Power and Performance of Native and Java Benchmarks on 130nm to 32nm Process Technologies

Hadi Esmaeilzadeh, Stephen M. Blackburn, Xi Yang, Kathryn S. Mckinley

Power and Performance of Native and Java Benchmarks on 130nm to 32nm Process Technologies

Hadi Esmaeilzadeh
The University of Texas at Austin
hadi@cs.utexas.edu

Stephen M. Blackburn Xi Yang
Australian National University
{Steve.Blackburn,Xi.Yang}@anu.edu.au

Kathryn S. McKinley
The University of Texas at Austin
mckinley@cs.utexas.edu

Abstract

Over the past decade, chip fabrication technology shrank from 130nm to 32nm. This reduction was generally considered to provide performance improvements *together with* chip power reductions. This paper examines how well process technology and microarchitecture delivered on this assumption. This paper evaluates power and performance of native and Java workloads across a selection of IA32 processors from five technology generations (130nm, 90nm, 65nm, 45nm, and 32nm). We use a Hall effect sensor to accurately measure chip power. This paper reports a range findings in three areas. 1) Methodology: TDP is unsurprisingly a poor predictor of application power consumption for a particular processor, but worse, TDP is a poor predictor of relative power consumption between processors. 2) Power-performance trends: Processors appear to have already hit the power wall at 45nm. 3) Native versus Java workloads and their relationship to processor technology: Single threaded Java workloads exploit multiple cores. These results indicate that Java workloads offer different opportunities and challenges compared to native workloads. Our findings challenge prevalent methodologies and offer new insight into how microarchitectures have traded power and performance as process technology shrank.

1. Introduction

This paper explores measured power and performance at a time when hardware and software are undergoing fundamental changes. Over the past seven years, process technology has followed Moore's law and shrunk from 130nm to 32nm, but today, rather than faster cores, more cores are delivering much of the performance dividend. Meanwhile demand for mobility, reliability, and ubiquity have resulted in an explosion of new applications written in *managed* programming languages, which are dynamically compiled, use safe pointer disciplines, and use garbage collection (automatic memory management). We evaluate processor power and performance in the midst of these fundamental changes.

Our contributions fall in three areas: *methodology* for power-performance analysis, *measurement and analysis of power-performance trends* over the past seven years, and *native versus Java workloads* over these seven years of process technologies, thus intersecting language and hardware technology changes.

The prevalent methodology for power-performance analysis has been based on the performance of C benchmarks and the use of power estimates such as TDP (thermal design power). In this work, we use a Hall effect sensor to accurately measure chip power consumption. As our representative of mananged languages, we use Java, which is currently the most popular managed language. Our results show that not only are proxies such as TDP and maximum power very poor estimates of actual power consumption on a given processor, but worse, that they are not predictive of relative power consumption among processors. We find that Java and C benchmarks perform differently in a number of important regards. In particular, we hypothesize that the use of a virtual machine is key to some of these differences. Given the commercial reality of managed languages and the importance of power analysis, our findings suggest that power analysis must consider managed languages and use measured power rather than estimates such as TDP. We also note that the non-determinism introduced by dynamic optimization and garbage collection in managed languages require different performance evaluation methodologies [7].

We perform power-performance analysis across a range of IA32 processors representing five technology generations from 130nm in 2003 to 32nm in 2010. To the best of our knowledge, this study is the first systematic exploration of power and performance across technology generations using measured power. As expected, our results show that performance improvements from generation to generation have decreased. In the transitions to 45nm and 32nm, processors appear to have already hit the power wall. Many performance improvements have come at a significant cost to power. In particular, performance improvements for multi-threaded Java workloads have come at a notable cost to power consumption on both generations of the most recent *Nahelem* microarchitecture family.

We use more than fifty *native* (C, C++ and Fortran) and Java benchmarks drawn from the SPEC CPU 2006, DaCapo, SPEC jvm98 suites, and SPEC jbb2005. We classify these benchmarks into four groups: native integer benchmarks (NINT) (C and C++), native floating point benchmarks (NFP) (C, C++ and Fortran), Java single-threaded benchmarks (JST), and Java multi-threaded benchmarks (JMT). Our results indicate that the past seven years of technology change have played out differently among the four groups. In particular, NFP has benefited most from technology changes, while NINT, JST and JMT have fared less well. Surprisingly, JST benchmarks exploit *more than* one core. We hypothesize that this result is due to parallelism within the managed runtime on which these applications run. Unfortunately, many JMT benchmarks scale very poorly.

Our results show that Java benchmarks, and more broadly programs using managed languages, are worthy of special attention because they behave differently from native benchmarks and they present new opportunities due to the parallelism inherent in the runtime. This finding and the commercial dominance of managed languages from JavaScript to F# and Java suggests that the use of managed languages should be integral to power-performance evaluation methodology. Our results also show that power needs to be measured directly since commonly used estimates such as TDP are both inaccurate and misrepresent relative power between processors. Thus we suggest changes to the prevalent methodology and offer new insight into how process technology and microarchitectural advances have played out in software power and performance

over the past seven years. Understanding the profound changes occurring in both hardware and software, and their interactions, clearly warrants further study.

2. Related Work

TDP (Thermal Design Power) is widely used in the literature as an estimate of chip power consumption. Horowitz et al. study power-performance across different process technologies using TDP to estimate power and SPECmark to estimate performance [11]. They study different aspects of CMOS scaling technology and its impact on power and performance. Hempstead et al. introduce an early stage modeling framework, Navigo, for multi-core architecture exploration across future process technologies from 65nm down to 11nm [10]. Navigo supports voltage and frequency scaling based on ITRS [1] and predictive technology models (PTM) [2]. As the starting point, Navigo uses reported TDP values and SPECmarks as the estimate of power and performance respectively and then predicts the power and performance of processors in the future technology nodes. Chakraborty uses TDP as the power envelope for studying the chip multiprocessor power consumption trends in the future technology nodes [8]. Our results show that TDP is not a good estimate for actual power consumption, suggesting that studies such as these [8, 10, 11] need to be reconsidered in light of measured power rather than TDP.

Li et al. explore the design space of chip multiprocessors under various area and thermal constraints [13]. They combine decoupled trace-driven cache simulation and cycle-accurate execution-driven simulation with detailed single-core simulation to achieve high accuracy in power and performance estimation for chip multiprocessors. While Li et al.'s work uses simulation and is prospective, ours uses direct measures and is retrospective, measuring the power and performance characteristics of existing processors.

Isci and Martonosi introduce a technique for a coordinated measurement approach that combines real total power measurement using a clamp ammeter with performance-counter-based, per unit power estimation [12]. They measure total power for an Intel Pentium 4 on the SPEC CPU2000 benchmark suite.

Bircher and John [5] perform a detailed study of power and performance on AMD quad core Opteron and Phenom processors. They measure power directly using a series resistor, sampling the voltage across the resistor at 1KHz. Our work is complimentary. While they take a very close look at two specific processors, we examine power-performance trends across multiple generations of microarchitecture and process technology.

Fan et al. [9] study the challenge of accurately estimating whole system power in the setting of large scale data centers. They find that when running the most power-consuming workloads, the studied systems drew less than 60% of the nominal 'nameplate' peak power consumption for the system. Similar to our study, they find that actual power is very different to nominal power. However, our concern is chip rather than whole system power, and our objective is in studying power-performance trends across processor generations while theirs is in estimating the power needs of large data centers.

In this paper, first we confirm that thermal design power (TDP) is not a reliable estimate of the processor power consumption. Our power measurements show that TDP, TDP per core, and maximum power are all very poor models of actual measured power consumption. We study power-performance trends for both single-threaded SPEC CPU2006 native benchmarks, and single and multi-threaded Java benchmarks, including SPEC JVM98, SPEC JBB2005, and DaCapo across five process technologies from 130nm to 32nm. As far as we are aware, this paper is the first to quantitatively study power and performance trends across hardware generations using single and multi-threaded, native and managed workloads.

3. Methodology

This section describes the hardware, power measurement methodologies, benchmarks, compiler, Virtual Machines, and performance measurement methodologies using which we report the results.

3.1 Hardware Platforms

To evaluate the power and performance trends across various process technology nodes, we use nine IA32 chips, manufactured using five different process technologies (130nm, 90nm, 65nm, 45nm, and 32nm). Table 1 lists the processors, the number of cores on each processor, their clock speed, last level cache size, TDP, maximum power, sSpec number, release date and release price. TDP is the thermal design power, i.e., the amount of power the chip can dissipate without exceeding the maximum junction temperature, and in fact it does not correspond to the power consumed during the execution of the typical realistic workloads. Manufacturers usually report the same TDP for a family of microarchitectures (see Core 2 Duo *Conroe* (65nm) and Core 2 Duo *Wolfdale* (45nm) in Table 1). However, as shown in Table 1, the maximum amount of power that a chip can draw is higher than the reported TDP. This paper compares TDP and TDP per core to measured power and explores using maximum power, which is equally inaccurate. We report the sSpec number which Intel uses to uniquely identify a chip. We report the release date and nominal release price to provide context regarding Intel's placement of each processor in the market. The two Atom machines and the Core 2 Quad *Kentsfield* (65nm) are outliers at the bottom and top of the market respectively.

3.2 Power Measurements

To measure the power consumption, we use Pololu's ACS714 current sensor board, following the same methodology as Pallipadi and Starikovskiy [14]. The board is a carrier for Allegro's $\pm 5A$ ACS714 Hall effect-based linear current sensor. The sensor accepts a bidirectional current input with a magnitude up to $5A$. The output is an analog voltage ($185mV/A$) centered at $2.5V$ with a typical error of less than 1.5%. The sensor is placed on the $12V$ power line that merely drives the processor chip.

We use Sparkfun's Atmel AVR Stick, which is a simple data-logging device, to send the measured values from the current sensor to the USB port. We use a data-sampling rate of 5Hz, except for the Core i5, which requires a 1Hz data-sampling rate. We execute each benchmark, log all its measured power values, and then compute the average power consumption. For the Java benchmarks, which perform dynamic optimization, we measure the fifth iteration (reflecting steady state), and we take ten such measurements for each benchmark, reporting the arithmetic mean of the ten measurements.

3.3 Benchmarks

We use 52 benchmarks drawn from five suites. The benchmarks are listed in Table 2. We group the benchmarks as follows: C and C++ integer benchmarks, NINT, drawn from SPEC CINT2006; C, C++ and Fortran floating point benchmarks, NFP, drawn from SPEC CFP2006; Java single-threaded benchmarks, JST, drawn from SPEC JVM98, DaCapo-2006-10-MR2, and DaCapo-9.12; and Java multi-threaded benchmarks, JMT, drawn from DaCapo-9.12 and PJBB2005. Table 2 shows the benchmarks, their groupings, the suites the benchmarks were drawn from, the running time for each benchmark on the Atom 230 *Diamondville* (45nm) which we normalize our results to, and a short description. The sections below describe the measurement methodology in detail.

3.3.1 Native Integer and FP Benchmarks (NINT, NFP)

The NINT and NFP groups map directly to the SPEC CINT2006 and SPEC CFP2006 suites respectively [15]. All of these benchmarks

Processor	µArch	Microprocessor	Process nm	# of Cores	Clock GHz	LLC MB	TDP W	Max Power W	sSpec Number	Release Date	Release Price USD
Pentium 4	*NetBurst*	*Northwood*	130	1	2.40	0.5	66.2	—	SL6WF	May '03	—
Pentium M 760	*P6M*	*Dothan*	90	1	2.00	2.0	27.0	38.20	SL7SQ	Jan '05	$423
Core 2 Duo E6600	*Core*	*Conroe*	65	2	2.40	4.0	65.0	105.15	SL9S8	Jul '06	$316
Core 2 Quad Q6600	*Core*	*Kentsfield*	65	4	2.40	8.0	105.0	155.25	SL9UM	Jan '07	$851
Atom 230	*Atom*	*Diamondville*	45	1	1.66	0.5	4.0	6.73	SLB6Z	Jun '08	$29
Atom D510	*Atom*	*Pineview*	45	2	1.66	1.0	13.0	—	SLBLA	Dec '09	$63
Core 2 Duo E7600	*Core*	*Wolfdale*	45	2	3.06	3.0	65.0	98.89	SLGTD	May '09	$133
Core i7 920	*Nehalem*	*Bloomfield*	45	4	2.66	8.0	130.0	230.14	SLBCH	Nov '08	$284
Core i5 670	*Nehalem*	*Clarkdale*	32	2	3.40	4.0	73.0	128.71	SLBLT	Jan '10	$284

Table 1. Experimental Processor Technologies

are *single-threaded* and all are *native* (C, C++, or Fortran) [3]. The NINT benchmarks are intended to represent compute-intensive integer applications that contain sophisticated control flow logic. The NFP benchmarks represent compute-intensive floating-point applications. We complied all of these benchmarks with version 11.1 of the 32-bit Intel compiler suite using the `-o3` optimization flag. We compiled each benchmark once, using the default Intel compiler configuration, without setting any microarchitecture-specific optimizations, and used the resulting binary on all platforms. We exclude 410.bwaves and 481.wrf because they failed to execute when compiled with the Intel compiler. We report the mean of three executions of each benchmark by invoking each benchmark three times in succession.

3.3.2 Java Single-Threaded Benchmarks (JST)

The JST benchmark group includes all single threaded benchmarks from SPEC JVM98, and two releases of DaCapo. SPEC JVM98 is intended to be representative of client-side Java programs. Java programs execute on a JVM (Java Virtual Machine) with a Just-In-Time (JIT) compiler and garbage collector. Although the SPEC JVM98 benchmarks are over ten years old and Blackburn et al. have shown they are simple and have a very small instruction cache and data footprint [6], many researchers still use them. The DaCapo Java benchmarks are a suite of a forward-looking, widely used, nontrivial, and diverse set of client-side applications [6, 16]. The DaCapo suite selects its benchmarks from major open source projects under active development. Researchers have not yet reported extensively on the latest release, but it was designed to expose richer behavior and concurrency on large working sets. We extract the following single-threaded benchmarks from the 2006 DaCapo benchmark release (DaCapo-2006-10-MR2) and from the most current release (DaCapo-9.12): antlr and bloat from DaCapo-2006-10-MR2; and fop and luindex from DaCapo-9.12.

3.3.3 Java Multi-Threaded Benchmarks (JMT)

The JMT benchmark group includes all multi-threaded benchmarks from SPEC JVM98, PJBB2005, and DaCapo-9.12 (DaCapo-2006-10-MR2 does not include any multi-threaded benchmarks that are not in DaCapo-9.12). PJBB2005, is a variant of SPEC JBB2005 [15], that holds the workload, instead of time, constant. We configure PJBB2005 with 8 warehouses and 10,000 transactions per warehouse. We exclude tradesoap from DaCapo because we found it difficult to run successfully on our slowest machines (it makes heavy use of sockets, which time out on slow machines).

3.4 Java Virtual Machines

We use two Java Virtual Machines, Sun (Oracle) HotSpot version 1.7 and Intel JRockit version 1.6, to execute the Java benchmarks. We follow the recommended methodologies for measuring Java [4, 7]. We measure and report the third iteration of each benchmark, which is approximates steady state performance. This execution may still have compiler activity, but has sufficient time to create optimized frequent executed code. We perform this process ten times and report the geometric mean. We require at least ten iterations because the adaptive JIT compilation and garbage collection induce non-determinism. We interleave every execution of each benchmark with the other benchmarks to reduce the probability that one interfering event, such as an operating system demon, will disrupt all the executions of the same benchmark. In contrast to the native workloads, which are compiled a head of time, for the Java benchmarks the compilers may dynamically produce microarchitecture-specific code.

3.5 Operating System and Performance Methodology

We perform all the experiments with Ubuntu 9.10 Karmic with the 2.6.31 Linux kernel. We normalize the execution times for Java benchmarks with HotSpot and all the native benchmarks to the execution time on Atom 230 *Diamondville* (45nm) with HotSpot. The Atom 230 *Diamondville* (45nm) is the lowest performing processor in the set of evaluation platforms. Table 2 lists the average execution time of each benchmark on the Atom 230 *Diamondville* (45nm). We calculate the performance of each benchmark group using the geometric mean.

4. Experimental Results

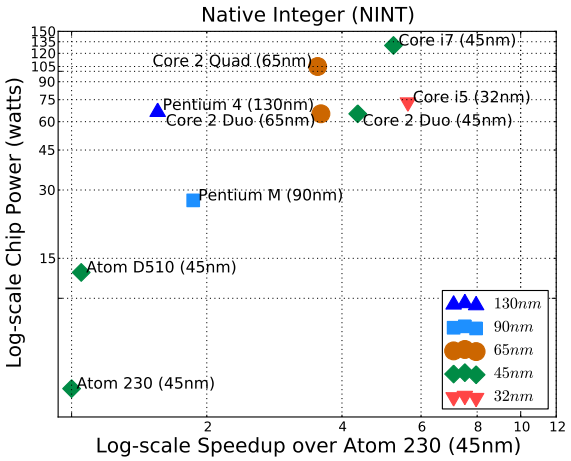
4.1 Power Dissipation Estimation for Native Benchmarks

We start by exploring the power-performance characteristics of the NINT and NFP benchmarks across our range of microarchitectures, using three different measures of power: nominal chip power, nominal chip power per core, and measured power. For nominal chip power, we use the manufacturer's published TDP (thermal design power). For measured power we use the Hall effect sensor described above. We have not been able to measure power on the Atom or Pentium M processors due to their unique power delivery mechanisms.

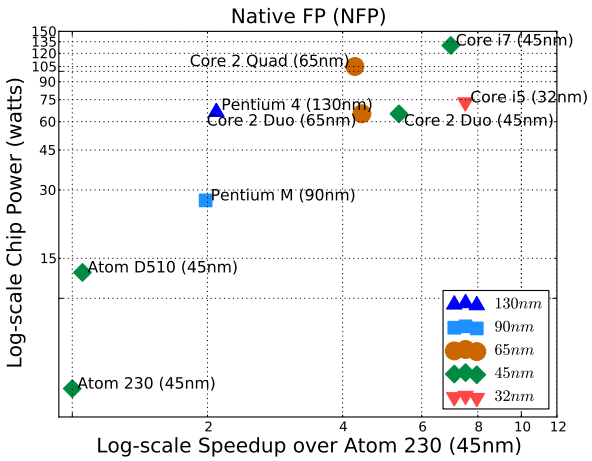
Figure 1 depicts average power and performance for NINT (SPEC CINT2006, left column) and NFP (SPEC CFP2006, right column) with respect to each of the three power metrics: TDP, TDP divided by the number of cores, and measured power. Results are plotted on a log-log scale and performance is normalized to the performance of the single core, low-power, Atom 230 *Diamondville* (45nm) processor, as show in Table 2. Each plotted point reflects the average power and performance for the entire benchmark group on a given processor.

Group	Source	Benchmark	Run Time (secs)	Description
NINT	SPEC CINT2006	400.perlbench	2007	Perl programming language
		401.bzip2	2927	bzip2 Compression
		403.gcc	1530	C optimizing compiler
		429.mcf	1983	Combinatorial optimization/singledepot vehicle scheduling
		445.gobmk	1947	Artificial intelligence (Go game playing)
		456.hmmer	1680	Search a gene sequence database
		458.sjeng	2340	Artificial Intelligence (game tree searchand pattern recognition)
		462.libquantum	1180	Physics / Quantum Computing
		464.h264ref	3017	H.264/AVC video compression
		471.omnetpp	3017	Ethernet network simulation based o then OMNeT++ simulator
		473.astar	2013	Portable 2D path-finding library that is used in game's AI
		483.xalancbmk	1650	XSLT processor for transforming XML documents
NFP	SPEC CFP2006	416.gamess	8950	Quantum chemical computations
		433.milc	1183	Physics/quantum chromodynamics (QCD)
		434.zeusmp	3877	Physics/Magnetohydrodynamics based on ZEUS-MP
		435.gromacs	2013	Molecular dynamics simulation
		436.cactusADM	4620	Cactus and BenchADM physics/general relativity kernels
		437.leslie3d	3567	Large-Eddy Simulations with Linear-Eddy Model in 3D computational fluid dynamics
		444.namd	2680	Parallel program for the simulation of large biomolecular systems
		447.dealII	1700	Partial differential equations with the adaptive finite element method
		450.soplex	1913	Simplex linear program (LP) solver
		453.povray	1280	A ray-tracer
		454.calculix	2600	Finite element code for linear and nonlinear 3D structural applications
		459.GemsFDTD	3273	Solves the Maxwell equations in 3D in the time domain
		465.tonto	3050	Quantum crystallography
		470.lbm	2260	Lattice Boltzmann Method (LBM) to simulate incompressible fluids
		482.sphinx3	4303	Speech recognition
JST	SPEC JVM98	_201_compress	9.9	Compress/uncompress large files based on Lempel-Ziv method
		_202_jess	3.1	Java expert system shell
		_209_db	9.6	Small data management program
		_213_javac	6.5	The JDK 1.0.2 Java compiler compiling 225000 lines of code
		_222_mpegaudio	6.7	MPEG-3 audio stream decoder
		_228_jack	5.0	Parser generator with lexical analysis (early version of JavaCC)
	DaCapo-2006-10-MR2	antlr	8.2	A parser generator and translator generator
		bloat	14.7	A Bytecode-level optimization and analysis tool for Java
	DaCapo-9.12	fop	4.8	An output-independent print formatter
		luindex	4.3	A text indexing tool
JMT	SPEC JVM98	_227_mtrt	1.6	Dual-threaded raytracer
	DaCapo-9.12	avrora	20.2	Simulates the AVR microcontroller
		batik	8.5	A Scalable Vector Graphics (SVG) toolkit
		eclipse	101.6	An integrated development environment
		h2	26.9	An SQL relational database engine written in Java
		jython	27.8	A python interpreter written in Java
		lusearch	15.9	A text search tool
		pmd	14.7	A source code analyzer for Java
		sunflow	37.2	A photo-realistic rendering system
		tomcat	17.6	Tomcat servlet container
		tradebeans	34.6	Tradebeans Daytrader benchmark
		xalan	15.1	An XSLT processor for transforming XML documents
	PJBB2005	pjbb2005	14.8	A fixed workload version of SPEC JBB2005, which is a server application consisting of a wholesale company with warehouses that serve different districts

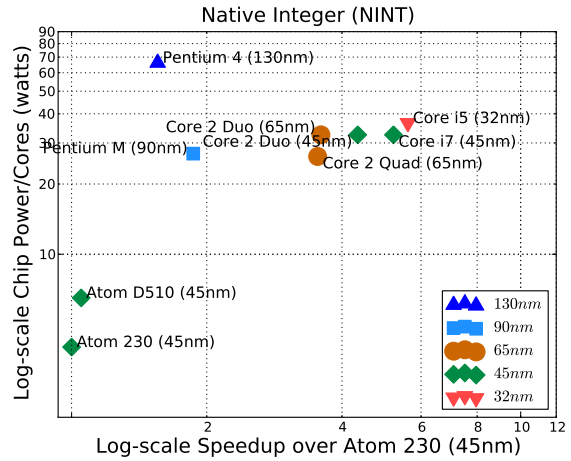
Table 2. Benchmark group composition, run times on the baseline architecture, Atom 230 *Diamondville* (45nm), and short description.



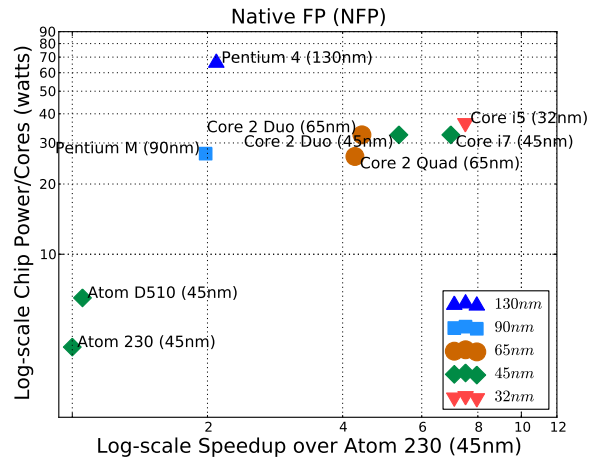
(a) TDP vs Performance



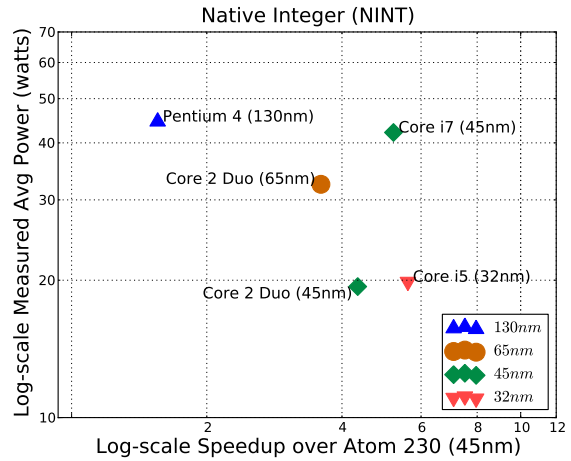
(b) TDP vs Performance



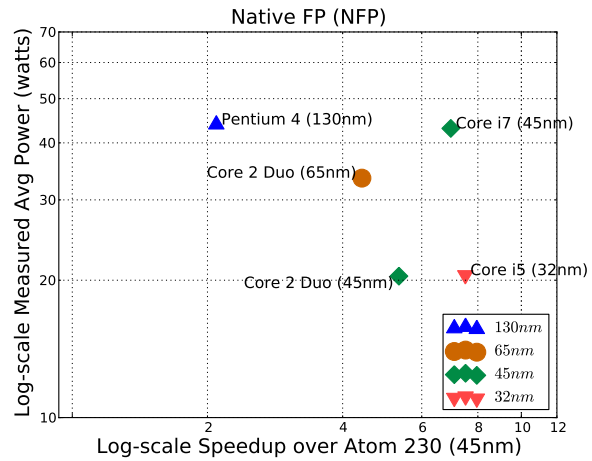
(c) TDP / Core vs Performance



(d) TDP / Core vs Performance



(e) Measured Power vs Performance



(f) Measured Power vs Performance

Figure 1. Power versus performance across different process technologies and microarchitectures for SPEC CINT2006 and SPEC CFP2006 using TDP and measurement. (a), (b) TDP: total chip power. (c), (d) TDP/cores: chip power/number of cores. (e), (f) Measured average power.

In Figure 1(a) and (b), TDP is used in the power-performance plot. However, since the NINT and NFP benchmarks are single-threaded, the extra cores are not being utilized during the execution. For example, the Core 2 Quad *Kentsfield* (65nm) and the Core 2 Duo *Conroe* (65nm) quad and dual core variants of the same microarchitecture deliver the same performance, however, this approach estimates that the Core 2 Quad *Kentsfield* (65nm) may consume 62% more power. Similarly, Figure 1(a)-(b) suggests the Core i7 *Bloomfield* (45nm) may consume 41% more power while delivering less performance on both benchmark suites than its dual-core cousin, the Core i5 *Clarkdale* (32nm) processor .

The NINT and NFP benchmarks are all single-threaded, so it is tempting to divide total chip power by the number of cores when assessing the power-performance characteristics of a given benchmark, since only one core is utilized. Thus Figure 1(c) and (d) show the total chip power (TDP) divided by the number of cores versus speedup over the Atom 230 *Diamondville* (45nm) processor for different architectures. Dividing the TDP by the number of cores is the first-order approximation of the single core power dissipation in a chip multiprocessor architecture. However, it ignores the fact that the 'uncore' (shared) on-chip components (such as last level cache) account for a significant fraction of the chip power and are uncontended when the other cores are inactive and thus may be fully utilized by the single core. The assumption also ignores the reality that background tasks may utilize the otherwise unused cores. Nonetheless, this approach to power analysis is arguably more useful than whole chip power for single thread benchmarks, so it is used. If we use this model of power, the results from Figure 1(c)-(d) invert the findings above, suggesting that Core i5 *Clarkdale* (32nm) may consume *more* power than Core i7 *Bloomfield* (45nm) and that similarly, Core 2 Duo *Conroe* (65nm) may consume *more* power than Core 2 Quad *Kentsfield* (65nm).

Figures 1(e) and (f) show the real measured power. This data paints a very different power-performance picture with respect to each processor and with respect to trends across processor generations. For example, the Core i5 *Clarkdale* (32nm) consumes around 20W on average, about half that predicted by TDP/core and around one quarter that predicted by TDP. Worse, the *relative* performance of the Core i5 *Clarkdale* (32nm) when compared to Core i7 *Bloomfield* (45nm) and Pentium 4 *Northwood* (130nm) is entirely different from that predicted by Figures 1(a)–(d). Also, Figures 1(e) and (f) reveal that Core 2 Duo *Wolfdale* (45nm) has *substantially* lower power consumption than Core 2 Duo *Conroe* (65nm) whereas they share a TDP of 65W.

Our study confirms that TDP (chip total power) and TDP per core consumption are very poor metrics for use in studies of power dissipation trends across different process technology nodes.

Figures 1(e) and (f) reveal some interesting changes in actual power and performance across generations. For NINT (NFP), the Core 2 Duo *Conroe* (65nm) consumes 37% (34%) less power than the Pentium 4 *Northwood* (130nm), while improving performance by a factor of 2.35 (2.11). Both architectures are operating at 2.40GHz. Furthermore, the Core 2 Duo *Wolfdale* (45nm), a 45nm variant of the Core 2 Duo *Conroe* (65nm) with a smaller LLC operating at 3.06GHz, consumes 40% (39%) less power compared to Core 2 Duo *Conroe* (65nm), while improving performance 30% (21%). Similarly, when the Core i7 *Bloomfield* (45nm) is die shrunk from 45nm to 32nm, which yields the Core i5 *Clarkdale* (32nm), performance improves as the clock increases from 2.66GHz to 3.40GHz, while total power consumption drops significantly. The reduction in power from the Core i7 *Bloomfield* (45nm) to the Core i5 *Clarkdale* (32nm) is not due to the technology shrink alone, but includes different power management strategies and significantly smaller uncore power demands. For example, instead of the Core i7 *Bloomfield* (45nm)'s four cores and 8MB LLC, the Core i5 *Clarkdale* (32nm) has two cores and 4MB LLC. The very large last level cache is unlikely to help NINT or NFP benchmarks since they generally place modest demands on memory, especially compared to the Java benchmarks, which we explore next.

Finally, looking more broadly at the *performance* trends across all architectures, NFP speeds up more than NINT relative to the Atom 230 *Diamondville* (45nm). This result is likely due to the greater instruction level parallelism available in the floating point benchmarks, which the out-of-order processors exploit to perform faster and deliver higher speedup compared, for example, to the Atom 230 *Diamondville* (45nm) in-order processor.

4.2 Contrasting Native and Java Single-Threaded Benchmarks

This section explores how the choice of a managed language affects the power-performance trade off. It starts by comparing single-threaded Java benchmarks (JST) to NINT and NFP, since they are strictly single threaded, and then examines multi-threaded Java benchmarks (JMT).

The left column of Figure 2 depicts speedup for each architecture with respect to: (a) native benchmarks (SPEC CPU2006, with both NINT and NFP together); (c) Java single-threaded benchmarks (JST); and (e) Java multi-threaded benchmarks (JMT). We use box-and-whisker plots to represent the range of speedups among the benchmarks constituting each group. The bottom and top of each box represent the lower and upper quartiles, respectively, and the band near the middle of the box is the 50th percentile (the median). The whiskers represent the lowest datum that is still within 1.5 inter quartile range (IQR) of the lower quartile, and the highest datum that is still within 1.5 IQR of the upper quartile. The 'x' indicates the geometric mean for each group.

Figure 2(a) shows native SPEC CPU2006 performance. With the notable exception of the Atom processors, which are designed for low power consumption rather than high performance, speedups fairly consistently rise as technology shrinks. The improvements compared to Pentium 4 *Northwood* (130nm) are the result of microarchitectural innovations, inclusion of more last-level cache, lower-latency memory access, and in the case of Core 2 Duo *Wolfdale* (45nm) and Core i5 *Clarkdale* (32nm), increased clock frequency. The distribution of the speedup for Pentium M *Dothan* (90nm) shows that even though on average it outperforms the Atom 230 *Diamondville* (45nm) by 93%, in some of cases the Atom 230 *Diamondville* (45nm) processor delivers better performance. The fact that Pentium M *Dothan* (90nm) is also optimized for power consumption can explain the slow-down in some cases. The Atom 230 *Diamondville* (45nm) processor has lower latency memory access, which can result in higher performance. As expected, since the benchmarks are single-threaded, additional cores do not improve performance.

Figure 2(c) illustrates a box plot for Java single-threaded benchmarks (JST). Similar to the case of NINT and NFP, the results show that except for the Atom processors, as the technology shrinks, the processors achieve higher performance. However, Core i5 *Clarkdale* (32nm) performs slightly worse than Core i7 *Bloomfield* (45nm). Unlike with SPEC CPU2006, the dual core Atom D510 *Pineview* (45nm) achieves 22% *higher performance* compared to the single-core Atom 230 *Diamondville* (45nm). Both Atom processors operate at 1.66GHz and the major differences are the number of cores and increased cache capacity. This result is intriguing because the benchmarks are single-threaded.

While a single-threaded benchmark itself may not directly exploit parallelism, the underlying JVM often may. This parallelism can come from several sources. The JVM provides rich libraries for the application, and these libraries may be parallelized. The JVM also performs services such as profiling (for feedback directed optimization), compilation, and garbage collection, all of which may

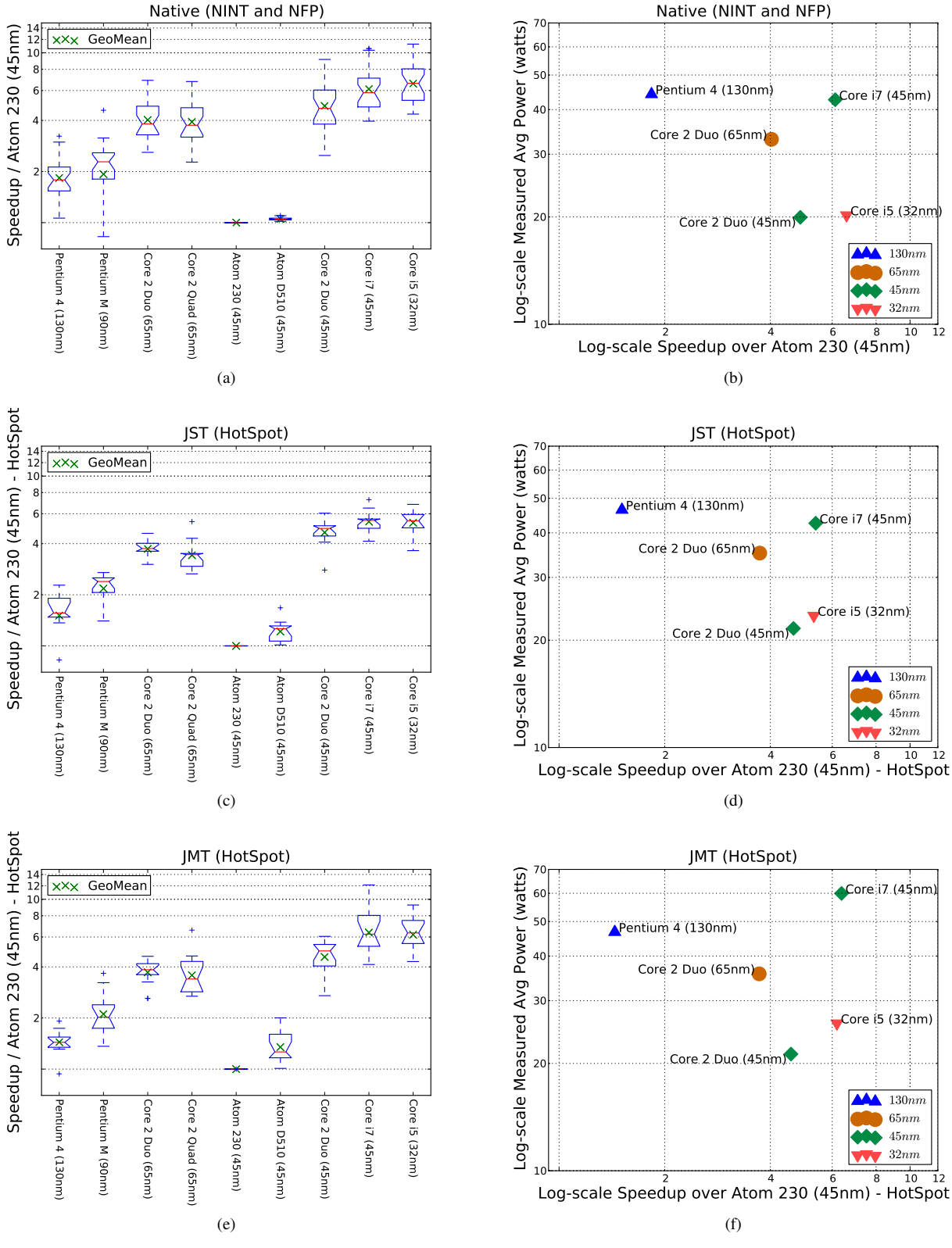


Figure 2. Speedup box plots across different process technologies and microarchitectures for (a) SPEC CPU2006, (b) JST, and (c) JMT. Measured power versus performance for (b) SPEC CPU2006, (d) JST, and (f) JMT.

be parallel or performed concurrently with the application in separate threads. Thus the JVM may leverage the additional cores even when the application itself is explicitly single threaded. This result highlights an important difference between managed and unmanaged languages.

Figure 2(b) presents the measured average power dissipation versus performance of various microarchitectures for SPEC CPU2006. As depicted, from 130nm (Pentium 4 *Northwood* (130nm)) to 65nm (Core 2 Duo *Conroe* (65nm)) process technology (two technology generations), performance improved by a factor of 2.21, while power consumption decreased by 34%. Core 2 Duo *Wolfdale* (45nm) uses the same Core microarchitecture as Core 2 Duo *Conroe* (65nm), shrunk from 65nm to 45nm, with a 3MB LLC rather than 4MB, and operating at 3.06GHz rather than 2.4GHz, which yields a 20% performance improvement and a 66% power reduction. On the other hand, Core i7 *Bloomfield* (45nm) achieves a 52% performance improvement over the Core 2 Duo *Conroe* (65nm), but by *increasing* power dissipation by 29%. These results show that in the transition from 65nm to 45nm the process shrink lead to good power/performance improvements, but in the case of the Core i7 *Bloomfield* (45nm), the balance of that dividend went to performance, rather than power savings. From 45nm to 32nm, Core i5 *Clarkdale* (32nm) improves performance by 35% compared to Core 2 Duo *Wolfdale* (45nm), while increasing power negligibly.

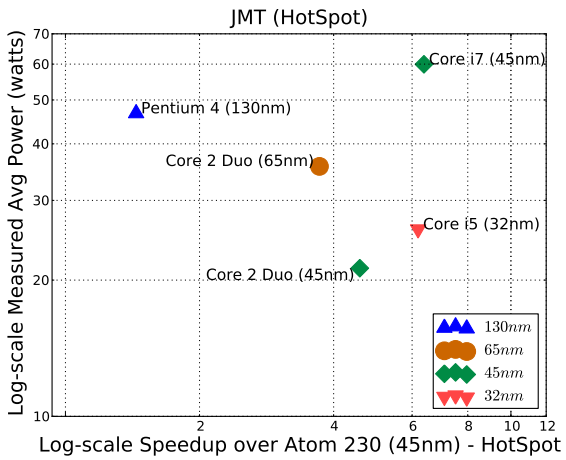
Compared to the Core i7 *Bloomfield* (45nm), the Core i5 *Clarkdale* (32nm) processor improves performance by 8% while halving power consumption. The Core i7 *Bloomfield* (45nm) and the Core i5 *Clarkdale* (32nm) share the *Nehalem* microarchitecture, but the Core i7 *Bloomfield* (45nm) has 4 cores, 8 hardware threads, 8MB L3 and is operating at 2.6GHz, while the Core i5 *Clarkdale* (32nm) has 2 cores, 4 hardware threads, 4MB L3 and is operating at 3.4GHz. The primary performance difference between these two processors is the clock speed. However, it is hard to make direct comparisons between the two since the Core i7 *Bloomfield* (45nm) has twice as many cores, twice the L3, and twice the memory bandwidth compared to the Core i5 *Clarkdale* (32nm). It is therefore unsurprising that the Core i5 *Clarkdale* (32nm) is more power/performance efficient than the Core i7 *Bloomfield* (45nm).

Comparing Figures 2(b), (d), the trends for the Pentium 4 *Northwood* (130nm), Core 2 Duo *Conroe* (65nm), and Core i7 *Bloomfield* (45nm) are very similar across the benchmark groups. It is notable that while the Core i5 *Clarkdale* (32nm) improved over the Core i7 *Bloomfield* (45nm) on SPEC CPU2006, it degraded on JST. We speculate that this is because the larger, more complex workloads present in JST, more specifically DaCapo, are more sensitive to cache size than to clock speed, which are the two principle variables differentiating the Core i7 *Bloomfield* (45nm) and Core i5 *Clarkdale* (32nm) in this single-threaded setting.

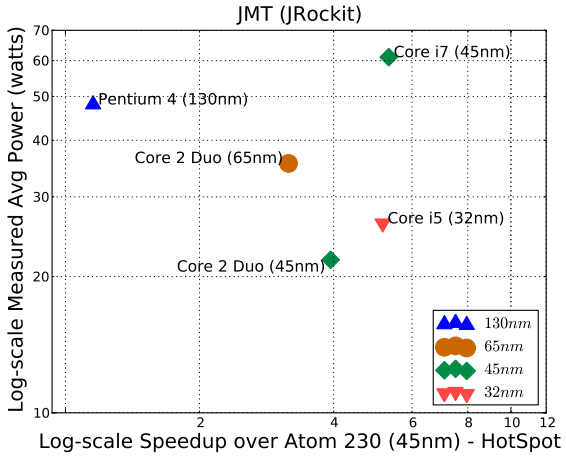
4.3 Java Multi-Threaded Benchmarks

Figure 2(e) shows speedup for JMT with box plots. The performance trends for JMT are almost the same as for JST; however, as expected, the performance improvements are higher when more cores are available. The Atom D510 *Pineview* (45nm) delivers 35% more performance compared to the Atom 230 *Diamondville* (45nm), while the microarchitectures are essentially the same in both cases. The Core i7 *Bloomfield* (45nm) and the Core i5 *Clarkdale* (32nm) deliver almost the same average performance, even though the Core i5 *Clarkdale* (32nm) is operating at a higher frequency. The Core i7 *Bloomfield* (45nm) provides four cores and supports eight hardware threads while the Core i5 *Clarkdale* (32nm) has two core and supports four hardware threads.

Comparing Figures 2(d) and 2(f), the most noticeable difference is the increased average power consumption of the Core i7 *Bloomfield* (45nm), which increases from around 40W to 60W. This result



(a)



(b)

Figure 3. Measured power versus performance for JMT with the HotSpot and JRockit Java virtual machines.

is consistent with the fact that multi-threaded benchmarks making greater use of the cores on chip multiprocessors. Perhaps more interesting is that the other chip multiprocessors do *not* exhibit greater power consumption even when more of their cores are exercised. This may reflect the greater complexity of the multithreaded benchmarks and their tendency to be memory bound, which may leave the extra cores relatively underutilized.

Figure 3 shows the power versus performance trends across different process technologies for the two JVMs on JMT. Figure 3 shows the choice of JVM affects the power-performance behavior of the workloads. In these results, JRockit 1.6.0 JVM delivers lower performance compared to HotSpot 1.7.0 while consuming about the same power. The experiments show that across all the benchmarks, JRockit is slower than HotSpot while consuming slightly less or the same amount of power. For example on Core i7 *Bloomfield* (45nm), HotSpot outperforms JRockit by 19% while dissipating 2% less power. Similarly on Core i5 *Clarkdale* (32nm), HotSpot delivers 20% higher performance and consumes 1% less power. The comparison shows that the choice of runtime has a significant effect on the performance of JMT applications, and needs to be considered

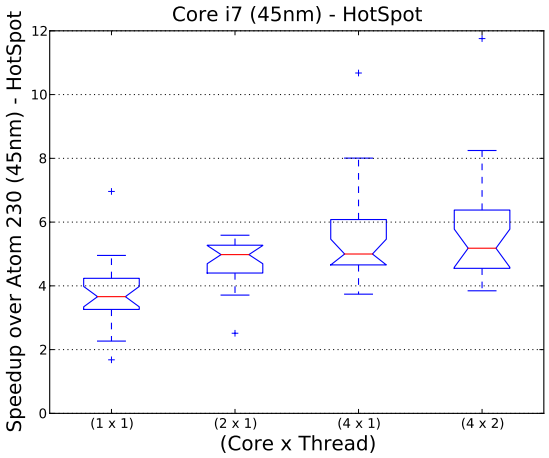
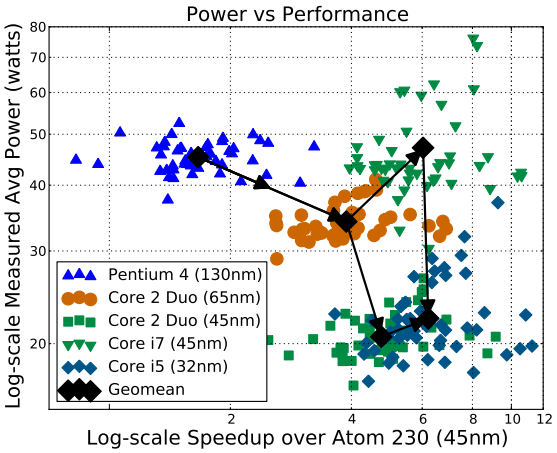
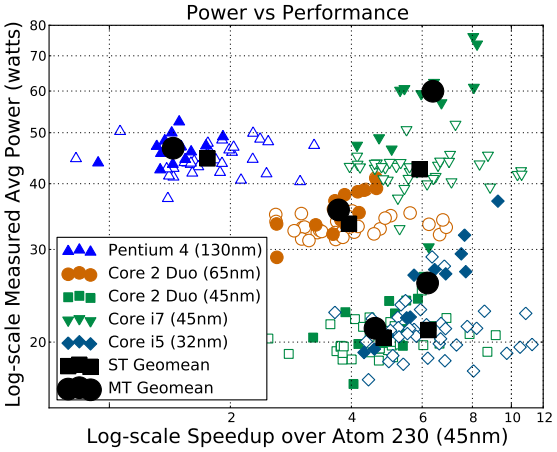


Figure 4. Scalability of all Java benchmarks (JST and JMT) on the Core i7 *Bloomfield* (45nm), with HotSpot.



(a)



(b)

Figure 5. Measured power versus performance scatter plot for different process technologies. Each point denotes one of the 52 benchmarks. (a) The arrows connect the geometric mean of average power for all 52 benchmarks. The bifurcation from 65nm to 45nm shows the two measured 45nm processors (Core 2 Duo *Wolfdale* (45nm) and Core i7 *Bloomfield* (45nm)). (b) Delineates the single-threaded (ST) benchmarks with hollow marks, the multi-threaded (MT) benchmarks with solid marks, and their individual geometric means with a large triangle (ST) and circle (MT), respectively.

as part of the methodology for power-performance evaluation for managed languages.

These figures indicate a surprising lack of scalability on average for the multi-threaded Java benchmarks. It is beyond the scope of this paper to explore the root causes for this result, but we did conduct a basic analysis to understand performance scalability with respect to the number of cores more using the Core i7 *Bloomfield* (45nm), which has four cores and eight hardware contexts.This experiment restricts the available hardware contexts via an operating system API and evaluates the resulting performance changes. Figure 4 uses box plots to reflect the speedup of the Java benchmarks as a function of available hardware contexts. We use all the Java benchmarks, i.e., both the JST and JMT. The hardware resources increase from left to right, starting with a single hardware context and ending with all eight hardware contexts. An interesting finding is that only one Java benchmark, h2, did not speed up as more hardware became available. The h2 benchmark is a transaction-oriented database, which may be limited by lock contention. Many single threaded benchmarks sped up appreciably when a second hardware context became available. The performance of I/O-intensive, single-threaded antlr *doubled* when a second core was available. Conversely, few of the multi-threaded benchmarks scaled with the number of hardware contexts. Only sunflow, tomcat, lusearch and eclipse showed near-linear speedup. We found these results surprising.

4.4 Have Processors Already Hit the Power-Performance Wall?

Figure 5 shows power versus performance scatter plots for all 52 benchmarks on each of the machines where we could measure power directly. Figures 5 (a) and (b) show exactly the same data points, but render them differently to illustrate different findings. Figure 5(a) uses red arrows to indicate the trends across technology nodes. The line bifurcates at 65nm to the divergent Core 2 Duo *Wolfdale* (45nm) (bottom-most) and Core i7 *Bloomfield* (45nm) (upper-right) processors. The arrows show two paths taken at the measured 45nm processors. Compared to the Core 2 Duo *Conroe* (65nm) (center), the Core 2 Duo *Wolfdale* (45nm) provides 22% higher performance with 66% less power. On the other hand, the Core i7 *Bloomfield* (45nm) provides 55% higher performance while consuming 38% more power. From 130nm to 65nm (two generations), we see performance improve by 134%, while from 65nm to 32nm (two generations), we see only 66% performance improvement, just half of what is achieved from 130nm to 65nm.

Figure 5(b) illustrates the affect the technology changes have on multi-threaded (solid symbols) and single-threaded (hollow symbols) benchmarks respectively, using all 52 benchmarks. It is notable that the newer Core i5 *Clarkdale* (32nm) and Core i7 *Bloomfield* (45nm) (*Nehalem*) processors present a similar pattern, with substantial variation in their actual power consumption. The multi-threaded benchmarks sometimes consume twice as much power as their single-threaded counterparts. In contrast, the power consumption of the *Core* (Core 2 Duo *Conroe* (65nm) and Core 2 Duo *Wolfdale* (45nm)) processors are much less sensitive to whether the benchmark is single or multi-threaded. The Pentium 4 *Northwood* (130nm) has only one hardware context, so it is unsurprising that it is largely

unaffected by the number of threads. The *Nehalem* processors also appear to have greater power variability, even among the single-threaded benchmarks, but this result may simply reflect the single-threaded Java benchmarks' capacity to utilize extra cores. It is intriguing that higher power consumption on multi-threaded Java benchmarks does not translate to considerable average performance improvement.

5. Conclusion

As far as we are aware, this paper is the first to quantitatively study measured power and performance across hardware generations using single and multi-threaded, native and managed workloads.

With respect to methodology for power-performance analysis, we show that relative power across hardware generations cannot be accurately estimated from TDP, TDP per core, or maximum power specifications. We also show that Java benchmarks behave differently to native benchmarks with respect to power and performance trends. Given the commercial importance and widespread use of managed workloads, this finding indicates that future studies should include Java or other managed languages.

We find that performance gains from generation to generation have slowed and that in the transition from 45nm to 32nm processors appear to have already hit the power wall. Furthermore, many performance improvements have come at a significant cost to power. In particular, in the most recent processors, multi-threaded benchmarks are notably more power-intensive relative to their single-threaded counterparts. We also find that subsequent hardware generations improve native floating point benchmarks more than integer and Java benchmarks.

We study both native and Java benchmarks and are encouraged to see that the Java runtime exploits multiple cores, even when executing single-threaded workloads. On the other hand, we found that many parallel benchmarks do not scale with the number of hardware contexts. Much further analysis, including cache and bus measurements, are need to explain the root causes of these problems.

In summary, our findings challenge widely used methodology, offer new insight into how microarchitectures have traded power and performance as process technology has shrunk, and point to the need for much additional analysis.

Acknowledgements

We gratefully acknowledge the generous assistance of Bob Edwards of ANU for his work in fabricating and calibrating the current sensors we used throughout this work.

References

[1] International technology roadmap for semiconductors, 2010. http://www.itrs.net.

[2] Predictive technology model, 2010. http://ptm.asu.edu.

[3] SPEC CPU2006 benchmark descriptions. *ACM SIGARCH newsletter, Computer Architecture News*, 34(4), September 2006.

[4] L. E. Andy Georges, Dries Buytaert. Statistically rigorous Java performance evaluation. In *ACM Conference on Object–Oriented Programming Systems, Languages, and Applications*, pages 57–76, 2007.

[5] W. L. Bircher and L. K. John. Analysis of dynamic power management on multi-core processors. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, pages 327–338, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-158-3. doi: http://doi.acm.org/10.1145/1375527.1375575.

[6] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis. In *ACM SIGPLAN Conference on Object-Oriented Programing, Systems, Languages, and Applications*, pages 169–190, Oct. 2006.

[7] S. M. Blackburn, K. S. McKinley, R. Garner, C. Hoffman, A. M. Khan, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. Wake up and smell the coffee: Evaluation methodologies for the 21st century. *Communications of the ACM*, 51(8):83–89, Aug. 2008.

[8] K. Chakraborty. *Over-provisioned Multicore Systems*. PhD thesis, University of Wisconsin-Madison, 2008.

[9] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-706-3. doi: http://doi.acm.org/10.1145/1250662.1250665.

[10] M. Hempstead, G.-Y. Wei, and D. Brooks. Navigo: An early-stage model to study power-contrained architectures and specialization. In *Proceedings of Workshop on Modeling, Benchmarking, and Simulations (MoBS) (Held in conjunction with ISCA-36)*, June 21 2009.

[11] M. Horowitz, E. Alon, D. Patil, S. Naffziger, R. Kumar, and K. Bernstein. Scaling, power, and the future of CMOS. In *Proceedings of International Electron Devices Meeting (IEDM)*, pages 7–15, December 2005.

[12] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *International Symposium on Microarchitecture*, pages 93–104, December 3–5 2003.

[13] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP design space exploration subject to physical contraints. In *Proceedings of International Sympsoium on High-Performance Computer Architecture (HPCA-12)*, pages 17–28, Feb 2006.

[14] V. Pallipadi and A. Starikovskiy. The ondemand governor: past, present and future. In *Proceedings of Linux Symposium*, volume 2, pages 223–238, July 19–22 2006.

[15] Standard Performance Evaluation Corp. SPEC Benchmarks, 2010. http://www.spec.org.

[16] The DaCapo Research Group. The DaCapo Benchmarks, beta-2006-08, 2006. http://www.dacapobench.org.