

## CIS501

### Introduction to Computer Architecture

**Prof. Milo Martin**

Unit 1: Technology, Cost, Performance, Power, and Reliability

---

### Readings

- **H+P**
  - Chapters 1

- **Paper**
  - G. Moore, “Cramming More Components onto Integrated Circuits”

- **Reminders**
  - Pre-quiz
  - Paper review
    - Groups of 3-4, send via e-mail to cis501-review@cis.upenn.edu
    - Don’t worry (much) about power question, as we might not get to it today

---

### What is Computer Architecture? (review)

- Design of interfaces and implementations...
- Under constantly changing set of external forces...
  - **Applications**: change from above (discussed last time)
  - **Technology**: changes transistor characteristics from below
  - **Inertia**: resists changing all levels of system at once

- To satisfy different constraints
  - **CIS 501** mostly about **performance**
  - Cost
  - Power
  - Reliability

- Iterative process driven by **empirical evaluation**
  - The art/science of tradeoffs

---

### This Unit

- What is a computer and what is computer architecture
- Forces that shape computer architecture
  - Applications (covered last time)
  - Semiconductor technology
- Evaluation metrics: parameters and technology basis
  - Cost
  - Performance
  - Power
  - Reliability

---
Abstraction and Layering

- **Abstraction**: only way of dealing with complex systems
  - Divide world into objects, each with an...
  - **Interface**: knobs, behaviors, knobs → behaviors
  - **Implementation**: "black box" (ignorance+apathy)
  - Only specialists deal with implementation, rest of us with interface
  - Example: car, only mechanics know how implementation works

- **Layering**: abstraction discipline makes life even simpler
  - Removes need to even know interfaces of most objects
  - Divide objects in system into layers
  - Layer X objects
    - Implemented in terms of interfaces of layer X-1 objects
    - Don't even need to know interfaces of layer X-2 objects
    - But sometimes helps if they do

Abstraction, Layering, and Computers

- Computers are complex systems, built in layers
  - Applications
  - O/S, compiler
  - Firmware, device drivers
  - Processor, memory, raw I/O devices
  - Digital circuits, digital/analog converters
  - Gates
  - Transistors
  - 99% of users don't know hardware layers implementation
  - 90% of users don't know implementation of any layer
  - That's OK, world still works just fine
    - But unfortunately, the layers sometimes breakdown
    - Someone needs to understand what's "under the hood"

CIS501: A Picture

- Computer architecture
  - Definition of ISA to facilitate implementation of software layers
- CIS 501 mostly about **computer micro-architecture**
  - Design CPU, Memory, I/O to implement ISA...
Shaping Force: Technology

- Basic technology element: **MOSFET**
  - Invention of 20th century
  - **MOS**: metal-oxide-semiconductor
  - Conductor, insulator, semi-conductor
  - **FET**: field-effect transistor
  - Solid-state component acts like electrical switch
  - Channel conducts source-drain when voltage applied to gate

- **Channel length**: characteristic parameter (short → fast)
  - Aka “feature size” or “technology”
  - Currently: 0.09 μm (0.09 micron), 90 nm
  - Continued miniaturization (scaling) known as “Moore’s Law”
  - Won’t last forever, physical limits approaching (or are they?)

CMOS Examples

- Example I: inverter
  - Case I: input = 0
    - P-transistor closed, n-transistor open
    - Power charges output (1)
  - Case II: input = 1
    - P-transistor open, n-transistor closed
    - Output discharges to ground (0)

- Example II: look at **truth table**
  - 0, 0 → 1  0, 1 → 1
  - 1, 0 → 1  1, 1 → 0
  - Result: this is a **NAND** (NOT AND)
  - NAND is universal (can build any logic function)

- More examples, details
  - [http://.../amir/cse371/lecture_slides/tech.pdf](http://.../amir/cse371/lecture_slides/tech.pdf)

Complementary MOS (CMOS)

- Voltages as values
  - Power ($V_{DD}$) = 1, Ground = 0

- Two kinds of MOSFETs
  - **N-transistors**
    - Conduct when gate voltage is 1
    - Good at passing 0s
  - **P-transistors**
    - Conduct when gate voltage is 0
    - Good at passing 1s

- **CMOS**: complementary n-/p- networks form boolean logic

More About CMOS and Technology

- Two different CMOS families

- **SRAM (logic)**: used to make processors
  - Storage implemented as inverter pairs
  - Optimized for speed

- **DRAM (memory)**: used to make memory
  - Storage implemented as capacitors
  - Optimized for density, cost, power

- Disk is also a “technology”, but isn’t transistor-based
Aside: VLSI + Manufacturing

- **VLSI (very large scale integration)**
  - MOSFET manufacturing process
  - As important as invention of MOSFET itself
  - Multi-step photochemical and electrochemical process
  - Fixed cost per step
  - Cost per transistor shrinks with transistor size

- Other production costs
  - Packaging
  - Test
  - Mask set
  - Design

MOSFET Side View

- **MOS**: three materials needed to make a transistor
  - **Metal** - Aluminum, Tungsten, Copper: conductor
  - **Oxide** - Silicon Dioxide (SiO₂): insulator
  - ** Semiconductor** - doped Si: conducts under certain conditions

- **FET**: field effect (the mechanism) transistor
  - Voltage on gate: current flows source to drain (transistor on)
  - No voltage on gate: no current (transistor off)

Manufacturing Process

- Start with silicon wafer
- “Grow” photo-resist
  - Molecular beam epitaxy
- Burn positive bias mask
  - Ultraviolet light lithography
- Dissolve unburned photo-resist
  - Chemically
- Bomb wafer with negative ions (P)
  - Doping
- Dissolve remaining photo-resist
  - Chemically
- Continue with next layer
Manufacturing Process

- Grow SiO₂
- Grow photo-resist
- Burn "wire-level-1" mask
- Dissolve unburned photo-resist
  - And underlying SiO₂
- Grow copper "wires"
- Dissolve remaining photo-resist
- Continue with next wire layer...
- Typical number of wire layers: 3-6

Empirical Evaluation

- Metrics
  - Cost
  - Performance
  - Power
  - Reliability

  - Often more important in combination than individually
    - Performance/cost (MIPS/$)
    - Performance/power (MIPS/W)

  - Basis for
    - Design decisions
    - Purchasing decisions

Defects

- Defects can arise
  - Under-/over-doping
  - Over-/under-dissolved insulator
  - Mask mis-alignment
  - Particle contaminants

- Try to minimize defects
  - Process margins
  - Design rules
    - Minimal transistor size, separation

- Or, tolerate defects
  - Redundant or "spare" memory cells

Cost

- Metric: $

  - In grand scheme: CPU accounts for fraction of cost
    - Some of that is profit (Intel's, Dell's)

<table>
<thead>
<tr>
<th></th>
<th>Desktop</th>
<th>Laptop</th>
<th>PDA</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>$100–$300</td>
<td>$150–$350</td>
<td>$50–$100</td>
<td>$10–$20</td>
</tr>
<tr>
<td>% of total</td>
<td>10–20%</td>
<td>10–20%</td>
<td>20–30%</td>
<td>20-30%</td>
</tr>
<tr>
<td>Other costs</td>
<td>Memory, display, power supply/battery, disk, packaging</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We are concerned about Intel's cost (transfers to you)
  - Unit cost: costs to manufacture individual chips
  - Startup cost: cost to design chip, build the fab line, marketing
Unit Cost: Integrated Circuit (IC)

- Chips built in multi-step chemical processes on wafers
  - Cost / wafer is constant, f(wafer size, number of steps)
- Chip (die) cost is proportional to area
  - Larger chips means fewer of them
  - Larger chips means fewer working ones
  - Why? Uniform defect density
  - Chip cost ∼ chip area
    - α = 2–3
- Wafer yield: % wafer that is chips
- Die yield: % chips that work
- Yield is increasingly non-binary - fast vs slow chips

Startup Costs

- **Startup costs**: must be amortized over chips sold
  - Research and development: ∼$100M per chip
    - 500 person-years @ $200K per
  - Fabrication facilities: ∼$2B per new line
    - Clean rooms (bunny suits), lithography, testing equipment
  - If you sell 10M chips, startup adds ∼$200 to cost of each
    - Companies (e.g., Intel) don't make money on new chips
    - They make money on proliferations (shrinks and frequency)
    - No startup cost for these

Yield/Cost Examples

- **Parameters**
  - wafer yield = 90%, α = 2, defect density = 2/cm²

<table>
<thead>
<tr>
<th>Die size (mm²)</th>
<th>100</th>
<th>144</th>
<th>196</th>
<th>256</th>
<th>324</th>
<th>400</th>
</tr>
</thead>
<tbody>
<tr>
<td>Die yield</td>
<td>23%</td>
<td>19%</td>
<td>16%</td>
<td>12%</td>
<td>11%</td>
<td>10%</td>
</tr>
<tr>
<td>6” Wafer</td>
<td>139(31)</td>
<td>90(16)</td>
<td>62(9)</td>
<td>44(5)</td>
<td>32(3)</td>
<td>23(2)</td>
</tr>
<tr>
<td>8” Wafer</td>
<td>256(59)</td>
<td>177(32)</td>
<td>124(19)</td>
<td>90(11)</td>
<td>68(7)</td>
<td>52(5)</td>
</tr>
<tr>
<td>10” Wafer</td>
<td>431(96)</td>
<td>290(53)</td>
<td>206(32)</td>
<td>153(20)</td>
<td>116(13)</td>
<td>90(9)</td>
</tr>
</tbody>
</table>

Moore’s Effect on Cost

- Scaling has opposite effects on unit and startup costs
  - **Reduces unit integrated circuit cost**
    - Either lower cost for same functionality...
    - Or same cost for more functionality
  - **Increases startup cost**
    - More expensive fabrication equipment
    - Takes longer to design, verify, and test chips
Performance

- Two definitions
  - **Latency (execution time)**: time to finish a fixed task
  - **Throughput (bandwidth)**: number of tasks in fixed time
- Very different: throughput can exploit parallelism, latency cannot
  - Baking bread analogy
  - Often contradictory
  - Choose definition that matches goals (most frequently throughput)

- Example: move people from A to B, 10 miles
  - Car: capacity = 5, speed = 60 miles/hour
  - Bus: capacity = 60, speed = 20 miles/hour
  - Latency: **car = 10 min**, bus = 30 min
  - Throughput: car = 15 PPH (count return trip), **bus = 60 PPH**

Performance Improvement

- Processor A is X times faster than processor B if
  - Latency(P,A) = Latency(P,B) / X
  - Throughput(P,A) = Throughput(P,B) * X
- Processor A is X% faster than processor B if
  - Latency(P,A) = Latency(P,B) / (1+X/100)
  - Throughput(P,A) = Throughput(P,B) * (1+X/100)

- Car/bus example
  - Latency? Car is 3 times (and 200%) faster than bus
  - Throughput? Bus is 4 times (and 300%) faster than car

What Is ‘P’ in Latency(P,A)?

- Program
  - Latency(A) makes no sense, processor executes **some program**
  - But which one?
  - Actual target workload?
    + Accurate
      - Not portable/repeatable, overly specific, hard to pinpoint problems
  - **Some representative benchmark program(s)?**
    + Portable/repeatable, pretty accurate
      - Hard to pinpoint problems, may not be exactly what you run
  - Some small kernel benchmarks (micro-benchmarks)
    + Portable/repeatable, easy to run, easy to pinpoint problems
      - Not representative of complex behaviors of real programs

SPEC Benchmarks

- **SPEC (Standard Performance Evaluation Corporation)**
  - [http://www.spec.org/](http://www.spec.org/)
  - Consortium of companies that collects, standardizes, and distributes benchmark programs
  - Post **SPECmark** results for different processors
    - 1 number that represents performance for entire suite
    - Benchmark suites for CPU, Java, I/O, Web, Mail, etc.
  - Updated every few years: so companies don’t target benchmarks

- **SPEC CPU 2000**
  - 12 “integer”: gzip, gcc, perl, crafty (chess), vortex (DB), etc.
  - 14 “floating point”: mesa (openGL), equake, facerec, etc.
  - Written in C and Fortran (a few in C++)
Other Benchmarks

- Parallel benchmarks
  - SPLASH2 - Stanford Parallel Applications for Shared Memory
  - NAS
  - SPEC’s OpenMP benchmarks
  - SPECjbb - Java multithreaded database-like workload

- Transaction Processing Council (TPC)
  - TPC-C: On-line transaction processing (OLTP)
  - TPC-H/R: Decision support systems (DSS)
  - TPC-W: E-commerce database backend workload
  - Have parallelism (intra-query and inter-query)
  - Heavy I/O and memory components

Other Benchmarks

- Transaction Processing Council (TPC)
  - TPC-C: On-line transaction processing (OLTP)
  - TPC-H/R: Decision support systems (DSS)
  - TPC-W: E-commerce database backend workload
  - Have parallelism (intra-query and inter-query)
  - Heavy I/O and memory components

SPECmark

- Reference machine: Sun SPARC 10
- Latency SPECmark
  - For each benchmark
    - Take odd number of samples: on both machines
    - Choose median
    - Take latency ratio (Sun SPARC 10 / your machine)
  - Take GMEAN of ratios over all benchmarks
- Throughput SPECmark
  - Run multiple benchmarks in parallel on multiple-processor system
- Recent (latency) leaders
  - SPECint: Intel 3.4 GHz Pentium4 (1705)
  - SPECfp: IBM 1.9 GHz Power5 (2702)

Adding/Averaging Performance Numbers

- You can add latencies, but not throughput
  - Latency(P1+P2, A) = Latency(P1,A) + Latency(P2,A)
  - Throughput(P1+P2,A) = Throughput(P1,A) + Throughput(P2,A)
    - 1 mile @ 30 miles/hour + 1 mile @ 90 miles/hour
      - Average is **not** 60 miles/hour
    - 0.033 hours at 30 miles/hour + 0.01 hours at 90 miles/hour
      - Average is only 47 miles/hour (2 miles / (0.033 + 0.01 hours))
  - Throughput((P1+P2,A) = 1 / [(1/ Throughput(P1,A)) + (1/ Throughput(P2,A))]

- Same goes for means (averages)
  - **Arithmetic**: \((1/N) \times \sum_{p=1..N} \text{Latency}(P)\)
    - For units that are proportional to time (e.g., latency)
  - **Harmonic**: \(N / \sum_{p=1..N} 1/\text{Throughput}(P)\)
    - For units that are inversely proportional to time (e.g., throughput)
  - **Geometric**: \(\sqrt[N]{\prod_{p=1..N} \text{Speedup}(P)}\)
    - For unitless quantities (e.g., speedups)

CPU Performance Equation

- Multiple aspects to performance: helps to isolate them
  - Latency(P,A) = seconds / program =
    - (instructions / program) * (cycles / instruction) * (seconds / cycle)
  - **Instructions / program**: dynamic instruction count
    - Function of program, compiler, instruction set architecture (ISA)
  - **Cycles / instruction**: CPI
    - Function of program, compiler, ISA, micro-architecture
  - **Seconds / cycle**: clock period
    - Function of micro-architecture, technology parameters

- For low latency (better performance) minimize all three
  - Hard: often pull against the other
Danger: Partial Performance Metrics

- Micro-architects often ignore dynamic instruction count
  - Typically work in one ISA/one compiler → treat it as fixed

- CPU performance equation becomes
  - seconds / instruction = (cycles / instruction) * (seconds / cycle)
  - This is a latency measure, if we care about throughput ...
  - \textbf{Instructions / second} = (instructions / cycle) * (cycles / second)

- \textbf{MIPS} (millions of instructions per second)
  - Instructions / second * 10^6
  - \textbf{Cycles / second}: clock frequency (in MHz)
  - Example: CPI = 2, clock = 500 MHz, what is MIPS?
    - 0.5 * 500 MHz * 10^6 = 250 MIPS
  - Example problem situation:
    - compiler removes instructions, program faster
    - However, “MIPS” goes down (misleading)

MIPS and MFLOPS (MegaFLOPS)

- Problem: MIPS may vary inversely with performance
  - Some optimizations actually add instructions
  - Work per instruction varies (e.g., FP mult vs. integer add)
  - ISAs are not equivalent

- \textbf{MFLOPS}: like MIPS, but counts only FP ops, because...
  - + FP ops can’t be optimized away
  - + FP ops have longest latencies anyway
  - + FP ops are same across machines

- May have been valid in 1980, but today...
  - Most programs are “integer”, i.e., light on FP
  - Loads from memory take much longer than FP divide
  - Even FP instructions sets are not equivalent

- Upshot: MIPS not perfect, but more useful than MFLOPS

Danger: Partial Performance Metrics II

- Micro-architects often ignore dynamic instruction count...
  - ... but general public (mostly) also ignores CPI
    - Equates clock frequency with performance!!

- Which processor would you buy?
  - Processor A: CPI = 2, clock = 500 MHz
  - Processor B: CPI = 1, clock = 300 MHz
  - Probably A, but B is faster (assuming same ISA/compiler)

- Classic example
  - 800 MHz Pentium III faster than 1 GHz Pentium 4
  - Same ISA and compiler

Cycles per Instruction (CPI)

- \textbf{CIS501} is mostly about improving \textbf{CPI}
  - Cycle/instruction for \textbf{average instruction}
  - \textbf{IPC} = 1/CPI
    - Used more frequently than CPI, but harder to compute with
    - Different instructions have different cycle costs
      - E.g., integer add typically takes 1 cycle, FP divide takes > 10
    - Assumes you know something about instruction frequencies

- \textbf{CPI} example
  - A program executes equal integer, FP, and memory operations
  - Cycles per instruction type: integer = 1, memory = 2, FP = 3
  - What is the CPI? (0.33 * 1) + (0.33 * 2) + (0.33 * 3) = 2
  - \textbf{Caveat}: this sort of calculation ignores dependences completely
    - Back-of-the-envelope arguments only
Another CPI Example

- Assume a processor with instruction frequencies and costs
  - Integer ALU: 50%, 1 cycle
  - Load: 20%, 5 cycle
  - Store: 10%, 1 cycle
  - Branch: 20%, 2 cycle
- Which change would improve performance more?
  - A. Branch prediction to reduce branch cost to 1 cycle?
  - B. A bigger data cache to reduce load cost to 3 cycles?
- Compute CPI
  - Base = 0.5*1 + 0.2*5 + 0.1*1 + 0.2*2 = 2
  - A = 0.5*1 + 0.2*5 + 0.1*1 + 0.2*1 = 1.8
  - B = 0.5*1 + 0.2*3 + 0.1*1 + 0.2*2 = 1.6 (winner)

Increasing Clock Frequency: Pipelining

- CPU is a pipeline: compute stages separated by latches
  - http://.../-amir/cse371/lecture_slides/pipeline.pdf
- Clock period: maximum delay of any stage
  - Number of gate levels in stage
  - Delay of individual gates (these days, wire delay more important)

CPI and Clock Frequency

- System components “clocked” independently
  - E.g., Increasing processor clock frequency doesn’t improve memory performance

- Example
  - Processor A: CPI\text{CPU} = 1, CPI\text{MEM} = 1, clock = 500 MHz
  - What is the speedup if we double clock frequency?
    - Base: CPI = 2 → IPC = 0.5 → MIPS = 250
    - New: CPI = 3 → IPC = 0.33 → MIPS = 333
    - Clock *= 2 → CPI\text{MEM} *= 2
    - Speedup = 333/250 = 1.33 << 2

- What about an infinite clock frequency?
  - Only a x2 speedup (Example of Amdahl’s Law)
Measuring CPI

- How are CPI and execution-time actually measured?
  - Execution time: time (Unix): wall clock + CPU + system
  - CPI = CPU time / (clock frequency * dynamic insn count)
  - How is dynamic instruction count measured?
  - More useful is CPI breakdown (CPI_{CPU}, CPI_{MEM}, etc.)
    - So we know what performance problems are and what to fix

- CPI breakdowns
  - Hardware event counters
    - Calculate CPI using counter frequencies/event costs
  - Cycle-level micro-architecture simulation (e.g., SimpleScalar)
    + Measure exactly what you want
    + Measure impact of potential fixes
  - Must model micro-architecture faithfully
  - Method of choice for many micro-architects (and you)

Moore’s Effect on Performance

- Moore’s Curve: common interpretation of Moore’s Law
  - “CPU performance doubles every 18 months”
  - Self fulfilling prophecy
    - 2X every 18 months is ~1% per week
    - Q: Would you add a feature that improved performance 20% if it took 8 months to design and test?
  - Processors under Moore’s Curve (arrive too late) fail spectacularly
    - E.g., Intel’s Itanium, Sun’s Millennium

Improving CPI

- CIS501 is more about improving CPI than frequency
  - Historically, clock accounts for 70%+ of performance improvement
    - Achieved via deeper pipelines
  - That will (have to) change
    - Deep pipelining is not power efficient
    - Physical speed limits are approaching
    - Techniques we will look at
      - Caching, speculation, multiple issue, out-of-order issue
      - Vectors, multiprocessing, more...
  - Moore helps because CPI reduction requires transistors
    - The definition of parallelism is “more transistors”
    - But best example is caches

Performance Rules of Thumb

- Make common case fast
  - Sometimes called “Amdahl’s Law”
  - Corollary: don’t optimize 1% to the detriment of other 99%

- Build a balanced system
  - Don’t over-engineer capabilities that cannot be utilized

- Design for actual, not peak, performance
  - For actual performance X, machine capability must be > X
Transistor Speed, Power, and Reliability

- Transistor characteristics and scaling impact:
  - Switching speed
  - Power
  - Reliability

- “Undergrad” gate delay model for **architecture**
  - Each Not, NAND, NOR, AND, OR gate has delay of “1”
  - Reality is not so simple

Simple RC Delay Model

- Switching time is a RC circuit (charge or discharge)
  - R - Resistance: slows rate of current flow
  - Depends on material, length, cross-section area
  - C - Capacitance: electrical charge storage
  - Depends on material, area, distance
  - Voltage affects speed, too

IBM CMOS7, 6 layers of copper wiring

IBM SOI Technology

Transistors and Wires

IBM CMOS7, 6 layers of copper wiring
Resistance

- Transistor channel resistance
  - function of $V_g$ (gate voltage)
- Wire resistance (negligible for short wires)

Capacitance

- Source/Drain capacitance
- Gate capacitance
- Wire capacitance (negligible for short wires)

Transistor Width

- “Wider” transistors have lower resistance, more drive
  - Specified per-device
- Useful for driving large “loads” like long or off-chip wires

Which is faster? Why?

- Use for driving large “loads” like long or off-chip wires
RC Delay Model Ramifications

- Want to reduce resistance
  - “wide” drive transistors (width specified per device)
  - Short wires
- Want to reduce capacitance
  - Number of connected devices
  - Less-wide transistors (gate capacitance of next stage)
  - Short wires

Transistor Scaling

- Transistor length is key property of a “process generation”
  - 90nm refers to the transistor gate length, same for all transistors
- Shrink transistor length:
  - Lower resistance of channel (shorter)
  - Lower gate/source/drain capacitance
- Result: transistor drive strength linear as gate length shrinks

Wires

- Resistance fixed by \((\text{length} \times \text{resistivity}) / (\text{height} \times \text{width})\)
  - bulk aluminum 2.8 \(\mu\)Ω-cm, bulk copper 1.7 \(\mu\)Ω-cm
- Capacitance depends on geometry of surrounding wires and relative permittivity, \(\varepsilon_r\) of dielectric
  - silicon dioxide \(\varepsilon_r = 3.9\), new low-k dielectrics in range 1.2-3.1

Wire Delay

- RC Delay of wires
  - Resistance proportional to length
  - Capacitance proportional to length
- Result: delay of a wire is quadratic in length
  - Insert “inverter” repeaters for long wires to
  - Bring it back to linear delay
Moore’s Effect on RC Delay

- Scaling helps reduce wire and gate delays
  - In some ways, hurts in others
    + Wires become shorter (Length ↓ → Resistance ↓)
    + Wire “surface areas” become smaller (Capacitance ↓)
    + Transistors become shorter (Resistance ↓)
    + Transistors become narrower (Capacitance ↓, Resistance ↑)
  - Gate insulator thickness becomes smaller (Capacitance ↑)
  - Distance between wires becomes smaller (Capacitance ↑)

Improving RC Delay

- Exploit good effects of scaling
- Fabrication technology improvements
  + Use copper instead of aluminum for wires (ρ ↓ → Resistance ↓)
  + Use lower-dielectric insulators (κ ↓ → Capacitance ↓)
  + Increase Voltage
- Design implications
  + Use bigger cross-section wires (Area ↑ → Resistance ↓)
    - Typically means taller, otherwise fewer of them
    - Increases “surface area” and capacitance (Capacitance ↑)
  + Use wider transistors (Area ↑ → Resistance ↓)
    - Increases capacitance (not for you, for upstream transistors)
    - Use selectively

Another Constraint: Power and Energy

- **Power** (Watt or Joule/Second): short-term (peak, max)
  - Mostly a dissipation (heat) concern
    - Power-density (Watt/cm²): important related metric
      - Thermal cycle: power dissipation ↑ → power density ↑ → temperature ↑ → resistance ↑ → power dissipation ↑...
    - Cost (and form factor): packaging, heat sink, fan, etc.

- **Energy** (Joule): long-term
  - Mostly a consumption concern
    - Primary issue is battery life (cost, weight of battery, too)
    - Low-power implies low-energy, but not the other way around

- 10 years ago, nobody cared

Sources of Energy Consumption

Dynamic power:
- Capacitor Charging (85-90% of active power)
  - Energy is ½ CV² per transition
- Short-Circuit Current (10-15% of active power)
  - When both p and n transistors turn on during signal transition

Static power:
- Subthreshold Leakage (dominates when inactive)
  - Transistors don’t turn off completely
- Diode Leakage (negligible)
  - Parasitic source and drain diodes leak to substrate
Moore’s Effect on Power

- Scaling has largely **good** effects on local power
  - Shorter wires/smaller transistors (Length ↓ → Capacitance ↓)
  - Shorter transistor length (Resistance ↓, Capacitance ↓)
  - Global effects largely undone by increased transistor counts

- Scaling has a largely negative effect on **power density**
  - Transistor/wire power decreases linearly
  - Transistor/wire density decreases quadratically
  - Power-density increases linearly
    - Thermal cycle
    - Controlled somewhat by reduced $V_{DD}$ (5 → 3.3 → 1.6 → 1.3 → 1.1)
    - Reduced $V_{DD}$ sacrifices some switching speed

---

Dynamic Voltage Scaling (DVS)

- **Dynamic voltage scaling (DVS)**
  - OS reduces voltage/frequency when peak performance not needed

<table>
<thead>
<tr>
<th>Mobile PentiumIII “SpeedStep”</th>
<th>TM5400 “LongRun”</th>
<th>Intel X-SCALE (StrongARM2)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frequency</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>300–1000MHz (50MHz steps)</td>
<td>200–700MHz (33MHz steps)</td>
<td>50–800MHz (50MHz steps)</td>
</tr>
<tr>
<td><strong>Voltage</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.9–1.7V (0.1V steps)</td>
<td>1.1–1.6V (continuous)</td>
<td>0.7–1.6V (continuous)</td>
</tr>
<tr>
<td><strong>High-speed</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3400MIPS @ 34W</td>
<td>1600MIPS @ 2W</td>
<td>800MIPS @ 0.9W</td>
</tr>
<tr>
<td><strong>Low-power</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1100MIPS @ 4.5W</td>
<td>300MIPS @ 0.25W</td>
<td>62MIPS @ 0.01W</td>
</tr>
</tbody>
</table>

± X-SCALE is power efficient (6200 MIPS/W), but not IA32 compatible

---

Reducing Power

- **Reduce supply voltage ($V_{DD}$)**
  - Reduces dynamic power quadratically and static power linearly
  - But poses a tough choice regarding $V_{TH}$
  - Constant $V_{TH}$ slows circuit speed $→$ clock frequency $→$ performance
  - Reduced $V_{TH}$ increases static power **exponentially**

- **Reduce clock frequency ($f$)**
  - Reduces dynamic power linearly
  - Doesn’t reduce static power
  - Reduces performance linearly
  - Generally doesn’t make sense without also reduced $V_{DD}$...
    - Except that frequency can be adjusted cycle-to-cycle and locally
    - More on this later

---

Reducing Power: Processor Modes

- **Modern electrical components have low-power modes**
  - Note: no low-power disk mode, magnetic (non-volatile)
  - **“Standby” mode**
    - Turn off internal clock
    - Leave external signal controller and pins on
    - Restart clock on interrupt
    - Cuts dynamic power linearly, doesn’t effect static power
    - Laptops go into this mode between keystrokes
  - **“Sleep” mode**
    - Flush caches, OS may also flush DRAM to disk
    - Turn off processor power plane
    - Needs a “hard” restart
    - Cuts dynamic and static power
    - Laptops go into this mode after ~10 idle minutes
Reliability

- **Mean Time Between Failures (MTBF)**
  - How long before you have to reboot or buy a new one
  - Not very quantitative yet, people just starting to think about this

- CPU reliability small in grand scheme
  - Software most unreliable component in a system
  - Much more difficult to specify & test
  - Much more of it
  - Most unreliable hardware component ... disk
  - Subject to mechanical wear

Moore’s Bad Effect on Reliability

- CMOS devices: CPU and memory
  - Historically almost perfectly reliable
  - Moore has made them less reliable over time

- Two sources of electrical faults
  - Energetic particle strikes (from sun)
    - Randomly charge nodes, cause bits to flip, **transient**
  - Electro-migration: change in electrical interfaces/properties
    - Temperature-driven, happens gradually, **permanent**

- Large, high-energy transistors are immune to these effects
  - Scaling makes node energy closer to particle energy
  - Scaling increases power-density which increases temperature
  - Memory (DRAM) was hit first: denser, smaller devices than SRAM

Moore’s Good Effect on Reliability

- The key to providing reliability is **redundancy**
  - The same scaling that makes devices less reliable...
  - Also increase device density to enable redundancy

- Classic example
  - Error correcting code (ECC) for DRAM
  - ECC also starting to appear for caches
  - More reliability techniques later

- Today’s big open questions
  - Can we protect logic?
  - Can architectural techniques help hardware reliability?
  - Can architectural techniques help with software reliability?

Summary: A Global Look at Moore

- Device scaling (Moore’s Law)
  + Increases performance
    - Reduces transistor/wire delay
    - Gives us more transistors with which to reduce CPI
  + Reduces local power consumption
    - Which is quickly undone by increased integration
    - Aggravates power-density and temperature problems
  + Aggravates reliability problem
    - But gives us the transistors to solve it via redundancy
  + Reduces unit cost
    - But increases startup cost

- Will we fall off Moore’s Cliff? (for real, this time?)
  - What’s next: nanotubes, quantum-dots, optical, spin-tronics, DNA?
Summary

- What is computer architecture
  - Abstraction and layering: interface and implementation, ISA
  - Shaping forces: application and semiconductor technology
  - Moore’s Law
- Cost
  - Unit and startup
- Performance
  - Latency and throughput
  - CPU performance equation: insn count * CPI * clock frequency
- Power and energy
  - Dynamic and static power
- Reliability

CIS501

- CIS501: Computer Architecture
  - Mostly about micro-architecture
  - Mostly about CPU/Memory
  - Mostly about general-purpose
  - Mostly about performance
  - We’ll still only scratch the surface

- Next time
  - Instruction set architecture