Proactive Energy-Aware System Software Design with SEEP

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

Björn Cassens and Rüdiger Kapitza
TU Braunschweig
{b.cassens,rkapitza}@tu-bs.de

1 Introduction and Motivation

Designing system software currently optimizes program code for correctness and speed. While this is essential for the reliable operation of computer systems, these two characteristics alone are often not sufficient. Moreover, it is important to ensure that a third characteristic is being considered during the process of designing system software: energy efficiency.

As optimizing program code for energy efficiency is a tedious and time-consuming task we are working on SEEP [1], a project which provides a programming framework to assist developers at the task of energy-aware programming. The framework is named after two of its key components: symbolic execution and energy profiles. In this position paper, we introduce the SEEP approach, detail our current work, and discuss future challenges. We believe that it is essential to supply software developers and software designers with the right set of tools in order to ease the process of energy-aware programming.

We have identified the current modus operandi to be hindering for energy-efficient software development. Today, developers need to analyze program code for energy hotspots manually. This task is being performed in a *reactive* manner. Program code is first being developed and afterwards being analyzed for defects with regard to unusually high energy consumption. This manual task is cumbersome for two reasons. First, the efforts required to analyze program code for energy efficiency grow exponentially with the number of program paths of the application. Second, the amount of energy consumed differs among heterogeneous hardware platforms. Developers are required to evaluate the software on various platforms which makes the task of identifying and solving energy bugs even more unappealing.

This work was partly supported by the German Research Foundation (DFG) under grants no. FOR 1508 (subproject TP2) and SFB/TR 39 (subproject C1).

With SEEP, we provide the tooling infrastructure required to overcome current limitations. We exploit symbolic execution techniques [2] for automatic analysis of program code. Combined with energy models and platform-specific energy profiles we provide energy estimates for program code to developers as early as during the time of software development.

2 Proactive Energy-Aware Programming Using SEEP

The SEEP framework (see Figure 1) is motivated by instantly providing energy estimates, which have direct influence on the development process. Hence, commonly required feedback-based code modifications after deployment can be reduced by turning the modus operandi into a *proactive* approach.

One major effort is to offer a high degree of automation, that is, requiring as little user interaction as possible. At best, no code annotations or other changes to the program under test are necessary. With regards to programming languages, developers are free in their choice and are not forced to use special energy-aware programming languages as proposed in [3].

In order to provide precise and exhaustive energy consumption estimates, program code that is potentially being executed should be incorporated into the energy estimation process. SEEP uses symbolic execution, a technique that is effective in exploring program paths automatically. This multi-path analysis ensures that energy estimates cover a program in all its facets, and as a consequence, increase the chance to unveil hidden energy hotspots. Beforehand, executables that correspond to specific code paths (so-called path entities) need to be concretely executed to extract runtime characteristics required in subsequent steps during the analysis phase.

SEEP needs to keep the complexity of the estimation process at an absolute minimum. For this purpose, the framework relies on several different energy profiles. Besides instruction profiles, which depend on

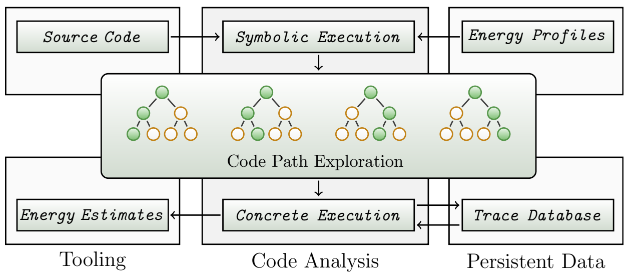

Figure 1: Overview of the SEEP architecture.

a CPU's instruction set architecture, this includes energy profiles for device-specific peripherals (e.g., network or file transfer costs). By means of virtualization environments, energy estimates can be calculated without the need to execute code on target platforms.

This profile-driven approach is extended by further persistent data, which is populated iteratively with entries for functions that have been analyzed by the framework. Such function entries consist, amongst others, of symbolic expressions, which can be exploited to interpolate a function's energy consumption. Thus, whenever the control flow of a consecutive execution run reaches functions that have been analyzed previously, symbolic execution can be omitted. This shortcut saves great amounts of analysis time.

Furthermore, concretely executing path entities to deduct a target's runtime behavior can be parallelized for heterogeneous platforms using virtualization techniques. At this, our approach does not make any restrictions as long as target platforms and their peripheral devices can be measured accurately according to a precise energy model. From a CPU's point of view, such models must contain both basic and inter-instruction energy costs which vary in dependence of a CPU's capabilities (e.g., instruction pipelining).

3 Development Process Integration

Currently, we explore different possibilities to consolidate SEEP with integrated development environments (IDEs) such as Eclipse. As basis of decision-making, energy estimates are provided at function level. These estimates are displayed in the IDE so that developers can correlate program code (i.e., source code of a function and input parameters) with energy consumption estimates.

During the development phase, code changes are being reported to the backend of the SEEP framework. Functional changes trigger a reevaluation of affected program paths. To ensure consistency the trace database is being updated incrementally. Whenever code changes cause a significant negative impact on the program code currently in development, the developer is being notified concerning this matter. If alternatives for functionally equal implementations are available (e.g., different libraries implementing the same algorithm), SEEP proposes to use the more efficient alternative. This may depend on the target platform and conditionally needs to be incorporated into the build infrastructure of the program code.

To adequately represent a multitude of energy consumption estimates (e.g., several distinct input parameters for a single function) we currently evaluate how to graphically illustrate the energy estimates within IDEs. This helps developers to easily discover energy hotspots in the program code.

4 Position Statement and Outlook

Down to the present day, program code is commonly not optimized for energy-efficiency. As developers improve their program code merely with regards to speed and correctness, it leads to the situation that system software components needlessly waste energy resources. To address this, we are convinced that new concepts for energy-aware programming need to be established. Most of all it is required to provide strong tooling support for developers to ease the task of increasing the energy-efficiency of software. Such tooling support relieves developers from manually examining software for energy hotspots by providing a high degree of automation. This is a challenging endeavor as the diversification of hardware platforms steadily increases and analyzing program code asks for high analysis efforts. In order to propagate energy-aware programming we propose SEEP, a proactive approach to address these challenges. By applying the SEEP approach, we currently increase the energy-efficiency of the Sloth operating system [4] used in the research project BATS which is founded by the German Research Foundation (DFG-Forschergruppe 1508).

References

[1] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat. SEEP: Exploiting symbolic execution for energy-aware programming. In *Proc. of the 4th Workshop on Power-Aware Computing and Systems*, pages 17–22, 2011.

[2] C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proc. of the 8th Symp. on Operating Systems Design and Implementation*, pages 209–224, 2008.

[3] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. Corner, and E. Berger. Eon: A language and runtime system for perpetual systems. In *Proc. of the 5th Intl. Conf. on Embedded Networked Sensor Systems*, pages 161–174, 2007.

[4] W. Hofer, D. Lohmann, F. Scheler, and W. Schröder-Preikschat. Sloth: Threads as interrupts. In *Proc. of the 30th Real-Time Systems Symp.*, pages 204–213, 2009.