Self-aware Computing in the Angstrom Processor

Henry Hoffmann[1] Jim Holt[1,2] George Kurian[1] Eric Lau[1] Martina Maggio[3]
Jason E. Miller[1] Sabrina M. Neuman[1] Mahmut Sinangil[4] Yildiz Sinangil[4]
Anant Agarwal[1] Anantha P. Chandrakasan[4] Srinivas Devadas[1]

[1]MIT Computer Science and Artificial Intelligence Laboratory [2]Freescale Semiconductor
[3]Lund University [4]MIT Microsystems Technology Laboratories

[1] {hank,jholt,gkurian,elau,jasonm,sneuman,agarwal,devadas}@csail.mit.edu, [2]jim.holt@freescale.com
[3]maggio.martina@gmail.com, [4]{sinangil,koken,anantha}@mit.edu

ABSTRACT

Addressing the challenges of extreme scale computing requires holistic design of new programming models and systems that support those models. This paper discusses the Angstrom processor, which is designed to support a new Self-aware Computing (SEEC) model. In SEEC, applications explicitly state goals, while other systems components provide actions that the SEEC runtime system can use to meet those goals. Angstrom supports this model by exposing sensors and adaptations that traditionally would be managed independently by hardware. This exposure allows SEEC to coordinate hardware actions with actions specified by other parts of the system, and allows the SEEC runtime system to meet application goals while reducing costs (*e.g.,* power consumption).

Categories and Subject Descriptors

C.1.3 [**Other Architectural Styles**]: Adaptable architectures

General Terms

Performance, Design, Experimentation

Keywords

Adaptive Systems, Self-aware Computing

1. INTRODUCTION

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

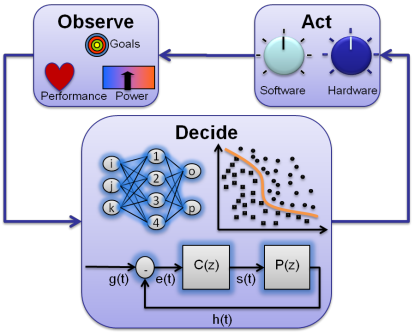

Figure 1: The Observe-Decide-Act loop in SEEC.

Programming extreme scale systems to meet multiple constraints and modify behavior in the face of unforeseen events is a challenge beyond most application developers. Doing so requires expertise in the application domain, but also a deep systems knowledge to balance performance against competing goals like power efficiency. 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 Angstrom project [29] addresses the challenge of programming extreme-scale systems by supporting a novel *self-aware computing* model, called SEEC [15]. SEEC makes it easy to create self-adaptive, or autonomic, computing systems which can alter their 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 [18, 23]. The Angstrom system continuously monitors its goals (observe) and available resources (actions) using a decision engine to determine how best to use resources to meet goals (decide) given the current state of the system.

One of the unique features of Angstrom compared to prior work in adaptive systems is that every component of the system, from applications to hardware, is designed to support adaptation as a *first-class object*. In practice, this means that these components all contribute to the specification of the observe-decide-act loop as illustrated in Figure 1. In this model, applications use one interface to specify goals (*e.g.,* a video encoder should run at thirty frames per second),

This work was funded by the U.S. Government under the DARPA UHPC program. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

DAC 2012, June 3-7,2012, San Francisco, California, USA.

while system software and hardware use a separate interface to specify actions (*e.g.,* allocating processing cores, changing cache configuration). A runtime decision system then determines how to use the available actions to meet goals while reducing cost.

This paper provides an overview of the SEEC model and describes the current design of the Angstrom processor, with a focus on those features that explicitly support SEEC. Angstrom is a proposed 1000-core, massively manycore processor that supports self-aware computing by exposing a wide array of actions (in the form of different hardware configurations) and observations (including both traditional performance counters [34] and energy counters [31]) to the SEEC runtime system. Simulations of 256 core Angstrom systems show that exposing hardware adaptation to a software management system has the potential to improve performance per watt by an average of over 100% compared to a non-adaptive system.

The rest of this paper is organized as follows. Section 2 discusses related work in adaptive computing. Section 3 summarizes the SEEC model, while Section 4 describes the Angstrom processor's support for this model. Section 5 presents an evaluation of the SEEC model on both an existing system and on simulations of future Angstrom systems. The paper concludes in Section 6.

2. BACKGROUND

Self-aware, or autonomic, computing has been proposed as one method to deal with the rising complexity of computer systems [18, 23], and adaptive systems have been implemented in both hardware [2, 5, 10, 11] and software [32]. One limitation of these approaches is that they typically do not support adaptation as a first-class object. Instead, they propose *closed* adaptive systems that are not accessible by other components of the system. While this approach completely insulates application programmers from the complexity of adaptive system design, it can lead to other difficulties. Specifically, many hardware-based approaches assume a fixed set of adaptations, which exist exclusively in hardware and are unable to incorporate application specific goals. Similarly, many software-based approaches assume that the hardware is fixed and thus make no attempt to coordinate with lower-level adaptations.

As an example of a closed adaptive hardware system, consider the resource manager described by Bitirgen et al [5]. This system uses a neural network to allocate resources to a multi-programmed multicore system, with a goal of optimizing total system throughput, or total number of programs completed. This system does an excellent job of coordinating resources to meet goals, but it works with a fixed set of hardware resources and optimizes only a fixed goal of total system throughput. This system would not be able to incorporate additional adaptations specified at the software level or work with application specific goals (like achieving a desired frame rate for a video encoder).

To illustrate the problems of composing closed adaptive systems, we present the following experiment. Using the Graphite simulator [28], we run the barnes application from the SPLASH2 benchmark suite on a multicore system with two possible adaptations: the total number of cores assigned to it (from 1-64, by powers of 2), and the size of the L2-cache on each core (from 16-256 KB, by powers of 2). For each combination of core allocation and cache size, we measure

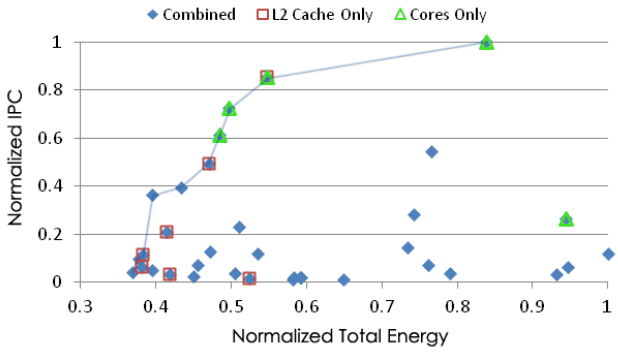

Figure 2: Efficiency of closed adaptive systems.

the performance of the application and the total energy consumed. The results are shown in Figure 2, where the x-axis shows energy and the y-axis shows instructions per second. The solid diamond points represent all tested configurations. The squares show configurations that appear optimal for a closed system which only considers cache adaptations. The triangles show possible configurations for a system that only considers core allocations. The best configurations are the ones with highest performance and lowest total energy; *i.e.,* the Pareto-optimal frontier which is depicted by those diamond points that are connected by a line in the figure. Notice that both triangles and squares appear to the right of the Pareto frontier, and these points represent configurations that closed systems would believe to be optimal, but, in fact, are sub-optimal for the overall system.

To avoid sub-optimal configurations, the SEEC model provides a general interface allowing adaptations supported by different system components to be described by their designer and then manipulated by the SEEC runtime decision system. For example, this interface can be used to describe both operating system-level actions (*e.g.,* allocation of cores to an application [26]) and hardware-level actions (*e.g.,* reconfiguration of the hardware data cache [4]). Given this information, the SEEC runtime system can coordinate adaptation to keep the system on the Pareto optimal curve shown in Figure 2. To support this model, hardware must be explicitly designed to expose adaptations instead of attempting to adapt as a closed system.

3. THE SEEC MODEL

The key feature of the SEEC model is its open ODA loop where different components of the system contribute to the specification of observations, actions, and decisions. This section summarizes the model; more detailed information is available in [15].

3.1 Observe

In order to change something, it is essential to measure the value to be changed. In SEEC, applications explicitly state their goals and other system components measure whether those goals are being met. SEEC uses the Application Heartbeats API [14] to specify application goals and progress. 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 [16], measured over some set of heartbeats. Power and energy goals can be specified as target average power for a given heartrate or as a target energy between tagged heartbeats. All goals are expressed using a C/C++API so that applications can explicitly state these goals. A second API allows other components of the system to observe the current value of any of these metrics and thus evaluate whether or not the goals are being met.

3.2 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 [3, 16], system software level [12, 25], and the hardware level as exposed by the Angstrom processor (see Section 4).

As discussed in Section 2, SEEC works to avoid suboptimal combinations of actions by providing an interface that all system components use to specify available actions. This interface must be general enough to 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).

3.3 Decide

SEEC's runtime system automatically and dynamically selects actions to meet goals while attempting to minimize cost. The SEEC decision engine is designed to handle general-purpose environments and the SEEC runtime system will often have to make decisions about actions and applications with which it has no prior experience. 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 general and volatile environments, the SEEC decision engine is designed with multiple layers of adaptation as described in [15]. 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 [26]. 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 [27], 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.

4. ANGSTROM SUPPORT FOR SEEC

Angstrom is a manycore system design targeting the integration of 1000 cores onto a single chip. A full description of the design is beyond the scope of this paper, so we focus on features of the architecture explicitly designed to support the SEEC model.

4.1 Observation

The Angstrom processor design supports SEEC by providing visibility into the hardware in the form of traditional performance counters, event probes, and non-traditional sensors. This information allows SEEC's runtime decision engine to diagnose either why goals are not being met, or whether there might be a lower cost set of actions that would achieve the same goal.

Performance counters provide valuable insight into the behavior of an application on a particular hardware architecture. Unfortunately, many existing systems limit the number of counters that can be read simultaneously by software. This limitation means that application tuning requires multiple profiling runs and prevents dynamic exploitation of performance counter information. The Angstrom design exposes multiple performance counters that are memory-mapped and can be read by any level of the software stack. These count simple events such as: memory operations, cache hits and misses, pipeline stall cycles, network flits sent and received, etc. These are useful for assessing average behavior over a period of time but since they must be polled by software, they cannot be queried too frequently.

To reduce the overhead of monitoring the performance counters and watch for rare events, Angstrom's design includes *event probes*, that can be associated with a performance counter or some other piece of state with the processor. They contain a *trigger* register and a programmable comparator that continually watches for the state to match the trigger. The comparator can be set to different operations including: equal, less than, greater than and their logical inverses. In addition, a mask can be specified to compare only selected bits. When a match occurs, an interrupt can be generated or an event record can be placed in a small hardware queue.

Besides observing processor state, Angstrom includes sensors to monitor things like temperature, voltage, battery charge, and energy consumption [31]. This allows the runtime decision engine to react to changing environmental conditions (such as cooling failures or dying batteries) as well as observe how its actions impact these quantities to handle goals like minimizing power consumption or limiting temperature extremes. We expect some of these sensors to be deployed in a fine-grained manner to measure variations between the 1000 cores.

4.2 Action

As further support for the SEEC model, the Angstrom processor exposes a number of different actions or different hardware configurations. Keeping with the theme of building an open adaptive system, Angstrom provides these "knobs" but relies on the SEEC runtime system to set them in coordination with other adaptations specified at the software level. This section discusses some of the adaptations exposed by the Angstrom processor at both the intra- and inter-core level.

4.2.1 Intra-core Adaptation

In the Angstrom design, each core is capable of running at different voltages and frequencies. Operating the processor

designs at lower voltage levels has been shown to increase energy efficiency, as with the voltage-scalable 32-bit microprocessor design demonstrated in [17]. This processor operates at peak performance with nominal voltages while supporting an energy efficient mode at 0.54 V with only 10.2 pJ/cycle energy consumption. Similarly, making each Angstrom core capable of running at different voltage and frequency levels will optimize them for applications with limited energy budgets and time varying processing loads.

Technology scaling is fueling integration of larger on chip caches in processor design (*e.g.,* up to 50 MB [30]). In order to enable ultra-low power consumption, Angstrom cores need to feature voltage-scalable SRAMs. Conventional SRAMs cannot work at low-voltage levels due to stability problems. Thus, recent work has focused on implementing different bit-cell topologies [7, 6] and peripheral assist circuits [21, 33] to enable operation down to sub-VT levels.

Reconfiguration of the local caches is shown to reduce power consumption for the same performance [4]. Disabling unnecessary parts of the Angstrom caches (sets and ways) will help SEEC to optimize power and performance tradeoffs. This adaptation can be beneficial both for applications with small working sets and applications with large working sets that do not achieve much locality on their data.

4.2.2 Inter-core Adaptations

Angstrom supports dynamic adaptation of the on-chip network by enabling software and hardware to interact in achieving goal-driven tradeoffs between performance and efficiency. This is accomplished with three architecture features: express virtual channels (EVC) [8], bandwidth adaptive networks (BAN) [9], and application-aware oblivious routing (AOR) [22].

EVC allows flits to attempt bypassing of buffering and arbitration within a network router by proceeding straight to switch and link traversals. This can significantly reduce latency and can also reduce energy spent in buffering flits. Previous EVC research has demonstrated significant performance/energy tradeoff capability; Angstrom enhances this by introducing a software interface to routing tables maintained in the network hardware. This information is then used by EVC logic to manage virtual channels.

A BAN can rapidly adjust bisection bandwidth to adapt to changing network conditions. This is done by using bidirectional links with arbitration logic and tristate buffers that prevent simultaneous writes to the same wire. A hardware bandwidth allocator governs the direction of a link. Angstrom expands on prior BAN approaches by exposing configuration parameters in the bandwidth allocator to software while still providing for fine-grained bandwidth management in hardware.

AOR algorithms can produce deadlock-free routes while maximizing satisfaction of flow demands on the network, achieving better throughput than traditional oblivious routing approaches because optimization is done using global application knowledge. At the same time the router remains simple because routes are configured via a routing table. Unlike previous offline routing algorithm approaches, Angstrom provides for online routing computation by exposing the routing table to software, allowing the system to adapt to changing application characteristics over time.

Angstrom also supports adaptation of the cache-coherence protocol used between cores. For some applications, directory-based cache-coherence provides the best performance and energy consumption [13]. However, for other applications it is more efficient to use a shared-NUCA (non-uniform cache access) protocols because it provides for a large shared cache capacity and reduces the total number of off-chip accesses [20]. The ARCc architecture has shown that combining these protocols and adaptively selecting the best on a per application basis can improve performance and energy efficiency [19]. Angstrom adopts the ARCc approach of providing multiple coherence protocols and exposes these adaptations to SEEC for management.

4.3 Decision

Although self-aware optimizations are capable of dramatically improving the behavior of applications, they do not come for free. Some resources must be devoted to making runtime decisions to have a dynamic, adaptive system. To help reduce the costs of runtime decision making the Angstrom processor contains specialized, low-power cores called *partner cores*, which we describe below. More detail is available in [24].

Each main core in the Angstrom design has a partner core associated with it. These two cores are tightly integrated so that the partner core can inspect and manipulate state (including performance counters and configuration registers) within the main core. The partner core also has access to the event queues fed by event probes. The partner core targets a lower performance point than the main core and is designed to take much less area and energy. It has a simplified pipeline, smaller caches and fewer functional units. It is designed to run at lower clock frequencies and makes heavy use of low-power circuit techniques, requiring less energy per operation, and making it more efficient to run dynamic optimization code on the partner core than the main core. We estimate that each partner core will consume about 10% of the area and 10% of the power of a main core.

5. EVALUATION

This section presents experiments designed to evaluate the SEEC model and highlight the benefits of Angstrom's support for it. We first use SEEC to control applications on an existing Linux/x86 system. We then collect data on simulations of a 256 core Angstrom processor and use that data to build analytical models that predict how SEEC will behave on such a system.

5.1 Benchmark Applications

Both experiments use the same set of five benchmark applications selected from the SPLASH2 benchmark suite: barnes, ocean (non contiguous), raytrace, water (spatial), and volrend [35]. Each application is instrumented with the Application Heartbeats API to emit goals. In both experiments, applications request a target performance and SEEC is tasked with meeting that performance while minimizing power consumption.

5.2 SEEC on Existing Systems

We begin by presenting results using SEEC to control these benchmarks on an existing architecture with some adaptive capabilities, but which was not explicitly designed to support SEEC. Specifically, we use a Dell PowerEdge R410 server with two quad-core Intel Xeon E5530 processors running Linux 2.6.26. The processors support seven power

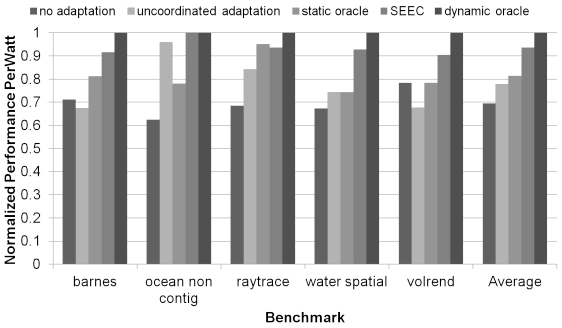

Figure 3: SEEC on a Linux/x86 system.

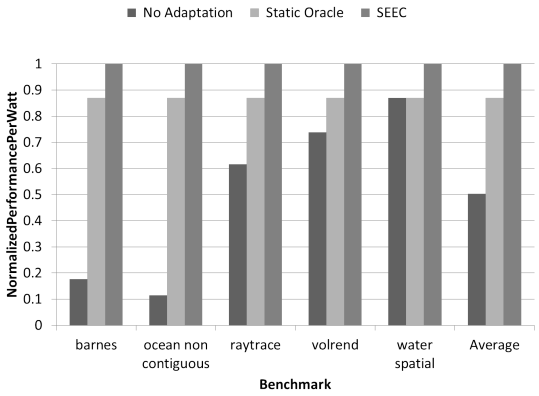

Figure 4: Anticipated SEEC results on Angstrom.

states with clock frequencies from 2.4 GHzto 1.6 GHz. The `cpufrequtils` package enables software control of the clock frequency (and thus the power state). We use a WattsUp device to sample and store the average consumed power over 1 second intervals [1]. All benchmark inputs have been expanded to allow them to run for significantly more than 1 second. The measured power ranges from 220 watts (at full load) to 80 watts (idle), with a typical idle power consumption of approximately 90 watts.

Each benchmark is launched on a single core set to the minimum clock speed and it requests a performance equal to half the maximum achievable. SEEC's runtime attempts to meet this application specified performance goal while minimizing power consumption using the following actions: changing the number of cores assigned to the application, changing the clock speed of the cores assigned to an application, and changing the number of active (or non-idle) cycles assigned to the application. For each application, we measure the average performance and power consumption over the entire execution of the application. We then subtract out the idle power of the processor and compute performance per Watt as the minimum of the achieved and desired performance divided by the power beyond idle.

To show the benefits of SEEC we compare to several other approaches. First, we compute the best that could be achieved in a system with *no adaptation*; *i.e.,* when all applications use the same number of cores and the same clock speed, which provides a baseline that any adaptive system should hope to beat. Next, we compare to a system where adaptation is *uncoordinated*; *i.e.,* separate instances of the SEEC runtime system control cores, clock speed, and idle cycles but do not coordinate with each other. Uncoordinated adaptation represents what happens when multiple closed adaptive systems work together. Third, we compare to a *static oracle*, which adapts resource usage to the individual application, but only assigns resources once per application. The static oracle represents what is possible with an adaptive system which does not make fine-grained adaptation. Finally, we compare to a *dynamic oracle* which adapts resource allocation at every heartbeat to the best configuration for the application's next heartbeat. Obviously, the dynamic oracle cannot be built in practice, but instead its actions are computed after the fact by post processing empirical data for each application. The dynamic oracle represents an upper bound on the possible benefits of adaptation because it has 1) no overhead and 2) perfect knowledge of the future.

The results of this experiment are shown in Figure 3, where benchmarks are labeled on the x-axis and performance per Watt is on the y-axis. For each benchmark, performance per Watt is normalized to that achieved by the dynamic oracle (so the maximum possible is one). Separate bars show the value achieved for each approach of: no adaptation, uncoordinated adaptation, SEEC, and dynamic oracle.

The results show the benefits of using SEEC to provide coordinated management of adaptive systems. SEEC is able to outperform the uncoordinated adaptive system by over 20% and the static oracle by over 15%. Indeed, for some applications, uncoordinated adaptation is actually worse than not adapting because uncoordinated adaptations oscillate through suboptimal resource allocations. In contrast, the SEEC model exposes these adaptations as first class objects so it can coordinate and avoid suboptimal configurations. SEEC is able to achieve almost 94% of the performance of the dynamic oracle, which demonstrates that SEEC is responsive and low overhead.

5.3 SEEC on Angstrom

This section explores the benefits of running SEEC on a future Angstrom architecture with 256 cores. We use the Graphite simulator [28] to model an Angstrom processor that can adapt its cache size (from 32-128 KB, by powers of 2), the number of cores assigned to an application (from 1-256, by powers of 2), and the voltage (0.4, 0.8 V) and frequency (100, 500 MHz) of those cores. We use simulation because existing processors do not have as many cores or available adaptations.

We run each benchmark in every possible configuration and use this data to compute the performance per watt as described above and construct the performance of a system with no adaptation, and the performance of a system which adapts using the static oracle. We then compute the predicted value of using SEEC by multiplying the value of the static oracle by the multiplier that SEEC achieved on the x86 architecture in the above section.

The results for this experiment are shown in Figure 4. This figure shows the benchmark on the x-axis and the performance per Watt on the y-axis. For each benchmark, performance per Watt is normalized to that achieved by the predicted SEEC result. Separate bars show the value achieved for: no adaptation, the static oracle, and our estimate of how SEEC will behave on the Angstrom architecture.

The results show that the Angstrom architecture indeed benefits more from adaptation than modern architectures, because of the increased number of available adaptations. As a specific example, barnes contains large amounts of par-

allelism that can be exploited by allocating cores, but that is not the case for all applications. In fact, the non-adaptive system allocates 64 cores out of a possible 256 to the system because it provides the best average performance per watt efficiency across all applications. On the other hand, a static oracle allocates 256 cores for running barnes, outperforming the non-adaptive configuration by over 5x.

Overall, the static oracle outperforms the non-adaptive system by 72% on average. Under the assumption that SEEC outperforms the static oracle by 15%, we predict that SEEC on the Angstrom architecture can achieve an improvement of 100% over a standard non-adaptive system. Indeed, future manycore architectures that are designed explicitly to leverage the self-aware computing model will enjoy SEEC considerably more than current systems.

6. CONCLUSION

This paper has presented the design for the Angstrom architecture. Angstrom is a multicore which addresses the challenge of extreme-scale computing by explicitly supporting a self-aware computational model. This support comes in two forms. First, Angstrom provides additional visibility into the hardware state through performance and energy counters. Second, Angstrom exposes hardware adaptations so that they can be managed by the SEEC runtime system in coordination with other, software based actions. We have demonstrated how self-aware computing provides a benefit on existing architectures and presented evidence that this approach will provide an even greater benefit when implemented on an architecture like Angstrom with explicit support for the model.

7. REFERENCES

[1] Wattsup .net meter. `http://www.wattsupmeters.com/`.

[2] D. H. Albonesi, 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.

[3] 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.

[4] R. Balasubramonian, D. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In *MICRO*, 2000.

[5] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *MICRO*, 2008.

[6] B. Calhoun and A. Chandrakasan. A 256kb sub-threshold SRAM in 65nm CMOS. In *ISSCC*, 2006.

[7] L. Chang, D. Fried, J. Hergenrother, J. Sleight, R. Dennard, R. Montoye, L. Sekaric, S. McNab, A. Topol, C. Adams, K. Guarini, and W. Haensch. Stable SRAM cell design for the 32 nm node and beyond. In *Symposium on VLSI Technology*, 2005.

[8] C.-H. O. Chen, N. Agarwal, T. Krishna, K.-H. Koo, L.-S. Peh, and K. C. Saraswat. Physical vs. Virtual Express Topologies with Low-Swing Links for Future Many-core NoCs. In *NOCS*, 2010.

[9] M. H. Cho, M. Lis, K. S. Shim, M. Kinsy, T. Wen, and S. Devadas. Oblivious Routing in On-Chip Bandwidth-Adaptive Networks. In *PACT*, 2009.

[10] S. Choi and D. Yeung. Learning-Based SMT Processor Resource Distribution via Hill-Climbing. In *ISCA*, 2006.

[11] C. Dubach, T. M. Jones, E. V. Bonilla, and M. F. P. O'Boyle. A predictive model for dynamic microarchitectural adaptivity control. In *MICRO*, 2010.

[12] J. Eastep, D. Wingate, M. D. Santambrogio, and A. Agarwal. Smartlocks: lock acquisition scheduling for self-aware synchronization. In *ICAC*, 2010.

[13] A. Gupta, W. Weber, and T. Mowry. Reducing memory and traffic requirements for scalable directory-based cache coherence schemes. In *ICPP*, 1990.

[14] 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.

[15] H. Hoffmann, M. Maggio, M. D. Santambrogio, A. Leva, and A. Agarwal. SEEC: A General and Extensible Framework for Self-Aware Computing. Technical Report MIT-CSAIL-TR-2011-046, MIT, November 2011.

[16] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard. Dynamic knobs for responsive power-aware computing. In *ASPLOS*, 2011.

[17] N. Ickes, Y. Sinangil, F. Pappalardo, E. Guidetti, and A. Chandrakasan. A 10 pJ/cycle ultra-low-voltage 32-bit microprocessor system-on-chip. In *ESSCIRC*, sept. 2011.

[18] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, January 2003.

[19] O. Khan, H. Hoffmann, M. Lis, F. Hijaz, A. Agarwal, and S. Devadas. ARCc: A case for an architecturally redundant cache-coherence architecture for large multicores. In *ICCD*, 2011.

[20] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *ASPLOS*, 2002.

[21] T.-H. Kim, J. Liu, J. Keane, and C. Kim. A High-Density Subthreshold SRAM with Data-Independent Bitline Leakage and Virtual Ground Replica Scheme. In *ISSCC*, 2007.

[22] M. Kinsy, M. H. Cho, T. Wen, E. Suh, M. van Dijk, and S. Devadas. Application-Aware Deadlock-Free Oblivious Routing. In *ISCA*, 2009.

[23] R. Laddaga. Guest editor's introduction: Creating robust software through self-adaptation. *IEEE Intelligent Systems*, 14:26–29, May 1999.

[24] E. Lau, J. E. Miller, I. Choi, D. Yeung, S. Amarasinghe, and A. Agarwal. Multicore performance optimization using partner cores. In *HotPar*, 2011.

[25] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Power optimization in embedded systems via feedback control of resource allocation. *IEEE Transactions on Control Systems Technology*, PP(99):1–8.

[26] 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.

[27] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva. Decision making in autonomic computing systems: comparison of approaches and techniques. In *ICAC*, 2011.

[28] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald III, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A distributed parallel simulator for multicores. In *HPCA*, 2010.

[29] MIT. The MIT angstrom project. `http://projects.csail.mit.edu/angstrom`, 2012.

[30] R. Riedlinger, R. Bhatia, L. Biro, B. Bowhill, E. Fetzer, P. Gronowski, and T. Grutkowski. A 32nm 3.1 billion transistor 12-wide-issue Itanium processor for mission-critical servers. In *ISSCC*, 2011.

[31] E. Rotem, A. Naveh, D. R. amd Avinash Ananthakrishnan, and E. Weissmann. Power management architecture of the 2nd generation Intel Core microarchitecture, formerly codenamed Sandy Bridge. In *Hot Chips*, Aug. 2011.

[32] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–42, 2009.

[33] M. Sinangil, H. Mair, and A. Chandrakasan. A 28nm high-density 6T SRAM with optimized peripheral-assist circuits for operation down to 0.6V. In *ISSCC*, 2011.

[34] P. Team. Online document, `http://icl.cs.utk.edu/papi/`.

[35] 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.