Remote Performance Monitoring (RPM): Full-System Power, Energy, and Performance Profiling for Resource-Constrained Devices

UCSB Technical Report #2006-02

Selim Gurun Priya Nagpurkar Chandra Krintz

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

ABSTRACT

Understanding the full-system power and energy behavior of real, resource-constrained, battery-powered devices is vital to accurately characterize, model, and develop effective techniques for extending battery life. Unfortunately, extant approaches to measuring and characterizing power and energy consumption focus on high-end processors, do not consider the complete device, employ inaccurate (program-only) simulation, rely on inaccurate, course-grained battery level data from the device, or employ expensive power measurement tools that are difficult to share across research groups and students.

In this paper, we present RPM, a remote performance monitoring system, that enables fine grained characterization of embedded computers. RPM consists of a tightly connected set of components which (1) control lab equipment for power measurements and analysis, (2) configure target system characteristics at run-time (such as CPU and memory bus speed), (3) collect target system data using on-board hardware performance monitors (HPMs) and (4) provide a remote access interface. Users of RPM can submit and configure experiments that execute programs on the RPM target device (currently a Stargate sensor platform that is very similar to an HP iPAQ) to collect very accurate power, energy, and CPU performance data with high resolution.

We use RPM to investigate whether CPU-based performance data in the form of HPM metrics or program phase behavior correlates well with full-system energy or power behavior. Prior work shows that both accurately estimate processor power consumption for high-end CPUs. In resource-constrained devices, such as the one we study, however, the processor consumes a much smaller portion of the total power in the system than for high-end processors. Our experimentation with RPM for the Stargate and set of embedded system benchmarks, show that CPU-based metrics do not correlate well with full-system energy and power consumption. Moreover, we find that full-system energy and power varies significantly with the type of memory device and file system.

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).

Simulation and CPU-based techniques are most commonly used to evaluate power and energy consumption of a program executing on a particular device. Simulation is limited in that the simulation process introduces error in both performance and power estimation. Moreover, most simulation systems emulate a single program as opposed to a complete system including operating system and external devices – each of which can significantly impact power, energy, and performance.

Hardware performance monitors (HPMs) have gained wide-spread use recently for estimation of CPU processing power [24, 16, 4, 15, 12, 13, 14]. 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 [23, 3, 13, 14]. These processor-level metrics have been shown in these prior works to correlate well with processor power consumption [4, 12, 13, 14, 15]. 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). Full-system energy is important for techniques that attempt to extend battery life in resource-constrained, mobile devices.

Moreover, most prior work on HPM-based power prediction focuses on high-end processors such as the Intel Pentium class of processors. These processors consume a much larger portion of the total system power than the processors used in resource-constrained devices such as those in which we are interested: hand-helds, sensor network nodes, and cellular phones. These devices commonly implement energy-efficient processors such as those from the StrongARM [10] and XScale [11] family of processors.

Our prior work employed internal battery monitors to measure and characterize power and energy consumption of resource constrained devices [18, 26, 25]. Unfortunately, all extant internal battery monitors or mobile devices are coarse grained, not program specific, and inaccurate. A more accurate approach to measuring

the power and energy consumed by a device is to use high-end current meters such as oscilloscopes and programmable power supplies that sample a system via external probes with high resolution and accuracy. External systems, however, are expensive, immobile, and cannot be used remotely. These factors limit the degree to which such systems can be shared by geographically separated researchers and students.

To enable the characterization of a full-system as easily and accurately as possible, we developed a toolset called the *Remote Performance Monitoring (RPM)* system. 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.

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.
- Branch-based phases do not map well to power and energy phases.
- 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.

In summary, with this paper, we contribute

- A remotely accessible and freely available toolset for remote performance monitoring of XScale programs.
- An RPM system that automatically collects accurate power, energy, and hardware performance monitor profiles.
- An RPM web interface through which users can submit jobs to the system for profile collection. User control what metrics RPM employs, interval size, and which intervals to profile.
- An analysis enabled by RPM of the efficacy of commonly used processor-level metrics for estimating full-system power and energy consumption. Prior work has shown such techniques to be effective for processor-level estimation of power.
- An analysis of how well program-level phase behavior maps to full-system power and energy behavior.
- An evaluation of the impact of memory types and file system implementations on power and energy consumption.

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 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 [24, 16, 4, 15, 12, 13, 14]. Similarly, estimation based on patterns in the executing code, i.e., phase behavior, is also successful for CPU power estimation [3, 13, 14, 23]. 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:

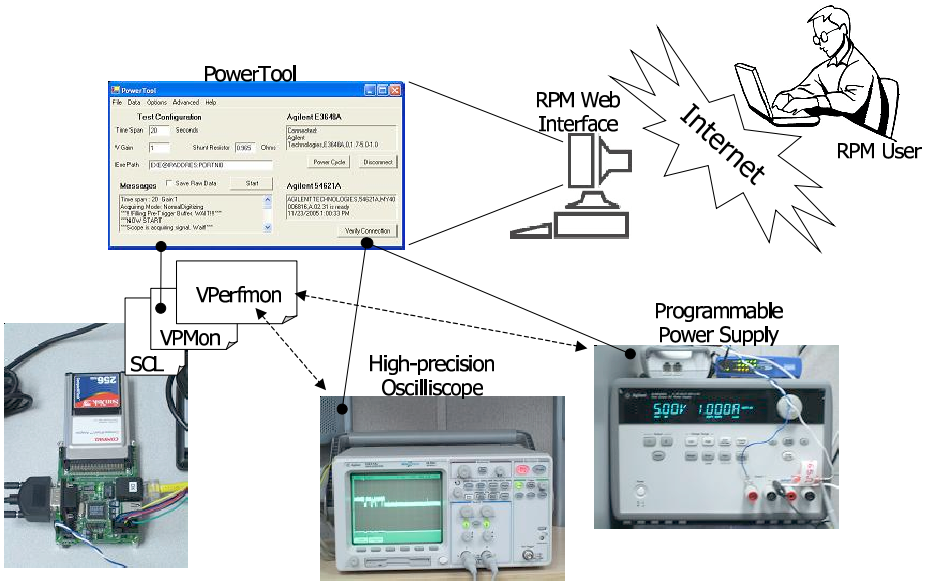


Figure 1: RPM Overview.

- 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. We describe VPerfmon in greater detail in the next subsection.

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.

The Stargate is used extensively in sensor network research as a gateway for communication and data collection from sensors. The Stargate supports many important mechanisms that enable instrumentation and analysis of the system. For example, the Stargate implements both JTAG ports (a standardized interface that enables users to test and debug at the chip level) and general purpose I/O (GPIO) ports.

The RPM uses the SCL driver to scale the CPU speed of the Stargate if desired. For target devices that do not support frequency or voltage scaling of the processor, we simply do not load the SCL driver module into the Linux system. The Stargate processor, the PXA-255, has a very flexible CPU clock implementation that users can configure to set memory, bus, and CPU core speed independently. There are currently five valid configurations (due to timing constraints). SCL enables users to manipulate the configurations at runtime and compiles a log of the new speed, device, and the time at which it implemented the changes the clock speed (using microsecond resolution).

2.1 VPerfmon

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

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.

virtual instruction per cycle (IPC) counter by tracking instructions (cycles are tracked by default on most devices). The virtual counters are 64bits in size to reduce overflow problems. It is possible to selectively enable/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 interfaces to and monitors other system events to increase the accuracy of the HPM profiles. When the VPMon initiates a new program, it contacts VPerfmon The VPerfmon driver allocates a set of virtual counters for the new task. Similarly, VPerfmon allocates a set of virtual counter when a process under the control of VPerfmon forks a child process. When the kernel performs a context switch to a task under VPerfmon control, VPerfmon configures and enables the counters. When the task is suspended or terminates, VPerfmon stores the virtual HPMs.

To isolate application and operating system performance, the VPerfmon kernel patch disables HPMs on interrupt entry and reenables them on exit. This operation requires a read-modify-write cycle that is equal to three XScale instructions. As a result, the patch does not significantly increase interrupt latency.

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 (off by default). If on, VPerfmon continues to monitor system HPMs during system calls. One useful way to compare simulated data with real data collected by RPM is to turn system call monitoring off when the simulation system under investigation does not fully support system calls (as is common).

- Exceptions and floating point operation monitoring (off by default): Many resource-constrained devices, including those with StrongARM and XScale processors, do not implement a floating point co-processor. For such devices, floating point operations are implemented as user level libraries or, more commonly, as undefined instructions. For the latter, when the kernel executes one of these instructions, it emulates floating point hardware in software. The VPerfmon driver can disable performance monitoring during the processing of floating point emulation and other exceptions. When this option and the previous option are disabled, the VPerfmon virtual HPMs reflect the performance of only user-space execution. There is a 2% difference at most between RPM and Simplescalar [2] simulation. This difference is due to the instructions required to turn HPM profiling on and off.

- Fixed versus arbitrary interval lengths (fixed intervals each with length 10 million instructions, by default): VPerfmon can monitor and log the events for fixed or arbitrary length intervals. For fixed-length intervals, the user specifies the length in terms of some HPM count. Arbitrary intervals provide a way to the user to set interval boundaries without using a fixed length. It is an array of long long integers. The VPmon program reads this information from a file and passes to the VPerfmon using an device driver command.

VPerfmon facilitates interval-based data collection via the GPIO pin on the development board. Initially the GPIO pin is reset to logic 0 on program start. 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.

2.2 RPM Web Interface

The WWW interface export 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.

We password protect the web site to limit access and to limit security problems. The web page is currently available at `http:`

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)

Benchmark	Instr. Count	Time seconds		Energy joules		Diff. %	RPM ovhd
	10^6	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

`//www.cs.ucsb.edu/~racelab/RPM`. The login for interested reviewers is `rev1` and the password is `mobisys06`. We plan to make the webpage available to the research community if this paper should be accepted.

3. EVALUATION AND ANALYSIS

To show the utility of RPM, we use it to evaluate the relationships between the full-system energy and power consumption and CPU-based events. Prior work considered the relationship between the events and microprocessor energy behavior – for both high-end and energy efficient CPUs such as those in which we are interested, e.g., the Intel XScale processor. As such, 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. The Stargate is similar in functionality and performance (and is similarly equipped in terms of and external devices) to the HP iPAQ hand-held.

In the following subsection, we describe our experimental methodology and benchmarks. We then show how we (and others) can use RPM to measure and analyze the power and energy behavior of complex programs. We then use RPM to investigate the relationship between CPU-level metrics (HPM data and phase data collected via simulation) 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.

In our experiments, we investigate both power and the energy. For real devices, the capacity of a battery is expressed in Coulombs, i.e., Ampere-hours, that the battery can deliver to the device [5]. We use the standard power and energy functions:

$$\begin{aligned} Power &= V \times I \\ Energy &= V \times I \times t \end{aligned}$$

where I is the amperes running through a circuit and V is the potential drop. The battery voltage, V, is dependent on internal chemical components and diminishes as the capacity of battery decreases. A program can reduce its energy consumption either by reducing the rate of discharge, I, or the duration of discharge, t, or both. The energy consumed by a program decreases the battery lifetime. Power consumption is also important because typically, the relation between battery capacity and rate of discharge has a non-linear component and battery capacity decreases much faster if a program draws current at a larger rate. However, it is possible for one program to consume more power than another yet consume less energy (due to the time component in the energy computation). Power is also important, since its fluctuation relates directly to the heat produced by the system.

In order to plot power, energy, and HPM data on the same graph, we normalize the data so that the mean is 0. We do so by subtracting the average from the measured value and dividing the result by the standard deviation. When we compute the correlation coefficient for two datasets, we do so using the normalized values.

In our evaluations, we use six popular, embedded systems benchmarks from the MediaBench benchmark suite. We show the benchmark programs and their characteristics that we collected using 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, to limit the amount of data we present 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 [4]. 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

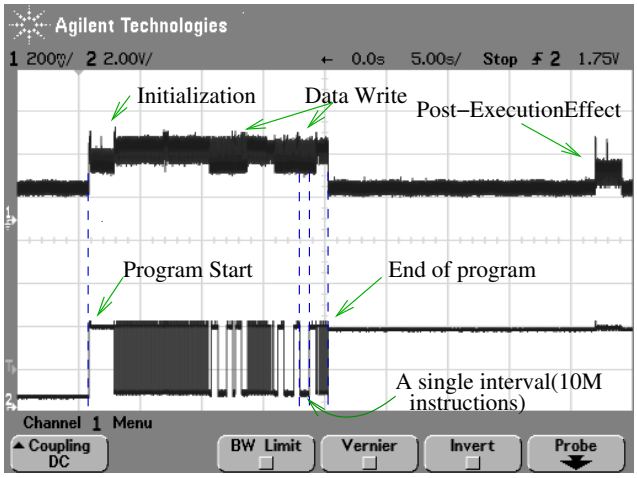


Figure 2: Power consumption of JPEGDecode on Ext2 file system. The top line shows the power consumption, and the bottom line shows the interval detection output pin voltage readings. The power phases (during the initialization and file write) are marked with an arrow. A post-execution effect, due to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

(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.

Figure 3 shows graphs for two representative benchmarks (GSMEncode and MPEGEncode). For each graph, we plot each average metric value for each interval in the program over its lifetime. We produce this data by executing the benchmarks on the EXT2 files system; the performance of these benchmarks using the RAM

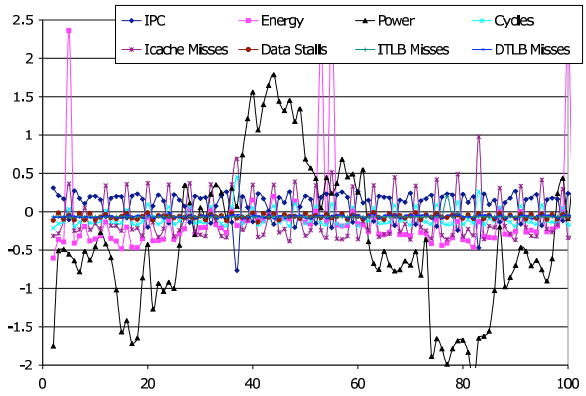


Figure 3: GSMEncode performance data from RPM over time for the first half of benchmark execution. We plot all metrics with energy and power. We normalize all data as described in Section 3.1.

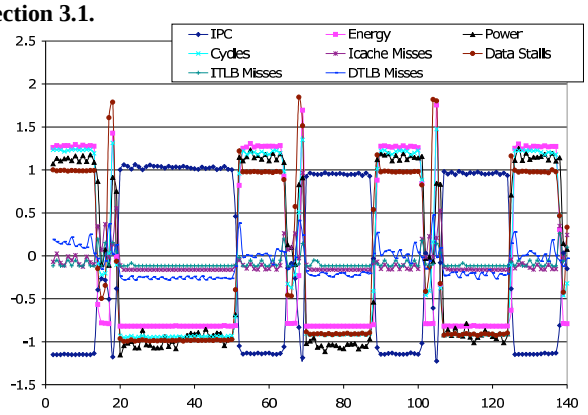


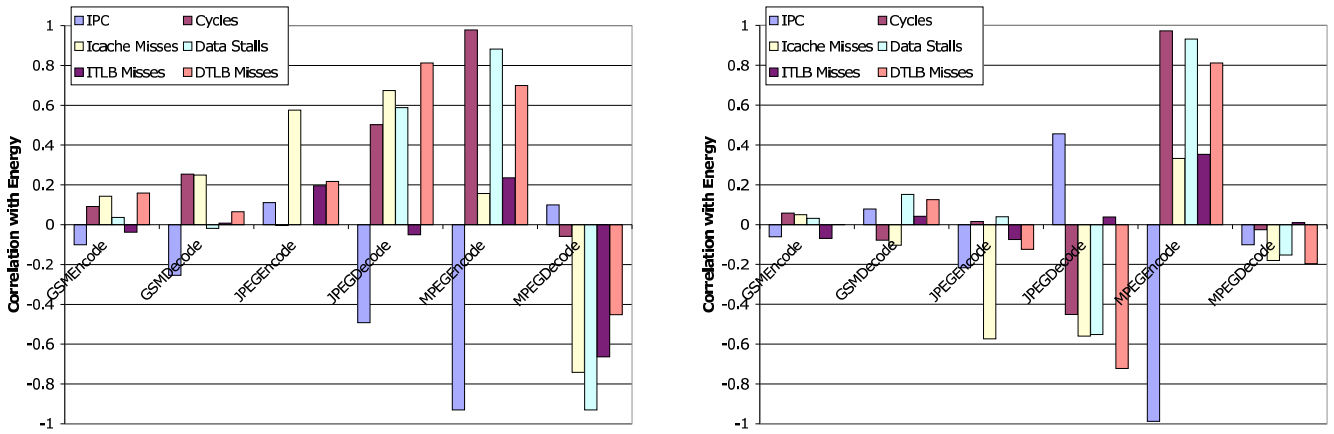
Figure 4: MPEGEncode performance data from RPM over the entire execution. We plot all metrics with energy and power. We normalize all data as described in Section 3.1.

drive is similar. For GSMEncode, we show only the first half of execution for clarity; the second half is also similar, however. We show data from the entire execution of MPEGEncode.

The data exhibits uniform behavior for GSMEncode and bi-modal behavior for most of the metrics in MPEGEncode. For GSMEncode, no HPM metrics appears to track power. However, it is unclear from the visual representation of the data whether other HPMs track energy. For MPEGEncode, both energy and power move in unison. Moreover, a few of the metrics appear to track this behavior either either directly or inversely (e.g., IPC).

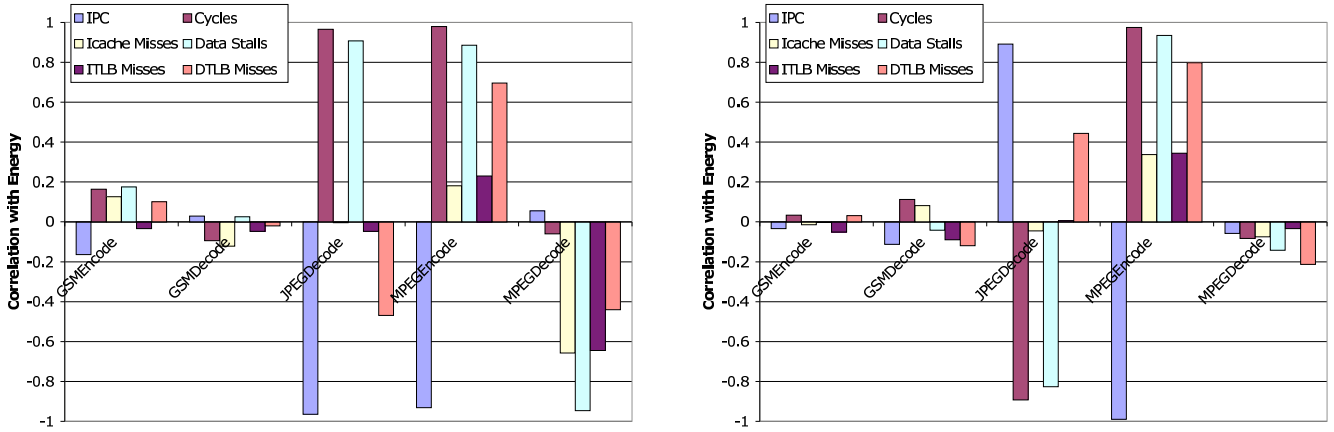
We next investigate the relationship between HPM behavior and that of power and energy more formally. To evaluate whether the HPM metrics explain power and energy variability, we computed the statistical correlation coefficients between the power dataset and each HPM dataset (likewise for energy). Correlation reports how well one dataset explains another. The resulting correlation value is between -1 and 1. Values near zero indicate very little correlation, while values at the other extremes indicate high correlation. A correlation of -1 means that as data in one set decreases, data in the other set increases similarly. A correlation of 1 means that the data in the two sets vary together in the same direction.

Figures 5 and 6 show the correlations for all of our benchmarks for each of the HPM metrics. Figure 5 shows the correlation for the EXT2 file system and Figure 6 shows the correlation of the RAM drive. The left graph in each figure shows the correlation of the



(a) VS Energy (b) VS Power

Figure 5: Correlation of HPM metrics for the EXT2 file system. Graph (a) shows the correlation of each metric with energy and (b) with power.



(a) VS Energy (b) VS Power

Figure 6: Correlation of HPM metrics for the RAM drive. Graph (a) shows the correlation of each metric with energy and (b) with power.

VS Energy	IPC	Cycles	ICache Misses	Data Stalls	ITLB Misses	DTLB Misses
Average	**-0.261**	**0.294**	**0.176**	**0.093**	**-0.052**	**0.250**
Min Name	MPEGEncode	MPEGDecode	MPEGDecode	MPEGDecode	MPEGDecode	MPEGDecode
Min	-0.930	-0.058	-0.741	-0.930	-0.663	-0.452
Max Name	JPEGEncode	MPEGEncode	JPEGDecode	MPEGEncode	MPEGEncode	JPEGDecode
Max	0.111	0.978	0.674	0.882	0.235	0.812
VS Power						
Average	**-0.139**	**0.082**	**-0.172**	**0.075**	**0.049**	**-0.018**
Min Name	MPEGEncode	JPEGDecode	JPEGEncode	JPEGDecode	JPEGEncode	JPEGDecode
Min	-0.987	-0.451	-0.573	-0.552	-0.075	-0.722
Max Name	JPEGDecode	MPEGEncode	MPEGEncode	MPEGEncode	MPEGEncode	MPEGEncode
Max	0.455	0.973	0.332	0.931	0.352	0.811

Figure 7: Correlation across benchmarks for the EXT2 file system. The table shows the average (row 2) as well as the minimum and maximum correlation across benchmarks. Above each minimum and maximum correlation value, we include the name of the benchmark for which the minimum or maximum occurred.

HPM metrics with energy. The right graph in each figure shows the correlation of the HPM metrics with power.

The data indicates that across benchmarks, for either storage device, the relationship between HPM metrics and power and energy is widely varied. GSMEncode shows the weakest relationship across all metric will all correlation values less than 0.2. All metric values that are near 0 (between -0.2 and 0.2) indicate very weak correlation.

For particular benchmarks, we can identify certain metrics that correlate well. For example, data stalls and data TLB misses correlate well with both energy and power for MPEGEncode. Other trends are less clear and are very particular to the benchmark and device type.

Across all benchmarks, the cycle based metrics (IPC and cycles – the first two bars for each benchmark) exhibit the largest correlation to energy. IPC correlates negatively for most programs, as is intuitive – an increase in IPC decreases energy or power. However, for JPEGEncode and MPEGDecode, IPC correlates positively. When a metric correlates positively with one benchmark and negatively for another, it indicates that the metric will be difficult to use accurately as part of a full-system power or energy model. That is, such metrics do not explain the variance in the power or energy data in the same way for all benchmarks. Most of the metrics produce such results (positive and negative values for different benchmarks).

We present the average correlation as well as the minimum and maximum correlations in Figures 7 (for EXT2) and 8 for (RAM). By showing the minimum and maximum, we can see which metrics produce both positive and negative correlations across benchmarks. The top half of each table shows the correlation of HPMs versus energy and the bottom half is versus power. The columns present the data for each metric. The first row of each section (top and bottom) shows the average correlation across all benchmarks for each metric. Below this, we show the minimum and the maximum correlation across benchmarks and identify the benchmark responsible for the minimum and maximum values. GSMEncode and GSMDecode are never the minimum or the maximum. This is because, all of the metrics are near 0 (uncorrelated) as the previous figures show.

Figure 9 shows the R^2 value which is also known as the *variation explained*. We compute this value by squaring the average correlation (shown in the previous tables) and multiplying by 100. This value indicates what percentage of the variance in the energy and power data, respectively, is explained by each HPM metric. The left graph shows the data 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 low 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, for the reasons we articulate above.

3.4 Further Analysis

In general, running the benchmarks on the RAM device reduces the variability in performance and energy data when compared to running the programs on the compact flash. Our evaluations indicate that the effect of the latter is most visible when large files are written. For example, the JPEGDecode, which generates an output file of almost $\approx 30MBytes$ uses 50% more energy when run on compact flash. This cost does not include the post-execution effect that we described above. Table 3 that we presented earlier, shows the differences in the average energy consumption for each benchmarks.

The MPEGEncode and MPEGDecode applications are the only benchmarks that include significant amount of floating point operations. The PXA-255 processor does not have a floating point co-processor thus, floating point instructions generates undefined instruction exceptions and the kernel acts as a software floating point emulator. As we are more interested in the applications effect on the whole system and to be able to compare our results across different processors and simulators such as Simplescalar, we decided to shut down performance counters during the undefined instruction exceptions. In this execution mode, the intervals that include heavy floating point operations are correctly identified as much more expensive (in terms of energy) phases than the others. Inevitably, each floating point instruction looks as if it is a conventional instruction with an extremely long clock cycle and much higher energy cost.

JPEGDecode and JPEGEncode, as mentioned previously, highly variable in their performance behavior. JPEGEncode reads a large file, and JPEGDecode writes a similar sized file, exploiting the behavior of file system. Figure 10 shows the energy consumption and instruction miss correlation for JPEGDecode (file write) and JPEGEncode (file read) on EXT2 and RAM devices. On datasets collected from EXT2 drive, the effect of heavy file system access (towards the initialization for JPEGEncode, and towards the final part for JPEGDecode) are easily visible. The increasing file system access increases the energy consumption. Furthermore, the I-Miss rate also increases with more file system calls due to more context switches, suggesting the reason for high correlation between these two parameters. Another interesting result is high negative correlation of energy and IPC. This is also due to the same reason: A high number of instruction misses reduces IPC, so more time and energy is needed to complete the execution of an interval. The JPEGEncode on EXT2 is not effected in the same way.

MPEG (both Encode and Decode), by far, exhibit the largest correlation between HPM metrics and both power and energy. That is because MPEG shows large variations in energy consumption: One reason is the existence of software floating point emulation. When these intervals are entered, the energy cost of an instruction suddenly explodes, however, IPC decreases, introducing large negative correlations. Another reason is the characteristics of MPEG. MPEGEncode, reads an input frame, encodes, writes the encoded data and moves to the next frame. The investigation of run-time data reveals very efficient processing phases where instruction cache and TLB misses are almost zero, and data stalls are at a minimum. Consequently, these intervals have the largest IPC, and smallest energy consumption. Even though a reduction in energy consumption appears nonintuitive, it is possible since each interval is 10 million instructions, any event increasing the IPC rate, can reduce the energy consumption.

3.5 Using RPM to Study Phase Behavior

We also use RPM to evaluate the efficacy of code-based phase characterization in capturing power and energy phases. A phase characterization attempts to group periods with similar execution characteristics together so that observed behavior is uniform within a phase and each phase represents a distinct behavior in the program's execution. Much prior research has focused on capturing and exploiting phase behavior, especially in runtime prediction and optimization. Recent studies show that phase behavior captured at the basic block level is indicative of a variety of execution characteristics at the architectural level. We perform a preliminary investigation to study the correlation between code-based and energy phases in this section.

As a first step, we generate similarity matrices from the per-interval basic block vector trace and per-interval energy measurements. Similarity matrices present a visual representation of time

VS Energy	IPC	Cycles	ICache Misses	Data Stalls	ITLB Misses	DTLB Misses
Average	**-0.395**	**0.391**	**-0.095**	**0.209**	**-0.109**	**-0.026**
Min Name	JPEGDecode	GSMEncode	MPEGDecode	MPEGDecode	MPEGDecode	JPEGDecode
Min	-0.964	-0.094	-0.657	-0.947	-0.645	-0.469
Max Name	MPEGDecode	MPEGEncode	MPEGEncode	JPEGDecode	MPEGEncode	MPEGEncode
Max	0.056	0.979	0.181	0.907	0.230	0.696
VS Power						
Average	**-0.060**	**0.029**	**0.057**	**-0.016**	**0.035**	**0.188**
Min Name	MPEGEncode	JPEGDecode	MPEGDecode	JPEGDecode	GSMEncode	MPEGDecode
Min	-0.989	-0.893	-0.075	-0.826	-0.089	-0.213
Max Name	JPEGDecode	MPEGEncode	MPEGEncode	MPEGEncode	MPEGEncode	MPEGEncode
Max	0.891	0.975	0.338	0.935	0.345	0.798

Figure 8: Correlation across benchmarks for the RAM drive. The table shows the average (row 2) as well as the minimum and maximum correlation across benchmarks. Above each minimum and maximum correlation value, we include the name of the benchmark for which the minimum or maximum occurred.

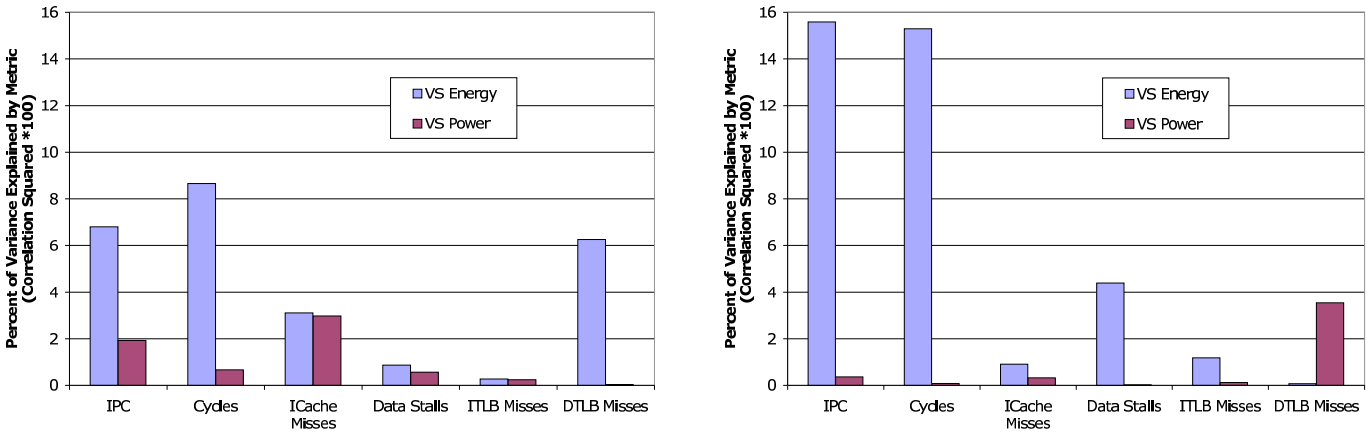


Figure 9: R^2 Correlation Statistic: Correlation squared times 100. This value shows the percent of the variability in energy (first bar) or power (second bar) that is explained by the metric (x-axis). (a) shows the data for the EXT2 file system; (b) shows the data for the RAM drive.

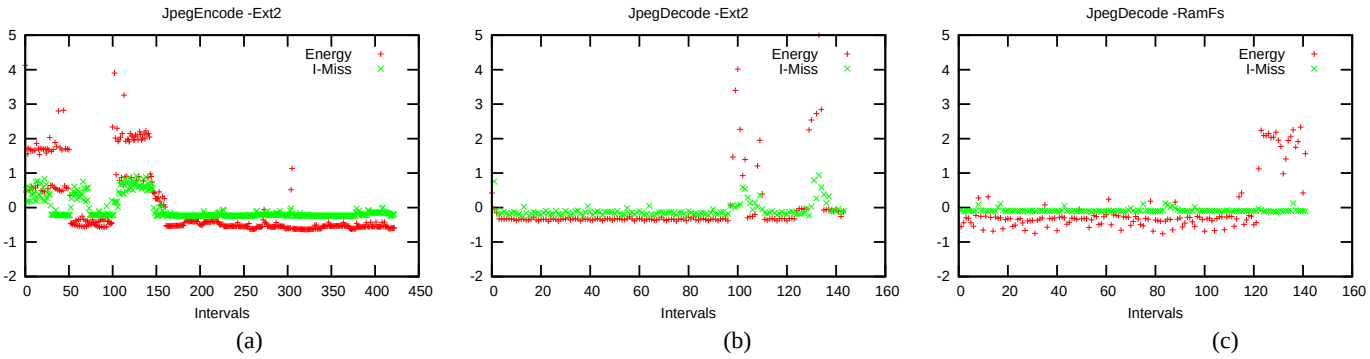


Figure 10: Correlation of Instruction Miss and CPU energy Consumption. JPEG was the only application that showed a correlation between energy consumption and instruction cache misses. Surprisingly this correlation is very obvious when we run the application on EXT2, but non-existent when we execute in RAM, (a) and (b) graphs, respectively. Most instruction cache misses occur during heavy file system read/write operations. (c) File system write operation has almost no effect on instruction cache misses.

varying behavior. Each entry in a row or column represents an interval. We list intervals in each row or column in the order in which they occur in the program. An entry in the matrix at position (x,y) is a pixel colored to represent the similarity between interval x and interval y (black is similar, white is completely dissimilar). Figure 11(a) shows the similarity matrices for four benchmarks (one per column). The first row of is the BBV-baseed matrices; the second row shows those computed using real energy data. GSMEncode and GSMDecode show very little variation for either type of matrix. JPEGEncode and JPEGDecode however, do show some differences; in particular, the energy matrices show more detail and resolution, i.e., differences between intervals. This is because the differences in energy data is more pronounced than the differences in the code executed during the intervals in the last 1/3 of execution for JPEGDecode and the initial 1/4 of execution for JPEGEncode. The changes in energy are due to the use of the file system for file writes and reads for JPEGDecode and JPEGEncode, respectively, as we explained in the previous section.

We use the basic block distribution analysis based technique described in [21] to generate code-based phases given an interval length of 10 million instructions. We use the Simplerscalar simulator to generate per-interval basic block vector traces, and the Simpoint framework [22] to obtain a phase classification. To generate energy phases, we cluster per-interval RPM power measurements using the K-means clustering algorithm. The method above has one caveat. Even though XScale and Simplescalar use ARM ISA, the dynamic instruction counts in XScale and Simplescalar are not 100% compatible. Furthermore due to the OS overheads (even though the HPMs are disabled during interrupts and scheduling, it is not possible to eliminate all the overhead), XScale counters tend to increase a little faster. For most of our benchmarks, we found the variation in instruction counts to be within 1%. We exclude MPEG (encode and decode) from this study since its use of floating point operations yields a larger variation across the simulated environment and the real-time measurement system. For the four benchmarks studied, we find a fixed number of phases (3 – chosen arbitrarily) using both the basic-block-vector and K-means clustering techniques. In addition to visual correlation, we quantify the difference between the two by computing the error in estimating measured power using basic-block-vector phases. The table in Figure 11 lists the percentage error in estimating power using basic-block vector phases in column three. The error is a measure of the deviation of estimated energy from measured energy and is omputed as: $PowerDev_b$, as:

$$Error = \sqrt{\frac{\sum_{i=1}^{N}(P_i - R_{ji})^2}{N}}$$

where P_i is the measured energy for interval i, R_{ji} is the representative energy for the phase that intervals i belongs to. The representative energy for a phase is computed as an average over all intervals belonging to that phase. The table also provides details about the three phases found by each of the two clustering schemes. We can see that basic-block-vector based phases and energy phases not only yield different clusterings, but also differ in the characteristics of phases found. An important point to note is that, although the estimation error is very low, basic-block-vector based phases do not necessarily yield distinct phases in terms of energy behavior although variations in energy behavior do exist.

4. RELATED WORK

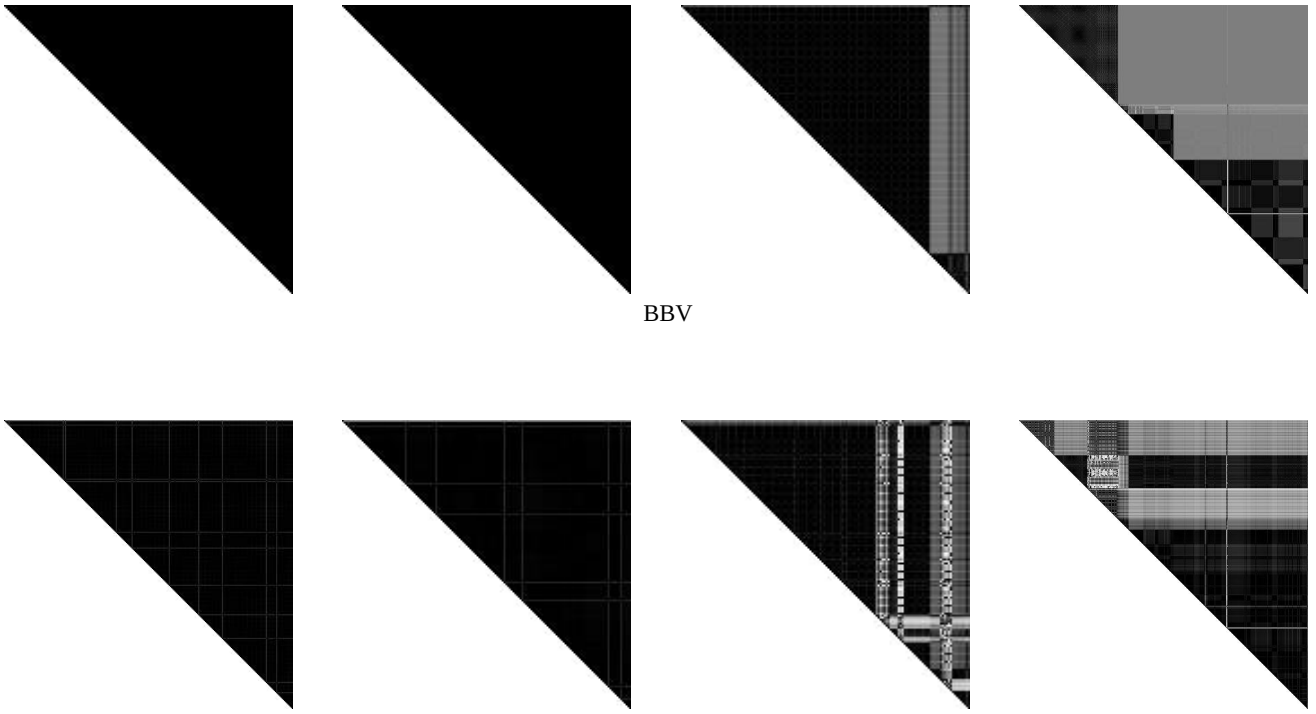
The work that we describe herein is a remotely accessible toolset for highly accurate power and energy measurement and CPU-based profiling of a target device. The toolset consists of an Agilent high-end deep-memory oscilloscope, an Agilent programmable power supply, and a software system that we developed to enable remote access and programmable experimentation, that couples of power and energy profile collection with HPM profile data. The target device that we currently have plugged into the RPM is a Crossbow Stargate sensor network intermediate node. We are currently working on supporting a very similar device, the HP iPAQ hand-held. The primary difference between these devices is the LCD display and user interface (key pad, buttons, and touch screen).

We show the utility of the toolset and its importance for resource-constrained and battery powered systems by using it to study currently open research questions in this area. In particular, we investigate how well HPM and phase behavior data correlate with power and energy data. In addition, we investigate the impact of memory devices and file system implementations on power and energy.

The most closely related work in this area include systems that employ HPM data to estimate power and energy behavior and techniques for measurement and characterization of power and energy behavior using real devices. To our knowledge, extant approaches to measurement and characterization of power and energy behavior are different from ours in that they either consider only high-end processors (e.g., the Intel Pentium class), or do not investigate the full-system power consumption (i.e. they consider only CPU power consumption). Another key difference is that many prior studies use the terms power and energy interchangeably and focus primarily on power. We focus on popular resource-constrained devices with energy-efficient processors and show that it is important to measure, understand, and characterize both to understand full system battery consumption. No extant system provides remote access for highly programmable experimentation with a set of remote power and performance profiling tools.

There is much work that makes use of HPM data to estimate, characterize, and optimize performance (as opposed to power), e.g. [8, 19, 1]. However, those that consider how HPM data correlates with processor power and energy consumption is more related to our research and goals. Bellosa et. al [24] first proposed the use of of CPU event counters to estimate power consumption and to guide dynamic voltage scaling. The authors focus on memory and CPU power only; these are the only components that they can monitor via the counters. The authors make two observations: (i) Many memory requests per second indicate heavy use of memory, so energy performance will benefit if CPU speed is reduced; and (ii) The IPC indicates the sensitivity for performance loss. If the IPC is low, the thread will be less sensitive to clock speed reduction. The authors propose a modular system that samples HPM information every timer interrupt and then uses the policies above for processor voltage switching. The support their case, they measured the energy consumption of the CPU (using a current meter attached to the CPU) for several applications, using different speed levels ranging from 333MHz, 400MHz, 466MHz, 533MHz, 600MHz and 733 MHz. They maintain the same speed throughout the lifetime of the program (i.e. they do not switch voltages during program execution). Their results show that most applications experience high performance losses with lower CPU speeds.

Many other prior studies construct models of CPU power consumption using HPM data. In [16], Kadayif et al. describe vEC, a model that estimates energy consumption of memory on UltraSPARC CPU. vEC uses HPMs to determine rate of use of each component in the memory hierarchy, including as the bus, cache, and main memory. They estimate energy using the analytical model defined in [17] which models the energy consumption of CPU components with an average 2.4% error relative to a circuit-level sim-



(a) Similarity Matrices

Benchmark	Total Energy	% BBV Estimation Error	Phase	BBV		Measured	
				# Intervals in Phase	Average Energy per Interval	# Intervals in Phase	Average Energy per Interval
			1	120	0.0593	11	0.0728
			2	62	0.0598	82	0.0597
gsmencode	15.254	0.025	3	73	0.0599	163	0.0586
			1	33	0.0686	1	0.1637
			2	25	0.0690	7	0.0777
gsmdecode	11.144	0.069	3	102	0.0695	153	0.0682
			1	93	0.1015	52	0.1958
			2	227	0.1884	310	0.1059
jpegencode	63.118	0.114	3	100	0.1048	62	0.3243
			1	87	0.1539	130	0.1367
			2	19	0.3544	4	0.9125
jpegdecode	26.374	0.546	3	35	0.1650	10	0.4953

(b) Power Estimation using BBV Phases

Figure 11: (a) Visualization of bbv and energy phases using similarity matrices. The interval length used is 10 million instructions. The number of intervals, n, vary and each graph is a n x n matrix with the x and y axes representing the interval identifier. The lower triangle is a mirror image of the upper one and is masked for clarity. Each point on the graph indicates the similarity between the intervals represented by that point. Dark implies similar and light implies dissimilar. The diagonal is dark, since every interval is entirely similar to itself.

ulation. Similarly, the authors in [15], describe a general scheme for estimating runtime power of the CPU and its constituent components using HPMs for the Pentium Pro processor. The authors measure power via an external multi-meter and shunt resistor attached to the CPU of the device.

Isci et al [12] build upon this scheme to estimate the power consumed by a Pentium-IV processor. They identify 22 components of the processor and estimate power cost of each of the component using external power measurement tools attached to the processor. They use the HPMs to compute the rate of use of each component. The sum of power consumption of each component gives the power consumption of CPU. Bricher et al [3] proposes a similar, but much simpler model that uses only two performance counters to estimate power consumption of Pentium-IV processors. As a high-end processor, Pentium-IV offers a large set of event counters (i.e. 18 counters and 59 event classes). Embedded processors are much more limited in terms of the number of counters supported. This increases the difficulty of establishing an accurate power estimation model. Contreras et al [4] investigates different model for CPU and memory on a XScale development board. Their experiences show that a model using five HPMs can correctly estimate the power. However, they also show that the error in their model can be as high as 70%. This error is due to inaccurate modeling of memory accesses. Moreover, this model uses CPU HPM data to model CPU power consumption, not full system power consumption.

In [12], a runtime power monitoring methodology was proposed for runtime microprocessors. Even though this setup is similar to ours in terms of design; there are a couple of significant differences. (i) The proposed system is designed for CPU power monitoring in high-end systems, whereas our system is designed for monitoring the energy consumption of the whole system in embedded devices. (ii) The proposed methodology collects HPM data at run-time from the monitored system, via ethernet. The run time data collection and network connection perturb power, energy, and performance, which is undesirable. Since the authors only monitor and estimate CPU power consumption using a very fast processor, this perturbation can be negligible. In our setup, we are monitoring the energy consumption of a whole system, thus, the overhead of such an online monitoring system is not acceptable. The same authors, extended this system in [13, 14], to evaluate the correlation between the CPU power consumed and that estimated by HPM and phase profiles. Again, the studies are limited to CPU power only, not full system power. They show that HPM-based estimations produce accuracy errors of 2-7% and that phase-based estimations produce errors of 3-12%.

Other work on phase-based power characterization have been proposed to reduce energy via phase-guided dynamic cache and processor bus width reconfiguration [23] and phase-aware remote program profiling [20]. In the latter, we sample the program once per phase to reduce profiling overhead. RPM, described herein, can also do the same (sample the intervals of interest) via its user interface. The user must specify which intervals to sample (collect data for power, energy, and HPM performance). The user can generate this interval list using our tools from this prior work. We use these tools to do so to investigate the similarities between code-based phases (i.e. periods of stability) and phases in power and energy data. Most prior work on phase behavior employs simulation to measure, estimate, and evaluate the efficacy of phase characterization [20, 23, 22, 7, 6]. Our work herein, uses only real data, collected online, using a remotely accessible, programmable, highly accurate and and low overhead measurement system called RPM.

5. CONCLUSIONS AND FUTURE WORK

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. Moreover, an RPM user can evaluate the programs using any of the valid five clock configuration options.

We use RPM to investigate a number of open research questions regarding the correlation between CPU-based metrics and power and energy consumption. The relation between HPM statistics and CPU and memory power consumption has been investigated many times, however, the correlation between HPM monitors and the energy consumption is generally overlooked.

We find that HPM metrics do not correlate well with full-system energy and power consumption for most benchmarks. We also find that it is important to consider system characteristics such as file system type, system calls, I/O device types, etc., to capture energy and power behavior. We also investigate how well code-level phase identification maps to the phases in power and energy. We find that there are many more phases identified by code-level techniques than are actually exhibited by full-system energy performance.

As part of future work, we are adding additional devices to RPM. We are currently extending RPM to support two different types of HP iPAQ handheld. We will provide a mechanism to switch between the devices as part of the web experiment request. RPM supports any device that runs Familiar Linux. Since RPM couples power and performance monitoring, devices that export hardware performance monitors exercise all of the RPM functionality. We are interested in understanding the full-system power behavior of handheld with LCD displays and other types of I/O devices.

Also as part of future work, we are extending RPM to enable automatic dynamic clock scaling and using RPM to investigate a number of interesting questions. The most important is to identify which set of device behaviors does correlate well with full system energy and power. We are currently developing methodologies to characterize the systems that infer application characteristics to manage global resources. One such system is AutoDVS [9] that we developed in prior work. AutoDVS uses application interactivity to manage CPU clock speed. We are interested in combining these behaviors and effectively predicting future patterns in full-system power and energy consumption.

Acknowledgments

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

6. REFERENCES

[1] A. Adl-Tabatabal, R. Hudson, M. Serrano, and S. Subramoney. Prefetch injection based on hardware monitoring and object metadata. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, June 2004.

[2] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An Infrastructure for Computer System Modeling. In *IEEE Computer*, February 2002.

[3] 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.

[4] 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.

[5] R.M. Dell and D.A.J. Rand. *Understanding Batteries*. RSC Paperbacks, 2001.

[6] A. Dhodapkar and J. Smith. Managing multi-configuration hardware via dynamic working set analysis. In *29th Annual International Symposium on Computer Architecture*, May 2002.

[7] A. Dhodapkar and J. Smith. Comparing program phase detection techniques. In *36th Annual International Symposium on Microarchitecture*, December 2003.

[8] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *International Conference on Parallel Architecture and Compilation Techniques*, September 2003.

[9] S. Gurun and C. Krintz. Autodvs: An automatic, general-purpose, dynamic clock scheduling system for hand-held devices. In *ACM SIGBED International Conference on Embedded Systems Software (EMSOFT)*, September 2005.

[10] Intel. *StrongARM SA-1110 Microprocessor Developer's Manual*, October 2001. Order Number:278240-004.

[11] Intel Corporation. *Xscale*. www.intel.com/design/intelxscale/.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] H. Kim, N. Vijaykrishnan, M. Kandemir, and M. Irwin. Multiple access caches: Energy implications. In *IEEE CS Annual Workshop on VLSI*, April 2000.

[18] C. Krintz, Y. Wen, and R. Wolski. Application-level Prediction of Battery Dissipation. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, August 2004.

[19] Jiwei Lu, Howard Chen, Rao Fu, Wei Hsu, Pen-Chung Yew, and D. Chen. The performance of runtime data cache prefetching in a dynamic optimization system. In *International Symposium on Microarchitecture*, December 2003.

[20] Priya Nagpurkar, Chandra Krintz, and Timothy Sherwood. Phase-aware remote profiling. In *ACM Conference on Code Generation and Optimization*, March 2005.

[21] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *International Conference on Parallel Architectures and Compilation Techniques*, September 2001.

[22] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *10th International Conference on Architectural Support for Programming Languages*, October 2002.

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

[24] 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.

[25] Y. Wen, R. Wolski, and C. Krintz. History-based, Online, Battery Lifetime Prediction for Embedded and Mobile Devices. In *Workshop on Power-Aware Computer Systems (PACS)*, April 2003.

[26] Y. Wen, R. Wolski, and C. Krintz. Online Prediction of Battery Lifetime for Embedded and Mobile Devices. *Special Issue on Embedded Systems: Springer-Verlag Heidelberg Lecture Notes in Computer Science*, 3164(2004), 2004.