Playing Hare and Tortoise: The FIGAROS Kernel for Fine-Grained System-Level Energy Optimizations

Timo Hönig, Christopher Eibel, Benedict Herzog,
Heiko Janker, Peter Wägemann, and Wolfgang Schröder-Preikschat

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)

***Abstract*—Energy has emerged to be the most important resource for computing systems. Despite the exceptional importance of energy, reducing its demand at application and system level remains a challenging task for programmers and engineers. This is aggravated by the fact that traditional energy-saving approaches are not only error-prone but even lead to adverse consequences (i.e., increased energy consumption). To address this concern, we present the FIGAROS operating system for fine-grained system-level energy optimizations. The evaluation of our FIGAROS implementation shows that the operating system lowers the energy consumption of processes by up to 2.9 x.**

Keywords*—*Operating Systems, Energy Measurements, Energy Optimizations, Energy-Aware Scheduling

I. INTRODUCTION

Energy has become the most important operating resource for computing systems of all sizes—from embedded systems to large-scale high-performance computing systems. Yet, at application and system level, programmers and engineers remain challenged at efficiently handling energy as first-class operating system resource [1]. Accordingly, to reduce the energy demand of computing systems, different energy-optimization strategies have emerged, which either tackle the challenge *ahead of runtime* [2], [3] or *at runtime* [4], [5]. To reduce the energy demand of processes (i.e., a program being executed) at runtime, today's operating systems commonly use race-to-sleep strategies [4], [6], [7]. Even though the average power demand increases during code execution with race-to-sleep strategies, the energy consumption (integral of power over execution time) is actually decreased. However, recent research [8], [9] shows that race-to-sleep strategies are prone to miss their target and eventually lead to *higher* energy demand.

In this paper we propose to tie a link between ahead-of-runtime and at-runtime energy-optimization strategies to reduce the energy consumption of processes in situations where traditional energy-saving strategies fail to succeed. To close the gap between ahead-of-runtime and at-runtime energy optimization, we present the FIGAROS operating system. FIGAROS orchestrates energy measurements of processes and applies energy optimizations to reduce the system's energy consumption. Our work makes the following contributions: First, we present the concept of the FIGAROS operating-system kernel, which provides basic operations for system-level energy analysis and associated energy optimizations. Second, we implemented a prototype of FIGAROS that exploits hardware-based energy measurements at runtime to validate our approach. Third, we evaluate FIGAROS and discuss analysis results which show that FIGAROS lowers the energy consumption by up to 2.9 x.

II. TIME VS. ENERGY: PLAYING HARE AND TORTOISE

The energy demand E_{exec} of a computer to execute a given program code is determined by integrating the time function of the power consumption $p(t)$ over the time $t = t_1 - t_0$: $E_{\text{exec}} = \int_{t_0}^{t_1} p(t) \cdot \mathrm{d}\,t$. The energy demand of program code is therefore lowered by reducing the *power demand* during the execution time and by reducing the *execution time* itself. Program code is commonly executed with race-to-sleep (alias race-to-idle) strategies. Such strategies execute the program code at maximum performance with the objective to reduce the execution time. Although the power demand is maximized during the execution period, the reduced execution time allows the system to be transfered to a low-power or sleep state for a longer period of time. Eventually, this reduces the energy consumption needed for executing the program code compared to normal execution. In contrast to these best practices, recent works [8], [9] have revealed that race-to-sleep strategies may not ultimately lead to the desired effect (i.e., minimization of energy demand). In fact, crawl-to-sleep strategies that lead to longer execution times may decrease the energy consumption needed for executing the program code.

Thus, optimizing for performance (i.e., minimizing execution time) does not necessarily lead to an optimum with regards to energy consumption (i.e., minimization of energy demand). This situation can be compared to Aesop's fable *The Hare and the Tortoise*, where it is beyond question that the hare is quicker than the tortoise. However, while the tortoise tries to reach the other end with low but nearly constant speed, in his arrogance the hare takes a nap believing that he is still able to race across the finish line before the tortoise. In the end, although the tortoise requires a lot more time than the hare, he wins the race nonetheless. Similarly, it needs novel sophisticated strategies to reach goals in today's computing systems (i.e., progress guarantees, task execution) with less energy—even if other non-functional aspects are sacrificed by tolerable amounts (i.e., lower energy demand at the cost of acceptable, though longer execution times).

III. THE FIGAROS OPERATING-SYSTEM KERNEL

With FIGAROS we present a systems approach to provide a reliable link between ahead-of-runtime and at-runtime energy optimizations to be used for novel energy optimizations for program code. FIGAROS addresses the challenges of combining ahead-of-runtime and at-runtime energy optimizations. Further, our approach solves the problem of minimizing the energy demand (as outlined in Section II) of program code by making it possible to execute tasks with predefined, application-specific sets of energy-saving features. The core of

2324-7894/15 $31.00 © 2015 IEEE
DOI 10.1109/SBESC.2015.22

Device	Energy Resolution [µJ]	Temporal Resolution [µs]	Current Range [A]	Operability
Fuel Gauge	116,000	3,515,000	Up to 2.55	Hard (no time/trigger)
MeasureAlot	0.04	0.006	Up to 0.15	Easy
Multimeter	1	ca. 300,000	Up to 10	Hard (no time/trigger)
Oscilloscope	0.0001	0.001	0.005–150	Hard (setup, streaming)

TABLE I. LIST OF DISTINCT ENERGY-MEASUREMENT DEVICES

FIGAROS is an operating-system kernel supplemented with an energy-measurement infrastructure (Section III-A). In line with the concept of our approach we present an implementation of the FIGAROS operating-system kernel that is based on Linux (Section III-B). Our prototype exploits a programmable energy-measurement device [9] to provide runtime energy measurements (Section III-C).

A. Concept. FIGAROS consists of two main components: On the one hand, FIGAROS provides a runtime environment (i.e., operating-system kernel) which has full control of the energy-saving features of the underlying system hardware. On the other hand, FIGAROS provides an energy-measurement subsystem that implements basic operations to run energy analysis of program code at runtime.

The runtime environment reduces the energy consumption required to execute a given program code by exploiting energy-saving features available to the system that executes the program code. Such energy-saving features include, but are not limited to, processor-specific energy-saving features (i.e., DVFS, sleep states) as well as energy-saving features specific to peripheral devices (i.e., input/output subsystem, graphics processing unit). To execute program code, the runtime environment either operates in manual or automatic operation. In the manual operation mode, FIGAROS determines the energy-saving features by reading metadata tied to the program code. This metadata is deposited by the programmers ahead of runtime and originates from a code analysis performed at development time of the code. In the automatic operation mode, FIGAROS varies different sets of energy-saving features to identify the most suitable set for a given task. During each execution of a specific task the active set of energy-saving features is varied and the resulting energy consumption required to execute the task is measured.

The energy-measurement subsystem of FIGAROS implements basic operations for energy measurements and provides corresponding programming interfaces. To implement energy-measurement operations, FIGAROS either uses existing energy models or runs energy measurements at runtime of the program code. Thus, the energy analysis of FIGAROS is either software based (energy models) or hardware based (energy measurements). Software-based energy-analysis methods use different ways of modeling the energy demand. For example, the energy demand can be estimated by evaluating performance counters [10] or by using instruction-based energy profiles. Hardware-based energy-analysis methods use an energy-measurement device (i.e., multimeter, oscilloscope) to determine the system's energy consumption during the execution of program code. In either way, the energy-analysis method as implemented by the FIGAROS subsystem can be controlled through a well-defined programming interface during runtime.

B. Implementation of the FigarOS Kernel. Our implementation of the FIGAROS operating-system kernel is based on the Linux kernel and introduces energy awareness of the operating system at its core. We chose the Linux kernel to explore the capabilities of FIGAROS on a wide variety of different computing-system classes (i.e., embedded, mobile/desktop, and server systems), which provide various distinct energy-saving opportunities. The energy-measurement operations of FIGAROS enable the operating system to be self-reflective and perform online energy measurements for different system components (e.g., individual kernel units, user-space processes). Thus, we implemented an energy-measurement module, which encapsulates functionality to drive energy measurements and can be used to trigger single or continuous measurements from arbitrary parts of the operating-system kernel.

For our current prototype of FIGAROS we extended the Linux process scheduler with the ability to monitor the energy demand of user-space tasks of the system. Linux uses the Completely Fair Scheduler (CFS) to distribute available processor execution time completely fair among all runnable processes. To do so, Linux manages all runnable processes in a red–black tree sorted by the total spent execution time per process. On rescheduling, the process with the lowest spent execution time so far is chosen. The implementation leads to time slices of variable length, though the minimum length of a time slice can be specified. Our extended process scheduler decides upon different criteria (i.e., process ID, binary name) whether a measurement is triggered for the selected task. Over time, the consecutive measurements build a detailed view of the past and current power demand of a single or a group of processes and enables power-demand forecasts. Combining the power demand of the process over its execution time gives FIGAROS detailed information about its energy demand. The gained detailed knowledge allows the operating system to detect critical situations (e.g., thermal overheating, malicious waste of energy) early and take appropriate countermeasures at hardware and software level. When a lasting, increased power demand occurs, FIGAROS initiates a countermeasure at hardware level (e.g., switching to a new set of energy-saving features) early enough to circumvent a system failure. Further, FIGAROS considers software measures to mitigate a menacing critical situation, for example, by penalizing specific processes (e.g., activity suspension or even termination).

The basic energy-measurement operations provide the core functionality for complex energy-measurement strategies. For example, the operations can be exploited for verification purposes (e.g., energy-consumption capping) and novel operation modes including energy-driven priority schemes instead of traditional time- and priority-driven ones. Further, the type of energy measurement is of great importance as it has a major impact on the gained information (i.e., measurement data).

C. Energy-Measurement Infrastructure. Accurate energy-measurement methods are required for the control logic of efficient power and energy management. Unfortunately, available measurement implementations are often unsuitable for this task. Table I shows basic parameters for three common approaches: Multimeters and oscilloscopes paired with a current probe are common commercial measurement devices, whereas fuel gauges are low-power integrated circuits with a small footprint, employed in many mobile devices.

These methods typically struggle with the huge dynamic demand of current in today's computing systems. Currents can span by orders of magnitude between deep sleep and

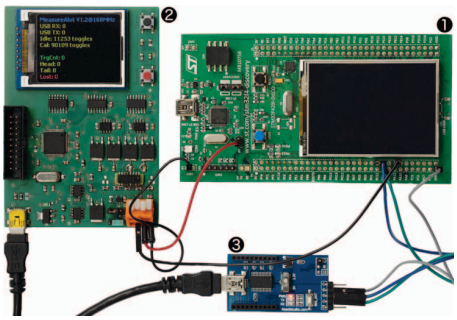

Fig. 1. We evaluate energy optimizations of our FIGAROS operating system with benchmarks running on a hardware platform with an ARM Cortex-M4.

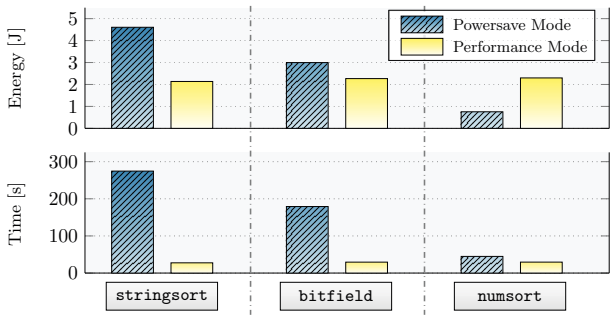

Fig. 2. The energy consumption (top) and execution time (bottom) do *not* always correlate and depend on the run mode (i.e., powersave, performance).

run mode. If the measurement range of these devices is, for example, set up to observe a specific run mode (i.e., operating level at a given performance level), the current draw of another run mode (i.e., sleep mode) is typically missed. To provide measurement capabilities for short code paths (i.e., time slice of a process scheduler), a measurement device must implement a trigger input to start and end measurements and needs to provide a high temporal resolution. From the aforementioned methods of measurement, only oscilloscopes offer this capability. However, oscilloscopes are not suitable, as continuous streaming of measurements is difficult to achieve, if at all. For FIGAROS we therefore rely on the MeasureAlot energy-measurement device [9], which addresses these issues. Internally, the device operates using a current-to-frequency conversion, offering a usable measurement range over several decades. The energy-measurement resolution is 0.1 µJ and the device offers a dedicated trigger signal to support measurements of very short periods (down to approximately 6 ns). The energy-measurement results are streamed to a host computer over USB and retrieved using an open-source library.

IV. EVALUATION

We evaluate energy optimizations of our FIGAROS implementation with the nBench Linux benchmark and present the energy-analysis and energy-optimization results of FIGAROS for three unique benchmark modules in the following.

A. Setup. The setup of our evaluation is depicted in Figure 1. The FIGAROS operating-system kernel is running on our target system, an STMicroelectronics STM32F429 Discovery development board featuring an ARM Cortex-M4 (❶). The power to the microcontroller is supplied by the MeasureAlot measurement device (❷). A trigger signal from the STM32F429 Discovery development board to the MeasureAlot is used to control the measurements. Lastly, a USB-to-serial converter (❸) provides a serial console to the development board for controlling purposes.

B. Time vs. Energy. In the first evaluation scenario we optimize the execution of the nBench benchmark modules to achieve the lowest energy consumption possible. Each individual benchmark module remains entirely unchanged (i.e., identical machine code) across all evaluation runs. However, across several execution runs FIGAROS varies the configuration of different energy-saving features (i.e., CPU clock) according to previously deposited information. The resulting execution times and energy consumptions of each run are recorded and shown in Figure 2. In performance mode (i.e., race-to-sleep), the CPU is clocked 22.5 times higher compared to powersave mode. Thus, the CPU performance is significantly higher in performance mode. As expected, this performance gain is reflected by shorter execution times for all three benchmark modules. It is surprisingly, however, that executing the `numsort` benchmark module consumes 2.9 x more energy in performance mode. Ergo, executing this program code in powersave mode (i.e., crawl-to-sleep) significantly reduces the energy consumption compared to traditional race-to-sleep execution strategies and confirms the importance of ahead-of-runtime program-code analysis to proactively optimize program-code execution for the lowest energy consumption at runtime.

C. Energy Demand per Time Slice. To develop energy-aware scheduling strategies, we analyze the energy demand per time slice of the user-space tasks executing the three benchmark modules (cf. Section IV-A). Figure 3 shows the length of all time slices for all three examined algorithms executed in performance mode. The y-axis reflects the elapsed time and the x-axis the corresponding time slice. The Linux process scheduler uses time slices of variable length: at the beginning (time slices number 1 to 13) and at the end (time slices number 41 to 43) of the task execution, the CFS scheduler decides to use very short time slices. This may be caused by initialization and cleanup work required during task creation and task termination, respectively. During the actual workload phase (time slices number 14 to 40) the scheduler uses comparably long time slices, which are heaped among several levels. Presumably, the different levels reflect the scheduler granularity for the length of a time slice.

Figure 4 illustrates the power demand (energy per time slice) of every time slice executed in performance and powersave mode. In performance mode the number of time slices and the total execution time is similar for all three benchmark modules (see Figure 2). However, in powersave mode the number of time slices and the total execution time of each benchmark module differs, which leads to graphs of different lengths in Figure 4 (right-hand graph). Nevertheless, all three graphs have similar shapes in *both* run modes (i.e., performance and powersave). The short time slices at task initialization and at task termination lead to higher power demand and higher energy consumption. Possible reasons for this are frequent context switches, cache effects, and a different type of instructions pattern. However, during workload phase, the power demand remains at a very constant, low level. Following from that we conclude that disturbing a task as little as possible leads to a minimization of energy consumption.

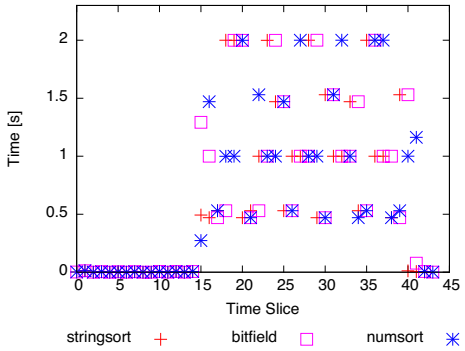

Fig. 3. The time per time slice differs during execution and is lower during task initialization and termination.

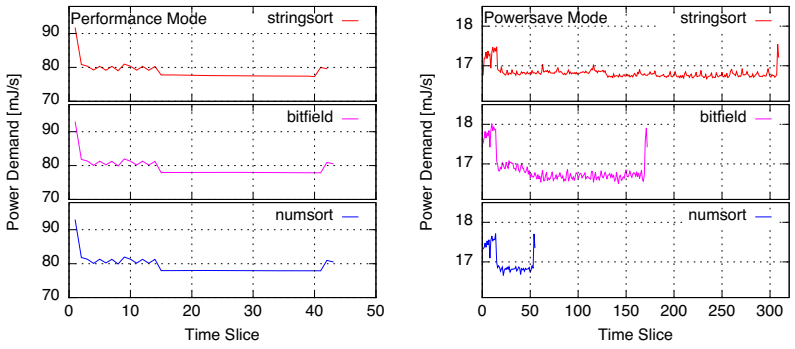

Fig. 4. The power demand per time slice of the benchmark modules running on the FIGAROS operating-system kernel in performance mode (left) and powersave mode (right).

V. Related and Future Work

ECOSystem [1] was first to propose that energy should be treated as a first-class operating-system resource. The idea of the ECOSystem approach is a unified methodology to energy accounting and distributing energy in a fair way among applications in the system. According to the authors of ECOSystem, however, smart-battery interfaces of modern, general-purpose desktop systems have shortcomings. These shortcomings inhibit fine-grained energy scheduling, since the interfaces are considered to be slow and only yield coarse-grained information about energy consumption. This problem is solved through the FIGAROS kernel with its accurate, fine-grained measurement technique for tasks.

These adaptive scheduling methods require information about the energy demand of tasks at runtime in order to identify suitable scheduling decisions. Only considering the timing behavior is not sufficient if energy is treated as a first-class resource [1] in the operating system. As demonstrated in the evaluation (cf. Section IV-A), execution time does not necessarily correlate with the energy consumption. The coherence between time and energy even decreases if a task makes extensive use of external peripherals (i.e., sensors, transceivers) causing a significantly higher power consumption than other tasks within a fixed time slice. A possible scheduling strategy is a completely fair distribution of energy to all tasks controlled in a feedback loop. The energy consumption is recorded for each task by the FIGAROS kernel. If a higher energy consumption is traced during task execution in the assigned time slice, the time slice of this task is reduced for the following dispatching process. Thus, adapting time slices is controlled in a feedback loop leading to a fair energy distribution among all tasks.

However, the *power consumption* is not yet respected in the discussed energy-aware task-scheduling approach. For example, a task can still consume its own entire energy budget within a timespan which is shorter than the length of the actual time slice. As such peaks in the power consumption can possibly occur before triggering a new energy measurement, we are potentially unable to counteract with a rescheduling decision in time. We aim to tackle this problem by applying both hardware- and software-based techniques. At software level, it is possible to treat external activities following our ahead-of-runtime approach. Being aware of the energy consumption and power-consumption peaks of complex operations (e.g., I/O) beforehand allows us to incorporate them right into the scheduling-decision process. At hardware level, power capping is a countermeasure to prevent power-consumption peaks and enable current state-of-the-art processor architectures (e.g., Intel Sandy Bridge and later: RAPL [11]) to adhere to a certain power limit which is not exceeded within a specified time interval. In combination with FIGAROS, power capping gives the possibility to further penalize a task's energy consumption compared to just reducing its assigned time slices.

VI. Conclusion

In this paper we presented FIGAROS, a highly energy-aware operating-system kernel that implements fine-grained energy optimizations at system level. By tying a new link between ahead-of-runtime and at-runtime energy optimizations our approach cuts down the energy consumption by up to 2.9 x.

Acknowledgements. This work was supported by the German Research Foundation (DFG) under grants no. SCHR 603/13-1, SCHR 603/10-1, and the Transregional Collaborative Research Centre 89 (SFB/TR 89, C1).

References

[1] H. Zeng, X. Fan, C. Ellis, A. Lebeck, and A. Vahdat, "ECOSystem: Managing energy as a first class operating system resource," in *Proc. of the 10th Conf. on Architectural Support for Programming Languages and Operating Systems*, 2002, pp. 123–132.

[2] M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye, "Influence of compiler optimizations on system power," in *Proc. of the 37th Annual Design Automation Conf.*, 2000, pp. 304–307.

[3] T. Šimunić, L. Benini, G. De Micheli, and M. Hans, "Source code optimization and profiling of energy consumption in embedded systems," in *Proc. of the 13th Intl. Symp. on System Synthesis*, 2000, pp. 193–198.

[4] A. S. V. Palladi and A. Starikovskiy, "The ondemand governor: Past, present and future," in *Proc. of Linux Symp.*, vol. 2, 2006, pp. 223–238.

[5] L. A. Barroso and U. Hölzle, "The case for energy–proportional computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[6] S. Albers and A. Antoniadis, "Race to idle: New algorithms for speed scaling with a sleep state," *ACM Transactions on Algorithms (TALG)*, vol. 10, no. 2, p. 9, 2014.

[7] A. Jimborean, K. Koukos, V. Spiliopoulos, D. Black-Schaffer, and S. Kaxiras, "Fix the code. Don't tweak the hardware: A new compiler approach to voltage-frequency scaling," in *Proc. of the 2014 Intl. Symp. on Code Generation and Optimization*, 2014, pp. 262–272.

[8] H. Amur, R. Nathuji, M. Ghosh, K. Schwan, and H.-H. S. Lee, "Idlepower: Application-aware management of processor idle states," in *Proc. of the 2008 Works. on Mgd. Many-Core Systems*, vol. 8, 2008.

[9] T. Hönig, H. Janker, C. Eibel, O. Mihelic, R. Kapitza, and W. Schröder-Preikschat, "Proactive energy-aware programming with PEEK," in *Proc. of the 2014 Conf. on Timely Results in Operating Syst.*, 2014, pp. 1–14.

[10] F. Bellosa, "The benefits of event-driven energy accounting in power-sensitive systems," in *Proc. of 9th ACM SIGOPS European Workshop—Beyond the PC: New Chall. for the Operating Syst.*, 2000, pp. 37–42.

[11] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping," in *Proc. of the Intl. Symp. on Low Power Electronics and Design*, 2010, pp. 189–194.