A Methodology for Modeling Dynamic and Static Power Consumption for Multicore Processors

Bhavishya Goel and Sally A. McKee

Chalmers University of Technology
Gothenburg, Sweden
{goelb, mckee}@chalmers.se

***Abstract*—System designers and application programmers must consider trade-offs between performance and energy. Making energy-aware decisions when designing an application or runtime system requires quantitative information about power consumed by different processor components. We present a methodology to model static and dynamic power consumption of individual cores and the uncore components, and we validate our power model for both sequential and parallel benchmarks at different voltage-frequency pairs on an Intel® Haswell platform.**

Our power models yield the following insights about energy-efficient scaling. (1) We show that uncore energy accounts for up to 74% of total energy. In particular, uncore static energy can be as high as 61% of total energy, potentially making it a major source of energy inefficiency. (2) We find that the frequency at which an application expends the lowest energy depends on how memory-bound it is. (3) We demonstrate that even though using more cores may improve performance, the energy consumed by stalled cores during serial portions of the program can make using fewer cores more energy-efficient.

***Keywords*-Power Modeling; Static and Dynamic Power; Energy Characterization.**

I. Introduction

Modern systems from petascale supercomputers to handheld devices must balance performance and power consumption. This often requires that the system have access to real-time power information. Hypervisors, operating systems, and runtime software can all use such information to execute workloads more efficiently.

Software acquires power usage information either from actual power meters or via estimation models implemented in hardware or software. For instance, Intel Xeon® chips can use Node Manager® [1] to measure power, but this off-die infrastructure only provides information about processors, main memory, and the system as a whole. It cannot report on individual chip components, i.e., cores, functional units, other microarchitectural components, or caches.

Model-based power estimation is a common alternative to actual measurement. For instance, Intel's Running Average Power Limit (RAPL) [2], AMD's Application Power Management (APM) [3], and NVIDIA's Management Library (NVML) [4] all implement energy estimation models in hardware. These interfaces provide composite power values for the entire chip but not for individual compute units. RAPL provides separate aggregate values for the cores versus the uncore, but it does not provide a power breakdown for individual cores. The "uncore" is Intel's term for the CPU components that are outside but closely associated with the cores (e.g., the last-level cache, memory controller, and interconnect),

Implementing techniques like energy-aware scheduling and software-controlled DVFS requires power information at a finer granularity. Given the limited utility of power meters in commodity hardware, many software power estimation models have been proposed. These models strive to provide detailed power estimation at granularities ranging from the entire CPU [5]–[7] to individual cores [8]–[10], individual microarchitectural components [11]–[13], or individual cores plus the uncore [14]. To the best of our knowledge, none of these models makes an explicit distinction between static and dynamic power consumption.

Mair et al. [15] emphasize the importance of distinguishing between static and dynamic power due to the fundamental differences in their impact and their parameters. Both depend on supply voltage, but static power depends on temperature and dynamic power depends on frequency. These power components scale differently from one voltage-frequency step to another, and accurate information on the breakdown between them is critical for understanding energy expenditure trends when applying DVFS.

Another reason to model static and dynamic power separately is to characterize system energy efficiency. Static power consumption is due to leakage current in the transistors, and hence it does not contribute towards performing useful work. The static energy thus represents "wasted energy" in the system. The ratio of static to total system energy represents energy inefficiency of the system and can be useful in making qualitative architectural comparisons.

We present a systematic methodology for deriving models that calculate static and dynamic power consumption for the uncore and the cores. We show how to isolate and quantify power consumption of different processor components. We validate our power models at four different voltage-frequency steps for single-threaded and parallel benchmarks, and show that our models estimate power with good accuracy (mean absolute error of 3.14%) across all benchmarks.

We use our models to characterize the energy efficiency of scaling clock frequency and thread-level parallelism on the Haswell processor. We find that uncore static energy represents a significant portion of total system energy (up to 61%), making it a major source of energy inefficiency. We also show that running an application at lower frequencies does not always expend less energy: the degree to which an application is memory-bound must be considered when choosing energy-efficient DVFS policies. Finally, we demonstrate that the serial fraction of a parallel workload determines the level of concurrency at which the least energy is expended.

II. Methodology

We run experiments on an Intel Core™ i7 4770 (Haswell) processor that has four physical cores with a maximum clock frequency of 3.4 GHz. Each core has two eight-way 32 KB private L1 caches (separate I and D), a 256 KB private L2 cache (combined I and D), and an 8 MB shared L3 cache, with 16 GB of physical memory on board.

Since we want to focus on the CPU portion of the chip, our experiments do not use the on-chip GPU and eDRAM (which are power-gated and cannot affect results). Power models for these components would be interesting and useful, but deriving them is beyond the scope of the work presented here. We disable hyper-threading and Turbo Boost for all experiments. In the rest of our discussion, we use chip and CPU interchangeably.

We measure power consumption at the ATX CPU power rails using infrastructure that we built for previous work [10]. This infrastructure uses current transducers to convert measured current to voltage, which a data acquisition unit then logs at high accuracy with high sampling rates. To minimize interference, we log power measurements on a machine other than the one under test.

We collect performance counter values using the `pfmon` tool provided by the `libpfm4` library. We use the Linux® kernel driver `coretemp` to read package temperatures via the on-die Digital Temperature Sensor (DTS) [16].

Our models break total power into four components:

1) uncore dynamic power,
2) core dynamic power,
3) uncore static power, and
4) core static power.

We strive to develop models for each component in isolation in order to prevent model estimation error of any component from affecting the estimation accuracy of other components.

A. Uncore Dynamic Power Model

We break the uncore dynamic power into uncore idle dynamic power and uncore active dynamic power. Our experiments indicate that the uncore is neither clock-gated nor power-gated when idle. Eq. 1 gives the formula for idle uncore dynamic power consumption.

$$P(uncore)_{idle\_dynamic} = \alpha * C * V^2 * F \quad (1)$$

where α = idle uncore activity factor
C = uncore capacitance
V = uncore supply voltage
F = uncore frequency

When idle, the uncore effective capacitance (αC in Eq. 1) remains virtually constant across frequency changes (we disregard the insignificant uncore activity caused by OS housekeeping threads). Recall that uncore static power depends on uncore voltage and package temperature. We fix uncore voltage from the BIOS and measure idle chip power at the same package temperature at different uncore frequencies. We can thus safely assume that static power remains constant across frequency changes. Any differences in measured power must therefore be due to changes in uncore dynamic power. We measure idle CPU power consumption at frequencies F_1 and F_2 while making sure that the package temperature is same across two readings. Then, the idle CPU power at uncore frequency F_n can be calculated as $P_n = \alpha * C * V^2 * F_n + P_{static}$. We then calculate αC for the idle uncore using Eq. 2.

$$\alpha * C = \frac{P_2 - P_1}{V^2 F_2 - V^2 F_1} \quad (2)$$

Eq. 3 shows the model we generate for uncore idle dynamic power. We verify the value of αC by repeating our measurements for different values of F_1 and F_2.

$$P(uncore)_{idle\_dynamic} = 1.41 * 10^{-9} * V^2 * F \quad (3)$$

where V = uncore supply voltage
F = uncore frequency

We next analyze the effects of L2 cache misses on uncore dynamic power consumption. An L2 miss causes activity on the ring interconnect and in the L3 last-level cache. To measure the effects of L2 misses on CPU power consumption, we create a microbenchmark that interleaves memory accesses with enough integer and floating point operations to hide L3 latency. We measure CPU power consumption as we gradually increase the benchmark's working-set size such that core activity (micro-operations executed per cycle) and L2 activity (L2 requests per cycle) remain constant but the ratio of L2 misses to L2 requests increases. We attribute differences in consumed power to increases in uncore activity from the increased L2 misses. We run linear regression on the L2 miss rate and increase in power consumption to generate a model for the L2-miss contribution to uncore dynamic power. Eq. 4 shows this

model, and Figure 1 shows its accuracy with respect to our measured values. The R^2 coefficient measuring the goodness of the fit for the estimation model is 0.999.

$$P(uncore)_{L2\_miss\_dynamic} = (3.043 * 10^{-9} * x^2 + 2.881 * 10^{-9} * x) * V^2 * F \quad (4)$$

where x = chip-wide L2 misses per uncore cycle
V = uncore supply voltage
F = uncore frequency

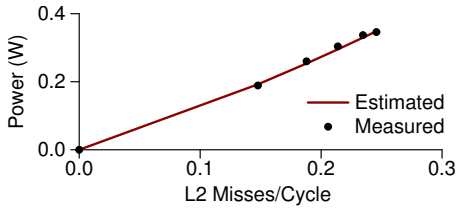


Figure 1: Model Fitness for Power Consumed due to L2 Misses at F=800 MHz and V=0.7V

We perform a similar experiment to measure effects of L3 misses on uncore power. For realistic L3 miss rates ($\leq$ 50 MPKI), CPU power consumption increases negligibly even though DRAM power consumption increases significantly. Our models thus omit L3 misses, and we compute total uncore dynamic power consumption as the sum of uncore idle dynamic power and uncore power due to L2 misses.

B. Core Dynamic Power Model

To model core dynamic power consumption, we study the effects of microarchitectural events like micro-operations (uops) executed, L1 accesses, and L2 accesses. We start by quantifying the effects of non-memory operations on core dynamic power. We would prefer to analyze integer and floating point operations separately, but the Haswell microarchitecture provides no performance counter for floating point operations. We thus average over both instruction types. We create microbenchmarks that loop over the following instructions, alone and in combination:

- x87 floating point multiplications,
- x87 floating point additions,
- integer multiplications,
- integer additions,
- SIMD floating-point multiplications, and
- SIMD floating-point additions.

For each microbenchmark, we calculate αC using the same technique as for calculating idle uncore power: we fix uncore voltage, uncore frequency, and core voltage from the BIOS. We then initiate microbenchmark execution on all cores, measure power consumption, switch the cores to a higher frequency, and measure again within 10 ms.

Since the uncore voltage, uncore frequency, and package temperature are stable across readings (assuming that temperature change is insignificant within the very small sampling period), the difference in measured power must be due to changes in core dynamic power consumption. αC can now be calculated using Eq. 2. We calculate αC for multiple frequency pairs and take an average to reduce estimation error. We follow this approach to calculate αC for all microbenchmarks. We use linear regression to find the best fit for estimating the impact of non-memory instructions on core dynamic power. Eq. 5 shows the derived model, and Figure 2 shows its fitness with respect to our measured values. The R^2 fitness coefficient is 0.801.

$$P(core)_{non-mem\_dynamic} = (2.448 * 10^{-10} * x + 1.1809 * 10^{-9}) * V^2 * F \quad (5)$$

where x = Non-memory instructions per cycle
V = core supply voltage
F = core frequency

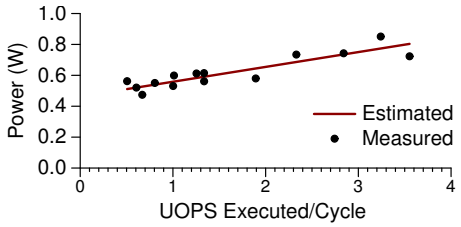


Figure 2: Model Fitness for Power Consumed due to Non-Memory Execution Units at F=800 MHz and V=0.7V

We next study effects of memory operations on core dynamic power. We maintain constant activity on the non-memory execution ports (ports 0, 1, 5, and 6) and gradually increase it on the load/store units (ports 2, 3, 4, and 7). We correlate increases in memory operations with growth in core dynamic power to generate the model in Eq. 6. Figure 3 shows the model fitness. The R^2 coefficient is 0.999.

$$P(core)_{mem\_dynamic} = (2.780 * 10^{-10} * x + 1.497 * 10^{-10}) * V^2 * F \quad (6)$$

where x = L1 accesses per cycle
V = core supply voltage
F = core frequency

To study the effects of L2 accesses on core dynamic power consumption, we follow the same approach as for studying the effects of L3 accesses. We again use our microbenchmark to increase the number of L2 accesses while keeping core activity constant. We measure the increase in CPU power consumption as we increase the L2 access rate. We then run linear regression on the L2 access rate and increase in power consumption to generate the model for the

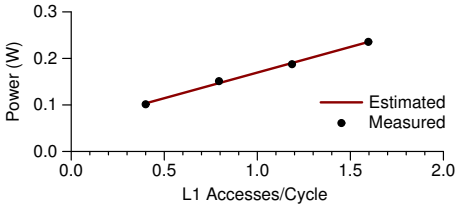


Figure 3: Model Fitness for Power consumed due to L1 Accesses at F=800 MHz and V=0.7V

L2-access contribution to core dynamic power consumption shown in Eq. 7. Figure 4 shows the model fitness with respect to our measured values. The R^2 coefficient that measures the goodness of the fit is 0.997.

$$P(core)_{L2\_dynamic} = 2.829 * 10^{-9} * x * V^2 * F \quad (7)$$

where x = per-core L2 accesses per uncore cycle
V = uncore supply voltage
F = uncore frequency

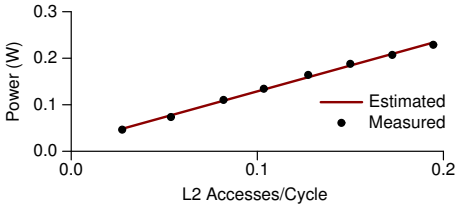


Figure 4: Model Fitness for Power Consumed due to L2 Accesses at F=800 MHz and V=0.7V

The total core dynamic power consumption can now be expressed by Eq. 8.

$$P(core)_{dynamic} = P(core)_{mem\_dynamic} + P(core)_{non-mem\_dynamic} + P(core)_{L2\_dynamic} \quad (8)$$

C. Uncore Static Power Model

The Haswell microarchitecture uses a deep-sleep C-state, C7, when a core is idle. In the C7 state, the cores are power-gated, and so they consume negligible power. In contrast, our experiments show that the Haswell uncore is neither power-gated nor clock-gated when the chip is idle. The idle power consumption of the chip is thus almost entirely due to the uncore. Chip idle power can be expressed by Eq. 9.

$$P(CPU)_{idle} = P(uncore)_{idle\_dynamic} + P(uncore)_{static} \quad (9)$$

We use the model for $P(uncore)_{idle\_dynamic}$ derived in Section II-A to isolate $P(uncore)_{static}$. Recall that static power consumption depends on supply voltage and temperature. We use CPU package temperature to approximate average temperature across the uncore. To measure effects of uncore voltage and uncore temperature on uncore static power, we set the uncore voltage to values ranging from 0.7V to 1.0V with increments of 0.05V. At each voltage, we vary CPU temperature using a hot-air gun, and we measure power consumption at the granularity of one sample per second. We subtract uncore idle dynamic power from the measured values and run non-linear regression on this difference, the voltage, and the temperature to create the uncore static power model in Eq. 10. This equation uses the Poole-Frenkel effect [17]. Figure 5 shows measured versus estimated uncore static values at three voltage points. We ran each experiment until temperature stopped dropping (the figure shows time windows of 200 seconds for ease of comparison — experiments that ran longer continued to show high estimation accuracy during the time not shown). The R^2 coefficient for the estimation model is 0.999 across all sample points.

$$P(uncore)_{static} = 141.615 * e^{-(\frac{-169.083+1202.02*\sqrt{V}}{273.15+T})} * V^2 \quad (10)$$

where V = uncore supply voltage
T = package temperature

D. Core Static Power Model

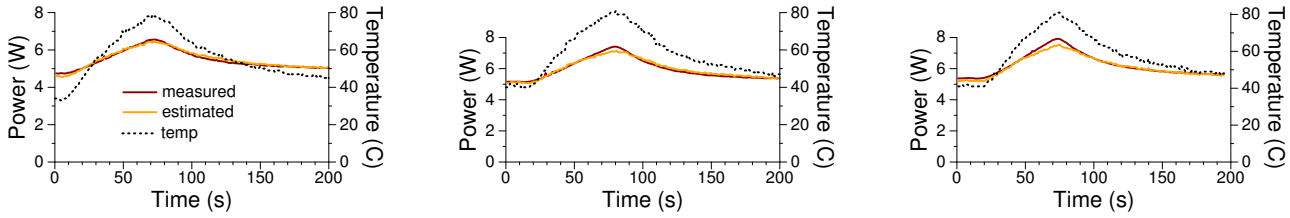
To generate a model for core static power, we force the core to remain in state C0. In this state, the Linux kernel runs a polling idle loop that consumes constant dynamic power. We calculate this power with the same strategy we used to calculate uncore idle dynamic power. We calculate αC for idle cores in state C0 using Eq. 2. Eq. 11 shows the model for per-core dynamic power in state C0. Eq. 12 uses Eq. 11 to isolate per-core static power consumption.

$$P(core)_{C0\_dynamic} = 1.28 * 10^{-9} * V^2 * F \quad (11)$$

where V = core supply voltage
F = core frequency

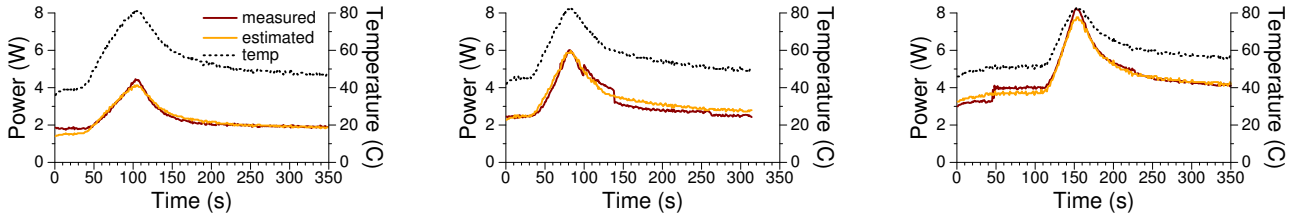
$$P(core)_{static} = \frac{1}{4} * (P(CPU)_{C0} - P(uncore)_{idle\_dynamic} - P(uncore)_{static} - 4 * P(core)_{C0\_dynamic}) \quad (12)$$

To create the core static power model we first measure the effects of core voltage and temperature. We use package temperature as an approximation for core temperature. We fix cache voltage at 0.7V and frequency at 800 MHz from the BIOS. We again incrementally increase core voltage from 0.7V to 1.05V in steps of 0.05V. At each voltage, we vary package temperature using a hot-air gun, and we measure changes in power consumption and temperature. We run non-linear regression on collected data to create the core static model in Eq. 13. Figure 6 shows measured versus



(a) V=0.70 (b) V=0.85 (c) V=1.0

Figure 5: Uncore Static Model Fitness



(a) V=0.70 (b) V=0.85 (c) V=1.0

Figure 6: Core Static Model Fitness

estimated core static values at three voltage points. The estimated model fits measured values with an R^2 coefficient of 0.996.

$$P(core)_{static} = 1525.07 * e^{-(\frac{1884.1+525.556*\sqrt{V}}{273.15+T})} * V^2 \quad (13)$$

where V = core supply voltage
T = package temperature

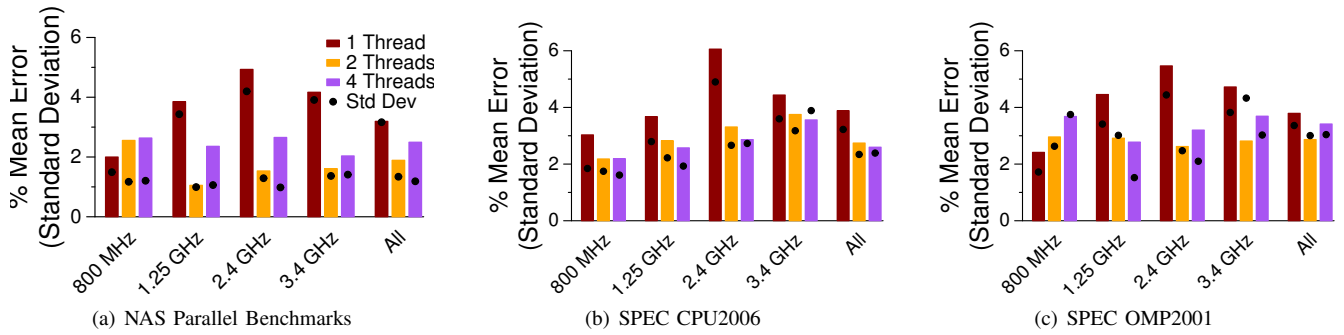
E. Total Chip Power Model

Based on our findings, we must consider two more components in constructing a full-chip power model. First, our measurements show that when core frequency is increased, chip power consumption increases more than expected at certain frequency steps. These increases appear to depend on the number of active cores but not on core activity. We believe this phenomenon is due to the core PLL switching to higher supply voltages above certain frequencies. Second, chip power consumption increases abruptly at higher temperatures — by up to 0.5W per active core — but our experiments and research have been unable to determine the source of this phenomenon. These two power-consumption components are difficult to model without more information about their causes. We therefore construct an offline table of empirically determined increases in power consumption due to these factors, acknowledging that deeper understanding is required to accurately predict their behavior. We represent these two components collectively with $P(misc)$. Our experiments indicate that $P(misc)$ can amount to as much as 10% of total chip power consumption, depending on the level of core and uncore activity. Total chip power consumption is then given as:

$$P(CPU) = P(uncore)_{dynamic} + P(uncore)_{static} + P(core)_{dynamic} + P(core)_{static} + P(misc) \quad (14)$$

III. Validation

We validate our power models on the single-threaded SPEC CPU2006 [18] benchmarks, NAS Parallel Benchmarks [19], and multithreaded SPEC OMP2001 [20] applications in Table I. We omit *bwaves* from SPEC CPU2006 and *art* and *fma3d* from SPEC OMP2001 because they do not run on our system. We use the NAS class B inputs, and we omit *IS* because it does not run for long enough time periods to gather reliable measurements. We run the parallel applications with one, two, and four threads. Similarly, we run one, two, and four concurrent instances of the sequential applications. Once per second we read package temperature and core voltage, and collect performance counter as described in Section II. We measure processor power every 1 ms and average the values over one second to sync the power values with other parameters. To verify that our model estimates power accurately across different voltage-frequency steps, we validate it at four DVFS points: 800 MHz at 0.7V, 1500 MHz at 0.78V, 2400 MHz at 0.88V, and 3400 MHz at 1.01V. We again disable hyper-threading and Turbo Boost.



(a) NAS Parallel Benchmarks (b) SPEC CPU2006 (c) SPEC OMP2001

Figure 7: Validation of Total Chip Power

Suite	Benchmarks
NAS	SP, EP, BT, MG, DC, UA, CG
SPEC OMP2001	quake, swim, wupwise, ammp, apsi, applu, mgrid
SPEC CPU2006	cactusADM, calculix, dealII, gamess, GemsFDTD, gromacs, lbm, leslie3d, milc, namd, povray, soplex, zeusmp, astar, bzip2, gcc, gobmk, h264ref, hmmer, libquantum, mcf, omnetpp, perlbench, sjeng, xalancbmk

Table I: Benchmarks Used for Validation

Figure 7 shows estimation error for our benchmark suites. Mean absolute error (MAE) for all NAS benchmarks across the four DVFS points is 3.19% for a single thread, 1.89% for two threads, and 2.50% for four threads. MAE for SPEC CPU2006 is 3.88% for single-instance runs, 2.74% for double-instance runs, and 2.60% for quad-instance runs. MAE for SPEC OMP2001 is 3.79% for a single thread, 2.87% for two threads, and 3.42% for four threads. Mean absolute error across all sample points for all benchmarks and voltage-frequency states is 3.14%, and the standard deviation is 2.87%.

Our model predicts power estimation with good accuracy for most benchmarks. For the benchmarks with rapid phase changes (*DC* in NAS and *ammp* in SPEC OMP2001), we had to re-run the validation experiments at the higher measurement granularity of 100 ms to accurately estimate their rapidly changing power consumption values.

We know that model error arises from at least two sources. First, the Haswell microarchitecture provides no per-core event to track executed floating point operations, which prevents us from creating separate models for floating point and integer instruction executions. This creates slight model inaccuracies for compute-intensive phases. Second, values in the offline table we create to account for $P(misc)$ are not always accurate, especially at higher frequencies during periods of rapid phase change. Despite these known sources of error, our model accurately predicts workload energy trends when scaling both frequency and numbers of threads.

IV. Energy Characterization

Dynamic voltage-frequency scaling permits hardware or software to adjust clock speed and/or voltage levels to try increase performance or save power. The degree of parallelism in a workload (i.e., the number of parallel threads) can also be increased to try to improve performance, but such scaling often comes at an energy cost. Finding the frequencies or numbers of threads at which an application meets performance goals while maintaining a given energy budget is not necessarily straightforward. We use our power models to characterize energy efficiency by showing how static and dynamic energy from the core and uncore components scale in conjunction with DVFS and thread-level parallelism.

We perform sensitivity analyses of energy consumption at different voltage-frequency points and thread concurrency levels. In these studies we employ a parameterized synthetic workload that lets us we vary both memory-access intensity and thread-level parallelism. We validate results from this synthetic workload against those from real applications.

A. Energy Effects of DVFS

We first examine how voltage-frequency scaling affects energy consumption. Conventional wisdom says that running at higher frequencies boosts performance at the expense of expending more energy [14] because dynamic power scales quadratically with voltage (Eq. 1). But when we take into account the effects of energy consumed by the uncore, we find that running at a lower voltage-frequency step can sometimes expend *m*ore energy. Figure 8 shows how energy scales for our synthetic workload as we vary its memory intensity and the frequency at which we execute. Energy numbers are normalized to the energy expenditure at the lowest frequency (800 MHz).

Figure 8(a) shows that a completely sequential, CPU-bound application (which hence scales linearly with frequency) running at the lowest frequency expends the highest total energy, even though the core dynamic energy is lowest at this frequency. This happens because the uncore energy accounts for 74% of the total, but it contributes nothing

to application performance. Specifically, uncore static energy constitutes 61% of the total. When the frequency is increased, core dynamic energy goes up. However, reducing execution time drastically reduces uncore static energy, which lowers overall energy. After a certain frequency, increases in (core and uncore) dynamic energy dwarf decreases in uncore static energy, causing total energy expenditure to rise again.

The more memory-bound an application, the less performance it gains from running at higher frequencies, and so reductions in uncore static energy become less significant with increasing frequency. When the memory-bound portion of a single-threaded application exceeds 40%, we see the expected trend: increasing the frequency increases energy expenditure. Figure 8(b) shows energy-scaling results for a four-threaded workload. Uncore static energy is less significant for multi-threaded, CPU-bound applications (we measure 49% at two threads and 35% at four) because of the increased ratio of core power to total chip power. As a result, energy scaling correlates with frequency scaling when a two-thread application is 30% memory-bound and a four-thread application is 20% memory-bound (compared to 40% for a single-threaded application). This analysis shows that the extent to which an application is memory bound must be taken into account when choosing a frequency at which to run the application such that it expends the least energy.

B. Energy Effects of Scaling the Number of Threads

On a processor with ideal energy efficiency, performing a given amount of work should expend roughly the same amount of energy, regardless of the number of cores used. But the sources of energy inefficiency in modern processors result in energy trends that differ from this "perfect" scenario. We use our power models to analyze how scaling concurrency levels affects total energy expenditure. Figure 9 shows energy scaling across different levels of parallelism when we vary the proportion of serial code while keeping the amount of work constant. Energy values are normalized to the energy expenditure of a single-threaded run. When the workload is completely parallel, using all available cores expends the least energy, since increasing performance reduces both uncore static and uncore idle dynamic energy. Core static energy increases slightly due to higher package temperatures when using more cores, but core dynamic energy remains the same. At 3.4 GHz, going from one to two threads expends 26% less energy, and going from one to four threads expends 40% less.

As we increase the relative portion of serial execution, core dynamic energy increases even though the amount of work does not change. Even though the processor executes the same number of instructions, total active core time grows due to pipeline stalls from increased inter-core communication. This increases core idle dynamic energy. At 3.4 GHz (Figure 9(a)), when the serial portion of the application exceeds 20%, using two cores becomes more efficient than four. When over 40% of the application is serial, running a single thread becomes more energy-efficient than running parallel threads, even though speedup when going from one to four cores is close to 100%.

The ratio of uncore static energy to total energy is much higher at 800 MHz (Figure 9(b)) than at 3.4 GHz, and thus increasing the number of threads causes energy savings from lower uncore static energy consumption to become more prominent. At 800 MHz, running at four threads is still most energy efficient, even when 30% of the application is serial (compared to 20% at 3.4 GHz). A workload must be almost 70% serial before the serial version becomes more energy-efficient than multithreading (compared to 40% at 3.4 GHz).

C. Benchmark Energy Scaling

To validate the energy trends predicted using our model's sensitivity analyses, we examine the energy expenditure of six of the NAS benchmarks. Figure 10 shows the scaling of energy expenditure and execution time for NAS benchmarks at different frequencies and different levels of concurrency. Please note that these are measured values.

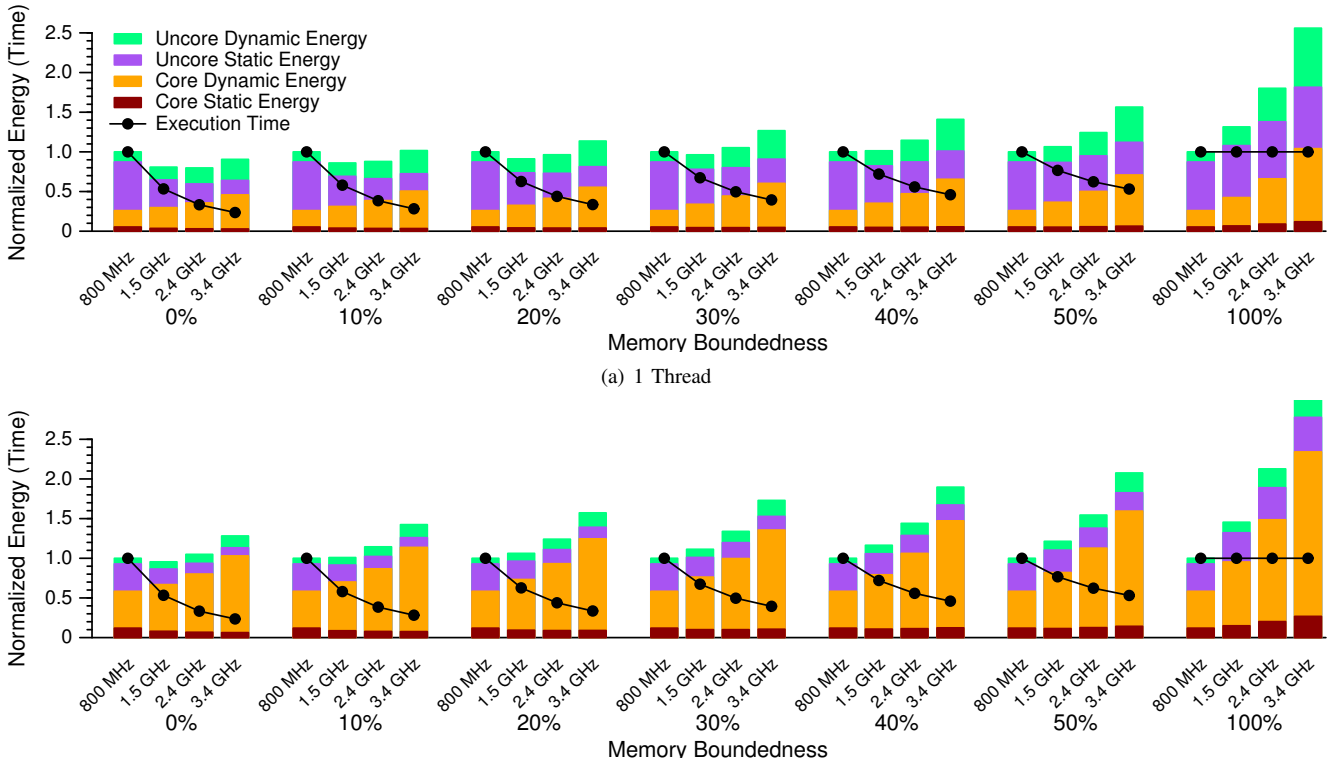
EP, the Embarrassingly Parallel benchmark, is CPU bound: it has a small working set with no serial sections. Figure 10(a) shows that sequential runs consume maximum energy at the lowest frequency, as we expect from the trends in Figure 8(a). For the four-thread run, *EP* expends the most energy at 3.4 GHz and the least at 1.5 GHz, which follows the trends in Figure 8(b). Energy expenditure decreases when scaling to more threads, which validates our analyses.

BT has a larger working-set than *EP*. For sequential executions, the energy scaling trends in Figure 10(b) resemble those of the 10% memory-bound workload in Figure 8(a). At four threads, energy expenditures closely match trends for the 10% memory-bound workload in Figure 8(b). Our remaining benchmarks (*CG*, *SP*, *MG*, and *LU*) have even larger working sets, and their energy scaling trends echo those for the 20% memory-bound workload in Figure 8.

In terms of thread scaling, execution times of most of the NAS benchmarks scale almost linearly, and energy expenditures decrease with higher concurrency. One exception is *SP* running at 3.4 GHz. Scaling from one to two threads yields an almost linear speedup (48% less execution time), but scaling from two to four threads reduces execution time by just 36% percent due to pipeline stalls. Core dynamic energy increases, making *SP* expend more total energy at four threads than two, as our model predicts.

V. Related Work

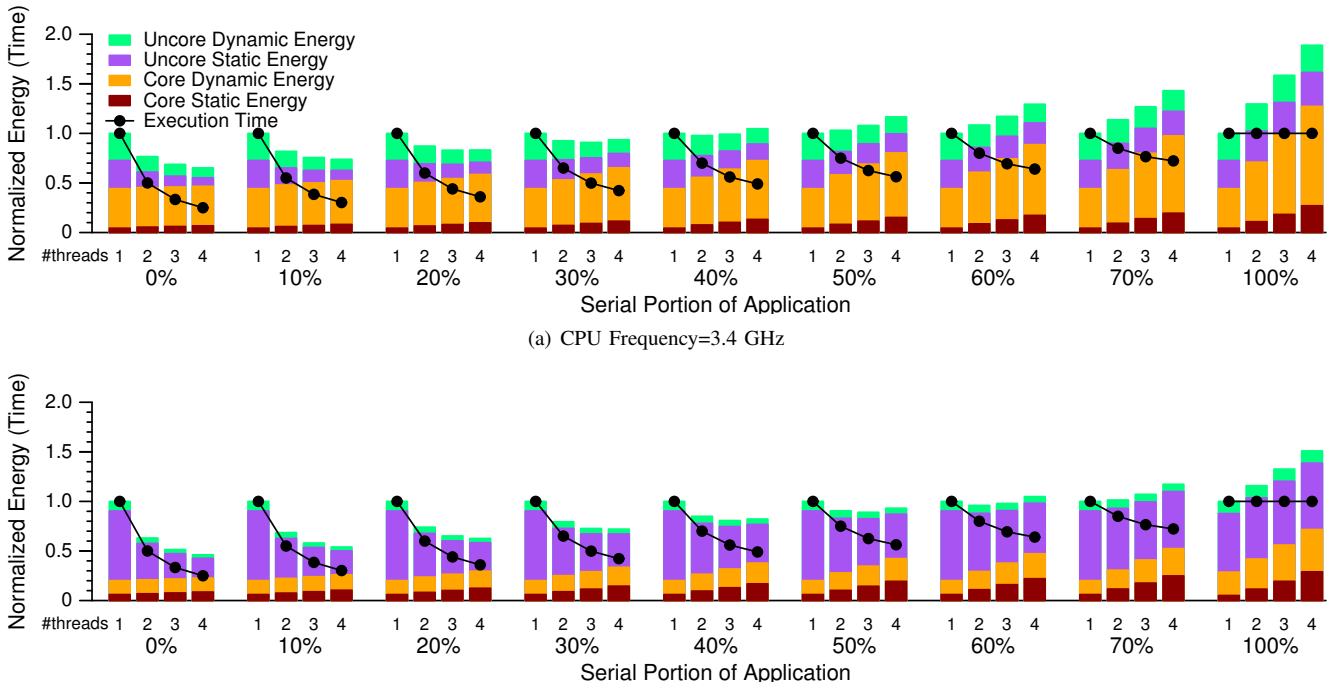
The difficulty of obtaining accurate, real-time power measurements together with the growing emphasis on green computing have sparked much research on power-estimation models. An early effort by Tivari et al. [21] develops a measurement-based approach to model instruction-level



(a) 1 Thread

(b) 4 Threads

Figure 8: Effects of DVFS on Total Energy and Performance



(a) CPU Frequency=3.4 GHz

(b) CPU Frequency=800 MHz

Figure 9: Effects of Thread Scaling on Total Energy

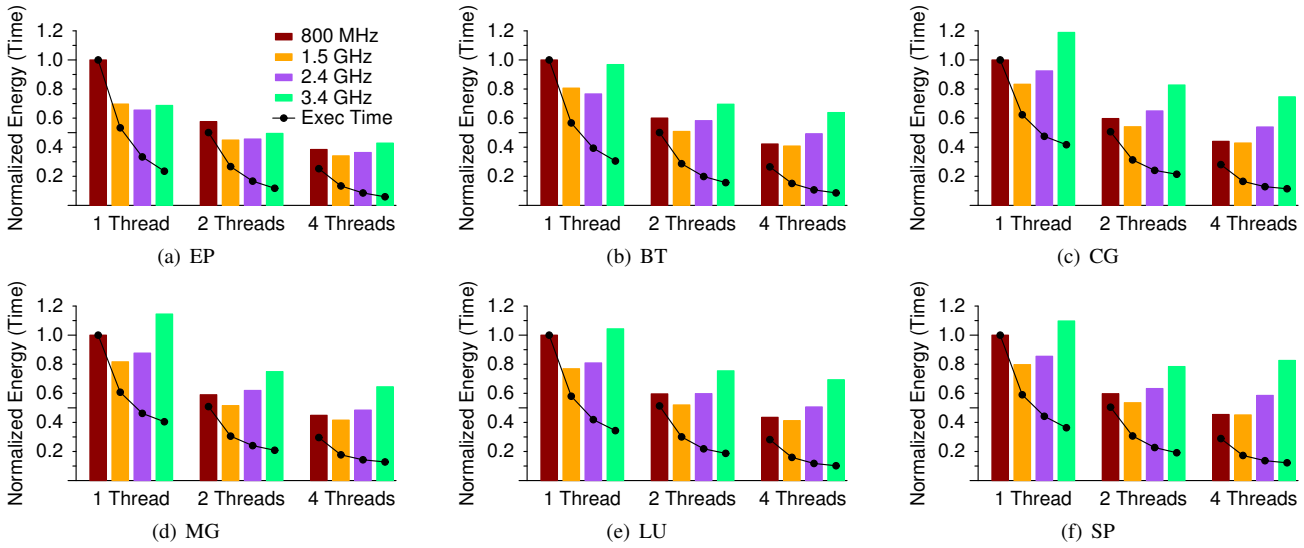


Figure 10: Energy Scaling for NAS Benchmarks

power effects. They associate energy cost for each type of instruction, instruction pairs and inter-instruction effects like pipeline stalls and cache misses. Unlike us, they do not distinguish between static and dynamic power components. Their methodology requires forming the models separately at each processor frequency.

More recent work increasingly relies on hardware performance monitoring counters (PMCs) to build the processor power model. Joseph and Martonosi [11] model power consumption of individual microarchitectural components by choosing performance counters that intuitively correlate with component utilization and deriving weights for each component's activity factor. They run their experiments at a single frequency (since DVFS was not common in 2001) and make no distinction between static and dynamic power.

Singh et al. [8] estimate per-core power consumption for an AMD Phenom [22] by running multiple linear regression on just four PMCs (representing integer operations, floating point operations, memory accesses, and pipeline stalls). Singh [23] later incorporates temperature and frequency scaling. Goel et al. [9] further refine and validate this general approach, demonstrating consistently high model accuracy across several architectures. Goel et al. [10] investigate the impact of different power measurement infrastructures on model accuracy.

Like us, Bertran et al. [13] develop a power model that sums dynamic and static power. However, they define static power as idle power and use regression on the values of their chosen PMCs to derive both static (idle) and dynamic power. Their power models disregard both voltage and temperature.

Spiliopoulos et al. [24] address static and dynamic power separately in their approach to power estimation, but they also define static power as idle power. They create a table (offline) of idle power values at all frequency steps and "typical core temperature ranges". For the dynamic power component in their models, they estimate the effective processor activity factor based only on instructions retired. Our experience indicates that models ignoring memory operations are prone to error on memory-bound workloads.

Su et al. [14] also address idle and dynamic power separately. Like us, they use voltage and temperature to develop a static power model, but they group idle dynamic power into their static power model, which prevents them from reflecting the true static power consumption of the core and the northbridge (equivalent to the uncore in our models). They use multiple linear regression on nine PMCs to derive their dynamic power model, but relying on so many counters requires time multiplexing, which sacrifices accuracy. They conclude that their AMD FX-8320® chip always expends least energy at the lowest voltage-frequency state, which our Haswell findings contradict.

VI. Conclusions

Accurately estimating the power consumption of processor components is important for supporting better power-aware resource management. We present a methodology to estimate static and dynamic power consumption of the core and the uncore processor components. The methodology uses core and uncore voltage, package temperature, and performance counters to create models that can estimate power consumption for sequential and parallel applications across all system frequencies. We validate our models on benchmarks from NAS, SPEC CPU2006, and SPEC OMP2001, and we show that our models can estimate power with high accuracy (3.14% mean absolute error) across all voltage-frequency pairs and different concurrency levels.

We use our power models to study the impact of DVFS on energy consumption, showing that — contrary to conventional wisdom — it is not always most energy efficient to run applications at the lowest frequency. Uncore static energy effects must be taken into consideration. The frequency at which an application expends the lowest energy depends on how memory bound it is and how many concurrent threads it uses. We study the impact of thread scaling on energy expenditure, showing that the relative serial portion of a program influences the level of concurrency at which the program will expend the least energy.

ACKNOWLEDGMENTS

This work was funded in part by the European Union FP7 project EUROSERVER under grant agreement 610456.

REFERENCES

[1] D. Jenkins, "Node Manager — a dynamic approach to managing power in the data center," White Paper, Intel Corporation. http://communities.intel.com/docs/DOC-4766, Jan. 2010.

[2] E. Rotem, A. Naveh, D. Rajwan, A. Ananthadrishnan, and E. Weissmann, "Power management architecture of the 2nd generation Intel® Core™ microarchitecture, formerly codenamed Sandy Bridge," in *Proc. 23rd HotChips Symposium on High Performance Chips*, Aug. 2011.

[3] Advanced Micro Devices, "AMD Opteron™ 6200 series processors Linux tuning guide," 2012, http://developer.amd.com/wordpress/media/2012/10/51803A_OpteronLinuxTuningGuide_SCREEN.pdf.

[4] *Nvidia NVML API Reference Manual*, NVIDIA, 2012.

[5] K. Rajamani, H. Hanson, J. Rubio, S. Ghiasi, and F. Rawson, "Application-aware power management," in *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, Oct. 2006, pp. 39–48.

[6] G. Contreras and M. Martonosi, "Power prediction for Intel XScale processors using performance monitoring unit events," in *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, Aug. 2005, pp. 221–226.

[7] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system power analysis and modeling for server environments," in *Proc. Workshop on Modeling, Benchmarking, and Simulation*, Jun. 2006.

[8] K. Singh, M. Bhadauria, and S. McKee, "Real time power estimation and thread scheduling via performance counters," in *Proc. Workshop on Design, Architecture and Simulation of Chip Multi-Processors*, Nov. 2008.

[9] B. Goel, S. McKee, R. Gioiosa, K. Singh, M. Bhadauria, and M. Cesati, "Portable, scalable, per-core power estimation for intelligent resource management," in *Proc. 1st International Green Computing Conference*, Aug. 2010, pp. 135–146.

[10] B. Goel, S. McKee, and M. Själander, *Techniques to Measure, Model, and Manage Power*, ser. Advances in Computers. Elsevier, Nov. 2012, vol. 87, ch. 2, pp. 7–54.

[11] R. Joseph and M. Martonosi, "Run-time power estimation in high-performance microprocessors," in *Proc. IEEE/ACM International Symposium on Low Power Electronics and Design*, Aug. 2001, pp. 135–140.

[12] F. Bellosa, S. Kellner, M. Waitz, and A. Weissel, "Event-driven energy accounting for dynamic thermal management," in *Proc. Workshop on Compilers and Operating Systems for Low Power*, Sep. 2003.

[13] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *Proc. 24th ACM International Conference on Supercomputing*, Jun. 2010, pp. 147–158.

[14] B. Su, J. Gu, L. Shen, W. Huang, J. Greathouse, and Z. Wang, "PPEP: Online performance, power, and energy prediction framework and DVFS space exploration," in *Proc. IEEE/ACM 47th Annual International Symposium on Microarchitecture*, Dec. 2014, pp. 445–457.

[15] J. Mair, D. Eyers, Z. Huang, and H. Zhang, "Myths in power estimation with Performance Monitoring Counters," *Elsevier Sustainable Computing: Informatics and Systems*, vol. 4, no. 2, pp. 83–93, Jun. 2014.

[16] M. Berktold and T. Tian, "CPU monitoring with DTS/PECI," White Paper 322683, Intel Corporation. http://download.intel.com/design/intarch/papers/322683.pdf, Sep. 2010.

[17] R. B. Hall, *Thin Solid Films*. Elsevier, Oct. 1971, vol. 8, ch. 2, pp. 263–271.

[18] Standard Performance Evaluation Corporation, "SPEC CPU benchmark suite," http://www.specbench.org/osg/cpu2006/, 2006.

[19] D. Bailey, T. Harris, W. Saphir, R. Van der Wijngaart, A. Woo, and M. Yarrow, "The NAS parallel benchmarks 2.0," NASA Ames Research Center, Report NAS-95-020, Dec. 1995.

[20] Standard Performance Evaluation Corporation, "SPEC OMP benchmark suite," http://www.specbench.org/hpg/omp2001/, 2001.

[21] V. Tivari, S. Malik, A. Wolfe, and M.-C. Lee, "Instruction level power analysis and optimization of software," *Kluwer Journal of VLSI Design*, vol. 13, no. 3, pp. 223–238, 1996.

[22] AMD Corporation, "Model number and feature comparisons — AMD Phenom. Processors," http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_15331_15332\%5E15347,00.html, Dec. 2007.

[23] K. Singh, "Prediction strategies for power-aware computing on multicore processors," Ph.D. dissertation, Cornell University, Jun. 2009.

[24] V. Spiliopoulos, S. Kaxiras, and G. Keramidas, "Green governors: A framework for continuously adaptive DVFS," in *Proc. 2nd International Green Computing Conference*, Jul. 2011, pp. 1–8.