

Managing Competing Goals with a Self-aware Runtime System

Henry Hoffmann CSAIL, MIT hank@csail.mit.edu

Introduction

The constraints and complexity of extreme-scale computing systems make them extremely difficult to program. The challenge stems partly from the need to meet multiple – often competing – goals, such as maximizing performance while minimizing energy consumption. Additional difficulty arises because these systems are dynamic and must continue to function in the face of both changing application workloads and unreliable, failure-prone components.

Programming extreme scale systems to meet multiple constraints and function in the face of unforeseen events is a challenge beyond most application developers. Meeting this challenge requires expertise in the application domain and a deep systems knowledge to balance competing goals. In addition, adjusting to dynamic fluctuations, such as workload variance or component failure, requires further knowledge in the design and implementation of adaptive systems.

The SEEC Model and Runtime System

To address the challenge of programming extreme-scale systems, we propose a novel *self-aware computing* model and runtime system, called SEEC [10]. SEEC allows multiple, independent components to be combined into self-adaptive, or autonomic, computing system which adjusts its behavior to meet multiple goals and automatically adapt to environmental changes. Like all self-adaptive systems, SEEC is characterized by the presence of an *observe-decide-act*, or ODA, loop [12, 13]. In the SEEC model, a dedicated runtime system continuously monitors application level goals (observe) and system-level resources (actions) and determines how to use resources to meet goals (decide). One of SEEC’s unique features is that multiple developers contribute to the construction of the ODA loop, with each concentrating on their area of expertise.

Observe

In SEEC, applications explicitly state their goals and progress using the Application Heartbeats API [8]. The API’s key abstraction is a heartbeat; applications use a function to emit heartbeats at important intervals, while additional API calls specify goals in terms of this heartbeat. SEEC currently supports three application specified goals: performance, accuracy, and power. Performance is specified as a target heart rate or a target latency between specially tagged heartbeats. Accuracy goals are specified as a *distortion*, or linear distance from an application defined nominal value [11], measured over some set of heartbeats. Power and energy goals can be specified as target average power for a given heart rate or as a target energy between tagged heartbeats. Heartbeat data can be read by any other process in the system.

Act

In the SEEC model, applications provide goals while system components specify *actions* that change the behavior of the system. SEEC supports a range of actions specified from the application-level [2, 11], system software level [7, 14], and the hardware level [9]. SEEC does so by providing an interface that all system components use to specify available actions. This interface is designed to be general and support actions exposed by different developers working at different levels of the system stack. Actions are specified by describing the *actuators* that implement them. In SEEC, an actuator is a data object with: a name, a list of allowable settings, a function that changes the setting, a set of axes which the actuator affects (*e.g.*, performance and power), and the effects of each setting on each axis. These effects are listed as multipliers over a nominal setting, whose effects are 1 on all axes. Each actuator specifies a *delay*, which is the time between when it is set and when its effects can be observed. Finally, each actuator specifies whether it works on only the application that registered it or if it works on all applications, so that SEEC can distinguish between adaptations specified at application-level (*e.g.*, changing algorithms) and adaptations that affect the whole system (*e.g.*, allocating cores).

Decide

SEEC’s runtime system automatically selects actions to meet goals while reducing cost. The SEEC decision engine is designed to work without prior knowledge of the applications which it will support. In addition, the runtime system will need to react quickly to changes in application load and fluctuations in available resources. To meet these requirements for handling unknown applications and volatile environments, the SEEC decision engine is designed with multiple layers of adaptation as described in [10]. At the lowest-level, SEEC acts as a classical control system, taking feedback, in the form of heartbeats, and using it to tune actuators to meet goals [15]. The classical control system works well given prior knowledge about the application’s behavior. Additional layers of adaptation, including adaptive control and machine learning based techniques [16], allow the SEEC runtime to allocate resources efficiently without prior knowledge of the application, or when the behavior of the actuator diverges from the predicted behavior.

Related Work

Self-aware, or autonomic, computing has been proposed as one method to deal with the rising complexity of computer systems [12, 13], and adaptive systems have been implemented in both hardware [1, 4–6] and software [17]. One limitation of existing approaches is that they typically do not

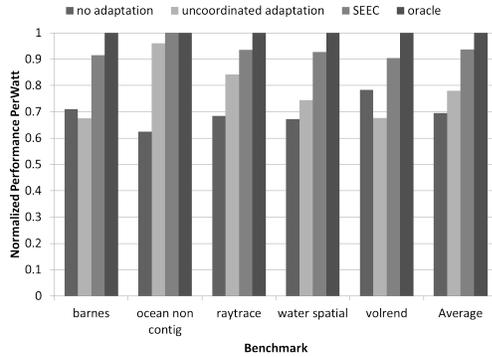


Figure 1. SPLASH benchmarks under adaptive schemes.

support adaptation as a first-class object. Instead, they adopt a *closed* design which is inaccessible to other components of the system. While this approach completely insulates other developers from the complexity of adaptive system design, it leads to additional difficulties. Specifically, closed adaptive systems 1) a fixed set of adaptations and 2) that they are the only adaptive component in the system. These assumptions can lead to problems if the set of available adaptations changes or if multiple adaptive components are deployed on the same system.

Figure 1 shows experimental results demonstrating how SEEC overcomes some limitations of uncoordinated, closed adaptive systems. The figure shows performance per Watt for 5 of the SPLASH2 [18] benchmarks running on an eight core Linux x86 system, which supports adapting clock speed and voltage (through DVFS) and assignment of application threads to cores. Results are shown for four different adaptive schemes. “no adaptation” shows results when the same speed and core count must be used for all applications. “uncoordinated adaptation” shows the results when independent systems adapt DVFS and core assignment. “SEEC” shows the results when SEEC controls both adaptations. Finally, “oracle” shows the best that can possibly be achieved with no overhead and perfect knowledge of the future.

The results demonstrate that uncoordinated adaptation is, on average, only slightly better than not adapting at all. In fact for 2 of the 5 benchmarks, uncoordinated adaptation is worse than not adapting. This phenomenon occurs because uncoordinated adaptation can enter combinations of states that are suboptimal even though the individual components are behaving optimally with their local knowledge.

In contrast, SEEC avoids suboptimal configurations because the ODA loop is open and different adaptive components contribute to its construction. This allows adaptations at different levels to be coordinated. For example, this interface can be used to describe both operating system-level actions (*e.g.*, allocation of cores to an application [15]) and hardware-level actions (*e.g.*, reconfiguration of the hardware data cache [3]). To support this model, adaptive components must be designed to expose adaptations instead of attempting to adapt as a closed system.

Assessment of Approach

Challenges Addressed One challenge of exascale computing is that systems are evaluated on multiple, frequently competing, metrics. This creates an additional programming burden to understand not just performance, but also power, and reliability goals. The SEEC approach addresses this issue by having programs explicitly state high-level goals and dynamically adjusting the system to meet those goals.

Maturity SEEC has been used to manage power on embedded systems [14] and to create adaptive applications [11]. A great deal of work has been done to evaluate different decision mechanisms for the SEEC runtime [16]. To date, SEEC has been tested on single node systems. It has been tested on up to 16-core physical systems and up to 256-core simulated systems. However, there are still several challenges that need to be addressed to bring the approach to exascale.

Uniqueness One challenge of bringing SEEC to exascale is that decisions will have to be made across nodes. The Application Heartbeats Interface already supports transferring data to remote nodes. However, there are still open questions about how to make decisions across multiple nodes. Some possibilities include adopting a hierarchical decision mechanism and exploring the use of economic models that allows distributed decisions to be made without moving the system into a suboptimal state.

Novelty SEEC is distinguished by its treatment of the ODA loop as a first class object which is assembled from disparate parts. Application developers contribute by setting goals and making it easy to observe key metrics of application success. Systems developers contribute by describing actuators available in the system. The SEEC runtime system contributes by coordinating the efforts of different system components to reach good global outcomes based on user-defined goals. The SEEC approach differs from other approaches featuring closed adaptive components which cannot coordinate with the rest of the system.

Applicability Many of the challenges of exascale systems are faced by other systems and many of the solutions should benefit these areas as well. As mentioned above, SEEC has applicability to embedded systems and the development of adaptive applications. A distributed SEEC which works at scale across nodes could be applicable to high-performance embedded computing as well as data centers.

Effort One drawback of this approach is that it is not incremental, but requires modification to several parts of the existing system stack. Applications need to be modified to emit heartbeats and goals. System components need to be modified to describe the actuators they support. So far, we have found these types of modifications fairly unobtrusive and straightforward. In addition, the SEEC runtime system needs to be modified to support distributed decision making. This last issue is likely where the bulk of the effort will be spent.

References

- [1] D. H. Albonese, R. Balasubramonian, S. G. Dropsho, S. Dwarkadas, E. G. Friedman, M. C. Huang, V. Kursun, G. Magklis, M. L. Scott, G. Semeraro, P. Bose, A. Buyuktosunoglu, P. W. Cook, and S. E. Schuster. Dynamically tuning processor resources with adaptive processing. *Computer*, 36:49–58, December 2003.
- [2] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. PetaBricks: A language and compiler for algorithmic choice. In *PLDI*, 2009.
- [3] R. Balasubramonian, D. Albonese, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, MICRO 33, pages 245–257, New York, NY, USA, 2000. ACM.
- [4] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *MICRO*, 2008.
- [5] S. Choi and D. Yeung. Learning-based smt processor resource distribution via hill-climbing. In *ISCA*, 2006.
- [6] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O’Boyle. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO ’43, pages 485–496, Washington, DC, USA, 2010. IEEE Computer Society.
- [7] J. Eastep, D. Wingate, M. D. Santambrogio, and A. Agarwal. Smartlocks: lock acquisition scheduling for self-aware synchronization. In *ICAC*, 2010.
- [8] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments. In *ICAC*, 2010.
- [9] H. Hoffmann, J. Holt, G. Kurian, E. Lau, M. Maggio, J. E. Miller, S. M. Neuman, M. E. Sinangil, Y. Sinangil, A. Agarwal, A. P. Chandrakasan, and S. Devadas. Self-aware computing in the angstrom processor. In *DAC*, 2012.
- [10] H. Hoffmann, M. Maggio, M. D. S. and Alberto Leva, and A. Agarwal. SEEC: A General and Extensible Framework for Self-Aware Computing. Technical Report MIT-CSAIL-TR-2011-046, MIT, November 2011.
- [11] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for responsive power-aware computing. In *ASPLOS*, 2011.
- [12] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, January 2003.
- [13] R. Laddaga. Guest editor’s introduction: Creating robust software through self-adaptation. *IEEE Intelligent Systems*, 14:26–29, May 1999.
- [14] M. Maggio, H. Hoffmann, M. D. S. and Anant Agarwal, and A. Leva. Power optimization in embedded systems via feedback control of resource allocation. *IEEE Transactions on Control Systems Technology (to appear)*.
- [15] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Controlling software applications via resource allocation within the heartbeats framework. In *CDC*, 2010.
- [16] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Decision making in autonomic computing systems: comparison of approaches and techniques. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC ’11, pages 201–204, New York, NY, USA, 2011. ACM.
- [17] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–42, 2009.
- [18] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. *SIGARCH Comput. Archit. News*, 23:24–36, May 1995.