2nd Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS'13)

5. September 2013
Porto, Portugal

edited by
Tobias Becker

Preface

The field of modern computing sees a continuing trend towards increasingly complex, parallel and heterogeneous architectures as well as distributed and dynamic applications. These developments call for new approaches to design and operate systems that are capable of dealing with uncertainty and changing behaviour. Self-awareness is an emerging field of research in computing that considers systems and applications that gather and maintain information about their current state and environment, reason about their behaviour, and adapt themselves if necessary.

Reconfigurable computing systems, such as the ones using FPGAs, are capable of delivering high performance and efficiency combined with flexibility, and research in reconfigurable applications is well established. We are, however, interested in further reaching applications of reconfigurability to address the challenges mentioned above. Properties such as self-organisation, self-optimisation or self-healing can act as a means to improve flexibility, performance or reliability of applications targeting reconfigurable hardware. Self-awareness extends this line of research and includes aspects such as reasoning, learning and intelligence to a run-time adaptive system.

The Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS) was created to bring together researchers who are active in this field, present their current work, and share their concepts and visions of self-aware systems. The topics of interest for this workshop are:

- Concepts and foundations of self-aware systems.
- Architectures, control, instrumentation and infrastructure for self-aware systems.
- Algorithmic approaches for self-awareness.
- Tools for engineering self-aware systems.
- Advanced autonomous and self-adaptive systems.
- Self-awareness and adaptation in heterogeneous and distributed systems.
- Run-time techniques for adaptive behaviour, including dynamic reconfiguration.
- Applications using self-awareness or self-adaptivity.
- Emergence of self-awareness in adaptive systems.

The second edition of this workshop was held on 5. September 2013 in Porto, Portugal, and co-located with the 2013 International Conference on Field Programmable Logic and Applications (FPL). Of all papers submitted to this workshop, 7 were selected for presentation. In addition, we were able to attract 2 invited talks from established academics in the field, resulting in a diverse program that covers many aspects of self-aware systems. We would like to thank all authors for submitting their work to the workshop. We would also like to thank the program committee for reviewing papers and helping with the paper selection. We gratefully acknowledge the financial support of Awareness, a FET coordination action funded by the European Commission under FP7. Special thanks go the the FPL organisers who helped us co-locating this workshop with FPL 2013.

Tobias Becker, Imperial College London
Marco Platzner, University of Paderborn
Markus Happe, ETH Zurich

Program Committee

David Andrews	University of Arkansas
Tobias Becker	Imperial College London
Markus Happe	ETH Zurich
Michael Hübner	Ruhr-University of Bochum
Enno Lübbers	Intel
Marco Platzner	University of Paderborn
Marco Santambrogio	Politecnico di Milano
Stephan Stilkerich	EADS
David Thomas	Imperial College London
Andy Tyrrell	University of York
Stefan Wildermann	University of Erlangen-Nuremberg
Dong Ping Zhang	AMD

Author Index

Borkmann, Daniel 3

Carro, Luigi 17
Chibamu, Walter 22

Happe, Markus 3
Hinchey, Mike 1

Keller, Ariane 3
Kurek, Maciej 8

Lewis, Peter 22
Liu, Tianchi 8
Luk, Wayne 8, 28

Nacci, Alessandro A. 33
Nazar, Gabriel 17
Neuhaus, Stephan 3

Pilato, Christian 33

Santambrogio, Marco D. 33
Santos, Leonardo 17
Sciuto, Donatella 33
Stilkerich, Stephan 28

Teich, Jürgen 13
Tyrrell, Andy 2
Todman, Tim 28

Wildermann, Stefan 13

Yao, Xin 22

Table of Contents

Invited Talks

Building Intelligent Space Exploration Missions 1
Mike Hinchey . . . . .
From Self-Aware Robotics to Adaptive Silicon Chips: Knobs and Monitors 2
Andy Tyrrell . . . . .

Papers

Autonomic Configuration of Dynamic Protocol Stacks 3
Ariane Keller, Daniel Borkmann, Stephan Neuhaus and Markus Happe . . . . .
Multi-Objective Self-Optimization of Reconfigurable Designs with Machine Learning . . 8
Maciej Kurek, Tianchi Liu and Wayne Luk . . . . .
Decomposing Run-time Resource Management in Heterogeneous Reconfigurable Systems 13
Stefan Wildermann and Juergen Teich . . . . .
Dynamically Shifted Scrubbing for Fast FPGA Repair 17
Leonardo Santos, Gabriel Nazar and Luigi Carro . . . . .
Towards a Dynamic Evolutionary Approach to FPGA Temperature Management 22
Peter Lewis, Walter Chibamu and Xin Yao . . . . .
Using Statistical Assertions to Guide Self-Adaptive Systems 28
Tim Todman, Stephan Stilkerich and Wayne Luk . . . . .
Designing Self-Adaptive Smart Spaces for Energy Saving 33
Alessandro A. Nacci, Christian Pilato, Marco Domenico Santambrogio and Donatella Sciuto . . . . .

INVITED TALK: BUILDING INTELLIGENT SPACE EXPLORATION MISSIONS

Mike Hinchey

Lero–the Irish Software Engineering Research Centre
Limerick, Ireland
email: mike.hinchey@lero.ie

Talk summary

NASA's new age of space exploration augurs great promise for deep space exploration missions whereby spacecraft should be independent, autonomous, and smart. Nowadays NASA increasingly relies on the concepts of autonomic computing, exploiting these to increase the survivability of remote missions, particularly when human tending is not feasible. Autonomic computing has been recognized as a promising approach to the development of self-managing spacecraft systems that employ onboard intelligence and rely less on control links. We describe our work on developing self-management concepts inspired by biological concepts and formally specifying these with particular reference to a NASA concept exploration mission.

About the speaker

Mike Hinchey is Director of Lero–the Irish Software Engineering Research Centre, a multi-institutional research centre funded by Science Foundation Ireland, and Professor of Software Engineering at University of Limerick, Ireland. He was previously Director of the NASA Software Engineering Laboratory at Goddard Space Flight Center and continues to serve as a NASA expert consultant. He is editor-in-chief of Innovations in Systems and Software Engineering: a NASA Journal (Springer) and is currently a Vice President of IFIP and Chair of the IFIP Technical Assembly.

INVITED TALK: FROM SELF-AWARE ROBOTICS TO ADAPTIVE SILICON CHIPS: KNOBS AND MONITORS

Andy Tyrrell

Department of Electronics
University of York, York, UK
email: andy.tyrrell@york.ac.uk

Talk summary

Biological inspiration in the design of computing machines finds its source in essentially three biological models: phylogenesis, the history of the evolution of the species, ontogenesis, the development of an individual as directed by his genetic code, and epigenesis, the development of an individual through learning processes influenced both by their genetic code and by the environment. These three models share a common basis: a one-dimensional description of the organism, the genome and contribute explicitly or implicitly to self-awareness in biological organisms. If one would like to implement some or all of these ideas in hardware (e.g. robots, silicon) can we achieve self-awareness? Do we need specifically designed-for-purpose hardware? This talk will consider some historical work on bio-inspired architectures before moving on to consider some recent work in collective robotics showing forms of self-repair and a new FPGA designed and fabricated specifically for bio-inspired work. It will consider some of the novel features present in this device, such as reconfigurable analogue components, which assist the implementation of capabilities such as self-repair and self-tuning.

About the speaker

Andy Tyrrell received a 1st class honours degree in 1982 and a PhD in 1985 (Aston University), both in Electrical and Electronic Engineering. He joined the Electronics Department at York University in April 1990, he was promoted to the Chair of Digital Electronics in 1998. His main research interests are in the design of biologically-inspired architectures, artificial immune systems, evolvable hardware, FPGA system design and real-time systems. This work has included the creation of embryonic processing array, intrinsic evolvable hardware systems and the immunotronics hardware architecture. He is Head of the Intelligent Systems research group at York. He has published over 280 papers in these areas. He is a Senior member of the IEEE and a Fellow of the IET.

AUTONOMIC CONFIGURATION OF DYNAMIC PROTOCOL STACKS

Ariane Keller, Stephan Neuhaus, Markus Happe [*]

Communication Systems Group
ETH Zurich, Zurich, Switzerland
email: first.last@tik.ee.ethz.ch

Daniel Borkmann [†]

Red Hat
Zurich, Switzerland
email: borkmann@redhat.com

ABSTRACT

The Internet architecture works well for a wide variety of communication scenarios. However, communication in constrained environments with embedded and/or mobile devices requires specialized communication protocols. Additionally, network characteristics often vary in those scenarios, which makes it difficult for a static set of protocols to provide the required functionality. Therefore, we propose a self-aware configuration method for dynamic protocol stacks that allows for the autonomic configuration of individual protocols into a protocol stack. This adaptation happens at run-time and might be triggered by policy changes or by changing network conditions. We demonstrate the efficiency of our self-aware architecture for a networking scenario where the link quality changes over time. In contrast to a static reliable protocol stack we can reduce the communication overhead in terms of sent packets by 28% for a given scenario.

1. INTRODUCTION

In contrast to the beginning of the computing age, nowadays most applications are distributed and interact with other devices. Today's applications are executed on a variety of devices (such as workstations, notebooks, cellphones, sensor nodes) with different processing power and in varying network conditions (such as wireless or wired, trusted or untrusted, etc.). We can no longer assume that a static networking architecture always provides robust and secure communication links with high throughput at low performance overhead and power consumption.

For instance, mobile devices are usually used in dynamic network environments, where the link quality can vary dramatically over time, e.g., when a user moves away from or approaches a WLAN hotspot. Moreover, the user may switch between private and public networks, which may require different privacy modes. Static networking architectures usually lack the flexibility to adapt themselves to dynamic environments in order provide the required communication functionalities at minimal cost.

In the current Internet architecture certain protocols can already adapt themselves to changing communication conditions. However, the overall functionality to be provided by a communication link has to be specified while writing an application and can only be selected from a small pre-defined range. We argue that in order to execute applications optimally in dynamic network environments, we need a self-aware communication architecture, in which the overall functionality autonomously adapts itself to the current network characteristics. Examples of such an adaptation could be the dynamic inclusion of a reliability and/or a privacy block in the protocol stack whenever the network conditions demand them.

In previous work [1, 2] we have already proposed to use dynamic protocol stacks instead of static protocol stacks. Dynamic protocol stacks (DPS) split the networking functionalities into individual functional blocks, which can be dynamically linked with each other in order to form arbitrary protocol stacks. In this paper we extend our work by introducing a self-aware networking architecture that adapts the protocol stacks at run-time to a changing environment.

Specifically, our contributions are:

- We have developed a self-aware network node architecture that supports the autonomic configuration of dynamic protocol stacks.
- We have developed techniques to set up and adapt protocol stacks based on application requirements and the current network condition.
- We have evaluated our self-aware architecture with a real-world scenario and shown that self-adaptation of the protocol stack can reduce the communication overhead in terms of sent packets as compared to static stacks.

The rest of this paper is structured as follows: We first give an overview of related work (Section 2). Then, we present our self-aware networking architecture and our self-adaptation strategies in Section 3. Next, we demonstrate the efficiency of our approach in a real-world networking sce-

[*]The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n° 257906.

[†]This work was performed while affiliated with ETH Zurich.

nario with changing link qualities (Section 4). Finally, Section 5 concludes the paper.

2. RELATED WORK

Already in the early 1990s, Tennenhouse and Wetherall proposed *Active Networks* in which users could inject custom code into the network [3]. This code was associated with a set of packets that traversed the network from the source over several routers to the destination. The code was executed on intermediate nodes and could modify the packets on-the-fly as desired. Less flexible architectures were proposed by the `Click` modular router [4] and `netgraph` [5]. Both `Click` and `netgraph` offer the possibility to combine networking functionalities flexibly. However, they did not focus on run-time reconfiguration. The concepts of *flexibility, modularity, and extensibility* were also recently presented by Ghodsi et al. [6] as the basic requirements for a network architecture that is able to evolve. Wolf et al. [7] argue that a user should be able to *choose* the service that best fits his requirements. In contrast to related work we present a novel self-aware networking architecture which adapts its protocol stack autonomously to react to a changing environment without fine-granular user interaction.

3. SELF-AWARE NETWORKING ARCHITECTURE

In this section we describe our self-aware networking architecture that enables us to dynamically configure protocol stacks. We first discuss the architecture developed for the node-local adaptation and then we focus on the setup and adaptation of the protocol stack between nodes.

3.1. Node-Local Adaptation

Figure 1 shows our self-aware network node architecture, which consists of the following building blocks:

- The **network models** contain the network protocols, the network characteristics, as well as some predictions on how the world might look like in the future.
- The **sensors** provide information such as the signal-to-noise ratio, available energy, or throughput. Sensors can be passive (just observing) or active (inserting probes in the network, observing the reaction).
- The **sensor daemon** collects data from the individual sensors. It offers additional functionality such as sending notifications whenever a monitored value exceeds a specified threshold.
- The **self-adaptation engine** contains a *strategy finder*, which selects the current strategy (minimize power, maximize throughput, etc.), and a *stack builder*, which determines the best stack and adapts the networking core accordingly.
- The **networking core** is responsible for processing network packets. Therefore, it passes a network packet between functional blocks. The details of our networking core are described in [2].

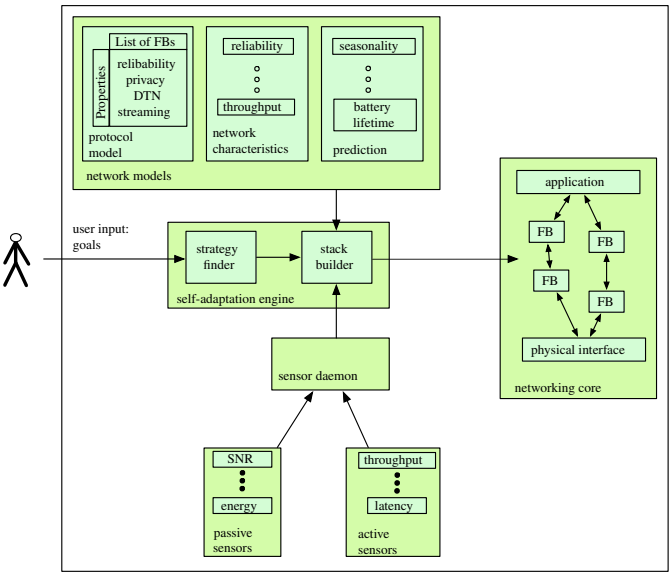


Fig. 1. Overview of the self-aware node architecture.

Building a protocol stack requires the knowledge of (a) the available protocols and (b) the communication requirements. In the Internet architecture, an application solves this problem by using a specific BSD socket type and additional libraries as needed, e.g., for encryption. This setup implies that once the application is written it will always use the same protocols and it cannot make use of newly developed protocols that might fit its needs just as well or even better. In our self-aware networking architecture, the application can specify a set of properties that need to be fulfilled for a given communication. The stack builder then examines the protocol models and finds all protocol stacks that match the requirements. In the current implementation, both protocols and requirements, are specified with simple key words.

3.2. Inter-Node Adaptation

Once all possible stacks are known, a connection to the destination node has to be established. The destination node might not have the required protocols available; therefore, before the communication starts, a protocol stack negotiation phase is executed. First, all possible protocol stacks are sent to the destination node. The destination node decides which protocol stack to use, sets up this protocol stack and sends the chosen configuration back to the source. If the source never receives a reply from the destination, which could happen on a lossy link, the source re-sends the con-

figuration message and waits for the confirmation. After the completion of the negotiation phase, the actual data transmission starts. In order to solve the "chicken and egg problem" of the protocol used for the protocol negotiation phase, we assume that all nodes in a given network segment use the Ethernet protocol. Similarly, if a connection to a node in another segment should be established, the intermediate nodes have to use the same routing protocol.

Upon receiving a data packet, a node has to decide how to process it. In the Internet architecture this decision is based on *next header fields* that are part of each protocol header. For example, in the next header field of the Ethernet protocol it is specified whether the next protocol is `IPv4`, `IPv6`, `ARP`, etc. If the protocol stack is negotiated upfront, this step by step resolution of the next protocol is not necessary, instead, a single identifier per connection can be used. This identifier is calculated by the stack builder as follows: Every functional block has a unique name. In order to obtain a unique name the inverted `url` that is associated with the developer is used. This is similar to the convention for package names in the Java programming language. The unique identifier for the overall protocol stack is then obtained by concatenating the individual names and hashing them. If the identical protocol is implemented by several developers, and their implementations pass an interoperability test, a special interoperability name should be used. Upon packet reception, the Ethernet functional block checks the hash and forwards the packet to the corresponding stack "pipeline".

When the networking conditions change, the self-aware nodes might want to change the protocol stack to add or remove networking functionalities. Identifying a given stack by a unique identifier is also valuable when changing the protocol stack on-the-fly. The negotiation of the new protocol is similar to the negotiation for setting up a protocol. The re-negotiation is executed over the currently used protocol. While adapting the protocol stack packets might be reordered on their way from source to destination. Therefore, special care has to be taken that packets still belonging to the old stack are not processed by the new stack and vice versa. Since the hash that identifies a given stack will change when the protocol stack is changed, also the packets sent over the new stack will be identified with a different hash. This hash is used to dispatch the packet either to the new or the old protocol stack. Figure 2 depicts this change of the protocol stack.

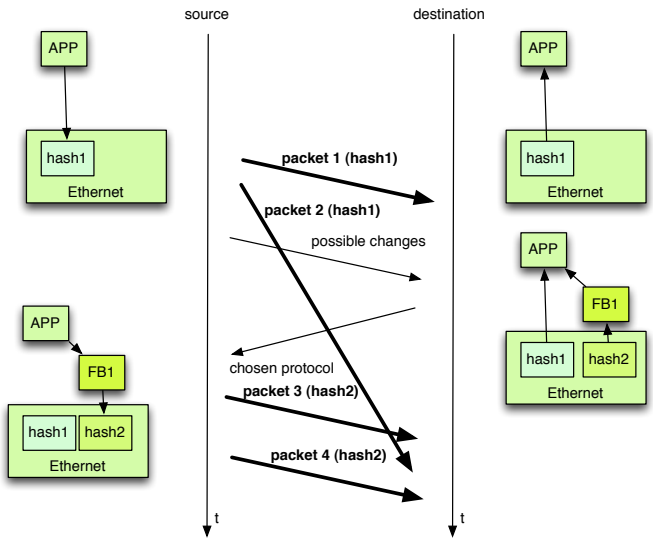


Fig. 2. Updating the dynamic protocol stack over time.

4. EXPERIMENTAL RESULTS

We implemented the self-aware network node architecture as a combination of Linux kernel modules (for the networking core) and user-space tools (for monitoring and configuration). However, it could be implemented on any other operating system as well. Our implementation is designed to scale from small embedded systems up to high-end SMP servers. Applications interact with the network architecture over a new BSD socket family that supports the following socket calls: *open, ioctl, sendto, poll, recvfrom, close*. A library is provided that allows for specifying the communication requirements. The source code of our architecture together with getting started information is available in github at `http://github.org/epics/reconos`.

In order to evaluate the benefits of a self-aware network architecture, we show how our system autonomously adapts itself to changing network conditions. We developed a simple application that mimics a sensor that sends measurement data periodically to a server. We argue that transmitting a packet over a wireless interface costs energy, and therefore should only be performed when necessary. Therefore, we implemented a stack builder that includes an idle repeat request (IRR) reliability protocol in the protocol stack, only when sensors report low link quality. The link quality is determined by a sensor that divides the current with the maximum possible wireless link quality. Our link-quality-aware networking architecture is shown in Figure 3.

We evaluated our architecture on commodity notebooks. In order to obtain reproducible results, we used a wired connection between the test machines and used the Linux traffic control tool `tc` with the `netem discipline` [8] to emulate packet loss. We recorded the link quality between two nodes while walking around in our office building, see Figure 4. We have used this recording as realistic input for our emulation. Simultaneously, we measured that packets got lost, when the link quality was below 35%.

Our stack builder requests to be notified by the sensor daemon when the signal strength falls below a threshold of 40% or increases beyond 50%, see Figure 4. Upon such an

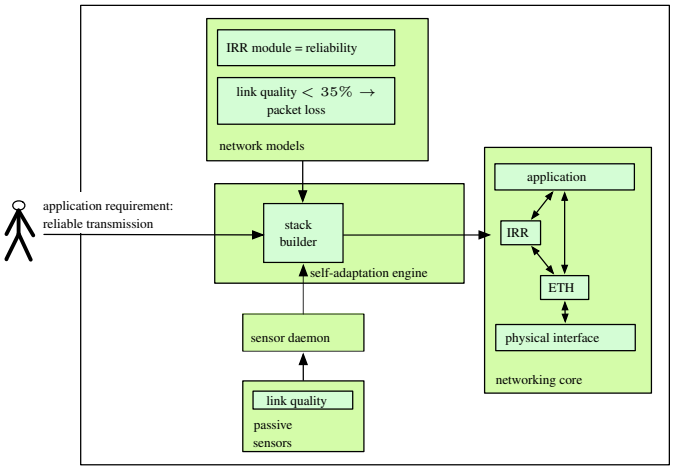


Fig. 3. Implementation of the node architecture for the link-quality-aware protocol stack.

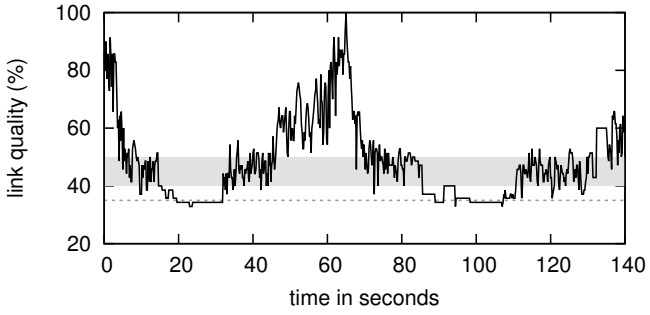


Fig. 4. Measured link quality over 140 seconds. Packets got lost when the link quality was below the dashed line. The DPS is updated when the graph crosses the gray bar.

event, it either inserts the reliability module or it removes the reliability module, and renegotiates the protocol stack with the neighboring node. The lower threshold for renegotiation ensures that the reliability protocol is inserted to the protocol stack before the link quality reaches the critical value of 35%. The upper threshold is used to avoid frequent adaptations of the protocol stack.

For evaluation purposes, we compared the data loss rate and the total number of packets sent for (i) a protocol stack that dynamically adapts itself to the link quality, (ii) a protocol stack that never uses reliability, and (iii) a protocol stack that always uses reliability. We used these measured values to emulate the network conditions on a machine that connected the two test machines.

Table 1 summarizes our results. The configuration with no reliability lost on average 31% of the packets, whereas we didn't observe packet loss in the other two configurations. However, this reliability comes at a price. The overhead (in terms of sent packets) for achieving reliability was 128% for the configuration that was statically configured to use the reliability protocol. The total overhead for the dynamic configuration was 100% split in 60% for sending acknowledgement and retransmission packets and 40% for sending the protocol stack reconfiguration messages. This clearly shows that adaptive protocol stacks can reduce the total communication overhead in dynamic scenarios. However, the adaptation algorithm has to be designed carefully to avoid increasing the total overhead by sending too many stack reconfiguration messages.

Table 1. Comparison between static and autonomous configurations over 140 seconds.

		overhead	
config.	packet loss rate	reliability	reconfig.
unreliable	31%	-	-
reliable	0%	128%	-
autonomous	0%	60%	40%

We also measured the protocol stack reconfiguration time, i.e., the time it takes from an event that triggers a reconfiguration until data can be sent over the new protocol stack. This time is composed of (i) the time to determine and reconfigure the stack on both sides of the communication and (ii) the time to send the reconfiguration messages. We measured a protocol stack reconfiguration time of 806μs whereof 286μs were required for the transmission of the packets (round trip time).

5. CONCLUSION

In this paper we presented a novel self-aware network node architecture. The self-adaptation of the protocol stack is triggered by combining the sensor input with models and goals. The currently implemented self-adaptation techniques allow to insert or remove protocols, such as encryption or reliability, at run-time. We demonstrated that our self-aware networking architecture can autonomously adapt its protocol stack to varying link qualities without loosing any packets while reducing the communication overhead (in terms of sent packets) by 28% compared to a static networking architecture.

In future work we will focus on more advanced self-adaptation algorithms and we will apply our architecture to a smart camera network. The platform for the smart cameras will be ReconOS [9], which allows us to execute some parts of the network functionality, such as encryption or compression algorithms, in hardware, while still being able to freely compose and adapt the protocol stack.

6. REFERENCES

[1] G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, and M. May, "The autonomic network architecture (ana)," *Selected Areas in Communications, IEEE Journal on*, vol. 28, no. 1, pp. 4–14, Jan. 2010.

[2] A. Keller, D. Borkmann, and W. Mühlbauer, "Efficient implementation of dynamic protocol stacks (poster)," in *Proc. ACM/IEEE Symp. on Architecture for Networking and Communications Systems (ANCS)*. Washington, DC, USA: IEEE Computer Society, Oct. 2011, pp. 83–84.

[3] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *Computer Communication Review*, vol. 26, pp. 5–18, 1996.

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[5] "netgraph – graph based kernel networking subsystem," (accessed in Sept. 2012). [Online]. Available: http://www.freebsd.org/cgi/man.cgi?query=netgraph\&sektion=4

[6] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Intelligent design enables architectural evolution," in *Proc. of the ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. NY, USA: ACM, 2011, pp. 3:1–3:6.

[7] T. Wolf, J. Griffioen, K. L. Calvert, R. Dutta, G. N. Rouskas, I. Baldine, and A. Nagurney, "Choice as a principle in network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 105–106, Aug. 2012.

[8] "netem," (accessed June 2013). [Online]. Available: http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[9] E. Lübbers and M. Platzner, "Reconos: Multithreaded programming for reconfigurable computers," *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 1, pp. 8:1–8:33, Oct. 2009.

MULTI-OBJECTIVE SELF-OPTIMIZATION OF RECONFIGURABLE DESIGNS WITH MACHINE LEARNING

Maciej Kurek, Tianchi Liu, Wayne Luk *

Department of Computing
Imperial College London
180 Queen's Gate, London SW7 2BZ, England
email: mk306@imperial.ac.uk, tianchi.liu12@imperial.ac.uk, wl@imperial.ac.uk

ABSTRACT

Optimizing reconfigurable designs is a complex task that usually involves manual design analysis and subsequent tweaking. We present a new Multi-Objective Machine Learning Optimizer (MOMLO) which supports self-optimization of reconfigurable designs through automatic analysis and adaptation of design parameters. From a number of benchmark executions, our tool automatically derives the characteristics of the parameter space and creates a surrogate model covering the multiple objectives of the design. The resulting Pareto fronts of possible design configurations can be used for self-optimization at run time. For example, we can switch between a fast but power hungry design and a relatively slow but low power alternative. We evaluate the algorithm using a multi-objective example consisting of power and throughput benchmarks.

1. INTRODUCTION

In previous work [1, 2] we have demonstrated automatic optimization for reconfigurable designs by constructing surrogate models of fitness functions which represent the design quality of parameterized designs. We now extend our work to optimize for multiple competing design aspects such as power efficiency, performance and accuracy. Our new Multi-Objective Machine Learning Optimizer (MOMLO) aims to discover a set of balanced solutions with respect to several objectives and represent them in a Pareto optimal front. A surrogate model of all the design objectives is constructed, which brings substantial savings since its evaluation is orders of magnitude faster than generation of bitstreams and code execution of benchmarks. Our MOMLO approach results in a substantially reduced design effort compared to traditional approaches which require the designer to manually analyze the application, create models and benchmarks, and subsequently optimize the design [3, 4, 5, 6]. Furthermore, we support self-optimization at run time where an optimal design variant can be reconfigured based on dynamically changing operating conditions or environments by repeatedly extracting the most suitable design from the discovered Pareto optimal front. The contributions of this paper are:

- The new MOMLO approach. We show how multiple Bayesian regressors, classifiers and multi-objective meta-heuristics can be interlinked (Section 3).
- An evaluation of the extended MOMLO approach using a case study where a quadrature based financial application with varied precision is optimized for throughput and power consumption (Section 4).

2. BACKGROUND

When developing reconfigurable applications, designers are often confronted with a very large parameter space. As a result parameter space exploration can take an immense amount of time. A number of researchers approach the problem of high-cost fitness functions and large design spaces in various fields by having fitness functions combined with fast-to-compute Gaussian Process (GP) surrogate models for decreasing evaluation time [7, 8, 9, 10, 11]. However most current surrogate models only consist of a regressor and rarely take into account invalid configurations within the design space. Surrogate models, which approximate fitness functions by substituting lengthy evaluations with estimations based on closeness in a design space, have been investigated in reconfigurable computing [12]. The work covers surrogate models for circuit synthesis from higher level languages, rather than parameter optimization. In previous work [1, 2] we have shown that it is useful to construct surrogate models of fitness functions representing the design quality of reconfigurable parameterized designs. The optimization approach we developed replaces the following steps:

1. Build application and a benchmark returning design

*This work is supported by the European Union Seventh Framework Programme under grant agreement number 248976, 257906, 287804 and 318521, by UK EPSRC, by Maxeler University Programme, and by Xilinx.

quality metrics.

2. Specify search space boundaries and optimization goal.

3. Create analytical models for the design.

4. Create tools to explore the parameter space.

5. Use the tools to find optimal configurations, guided by the models in step 3.

6. If result is not satisfactory, redesign.

When using the Machine Learning Optimizer (MLO) the user supplies a benchmark along with constraints and goals, and the MLO automatically carries out the optimization. The approach consists of the following steps:

1. Build application and benchmark returning design quality metrics.

2. Specify search space boundaries and optimization goal.

3. Automatically optimize design with MLO.

4. If result is not satisfactory, redesign or revise time budget and search space.

2.1. Gaussian Process Regression

GP is a machine learning technology based on strict theoretical fundamentals and Bayesian theory [13]. GP does not require a predefined structure; it can approximate arbitrary function landscapes including discontinuities, and includes a theoretical framework for obtaining optimum hyperparameters [10]. An advantage of GP is that it provides a predictive distribution, not a point estimate.

A Gaussian process is a collection of random variables, a finite set of which have a joint Gaussian distribution. A Gaussian process is completely specified by its mean function $m(\mathbf{x})$ and the covariance (kernel) function $k(\mathbf{x}, \mathbf{x}')$. The goal is to compute regression: $\hat{f}(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$

The function $k(\mathbf{x}, \mathbf{x}')$ describes the covariance between pairs of random variables, and in regression analysis it expresses the relation between input-output pairs. This is based on a training set $\mathcal{D}$ of n observations, $\mathcal{D} = (\mathbf{x}_i, y_i)|i = 1, ...n$, where $\mathbf{x}$ denotes an input vector, and y denotes a scalar output. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix X, and the outputs are collected in the vector $\mathbf{y}$. The goal of Bayesian forecasting is to compute the distribution $p(\hat{f}|\mathbf{x}_*, \mathbf{y}, X)$ of the function $\hat{f}$ at unseen input $\mathbf{x}_*$ given a set of training points $\mathcal{D}$. Using Bayes rule, the predictive posterior for the Gaussian process $\hat{f}$ and the predicted scalar outputs $\hat{f}(\mathbf{x}_*) = y_*$ can be obtained.

2.2. Support Vector Machines Classification

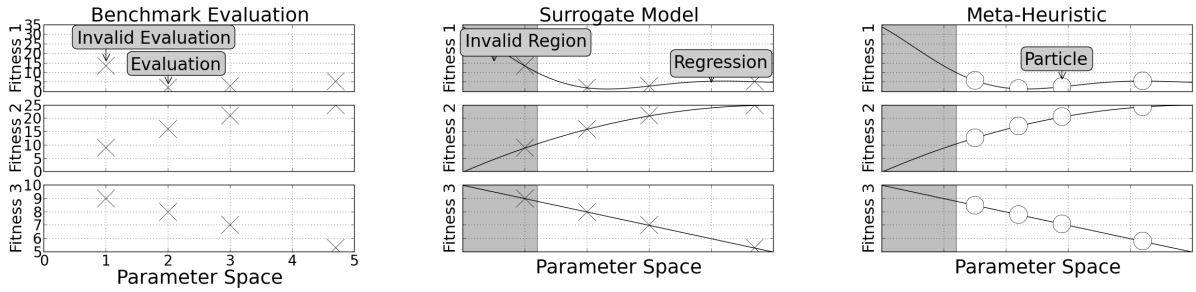
Support Vector Machine (SVM) is a maximum margin classifier, which constructs a hyperplane used for classification (or regression) [14]. SVMs use kernel functions $k(\mathbf{x}, \mathbf{x}')$ to transform the original feature space to a different space with a linear model used for classification. SVMs are a class of decision machines and do not provide posterior probabilities. There is a training set $\mathcal{D}$ of n observations, $\mathcal{D} = (\mathbf{x}_i, t_i)|i = 1, ...n$, where $\mathbf{x}$ denotes an input vector, t denotes a target value. The column vector inputs for all n cases are aggregated in the $D \times n$ design matrix X, and the targets in the vector $\mathbf{t}$. The goal is to classify an unseen input $\mathbf{x}_*$ based on X and $\mathbf{t}$ by computing a decision boundary.

2.3. PSO

Particle Swarm Optimization (PSO) is a population-based meta-heuristic based on the simulation of the social behavior of birds within a flock [15]. The algorithm starts by randomly initializing N particles where each individual is a point in the $\mathcal{X} = \mathbb{R} \times ... \times \mathbb{R}$ search space. The population is updated in an iterative manner, with each particle displaced based on its velocity v_{id}. The criteria for termination of the PSO algorithm can vary, and usually are determined by a time budget. The variable x_{id} represents the dth coordinate of particle i from the set X_* of N particles, where each particle is a point within $\mathcal{X}$. Multi-objective optimization is usually approached by finding a Pareto optimal set of the underlying fitness functions. The original PSO algorithm was designed to cope with single-objective optimization problems, multiple different flavors have been developed to cope with multi-objective optimization [16, 17]. Many more sophisticated multi-objective meta-heuristic algorithms have been developed [18]. The designer has to assess his requirements in terms of performance and robustness when deciding which algorithm to use. In such problems, the objectives to be optimized are normally in conflict with respect to each other, which indicates that there is no single solution for all of these problems. Instead, we aim to find "trade-off" solutions that achieve the best possible compromise among the objectives. In other words, we wish to find the Pareto optimal set $\mathcal{P}*$ which is an approximation of the Pareto Front $\mathcal{PF}*$ [19].

3. OPTIMIZATION APPROACH

The optimization approach of MOMLO is inspired by that of MLO. The idea of multi-objective surrogate modeling is illustrated in Fig. 1. The MOMLO algorithm explores the parameter space by evaluating different benchmark configurations as presented in Fig. 1a. Fig. 1b shows the results obtained during evaluations are used to build surrogate model which provides regressions of the fitness function multiple-metrics and identifies invalid regions of the parameter space.



(a) (b) (c)

Fig. 1: Benchmark evaluations, surrogate model and model guided search for a problem with three conflicting objectives.

A multi-objective PSO guides the exploration of the parameter space using the surrogate model, as shown in Fig. 1c. The main novelty is that the result of optimization is a Pareto optimal set of designs $\mathcal{P}*$, allowing the design to self-adapt when circumstances change.

3.1. Fitness Function

The parameter space $\mathcal{X}$ of a reconfigurable design is spanned by discrete and continuous parameters determining both the architecture and physical settings of Field programmable gate array (FPGA) designs. Given a parameter setting $\mathbf{x} \in \mathcal{X}$, a benchmark consists of a vector of fitness metric $[y_1, y_2, .., y_i] = \mathbf{y}$ and t, the exit code of the application. A function $b_i(\mathbf{x}) = y_i$ represents one of the K objectives. Execution time and power consumption are examples of possible objectives. Vector $\mathbf{y}$ consists of multiple fitness measures when the designer wants to find an optimal design defined in terms of a number of qualities. For example the $\mathbf{y}$ vector could constitute of execution time and power usage, if the aim is to find the set of power efficient designs. There are many possible exit codes t, with 0 indicating valid $\mathbf{x}$'s. The designer can choose to extend the benchmark to return additional exit codes depending on the failure cause, such as configurations producing inaccurate results or failing to build.

We distinguish three different types of exit codes. The first type is exit code 0 indicating a valid design. The second type of exit codes indicate configurations that produce results yet fail at least one constraint making them undesirable. The third type of exit codes are used for configurations that fail to produce any results. The region of $\mathcal{X}$ that defines configurations $\mathbf{x}$ that produce $\mathbf{y}$ and satisfy all constraints is defined as valid region $\mathcal{V}$, regions with designs failing at least one constraint yet producing $\mathbf{y}$ are part of failed region $\mathcal{F}$, and the region with designs failing to produce $\mathbf{y}$ is the invalid region $\mathcal{I}$. If $\mathbf{x}_*$ does not produce a valid result, we assign a value that the designer assumes to be the most disadvantageous. Depending on whether we face a minimization or a maximization problem for a given objective function f_i either a ∞ or $-\infty$ value will be assigned as presented in Eq. 1.

$$f_i(\mathbf{x}) = \begin{cases} y_i & \mathbf{x} \in \mathcal{V} \\ \pm\infty & otherwise \end{cases} \tag{1}$$

3.2. The MOMLO Algorithm

We integrate a GP regressor $\hat{f}$ and an SVM classifier to create a novel surrogate model of fitness function f. As illustrated in Fig. 1, the problem we face is regression of f over $\mathcal{V}$ and $\mathcal{F}$ as well as classification of $\mathcal{X}$. We make use of GP to access the standard deviation estimate $\sigma(\mathbf{x}_*)$ of non-examined parameter configurations $\mathbf{x}_*$. We use SVMs to predict exit codes of X_* across $\mathcal{X}$. Regression $\hat{f}_i$ for a function f_i is created using the training set obtained from benchmark execution $\mathcal{D}_{Ri}$, while classification is done using the training set $\mathcal{D}_C$. We invoke regressions $\hat{f}_i(\mathbf{x}_*)$ for every particle in $\mathbf{X}_*$ and for every function f_i and aggregate the results to obtain the regression $[\hat{f}_1(\mathbf{x}_*), \hat{f}_2(\mathbf{x}_*), .., \hat{f}_n(\mathbf{x})] = \hat{f}(\mathbf{x}_*) = \mathbf{y}_*$ and its uncertainty vector $[\sigma_1(\mathbf{x}_*), \sigma_2(\mathbf{x}_*), .., \sigma_n(\mathbf{x})] = \sigma_*(\mathbf{x})$, which is the standard deviation estimate. Exit code t_* of particle $\mathbf{x}_*$ is predicted by the classifier.

In our MOMLO algorithm, we adopt 1) density measure [20] (indicates the closeness of the particles within the swarm) as the criterion to choose the leader for particles, i.e. guide the population to spread out along real Pareto frontier as fast as possible; 2) "ϵ-dominance" method [21] to retain a non-dominated solution to the Pareto Front, which is believed to be able to generate well-formed Pareto optimal set as well as to generate the front evaluating fewer fitness functions. We present the MOMLO approach in Algorithm 1. The algorithm includes a classifier to account for invalid regions of $\mathcal{X}$. We initialize the meta-heuristic of our choice with N particles X_* randomly distributed across the parameter space. Each particle has an associated fitness $\mathbf{x}.fit$ and a position $\mathbf{x}$. For all $\mathbf{x}_*$ predicted to lie in $\mathcal{V}$ we proceed as follows: whenever $\sigma_{max}(\mathbf{x}_*)$, the largest value out of all σ_i, returned by the GP is below a credible interval min_σ we use the prediction y_*; otherwise we assume the prediction to

Algorithm 1 MOMLO

for $\mathbf{x}_* \in X_*$ **do**
  $\mathbf{x}_*.\mathbf{fit} \leftarrow f(\mathbf{x}_*)$ ▷ Initialize with a uniformly randomized set for every objectives f_i* in the fitness function.
end for
repeat
  for $\mathbf{x}_* \in X_*$ **do**
    if $\sigma_{max}(\mathbf{x}_*) < min_\sigma$ and $t_* = 0$ **then**
      $\mathbf{x}_*.\mathbf{fit} \leftarrow \mathbf{y}_*$
    else
      if $t_* = 0$ **then**
        $\mathbf{x}_*.\mathbf{fit} \leftarrow f(\mathbf{x}_*)$
      else
        for $i \in 1, 2, ..., K$ **do** ▷ Depending on the objective of each of the fitness function either ∞ or $-\infty$ is assigned
          $\mathbf{x}_*.fit_i \leftarrow \pm\infty$
        end for
      end if
    end if
  end for
  $X_* \leftarrow Meta(X_*)$ ▷ Iteration of the meta-heuristic
until Termination Criteria Satisfied

be inaccurate and evaluate $f(\mathbf{x}_*)$. This step is required and happens in a situation when at least one of the underlying f_i functions is not modeled accurately. Although individual $f_i(\mathbf{x}_*)$ could be evaluated, usually the cost of evaluation of a single f_i is marginally smaller than the cost of evaluation of f. Based on our experience values within the range of 0.01 and 0.1 are the most practical for min_σ. Larger credible interval will usually hinder MOMLO performance due to high admissible uncertainty which is especially problematic when the mean estimate is relatively small. The meta-heuristic will avoid $\mathcal{I}$ and $\mathcal{F}$ regions as they are both assigned unfavorable $\pm\infty$ values. Whenever $f(\mathbf{x})$ is evaluated, $(\mathbf{x}, t)$ is included within the classifier training set $\mathcal{D}_C$. If the exit code is valid $(t = 0)$, then $(\mathbf{x}, y_i)$ is added to $\mathcal{D}_{Ri}$.

4. EVALUATION

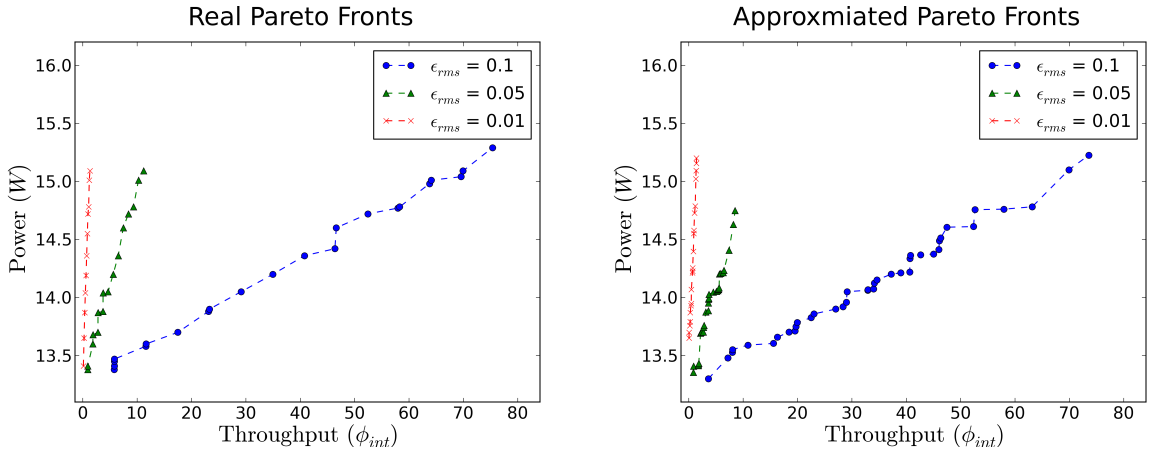
In [4] the designer explores trade-off between accuracy and throughput in a quadrature-based financial application with three parameters. The first two parameters are mantissa width m_w of the floating point operators and the number of computational cores $cores$. Having more m_w bits increases computation accuracy, but limits the maximum number of $cores$ that can be implemented on the chip due to the increased size of the individual core. The third parameter is the density factor d_f which is inversely proportional to the integration grid spacing. It is an application parameter and is independent of the FPGA device used. The density factor d_f increases computation time per integration while improving the accuracy of the results due to having a finer integration grid.

The optimization goal is to find the design offering the highest throughput of integrations per second ϕ_{int} (f_1) and the lowest power consumption W (f_2) given a required minimum accuracy defined in terms of root mean square error ϵ_{rms}. The error is defined with respect to results obtained by calculating a set of reference integrals at the highest possible precision. MOMLO terminates when the globally optimal configuration for a given ϵ_{rms} is found. The $\mathcal{F}$ region contains the inaccurate result class. The design space $\mathcal{X}$ is defined as $m_w \times cores \times d_f$: $\{11-53\} \times \{1-16\} \times \{4-32\}$. We repeat the experiment for different error limits ϵ_{rms} 10 times; we find that in order for the approximate front to cover the real Pareto front we require around 158 ($\epsilon_{rms} = 0.1$), 116 ($\epsilon_{rms} = 0.05$) and 91 ($\epsilon_{rms} = 0.01$) fitness function evaluations. By coverage we understand that around 30% of designs within the approximate front will reside on the real front and around 35% will match it within a 5% performance limit. The approximate front includes more designs, and 50% of the designs from the real front reside within it. The rest of the designs have a higher discrepancy (around 10%) due to surrogate model inaccuracies. The coverage can be improved by increasing the number of fitness evaluations.

When comparing MOMLO to the single-objective MLO [2] the increase of the number of required fitness function evaluations to reach termination criteria is noticeable and dependant on the size of valid area. The increase in fitness function evaluations is 15% (0.1), 73% (0.05) and 94% (0.01) for the evaluated error limits. Longer optimization time of multi-objective problems is expected since the problems complexity increases with respect to single-objective optimization. Although the overhead can be significant, it seems to decrease as the size of valid area increases (increased ϵ_{rms}). As presented in [2] the manual optimization procedure requires 420 fitness function evaluations to find an optimal design for a given ϵ_{rms}. In best case scenario the number would not be increased for multiple objectives meaning MOMLO would still offer superior performance. The drawback of MOMLO is the lack of guaranty of finding the true Pareto optimal front.

5. CONCLUSIONS AND FUTURE WORK

Our MOMLO approach can be used to create a self-adaptive system which can constantly improve its knowledge of the design's Pareto optimal configuration set, and switch between design configurations depending on the current environment. The algorithm shows much promise, however its capability and scope require further investigation. We are preparing a number of new multi-objective evaluation cases which should help us to assess MOMLO's robustness and performance. Furthermore we are investigating a distributed version of the algorithm, enabling a parallel approach and hence faster optimization when the compute resources are available. This allows for an optimization approach where the algorithm self-adapts the optimization strategy to balance its search speed and efficiency.



(a) (b)

Fig. 2: The Real and Approximated Pareto Fronts for different ϵ_{rms} limits.

6. REFERENCES

[1] M. Kurek and W. Luk, "Parametric reconfigurable designs with machine learning optimizer," in *FPT*, 2012, pp. 109–112.

[2] M. Kurek, T. Becker, and W. Luk, "Parametric optimization of reconfigurable designs using machine learning," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. LNCS 7806. Springer, 2013, pp. 134–145.

[3] X. Niu et al., "Exploiting run-time reconfiguration in stencil computation," in *FPL 2012*, 2012, pp. 173–180.

[4] A. H. T. Tse et al., "Optimising performance of quadrature methods with reduced precision," in *ARC*, ser. LNCS 7199. Springer, 2012, pp. 251–263.

[5] T. Becker, W. Luk, and P. Y. Cheung, "Parametric design for reconfigurable software-defined radio," in *ARC*. Springer, 2009, pp. 15–26.

[6] Q. Jin et al., "Optimising explicit finite difference option pricing for dynamic constant reconfiguration," in *FPL*, 2012, pp. 165–172.

[7] A. I. Forrester and D. R. Jones, "Global optimization of deceptive functions with sparse sampling," in *AIAA/ISSMO*. American Institute of Aeronautics and Astronautics, September 2008.

[8] Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 481–494, 2002.

[9] Y. S. Ong, P. B. Nair, and A. J. Keane, "Evolutionary optimization of computationally expensive problems via surrogate modeling," *AIAA*, vol. 41, no. 4, pp. 689–696, 2003.

[10] S. Guoshao and J. Quan, "A cooperative optimization algorithm based on gaussian process and particle swarm optimization for optimizing expensive problems," in *CSO*, vol. 2, 2009, pp. 929–933.

[11] H.A.L. Thi, D.T. Pham, and N.V. Thoai, "Combination between global and local methods for solving an optimization problem over the efficient set," *EJOR*, vol. 142, no. 2, pp. 258–270, 2002.

[12] C. Pilato, A. Tumeo, G. Palermo, F. Ferrandi, P. L. Lanzi, and D. Sciuto, "Improving evolutionary exploration to area-time optimization of FPGA designs," *J. Syst. Archit.*, vol. 54, no. 11, pp. 1046–1057, 2008.

[13] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer-Verlag, 2006.

[15] F. Van Den Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, University of Pretoria, South Africa, 2002.

[16] X. Hu and R. Eberhart, "Multiobjective optimization using dynamic neighborhood particle swarm optimization," in *IEEE CEC*, vol. 2, 2002, pp. 1677–1681.

[17] J. Liang, B. Qu, P. Suganthan, and B. Niu, "Dynamic multi-swarm particle swarm optimization for multi-objective optimization problems," in *IEEE CEC*, 2012, pp. 1–8.

[18] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, Jun. 2000.

[19] M. Reyes-Sierra and C. C. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *International Journal of Computational Intelligence Research*, vol. 2, pp. 287–308, 2006.

[20] K. Deb et al., "A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2000.

[21] M. Laumanns et al., "Combining convergence and diversity in evolutionary multiobjective optimization," *Evol. Comput.*, vol. 10, no. 3, pp. 263–282, Sep. 2002.

DECOMPOSING RUN-TIME RESOURCE MANAGEMENT IN HETEROGENEOUS RECONFIGURABLE SYSTEMS

Stefan Wildermann, Jürgen Teich

Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
{stefan.wildermann, teich}@fau.de

ABSTRACT

Mixed workload and multi-application scenarios characterize modern and future reconfigurable systems. On the one hand, such systems consist of applications with objectives which may dynamically change during different execution phases. On the other hand, the designer or user of the system specifies requirements, e.g., regarding its power consumption, which have to be filled at any time. The main challenge is to partition the resources of the heterogeneous hardware architecture between the applications such that their objectives are optimized while fulfilling the system requirements. In this context, dynamic application objectives and system requirements can only be handled by providing self-adaptive resource management at run-time.

This paper discusses a distributed approach to resource management which is derived by applying Lagrangian dual decomposition. Each application is only aware of its own objectives and determines the desired amount of resources solely based on local information. The proposed mechanism steers the decisions of applications to be in compliance with the objectives and requirements of the overall system. We show that this approach achieves results which are competitive, and in many cases even significantly better than the results of a centralized heuristic used as state-of-the-art for resource management in reconfigurable systems while having the advantages of a distributed approach.

1. INTRODUCTION

Driven by the constant increase of the clock frequency, the functionality and usage scenarios of embedded systems have grown more and more complex and dynamic over the past years. Since 2005 however, semiconductors are increasingly confronted with physical issues concerning heat, power consumption, and leakage problems so that the increase in clock frequency of the computational resources has stagnated. Heterogeneous and highly parallel hardware architectures have emerged as a consequence. Obtaining further performance gains on these architectures requires that programs exploit the available parallelism. At the same time, the architectures have to provide reconfigurability so that the resources can be shared optimally between applications for varying workloads and other dynamic usage scenarios.

In this context, resource allocation is an optimization problem with further objectives in addition to the performance increase of individual applications. For instance, mobile systems usually need an allocation of computational resources with the requirement of the power consumption staying within a power budget. This means that it is required to select an implementation for each application such that their objectives are optimized while fulfilling such system requirements.

In this paper, we provide a mechanism for distributed resource allocation based on a round based negotiation scheme. We derive this negotiation scheme from the formulation of the original optimization problem by applying Lagrangian dual decomposition. This results in a mechanism where self-aware applications optimize their resource requirements based on local information only, while a master steers their decisions to be in compliance with the objectives and requirements of the overall system. One interesting aspect of this approach is the decoupling of master and applications since they do not require any local information of each other. This self-awareness and separation of concerns provides the scalability and flexibility required for dynamic usage scenarios.

2. RELATED WORK

Resource allocation for dynamic embedded systems is often tackled by design-time methodologies in the form of *scenario-based design*, e.g., [1], or *multi-mode system synthesis*, e.g., [2], as it is possible to apply powerful verification and optimization techniques to generate feasible and highly optimized implementations. Nonetheless, near-future embedded systems cannot be fully predicted at design-time due to dynamic usage scenarios and unexpected unavailability of hardware resources because of aging, reliability, or temperature effects.

The only way for being able to deal with this is to provide the system with a run-time resource management (RRM) layer which dispatches the reconfigurable resources to the

applications. *Centralized RRMs* collect all information relevant for calculating an optimized resource allocation. However, they have to deal with scalability and reliability issues as they form a single point of failure and produce communication and computation hot-spots and bottlenecks at the processors running the RRM. Even security issues can arise when an application has to reveal all internal information, and thus introduce the possibility of side-channel attacks. Consequently, several *decentralized RRM* approaches have been proposed, which are often based on multi-agent systems in the embedded domain [3, 4]. Here, also mechanisms from distributed computing (grid computing, cloud computing) could be adopted, which are mostly provided through auctions. In this paper, we present a formal approach to establish a distributed resource allocation approach where we apply techniques from convex optimization to incorporate self-awareness into the RRM.

3. DECOMPOSITION OF THE RUN-TIME RESOURCE MANAGEMENT

An *implementation* of an application i can be characterized by a vector x_i. It contains the implementation's quality numbers regarding application and system objectives, as well as the amount of all resource types required to run this implementation on the heterogeneous system. The set of all possible implementations is denoted by D_i. This turns out to be a Pareto front, only containing implementations which are non-dominated regarding the objectives and resource requirements. Figure 1 illustrates an example. There is of course the question of how to determine D_i, and we observe two major directions in the related work. The first one is to determine the non-dominated implementations D_i at design-time by performing design space exploration (DSE) and applying profiling techniques [5, 6]. The second one is to perform adaptive auto-tuning which uses parametrized code variants [7] for being able to generate various implementations at run-time. They can then be evaluated by monitoring the values of the objectives during their execution.

An application has different *utilities* for running in one of the implementations, which is expressed by utility function $f_i : D_i \to \mathbb{R}$. This utility may depend on the current execution phases of the application. The purpose of the utility function is to assign each implementation with a scalar value, which defines a total order over all elements in D_i. This is necessary for being able to make decisions at run-time. Such functions are commonly achieved by performing a *scalarization* of all relevant application objectives, e.g., [5, 6].

3.1. Resource Allocation Problem

Resource allocation in heterogeneous reconfigurable systems can be formulated as a combinatorial problem with the goal

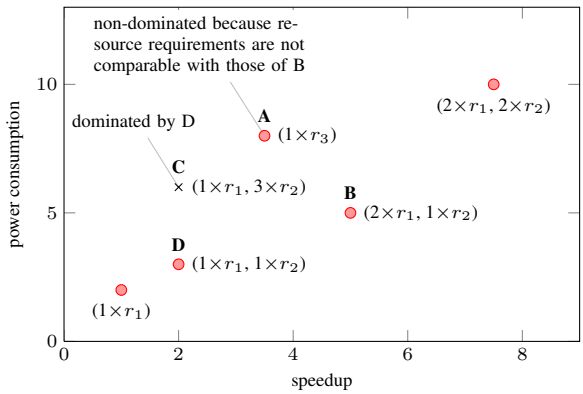


Fig. 1. Example of implementations depicted as a Pareto front for objectives speedup, power consumption, and usage of three resource types (r_1, r_2, r_3). Each point is annotated with the amount of required resources of each type.

of selecting implementations of all applications which maximize their utilities while adhering to the system constraints. Constraints originate from restricted physical and abstract resources (e.g., available amount of computational resources of a specific resource type or restricted power budgets). The upper bound of a constraint j is specified by $\overline{r}_j$, and the amount required by an implementation x_i is given by $r_j(x_i)$.

Whenever an application switches its execution phase or the system environment changes, the system should be reconfigured to optimally utilize the available resources. This problem can be formalized acc. to the following definition.

Definition 1. *Resource allocation problem*

$$\text{maximize} \quad \sum_{i=1}^{n} f_i(x_i) \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} r_j(x_i) \leq \overline{r}_j, \quad j = 1, ..., m \tag{2}$$

Eq. (1) *represents the objective to maximize the average utility of all applications*[1]*, and Eq.* (2) *the* m *constraints.*

3.2. Decomposing the Resource Allocation Problem

Generally, the optimization problem formulated in Def. 1 is NP-hard. In this section, we therefore propose an approach which is based on solving the Lagrangian dual optimization problem. The main benefits are that this results in a convex optimization problem, which can be solved much more efficiently than the primal problem, and that it can be decomposed to enable a distributed solution method. The mathematical background applied in this section is, e.g., summarized in [8].

The *Lagrangian* of the resource allocation problem is

$$\mathcal{L}(x, \lambda) = -\left(\sum_{i=1}^{n} f_i(x_i)\right) + \sum_{j=1}^{m} \lambda_j \left(\sum_{i=1}^{n} r_j(x_i) - \overline{r}_j\right) \tag{3}$$

[1]The constant normalizing factor $1/n$ can be omitted.

where λ_j is the *Lagrangian multiplier* associated with the j-th constraint.

The *Langange dual function* is then defined as

$$\begin{aligned} g(\lambda) &= \inf_x \mathcal{L}(x,\lambda) = \\ &= \sum_{i=1}^{n} \inf_{x_i} \left(-f_i(x_i) + \sum_{j=1}^{m} \lambda_j \cdot r_j(x_i) \right) - \\ &\quad - \sum_{j=1}^{m} \lambda_j \cdot \overline{r}_j. \end{aligned} \tag{4}$$

The interesting aspects of the dual function are twofold. First, the optimal implementation x_i for given multipliers $\lambda = (\lambda_1, ..., \lambda_m)$ can be calculated by application i independent of other applications. Second, $g(\lambda)$ is concave and continuous in λ even when the primal objective function $\sum_{i=1}^{n} f_i(x_i)$ is not.

Now, the *Lagrange dual optimization problem* is given as

$$\begin{aligned} &\text{maximize} && g(\lambda) \\ &\text{subject to} && \lambda \geq 0, \end{aligned} \tag{5}$$

which, due to the nature of $g(\lambda)$, is a convex optimization problem. As such, it is possible to apply standard methods to determine the optimal value for λ. An algorithm based on the *subgradient method* [8] is summarized in Algorithm 1. All application subproblems can be solved independently, and the master problem of maximizing the dual function is solved by applying the subgradient method for all Lagrange multipliers.

Algorithm 1: Algorithm for solving the dual optimization problem.

```
while !stopping criterion do
    // application subproblems
    for each i = 1,...n do
        Find x_i that minimizes
        ( -f_i(x_i) + sum_{j=1}^{m} λ_j · f_j(x_i) );
    // master problem
    for each j = 1,...m do
        // Calculate subgradient of λ_j
        Δ_j = r̄_j - sum_{i=1}^{n} r_j(x_i);
        // Apply update rule acc. to
           subgradient method
        λ_j = max{0, λ_j - α_{t,j} · Δ_j} ;
```

The algorithm proposes a negotiation scheme, as illustrated in Figure 2. The Lagrange multipliers can be interpreted as the *price* of the respective resource (cf. [9]): The master tries to maximize the *costs*, while each application determines how much resources it wants to *buy* to maximize its *asset* for the current price. The big advantage is the self-awareness inherent in the algorithm: For no component is it necessary to have any internal details about another component.

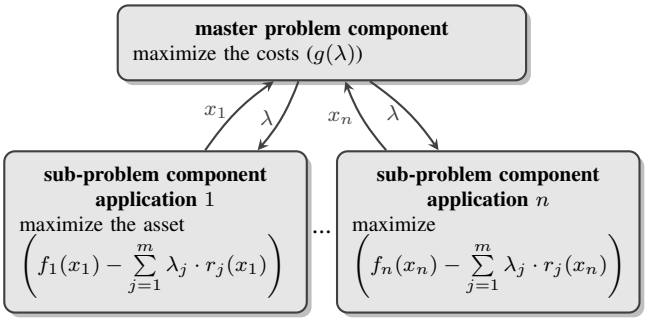


Fig. 2. Schematic illustration of resource allocation based on dual decomposition.

A disadvantage is that due to the Lagrangian relaxation in Eq. (3) the optimum is only approximated. This induces that there might be a gap between the optimal value f^* of the original problem and the optimal value g^* of the dual problem, so that $(-f^*) - g^* \geq 0$ could be non-zero. As a consequence, the negotiated outcome may not be achievable or feasible. We therefore propose the following heuristic. Applications can claim resources, e.g., by using mechanisms known from *invasive computing* [10], which enables the exclusive reservation of resources. Whenever resource conflicts arise during this embedding, applications are prioritized to resolve these conflicts. We choose the priority to be proportional to

$$f_i(x_i) - \sum_{j=1}^{m} \lambda_j \cdot r_j(x_i), \tag{6}$$

which represents the negotiated *asset* of application i. Applications which do not find sufficient resources for their negotiated implementation x_i choose this implementation x_i' that can be feasibly implemented on the remaining resources with maximal asset acc. to Eq. (6).

4. EXPERIMENTS

In this section, we present the results of our approach. In one version, the negotiation is performed for 20 rounds (*rrm20*) and in the other version for 100 rounds (*rrm100*) before the applications are embedded. We compare our approach to the knapsack heuristic from [5] (*knap*). For the first experiments, we generated test cases from the *e3s benchmark* [11]. It contains five applications. The architecture contains three different resource types. For each application, we generated the Pareto sets D_i by performing a DSE using the Opt4J

Table 1. Results for e3s benchmark case study.

α	approach	speedup	power[W]	utility $\sum_i f_i(x_i)$
0	*knap* [5]	3.21	8.25	**-8.25**
	rrm20	5.00	8.80	-8.80
	rrm100	5.00	8.80	-8.80
0.5	*knap* [5]	3.59	6.59	**-1.50**
	rrm20	5.00	8.80	-1.90
	rrm100	5.00	8.80	-1.90
0.75	*knap* [5]	24.14	40.07	8.09
	rrm20	27.07	45.44	**8.94**
	rrm100	26.85	45.08	8.86
1.0	*knap* [5]	27.92	57.67	27.92
	rrm20	31.20	65.39	31.20
	rrm100	31.48	63.66	**31.48**

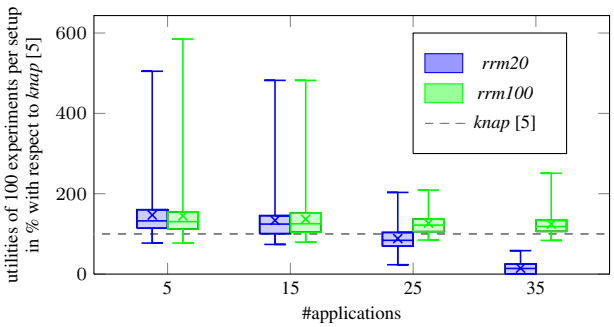


Fig. 3. Boxplots of the results for synthetic test cases relative to the results of *knap* (indicated by dashed line) for 100 experiments per setup.

framework [12] and optimized for resource usage, speedup (compared to the implementation with highest latency) and power consumption. The utility function is chosen according to $f_i(x_i) = \alpha \cdot speedup(x_i) - (1-\alpha) \cdot power(x_i)$, where α is a weight for scalarizing the two objectives to maximize the speedup and to minimize the power consumption. Results for different values of α are shown in Table 1. In all cases, the results are close together, showing that the proposed distributed approach is competitive with a fully centralized heuristic.

We furthermore performed test runs on synthetic test cases with 5, 15, 25, and 35 applications, and an architecture consisting of four resource types. For each such setup, we generated 100 cases and evaluated them. Fig. 3 shows the boxplots of the results for each setup, where the results of *knap* serve as baseline and all other results are given relative to this in percent. The results show that the number of negotiation rounds has to increase with the number of applications as *rrm20* degradates with the number of applications. In case of *rrm100* however, the proposed approach even performs better in a significant amount of experiments.

5. CONCLUSION

This paper demonstrates the application of formal mechanisms to incorporate self-awareness into run-time resource management (RRM) for embedded reconfigurable systems. We have presented the use of Lagrangian relaxation and dual decomposition. But also other techniques, such as game theory [13], are promising mathematical tools to perform this task. We discussed several advantages of distributed and self-aware approaches for RRM compared to centralized heuristics. Nonetheless, they usually perform better than distributed approaches as all details are available. However, the experiments have shown that the proposed distributed approach is competitive and in many cases even significantly better than a state-of-the-art centralized heuristic.

6. REFERENCES

[1] P. van Stralen and A. Pimentel, "Scenario-based design space exploration of MPSoCs," in *Proceedings of ICCD*, oct. 2010, pp. 305 –312.

[2] S. Wildermann, F. Reimann, D. Ziener, and J. Teich, "Symbolic design space exploration for multi-mode reconfigurable systems," in *Proceedings of CODES+ISSS*, 2011, pp. 129–138.

[3] M. Al Faruque *et al.*, "ADAM: Run-time agent-based distributed application mapping for on-chip communication," in *Proceedings of Design Automation Conference (DAC)*, 2008, pp. 760–765.

[4] S. Kobbe *et al.*, "DistRM: distributed resource management for on-chip many-core systems," in *Proceedings of CODES+ISSS*, 2011, pp. 119–128.

[5] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal, "Fast multi-dimension multi-choice knapsack heuristic for mp-soc run-time management," in *Proc. of SOC*, nov. 2006, pp. 1 –4.

[6] G. Marianik *et al.*, "Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures," in *Proc. of DATE*, 2012, pp. 1379–1384.

[7] Y. Li, J. Dongarra, and S. Tomov, "A note on auto-tuning GEMM for GPUs," in *Proc. of ICCS*. Springer-Verlag, 2009, pp. 884–892.

[8] D. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE JSAC*, vol. 24, no. 8, pp. 1439 –1451, aug. 2006.

[9] S. Boyd and L. Vandenberghe, *Convex optimization*. New York, NY, USA: Cambridge University Press, 2004.

[10] J. Teich *et al.*, "Invasive computing: An overview," in *Multiprocessor System-on-Chip – Hardware Design and Tool Integration*, M. Hübner and J. Becker, Eds. Springer, Berlin, Heidelberg, 2011, pp. 241–268.

[11] R. Dick, "Embedded system synthesis benchmarks suite," 2010, http://ziyang.eecs.umich.edu/dickrp/e3s/.

[12] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J – a modular framework for meta-heuristic optimization," in *Proc. of GECCO*, Dublin, Ireland, 2011, pp. 1723–1730.

[13] S. Wildermann, T. Ziermann, and J. Teich, "Game-theoretic analysis of decentralized core allocation schemes on many-core systems," in *Proceedings of DATE*, 2013, pp. 1498–1503.

DYNAMICALLY SHIFTED SCRUBBING FOR FAST FPGA REPAIR

Leonardo P. Santos, Gabriel L. Nazar and Luigi Carro

Instituto de Informática
Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre, RS - Brazil
pereira.santos@ufrgs.br, {glnazar, carro}@inf.ufrgs.br

ABSTRACT

Field Programmable Gate Arrays (FPGAs) are very successful platforms that rely on large configuration memories to store the circuit functions required by users. Faults affecting such memories are a major dependability threat for these devices, and the applicability of FPGAs on critical systems depends on efficient means to mitigate their effects. The usual means to effectively remove such faults, namely configuration scrubbing, consists in rewriting the desired contents of the configuration memory. The scrubbing process suffers from high power consumption and a long mean time to repair (MTTR). In this work we propose a novel approach to enable self-diagnosed circuits that, by being aware of their own disposition on the FPGA fabric are able to greatly reduce the MTTR.

1. INTRODUCTION

SRAM-based FPGA play an important role in self-aware systems by adding several attractive characteristics to logic designers: flexibility, high density and high pin count. However, these devices suffer reliability problems caused by Single Event Upsets (SEUs). SEUs in SRAM-based FPGAs are especially dangerous; because flipped bits in a configuration cell might change the device's programmed functionality, creating a persistent error.

Redundancy techniques as Dual Module Redundancy (DMR) and Triple Module Redundancy (TMR) can be used to hide the effects of SEUs, thus enabling the use of SRAM-based FPGAs in critical applications. The use of redundancy comes at a price; as the respective area overheads for DMR and TMR are 100 % and 200 % at least, redundancy also adds to power consumption. As redundancy works by detecting and/or masking the errors, it is possible to accumulate enough SEUs to overwhelm it and cause a failure.

Self-awareness is explored in this work through a system that detects and repairs faults on itself before they become functional failures. Currently, the standard way to achieve this in a SRAM-based FPGA is to use partial reconfiguration [1], [2], to re-write the configuration memory before the chosen redundancy is overwhelmed. This is called scrubbing and is usually accomplished by periodically re-writing the device's configuration memory from start to end. The periodicity is calculated based on a statistical estimate of the SEU rate per time unit on the device's operating environment. This means that a higher than anticipated SEU rate can leave a circuit with a configuration error. Also, scrubbing is not instantaneous, as the configuration bit streams sizes for modern devices are on the order of several megabits [3]; event the fastest configuration interface can pose unacceptable delays. The time required to fix an error with the scrubbing process is called Mean Time To Repair (MTTR).

Due to FPGAs' required flexibility, most configuration bits do not have an effect on the circuit, even for applications that use most of a device. This is due to most of them being routing configuration bits or bits that control unused resources. So SEUs on these idle configuration bits have no practical effects. The work in [4] exploits this fact to discover areas with high concentration of bits that affect the implemented circuit and then to choose an optimum frame start position for the scrubbing process, minimizing the MTTR. In this work we extend this concept to improve the gain in MTTR obtained with a fine-grained error detection technique, which provides enhanced diagnosis. Thus, the FPGA circuits are aware of their own placement on the reconfigurable fabric and of the relation between configuration bits and error detection signals. By not having a single start position, but instead a dynamic one based on fine-grained diagnosis, significant improvements are attainable.

The remainder of this paper is organized as follows: in section 2 we discuss related works. Section 3 presents the proposed technique. The validation and measurement setup is explained in section 4, while section 5 contains the results and their discussion. We close this paper with the conclusions in section 6.

2. RELATED WORK

The opportunities provided from coupling error detection techniques and partial reconfiguration have been explored in the past a mean to provide high availability in SRAM-

This work is sponsored by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq)

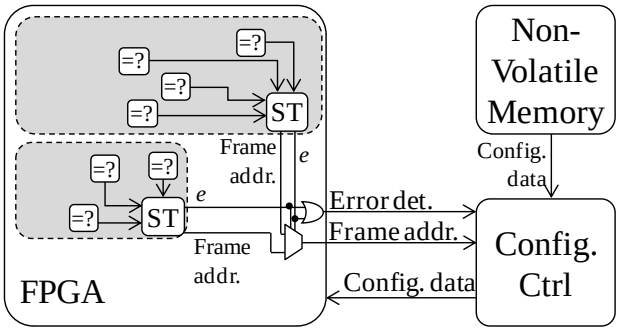


Fig. 1. Embedded detectors and translators in a circuit

based FPGAs. By using configuration readback and a per-frame CRC [5], it is possible to have a high precision on which frame should be corrected; but there's still need for a time-consuming readback and thus a high correction latency. Other works like [6] rely on automatically exploring the design space, using DMR and TMR to meet reliability constrains while minimizing area and repair time. This exploration tests different partitioning schemes and granularities, with different trade-offs between correction latency and area overhead.

Fine-grained DMR is also used in [7], with a focus in softcore processors. The authors propose using precompiled bit streams to bypass faulty components, while halting the processor to avoid corrupting its current state and memory. As is discussed in [6], the extra precision afforded by finer-grain techniques create a greater area and power overhead. One way to mitigate these overheads is offered in [8]. The use of hardwired resources, in this case the carry chains of each slice, hides some of the costs, as this chain is part of the device itself and underused in many situations.

3. DYNAMICALLY SHIFTED SCRUBBING

Because in [8] there is approximately one error detection bit for each of the device's slices, we can create the concept of an "error signature" that is formed by the concatenation of all error bits. These error signatures provide a more precise diagnosis information that thus can be used to guide a local repair procedure, provided the system is aware of its own signature-to-frame relations. The concept explored in this work is that a scrubbing procedure does not necessarily have to start at the first frame of the partition, as proposed in [4]. That work makes use of a previous error analysis to choose a single starting position for the scrubbing process, thus requiring only a very simple error detection scheme (primary output voters, watchdog). In this work, we make use of error signatures generated by a fine-grain error detection technique to dynamically guide the choice of the optimum starting frame, instead of relying on a statically chosen address. We aim for the ideal situation of having a fine-grained technique embedded on the final circuit and of using the

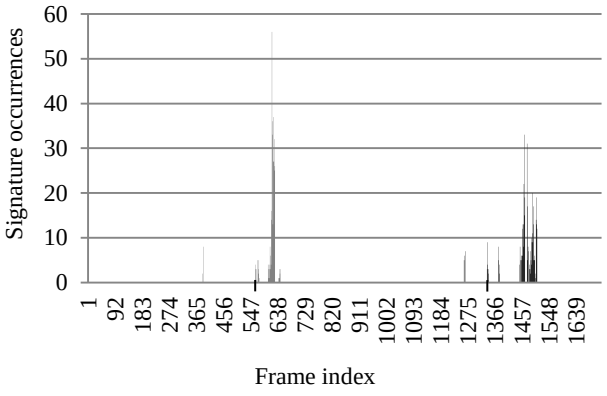


Fig. 2. Example of a histogram for *misex3*

error signature to jump to the best frame possible, MTTR-wise. Figure 1 shows the fine-grained error detectors (represented by the "=?" boxes) and Signature Translators (*ST*) embedded on a circuit.

To collect the error signatures, an error injection block is used, as described in the next section. The injector allows us to collect not only the signatures, but the frame address associated with them and when a different bit is tested within a frame. With this information, we can construct a histogram for each signature, with the frame number of the horizontal axis and the number of occurrences of that particular signature on the vertical axis. Figure 2 show the histograms for two different signatures for the *misex3* circuit.

It is possible to see in the histogram that one signature happens over 50 times for the same frame, frame 617. So it is fair to say that if that signature is detected, we could achieve a good precision if we simply corrected this frame. But it can also be seen that other frames generate the same signature as well and that they are near each other. So we can speculate that by starting the scrubbing by frame 617 we might achieve a low MTTR, but it might not be lowest possible. Because the scrubbing would not start at the first frame, we call this technique *shifted* scrubbing. To find the best starting position, we calculate the MTTR for each possible starting frame *f*:

$$MTTR_s(f) = \frac{FS}{BR} \sum_{i=PB}^{PE} \frac{h_s[i]}{O_s} \cdot (dist(i, f) + 1) \qquad (1)$$

Where $MTTR_s(f)$ is the MTTR for a given signature *s* and starting frame *f*, *FS* is the frame's configuration size in bits, *BR* is the scrubbing bit rate, *PB* is the partition beginning and *PE* is the partition end. $h_s[i]$ is histogram value for *s* for the *i*-th frame and O_s is the total amount of occurrences of *s*. Therefore, $h_s[i]/O_s$ is the probability that the error is located in the *i*-th frame, whenever *s* is received. *dist*(*i*, *f*) is the distance between *f* and the *i*-th frame, i.e., the amount of frames that have to be written before reaching the *i*-th. It is defined as:

$$dist(i, f) = \begin{cases} i - f, & \text{if } i \geq f \\ PE - f + 1 + i - PB, & \text{otherwise.} \end{cases} \quad (2)$$

The sum in (1) is, therefore, the "mean frames to repair" when signature *s* is received and *f* is used as starting frame. It is converted to a time unit with the time required to write a frame (*FS*/*BR*). The scrubbing controller would start on the best frame and reconfigure the whole device. If during the scrubbing it reaches the end of the partition, it would continue the scrubbing on the partition's beginning, until it reaches the last frame before the starting frame. This can be seen in equation (2), the first condition is the distance between *f* and *i* if *f*, the starting frame, is before *i*. In this case, the error is corrected before reaching the end of the partition. The second condition occurs when the error is only corrected after reaching the end of the partition and returning to its beginning. In this case, $PE - f + 1$ is the amount of frames written until the partition end and $i - PB$ is the distance between the partition beginning and *i*. One improvement would be stopping the scrubbing process after the error detection signals turn off, saving power and readying the controller for a new scrubbing round faster.

It is possible to leverage on the FPGAs high density if the error detectors and the blocks that translate the error signature to the optimum frame address, indicated as *ST* in Figure 1, are embedded on the device itself. This arrangement gives designers a high density device with self-error identification.

4. EXPERIMENTAL SETUP

In order to extract the error signatures, and thus identify which bits are critical in a design, it was used an error injection platform run on a Xilinx XUPV5-LX110T board, containing a Xilinx Virtex 5 XC5VLX110T FPGA device. This error injection platform relies on an error detection scheme, in our case, the one presented in [8]. It uses LUT-level DMR and the device's embedded carry chain to create an error detection bit for each of the device's slices. Bundling these all the error detection bits together, we form the error signature for that bit.

With the error detection in place, the injector platform exercises the Circuit Under Test (CUT) buy reading the configuration memory of a single frame through the Internal Configuration Access Port (ICAP); it then flips one bit in the read configuration and writes back this "errored" configuration on the device. The platform then excites the CUT by creating several pseudo-random input vectors by means of LSFR. While exciting the circuit, if one or more bits on the error signature turn on, the platform sends to a host PC the frame address being tested, the error signature itself and a flag bit if that signature is the first one for the bit being tested using a serial interface.

Table 1. Benchmark circuits

Circuit	LUTs	FFs	PIs	POs	S_{Size}
alu4	402	0	14	8	192
apex2	798	0	39	3	395
apex4	655	0	9	18	332
bigkey	575	224	264	197	354
clma	1269	34	384	82	609
des	550	0	256	245	355
diffeq	470	244	29	3	234
dsip	635	224	230	197	370
elliptic	143	71	20	2	73
ex1010	487	0	10	10	215
ex5p	128	0	8	63	81
frisc	1718	853	21	116	894
misex3	699	0	14	14	349
pdc	1253	0	16	40	603
s298	17	14	5	6	11
s38417	1709	1447	30	106	884
s38584.1	2001	1233	40	304	1080
seq	846	0	41	35	430
spla	221	0	16	46	114
tseng	598	260	53	122	337
Avg.	758.7	230.2	74.95	80.85	395.60

After all the bits in a frame's configuration frame are tested, that frame's original configuration is written back and the test of a new frame is begins. Because the signatures are sensible both to the flipped bit and to the input vector, it was chosen to limit the number of different signatures for the same bit to 20.

To determine the signatures' behavior for different types of circuits, we selected a set of 20 benchmark circuits from the MCNC suite; obtained at [9]. As the CUT and the injection platform are placed on the same device, it was necessary to limit the action of the injection platform on just the CUT and not on itself by the use of placement constrains to create an Area Under Test (AUT) in which the CUT is placed completely and exclusively.

To analyze the data collected, we wrote a C++ application to map the different signatures and then calculate for each signature the optimal beginning frame for scrubbing process. The application also calculates the MTTR for the standard scrubbing approach. An example of the optimum starting position for two signatures is shown in Figure 2 as the two black marks on the horizontal axis. As the errors are sensitive to the routing and placement choices of the synthesis tools, it is essential that this information is kept in the final design. It is possible to achieve this by the use of incremental design flow, among other means such as placement and routing constraints.

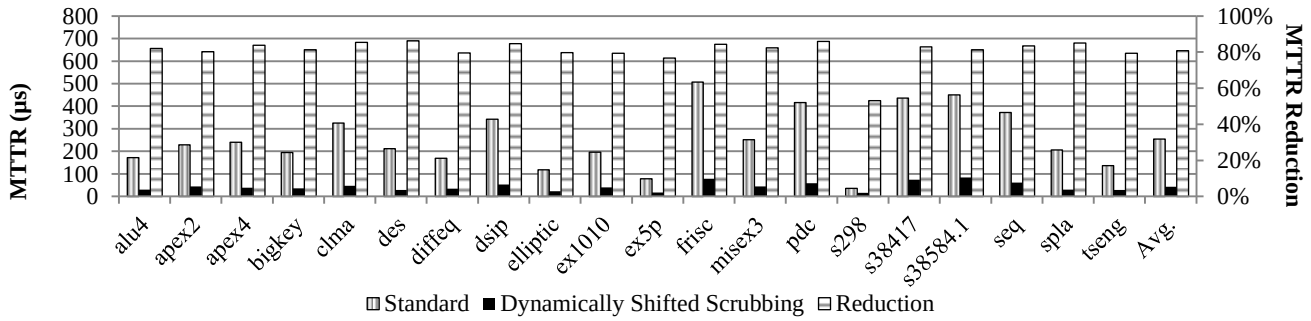


Fig. 3. MTTR for the standard and shifted scrubbing approaches and relative reduction

5. EXPERIMENTAL RESULTS

The list of the tested circuits from the MCNC suite is shown in Table 1, along with the resources used (pre-DMR), number of Primary Inputs (PI), number of POs and the signature size (post DMR) in bits. The circuits were tested according with the procedure described in section 4 and the error signatures were recorded and processed in a host PC.

All results assume a scrubbing interface operating at the maximum speed of the Virtex 5 SelectMAP interface, which is a 32-bit wide port at 100 MHz. It is also taken into account the time required to issue a write command to the interface (25 cycles in our implementation) and to write a dummy frame, which is required by SelectMAP. Such costs represent only 0.39 % and 1.9% of the total MTTR for standard and shifted scrubbing respectively. The MTTR was measured, in µs, for a standard scrubbing approach and for the shifted scrubbing. The obtained results are presented in Figure 3, together with the measured reduction in the MTTR. It can be seen that the gains in MTTR reduction are significant, with a minimum reduction of 77 % for the *ex5p* circuit and a maximum reduction of 86 % for the *des* and *pdc* circuits. The mean reduction for the 20 benchmark circuits was 80.85 %.

6. CONCLUSION

In this paper we have examined the possibility of reducing time needed to repair the configuration of a SRAM-based FPGA with a novel approach, using a shifted scrubbing process. By using a fine-grain error detection scheme allied with partial reconfiguration, it is possible analyze the circuit and discover information that allows us to precisely identify the frame with a configuration error and restore its correct state. The technique was evaluated through exhaustive testing with an error injection platform. The obtained results show that is possible to expect MTTR reductions of over 85 % for many of the benchmarked circuits. These results are very encouraging to further pursue optimizations of this technique.

Such a future work could see the detection and the signature translator circuits allied with a TMR scrubbing controller offering logic designers the advantages of SRAM-based FPGAs with self-repair capabilities.

7. REFERENCES

[1] Altera, "Increasing Design Functionality with Partial and Dynamic Reconfiguration in 28-nm FPGAs ", ed.

[2] Xilinx. *Partial Reconfiguration User Guide* Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug702.pdf

[3] Xilinx. *7 Series FPGAs Configuration User Guide* Available: http://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf

[4] G. Nazar and L. Carro, "Accelerated fpga repair through shifted scrubbing," in *Field Programmable Logic and Applications, 2013. FPL 2013. International Conference on*, 2013.

[5] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin, "Dynamic reconfiguration for management of radiation-induced faults in FPGAs," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, 2004, p. 145.

[6] C. Bolchini, A. Miele, and C. Sandionigi, "A Novel Design Methodology for Implementing Reliability-Aware Systems on SRAM-Based FPGAs," *Computers, IEEE Transactions on*, vol. 60, pp. 1744-1758, 2011.

[7] M. Psarakis and A. Apostolakis, "Fault tolerant FPGA processor based on runtime reconfigurable modules," in *Test Symposium (ETS), 2012 17th IEEE European*, 2012, pp. 1-6.

[8] G. L. Nazar and L. Carro, "Exploiting Modified Placement and Hardwired Resources to Provide High Reliability in FPGAs," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, 2012, pp. 149-152.

[9] K. Minkovich. *Kirill Minkovich's Home Page*. Available: http://cadlab.cs.ucla.edu/~kirill/

TOWARDS A DYNAMIC EVOLUTIONARY APPROACH TO FPGA TEMPERATURE MANAGEMENT

Peter R. Lewis, Walter C. Chibamu and Xin Yao

CERCIA, School of Computer Science
University of Birmingham, UK
{p.r.lewis – wcc081 – x.yao}@cs.bham.ac.uk

ABSTRACT

Reconfigurable computing systems, such as FPGAs, provide great promise for on-the-fly adaptive computation. Such systems can be modified while in use, to deal with newly arriving computational tasks. However, how to adapt, and with respect to what objectives, is not yet well established. Typically, one might be interested to achieve an allocation of tasks, as they arrive, to differently sized regions of the FPGA, in order to provide the most efficient computation. At the same time, it is important to also maintain healthy on-chip temperature. In this paper, we propose a novel description of this dynamic FPGA temperature management problem. We formulate the problem as a dynamic multi-objective optimisation problem, with three objectives and a time-linkage characteristic. We discuss the implications of the availability or otherwise of a model of the FPGA, and propose the use of evolutionary algorithms as a method to tackle the problem.

1. INTRODUCTION

A current trend in field-programmable gate array (FPGA) technology is an increase in the number and density of programmable logic components. However, increased density results in both higher and uneven chip temperature distributions (hot spots), the management of which is becoming increasingly important [1]. One factor that influences temperature is the distribution of tasks across the chip. This calls for efficient, temperature-aware methods, able to map incoming tasks and migrate existing tasks to cooler regions, such that the overall temperature is kept as low as possible and hot spots are avoided. However, migrating tasks which are already executing incurs an overhead in terms of execution time that should also be avoided. In this paper, we formalise the thermal management of FPGAs as a dynamic multi-objective optimisation problem. The task is to find a mapping of tasks to computational cores at a series of decision points, with the aim of achieving an efficient trade-off between conflicting objectives. Though in this paper we focus on FPGAs, the problem definition and proposed solution generalise beyond current technology, and we expect will apply equally to future massively parallel reconfigurable computing platforms, where dynamic temperature management will have an even bigger role to play.

This research was conducted in the EPiCS project and received funding from the European Union Seventh Framework Programme under grant agreement n^{o} 257906. http://www.epics-project.eu/ The authors would like to thank Markus Happe, Andreas Agne, Christian Plessl and Marco Platzner for their contribution to fruitful discussions.

2. BACKGROUND AND RELATED WORK

Modern FPGAs allow partial reconfiguration, where only a part of the FPGA is reconfigured (for example, a new piece of functionality is placed onto a currently idle region of the chip) while the rest of the FPGA remains active and continues to operate [2]. To enable granular chip temperature sensing, the FPGA may be partitioned into a regular grid of tiles [3]. A sensor is placed on each tile, which records the tile's temperature periodically. Processing elements (called *cores* in this paper) are laid over at least one tile. Figure 1 illustrates this idea whereby a 10×15 tile FPGA is configured into eight cores, occupying varying sized regions.

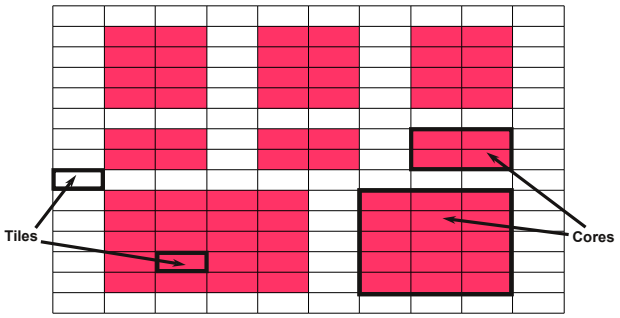


Fig. 1: Illustration of a multi-core FPGA with different sized cores and temperature sensing tiles. Example tiles and cores are indicated.

A number of *dynamic thermal management* techniques have been developed, which aim to control a chip's temperature [4, 5] at runtime. Our main contribution is to formulate the

problem formally, as a dynamic multi-objective optimisation problem. A clear formalisation of the FPGA temperature management problem helps us to gain a deeper understanding of the problem and various hidden factors and assumptions that are associated with it. It helps us to focus on the most important challenges and thus to propose most appropriate solutions.

Evolutionary computation (EC) has a long and successful history of application to complex optimisation problems [6]. The main idea of evolutionary algorithms, of which there are now many variants, is to maintain a population of candidate solutions to a problem (called *individuals*), which are evaluated against an objective function (called a *fitness* function in the parlance of EC), which defines the problem to be optimised. Based on the fitness of candidate solutions, they are selected (analogously to natural selection in biological evolution), before undergoing a perturbation (called *mutation* in EC), and optionally some recombination between individuals in the population. This results in a further *generation* of candidate solutions, which are then evaluated against the fitness function. The process iterates, typically until a time budget is exhausted, or a suitably optimal solution is found.

Relatively recently, EC has also begun to be applied to *dynamic optimisation problems* [7]. These are problems where there are multiple decision points, each at which a solution is implemented. Crucially, between these multiple decision points, the problem may have changed. In the case where the changes are incremental, i.e. the objective function at time t, f_t has some similarity to that at a previous time, e.g. f_{t-1}, then the incremental exploratory nature of evolutionary algorithms lends them well to the adaptation of solutions between decision points, since it acts to transfer knowledge from past problem instances to the current one. A more formal definition of a dynamic optimisation problem is given in [8]. A more comprehensive treatment of the topic is provided by a recently edited volume [9].

Several performance measures have been developed for evolutionary dynamic optimisation [7, 8]. Depending on the assumptions present in the real world problem, different performance measures may be more appropriate. Firstly, the *online performance metric* [7] is defined as the sum or average of all fitness evaluations over the entire problem:

$$F_{online} = \sum_{t \in T'} f_t(x) \tag{2.1}$$

where T' is the set of all time points at which fitness may be evaluated, f_t is the fitness function at time $t \in T'$ and x is a candidate solution to be evaluated.

The online measure is appropriate in the case when there exists no simulation or mathematical model of the problem, and therefore each and every evaluation must be carried out in the real world. In this case, every fitness evaluation counts. This has significant implications for the exploratory behaviour of the algorithm, since any bad individuals generated through exploration of new regions of the search space, impact directly upon overall performance.

In the case when either a model of the problem may be assumed, or else when the algorithm may obtain the use of the real-world problem instance during a *training phase* that does not impact upon problem performance, we may employ an offline performance measure. An original *offline performance metric* was defined by Branke [7], which takes the average of the best individual found so far, at each time step. As Branke points out however, this does not account for the case when an individual from a previous time has a reduced fitness at later time points. Several variants of offline performance have since been defined in the literature, though they are typically tied either to generations (as in the *best of generation* measure) or else to detected changes (as in the *modified offline* measure) [8].

In the FPGA temperature management problem as described in this paper, it appears more intuitive to tie a possible offline performance evaluation to discrete decision points, which correspond to opportunities to reconfigure the FPGA. In this case, we define offline decision-based performance to be:

$$F_{offline} = \sum_{t \in T} f_t(x) \tag{2.2}$$

where $T \subset T'$ is the set of time points at which a decision is made and a solution implemented in the FPGA, a subset of the full set of time points at which the evolutionary algorithm performs fitness evaluations (on the model). Note that a decision may involve making no change to the FPGA relative to the previous decision, which may be desirable in the case when no better solution has been found in the interim.

In addition to being a dynamic optimisation problem, the FPGA temperature management problem is inherently multi-objective, since we are concerned both with computational performance of the chip as well as managing the chip's temperature. Multi-objective optimisation problems are a class of optimisation problems in which our aim is to simultaneously optimise two or more objectives [10]. Since the objectives are often in conflict, we cannot typically find solutions which obtain the global optimum for all objectives simultaneously. Instead, we aim to find Pareto-optimal solutions, i.e. those which cannot be further improved on one objective, without being strictly worse on at least one other objective. A preference model is then used to rank the found Pareto-optimal solutions, and to select one for implementation. Evolutionary computation has also long been successfully applied to multi-objective optimisation problems [11].

Evolutionary algorithms have been used to solve several optimisation problems arising in FPGA configuration (e.g. [12, 13]). However, prior work has not considered run time reconfiguration as a dynamic optimisation problem, where multiple decisions are made during run time, as the problem changes with unforseen incoming tasks and the effect of past decisions. Instead, these problems consider the entire configuration over time as a static optimisation problem to be solved ahead of time.

The problem of determining task placement online in partially reconfigurable FPGAs has also been tackled using evolutionary algorithms [14]. However, this early work did not consider the problem over time, as a dynamic optimisation problem. Instead, evolutionary algorithms were used to solve multiple independent task rearrangement problems during run time. This assumption of independence ignores the relationship between decisions, and cannot account for time linkage. Additionally, the work did not consider temperature, and though multiple performance-relevant metrics were used, a multi-objective approach was not taken. The metrics were instead combined into a single scalar objective to be optimised. Crucially however, Middendorf et al. [14], showed the effectiveness of using evolutionary computation for online management of partially reconfigurable FPGAs. In this paper, we present the first formulation of the problem as a dynamic multi-objective optimisation problem, which allows for run time optimisation, considering trade offs between conflicting objectives as well as time linkage. Our formulation is also the first to consider temperature as an additional explicit objective in such an optimisation problem, using real time readings. This is considered alongside task performance objectives, in a multi-objective setting.

3. PROBLEM FORMULATION

In the dynamic FPGA temperature management problem, we seek to schedule incoming tasks and migrate tasks currently in execution, during run time, in order to:

1. Minimise average chip temperature,
2. Minimise spatial temperature variations (hot spots),
3. Minimise the total time taken to execute all tasks.

We consider temperature measurement at the granularity of a *tile*, a small surface area over which a physical sensor records the temperature. A *core* is a processing element on which computation can be carried out. These are depicted in Figure 1. The configuration in which tasks are assigned to cores has an impact on the resultant chip temperature landscape over time. In addition, cores may be heterogeneous, such that, depending on which core it is executed, a task may take a different amount of time to complete, and may generate a thermal contribution with a different distribution.

3.1. Notation

Our problem definition makes use of the following notation:

- A finite set of discrete time intervals over which the FPGA is to be managed, $T = \{0, 1, 2, 3, \dots t_{max}\}$, s.t. $T \subset \mathbb{N}_0$ and $t_{max} = |T| + 1$.
- A single task in the set of tasks which, at time t, have arrived but have not yet been completed, $\tau \in \mathcal{T}_t$.
- The arrival time of a task, $\alpha : \bigcup_{t \in T} \mathcal{T}_t \rightarrow T$.
- The completion time of a task, $\phi : \bigcup_{t \in T} \mathcal{T}_t \rightarrow T$.
- A fixed set of *processing* cores, $C = \{c_0, \dots, c_{|C|-1}\}$, able to process incoming tasks.
- A notional *waiting* core, w, to which a task may be assigned to indicate that it is in a waiting state (i.e. the task is not currently being executed).
- A notional *unusable* core, u, to which a grid tile may be mapped to indicate that it measures an unused area of the FPGA.
- A set of grid tiles, $G = \{g_{0,0}, g_{0,1} \dots, g_{x,y}\}$, used for temperature sensing; x and y denote the number of horizontal and vertical tiles, respectively.
- A mapping of grid tiles to cores, $\sigma : G \rightarrow C \cup \{n\}$, indicating the layout of the FPGA. σ is required to be surjective.
- A mapping of tasks to cores at time t, $\mu_t : \mathcal{T}_t \rightarrow C \cup \{w\}$. μ_t is required to be injective, except that many tasks may be mapped to the *waiting* core, w.
- A reading of each tile's temperature at time t, $\theta_t : G \rightarrow \mathbb{R}$, s.t. $t \in T$.
- The ambient chip temperature, $\Theta_{ambient}$.
- The total workload of a task, in (arbitrary) workload units, $\omega : \bigcup_{t \in T} \mathcal{T}_t \rightarrow \mathbb{N}$
- The current progress of a task, more precisely a counter indicating the next workload unit of a task to be executed, $\lambda : \bigcup_{t \in T} \mathcal{T}_t \rightarrow \mathbb{N}$.
- The task-core workload rate, $\rho : \cup_{t \in T} \mathcal{T}_t \times C \cup \{w\} \rightarrow \mathbb{N}$ (i.e. how much of a task's workload is completed per time interval by a given core). $\rho(\tau, w)$ is defined to be 0 for any τ.
- The task-core reconfiguration penalty, $\pi : \cup_{t \in T} \mathcal{T}_t \times C \cup \{w\} \rightarrow \mathbb{N}$ (i.e. the time taken, in workload units, to reconfigure a given core to execute a given task). E.g. $\pi(\tau, c_1)$ indicates the workload not carried out during a time instance, when migrating task τ to core c_1. It is assumed that $\pi(\tau, c) \leq \rho(\tau, c)$ for all τ and c,

i.e. that all migrations may be completed within one discrete time step.

- The task-tile thermal contribution, $\delta : \cup_{t \in T} \mathcal{T}_t \times G \times \mathbb{N} \times C \times \Sigma \to \mathbb{R}$ (i.e. how much temperature contribution a task makes to a tile, for a given workload unit of that task, when running on a given core). Σ represents the space of possible grid-tile mappings, i.e. possible instances of σ. $\delta(\tau, g, x, c, \sigma)$ is undefined if $x > \omega(\tau)$, for all τ and any g, c and σ.

  Typically, the thermal contribution of a task τ to tile g occupied by core c running task τ (i.e. $\sigma(g) = c$ and if $\mu(\tau) = c$) during one time step will be $\sum_{x=\lambda(\tau)}^{\lambda(\tau)+\rho(\tau,c)} \delta(\tau, g, x, c, \sigma)$, except when τ is migrated to c in that time step, in which case the thermal contribution will be $\sum_{x=\lambda(\tau)+\pi(\tau,c)}^{\lambda(\tau)+\rho(\tau,c)} \delta(\tau, g, x, c, \sigma)$.

  If the core does not occupy the given grid tile, i.e. $\sigma(g) \neq c$ then $\delta(\tau, g, x, c, \sigma) = 0$ for any τ and x. In this case, any heat on tile g due to task τ will be through dissipation through the chip, and not due to a direct contribution.

3.2. Assumptions

We make the following simplifying assumptions:

- Workload rates ρ do not vary with temperature.
- For a given task and core, the task-core workload is constant throughout the lifetime of the task.
- Cores which complete a task during one time interval generate a thermal contribution as if they had executed that task for the entire time interval.
- Reconfiguration itself does not generate any heat.
- There are no data dependencies between tasks.

Additionally, we assume that since tasks arrive over time, no information about them is available until they have arrived. Therefore, one cannot reason about $\mathcal{T}_{t+n}$, where $n \geq 1$, at time t, since those tasks have not yet been seen.

3.3. Problem Framework

A problem instance begins with initial temperatures, $\theta_t(g) = \Theta_{ambient}$ for all $g \in G$. The problem instance iterates:

1. Initialisation:
   (a) $t = 0$.
   (b) **Arrival of first tasks:**
   $\mathcal{T}_t = \{\tau : \tau \in \bigcup_{s \in T} \mathcal{T}_s, \alpha(\tau) = 0\}$,
   the set of tasks arriving at time 0.
   (c) Since no task is running when it arrives,
   $\mu_t(\tau) = w$, for all $\tau \in \mathcal{T}_t$.
   (d) Set counters to the start of each arriving task: for all $\tau \in \mathcal{T}_t$, $\lambda(\tau) = 1$.
2. $t = t + 1$.
3. **Measure temperature on each tile:**
   Record values for $\theta_t(g)$, for all $g \in G$.
4. **Decide on mapping** $\mu_t(\tau)$ for each $\tau \in \mathcal{T}_t$.
5. **Reconfigure FPGA** according to μ_t, if necessary.
6. **Carry out computational work:**
   For each $\tau \in \mathcal{T}_t$,
   $\lambda(\tau) = \begin{cases} \lambda(\tau) - \rho(\tau, \mu(\tau)), \text{ if } \mu_t(\tau) = \mu_{t-1}(\tau) \\ \lambda(\tau) - \rho(\tau, \mu(\tau)) + \pi(\tau, \mu(\tau)), \text{ otherwise.} \end{cases}$
   During this step, the thermal contributions are made and heat dissipates.
7. Build $\mathcal{T}_{t+1}$:
   (a) **Remove completed tasks:**
   $\mathcal{T}_{t+1} = \mathcal{T}_t \setminus \{\tau : \tau \in \mathcal{T}_t, \lambda(\tau) > \omega(\tau)\}$
   (b) **Record completion times of completed tasks:**
   $\phi(\tau_c) = t$,
   for all $\tau_c \in \{\tau : \tau \in \mathcal{T}_t, \lambda(\tau) > \omega(\tau)\}$
   (c) **Arrival of new tasks:**
   $\mathcal{T}_{t+1} = \mathcal{T}_{t+1} \cup \{\tau : \tau \in \cup_{s \in T} \mathcal{T}_s, \alpha(\tau) = t+1\}$.
   (d) Set initial task state for new tasks:
   For all $\tau \in \mathcal{T}_{t+1}$, if $\alpha(\tau) = t + 1$:
      i. $\mu_t(\tau) = w$, and
      ii. $\lambda(\tau) = 1$.
8. If $t < t_{max}$, go to step 2, otherwise end.

3.4. Instantaneous Optimisation Task

In the above problem framework, the task is to make the decision at each point 4, to decide the mapping from tasks to cores. This is encapsulated at each time point t, in the mapping μ_t, which an optimisation algorithm is free to set. Informally, we described the objectives which we would like to make this decision with respect to, at the start of section 3. We now formalise these objectives for a given time point, t:

1. Minimise average chip temperature, specifically:

$$\textit{minimise } f_{1t}(\mu_t) = \frac{1}{|G|} \sum_{g \in G} \theta_t(g) \tag{3.1}$$

2. Minimise spatial temperature variations (hot spots). There are various ways in which this could be tackled. Here we take a minimax approach:

$$\textit{minimise } f_{2t}(\mu_t) = \max\left(|\theta_t(g) - \theta_t(h)|\right) \tag{3.2}$$

$$\forall g \in G, \forall h \in \nu(g)$$

where $\nu(g)$ is the set of all tiles in the Moore neighbourhood of g.

3. Minimise the total time taken to execute all tasks. At a particular time instance, this may be seen as maximising the workload carried out at this time, for all currently existing tasks:

$$\textit{maximise } f_{3t}(\mu_t) = \sum_{\tau \in \mathcal{T}_t} \rho(\tau, \mu(\tau)) - \begin{cases} \pi(\tau, \mu(\tau)) & \text{if } \mu_t(\tau) \neq \mu_{t-1}(\tau) \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

The performance of a given mapping μ_t, with respect to the instantaneous optimisation objectives, f_{1t}, f_{2t} and f_{3t} is calculated based on temperature readings θ_t, the workload and penalty rates ρ and π, and an observation of required reconfigurations (mappings and migrations), due to a difference between μ_{t-1} and μ_t. In a naive way, we may now tackle the problem of selecting a μ_t for step 4, evaluating candidates for μ_t against the three objectives.

3.5. Objectives Over Time

Although decisions are made at each of the time points in T, from the problem perspective, we are interested in the objectives over the entire lifetime of the management of the FPGA, i.e. over all of T. Therefore, when evaluating the performance of a particular algorithm for determining task to core mappings, we must define metrics which quantify these over time. In section 3.4 we defined objectives for a naive decision at time t. There are many ways in which we could define complementary performance metrics for an entire problem instance, e.g. by borrowing some ideas from robust optimisation over time [15, 16, 17].

Here, we propose three performance metrics over time:

1. Minimise average chip temperature over time:

$$\textit{minimise } f_1 = \frac{1}{T} \sum_{t \in T} f_{1t}(\mu_t) \tag{3.4}$$

2. Minimise spatial temperature variations over time:

$$\textit{minimise } f_2 = \frac{1}{T} \sum_{t \in T} f_{2t}(\mu_t) \tag{3.5}$$

3. Minimise the total time taken to execute all tasks. Since, assuming all tasks completed, a simple summation of f_{3t} would not capture penalties (i.e. time wasted performing migrations), and a simple count of migrations would ignore differences in $\rho(\tau, c)$ as c varies, we instead measure finish times of tasks:

$$\textit{minimise } f_3 = \sum_{\tau \in \bigcup_{t \in T} \mathcal{T}_t} \phi(\tau) \tag{3.6}$$

It is important to note, that while decisions are made locally within each time step, the performance of any algorithm generating such decisions should be evaluated in terms of these three objectives calculated over all of T. Importantly, whether these objectives are measured online or offline will depend on the availability of a simulation or mathematical model of the problem at each decision point, with sufficient fidelity to provide meaningful fitness evaluations between decision points. Since f_1 and f_2 depend on θ_t values, which must either be measurements from the real system at time t or predictions from a model, then the lack of a model means that each and every fitness evaluation must be carried out on the real system. Furthermore, in a possible extended version of this problem, ρ may not be known, or may vary, and may indeed also need to be a runtime observation.

3.6. Time linkage

Though f_{3t} considers μ_{t-1} explicitly, f_{1t}, f_{2t} and f_{3t} are driven by various μ_s where $s < t$. This is since the current state of the FPGA at time t will be dependent on work carried out prior to time t. Specifically, both the current temperature of the grid tiles and the currently running tasks on each core, can alter the values for each of f_{1t}, f_{2t} and f_{3t} for a given input. In the language of evolutionary dynamic optimisation, this is referred to as *time linkage* [18].

The implication of this is important, since the quality of decisions made at time t is affected directly by decisions made prior to t. In some cases, this means that a seemingly good decision at one point in time, acts to limit the quality of a decision at a later point in time. For example, greedily placing an incoming task onto the core which will complete that task the quickest, may mean that a later task, which would benefit more from being placed on that core, cannot be placed there without incurring a penalty incurred by migrating the first task away first. Thus, greedily optimising the instantaneous objective functions, f_{1t}, f_{2t} and f_{3t}, is very likely not to be optimal in terms of the objectives specified in f_1, f_2 and f_3. An high performing optimisation algorithm should therefore be designed to account for this time-linkage, in the presence of uncertainty over future incoming tasks.

4. CONCLUSIONS AND OUTLOOK

In this paper we have defined a novel formulation for the problem of dynamically managing partially reconfigurable

FPGAs, with respect to completion time of incoming tasks and the management of on-chip temperature. The problem is formulated as a dynamic multi-objective optimisation problem, which enables us to explore the trade-off between computational efficiency, heat generation and temperature imbalance. Although three objectives were proposed in this paper, our formulation can easily accommodate additional objectives, e.g., energy consumption of the chip, security, reliability, etc. All these are important issues, which can now be considered under the same problem formulation.

In tackling this problem, we propose the use of evolutionary algorithms, since these have been used successfully for both dynamic and multi-objective problems for many years. Their incremental exploratory nature enables them to re-use information from previous time instances, for solutions in the present, when the problem consists of a sequence of related problem instances over time. The performance of such algorithms should be compared with that of existing heuristics, such as those described by Happe [5, pp.73–81].

We highlighted a number of challenges which need to be addressed in order to tackle this problem. Firstly, the issue of time linkage means that optimisation algorithms which are greedy with respect to time, will likely not be optimal over the entire problem instance. Instead, algorithms will need to make predictions about future tasks and chip behaviour. Secondly, techniques from both dynamic and multi-objective optimisation will need to be combined in this problem which contains both characteristics. Thirdly, a question arises on the availability of a suitable simulation or mathematical model, which would enable an evolutionary algorithm to evaluate candidate solutions between decision points. Ultimately, the availability (or otherwise) of such a model will drive the selection of appropriate evolutionary techniques.

Finally, it will be important to evaluate the overhead of the computational work required to apply evolutionary techniques to tackle this problem at runtime, in terms of the objectives considered here. Depending on the exact form the evolutionary algorithm takes (importantly, e.g. either online or offline), this overhead may actually make things worse. Indeed, regardless of the approach taken, the costs as well as benefits of a given technique should be considered. One approach could be for the decision process to be seen as a task itself, whose impact is evaluated as part of the whole system.

5. REFERENCES

[1] S. Borkar, "Thousand core chips: a technology perspective," in *Proc. 44th Annual Design Automation Conference*, ser. DAC '07. New York, NY, USA: ACM, 2007, pp. 746–749.

[2] A. Gupte and P. Jones, "Hotspot mitigation using dynamic partial reconfiguration for improved performance," in *Reconfigurable Computing and FPGAs, 2009. ReConFig '09. International Conference on*, 2009, pp. 89 –94.

[3] M. Happe, H. Hangmann, A. Agne, and C. Plessl, "Eight ways to put your FPGA on fire – A systematic study of heat generators," in *Proc. Int. Conf. on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE Computer Society, 2012.

[4] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *7th Int. Symp. on High-Performance Computer Architecture (HPCA)*, 2001, pp. 171–182.

[5] M. Happe, "Performance and thermal management on self-adaptive hybrid multi-cores," Ph.D. dissertation, Paderborn University, 2013.

[6] R. Chiong, T. Weise, and Z. Michalewicz, Eds., *Variants of Evolutionary Algorithms for Real-World Applications*. Springer, 2012.

[7] J. Branke, *Evolutionary Optimization in Dynamic Environments*. Kluwer, 2001.

[8] T. T. Nguyen, S. Yang, and J. Branke, "Evolutionary dynamic optimization: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 6, pp. 1–24, 2012.

[9] S. Yang and X. Yao, Eds., *Evolutionary Computation for Dynamic Optimization Problems*. Springer, 2013.

[10] J. Branke, K. Deb, K. Miettinen, and R. Slowiski, Eds., *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Springer, 2008.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. on Evol. Comp.*, vol. 6, no. 2, pp. 182–197, 2002.

[12] J. Harkin, T. McGinnity, and L. Maguire, "Genetic algorithm driven hardware-software partitioning for dynamically reconfigurable embedded systems," *Microprocessors and Microsystems*, vol. 25, no. 5, pp. 263–274, 2001.

[13] H. Fröhlich, A. Kosir, and B. Zajc, "Optimization of FPGA configurations using parallel genetic algorithm," *Information Sciences*, vol. 133, no. 3–4, pp. 195–219.

[14] M. Middendorf, B. Scheuermann, H. Schmeck, and H. El-Gindy, "An evolutionary approach to dynamic task scheduling on FPGAs with restricted buffer," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1407 – 1420, 2002.

[15] H. Fu, B. Sendhoff, K. Tang, and X. Yao, "Finding robust solutions to dynamic optimization problems," in *Proc. EvoApplications 2013, LNCS 7835*, A. I. Esparcia-Alcázar *et al.*, Eds. Springer-Verlag, 2013, pp. 616–625.

[16] ——, "Characterizing environmental changes in robust optimization over time," in *Proc. 2012 IEEE Congress on Evolutionary Computation (CEC'12)*. IEEE Press, 2012, pp. 551–558.

[17] X. Yu, Y. Jin, K. Tang, and X. Yao, "Robust optimization over time – a new perspective on dynamic optimization problems," in *Proc. 2010 IEEE Congress on Evolutionary Computation (CEC2010)*. IEEE Press, 2010, pp. 3998–4003.

[18] T. T. Nguyen and X. Yao, "Dynamic time-linkage evolutionary optimization: Definitions and potential solutions," in *Metaheuristics for Dynamic Optimization*, ser. Studies in Computational Intelligence, E. Alba, A. Nakib, and P. Siarry, Eds. Springer, 2013, vol. 433, pp. 371–395.

USING STATISTICAL ASSERTIONS TO GUIDE SELF-ADAPTIVE SYSTEMS

Tim Todman, Wayne Luk

Department of Computing
Imperial College London
180 Queen's Gate, London SW7 2AZ
email: {timothy.todman, w.luk}@imperial.ac.uk

Stephan Stilkerich

Software Engineering
EADS Innovation Works
Willy-Messerschmitt Str. 1, 85521 Ottobrunn
email: stephan.stilkerich@eads.net

ABSTRACT

Self-adaptive systems need to monitor themselves, to check their internal behaviour and design assumptions about run-time inputs and conditions. This kind of monitoring for self-adaptive systems can include collecting statistics about systems themselves which can be computationally-intensive (for detailed statistics) and hence time-consuming, with possible negative impact on self-adaptive response time. To mitigate this limitation, we extend the technique of in-circuit run-time assertions to cover statistical assertions in hardware. The presented designs implement several useful statistical operators that can be exploited by self-adaptive systems. To illustrate the practicability and industrial relevance of our proposed approach, we evaluate our designs, chosen from a class of possible application scenarios, for their resource usage and the tradeoffs between hardware and software implementations.

I. INTRODUCTION

Self-adaptive systems can configure themselves to flexibly deal with changing environments after they are deployed. The configuration itself is systematically guided by means of system self-monitoring to aid decisions about changing modes, or to check design assumptions about runtime data and conditions or their internal operation. Such monitoring could check elementary Boolean conditions or, more generally, could process collected run-time system data, feeding a process of deciding whether or how the system can be adapted. The response time to adaptation is a fundamental feature characterizing self-adaptive systems. For the class of applications from the avionics domain we investigate, a fast response time to adaptation is crucial and motivates our advocated approach, presented in the rest of the paper. Gathered system data can be used for many purposes: for example, design assumptions about input range, used to optimize operator bit-widths, can be checked by assertions about the standard deviation of the input.

In this paper, we propose *in-circuit, statistical assertions*, compiled into the hardware part of a software-hardware design as a dedicated self-monitoring facility for self-adaptive systems, with a fast response time to adaptation. Compared to the proposed in-circuit assertions that can compute in parallel with the rest of the design, purely software-implemented assertions need to wait until the hardware has finished computing its results before they can process their own tasks. Moreover, efficient hardware designs are often deeply-pipelined, operating on large batches of data, further prolonging the time until software assertions can start processing. Additionally, by preprocessing potentially large amounts of data, in-circuit data gathering can improve use of limited bandwidth between hardware and software of the self-adaptive system triggering and controlling system adaptation. In summary: in-circuit assertions are the necessary precondition to realize fast response times to adaptation not realizable by pure software assertions.

Figure I provides a structural overview of our approach. A hardware datapath is instrumented by in-circuit statistical operators which compute relevant statistics about the design. These are then sent back to a software engine running a self-adaptive system. The software builds up the self-adaptive representation which is used to control reconfiguration of the system. It should be mentioned that whilst we target a software-hardware system setting, our approach is not limited to this setting at the outset. The software could likewise run on a soft processor within a Field Programmable Gate Array (FPGA) fabric.

This paper makes the following contributions:

- The design and implementation of in-circuit statistical assertions, which can be used by self-adaptive systems to monitor themselves and control system adaptation;
- A case study on avionics systems, showing the potential of in-circuit statistical assertions;
- Evaluation of tradeoffs between assertion implementations in software and in hardware, showing the advantages of our proposed in-circuit assertions.

The rest of the paper is organized as follows: the next section describes related work. Section III shows our designs for assertions and implementations for Maxeler systems;

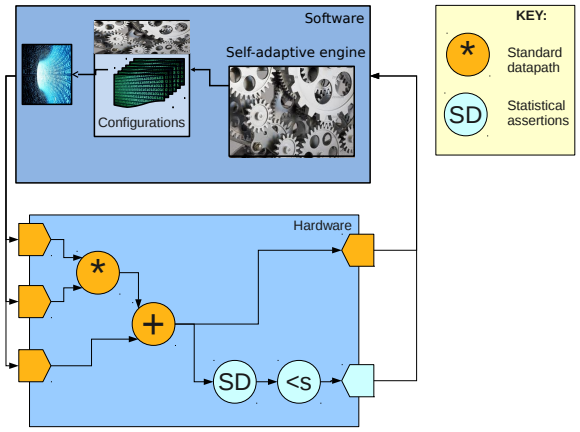


Fig. 1. Our approach: hardware datapath augmented with in-circuit statistical assertions feeding an engine running a self-adaptive algorithm in software.

section IV is a brief case study for avionics. Section V evaluates our implementation; section VI concludes and suggests future work.

II. BACKGROUND

Runtime verification: several researchers have used temporal logic for runtime verification; for example, Reinbacher et al. [1] implement hardware temporal logic monitors for a software system running on a soft processor on the same device. Calinescu et al. [2] propose that self-adaptive software needs quantitative runtime verification; our statistical in-circuit assertions could complement such approaches.

Assertion-based verification allows the use of Boolean and temporal assertions for debugging designs in simulation [3], extended to in-circuit assertions by Curreri [4]. We extend in-circuit assertions with statistical operators; in our approach, failed assertions do not necessarily indicate errors, but may be the trigger for a self-adaptive system to adapt or reconfigure itself.

Statistical assertions have been proposed by Dinh et al. [5], as a debug-time method to reason about large parallel programs – users can reason with derived metrics, rather than raw program output. The assertions are implemented efficiently using a map-reduce style computation. We use statistical assertions for run-time monitoring of reconfigurable hardware-accelerated systems.

III. IN-CIRCUIT STATISTICAL ASSERTIONS

This section shows our designs for in-circuit statistical assertions. Our assertion language comprises C-language style Boolean operators, augmented by statistical primitives. We choose the C language as it is familiar to many designers. The set of statistical primitives is as follows:

- $mean(e1)$, the mean value of expression $e1$;
- $stdev(e1)$, the standard deviation of expression $e1$;
- $variance(e1)$, the variance of expression $e1$.

We choose these as a useful set for expressing statistical conditions; future work could add further statistical operators such as covariance, skewness and kurtosis, or limit the number of cycles over which the statistics are calculated, potentially reducing hardware resources.

The following shows the grammar of our statistical assertions language in extended Backus-Naur form:

```
e = a
  | e bop e
  | uop e
  | mean(e)
  | stdev(e)
  | variance(e)
bop = == | != | < | > | ...
uop = + | - | ! | ~
```

where *bop* represents any C binary operator, *uop* any C unary operator and a any atomic expression (literals, variables, constants). This language allows the user to combine both Boolean and statistical operators.

Online algorithms for calculation of statistical metrics such as mean, variance and standard deviation are known [6] [7], which involve a single pass over the input data, using an accumulator and the current input element. Whilst this may seem suitable for streaming implementations, they contain feedback owing to the accumulator. Chan et al. developed a pairwise algorithm for variance [8] which can be parallelized; for N input elements, naively implementing this algorithm on streaming systems requires $O(NlogN)$ hardware. Chan et al's algorithm denotes the sum and mean of data points x_i as T_{ij} and M_{ij} respectively:

$$T_{ij} = \sum_{k=i}^{j} x_k \qquad M_{ij} = \frac{1}{j-i+1} T_{ij}$$

and the sum of squares S_{ij}:

$$S_{ij} = \sum_{k=i}^{j} (x_k - M_{ij})^2$$

calculated by their pairwise algorithm:

$$S_{1,2m} = S_{1,m} + S_{m+1,2m} + \frac{1}{2m}(T_{1,m} - T_{m+1,2m})^2$$

We propose two designs suitable for streaming systems: a systolic design adapted from Chan's parallel algorithm, and a C-slowed variant of the online algorithm. Figure 2 shows the datapath for the systolic design, combining stream offsets with Chan's pairwise operators for calculating variance or mean; for clarity, we omit the calculation of $T_{i,j}$, which

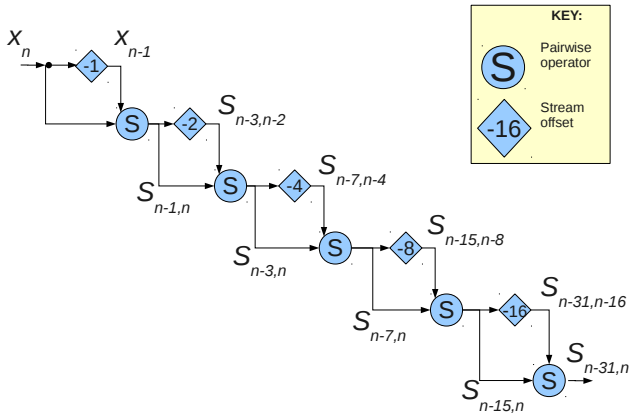


Fig. 2. Systolic partial calculation of variance using pairwise operators.

has the same pattern. Note that the leftmost operator can be optimized, because $S_{i,i} = 0$ (the variance of a single point is zero). The systolic design uses the observation that in a streaming system, iterating through the input data in order, sums of neighbouring elements can be accessed by stream offsets, so using Chan et al's notation:

$$\begin{aligned} T_{1,2^k} &= T_{1,2^{k-1}} + T_{2^{k-1}+1,2^k} \\ &= T_{1,2^{k-1}} + offset(T_{1,2^{k-1}}, -2^k) \end{aligned}$$

where $offset(e, n)$ means the value of expression e sampled n cycles in the past; $S_{1,2^k}$ is calculated in the same way. Unlike the naive implementation of Chan et al's algorithm, which requires $O(NlogN)$ hardware, this requires $O(logN)$ statistical operators plus $O(N)$ delay elements used to implement the $offset$ operation.

Note that this only calculates part of the variance, specifically the local variance around each sample; however, it greatly reduces the amount of data sent back to software. The design consists of repeating units of the pairwise operator and stream offsets to delay the input. Each repeating unit reduces both the output data and the remaining calculations to be done in software by half, so K units reduces it 2^K-fold.

Implementation targeting Maxeler streaming designs: we choose Maxeler streaming systems to implement our designs, though the approach is not Maxeler-specific, and can be ported to other design descriptions such as Verilog and VHDL. We focus on a systematic approach to translating assertions into Maxeler designs; future work could implement a compiler from Maxeler designs extended with statistical assertions into the base language.

The Maxeler system generates *streaming* designs, where inputs and outputs are large arrays used as streams. Each output element is calculated from corresponding elements in one or more input streams; offsets allow reading from neighbourhood stream elements. The user programatically builds a datapath using a domain-specific language based on Java. The control path may be counters or state machines generated from another domain-specific language.

Maxeler tools compile designs into hardware description languages and control FPGA vendor tools to build a reconfigurable device bitstream implementing a design. Software can interact with the generated hardware using a Maxeler application programming interface to configure the FPGA device with the bitstream and run on user data stored in C arrays. The Maxeler tools automatically pipeline the datapath, resulting in deeply-pipelined operators at a high clock rate. This works well for feed-forward designs, but feedback requires some manual intervention and reordering or duplicating of input data.

IV. CASE STUDY: AVIONICS SYSTEMS

Avionics systems are electronic systems used for control or information in the aviation or aerospace industries [9].

Self-adaptive systems with a fast response to adaptation (where fast means quicker than $500ms$), are promising architectures for dedicated application scenarios in the avionics and space-flight industry. Systems that profit from architectures with fast response time to adaptation are:

- autonomous flying systems,
- special satellites,
- deep-space mission systems,
- exploratory space mission systems.

All these systems operate in environments that can not fully be described right from the beginning and hence the systems cannot be statically designed to cover and handle all environmental settings. Furthermore, these systems have strong constraints on power consumption, weight and packaging volume. Additionally, these systems may never be reachable after deployment.

We choose a $500ms$ limit as this duration fits perfectly into most processing and control-loops of the systems and application scenarios mentioned in the paper. Hence, if we realize our self-adaptation and self-expression with the configuration within this limit, it would seamlessly fit into our systems, applications and the already available developed systems.

We analyze the processing structure of these systems for the functionality of guidance, navigation and orientation, revealing that the processing is composed of different blocks/kernels with inputs and outputs. Determining the adequate bit-widths and hence precision for the inputs and outputs is difficult and is today based on worst-case assumptions involving unnecessary resources. An alternative is to start with an initial, more optimistic design assumption about the input/output range, used to optimize operator

bit-widths. Such assumptions can be checked by assertions about the standard deviation of the input and adapted by another kernel-version accordingly if required. Obviously, fast response time to adaptation is to avoid compromising system functionality, and to simultaneously optimize the system at run-time with respect to energy efficiency and environmental adaptability.

V. EVALUATION

We evaluate our implementations of on-chip statistical assertions showing the tradeoff between hardware and software implementations. We compare: 1) scalability: operator size versus hardware size; 2) software statistics versus hardware-assisted: speed, bandwidth.

Experimental setup: we implement our designs using Maxeler compiler version 2012.1 and Xilinx ISE 13.1. Designs target a MAX3 system, containing a Xilinx xc6vsx475t FPGA, with a speed goal of 200MHz. We implement a single variance assertion, with 32-bit input data in IEEE Single-Precision (SP) floating-point format, one data element per cycle.

Figure 3 shows the effect of unroll factor on the area resources for the systolic variance operator; the area is measured in Look-Up Tables (LUTs), Flip-Flops (FFs) and Digital Signal Processing (DSP) blocks. The area cost is linearly proportional to the unroll factor (for LUTs and FFs), but the output data reduction factor is exponential: increasing the unroll factor by one halves the output volume. For unroll factor 15, the data reduces by 2^{15} and the variance takes about 5% of flip-flops, 8% of other resources. For LUTs and FFs there is also a small fixed cost which is due to the Maxeler runtime system used to communicate with the host. The cost in block RAMs (BRAMs) is exponential in the unroll factor, as they are used to store delayed stream elements used to calculate the $offset$ expressions; however, the cost is still modest even for large unroll factors.

The C-slowed design uses a small fixed area per assertion (about 3.5% of LUTs, 2% of FFs for 32-bit SP variance). For 32-bit SP data, the pipeline is 85 stages long, padded to 128 stages. The data are reduced to 128 partial variances, which can be further reduced to a single variance by Chan et al's method. The design runs at 300MHz.

Case study: avionics: we assume a hard $0.5s$ limit for hardware run time. Figure 4 shows estimated run times versus number of statistical assertions for both software and hardware implementations. We assume the design is limited by the bandwidth between software and hardware (MAX3 has 2GB/s maximum speed); stream length is 2^{26}, each output is 4 bytes wide, so the run time with no assertions is 0.15 seconds. Software calculations are limited to two assertions within the time limit, because all 2^{26} values must be streamed across the bus for each exception. In contrast, the C-slowed design summarizes 2^{26} data to 128 values,

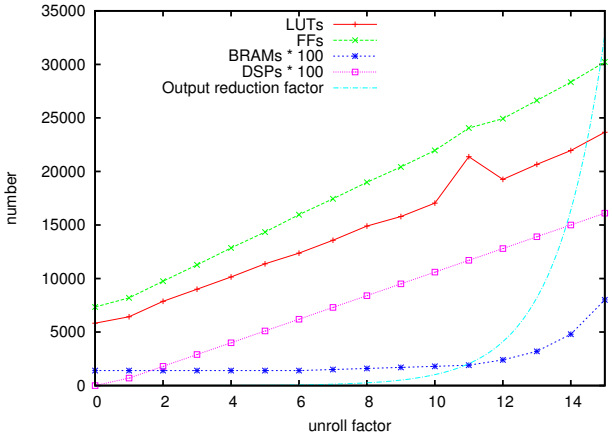


Fig. 3. Area usage and output reduction versus unroll factor for systolic variance operator.

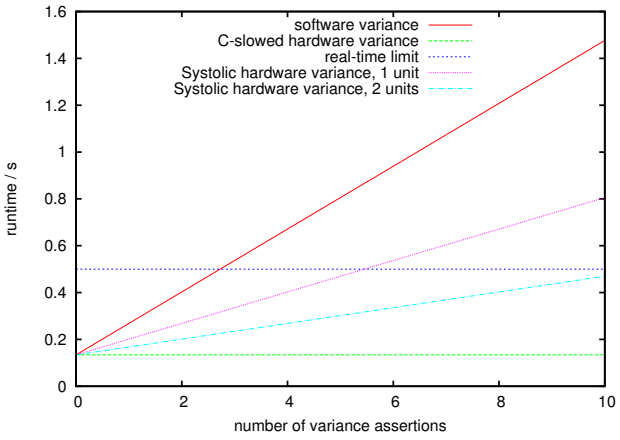


Fig. 4. Estimated time taken by software and C-slowed hardware variance assertions versus number of assertions.

meaning the time cost of each exception is much lower. Systolic hardware designs allow the number of assertions to be traded for hardware area. Note we do not include time to calculate the variance on the host.

VI. CONCLUSION

To allow efficient monitoring for self-adaptive systems, we design and implement in-circuit statistical assertions, allowing designs to use several frequently-occurring statistical operators to express desired runtime properties of design inputs, outputs and internal signals. Results show that response time can be greatly reduced at a modest cost in hardware area per exception.

Current and future work includes enlarging the set of statistical primitives to allow more general assertions on the state of the design. We would also like to explore the interaction of the statistical operators with run-time

reconfiguration. Statistical conditions can be used to decide when to reconfigure. More generally, the statistics operators themselves can be reconfigured, allowing the system to alter the balance of configurable hardware between assertions and computation depending on runtime conditions.

Acknowledgements: The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement number 257906.

VII. REFERENCES

[1] T. Reinbacher, M. Függer, and J. Brauer, "Real-time runtime verification on chip," in *Runtime Verification*, ser. Lecture Notes in Computer Science, S. Qadeer and S. Tasiran, Eds. Springer Berlin Heidelberg, 2013, vol. 7687, pp. 110–125.

[2] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Communications of the ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[3] S. Vasudevan, "What is assertion-based verification?" *SIGDA E-News*, vol. 42, no. 12, December 2012.

[4] J. Curreri, G. Stitt, and A. D. George, "High-level synthesis of in-circuit assertions for verification, debugging, and timing analysis," *International Journal of Reconfigurable Computing*, vol. 2011, 2011.

[5] M. N. Dinh, D. Abramson, J. Chao, D. Kurniawan, A. Gontarek, B. Moench, and L. DeRose, "Debugging scientific applications with statistical assertions," *Procedia Computer Science*, vol. 9, no. 0, pp. 1940 – 1949, 2012, proceedings of the International Conference on Computational Science, (ICCS) 2012.

[6] D. E. Knuth, *The Art of Computer Programming, volume 2: Seminumerical Algorithms*, 3rd ed. Addison-Wesley, 1998.

[7] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. pp. 419–420, 1962.

[8] T. F. Chan, G. H. Golub, and R. J. LeVeque, "Updating formulae and a pairwise algorithm for computing sample variances," Department of Computer Science, School of Humanities and Sciences, Stanford University, Tech. Rep. STAN-CS-79-773, November 1979.

[9] R. P. G. Collinson, *Introduction to avionics systems*, 2nd ed. Kluwer, 2003.

DESIGNING SELF-ADAPTIVE SMART SPACES FOR ENERGY SAVING

Alessandro A. Nacci, Christian Pilato, Marco D. Santambrogio, Donatella Sciuto

Politecnico di Milano – Dip. di Elettronica, Informazione e Bioingegneria
P.zza Leonardo da Vinci, 32 - 20133 - Milano (Italy)
nacci@elet.polimi.it, {christian.pilato,marco.santambrogio,donatella.sciuto}@polimi.it

ABSTRACT

This paper presents a SystemC-based framework for the simulation of smart spaces that focuses on energy aspects. Indeed, an efficient energy management plays a key role for the global reduction of carbon emissions. For this reason, the design of energy-aware smart spaces, able to self adapt to the surrounding environment, is becoming more and more interesting from the research point of view. However, it presents several challenges for their design, mainly due to the complexity of these complex and distributed systems and their cooperative behavior. This framework thus aims at facilitating the design of such complex cyber-physical systems.

1. INTRODUCTION AND RELATED WORK

Energy sustainability is becoming a central point in research to move towards a sustainable planet and research frameworks, like European Union Horizon 2020 [1], are currently encouraging to develop solutions to pursue these objectives. In this context, in both academia and industry, there is a growing interest for *smart spaces* [2]: they are environments such as apartments, offices, museums, hospitals, schools, malls, university campuses, and outdoor areas that are enabled for the co-operation of objects (e.g, sensors, devices, appliances) and systems that have the capability to self organize themselves, based on given policies.

Thanks to their main characteristics, these cyber-physical systems [3] are self-aware, heterogeneous and distributed for an efficient management of the energy consumption. Indeed, to realize these smart spaces, dozens of distributed computation, perception and actuation modules are usually adopted [4]. Sensors will gather a huge quantity of information that must be elaborated from the computation nodes and then efficiently transmitted to the actuators. For these reasons, efficient hardware infrastructures, software systems and design flows are in great needs. Dedicated techniques for their design, implementation and validation are definitively required, especially by means of simulation methodologies to analyze how the components affect each other and the interaction with the external physical environment, especially when considering energy aspects.

Even if there exist many approaches for the energy management of smart spaces, also the evaluation of the dynamic policies would be very attractive to self adapt the system to the surrounding environment. In such a scenario, system-level modeling through SystemC and TLM [11] has been demonstrated [12, 13] a viable solution to easily describe an overall system architecture and to simulate the computation of big amounts of data. On the other hand, an example of simulation environment can be found in [14] where a time-domain simulation environment for smart spaces with probabilistic appliance events is presented, but the appliance behavior is only represented as a Poisson random variable. In [15] the *Bit-Watt system* is presented to evaluate the performance of energy management in a home environment by computer simulations. However, the physical behavior of the surrounding environment is not actually simulated, potentially limiting the analysis that the designer can perform. Another simulator is presented in [16] to coordinate the execution of the residence appliances to minimize system cost assuming a time-of-use electricity rate structure.

In conclusion, to the best of our knowledge, it does not exist any modular and extensible framework that allows to efficiently design the systems (e.g., also integrating models of newer appliances) and evaluate them (and their cooperation) at different levels of abstraction, including the interaction with the physical environment.

This paper thus introduces *EA-SIM*, a novel and open-source framework for designing, simulating and validating self-adaptive and energy-aware smart spaces. EA-SIM is based on SystemC TLM-2.0 and it allows a functional validation of the generated systems and also the analysis of non-functional properties, such as performance and power consumption, at different levels of abstraction. Moreover, it can integrate the simulation of the physical environment (e.g., thermal trends, power consumption) to validate the decisions taken by the control units with respect to the information gathered by the sensors. *EA-SIM* offers an extensible component library that allows to model the different elements of a smart space: the control units, the computational nodes, the appliances, along with their sensors and actuators. Moreover, it is also possible to associate local batteries with each appliance and control their usage to reduce peaks.

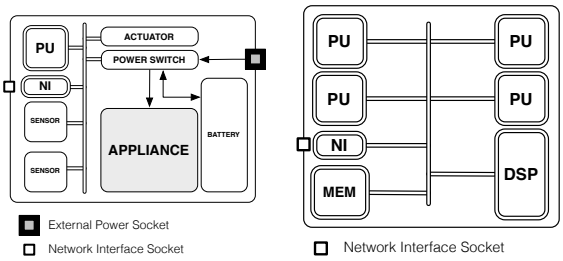


Fig. 1. High-level organization of the smart appliance (left) and the energy box (right).

2. EA-SIM SIMULATION FRAMEWORK

An energy-aware system can be defined as a context-aware architecture that can sense its physical environment, and adapt its behavior accordingly. Therefore, in order to simulate such kind of systems, it is necessary to create a framework able to evaluate the concurrent activity of all the components (i.e. the nodes), how they exchange the data and how they consume energy with respect to the users' requests and the policies defined by the control unit. For this reason, we propose *EA-SIM*, a simulation framework based on SystemC TLM-2.0 [11] that allows to easily evaluate the behavior of an energy-aware smart space. In particular, different architectural solutions can be evaluated, as well as different policies to control the dynamic behavior of the overall system. EA-SIM is an under development framework that allows to design and to simulate energy-aware smart spaces that are represented as composed of *power generator nodes*, *appliance nodes* and *energy boxes*. It should be noticed that this representation is flexible enough to represent a great variety of different spaces but simple to use at the same time.

A **power generator** is a component that produces electric current as, for example, the one representing the incoming current from the national power system. Also **accumulators** (batteries) are modeled in the proposed simulation framework. The use of batteries in a grid of appliances has been proved to be effective [9]. In fact, supplying energy from accumulators during peak electricity consumption times allows lowering peak demands and reduce both energy costs and carbon emissions. Furthermore, they can be used to compensate for the variability of typical renewable resources (e.g., wind, wave, solar), thus making the integration of such facilities more viable in practice [17].

In our view, two different appliances can be inserted in the simulation: standard and "smart" appliances. A simple **standard appliance** does not provide any sensors, actuators or digital interface to customize the behavior that can be modeled through mathematical models (including electro-mechanical behavior) or through the profiling of actual users' experience. A **smart appliance** is a more sophisticated appliance, also providing a socket interface to receive commands. It can have sensors to read the current status of the appliance and actuators to modify the status. The representation of such smart applicances is shown on the left side of Figure 1. It is thus possible to lower peak demand and reduce both energy costs and carbon emissions. Furthermore, storage devices can be used to compensate for the variability of typical renewable electricity generation (e.g., wind, wave, solar) and supporting their practical integration [17]. The *power switch* is used to select which is the the current power supply: the battery or the external power generator. A *processing unit* (PU) coordinates all the other elements: this can be connected to a *Network Interface* (NI - to communicate with an external energy box), sensors and actuators through a bus or similar communication infrastructures. The power switch is then responsible for the selection of the right power source for the appliance. The selection is performed by the PU and it is based on the policies suggested by the external *energy box*. Three situations are envisioned in a smart appliance: (1) the appliance is powered through the external power socket and the battery is not recharging; (2) the appliance is powered through the external power socket and the battery is recharging; (3) the appliance is powered through the internal battery. The processing unit is instead the module responsible for the coordination of the entire smart appliance.

The **energy box** nodes (as shown on the right side of Figure 1) are adopted to monitor and manage all installed appliances, generators and accumulators (batteries). They are designed to potentially elaborate a large amount of sensing data and implement dynamic control policies. For such reason, an energy box is an embedded and heterogeneous multicore architecture. Also a memory module can be available in order to store temporary data. A network interface is responsible for the connection with the other nodes. Moreover, a energy box can be connected to the NI to provide a remote interface to access features exported by the appliances. It will also build profiles inferred from the behavior of connected devices, for example to recommend and actuate energy conservation policies.

3. CASE STUDY

This section presents two case studies as preliminary examples to show the validity of the proposed simulation framework. Figure 2 shows the corresponding architecture. This architecture contains one *power generator* and one *energy box* to control the energy consumption of two *appliances*. Then, only one of them is "smart", i.e., it has been enhanced with a *power switch*, a *battery* and finally a *CPU* to implement the policies decided by the *energy box*. To show the effectiveness of the proposed framework, different situations have been evaluated. In all these cases, the charge of the battery is always maintained between 30% and 60%, as soon as it enters in this interval. Moreover, the *energy box* imposes

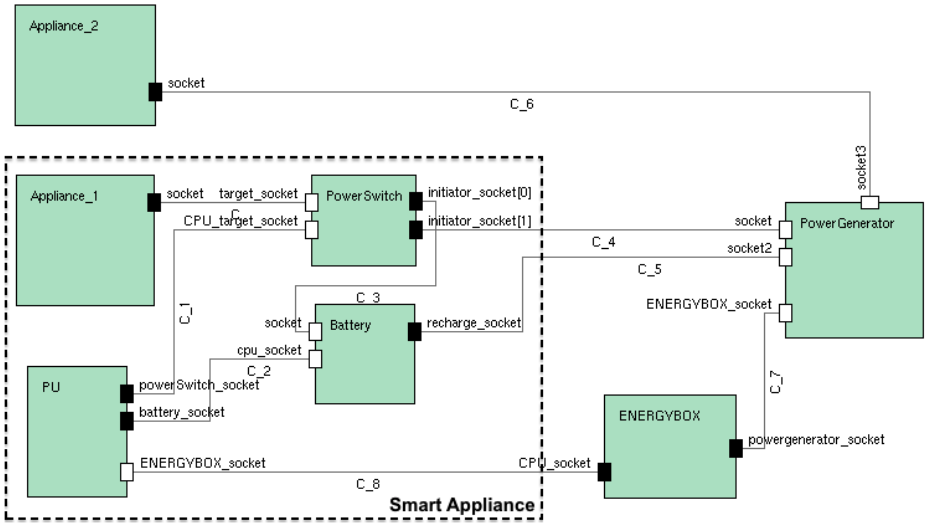


Fig. 2. Schema of the simulated architecture.

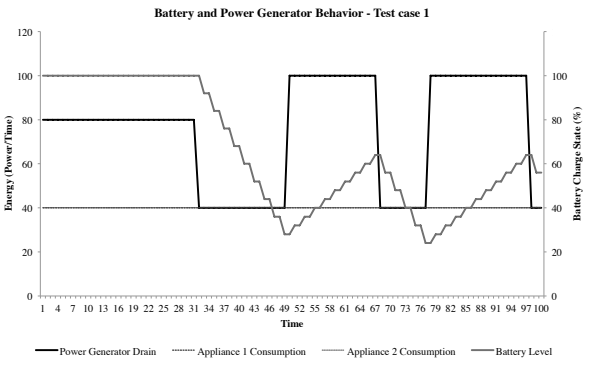


Fig. 3. Results of experiment #1: two appliances with constant energy consumption, where one of them is equipped with a battery.

the *smart appliance* to use the *power generator* instead of the *battery* for the first 30*ut*s in the first example and for 20*ut* in the other ones. All the experiments have been conducted through Synopsys Platform Architect [18] that offers a GUI to easily connect all the components of the system, execute the simulations and analyze the results.

In order to better understand the meaning of the case study, it must be known that using EA-SIM it is possible to set the desired units of measurement for both the energy consumption and the time. In fact, based on the appliances under analysis, the designer can configure the system to simulate the power, for instance, as *mW* or *W*. Then, based on the granularity of the dynamic behavior to be simulated, the evolution of time can be represented, for instance, in *seconds* or *minutes*. As a consequence, since in this paper we are only interested in analyzing the dynamic behavior of the energy-aware system not from a quantitative but from a qualitative point of view, in the rest of this work, the quantity of energy is indicated in *unit of energy* (*ue*) and the time is represented in *unit of time* (*ut*).

In the first experiment, the two appliances have a constant consumption of 40*ue/ut* and the battery has a consumption of 20*ue/ut* when it recharges. As shown in Figure 3, for the first 30*ut*, the two appliances are powered through the generator that thus consumes 80*ue/ut* (40*ue/ut* for each of the two appliances). After 30*ut*, the energy box switches the power source of the smart appliance to the battery and, as it is possible to see, the battery starts to discharge and the consumption of the power generator drops down to 40*ue/ut* (since one appliance is still powered using the generator). As soon the battery charge level is less than 30%, the energy box decides to switch again the smart appliance to the generator. The energy provided by the generator is now 100*ue/ut* since it has to power two appliances (40*ue/ut* for each of the 2 appliances) and the battery (40*ue/ut*).

Let us consider another experiment. While the the behavior of the battery is the same as before (it consumes 20*ue/ut* during the recharge phase), in this case the standard appliance (i.e., the one that is directly connected to the power generator) has a different dynamic. In fact, it has a step-based energy consumption, instead of a constant one. More precisely the step function has a value of 10*ue/ut* in the intervals (0;10) and (40;100) and 60*ue/ut* in the interval (10;40). In this case, the behavior of the power generator becomes really interesting, as shown from its curve in Figure 4. In fact, in the first 10*ut*, the drained energy is 50*ue/ut* since the two appliances are consuming 10*ue/ut* and 40 *ue/ut*, respectively, and the battery is not used (due to the pre-defined policy of the energy box). In the interval (10;20), the battery

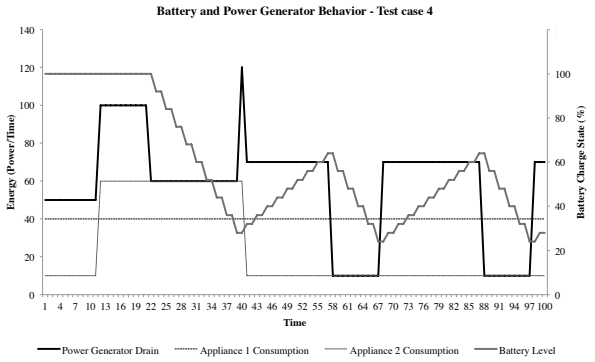


Fig. 4. Results of experiment #2: the energy consumption of one of the appliances is characterized by a step.

is then activated and the curve of the power generator drops down to 60*ue/ut* (i.e., the consumption of the standard appliance). An interesting phenomenon occurs at 10*ut*, where it is possible to find a peak of 120*ue/ut*: this value is due the fact that in that moment the battery starts to recharge (20*ue/ut*), the smart appliance is not using the battery since it is in charging (40*ue/ut*) and the standard appliance is still draining 60*ue/ut*. The other important interval that must be considered is the one between 57*ut* and 67*ut*: here the battery is in use and consequently the power generator has to take care only of the standard appliance (10*ue/ut*). These experiments demonstrated that the proposed EA-SIM framework is effectively able to simulate complex behaviors of interacting appliances in a smart environment, where the designer can evaluate simple yet effective policies to control the energy consumption of the system. For instance, it is possible to use EA-SIM to explore different solutions to mitigate peaks in the architecture, e.g., by changing the battery policies or evaluating the behavior of different possible components.

4. CONCLUSIONS AND FUTURE WORK

We presented EA-SIM, a modular, open-source and extensible framework for supporting the design and the validation of energy-aware smart spaces, allowing the easy evaluation of different aspects at both hardware and software levels.

Future research is oriented to design efficient system architectures and policies for supporting energy sustainability of existing spaces. Moreover, novel guidelines will be also investigated to suggest the design of future smart spaces.

Acknowledgments

The authors would thank Synopsys, Inc. that kindly provided the tools to support the design of the library components and to perform the system-level simulations.

5. REFERENCES

[1] "European Commision Research and Innovation - horizon 2020," http://ec.europa.eu/research/horizon2020/, accessed: 26/03/2013.

[2] "European Institute for Innovation and Technology ICT Labs," http://www.eitictlabs.eu/innovation-areas/smart-spaces/, accessed: 26/03/2013.

[3] E. A. Lee, "Cyber physical systems: Design challenges," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-8, Jan 2008.

[4] W. Xie, Y. Shi, G. Xu, and Y. Mao, "Smart Platform - a software infrastructure for Smart Space (SISS)," in *Proceedings of ICMI*, 2002, pp. 429–434.

[5] G. Wood and M. Newborough, "Dynamic energy-consumption indicators for domestic appliances: environment, behaviour and design," *Energy and Buildings*, vol. 35, no. 8, pp. 821 – 841, 2003.

[6] M. Erol-Kantarci and H. Mouftah, "Using wireless sensor networks for energy-aware homes in smart grids," in *Proceedings of ISCC*, 2010, pp. 456–458.

[7] "AlertMe - creating smart homes," http://www.alertme.com, accessed: 12/04/2013.

[8] "CurrentCost - save money and cut your electricity waste," http://www.currentcost.com, accessed: 12/04/2013.

[9] A. Mohd, E. Ortjohann, A. Schmelter, N. Hamsic, and D. Morton, "Challenges in integrating distributed energy storage systems into future smart grid," in *Proceedings of ISIE*, 2008, pp. 1627–1632.

[10] N. Wade, P. Taylor, P. Lang, and P. Jones, "Evaluating the benefits of an electrical energy storage system in a future smart grid," *Energy Policy*, vol. 38, no. 11, pp. 7180 – 7188, 2010.

[11] Accelera Systems Initiative, "http://www.accellera.org."

[12] M. Damm, J. Moreno, J. Haase, and C. Grimm, "Using transaction level modeling techniques for wireless sensor network simulation," in *Proceedings of DATE*, 2010, pp. 1047–1052.

[13] F. Fummi, G. Perbellini, D. Quaglia, and A. Acquaviva, "Flexible energy-aware simulation of heterogenous wireless sensor networks," in *Proceedings of DATE*, 2009, pp. 1638–1643.

[14] A. Roscoe and G. Ault, "Supporting high penetrations of renewable generation via implementation of real-time electricity pricing and demand response," *Renewable Power Generation, IET*, vol. 4, no. 4, pp. 369–382, July.

[15] R. Teng and T. Yamazaki, "Bit-watt home system with hybrid power supply," in *Proceedings of ICCAE*, 2010, pp. 59–63.

[16] N. Gudi, L. Wang, V. Devabhaktuni, and S. Depuru, "Demand response simulation implementing heuristic optimization for home energy management," in *Proceedings of NAPS*, 2010, pp. 1–6.

[17] P. Vytelingum, T. D. Voice, S. D. Ramchurn, A. Rogers, and N. R. Jennings, "Agent-based micro-storage management for the smart grid," in *Proceedings of AAMAS*, 2010, pp. 39–46.

[18] Synopsys, Inc., "http://www.synopsys.com."