Remote Performance Monitoring (RPM)

Chandra Krintz and Selim Gurun

Computer Science Department
University of California, Santa Barbara
{ckrintz,gurun}@cs.ucsb.edu

1. INTRODUCTION

As battery-powered, resource-constrained systems continue to grow in capability and complexity, it is increasingly difficult to accurately measure and characterize the full-system power consumption of real devices. However, we must do so if we are to effectively model, predict, and optimize programs and systems to increase battery life. Extant approaches to measurement and characterization of power and energy behavior include simulation, processor-level metrics, and measurement via external monitoring devices (e.g. multi-meters).

Hardware performance monitors (HPMs) have gained wide-spread use recently for estimation of CPU processing power [9, 7, 2, 6, 3, 4, 5]. In addition, other types of processor-level metrics have been shown to be effective for predicting CPU performance and power consumption, in particular those related to program phase behavior [8, 1, 4, 5]. These processor-level metrics have been shown in these prior works to correlate well with processor power consumption [2, 3, 4, 5, 6]. Unfortunately, prior work does not evaluate how well processor-level metrics correlate with or estimate the power and energy consumed by the entire system (as opposed to simply the CPU power and energy consumption) or focus on high-end processors such as the Intel Pentium class of processors. We consider Our understanding of full-system energy and of the behavior of energy-efficient processors (e.g. StrongARM and XScale CPUs) is vital if we are to develop techniques for extending the batter life of resource-constrained, mobile devices.

To enable the characterization of a full-system as easily and accurately as possible, we developed a toolset called the *Remote Performance Monitor (RPM)*. RPM consists of both hardware and software components. The hardware components include a set of high-end tools to monitor the target microcomputer. The software tools include utility programs to configure various system characteristics of the monitored device, and operating system extensions and device drivers to collect performance data (such as HPM counters). A GUI program and a web interface enables remote users (e.g. students and researchers) to submit jobs for performance profiling to our system, i.e, to extract accurate performance profiles without investing in, installing, and managing their own system. RPM collects power, energy, and HPM data for fixed- or variable-length intervals. Interval lengths are in terms of dynamic binary instructions and can be set by the user upon job submission. Users of our system can also control which metrics RPM collects and which intervals RPM samples.

At present, we use a RPM to characterize a Crossbow Stargate embedded microcomputer. The Stargate implements an Intel XScale processor and a number of I/O devices. The Stargate is very similar in functionality to an HP iPAQ handheld device (without the LCD display), and it is used extensively in sensor network research.

In this paper, we employ RPM to investigate how well processor-level metrics correlate with full-system power and energy consumption by programs. We consider a number of different HPMs as well as a technique that identifies code-based phases in program behavior using simulation. We make many interesting observations using RPM: We find that

- HPMs do not explain the variance in full-system power and energy consumed by the device for the programs that we have studied. This is in contrast to prior works that show that HPMs are effective for explaining variance in *processor* power consumption.
- IPC is also not highly correlated with power and that for some programs IPC is correlated to some degree with energy. Prior work shows that IPC is a good measure of processor power consumption.
- I/O types and their OS support, e.g., volatile memory and file systems, can impact the power and energy consumed by a program significantly.

We believe that our work provides a shared infrastructure that will enable researchers and students to collect fine-grained, highly accurate power, energy, and HPM profiles from a real system using real programs without investing in the necessary hardware. Moreover, our measurement analysis using RPM for real programs reveals that current approaches for processor-level power estimation do not correlate well with full-system power and energy behavior.

2. REMOTE PERFORMANCE MONITORING (RPM)

One of the primary goals of RPM is to provide a research testbed for power studies on embedded systems. Understanding and characterizing energy behavior is critical for techniques that extend battery life in embedded and mobile systems. To enable this, we require mechanisms that measure the power and energy a device consumes at a high resolution (fine grain) with high accuracy. Moreover, we must understand the power and energy behavior of

Dagstuhl Seminar Proceedings 05501
Automatic Performance Analysis
http://drops.dagstuhl.de/opus/volltexte/2006/504

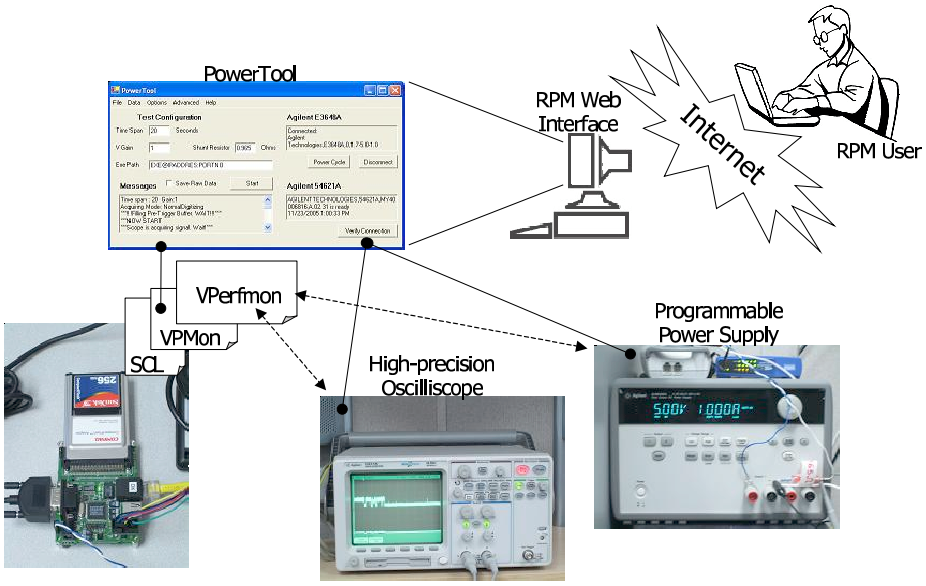

Figure 1: RPM Overview.

the device as a whole to ensure that we identify the primary contributing factors of battery drain and that the techniques we develop reduce this drain (and do not accelerate it).

Recent research using real systems has shown that hardware performance monitors correlate well with, and thus, can be used to estimate, the power consumption of the CPU [9, 7, 2, 6, 3, 4, 5]. Similarly, estimation based on patterns in the executing code, i.e., phase behavior, is also successful for CPU power estimation [1, 4, 5, 8]. For systems for which the CPU is the primary consumer of energy, these techniques may be adequate. However, processors vary greatly in capability, energy consumption, and the portion of the full-system power and energy consumption to which they contribute. Moreover, prior work has focused on power alone. However, high power consumption can result in lower total energy consumed if the execution time is significantly decreased. Total energy consumption is key to understanding and prolonging battery life in resource-constrained systems – so both must be measured, studied, characterized, and accurately understood.

An alternative approach to measurement of power and energy behavior for real systems is to employ a set of external measurement tools. Such tools include a multi-meter, oscilloscope, and programmable power supply and enable highly accurate and very fine-grained measurement (i.e., a large number of measurements per millisecond) of power and energy consumed by a device. Unfortunately, these tools are costly and immobile, making them less than ideal for sharing between geographically disjoint research groups and students. Moreover, these systems only collect power-related metrics; access to the performance profiling capabilities that a device may have is not supported.

The goal of our work is to extend such a system to enable concurrent performance profiling and shared access to the system by remote users. We refer to this system as the *Remote Performance Monitor (RPM)* and provide an overview of its primary components in Figure 1. RPM is a tightly integrated suite of tools to monitor program energy, power, and CPU performance. The RPM includes four components:

- A device driver and Linux kernel patches, called VPerfmon, that enable and control HPM, power, and energy profiling.
- A user program, called VPMon, that executes a submitted program under the control of VPerfmon.
- A user program, called SCL, that dynamically switches CPU frequency level.
- A Windows XP GUI program called the PowerTool, that monitors and controls the lab equipment (oscilloscope and power supply), and sets the experimental parameters.
- A web interface through which remote users can submit programs to the system for execution and profile collection. Users specify a set of parameters that control how often RPM profiles the program, the duration of profiling, profile granularity and accuracy, CPU frequency, and the metrics that RPM collects.

RPM monitors program power consumption at a very fine granularity (2K measurements/second) and high accuracy (1mW resolution) by default. A key difference between RPM and past measurement systems, is that RPM monitors the energy and power consumed by the entire device. We can extend RPM to monitor individual elements such as memory and CPU; however, our focus in this work is full-system power consumption.

RPM consists of an Agilent deep-memory oscilloscope that monitors the current passing through a high-precision resistor connected to the target computer power supply. We connect the oscilloscope to a workstation through a general purpose interface bus (GPIB). The PowerTool executes on the workstation and consumes, analyzes, and packages the collected data. The PowerTool also controls a high-precision, programmable power supply, the Agilent E3648A. In addition, RPM users can investigate and rewrite the boot-loader on the target devices using the PowerTool. We wrote the PowerTool software in the portable C# language using the Microsoft .Net platform.

RPM monitors a *target device* on which we execute the VPMon. The VPMon is the user interface to the target device that executes the submitted program and controls HPM profiling by interacting with VPerfmon. VPMon and VPerfmon are also portable to any architecture that supports Linux and implements hardware performance counters.

Event	Description
0x0	Instruction cache miss requires fetch from external memory.
0x1*	Instruction cache cannot deliver an instruction. This could indicate an ICache miss or an ITLB miss.
0x2*	Stall due to a data dependency.
0x3	Instruction TLB miss.
0x4	Data TLB Miss
0x5	Branch instruction executed, branch may or may not have changed program flow.
0x6	Branch mispredicted
0x7	Instructions executed
0x8*	Stall because the data cache buffers are full.
0x9	Stall because the data cache buffers are full.
0xa	Data cache access, not including Cache Operations.
0xb	Data cache miss, not including Cache Operations.
0xc	Data cache write-back. This event occurs once for each 1/2 line (four words) that are written back from the cache.
0xd	PC Modified

Table 1: PXA-255 Performance Monitoring Events. The events marked with a * counts the number of cycles that the condition is present.

The target device that we currently support is the Stargate sensor network intermediate node. The Stargate is representative of modern battery-powered, resource constrained devices as it implements the recent PXA-255 XScale processor and a wide range of popular I/O devices. We detail the components of this system in Section 3. We show the range of HPMs available (and thus available for RPM profiling) in Table 1. The Stargate is very similar to an HP iPAQ device without an LCD display.

VPerfmon is the control center for program profiling. VPerfmon provides virtual hardware performance counters to each application. The HPMs by default count global CPU events, i.e. they do not track events at the program or thread level. VPerfmon provides a layer that multiplexes the counters and that enables selective monitoring of particular programs and threads. VPerfmon implements a virtual instruction per cycle (IPC) counter by tracking instructions (cycles are tracked by default on most devices). The virtual counters are 64bits in size which reduces overflow problems. Users can selectively enable and disable sampling during the monitoring.

In our target device, the Stargate processor, the PXA-255, implements three 32-bit event counters; the hardware uses one to monitor dynamic clock cycles. VPerfmon sets the remaining counters to any two of the 14 events supported. The VPerfmon virtual counters reflect the same architecture ((i.e. extended to 64 bits), it uses one counter to count CPU clock cycles and the other two to monitor events.

VPerfmon also manages the profiling parameters set by default or by the RPM user. These parameters are forwarded to VPerfmon by VPmon upon program instantiation. The parameters control system call monitoring, exceptions and floating point operation monitoring, interval characteristics (size, variable versus fixed), and callbacks to user code.

VPerfmon facilitates interval-based data collection via the GPIO pin on the development board. Initially, RPM sets the GPIO pin to logic 0 when a program starts. During program execution, VPerfmon toggles the pin's value at then end of every interval. VPerfmon, as mentioned above tracks interval lengths (arbitrary or fixed) using some performance event specified by the user. For the data in this paper, we use instruction counts as the event and fixed-length intervals of 10 million instructions. The oscilliscope is equipped with two channels. One channel monitors the voltage shunt resistor to measure power consumption. The second channel monitors the GPIO pin that VPerfmon toggles. Using this setup, RPM is able to log and track power, energy, and performance data at interval boundaries.

Processor	32 Bit, 400 MHz Intel PXA-255 Xscale Arm architecture Version 5TE ISA 32 KByte Instruction and 32 KByte Data cache 2 KByte Mini Data cache 2 KByte Mini Instruction cache
Memory	32 MB Intel StrataFlash
Expansion Ports	1 Type II CompactFlash Slot *(populated with 256 MB CF card)* 1 PCMCIA slot
Network & Others	10 Base-T Wired Ethernet RS-232 JTAG USB *(disabled at present)* I2C *(disabled at present)*

Table 2: Stargate device characteristics (RPM target device)

The WWW interface exports most RPM functionality to the research and educational community. The features that we support via the interface include:

- A tool chain for cross-compilation of programs for the target device.
- An form to download the benchmark package. The package is a gzipped-compressed UNIX tar archive. The package contains all of the necessary target binaries and input files. In addition, the package includes (in its root directory) a shell script, called start.sh, that initiates execution. We currently support programs with execution durations of less than 10 minutes.
- An interface to control the execution (such as start, cancel, and the number of times to repeat the experiment (currently the max is 5)).
- An interface to control the VPerfmon configuration (fixed or arbitrary intervals, the interval start data (if arbitrary intervals are used), interval length (if fixed intervals are used), events to monitor, etc.
- An interface to the measurement equipment to direct to access experimental results and to power cycle the board before or during the user's experiments.

3. ANALYSIS

We are interested in the degree to which CPU-based events explain observed, full system power and energy performance. The RPM target device that we study is the XScale-based Crossbow Stargate sensor network intermediate node. We first present our empirical methodology and benchmarks. We then use RPM to investigate the relationship between CPU-level HPM metrics and full-system power and energy behavior.

3.1 Experimental Methodology

We present the characteristics of the RPM target device, the Crossbow Stargate, in Figure 2. We list the various components that the device implements broken down by those specific to the processor, memory, expansion ports, and other components.

In our evaluation, we use six popular, embedded systems benchmarks from the MediaBench benchmark suite. We show the benchmark programs and their characteristics that we collected using

Benchmark	Instr. Count 10^6	Time seconds		Energy joules		Diff. %	RPM ovhd %
		EXT2	RAM	EXT2	RAM		
gsmencode	2.59	10.88	10.87	15.30	15.21	0.63	7.1
gsmdecode	1.64	6.95	6.61	11.19	10.86	3.05	11.2
jpegencode	4.28	48.53	N/A	63.20	NA	NA	7.2
jpegdecode	1.45	19.46	11.36	26.45	18.43	43.49	8.2
mpegencode	1.43	107.24	107.49	195.02	195.37	-0.18	3.6
mpegdecode	2.13	311.80	312.01	570.27	568.60	0.29	0.9

Table 3: Benchmark characteristics

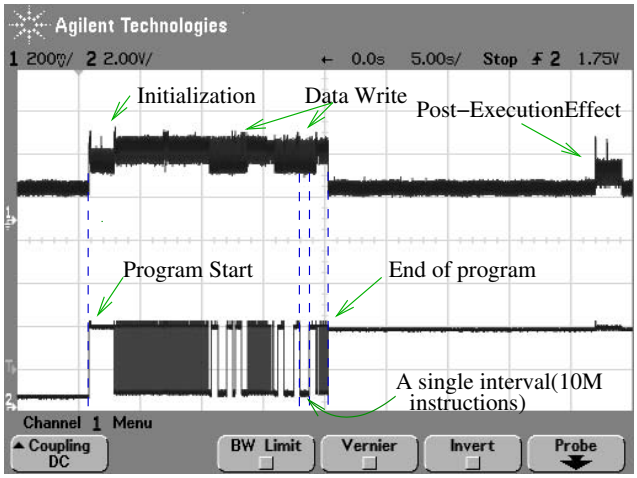

Figure 2: Power consumption of JPEGDecode on Ext2 file system. The top line shows the power consumption, and the bottom line shows the interval detection output pin voltage readings. The power phases (during the initialization and file write) are marked with an arrow. A post-execution effect, due to writes on compact flash, follows approximately 20 seconds after execution and very consistent across runs.

RPM in Table 3. The first three columns show the instruction count, execution time, and energy for each benchmark. The fourth column shows the difference between EXT2 drive and RAM. The fifth column shows the RPM overhead (in energy). The overhead (relatively) decreases as the application becomes larger. To collect benchmark energy characteristics, we run each benchmark five times with RPM using the same input, delete the first run (due to the high variability in performance due to system warmup), and average the results. We collect power data in fixed intervals each with length 10 million instructions. We use this methodology throughout our experimentation section.

We study the energy and power behavior for the benchmarks using two memory technologies: the compact flash card attached via a PCMCIA bus and the internal RAM. The flash is supported by the EXT2 file system. JPEGEncode benchmark does not fit in RAM on this device, so we exclude it from our RAM-based experimental results. During the experiments, the wired network interface is connected but idle and there are no other tasks running.

RPM supports all of the performance monitoring events that we showed previously in Table 1. However, in this paper, we only consider HPMs for instructions per cycle (IPC), instruction cache miss, data stalls, instruction TLB misses, and data TLB misses. These metrics have been shown to be important in modeling the CPU power consumption [2]. To collect HPM event statistics, we run the program repeatedly, collecting one statistic at a time.

3.2 Complexities in Full-System, Real Device Behavior

Resource-constrained, battery-powered devices and their software exhibit complex interactions and behaviors that RPM is able to capture. As an example, Figure 2 displays the RPM output for one of our benchmarks (JPEGDecode). The benchmark decodes a large file (30MB) and writes to an EXT2 Linux file system. The horizontal axis of the figure is time. There are two sets of data, one per oscilloscope channel: Power (at the top) and execution progress (at the bottom). The power data shows periods of stable behavior (phases) and periods of instable behavior (transitions).

We indicate execution progress by toggling a binary switch each time an interval completes. Interval sizes are fixed for this experiment at 10 million instructions. The second channel simply outputs a line of ones and zeros. and is set to 1 when the program starts as indicated in the figure. Whitespace between interval toggle values indicate that the interval takes more time than other intervals which appear to be blocks of adjacent lines. For example the first interval in the program takes significantly more time than the intervals that follow it.

Another interesting behavior occurs approximately 25 seconds execution terminates at which point there is an increase in power consumption. We refer to this as a *post-execution effect*. This behavior is consistent across runs and we do not observe this behavior when we execute the application from the RAM device. Since most HPM measurement ends when the program ends, HPM data is unable to capture such activities (and even this assumes that the HPMs are operational during operating system execution). Similarly, simulation cannot capture such behavior unless the system is power accurate and supports OS execution. Such phenomenon are real and motivate the need for full system monitoring of energy, power, and performance in a unified experimentation framework such as RPM.

3.3 The Relationship Between HPMs and Energy/Power

The overall power, energy, and performance behavior of the individual benchmarks varies significantly. The GSM benchmarks are very stable and produce uniform behavior. MPEGEncode exhibits a very regular bi-modal patterns. JPEGDecode, as we showed in the example above, varies significantly over the life of the program.

To evaluate the degree to which HPM metrics explain power and energy behavior, we computed the correlation between each. Figure 3 shows the square of the correlation (multiplied by 100) which is also known as the *variation explained* and R^2 correlation statistic. The R^2 value indicates the percentage of the variance present in the energy and power data, respectively, that is explained by each HPM metric. Small R^2 values indicate that little variance is explained by the metric and thus, the metric may not be a good predictor of the behavior in the energy power data.

We investigated two different file systems and I/O devices for storage, as we describe above. The left graph shows the R^2 values for the EXT2 file system and the right graph shows the data when we use the RAM drive. The data shows that across benchmarks, each HPM metric explains a very small percentage of the variance in either energy or power behavior and for either storage device. The clock cycle metrics explain the largest percentage of variance, i.e., cycles and IPC – however, the percent of the variance that is explained by these metrics is still very low.

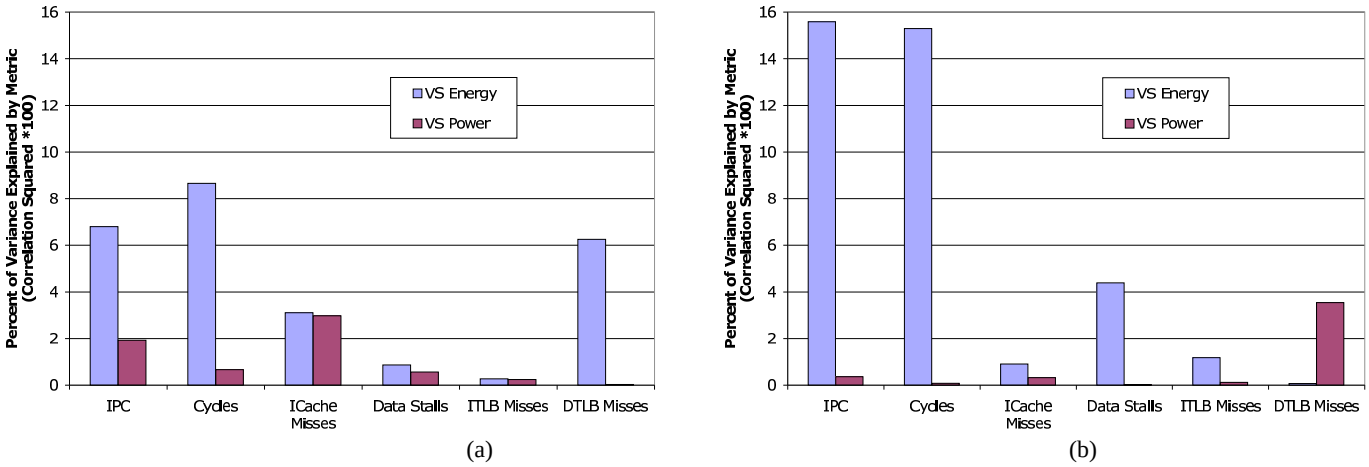

Figure 3: R^2 Correlation Statistic: Correlation squared times 100. This value shows the percent of the variability in energy (first bar) or power (second bar) that is explained by the metric (x-axis). (a) shows the data for the EXT2 file system; (b) shows the data for the RAM drive.

4. CONCLUSIONS

As resource-constrained, battery-powered devices and their software continue to increase in complexity and capability, it is important for us to understand full system energy and power behavior, if we are to identify techniques that extend battery life. To facilitate better understanding of the energy and performance characteristics of these complex systems, we present RPM, the Remote Performance Monitor.

RPM is a remotely accessible system to characterize an *real* embedded devices. We provide remote access via a user-friendly web interface and hide most of the cumbersome lab equipment details from the end user. We couple high-end external power and energy measurement with device-level CPU performance monitors. RPM characterizes the system in a number of different levels. For example, users can monitor a single application or multiple applications by including or excluding the effect of system calls. It is also possible for users to change the characteristics of the remote system.

We use RPM to investigate the degree to which commonly used HPM metrics explain the variance in the power and energy consumption behavior of programs. We find that only a very small portion of the variance in power and energy curves is explained by HPM behavior. Our results indicate that HPMs alone may not be sufficient or accurate in estimating full-system energy performance of resource constrained devices.

Acknowledgments

This work was funded in part by Intel, Microsoft, and NSF grant Nos. ST-HEC-0444412, ITR/CCF-0205712, and CNF-0423336.

5. REFERENCES

[1] W. L. Bircher, M. Valluri, J. Law, and L. K. John. Runtime identification of microprocessor energy saving opportunities. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 275–280, New York, NY, USA, 2005. ACM Press.

[2] G. Contreras and M. Martonosi. Power prediction for intel xscale® processors using performance monitoring unit events. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 221–226, 2005.

[3] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO '03: Proceedings of the 36th ACM/IEEE International Symposium on Microarchitecture*, 2003.

[4] Canturk Isci and Margaret Martonosi. Detecting recurrent phase behavior under real-system variability. In *IISWC '05: Proceedings of the 2005 International Symposium on Workload Characterization*, 2005.

[5] Canturk Isci and Margaret Martonosi. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *HPCA '06: Proceedings of the Twelfth International Symposium on High-Performance Computer Architecture*, 2006.

[6] Russ Joseph and Margaret Martonosi. Runtime power estimation in high-performance microprocessors. In *ISPLED '01: Proceedings of the International Symposium on Low Power Electronics and Design*, 2001.

[7] I. Kadayif, T. Chinoda, M. Kandemir, N. Vijaykirsnan, M. J. Irwin, and A. Sivasubramaniam. vec: virtual energy counters. In *PASTE '01: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 28–31, New York, NY, USA, 2001. ACM Press.

[8] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *30th Annual International Symposium on Computer Architecture*, June 2003.

[9] Andreas Weissel and Frank Bellosa. Process cruise control: event-driven clock scaling for dynamic power management. In *CASES '02: Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246, New York, NY, USA, 2002. ACM Press.