XTREM: A Power Simulator for the Intel XScale® Core

Gilberto Contreras, Margaret Martonosi
Department of Electrical Engineering
Princeton University
(gcontrer,mrm)@princeton.edu

Jinzhan Peng, Roy Ju, Guei-Yuan Lueh
Microprocessor Technology Lab
Intel Corp.
(paul.peng,roy.ju,guei-yuan.lueh)@intel.com

ABSTRACT

Managing power concerns in microprocessors has become a pressing research problem across the domains of computer architecture, CAD, and compilers. As a result, several parameterized cycle-level power simulators have been introduced. While these simulators can be quite useful for microarchitectural studies, their generality limits how accurate they can be for any one chip family. Furthermore, their hardware focus means that they do not explicitly enable studying the interaction of different software layers, such as Java applications and their underlying Runtime system software.

This paper describes and evaluates XTREM, a power simulation tool tailored for the Intel XScale microarchitecture. In building XTREM, our goals were to develop a microarchitecture simulator that, while still offering size parameterizations for cache, TLB, etc., more accurately reflected a realistic processor pipeline. We present a detailed set of validations based on multimeter power measurements and hardware performance counter sampling. Based on these validations across a wide range of stressmarks, Java benchmarks, and non-Java benchmarks, XTREM has an average performance error of only 6.5% and an even smaller average power error: 4%. The paper goes on to present a selection of application studies enabled by the simulator. For example, presenting power behavior vs. time for selected embedded C and Java CLDC benchmarks, we can make power distinctions between the two programming domains as well as distinguishing Java application (JITted code) power from Java Runtime system power. We also study how the Intel XScale core's power consumption varies for different data activity factors, creating power swings as large as 50mW for a 200Mhz core. We are planning to release XTREM for wider use, and feel that it offers a useful step forward for compiler and embedded software designers.

LCTES'04, June 11–13, 2004, Washington, DC, USA.

Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors — Runtime environments; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems.

General Terms

Measurements, Performance, Experimentation, Languages.

Keywords

XScale, XORP, Power Measurements, Power Modeling, Java, Hardware Performance Counters

1. INTRODUCTION

Recent years have seen a proliferation of embedded devices in many aspects of life, from cell phones to automated controllers. Each new generation of embedded devices includes new features and capabilities, which are made possible by the greater data processing speeds of embedded microprocessors and larger data storage capacities of RAM and FLASH chips. For device designers and software engineers however, these attractive features can mean more challenging power and thermal design issues, resulting in added complexity and design/test time of target applications.

Under such constraints, understanding the power consumption of running software during the first design stages is of extreme importance. This is because knowing power consumption can help realize early power/performance optimizations. This scenario, in reality, is difficult to achieve since software performance and power consumption are very dependent on implementation details of the processing device, which at early stages might not be fully defined. Furthermore, with complex software platforms like Java Runtime systems running on embedded devices, the task of understanding power and performance becomes even more challenging. This is where high-level design tools come into play.

Research in low-power architectures and energy-efficient programming techniques has led to an extensive suite of high-level tools with the purpose of estimating power and performance characteristics of existing and theoretical microprocessor designs. Some of these tools have been used to estimate power consumption of high-end superscalar processors while others have been used to investigate the effects of software transformations on power consumption [2][13][17][20]. Existing power estimation tools generally offer great flexibility in their usage and configuration since they do not model a particular pipeline implementation. Furthermore, they are mostly aimed at studying C and assembly-based

benchmarks; we are more interested in Java applications and Java Runtime systems.

This paper introduces XTREM, a microarchitectural functional power simulator for the Intel XScale core. XTREM is a powerful infrastructure capable of providing power and performance estimates of software applications designed to run on Intel XScale technology-based platforms. XTREM models the effective switching node capacitance of various functional units inside the core, following a similar modeling methodology to the one found in [2].

The entire XTREM infrastructure has been tailored for the Intel XScale core and validated against real hardware for improved accuracy, yet it has been kept flexible enough to be used during the first design and exploration stages of software and hardware design ideas. We were able to obtain a 4% average error on power estimates and an error of less than 7% on average performance (IPC) across a diverse set of nine different benchmarks composed of C-based embedded benchmarks and Java CLDC applications.

Cycle-level simulation of Java applications by our base simulator is made possible through added system calls and soft-floating point support. This adds an extra dimension of analysis to XTREM since it is possible to analyze separately power and performance characteristics of the JVM and Java application code (JITted code). In fact, detailed knowledge of the JVM memory map allows performance/power analysis of individual JVM phase components like the class loader, the executing engine, the JIT compiler and/or the garbage collector. (Due to space constrains however, the details of this analysis are deferred to future work.) For tested CLDC Java benchmarks, we observed that power usage between JITted code and JVM code is quite varied, as our simulation results show the JVM can consume as little as 14% and as much as 70% of the total average power.

This paper makes the following contributions to the area of embedded power estimation and performance analysis tools:

- We have developed power models for the various functional units of the Intel XScale core.
- Our tools enable one to run a JVM on top of a functional simulator, which allows a much broader application analysis of many important Java embedded applications.
- Our simulation framework is the first to support deeper application-aware analysis — such as distinguishing execution of Java JITted application code and Java Runtime system procedures.
- Having developed a data acquisition methodology consisting of physical multimeter measurements, Hardware Performance Counter (HPC) statistics and simulation results, we present a broad comprehensive analysis of Java and Non-Java systems.

This paper is organized as follows: Section 2 begins with a brief description of the Intel XScale microarchitecture, detailing some architectural features that make the Intel XScale core unique. Section 3 describes the heart of XTREM: Sim-XScale, a microarchitectural functional simulator that models the pipeline structure of the Intel XScale core. Section 4 describes various measuring techniques used in the acquisition of necessary run-time information for the study of the Intel XScale core and validation of XTREM's power models. Section 5 and 6 describe in detail performance and power modeling validation results for various Java and Non-Java applications. Section 7 describes related work and highlights existing differences between XTREM and other power estimation tools. Future work is described in section 8. The summary for our work can be found in Section 9.

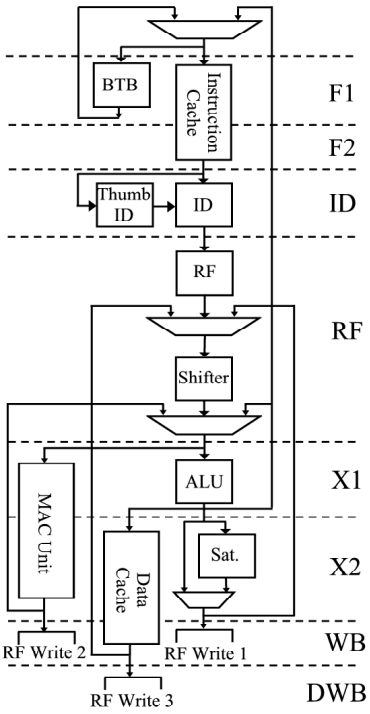

Figure 1: Block diagram of the Intel XScale microarchitecture pipeline.

2. THE INTEL XSCALE CORE

The Intel XScale core is a high-performance, low-power microarchitecture specifically targeted for embedded applications [9]. It is compatible with the ARMv5TE instruction set and includes support for eight new DSP instructions that take advantage of a fast DSP coprocessor.

The Intel XScale core is a seven to eight stage (depending on the type of executing instruction) single issue superpipelined microprocessor with many architectural features that make it suitable for general purpose embedded applications. Figure 1 shows the Intel XScale microarchitecture pipeline organization.

Among the most characteristic features of the Intel XScale core we find a 32KB 32-way set associative instruction cache and a 32KB 32-way set associative data cache. Access to data and instruction caches is distributed between two pipeline stages. The first access stage is dedicated for address TAG comparison and verifying memory access permissions, which are stored in a 32-entry fully-associative Translation Lookaside Buffer (TLB). The second stage is spent retrieving data from the cache. This two-cycle cache access distribution allows the Intel XScale core to be clocked at faster rates than previous ARM cores.

As illustrated in Figure 1, instruction decoding and register file data access are performed in separate stages, as opposed to a unified decode-read stage commonly found in other ARM devices [15]. The decoding engine of the Intel XScale core supports 32-bit ARMv5 and 16-bit ARMv5T Thumb instructions by including a special decoding unit that expands 16-bit instructions into 32-bit instructions. This assists devices with a very limited amount of memory since 16-bit instructions can yield more compact program code. A 128-entry direct mapped Branch Target Buffer (BTB) with a 2-bit branch predictor is included in the Intel XScale microarchitecture to improve performance.

High clock rates achieved by the Intel XScale core come at the expense of increasing main memory access latency. To alleviate this problem, the Intel XScale core includes two specialized data buffers called the fill buffer and the write buffer that sit between the processor's core and main memory. The 32-byte, four-entry fill buffer is responsible for sending and receiving all external memory requests, allowing up to four outstanding memory request before the core needs to stall. The coalescing 16-byte, eight-entry write buffer captures all data write operations from the core to external memory, storing data temporarily until the memory bus becomes available.

The architects of the Intel XScale core have added support for demanding DSP applications by including a 40-bit Multiply-Accumulate (MAC) unit. The MAC is a variable-latency, high-speed, low-power multiply unit. It takes two to five clock cycles to complete an operation depending on instruction type and data width. Intel XScale technology engineers have also extended memory page attributes of the microprocessor memory management unit to enhance memory-caching dynamics. Memory pages can be configured to be non-cacheable, cacheable by the data cache or cacheable by a 2KB 32-way set associative mini-data cache.

We have integrated many of the above architectural features into Sim-XScale: Data and instruction cache accesses have been split into two stages, buffers assimilating fill and write buffers have been installed into our simulator and our branch predictor has been modified to match the hardware's 2-bit prediction algorithm. Thumb instructions and special memory page attributes are not supported by our simulator since none of our tested benchmarks make use of these features. The addition of supported microarchitectural features into Sim-XScale provided us with an accurate simulator that reports an average performance error of less than 1% for micro-kernels and an average error of less than 7% for our tested set of benchmarks as described in Section 5.

2.1 Performance Counters

The Intel XScale core includes two specialized 32-bit registers, CNT0 and CNT1, that can be configured to monitor and count any of the 14 possible performance events shown in Table 1. These performance counters are accessible in privileged OS mode and only two events may be monitored at a time. A third specialized 32-bit register, CLKCNT, is triggered on every clock cycle and its value can be used in conjunction with CNT0 and CNT1 to compute interesting performance information that can reveal major performance losses of running applications. Since performance monitoring of software happens during runtime, performance counters are a reliable source of performance data. We use these counters to validate our performance and power models.

Event	Description
0x0	Instruction Cache miss count
0x1	Number of stall cycles for fetch unit
0x2	Data dependency duration count
0x3	Instruction TLB miss count
0x4	Data TLB miss count
0x5	Branch instruction executed
0x6	Misspredicted branch count
0x7	Number of instructions executed
0x8	Number of stall cycles due to buffers full
0x9	Number of times buffers are detected full
0xA	Data cache access count
0xB	Data cache miss count
0xC	Number of write-back events
0xD	Number of times software changed the PC

Table 1: Comprehensive list of performance events for the Intel XScale core.

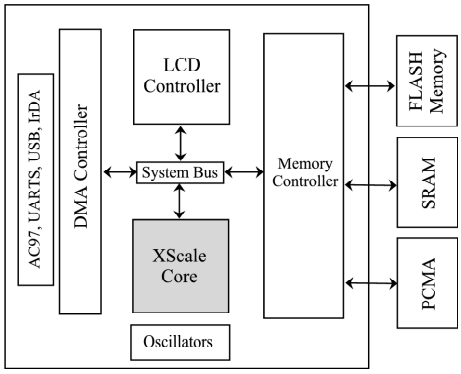

Figure 2: Processor block diagram. The Intel XScale core is surrounded by support functional units that contribute to the processor's overall power consumption. Not all peripheral units are shown in this figure.

3. XTREM POWER AND PERFORMANCE SIMULATOR

XTREM can be divided into two primary components: an accurate microarchitectural functional simulator called Sim-XScale and a set of power models used to obtain power consumption estimates of various functional units of the Intel XScale core on a per-cycle basis.

The methodology for defining the granularity of power distribution (i.e. the number of functional units to model) is not straightforward since the Intel XScale core is embedded inside a complex processor composed of various peripherals that interface to the external world. For example, Figure 2 shows a block diagram of the Intel XScale PXA255 processor showing the spatial placement of the main core within the processor. This work focuses on power behavior of the main processing core and less so on the energy usage of the external components like UARTs, LCD drivers and PCMCIA drivers. It is, of course, to some extent impossible to ignore the effects of these units when power sampling is done on the processor in real time. We minimize these effects by turning off as many unused peripherals as possible.

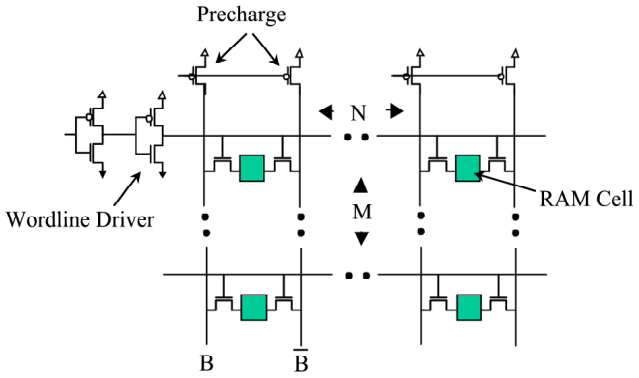

Figure 3: RAM array schematic. Transistor-level schematics like the one shown here are used to create mathematical equations that estimate internal node switching capacitance.

The starting point in the construction of power models is a detailed description of the Intel XScale technology and microarchitecture as described in [4]. This paper describes with great detail logic-level implementation of caches and clock distribution logic, among other important functional blocks. From this lower-level description and available Intel XScale technology documentation, an initial set of power models using Wattch [2] power models as templates were created. Some of Wattch's power models have been adapted into XTREM with minor changes, while other power models were adjusted and revised to reflect more accurately the Intel XScale microarchitecture and new technology implementations (in the case of memory arrays and caches).

Power models are mathematical equations that provide node switching capacitance estimates. We constructed power equations based on transistor-level schematics of functional units and a high-level view of transistor gate and drain capacitances. A single equation does not describe an entire functional unit, but rather basic sub-blocks that can be reused. For example, the register file unit is sub-divided into a row decoder, an SRAM array and pre-charge logic.

Equations 1 and 2 are mathematical model equations that estimate the internal node capacitance of bitlines and wordlines in an SRAM array. Figure 3 shows the RAM array schematic model used to derive these equations. Equations 1 and 2 are given as an example of the analysis and modeling detail used in the construction of power models for various functional units.

$$C_{bitline} = C_{diff}(PreCharge) + C_{diff}(CellAccess) * M + C_{metal} * BitLineLength \quad (1)$$

$$C_{wordline} = C_{diff}(WordLineDriver) + C_{gate}(CellAccess) * N + C_{metal} * WordLineLength \quad (2)$$

Not all necessary power models for the Intel XScale core were found in Wattch. For example, the unique T-shaped clock structure common in the Intel XScale core, for example, had to be created based on [4] since Wattch power models assume an H-tree clock distribution network.

3.1 The Sim-XScale Functional Simulator

Sim-XScale is in part derived from the ARM-SimpleScalar simulator [18], a highly flexible microarchitectural-level simulator with a five-stage superscalar-like pipeline organization. The ARM-SimpleScalar simulator provides reasonably good accuracy for many applications, but for some stressmarks with heavy emphasis on the memory subsystem, significant differences can be observed between the IPC it predicts and that measured by the hardware performance counters of the Intel XScale core. This large performance error is primarily caused by the pipeline and memory sub-system differences between the general ARM-SimpleScalar microarchitecture and that of the Intel XScale microarchitecture.

Sim-XScale includes architectural features not available in ARM-SimpleScalar such as fill and write buffers, a 4-entry pend buffer, a revised version of the well-known 2-bit branch predictor algorithm and a read/write cache-line allocation policy. Architectural features such as these make Sim-XScale more closely-matched to the microprocessors we target.

In the same way we implemented new units into Sim-XScale we also removed microarchitectural units inherited from ARM-SimpleScalar that have no parallel with a true Intel XScale microarchitecture. For example, the Register Update Unit common to many SimpleScalar simulations is not present in our framework.

Sim-XScale allows monitoring of 14 different functional units of the Intel XScale core: Instruction Decoder, BTB, Fill Buffer, Write Buffer, Pend Buffer, Register File, Instruction Cache, Data Cache, Arithmetic-Logic Unit, Shift Unit, Multiplier Accumulator, Internal Memory Bus, Memory Control and Clock.

3.2 Sim-Xscale JVM Simulation Support

To support research on power and performance of Java Runtime system for embedded devices, Sim-XScale gives researchers the ability to run complex Java Runtime systems like Sun's KVM reference CLDC design [11] or Intel's XORP[1] JVM. It is required, however, that the JVM binary be compiled using the `-static` linking flag, meaning no dynamic libraries can be used. We have used a statically linked XORP JVM for all the experiments presented in this paper.

A dynamically linked XORP JVM (designed to run on top of an OS) employs a one-to-one Java thread to native thread policy, which means that each Java thread is mapped to a native thread. In order for us to be able to run a statically linked XORP JVM directly on top of our functional simulator without emulating an OS, we have to remove multi-thread support and thread synchronization from the JVM. The direct implication of this modification is multi-threaded Java applications cannot run using our statically linked (modified) JVM. We hope to add multi-thread support to our infrastructure in the near future by either emulating an OS between our simulator and the JVM or by mapping M Java threads into a single native thread.

Our "static" XORP should not affect the performance of the running Java application significantly. This was verified by comparing hardware performance counter statistics from various Java applications using a modified "static" link and

[1]The XORP JVM is a clean-room Runtime system designed specifically for high performance and small memory footprint. At current development stage, XORP has full-fledged support for J2ME CLDC/CDC on XScale platforms.

an unmodified "dynamic" link of the XORP. For four CLDC Java benchmarks, our experiments showed an average difference of 1.78% between hardware performance counter values. Physical power measurements between the two XORP configurations are also very similar, with an average difference of 0.21% across the same set of four CLDC Java benchmarks.

4. MEASURING TECHNIQUES FOR XTREM VALIDATION

4.1 The DBPXA255 Development Board

Our testing equipment consists of the DBPXA255 development board [10] powered by the PXA255 processor running the Linux 2.4.19-rmk7-pxa1 patched kernel. The DBPXA255 is a multi-purpose development board that includes many of the device features commonly found in embedded devices such as SRAM and FLASH memory banks, a LCD touch screen, ethernet adapter and keyboard.

The DBPXA255 development board does not include a tertiary storage device like a hard disk or a CD-ROM. Since it is infeasible to store all benchmarks and their associated data sets on the board's limited flash memory, a network connection was set up between the development board and a host PC.

The DBPXA255 board has two sets of jumpers that facilitate voltage and current measurements of hardware. The first set allows the user to tap into the main power supply of the processor. The second set exposes the CPU's memory bus voltage pins. We use the Agilent 34401A digital multimeter to sample current consumption of the PXA255 processor while running our selected set of benchmarks. Voltage can be measured in a similar way, but in all measurements the voltage remained nearly constant throughout our experiments, so we performed our calculations assuming using a constant voltage of 1.5V. Since voltage is assumed to remain constant, we only sample current, which is then multiplied by voltage to get a power figure as defined by the well-known equation $P = V \cdot I$.

We set our development board to use a core frequency of 200Mhz, a bus clock frequency of 100MHz and memory frequency of 100Mhz. The microprocessor core's voltage was adjusted to 1.5V and all I/O pins were driven by a 3.3V power supply.

4.2 Runtime Power Sampling

We performed power sampling at runtime of various test benchmarks also using the Agilent 34401A digital multimeter. The digital multimeter interfaces via a GPIB cable to a PC. A GPIB interface allows us to obtain sampling rates of up to 1000 samples per second. Figure 4 shows the the physical measurements setup. In order to better focus on the microarchitectural core, we turned off various unused peripherals like the LCD clock driver, UART clock drivers and the AC97 clock driver. Turning these components off reduced idle power consumption by as much as 7 to 8mW (out of a total of roughly 300mW) for the 200Mhz and 1.5V processor setup.

4.3 Runtime Performance Sampling

In addition to power sampling via multimeter, we also used hardware performance counters for simulator validation. We interfaced to the hardware performance counters

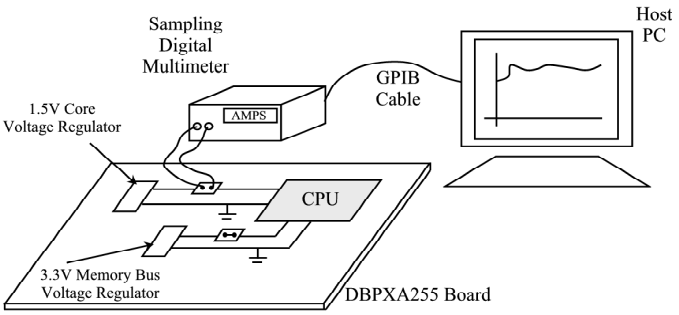

Figure 4: Physical measurements setup. The Agilent 34401A Digital Multimeter (DMM) is used to sample current consumption of the Intel PXA255 CPU. High sampling speeds are achieved through a GPIB connection between the DMM and a host PC.

using a Loadable Kernel Module (LKM) that adds system calls to the Linux OS kernel. These additional system calls are used to configure and read hardware performance counters. Each call is no more than three assembly instructions long, keeping performance overhead low. Overflow of performance counters is detected by a special bit location on the performance's counters configuration register. Our LKM is based on a similar performance counter reader previously designed for Pentium 4 systems [3].

Performance sampling is done by `avgsample`, a C program that creates two working threads. The first thread runs the target program that we wish to measure. The second thread is the main sampling thread. The sampling thread is in charge of clearing and activating hardware performance counters before the target program thread runs. Once the target application thread is running, the sampling thread waits until the application thread exits and immediately reports results to the user. Counter overflow is automatically detected by an interrupt service routine, which adjusts counter values beyond 32-bit accuracy when needed.

5. XTREM PERFORMANCE VALIDATION

Performance counters are very helpful in analyzing runtime performance of applications as well as for exposing different memory system and pipeline latencies of the intel XScale core. Their versatility and accuracy have proven invaluable in validating Sim-XScale. To this end we compared performance counter results and equivalent performance metrics reported by Sim-XScale for several stressmark programs. Stressmark are highly predictable programs written to validate the pipeline structure of our simulator. We wrote nine stressmarks as described below.

`Dcache` is a small kernel that works with a data set that fits entirely in the data cache, thus minimizing data cache misses. `Dcache_trash` works with a data set that extends beyond the capacity of the data cache. This stressmark has been designed to have a very large number data cache misses, ideally a cache miss for every loop iteration. `Dcache_write` stresses data throughput to main memory. The main loop writes consecutive integers into memory. The purpose of this kernel is to stress the memory coalescing feature of the Intel XScale core. `Dcache_writeline` is very similar to `dcache_write`, except that `Dcache_writeline` writes one in-

teger in every memory address location corresponding to the beginning of one cache line so that memory address coalescing is not possible. `Dcache_twriteline` and `dcachet_twrite` are similar to `dcache_writeline` and `dcache_twrite` except that the working data set does not fit entirely in the cache. `Mult_dep` and `mult_nodep` are two kernels that exercise and measure the latency of the multiplier when data dependencies exist and when they are absent, respectively. `ADD` stresses the arithmetic unit of the Intel XScale core. The `ALU` benchmark has a very high IPC since no data dependencies exist. The `BTB` stressmark has been designed to stress the Branch Target Buffer by implementing two mutually exclusive inner branches that miss on every loop iteration. `Matrix` is a small benchmark that multiplies an n by n array. It has been constructed to measure the overall quality of our simulator.

Figure 5 shows our stressmark validation results by comparing the IPC reported by Sim-XScale and the IPC computed from hardware performance counter readings. The average IPC difference for the nine stressmarks is less than 1%.

A 1% average performance error in stressmark testing is great news for our microarchitectural simulator, but in order to further quantify the accuracy of Sim-Xscale we need to employ more realistic, complex benchmarks. For this end we have selected five benchmarks from Mibench[8], an embedded benchmark suite developed by the University of Michigan. The benchmarks are: `JPEG_compress`, `Bitcount`, `CRC`, `SHA` and `Dijkstra`. These benchmarks were chosen based on their large percentage of CPU time and work done in user space. We also wanted to test how well our simulator is able to track hardware performance for Java applications. `Jzlib` [12], `Crypto` [14], `GIF` [7] and `REX` [1] are the four open Java CLDC benchmarks selected for this task. These four Java benchmarks are similar to the set of Java CLDC benchmarks created by EEMBC [5]. `Jzlib` is a ZLIB implementation in pure Java, `Crypto` is a Java implementation of the cryptographic algorithms, `GIF` is a GIF-format picture decoder and `REX` is a Java implementation for regular expressions.

Figure 6 shows performance results for Mibench and Java CLDC benchmarks in the form of an IPC comparison graph. The white bar in Figure 6 corresponds to the IPC reported by Sim-XScale; the gray bar corresponds to IPC derived from hardware performance counters. As seen from the figure, Sim-XScale provides reasonable performance accuracy for complex benchmarks, reporting a maximum IPC difference for `CRC` of 14.2% with respect to hardware-measured IPC. This relatively large error is caused by differences between our simulator and real hardware when it comes to decomposing complex instructions (such as the LDMIA and STMIA instructions) into uops. `CJPEG` has the lowest IPC error with less than 1% difference. Average IPC difference for all nine tested benchmarks is 6.5%.

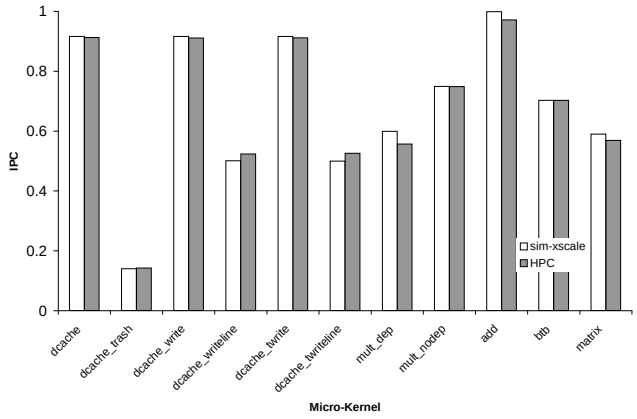

Figure 5: IPC comparison for stressmarks as reported by Sim-Xscale and Hardware Performance Counters (HPC). Sim-XScale simulates adequately many aspects of the Intel XScale microarchitecture.

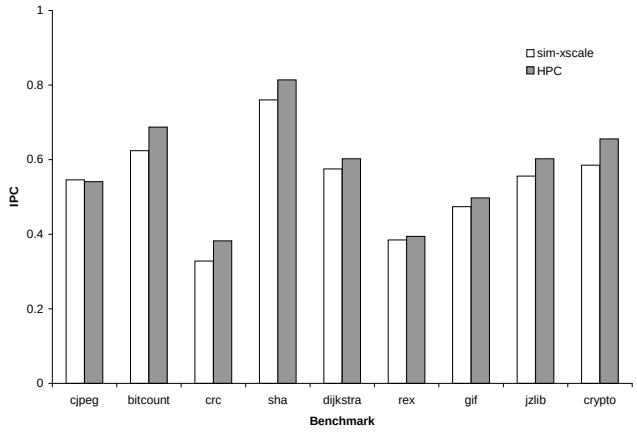

Figure 6: Comparison between simulated and hardware-measured IPC performance for a set of MiBench and open CLDC benchmarks.

6. XTREM POWER VALIDATION

Validation of XTREM power models is a necessary step in the development of a trustworthy power estimation tool. Good performance accuracy is of little importance if we cannot guarantee a tool that also provides power estimates within a tolerable error.

The first step taken toward validation of our power models was to isolate power consumption of individual functional units. Decomposing power usage into various utilized functional units provides the most comprehensive way of validating XTREM power models since model accuracy can be traced to a single functional unit, thus giving us the opportunity to pinpoint individual power models that do not follow the expected power behavior. This, of course, may be difficult or even impossible to accomplish for every functional unit since some units are always used independently of the instruction being executed; functional units like instruction decoder, TLBs and instruction cache are difficult to isolate under normal execution conditions. On the other hand, some functional units can be orchestrated in very predictable ways. These include the register file, ALU, MAC, Data Cache, Fill and Write buffers.

As described in Section 5, our stressmarks have been designed to make specific use of various functional units within the core. Serving once again as validating agents, small stressmarks were used to dissect the processor's power utilization into functional unit power consumption. This methodology assumes that no power is consumed by units that have been turned off or are not being used. This assump-

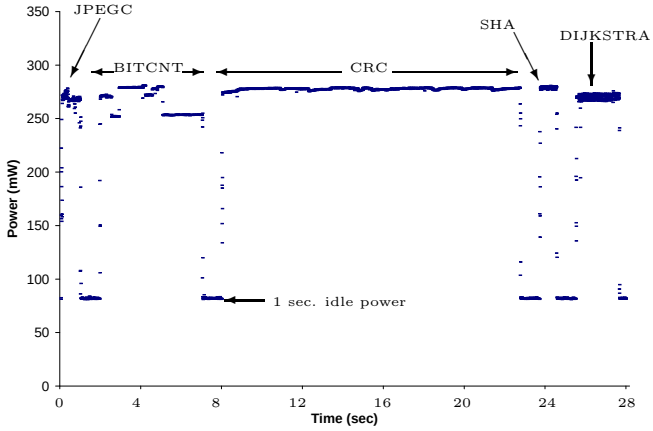

Figure 7: Hardware power sampling results for five MiBench benchmarks. Benchmarks are separated by a 1 second delay seen as 80mW idle time.

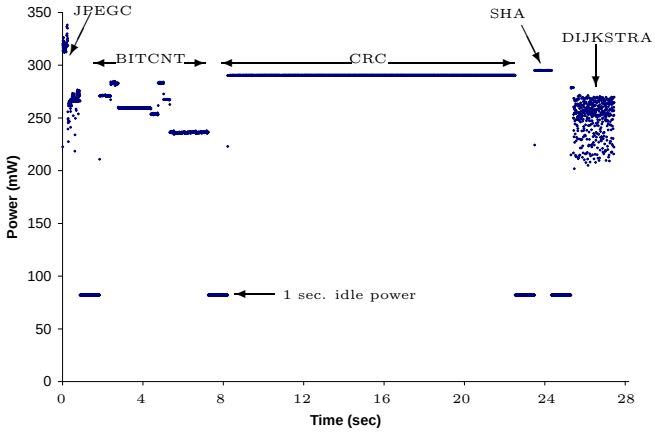

Figure 8: Simulated power traces for five Mibench benchmarks. The maximum error of 11% for `JPEG` and a minimum of 3.66% for `Bitcount` was calculated for average benchmark power consumption.

tion seems plausible since the Intel XScale core is a highly power-efficient core that makes use of multiple levels of clock gating, allowing entire units to be disabled when not in use [4].

Our power-isolation methodology does not guarantee the exact per-functional unit power consumption of the entire core, but rather helps understand how power is distributed across various functional units and how software affects overall power consumption. The second step in power model validation included a second revision of power models to reduce estimation errors discovered during the first validation step.

A third and last step in power model validation involves simulating Mibench and Java CLDC benchmarks. Instead of simply comparing "average estimated power" and "average measure power" as a validation approach, we believe power behavior in the time domain better describes how accurately XTREM tracks real-hardware power behavior.

We start by describing Figure 7, which shows a power vs. time plot for five Mibench benchmarks running on real hardware. The benchmark ordering from left to right is: `JPEG_Compress`, `Bitcount`, `CRC`, `SHA` and `Dijkstra`. Benchmarks are separated from each other by a one-second delay, visible on the graph in the form of 82mW idle power consumption.

Figure 8 shows XTREM's simulated power results assuming the same benchmark ordering as Figure 7. The similarities between Figure 7 and Figure 8 are encouraging: XTREM is able to capture not only power behavior within a benchmark, but also the power relationship among the set of tested benchmarks.

Simulating CLDC Java benchmarks not only helped validate XTREM's CLDC power estimation capability, but also helped revealed many interesting characteristics of Java applications. Figure 9 is a power vs time plot of `REX` Java benchmark running on real hardware. From the graph we can observe that Java applications are characterized by an initialization process where the JVM is allocating the heap and initializing the Runtime system. This JVM initialization phase is visible in the figure in the form of a random-like distribution of points at the start of the plot.

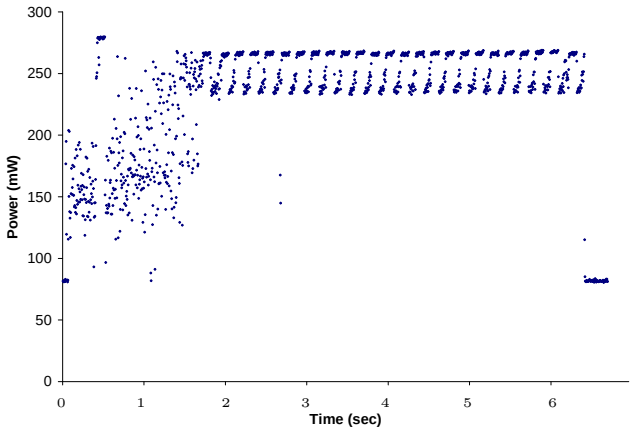

Figure 9: `REX` live power measurements. The start of the benchmark is characterized by very varied power behavior corresponding to JVM initialization.

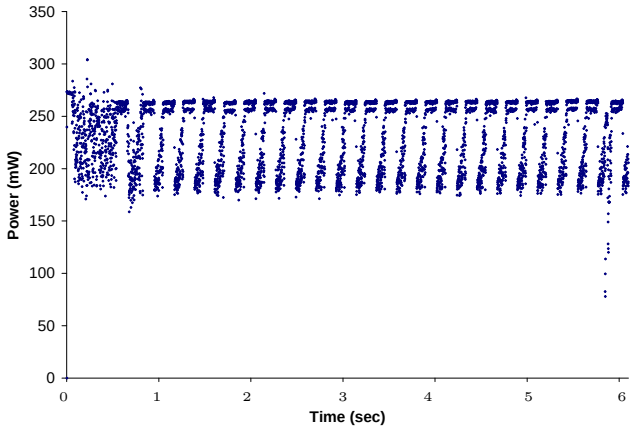

Figure 10: `REX` simulated power measurements.

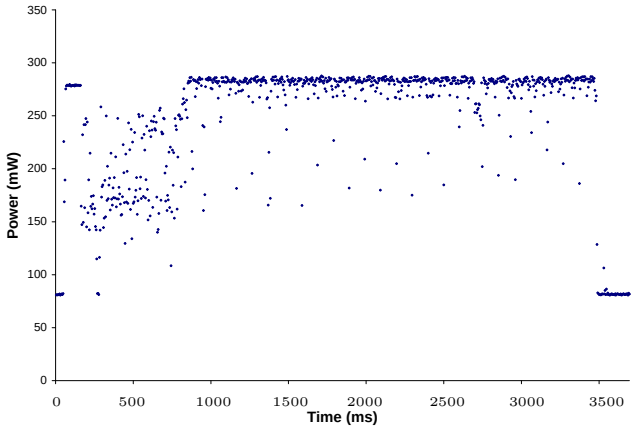

Figure 11: GIF hardware-measured power behavior.

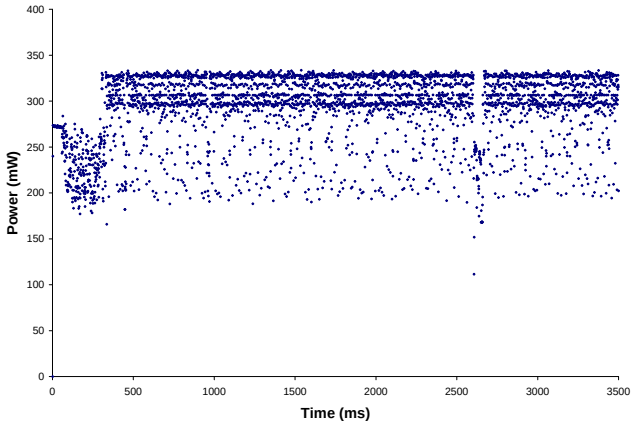

Figure 12: GIF simulated power measurements. A call to garbage collection creates a slight power spike at around 2600 ms.

Figure 10 displays simulated power vs time for REX. This figure was constructed by reporting average power every 200,000 instructions — 4x faster than physical power sampling. Higher sampling rates allow XTREM to capture many low-latency events not visible by the power sampling hardware. An example of this is shown towards the end of Figure 10 in the form of a small power spike. This power spike is a consequence of garbage collection.

Figure 11 and 12 show physical power sampling and simulated sampling results for GIF, respectively. Both plots are described by a JVM initialization phase followed by an almost flat power consumption trace with snowfall-like traces on the bottom. This snowfall-like behavior is more visible in simulation traces as a consequence of higher sampling rates. As with Figure 10, Figure 12 shows the effects of calling the garbage collector, which creates a characteristic power spike two-thirds into the benchmark.

As a summary to our Java CLDC and C benchmark validation experiments, Figure 13 gives a graphical comparison between simulated average power and hardware-measured average power consumption for five Mibench and four Java CLDC benchmarks. Table 2 shows our results in tabular form.

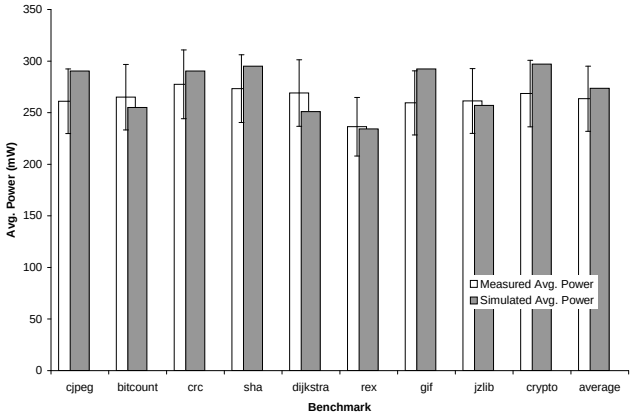

Figure 13: Average power consumption for MiBench and open CLDC Java benchmarks. 12 percent error bars are shown.

	Measured		Simulated	
Benchmark	Avg. mW	Std. Dev.	Avg. mW	Std. Dev.
JPEGC	261.01	28.43	290.37	26.78
Bitcount	265.02	15.25	255.02	16.13
CRC	277.44	5.88	290.37	1.20
SHA	273.26	24.59	295.03	5.16
Dijkstra	269.06	11.85	251.06	17.80
Rex	236.41	39.90	234.33	32.41
GIF	259.49	42.30	292.33	38.13
Jzlib	261.34	19.79	257.05	16.40
Crypto	268.54	22.44	297.03	27.78

Table 2: Average power and standard deviation comparison between hardware-measured power traces and simulated power consumption traces.

XTREM goes a step beyond power estimation. It provides researchers with the ability to dissect Java application power consumption into JVM and non-JVM power, a skill not yet available to conventional power measuring techniques. With the capability of discriminating between JVM and non-JVM power consumption at hand, software designers and architects can better understand how to increase power and performance efficiency of Java Runtime systems during early development stages.

Figures 14 and 15 show the simulated JVM power consumption and the simulated non-JVM power behavior for the REX benchmark, respectively. We previously mentioned the initialization stage of Java benchmarks is characterized by almost pure JVM activity. This is demonstrated in the first 500ms of Figure 15, where non-JVM or "Java application" involvement is minimum during the 500ms time frame. After JVM initialization, JVM power for REX drops to an average of about 75mW and Java application power rises to an average of about 160mW. Division of JVM and non-JVM power introduces an extra dimension in the analysis of "interesting" events. For example, Figure 14 makes it clear that the power spike seen from Figure 10 originates within the JVM Runtime system. For REX, the average power required by the JVM represents 37% of the total average power. The GIF benchmark has 52mW of average power assigned for the

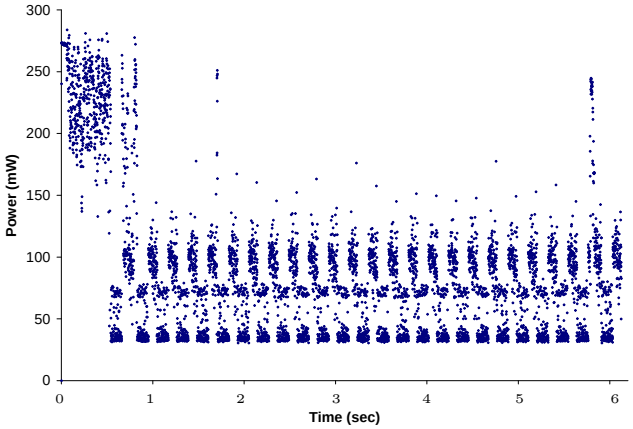

Figure 14: Power consumption of the JVM system while running the `REX` benchmark.

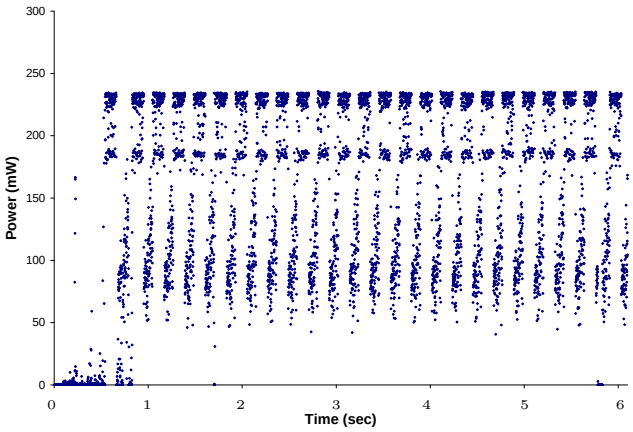

Figure 15: Simulated application (JITted code) power traces for the benchmark `REX`.

JVM, or 18% of the total average. `Crypto`'s JVM consumes 44mW of average power, equivalent to 15% of the total average power. `Jzlib`, on the other and, has more of its total average power consumed by the JVM, with 181mW or 70% of its total average consumed power.

For `Jzlib`, the high percentage of power dedicated to JVM execution is caused by support functions. Support functions, such as integer remainder and integer divide functions, are called within JITted code to perform a specific task, but the actual function execution occurs within the JVM, thus increasing its average power requirement.

Last but not least is XTREM's ability to provide a breakdown of power consumption among various microarchitectural components. When experimenting with novel low-power architectures, it is often necessary to quantify power distribution among the various functional units of the microprocessor. By combining accurate runtime power estimation of benchmarks with functional unit power breakdown, XTREM promises to be a versatile tool in early system design exploration.

Figure 16 is a unit-by-unit power breakdown for `crypto`. Out of the fourteen functional units that XTREM models, only nine units are shown in this figure. These nine units

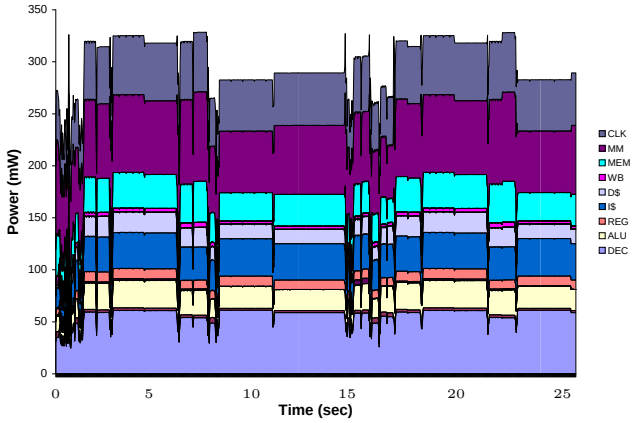

Figure 16: `Crypto`'s functional unit power distribution across time. Nine out of fourteen simulated functional units are responsible for more than 95% of total average power consumption.

account for 94% of the total average power: Memory controller unit (24% total average power), Instruction Decoder (19%), Clock Structure (17%), the Instruction Cache (11%) Memory Bus (10%), ALU (8%) and Data Cache (6%). The rest of the power (6%) is distributed among the rest of the modeled functional units.

6.1 Activity Factor influence on Power Consumption

Power consumption of functional units can be very dependent on input/output data. This power consumption dependency on activity factors is expected since dynamic power, a consequence of charging and discharging of capacitive nodes, encompasses a large percentage of overall power consumption. Knowledge of power dependency on activity factors is of great importance for small portable embedded processors where there is a limited amount of energy available. System designers can, for example, strategically place high access memory regions in such a way that the activity factors for memory address pins, and consequently internal address bus, is reduced, thus saving power and extending battery life.

We once again made use of specific stressmarks to study power consumption's dependence on input data. Activity factor stressmarks expose activity factor dependency of various functional units by performing the same operation on a predetermined input data set. Using known input values allows the functional unit to work with a known data input activity factor, which is defined as the variability of ones and zeros in the binary representation of an input data vector when it transitions from data A to data B. Among the various functional units for which this experiment was realized, the MAC, Data Cache, Shift Unit and ALU, the ALU datapath unit showed the largest activity factor influence on power consumption.

Figure 17 is a power vs time plot of the ALU-datapath activity factor stressmark running on the PXA255 microcontroller. The experiment, as seen from the plot, is divided in two parts. During the first part of the experiment, the A input varies in the vertical direction by increasing the number of '1' of the datum while B changes in the horizontal direc-

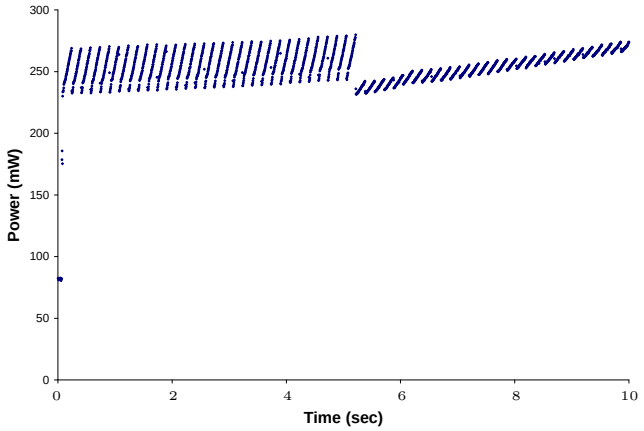

Figure 17: Power dependence on input Activity Factors for the "ADD" operation.

tion in a similar way. For the second part of the experiment the roles of A and B are interchanged. The lower-left corner of the first half of the experiment corresponds to adding 0 + 0. The upper-right corner corresponds to adding two registers with all ones (binary negative one). In between each addition operation of the stressmark, input values are reset to null values in order to increase one-to-zero transitions. It is interesting to observe how interchanging the roles of the A and B inputs changes the power behavior of the experiment.

Figure 17 demonstrates it is possible to create power swings as large as 50mW in the Intel XScale core by carefully adjusting the data activity factor of the ALU datapath. Power swings of double this magnitude are possible in a 400Mhz Intel XScale core.

7. RELATED WORK

Previous research in the area of power consumption for Java systems has been done by Farkas et al. [6]. These authors use a hardware-based approach in the study of power behavior for Java applications. This work analyzes power consumption of Itsy, a pocket computer developed by Compaq based on the StrongARM SA-1100 processor. The study performs live measurements of power consumption for various applications running on the Itsy pocket computer. They also present initial data on Java features such as preloading Java classes, JIT vs non-JIT compilation and multi-JVM support.

A substantial number of studies have focused on simulation techniques for power estimation. Wattch [2] and SimplePower [19] are two infrastructures used to study energy and performance efficiency of microprocessors. Wattch uses mathematical equations to model the effective capacitance of functional units. This flexible tool has been used to estimate power consumption of high-performance microprocessors such as the Pentium Pro, the MIPS R10K and the Alpha 21264. SimplePower is another high-level tool used in the study of system-energy estimation and compiler optimizations effects on the processor's power consumption. Power models for this work are based on analytical power models and energy tables that capture data switching activity.

Research studies using hardware-based measuring techniques have proven to be very efficient in modeling complex architectures. Isci et al. [3] used real-hardware power measurements along with functional unit utilization heuristics derived from hardware performance counters to construct analytical power models for a Pentium 4 processor. Chang et al. [16] performs cycle-accurate energy characterization using the ARM7TDMI microprocessor. For this work, fast energy characterization of hardware is possible using hardware measurement techniques involving charge-transfer measurements. A multidimensional energy characterization is done on the ARM7TDMI processor based on seven energy-sensitive microprocessor factors.

Accurate power modeling for the Intel XScale core requires a more specific hardware description than the approach employed by previous simulation-based tools. XTREM differs from previous work in various ways. First, XTREM is intended to model a specific microarchitecture family. While many of the core's configuration parameters are still user-specified (cache sizes, BTB entries, TLB size), the pipeline structure modeled by XTREM has been designed to closely match the microarchitecture of the Intel XScale core. This provides added performance accuracy over currently existing tools. Second, XTREM has been validated against real hardware using physical power measurements, hardware performance counters and stress kernels, which makes XTREM a reliable performance/power estimation tool for XScale-based systems. Last, increasing popularity for Java-supported devices motivated us to design XTREM to support both C and Java applications. Other tools just focus on C benchmarks and do not support a unified power and performance, Java Runtime system research environment.

8. FUTURE WORK

Future work for this study includes a more in-depth study of power behavior for Java Run-time systems. We wish to quantify the power consumed by various JVM phases (components) such as the Java class loader, the JIT compiler, JITted code execution and the garbage collector phase. Such study can potentially help identify opportunities for energy savings by determining phases with large energy requirements.

Studies for CDC Java benchmarks are also in the near future. CDC benchmarks are more sophisticated benchmarks that behave differently from CLDC applications from both performance and power standpoints. We have done preliminary run-time hardware performance counter measurements and have found such experiments extremely useful for identifying major performance losses. By analyzing simulation and hardware performance counter data, we hope to recommend power-efficient architectural changes to the Intel XScale core that improve the overall performance of both CDC and CLDC Java applications.

We are also interested in identifying opportunities for Dynamic Voltage Scaling (DVS) on XScale-based systems running Java applications. We have implemented a DVS algorithm on the DBPXA255 development board which uses HPC metrics to decide when voltage and frequency changes should take place. We hope to investigate ways to apply these DVS algorithms to re-curring Java phases.

9. SUMMARY

This paper has introduced XTREM, a high-level functional power simulator tailored for the Intel XScale core.

XTREM has been validated using hardware performance counters and real-hardware power measurements. We have presented simulated power results for five MiBench benchmarks and four CLDC Java benchmarks, reporting an average performance error of 6.5% and an average power estimation error of 4%. XTREM is capable of quantifying power requirements for the JVM and non-JVM sections of Java applications, giving software engineers and extra dimension in power analysis. This paper has described how XTREM can help identify "power-hungry" functional units by providing a breakdown of power consumption and how bit-switching activity within the Intel XScale core can produce power swings as large as 50mW for a 200Mhz processor.

The research presented here has provided the tools to obtain a broad and comprehensive view of how modern embedded systems work. Our approach employs a data acquisition methodology consisting of physical power measurements, hardware performance counter statistics and simulation results. This study has focused on Java, C-based embedded and stressmark applications targeted for Intel XScale Technology-based systems. We are planning to release XTREM for wider use. We feel XTREM offers a useful step forward for compiler and embedded software designers as it promises to help explore a broader design space targeted for low energy consumption and high performance.

10. ACKNOWLEDGMENTS

This work was supported in part by an NSF Information Technology Research Grant CCR-0086031 and by SRC contract number 2003-HJ-1121. In addition, we gratefully acknowledge funding and equipment donations from Intel Corp.

11. REFERENCES

[1] Java Regular Expressions. http://www.crocodile.org/~sts/Rex/.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture*, June 2000.

[3] Canturk Isci and Margaret Martonosi. Runtime Power Monitoring using High-End Processors: Methodology and Empirical Data, 2003. MICRO'36.

[4] Clark, L.T.et al. An embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications. *Solid-State Circuits, IEEE Journal o*, 36(11):1599–1608, November 2001.

[5] Embedded Microprocessor Benchmark Consortium. EEMBC Benchmarks for the Java 2 Micro Edition (J2ME) Platform. http://www.eembc.org.

[6] K. I. Farkas, J. Flinn, G. Back, D. Grunwald, and J.-A. M. Anderson. Quantifying The Energy Consumption of a Pocket Computer and a Java Virtual Machine. In *Measurement and Modeling of Computer Systems*, pages 252–263, 2000.

[7] FM Software. GIF Picture Decoder. http://www.fmsware.com/stuff/gif.html.

[8] M. R. Guthaus et al. MiBench: A free, Commercially Representative Embedded Benchmark Suite. July 2001. IEEE 4th Annual Workshop on Workload Characterization,.

[9] Intel Corporation. *Intel XScale Microarchitecture for the PXA255 Processor: User's Manual*, March 2003. Order No. 278796.

[10] Intel DBPXA255 Development Platform for the Intel Personal Internet Client Architecture. *Intel Corporation*, February 2003. Order No. 278701-001.

[11] J2ME Building Block For Mobile Devices: White Paper on KVM and the Connected Limited Device Configuration (CLDC). *Sun Microsystems*, May 2000. http://java.sun.com/j2me/docs/index.html.

[12] Jean-loup Gailly and Mark Adler. Zlib Java Implementation. http://www.jcraft.com/jzlib/.

[13] A. Krishnaswamy and R. Gupta. Profile Guided Selection of ARM and Thumb Instructions, 2002. In ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'02).

[14] Legion of the Bouncy Castle. Bouncy Castle Crypto 1.18. http://www.bouncycastle.org/.

[15] M. Levy. Exploring the ARM1026EJ-S pipeline. http://www.MPRonline.com.

[16] Naehyuck Chang; Kwanho Kim; Hyung Gyu Lee. Cycle-Accurate Energy Measurement and Characterization With a Case Study of the ARM7TDMI [microprocessors]. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2000.

[17] The SimpleScalar-ARM Power Modeling Project. PowerAnalyzer. http://www.eecs.umich.edu/~panalyzer.

[18] The SimpleScalar Toolset. SimpleScalar LLC. http://www.simplescalar.com.

[19] N. Vijaykrishnan. Energy-driven integrated hardware-software optimizations using SimplePower. In *Proceedings of the 27th International Symposium on Computer Architecture*, 2000.

[20] W. Ye, N. Vijaykrishnan, M. T. Kandemir, and M. J. Irwin. The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool. In *Design Automation Conference*, pages 340–345, 2000.