

# Fast, Flexible, Cycle-Accurate Energy Estimation

Phillip Stanley-Marbell, Michael S. Hsiao  
Department of Electrical and Computer Engineering  
Rutgers University  
Piscataway, NJ 08854

{narteh, mhsiao}@ece.rutgers.edu

## ABSTRACT

Designing energy efficient hardware and software systems demands different tools at various levels in the design hierarchy. There is however a dearth of tools to enable investigation and implementation of energy efficient software and hardware architectures. Presented is a fast, flexible, cycle-accurate architectural simulator, Myrmigki, that models a commercial microcontroller and microprocessor family, and enables cycle-accurate power dissipation analyses through a combination of instruction level power analysis and circuit activity estimation.

Myrmigki is intended to be used to study the effect of microarchitectural features on the energy efficiency of hardware and software systems. It provides facilities for dynamic voltage scaling, clock speed setting and per-cycle architecture reconfiguration, and is easily extended to add new microarchitectural features and model new instruction set architectures. The simulator provides over an order of magnitude speedup over a contemporary state-of-the-art power estimating simulator, while providing estimates within 10% of measurements from prototype hardware that it models.

## 1. INTRODUCTION

Increasing device integration has heightened the desire to reduce device power consumption. Besides mobile applications where battery life is a key utility metric, power consumption is becoming increasingly important for tethered applications, due to packaging and cooling costs. There is therefore a growing need to investigate the impact of both hardware and software architectures with low power consumption, and the interactions therein. On the side of hardware, system architectures and processor microarchitectures for low power must be investigated, and on the software end, compiler writers are beginning to investigate avenues for exploiting new low power hardware architectures. In order to bridge the gap between the investigation of hardware architectures and compiler techniques for low power, an infrastructure for architectural tradeoff investigation is needed.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*ISLPED'01*, August 6-7, 2001, Huntington Beach, California, USA.

Copyright 2001 ACM 1-58113-371-5/01/0008 ...\$5.00.

Such an infrastructure must be functionally correct, fast and flexible. Functional correctness and speed are essential to be able to run realistic workloads over the simulator, while obtaining correct results. Speed is also key to making techniques such as energy-profiling compilation a reality. Flexibility is important since the level of detail in the information required from the simulator will vary between applications, and between different phases of the same application. In general, the greater the degree of detail required, the slower the simulation, making it necessary to tradeoff simulation speed for detail.

Presented is a simulation tool, Myrmigki, an execution driven architectural simulator and energy estimation framework, which models an embedded system based on the Hitachi SH3 architecture [9]. It is a complete architectural simulator capable of booting unmodified commodity operating systems and embedded applications targeted at the SH3-LCEVB [2], a hardware evaluation board for the SH7708F60 microcontroller. Although it currently only simulates the Hitachi SH architecture family, the simulator can easily be extended to model other processor families, due to its modular construction. It models the processor (CPU core, on-chip cache, various on-chip peripherals), off-chip memory, and RS-232 serial communications interface. In addition to an accurate functional model, it includes two complementary means of estimating energy cost of application software – An empirical instruction level power model similar to [13] and circuit activity estimation. Also modeled are features not present on the target hardware, such as dynamic voltage scaling, clock speed setting and a broad range of on-chip cache configurations, to permit the investigation of architectural tradeoffs for energy efficiency, and the investigation of software and hardware architectures for dynamic voltage scaling and clock speed setting.

The simulator provides over an order of magnitude performance improvement over a contemporary state-of-the-art power estimating simulator that models similar hardware. The functionality of the simulator has been validated with several benchmark programs, including a commercial operating system [1]. The accuracy of the simulated energy estimates were validated with hardware measurements, and are shown to be within 10% of the measured values.

The remainder of the paper is structured as follows. Section 2 describes related work. Section 3 describes the SH3 architecture and the overall architecture of the simulator. Sections 4 and 5 describe the power estimation facilities provided, and the facilities for investigating architectural tradeoffs. Section 6 details a performance evaluation and

comparison with a contemporary power estimating simulator, and Section 7 concludes the paper.

## 2. RELATED WORK

There has been a considerable amount of work in the field of cycle accurate simulators for architectural investigation and performance evaluation of application and system software [3, 7, 15]. These have typically focused on fast functional simulation for architecture emulation, address tracing for memory hierarchy analysis and microarchitectural trade-off investigations.

The level of detail in the modeling performed by a simulator is a deciding factor in its performance. Generally, the larger the amount of detail, the slower the simulator. Fast functional simulators employing dynamic compilation such as Shade [7] and Embra [15] can simulate commercial microprocessors functionally at overheads of less than 10 host instructions per simulated instruction, whereas power estimating simulators such as Wattch, SyCHOSys and SimplePower [5, 11, 16] provide cycle accurate circuit activity estimation at overheads of hundreds of host instructions per simulated instruction.

Few tools exist for estimating the impact of hardware and software architectures on energy consumption in embedded systems. The majority of energy estimation tools available are at lower levels of the system design hierarchy. Though indispensable for gate level simulation, tools such as SPICE can only be applied in the final stages of a design, where it is often impossible to make sweeping microarchitectural changes. Furthermore, simulation speeds of such tools are very slow compared to application execution speeds on the target architecture. Tools such as PowerMill [10] provide higher simulation speeds with competitive accuracy, however still simulate designs in too great detail to be suitable for power estimation and simulation of real workloads on commercial processors.

Tools such as SyCHOSys [11] simulate at speeds of up to 8 orders of magnitude faster than SPICE, at the cost of reduced accuracy. SyCHOSys simulates portions of a design by automatically generating a cycle accurate simulator from a user supplied processor description. SimplePower [16] simulates a subset of the SimpleScalar [6] architecture. Wattch [5], like SimplePower simulates a derivative of the SimpleScalar architecture. They both derive power estimates from analysis of the circuit activity induced by application programs and from detailed capacitive models for the architectures they emulate. In [14] the authors extend a commercial instruction-level simulator with an instruction power model for a specific hardware device and its peripherals, obtaining energy estimates within 5% of hardware measurements.

Myrmigki is functionally comparable to SyCHOSys, SimplePower, Wattch and the simulator described in [14]. Unlike Wattch and SimplePower, Myrmigki does not incorporate elaborate capacitive macro-models for the architecture it simulates, but rather provides an equivalent amount of insight by providing per-component transition counts for a given workload. These may then be used by a system designer off-line to obtain per-component and system-wide energy consumption estimates. Furthermore, Myrmigki augments this circuit activity estimate with an instruction level power model for the processor it models, enabling it to per-

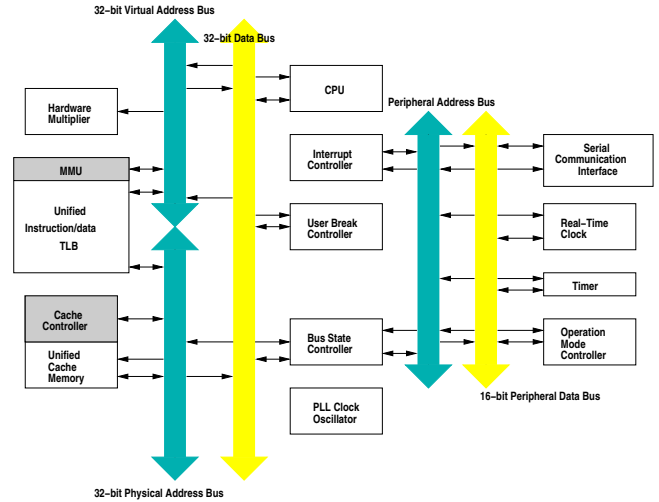


Figure 1: Hitachi SH3 architecture

form energy cost estimation within 10% accuracy of measured values.

## 3. MODELED ARCHITECTURE

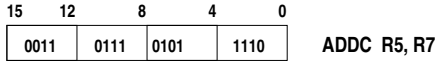
The SH3 is the third generation of the Hitachi SuperH RISC architecture [9]. It is a 32-bit RISC architecture, with a 5-stage pipeline, most instructions completing in a single cycle. Instructions are 16-bit for high code density. Figure 1 shows the functional layout of the SH3 architecture.

The simulator is controlled through a simple scripting language, which includes the entire instruction set of the modeled architecture, along with commands specific to the simulator such as commands for reconfiguring the simulated cache architecture, toggling pipeline simulation, loading and executing files containing commands in the command language and loading and executing binaries compiled for the simulated architecture. The language interface is also accessible to applications running over the simulator through a memory-mapped command register described further in Section 5.1. A command line interface permits interactive execution of commands in this command language. The simulator is also available as a library, that can be linked into applications that wish to use the functionality of the simulator, for example an energy-aware profiling compiler may use the simulation engine to determine tradeoffs between various optimizations, in terms of energy usage.

The simulator models a five stage pipeline, with pipeline stages for instruction fetch, decode, execute, memory access and register file write-back. The simulation of the pipeline can be enabled or disabled on a per-cycle basis. Simulation of the pipeline provides an accurate account of the number of cycles needed to execute an application, and it may be turned off to obtain a faster functional simulation. The fast functional simulation provides functionally correct application execution results at the cost of a loss in the accuracy of application execution time and induced circuit switching activity estimation. The signal transition activity modeling is discussed further in Section 4.1, and control over the modeled architecture is discussed further in Section 5.

### 3.1 Shared Decode Cache

Myrmigki performs simulation of applications using a technique similar to threaded code [4]. Object code for the target



Integer Value of Instruction = 0011011101011110 = 0x375E

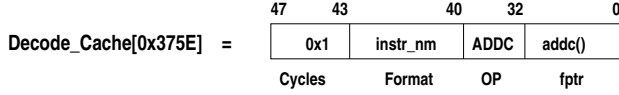


Figure 2: Decode Cache entry

architecture is used to index into a structure called the *decode cache*, which maintains per-instruction information for the quick simulation of instructions. Instructions in the SH architecture are fixed length, 16-bit. This permits the use of a small (64K-entry) software structure to maintain decode information for every possible instruction encoding.

Each entry in the decode cache consists of four fields: Cycles, Format, OP and Fptr. The Cycles field specifies the execution latency of the instruction. The latency of memory access instructions is determined by the cache configuration, and also by the simulated operating voltage discussed further in Section 5. The Format field specifies the instruction format of the indexing instruction, and is used to extract operand fields from the instruction word. Instruction formats in the Hitachi SH architecture map roughly onto addressing modes. The Fptr field is a pointer to a function that simulates the instruction. The OP field is used for internal simulator bookkeeping. Figure 2 shows how an SH3 instruction is used to index into the decode cache. In the example, the instruction used to index into the decode cache is an ADDC (Add with carry) instruction, and the Fptr field of the entry points to a function implementing ADDC. Each entry in the decode cache is 6 bytes, for an overall decode cache size of 384KB.

## 4. POWER ESTIMATION

The simulator's switching activity estimation models the on- and off-chip address and data busses of the SH3, the peripheral address and data busses, register file, program counter and pipeline registers. Circuit activity estimation tallies are maintained independently for the different busses and hardware components, to enable energy estimation to be performed in the presence of capacitive models for the different circuit components, as is done in SimplePower [16]. Signal transitions in the control logic and within each functional unit are presently not modeled. Given appropriate models for these components, it is straightforward to add such models to the simulator infrastructure.

### 4.1 Efficient Transition Counting

In performing circuit activity estimation, it is important that the counting of transitions be performed correctly, and as efficiently as possible. Contemporary cycle accurate energy estimators which estimate circuit power via transition counting during simulation such as [11] and [16] acknowledge the need for efficient means of transition counting. There are two primary methods employed in contemporary tools for transition counting: A simple linear scan approach (SimplePower Release 1.0) and an early terminating linear scan [11]. Myrmigki employs a constant time hierarchical approach described in [8].

```

flips(a, b)
{
    if (a == b)
        return 0;

    count = a ^ b;
    count = count&0x55555555 + (count&0xaaaaaaaa >> 1);
    count = count&0x33333333 + (count&0xcccccccc >> 2);
    count = count&0x0f0f0f0f + (count&0xf0f0f0f0 >> 4);
    count = count&0x00ff00ff + (count&0xff00ff00 >> 8);
    count = count&0x0000ffff + (count&0xffff0000 >> 16);

    return count;
}

```

Figure 3: Constant-time transition counting algorithm

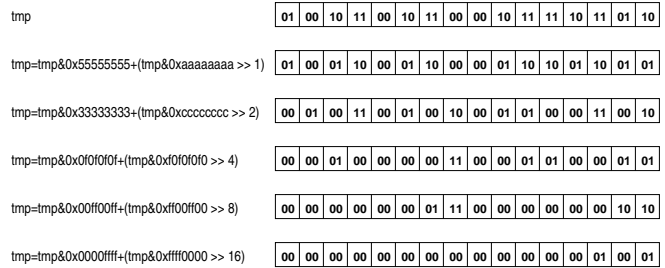


Figure 4: Illustrating the constant-time transition counting algorithm

For determining the number of bits that change between two 32-bit words, a simple linear scan approach requires at least 1 XOR, 32 SHIFTS, 32 ANDs and loop overhead, a minimum of 65 operations. The approach described in [11] as an optimization over this approach terminates the linear scan as soon as its data structures determine that the loop index is within the sign bits of both words. The loop overhead is still incurred, the algorithm may need to perform all 32 loop iterations, and there is additional memory overhead in maintaining data structures to determine when to terminate.

The algorithm employed in Myrmigki is a hierarchical bit counting scheme. It performs counting of the number of bits set in a word in-place. The algorithm requires 21 operations (1 XOR, 10 ANDs, 5 SHIFTS and 5 ADDs), and incurs no loop overhead. The pseudo-code in Figure 3 illustrates the implementation of a routine to determine the number of bits that flip between two 32-bit words. The operation of the algorithm involves XOR-ing two words to obtain a bit-vector with bits set at positions where the two words differ, then counting the number of bits set in this bit vector using the hierarchical constant time bit counting algorithm. Figure 4 illustrates the operation of the transition counting algorithm on a 32-bit word, tmp, obtained after XOR-ing two 32-bit words, and indicating the bit positions in the two words that differed. The final value of tmp after the last step in the algorithm, is the count of the number of bits that were set before operation of the algorithm.

### 4.2 Instruction Level Power Analysis

The simulator permits estimation of per-cycle power consumption of simulated software by an instruction level power analysis technique similar to [13]. The simulator incorpo-

rates per-instruction power models for all instructions in the instruction set of the SH3, the power values being specific to a SH7708F60 microcontroller, running at 60MHz and 3.3V. The instruction power models provide an empirical estimate of the average current that is drawn by the processor and memory subsystem when executing a specific instruction. To obtain the model data, a loop of 100 identical instances of each instruction was run on a processor evaluation board, and the average current draw of the processor and memory subsystem measured.

For each simulated instruction, a lookup is performed for the average current that would be drawn by the processor and memory, and this is used to calculate the corresponding power consumption estimate for that cycle. No inter-instruction effects such as the presence of different instruction types in the pipeline are modeled. In the presence of more accurate instruction power data, or accurate models for modeling inter-instruction and instruction operand effects, the simulator can easily be extended.

## 5. ARCHITECTURAL EXTENSIONS

The simulator models the operating voltage of the processor, the default being 3.3V, and can be varied as described in section 5.1. When the modeled operating voltage is changed, the latency of memory operations is scaled accordingly. In scaling the operating voltage of the processor, memory continues to operate at the same voltage and speed, and the latency of a memory access in milliseconds remains fixed. However, with respect to the simulation, a memory access instruction that incurred a cache miss has to wait for fewer cycles (at the new lower operating frequency), for data to be available from the memory subsystem.

### 5.1 Flexible Simulation Control

The simulator takes advantage of an unused opcode in the SH3 architecture to provide fine-grained control over simulation. This new instruction can be used to change the simulated VDD, clock frequency, cache configuration (cache size, block size, associativity) or may be used to enable or disable profiling activities, all on a per-cycle basis.

Insertion of reconfigure instructions into the instruction stream could be performed by a compiler to investigate the effects of application or operating system controlled dynamic voltage scaling / clock speed setting, or other system or hardware controlled architecture reconfiguration. In the absence of explicit compiler support, reconfigure instructions may be inserted into compiled binaries by replacing NOPs such as those that often occur in the branch delay slots of delayed control transfer instructions.

Applications running over the simulator may also interact with it through a memory mapped command register. Any string written to this memory region is passed to the simulator’s command language interface, and handled as would any command issued from the simulator’s interactive command line. This is an extremely useful interface as it is easily programmed at the application level, and can be used to perform such varied tasks as executing assembly instructions in-line, or may be used as a portal for applications running over the simulator to access resources outside the simulator.

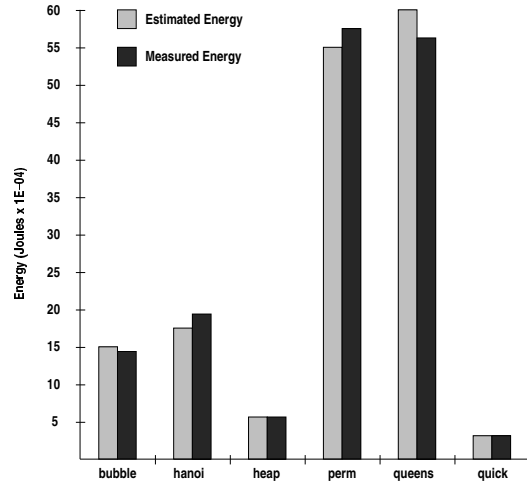


Figure 5: Comparing energy estimates from simulation and measurements from hardware

## 6. RESULTS

This section presents a comparison between the measured and estimated energy consumption for a suite of applications, an illustration of the range of simulation speeds and the accompanying tradeoff in Myrmigki, and compares its performance to a contemporary state-of-the-art cycle-accurate power estimating simulator, SimplePower Release 1.0.

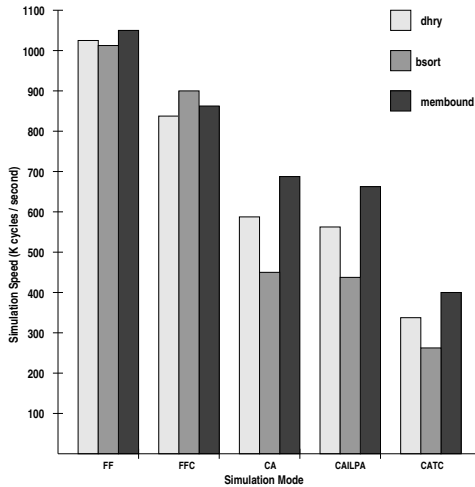
### 6.1 Power Estimation

Figure 5 presents results on running six applications over the simulator to obtain energy estimates, and also running the same applications directly on the hardware. To obtain the energy cost of the applications on the prototype hardware, each application was run in an infinite loop, and the average current draw of the processor and memory measured. The application was then run over the simulator to obtain the number of cycles necessary for completion, which was then used to calculate the overall running time and hence energy consumption of the application. Both the simulation and the empirical measurements were performed with the cache enabled. The estimated energy values in the figure are all within 6.5% of the measured values, which is competitive with results obtained in contemporary studies [5, 14, 16] on similar processor architectures.

### 6.2 Simulation Mode Tradeoff

It is often the case that not all portions of a simulation are of interest in architectural investigations, and it is desirable to be able to simulate portions of an application correctly, but at a reduced level of accuracy. This concept has previously been successfully employed in the SimOS complete machine simulator [12]. Myrmigki permits six different levels of simulation detail, *Fast Functional*, *Fast Functional with Cache*, *Cycle-Accurate*, *Cycle-Accurate with Transition Counting*, *Cycle-Accurate with Instruction Level Power Analysis* and *Cycle-Accurate with Transition Counting and Instruction Level Power Analysis*, referred to as FF, FFC, CA, CATC, CAILPA and CATC+CAILPA respectively.

The FF simulation mode provides the highest simulator performance. In FF, the simulator performs purely functional simulation – The application is simulated in terms of



**Figure 6: Trading off simulation detail for simulation speed**

its effect on the state of the machine registers, main memory and peripherals, but the motion of instructions through the pipeline, instruction latencies and cache and main memory latencies are not modeled. The FFC mode adds cache simulation. The CA, CATC and CAILPA modes provide cycle-accurate simulation. These modes model the motion of instructions in the pipeline and also model ALU and memory latencies. The CATC mode adds transition counting over the basic CA mode. The CAILPA simulation mode provides per-cycle power consumption estimates by instruction level power analysis as described in Section 4.2.

Figure 6 illustrates the simulator performance for five different simulation modes, for three sample applications – the Dhrystone 2.1 benchmark (dhry), a memory-bound application (membound) and bubblesort (bsort). For example, from the figure, the Dhrystone 2.1 benchmark simulates at 1023,000 simulated cycles/second for the FF mode and 336,000 simulated cycles/second for CATC mode.

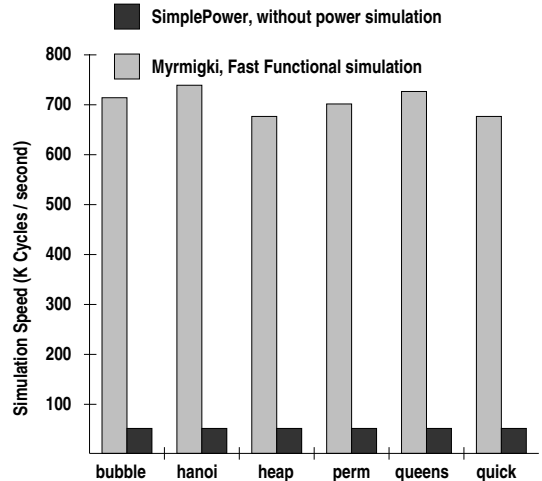
The different simulation modes provide different amounts of detail at a tradeoff in simulation speed. For the Dhrystone 2.1 benchmark, enabling the cache simulation in FF mode (i.e. going from FF to FFC mode) leads to a 18% reduction in the number of cycles simulated per second. Enabling pipeline simulation and simulating instruction latencies further decreases the simulated cycles per second by 30%. Enabling instruction level power analysis in CA mode leads to a simulation speed degradation of only 3%. Disabling instruction level power analysis in CA mode, and enabling only transition counting analysis leads to a performance degradation of 42%. Overall, for the Dhrystone 2.1 benchmark, simulation can be sped up by 204% over the CATC mode, by changing the simulation to the FF mode.

### 6.3 Performance Comparison

Simulations comparing SimplePower and Myrmigki were run on a Sun Ultra 10 running SunOS 5.7, with 256MB RAM and a clock speed of 440MHz. In this comparison, Myrmigki was configured to match the default configuration of SimplePower as closely as possible. In the following comparison, both simulators were configured to simulate processors with in-order execution and perfect caches – all memory accesses incur no extra latency. Table 1 shows the distribution of exe-

	SimplePower	Myrmigki
bubble	391,398	355,890
hanoi	220,820	231,034
heap	105,380	118,150
perm	751,789	878,080
queens	468,459	527,216
quick	67,543	65,668

**Table 1: Simulated cycles on SimplePower and Myrmigki for example applications. SimplePower application compiled with `ssbig-na-sstrix-gcc -O0` (gcc version 2.6.3), and Myrmigki applications compiled with `sh-coff-gcc -O0` (gcc version 2.95.2)**



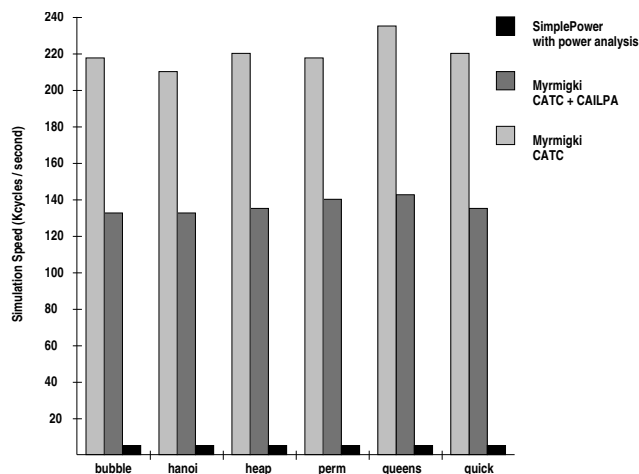
**Figure 7: Comparing fastest simulation modes in Myrmigki and SimplePower**

cution time in simulated clock cycles for the six applications used in the performance comparison with SimplePower, for both simulators running in cycle-accurate power estimating mode. As can be seen from the table, even though the simulated microarchitectures differ in instruction sets and processor microarchitectures, the number of cycles required on each architecture for the completion of the applications differed on the average by no more than 10%. It is therefore reasonable to compare the simulation speeds of this set of applications on the two architectures.

Figure 7 depicts the simulation speeds for SimplePower and Myrmigki with both simulators running at their fastest possible configuration. This mode is of interest especially when large workloads need to be positioned prior to actual estimation e.g. the boot-up sequence of an OS running over the simulator might not be of interest to an investigator, who might wish to “fast-forward” to a simulation region of interest.

In its fastest simulation mode, even though it performs no power estimation, SimplePower still generates statistics on the signal transition activity on the instruction and data cache buses, while Myrmigki performs no signal transition estimation at all. Furthermore, the Fast Functional mode for Myrmigki does not model the the pipeline, and all instructions complete in a single cycle. For this configuration, Myrmigki simulates the applications an average of 12.9 times faster than SimplePower.

Figure 8 illustrates the performance of the two modes



**Figure 8: Cycle-accurate power estimation speeds in Myrmigki and SimplePower**

of cycle-accurate power estimation in Myrmigki against the performance of SimplePower with power estimation enabled. In CATC mode, in which Myrmigki performs cycle-accurate simulation with transition counting, it simulates applications an average of over 36.5 times faster than SimplePower. In SimplePower, the power estimation is performed in-line, whereas in Myrmigki, an equally rich set of transition activity information is generated that may be used to perform energy estimation off-line. The Figure also shows the performance of Myrmigki's CATC+CAILPA mode, in which in addition to transition activity estimation, Myrmigki performs online power estimation using an instruction level power analysis technique. The resulting estimate was shown in Section 4 to be within 6.5% of measured values on hardware. In this simulation mode, Myrmigki simulates applications an average of 22.7 times faster than SimplePower.

## 7. CONCLUSION

Presented is a simulation tool, Myrmigki, an execution driven architectural simulator and energy estimation framework, which models an embedded system based on the Hitachi SH3 architecture [9]. It includes two complementary means of estimating energy cost of application software – An empirical instruction level power model and circuit activity estimation. The simulator provides a flexible range of simulation detail levels, and permits per-cycle reconfiguration of the simulated architecture.

The simulator has been shown to provide high performance and great flexibility, being up to 36.5 times faster than a contemporary power estimating simulator in one simulation mode, and up to 22.7 times faster than the same simulator in a comparable power estimating simulation mode. The accuracy of the energy estimates generated by the simulator were verified by hardware measurements and found to be within 6.5% of measured values for the example applications presented.

We are currently using the simulator as a platform for investigation into the effect of microarchitectural features on power consumption, as well as a platform for investigating the effect of compiler optimizations on application energy efficiency. Source and documentation for the simulator is available from <http://www.myrmigki.org>.

## 8. REFERENCES

- [1] eCos : The embedded Cygnus Operating System. <http://sources.redhat.com/ecos/hardware.html#sh>.
- [2] Hitachi SH7708 Development Board. <http://semiconductor.hitachi.com/tools>.
- [3] R. Bedichek. Some Efficient Architecture Simulation Techniques. In *USENIX Winter Technical Conference*, pages 53–63, January 1990.
- [4] J. R. Bell. Threaded Code. *Communications of the ACM*, 16(6):370–372, June 1973.
- [5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.
- [6] D. Burger, T. Austin, and S. Bennett. Evaluating Future Microprocessors: The SimpleScalar ToolSet. Technical Report CS-TR-1308, Computer Sciences Department, University of Wisconsin-Madison, 1996.
- [7] R. Cmelik. Shade: A Fast Instruction-Set Simulator for Execution Profiling. In *Proceedings of the 1994 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 128–137, May 1994.
- [8] R. Gutman. Exploiting 64-Bit Parallelism. *Dr. Dobbs Journal*, (316):133–136, September 2000.
- [9] A. Hasegawa, I. Kawasaki, K. Yamada, S. Yoshioka, S. Kawasaki, and P. Biswas. SH3: High Code Density, Low Power. *IEEE Micro*, 15(6):11–19, December 1995.
- [10] C. X. Huang, B. Zhang, A.-C. Deng, and B. Swirski. The Design and Implementation of PowerMill. In *Proc. Intl. Workshop on Low Power Design*, pages 105–110, April 1995.
- [11] R. Krashinsky, S. Heo, M. Zhang, and K. Asanovic. SyCHOSys: Compiled Energy-Performance Cycle Simulation. In *Workshop on Complexity-Effective Design, 37th Conference on Design Automation*, 2000.
- [12] M. Rosenblum, E. Bugnion, S. Devine, and S. A. Herrod. Using the SimOS Machine Simulator to Study Complex Computer Systems. *ACM Transactions on Modeling and Computer Simulation*, 7(1):78–103, January 1997.
- [13] V. Tiwari, S. Malik, and A. Wolfe. Power Analysis of Embedded Software: A first Step Towards Software Power Estimation. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 384–390, August 1994.
- [14] T. Šimunić, L. Benini, and G. D. Micheli. Cycle-Accurate Simulation of Energy Consumption in Embedded Systems. In *Proceedings of the 36th ACM/IEEE Design Automation Conference*, pages 867 – 872, June 1999.
- [15] E. Witchel and M. Rosenblum. Embra: Fast and Flexible Machine Simulation. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 68–79, 1996.
- [16] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool. In *Proceedings of the 37th Conference on Design Automation*, pages 340–345, 2000.