

Energy-Aware Software Design for Exascale Computing Systems*

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

System designers, operating system programmers and application developers are equally challenged by the advent of exascale computing systems. Fundamental concepts have to be revisited and emerging issues often demand entirely new approaches to succeed. To keep the consumption of limited energy resources and the associated costs at a minimum it is essential to build energy-efficient exascale systems. For this it is required that all parties involved are working towards a common goal in order not to impede each other.

To increase energy efficiency, current proposals are mostly targeting at runtime power management concepts which control exascale applications. In contrast to this, we propose a tooling infrastructure for *energy-aware programming* which is tightly integrated into the design process of exascale computing systems. Early during each development phase our approach assists system designers and application developers to make energy-aware design decisions targeting at improving the energy efficiency of the overall system.

2 Motivation

Today’s operating systems have sophisticated subsystems responsible for runtime power management and careful resource consumption to increase the overall system’s energy efficiency. Usually this is achieved by exploiting energy saving features exposed by underlying hardware components. Energy saving features such as dynamic voltage and frequency scaling [1, 2] and device-specific sleep states [3, 4] are common approaches and well-studied areas in both academia and practice. In contrast to runtime-driven approaches, energy-aware compilers optimize code prior to execution. Loop optimizations [5, 6] and architecture-specific instruction set extensions [7, 8] have proven to be effective.

Individual nodes of large-scale distributed systems perform local power management decisions to achieve a local optimum with regard to energy efficiency. Distributed power management components make sure that a global power management policy is enforced jointly across all participating nodes. Resources of individual physical nodes are being utilized optimally by migrating processes and aggregating work.

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

**Contact author: Timo Hönig <thoenig@cs.fau.de>

From a global point of view, distributed processes cooperatively pursue a common goal for saving energy by reacting on dynamic runtime aspects of the system. Low workload situations are handled by consolidating work first, which is then followed by taking down spare computing nodes. Vice versa, excess load is encountered by distributing work to an increased number of nodes. For efficient power management components, exascale computing systems will require fine-grained control over devices [9]. Power management capabilities need to reach out to many components of the system, besides components such as CPUs this includes network infrastructure components required for operation. This implies that today’s runtime power management approaches need to be scaled to the dimension of exascale systems. However, this alone will not be enough.

The tremendous energy requirements of exascale systems [10] need to be addressed at different layers and demand for a holistic approach: *application and operating system software co-design*. To achieve this, we propose to augment the design processes of both, operating system software and application software with so-called *energy consumption indicators* which system designers and developers can rely on during development. The importance of such measures is based on the fact that energy consumption depends on the actual application logic which is a result from design decisions during software development. Assuming that developers are being supplied with meaningful energy consumption estimates for program code (e. g., within the development environment), their knowledge could be exploited to reduce the resulting program code’s energy footprint.

Much the same as unit tests are used to automatically verify the correctness of program code, and static analysis tools are used to verify timing constraints, energy consumption indicators can be used to check whether program code meets certain energy consumption constraints. Designing *energy-efficient* exascale systems will require corresponding tooling support, so that developers can rely on energy consumption indicators in order to optimize program code of system software for energy efficiency. Usually this implies restructuring the program logic or choosing a different algorithm to achieve the required functionality in a more energy-efficient manner. Further, operating system runtime components can exploit this a priori knowledge for switching into more efficient operating modes in order to save resources and therefore increase the system’s energy efficiency.

3 The SEEP Approach

In order to reason about the energy efficiency of program code it is required to gain insight into its runtime behavior. Recently, we have presented results of our research targeting at energy-aware programming [11, 12] using the SEEP framework which analyzes source code in a multistep process (see Figure 1) for determining energy consumption estimates. The framework provides energy consumption estimates for heterogeneous target platforms, without the need to execute the program code on the actual targets.

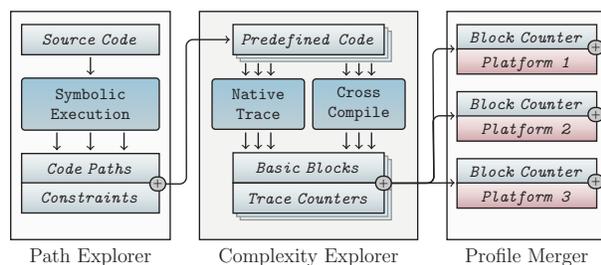


Figure 1: The SEEP architecture [11]

SEEP explores source code using symbolic execution techniques [13, 14]. Possible code paths of the source code under test are being extracted hereby. Further analyses of the code paths eventually lead to parametrized energy consumption estimates. To improve program code, analysis results (i. e., energy consumption estimates) are used by developers either immediately („online”) or results are used to create energy estimation profiles for later use („offline”). Programming libraries can be augmented with these energy estimation profiles in order to provide quick energy consumption estimates at function level and to shortcut future analysis processes.

4 Energy-Aware Exascale System Software

Developing system software for exascale systems not only means to create custom-made solutions in terms of high performance (in all its various facets) as past experience shows [15, 16], but also asks for energy estimation mechanisms deeply embedded into the design process of all software components. Only with such mechanisms system designers can evaluate consequences of their decisions during programming. Energy hogs revealed by applying techniques such as SEEP can be resolved by different measures. Either the application code is being optimized or restructured in a more energy-efficient way without sacrificing other mandatory system properties (e. g., by choosing algorithms which are more energy-efficient) or the composition of the system as a whole needs to be reconfigured (e. g., by deploying better suited operating system components or by exchanging hardware components).

In an iterative process, changes applied to the program code and the system architecture can be reviewed for effectiveness. As a result, both program logic and composition of the system are being optimized for energy efficiency proactively during the time of development. Only after this process has been carried out, common energy saving techniques are being applied. These commonly include static approaches (e. g., compiler optimizations) and runtime approaches (e. g., dynamic voltage and frequency scaling and hardware sleep states).

5 Applicability

The described SEEP approach is well suited for model driven software development methods which largely benefit from automated tooling support. Guided by developers, software models can be refined with the help of corresponding tools. For distributed computing scenarios the approach can be applied to commonly used HPC computing frameworks (e. g., Open MPI). By augmenting the corresponding programming libraries of the framework with energy consumption indicators, energy consumption estimates would be available during the time of creation of HPC applications.

In conjunction with further performance and code analysis tools developers are supplied with the information required to reason about possible trade-offs which effect performance, portability, and energy consumption. Even minor modifications of the system can have a major impact on the overall system performance as small energy savings add up and integrate over time. This is important, as recent advances in semiconductor technology and processor design [17] have yielded novel power characteristics which are important for system developers. The processor’s operating mode (determined by the program code) varies by up to a factor of five with regard to energy efficiency.

6 Conclusion

Being confronted with the sheer complexity of exascale systems, tooling support for creating energy-efficient software will be inevitable for developers. Complementing today’s common energy-saving techniques (i. e., runtime power management, compiler optimizations) we propose an upstream process which assists developers in optimizing program code and system structure during the time of development. Challenges induced by the massive scale of exascale systems must be answered with holistic approaches such as SEEP, as the problem of creating energy-aware software needs to be attacked beginning at the earliest stage of development in order to exploit the vital knowledge of system developers.

7 References

- [1] A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proceedings of the 16th International Conference on Supercomputing*, pages 35–44, 2002.
- [2] A. Weissel and F. Bellosa. Process cruise control: Event-driven clock scaling for dynamic power management. In *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 238–246, 2002.
- [3] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.
- [4] D. Meisner, B. T. Gold, and T. F. Wenisch. Powernap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 205–216, 2009.
- [5] V. Delaluz, M. T. Kandemir, N. Vijaykrishnan, and M. J. Irwin. Energy-oriented compiler optimizations for partitioned memory architectures. In *Proceedings of the 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 138–147, 2000.
- [6] V. Delaluz, M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, A. Sivasubramaniam, and I. Kolcu. Compiler-directed array interleaving for reducing energy in multi-bank memories. In *Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, pages 288–293, 2002.
- [7] N. Vijaykrishnan, M. T. Kandemir, M. J. Irwin, H. S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using SimplePower. In *Proc. of the 27th Annual International Symp. on Computer Architecture*, pages 95–106, 2000.
- [8] M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and W. Ye. Influence of compiler optimizations on system power. In *Proceedings of the 37th Annual Design Automation Conference (DAC)*, pages 304–307, 2000.
- [9] C. Hsu and W. Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 1–9, 2005.
- [10] K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, et al. Exascale computing study: Technology challenges in achieving exascale systems, 2008.
- [11] T. Höning, C. Eibel, R. Kapitza, and W. Schröder-Preikschat. SEEP: Exploiting symbolic execution for energy-aware programming. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, pages 17–22, 2011.
- [12] T. Höning, R. Kapitza, and W. Schröder-Preikschat. ProSEEP: A proactive approach to energy-aware programming. In *Proceedings of the USENIX Annual Technical Conference, Poster Session*, pages 1–2, 2012.
- [13] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, 19(7):385–394, 1976.
- [14] C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation*, pages 209–224, 2008.
- [15] S. Wheat, A. Maccabe, R. Riesen, D. Van Dresser, and T. Stallcup. PUMA: An operating system for massively parallel systems. *Scientific Programming*, 3(4):275–288, 1994.
- [16] J. Cordsen, T. Garnatz, A. Gerischer, M. Gubitoso, U. Haack, M. Sander, and W. Schröder-Preikschat. Vote for PEACE – Implementation and performance of a parallel operating system. *IEEE concurrency*, 5(2):16–27, 1997.
- [17] S. Jain, S. Khare, S. Yada, V. Ambili, P. Salihundam, S. Ramani, S. Muthukumar, M. Srinivasan, A. Kumar, S. Gb, et al. A 280mv-to-1.2 v wide-operating-range ia-32 processor in 32nm cmos. In *IEEE Solid-State Circuits Conference Digest of Technical Papers*, pages 66–68, 2012.