A Self-Adaptive Approach to Efficiently Manage Energy and Performance in Tomorrow's Heterogeneous Computing Systems

E. M. G. Trainiti, G. C. Durelli, A. Miele, C. Bolchini, M. D. Santambrogio
Politecnico di Milano, Italy
Dipartimento di Elettronica Informazione e Bioingegneria
ettore.trainiti@mail.polimi.it
{gianlucacarlo.durelli, antonio.miele, cristiana.bolchini, marco.santambrogio}@polimi.it

***Abstract*—ICT adoption rate boomed during the last decades as well as the power consumption footprint that generates from those technologies. This footprint is expected to more than triple by 2020. Moreover, we are moving towards an on-demand computing scenario, characterized by varying workloads, constituted of diverse applications with different performance requirements, and criticality. A promising approach to address the challenges posed by this scenario is to better exploit specialized computing resources integrated in a heterogeneous system architecture (HSA) by taking advantage of their individual characteristics to optimize the performance/energy trade-off of the overall system. Better exploitation although comes with higher complexity. System architects need to take into account the efficiency of systems units, i.e. GPP(s) either alone or with a single family of accelerators (e.g., GPUs or FPGAs), as well as the applications workload, which often leads to inefficiency in their exploitation, and therefore in performance/energy. The work presented in this paper will address these limitations by exploiting self-adaptivity to allow the system to autonomously decide which specialized resource to exploit for a carbon footprint reduction, due to a more effective execution of the application, optimizing goals that the user can set (e.g., performance, energy, reliability).**

I. Introduction

Computing systems have to operate in uncertain scenarios, where some of the characteristics of the surrounding environment can affect the device behavior. Let's consider, for example, a mobile phone that has to operate in different signal conditions, trying to consume the least amount of battery possible. Another important issue to be considered in the computing systems domain is that devices are usually designed to have a service life of several years. In such a scenario, the management of performance and delivered Quality of Service (QoS) is one of the main problems which is investigated with self-adaptive computing systems [1]. In these systems, the activity of gathering runtime information, the *observe* or *monitoring* phase [2], is crucial. In fact, the availability of accurate and appropriate information highly impacts the efficacy of their self-management capabilities.

On one hand there is the increasing importance of non-functional requirements: the *perceived value* of a digital system, i.e. features that are not completely reducible to the functionalities, are getting ever more important. Two relevant examples of such non-functional constraints are power consumption and reliability. However there are many other potential dimensions, that lie at the border of what can be called functionality, that impact the user experience of a digital system; an example can be the quality of the outputs, like in different audio and video qualities for a multimedia device. Meeting such constraints (or optimizing the associated figures) is getting more and more difficult, mainly because of the exponential increase of environmental interactions and conditions in which devices are required to operate.

On the other hand, devices structure is evolving towards solutions characterized by an increase in number and complexity of interacting "peer" elements, at various levels (e.g.: cores on a multicore architecture, concurrent programs in a multiprogrammed operating system, number of threads within a single application). Meeting non functional constraints requires, most of the times, a coordination among all those elements, for any possible working condition. It is evident that statically foreseeing, at design time, the actions that must be taken in order to maximize non-functional constraint satisfaction for all the possible scenarios is already way beyond feasibility.

Within this context, dynamic resources allocation has become even more interesting as a research direction with commodity computing gaining momentum and bringing new requirements in terms of elasticity and flexibility and with the growing importance of cloud computing [3]. Control structures are being re-engineered to work in a decentralized fashion at data center scale [4] and research is going towards practical (i.e., effective and low-overhead) use of software monitoring up to that same scale [5]. However, efficient monitoring data aggregation methods become useless if flexibility and efficiency are not retained starting from the basic blocks, i.e. the sensors. Efficiency and low overhead come embedded in hardware counters, but can be difficult to obtain when building a software infrastructure for higher-level metrics, which has the advantage of being directly meaningful to users and administrators, easing the task of setting the objectives [6, 7].

In this paper we detail the key features of a self-adaptive framework to efficiently manage heterogeneous systems. In particular we focus our attention on the description of the different components participating in the three control phases actuated by the framework: (i) the monitoring phase, (ii) the decision phase, and (iii) the actuation. The framework has been implemented and tested on a real machine with the possibility to exploit two different controlling mechanisms based on heterogeneous mapping and thread scaling.

The remainder of the paper will present the work related to our research topic (Section II), and then will detail the structure of the framework we propose (Section III). Section IV will illustrate the results of the deployment of the framework on a real machine discussing how two adaptation policies might be used to control different applications competing for shared resources. Finally, Section V concludes the paper.

II. Related Work

Several works in literature (e.g. [8], [9],[10], [11]) have presented approaches to design and implement adaptive computing systems that have been proven to be quite effective. The key difference with respect to the approach proposed in our work is that most of these previous works were not focusing their approach/technique to the online, self-aware adaptation, which is a key aspect of our approach.

A self-aware, adaptive, or autonomic computing system is able to alter its behavior in some beneficial way without the need for human intervention [12, 13, 14]. Such a system can observe its behavior, make decisions, and take actions to meet desired goals at runtime. This adaptability promises to reduce the burden modern computing systems place on application developers; however it also rises new challenges for the creation and the usage of such systems [15]. Some example systems that adapt their behavior include multicore chips [16, 17] that manage resource allocation [18], provide resources for critical sections [19], optimize for power [16], and assemble heterogeneous cores from many small cores[17]. In addition, languages and compilers have been developed to support adapting application implementation for performance [20, 21] or power [22, 23]. Operating systems are also a natural fit for self-aware computation [24, 25, 26, 27]. Self-aware techniques are also prominent in industry as shown by companies such as IBM [14] (e.g., IBM Touchpoint Simulator, the K42 Operating System [25]), Oracle (e.g., Oracle Automatic Workload Repository [28]), and Intel (e.g., Intel RAS Technologies for Enterprise[29]).

Several important self-optimizing hardware approaches have been implemented using machine learning approaches, eg. [30], [31] and [32]. In [30] the authors proposed a priority-lock library for synchronization for heterogeneous multicores that uses machine learning and feedback from applications to reassign at runtime priorities to the locks leveraging the observed performances. The self-optimizing memory controller presented in [31] can optimize its scheduling policy using a reinforcement learning approach which allows it to estimate the performance impact of each action it can take to better respond to the observations made on the system state. In [32], a framework for management of resources in a multiprocessor chip is presented. This framework uses a neural network to optimize on-chip resource allocation with the goal of maximizing system throughput.

In [33, 34] a control theory based approach has been presented to design, in a unified framework, adaptive real time systems based on specification of desired dynamic behavior. These two papers presented very interesting results that we would like to use in supporting our decision of using a control theoretical approach as the basis of the decision phase of our framework. Even if we do also see many relations between the two research areas, we think that at this stage of the work we can avoid to compare our work to the ones belonging to this area. In a real time system (e.g RTOS), a system able to guarantee a maximum most of the time for each operation that it is performing, the designers of an application, which has to respond to events, has to guarantee that these responses will happen deterministically. The kind of problems that we are addressing do not have to guarantee such properties but we want to be able to prove the ability of the system to adapt itself to meet the desired goals, which is not a required characteristic of a RT system.

In [35, 36, 37] the authors presented a very interesting approach to adapt processor resources while achieving Quality of Service (QoS) on contemporary multicore processors that run media streaming applications. The key idea [35] behind these works is to use a pre-defined set of scenarios to predict resources for decoding media streams and to be able to annotate the applications that have to be executed with this scenario information. A scenario is an entry in a provider's database which is defined as a group of frames with similar decode complexity from various media streams [36]. As presented in [37] a crucial aspect of this approach relies on the ability of identifying the scenarios that have to be used to annotate the media streams and to predict the required resources at runtime. In contrast to these works, our proposed framework is designed to react and prevent dangerous situations at runtime. Thanks to its decision engine it can predict the required resources and manage/organize them among several applications, without the need of a pre-defined database.

III. The Proposed Self-Aware Framework

To realize our vision of a self-aware computing system we must: (i) enable system adaptation policies to determine whether the specified goals are met, and (ii) enable adaptive systems to make informed decisions among multiple possible actions. Addressing the first challenge requires a general framework that allows applications to communicate their goals to the adaptive systems, that must in turn be able to measure applications progress toward them. Addressing the second challenge requires imbuing adaptive systems with the ability to dynamically and automatically react, adapting itself to the new conditions.

A. A bird's eye view on the proposed mechanism

The self-adaptive framework uses input from applications and systems developers to implement a closed-loop system with three distinct phases: *Observation*, *Decision*, and *Action*. This ODA loop is characteristic of control systems; during the observation phase the system collects information, which is fed to the decision phase. During the decision phase the system determines whether recent observations warrant a change in behavior and, if needed, what form this change should take. In particular the entity in charge of taking decisions, the *Orchestrator* - shown in Figure 1, constantly monitors the system and the applications running on it, and uses these observations to take informed decisions on how to manage the system and the applications. A preliminary version of this mechanism has been presented in [38]. If adaptation is

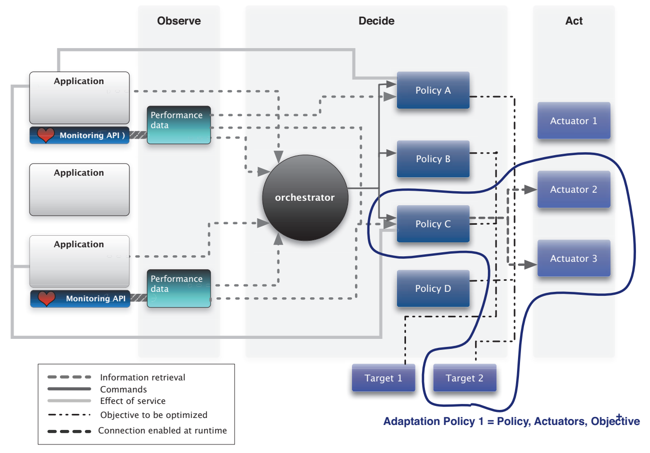

Fig. 1. Self-Aware system based on the ODA-loop. The orchestrator is the central component of the decision phase. Collecting data through the monitoring APIs, it takes informed decision on what action can be activated to improve the status of the systems (apps under execution and the system itself).

desired, the action phase implements the adaptation dictated by the Orchestrator. The proposed framework supports this form of closed-loop execution by generalizing the observation and decision phases, providing standard techniques that work with a broad range of actions.

In the proposed solution, it is possible to identify three distinct participants: application developer, system developer or system administrator, and the Orchestrator. The application developer's only responsibility is to indicate the application's goals and current progress toward those goals [1]. To simplify this task, we make use of the HRM monitoring infrastructure[2]. Application developers use HRM functions to register goals and progress and the self-aware framework automatically reads this data using additional HRM functions.

The system developer is responsible for specifying a set of actions and a function that can affect those actions. System developers use the proposed framework to design new system adaptation policies by specifying a set of actions that can be taken to manage the specific system, allowing the self-aware framework to automatically handle the observation and decision making phases of closed loop execution. In our case, the actions to adjust the application behavior are based on a control-theoretical framework. This means that a simple model for the application has been developed and a regulator to control the application speed towards the goal is devised. Furthermore, system developers are also in charge of providing the system the possibility to express goals. With respect to the system goals, as an example, the system developers might set temperature or power caps to optimize power related costs in the datacenter and it is up to the *Orchestrator* to manage computational resources to fulfill these goals. In this scenario, the Orchestrator consists of (i) a programming model that permits the applications to expose a goal (in a high level metric), and the system to present some overall requirements (e.g. the energy budget or a power cap); (ii) a set of knobs that the *Orchestrator* itself can tune to adapt the behavior of the application to run-time needs; finally, a set of adaptation mechanisms for controlling heterogeneous and parallel workloads.

[1]In this work, to make an application's goals and progress known to the rest of the system, we are using the HRM framework [7].

[2]There are other solutions that can be used for the this phase (e.g., Application Heartbeats API [6], or solution based on performance counters [39, 40], PAPI [41]) but we decided to use HRM for three main reasons: (i) the source code is public available, (ii) the modifications that have to be done to the application source code are minimal (iii) we can adopt just one solution to express application's goals and to monitor the system at runtime.

B. *The Orchestrator*

The Orchestrator relies on the concept of feedback-loops that are at the basis of a self-aware and adaptive system. These loops are generally referred to as **ODA** (Observe, Decide and Act) loops, as shown in Figure 1.

The **O**bserve phase collects data from all the adaptive applications and from various system components. Observation is what makes a system self-aware. Based on the observation, the system understands its state, its current progress, and its possible future actions. The type of data collected in this phase strictly depends on the objectives that have to be optimized by the feedback loop. As an example, we can think of collecting performance measurement for the applications QoS and system wide measurements as, for example, power consumption of the various elements on the chip for energy saving optimizations. Such data are collected by the Orchestrator, as shown in Figure 2, that has a global vision of the system, being aware of which applications are currently being monitored, their goals and which policies and actuators are available to adapt the applications behaviour and the system itself in order to meet local and global goals.

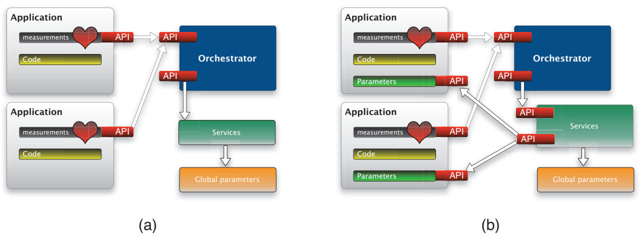

Fig. 2. (a) The adaptation policies enabled by the Orchestrator tune global parameters (i.e., DVFS, threads scheduling/core mapping, etc.) at runtime. (b) An adaptive-enabled application – orchestrator interaction. An adaptive-enabled application exposes to the Orchestrator a set of parameters that are tuned at runtime as well as the global parameters available in the system.

Participants in the **D**ecide phase are the Orchestrator, a set of policies that could be activated to perform adaptation and a set of targets to be met. A target is a specific goal that an application wants to reach or that must be enforced on the system. The **A**ct phase is performed by one or more actuators, in charge of executing specific actions on the system to vary actual performance. The combination of a policy (a way to take decisions), a target (the quantity to be optimised and its desired value) and one or more actuators (tuning points or knobs) is called a *adaptation policy*. For instance, we may want to optimize performance or temperature by acting on the frequency of the processors; thus, the actuator is the DVFS knob, the target is the performance or the temperature and the policy is a mechanism (based on a PID, a neural network or any other decision strategies) that decides how to turn the knob.

Finally the adaptation policy is the triplet of the mentioned items. Actually, various adaptation policies may be active at the same time and their decisions may influence each other risking to create conflicts; the Orchestrator is in charge of the cooperation and coexistence of several active adaptation policies.

Information is exchanged between the different components by means of shared memory or message passing primitives, and all the complexity is hidden behind a set of APIs which ease the usage of the system. A central entity is in charge of forwarding information of monitors and actuators to adaptation policies upon changes in the system (e.g., arrival of new controlled applications). The decoupling between the parts permits to easily extend the current implementation with new components.

IV. A WORKING EXAMPLE: ADAPTIVE POLICIES FOR RUNTIME RESOURCE ALLOCATION

This section presents a working example where the proposed Orchestrator has been used to manage runtime resource allocation to execute various kinds of workloads. The heterogeneous system architecture used in our evaluation is a workstation running the Debian GNU/Linux operating system, equipped with an Intel Core i5 750 dual core CPU running at 2.7 GHz (with Hyperthreading enabled), and a Nvidia GeForce GT 240 GPU capable of executing OpenCL and CUDA kernels. As a benchmark, we use an option pricing application based on the Black and Scholes formula.

We will now show how two different solutions have been integrated in the system illustrated in the previous section. The first one aims at managing heterogeneous workloads, while the second one allows to handle multi-threaded applications.

Within our approach, each of these solutions implements an adaptation policy that exploits a set of monitors and actuators to achieve a given goal. The possibility to continuously monitor the environment allows one to implement control theoretical approaches exploiting a closed feedback loop and also to learn the effect of the action that can be taken using a trial-and-error approach. The monitoring infrastructure used in this work is inspired by the HRM [7].

A. Heterogeneous Mapping

The first scenario considers applications containing computational kernels that can run on different heterogeneous resources. The system exposes a user-level API that the application designer uses to provide some information about the computational kernels of the application. Using this interface, the designer specifies the available implementations (e.g., CPU, GPU, FPGA) of each computational kernel in the application registering a function handler for the implementation and a function handler (if needed) that performs data movement between resources. Figure 3 illustrates an example of a service able to control the throughput of an application deciding at runtime on which computational unit it has to execute.

The programmer is in charge of registering the different implementations for a given kernel through an API and set the desired goal. The Orchestrator will then be aware of all the running applications that want to exploit heterogeneous resources and will autonomously manage the resource assignment among them in order to achieve their goals. Upon each execution of the kernel, the system automatically decides on which of the available units (CPU or GPU in the example) to execute in order to respect the desired throughput. The example of Figure 3 shows the performance of the application when running only on the GPU (in green) and only on the CPU (in red). If our user sets a goal (blue area in the figure) the Orchestrator tries to meet the specified goal by switching between CPU and GPU (blue line).

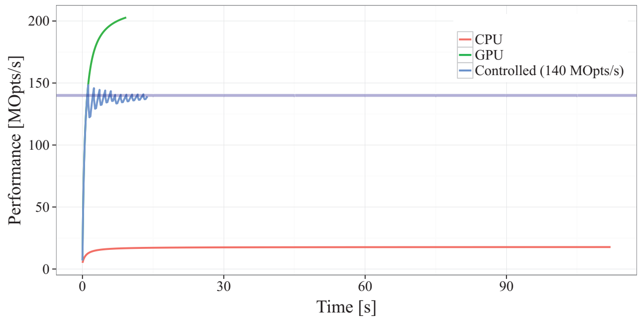

Fig. 3. The Orchestrator controlling one application. The lines represent three independent runs of the Black and Scholes benchmark: one running on the GPU (green line), the second running on CPU (red line) and the last one using the Orchestrator to control the application (blue line). The performance goal is set to be 140 MOptions/s.

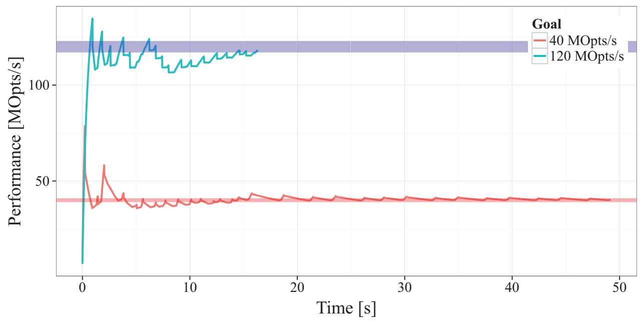

Fig. 4. The Orchestrator managing two instances of the Black and Scholes benchmark with different goals. CPU and GPU mapping is managed directly by the Orchestrator to meet the applications goals.

This policy allows the use of the minimum amount of resources needed to execute an application, leading to the possibility to consolidate multiple user applications on the same machine to increase its utilization. Figure 4 shows two applications being executed on the same machine, belonging to different users who set different goals. At runtime the Orchestrator allocates the CPU and GPU to the applications in order to meet their goals. The applications are two instances of the Black and Scholes algorithm used in the previous test case, but the problem here is that the testing machine has only one GPU which is contended by the two applications. The applications set two different performance goals of 40 and 120 MOptions/s through the monitoring infrastructure (colored band in the figure). The average performance is represented by the continuous stroke and meeting the expressed goal means that at the end of the execution this line should match the set goal for each one of the applications. As the

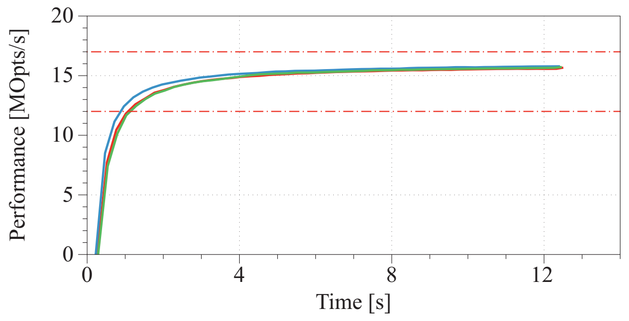

Fig. 5. Standard OpenMP behavior when application are consolidated.

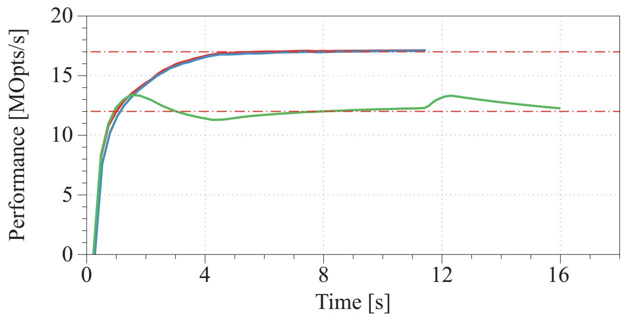

Fig. 6. Effect of adaptive thread scaling enforced by the Orchestrator.

figure shows both applications end meeting their goals; in this case the Orchestrator reduces the contention on the GPU allowing an application to execute on it only when it needs to increase its throughput. Note that as opposed to Figure 3 the performance of the controlled applications are not smooth when they are colocated. This is due both to the fact that with the performance goal set they both need to access the GPU and also to the fact that when executing on the CPU they both use OpenMP to parallelize the execution on the available cores. If used unconstrained, OpenMP supposes that only one multithreaded application is running generating a number of threads equal to the number of available cores, leading to an high contention when multiple applications are colocated. The developed allocation mechanism is then not only able to allocate CPU and GPU to meet the performance, but is able to do that in situation where there is an high contention on CPU resources. The next section will describe how we can manage the problem of high contention in case of multithreaded applications.

B. *Thread Scaling*

Another scenario where the Orchestrator can be usefully exploited is to reduce the contention on resources for parallel workloads. We developed a service that decides at runtime the number of threads that an application has to use to execute a given computational kernel. This service monitors at runtime the performance of the applications running in the system and varies the number of threads an application can use. Each execution of the kernel uses the number of threads decided by the Orchestrator. Figure 5 shows the behavior of a workload composed of three instances of the Black and Scholes benchmark parallelized using OpenMP when executed on the same machine at the same time.

The users' goals are 17 MOpts/s for two of them and 12 MOpts/s for the third one. The OpenMP default behavior consists in executing the applications sharing fairly the available resources, thus all the three instances end up with an average throughput of 15 MOpts/s. This means that two applications missed their goals. Figure 6 shows the effect of the policy implemented to manage the thread scaling of OpenMP applications. In this situation, the system during the execution of the test is able to identify the computational power needed by each application, and consequently instructs them to execute with the correct number of threads in order to fulfill the set goals.

It is possible to conclude that exploiting self-adaptiveness with the Orchestrator allows the system to achieve the desired goals for all of the analyzed applications.

V. Conclusions

In this paper we presented the key features of the Orchestrator, a runtime resource manager for heterogeneous architectures, and we illustrated the behavior of the system in controlling diversified applications running in a multiprogrammed environment through some examples. Results show the ability of the Orchestrator to control this kind of applications to meet the goals expressed by the final user. A key point of the research is the use of high level metrics in the monitoring infrastructure: this allows the user to easily express goals on the applications, and the Orchestrator to clearly understand whether the goals are met or not. The system is well suited for managing parallel workloads and can be deployed on desktop workstations and nodes of a computer cluster to control the execution of applications consolidated on a single node.

Acknowledgment

This work was partially funded by the European Commission in the context of the FP7 SAVE project (#610996-SAVE).

References

[1] M. Salehie and L. Tahvildari, "Self-Adaptive Software: Landscape and Research Challenges," vol. 4, no. 2, 2009.
[2] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," vol. 36, no. 1, 2003.
[3] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards Characterizing Cloud Backend Workloads: Insights from Google Compute Clusters," vol. 37, no. 4, 2010.
[4] R. Wang and N. Kandasamy, "On the design of decentralized control architectures for workload consolidation in large-scale server clusters," in *9th*, 2012.
[5] T. Huang, N. Kandasamy, and H. Sethu, "Evaluating compressive sampling strategies for performance monitoring of data centers," in *9th*, 2012.
[6] 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 *7th*, 2010.
[7] F. Sironi, D. B. Bartolini, S. Campanoni, F. Cancare, H. Hoffmann, D. Sciuto, and M. D. Santambrogio, "Metronome: Operating System Level Performance Management via Self-Adaptive Computing," in *49th*, 2012.

[8] G. Mariani, A. Brankovic, G. Palermo, J. Jovic, V. Zaccaria, and C. Silvano, "A correlation-based design space exploration methodology for multi-processor systems-on-chip," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, 2010, pp. 120 –125.

[9] I. Beretta, V. Rana, M. Santambrogio, and D. Sciuto, "On-line task management for a reconfigurable cryptographic architecture," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1 –4.

[10] V.-M. Sima and K. Bertels, "Runtime decision of hardware or software execution on a heterogeneous reconfigurable platform," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1 –6.

[11] K. Sigdel, M. Thompson, A. D. Pimentel, C. Galuzzi, and K. Bertels, "System-level runtime mapping exploration of reconfigurable architectures," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1586640.1587726

[12] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, pp. 1–42, 2009.

[13] "Organic Computing Initiative OCI," http://www.sra.uni-hannover.de/orgcomp/, 2010.

[14] IBM Inc., "IBM autonomic computing website," http://www.research.ibm.com/autonomic/, 2009.

[15] P. Dini, W. Gentzsch, M. Potts, A. Clemm, M. Yousif, and A. Polze, "Internet, GRID, self-adaptability and beyond: Are we ready?" Aug 2004.

[16] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Processor power reduction via single-isa heterogeneous multi-core architectures," *Computer Architecture Letters*, vol. 2, no. 1, pp. 2–2, Jan-Dec 2003.

[17] E. Ipek, M. Kirman, N. Kirman, and J. F. Martinez, "Core fusion: accommodating software diversity in chip multiprocessors," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 186–197, 2007.

[18] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *MICRO 41: Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 318–329.

[19] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt, "Accelerating critical section execution with asymmetric multi-core architectures," in *ASPLOS*, 2009, pp. 253–264.

[20] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N. M. Amato, and L. Rauchwerger, "A framework for adaptive algorithm selection in STAPL," in *PPoPP '05: Proceedings of the 10th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*. New York, NY, USA: ACM, 2005, pp. 277–288.

[21] 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 *Conf. on Programming Language Design and Implementation*, Jun 2009.

[22] W. Baek and T. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *ACM SIGPLAN Conference on Programming Language Design and Implementation*, Jun. 2010.

[23] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger, "Eon: a language and runtime system for perpetual systems," in *SenSys*, 2007, pp. 161–174.

[24] C. Cascaval, E. Duesterwald, P. F. Sweeney, and R. W. Wisniewski, "Performance and environment monitoring for continuous program optimization," *IBM J. Res. Dev.*, vol. 50, no. 2/3, pp. 239–248, 2006.

[25] O. Krieger, M. Auslander, B. Rosenburg, R. W. J. W., Xenidis, D. D. Silva, M. Ostrowski, J. Appavoo, M. Butrico, M. Mergen, A. Waterland, and V. Uhlig, "K42: Building a complete operating system," in *EuroSys '06: Proc. of the 1st ACM SIGOPS/EuroSys Euro. Conf. on Computer Systems*, 2006.

[26] S. Oberthür, C. Böke, and B. Griese, "Dynamic online reconfiguration for customizable and self-optimizing operating systems," in *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*. New York, NY, USA: ACM, 2005, pp. 335–338.

[27] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing controllable computer systems," in *Proceedings of the 10th conference on Hot Topics in Operating Systems*. Berkeley, CA, USA: USENIX Association, 2005, pp. 9–15.

[28] Oracle Corp., "Automatic Workload Repository (AWR) in Oracle Database 10g," http://www.oracle-base.com/articles/10g/AutomaticWorkloadRepository10g.php.

[29] Intel Inc., "Reliability, availability, and serviceability for the always-on enterprise," www.intel.com/assets/pdf/whitepaper/ras.pdf, 2005.

[30] J. Eastep, D. Wingate, M. D. Santambrogio, and A. Agarwal, "Smartlocks: lock acquisition scheduling for self-aware synchronization," in *ICAC '10: Proceeding of the 7th international conference on Autonomic computing*. New York, NY, USA: ACM, 2010, pp. 215–224.

[31] E. Ipek, O. Mutlu, J. F. Martnez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *ISCA '08: Proc. of the 35th Inter. Symp. on Comp. Arch.*, 2008.

[32] R. Bitirgen, E. Ipek, and J. F. Martinez, "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *MICRO 41: Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 318–329.

[33] A. Block, B. Brandenburg, J. Anderson, and S. Quint, "An adaptive framework for multiprocessor real-time system," in *Real-Time Systems, 2008. ECRTS '08. Euromicro Conference on*, 2008, pp. 23 –33.

[34] C. Lu, J. Stankovic, T. Abdelzaher, G. Tao, S. Son, and M. Marley, "Performance specifications and metrics for adaptive real-time systems," in *Real-Time Systems Symposium, 2000. Proceedings. The 21st IEEE*, 2000, pp. 13 –23.

[35] J. Hamers and L. Eeckhout, "Scenario-based resource prediction for qos-aware media processing," *Computer*, vol. 43, no. 10, pp. 56 –63, 2010.

[36] ——, "Resource prediction for media stream decoding," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, 2007, pp. 1 –6.

[37] ——, "Automated hardware-independent scenario identification," in *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, 2008, pp. 954 –959.

[38] C. Bolchini, G. C. Durelli, A. Miele, G. Pallotta, and M. D. Santambrogio, "An orchestrated approach to efficiently manage resources in heterogeneous system architectures," in *In Proc. of IEEE International Conference on Computer Design (ICCD)*, 2015, pp. 221–228.

[39] R. Azimi, M. Stumm, and R. W. Wisniewski, "Online performance analysis by statistical sampling of microprocessor performance counters," in *ICS '05: Proceedings of the 19th Inter. Conf. on Supercomputing*, 2005, pp. 101–110.

[40] B. Sprunt, "Pentium 4 performance-monitoring features," *IEEE Micro*, vol. 22, no. 4, pp. 72–82, Jul/Aug 2002.

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