ProSEEP: A Proactive Approach to Energy-Aware Programming

Timo Hönig and Wolfgang Schröder-Preikschat
Friedrich–Alexander University Erlangen–Nuremberg
{*thoenig,wosch*}*@cs.fau.de*

Rüdiger Kapitza
TU Braunschweig
rrkapitz@ibr.cs.tu-bs.de

1 Introduction and Motivation

Today, optimizing software for energy efficiency is an expensive task. First, there are only few tools available that assist developers to optimize their code at hand. Second, due to the complexity of systems examining applications for energy hogs is a very time-consuming task.

Designing energy-efficient applications currently is a backward-looking process. Energy bugs, which are usually experienced and reported by users past application deployment need to be analyzed and tracked down by developers manually.

We present ProSEEP, an approach for proactive energy-aware programming, which turns the modus operandi of energy-aware programming into a forward-looking process. Taking advantage of symbolic execution engines and platform-specific energy profiles ProSEEP assists software developers in making application design decisions in the awareness of the decisions' impact on the energy footprint of the code under development.

2 Energy-Aware Programming at Present

With the advances of the last two decades in mobile computing we are today progressing towards energy-proportional system designs. When no or only little work needs to be performed, systems spend only a fraction of the energy they consume when utilization is high [1]. Currently, this trend is trespassing upon other technology sectors such as server and data-center design [2] and exascale computing [3].

By contrast, application design targeting energy efficiency is an understudied research topic. Resulting from this, developers can not take advantage of well-engineered tools when optimizing their code for energy efficiency, or when trying to resolve energy hogs. Existing work such as energy profilers [4, 5] are limited, as they pursue a backward-looking approach. The major drawback of energy profilers is the fact that a single profiling run yields exactly one profiling result only. Each result is specific to a unique code path together with concrete input parameters of the application under test. The efforts required to create an exhaustive energy profile for an application therefore grow exponentially with the amount of code paths and their input parameters. Moreover, such profiling results are platform specific and can not be transferred to another platform.

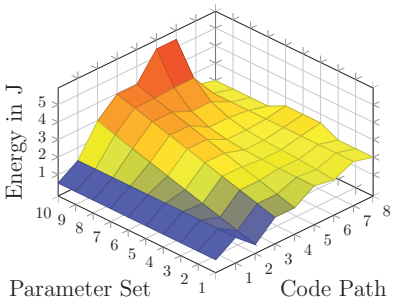

Figure 1: Energy consumption varies significantly for different code paths and depending on parameter sets.

Applications usually have a drastically different energy footprint depending on the code path taken during execution. For example, a regular code path (i. e., normal operation) is likely to consume less energy in comparison to an exception-handling code path which triggers a complex operation for recovery purposes (i. e., special case operation). In addition to this, two distinct execution runs of one and the same code path may show a varying energy footprint depending on input parameters (e. g., size of a batch job) and other path constraints. As an example, Figure 1 illustrates the overall energy footprint of a micro benchmark with eight code paths. Each code path (x-axis) together with its input parameters and path constraints (y-axis) can be assigned to a specific amount of energy consumed during execution (z-axis). The visualized data is an extended data set of our SEEP prototype which we have presented and evaluated in [6].

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre „Invasive Computing" (SFB/TR 89).

3 The ProSEEP Approach

With ProSEEP we present a forward-looking approach to assist developers at the task of energy-aware programming. Our approach is based on two major concepts. First, we exploit symbolic execution to explore the runtime complexity of program code automatically. Second, we use energy profiles to make projections of the expected energy consumption for heterogeneous target platforms. Making the projected energy consumption available to developers helps to identify unfavorable design decisions early (e. g., use of an inappropriate program library). Our ongoing research is based on SEEP which is most suitable for low noise execution environments like embedded systems and high performance computing where computing cores operate on single tasks. Figure 2 shows the ProSEEP architecture.

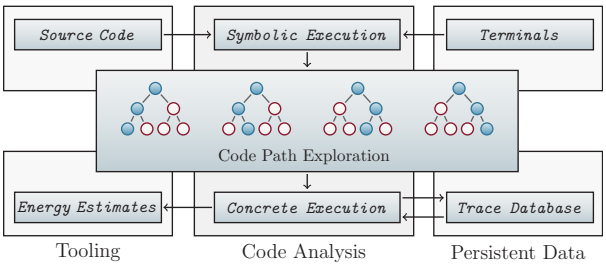

Figure 2: Overview of the ProSEEP architecture.

Currently, our work focuses on three aspects. First, to speed up code examination ProSEEP supports selective execution which allows us to calculate energy estimates for code paths iteratively. Second, our framework utilizes so-called *terminals* which are required to model energy consumption caused by peripheral devices. Third, we evaluate appropriate ways to make energy estimates available to developers (e. g., integration into development frameworks).

Iterative Energy Estimation. As the effort for exploring code paths grows exponentially with their quantity and with the number of their input parameters we introduce an iterative energy estimation technique. During the analysis phase of code paths we first query a database containing trace information of previous runs before executing and examining the code paths further. Subpaths that have been analyzed previously are not required to be examined a second time as tracing information is already available. In contrast, new subpaths are being executed during the analysis phase and the corresponding tracing information is stored in a database. This speeds up the overall process significantly, which in turn brings two major improvements. On the one hand, our code analysis scales much better as redundant analysis steps are being avoided, and, on the other hand, we can return energy estimates much quicker to the developer.

Terminals. Energy consumption is not strictly proportional to execution time as concurrent system activities caused by peripheral devices need to be considered. To incorporate such energy consumption into our energy estimates, we accordingly extract relevant data when analyzing program code. We identify low level interfaces leading to device activities and extract parameters which are eventually fed to device-specific energy models. We refer to such interfaces as terminals.

Tooling Integration. With the demand to turn energy-aware programming from a backward-looking into a forward-looking process, we are evaluating ways to integrate our approach into existing programming environments in order to assist developers in the most efficient way. Beside integrating ProSEEP into development environments like Eclipse we consider our approach especially useful for new programming paradigms as pursued in invasive computing [7] which focuses on resource-aware programming.

4 Conclusion

With ProSEEP we have presented early results of our continued research efforts targeting tool-supported energy-aware programming. To eliminate shortcomings of today's best practice in energy profiling, we propose a proactive approach to assist developers at the task of energy-aware programming.

In particular, our approach provides iterative energy estimation which speeds up the calculation of energy estimates and ensures scalability of our code analysis. Energy consumption of peripheral devices is incorporated by using data extracted during code analysis and device-specific energy models. With the appropriate integration into programming environments we see ProSEEP as a profound approach to turn energy-aware programming into a forward-looking process.

5 References

[1] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[2] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. In *Proceedings of ISCA'10*, pages 338–347, 2010.

[3] R. Murphy, T. Sterling, and C. Dekate. Advanced architectures and execution models to support green computing. *Computing in Science and Engineering*, pages 38–47, 2010.

[4] A. Kansal and F. Zhao. Fine-grained energy profiling for power-aware application design. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):26–31, 2008.

[5] A. Pathak, Y. Hu, and M. Zhang. Where is the energy spent inside my app? Fine grained energy accounting on smartphones with Eprof. In *Proceedings of EuroSys'12*, pages 29–42, 2012.

[6] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat. SEEP: Exploiting symbolic execution for energy-aware programming. In *Proceedings of HotPower'11*, pages 17–22, 2011.

[7] J. Teich, J. Henkel, A. Herkersdorf, D. Schmitt-Landsiedel, W. Schröder-Preikschat, and G. Snelting. Invasive computing: An overview. In *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*, pages 241–268. 2011.