Trading Functionality for Power within Applications

Melanie Kambadur
Columbia University
melanie@cs.columbia.edu

Martha A. Kim
Columbia University
martha@cs.columbia.edu

Abstract

When operating systems and hardware manage power and energy, they must be conservative in order to deliver requested resources while maintaining an expected rate of system throughput. Application-level energy management is more flexible, because applications can choose to request fewer resources or expect less performance, effectively trading accuracy or runtime for power. We propose to leverage this flexibility with *energy exchanges*, a C++ library extension that allows software to dynamically react to measured power and energy use by reducing functionality.

1. Background and Problem

Given energy's status as a precious commodity, many have ideas about how to police its use. Solutions for power and energy management abound, from hardware to the operating system to the virtual machine and the compiler. At the base of the stack, hardware has become dynamically adjustable, offering a range of supply voltages, operating frequencies, and sleep states. In the middle, operating systems tune hardware based on the measured and expected needs of applications running above. For example, Linux has policies to manage processor idle states and frequency scaling [5], power-aware scheduling algorithms have been proposed to schedule applications such that resource slow-downs can be made for longer periods or at a larger scale [4].

Such hardware and operating system adjustments need no application-level changes. Sometimes, this is desirable: many application developers cannot or will not modify their programs. However, keeping applications out of the energy management picture limits efficiency potentials, because applications have *three* energy control mechanisms that lower levels do not. First, only applications have the ability to request fewer processing and memory resources; hardware and operating systems miss this opportunity to save power and energy. Second, hardware and operating systems do not know when or by how much it is acceptable to slow down a program, so they must err on the side of caution when trading performance for power. Compounding their circumspection, existing power tuning controls often operate on a whole-core or whole-socket granularity, with potentially more than one application sharing the core or socket. At times when it is appropriate for one application to trade performance for power, it may not be for another, preventing the system from taking advantage. Conversely, applications know their own needs, and can decide exactly when, where, and to what degree performance should be traded for power. Third, while the system can trade only performance for power, the application has a second currency at its disposal: accuracy. Absent application-level information, the system cannot interfere with the function of a program. For these reasons, *application-level energy management must be used for optimal energy and power efficiency.*

2. Our Solution

We propose *energy exchanges*, a tool that empowers programmers to mandate when, where, and how to trade accuracy and performance for power and energy use. Using the short but expressive application-level directives provided by energy exchanges, programmers can implement feedback-driven changes to the application's behavior, for example, switching from precise to sloppy decoding in a video player as battery life declines, prioritizing a mobile games energy use over third-party advertisements, or bypassing some nonessential part of a robot's activity for a time so that it can run longer in between charges. We have implemented a prototype version of energy exchanges as a software only C++ library. The library monitors system-wide power and energy consumption by sampling standard hardware energy counters [3], then attributes measured usage to applications and threads using a special method of accounting.

3. Relationship to Prior Work

Energy exchanges offer several new features over standalone energy profiling and existing application-level energy management techniques [1, 6–10]. To begin, the exchanges are simple to use and integrate into existing programs and sys-

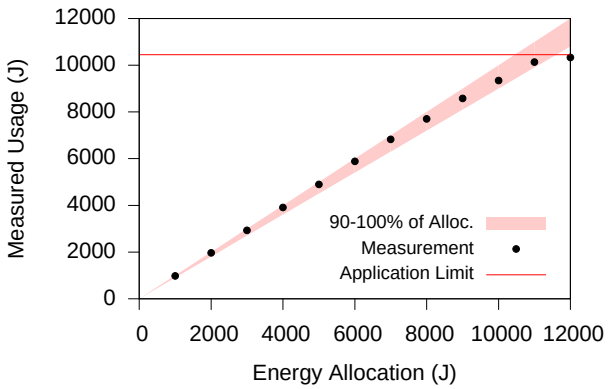

Figure 1: **Energy exchanges allow code to adapt to meet a program's energy goals.** Here, an energy exchange augmented `bodytrack` maximizes quality while never exceeding various allocated energy budgets. The application adapts by dropping frames as needed based on the energy consumed so far, the total allocation, and the number of remaining frames to process.

tems. They require no new hardware, operating systems, languages, or compilers. Instead, they require only short software annotations supported by the library extension. Next, unlike many software management strategies, e.g., compiler-inserted DVFS hints, that tune the same hardware knobs as the OS but in an application-specific way, energy exchanges allow the application to adjust itself. One benefit of this is that energy exchanges complement rather than replace existing system level energy management strategies. Additionally, self-adjustment means behavior can be based on dynamic energy, power, and runtime, so a program only makes performance or accuracy trades when necessary, and not every time the program runs. Finally, energy exchanges are extremely flexible. The energy and power usage feedback they provide can be used to modify just about anything a developer might want to change about their program's behavior. This means that a single energy-exchange augmented application will function across a range of different inputs, architectures, and degrees of parallelism.

4. A First Example

To demonstrate energy exchanges, we used them to augment `bodytrack`, a program from the Parsec benchmark suite [2]. The program tracks poses of a person recorded on multiple video cameras, with most of the program's computation inside a for-loop that processes video frames one at a time. Energy exchanges insert just a few lines of new code around and inside this loop to modify the program to drop as many (or as few) frames as necessary to meet a pre-specified energy allocation. Figure 1 demonstrates how bodytrack can be made to perform within various allocated budgets from 1000 to 12000 joules, with just one set of energy exchange annotations.

Completely and dynamically modifying an 11,000 line program's energy use with just 12 added lines of code demonstrates exactly the kind of simplicity and potency we hope energy exchanges will add to the energy management landscape.

References

[1] W. Baek and T. M. Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, June 2010.

[2] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[3] I. Corporation. Intel 64®and IA-32 architectures software developer's manual. `http://download.intel.com/products/processor/manual/253669.pdf`.

[4] I. n. Goiri, R. Beauchea, K. Le, T. D. Nguyen, M. E. Haque, J. Guitart, J. Torres, and R. Bianchini. Greenslot: Scheduling energy consumption in green datacenters. In *SC*, 2011.

[5] T. Hornbeck and P. Hokanson. Power management in the linux kernel. 2011.

[6] A. Kansal, S. Saponas, A. Brush, K. S. McKinley, T. Mytkowicz, and R. Ziola. The latency, accuracy, and battery (LAB) abstraction: programmer productivity and energy efficiency for continuous mobile context sensing. In *OOPSLA*, 2013.

[7] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn. Flikker: saving dram refresh-power through critical data partitioning. In *ASPLOS*, March 2011.

[8] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with Eprof. In *EuroSys*, 2012.

[9] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: approximate data types for safe and general low-power computation. In *PLDI*, June 2011.

[10] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger. Eon: a language and runtime system for perpetual systems. In *SenSys*, November 2007.