Full-System Chip Multiprocessor Power Evaluations Using FPGA-Based Emulation

Abhishek Bhattacharjee, Gilberto Contreras and Margaret Martonosi
Department of Electrical Engineering
Princeton University
{abhattac, gcontrer, mrm}@princeton.edu

ABSTRACT

The design process for chip multiprocessors (CMPs) requires extremely long simulation times to explore performance, power, and thermal issues, particularly when operating system (OS) effects are included. In response, our novel FPGA-based emulation methodology models a full CMP design including applications and an OS. Activity counters programmed into the cores feed per-component microarchitectural power models. These models achieve under 10% error compared to detailed gate-level simulations. Our method retains software flexibility, but offers up to 35× speedup compared to corresponding full-system software simulations. We present our approach by emulating a 2-core Leon3 cache-coherent multiprocessor running Linux and parallel benchmarks. In an example case study, our emulated system uses activity counts (a proxy for temperature) to guide process migration between the CMP cores. Overall, this paper's methodology makes possible detailed power and thermal studies of CMPs and their operating systems.

Categories and Subject Descriptors

B.8 [**Performance and Reliability**]: Performance Analysis and Design Aids; C.1 [**Processor Architectures**]: Parallel Architectures; C.4 [**Performance of Systems**]: Measurement Techniques

General Terms

Design, Measurement, Performance

1. INTRODUCTION

Increasing transistor counts and superlinear clock scaling have led to a significant rise in microprocessor power density, introducing higher operating temperatures and potential hotspots. Higher temperatures result in reliability issues [23] and prohibitively expensive packaging solutions [17]. As chip multiprocessors (CMPs) become the dominant microprocessor approach, development of efficient and accurate CMP power estimation becomes crucial.

Power estimation in the early design stages is a critical component of processor design. From this, various circuit-level and architectural techniques can be employed to mitigate thermal and power problems [4]. Conventionally, software simulation is used at the architectural [4], RTL, gate, and layout levels of design. However, full system simulations come at the expense of extremely long simulation times. To counter this, application snippets are employed and operating system effects ignored, compromising the credibility and accuracy of results. The problem is compounded for thermal studies, which require the simulation of tens to hundreds of milliseconds to study equilibrium operating points [21]. Although existing simulators may be parallelized for speedup, shared data structures such as coherence engines and the L2 cache limit the scalability of this approach [6]. Moreover, simulating CMPs with increasing core-counts results in a commensurate slow-down in simulation speed.

One solution uses hardware performance counters (HPCs) in existing microprocessors for accurate component and total power estimates [9, 14, 16]. Orders of magnitude faster, runtime power estimation using well-tuned HPC-based power models can be highly accurate. While this is an attractive alternative to software simulation, it is infeasible in early design stages since it requires that the processor already exist.

The increasing capability of Field Programmable Gate Arrays (FPGAs) offers emulation-based alternatives. Due to their inherently programmable fabric and because they run at hardware speeds, it is possible to combine fast simulation time with high accuracy and flexibility for early power estimation [11, 24]. Therefore, the concept of power emulation on FPGAs warrants examination. In this context, the novel contributions of this paper are:

- We develop an FPGA-based power emulator for a hypothetical CMP based on the Leon3 Sparc V8 core [1]. Our system runs 2 cache-coherent cores at 65 MHz and boots Linux 2.6. This achieves a speedup of up to 35× over full-system architectural software simulators. To our knowledge, this is the first FPGA-based full-system power emulation framework to be built.
- We evaluate the accuracy of our approach by comparing the emulated power with simulated power from Synopsys PrimeTime PX across a series of benchmarks. Our power models have under 10% average error.
- A case study involving runtime power profiling and OS regulated process migration highlights the benefits of our approach.

Although the emulation framework currently models Leon3 cores, we envision our methodology as general enough for early power estimation of any proposed architecture.

Our paper is organized as follows. We discuss related work in the next section and the novelty of our approach in Section 3. Section 4 details infrastructure methodology while Section 5 discusses results. Section 6 illustrates a case study, and Section 7 concludes.

2. RELATED WORK

A framework for power density and thermal studies in early design stages must satisfy three requirements. First, it must yield accurate per-component power estimates based on architectural events of interest. Second, fast data collection is critical to achieve high efficiency. Finally, the platform must provide power estimation of a *proposed* architecture that has not been implemented yet. We review prior work in the context of these three requirements.

2.1 Software Simulators

Early architecture-level power simulators included Wattch [4] (for out-of-order processors) and SimplePower [22] (for

ISLPED'08, August 11–13, 2008, Bangalore, India.

in-order processors). Wattch, for example, used capacitance-based power models for modules such as arrays, CAMs, combinational logic, and clocking components. Similarly, Skadron et al. pioneered work in the thermal space by developing Hotspot [21]. These studies demonstrate the strengths and limitations of software simulators. First, component power estimates can be gathered easily by instrumenting appropriate modules in the simulator with event counters. Furthermore, proposed designs can be readily created within the simulator environment. However, due to long simulation times, accuracy may be compromised. Application snippets are employed to reduce the runtime while operating system effects are often ignored. This long simulation time is particularly problematic for thermal studies which require simulation of tens to hundreds of milliseconds of the target design [21]. A related problem is that the number of statistics being collected has a significant impact on simulation speeds. An increase in the former typically leads to a decrease in the latter.

Our emulation approach addresses these weaknesses by using a programmable hardware substrate to provide a 35× speedup, thereby allowing for full workloads and operating system effects to be included. As with software simulators, our infrastructure retains flexibility in designing the event counters to feed into component power models. However, we do this with no degradation in the emulation speed, unlike full-system simulators.

2.2 Hardware Runtime Monitoring

Runtime power estimation on existing processors is an attractive alternative. Feeding HPC counts from existing processors into highly-tuned power models has been demonstrated as a fast and accurate means of power estimation [2, 9, 14, 16]. However, runtime power estimation suffers from the glaring disadvantage of requiring an existing processor. Direct access to architectural components for live power measurements is also unavailable. Therefore, HPCs are used to infer component-wise power consumption. While greater speed does offer higher potential accuracy, well-tuned power models are necessary. The number and variety of HPCs can restrict the development of these component power models, as demonstrated by Contreras and Martonosi [9]. Moreover, since HPCs are designed to measure *performance*, they may not cover the important *power* events. It is precisely these weaknesses that our framework addresses due to its *programmable* nature and access to individual components.

2.3 Hardware Emulation

Hardware emulation has been viewed as a mechanism to attain the flexibility of software with the speed of hardware. However, the restrictive capacities of earlier generation FPGAs posed problems. For this reason, the RPM project [20], one of the first FPGA-based emulators, evaluated only the memory subsystem for a maximum of 8 processors. Additionally, no OS was supported. With newer, more powerful FPGA generations, interest in emulation has been rekindled. Large-scale efforts such as RAMP [24] and HAsim [11] have primarily been concerned with the methodology involved in creating modular, parameterizable performance models of complex architectures. Hybrid hardware/software approaches such as Protoflex [8] emulate common processor functionality such as pipeline operations on FPGAs while offloading infrequent operations such as I/O to software simulation. Despite these advances, a full-system emulation framework focusing on power/thermal studies remains unaddressed. While Coburn et al. [15] investigate power emulation, they do so at the RTL level. Furthermore, they address the implementation of power models at the hardware level itself and techniques to decrease area/latency overheads arising from this. In contrast, Ghodrat et al. [19] propose a hybrid approach with power emulation for a subset of SoC components complementing more general estimation with an HDL simulator. Similarly, Atienza et al. [10] focus on thermal emulation of MPSoCs. They analyze existing cores for which power characteristics are already available and use runtime communication between the FPGA and software on a host PC instead of a full-fledged OS on the emulated system.

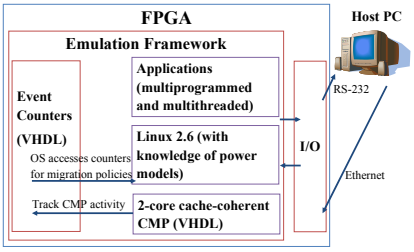


Figure 1: Emulator consists of power models, emulated CMP, OS, applications, event counters and host PC.

In contrast to all these approaches, we develop a methodology for emulation of a *proposed* CMP architecture in its early design stages and demonstrate interaction with complex applications and a full OS. Our target is to develop a full-system scalable architectural power emulation platform that operates without an HDL simulator backbone.

Overall, both software simulators and runtime monitoring systems present unique problems. Moreover, current emulation research has not focused on power/thermal studies in the early design stages of novel microprocessor designs. This represents a fundamental gap in the infrastructure available for studies of proposed architectures. Additionally, we expect this situation to worsen as designs increase in complexity, forcing software simulators to slow down and making it even more difficult to formulate component power estimates through runtime monitoring.

3. OUR APPROACH: FPGA-BASED FULL-SYSTEM POWER EMULATION

We address architecture's infrastructure gap by using FPGA-based emulation to combine the advantages of the software simulator and hardware runtime monitoring approaches for power and thermal studies. There are four novel traits in our work. First, unlike other emulation studies, we focus on power/thermal evaluations of a proposed cache-coherent CMP running complex workloads and Linux. Second, as with software simulators, we can instrument event counters of interest to estimate the power of individual architectural components. Third, due to its hardware speeds and because HPCs relevant to *power* can be implemented, we achieve potentially higher power estimation accuracy than either software simulators or hardware prototypes. Furthermore, the HPCs do not affect emulation speeds, as opposed to software simulators which slow down with statistics gathering. Fourth, as with software simulators, this approach can be used for power estimates of a *proposed* design.

Figure 1 details the interaction among components in our targeted emulation platform. We use the component power models (discussed in subsequent sections) in conjunction with our FPGA-based emulator to yield runtime power estimates of applications running on a cache-coherent CMP. As indicated, Linux runs on our emulated cores. The host PC uses the ethernet connection to upload applications on the emulator and receives statistics and OS/program output via the RS-232 connection. During application execution, event counters track the usage of different components in the design. As will be shown in the case study, the OS can feed these counter values into the power models to make decisions on process migration policies.

4. DETAILS OF OUR APPROACH

Our goal is to develop an emulation system which can accurately estimate CMP component power. To this end, we need to accomplish four objectives. First, we select a target CMP configuration based on a core design of interest. Second, we design component-specific event counters that will usefully reflect power dissipation in the core and the CMP system. Third, we develop component-specific power equations by assigning power weights to relevant counters. Finally, we validate our power models and refine them based on designed microbenchmarks.

This flow is a one-time investment. Once the appropriate power models have been developed, the emulation platform

Table 1: 2-core Leon3 Multiprocessor Configuration

Clock Rate	65 MHz
Organization	2-core, snooping for L1 cache coherence
Pipeline	Single-issue, in-order, 7-stage
Functional Units	Adder, Shifter, Pipelined Mul(5cc), Div(35cc)
L1 I-Cache	4KB, 2-way, 32-byte lines, LRR, 1 cycle hit
L1 D-Cache	4KB, 2-way, 32-byte lines, LRR write-through, virtually addressed, 1 cycle hit
MMU	8-entry I and D TLBs, LRU, 2 cycle TLB hit
Bus	ARM AMBA AHB bus with snooping

can be readily used to study the power characteristics of complex workloads with operating system effects.

4.1 Emulation Infrastructure

Our infrastructure platform is the Berkeley Emulation Engine 2 (BEE2) [5], which is comprised of five Xilinx Virtex II Pro 70 FPGAs. The FPGAs are laid out in a star topology with four user FPGAs in a ring and one control FPGA communicating with each user and the host PC. We currently use only the control FPGA. As we scale our design to emulate more complicated cores and higher core counts, we will use the other FPGAs as well.

Our approach is not specific to the BEE2. While we use it due to its high capacity and support, any FPGA board with sufficient resources would be usable.

4.2 System to be Emulated

Currently, we emulate a cache-coherent, bus-based CMP with Leon3 cores. The Leon3 is a VHDL model of a 32-bit SparcV8 architecture [1]. Details are listed in Table 1. In the future, we will add more cores, larger L1 caches, an L2 cache, and a floating point unit. The added complexity is feasible because the current design uses under 60% of the look-up tables (LUTs) and under 20% of the on-chip Block RAM (BRAM) for the caches on a single Virtex II Pro 70.

We stress here that the purpose of our work is to demonstrate a general approach. Therefore, other core designs or CMP system configurations would also be applicable.

4.3 Performance Counters for Power Models

The choice of events to monitor hinges on the specific components of interest. To that end, we develop power models for the integer pipeline, the integer register file, the caches, and the AHB bus controller. For each component, we modify the VHDL to assert pulses on events that cause the relevant 64-bit counters to increment. For example, a cache read hit asserts a pulse on a wire connected to the read hit event counter, resulting in its increment. All the counters reside in a distinct module on the bus and are hence non-obtrusive.

To guarantee cycle accuracy, we modify the pipeline to add counter start, counter stop, and counter reset instructions to the ISA. It takes 1 cycle to respond to these instructions (the time taken to exit the pipeline decode stage). Furthermore, the counters are memory-mapped allowing for a simple interface to gather statistics.

We currently model 36 counters for our power models. They cover events ranging from pipeline operations (instructions fetched, number of interlock stalls, instructions using different execution units), register file events, cache events (load hits and misses, store hits and misses, number of snoops), and bus transactions. Due to its simplicity, the counter module and associated logic uses less than 3% of the LUTs and has no impact on the operating frequency of the design.

4.4 Component Power Models

We next develop component power models for the pipeline, register file, caches, and AHB bus controller. Each power equation takes the form:

$$P_{component} = P_{idle} + \sum_i (E_i n_i f)/cycles \qquad (1)$$

Idle power is significant in Leon3 due to the absence of clock-gating. The second term of the relation, the events power, is calculated by finding the composite energy across all architectural events and scaling by the time taken to run the benchmark. Here, E_i is the energy per event i (for example, a register file write) and n_i is the event count. For technologies with significant leakage power, we could include a leakage term in the equation.

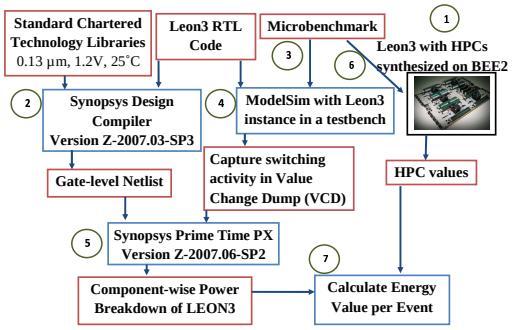


Figure 2: Flow for assigning energy to events.

Next we assign appropriate E_i values to the different events. Since we are interested in estimating the power of the *emulated* machine, measuring the power of the FPGA itself is meaningless. Instead, we calibrate our events with power numbers expected from a custom Leon3 microprocessor. Due to the absence of a commercial Leon3 multiprocessor, we calibrate with power numbers extracted from gate-level simulation. Our calibration flow is outlined in the following section.

4.5 Determining Event Energy Assignments

Figure 2 shows our calibration of event counter energies from detailed gate-level simulations. We start at point 1, with a synthesized version of our system with HPCs on the BEE2. We then synthesize an unmodified copy of the Leon3 RTL with Synopsys Design Compiler in step 2. This generates a gate-level netlist of the *emulated* machine with the 0.13μm Standard Chartered libraries. Step 3 creates microbenchmarks used to exercise certain portions of the system. To determine the energy of the microbenchmark, we capture its switching activity in a VCD file at step 4 via simulation on the Leon3 RTL in Modelsim. Now, we have both the gate-level netlist from step 2 and the switching activity from our microbenchmark. These become inputs to Synopsys PrimeTime PX in step 5, yielding a component-wise power estimate. We also run our microbenchmark on the BEE2 and gather performance statistics in step 6. At step 7, we use the counter values and the power breakdown to assign energy numbers to the events exercised in the microbenchmark. Steps 3-7 are repeated for different microbenchmarks to assign energy numbers to all the events of interest. Once all energy numbers for a component are assigned, the power model is complete.

A gate-level netlist rather than a placed-and-routed design is used for our calibration. While the latter yields higher accuracy, it also results in an exponential increase in simulation time. Since the microbenchmarks must run 500-1000 instructions, this approach is impractical. Instead, the gate-level netlist approach combines high accuracy with efficiency.

Although the eventual power models are behavioral in the sense that they track activity counts, they are faithful to real power estimates because of the accurate calibration with gate-level synthesis.

4.6 Discussion of Event Energy Assignments

Table 2 illustrates the idle power and event energy assignments per component. With these values, power models based on Equation 1 can be created. Due to the absence of clock-gating, the idle power numbers track the area of each component. Therefore, larger modules such as the caches consume significantly higher idle power.

Pipeline energy is assigned relative to the different execution units activated by the instructions. Instruction and data cache stalls freeze the entire pipeline and therefore only consume idle power. Dependency stalls result in the first 4 pipe stages shutting down while the last 3 drain. Hence, the energy for these stalls is low.

The I and D cache power models are primarily dependent on access hits and misses. The I-cache streaming cycles met-

Table 2: Event Energy Assignments

Pipeline	Idle: 19.97mW Add: 5.03nJ Shift: 3.43nJ Logical: 1.7nJ Br/Jmp: 1.93nJ Mul: 13.67nJ Div: 53.63nJ Dep Stall: 0.56nJ Other: 0.61nJ
Register File	Idle: 18.83mW Write: 0.53nJ Read Single: 0.29nJ Read Double: 0.39nJ
I-Cache	Idle: 82.34mW Hit: 1.46nJ Miss: 1.12nJ Streaming: 1.82nJ
D-Cache	Idle: 79.71mW Read Hit: 1.88nJ Read Miss: 2.08nJ Write Hit 2.37nJ Write Miss: 1.90nJ Snoops: 1.04nJ

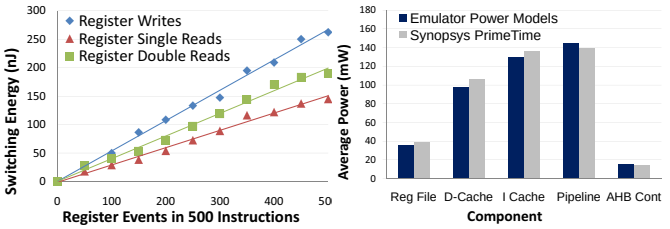


Figure 3: (a)Switching energy per register file event (b)Power model validation using `Libquantum` with all components under 10% error

ric accounts for energy expended during a cache line refill from main memory. Since the D-cache is write-through, there is little variation between read misses and write hits. However, a write miss uses a smaller energy quantum because the value is propagated to main memory without requiring updates in the cache.

To better understand the development of the microbenchmarks, we now focus on the register file power model.

4.6.1 *Case Study: Register File Power Model*

The register file has three architectural events: write, read single operand, and read double operand. To gauge the energy for each of these events, we write a series of microbenchmarks with 500 instructions. Only instructions causing the event under investigation and nops are present. We vary the number of event instructions and nops to study the energy change. We then run these microbenchmarks using the flow described in Figure 2. Figure 3a exhibits the linear relationship between the event instruction count and the energy expended by the register file. The energy per event may now be computed from the slopes.

4.7 Power Model Validation

Our power models have been extensively validated against both microbenchmarks and SPEC 2006 benchmarks. All the validation microbenchmarks run at least twice as long as the microbenchmarks used to develop the power models. Furthermore, the microbenchmarks now exercise multiple event types simultaneously. This tests the accuracy of the models when running complex workloads over longer periods of time. In addition, we validate our power models against PrimeTime power estimates for `Mcf` and `Libquantum` from SPEC 2006. Due to extremely long PrimeTime simulation times (in the order of days), we take 5 distinct 1-million-instruction snapshots of the SPEC 2006 benchmarks for validation.

We run the microbenchmarks and SPEC 2006 applications on the emulator to gather statistics. These statistics, using the power models, provide average power estimates per component. Concomitantly, the benchmarks are fed to PrimeTime for component power numbers. We then compare the estimates from our models against those from PrimeTime.

Table 3 demonstrates that our power models closely match PrimeTime power results with an error of under 10% for all components. Figure 3b reveals the accuracy of our estimates to PrimeTime results for `Libquantum`. Not only do component estimates closely match the PrimeTime power averages, they also track the power trends accurately.

This component-wise accuracy is a key benefit of our system. It will be particularly important as we develop thermal models in the future, in which per-component estimates and related floorplan information are crucial. This is in stark

Table 3: Avg. Power Model Errors Against PrimeTime

Component	Microbenchmarks	SPEC 2006
Pipeline	7.51%	5.74%
Register File	7.02%	6.31%
I-Cache	8.57%	8.85%
D-Cache	7.24%	7.44%
AHB Controller	5.66%	7.30%

contrast to works such as [14] where components such as the register file have no dedicated event counters. Instead, the number of micro-operations is scaled by an empirical factor to estimate switching activity. Thus, our approach combines high accuracy with high speed.

While the curve-fitting strategy is appropriate for average power, it may not enjoy the same level of accuracy in tracking peak and transient power. That being said, we designed our microbenchmark lengths and instruction mixes to sensitize the power models to peak and transient behavior. Furthermore, our experience with profiling runtime power of applications indicates that power surges and hotspots are tracked.

5. RESULTS

5.1 Emulation Speedup over Simulation

We measure the speedup our emulation framework offers over a gate-level simulator (Synopsys PrimeTime), and over an architectural simulator (Multifacet GEMS [18]). For the first case, we use the runtimes of the microbenchmarks and the 1 million instruction snapshots from `Mcf` and `Libquantum`. Compared to PrimeTime running on a lightly-loaded 64-bit, 2.2 GHz AMD Opteron 848, our approach achieves an average speedup of 33Million×! For perspective, while it takes 6 days to simulate `Mcf` on PrimeTime, it takes 10ms on our emulator.

We also compare our emulator to a full-system, cycle-accurate multiprocessor simulator, GEMS, running on a 64-bit, 2 GHz dual-core AMD Athlon processor. We simulate 2 cores with L1 caches configured to match our Leon3 emulation setup and remove the L2 cache [18]. We then run SPEC 2006 benchmarks (`Mcf`, `Libquantum` and `Bzip2`) with the train workloads to evaluate the speedup of the emulator over GEMS. We observe speedups of up to 35× across the tested benchmarks.

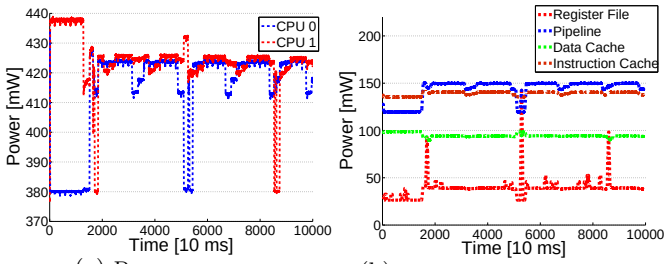
The speedup varies with application characteristics: for example, memory-intensive `Mcf` takes over 26 hours to run on GEMS but 46 minutes on our emulator, resulting in a speedup of 35. Since our system models the pipeline interactions in detail as opposed to our configuration of GEMS with only Ruby loaded [18], the speedup numbers are actually a conservative estimate. Moreover, the speedup quoted is dependent on our current cache configuration and emulator frequency. We anticipate even higher gains as we implement larger caches and increase our core frequencies.

We stress that we characterize our system benefits as conservatively as possible. Therefore, the power estimates are compared to highly accurate gate-level simulations rather than architectural power simulators. At the same time, the speedup of the emulator is compared against architectural simulators, which are much faster than gate-level simulation.

5.2 Runtime Power Profiling

5.2.1 *Kernel Modifications*

In preparation for our case study on power-aware OS-regulated process migration, we now detail our implementation of runtime power profiling. We modify the Linux kernel to accomplish this by adding instructions that read the HPCs within the 10ms kernel timer interrupt. We design system calls to initiate profiling and parameterize the sampling rate in multiples of 10ms. The counter values for each sample are stored in a kernel buffer. To minimize the cycles required for HPC access, we bypass the virtual memory system by using special macros that reference physical addresses directly. This optimization allows for the access of all 36 counters within the timer interrupt. The overhead as-



(a) Per-core power (b) CPU0 component power

Figure 4: Runtime power profiles for LU with 2 threads on emulator: while component profiles indicate sudden power surges for the register file, this is undetected on the composite core profiles.

sociated with our profiling is roughly 5700 clock cycles. Even for the finest sampling granularity of 10ms, this corresponds to a negligible 0.87% perturbation of runtime.

5.2.2 Power Profiling Results

We have profiled the runtime power of a number of SPEC 2006 benchmarks as well as multithreaded applications from Splash 2 and the PARSEC suite [3] using our emulation framework. For example, Figure 4 displays the runtime power characteristics of LU from Splash 2 running with 2 threads on our emulation system. The raw power numbers and swings are low compared to current microprocessors for several reasons: a 65MHz clock, a simple in-order, single-issue pipeline, small caches, the absence of a floating point unit, and lack of clock-gating. However, this does not detract from the generality of our approach and if each core dissipated more power, we would also see a larger power swing.

Figure 4a illustrates the power variation between the two cores. There is a visible phase for the first 15s when only core 1, which runs the master thread, is active. It consumes close to 440 mW while core 0 spins on an idle thread. From our measurements, the idle thread consumes roughly 380 mW. When the second thread is spawned, computation begins and periodic transitions in the power consumption for both cores are seen. Figure 4b provides a component-wise breakdown of the power for core 0. Most of the component power numbers are low initially because the core runs the idle thread. The data cache is the sole exception primarily because most of the idle thread's accesses are cached. Since cache accesses take a single cycle to expend the read hit energy quantum, more power is consumed. Once the computation begins however, cache misses lead to additional stalls which effectively lower the power. The pipeline and the I-cache profiles track each other. This is expected since an I-cache miss stalls the pipeline, leaving just the idle power. Finally, the register file profile is of significant interest. There are clearly points where the power rises drastically for time-frames in the range of 100ms. For example, we see a a peak register file power of 136mW at 52s. This would lead to a localized hot spot in the register file. However, a corresponding spike is not visible in the core 0 profile on Figure 4a because of the decrease in the pipeline and I-cache power profiles at that point. For example, at 52s core 0 is roughly 420mW which falls within the average range. The value of our emulation framework is apparent in this scenario. Unlike software simulators, the emulator runs fast enough to observe these sudden variations. At the same time, unlike runtime monitoring approaches on commercial processors, we have direct access to accurate component power profiles. We thus identify hotspots like in the register file which would otherwise remain undetected when looking at purely composite core power.

6. CASE STUDY: ACTIVITY MIGRATION

We now demonstrate our platform's capabilities in a case study on power density-based activity migration (AM). Dynamic thermal management (DTM) has been shown to be effective in limiting peak temperatures [7, 13]. Among many proposed DTM approaches, AM successfully counters local hot-spots within cores [7, 12, 13]. While effective at tackling thermal imbalances, there is an associated performance penalty. As such, AM offers a rich area of research for CMPs.

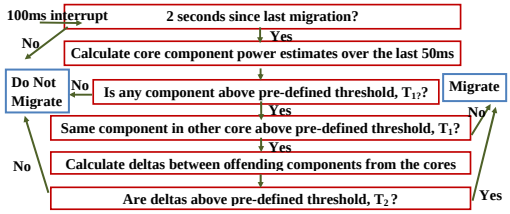


Figure 5: Scheduler algorithm for power-based migration as implemented in Linux 2.6 kernel

The use of our emulation environment for AM studies is motivated by a couple of observations. First, on-chip temperature hot-spots and power profiles are highly correlated with the utilization of individual architectural units [7]. Therefore, migration policies are highly dependent on accurate runtime power/thermal estimates of modules such as caches, register files etc. Extraction of this information is an intrinsic benefit of our infrastructure.

Second, on-chip temperatures experience rise and fall times in the order of hundreds of milliseconds [7, 14]. Our emulation scheme provides a natural alternative to software simulators which would take much longer to profile these timeframes. Furthermore, operating systems such as Linux are prime candidates for managing AM schemes due to their scheduler ticks in the tens of milliseconds range. In this vein, our framework is fast enough to handle full OS effects.

While our studies are currently power-based, we intend extending this work to incorporate temperature models for thermal migration. However, the benefits of our system are clearly assayed from our preliminary power-based studies.

6.1 Scheduling Algorithm

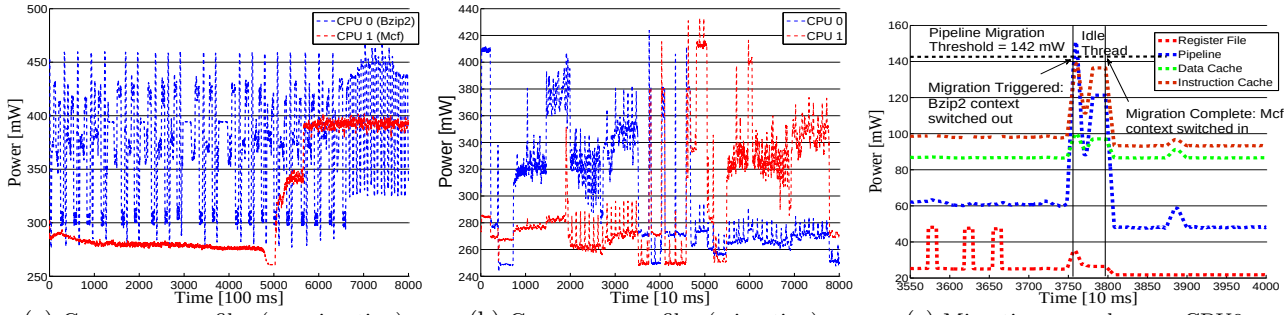
It is now possible to use the power profiling infrastructure to implement a power-aware task scheduling policy. The scheduling algorithm used is detailed in Figure 5. We instrument a 100ms hardware interrupt to initiate the migration scheduler. This timeframe is based on thermal rise and fall times documented in [7]. To equip our policy with some notion of performance, we require a 2s interval before a recently migrated process becomes a candidate for migration again. This interval is chosen because migration takes 300ms on average. This allows up to a 15% performance penalty due to migration overhead (discounting the impact of cold caches after migration). The 300ms required for migration is heavily dependent upon our 65MHz configuration and 4KB caches. As we scale these numbers, we expect lower migration times.

If the required time has elapsed, the component powers are computed over the last 50ms interval. This corresponds to a 5 sample window of the 10ms sample buffer. If a component dissipates power above a pre-defined threshold during this window, it becomes a candidate for migration. Should the same component in the other core also be above the threshold, the difference between the corresponding power numbers has to be higher than a second threshold for migration to take place. This prevents one high power process replacing another with an associated performance cost. The alternate case (an offending component in a single core) is trivial and immediately activates the migration procedure. When multiple component types are above the threshold, migration is allowed should any one of the components conform to the stated requirements.

As a final note, the algorithm favors a swap operation if there are processes present on both cores. This is consistent with the goal of maintaining performance while striving for power-aware scheduling. From here on, our discussion will assume a swap-based migration without loss of generality.

6.2 Kernel Modifications

We modify the Linux kernel by adding a 100ms interrupt and corresponding handler. The handler uses the scheduling algorithm discussed to decide on potential migrations. Per-core runqueue data structures maintain a list of the active and expired processes for their corresponding cores. We change the runqueues to include a flag that the handler sets should migration be initiated. These flags are checked when the main kernel scheduler is invoked. We add code in the kernel scheduler that forces the migration processes to be



(a) Core power profiles (no migration) (b) Core power profiles (migration) (c) Migration procedure on CPU0

Figure 6: `Bzip2` and `Mcf` on the emulator: initial inter-core power imbalance is corrected by activity migration

context switched out of each processor. This guarantees a safe swap. Finally, a custom routine to swap task descriptors between runqueues completes the migration.

6.3 Results

We have successfully tested our scheduling policy on a number of benchmarks. Here we focus on migration while running `Bzip2` and `Mcf` from SPEC 2006. Figure 6a illustrates the power profiles of `Bzip2` on core 0 and `Mcf` on core 1. While `Bzip2` experiences a maximum fluctuation of 180mW periodically, `Mcf` has phases of low power (around 280mW) prior to phases when the power increases close to 350mW followed by 390mW. This behavior is due to a high memory access count which results in low initial power. Once most of the working set is cached, computation proceeds, leading to the higher power profile. Due to the vastly different power characteristics of the two benchmarks, there is ample scope for better inter-core power distribution from migration.

While migration due to hotspots on any component is easily achievable by our scheduling algorithm, we will focus here on migration due to power surges on the integer pipeline. Figure 6b displays the first 80s of the runtime shown in Figure 6a with the power-based migration policy activated. Clearly, the power is now well balanced between the cores. Once again, core 0 starts with power-hungry `Bzip2` while core 1 runs `Mcf`. Migration is activated when the pipeline power of core 0 overshoots the pre-defined threshold around 37.6s into execution. In response, `Bzip2` and `Mcf` are swapped and core 1 begins to dissipate more power while core 0 cools off.

Figure 6c zooms into the exact point of migration on core 0. Here, the pipeline power surges above the 142mW threshold triggering the detection of a likely hotspot. `Bzip2` is immediately context switched out and the idle thread is switched in. Although not shown, there is a corresponding replacement of `Mcf` by the idle thread on core 1 at the same time. The task swap between the core runqueues is then accomplished. Approximately 300ms later, the newly swapped processes are context switched in, lowering the power on core 0.

Our emulation setup enjoys unique capabilities even for this simple experiment. The nuances of migration can be studied on complex workloads interacting with an OS. Furthermore, adaptive policies can readily be developed to cope with hotspots because of accurate component powers. Thus, this platform affords efficient, yet credible, studies of architectural and software solutions to power/thermal issues.

7. CONCLUSIONS

Our power emulation framework offers a novel early-stage alternative to software simulators and runtime monitoring. While it enables hardware speeds and fast data collection (up to 35× speedup over software), it retains software's flexibility in instrumenting event probes, incorporating new design proposals, and detailing accurate component power breakdowns (within 10% of gate-level simulators). As such, this presents a viable platform to carry out early-design-stage power and thermal studies on complex future microprocessors. As shown by the case study, our emulator can be used to study various DTM policies such as activity migration on complex workloads with operating system effects. Moreover, because they too follow Moore's Law, FPGAs will become more powerful as processors become more complex. Thus, they will remain useful for effective modeling for CMPs with large core counts and complicated functionality.

8. ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback. This work was supported in part by the Gigascale Systems Research Center, funded under the Focus Center Research Program, a Semiconductor Research Corporation program. In addition, this work was supported by the National Science Foundation under grant CNS-0720561.

9. REFERENCES

[1] Gaisler research. GRLIB IP core user's manual. Nov 2007.
[2] F. Bellosa. The benefits of event-driven energy accounting in power-sensitive systems. *Proc. 9th ACM SIGOPS European Wkshp.*, 2002.
[3] C. Bienia et al. The PARSEC benchmark suite: Characterization and architectural implications. *Princeton Univ. Tech. Rep.*, (TR81108), Jan 2008.
[4] D. Brooks et al. Wattch: A framework for architectural-level power analysis and optimizations. *Proc. 27th Intl. Symp. on Computer Architecture*, June 2000.
[5] C. Chang et al. A high-end reconfigurable computing system. *IEEE Design and Test of Computers*, 2005.
[6] M. Chidester and A. George. Parallel simulation of chip-multiprocessor architectures. *ACM Trans. on Modeling and Computer Simulation*, 12(3):176–200, July 2002.
[7] J. Choi et al. Thermal aware task scheduling at the system software level. *Proc. 2007 Intl. Symp. on Low Power Electronics and Design*, 2007.
[8] E. Chung et al. PROTOFLEX: FPGA-accelerated hybrid functional simulator. *Computer Architecture Lab at Carnegie Mellon (CALCM) Technical Report*, (2007-2), Feb 2007.
[9] G. Contreras and M. Martonosi. Power prediction for Intel XScale processors using performance monitoring units. *Proc. 2005 Intl Symp. on Low Power Electronics and Design*, 2005.
[10] D.Atienza et al. A fast HW/SW FPGA-based thermal emulation framework for multiprocessor system-on-chip. *Proc. 43rd Conf. on Design Automation*, July 2006.
[11] N. Dave et al. Implementing a functional/timing partitioned microprocessor simulator with an FPGA. *Wkshp. on Architecture Research using FPGA platforms*, Feb 2006.
[12] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. *Proc. 33rd Intl. Symp. on Computer Architecture*, 2006.
[13] S. Heo et al. Reducing power density through activity migration. *Proc. 2003 Intl. Symp. on Low Power Electronics and Design*, 2003.
[14] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: methodology and empirical data. *Proc. 36th Intl. Symp. on Microarchitecture*, Dec 2003.
[15] J.Coburn et al. Power emulation: A new paradigm for power estimation. *Proc. 42nd Conf. on Design Automation*, June 2005.
[16] R. Joseph and M. Martonosi. Runtime power estimation in high performance microprocessors. *Proc. 2001 Intl. Symp. on Low Power Electronics and Design*, 2001.
[17] R. Mahajan et al. The evolution of microprocessor packaging. *Intel Tech. Jnl.*, 2001.
[18] M. Martin et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, Sept 2005.
[19] M.Ghodrat et al. Accelerating system-on-chip power analysis using hybrid power estimation. *Proc. 44nd Conf. on Design Automation*, June 2007.
[20] K. Oner et al. The design of RPM: An FPGA-based multiprocessor emulator. *Proc. 3rd Intl. Symp. on Field Programmable Gate Arrays*, Feb 1995.
[21] K. Skadron et al. Temperature-aware microarchitecture. *Proc. 30th Intl. Symp. on Computer Architecture*, June 2003.
[22] N. Vijaykrishnan et al. Energy-driven integrated hardware-sotware optimizations using SimplePower. *Proc. 27th Intl. Symp. on Computer Architecture*, 2000.
[23] R. Viswanath et al. Thermal performance challenges from silicon to systems. *Intel Tech. Jnl.*, 4(3):16, 2000.
[24] J. Wawrzynek et al. RAMP: A research accelerator for multiple processors. *Tech. Rep. UC Berkeley, EECS-2006-158*, Nov 2006.