Hardware Implementations of Evolvable Systems
A critical analysis on self-adaptive autonomic systems on reconfigurable architectures

Imara Speek
Aimee Ferouge
Delft University of Technology
November 4, 2013

Abstract

Morbi tempor congue porta. Proin semper, leo vitae faucibus dictum, metus mauris lacinia lorem, ac congue leo felis eu turpis. Sed nec nunc pellentesque, gravida eros a. lobortis ultrices eget ac metus. In tempus hendrerit rhoncus. Mauris dignissim turpis id sollicitudin lacinia. Praesent libero tellus, fringilla nec ullamcorper at, ultrices i nulla. Phasellus placerat a tellus a malesuada.

Keywords: bio-inspired , self-aware , computing systems, machine learning

Blabla

1 Introduction

Introduction [5] blabla [**?**]

2 Prior knowledge

[still need to give an introduction to all the matters spoken. what is evolvable hardware?] [firstly introduce the self properties as they are mentioned right into the discussion]*

Evolvable systems exploiting self-adaptive techniques are self-configuring and self-optimizing systems capable of changing their operations to meet the given performance goals by modifying either the underlying heterogeneous architecture, the operating system and the self-adaptive applications. [4] Meeting efficiency and accuracy constraints is getting more and more difficult , mainly because of the exponential increase of interactions among systems and the environments in which the systems are required to work.

The monitoring process is central to self-adaptive systems and is done by the application HeartBeats. Using machine learning, one can enhance synchronization techniques. K32 is a modern object-oriented operating system that responds to changing and challenging environments adapting its operations. It supports on line reconfiguration of functionalities by interposition, hot-swap and dynamic update. [4]

The autonomic concept is inpired by the human body's nervous system. This system is

known to be adapt to environment aware and capable of adapting to this, but also be able to repair minor physical damage. An autonomic system is stated to contain all these properties, also known as self-properties: *self-managing, self-protecting, self-healing* and *self-optimizing* [9].

3 Discussion of the Different Papers

[introduction to discussions and seperation of the topics, maybe seperate hardware from software solution from these papers?] [Give some small introduction to each paper in 3 sentences and then maximum of 15 for every paper]

Considerable research has already been done in order to efficiently accelerate hardware while still maintaining virtually unlimited adaptability. Software techniques in autonomic computing systems such as hot-swapping and data clustering are discussed in [1]. These self-aware systems can adapt their behavior on FPGA-based system as discussed in [6]. Other approaches start with a FPGA-based architecture with a reconfigurable core [**?**], added programming schemes and new cell structures [3], [7] or even bio-inspired hardware using the POE-model [8]. More recent papers put effort a combined approach by either implementing autonomic systems on reconfigurable architectures [10] or creating evolvable systems via self-aware applications [4].

3.1 Software based flexible approaches

The results of [6] presents a system built on top of a set of enabling technology that proves the effectiveness of using self-aware adaptive computing systems. Self-aware adaptive computing focuses on creating a balance of resources to improve performance, utilization, reliability and programmability. A programmer will ideally only have to provide the system its goal, rather than a description of tasks, provided with some constraints.

In a system like this, the hardware, the application and the operating system have to be seen as unique entities to autonomously adapt itself. The underlying architecture has cognitive hardware mechanisms in its core to observe and affect the execution. The self-aware adaptive computing system also implements learning and decision making engines to determine appropriate actions. A key challenge is to identify what parts of a computer need to be adapted.

3.2 Hardware based fast approaches

FPGAs are a commonly used device for implementing the reconfigurable architecture of evolvable systems,especially Virtex FPGAs produced by Xilinx. Unfortunately, these devices have some drawbacks. The Erlangen Slot Machine from [7] tackles the three major limitations of the Virtex-II FPGA. A pipelined data flow architecture has been used, replacing the finite state machine by a MicroBlaze microcontroller and employing a data crossbar between plug-ins. Providing a new architecture to avoid the current physical problems of reconfigurable FPGAs, a new inter-module communication concept, as well as an intelligent module reconfiguration management has made the ESM an alternative for the Virtex FPGAs.

In [3] and [2] a Virtex-4 FPGA implementation is introduced for evolvable systems. Other then earlier versions of Virtex (e.g. the Virtex-II Pro), Virtex4 devices enable two-dimensional dynamic reconfiguration, a feature which considerably reduces the reconfiguration time an thus the evolution time ([3]). A big limitations of Virtex FPGAs is an almost unknown and undocumented bitstream data format and a

unsafe configurations schema. By using both VRCs (Virtual Reconfigurable Circuits) and direct bitstream manipulation, this architecture eliminates this limitation. This Virtex-4 based device, which takes advantage of 2D reconfiguration capabilities and direct manipulation of the bitstreams is the first one of its kind and will be discussed later.

A second Virtex-4 based architecture introduced in [2] also applies a 2D reconfiguration core. Rather than direct bitstream manipulation, an optimized Dynamically and Partial Reconfiguration (DPR) control engine has been integrated. As a result, the processing elements (PEs) of the reconfigurable core are structured as a 2D systolic array, known for its high performance and restrained use of resources. Also, the reconfiguration engine has been optimized by applying three enhancements that will be discussed in 4.3.3.

3.3 Co-design based systems

In [10] the importance of hardware/software co-design during design space exploration is urged on. Right now this co-design emphasizes on identifying intensive kernel tasks and implementing these tasks on the reconfigurable hardware. Existing adaptive systems often fail because of they are largely ad hoc and fail to incorporate true goals. The current performance model of the hardware however, depends on the degree of parallelism while the performance model for software execution is static and does not become affected by external factors. Since the introduction of reconfigurable hardware platforms such as FPGAs, the hardware domain shifted into the software domain: the possibility of implementing a reconfigurable architecture increased the flexibility of the hardware.

Partial dynamic reconfiguration is a key feature that makes FPGAs unique. This addresses the lack of resources to implement an application and its adaptability needs. Reconfigurable hardware taking advantage of partial dynamic reconfiguration is the perfect trade-off between the speed of HW and the flexibility of SW. Other aspects that motivate the use of online self-adaptable systems are the QoS and the reliability and continuity of the service.

However, an important problem often neglected is the time overhead the reconfiguration process introduces and the two-dimensional partitioning strategy reconfigurable devices need: spatial and temporal. [10] presents the urge to carefully evaluate the overhead created as a negative impact of reconfiguration latency which is not always discussed in present papers. In [4] an evolvable system running self-adaptive application on top of a heterogeneous systems is proposed. The operating system running on top is responsible for providing the self-* proporties and runs this on a general purpose processor and a reconfigurable device

In [4], the underlying hardware architecture is made up of static area and a dynamic area. The reconfigurable device can be configured to implement different functionality through dynamic reconfiguration support provided by the operating system. This is provided through standard libraries and the OS implements parts of the loop. The OS is therefor capable of choosing at run-time the best implementation for the required functionality among the available. The switchable units in the hot-swap method in [4] are identified as the libraries that export an implementation of a certain functionality. The self-adaptive library or Dynamic-link library(DLL) and the software implementation library target the reconfigurable FPGA and multicore respectively.

4 Analysis of techniques

This section provides an overview of techniques that are used in current research to create evolv-

able hardware implementations of evolvable systems. To create such a system, the system has to be self-aware, able to make decisions, self-adapting and the hardware has to support a certain amount of reconfiguration. In section 5 the overall best combination will be presented, utlizing the flexibility of software and the speed of hardware.

4.1 Self-awareness techniques

Self-aware adaptive computing systems are capable of adapting their behavior and resources thousands of times based on changing environmental conditions and demands [6]. This allows them to automatically accomplish their goals in the best way possible. Autonomic systems are build up of autonomic elements. These elements must include sensors and effectors [9] in order to monitor its behavior and in turn become self-aware. *Closed-control loops* are often incoorporated to monitor the behavior considering the given performance goals. In subsection 4.2 closed-control loops are further discussed.

The *Heartbeats application* is a monitoring application which makes it possible to assert performance goals as heart-rate windows delimited by a minimum and maximum performance, or *heart-rate* [4]. The Heartbeats API is made of small straightforward functions and allows declaring performance goals. Software components first have to register, specifying parameters such as minimum and maxmim heart-reate, size of the windows of observation and history buffers [6]. The application then updates the progress of the execution calling the function that signifies a heartbeat [4]. A framework like this is particularly convenient because it allows to automatically update all information about the global heart-rate which is then made available to external observers, such as decision making applcations. However it introduces an average overhead of 3,52% due to system calls in order to initialize its data structures and updating the global heart rate [6].

4.2 Decision making techniques

For a self-adaptive system the intelligence in using the monitoring data is equally important to collecting the data. Results from the monitoring framework are fed to a decision making application. Making decisions for autonomic systems is the second action in the closed-control loops such as the *observe-decide-act (ODA) loop*. Decision making all depends on the input and constraints given to the system. It can either concern a best-performance decision making or a sufficiently good performance that respects the user's performance goals [4]. Closed-control loops are implemented by the operating system reciding on the reconfigurable hardware. By using dynamic self-adaptive libraries, the operating system can choose at runtime which is the best suiting implementation for the required functionality among the available [4].

A desicioning framework can be divided into 2 categories: analytical models and empirical models [4]. The analytical models are good when working in the same environment as they are manually generated and are very precise because of this prior knowledge. However when considering evolvable systems in dynamic environments, empirical models are favourable as they exploit information collected at run-time, but are in turn less accurate. Current trends in supporting decision making are using heuristic policies, machine learning, control theory and competitive algorithms. Heuristics is the application of experience-derived knowledge to a problem. Heuristic software can be easily developed, applied and reused when working within a known environment including static condition. However within the context of a dynamic live application with changing environments, it is very easy to miss unforeseen problems using software that looks for known problems.

4.3 Self-adapting techniques

The final step of the closed-control loop is acting upon the decision that is made. Changing the operating of a evolvable system to meet specific performance goals can be done by modifying either the underlying heterogeneous architecture, the operating systems and the self-adaptive applications [4]. A fundamental part of a self-aware adaptive computing system is the ability to switch between implementations of the same functionality while the system is running. These implementation can either switch from software to hardware, running different applications, from hardware to software or considering different configurations. [6]

Criteria to choose between different implementations can be expected performance, available resources, input data type and size, the functionalities that are already implemented and the availability of hardware components. The *hot-swap mechanism* is a popular tool to implement self-configuration and self-optimization. It provides the ability of switching among different implementations of the same functionality in a transparent fashion. However state quiescence and state translation are two big issues when using this mechanism and need a framework solution to be reliable.

Other works present *Partial Dynamic Reconfiguration* [10] which only uses the already preconfigured elements to reduce the amount overhead introduces when setting up elements to be configured.

4.3.1 Virtex FPGA with a Two-Dimensional Reconfiguration Core

On Virtex-4 FPGAs, two-dimensional dynamic reconfiguration like figure 1 is supported. With 2D reconfiguration it becomes possible to reconfigure device portions whose height is not constrained to be the device height. This architecture of the reconfiguration core (or RC) speeds up the reconfiguration time and thus the evolution time. The RC can also deploy more candidate solutions as discussed in [3], which are arrays of bidimensional cell (see figure 4.3.2). An alternative for candidate solutions is a mesh-type systolic array of parallel processing elements (PEs) from [2], also following a 2D architecture for the RC. A major feature is the possibility to change the functionality of the PEs by means of Dynamic and Partial Reconfiguration (DPR). This gives the system the capability to adapt. The outputs of the PEs (east and south side) are connected to the close neighbour's input (west and north side), such that only the lowest and right-most PE has to be read for data output. This systolic approach of communication reduces the reconfiguration time and makes the architecture easy to extend.

A drawback of using Virtex FPGAs are the feed-through signales, mentioned in [7]. Each module must be implemented with all possible feed-through channels needed by other modules. Because we only know at run-time which module needs to feed through a signal, many channels reserved for a possible feed-through become redundant. Also, modules accessing external pins are no longer relocatable, because they are complied for fixed locations where a direct signal line to these pins is established.

4.3.2 Candidate Solutions and Direct Bitstream Manipulation

With a 2D architecture for the RC, [3] uses safe manipulation of the bitstream to overcome the unknown and undocumented bitstream mentioned in 3.2. Candidate solutions are used to fill the RC with cells, which have an internal flip-flops allowing the evolution of synchronous circuits. This is a common structure for evolvable hardware (EHW) systems making use of direct bitstream manipulation [3]. For the cell

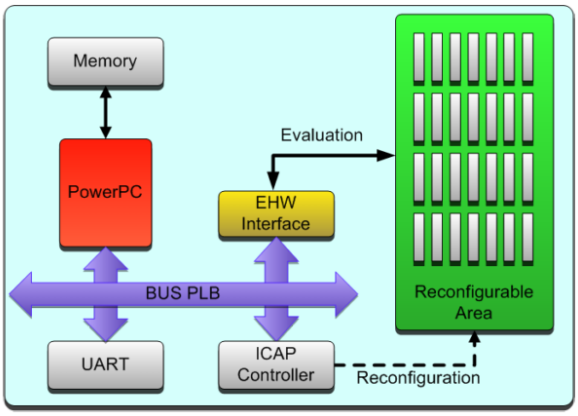

Figure 1: High-level structure of a Two-Dimensional Architecture depicted in [3]

structure of the candidate solutions the use of LUTs and a MUX is proposed, in order to provide direct bitstream manipulation. Combined with the 2D-reconfiguration mechanism, the new architecture causes a speed up of 16x factor. For this system, only Virtex-4 or Virtex-5 FPGAs can be used since Virtex-II does not support 2D reconfiguation ([3].

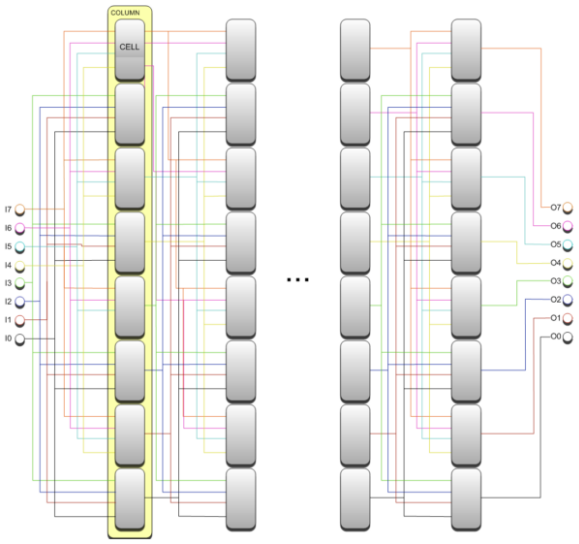

Figure 2: Internal structure of a Candidate Solution proposed in [3]

4.3.3 Systolic array of PEs and Optimized DPR

Other than [3], [2] uses a systolic array of PEs. In this approach, each PE is a basic computation unit able to perform a single operation on the data take from their close neighbors. The architecture can be easily extended to any other processing purposes, since new PEs can be added to the library. In addition, PEs included in this library can be reused among applications. As can be seen in fig. 3, the size of the implemented structure is 4x4, but it can be completely and easily scaled. This DPR with elements relocation is carried out using a special HW block, the reconfiguration engine (RE).

[2] describes an implementation of this RE. By storing only the body of the bitstream (cutting of the header and the tail), overclocking the FPGA by 2,5x and including internal memory (to avoid pasting the same configuration module in different positions of the architecture) the DPR is optimized. Due to this the reconfiguration time is greatly reduced. Adding the header and the tail of the bitstream at runtime has two additional advantages: it is allowed having a unique bitstream for each PE that can be configured in any position of the array. Also, bitstream reduction reduces the data transference time from the external memory.

4.3.4 Erlangen Slot Machine

The architecture of [7] the Erlangen Slot Machine (ESM) deals with the three drawbacks of FPGAs mentioned in 4.3.1, being fixed pins which are spread around the device (I/O dilemma0, the inter-module dilemma and the local memory dilemma.

First, the I/O dilemma caused by fixed pins spread around the device is solved by connecting all bottom pins from the FPGA to an interface controller realizing a crossbar, as can

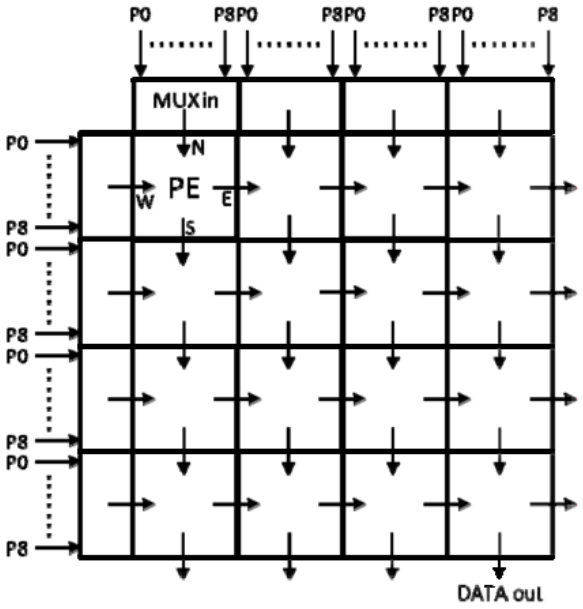

Figure 3: Systolic array of PEs introduced by [2]

be seen in figure 4. It connects FPGA pins to peripherals automatically based on the slot position of a placed module. This I/O rerouting principle is done without reconfiguration of the crossbar FPGA.

Second, the memory dilemma has been solved. In normal Virtex-II FPGAs, a module can only occupy the memory inside its physical slot boundary. Storing data in off-chip memories is therefore the only solution. In the ESM, six SRAM banks are connected to the FPGA. Since these banks are placed at the opposite side as the crossbar, a module will connect to peripherals from one side, while the other side will be used for temporally storing computational data. In order to use a SRAM bank (called a slot), the module must have at least a width of three micro-slots, in which the total device is divided (see 4). This organization simplifies relocation, enabling a partially reconfigurable computing system. Also, equal resources will be available for each module.

Finally, the inter-module communication dilemma is dealt with. Dynamically routing signal lines on the hardware is a very difficult task. The ESM uses a combination of bus-macros, shared memory, RMB (Reconfigurable Multiple Bus) and a crossbar to take away the limiting factor for the wide use of partial dynamic reconfiguration ([7]).

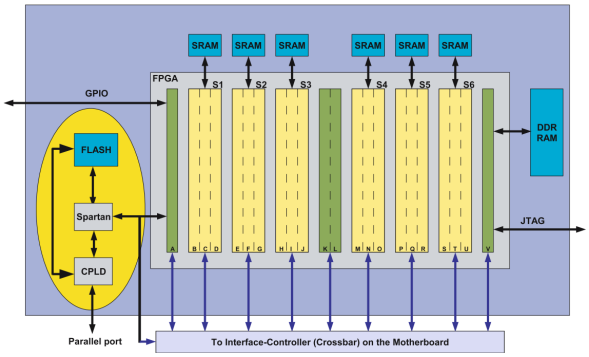

Figure 4: Architecture of the ESM board. Three consecutive micro-slots define a macro-slot, which can access one full external SRAM bank.

5 Proposed adaptive system design

* set up of a proposed system *

6 Related works

a presentation of related combination of techniques presented in previous paper [I think the POE model paper should be mentioned here instead of Discussion, since its not a HW implementation]

Inspired by life on Earth and the natural processes of living organisms, *bio-inspired* hardware systems have evolved. [8] introduces the POE model, that classifies bio-inspired systems according to three axes:

- Phylogeny, where evolvable hardware can be found.
- Ontogeny, where systems have the ability to self-repair as the main goal of ontogeny is (cell) growth or construction.
- Epigenesis, which is more software-based.

By looking at these biological phenomena, researchers can be inspired when developing evolvable hardware systems.

* present 3 tier architecture of [4] where the observe-decide-act loop resides in the self-adaptive libraries *

The K42 object oriented operating system created by IBM supports online reconfiguration [4]

Smartlocks is a spinlock library that adapts its internal implementation during execution using heuristics and machine learning. It optimizes towards a user-defined goal, programmed using the application heartbeats framework, which may relate to performance, power, problem-specific criteria or combinations. [10]

7 Conclusion

Comparison

References

[1] M. Khan A. Khalid, M. Haye and S. Shamail. Survey of frameworks, architectures and techniques in autonomic computing. In *Fifth International Conference on Autonomic and Autonomous Systems (ICAS)*.

[2] J. Mora A. Oter, R. Salvador and L. Sekanina. A fast reconfigurable 2d hw core architecture on fpgas for evolvable self-adaptive systems. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*.

[3] M. Santambrogio F. Cancare and D. Sciuto. A direct bitstream manipulation approach for virtex4-based evolvable systems. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*.

[4] H. Hoffman F. Sironi, A. Cuoccio and M.D. Santambrogio M. Maggio. Evolvable systems on reconfigurable architecture via self-aware adaptive applications. In *NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2011)*.

[5] H. Hoffmann et al. F. Sironi, M. Triverio. Self-aware adaptation in fpga-based systems. In *International Conference on Field Programmable Logic and Applications (FPL)*.

[6] H. Hoffmann M. Maggio M.D. Santambrogio F. Sironi, M. Triverio. Self-aware adaptation in fpga-based systems. In *2010 International Conference on Field Programmable Logic and Applications*.

[7] A. Ahmadinia M. Majer, J. Teich and C. Bobda. The erlangen slot machine: A dynamically reconfigurable fpga-based computer. *Journal of VSLI Signal Processing*.

[8] D. Mange et al. M. Sipper, E. Sanchez. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, August 2002.

[9] D R. Sterritt and Bustard. Towards an autonomic computing environment.

[10] M.D. Santambrogio. From reconfigurable architectures to self-adaptive autonomic systems. In *Int. J. Embedded Systems*.