Modern Computer Architecture

Lecture1 Fundamentals of Quantitative Design and Analysis (II)

Hongbin Sun

国家集成电路人才培养基地

Xi’an Jiaotong University
1.4 Trends in Technology

Logic: transistor density 35%/year, die size 10-20%/year, capacity 40-55%/year
DRAM: capacity 25-40%/year, and maybe stop in the middle of this decade.
Processor Technology

65nm 45nm 32nm 22nm 15nm 11nm 8nm
MANUFACTURING DEVELOPMENT RESEARCH

- 3-D Computational Lithography
- III-V Nanowire
- Optical Interconnect
- Copper (Cu) Barrier
- High-k Dense SRAM
- Carbon Nanotube FET
Semiconductor Flash

Conventional FG NAND cell has been scaled down over 18 years.

- 0.7um → 2Xnm (Cell size: ~1/2000)
- 1.5year/gen. (18 years / 12 gen.)
Magnetic Disk Technology

Areal Density of Magnetic HDD and DRAM

25% = 2X per 3 years
40  2
60  1.5

25% CGR

3380E  3380K
40M  1M

40% CGR

60% CGR

Travelstar 18GT
Travelstar 6GT
Deskstar 37GP
Ultrastar 36Z6
Ultrastar 36XP
Ultrastar 18XP
Ultrastar 2XP
Corsair

Year


IBM

Ed Grochowski at Almaden
Network Technology

Bus (core<=8) → Ring (cores<10) → Crossbar (cores<16) → Crossbar (cores<=100) → Optical Network (cores >100)

IBM Cell (8-core) → Sun SPAC T1 (8-core) → Intel TeraFlop80 (80-core) → Intel SCC48 (48-core) → Tilera (64/100 cores) → Sun SPAC T5 (16-core) → ATAC (1024 core)
Bandwidth over Latency

- **Bandwidth** or throughput is the total work done in a given time.

- **Latency** or response time is the time between the start and the completion of an event.

Bandwidth has outpaced latency and will likely continue to do so.
Technology Scaling

• The only constant in VLSI is constant change

• Feature size shrinks by 30% every 2-3 years
  – Transistors become cheaper
  – Transistors become faster
  – Wires do not improve (and may get worse)

• Scale factor $S$
  – Typically $S = \sqrt{2}$
  – Technology nodes

![Graph showing feature size shrinking over years with scale factor $S$]
Device Scaling Assumption

• What changes between technology nodes?

• Constant Field Scaling
  – All dimensions \((x, y, z \Rightarrow W, L, t_{ox})\)
  – Voltage \((V_{DD})\)
  – Doping levels

• Lateral Scaling
  – Only gate length \(L\)
  – Often done as a quick gate shrink \((S = 1.05)\)
Device Scaling

Table 4.15 Influence of scaling on MOS device characteristics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sensitivity</th>
<th>Constant Field</th>
<th>Lateral</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling Parameters</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length: $L$</td>
<td>1/$S$</td>
<td>1/$S$</td>
<td>$S$</td>
</tr>
<tr>
<td>Width: $W$</td>
<td>1/$S$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Gate oxide thickness: $t_{ox}$</td>
<td>1/$S$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Supply voltage: $V_{DD}$</td>
<td>1/$S$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Threshold voltage: $V_{th}$, $V_{tp}$</td>
<td>1/$S$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Substrate doping: $N_A$</td>
<td>$S$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Device Characteristics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\beta$</td>
<td>$W \frac{1}{L : t_{ox}}$</td>
<td>$S$</td>
<td>$S$</td>
</tr>
<tr>
<td>Current: $I_{ds}$</td>
<td>$\beta(V_{DD} - V_t)^2$</td>
<td>1/$S$</td>
<td>$S$</td>
</tr>
<tr>
<td>Resistance: $R$</td>
<td>$\frac{V_{DD}}{I_{ds}}$</td>
<td>1</td>
<td>1/$S$</td>
</tr>
<tr>
<td>Gate capacitance: $C$</td>
<td>$\frac{WL}{t_{ox}}$</td>
<td>1/$S$</td>
<td>1/$S$</td>
</tr>
<tr>
<td>Gate delay: $\tau$</td>
<td>$RC$</td>
<td>1/$S$</td>
<td>1/$S^2$</td>
</tr>
<tr>
<td>Clock frequency: $f$</td>
<td>$1/\tau$</td>
<td>$S$</td>
<td>$S^2$</td>
</tr>
<tr>
<td>Dynamic power dissipation (per gate): $P$</td>
<td>$CV^2f$</td>
<td>1/$S^2$</td>
<td>$S$</td>
</tr>
<tr>
<td>Chip area: $A$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power density</td>
<td>$P/A$</td>
<td>1</td>
<td>$S$</td>
</tr>
<tr>
<td>Current density</td>
<td>$I_{ds}/A$</td>
<td>$S$</td>
<td>$S$</td>
</tr>
</tbody>
</table>

- Gate capacitance per micron is nearly independent of process
- But ON resistance * micron improves with process
- Gates get faster with scaling (good)
- Dynamic power goes down with scaling (good)
- Current density goes up with scaling (bad)
- Velocity saturation makes lateral scaling unsustainable
Results of Device Scaling

- The fact that transistor count improves quadratically with a linear improvement in transistor performance is both the challenge and the opportunity.
  - 4-bit, 8-bit, 16-bit, 32-bit, to 64-bit microprocessor.
  - Multiple processors per chip
  - Wider SIMD units
  - Speculative execution
  - Caches
Wire Scaling Assumption

• Wire thickness
  – Hold constant vs. reduce in thickness

• Wire length
  – Local / scaled interconnect
  – Global interconnect
    • Die size scaled by $D_c \approx 1.1$
# Wire Scaling

## Table 4.16 Influence of scaling on interconnect characteristics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Sensitivity</th>
<th>Reduced Thickness</th>
<th>Constant Thickness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scaling Parameters</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Width: $\omega$</td>
<td></td>
<td>1/$S$</td>
<td></td>
</tr>
<tr>
<td>Spacing: $s$</td>
<td></td>
<td>1/$S$</td>
<td></td>
</tr>
<tr>
<td>Thickness: $t$</td>
<td></td>
<td>1/$S$</td>
<td>1</td>
</tr>
<tr>
<td>Interlayer oxide height: $h$</td>
<td></td>
<td>1/$S$</td>
<td></td>
</tr>
<tr>
<td><strong>Local/Scaled Interconnect Characteristics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length: $l$</td>
<td></td>
<td>1/$S$</td>
<td></td>
</tr>
<tr>
<td>Unrepeated wire RC delay</td>
<td>$P_{t_{wu}}$</td>
<td>1</td>
<td>between 1/$S$, 1</td>
</tr>
<tr>
<td>Repeated wire delay</td>
<td>$l_{t_{wr}}$</td>
<td>$\sqrt{1/S}$</td>
<td>between 1/$S$, $\sqrt{1/S}$</td>
</tr>
<tr>
<td><strong>Global Interconnect Characteristics</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Length: $l$</td>
<td></td>
<td>$D_c$</td>
<td></td>
</tr>
<tr>
<td>Unrepeated wire RC delay</td>
<td>$P_{t_{wu}}$</td>
<td>$S^2D_c^2$</td>
<td>between $SD_c^2$, $S^2D_c^2$</td>
</tr>
<tr>
<td>Repeated wire delay</td>
<td>$l_{t_{wr}}$</td>
<td>$D_c\sqrt{S}$</td>
<td>between $D_c$, $D_c\sqrt{S}$</td>
</tr>
</tbody>
</table>
Observations

• Capacitance per micron is remaining constant
  – About 0.2 fF/µm
  – Roughly 1/10 of gate capacitance

• Local wires are getting faster
  – Not quite tracking transistor improvement
  – But not a major problem

• Global wires are getting slower
  – No longer possible to cross chip in one cycle

• Wire delay has become a major design limitation for large integrated circuits and is often more critical than transistor switching delay.
ITRS

- Semiconductor Industry Association forecast
  - Intl. Technology Roadmap for Semiconductors

<table>
<thead>
<tr>
<th>Year</th>
<th>2009</th>
<th>2012</th>
<th>2015</th>
<th>2018</th>
<th>2021</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature size (nm)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$L_{\text{gate}}$ (nm)</td>
<td>20</td>
<td>14</td>
<td>10</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>$V_{\text{DD}}$ (V)</td>
<td>1.0</td>
<td>0.9</td>
<td>0.8</td>
<td>0.7</td>
<td>0.65</td>
</tr>
<tr>
<td>Billions of transistors/die</td>
<td>1.5</td>
<td>3.1</td>
<td>6.2</td>
<td>12.4</td>
<td>24.7</td>
</tr>
<tr>
<td>Wiring levels</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>Maximum power (W)</td>
<td>198</td>
<td>198</td>
<td>198</td>
<td>198</td>
<td>198</td>
</tr>
<tr>
<td>DRAM capacity (Gb)</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Flash capacity (Gb)</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
</tbody>
</table>

2007
1.5 Trends in Power and Energy

- Three primary concerns about power and energy:
  - What is the maximum power a processor ever requires?
  - What is the sustained power consumption?
    - Thermal design power (TPD)
  - Energy and energy efficiency

- Dynamic power:
  \[
  \text{Power}_{\text{dynamic}} = 0.5 \times \text{CapacitiveLoad} \times \text{Voltage}^2 \times \text{FrequencySwitched}
  \]

- For mobile devices, energy is the better metric
  \[
  \text{Energy}_{\text{dynamic}} = \text{CapacitiveLoad} \times \text{Voltage}^2
  \]

- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy
Dynamic Power and Energy

• Capacitive load a function of number of transistors connected to output and technology, which determines capacitance of wires and transistors

• Dropping voltage helps both, so went from 5V to 1V

• To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)

• Example: Suppose 15% reduction in voltage results in a 15% reduction in frequency. What is impact on dynamic power?

\[
\text{Power}_{\text{dynamic}} = \frac{1}{2} \times \text{CapacitiveLoad} \times \text{Voltage}^2 \times \text{FrequencySwitched}
\]

\[
= \frac{1}{2} \times .85 \times \text{CapacitiveLoad} \times (.85\times\text{Voltage})^2 \times \text{FrequencySwitched}
\]

\[
= (.85)^3 \times \text{OldPower}_{\text{dynamic}}
\]

\[
\approx 0.6 \times \text{OldPower}_{\text{dynamic}}
\]
Static Power and Energy

- Because leakage current flows even when a transistor is off, now static power becomes important too.

\[
\text{Power}_{\text{static}} = \text{Current}_{\text{static}} \times \text{Voltage}
\]

- Leakage current increases in processors with smaller transistor sizes
- Increasing the number of transistors increases power even if they are turned off
- In 2006, goal for leakage is 25% of total power consumption; high performance designs at 40%
- Very low power systems even gate voltage to inactive modules to control loss due to leakage
Power dissipation

Lead Microprocessors power continues to increase
Power will be a major problem

Power delivery and dissipation will be prohibitive
Power density

Power density too high to keep junctions at low temp
Low Power Tech. in Modern Processor

- **Do nothing well**: turn off the clock of inactive modules to save energy and dynamic power.

- **Dynamic Voltage-Frequency Scaling (DVFS)**

- **Design for typical case**: offer low power modes to save energy.

- **Overclocking**: Intel started offering *Turbo mode* in 2008.
1.6 Trends in Cost

- **Time**: The price drops with time, learning curve increases.

- **Volume**: The price drops with volume increase.

- **Commodities**: Many manufacturers produce the same product, Competition brings prices down.

In the past 25 years, much of the personal computer industry has become a commodity business.
Die and Wafer

Photograph of an Intel Core i7 microprocessor die. The dimensions are 18.9 mm by 13.6 mm (257 mm²) in a 45 nm process. (Courtesy Intel.)

This 300 mm wafer contains 280 full sandy bridge dies, each 20.7 by 10.5 mm in a 32 nm processor.
Cost of Integrated Circuit

\[ \text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}} \]

\[ \text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die yield}} \]

\[ \text{Dies per wafer} = \frac{\pi \times (\text{Wafer diam} / 2)^2}{\text{Die Area}} - \frac{\pi \times \text{Wafer diam}}{\sqrt{2} \times \text{Die Area}} - \text{Test dies} \]

\[ \text{Die Yield} = \text{Wafer yield} \times \left\{1 + \frac{\text{Defects per unit area} \times \text{Die Area}}{\alpha} \right\} \]

Die Cost goes roughly with die area\(^4\)
Cost of Integrated Circuit

• **Example:** Find the number of dies per 300mm wafer for a die that is 1.5 cm on a side and for a die that is 1.0 on a side.

• **Example:** Find the die yield for dies that are 1.5cm on a side and 1.0cm on a side, assuming a defect density of 0.031 per cm² and N is 13.5.
Cost of Manufacturing vs. Cost of Operation

- Google’s data center electricity use is about 0.01% of total worldwide electricity use and less than 1 percent of global data center electricity use in 2010
- Green Power
1.7 Dependability

- How decide when a system is operating properly?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- Systems alternate between 2 states of service with respect to an SLA:
  1. Service accomplishment, where the service is delivered as specified in SLA
  2. Service interruption, where the delivered service is different from the SLA
- Failure = transition from state 1 to state 2
- Restoration = transition from state 2 to state 1
Module reliability = measure of continuous service accomplishment (or time to failure). 2 metrics
1. Mean Time To Failure (MTTF) measures Reliability
2. Failures In Time (FIT) = 1/MTTF, the rate of failures
   • Traditionally reported as failures per billion hours of operation

Mean Time To Repair (MTTR) measures Service Interruption
   • Mean Time Between Failures (MTBF) = MTTF+MTTR

Module availability measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)

Module availability = MTTF / (MTTF + MTTR)
1.8 Measuring, Reporting, and Summarizing Performance

<table>
<thead>
<tr>
<th>Plane</th>
<th>DC to Paris</th>
<th>Speed</th>
<th>Passengers</th>
<th>Throughput (pmph)</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>BAD/Sud 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
Performance Comparison

• "X is n times faster than Y" means
  \[
  \frac{\text{ExTime}(Y)}{\text{ExTime}(X)} = \frac{\text{Performance}(X)}{\text{Performance}(Y)}
  \]

• Speed of Concorde vs. Boeing 747
• Throughput of Boeing 747 vs. Concorde
Benchmarks

• Real applications and application suites
  – E.g., SPEC CPU2000, SPEC2006, TPC-C, TPC-H

• Kernels
  – “Representative” parts of real applications
  – Easier and quicker to set up and run
  – Often not really representative of the entire app

• Toy programs, synthetic benchmarks, etc.
  – Not very useful for reporting
  – Sometimes used to test/stress specific functions/features
### SPEC CPU (Integer)

<table>
<thead>
<tr>
<th>SPEC2006 benchmark description</th>
<th>SPEC2006</th>
<th>SPEC2000</th>
<th>SPEC95</th>
<th>SPEC92</th>
<th>SPEC89</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU C compiler</td>
<td></td>
<td></td>
<td></td>
<td>gcc</td>
<td></td>
</tr>
<tr>
<td>Interpreted string processing</td>
<td></td>
<td></td>
<td></td>
<td>perl</td>
<td>espresso</td>
</tr>
<tr>
<td>Combinatorial optimization</td>
<td></td>
<td>mcf</td>
<td></td>
<td></td>
<td>li</td>
</tr>
<tr>
<td>Block-sorting compression</td>
<td></td>
<td>bzip2</td>
<td></td>
<td></td>
<td>eqntott</td>
</tr>
<tr>
<td>Go game (AI)</td>
<td>go</td>
<td>vortex</td>
<td></td>
<td>go</td>
<td>sc</td>
</tr>
<tr>
<td>Video compression</td>
<td>h264avc</td>
<td>gzip</td>
<td></td>
<td>ijpeg</td>
<td></td>
</tr>
<tr>
<td>Games/path finding</td>
<td>astar</td>
<td>eon</td>
<td></td>
<td></td>
<td>m88ksim</td>
</tr>
<tr>
<td>Search gene sequence</td>
<td>hmmer</td>
<td>twolf</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quantum computer simulation</td>
<td>libquantum</td>
<td>vortex</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete event simulation library</td>
<td>omnetpp</td>
<td>vpr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chess game (AI)</td>
<td>sjeng</td>
<td>crafty</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML parsing</td>
<td>xalancbmk</td>
<td>parser</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“Representative” applications keeps growing with time!
SPEC CPU (Floating Point)

- CFD/blast waves
- Numerical relativity
- Finite element code
- Differential equation solver framework
- Quantum chemistry
- EM solver (freq/time domain)
- Scalable molecular dynamics (~NAMD)
- Lattice Boltzmann method (fluid/air flow)
- Large eddie simulation/turbulent CFD
- Lattice quantum chromodynamics
- Molecular dynamics
- Image ray tracing
- Spare linear algebra
- Speech recognition
- Quantum chemistry/object oriented
- Weather research and forecasting
- Magneto hydrodynamics (astrophysics)

- bwaves
- cactusADM
- calculix
- dealll
- gamess
- GemsFDTD
- gromacs
- lbm
- LESlie3d
- milc
- namd
- povray
- soplex
- sphinx3
- tonto
- wrf
- zeusmp

- apsi
- mgrid
- applu
- turb3d
- wupwise
- apply
- galgel
- mesa
- art
- equake
- facerec
- ammp
- lucas
- fma3d
- sixtrack
- swim
- hydro2d
- su2cor
- wave5
- fpppp
- tomcatv
- doduc
- nasa7
- spice
- matrix300

© 2007 Elsevier, Inc. All rights reserved.
TPC Benchmarks

• Measure transaction-processing throughput

• Benchmarks for different scenarios
  – TPC-C: warehouses and sales transactions
  – TPC-H: ad-hoc decision support
  – TPC-W: web-based business transactions

• Difficult to set up and run on a simulator
  – Requires full OS support, a working DBMS
  – Long simulations to get stable results
High performance
Very expensive!
1.9 Quantitative Principles of Computer Design

- Take Advantage of Parallelism
  - Data level parallelism
  - Request level parallelism
  - Instruction level parallelism

- Principle of Locality, program property
  - Temporal locality
  - Spatial locality

- Focus on the Common Case
Amdahl’s Law (I)

- Amdahl’s law defines the *speedup* that can be gained by using a particular feature.

\[
\text{Speedup} = \frac{\text{Execution Time without Enhancement}}{\text{Execution Time with Enhancement}} = \frac{\text{Execution Time}_{\text{old}}}{\text{Execution Time}_{\text{new}}}
\]

**What if enhancement does not enhance everything?**

\[
\text{Speedup} = \frac{\text{Execution Time without using Enhancement at all}}{\text{Execution Time using Enhancement when Possible}}
\]

\[
\text{Execution Time}_{\text{new}} = \text{Execution Time}_{\text{old}} \times \left(1 - \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}}\right) + \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}}
\]

Overall Speedup = \[
\frac{1}{\left(1 - \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}}\right) + \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}}}
\]

**Caution: fraction of What?**
Amdahl’s Law (II)

• Make the Common Case Fast

\[
\text{Overall Speedup} = \frac{1}{(1 - \text{Fraction}_{\text{Enhanced}}) + \frac{\text{Fraction}_{\text{Enhanced}}}{\text{Speedup}_{\text{Enhanced}}}}
\]

\[
\text{Speedup}_{\text{Enhanced}} = 20 \quad \text{Fraction}_{\text{Enhanced}} = 0.1 \quad \text{VS} \quad \text{Speedup}_{\text{Enhanced}} = 1.2 \quad \text{Fraction}_{\text{Enhanced}} = 0.9
\]

\[
\text{Speedup} = \frac{1}{(1 - 0.1) + \frac{0.1}{20}} = 1.105
\]

\[
\text{Speedup} = \frac{1}{(1 - 0.9) + \frac{0.9}{1.2}} = 1.176
\]

Important: Principle of locality

Approx. 90% of the time spent in 10% of the code
Amdahl’s Law (III)

- Diminishing Returns

**Generation 1**

- Total Execution Time
  - Green Phase
  - Blue Phase
  - Speedup\text{Green} = 2
  - Fraction\text{Green} = \frac{1}{2}

**Generation 2**

- Speedup\text{Overall} = 1.33
  - Total Execution Time
  - Green
  - Blue
  - Speedup\text{Green} = 2
  - Fraction\text{Green} = \frac{1}{3}

**Generation 3**

- Speedup\text{Overall} = 1.2
  - Total Execution Time
  - Blue

Generation 2 over Generation 1

Generation 3 over Generation 2
Car Analogy

• From GT to Mall of Georgia (35mi)
  – you’ve got a “Turbo” for your car, but can only use on highway

• Spaghetti Junction to Mall of GA (23mi)
  – avg. speed of 60mph
  – avg. speed of 120mph with Turbo

• GT to Spaghetti junction (12 mi)
  – stuck in bad rush hour traffic
    • avg. speed of 5 mph

Turbo gives 100% speedup across 66% of the distance… … but only results in <10% reduction on total trip time (which is a <11% speedup)
Now Consider Price-Performance

• Without Turbo
  – Car costs $8,000 to manufacture
  – Selling price is $12,000 → $4K profit per car
  – If we sell 10,000 cars, that’s $40M in profit

• With Turbo
  – Car costs extra $3,000
  – Selling price is $16,000 → $5K profit per car
  – But only a few gear heads buy the car:
    • We only sell 400 cars and make $2M in profit
CPU Design is Similar

• What does it cost me to add some performance enhancement?

• How much effective performance do I get out of it?
  – 100% speedup for small fraction of time wasn’t a big win for the car example

• How much more do I have to charge for it?
  – Extra development, testing, marketing costs

• How much more can I charge for it?
  – Does the market even care?

• How does the price change affect volume?
The Processor Performance Equation

\[
\text{CPU time} = \text{CPU Clock Cycles} \times \text{Clock cycle time}
\]

\[
\text{CPU time} = \text{Instruction Count} \times \text{Cycles Per Instruction} \times \text{Clock cycle time}
\]

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
\]

CPI: clock cycles per instruction

\[
\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}
\]

IPC: instructions per clock, the inverse of CPI
Aspects of CPU Performance

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

<table>
<thead>
<tr>
<th></th>
<th>Inst Count</th>
<th>CPI</th>
<th>Clock Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inst. Set.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Organization</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Car Analogy

- Need to drive from Klaus to CRC
  - “Clock Speed” = 3500 RPM
  - “CPI” = 5250 rotations/km or 0.19 m/rot
  - “Insts” = 800m

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
\]

\[
800 \text{ m} \times \frac{1 \text{ rotation}}{0.19 \text{ m}} \times \frac{1 \text{ minute}}{3500 \text{ rotations}} = 1.2 \text{ minutes}
\]
CPU Version

• Program takes 33 billion instructions to run
• CPU processes insts at 2 cycles per inst
• Clock speed of 3GHz

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock Cycle}}
\]

Sometimes clock cycle time given instead (ex. cycle = 333 ps)
IPC sometimes used instead of CPI

= 22 seconds
The Processor Performance Equation (2)

CPU time = CPU Clock Cycles × Clock cycle time

CPU time = \left( \sum_{i=1}^{n} IC_i \times CPI_i \right) \times \text{Clock cycle time}

For each kind of instruction

How many instructions of this kind are there in the program

How many cycles it takes to execute an instruction of this kind
CPU performance w/ different instructions

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>Frequency</th>
<th>CPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>40%</td>
<td>1.0</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>4.0</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>2.0</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3.0</td>
</tr>
</tbody>
</table>

CPU time = \left( \sum_{i=1}^{n} IC_i \times CPI_i \right) \times \text{Clock cycle time}

Total Insts = 50B, Clock speed = 2 GHz

The overall CPI

\[ \text{CPI} = \frac{\sum_{i=1}^{n} IC_i \times CPI_i}{\text{Instruction count}} = \sum_{i=1}^{n} \frac{IC_i}{\text{Instruction count}} \times CPI_i \]
Comparing Performance

• “X is n times faster than Y”
\[
\frac{\text{Execution time}_Y}{\text{Execution time}_X} = n
\]

• “Throughput of X is n times that of Y”
\[
\frac{\text{Tasks per unit time}_X}{\text{Tasks per unit time}_Y} = n
\]
If Only it Were That Simple

• “X is n times faster than Y on A”

\[
\frac{\text{Execution time of app A on machine Y}}{\text{Execution time of app A on machine X}} = n
\]

• But what about different applications (or even parts of the same application)
  – X is 10 times faster than Y on A, and 1.5 times on B, but Y is 2 times faster than X on C, and 3 times on D, and...

So does X have better performance than Y?

Which would you buy?
Summarizing Performance

• Arithmetic mean
  – Average execution time
  – Gives more weight to longer-running programs

• Weighted arithmetic mean
  – More important programs can be emphasized
  – But what do we use as weights?
  – Different weight will make different machines look better
# Speedup

<table>
<thead>
<tr>
<th>Program</th>
<th>Machine A</th>
<th>Machine B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 1</td>
<td>5 sec</td>
<td>4 sec</td>
</tr>
<tr>
<td>Program 2</td>
<td>3 sec</td>
<td>6 sec</td>
</tr>
</tbody>
</table>

What is the speedup of A compared to B on Program 1?

What is the speedup of A compared to B on Program 2?

What is the average speedup?

What is the speedup of A compared to B on Sum(Program 1, Program 2)?
Normalizing & the Geometric Mean

- Speedup of arithmetic means $\neq$ arithmetic mean of speedup

- Use geometric mean: $\sqrt[n]{\prod_{i=1}^{n} \text{Normalized execution time on } i}$

- Neat property of the geometric mean: *Consistent whatever the reference machine*

- Do not use the arithmetic mean for normalized execution times
CPI/IPC

• Often when making comparisons in comp-arch studies:
  – Program (or set of) is the same for two CPUs
  – The clock speed is the same for two CPUs

• So we can just directly compare CPI’s and often we use IPC’s
**Average CPI vs. “Average” IPC**

- Average CPI = \( \frac{\text{CPI}_1 + \text{CPI}_2 + \ldots + \text{CPI}_n}{n} \)

  **Not Equal to A.M. of CPI!!!**

- A.M. of IPC = \( \frac{\text{IPC}_1 + \text{IPC}_2 + \ldots + \text{IPC}_n}{n} \)

- Must use *Harmonic Mean* to remain \( \propto \) to runtime
Harmonic Mean

• $H.M.(x_1, x_2, x_3, \ldots, x_n) =$

\[
\frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \frac{1}{x_3} + \ldots + \frac{1}{x_n}}
\]

• What in the world is this?
  – Average of inverse relationships
A.M.(CPI) vs. H.M.(IPC)

• “Average” IPC = \[
\frac{1}{n} \sum_{i=1}^{n} \text{CPI}_i
\]

= \[
\frac{1}{n} \left( \frac{1}{\text{IPC}_1} + \frac{1}{\text{IPC}_2} + \frac{1}{\text{IPC}_3} + \ldots + \frac{1}{\text{IPC}_n} \right)
\] = H.M.(IPC)