The XTREM Power and Performance Simulator for the Intel XScale® Core: Design and Experiences

GILBERTO CONTRERAS and MARGARET MARTONOSI
Department of Electrical Engineering, Princeton University
JINZHANG PENG and GUEI-YUAN LUEH
Microprocessor Technology Lab, Intel Corp.
ROY JU*
Google

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 and other structures, more accurately reflected a realistic processor pipeline. We present a detailed set of validations based on multimeter power measurements and hardware performance counter sampling. XTREM exhibits an average performance error of only 6.5% and an even smaller average power error: 4%. The paper goes on to present an application study enabled by the simulator. Namely, we use XTREM to produce an energy consumption breakdown for Java CDC and CLDC applications. Our simulator measurements indicate that a large percentage of the total energy consumption (up to 35%) is devoted to the virtual machine's support functions.

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

Additional Key Words and Phrases: Power Modeling, Java, Power Measurements, Intel XScale Technology

This work was supported in part by an NSF Information Technology Research Grant CCR-0086031 and by SRC contract number 2003-HJ-1121. Authors' Address: Department of Electrical Engineering, Princeton University, Princeton, NJ 08540. Email: {gcontrer,mrm}@princeton.edu. *The author contributed to this work while working at Intel Corporation. We gratefully acknowledge funding and equipment donations from Intel Corp.

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 [Brooks et al. 2000][Krishnaswamy and Gupta 2002][The SimpleScalar-ARM Power Modeling Project 2004][Ye et al. 2000]. 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. 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 comprehensive 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 [Brooks et al. 2000].

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-float support. This adds an extra dimension of analysis to XTREM since it is possible to separately analyze 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 such as the class loader, the execution engine, the JIT compiler and/or the garbage collector. This is done for our case study in Section 8. 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% or 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 through the use of a validated cycle-accurate power and performance simulator.

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 microarchitectural functional simulator that models the pipeline structure of the Intel XScale core, which is the heart of XTREM. 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. Sections 5 and 6 describe in detail performance and power modeling validation results for various Java and Non-Java applications by comparing simulation versus hardware measurements. Section 7 discusses the influence of data activity factors on the power consumption of the Intel XScale-based PXA255 microcontroller. Section 8 presents our case study where XTREM is used to study the energy consumption of a Java runtime system by dissecting the virtual machine into various software components. Section 9 describes related work and highlights existing differences between XTREM and other power estimation tools. Future work is described in Section 10. The summary for our work can be found in Section 11.

2. THE INTEL XSCALE CORE

The Intel XScale core is a high-performance, low-power microarchitecture specifically targeted for embedded applications [Intel Corporation 2003b]. It is compatible with the ARMv5TE instruction set and includes support for eight new DSP instructions that take advantage of a fast DSP coprocessor. We chose to study the Intel

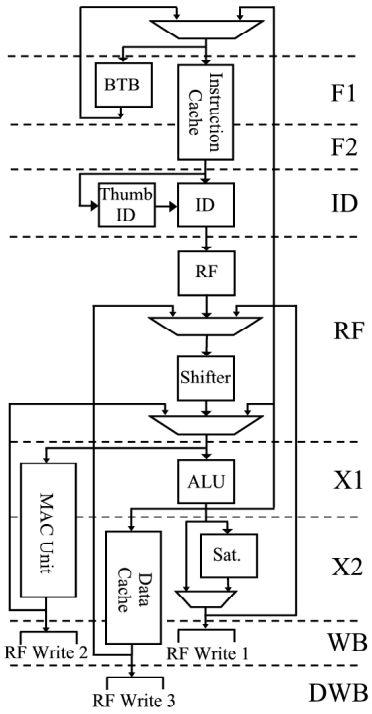

Fig. 1. Block diagram of the Intel XScale microarchitecture pipeline.

XScale core largely because of its wide use in embedded high-performance applications. Furthermore, its support for dynamic voltage and frequency scaling (DVFS) makes it a good candidate for power/performance research studies.

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 [M. Levy 2002]. 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.

The high clock rates of the Intel XScale core exacerbate the latency difference between core and main memory. To alleviate this problem, the core includes two specialized data buffers called the *fill buffer* and the *write buffer* that sit between the processor's core and main memory [Intel Corporation 2000]. 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 Intel XScale core supports 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 XTREM's models: Data and instruction cache accesses have been split into two stages, buffers assimilating fill and write buffers have been modeled 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 XTREM provides 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 available to the Intel XScale core [Intel Corporation 2003b]. 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 metrics 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.

3. XTREM POWER AND PERFORMANCE SIMULATOR

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 microcontroller surrounded by various peripherals that

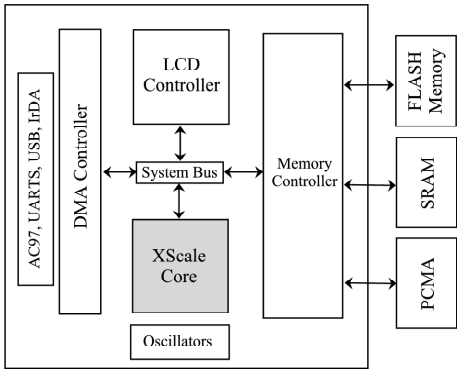

Fig. 2. Processor block diagram. The Intel XScale core is surrounded by support functional units that contribute to the processor's overall power consumption. Not all peripheral units are shown in this figure.

interface to external components. For example, Figure 2 shows a block diagram of the Intel XScale-based PXA255 microcontroller showing the spatial placement of the main core within the processor. This work focuses on the power behavior of the main processing core and less so on the energy usage of the external components such as the UARTs, LCD drivers and PCMCIA on-chip modules. It is, of course, to some extent impossible to ignore the effects of these units when power sampling is done on the microcontroller in real time. We minimize these effects by turning off as many unused peripherals as possible.

The starting point in the construction of power models is a detailed description of the Intel XScale technology and microarchitecture as described by Clark et al. [2001]. This paper describes the 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 [Brooks et al. 2000] 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 (as in the case of memory arrays and caches).

Our 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.

Our power models are based on the following CMOS energy equation:

$$E = C_L V^2 \tag{1}$$

Estimating the node capacitance is the first step to estimating energy (and consequently power) requirements of various processor structures. For example, the SRAM array shown in Figure 3 is used for fast data structures within the CPU such

as the register file and cache arrays. Therefore, given the mathematical equations that describe the power requirements of an SRAM array suffices to describe the power usage of variety of data structures within the core.

For the SRAM array shown, M data elements are arranged vertically and their constituent N bits are spread horizontally. On every clock cycle, all the bitlines (columns) of the structures are pre-charged. When a particular data element is accessed, the decoder selects a data element (a word in the structure) by enabling the wordline driver attached to the desired data element. Selecting a data element requires raising the wordline from 0 to 1, which according to Equation (1), will consume energy proportional to the capacitance seen by the power supply. Looking at Figure 3, we see that this capacitance consists of the diffusion capacitance of the wordline driver, two gate capacitances for every bit attached to the wordline, and finally the wire capacitance of the wordline. The addition of these capacitances is expressed in Equation (2).

Power dissipation by the pre-charge logic happens as follows: On every cycle, the pre-charge logic pre-charges all bitlines to a predetermined value, VDD. If a particular bitline is already charged, no energy is invested for that bitline. If a bitline was discharged by a previous element access, however, one of the bitlines will be discharged and the pre-charge logic will invest energy proportional to the capacitance of the bitline, which is equal to the diffusion capacitance of the pre-charge transistor, the diffusion capacitance of the bit cell access transistor, and the capacitance of the bitline wire. The addition of these capacitances is expressed in Equation (3). Equations (2) and (3) are given as an example of the analysis and modeling detail used in the construction of power models for various functional units.

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

$$C_{bitline} = C_{diff}(PreCharge) + C_{diff}(CellAccess) * M + C_{metal} * BitLineLength \quad (3)$$

Not all necessary power models for the Intel XScale core are found in Wattch. For example, the unique T-shaped clock structure common in the Intel XScale core had to be created based on work by Clark et al. [2001] since Wattch power models assume an H-tree clock distribution network. Functional units for which no complete circuit-level diagram is available are assumed to consume constant power on every access. The power consumption value of these units is either approximated using functional-unit isolation techniques or a power consumption estimate found in existing literature. The MAC unit is an example for which no complete circuit-level diagrams are published. Liao and Roberts [2002] describe the architecture of the MAC unit used by the Intel XScale core. The level of detail presented by Liao and Roberts [2002], however, is not enough to construct a comprehensive circuit-level description of the MAC unit. The paper, however, provides the nominal power consumption of the unit, which is used by XTREM as the per-access power consumption cost.

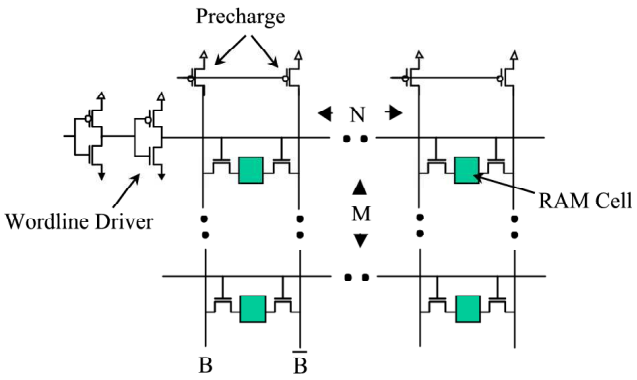

Fig. 3. RAM array schematic. Transistor-level schematics like the one shown here are used to create mathematical equations that estimate internal node switching capacitance.

The power consumption of other functional units for which no circuit-level description is available, such as the ALU and the memory manager unit (MMU), is derived empirically using functional-unit isolation techniques.

Functional unit isolation works by executing a carefully-crafted micro-stressmark on our development board. Since the Intel XScale core is a single-issue, in-order processor, the behavior of the execution pipeline as well as data interdependence between instructions can be accurately predicted based on the input instruction stream. An approximation of the power usage of a particular functional unit can thus be obtained by measuring the *delta* change in power consumption between a micro-stressmark that uses a partular functional unit, say, every cycle, and the power consumed by a second micro-stressmark that does not use the functional unit in question. Given this delta power difference and the access rate of the function unit, an average per-access power consumption can be estimated.

While functional unit power isolation is useful, it is not our primary approach to power modeling. As stated before, we only use power isolation techniques on selected functional units for which no other techniques were available. We do, however, use power isolation techniques more extensively for validation purposes, as covered in Section 6.

3.1 The XTREM Functional Simulator

XTREM is in part derived from the ARM-SimpleScalar simulator [The SimpleScalar Toolset 2001], 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 (instructions per cycle) 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.

XTREM 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. Supporting architectural features such as these makes XTREM more closely-matched to the microprocessors we target.

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

XTREM 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 XTREM JVM Simulation Support

To support research on power and performance of Java runtime system for embedded devices, XTREM gives researchers the ability to run complex Java runtime systems such as Sun's KVM reference CLDC design [Sun Microsystems 2000] 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 that 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 an unmodified "dynamic" link of the XORP. For four CLDC Java benchmarks, our experiments showed an average difference of less than 2% between hardware performance counter values. Physical power measurements between the two XORP configurations are also very similar, with an average difference of less than 0.5% 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 [Intel Corporation 2003a] powered by the Intel PXA255 microcontroller running the Linux

[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.

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 such as 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 Intel PXA255 microcontroller 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 perform our calculations 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 basic equation $P = V \cdot I$.

We set the PXA255 microcontroller 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 are driven by a 3.3V power supply. Under this setup we measured a CPU idle power of 80mW and an average CPU power consumption close to 270mW. Although main memory power consumption is not part of our study, we measured an idle SDRAM power consumption of 40mW and an active average power consumption close to 90mW (for one SDRAM HYB39S25610DT module).

4.2 Runtime Power Sampling

We perform power sampling at runtime of various test benchmarks using the Agilent 34401A digital multimeter. The digital multimeter interfaces to a PC through a GPIB cable. 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 such as the LCD clock driver, UART clock drivers, and the AC97 clock driver. Turning these components off reduces idle power consumption by as much as 7 to 8mW (out of a total of roughly 270mW non-idle power consumption) for the 200Mhz and 1.5V processor setup.

4.3 Runtime Performance Sampling

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

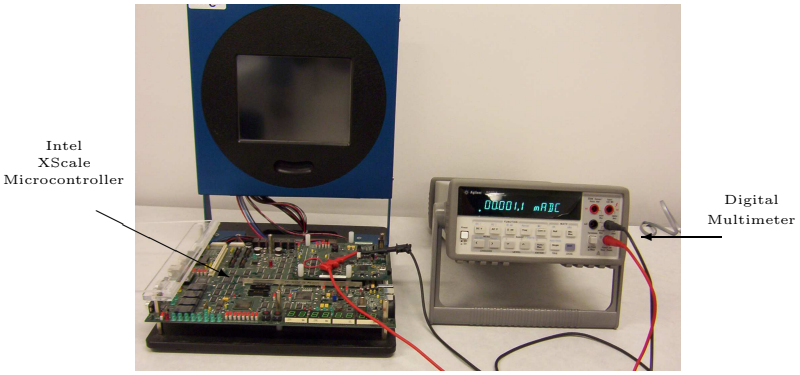

Fig. 4. Physical measurements setup. The Agilent 34401A Digital Multimeter (DMM) is used to sample current consumption of the Intel PXA255 CPU. High sampling speeds are achieved through a GPIB connection between the DMM and a host PC.

Pentium 4 systems [Isci and Martonosi 2003]. 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 XTREM. To this end we compare performance counter results and equivalent performance metrics reported by XTREM for several stressmark programs. Stressmarks 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_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_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_writeline` is very similar to `dcache_write`, except that `dcache_writeline` writes one integer in every memory address location corresponding to the beginning of one cache line so that memory address coalescing is not possible. `Add` stresses the arithmetic unit of the Intel XScale core. The `add` benchmark has a very high IPC (instructions per cycle) 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. `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. `icache_stress` is a micro-benchmark with a high number of instruction cache misses. `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 derived from hardware performance counter readings, IPC reported by XTREM and IPC from an unmodified ARM-SimpleScalar simulator. For our work, the average IPC difference for the nine stressmarks is less than 3%, while the average error between hardware performance counters and ARM-SimpleScalar is more than 17%, being the memory sub-system the main source of performance error.

A 3% average performance error in stressmark testing between XTREM and performance counters is good, but in order to further quantify the accuracy of XTREM we need to employ more realistic, complex benchmarks. For this end we have selected five benchmarks from MiBench by Guthaus et al. [2001], an embedded benchmark suite. The benchmarks are: `cjpeg` (jpeg compression application), `bitcount` (bit counting algorithm), `CRC` (a 32-bit cyclic redundancy check), `SHA` (secure hash algorithm), and `Dijkstra` (a shortest path algorithm). These benchmarks were chosen based on their large percentage of CPU time and work done in user space. We also want to test how well our simulator is able to track hardware performance for Java applications. `Jzlib` [Jean-loup Gailly and Mark Adler 2004], `Crypto` [Legion of the Bouncy Castle 2004], `GIF` [FM Software 2004] and `REX` [Zaliva 2004] 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 [Embedded Microprocessor Benchmark Consortium 2003]. `Jzlib` is a ZLIB implementation in pure Java, `Crypto` is a Java implementation of 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 graph compares the IPC derived from hardware performance counters (HPC), the IPC reported by XTREM, and the IPC obtained from ARM-SimpleScalar (SSS). As seen from the figure, XTREM 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%. For ARM-SimpleScalar, the largest error corresponds to `SHA` with more than 29% IPC difference, the lowest error is 1% for `CJPEG`.

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 impor-

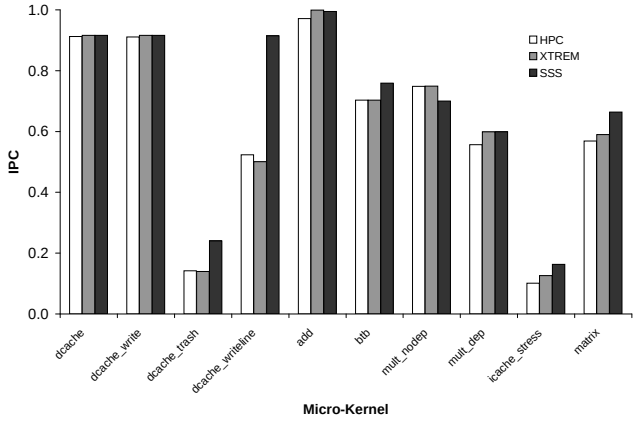

Fig. 5. IPC comparison for stressmarks as reported by XTREM, ARM-SimpleScalar (SSS) and Hardware Performance Counters (HPC). XTREM simulates adequately many aspects of the Intel XScale microarchitecture.

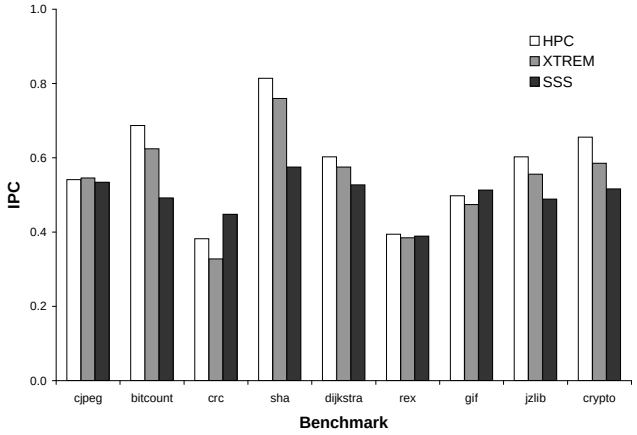

Fig. 6. Comparison between simulated and hardware-measured IPC performance for a set of MiBench and open CLDC benchmarks.

tance if we cannot guarantee a tool that also provides power estimates within a tolerable error.

The first step toward validation of our power models is 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 such as the instruction decoder, the TLBs and the instruction cache are difficult to isolate under normal execution conditions. On the other hand, some functional units can be orchestrated in very

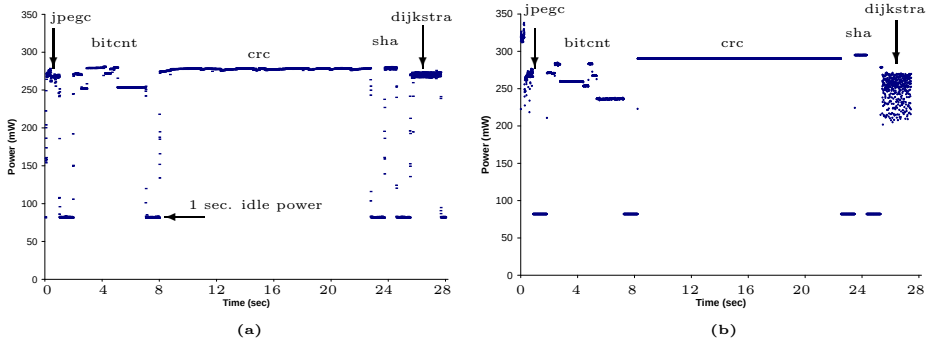

Fig. 7. Measured (left) and simulated (right) power consumption for five MiBench benchmarks. For this set of benchmark the maximum error is 11% (`JPEG`) and the minimum is 3.66% (`Bitcount`). Benchmarks are separated by a 1 second delay seen as 80mW idle power.

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 are 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 assumption 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 as described by Clark et al. [2001].

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 versus time plot for five MiBench benchmarks. Figure 7a shows real power measurements, 7b shows simulated power consumption. 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 80mW idle power consumption. The similarities between real and simulated power consumption 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 helps validate XTREM's CLDC power estimation capability, but also helps reveals many interesting characteristics

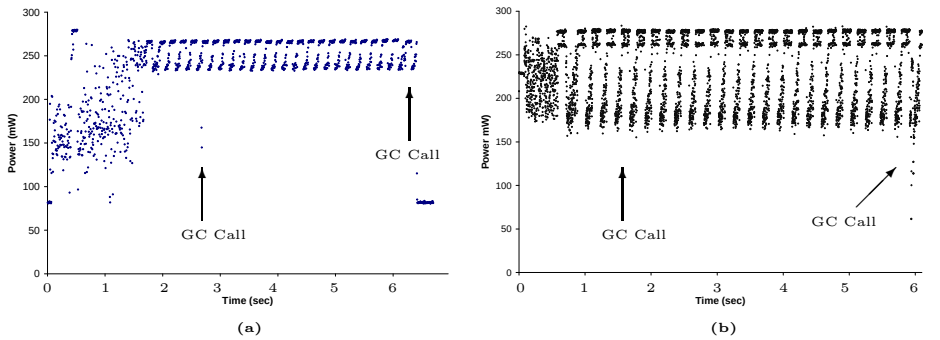

Fig. 8. Measured (left) and simulated (right) power consumption for `REX`. The start of CLDC benchmarks is characterized by very varied power behavior which corresponds to JVM initialization.

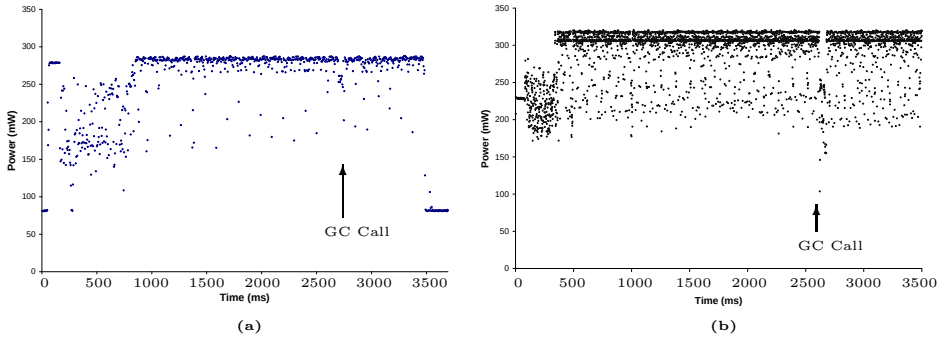

Fig. 9. Measured (left) and simulated (right) power behavior for `GIF`. A call to garbage collection creates a brief power drop around 2600ms.

of Java applications. Figure 8a is a power versus 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 8b displays simulated power versus 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 8b in the form of a small power spike. This power spike is a consequence of garbage collection.

Figure 9a and 9b 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 8b, Figure 9b shows the effects of

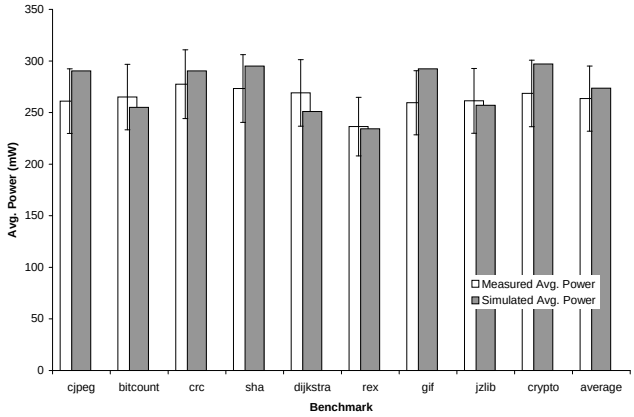

Fig. 10. Average power consumption for MiBench and open CLDC Java benchmarks. 12 percent error bars are shown.

	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 I. Average power and standard deviation comparison between hardware-measured power traces and simulated power consumption traces.

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 10 gives a graphical comparison between simulated average power and hardware-measured average power consumption for five MiBench and four Java CLDC benchmarks. Table I shows our results in tabular form.

7. ACTIVITY FACTORS AND 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

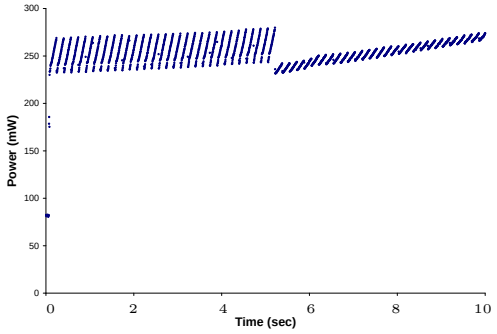

Fig. 11. Power dependence on input activity factors for the `ADD` operation.

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 (MAC, Data Cache, Shift Unit, and ALU), the ALU datapath unit showed the largest activity factor influence on power consumption.

To the right of Figure 11 is a power versus time plot of the ALU-datapath activity factor stressmark running on the Intel 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' bits in the datum while B changes in the horizontal direction 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 11 demonstrates it is possible to create power swings as large as 50mW on the Intel XScale core through a stressmark that exercises specific data activity factors on the ALU datapath. Power swings of double this magnitude are possible in a 400Mhz Intel XScale core.

8. CASE STUDY: JAVA CDC AND CLDC

In addition to power estimation, XTREM 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.

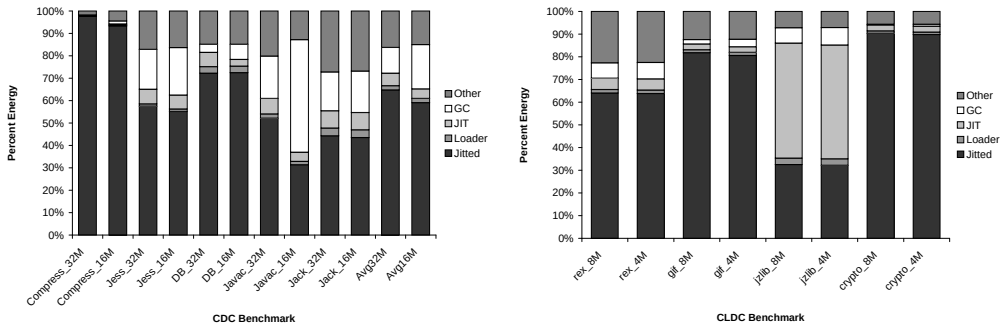

Fig. 12. Energy distribution among five JVM components for CDC (left) and CLDC (right) Java applications. The suffix attached to the benchmark name specifies the maximum heap size used by the JVM.

We utilize XTREM to understand how energy is distributed among the various functional parts of the JVM environment. For this purpose we divide the virtual machine into five main components: *JITted* code, Class Loader (*Loader*), JIT compiler (*JIT*), Garbage collector (*GC*), and *other*. This is similar to the approach followed by Vijaykrishnan et al. [2001]. The energy usage of each component is determined by the memory region from which the machine is fetching instructions. In other words, if instruction i has been fetched from the memory region occupied by garbage collector support functions, the power consumed by instruction i is assigned to garbage collection. The precise placement of all JVM functions and code in memory is known since our JVM has been statically compiled. We also want to understand how energy distribution varies when the maximum permitted heap size is varied. We expect changes in energy distribution since tighter memory constraints require the garbage collector to be invoked more frequently. Previous work in this field has been done by Vijaykrishnan et al. [2001], where the energy consumption of different JVM components is examined from the memory perspective—caches and main memory. Our case study focuses on providing energy consumption from the perspective of an embedded processor, not only taking cache power into effect, but also CPU functional unit energy consumption.

We employ our energy analysis technique on the previously described CLDC Java applications plus five SPECJVM98 Java CDC benchmarks [Standard Performance Evaluation Corporation 1998]. The CDC Java benchmarks are run until completion using the $-s100$ flag (full data set).

Figure 12 shows our case study results. The left side of the figure shows energy distribution for five CDC Java benchmarks and the right side shows results for CLDC benchmarks. The suffix attached to the name of the benchmark indicates the maximum heap size allowed. One of the most immediate observations from this figure is the fact that the `JITted` component is one of the highest energy consumers across all benchmarks. This is good news since the `JITted` component represents the actual user Java program being executed by the JVM. For CDC applications, the average energy consumption for the `JITted` component is 65% and 58% with 32M and 16M maximum heap size, respectively. Next to the `JITted`

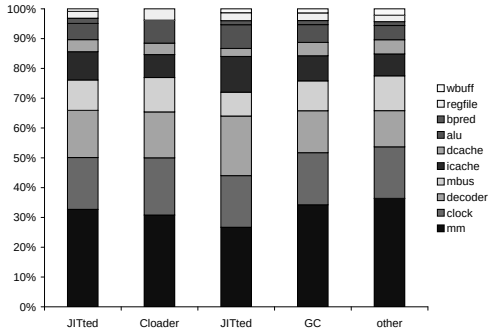

Fig. 13. Functional-unit energy breakdown of JVM components for `javac` with 16M maximum heap size.

component, `other` is the second largest energy consumer for CDC benchmarks with an average energy consumption of 16% for 32M heap size. The `other` component covers JVM functions that are needed for the correct functionality of the JVM, composed mainly of C library calls. The garbage collector is the third largest energy consumer utilizing more than 11% of the total energy with a heap size of 32M, only overtaking `other` when the maximum heap size is reduced to 16M. `javac` shows the largest variation in GC energy consumption. This is caused by the large increase of GC calls: 9 calls for 32M heap to 58 calls with 16M heap.

CLDC applications are less memory-hungry and are less affected when halving the maximum heap size (0.6% energy decrease for the `JITted` component). Overall, CLDC applications invest relatively more energy in the execution component (`JITted`) than CDC applications. A visible exception is `Jzlib`. The high percentage of power dedicated to `JIT` in `Jzlib` 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 energy requirement. Figure 13 shows how the energy consumption of the JVM components is distributed among various processor functional units when running the `javac` CDC benchmark with 16M heap size. The memory manager (`mm`) is one of the most power-hungry components, utilizing an average of 32% of the total energy across the five JVM components. In XTREM, the `mm` unit is activated when the core is stalled due to a main memory data or instruction fetch. Next to the `mm` unit we find the `clock` structure as the second highest energy contributor with an average of 17%. The instruction decoder and the memory bus follow with an average energy consumption of 15% and 10% respectively. Overall, the results presented here show that a significant percentage of time and energy (an average of 35% of the total consumed energy across simulated Java CDC benchmarks) is devoted to the virtual machine.

We can group everything not covered by the `JITted` component as energy consumed by the JVM. This grouping would tell us how much energy goes into the actual execution of the Java application and how much energy is consumed by the virtual machine and associated support functions. This is done in Figure 14. Figure 14 shows JVM power consumption and `JITted` power behavior for the `REX`

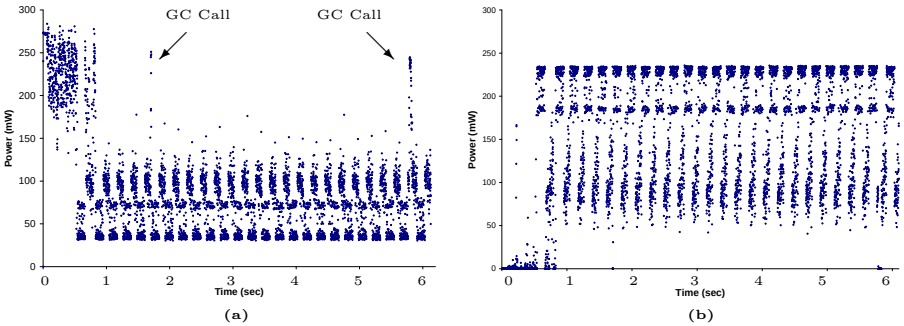

Fig. 14. Simulated power consumption of the JVM system (left) and JITted code (right) for `REX`

benchmark. The initialization stage of Java benchmarks is characterized by almost pure JVM activity. This is demonstrated in the first 500ms of Figure 14b, 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 14a makes it clear that the power spike seen from Figure 8b originates within the JVM runtime system (the garbage collector in this case).

The results obtained in our case study clearly indicate that the virtual machine is responsible for a significant fraction of the energy used by Java applications. In some cases, increasing the heap size helps decrease the energy consumed by the virtual machine since the garbage collector is invoked less frequently. For embedded devices, however, memory is a precious commodity and increasing the heap size may not always be possible.

Energy consumption of the class loader and JIT compiler may be reduced by pre-compiling and including system classes within the JVM bootloader. Doing so would reduce the number of classes loaded at runtime. Furthermore, since classes can be pre-compiled offline, the number of calls to the JIT compiler can also be reduced and consequently, its energy usage. This type of strategy is used by high-performance virtual machines such as the Jikes Virtual Machine by Alpern et al. [2000]. The drawback to this approach, however, is an increase in the binary size of the virtual machine due to included system classes. In a server-type virtual machine such as Jikes this may not be a problem since disk space and memory resources are abundant. For embedded devices, small binaries are preferred due to limited non-volatile memory.

9. RELATED WORK

A previous introduction of the XTREM simulator was done in [Contreras et al. 2004]. One of the key differences between this work and previous work is a deeper analysis of the capabilities of XTREM. This analysis is presented in Section 8 where we break down the energy consumption of the Java Virtual Machine among its constituent service components.

Previous research in the area of power consumption for Java systems has been done by Farkas et al. [2000]. For this work the authors use a hardware-based approach in the study of power behavior for Java applications. They analyze the 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.

Many studies have focused on simulation techniques for power estimation. Wattch by Brooks et al. [2000] and SimplePower by Vijaykrishnan [2000] 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 also proven to be very efficient in modeling complex architectures. Isci and Martonosi [2003] use 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. Contreras and Martonosi [2005] describe a linear power model based on hardware performance counters to estimate the power consumption of the PXA255 processor and associated external memory. While this approach can yield fast power consumption estimates, its accuracy is bounded by the amount of raw performance that hardware performance counters are able to expose: Applications with a large number of multiply instructions and/or diverse activity factors, for example, experience high estimation errors since performance counters are not able to track these events. In contrast, XTREM is able to model the entire pipeline of the core down to the functional unit level while taking activity factors into account. Chang et al. [2000] perform 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, exposing very detail levels of functional unit usage. 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.

10. FUTURE WORK

Future work for this study includes extending XTREM to support different voltage and frequency settings to more accurately describe the capabilities of Intel XScale technology cores. With this we hope to investigate ways of reducing energy consumption of JVM components such as the class loader, the garbage collector and the JIT compiler.

Furthermore we would like to include a run-time, phase-detection mechanism to our simulator infrastructure. This mechanism would be based on simulated hardware performance counters (HPCs) and would provide a way for detecting and predicting characteristic program execution phases. Such phase-detection mechanism, along with run-time power estimation, can potentially lead to sophisticated and accurate dynamic voltage and frequency algorithms (DVFS) for processors with support for DVFS and HPCs.

11. 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. Our case study has focused on analyzing the power consumption of the various software components in a Java runtime system, showing that the virtual machine itself can consume up to 35% of the total utilized energy of Java CDC applications. 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.

REFERENCES

Alpern, B., Attanasio, C. R., et al. 2000. The Jalapeno Virual Machine. *IBM System Journal 39,* 1.

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

Chang, N., Kim, K., and Lee, H. G. 2000. Cycle-Accurate Energy Measurement and Characterization With a Case Study of the ARM7TDMI. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*.

Clark, L., Hoffman, E., Miller, J., Biyani, M., Liao, L., Strazdus, S., Morrow, M., Velarde, K., and Yarch, M. 2001. An Embedded 32-b Microprocessor Core for Low-Power and High-Performance Applications. *Solid-State Circuits, IEEE Journal of 36,* 11 (November), 1599–1608.

Contreras, G. and Martonosi, M. 2005. Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events. In *Proceedings from the International Symposium on Low-Power Electronics and Design*.

Contreras, G., Martonosi, M., Peng, J., Ju, R., and Lueh, G.-Y. 2004. XTREM: A Power Simulator for the Intel XScale Core. In *Proceedings from the 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'02)*.

Embedded Microprocessor Benchmark Consortium. 2003. EEMBC Benchmarks for the Java 2 Micro Edition (J2ME) Platform. http://www.eembc.org.

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

FM Software. 2004. GIF Picture Decoder. http://www.fmsware.com/stuff/gif.html.

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

Intel Corporation 2000. *Intel XScale Core: Developer's Manual*. Intel Corporation. Order No. 273473-001.

Intel Corporation 2003a. *Intel DBPXA255 Development Platform for the Intel Personal Internet Client Architecture*. Intel Corporation. Order No. 278701-001.

Intel Corporation 2003b. *Intel XScale Microarchitecture for the PXA255 Processor: User's Manual*. Intel Corporation. Order No. 278796.

Isci, C. and Martonosi, M. 2003. Runtime Power Monitoring using High-End Processors: Methodology and Empirical Data. In *Proceedings from the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-36)*. 93–104.

Jean-loup Gailly and Mark Adler. 2004. Zlib Java Implementation. http://www.jcraft.com/jzlib.

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

Legion of the Bouncy Castle. 2004. Bouncy Castle Crypto 1.18. http://www.bouncycastle.org/.

Liao, Y. and Roberts, D. 2002. A High-Performance and Low-Power 32-bit Multiply-Accumulate Unit with Single-Instruction-Multiple-Data (SIMD) Feature. *Solid-State Circuits, IEEE Journal of 37,* 7 (July), 926–931.

M. Levy. 2002. Exploring the ARM1026EJ-S Pipeline: Extensive Architectural Modeling Highlights Frequency and IPC Tradeoffs. http://www.arm.com/miscPDFs/1752.pdf.

Standard Performance Evaluation Corporation. 1998. Spec JVM Client98. http://www.specbench.org/jvm98/jvm98/doc/benchmarks/index.html.

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

The SimpleScalar-ARM Power Modeling Project. 2004. PowerAnalyzer. http://www.eecs.umich.edu/~panalyzer.

The SimpleScalar Toolset. 2001. SimpleScalar LLC. http://www.simplescalar.com.

Vijaykrishnan, N. 2000. Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower. In *Proceedings of the 27th International Symposium on Computer Architecture*.

Vijaykrishnan, N., M. Kandemir, S. K., Tomar, S., Sivasubramaniam, A., and Irwin, M. J. 2001. Energy Behavior of Java Applications from the Memory Perspective. *The 1st USENIX Java Virtual Machine Research and Technology Symposium (JVM'01)*.

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

Zaliva, V. 2004. Java regular expressions. http://www.crocodile.org.