COMPUTING SYSTEMS BASED ON ADAPTIVE RECONFIGURABLE ARCHITECTURES

Andrea Cazzaniga, Filippo Sironi, Davide B. Bartolini, Marco D. Santambrogio

Dipartimento di Elettronica e Informazione, Politecnico di Milano

{acazzaniga,sironi,bartolini,santambrogio}@elet.polimi.it

ABSTRACT

Imagine further a computing system that performs better according to a user's preferred goal the longer it runs an application. Such an architecture will enable, for example, a hand-held radio or a cell phone that can run cooler the longer the connection time. Moreover, 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. Self-aware computer systems will be capable of adapting their behavior and resources thousands of times a second to automatically find the best way to accomplish a given goal despite changing environmental conditions and demands. In this work we present an attempt in presenting the key enabling technologies to realize such self-aware runtime system that can gain benefits from the presented paradigm.

1. INTRODUCTION

Reconfiguration capabilities and hardware-software codesign techniques are becoming just elements of a more complex scenario. The need for a systematic approach to the design of new architectures and systems enabling self-awareness is motivated by some trends that have gained momentum in the past few years. Research is pushing forward, looking for complex heterogeneous, reconfigurable multi-core architectures. 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. Different companies, i.e., IBM [2, 3], Oracle [4], and Intel [5] have invested in this research, creating several products characterized by a self-adaptive behavior. However, a lot of work still needs to be performed in defining effective self-adaptive and autonomic architectures in the embedded system domain.

On one hand there is the increasing importance of non-functional constraints: in the *perceived value* of a digital system, features that are not completely reducible to the functionalities are getting ever more important. Two famous examples of such non-functional constraints are power consumption and reliability, but there are many other potential dimensions, that lie at the border of what can be called functionality, that impact user experience of a digital system or device; examples can be results accuracy, like in different audio and video qualities for a multimedia device, or efficiency in understanding human signals in interactions (as already happens, for instance, in speech recognition software)). Meeting such constraints (or optimizing the associated figures) is getting more and more difficult, mainly because of the exponential increase of environmental interactions and conditions in which devices are required to operate.

On the other hand, devices structure evolution tends towards forms of complexity characterized by the increase in number and of complexness of interacting "peer" elements, at various levels (e.g.: cores on a multicore processor, concurrent programs in a multitask operating system, number of threads within a single application). Meeting non functional constraints requires, most of the times, a coordination among all those elements, for any possible working condition. It is evident that statically foreseeing, at design time, the actions that must be taken in order to maximize non-functional constraint satisfaction for all the possible scenarios is already way beyond feasibility. Think of the simplest problem that control engineering faces since a long time: controlling the temperature of a room to stay stable at a given value, within acceptable bounds. Room temperature can be determined or influenced by a plethora of different factors: outside weather, windows being open or closed, the presence of persons inside the room and so on. Knowing all such factors in advance is of course impossible. The conceptual solution developed was the closed loop control: the system reacts to deviations from the goal (differences between the temperature set and that measured) with actions somehow proportional to that distance (injecting thermal power

in the room).

The user of the control system just sets the goal, then the system dynamically and automatically reacts, adapting itself to the new conditions. This control task example can be used as a metaphor for the motivations towards implementation of the self-aware adaptive systems that are the focus of this project: as the temperature controller exploits information on its state and on the environment to pursue a goal that is dependent on a set of factors non foreseeable at design time, so should be able to do, on a much higher, behavioral level, embedded systems.

2. CONTEXT DEFINITION

Resources such as quantities of transistors and memory, the level of integration and the speed of components have increased dramatically over the years. Even though the technologies have improved, we continue to apply outdated approaches to our use of these resources. Within this context, imagine an interaction capability of digital systems by which designers and users can specify their desired goals rather than how to perform a task, along with constraints in terms of an energy budget, time, or simply a preference for an approximate answer over an exact answer. Imagine further a computing chip that performs better according to a user's preferred goal the longer it runs an application. Such an architecture will enable, for example, a handheld radio or a cell phone that can run cooler the longer the connection time. Or, a system that can perform reliably and continuously in a range of environments by tolerating hard and transient failures through self healing. Self-aware computer systems will be capable of adapting their behavior and resources thousands of times a second to automatically find the best way to accomplish a given goal despite changing environmental conditions and demands. Such a capability would benefit a broad spectrum of computer systems from embedded systems to supercomputers and is particularly useful for meeting power, performance, and resource-metering challenges in mobile computing [6, 7], grid and cloud computing [8, 9, 10], multicore computing [11, 12, 13], networks [14, 15], self-healing systems [16, 17, 18], complex distributed Internet services [19, 20, 21], distributed system [22], operating systems [23, 24, 25, 3, 26], and adaptive and dynamic compilation environments [27, 28].

3. RUNTIME SELF-AWARE SUPPORT

The operating system is in charge of choosing at runtime between the set of possible implementations (a software one or one of the available hardware implementations) according to different criteria, such as the available area (set of resources) on the FPGA, input data type and dimension, functionalities already implemented and available as hardware components. The runtime decision of the most suitable implementation (software or reconfigurable hardware) due to runtime conditions, allows this work to be considered as an attempt to the define a *self-aware computing system*. The operating system answers a request for a functionality by choosing a runtime the best implementation. *Best* does not mean the optimal solution but the one that can guarantee the best performance considering all the runtime conditions in which it has to be executed. Considering the scenario where an hardware solution is chosen as the best implementation, the corresponding hardware module has to be loaded by configuring the IP-Core on the FPGA and by creating a communication channel between the module and the software application in a transparent way. As a consequence, the IP-Core becomes accessible from the userspace when the control is returned to the user application. The online adaptability of the overall system is implemented in the OS by means of kernel modules implementing a closed control loop, called *Self-Aware Support* in Figure 1, and an Adaptive library. The *Self-Aware support* kernel extension, located between the userspace and the physical architecture, performs the online adaptation of the system, providing a common interface for software applications and hardware developers. Each software application communicates with the kernel using the API of the reconfiguration library, which allows also the access to the hardware component that physically implements specific functionalities, once they have been configured on the FPGA by the operating system.

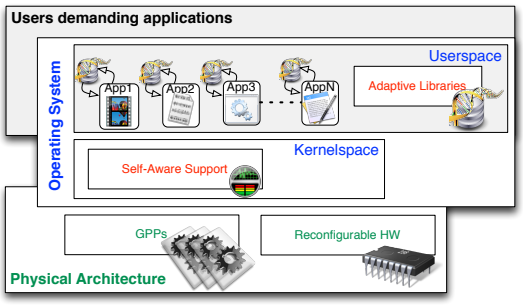

Fig. 1. The overview of a Self-Aware systems where the operating system is in charge on managing the online adaptation of the applications and of the underline architecture.

4. EXPERIMENTAL RESULTS

We designed a self-aware implementation [29] of the GNU/Linux operating system able to monitor itself to take autonomous decisions on the best implementation for the demanded functionalities. 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 classical software library, as an adaptive software, or as hardware

IP-Cores, or all of them. The operating system is in charge of choosing among the software or the hardware implementation according to different criteria, such as the amount of free area on the FPGA, or the dimension/number of data that has to be processed.

The case study that we would like to present, belongs to the cryptographic application domain. A cryptographic reconfigurable architecture, implementing the *Data Encryption Standard*, has been designed to evaluate the performance of the run-time decision of the best implementation for any demanded task. The proposed case study, as shown in Figure 2, compares the performance of different implementations of the DES algorithm. The FPGA-based solutions have

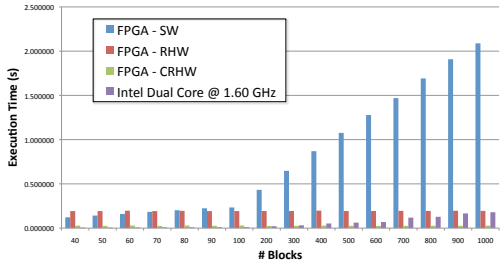

Fig. 2. Performance, in execution time, of the different implementations of the DES algorithm.

been implemented on a Xilinx Virtex-II Pro working with at 100MHz, while the data regarding the software solution has been taken using an Intel Pentium Dual Core working at 1.60GHz with Linux (kernel 2.6.27). Three different FPGA implementations have been implemented:

- SW: the DES algorithms has been executed in software on the processor on the FPGA;
- RHW: the algorithm has been implemented as a reconfigurable component and finally;
- CRHW: the reconfigurable component was already configured on the FPGA and ready to be used.

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. Therefore, the OS has not only to be able to understand on which *scenario* of the graph shown in Figure 2 it is working, but to foresee the impact of its decision on future calls. This will lead the OS to choose the most appropriate implementation for the demanded task, that may not lead to the best performance to that specific call, but that may provide better performance to the next ones.

In a scenario were we have enough area on the FPGA to configure the HW implementation of the DES algorithm, for a call on at least 300[1] blocks, it is not always the best decision to go for the Intel Dual Core solution even if we do not have the core algorithm already implemented as an HW IP-Core. To explain this situation we can consider the scenario characterized by two calls of the DES algorithm, the first one on 1000 blocks and the second one on 400. As shown in Figure 2, the best implementation, when the HW IP-Core has not been already configured on the FPGA, is the Dual Core one. Within this scenario, where the system has no knowledge of future calls (it is not aware of the fact that after the 1000 call it will serve a 400 one), the OS will always choose the Dual Core implementation of the DES. This is the solution already implemented in literature in different works [30, 31]. On the contrary, considering the history of the previous calls, the performance information of all the possible implementations, and the probability of receiving a certain call, our system will choose the reconfigurable HW solution (reconfiguration of the IP-Core and its execution) for the first call, since it will be payback for each consecutive call on at least 400 blocks. Table 1 presents the comparison between the two different approaches.

5. REFERENCES

[1] 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.

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

[3] 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," *EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pp. 133–145, 2006.

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

[5] 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.

[6] E. F. by the European Union's 7th Framework Program, "Exposing the features in ip version six protocols that can be exploited extended for the purposes of designing/building autonomic networks and services," 2009. [Online]. Available: http://www.efipsans.org

[7] M. M. Masters, "Exploring usability in mobile autonomic networks," in *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with*

[1] Which is the point where the CRHW implementation outperforms the Intel Dual Core one

Table 1. Different possible exe., for the same sequence of inputs

First call, #Blocks: 1000	**Second call, #Blocks: 400**	**Overall Execution TIme** (s)
Intel Core Duo: 0.179162 s	Intel Core Duo: 0.052382 s	**0.231544**
RHW: 0.194679 s	CRHW: 0.025710 s	**0.220389**

mobile devices and services. New York, NY, USA: ACM, 2008, pp. 549–550.

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

[9] 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.

[10] P. Reinecke and K. Wolter, "Adaptivity metric and performance for restart strategies in web services reliable messaging," in *WOSP '08: Proceedings of the 7th International Workshop on Software and Performance*. ACM, 2008, pp. 201–212.

[11] 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.

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

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

[14] HAGGLE, "A european union funded project in situated and autonomic communications," 2007. [Online]. Available: http://www.haggleproject.org

[15] ANA, "Autonomic network architecture," 2009. [Online]. Available: http://www.ana-project.org

[16] C. M. Garcia-Arellano, S. Lightstone, G. Lohman, V. Markl, and A.Storm, "A self-managing relational database server: Examples from IBM's DB2 universal database for linux unix and windows," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 36, no. 3, pp. 365– 376, 2006.

[17] 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.

[18] D. Breitgand, M. Goldstein, E. Henis, O. Shehory, and Y. Weinsberg, "Panacea towards a self-healing development framework," in *Integrated Network Management*. IEEE, 2007, pp. 169–178.

[19] E. U. F. P. CASCADAS, "Component-ware for autonomic situation-aware communications, and dynamically adaptable services," 2009. [Online]. Available: http://www.cascadas-project.org

[20] A. Fox, E. Kiciman, and D. Patterson, "Combining statistical monitoring and predictable recovery for self-management," in *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*. New York, NY, USA: ACM, 2004, pp. 49–53.

[21] J. Strassner, S.-S. Kim, and J. W.-K. Hong, "The design of an autonomic communication element to manage future internet services," in *APNOMS*, ser. Lecture Notes in Computer Science, C. S. Hong, T. Tonouchi, Y. Ma, and C.-S. Chao, Eds., vol. 5787. Springer, 2009, pp. 122–132.

[22] A. Quiroz, N. Gnanasambandam, M. Parashar, and N. Sharma, "Robust clustering analysis for the management of self-monitoring distributed systems," *Cluster Computing*, vol. 12, no. 1, pp. 73–85, 2009.

[23] J. Vetter and P. Worley, "Asserting performance expectations," in *Supercomputing, ACM/IEEE 2002 Conference*, Nov. 2002, pp. 33–33.

[24] M. Caporuscio, A. Di Marco, and P. Inverardi, "Run-time performance management of the siena publish/subscribe middleware," in *WOSP '05: Proc. of the 5th Inter. Work. on Software and performance*, 2005, pp. 65–74.

[25] 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.

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

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

[28] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe, "PetaBricks: A language and compiler for algorithmic choice," in *Conf. on Programming Language Design and Implementation*, Jun 2009.

[29] M. D. Santambrogio, "From reconfigurable architectures to self-adaptive autonomic systems," *IEEE International Conference on Computational Science and Engineering*, pp. 926 – 931, 2009.

[30] 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.

[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*, May 2009.