Processor Architectures for Ontogenesis

Gianluca Tempesti
Logic Systems Laboratory
Swiss Federal Institute of Technology at Lausanne (EPFL)
EPFL-IC-LSL, INN Ecublens, CH-1015 Lausanne, Switzerland
Email: Gianluca.Tempesti@epfl.ch ; Phone: +41-21-6932602

Abstract

In this article we present the outline of a novel processor architecture aimed to the design of systems that more closely resemble, within the limitations imposed by the capabilities of conventional silicon, the general modus operandi of multi-cellular organisms.

1. Introduction

One of the main motivations for the study of bio-inspired hardware is the astounding level of complexity achieved by biological organisms, a complexity far beyond that of even the most advanced silicon-based circuit. The promise of next-generation technologies [2][3][4] lies essentially in their ability to work at the same molecular level, and with the same component densities, as biological systems.

Among the many questions open for these technologies is how to exploit the wealth of hardware that will be at the disposal of the engineer. The study of how biology, and notably multi-cellular organisms, have successfully solved this issue is a possible avenue for finding approaches that could potentially be applied to these circuits.

The structure and operation of multi-cellular organisms relies, among other things, on the specialization of the cells to a finite set of specific operations. This specialization implies that the cells' physical structure is adapted to its function (e.g., a skin cell is physically very different from a liver cell). Structural differences notwithstanding, the same program (the genome) controls the operation of these cells. To realize bio-inspired systems, we must be able to achieve a similar degree of adaptation. The realization of processor architectures that can efficiently exploit the adaptive features of this approach remains an open problem.

This article describes the first results of a new project that, building on the results of the Embryonics [5] and POEtic [11] projects, will define a set of architectures specifically conceived for the realization of bio-inspired systems. In this paper, we will try to identify some of the main requirements of such architectures and present the outline of a novel architecture that represents an effort towards the designs of systems that more closely resemble, within the limitations imposed by the capabilities of conventional silicon, the general *modus operandi* of multi-cellular organisms.

2. Background

Many different approaches can be used to draw inspiration from nature in the design of electronic systems. Even within the much more restricted area of ontogenetic hardware (that is, hardware inspired by the growth of multicellular organisms), several valid approaches have been studied (for a partial review of such systems, see [8]).

Within the Embryonics project [5], we have been studying the application of biological ontogenesis to the design of digital hardware systems for several years. Among what we feel are the main contributions to the field brought about by this field is a self-contained representation of a possible mapping between the world of multi-cellular organisms in biology and the world of digital hardware systems (Fig. 1).

We define an artificial organism as a parallel array of cells, where each cell is a simple processor that contains the description of the operation of every other cell in the organism in the form of a program (the genome). The redundancy inherent in this approach is compensated by the added capabilities of the system, that can exploit this redundancy to implement properties such as growth and self-repair.

Our cells are *reconfigurable* processing elements, realized by programmable logic circuits (the *molecular* level) and structurally adapted to the application. For a given application, all cells are structurally identical and contain the same program, but different parts of the program and of the structure are activated depending on the cell's position.

Using this approach, we concentrated our efforts towards the demonstration of a number of basic properties of our systems such as self-repair [9] and growth [6].

Proceedings of the 2004 NASA/DoD Conference on Evolution Hardware (EH'04)
0-7695-2145-2/04 $ 20.00 © 2004 IEEE

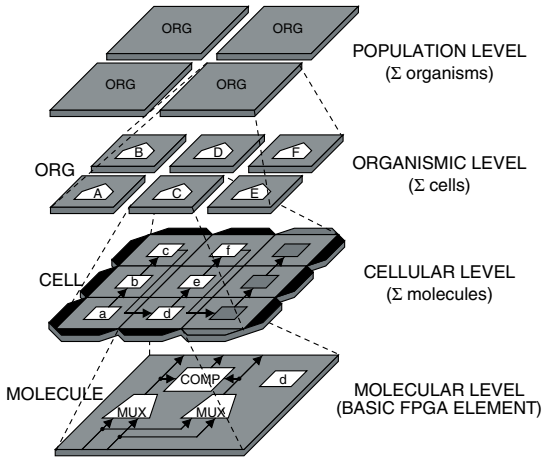

Figure 1. The four hierarchical levels of complexity of the Embryonics project

In 2001, we launched, together with the universities of York, Barcelona (UPC), Lausanne, and Glasgow, a project called "Reconfigurable POEtic Tissue" [11] funded by the Future and Emerging Technologies programme (IST-FET) for the European Community. This project aims at defining a novel programmable logic circuit specifically designed for the implementation of bio-inspired systems.

Within the POEtic project, we defined bio-inspired processors as three-layered structures: the *genotype layer* stores the genetic information of the cell (the genome), to which evolutionary approaches can be applied; the *mapping layer* implements developmental algorithms to realize growth-like processes; the *phenotype layer* is used for the actual execution of the application.

Within this project, we also defined a *dynamic routing network* [10] which, by setting up dynamically at runtime the communication channels between cells, renders unnecessary to explicitly define such connections at design time. This mechanism has major consequences for ontogenetic processes, since it allows cells to be created and connected to the rest of the network (or destroyed and removed from the network) at any time during the circuit's operation.

3. Ontogenetic Architectures

Keeping in mind the results of previous projects, we can analyze the structure of bio-inspired processors to try and identify some of the main features that are required in an ontogenetic processing element.

Of the three layers mentioned above, the genotype layer is probably the simplest, consisting essentially of the genome memory that stores the genetic information of the system as either a set of parameters or as a structured program. However, this simplicity hides some difficult architectural challenges relative to the *size* of the genome memory, introduced by the presence of a complete copy of the organism's genome in every cell.

The architecture should then be able to *reduce the size of the genome program* by reducing the number of instructions in the program and/or the size of the instructions.

The mapping layer of an ontogenetic processor is probably its most atypical feature. The function of this layer is essentially two-fold: implementing cellular differentiation and supporting most of the fault tolerance mechanisms.

Cellular differentiation is the process whereby each cell decides which portion (*gene*) of the entire genome to execute and thus which function to implement. A key aspect of this process is the *growth algorithm*, which requires the ability to dynamically add (or in some cases remove) processing elements from the array, assign them a function, and connect them to the pre-existing elements in a pattern that depends on the application's requirements. In reality, it is the last operation that presents the most practical difficulties, and it is in this context that a POEtic-style dynamic routing algorithm is extremely useful, allowing such dynamics to take place transparently to the user.

The architecture should then be able to *implement growth and cellular differentiation* by allowing processors to be seamlessly added (or removed) from the system

Fault tolerance introduces an additional level of complexity to the mapping layer. In reality, reconfiguration-based self-repair shares several mechanisms with the process of growth, as it implies the activation of a new processor and its connection to the network, as well as the deactivation of the faulty processor and its removal from the network. The main differences lay in the need to transfer the state of the faulty processor and the information concerning its connections to the new processor, a process that inevitably requires the introduction of additional mechanisms.

The architecture should then be able to *support fault tolerance through reconfiguration*, letting the system transparently recover after a fault has been detected.

In the phenotype layer resides the versatility of the system: depending on the application, this layer can be called upon to execute an extremely varied set of tasks, ranging from artificial neurons to image processing to robot control, and to alter its structure accordingly. At the architectural level, this implies that the processor should be able to exploit functional units that change from application to application. Moreover, the standard approach of defining application-specific instructions that, once decoded in the processor, activate the desired functional unit is not well suited for an architecture that needs to be used for a wide range of applications, as it would require a redefinition of the instruction set (and hence of the decoding circuitry in the processor) for each application.

The architecture should then be able to *adapt its structure to the application* by exploiting functional units that are targeted to the required functions without altering its programming language.

Furthermore, while a dynamic routing algorithm can be extremely useful for the implementation of application-specific communication patterns between processors, it also requires an architecture that is able to handle non-standard communication. The conventional approach is to design a communication network that can realize all of the possible communication patterns required by application. A dynamic routing algorithm, on the other hand, implies that the task of setting up the physical connection network falls to the processors themselves, which must then be capable of creating, destroying, or altering their own connections to the other cells in the system according to the instructions contained in their genome program.

The architecture should then be able to *implement complex communication patterns* that can be dynamically altered during operation.

Mapping an application onto the array of processors has always been a complex task, often requiring the complete redesign of the system, a major drawback for research groups interested in developing and testing novel bio-inspired approaches. It would then be a substantial advantage for an ontogenetic architecture to be integrated into a standardized design environment allowing the user to automatize the design of the application-specific processors.

Even if this requirement seems obvious, its implementation is far from simple: the complete configurability of ontogenetic systems is in itself an obstacle to their use. By providing too many degrees of freedom, the user is often required to completely re-design a system in order to test different approaches and mechanisms.

The architecture should then allow the user to generate the processor through an *automatic design flow* leading from a high-level description of an algorithm to the configuration bitstream that implements the circuit on the programmable logic substrate.

4. A MOVE Architecture

The requirements of our bio-inspired approach imply an architecture that is substantially different from conventional general-purpose processor architectures. Similarities, on the other hand, can be found between the requirements of the phenotype layer of our systems and those of application-specific processors. It is then in this direction that we moved to find a basis on which to build our ontogenetic systems. In order to simplify the development of our processors, we exploited the logical separation between the three layers (genotype, mapping, and phenotype) and developed solutions for each of the layers individually.

The implementation of the phenotype layer, indubitably the most application-dependent of the three layers, can exploit, as we have mentioned, the relatively vast knowledge-base associated with the design of application-specific processors. Notably, we have identified an architectural paradigm that we believe meets the major constraints of our system: the MOVE paradigm [1][7].

The MOVE approach is deceptively simple (Fig. 2). The processor executes the program using a set of functional units (FU), each with one or more inputs and one or more outputs. To each input and/or output corresponds a register. There exists only a single instruction, ”MOVE”, which transfers data from the output register of a unit to the input register of another unit. So, for example, to add two numbers the processor would use a FU that implements the sum, move one operand into the first input register of the unit, move the other operand into the second input, and move the result from the output register to the unit that needs it.

The MOVE approach, in and of itself, does not imply high performance: a simple addition requires three instructions. Its strength lies in its *modularity*: the architecture handles the functional units as ”black boxes”, without any knowledge of their functionality. This property implies that any functional unit can be handled using the same, simple programming language, and that the units can be tailored to the functions required by the application. The processor could contain a functional unit that implements, for example, the function `f(A,B)=16*A+3*(A+B)`, should such a function be needed by the application.

Moreover, communication channels can be handled exactly in the same way as a functional unit (Fig. 2): the address of the target cell can be moved to a dedicated input register in the communication units (CU), and the data similarly moved into a second input register. The unit can then autonomously handle the process of setting up the communication channel and transmit the data.

A MOVE architecture thus meets the requirements of the phenotype layer of our processors. In addition, it is also compatible with the main requirements of the mapping layer: the state of a cell, including its communication paths, resides completely within the I/O registers of the units. Being so grouped, it becomes possible (e.g., with a scan-chain mechanism) to read from and write into these registers to either activate a new processor or to implement self-repair.

This kind of architecture also meets some of the requirements of the genotype layer. The instruction set of a MOVE processor is extremely compact, since there is no explicit opcode and since all the moves are made between registers (the size of the instructions depends only on the number of registers in the processor, itself a function of the application). As far as the size of the program is concerned, the increase in number of instructions should be compensated, as we mentioned, by the specialization of the functional units.

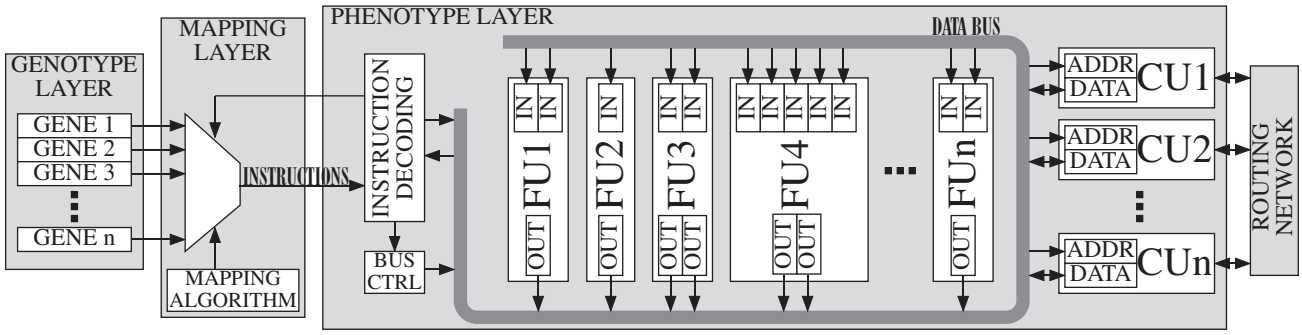

Figure 2. A MOVE processor consists of a set of functional and communication units tied together by one or more data busses.

Finally, by allowing the creation of functional units specifically designed for a given applications and by handling functional units as black boxes, a MOVE architecture can potentially be the object of an automated design flow, in which the functional units could be selected from pre-defined libraries and the architecture used to provide a framework in which the units are inserted. This approach could help simplify the design of ontogenetic processors without losing their versatility and universality.

5. Conclusions

The implementation of ontogenetic systems in silicon using the approach defined in the Embryonics and POEtic projects amounts to the creation of massively parallel arrays of application-specific processors with properties, such as growth and self-repair, typical of biological organisms.

Two very practical considerations stand in the way of such an implementation. The first is technological: current reconfigurable circuit densities do not allow the realization of massively parallel systems. However, improvements in silicon technology and, eventually, the development of molecular-level circuits should not only allow such systems to be built, but even *require* some of their features.

The second consideration concerns the implementation of ontogenetic systems: there exists today no universal architecture for application-specific processors that can be used to implement effectively our approach. In this article we present the background and first conclusions of a new project that shall attempt to remedy this lack. The results presented here represent the framework within which the project will evolve, and notably the outline of the processor architecture that will be used as a basis for our exploration of ontogenetic processes applied to the implementation of a wide range of bio-inspired systems.

References

[1] H. Corporaal. *Microprocessor Architectures – from VLIW to TTA*. John Wiley & Sons, 1998.

[2] K. E. Drexler. *Nanosystems: Molecular Machinery, Manufacturing and Computation*. John Wiley, New York, NY, 1992.

[3] W. Goddard III, D. Brenner, S. Lyshevski, and G. Lafrate, editors. *Handbook of Nanoscience, Engineering, and Technology*. CRC Press, Boca Raton, FL, 2002.

[4] C. Lent and P. Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85:4:541–557, 1997.

[5] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Towards robust integrated circuits: The embryonics approach. *Proceedings of the IEEE*, 88(4):516–541, 2000.

[6] D. Mange, A. Stauffer, E. Petraglio, and G. Tempesti. Embryonic machines that divide and differentiate. In *Proc. 1st Int. Workshop on Biologically Inspired Approaches to Advanced Information Technology (BioADIT04)*, 2004.

[7] D. Tabak and G. J. Lipovski. MOVE architecture in digital controllers. *IEEE Transactions on Computers*, C-29(2):180–190, Feb. 1980.

[8] G. Tempesti, D. Mange, E. Petraglio, A. Stauffer, and Y. Thoma. Developmental processes in silicon: An engineering perspective. In *Proc. 2003 NASA/DoD Conference on Evolvable Hardware (EH-2003)*, pages 255–264. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[9] G. Tempesti, D. Mange, and A. Stauffer. A robust multiplexer-based fpga inspired by biological systems. *Journal of Systems Architecture*, 43(10):719–733, 1997.

[10] Y. Thoma, E. Sanchez, J.-M. Moreno Arostegui, and G. Tempesti. A dynamic routing algorithm for a bio-inspired reconfigurable circuit. In *Proc. 13th Int. Conf. on Field-Programmable Logic and Applications (FPL03)*, volume 2778 of *LNCS*, pages 681–690. Springer Verlag, 2003.

[11] A. Tyrrell, E. Sanchez, D. Floreano, G. Tempesti, D. Mange, J.-M. Moreno, J. Rosenberg, and A. Villa. Poetic tissue: An integrated architecture for bio-inspired hardware. In *Proc. 5th Int. Conf. on Evolvable Systems: From Biology to Hardware (ICES2003)*, volume 2606 of *Lecture Notes in Computer Science*, pages 129–140. Springer Verlag, 2003.