From reconfigurable architectures to self-adaptive autonomic systems

2009 International Conference on Computational Science and Engineering

From reconfigurable architectures to self-adaptive autonomic systemss

Marco D. Santambrogio
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139
santambr@mit.edu

***Abstract*—Systems on a Chip (SoC) can draw various benefits such as adaptability and efficient acceleration of compute-intensive tasks from the inclusion of reconfigurable hardware as a system component. Dynamic reconfiguration capabilities of current reconfigurable devices create an additional dimension in the temporal domain. During the design space exploration phase, overheads associated with reconfiguration and hardware/software interfacing need to be evaluated carefully in order to harvest the full potential of dynamic reconfiguration. In order to overcome the limits deriving by the increasing complexity and the associated workload to maintain such complex infrastructure, one possibility is to adopt self-adaptive and autonomic computing systems [1]. A self-adaptive and autonomic computing system is a system able to configure, heal, optimize and protect itself without the need for human intervention.**

***Index Terms*—Performance, Reconfiguration, Codesign, Runtime Adaptability, Self-Adaptive Systems.**

I. Introduction

New application domains demand ever increasing adaptability and performance [2]–[4]. In order to cope with changing user requirements, improvements in system features, changing protocol [5] and data-coding standards, and demands for support of a variety of different user applications [6], many emerging applications in communication, computing and consumer electronics demand that their functionality stays flexible after the system has been manufactured. Reconfigurable SoCs employing different microprocessor cores [7] and different types of reconfigurable fabrics are one attractive solution for these domains. The increasing prominence of reconfigurable devices within such systems requires HW/SW codesign for SoCs to address the trade-off between software execution and reconfigurable hardware acceleration.

Inclusion of hardware reconfigurability allowed a deeper exploration of the design space, thanks to the ability to adapt the hardware part of the system in a simple and economical way. However, in order to harvest the true benefit from a system which employs dynamically reconfigurable hardware, codesign approaches need to pursue the best trade-off between hardware acceleration, communication cost, dynamic reconfiguration overhead, and system flexibility. In existing approaches to codesign, the emphasis is placed on identifying computationally intensive tasks, also called kernels, and then maximizing performance by implementing most of these tasks on reconfigurable hardware. In this scenario, software primarily performs the control dominated tasks. The performance model of the reconfigurable hardware is mainly defined by the degree of parallelism available in a given task and the amount of reconfiguration and communication cost that will be incurred. The performance model for software execution is on the other hand static and does not become affected by external factors.

Reconfiguration capabilities and hardware-software codesign techniques, are therefore becoming just elements of a more complex scenario. Nowadays information systems can be seen as aggregates of complex architectures, spanning from grids including thousands of geographically distributed systems dependent upon multi-site collaboration [8]–[10], and fast, reliable access to shared resources and data [11], to small and specialized embedded systems [7]. We achieved such a complexity in the infrastructure, no matter what kind of architecture it is based on, because of the increasing demanding for high performance computation, high reliability and for the need in providing always different, more complex, and high quality services to the users [12], [13]. Both in [10] and in [9] have been presented studies to adapt the applications workloads to the computational environment with respect to a given estimation of the performance of the grid. Adaptability is provided by migrating the operations among different components regarding the predicted time with respect to the gathered execution time using monitoring and analysis tools provided by other parts of their framework. Understanding the characteristics of the *users' workload* is an important aspect also when designing and providing web services [8]. Moreover, these complex, heterogenous and distributed systems can be characterized also by another factor: being able to provide service on-demand guaranteeing high system availability to properly support unpredictable workloads. Let's take into consideration the visa's transactions processing system which is routinely updated as many as $20,000$ times per year, yet tolerates less than 0.5% down-time [14].

The remaining of this paper is organized as follow. Section II describes the main characteristics and issues of nowadays reconfigurable systems, while Section III presents an overview of the literature where both hardware, not only FPGA-based

978-0-7695-3823-5/09 $26.00 © 2009 IEEE
DOI 10.1109/CSE.2009.490

architecture, and software components have been considered as reconfigurable elements. Section IV introduces the self-adaptive autonomic system concept showing how these systems belong and pervade different IT domains. Section V presents two examples of FPGA-based adaptable systems and finally, Section VI presents some concluding remarks.

II. Reconfigurable systems: characteristics and issues

Reconfigurable hardware is becoming a prominent component in a large variety of SoC designs. Reconfigurability allows efficient hardware acceleration and virtually unlimited adaptability. On the other hand, overheads associated with reconfiguration and interfaces with the software component need to be evaluated carefully during the exploration phase. This section provides some insights in the FPGA-based reconfigurable architecture research area, trying to describe how it can be utilized and some of its drawbacks.

A. Reconfiguration support

In order to configure an FPGA[1] with the desired functionality, one or more bitstreams are needed. A bitstream is a binary file in which configuration information for a particular FPGA device is stored, that is where all the data to be copied on to the configuration SRAM cells, the configuration memory, are stored, along with the proper commands for controlling the chip functionalities. Therefore Virtex [15] devices, such as Virtex II Pro [16] and Virtex 4 [17], are configured by loading application specific data into their configuration memory. I will refer to the physical implementation on the FPGA of a given functionality, configured on the device using a bitstream, with the term *configured task*. On the Virtex FPGAs the configuration memory is segmented into frames. A frame represents the smallest unit of reconfiguration. According to the device, this element can span the entire length of the FPGA, such as in the Virtex II Pro context, or just part of it, as in Virtex 4 devices. The number of frames and the bits per frame are specific for each device family. The number of frames is proportional to CLB width. Bitstreams can be either partial or full. A full bitstream configures the whole configuration memory and is used for static design or at the beginning of the execution of a dynamic reconfiguration system, to define the initial state of SRAM cells. Partial bitstreams configure only a portion of the device and are one of the end products of any partial reconfiguration flow.

FPGAs provide different means for configuration, under the form of different interfaces to the configuration logic on the chip. There are several modes and interfaces to configure a specific FPGA family, among them the JTAG download cable (which is the method used in this work), the SelectMAP interface, for daisy-chaining the configuration process of multiple FPGAs, configuration loading from PROMs or compact flash cards, micro-controller-based configuration, an internal configuration access port (ICAP) [18] and so on, depending on the specific family. The ICAP provides an interface which can be used by the internal logic to reconfigure and read back the configuration memory. A set of configuration registers defines the state of this configuration logic at a given moment in time. Actual configuration data is first written by the bitstream into these registers and then copied by the configuration logic on the configuration SRAMs.

[1]From now on, whenever using the term FPGA, I will refer to the Xilinx FPGA devices.

B. Partial Dynamic Reconfiguration

Partial dynamic reconfiguration is one of the key features that makes FPGAs unique devices, offering degrees of freedom not available in other kind of technologies and in some cases pushing FPGA-based solutions towards the standard application platform e.g., *network controller* capable of handling the TCP and UDP protocols by exploiting partial reconfiguration [5], cryptographic system [3], mechatronic systems [4]. In particular, two important benefits can be achieved by exploiting partial dynamic reconfiguration on reconfigurable hardware: (1) the reconfigurable area can be exploited more efficiently with respect to a static design, (2) some portion of the application must change over time and react to changes in its environment.

The two main advantages given by a Partial Dynamic Reconfiguration (PDR) solution thus address the lack of resources needed to implement an application and its adaptability needs; it must be pointed out that both of the advantages could be replaced by having a larger resource array, where all of the functionalities could be implemented. This solution is not always viable for non-trivial designs and a PDR strategy must be implemented. Reconfigurable hardware taking advantage of partial dynamic reconfiguration can be thus seen as the middle point in the trade-off between the speed of HW solutions and the flexibility of SW.

C. Issues in PDR design methodologies

Besides the benefits introduced by the usage of partial dynamic reconfiguration, some issues remain in the design methodologies used to achieve the final reconfigurable architecture. The issue that is impairing a wider diffusion of such architectures is certainly the lack of a complete software toolchain capable of taking into account partial dynamic reconfiguration in a sound manner, even if few novel methodologies have been proposed [19]. It is true that this is not the only issue in working with reconfigurable architecture, i.e., using partial dynamic reconfiguration leads to the introduction of time overhead due to the reconfiguration process, but without strong theoretical studies and the definition of the corresponding design methodologies and developing frameworks, it will be impossible to effectively used such a concept in designing the next generation of computer architectures, and not only in the SoC context.

The current methodologies for Xilinx FPGAs [20] [21], as an example, comprise a long series of steps that the developer has to undertake in order to convert the product of conventional CAD tools into the final full and partial bitstreams necessary to

deploy a partial dynamic reconfigurable architecture. During the various steps, moreover, the system designer has to keep clearly in mind partial reconfiguration constraints and use the software to enforce them. Due to the relative novelty of partial reconfiguration techniques these procedures are not yet included into the manufacturer's software, and many operations have to be done manually. This factor may deviate the focus of the development process away from the real application towards these details, thus resulting in an extended time-to-market of the developed application. Another issue related to the lack of support for partial reconfiguration in the manufacturer's design flow is that some tools that make up the development flow are not reconfiguration aware.

III. Hardware and software as reconfigurable resources

In the context of reconfigurable systems, many approaches focused on effective utilization of the dynamically reconfigurable hardware resources. Related works in this domain focus on various aspects of partitioning and context scheduling. A system called NIMBLE was proposed for this task [22]. As an alternative to conventional ASICs, a reconfigurable datapath has been used in this system. The partitioning problem for architectures containing reconfigurable devices has different requirements. It demands a two dimensional partitioning strategy, in both spatial and temporal domains, while conventional architectures only involve spatial partitioning. The partitioning engine has to perform temporal partitioning as the FPGA can be reconfigured at various stages of the program execution in order to implement different functionalities. Noguera and Badira [23] proposed a design framework for dynamically reconfigurable systems, introducing a dynamic context scheduler and hw/sw partitioner. Banerjee et al. [24] introduced a partitioning scheme that is aware of the placement constraints during the context scheduling of the partially reconfigurable datapath of the SoC.

In [25] the authors propose a new methodology to allow the platforms to hot-swap application specific modules without disturbing the operation of the rest of the system. This goal is achieved through the use of partial dynamic reconfiguration. The application presented in that paper has been implemented onto a Xilinx Virtex-E FPGA. According to this, the proposed methodology finds its physical implementation as an external reconfiguration that implies that a Virtex-E active array may be partially reconfigured by an external device such as a Personal Computer, while ensuring the correct operation of those active circuits that are not being changed [26]. The reconfigurable modules are called Dynamic Hardware Plugin, DHP. The methodology proposed in [25] transforms standard bitfiles, computed by common computer aided design tools, into new partial bitstreams that represent the DHP modules due to the PARtial BItfile Transform tool, PARBIT, [27].

Two interesting examples of reconfigurable chip multiprocessor architecture have been presented in [7] and [28]. In [7], an heterogeneous multi-core architecture has been proposed in order to assign to the most appropriate core, in terms of power consumption, the execution of a specific part of an application, since different cores have varying energy efficiencies for the same workload. This architecture has been designed taking into consideration the fact that typical *programs go through phases* with different execution characteristics, which can lead to different workloads, therefore the most appropriate core during one phase may not be the right one for a following phase. In [28] the Core Fusion architecture is proposed. This architecture is characterized by the presence of different tiny independent cores that can be used as distinct processing elements or that can be fused into a bigger CPU based on the software demand.

Nowadays, different works have also focused their attention of the effective utilization of dynamically reconfigurable software resources [29], [30]. In [31], [32] the runtime implementation, hardware or software, of specific elements of the systems is taken online during the system execution. A hot-swap-based approach has been used to implement software reconfiguration in the K42 Operating System [29]. Aims of the work presented in [33], is to characterize and understand the interactions between hardware and software and to affect optimizations based on those characterizations. To achieve this, they have designed and implemented a performance and environment monitoring (PEM) infrastructure that vertically integrates performance events from various layers in the execution stack. The authors have developed an architecture for continuous program optimization (CPO) [34] to assist in, and automate the challenging task of performance tuning a system. CPO utilizes the data provided by PEM to detect, diagnose, and eliminate performance problems. Tackling the PEM and CPO together, it is possible to obtain an efficient monitoring system able to improve operating system availability [35] with dynamic update based on hot-swapable objects [36], [37].

IV. Towards the definition of self-adaptive and autonomic computing systems

In order to overcome the limits deriving by the increasing complexity and the associated workload to maintain such complex infrastructure, one possibility is to adopt self-adaptive and autonomic computing systems [38]. A self-adaptive and autonomic computing system is a system able to configure, heal, optimize and protect itself without the need for human intervention [1]. Different companies, i.e., IBM [39], Oracle, and Intel have invested a lot of their efforts in this research, realizing several products characterized by a self-adaptive behavior. Examples of these products are the Oracle Automatic Workload Repository [6], the IBM Touchpoint Simulator, the K42 Operating System [29] and the Intel RAS Technologies for Enterprise [2].

Within this context, a self-adaptive and autonomic computing system is no longer view as a static bunch of hardware components with a passive set of applications running on top of an operating system used to properly coordinate the underling hardware architecture. It becomes an active system where either the hardware, the applications and the operating

system have to be seen as an unique entity that have to be able to autonomously adapt itself to achieve the best performance[2]. In order to achieve such a scenario, the self-adaptive and autonomic computing systems have to be able to monitor its behavior to self update itself, in one, or in a combination of several, of its components (hardware architecture, operating system and running applications), to overcome possible failure in accomplishing its tasks. Most current architectures include at least basic hardware assists for system monitoring, usually in the form of counter registers. Counter-based techniques suffer common shortcomings [40]: too few counters, sampling delay, and lack of address profiling. Modern systems [41] try to address these deficiencies, however they still suffer the fact that they can be applied only to collect aggregate statistics using sampling. It is not possible to react to single events or to collect additional data. The Itanium processor family [42], overcomes also this limit introducing microarchitectural event data that have to be delivered to the consuming software through an exception for each event. This solution implies that the process using this information has to experience frequent interrupts.

In [43], the FPGA has been used as a sort of filter to monitor, using the dependability analysis, the data flowing through a certain part of the system. Unfortunately, this approach, even without introducing overhead into the computation, cannot be considered as non intrusive with respect to the overall system. A partial reconfiguration approach, due to the reconfiguration capabilities of modern FPGAs [44]–[46], has been proposed in [47] to implement an online adaptive system, able to update its underlying architectural implementation to optimize the power consumption. This is an interesting approach which is only proving that partial reconfiguration can be used to implement an online solution, but the runtime environment has not been realized nor information on how to monitor online its behavior has been outlined.

V. FPGA-BASED ADAPTABLE SYSTEMS: TWO EXAMPLES

A. Adaptive software and reconfigurable hardware

In [30], the authors identify the best trade-off considering application-specific features in software, which can lead itself to software-based acceleration and lead to a revision of the view that certain computationally intensive tasks can only be accelerated through hardware. The presented methodology addresses the problem of the identification of the software and the hardware components of a complex dynamically reconfigurable SoC, introducing an adaptive computation approach. The authors aim to accomplish this goal by utilizing the concept of Adaptive[3] programming [48].

The proposed methodology has been introduced for exploiting application specific properties in purely software-based systems in order to accelerate execution time by up to three orders of magnitude for various applications [49]. The main idea behind this methodology is to examine the relationship between inputs and outputs of a computation as the input changes. An adaptive program, [48], responds to input changes by updating its output while only re-evaluating those portions of the program affected by the change. Adaptive programming is useful in situations where input changes lead to relatively small changes in the output. In limiting cases one cannot avoid a complete re-computation of the output, but in many cases the results of the previous computation may be re-used to obtain the updated output more quickly than a complete re-evaluation. Based on this observation it is possible to define the set of *adaptive metrics* that are able to provide information on how a software-based computation is going to be affected by an input change. In the proposed paradigm they use this concept to create a comprehensive analysis engine. The proposed solution for generating the software partition of a codesigned SoC is based on this generalized analysis scheme [30].

The authors adapt this paradigm into HW/SW codesign for reconfigurable SoCs, developing a new performance model and an associated evaluation metric to identify application specific input behavior thereby differentiating between various levels of performance across different portions of software modules. This general performance model is then embedded along with hardware performance models into the codesign environment, which will yield a highly flexible means to evaluate the performance impact of different partitioning and allocation decisions.

B. On-Line Task Management

In [50], [51] an extension of a well-known and portable kernel such as GNU/Linux, in order to introduce a support for dynamic reconfiguration and to simplify the interface between the user application and the reconfigurable hardware, has been proposed. Each software application, also named *process*, can issue one or more system calls in order to require a specific functionality, which may be available either as a software library, or as a hardware IP-Core, or both. The operating system is in charge of choosing among the software or the hardware implementation according to different criterias, such as the amount of free area on the FPGA.

In [51] a cryptographic reconfigurable architecture, involving two popular encryption algorithms (the *Advanced Encryption Standard* and the *Data Encryption Standard*) has been presented to evaluate the performance of the run-time decision of the best implementation for an demanded task. Both the algorithms have been implemented as software applications and IP-Cores, whose FPGA-implementation has been shown in Figure 1, and they have been made available through the reconfiguration manager.

The evaluation system, implemented on Xilinx Virtex-II Pro VP30, has been defined using an architecture composed of two reconfigurable cores reconfigured using a self partial dynamic reconfiguration technique managed by the Microblaze, running

[2]Where performance can have different meaning according to a specific scenario. It is possible to have a system trying to maximize the overall completion time of a set of applications, and, under different constraints, having the same system running trying to minimize its power consumption.

[3]Adaptivity implies that due to input changes the output of the system is updated only re-evaluating those portions of the program affected by the changes.

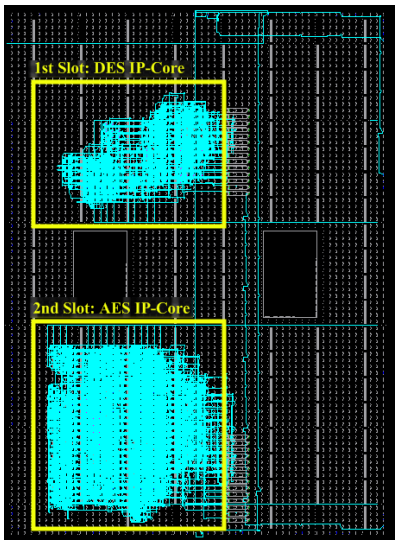

Figure 1. AES and DES IP-core implementations: FPGA-view.

the extended PetaLinux, with the 2.4 uCLinux kernel, operating system, and realized using the ICAP port. The proposed case study, as shown in Figure 2, compares the performance of different implementations of the AES algorithm.

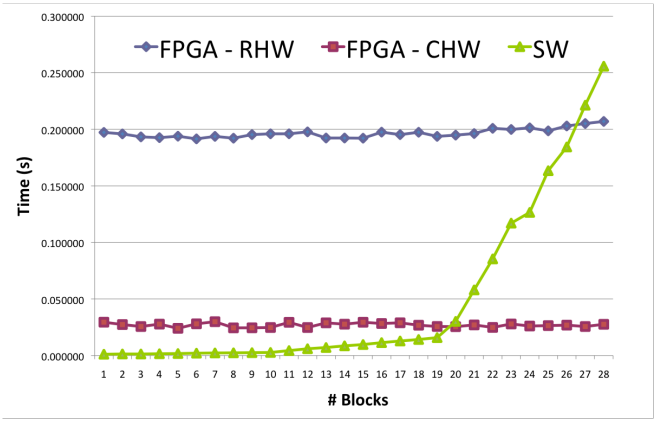

Figure 2. Execution time of the implementations of the AES algorithm.

The data related to the software solution has been computed using an Intel Pentium Dual Core working at 1.60GHz with Ubuntu 8.10 (kernel 2.6.27). Figure 2 shows the impact of the reconfiguration latency in the execution of a functionality, since it may dominate the overall execution time and makes the IP-Core implementation less efficient than a software execution. Additional elements can affect the behavior of the system at runtime, such as a situation where the entire FPGA area is being used, and therefore a new task cannot be executed until one of the configured IP-Cores completes its execution. In order to optimize the execution time of a functionality, it is important for the operating system to be able to choose the best implementation at runtime. It has to be able to understand on which point of the graph shown in Figure 2 it is working. This will lead the OS to choose the most appropriate implementation for the demanded task. The benefits of a combined use of implementation selection and IP-Core caching can be observed by evaluating the time required to serve a large number of requests.

VI. Conclusion

New application domains demand ever increasing adaptability and performance [2]–[4]. In order to cope with changing user requirements, improvements in system features, changing protocol [5] and data-coding standards, and demands for support of a variety of different user applications [6], many emerging applications in communication, computing and consumer electronics demand that their functionality stays flexible after the system has been manufactured. Furthermore, nowadays research is pushing forward, looking for complex heterogeneous, reconfigurable multi-cores architecture. Good examples of *heterogeneous systems, highly dynamic in content, workload and infrastructure* (i.e., nodes are continuously leaving and joining) are cloud computing, grid [9], [10], cluster and peer to peer architectures. In order to overcome the limits deriving by the increasing complexity and the associated workload to maintain such complex infrastructures, one possibility is to adopt self-adaptive [38] and autonomic computing systems [1]. These systems are able to configure, heal, optimize and protect themselves without the need for human intervention. Within this context, reconfigurable computing systems are moving to self-adaptive and autonomic computing systems where either hardware components [7], [28], [37], [52], the applications [36], [53] and the operating system [29], [54] have to be seen as an unique entity that have to be able to autonomously adapt itself to achieve the best performance.

Acknowledgment

I'd like to thank all the many people who I had the pleasure to work with over the last two years for all the useful discussion I had with them and for their thoughtful ideas. Special thanks to V. Rana, I. Beretta, D. Sciuto, S. Ogrenci Memik, A. Agarwal and all the members of the CARBON and the DRESD research groups.

References

[1] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[2] Intel, "Reliability, availability, and serviceability for the always-on enterprise. the enhanced ras capabilities of intel processor-based server platforms simplify 24x7 business solutions," Online document, www.intel.com/assets/pdf/whitepaper/ras.pdf, 2005.

[3] J. Castillo, P. Huerta, V. López, and J. I. Martínez, "A secure self-reconfiguring architecture based on open-source hardware," in *International Conference on Reconfigurable Computing and FPGAs (ReConFig'05)*, 2005.

[4] K. Danne, C. Bobda, and H. Kalte, "Run-time exchange of mechatronic controllers using partial hardware reconfiguration," 2003, pp. 272–281.

[5] A. P. Chaubal, "Design and implementation of an fpga-based partially reconfigurable network controller," Master's thesis, Virginia Polytechnic Institute and State University, 2004.

[6] Oracle, "Automatic workload repository (awr) in oracle database 10g," Website, http://www.oracle-base.com/articles/10g/AutomaticWorkloadRepository10g.php.

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

[8] R. Pena-Ortiz, J. Sahuquillo, A. Pont, and J. A. Gil, "Modeling continuous changes of the user's dynamic behavior in the www," pp. 175–180, 2005.

[9] F. A. J. Buisson and J. L. Pazat, "Dynamic adaptation for grid computing," *Lecture Notes in Computer Science. Advances in Grid Computing - EGC*, pp. 538–547, 2005.

[10] S. S. Vadhiyar and J. J. Dongarra, "Self adaptivity in grid computing," *Concurr. Comput. : Pract. Exper.*, vol. 17, no. 2-4, pp. 235–257, 2005.

[11] W. Gentzsch, K. Iwano, D. Johnston-Watt, M. Minhas, and M. Yousif, "Self-adaptable autonomic computing systems: An industry view," *Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on*, pp. 201–205, Aug. 2005.

[12] J. S. Lovstad and P. H. Hughes, "Run-time software configuration for mobile devices using an evolutionary quantifiable deployment model," pp. 189–200, 2008.

[13] A. Avritzer, R. G. Cole, and E. J. Weyuker, "Using performance signatures and software rejuvenation for worm mitigation in tactical manets," pp. 172–180, 2007.

[14] D. Pescovitz, "Monsters in a box. think you know what a supercomputer is? think again: The real thing will blow your mind." Wired, 8(13):341–347. Online document, http://www.wired.com/wired/archive/8.12/supercomputers.html.

[15] S. Kelem, "Virtex series configuration architecture user guide," *Xilinx XAPP151*, 2003.

[16] Xilinx, *Virtex-II Pro Data Sheet Virtex-II Pro™ Platform FPGA Data Sheet*. Xilinx, 2003.

[17] X. Inc., "Virtex-4 user guide," Xilinx Inc., Tech. Rep. ug70, March 2007. [Online]. Available: http://www.xilinx.com/bvdocs/userguides/ug70.pdf

[18] ——, "Opb hwicap (v1.00.b) product specification," Xilinx Inc., Tech. Rep., March 2005.

[19] V. Rana, M. Santambrogio, and D. Sciuto, "Dynamic reconfigurability in embedded system design," in *IEEE International Symposium on Circuits and Systems, 2007. ISCAS 2007*, May 2007, pp. 2734–2737.

[20] Xilinx Inc., *Application Notes 290. Two Flows for Partial Reconfiguration: Module Based or Small Bit Manipulations*, San Jose, California, 2004.

[21] ——, *Early Access Partial Reconfiguration Guide*, Xilinx Inc., 2006.

[22] Y. Li, T. Callahan, E. Darnell, R. E. Harr, U. Kurkure, and J. Stockwood, "Hardware/software codesign of embedded reconfigurable architectures," in *Proceedings of the 37th Conference on Design Automation*. ACM/IEEE, 2000, pp. 507–512.

[23] J. Noguera and R. M. Badia, "Hw/sw codesign techniques for dynamically reconfigurable architectures," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 10, no. 4, pp. 399–415, 2002.

[24] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-aware hw-sw partitioning for reconfigurable architectures with partial dynamic reconfiguration," in *DAC '05: Proceedings of the 42nd annual conference on Design automation*. ACM Press, 2005, pp. 335–340.

[25] E. L. Horta, J. W. Lockwood, and D. Parlour, "Dynamic hardware plugins in an fpga with partial run–time reconfigurtion," pp. 844–848, 2002.

[26] S. Tapp, "Configuration quick start guidelines," *XAPP151*, July 2003.

[27] E. Horta and J. W. Lockwood, "Parbit: A tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (fpgas)," *Washington University, Department of Computer Science, Technical Report WUCS–01–13*, July 2001.

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

[29] O. Krieger, M. Auslander, B. Rosenburg, R. W. J. W., Xenidis, D. D. Silva, M. Ostrowski, J. Appavoo, M. Butrico, M. Mergen, A. Waterland, and V. Uhlig, "K42: building a complete operating system," pp. 133–145, 2006.

[30] M. D. Santambrogio, S. O. Memik, V. Rana, U. A. Acar, and D. Sciuto, "A novel soc design methodology combining adaptive software and reconfigurable hardware," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design. ICCAD 2007*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 303–308.

[31] V. Sima and K. Bertels, "Runtime decision of hardware or software execution on a heterogeneous reconfigurable platform," in *IEEE International Symposium on Parallel and Distributed Processing, 2009. IPDPS 2009*, May 2009.

[32] K. Sigdel, M. Thompson, A. Pimente, K. Bertels, and C. Galuzzi, "System level runtime mapping exploration of reconfigurable architectures," in *IEEE International Symposium on Parallel and Distributed Processing, 2009. IPDPS 2009*, May 2009.

[33] R. Azimi, C. Cascaval, E. Duesterwald, M. Hauswirth, K. Sudeep, P. F. Sweeney, and R. W. Wisniewski, "Performance and environment monitoring for whole-system characterization and optimization," *Proceedings of the PAC2 Conference on Power/Performance Interaction with Architecture, Circuits, and Compilers*, pp. 15–24, 2004.

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

[35] A. Baumann, D. D. Silva, O. Krieger, and R. W. Wisniewski, "Improving operating system availability with dynamic update," pp. 21–27, 2004.

[36] C. A. N. Soules, J. Appavoo, K. Hui, R. W. Wisniewski, D. D. Silva, G. R. Ganger, O. Krieger, M. Stumm, M. Auslander, M. Ostrowski, B. Rosenburg, and J. Xenidis, "System support for online reconfiguration," 2003.

[37] J. Appavoo, K. Hui, M. Stumm, R. W. Wisniewski, D. D. Silva, O. Krieger, and C. A. N. Soules, "An infrastructure for multiprocessor run-time adaptation," in *WOSS '02: Proceedings of the first workshop on Self-healing systems*. New York, NY, USA: ACM, 2002, pp. 3–8.

[38] P. Dini, "Internet, grid, self-adaptability and beyond: are we ready?" *Database and Expert Systems Applications, 2004. Proceedings. 15th International Workshop on*, pp. 782–788, Aug.-3 Sept. 2004.

[39] IBM, "Ibm autonomic computing website," Website, http://www.research.ibm.com/autonomic/, 2009.

[40] B. Sprunt, "The basics of performance-monitoring hardware," *Micro, IEEE*, vol. 22, no. 4, pp. 64–71, Jul/Aug 2002.

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

[42] Intel, "Intel itanium architecture software developer's manual," Website, http://www.intel.com/design/itanium/manuals/iiasdmanual.htm, 2006.

[43] N. Bartzoudis and K. McDonald-Maier, "Online monitoring of fpga-based co-processing engines embedded in dependable workstations," *On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International*, pp. 79–84, July 2007.

[44] *Virtex-II Pro and Virtex-II ProX Virtex-II Pro and Virtex-II Pro X FPGA User Guide*. Xilinx, 28 March 2007.

[45] Xilinx, "Virtex-4 configuration user guide," no. ug71, January 2007. [Online]. Available: http://www.xilinx.com/bvdocs/userguides/ug71.pdf

[46] ——, "Virtex-5 configuration user guide," no. ug191, February 2007. [Online]. Available: http://www.xilinx.com/bvdocs/userguides/ug191.pdf

[47] K. Paulsson, M. Hübner, and J. Becker, "On-line optimization of fpga power-dissipation by exploiting run-time adaption of communication primitives," pp. 173–178, 2006.

[48] U. A. Acar, G. E. Blelloch, and R. Harper, "Adaptive functional programming." in *POPL*, 2002, pp. 247–259.

[49] U. A. Acar, G. E. Blelloch, R. Harper, J. L. Vittes, and S. L. M. Woo, "Dynamizing static algorithms, with applications to dynamic trees and history independence." in *SODA*, 2004, pp. 531–540.

[50] M. Santambrogio, V. Rana, and D. Sciuto, "Operating system support for online partial dynamic reconfiguration management," in *International Conference on Field Programmable Logic and Applications, 2008. FPL 2008*, Sept. 2008, pp. 455–458.

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

[52] E. Ipek, O. Mutlu, J. F. Martínez, and R. Caruana, "Self-optimizing memory controllers: A reinforcement learning approach," in *ISCA '08: Proceedings of the 35th International Symposium on Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 39–50.

[53] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N. M. Amato, and L. Rauchwerger, "A framework for adaptive algorithm selection in stapl," in *PPoPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM, 2005, pp. 277–288.

[54] D. Wentzlaff and A. Agarwal, "Factored operating systems (fos): the case for a scalable operating system for multicores," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 76–85, 2009.