A Detailed Analysis of Contemporary ARM and x86 Architectures

Emily Blem, Jaikrishnan Menon, and Karthikeyan Sankaralingam
University of Wisconsin - Madison
{blem,menon,karu}@cs.wisc.edu

Abstract

RISC vs. CISC wars raged in the 1980s when chip area and processor design complexity were the primary constraints and desktops and servers exclusively dominated the computing landscape. Today, energy and power are the primary design constraints and the computing landscape is significantly different: growth in tablets and smartphones running ARM (a RISC ISA) is surpassing that of desktops and laptops running x86 (a CISC ISA). Further, the traditionally low-power ARM ISA is entering the high-performance server market, while the traditionally high-performance x86 ISA is entering the mobile low-power device market. Thus, the question of whether ISA plays an intrinsic role in performance or energy efficiency is becoming important, and we seek to answer this question through a detailed measurement based study on real hardware running real applications. We analyze measurements on the ARM Cortex-A8 and Cortex-A9 and Intel Atom and Sandybridge i7 microprocessors over workloads spanning mobile, desktop, and server computing. Our methodical investigation demonstrates the role of ISA in modern microprocessors' performance and energy efficiency. We find that ARM and x86 processors are simply engineering design points optimized for different levels of performance, and there is nothing fundamentally more energy efficient in one ISA class or the other. The ISA being RISC or CISC seems irrelevant.

1. Introduction

The question of ISA design and specifically RISC vs. CISC ISA was an important concern in the 1980s and 1990s when chip area and processor design complexity were the primary constraints [24, 12, 17, 7]. It is questionable if the debate was settled in terms of technical issues. Regardless, both flourished commercially through the 1980s and 1990s. In the past decade, the ARM ISA (a RISC ISA) has dominated mobile and low-power embedded computing domains and the x86 ISA (a CISC ISA) has dominated desktops and servers.

Recent trends raise the question of the role of the ISA and make a case for revisiting the RISC vs. CISC question. First, the computing landscape has quite radically changed from when the previous studies were done. Rather than being exclusively desktops and servers, today's computing landscape is significantly shaped by smartphones and tablets. Second, while area and chip design complexity were previously the primary constraints, energy and power constraints now dominate. Third, from a commercial standpoint, both ISAs are appearing in new markets: ARM-based servers for energy efficiency and x86-based mobile and low power devices for higher performance. Thus, the question of whether ISA plays a role in performance, power, or energy efficiency is once again important.

Related Work: Early ISA studies are instructive, but miss key changes in today's microprocessors and design constraints that have shifted the ISA's effect. We review previous comparisons in chronological order, and observe that all prior comprehensive ISA studies considering commercially implemented processors focused exclusively on performance.

Bhandarkar and Clark compared the MIPS and VAX ISA by comparing the M/2000 to the Digital VAX 8700 implementations [7] and concluded: "RISC as exemplified by MIPS provides a significant processor performance advantage." In another study in 1995, Bhandarkar compared the Pentium-Pro to the Alpha 21164 [6], again focused exclusively on performance and concluded: "...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." Consensus had grown that RISC and CISC ISAs had fundamental differences that led to performance gaps that required aggressive microarchitecture optimization for CISC which only partially bridged the gap.

Isen et al. [22] compared the performance of Power5+ to Intel Woodcrest considering SPEC benchmarks and concluded x86 matches the POWER ISA. The consensus was that "with aggressive microarchitectural techniques for ILP, CISC and RISC ISAs can be implemented to yield very similar performance."

Many informal studies in recent years claim the x86's "crufty" CISC ISA incurs many power overheads and attribute the ARM processor's power efficiency to the ISA [1, 2]. These studies suggest that the microarchitecture optimizations from the past decades have led to RISC and CISC cores with similar performance, but the power overheads of CISC are intractable.

In light of the prior ISA studies from decades past, the significantly modified computing landscape, and the seemingly vastly different power consumption of ARM implementations (1-2 W) to x86 implementations (5 - 36 W), we feel there is need to revisit this debate with a rigorous methodology. Specifically, considering the dominance of ARM and x86 and the multi-pronged importance of the metrics of power, energy, and perfor-

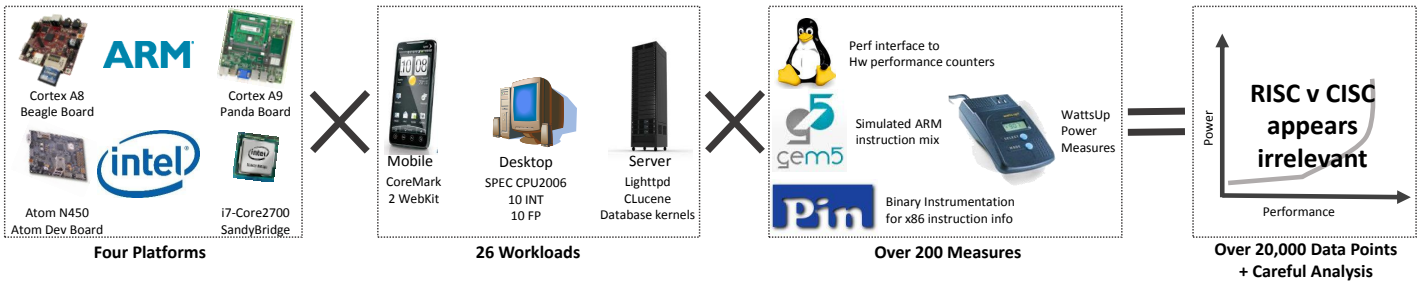


Figure 1. Summary of Approach.

mance, we need to compare ARM to x86 on those three metrics. Macro-op cracking and decades of research in high-performance microarchitecture techniques and compiler optimizations seemingly help overcome x86's performance and code-effectiveness bottlenecks, but these approaches are not free. The crux of our analysis is the following: *After decades of research to mitigate CISC performance overheads, do the new approaches introduce fundamental energy inefficiencies?*

Challenges: Any ISA study faces challenges in separating out the multiple implementation factors that are orthogonal to the ISA from the factors that are influenced or driven by the ISA. ISA-independent factors include chip process technology node, device optimization (high-performance, low-power, or low-standby power transistors), memory bandwidth, I/O device effects, operating system, compiler, and workloads executed. These issues are exacerbated when considering energy measurements/analysis, since chips implementing an ISA sit on boards and separating out chip energy from board energy presents additional challenges. Further, some microarchitecture features may be required by the ISA, while others may be dictated by performance and application domain targets that are ISA-independent.

To separate out the implementation and ISA effects, we consider multiple chips for each ISA with similar microarchitectures, use established technology models to separate out the technology impact, use the same operating system and compiler front-end on all chips, and construct workloads that do not rely significantly on the operating system. Figure 1 presents an overview of our approach: the four platforms, 26 workloads, and set of measures collected for each workload on each platform. We use multiple implementations of the ISAs and specifically consider the ARM and x86 ISAs representing RISC against CISC. We present an exhaustive and rigorous analysis using workloads that span smartphone, desktop, and server applications. In our study, we are primarily interested in *whether and, if so, how the ISA impacts performance and power.* We also discuss infrastructure and system challenges, missteps, and software/hardware bugs we encountered. Limitations are addressed in Section 3. Since there are many ways to analyze the raw data, this paper is accompanied by a public release of all data at `www.cs.wisc.edu/vertical/isa-power-struggles`.

Key Findings: The main findings from our study are:

- Large performance gaps exist across the *implementations*, although average cycle count gaps are $\leq 2.5\times$.
- Instruction count and mix are ISA-independent to first order.
- Performance differences are generated by ISA-independent microarchitecture differences.
- The energy consumption is again ISA-independent.
- ISA differences have implementation implications, but modern microarchitecture techniques render them moot; one ISA is not fundamentally more efficient.
- ARM and x86 *implementations* are simply design points optimized for different performance levels.

Implications: Our findings confirm known conventional (or suspected) wisdom, and add value by quantification. Our results imply that microarchitectural effects dominate performance, power, and energy impacts. The overall implication of this work is that the ISA being RISC or CISC is largely irrelevant for today's mature microprocessor design world.

Paper organization: Section 2 describes a framework we develop to understand the ISA's impacts on performance, power, and energy. Section 3 describes our overall infrastructure and rationale for the platforms for this study and our limitations, Section 4 discusses our methodology, and Section 5 presents the analysis of our data. Section 7 concludes.

2. Framing Key Impacts of the ISA

In this section, we present an intellectual framework in which to examine the impact of the ISA—assuming a von Neumann model—on performance, power, and energy. We consider the three key textbook ISA features that are central to the RISC/CISC debate: format, operations, and operands. We do not consider other textbook features, data types and control, as they are orthogonal to RISC/CISC design issues and RISC/CISC approaches are similar. Table 1 presents the three key ISA features in three columns and their general RISC and CISC characteristics in the first two rows. We then discuss contrasts for each feature and how the choice of RISC or CISC potentially and historically introduced significant trade-offs in performance and power. In the fourth row, we discuss how modern refinements have led to similarities, marginalizing the choice of RISC or CISC on performance and power. Finally, the last row raises empirical questions focused on each feature to quantify or validate this convergence. Overall, our approach is to understand all performance and power differences by using measured metrics to quantify the root cause of differences and whether or not

Table 1. Summary of RISC and CISC Trends.

	Format	Operations	Operands
RISC / ARM	◦ Fixed length instructions ◦ Relatively simple encoding ◦ ARM: 4B, THUMB(2B, optional)	◦ Simple, single function operations ◦ Single cycle	◦ Operands: registers, immediates ◦ Few addressing modes ◦ ARM: 16 general purpose registers
CISC / x86	◦ Variable length instructions ◦ Common insts shorter/simpler ◦ Special insts longer/complex ◦ x86: from 1B to 16B long	◦ Complex, multi-cycle instructions ◦ Transcendentals ◦ Encryption ◦ String manipulation	◦ Operands: memory, registers, immediates ◦ Many addressing modes ◦ x86: 8 32b & 6 16b registers
Historical Contrasts	◦ CISC decode latency prevents pipelining ◦ CISC decoders slower/more area ◦ Code density: RISC $<$ CISC	◦ Even w/ μcode, pipelining hard ◦ CISC latency may be longer than compiler's RISC equivalent	◦ CISC decoder complexity higher ◦ CISC has more per inst work, longer cycles ◦ Static code size: RISC $>$ CISC
Convergence Trends	◦ **μ-op cache minimizes decoding overheads** ◦ **x86 decode optimized for common insts** ◦ **I-cache minimizes code density impact**	◦ **CISC insts split into RISC-like micro-ops; optimizations eliminated inefficiencies** ◦ **Modern compilers pick mostly RISC insts; μ-op counts similar for ARM and x86**	◦ **x86 decode optimized for common insts** ◦ **CISC insts split into RISC-like micro-ops; x86 and ARM μ-op latencies similar** ◦ **Number of data cache accesses similar**
Empirical Questions	◦ How much variance in x86 inst length? Low variance ⇒ common insts optimized ◦ Are ARM and x86 code densities similar? Similar density ⇒ No ISA effect ◦ What are instruction cache miss rates? Low ⇒ caches hide low code densities	◦ Are macro-op counts similar? Similar ⇒ RISC-like on both ◦ Are complex instructions used by x86 ISA? Few complex ⇒ Compiler picks RISC-like ◦ Are μ-op counts similar? Similar ⇒ CISC split into RISC-like μ-ops	◦ Number of data accesses similar? Similar ⇒ no data access inefficiencies

ISA differences contribute. The remainder of this paper is centered around these empirical questions framed by the intuition presented as the convergence trends.

Although whether an ISA is RISC or CISC seems irrelevant, ISAs are evolving; expressing more semantic information has led to improved performance (x86 SSE, larger address space), better security (ARM Trustzone), better virtualization, etc. Examples in current research include extensions to allow the hardware to balance accuracy with energy efficiency [15, 13] and extensions to use specialized hardware for energy efficiency [18]. We revisit this issue in our conclusions.

3. Infrastructure

We now describe our infrastructure and tools. The key takeaway is that we pick four platforms, doing our best to keep them on equal footing, pick representative workloads, and use rigorous methodology and tools for measurement. Readers can skip ahead to Section 4 if uninterested in the details.

3.1. Implementation Rationale and Challenges

Choosing implementations presents multiple challenges due to differences in technology (technology node, frequency, high performance/low power transistors, etc.); ISA-independent microarchitecture (L2-cache, memory controller, memory size, etc.); and system effects (operating system, compiler, etc.). Finally, platforms must be commercially relevant and it is unfair to compare platforms from vastly different time-frames.

We investigated a wide spectrum of platforms spanning Intel Nehalem, Sandybridge, AMD Bobcat, NVIDIA Tegra-2, NVIDIA Tegra-3, and Qualcomm Snapdragon. However, we did not find implementations that met all of our criteria: same technology node across the different ISAs, identical or similar microarchitecture, development board that supported necessary measurements, a well-supported operating system, and similar I/O and memory subsystems. We ultimately picked the Beagleboard (Cortex-A8), Pandaboard (Cortex-A9), and Atom board, as they include processors with similar microarchitectural features like issue-width, caches, and main-memory and are from similar technology nodes, as described in Tables 2 and 7. They are all relevant commercially as shown by the last row in Table 2. For a high performance x86 processor, we use an Intel i7 Sandybridge processor; it is significantly more power-efficient than any 45nm offering, including Nehalem. Importantly, these choices provided usable software platforms in terms of operating system, cross-compilation, and driver support. Overall, our choice of platforms provides a reasonably equal footing, and we perform detailed analysis to isolate out microarchitecture and technology effects. We present system details of our platforms for context, although the focus of our work is the processor core.

A key challenge in running real workloads was the relatively small memory (512MB) on the Cortex-A8 Beagleboard. While representative of the typical target (e.g., iPhone 4 has 512MB RAM), it presents a challenge for workloads like SPEC-CPU2006; execution times are dominated by swapping and OS overheads, making the core irrelevant. Section 3.3 describes how we handled this. In the remainder of this section, we discuss the platforms, applications, and tools for this study in detail.

3.2. Implementation Platforms

Hardware platform: We consider two chip implementations each for the ARM and x86 ISAs as described in Table 2.
Intent: Keep non-processor features as similar as possible.

Table 2. Platform Summary.

	32/64b x86 ISA		ARMv7 ISA	
Architecture	Sandybridge	Atom	Cortex-A9	Cortex-A8
Processor	Core 2700	N450	OMAP4430	OMAP3530
Cores	4	1	2	1
Frequency	3.4 GHz	1.66 GHz	1 GHz	0.6 GHz
Width	4-way	2-way	2-way	2-way
Issue	OoO	In Order	OoO	In Order
L1 Data	32 KB	24 KB	32 KB	16 KB
L1 Inst	32 KB	32 KB	32 KB	16 KB
L2	256 KB/core	512 KB	1 MB/chip	256 KB
L3	8 MB/chip	—	—	—
Memory	16 GB	1 GB	1 GB	256 MB
SIMD	AVX	SSE	NEON	NEON
Area	216 mm^2	66 mm^2	70 mm^2	60 mm^2
Tech Node	32 nm	45 nm	45 nm	65 nm
Platform	Desktop	Dev Board	Pandaboard	Beagleboard
Products	Desktop	Netbook	Galaxy S-III	iPhone 4, 3GS
		Lava Xolo	Galaxy S-II	Motorola Droid

Data from TI OMAP3530, TI OMAP4430, Intel Atom N450, and Intel i7-2700 datasheets, `www.beagleboard.org` & `www.pandaboard.org`

Operating system: Across all platforms, we run the same stable Linux 2.6 LTS kernel with some minor board-specific patches to obtain accurate results when using the performance counter subsystem. We use `perf`'s[1] program sampling to find the fraction of time spent in the kernel while executing the SPEC benchmarks on all four boards; overheads were less than 5% for all but `GemsFDTD` and `perlbench` (both less than 10%) and the fraction of time spent in the operating system was virtually identical across platforms spanning ISAs.

Intent: Keep OS effects as similar as possible across platforms.

Compiler: Our toolchain is based on a validated `gcc` 4.4 based cross-compiler configuration. We intentionally chose `gcc` so that we can use the same front-end to generate all binaries. All target independent optimizations are enabled (O3); machine-specific tuning is disabled so there is a single set of ARM binaries and a single set of x86 binaries. For x86 we target 32-bit since 64-bit ARM platforms are still under development. For ARM, we disable THUMB instructions for a more RISC-like ISA. We ran experiments to determine the impact of machine-specific optimizations and found that these impacts were less than 5% for over half of the SPEC suite, and caused performance variations of $\pm$20% on the remaining with speed-ups and slow-downs equally likely. None of the benchmarks include SIMD code, and although we allow auto-vectorization, very few SIMD instructions are generated for either architecture. Floating point is done natively on the SSE (x86) and NEON (ARM) units. Vendor compilers may produce better code for a platform, but we use `gcc` to eliminate compiler influence. As seen in Table 12 in Appendix I, static code size is within 8% and average instruction lengths are within 4% using `gcc` and `icc` for SPEC INT, so we expect that compiler does not make a significant difference.

Intent: Hold compiler effects constant across platforms.

3.3. Applications

Since both ISAs are touted as candidates for mobile clients, desktops, and servers, we consider a suite of workloads that span these. We use prior workload studies to guide our choice, and where appropriate we pick equivalent workloads that can run on our evaluation platforms. A detailed description follows and is summarized in Table 3. All workloads are single-threaded to ensure our single-core focus.

Table 3. Benchmark Summary.

Domain	Benchmarks	Notes
Mobile client	CoreMark	Set to 4000 iterations
	WebKit	Similar to BBench
Desktop	SPECCPU2006	10 INT, 10 FP, test inputs
Server	lighttpd	Represents web-serving
	CLucene	Represents web-indexing
	Database kernels	Represents data-streaming and data-analytics

Mobile client: This category presented challenges as mobile client chipsets typically include several accelerators and careful analysis is required to determine the typical workload executed on the programmable general-purpose core. We used CoreMark (`www.coremark.org`), widely used in industry white-papers, and two WebKit regression tests informed by the BBench study [19]. BBench, a recently proposed smartphone benchmark suite, is a "a web-page rendering benchmark comprising 11 of the most popular sites on the internet today" [19]. To avoid web-browser differences across the platforms, we use the cross-platform WebKit with two of its built-in tests that mimic real-world HTML layout and performance scenarios for our study[2].

Desktop: We use the SPECCPU2006 suite (`www.spec.org`) as representative of desktop workloads. SPECCPU2006 is a well understood standard desktop benchmark, providing insights into core behavior. Due to the large memory footprint of the train and reference inputs, we found that for many benchmarks the memory constrained Cortex-A8, in particular, ran of memory and execution was dominated by system effects. Instead, we report results using the test inputs, which fit in the Cortex-A8's memory footprint for 10 of 12 INT and 10 of 17 FP benchmarks.

Server: We chose server workloads informed by the CloudSuite workloads recently proposed by Ferdman et al. [16]. Their study characterizes server/cloud workloads into data analytics, data streaming, media streaming, software testing, web search, and web serving. The actual software implementations they provide are targeted for large memory-footprint machines and their intent is to benchmark the *entire system and server cluster*. This is unsuitable for our study since we want to isolate processor effects. Hence, we pick implementations with small memory footprints and single-node behavior. To represent data-streaming and data-analytics, we use three database kernels commonly used in database evaluation work [26, 23] that capture the core computation in Bayes classification and data-store[3]. To represent web search, we use `CLucene` (`clucene.`

[1] `perf` is a Linux utility to access performance counters.

[2] Specifically `coreLayout` and `DOMPerformance`.

[3] CloudSuite uses Hadoop+Mahout plus additional software infrastructure, ultimately running Bayes classification and data-store; we feel this kernel approach is better suited for our study while capturing the domain's essence.

Table 4. Infrastructure Limitations.

	Limitation	Implications
Cores	Multicore effects: coherence, locking...	2nd order for core design
	No platform uniformity across ISAs	Best effort
	No platform diversity within ISAs	Best effort
	Design teams are different	μarch effect, not ISA
	"Pure" RISC, CISC implementations	Out of scope
Domain	Ultra low power microcontrollers	Out of scope
	Server style platforms	See server benchmarks
	Why SPEC on mobile platforms?	Tracks emerging uses
	Why not SPEC JBB or TPC-C?	CloudSuite more relevant
Tools	Proprietary compilers are optimized	`gcc` optimizations uniform
	Arch. specific compiler tuning	$<$10%
	No direct decoder power measure	Results show 2nd order
	Power includes non-core factors	4-17%
	Performance counters may have errors	Validated use (Table 5)
	Simulations have errors	Validated use (Table 5)
Scaling	Memory rate effects cycles nonlinearly	Second-order
	Vmin limit effects frequency scaling	Second-order
	ITRS scaling numbers are not exact	Best effort; extant nodes

`sourceforge.net`), an efficient, cross-platform indexing implementation similar to CloudSuite's Nutch. To represent web-serving (CloudSuite uses Apache), we use the `lighttpd` server (`www.lighttpd.net`) which is designed for "security, speed, compliance, and flexibility"[4]. We do not evaluate the media-streaming CloudSuite benchmark as it primarily stresses the I/O subsystem. CloudSuite's Software Testing benchmark is a batch coarse-grained parallel symbolic execution application; for our purposes, the SPEC suite's Perl parser, combinational optimization, and linear programming benchmarks are similar.

3.4. Tools

The four main tools we use in our work are described below and Table 5 in Section 4 describes how we use them.

Native execution time and microarchitectural events: We use wall-clock time and performance-counter-based clock-cycle measurements to determine execution time of programs. We also use performance counters to understand microarchitecture influences on the execution time. Each of the processors has different counters available, and we examined them to find comparable measures. Ultimately three counters explain much of the program behavior: branch mis-prediction rate, Level-1 data-cache miss rate, and Level-1 instruction-cache miss rate (all measured as misses per kilo-instructions). We use the `perf` tool for performance counter measurement.

Power: For power measurements, we connect a Wattsup (`www.wattsupmeters.com`) meter to the board (or desktop) power supply. This gives us system power. We run the benchmark repeatedly to find consistent average power as explained in Table 5. We use a control run to determine the board power alone when the processor is halted and subtract away this board power to determine chip power. Some recent power studies [14, 21, 9] accurately isolate the processor power alone by measuring the current supply line of the processor. This is not possible for the SoC-based ARM development boards, and hence we determine and then subtract out the board-power. This methodology allows us to eliminate the main memory and I/O power and examine only processor power. We validated our strategy for the i7 system using the exposed energy counters (the only platform we consider that includes isolated power measures). Across all three benchmark suites, our WattsUp methodology compared to the processor energy counter reports ranged from 4% to 17% less, averaging 12%. Our approach tends to under-estimate core power, so our results for power and energy are optimistic. We saw average power of 800mW, 1.2W, 5.5W, and 24W for A8, A9, Atom, and i7 (respectively) and these fall within the typical vendor-reported power numbers.

Technology scaling and projections: Since the i7 processor is 32nm and the Cortex-A8 is 65nm, we use technology node characteristics from the 2007 ITRS tables to normalize to the 45nm technology node in two results where we factor out technology; we do not account for device type (LOP, HP, LSTP). For our 45nm projections, the A8's power is scaled by 0.8$\times$ and the i7's power by 1.3$\times$. In some results, we scale frequency to 1 GHz, accounting for DVFS impact on voltage using the mappings disclosed for Intel SCC [5]. When frequency scaling, we assume that 20% of the i7's power is static and does not scale with frequency; all other cores are assumed to have negligible static power. When frequency scaling, A8's power is scaled by 1.2$\times$, Atom's power by 0.8$\times$, and i7's power by 0.6$\times$. We acknowledge that this scaling introduces some error to our technology-scaled power comparison, but feel it is a reasonable strategy and doesn't affect our primary findings (see Table 4).

Emulated instruction mix measurement: For the x86 ISA, we use DynamoRIO [11] to measure instruction mix. For the ARM ISA, we leverage the gem5 [8] simulator's functional emulator to derive instruction mixes (no ARM binary emulation available). Our server and mobile-client benchmarks use many system calls that do not work in the gem5 functional mode. We do not present detailed instruction-mix analysis for these, but instead present high-level mix determined from performance counters. We use the MICA tool to find the available ILP [20].

3.5. Limitations or Concerns

Our study's limitations are classified into core diversity, domain, tool, and scaling effects. The full list appears in Table 4, and details are discussed below. Throughout our work, we focus on what we believe to be the first order effects for performance, power, and energy and feel our analysis and methodology is rigorous. Other more detailed methods may exist, and we have made the data publicly available at `www.cs.wisc.edu/vertical/isa-power-struggles` to allow interested readers to pursue their own detailed analysis.

Cores: We considered four platforms, two from each ISA. A perfect study would include platforms at several performance levels with matched frequency, branch predictors, other microarchitectural features, and memory systems. Further, a pure RISC versus CISC study would use true RISC and CISC cores, while ARM and x86's ISA tweaks represent the current state-of-the-art. Our selections reflect the available, well-supported imple-

[4] Real users of lighttpd include YouTube.

Table 5. Methodology Summary.
(a) Native Execution on Real Hardware

Measures	Methodology
Execution time, Cycle counts	○ Approach: Use `perf` tool to sample cycle performance counters; sampling avoids potential counter overflow. ○ Analysis: 5 - 20 trials (dependent on variance and benchmark runtime); report minimum from trials that complete normally. ○ Validation: Compare against wall clock time.
Inst. count (ARM)	○ Approach: Use `perf` tool to collect macro-ops from performance counters ○ Analysis: At least 3 trials; report minimum from trials that complete normally. ○ Validation: Performance counters within 10% of gem5 ARM simulation. Table 9 elaborates on challenges.
Inst. count (x86)	○ Approach: Use `perf` to collect macro-ops and micro-ops from performance counters. ○ Analysis: At least 3 trials; report minimum from trials that complete normally. ○ Validation: Counters within 2% of DynamoRIO trace count (macro-ops only). Table 9 elaborates on challenges.
Inst. mix (Coarse)	○ Approach: SIMD + FP + load/store performance counters.
Inst. length (x86)	○ Approach: Wrote Pin tool to find length of each instruction and keep running average.
Microarch events	○ Approach: Branch mispredictions, cache misses, and other uarch events measured using `perf` performance counters. ○ Analysis: At least 3 trials; additional if a particular counter varies by $> 5\%$. Report minimum from normal trials.
Full system power	○ Set-up: Use Wattsup meter connected to board or desktop (no network connection, peripherals on separate supply, kernel DVFS disabled, cores at peak frequency, single-user mode). ○ Approach: Run benchmarks in loop to guarantee 3 minutes of samples (180 samples at maximum sampling rate). ○ Analysis: If outliers occur, rerun experiment; present average power across run without outliers.
Board power	○ Set-up: Use Wattsup meter connected to board or desktop (no network connection, peripherals on separate supply, kernel DVFS disabled, cores at peak frequency, single-user mode). ○ Approach: Run with kernel power saving enabled; force to lowest frequency. Issue halt; report power when it stabilizes. ○ Analysis: Report minimum observed power.
Processor power	○ Approach: Subtracting above two gives processor power. ○ Validation: compare core power against energy performance counters and/or reported TDP and power draw.

(b) Emulated Execution

Measures	Methodology
Inst. mix (Detailed)	○ Approach (ARM): Use gem5 instruction trace and analyze using python script. ○ Approach (x86): Use DynamoRIO instruction trace and analyze using python script. ○ Validation: Compare against coarse mix from SIMD + FP + load/store performance counters.
ILP	○ Approach: Pin based MICA tool which reports ILP with window size 32, 64, 128, 256.

mentations available to our group. The impact of higher performance emerging cores is included in our synthetic processor study.

Domain: We picked a representative set of workloads we feel captures a significant subset of modern workloads. We do not make broad domain-specific arguments, since that requires truly representative inputs and IO subsystem control for the mobile and server domains. Our study focused on single-core, and thus intentionally avoids multi-core system issues (e.g., consistency models, coherence, virtualization, etc.).

Measurement and tool errors: Our measurements are primarily on real hardware, and therefore include real world errors. We execute multiple runs and take a rigorous approach as detailed in Table 5. Eliminating all errors is impractical, and our final result trends are consistent and intuitive.

Analysis: We have presented our analysis of this rich data set, and will release the data and our analysis scripts to allow interested readers to pursue their own detailed analysis.

4. Methodology

In this section, we describe how we use our tools and the overall flow of our analysis. Section 5 presents our data and analysis. Table 5 describes how we employ the aforementioned tools and obtain the measures we are interested in, namely, execution time, execution cycles, instruction-mix, microarchitecture events, power, and energy.

Our overall approach is to understand all performance and power *differences* and use the measured metrics to quantify the root cause of differences and whether or not ISA differences contribute, answering empirical questions from Section 2. Unless otherwise explicitly stated, all data is measured on real hardware. The flow of the next section is outlined below.

4.1. Performance Analysis Flow

Step 1: Present execution time for each benchmark.
Step 2: Normalize frequency's impact using cycle counts.
Step 3: To understand differences in cycle count and the influence of the ISA, present the dynamic instruction count measures, measured in both macro-ops and micro-ops.
Step 4: Use instruction mix, code binary size, and average dynamic instruction length to understand ISA's influence.
Step 5: To understand performance differences not attributable to ISA, look at detailed microarchitecture events.
Step 6: Attribute performance gaps to frequency, ISA, or ISA-

independent microarchitecture features. Qualitatively reason about whether the ISA forces microarchitecture features.

4.2. Power and Energy Analysis Flow

Step 1: Present per benchmark raw power measurements.
Step 2: To factor out the impact of technology, present technology-independent power by scaling all processors to 45nm and normalizing the frequency to 1 GHz.
Step 3: To understand the interplay between power and performance, examine raw energy.
Step 4: Qualitatively reason about the ISA influence on microarchitecture in terms of energy.

4.3. Trade-off Analysis Flow

Step 1: Combining the performance and power measures, compare the processor implementations using Pareto-frontiers.
Step 2: Compare measured and synthetic processor implementations using Energy-Performance Pareto-frontiers.

5. Measured Data Analysis and Findings

We now present our measurements and analysis of performance, power, energy, and the trade-offs between them. We conclude the section with sensitivity studies projecting performance of additional implementations of the ARM and x86 ISA using a simple performance and power model.

We present our data for all four platforms, often comparing A8 to Atom (both dual-issue in-order) and A9 to i7 (both OOO) since their implementations are pair-wise similar. For each step, we present the average measured data, average in-order and OoO ratios if applicable, and then our main findings. *When our analysis suggests that some benchmarks are outliers, we give averages with the outliers included in parentheses.*

5.1. Performance Analysis

Step 1: Execution Time Comparison

Data: Figure 2 shows execution time normalized to i7; averages including outliers are given using parentheses. Average ratios are in table below. Per benchmark data is in Figure 16 of Appendix I.

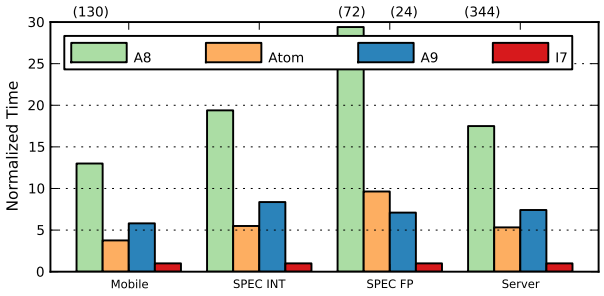


Figure 2. Execution Time Normalized to i7.

Ratio	Mobile	SPEC INT	SPEC FP	Server
A8 to Atom	3.4 (34)	3.5	4.2 (7.4)	3.7 (103)
A9 to i7	5.8	8.4	7.2 (23)	7.4

Outliers: A8 performs particularly poorly on WebKit tests and `lighttpd`, skewing A8/Atom differences in the mobile and server data, respectively; see details in Step 2. Five SPEC FP benchmarks are also considered outliers; see Table 8. Where outliers are listed, they are in this set.

Finding P1: Large performance gaps are platform and benchmark dependent: A9 to i7 performance gaps range from 5× to 102× and A8 to Atom gaps range from 2× to 997×.

Key Finding 1: Large performance gaps exist across the four platforms studied, as expected, since frequency ranges from 600 MHz to 3.4 GHz and microarchitectures are very different.

Step 2: Cycle-Count Comparison

Data: Figure 3 shows cycle counts normalized to i7. Per benchmark data is in Figure 7.

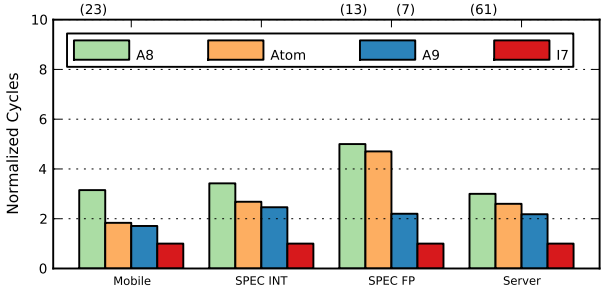


Figure 3. Cycle Count Normalized to i7.

Ratio	Mobile	SPEC INT	SPEC FP	Server
A8 to Atom	1.2 (12)	1.2	1.5 (2.7)	1.3 (23)
A9 to i7	1.7	2.5	2.1 (7.0)	2.2

Finding P2: Per suite cycle count gaps between out-of-order implementations A9 and i7 are less than 2.5× (no outliers).

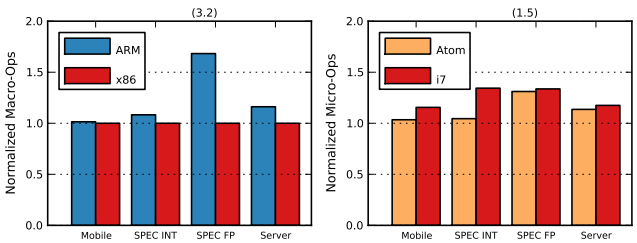
Finding P3: Per suite cycle count gaps between in-order implementations A8 and Atom are less than 1.5× (no outliers).

Key Finding 2: Performance gaps, when normalized to cycle counts, are less than 2.5× when comparing in-order cores to each other and out-of-order cores to each other.

Step 3: Instruction Count Comparison

Data: Figure 4a shows dynamic instruction (macro) counts on A8 and Atom normalized to Atom x86 macro-instructions. Per benchmark data is in Figure 17a of Appendix I. Per benchmark data for CPIs is in Table 11 in Appendix I.

Data: Figure 4b shows dynamic micro-op counts for Atom and i7 normalized to Atom macro-instructions[5]. Per benchmark data is in Figure 17b. of Appendix I



(a) Macro-Ops (b) Micro-Ops

Figure 4. Instructions Normalized to i7 macro-ops.

[5]For i7, we use issued micro-ops instead of retired micro-ops; we found that on average, this does not impact the micro-op/macro-op ratio.

Outliers: For `wkperf` and `lighttpd`, A8 executes more than twice as many instructions as A9[6]. We report A9 instruction counts for these two benchmarks. For `CLucene`, x86 machines execute 1.7× more instructions than ARM machines; this appears to be a pathological case of x86 code generation inefficiencies. For `cactusADM`, Atom executes 2.7× more micro-ops than macro-ops; this extreme is not seen for other benchmarks.

Finding P4: Instruction count similar across ISAs. Implies `gcc` picks the RISC-like instructions from the x86 ISA.

Finding P5: All ARM outliers in SPEC FP due to transcendental FP operations supported only by x86.

Finding P6: x86 micro-op to macro-op ratio is often less than 1.3×, again suggesting `gcc` picks the RISC-like instructions.

Key Finding 3: Instruction and cycle counts imply CPI is *less* on x86 implementations: geometric mean CPI is 3.4 for A8, 2.2 for A9, 2.1 for Atom, and 0.7 for i7 across all suites. x86 ISA overheads, if any, are overcome by microarchitecture.

Step 4: Instruction Format and Mix

Data: Table 6a shows average ARM and x86 static binary sizes, measuring only the binary's instruction segment. Per benchmark data is in Table 12a in Appendix I.

Data: Table 6b shows average dynamic ARM and x86 instruction lengths. Per benchmark data is in Table 12b in Appendix I.

Table 6. Instruction Size Summary.

		(a) Binary Size (MB)		(b) Instruction Length (B)	
		ARM	x86	ARM	x86
Mobile	Minimum	0.02	0.02	4.0	2.4
	Average	0.95	0.87	4.0	3.3
	Maximum	1.30	1.42	4.0	3.7
Desktop INT	Minimum	0.53	0.65	4.0	2.7
	Average	1.47	1.46	4.0	3.1
	Maximum	3.88	4.05	4.0	3.5
Desktop FP	Minimum	0.66	0.74	4.0	2.6
	Average	1.70	1.73	4.0	3.4
	Maximum	4.75	5.24	4.0	6.4
Server	Minimum	0.12	0.18	4.0	2.5
	Average	0.39	0.59	4.0	3.2
	Maximum	0.47	1.00	4.0	3.7

Outliers: `CLucene` binary (from server suite) is almost 2× larger for x86 than ARM; the server suite thus has the largest span in binary sizes. ARM executes correspondingly few instructions; see outliers discussion in Step 3.

Finding P7: Average ARM and x86 binary sizes are similar for SPEC INT, SPEC FP, and Mobile workloads, suggesting similar code densities.

Finding P8: Executed x86 instructions are on average up to 25% shorter than ARM instructions: short, simple x86 instructions are typical.

Finding P9: x86 FP benchmarks, which tend to have more complex instructions, have instructions with longer encodings (e.g., `cactusADM` with 6.4 Bytes/inst on average).

[6] A8 spins for IO, event-loops, and timeouts.

Data: Figure 5 shows average coarse-grained ARM and x86 instruction mixes for each benchmark suite[7].

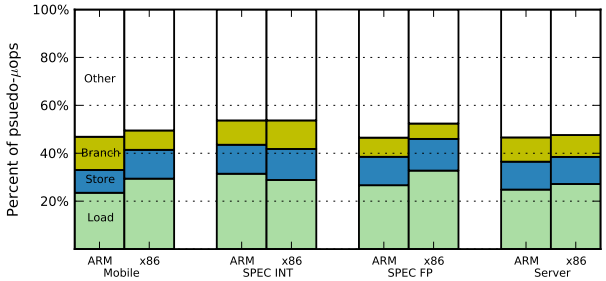


Figure 5. Instruction Mix (Performance Counters).

Data: Figure 6 shows fine-grained ARM and x86 instruction mixes normalized to x86 for a subset of SPEC benchmarks[7].

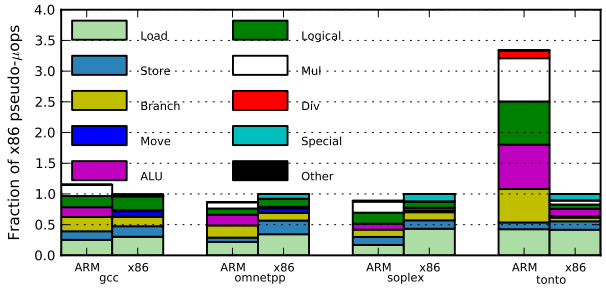


Figure 6. Selected Instruction Counts (Emulated).

Finding P10: Fraction of loads and stores similar across ISA for all suites, suggesting that the ISA does not lead to significant differences in data accesses.

Finding P11: Large instruction counts for ARM are due to absence of FP instructions like `fsincon, fyl2xpl`, (e.g., `tonto` in Figure 6's many special x86 instructions correspond to ALU/logical/multiply ARM instructions).

Key Finding 4: Combining the instruction-count and mix-findings, we conclude that ISA effects are indistinguishable between x86 and ARM implementations.

Step 5: Microarchitecture

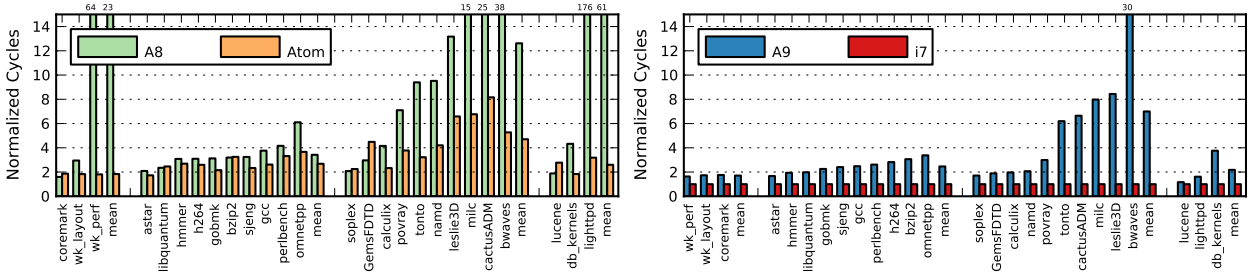
Data: Figure 7 shows the per-benchmark cycle counts for more detailed analysis where performance gaps are large. The raw data for this figure is in the `Cycles` worksheet of our publicly released spreadsheet [10].

Data: Table 7 compares the A8 microarchitecture to Atom, and A9 to i7, focusing on the primary structures. These details are from five Microprocessor Report articles[8] and the A9 numbers are estimates derived from publicly disclosed information on A15 and A9/A15 comparisons.

[7] x86 instructions with memory operands are cracked into a memory operation and the original operation.

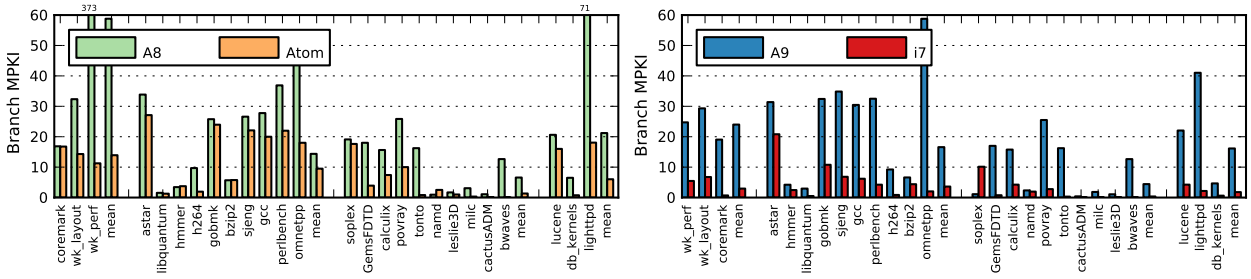
[8] "Cortex-A8 High speed, low power" (Nov 2005), "More applications for OMAP4" (Nov 2009), " Sandybridge spans generations" (Sept 2010), "Intel's Tiny Atom" (April 2008), "Cortex A-15 Eagle Flies the Coop" (Nov 2010).

[9] 60 for A15.



(a) In-Order (b) Out-of-Order

Figure 7. Cycle Counts Normalized to i7.



(a) In-Order (b) Out-of-Order

Figure 8. Branch Misses per 1000 ARM Instructions.

Table 7. Processor Microarchitecture Features.

(a) In-Order Cores

	Pipeline Depth	Issue Width	Threads	ALU/FP Units	Br. Pred. BTB Entries
A8	13	2	1	2/2 + NEON	512
Atom	16+2	2	2	2/2 + IMul	128

(b) Out-of-Order Cores

	Issue width	Threads	ROB Size	Entries for LD/ST	Rename	Scheduler	BTB
A9	4	1	- [9]	-/4	56	20	512
i7	4(6)	2	64/36	160	168	54	8K - 16K

Finding P12: A9 and i7's different issue widths (2 versus 4, respectively)[10] explain performance differences up to 2×, assuming sufficient ILP, a sufficient instruction window and a well balanced processor pipeline. We use MICA to confirm that our benchmarks all have limit ILP greater than 4 [20].

Finding P13: Even with different ISAs and significant differences in microarchitecture, for 12 benchmarks, the A9 is within 2× the cycle count of i7 and can be explained by the difference in issue width.

Data: Figures 8, 9, and 10 show branch mispredictions & L1 data and instruction cache misses per 1000 ARM instructions. The raw data for these figures is in the `Branch Misses`, `L1 Data Misses`, and `L1 Inst Misses` worksheets, respectively, of our publicly released spreadsheet [10].

[10]We assume the conventional wisdom that A9 is dual issue, although its pipeline diagrams indicate it is quad-issue.

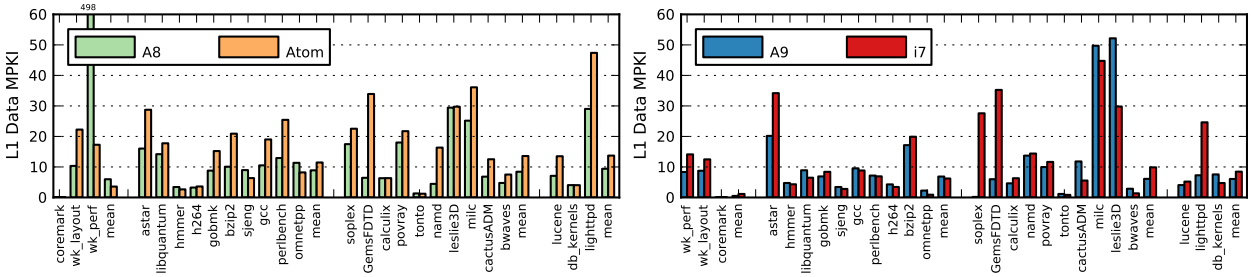
Finding P14: Observe large microarchitectural event count differences (e.g., A9 branch misses are more common than i7 branch misses). These differences are not because of the ISA, but rather due to microarchitectural design choices (e.g., A9's BTB has 512 entries versus i7's 16K entries).

Finding P15: Per benchmark, we can attribute the largest gaps in i7 to A9 performance (and in Atom to A8 performance) to specific microachitectural events. In the interest of space, we present example analyses for those benchmarks with gaps greater than 3.0× in Table 8; `bwaves` details are in Appendix II.

Key Finding 5: The microarchitecture has significant impact on performance. The ARM and x86 architectures have similar instruction counts. The highly accurate branch predictor and large caches, in particular, effectively allow x86 architectures to sustain high performance. x86 performance inefficiencies, if any, are not observed. The microarchitecture, not the ISA, is responsible for performance differences.

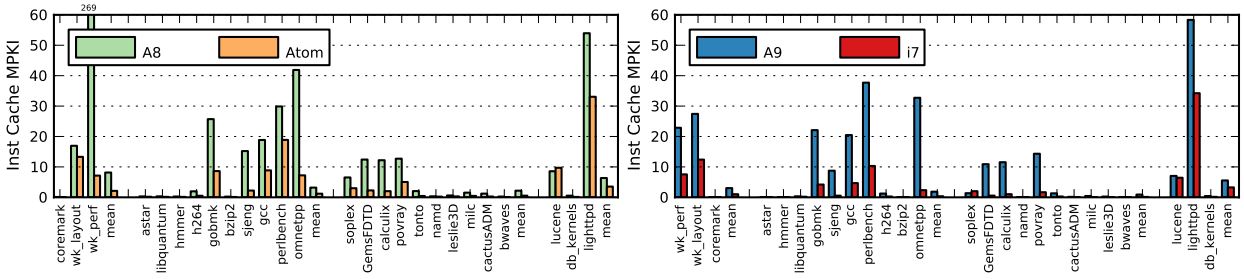
Step 6: ISA influence on microarchitecture

Key Finding 6: As shown in Table 7, there are significant differences in microarchitectures. Drawing upon instruction mix and instruction count analysis, we feel that the only case where the ISA forces larger structures is on the ROB size, physical rename file size, and scheduler size since there are almost the same number of x86 micro-ops in flight compared to ARM instructions. The difference is small enough that we argue it is not necessary to quantify further. Beyond the translation to micro-ops, pipelined implementation of an x86 ISA introduces no additional overheads over an ARM ISA for these performance levels.



(a) In-Order (b) Out-of-Order

Figure 9. Data L1 Misses per 1000 ARM Instructions.



(a) In-Order (b) Out-of-Order

Figure 10. Instruction Misses per 1000 ARM Instructions.

5.2. Power and Energy Analysis

In this section, we normalize to A8 as it uses the least power.

Step 1: Average Power

Data: Figure 11 shows average power normalized to the A8. Per benchmark data is in Figure 18 of Appendix I.

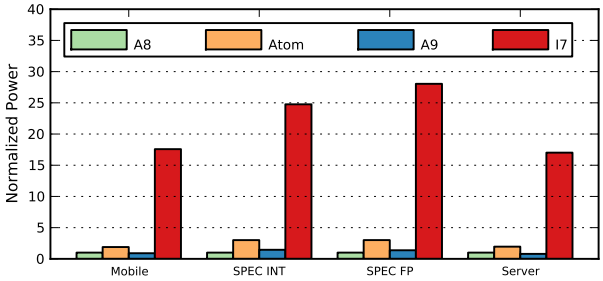


Figure 11. Raw Average Power Normalized to A8.

Ratio	Mobile	SPEC INT	SPEC FP	Server
Atom to A8	3.0	3.1	3.1	3.0
i7 to A9	20	17	20	21

Key Finding 7: Overall x86 implementations consume significantly more power than ARM implementations.

Step 2: Average Technology Independent Power

Data: Figure 12 shows technology-independent average power–cores are scaled to 1 GHz at 45nm (normalized to A8). Per benchmark data is in Figure 19 of Appendix I.

Ratio	Mobile	SPEC INT	SPEC FP	Server
Atom to A8	0.6	0.6	0.6	0.6
i7 to A9	7.0	6.1	7.4	7.6

Finding E1: With frequency and technology scaling, ISA appears irrelevant for power optimized cores: A8, A9, and Atom are all within 0.6× of each other (A8 consumes 29% *more* power than A9). Atom is actually lower power than A8 or A9.

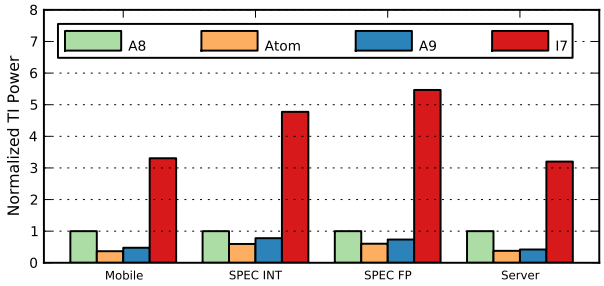


Figure 12. Tech. Independent Avg. Power Normalized to A8.

Finding E2: i7 is performance, not power, optimized. Per suite power costs are 6.1× to 7.6× higher for i7 than A9 with 1.7× to 7.0× higher frequency-independent performance (Figure 3 cycle count performance).

Key Finding 8: The choice of power or performance optimized core designs impacts core power use more than ISA.

Step 3: Average Energy

Data: Figure 13 shows energy (product of power and time). Per benchmark data is in Figure 20 of Appendix I.

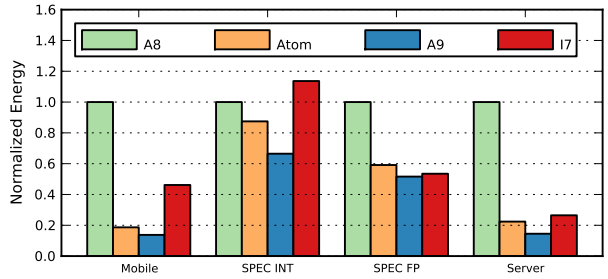


Figure 13. Raw Average Energy Normalized to A8.

Table 8. Detailed Analysis for Benchmarks with A9 to i7 Gap Greater Than 3×.

Benchmark	Gap	Analysis
`omnetpp`	3.4	Branch MPKI: 59 for A9 versus only 2.0 for i7; I-Cache MPKI: 33 for A9 versus only 2.2 for i7.
`db_kernels`	3.8	1.6× more instructions, 5× more branch MPKI for A9 than i7.
`tonto`	6.2	Instructions: 4× more for ARM than x86.
`cactusADM`	6.6	Instructions: 2.8× more for ARM than x86.
`milc`	8.0	A9 and i7 both experience more than 50 data cache MPKI. i7's microarchitecture hides these misses more effectively.
`leslie3D`	8.4	4× as many L2 cache misses using the A8 than using the Atom explains the 2× A8 to Atom gap. On the A9, the data cache MPKI is 55, compared to only 30 for the i7.
`bwaves`	30	324× more branch MPKI, 17.5× more instructions, 4.6× more instruction MPKI, and 6× more L2 cache misses on A8 than Atom. A9 has similar trends, including 1000× more branch MPKI than the i7.

Ratio	Mobile	SPEC INT	SPEC FP	Server
A8 to Atom	0.8(0.1)	0.9	0.8 (0.6)	0.8(0.2)
i7 to A9	3.3	1.7	1.7 (1.0)	1.8

Finding E3: Despite power differences, Atom consumes less energy than A8 and i7 uses only slightly more energy than A9 due primarily to faster execution times, not ISA.

Finding E4: For "hard" benchmarks with high cache miss rates that leave the core poorly utilized (e.g., many in SPEC FP), fixed energy costs from structures provided for high-performance make i7's energy 2× to 3× worse than A9.

Key Finding 9: Since power and performance are both primarily design choices, energy use is also primarily impacted by design choice. ISA's impact on energy is insignificant.

Step 4: ISA impact on microarchitecture.

Data: Table 7 outlined microarchitecture features.

Finding E5: The energy impact of the ISA is that it requires micro-ops translation and an additional micro-ops cache. Further, since the number of micro-ops is not significantly higher, the energy impact of x86 support is small.

Finding E6: Other power-hungry structures like a large L2-cache, highly associative TLB, aggressive prefetcher, and large branch predictor seem dictated primarily by the performance level and application domain targeted by the Atom and i7 processors and are not necessitated by x86 ISA features.

5.3. Trade-off Analysis

Step 1: Power- Performance Trade-offs

Data: Figure 14 shows the geometric mean power-performance trade-off for all benchmarks using technology node scaled power. We generate a cubic curve for the power-performance trade-off curve. Given our small sample set, a core's location on the frontier does not imply that it is optimal.

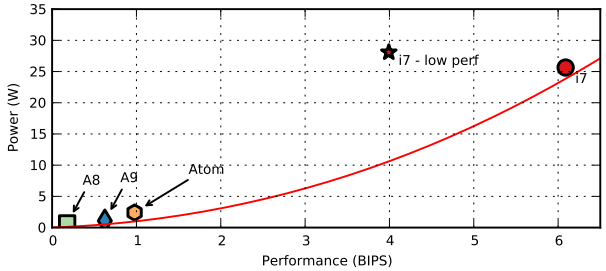


Figure 14. Power Performance Trade-offs.

Finding T1: A9 provides 3.5× better performance using 1.8× the power of A8.

Finding T2: i7 provides 6.2× better performance using 10.9× the power of Atom.

Finding T3: i7's microarchitecture has high energy cost when performance is low: benchmarks with the smallest performance gap between i7 and A9 (star in Figure 14) [11] have only 6× better performance than A9 but use more than 10× more power.

Key Finding 10: Regardless of ISA or energy-efficiency, high-performance processors require more power than lower-performance processors. They follow well established cubic power/performance trade-offs.

Step 2: Energy-Performance Trade-offs

Data: Figure 15 shows the geometric mean energy-performance trade-off using technology node scaled energy. We generate a quadratic energy-performance trade-off curve. Again, a core's location on the frontier does not imply optimality. Synthetic processor points beyond the four processors studied are shown using hollow points; we consider a performance targeted ARM core (A15) and frequency scaled A9, Atom, and i7 cores. A15 BIPS are from reported CoreMark scores; details on synthetic points are in Appendix III.

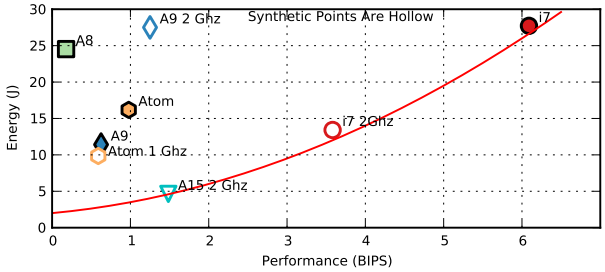


Figure 15. Energy Performance Trade-offs.

Finding T4: Regardless of ISA, power-only or performance-only optimized cores have high energy overheads (e.g., A8 and i7).

Finding T5: Balancing power and performance leads to energy-efficient cores, regardless of the ISA: A9 and Atom processor energy requirements are within 24% of each other and use up to 50% less energy than other cores.

[11] Seven SPEC, all mobile, and the non-database server benchmarks.

Finding T6: DVFS and microarchitectural techniques can provide high energy-efficiency to performance-optimized cores, regardless of the ISA: i7 at 2 GHz provides 6× performance at the same energy level as an A9.

Finding T7: We consider the energy-delay metric (ED) to capture both performance and power. Cores designed balancing power and performance constraints show the best energy-delay product: A15 is 46% lower than any other design we considered.

Finding T8: When weighting the importance of performance only slightly more than power, high-performance cores seem best suited. Considering $ED^{1.4}$, i7–a performance optimized core–is best (lowest product, and 6× higher performance). Considering ED^2, i7 is more than 2× better than the next best design. See Appendix IV for more details.

Key Finding 11: It is the microarchitecture and design methodologies that really matter.

6. Challenges

During this study, we encountered infrastructure and system challenges, missteps, and software/hardware bugs. Table 9 outlines these issues as a potentially useful guide for similar studies, and we describe them in more detail below.

Board cooling: The A8 and A9 boards lack active cooling, and repeatedly rebooted due to over-heating while under test. A fan-based laptop cooling pad fixed the problem.

Network over USB: The `ssh` connection to the A8 and A9 boards used up to 20% of the CPU, which was unsuitable for performance benchmarking. We instead used a serial terminal to access these boards. The Atom board does not support USB networking; we used the Atom as a stand-alone terminal.

Microprocessor PMU infrastructure: The performance counters on the ARM processor are poorly supported on community-supported boards. We backported over 150 TI patches to the Linux kernel to support performance counters and PMU interrupts.

Compilation: `gcc` works remarkably well as a cross-platform compiler for simple benchmarks like SPEC which relies on `libc`. However, for the ARM environment, the compiler often fails when compiling complex code bases that have not been rigorously tested on Linux, due to dependences on over 100 packages. Overcoming these linking errors is a tremendously tedious process. We either carefully choose equivalent highly portable workloads (e.g., `lighttpd`) or worked through the errors (e.g., CLucene and WebKit).

Tracing and debugging: ARM open-source tracing infrastructure is limited, and lacks dynamic binary translation tools like Pin or DynamoRIO. `ptrace` based approaches were too slow; QEMU correctly emulated, but its JIT obfuscated the instruction stream. We used gem5 for ARM traces; gem5 does not support all benchmarks (e.g., `lighttpd`).

Table 9. Summary of Challenges.

Challenge	Description
Board Cooling (ARM)	No active cooling leading to failures **Fix:** use a fan-based laptop cooling pad
Networking (ARM)	`ssh` connection used up to 20% of CPU **Fix:** use a serial terminal
Networking (Atom)	USB networking not supported **Fix:** use as standalone terminal
Perf Counters (ARM)	PMU poorly supported on selected boards **Fix:** backport over 150 TI patches
Compilation (ARM)	Failures due to dependences on >100 packages **Fix 1:** pick portable equivalent (`lighttpd`) **Fix 2:** work through errors (`CLucene` & `WebKit`)
Tracing (ARM)	No dynamic binary emulation **Fix:** Use gem5 to generate instruction traces

Table 10. Summary of Findings.

	#	Finding	Support	Representative Data: A8/Atom
Performance	1	Large performance gaps exist	Fig-2	2× to 997×
	2	Cycle-count gaps are less than 2.5× (A8 to Atom, A9 to i7)	Fig-3	$\leq$ 2.5×
	3	x86 CPI $<$ ARM CPI: x86 ISA overheads hidden by μarch	Fig-3 & 4	A8: 3.4 Atom: 2.2
	4	ISA performance effects indistinguishable between x86 and ARM	Table-6 Fig-5 & 6	inst. mix same short x86 insts
	5	μarchitecture, not the ISA, responsible for performance differences	Table-8	324× Br MPKI 4× L2-misses
	6	Beyond micro-op translation, x86 ISA introduces no overheads over ARM ISA	Table-7	
Power	1	x86 *implementations* draw more power than ARM *implementations*	Fig-11	Atom/A8 raw power: 3×
	2	Choice of power or perf. optimization impacts power use more than ISA	Fig-12	Atom/A8 power @1 GHz: 0.6×
	3	Energy use primarily a design choice; ISA's impact insignificant	Fig-13	Atom/A8 raw energy: 0.8×
Trade-offs	1	High-perf processors require more power than lower-performance processors	Fig-14	A8/A9: 1.8× i7/Atom: 10.9×
	2	It is the μ-architecture and design methodology that really matters	Fig-15	ED: i7@2GHz$<$A9 A15 best for ED i7 best for $ED^{1.4}$

7. Conclusions

In this work, we revisit the RISC vs. CISC debate considering contemporary ARM and x86 processors running modern workloads to understand the role of ISA on performance, power, and energy.Our study suggests that whether the ISA is RISC or CISC is irrelevant, as summarized in Table 10, which includes a key representative quantitative measure for each analysis step. We reflect on whether there are certain metrics for which RISC or CISC matters, and place our findings in the context of past ISA evolution and future ISA and microarchitecture evolution.

Considering area normalized to the 45nm technology node, we observe that A8's area is 4.3mm^2, AMD's Bobcat's area is 5.8mm^2, A9's area is 8.5 mm^2, and Intel's Atom is 9.7 mm^2 [4, 25, 27]. The smallest, the A8, is smaller than Bobcat by 25%. We feel much of this is explained by simpler core design (in-order vs OOO), and smaller caches, predictors, and TLBs. We also observe that the A9's area is in-between Bobcat

and Atom and is close to Atom's. Further detailed analysis is required to determine how much the ISA and the microarchitecture structures for performance contribute to these differences.

A related issue is the performance level for which our results hold. Considering very low performance processors, like the RISC ATmega324PA microcontroller with operating frequencies from 1 to 20 MHz and power consumption between 2 and 50mW [3], the overheads of a CISC ISA (specifically the complete x86 ISA) are clearly untenable. In similar domains, even ARM's full ISA is too rich; the Cortex-M0, meant for low power embedded markets, includes only a 56 instruction subset of Thumb-2. Our study suggests that at performance levels in the range of A8 and higher, RISC/CISC is irrelevant for performance, power, and energy. Determining the lowest performance level at which the RISC/CISC ISA effects are irrelevant for all metrics is interesting future work.

While our study shows that RISC and CISC ISA traits are irrelevant to power and performance characteristics of modern cores, ISAs continue to evolve to better support exposing workload-specific semantic information to the execution substrate. On x86, such changes include the transition to Intel64 (larger word sizes, optimized calling conventions and shared code support), wider vector extensions like AVX, integer crypto and security extensions (NX), hardware virtualization extensions and, more recently, architectural support for transactions in the form of HLE. Similarly, the ARM ISA has introduced shorter fixed length instructions for low power targets (Thumb), vector extensions (NEON), DSP and bytecode execution extensions (Jazelle DBX), Trustzone security, and hardware virtualization support. Thus, while ISA evolution has been continuous, it has focused on enabling specialization and has been largely agnostic of RISC or CISC. Other examples from recent research include extensions to allow the hardware to balance accuracy and reliability with energy efficiency [15, 13] and extensions to use specialized hardware for energy efficiency [18].

It appears decades of hardware and compiler research has enabled efficient handling of both RISC and CISC ISAs and both are equally positioned for the coming years of energy-constrained innovation.

Acknowledgments

We thank the anonymous reviewers, the Vertical group, and the PARSA group for comments. Thanks to Doug Burger, Mark Hill, Guri Sohi, David Wood, Mike Swift, Greg Wright, Jichuan Chang, and Brad Beckmann for comments on the paper and thought-provoking discussions on ISA impact. Thanks for various comments on the paper and valuable input on ISA evolution and area/cost overheads of implementing CISC ISAs provided by David Patterson. Support for this research was provided by NSF grants CCF-0845751, CCF-0917238, and CNS-0917213, and the Cisco Systems Distinguished Graduate Fellowship.

References

[1] ARM on Ubuntu 12.04 LTS battling Intel x86? `http://www.phoronix.com/scan.php?page=article&item=ubuntu_1204_armfeb&num=1`.

[2] The ARM vs x86 wars have begun: In-depth power analysis of Atom, Krait & Cortex A15 `http://www.anandtech.com/show/6536/arm-vs-x86-the-real-showdown/`.

[3] Atmel Datasheet, `http://www.atmel.com/Images/doc2503.pdf`.

[4] chip-architect, `http://www.chip-architect.com/news/AMD_Ontario_Bobcat_vs_Intel_Pineview_Atom.jpg`.

[5] M. Baron. The single-chip cloud computer. *Microprocessor Report*, April 2010.

[6] D. Bhandarkar. RISC versus CISC: a tale of two chips. *SIGARCH Comp. Arch. News*, 25(1):1–12, Mar. 1997.

[7] D. Bhandarkar and D. W. Clark. Performance from architecture: comparing a RISC and a CISC with similar hardware organization. In *ASPLOS '91*.

[8] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. Hill, and D. Wood. The gem5 simulator. *SIGARCH Comp. Arch. News*, 39(2), Aug. 2011.

[9] W. L. Bircher and L. K. John. Analysis of dynamic power management on multi-core processors. In *ICS '08*.

[10] E. Blem, J. Menon, and K. Sankaralingam. Data to accompany a detailed analysis of contemporary arm and x86 architectures, `www.cs.wisc.edu/vertical/isa-power-struggles`, 2013.

[11] D. Bruening, T. Garnett, and S. Amarasinghe. An infrastructure for adaptive dynamic optimization. In *CGO '03*.

[12] R. Colwell, C. Y. Hitchcock, III, E. Jensen, H. Brinkley Sprunt, and C. Kollar. Instruction sets and beyond: Computers, complexity, and controversy. *Computer*, 18(9):8–19, Sept. 1985.

[13] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *ISCA '10*.

[14] H. Esmaeilzadeh, T. Cao, Y. Xi, S. Blackburn, and K. McKinley. Looking back on the language and hardware revolutions: measured power, performance, and scaling. In *ASPLOS '11*.

[15] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *ASPLOS '12*.

[16] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ASPLOS '12*.

[17] M. J. Flynn, C. L. Mitchell, and J. M. Mulder. And now a case for more complex instruction sets. *Computer*, 20(9), 1987.

[18] V. Govindaraju, C.-H. Ho, and K. Sankaralingam. Dynamically specialized datapaths for energy efficient computing. In *HPCA '11*.

[19] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. Full-system analysis and characterization of interactive smartphone applications. In *IISWC '11*.

[20] K. Hoste and L. Eeckhout. Microarchitecture-independent workload characterization. *Micro, IEEE*, 27(3):63 –72, 2007.

[21] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO '03*.

[22] C. Isen, L. John, and E. John. A tale of two processors: Revisiting the RISC-CISC debate. In *2009 SPEC Benchmark Workshop*.

[23] C. Kim, T. Kaldewey, V. W. Lee, E. Sedlar, A. D. Nguyen, N. Satish, J. Chhugani, A. Di Blas, and P. Dubey. Sort vs. Hash revisited: fast join implementation on modern multi-core CPUs. *VLDB '09*.

[24] D. A. Patterson and D. R. Ditzel. The case for the reduced instruction set computer. *SIGARCH Comp. Arch. News*, 8(6), 1980.

[25] G. Quirk. Improved ARM core, other changes in TI mobile app processor, `http://www.cs.virginia.edu/~skadron/cs8535_s11/arm_cortex.pdf`.

[26] J. Rao and K. A. Ross. Making B+- trees cache conscious in main memory. In *SIGMOD '00*.

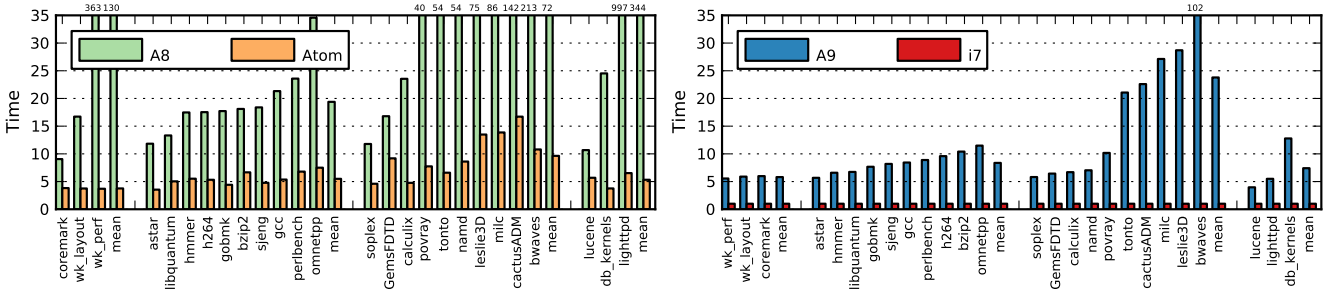
[27] W. Wang and T. Dey. `http://www.cs.virginia.edu/~skadron/cs8535_s11/ARM_Cortex.pdf`.

Appendices

All raw data mentioned below can be found in the Excel file attached to this document (right click on link to save).

Appendix I: Detailed Counts

Data: Figure 16 shows execution time normalized to i7. The raw data for this figure is in the `Time` worksheet of our publicly released spreadsheet [10].

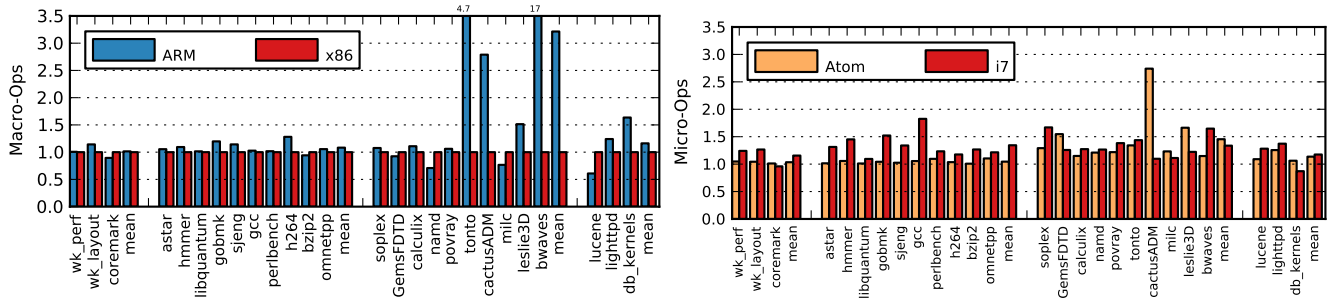


(a) In-Order (b) Out-of-Order

Figure 16. Execution Time Normalized to i7.

Data: Figure 17a shows dynamic instruction (macro) counts on A8 and Atom normalized to Atom x86 macro-instructions. The raw data for this figure is in the `Macro-ops` worksheet of our publicly released spreadsheet [10].

Data: Figure 17b shows dynamic micro-op counts for Atom and i7 normalized to Atom macro-instructions[12]. The raw data for this figure is in the `Micro-ops` worksheet of our publicly released spreadsheet [10]. Note: for `icc` results, preprocessor directives changed so that `gcc` and `icc` use the same alignment and compiler hints.



(a) In-Order (b) Out-of-Order

Figure 17. Instruction Counts Normalized to i7 macro-ops.

Data: Table 11 shows CPIs. This data is also in the `CPI` worksheet of our publicly released spreadsheet [10].

Table 11. Cycles per Instruction (CPI) Per Benchmark.

	Mobile			SPEC INT										SPEC FP										Server		
	coremark	wk - layout	wk - perf	astar	libquantum	hmmer	h264	gobmk	bzip2	sjeng	gcc	perlbench	omnetpp	soplex	GemsFDTD	calculix	povray	tonto	namd	leslie3D	milc	cactusADM	bwaves	lucene	db_kernel	lighttpd
ARM	0.02	1.3	1.3	0.6	0.6	0.9	1.7	2.1	0.5	0.6	3.9	1.9	2.0	1.4	1.3	2.7	2.0	4.8	0.9	0.9	0.7	1.7	0.8	0.4	0.47	0.1
x86	0.02	1.4	1.4	0.7	0.7	0.9	1.5	2.1	0.7	0.7	4.1	1.7	1.7	1.5	1.3	2.6	1.8	5.2	0.9	0.9	0.7	1.5	0.8	1.0	0.6	0.2

[12]For i7, we use issued micro-ops instead of retired micro-ops; we found that on average, this does not impact the micro-op/macro-op ratio.

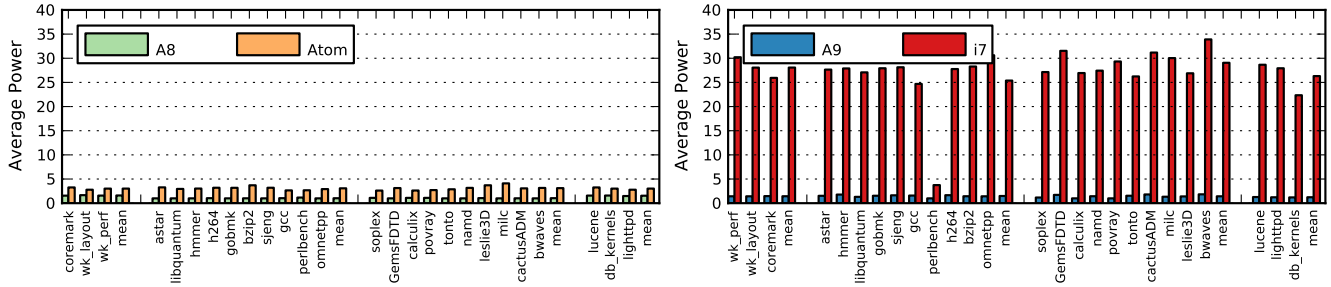
Data: Table 12a shows average ARM and x86 static binary sizes, measuring only the binary's instruction segment. Note that `bwaves` fails to run due incorrect code produced by `ifort`. This data is also in the `Code Size` worksheet of our publicly released spreadsheet [10].

Data: Table 12b shows average dynamic ARM and x86 instruction lengths. This data is also in the `Inst Length` worksheet of our publicly released spreadsheet [10].

Table 12. Instruction Size Details: (a) Static Binary (MB), (b) Average Dynamic Instruction (B).

		Mobile			SPEC INT										SPEC FP										Server		
		coremark	wk - layout	wk - perf	astar	libquantum	hmmer	h264	gobmk	bzip2	sjeng	gcc	perlbench	omnetpp	soplex	GemsFDTD	calculix	povray	tonto	namd	leslie3D	milc	cactusADM	bwaves	lucene	db_kernel	lighttpd
Bin	ARM	0.02	1.3	1.3	0.6	0.6	0.9	1.7	2.1	0.5	0.6	3.9	1.9	2.0	1.4	1.3	2.7	2.0	4.8	0.9	0.9	0.7	1.7	0.8	0.4	0.47	0.1
	x86 - `gcc`	0.02	1.4	1.4	0.7	0.7	0.9	1.5	2.1	0.7	0.7	4.1	1.7	1.7	1.5	1.3	2.6	1.8	5.2	0.9	0.9	0.7	1.5	0.8	1.0	0.6	0.2
	x86 - `icc`	0.6	1.5	1.5	0.7	0.7	1.0	1.3	2.2	0.7	0.8	4.3	1.9	2.2	1.5	1.7	3.1	2.2	6.8	1.0	1.4	0.8	2.0	—	1.4	1.8	0.2
Inst	ARM	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0	4.0
	x86 - `gcc`	2.4	3.7	3.7	2.9	3.0	3.0	3.5	3.1	3.6	3.5	2.8	2.9	2.7	2.7	3.4	2.9	2.6	3.4	3.3	4.1	2.6	6.4	3.0	3.7	2.6	3.7
	x86 - `icc`	2.5	3.2	3.2	3.2	2.9	3.6	3.3	3.3	3.4	3.6	2.9	3.2	2.8	3.1	3.6	3.3	3.5	4.2	4.9	5.0	4.1	6.1	—	2.7	2.7	2.8

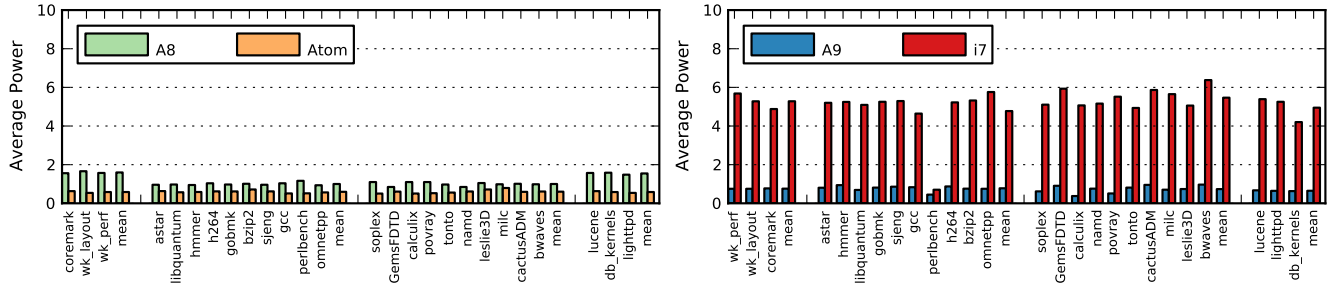
Data: Figure 18 shows average power normalized to the A8. The raw data for this figure is in the `Avg Power` worksheet of our publicly released spreadsheet [10].



(a) In-Order (b) Our-of-Order

Figure 18. Average Power Normalized to A8.

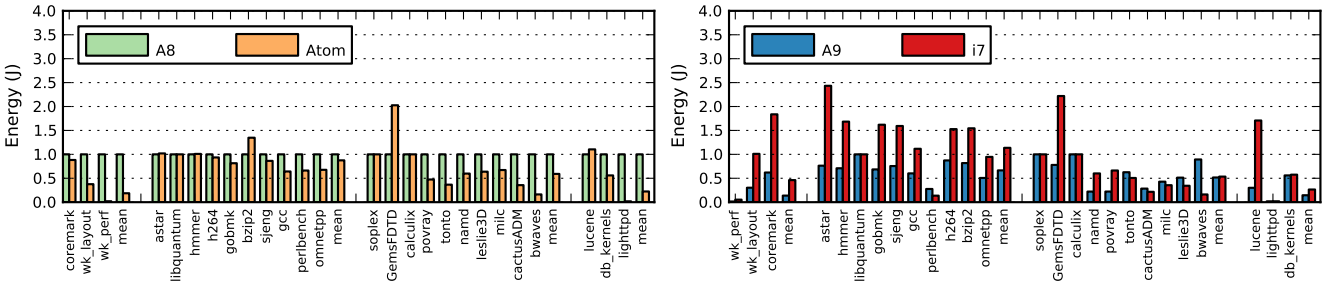
Data: Figure 19 shows technology-independent average power–cores are scaled to 1 GHz at 45 nm (normalized to A8). The raw data for this figure is in the `Avg TI Power` worksheet of our publicly released spreadsheet [10].



(a) In-Order (b) Out-of-Order

Figure 19. Technology Independent Average Power Normalized to A8.

Data: Figure 20 shows raw energy (power and time). The raw data for this figure is in the `Energy` worksheet of our publicly released spreadsheet [10].



(a) In-Order (b) Out-of-Order

Figure 20. Energy Normalized to A8.

Appendix II: Detailed `bwaves` Analysis

As noted in Section 5, the `bwaves` benchmark performed significantly worse (up to 30× more cycles) on ARM cores than on x86 cores. Contributing to this gap, we found that ARM cores executed 17.5× more instructions than x86 cores. We believe that most of the ARM to x86 gap for `bwaves` can be explained by this large differences in the number of instructions required to complete the same amount of work.

We performed detailed analysis to find the source of the instruction count discrepancies. To begin, we found from the execution profile that complex double floating point operations which the compiler translates to longer instruction sequences for ARM than for x86 are a significant source of additional instructions: 37% of all cycles for ARM cores are spent in `__aeabi_dadd` and 29% of all cycles are spent in `__aeabi_dmul`, while neither of these routines appear in the x86 summary).

We use flags to force `gcc` to compile floating point instructions to SSE 2 (x86) and NEON (ARM) instructions. This decision is the most fair in general, since ARM's VFP unit is known to be significantly slower than the NEON unit for single precision floating point operations. However, unlike the VFP unit, the NEON unit is not IEEE754 compliant, and double precision operations are mapped to library calls. The result is that for ARM architectures, `gcc`—in the absence of FP relaxation—compiles double-precision floating point arithmetic to library calls which add significant overhead compared to short instruction sequences on x86. One solution to `bwave`'s outlier status would be to use different compiler flags for benchmarks with significant amounts of double precision arithmetic.

Appendix III: Synthetic Points

A15: Using reported Coremark scores (Coremarks/MHz) and mW/MHz from Microprocessor Reports and ARM documentation, we assume a 2GHz operating frequency and compute the Coremark score and energy. We then scale A9 BIPS results by the ratio of the A15 Coremark score to the A9 Coremark score to get an A15 performance projection.

Frequency scaled cores: For frequency scaled cores, we project performance by assuming a linear relationship between performance and frequency. We scale energy projections as detailed in the Technology scaling and projections paragraph of Section 3.4.

Appendix IV: ED^x Analysis

Data: Figure 21 shows the impact of the exponent x on the product ED^x. Note that a lower product is better. When x is greater than 1.4, i7 outperforms all other cores.

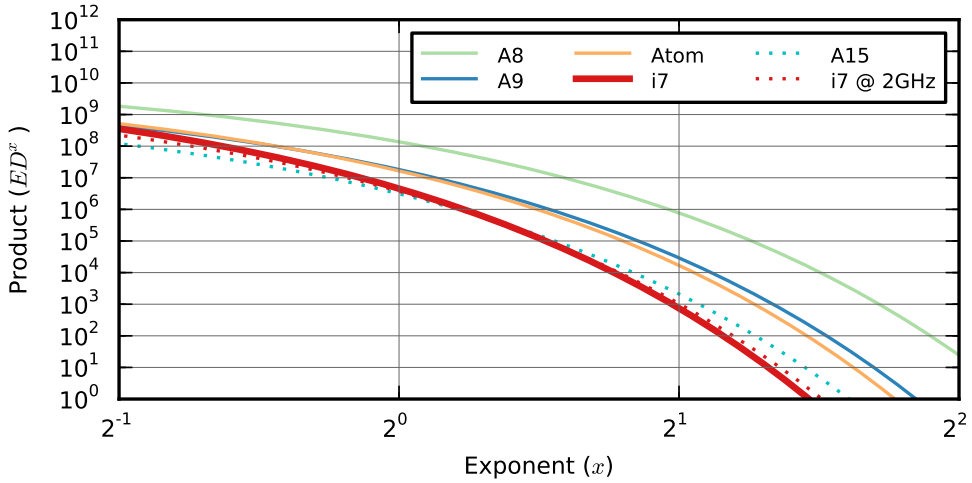


Figure 21. Impact of exponent, x, on product ED^x.