Power and Energy

Charles Li and Deepak Pallerla
Power: A First-Class Architectural Design Constraint
Motivations

● IT was 8% of US electricity usage in 2000
  ○ Increasing over time
● Chip die power density increasing linearly
  ○ Eventually can’t cool them
● Very general motivations
  ○ Appropriate for a general overview
CMOS Power Basics

- \( P = ACV^2 f + \tau AVI_{\text{short}} + VI_{\text{leak}} = P_{\text{switching}} + P_{\text{short}} + P_{\text{leakage}} \)
  - \( ACV^2 f = \text{Activity} \times \text{Capacitance} \times \text{Voltage}^2 \times \text{Frequency} \)
  - \( \tau AVI_{\text{short}} = \text{Short circuit time} \times \text{Activity} \times \text{Voltage} \times \text{Short circuit current} \)
  - \( VI_{\text{leak}} = \text{Voltage} \times \text{Leakage current} \)

- Reduce voltage?
  - Reduces max frequency unless you reduce MOSFET \( V_{\text{th}} \)
  - Reducing \( V_{\text{th}} \) increases \( I_{\text{leak}} \)

- Reducing \( V \) will decrease \( P_{\text{switching}} \) and increase \( P_{\text{leakage}} \) until \( P_{\text{leakage}} \) dominates
What Does Efficiency Mean?

- Portable devices carry a fixed amount of energy in the battery
  - Minimizing energy per operation better than minimizing power
  - MIPS/W a common metric (simplifies to instructions per Joule)
  - MIPS/W can be misleading for quadratic devices (CMOS)

- Non-portable devices should minimize power
  - Different from minimizing energy per operation
Power Reduction - Logic

Clock tree is a significant power consumer. What can you do about it?

- **Clock gating** - Turn off clocks to unused logic
  - Increases clock skew but solved by better tools
- **Half frequency** - Use rising *and* falling edges, run at half frequency
  - Increases logic complexity and area
- **Half swing** - Clock swing only half of supply voltage
  - “Increases the latch design’s requirements”
  - Hard to use when supply voltage is already low
Power Reduction - Logic (cont.)

- **Asynchronous logic** - Clocks use power, so don’t use clocks. Many problems.
  - Extra logic and wiring required for completion signals
  - Absence of design tools, difficult to test
    - Still true 20 years later?
  - Amulet - asynchronous ARM implementation

- **Globally asynchronous, locally synchronous logic**
  - Reduce clock power and skew on large chips
  - Ability to reduce frequency and voltage to specific parts of chip
  - Best of both worlds
Power Reduction - Architecture

Dynamic power loss upon memory access, leakage loss from being turned on.

- **Memory - Filter cache**
  - Extremely small cache ahead of L1 cache
  - Sacrifice performance but keep L1 cache at low power most of the time

- **Memory - Banking**
  - Split memory into banks, turn on bank being used
  - Requires spatial locality and disk backup for off banks
Power Reduction - Architecture (cont.)

Memory buses are a significant source of power usage.

- Gray code addresses reduces switching for sequential addresses.
- Compression reduces data transfer amounts
  - Presumably saves more power than compression and decompression
Power Reduction - Architecture (cont.)

- **Pipelining** is done to increase clock frequency (reduce critical path length)
  - Limits voltage reduction
- **Parallel processing** improves efficiency
  - General purpose computation (SPEC benchmarks) not very parallel
  - DSPs are highly parallel and power efficient
    - This points towards accelerators for further improvements
Power Reduction - Operating System

Operating system can support voltage scaling. How do we use it best?

- **Application controlled** - Apps use OS interface to scale voltage for itself
  - Requires app modification
- **OS controlled** - OS detects when to scale voltage
  - No app modification needed
  - Difficult to make detection optimal
Applications for Efficient Processors

- **High MIPS/W** (low energy per operation)
  - “The obvious applications [...] lie in mobile computing.”
  - “mobile phones will surpass the desktop as the defining application environment for computing”
    - Pretty accurate in 2020
- **Low power**
  - Servers and data centers
  - More compute for same power
Future Challenges

● Smaller FETs need lower $V_{th}$
● Lower $V_{th}$ increases leakage current
  ○ Use low $V_{th}$ FETs for high frequency paths
  ○ Use high $V_{th}$ FETs for low frequency paths
● In general power must be considered early in design process
  ○ Currently happening
● Tools must support power analysis
  ○ Currently happening
<table>
<thead>
<tr>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Broad overview of power saving techniques at different levels</td>
<td>● Individual techniques vaguely described</td>
</tr>
<tr>
<td>● Distinguishes between power and energy</td>
<td>● Heterogeneous designs not mentioned (ex. big.LITTLE)</td>
</tr>
<tr>
<td>● Predicts rise of mobile computing</td>
<td>● OS section only sort of discusses energy aware scheduling</td>
</tr>
<tr>
<td></td>
<td>● Nearly 20 years old, what’s new?</td>
</tr>
</tbody>
</table>
Power Struggles: Revisiting the RISC vs CISC Debate on Contemporary ARM and x86 Architectures
Motivation
RISC v. CISC pt.1

- First debates in 1980s
  - Focused on desktops and servers
  - Primary design constraints
    - Area
    - Chip design complexity
RISC v. CISC pt.1

- "RISC as exemplified by MIPS provides a significant processor performance advantage."
- "... the Pentium Pro processor achieves 80% to 90% of the performance of the Alpha 21164 ... It uses an aggressive out-of-order design to overcome the instruction set level limitations of a CISC architecture. On floating-point intensive benchmarks, the Alpha 21164 does achieve over twice the performance of the Pentium Pro processor."
- "with aggressive microarchitectural techniques for ILP, CISC and RISC ISAs can be implemented to yield very similar performance."
RISC v. CISC pt.2

- 2013
  - Smartphones and tablets in addition to desktops and servers
  - Primary design constraints
    - Energy
    - Power
  - New markets
    - ARM servers for energy efficiency
    - x86 for mobile and low power devices for performance
Does ISA affect performance, power, energy efficiency?
## Framing the Impacts

### Table 1. Summary of RISC and CISC Trends.

<table>
<thead>
<tr>
<th>Format</th>
<th>Operations</th>
<th>Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISC / ARM</td>
<td>▪ Fixed length instructions</td>
<td>▪ Operands: registers, immediates</td>
</tr>
<tr>
<td></td>
<td>▪ Relatively simple encoding</td>
<td>▪ Few addressing modes</td>
</tr>
<tr>
<td></td>
<td>▪ ARM: 4B, THUMB(2B, optional)</td>
<td>▪ ARM: 16 general purpose registers</td>
</tr>
<tr>
<td></td>
<td>▪ Variable length instructions</td>
<td>▪ Complex, multi-cycle instructions</td>
</tr>
<tr>
<td></td>
<td>▪ Common insts shorter/simpler</td>
<td>▪ Transcendental</td>
</tr>
<tr>
<td></td>
<td>▪ Special insts longer/complex</td>
<td>▪ Encryption</td>
</tr>
<tr>
<td></td>
<td>▪ x86: from 1B to 16B long</td>
<td>▪ String manipulation</td>
</tr>
<tr>
<td>CISC / x86</td>
<td>▪ CISC decode latency prevents pipelining</td>
<td>▪ Even w/ μcode, pipelining hard</td>
</tr>
<tr>
<td></td>
<td>▪ CISC decoders slower/more area</td>
<td>▪ CISC latency may be longer than compiler’s RISC equivalent</td>
</tr>
<tr>
<td></td>
<td>▪ Code density: RISC &lt; CISC</td>
<td>▪ CISC decoder complexity higher</td>
</tr>
<tr>
<td>Historical</td>
<td>▪ μ-op cache minimizes decoding overheads</td>
<td>▪ CISC has more per inst work, longer cycles</td>
</tr>
<tr>
<td>Contrasts</td>
<td>▪ x86 decode optimized for common insts</td>
<td>▪ Static code size: RISC &gt; CISC</td>
</tr>
<tr>
<td>Convergence</td>
<td>▪ I-cache minimizes code density impact</td>
<td></td>
</tr>
<tr>
<td>Trends</td>
<td>▪ How much variance in x86 inst length?</td>
<td>▪ x86 decode optimized for common insts</td>
</tr>
<tr>
<td></td>
<td>Low variance ⇒ common insts optimized</td>
<td>▪ CISC insts split into RISC-like micro-ops; optimizations eliminated inefficiencies</td>
</tr>
<tr>
<td>Empirical</td>
<td>▪ Are ARM and x86 code densities similar?</td>
<td>▪ Modern compilers pick mostly RISC insts; μ-op counts similar for ARM and x86</td>
</tr>
<tr>
<td>Questions</td>
<td>▪ Similar density ⇒ No ISA effect</td>
<td></td>
</tr>
<tr>
<td></td>
<td>▪ What are instruction cache miss rates?</td>
<td>▪ Are macro-op counts similar?</td>
</tr>
<tr>
<td></td>
<td>Low ⇒ caches hide low code densities</td>
<td>Similar ⇒ RISC-like on both</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Are complex instructions used by x86 ISA?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Few complex ⇒ Compiler picks RISC-like</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▪ Are μ-op counts similar?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Similar ⇒ CISC split into RISC-like μ-ops</td>
</tr>
</tbody>
</table>
Choosing Platforms

- Want as many similarities as possible
  - Technology node
  - Frequency
  - High performance/low power transistors
  - L2-Cache
  - Memory Controller
  - Memory Size
  - Operating System
  - Compiler

- Intent: Keep non-processor features as similar as possible.
Choosing Platforms: Best Effort

- **ARM/RISC**
  - Cortex-A9
  - Cortex-A8
- **x86/CISC**
  - Sandy Bridge (Core i7)
  - Atom
- Differences in tech node and frequency handled by estimate scaling to 45nm and 1GHz

<table>
<thead>
<tr>
<th>Table 2. Platform Summary.</th>
<th>32/64b x86 ISA</th>
<th>ARMv7 ISA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Sandybridge</td>
<td>Atom</td>
</tr>
<tr>
<td>Processor</td>
<td>Core 2700</td>
<td>N450</td>
</tr>
<tr>
<td>Cores</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Frequency</td>
<td>3.4 GHz</td>
<td>1.66 GHz</td>
</tr>
<tr>
<td>Width</td>
<td>4-way</td>
<td>2-way</td>
</tr>
<tr>
<td>Issue</td>
<td>OoO</td>
<td>In Order</td>
</tr>
<tr>
<td>L1 Data</td>
<td>32 KB</td>
<td>24 KB</td>
</tr>
<tr>
<td>L1 Inst</td>
<td>32 KB</td>
<td>32 KB</td>
</tr>
<tr>
<td>L2</td>
<td>256 KB/core</td>
<td>512 KB</td>
</tr>
<tr>
<td>Memory</td>
<td>16 GB</td>
<td>1 GB</td>
</tr>
<tr>
<td>SIMD</td>
<td>AVX</td>
<td>SSE</td>
</tr>
<tr>
<td>Area</td>
<td>216 mm²</td>
<td>66 mm²</td>
</tr>
<tr>
<td>Tech Node</td>
<td>32 nm</td>
<td>45 nm</td>
</tr>
<tr>
<td>Platform</td>
<td>Desktop</td>
<td>Dev Board</td>
</tr>
<tr>
<td>Products</td>
<td>Desktop</td>
<td>Netbook</td>
</tr>
</tbody>
</table>

- **Cortex-A9**
  - OMAP4430
- **Cortex-A8**
  - OMAP3530
- **Sandy Bridge (Core i7)**
- **Atom**
- **iPhone 4, 3GS**
- **Motorola Droid**
Choosing Workloads

- RISC and CISC both claim to be good for mobile, desktop, and server
- Single-threaded core-focused

Table 3. Benchmark Summary.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Benchmarks</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile</td>
<td>CoreMark</td>
<td>Set to 4000 iterations</td>
</tr>
<tr>
<td>client</td>
<td>WebKit</td>
<td>Similar to BBench</td>
</tr>
<tr>
<td>Desktop</td>
<td>SPECCPU2006</td>
<td>10 INT, 10 FP, test inputs</td>
</tr>
<tr>
<td>Server</td>
<td>lighttpd</td>
<td>Represents web-serving</td>
</tr>
<tr>
<td></td>
<td>CLucene</td>
<td>Represents web-indexing</td>
</tr>
<tr>
<td></td>
<td>Database kernels</td>
<td>Represents data-streaming and data-analytics</td>
</tr>
</tbody>
</table>
Metrics

- **Performance**
  - Wall-Clock Time
  - Built-In Cycle Counters

- **Power**
  - Wattsup
  - Multiple runs for average system power; control run for board power
  - Chip power = system power - board power
Key Findings (Perf)

- Execution time varies greatly
- Upon normalization to CPI and instruction count/mix, performance differences are explicable by microarchitectural differences (branch pred/cache size)
Key Findings (Power)

- i7 core is not power optimized so it has exceptionally high power
- Generally, core power is based on its optimization level
- Most differences in energy can be explained by differences in performance (e.g. BP) and power (Optimized for or not)
Trade-Off Analysis

- Cubic trade-off in power and performance
- Quadratic trade-off in energy and performance
- Pareto optimality not dependent on ISA
ISA does NOT affect performance, power, energy efficiency
Strengths

- Presents intuition first, then affirms with results
- Does a good job of drawing relevant data and conclusions with a severely limited scope
- Admit to several limitations in the paper itself
Weaknesses

- Comparison to performance optimized i7 Sandy Bridge core seems shaky -- could have used more similarly optimized technology for better results
  - Option 1: More test points so we can maybe group into power optimized, perf optimized, and somewhere in the middle
  - Option 2: Same number of test points but homogenous in use case
- Normalizing the cores to a specific frequency and technology node obfuscates the original purpose of the cores, which might differ from core to core (EDP?)
- Evaluation is now 7 years old, what differences might we expect to see in 2020 v 2013?