
<td>JPEG Decode</td>
<td>19s</td>
<td>12s</td>
</tr>
</tbody>
</table>
Amdahl’s Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it’s time doing JPEG decode
- How much does JOR2k help?

Performance: 30/21 = 1.4x Speedup != 10x
Amdahl’s Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of its time doing JPEG decode
- How much does JOR2k help?

Performance: $\frac{30}{21} = 1.4x$ Speedup != 10x

Is this worth the 45% increase in cost?
Amdahl’s Law in Action

- SuperJPEG-O-Rama2010 in the wild
- PictoBench spends 33% of it’s time doing JPEG decode
- How much does JOR2k help?

Performance: $\frac{30}{21} = 1.4x$ Speedup $\neq 10x$

Is this worth the 45% increase in cost?
Amdahl’s Law

• The second fundamental theorem of computer architecture.
• If we can speed up $X$ of the program by $S$ times.
• Amdahl’s Law gives the total speed up, $S_{tot}$

$$S_{tot} = \frac{1}{\left(\frac{x}{S} + (1-x)\right)}.$$
Amdahl’s Law

• The second fundamental theorem of computer architecture.
• If we can speed up $X$ of the program by $S$ times
• Amdahl’s Law gives the total speed up, $S_{tot}$

$$S_{tot} = \frac{1}{\frac{x}{S} + (1-x)}.$$ 

Sanity check:

$$x = 1 \Rightarrow S_{tot} = \frac{1}{\frac{1}{S} + (1-1)} = \frac{1}{\frac{1}{S}} = S$$
Amdahl’s Corollary #1

- Maximum possible speedup, $S_{\text{max}}$

\[
S = \text{infinity}
\]

\[
S_{\text{max}} = \frac{1}{(1-x)}
\]
Amdahl’s Law Example #1

- Protein String Matching Code
  - 200 hours to run on current machine, spends 20% of time doing integer instructions
  - How much faster must you make the integer unit to make the code run 10 hours faster?
  - How much faster must you make the integer unit to make the code run 50 hours faster?

A) 1.1
B) 1.25
C) 1.75
D) 1.33
E) 10.0
F) 50.0
G) 1 million times
H) Other
Amdahl’s Law Example #1

• Protein String Matching Code
  – 200 hours to run on current machine, spends 20% of time doing integer instructions
  – How much faster must you make the integer unit to make the code run 10 hours faster?
  – How much faster must you make the integer unit to make the code run 50 hours faster?

A) 1.1  
B) 1.25  
C) 1.75  
D) 1.33  
E) 10.0  
F) 50.0  
G) 1 million times  
H) Other
Amdahl’s Law Example #2

- Protein String Matching Code
  - 4 days execution time on current machine
    - 20% of time doing integer instructions
    - 35% percent of time doing I/O
  - Which is the better tradeoff?
    - Compiler optimization that reduces number of integer instructions by 25% (assume each integer inst takes the same amount of time)
    - Hardware optimization that reduces the latency of each IO operations from 6us to 5us.
  - How much will doing both improve performance?
Answer #2

- Integer ops: Speedup = 1.04
- IO: speedup = 1.06
- Together: Speedup = 1.109
  - This one’s tricky. See discussion of applying Amdahl’s law for multiple, simultaneous optimizations later in theses slides
Amdahl’s Corollary #2

• Make the common case fast (i.e., $x$ should be large)!
  – Common == “most time consuming” not necessarily “most frequent”
  – The uncommon case doesn’t make much difference
  – Be sure of what the common case is
  – The common case changes.

• Repeat…
  – With optimization, the common becomes uncommon and vice versa.
Amdahl’s Corollary #2: Example

Common case
Amdahl’s Corollary #2: Example

Common case

7x => 1.4x
Amdahl’s Corollary #2: Example

Common case

7x => 1.4x
4x => 1.3x
Amdahl’s Corollary #2: Example

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

Total = 20/10 = 2x
Amdahl’s Corollary #2: Example

- In the end, there is no common case!
- Options:
  - Global optimizations (faster clock, better compiler)
  - Find something common to work on (i.e. memory latency)
  - War of attrition
  - Total redesign (You are probably well-prepared for this)
Amdahl’s Corollary #3

- Benefits of parallel processing
- $p$ processors
- $x\%$ is $p$-way parallizable
- maximum speedup, $S_{par}$

\[
S_{par} = \frac{1}{\frac{x}{p} + (1-x)}
\]
Amdahl’s Corollary #3

- Benefits of parallel processing
- \( p \) processors
- \( x\% \) is \( p \)-way parallizable
- maximum speedup, \( S_{par} \)

\[
S_{par} = \frac{1}{\left(\frac{x}{p} + (1-x)\right)}
\]

\( x \) is pretty small for desktop applications, even for \( p = 2 \).
Example #3

• Recent advances in process technology have quadruple the number transistors you can fit on your die.

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

• You have two choices:
  – Increase the number of processors from 1 to 4
  – Use 2 processors but add features that will allow the applications to use them for 80% of execution.

• Which will you choose?
Answer: #3

- Speedup in the first case: 1.42
- Speedup in the second case: 1.667

- The second option is better

- Make sure you understand why.
Amdahl’s Corollary #4

- Amdahl’s law for latency (L)
  - By definition
    - Speedup = oldLatency/newLatency
    - newLatency = oldLatency * 1/Speedup
  - By Amdahl’s law:
    - newLatency = old Latency * (x/S + (1-x))
    - newLatency = oldLatency/S + oldLatency*(1-x)
  - Amdahl’s law for latency
    - newLatency = oldLatency/S + oldLatency*(1-x)
Announcements

• 240a is moving!
• To cse2154. Second floor, CS building, top of the stairs.
• We will start meeting there next Monday.
Last Time

• Amdahl’s law limits speedup!
• Know Amdahl’s law!
• Make the common case fast, it’s what matters most.
Amdahl’s Non-Corollary

- Amdahl’s law does not bound slowdown
  - \( \text{newLatency} = \frac{\text{oldLatency}}{S} + \text{oldLatency} \times (1-x) \)
  - \( \text{newLatency} \) is linear in \( \frac{1}{S} \)

- Example: \( x = 0.01 \) of execution, \( \text{oldLat} = 1 \)
  - \( S = 0.001; \)
    - \( \text{Newlat} = 1000 \times \text{Oldlat} \times 0.01 + \text{Oldlat} \times (0.99) \approx 10 \times \text{Oldlat} \)
  - \( S = 0.00001; \)
    - \( \text{Newlat} = 100000 \times \text{Oldlat} \times 0.01 + \text{Oldlat} \times (0.99) \approx 1000 \times \text{Oldlat} \)

- Things can only get so fast, but they can get arbitrarily slow.
  - Do not hurt the non-common case too much!
Amdahl’s Example #4

This one is tricky

- Memory operations currently take 30% of execution time.
- A new widget called a “cache” speeds up 80% of memory operations by a factor of 4.
- A second new widget called a “L2 cache” speeds up 1/2 the remaining 20% by a factor of 2.
- What is the total speed up?
Answer in Pictures

Speed up = 1.242
Amdahl’s Pitfall: This is wrong!

• You cannot trivially apply optimizations one at a time with Amdahl’s law.

• Just the L1 cache
  • $S_1 = 4$
  • $x_1 = 0.8 \cdot 0.3$
  • $S_{totL1} = 1/(x_1/S_1 + (1-x_1))$
  • $S_{totL1} = 1/(0.8 \cdot 0.3/4 + (1-(0.8 \cdot 0.3))) = 1/(0.06 + 0.76) = 1.2195$ times

• Just the L2 cache
  • $S_{L2} = 2$
  • $x_{L2} = 0.3 \cdot (1 - 0.8)/2 = 0.03$
  • $S_{totL2'} = 1/(0.03/2 + (1-0.03)) = 1/(0.015 + 0.97) = 1.015$ times

• Combine
  • $S_{totL2} = S_{totL2'} \cdot S_{totL1} = 1.02 \cdot 1.21 = 1.237 \; \text{<- This is wrong}$

• What’s wrong? -- after we do the L1 cache, the execution time changes, so the fraction of execution that the L2 effects actually grows
Answer in Pictures

Speed up = 1.242
Multiple optimizations: The right way

- We can apply the law for multiple optimizations
- Optimization 1 speeds up $x_1$ of the program by $S_1$
- Optimization 2 speeds up $x_2$ of the program by $S_2$

$$S_{\text{tot}} = 1/(x_1/S_1 + x_2/S_2 + (1-x_1-x_2))$$

Note that $x_1$ and $x_2$ must be disjoint! i.e., $S_1$ and $S_2$ must not apply to the same portion of execution.

If not then, treat the overlap as a separate portion of execution and measure it’s speed up independently

ex: we have $x_{1\text{only}}$, $x_{2\text{only}}$, and $x_{1&2}$ and $S_{1\text{only}}$, $S_{2\text{only}}$, and $S_{1&2}$, Then

$$S_{\text{tot}} = 1/(x_{1\text{only}}/S_{1\text{only}} + x_{2\text{only}}/S_{2\text{only}} + x_{1&2}/S_{1&2} + (1-x_{1\text{only}}-x_{2\text{only}}+x_{1&2}))$$
In Summary

• Understand Amdahl’s law and how to apply it.
• It’s one of few “rote” things you must learn in this class.
### Latency versus Bandwidth

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Bandwidth (p-mph)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boeing 747</td>
<td>6.5 hours</td>
<td>610 mph</td>
<td>470</td>
<td>286,700</td>
</tr>
<tr>
<td>Concorde</td>
<td>3 hours</td>
<td>1350 mph</td>
<td>132</td>
<td>178,200</td>
</tr>
</tbody>
</table>

- **Time to run the task** (ExTime)
  - Execution time, response time, latency
- **Tasks per day, hour, week, sec, ns ...** (Performance)
  - Throughput, bandwidth
Disks: ~1980 vs. ~2000

- CDC Wren I, 1983
- 3600 RPM
- 0.03 GBytes capacity
- Tracks/Inch: 800
- Bits/Inch: 9550
- Three 5.25” platters

- Bandwidth: 0.6 MBytes/sec
- Latency: 48.3 ms
- Cache: none
Disks: ~1980 vs. ~2000

- CDC Wren I, 1983
  - 3600 RPM
  - 0.03 GBytes capacity
  - Tracks/Inch: 800
  - Bits/Inch: 9550
  - Three 5.25” platters
  - Bandwidth: 0.6 MBytes/sec
  - Latency: 48.3 ms
  - Cache: none

- Seagate 373453, 2003

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Disks: ~1980 vs. ~2000

- CDC Wren I, 1983
  - 3600 RPM
  - 0.03 GBytes capacity
  - Tracks/Inch: 800
  - Bits/Inch: 9550
  - Three 5.25” platters
  - Bandwidth: 0.6 MBytes/sec
  - Latency: 48.3 ms
  - Cache: none

- Seagate 373453, 2003
  - 15000 RPM (4X)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Disks: ~1980 vs. ~2000

- CDC Wren I, 1983
  - 3600 RPM
  - 0.03 GBytes capacity
  - Tracks/Inch: 800
  - Bits/Inch: 9550
  - Three 5.25” platters
  - Bandwidth: 0.6 MBytes/sec
  - Latency: 48.3 ms
  - Cache: none

- Seagate 373453, 2003
  - 15000 RPM (4X)
  - 73.4 GBytes (2500X)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Disks: ~1980 vs. ~2000

<table>
<thead>
<tr>
<th>1980</th>
<th>2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDC Wren I, 1983</td>
<td>Seagate 373453, 2003</td>
</tr>
<tr>
<td>3600 RPM</td>
<td>15000 RPM (4X)</td>
</tr>
<tr>
<td>0.03 GBytes capacity</td>
<td>73.4 GBytes (2500X)</td>
</tr>
<tr>
<td>Tracks/Inch: 800</td>
<td>Tracks/Inch: 64000 (80X)</td>
</tr>
<tr>
<td>Bits/Inch: 9550</td>
<td></td>
</tr>
<tr>
<td>Three 5.25” platters</td>
<td></td>
</tr>
<tr>
<td>Bandwidth: 0.6 MBytes/sec</td>
<td></td>
</tr>
<tr>
<td>Latency: 48.3 ms</td>
<td></td>
</tr>
<tr>
<td>Cache: none</td>
<td></td>
</tr>
</tbody>
</table>

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Disks: ~1980 vs. ~2000

- CDC Wren I, 1983
  - 3600 RPM
  - 0.03 GBytes capacity
  - Tracks/Inch: 800
  - Bits/Inch: 9550
  - Three 5.25” platters
  - Bandwidth: 0.6 MBytes/sec
  - Latency: 48.3 ms
  - Cache: none

- Seagate 373453, 2003
  - 15000 RPM (4X)
  - 73.4 GBytes (2500X)
  - Tracks/Inch: 64000 (80X)
  - Bits/Inch: 533,000 (60X)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Disks: ~1980 vs. ~2000

- CDC Wren I, 1983
  - 3600 RPM
  - 0.03 GBytes capacity
  - Tracks/Inch: 800
  - Bits/Inch: 9550
  - Three 5.25” platters
  - Bandwidth: 0.6 MBytes/sec
  - Latency: 48.3 ms
  - Cache: none

- Seagate 373453, 2003 (4X)
  - 15000 RPM
  - 73.4 GBytes (2500X)
  - Tracks/Inch: 64000 (80X)
  - Bits/Inch: 533,000 (60X)
  - Four 2.5” platters (in 3.5” form factor)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Disks: ~1980 vs. ~2000

- **CDC Wren I, 1983**
  - 3600 RPM
  - 0.03 GBytes capacity
  - Tracks/Inch: 800
  - Bits/Inch: 9550
  - Three 5.25” platters
  - Bandwidth: 0.6 MBytes/sec
  - Latency: 48.3 ms
  - Cache: none

- **Seagate 373453, 2003**
  - 15000 RPM (4X)
  - 73.4 GBytes (2500X)
  - Tracks/Inch: 64000 (80X)
  - Bits/Inch: 533,000 (60X)
  - Four 2.5” platters (in 3.5” form factor)
  - Bandwidth: 86 MBytes/sec (140X)
  - Latency: 48.3 ms
  - Cache: none

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Disks: ~1980 vs. ~2000

- CDC Wren I, 1983
  - 3600 RPM
  - 0.03 GBytes capacity
  - Tracks/Inch: 800
  - Bits/Inch: 9550
  - Three 5.25” platters
- Bandwidth: 0.6 MBytes/sec
- Latency: 48.3 ms
- Cache: none

- Seagate 373453, 2003
  - 15000 RPM (4X)
  - 73.4 GBytes (2500X)
  - Tracks/Inch: 64000 (80X)
  - Bits/Inch: 533,000 (60X)
  - Four 2.5” platters
    (in 3.5” form factor)
  - Bandwidth: 86 MBytes/sec (140X)
  - Latency: 5.7 ms (8X)
  - Cache: none

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
## Disks: ~1980 vs. ~2000

<table>
<thead>
<tr>
<th>1980 (CDC Wren I)</th>
<th>2000 (Seagate 373453)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RPM</strong></td>
<td><strong>RPM</strong></td>
</tr>
<tr>
<td>3600</td>
<td>15000 (4X)</td>
</tr>
<tr>
<td><strong>Capacity</strong></td>
<td><strong>Capacity</strong></td>
</tr>
<tr>
<td>0.03 GBytes</td>
<td>73.4 GBytes (2500X)</td>
</tr>
<tr>
<td><strong>Tracks/Inch</strong></td>
<td><strong>Tracks/Inch</strong></td>
</tr>
<tr>
<td>800</td>
<td>64000 (80X)</td>
</tr>
<tr>
<td><strong>Bits/Inch</strong></td>
<td><strong>Bits/Inch</strong></td>
</tr>
<tr>
<td>9550</td>
<td>533,000 (60X)</td>
</tr>
<tr>
<td><strong>Platters</strong></td>
<td><strong>Platters</strong></td>
</tr>
<tr>
<td>Three 5.25”</td>
<td>Four 2.5” (in 3.5” form factor)</td>
</tr>
<tr>
<td><strong>Bandwidth</strong></td>
<td><strong>Bandwidth</strong></td>
</tr>
<tr>
<td>0.6 MBytes/sec</td>
<td>86 MBytes/sec (140X)</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td><strong>Latency</strong></td>
</tr>
<tr>
<td>48.3 ms</td>
<td>5.7 ms (8X)</td>
</tr>
<tr>
<td><strong>Cache</strong></td>
<td><strong>Cache</strong></td>
</tr>
<tr>
<td>None</td>
<td>8 MBytes</td>
</tr>
</tbody>
</table>

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Memory: 1980 versus 2000

- 1980 DRAM (asynchronous)
- 0.06 Mbits/chip
- 64,000 xtors, 35 mm²
- 16-bit data bus per module, 16 pins/chip
- BW: 13 Mbytes/sec
- Latency: 225 ns
- (no block transfer)
Memory: 1980 versus 2000

- 1980 DRAM (asynchronous)
  - 0.06 Mbits/chip
  - 64,000 xtors, 35 mm²
  - 16-bit data bus per module, 16 pins/chip
  - BW: 13 Mbytes/sec
  - Latency: 225 ns
  - (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Memory: 1980 versus 2000

- 1980 DRAM (asynchronous)
  - 0.06 Mbits/chip
  - 64,000 xtors, 35 mm²
  - 16-bit data bus per module, 16 pins/chip
  - BW: 13 Mbytes/sec
  - Latency: 225 ns
  - (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
  - 256.00 Mbits/chip (4000X)
Memory: 1980 versus 2000

- 1980 DRAM (asynchronous)
  - 0.06 Mbits/chip
  - 64,000 x tors, 35 mm²
  - 16-bit data bus per module, 16 pins/chip
  - BW: 13 Mbytes/sec
  - Latency: 225 ns
  - (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
  - 256.00 Mbits/chip (4000X)
  - 256,000,000 x tors, 204 mm²

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Memory: 1980 versus 2000

- 1980 DRAM (asynchronous)
  - 0.06 Mbits/chip
  - 64,000 xtors, 35 mm²
  - 16-bit data bus per module, 16 pins/chip
  - BW: 13 Mbytes/sec
  - Latency: 225 ns
  - (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
  - 256.00 Mbits/chip (4000X)
  - 256,000,000 xtors, 204 mm²
  - 64-bit data bus per DIMM, 66 pins/chip (4X)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Memory: 1980 versus 2000

• 1980 DRAM (asynchronous)
  • 0.06 Mbits/chip
  • 64,000 x tors, 35 mm²
  • 16-bit data bus per module;
    16 pins/chip
  • BW: 13 Mbytes/sec
  • Latency: 225 ns
  • (no block transfer)

• 2000 Double Data Rate Synchr. (clocked) DRAM
  • 256.00 Mbits/chip  (4000X)
  • 256,000,000 x tors, 204 mm²
  • 64-bit data bus per
    DIMM, 66 pins/chip  (4X)
  • BW: 1600 Mbytes/sec  (120X)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Memory: 1980 versus 2000

- **1980 DRAM (asynchronous)**
  - 0.06 Mbits/chip
  - 64,000 xtors, 35 mm$^2$
  - 16-bit data bus per module, 16 pins/chip
  - BW: 13 Mbytes/sec
  - Latency: 225 ns
  - (no block transfer)

- **2000 Double Data Rate Synchr. (clocked) DRAM**
  - 256.00 Mbits/chip (4000X)
  - 256,000,000 xtors, 204 mm$^2$
  - 64-bit data bus per DIMM, 66 pins/chip (4X)
  - BW: 1600 Mbytes/sec (120X)
  - Latency: 52 ns (4X)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Memory: 1980 versus 2000

- 1980 DRAM (asynchronous)
  - 0.06 Mbits/chip
  - 64,000 x tors, 35 mm²
  - 16-bit data bus per module;
    16 pins/chip
  - BW: 13 Mbytes/sec
  - Latency: 225 ns
  - (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
  - 256.00 Mbits/chip (4000X)
  - 256,000,000 x tors, 204 mm²
  - 64-bit data bus per DIMM, 66 pins/chip (4X)
  - BW: 1600 Mbytes/sec (120X)
  - Latency: 52 ns (4X)
  - Block transfers (page mode)
CPUs: ~1980 vs. ~2000

- 1982 Intel 80286
- 12.5 MHz
- 2 MIPS (peak)
- Latency 320 ns
- 134,000 xtors, 47 mm²
- 16-bit data bus, 68 pins
- Microcode interpreter, separate FPU chip
- (no caches)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
CPUs: ~1980 vs. ~2000

- 1982 Intel 80286
- 12.5 MHz
- 2 MIPS (peak)
- Latency 320 ns
- 134,000 xtors, 47 mm$^2$
- 16-bit data bus, 68 pins
- Microcode interpreter, separate FPU chip
- (no caches)
CPUs: ~1980 vs. ~2000

- 1982 Intel 80286
  - 12.5 MHz
  - 2 MIPS (peak)
  - Latency 320 ns
  - 134,000 xtors, 47 mm²
  - 16-bit data bus, 68 pins
  - Microcode interpreter, separate FPU chip
  - (no caches)

- 2001 Intel Pentium 4

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
CPUs: ~1980 vs. ~2000

- 1982 Intel 80286
  - 12.5 MHz
  - 2 MIPS (peak)
  - Latency 320 ns
  - 134,000 xtors, 47 mm²
  - 16-bit data bus, 68 pins
  - Microcode interpreter, separate FPU chip
  - (no caches)

- 2001 Intel Pentium 4
  - 1500 MHz (120X)
CPUs: ~1980 vs. ~2000

• 1982 Intel 80286
  • 12.5 MHz
  • 2 MIPS (peak)
  • Latency 320 ns
  • 134,000 xtors, 47 mm²
  • 16-bit data bus, 68 pins
  • Microcode interpreter, separate FPU chip
  • (no caches)

• 2001 Intel Pentium 4
  • 1500 MHz
  • 4500 MIPS (peak)
  • (120X)
  • 2250X

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
CPUs: ~1980 vs. ~2000

1982 Intel 80286
- 12.5 MHz
- 2 MIPS (peak)
- Latency 320 ns
- 134,000 xtors, 47 mm²
- 16-bit data bus, 68 pins
- Microcode interpreter, separate FPU chip
- (no caches)

2001 Intel Pentium 4
- 1500 MHz (120X)
- 4500 MIPS (peak) (2250X)
- Latency 15 ns (20X)

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
CPUs: ~1980 vs. ~2000

- 1982 Intel 80286
  - 12.5 MHz
  - 2 MIPS (peak)
  - Latency 320 ns
  - 134,000 xtors, 47 mm²
  - 16-bit data bus, 68 pins
  - Microcode interpreter, separate FPU chip
  - (no caches)

- 2001 Intel Pentium 4
  - 1500 MHz (120X)
  - 4500 MIPS (peak) (2250X)
  - Latency 15 ns (20X)
  - 42,000,000 xtors, 217 mm²

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
### CPUs: ~1980 vs. ~2000

<table>
<thead>
<tr>
<th>Year</th>
<th>Model</th>
<th>Frequency</th>
<th>Peak Performance</th>
<th>Latency</th>
<th>Transistors</th>
<th>Dimensions</th>
<th>Data Bus Width</th>
<th>Pins</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1982</td>
<td>Intel 80286</td>
<td>12.5 MHz</td>
<td>2 MIPS (peak)</td>
<td>320 ns</td>
<td>134,000</td>
<td>47 mm²</td>
<td>16-bit</td>
<td>68</td>
<td>Microcode interpreter, separate FPU chip (no caches)</td>
</tr>
<tr>
<td>2001</td>
<td>Intel Pentium 4</td>
<td>1500 MHz</td>
<td>4500 MIPS (peak)</td>
<td>15 ns</td>
<td>42,000,000</td>
<td>217 mm²</td>
<td>64-bit</td>
<td>423</td>
<td>64-bit data bus, separate FPU chip (no caches)</td>
</tr>
</tbody>
</table>
CPUs: ~1980 vs. ~2000

- 1982 Intel 80286
  - 12.5 MHz
  - 2 MIPS (peak)
  - Latency 320 ns
  - 134,000 xtors, 47 mm²
  - 16-bit data bus, 68 pins
  - Microcode interpreter, separate FPU chip
  - (no caches)

- 2001 Intel Pentium 4
  - 1500 MHz (120X)
  - 4500 MIPS (peak) (2250X)
  - Latency 15 ns (20X)
  - 42,000,000 xtors, 217 mm²
  - 64-bit data bus, 423 pins
  - 3-way superscalar, Dynamic translate to RISC, Superpipelined (22 stage), Out-of-Order execution

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
CPUs: ~1980 vs. ~2000

- 1982 Intel 80286
  - 12.5 MHz
  - 2 MIPS (peak)
  - Latency 320 ns
  - 134,000 xtors, 47 mm²
  - 16-bit data bus, 68 pins
  - Microcode interpreter, separate FPU chip
  - (no caches)

- 2001 Intel Pentium 4
  - 1500 MHz
  - 4500 MIPS (peak)
  - Latency 15 ns
  - 42,000,000 xtors, 217 mm²
  - 64-bit data bus, 423 pins
  - 3-way superscalar, Dynamic translate to RISC, Superpipelined (22 stage), Out-of-Order execution
  - On-chip 8KB Data caches, 96KB Instr. Trace cache, 256KB L2 cache

Adapted from Patterson, CSE 252 Sp06 Lecture 2 © 2006 UC Berkeley.
Latency-BW Trade-offs

- Often, increasing latency for one task and increase BW for many tasks.
  - Think of waiting in line for one of 4 bank tellers
  - If the line is empty, your response time is minimized, but throughput is low because utilization is low.
  - If there is always a line, you wait longer (your latency goes up), but there is always work available for tellers.

- Much of computer performance is about scheduling work onto resources
  - Network links.
  - Memory ports.
  - Processors, functional units, etc.
  - IO channels.
  - Increasing contention for these resources generally increases throughput but hurts latency.
Bandwidth, Latency, Utilization -- Choose two

• Useful results from queueing theory
• High bandwidth + Low latency (little waiting)
  • Poor utilization -- a ticket window must always be available to service a new request, so utilization will be poor
• High bandwidth + High utilization
  • Latency will be long, since all ticket windows are always occupied
• Low latency + High utilization
  • Very hard to achieve, unless the arrival times and service times for each request are very uniform.
Reliability Metrics

- Mean time to failure (MTTF)
  - Average time before a system stops working
  - Very complicated to calculate for complex systems
- Why would a processor fail?
  - Electromigration
  - High-energy particle strikes
  - Cracks due to heat/cooling
- It used to be that processors would last longer than their useful life time. This is becoming less true.
Power/Energy Metrics

• **Energy == joules**
  - You buy electricity in joules.
  - Battery capacity is in joules
  - To minimize operating costs, minimize energy
  - You can also think of this as the amount of work that computer must actually do

• **Power == joules/sec**
  - Power is how fast your machine uses joules
  - It determines battery life
  - It is also determines how much cooling you need. Big systems need 0.3-1 Watt of cooling for every watt of compute.
Metrics in the wild

- Millions of instructions per second (MIPS)
- Floating point operations per second (FLOPS)
- Giga-(integer)operations per second (GOPS)

Why are these all bandwidth metric?
- Peak bandwidth is workload independent, so these metrics describe a hardware capability
- When you see these, they are generally GNTE (Guaranteed not to exceed) numbers.
More Complex Metrics

• For instance, want low power and low latency
  • Power * Latency
• More concerned about Power?
  • Power^2 * Latency
• High bandwidth, low cost?
  • (MB/s)/$
• In general, put the good things in the numerator, the bad things in the denominator.
  • MIPS^2/W
Stationwagon Digression

- IPv6 Internet 2: 272,400 terabit-meters per second
  - 585GB in 30 minutes over 30,000 Km
  - 9.08 Gb/s

- Subaru outback wagon
  - Max load = 408Kg
  - 21Mpg

- Deskstar 5K3000
  - 3TB/Drive
  - 0.680Kg

- 1.8PB

- Legal speed: 75MPH (33.3 m/s)

- BW = 16 Gb/s

- Latency = 10 days

- 479,520 terabit-meters per second
Prius Digression

- IPv6 Internet 2: 272,400 terabit-meters per second
  - 585GB in 30 minutes over 30,000 Km
  - 9.08 Gb/s

- My Toyota Prius
  - Max load = 374Kg
  - 44Mpg (2x power efficiency)

MHX2 BT 300
- 300GB/Drive
- 0.135Kg

- 831TB
- Legal speed: 75MPH (33.3 m/s)
- BW = 14.4 Gb/s
- Latency = 10 days
- 439,560 terabit-meters per second (9% performance hit)
Benchmarks: Standard Candles for Performance

• It’s hard to convince manufacturers to run your program (unless you’re a BIG customer)
• A benchmark is a set of programs that are representative of a class of problems.
• To increase predictability, collections of benchmark applications, called benchmark suites, are popular
  – “Easy” to set up
  – Portable
  – Well-understood
  – Stand-alone
  – Standardized conditions
  – These are all things that real software is not.
• Benchmarks are created by a human (i.e., a political) process. The creators may not have your best interests at heart.
Classes of benchmarks

• **Microbenchmark** – measure one feature of system
  – e.g. memory accesses or communication speed

• **Kernels** – most compute-intensive part of applications
  – e.g. Linpack and NAS kernel b’marks (for supercomputers)

• **Full application:**
  – SpecInt / SpecFP (int and float) (for Unix workstations)
  – Other suites for databases, web servers, graphics,...

• I use benchmarks all the time. They are indispensable for research
  – All of them are flawed (although some are better than others)
  – You must understand their limitations and how those limitations affect your system.