Toward an Evaluation Infrastructure for Power and Energy Optimizations

Chunling Hu Daniel A. Jiménez Ulrich Kremer
Department of Computer Science
Rutgers University
Piscataway, NJ 08854
chunling, djimenez, uli@cs.rutgers.edu

Abstract

Execution-driven simulators are often used for power/energy and performance evaluation. Simulators can provide semantic details but they provide insufficient speed and accuracy for compiler and OS research. Physical measurement is fast and objective but lacks a semantic connection between the measurement result and the evaluated program. The objective of our research is to bring together the advantages of simulation and physical measurement to build an infrastructure for power and energy optimization. Power and energy behavior is obtained through physical measurement. Simulation is used for observing the connection between power and energy behavior and the evaluated program. Our preliminary results demonstrate the ability of this infrastructure to capture detailed power behavior of any region of a program. To simplify the power/energy evaluation of programs with long execution times and overcome the limitation of physical devices, we propose using the SimPoints methodology developed by researchers at UC San Diego to find representative slices of a program. Through simulation, we validate the feasibility of the SimPoint idea in simplifying power/energy evaluation. We expect that this infrastructure will help researchers in OS/compiler power/energy optimization to evaluate their optimizations more efficiently and observe more optimization opportunities.

1. Introduction

Process technology increases transistor density by about 35% per year. The effect of increasing die size and transistor density causes a growth rate in transistor count of about 55% per year with a commensurate improvement in transistor performance. High transistor density has supported the emergence of 64-bit microprocessors and many innovations in pipelining and caches [11]. At the same time, power management has emerged as a major challenge for device scaling. Most of the power dissipation of CMOS microprocessors comes from the switching (dynamic) power of transistors, which can be calculated as

$$P = f * C * V_{dd}^2 \qquad (1)$$

Here, f is the switching frequency, C is the load capacitance of the transistor and V_{dd} is the voltage. Decreased feature size of a transistor means a decrease in its load capacitance. However, the increases in the number of transistors and frequency dominate the decreases in load capacity and voltage, resulting in power dissipation growth. Power is likely to become the major limitation in the development of computer architecture [11].

Simulators are often used to evaluate the power and performance of programs. By simulating the execution of a program on some platform, we can get very detailed program behavior in both power and performance. Simulation is very important in the early stage of computer architecture design since no concrete architecture is available. However, simulators are very slow and inaccurate compared with physical measurement. In the OS and compiler communities, benefit analysis typically rely on physical measurement in order to get objective power/energy behavior with time cost proportional to program execution time. However, physical measurement results alone are often not able to explain the observed power/energy behavior, i.e., do not provide a semantic feedback. It is our goal to bring together the advantages of simulation and physical measurement to build an evaluation infrastructure for OS/compiler power/energy optimizations. By combining the results of physical measurement and simulation, the user can get an objective power/energy measurement with a semantic connection to the evaluated program.

For long-running programs it is hard to get very detailed power behavior of the whole program through physical measurement because of hardware limitations. Even with energy simulation, it is time-consuming to simulate the whole program. We plan to use the concept of SimPoints [15] in our infrastructure to overcome hardware limitations and simplify the power/energy measurement/simulation. That is, we find representative slices of a program and perform measurement/simulation only on these slices to get the power/energy behavior of the whole program.

The rest of the paper is organized as follows: Section 2 describes power/energy simulators and the SimPoint idea. Section 3 discusses our current evaluation infrastructure and preliminary results. Section 4 describes the validation of the feasibility of the SimPoint idea in power/energy evaluation through simulations. Section 5 gives directions for future work, and Section 6 concludes the paper.

2. Related work

2.1. Power Evaluation Techniques

In simulation-based power evaluation methods, the system is abstracted into various components. The energy consumption of a program is estimated as the sum of the energy consumption of all these components. Simulators can be classified according to their levels of system and component abstraction. They target different levels of detail and make different trade-offs between simulation time and accuracy. Most simulators are parameterized so they can be used to estimate the energy consumption of different system configurations. Simulators are very important in the early stages of architecture design and evaluation. Furthermore, many simulators can give details at a very fine level of semantic granularity.

2.1.1. Transistor-level These simulators characterize models of transistors and estimate voltage and current behavior over time. Power dissipation of transistors comes from three sources: switching power, short-circuit power, and leakage power. Such simulation is time-consuming but useful in integrated circuit design. Transistor-level simulators are not suitable in evaluating power consumption of large programs on complex systems.

2.1.2. Cycle-accurate Microarchitecture-level A microarchitecture-level simulator simulates the execution at the level of individual cycles, allowing to keep track of power behavior changes across cycles. They are often used for simulations of modern superscalar processors. Three examples of cycle-level simulator are Wattch [5], SimplePower [18] and Sim-Panalyzer [1]. We use Sim-Panalyzer in our research to estimate the power dissipation of the benchmarks.

2.1.3. Instruction-level Instruction-level simulators provide coarser power behavior than the above two. The simulation is based on the instruction-level energy profiling of the instruction set of the target processors and the assumption that the energy consumption of an instruction is mostly independent of the addressing mode or operands. Instruction-level simulators are normally faster than cycle-level simulators and useful when only total energy consumption is needed. One instruction-level simulator is JouleTrack [17].

2.1.4. System-level Hardware component system-level simulators characterize the energy consumption of each system component in different states. The simulator records the transitions between states and the time each component spends in each state during the simulation and calculates the energy consumption of the whole program. Such simulators do not provide detailed power behavior of a program, but they are useful for component selection and system partitioning during an architecture design. An example is the simulator from Duke University [6]. It is an extension of POSE, a palm OS Simulator.

There are also some software component system-level evaluators. PowerScope [8] is a time-driven statistical sampler that uses samples from a digital multimeter. An energy-driven statistical sampler from Compaq is similar to PowerScope except that the sampling period is determined by energy quanta.

SoftWatt [10] is the first simulator to target the complete system power profile of a high-end system. It extends SimOS [14] with validated analytical energy models for hardware components. This simulator identifies power hotspots in system components, and captures the relative contribution of the power profile to the user and kernel code and identifies power-hungry OS services.

ECOSystem [19] is a modified Linux that manages energy as an OS resource. Parameters of its "currentcy model" can be changed to support different platforms.

Isci *et al.* [12] propose a coordinated measurement approach that combines real total power measurement with performance-counter-based per-unit power estimation. This is useful for dynamic power/energy management but incurs runtime overhead due to the access of performance counters. This overhead may be responsible for a significant portion of the observed power dissipation, in particular for short programs or program regions. Our work is different from this approach because our objective is an evaluation infrastructure for OS/compiler optimizations. Our infrastructure can get the power measurement of any small region of a program as well as the power behavior of a long program. Furthermore, even through [12] provides power breakdown for CPU components, their is no semantic connection between the measurement result and the measured program, which is important for observing power/energy optimization opportunities and will be an important contribution of our infrastructure.

2.2. Disadvantages of Energy Simulators

The above simulators have some common features. Energy models for various components are characterized before the evaluation and energy consumption evaluations are done through looking up values in many tables by the simulator. The higher the precision is, the larger the tables are. So speed is usually decreased with the increase in precision. Simulator are valuable for power and energy estimation of unavailable architectures. For OS and compiler level power optimization on available architectures, physical measurement can be used for evaluation.

Performance modeling is subject to many sources of error [4]. *Modeling* errors are from the incorrect coding of the desired functionality. Desikan *et al.* measured the experimental error in microprocessor simulation and showed that the error in common simulators is often larger than the performance gains yielded by new architecture ideas reported in the literature [7]. From the construction of power simulators, we can see that power simulators are also subject to the errors mentioned in [4]. Some tables are usually simplified to accelerate simulation. There may be mismatches between reality and the simulation of the program execution. The effect of the OS is not considered in many simulators. All of these issues make accuracy a problem of simulators. Ghiasi *et al.* compared two architectural power models, the Cai-Lim power model and Wattch, and found that these models disagree on the efficacy of the design choices in each experiment and do not always produce statistically significant results [9].

The disadvantages of power simulation and physical measurement show that we need a faster, more precise power and energy evaluation infrastructure to correctly reflect the power behavior of a program and evaluate the benefit of an optimization. This is the motivation of our research.

2.3. The SimPoint Idea – Off-line Phase Clustering

Execution of a program falls into repeating behaviors called *phases*. If we can identify these phases then we can find segments of the program execution with similar behavior and focus our simulation or measurement only on several representative segments to get the behavior of the whole program execution. Sherwood *et al.* proposed off-line Phase Clustering Analysis and used it in finding simulation points of programs in their SimPoint research [16].

In SimPoint, a program execution is partitioned into intervals with a fixed number of instructions. Intervals with similar behavior are clustered into a phase. Basic Block Vector(BBV) profiling is used to collect a sort of fingerprint of each interval for off-line classification. This fingerprint is the number of instructions executed for each basic block during the interval. The phase behavior can be found by examining the ratios in which different regions of code are executed over time [15]. One cluster corresponds to a phase. One interval for each phase is chosen as a simulation point and the program behavior can be estimated from the simulation of all the simpoints. They validated this method on the SPEC2000 benchmarks by estimating their instructions-per-cycle (IPC), cache miss rate, branch misprediction rate *etc*[2].

We plan to borrow the SimPoint idea to simplify our physical measurement and enable the physical measurement of programs with long execution times in spite of the hardware limitations. We have validated the feasibility of this idea in power/energy evaluation through simulation on some benchmarks.

3. Evaluation Infrastructure

3.1. Current State of the Infrastructure

As depicted in Figure 1, our evaluation infrastructure has three components: a Skiff board(a Compaq Personal Server PCB with a StrongARM SA110 CPU and 32MB of SDRAM), a Tektronix TDS3014 DPO oscilloscope with a TDS3TRG advanced trigger module, and a dedicated data-acquisition Linux machine. The Skiff board has separate power planes and current measurement points for CPU and memory. The measured program runs on the Skiff board. The oscilloscope measures the current or voltage of the components(CPU, memory) of the Skiff board or the whole board. The data-collecting machine communicates with the oscilloscope to gather data and does offline analysis. Sampling is done by the oscilloscope and the data-acquisition ma-

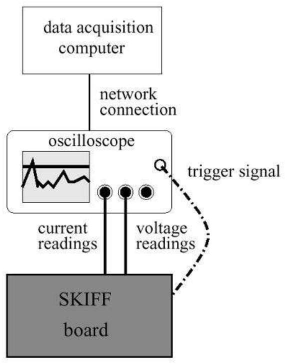


Figure 1. Prototype power measurement infrastructure for the StrongARM based Skiff board

chine only communicates with the oscilloscope.So there is no interference between the measured program, sampling and data collecting. The accuracy of the result is high.

3.2. Usage and Measured Parameters of Oscilloscope

The TDS3014 oscilloscope has four channels. Users can control its acquisition mode, record length, trigger mode, data encoding and other configurations through our user-level API. TDS3014 has two sample acquisition modes:

- Normal : record length=10,000 points
- Fast trigger: 500 points

Record length is the number of points that comprise a complete waveform record. It determines the amount of data that can be captured with each channel. In our initial work, we use fast trigger mode to collect the power behavior of the whole measured program. There are two problems: 1) The oscilloscope keeps acquiring samples all the time. It is hard to determine the beginning and the end of the measured program. 2) The communication cost to collect 500 samples from the oscilloscope to the data-acquisition machine is much longer than the time used to generate these samples when high resolution is used. In our experiments, the communication cost is usually about 135ms. This means, that if we adjust the oscilloscope to generate 500 samples in 135ms, the communication speed of the data acquisition machine is fast enough to retrieve all data from the buffer before the buffer is overwritten. For a 233Mhz machine such as the Skiff board, each sample covers about 62910 cycles. This resolution may not be good enough if we want to have a closer look at the power behavior of the measured program. However, if we use a higher sampling rate, some samples will be lost due to the overwriting mechanism of the sample buffer.

To solve the first problem we use the trigger module of the oscilloscope. In this mode, the oscilloscope starts sample acquisition only after a trigger event happens. The acquisition is stopped after an entire record is obtained. The trigger starting the measurement is generated by setting a pin on the Skiff board to a high voltage. We call this pin the *trigger pin*. After the measured part, the *trigger pin* is set to lower voltage to indicate the end of the measurement. Through running some specially designed micro-benchmarks, we determined that the delays of setting the high voltage and low voltage are 20ns and 47ns, respectively. Our current prototype user interface allows the specification of program regions by inserting compiler pragmas to mark the start and end of the region to be measured. Trigger generation code is inserted into the source code automatically. The execution of the measured program region is not affected, allowing *non-intrusive* power/energy observations.

Although the *normal* mode can help us identify the beginning and the end of the measured program, it is still hard to get the power behavior in high precision for programs with long execution time due to the high communication cost. A larger memory for the oscilloscope can help, but the problem persists if we want higher precision. This is why we plan to use the SimPoint idea to find representative slices of a program and get power evaluation for the whole program based on the physical measurement of the selected slices.

3.3. Measurement Example

Figure 2 shows a very simple program with a loop. Unrolling the loop 8 times and compiling it with GCC yields a new version, **version A**, which has a basic block with 16 loads followed by 16 additions. We reschedule the instructions of the assembly of **version A** by hand to get two other versions, **version B** and **version C**. Then we use GCC to compile them into executables. Table 1 shows the instruction schedule for each version.

Figure 3 shows the measured CPU current for the StrongARM SA110 processor (2.0V, 233MHz). The line marked "trigger" represents the trigger signal for the oscilloscope. At the beginning and end of the program region of interest, the *trigger pin* is set to high and low voltage, respectively.

In Figure 3, we see what we expect for the current behavior of the program: one cache miss every 8 iterations of the original version. **Version C**, the alternating schedule of memory and CPU instructions, leads to the short-

Version	Instruction Order
Original	2 loads followed by 2 additions in each loop, but 8 times of loops compared to other versions
Version A	16 loads followed by 16 additions
Version B	2 loads, followed by 2 additions, followed by 14 loads, followed by 14 additions
Version C	alternating groups of 2 loads, followed by 2 additions

Table 1. Instruction order of each version of the loop

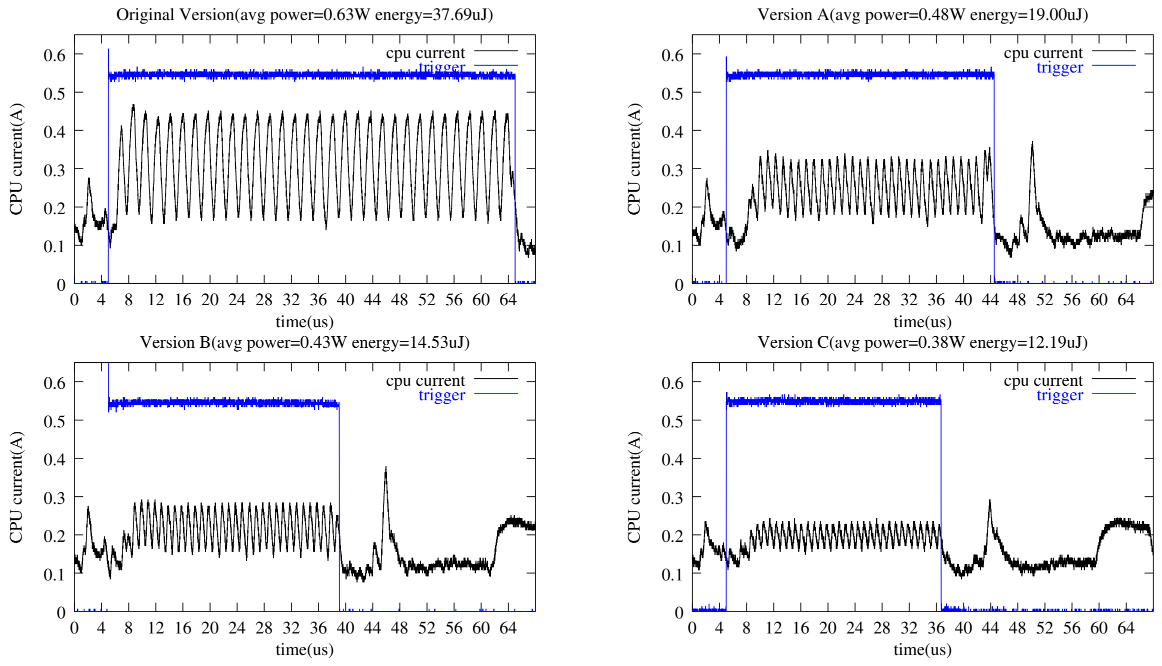
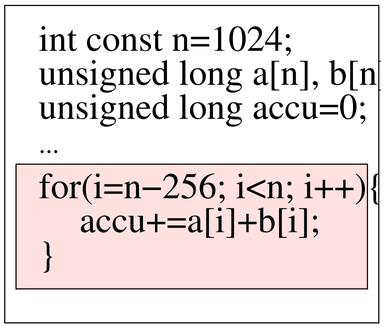


Figure 3. Physical measurement results for the four versions in Table 1.



```
int const n=1024;
unsigned long a[n], b[n
unsigned long accu=0;
...
for(i=n−256; i<n; i++){
    accu+=a[i]+b[i];
}
```

Figure 2. A simple program with loop. The part in the gray box is measured.

est execution time, the lowest energy consumption and the smoothest power dissipation profile. Choosing this schedule over the alternatives will lead to a fast program with low peak CPU power dissipation and small variations in power dissipation.

For comparison, we also simulate the same programs using Sim-Panalyer, a cycle-accurate architecture-level ARM power simulator. Since it is non-trivial to identify loops within an executable, we simulate the power dissipation of the whole program. Only the loop is different in different simulations, so we can say that the difference among simulation results are from our modification to the loop. Most of the architectural configuration values used in our simulation experiments are from the StrongARM v4 data-sheet. Others are from the default configuration provided by Sim-Panalyzer. We use the power configuration file provided by Sim-Panalyzer but change the frequencies to 233Mhz. Table 2 gives the

Version	power	cycles
Original	1.000	1.000
Version A	0.653	0.644
Version B	0.668	0.684
Version C	0.667	0.683

Table 2. Simulated power and cycles for the unrolled loop in Figure 2.

simulation results for the four versions of the program. Both the power dissipation and simulated cycles are normalized by the results of the original version.

From Table 2, we can see that **version A, B** and **C** all give better results compared to the **original version** without respect to power dissipation or execution cycles. However, based on the simulation result, **Version A** is the best of the four versions. This is different from the observation we observe from the measurement result. Furthermore, even through we simulate the whole program, the power dissipation of the loop is a big part of that of the whole program based on the comparison of the simulation results of the original version and **version A**. But we can not see significant difference between the simulation results of **version B** and **C**, which also disagrees with the physical measurement result. It is therefore hard to know when to trust the obtained simulation results.

4. Usage of the SimPoint Idea

Sometimes it is necessary to measure the power of the whole program in fine precision. Even though we can run the program many times to measure the power behavior of one small slice in each running and combine the results from the slices to get the final answer, it is time consuming and dealing with the overlap between two slices is non-trivial. In order to simplify the measurement work, we use off-line phase classification from SimPoint[16] to find representative intervals to simulate/measure.

4.1. Off-line Phase Clustering Analysis

The selection of simpoints includes the following steps:

1. **Basic Block Profiling** gets the basic block vector(BBV) for each interval with a fixed number of instructions.
2. **BBV Dimension Reduction** reduces the dimension of BBV to speedup the third step.
3. **Phase Classification Through K-Means Clustering** identifies the phases.
4. **Picking Simulation/Measurement Points** picks a representative from each cluster.

4.2. Feasibility Validation of the SimPoint Idea

In order to validate the suitability of the selected points from the above method, we choose 9 benchmarks from Mediabench [3] and compile them into ARM executables. We extend sim-outorder of SimpleScalar to perform BBV profiling on ARM benchmarks. The original SimPoint algorithm is also extended to support fixed number of simpoints instead of choose the best one among several numbers so that we can see the impact of the number of simpoints on the error rate of power/energy estimation. We then explore the error rate of each benchmark in a 3-dimensional space: interval size, number of simpoints, and error rate.

The following steps are performed on each benchmark:

1. Compile the benchmark locally on the skiff board with extended options -static and -msoft-float.
2. Run the modified sim-outorder on the benchmark to get BBVs.
3. Run the BBV analysis program on the BBVs from step2 to get the simpoints and their corresponding weights.
4. Run sim-panalyzer on the simpoints from step3.
5. Calculate whole-program power estimation based on the power values from step4 and the weights from step2.
6. Calculate the error rate.

Figure 4 shows the power behavior of *jpegencode* and the clusters obtained from the off-line phase clustering. The curve is the power behavior of the benchmark. Each power value is the simulated power dissipation of a fixed number(10000) of continuous instructions, so the x axis does not represent execution time precisely. The numbers on the primary x axis are phase numbers for the intervals of *jpegencode* when 5 phases are identified by the off-line phase classification algorithm. Intervals with the same phase number are classified into the same cluster and one of them is chosen as the representative of the phase. The numbers on the secondary x axis are phase numbers when 10 phases are identified. We can see that intervals from the same phase have similar power behavior. When 10 simpoints are simulated or measured, we get more representative slices of the program than when 5 phases are identified. This is evident especially for the second half of the program. When there are 5 phases, the

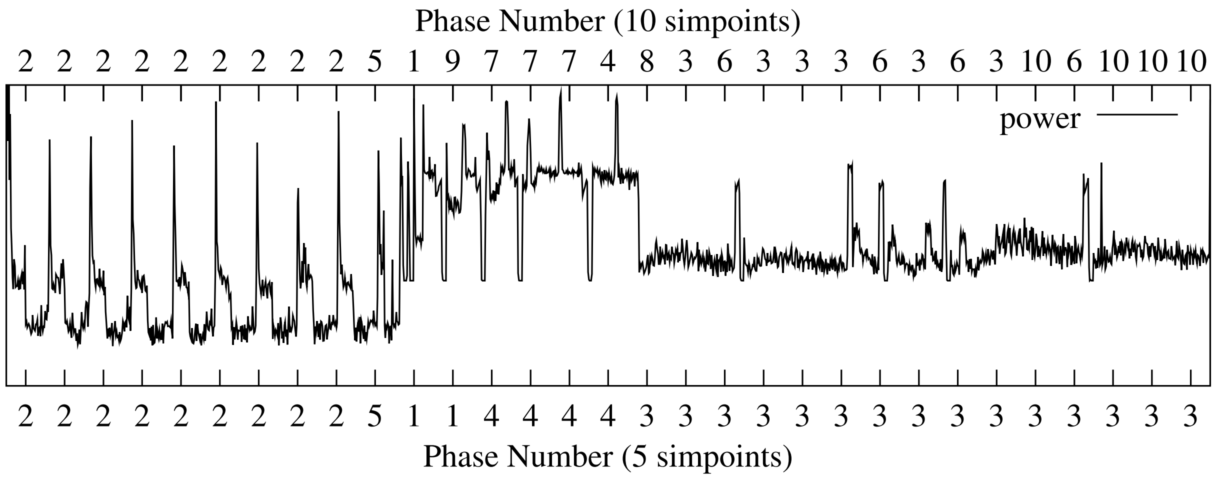


Figure 4. Power behavior and phase classification for ***jpegencode*****.**

intervals in the second half are clustered into the same phase, in spite of the power peaks in this part. When there are 10 phases, intervals with power peaks are clustered into a separate phase and the peaks can be found through the simulation or measurement of the simpoints. Phase 10 and phase 3 have very close power behavior, which means 10 simpoints are too many in this case.

We also explored the impacts of different numbers of simpoints and different interval sizes. For most benchmarks, when the number of simpoints is in the range of 1 to 10, the error rate decreases with the increase in the number of simpoints. When the number of simpoints is larger than 10, the error rate does not change significantly. An interval size of 1 million instructions is good enough for the benchmarks. For some benchmarks, we get the same error rate when smaller interval size is used, and the simulation time is largely reduced. It is hard to say what is the best interval size for a program without the behavior of the whole program. Figures are not available here due to space limitation.

Figure 5 shows the instructions-per-cycle (IPC) and power error rates of the simulated benchmarks and the average error rate. For each benchmark, the error rates of IPC and power are similar and the average error rate is below 2%.

Figure 4 and Figure 5 show the simulation-validated feasibility of the SimPoint idea in power and energy evaluation for large programs. In order to validate the feasibility of the SimPoint method in power evaluation, we need to identify each simpoint and trigger the physical measurement during the execution of the program. This work is in progress now.

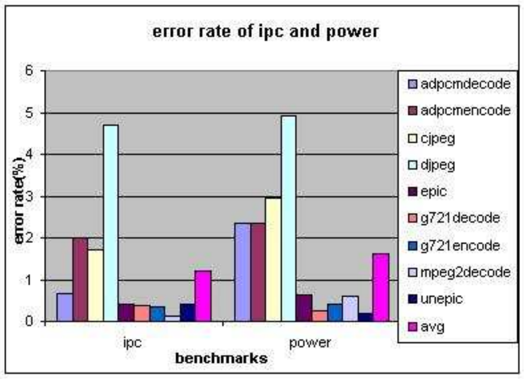


Figure 5. Error rates of IPC and power for each benchmark and their average error rates.

5. Future Work

Our focus in ongoing work is the placement of compiler generated code to identify simpoints during program execution. The identification should not have significant overhead or it will affect the accuracy of the measurement result. It may be possible to define some other form of program execution characterization that is different from Basic Block Vectors for phase classification, which may make the runtime identification of a simpoint easier. [13] describes a system using dynamic instrumentation for profiling, which may be helpful for the identification of the simpoints.

From the comparison of the physical measurement and simulation of the small program in Section 3, we can see that sometimes simulators can give us misleading results. Since our infrastructure can get precise measurement of a region in a program, we can design some specific micro-benchmarks to validate an energy simulator to help provide semantic connection between the physical measurement and the measured program.

After the above steps, we will have an integrated evaluation infrastructure for OS/compiler power and energy optimizations that will help in future power/energy optimization research. Our current work is concentrated on the ARM architecture. We will build a general-purpose power/energy evaluation infrastructure that can be applied to more architectures.

6. Conclusion

This paper describes an evaluation infrastructure for OS/compiler power and energy optimizations. This infrastructure brings together the advantages of simulation and physical measurement. It can provide objective evaluation of an optimization and semantic connection between measured power/energy and source code if needed. In order to overcome hardware limitations and measure long programs, we plan to use the SimPoint idea to find representative slices of a program and do physical measurement on these slices. The preliminary results show that our current infrastructure can do non-intrusive physical measurement and get precise power/energy behavior of the measured region of a program. Also, through simulation, we validated the feasibility of the SimPoint idea in power/energy evaluation. Validation through physical measurement is ongoing. We expect that the final infrastructure can help researchers find more optimization opportunities and avoid the effects of misleading evaluation results.

References

[1] `http://www.eecs.umich.edu/~panalyzer/`.

[2] `http://www.cs.ucsd.edu/~calder/simpoint/index.htm`.

[3] `http://cares.icsl.ucla.edu/MediaBench/`.

[4] B. Black and J. P. Shen. Calibration of microprocessor performance models. *IEEE Computer*, 31(5):59–65, May 1998.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. *Proceedings of the 27th Annual International Symposium on Computer Architecture*, page 83, June 2000.

[6] T. L. Cignetti, K. Komarov, and C. S. Ellis. Energy estimation tools for the palm. *International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, pages 96–103, August 2000.

[7] R. Desikan, D. Burger, and S. W. Keckler. Measuring experimental error in microprocessor simulation. *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 266–277, June 2001.

[8] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile application. *The Second IEEE Workshop on Mobile Computing Systems and Applications*, page 2, February 1999.

[9] S. Ghiasi and D. Grunwald. A comparison of two architectural power models. *Proceedings of the First International Workshop on Power-Aware Computer Systems*, pages 137–152, November 2000.

[10] S. Gurumurthi, A. Sivasubramaniam, M. J. Mary Jane Irwin, N. Vijaykrishnan, M. Kandemir, and J. L. K. Li, Tao. Using complete machine simulation for software power estimation: The softwatt approach. *International Symposium on High Performance Computer Architecture(HPCA)*, page 141, February 2002.

[11] Hennessy, John L. and Patterson, David A. Computer architecture: A quantitative approach(third edition). 2002.

[12] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, page 93, December 2003.

[13] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large intel itanium programs with dynamic instrumentation. *Proceedings of the 37th International Symposium on Microarchitecture*, pages 81–92, December 2004.

[14] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: The simos approach. *IEEE Parallel and Distributed Technology: Systems and Applications*, 3(4):34–43, December 1995.

[15] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. *International Conference on Parallel Architectures and Compilation Techniques*, pages 3–14, September 2001.

[16] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. *10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, October 2002.

[17] A. Sinha and A. P. Chandrakasan. Jouletrack - a web based tool for software energy profiling. *Design Automation Conference*, pages 220–225, June 2001.

[18] N. Vijaykrishnan, M. Kandemir, M. I. Irwin, H. S. Kim, and Y. W. Energy-driven integrated hardware-software optimizations using simplepower. *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 95–106, June 2000.

[19] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat. Ecosystem: Managing energy as a first class operating system resource. *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 36(5):123–132, December 2002.