# Architectures and Compilers to Support Reconfigurable Computing

by João M. P. Cardoso and Mário P. Véstias

## Introduction

The main characteristic of **Reconfigurable Computing (RC)** is the presence of hardware that can be reconfigured (**r**econfig**w**are - **RW**) to implement specific functionality more suitable for specially tailored hardware than on a simple uniprocessor. RC systems join microprocessors and programmable hardware in order to take advantage of the combined strengths of hardware and software [20, 5] and have been used in applications ranging from embedded systems to high performance computing. Many of the fundamental theories have been identified and used by the Hardware/Software Co-Design research field [16]. Although the same background ideas are shared in both areas, they have different goals and use different approaches.

Although the basic concept was proposed in the 1960s, RC has only recently been feasible; this is the result of the availability of high-density reconfigurable devices. These devices make the hardware characteristics of **Application Specific Integrated Circuits** (ASICs) much more flexible.

During the past 5 years a large number of RC systems developed by the research community have demonstrated the potential for achieving high performance for a range of applications. However, performance improvements possible with these systems typically are dependent on the skill and experience of hardware designers. This method of RW programming cannot fully exploit the increasing density of reconfigurable devices. Hence, a current challenge in this area is the establishment of an efficient RW compiler that would help the designer accomplish adequate performance improvements without the need to be involved in complex low level manipulations. Although mature design tools exist for logic and layout synthesis for programmable devices, **High-Level Synthesis** (HLS) and multi-unit partitioning (both spatial and temporal) need to be further developed.

## Reconfigurable computing: Why and How?

Usually the target RC system is based on multiple SRAM-based **Field Programmable Gate-Arrays** (SRAM-FPGAs), which act as **Reconfigurable Processing Units** (RPUs), coupled to a personal computer. The coupling of dedicated hardware to a host computer (as shown in Figure 1) in order to accelerate some computationally intensive tasks, is not a new concept. A common example in the PC industry is the use of graphic boards to accelerate graphical transformations. However, such hardwired

application-specific accelerator circuits require great design effort and typically use more silicon area than the microprocessor itself.

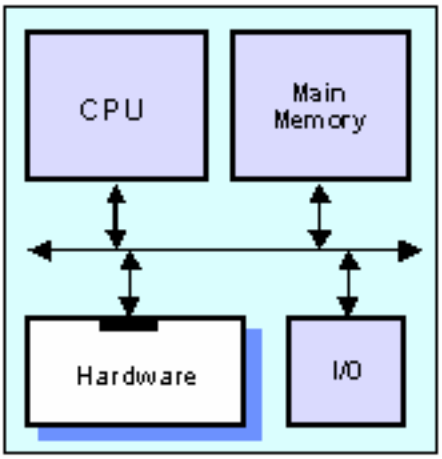



Figure 1. A hardware device added to an existing architecture for acceleration purposes.

The use of reconfigurable hardware makes it possible to couple the host computer with more flexible hardware resources, better adapted to the application under current execution, and with the possibility to adapt to new versions of an application. Moreover, the dynamic reconfiguration capability allows for time-sharing of different tasks, which may significantly reduce the required silicon area. These devices have a faster growth of transistor density than that of general processors, as can be seen in Figure 2. The newest FPGAs can support circuits with about 500,000 equivalent gates in the same device and improved FPGAs (such as the Xilinx Virtex(tm) series [29]) with one-million gates have been announced.

With the introduction of FPGAs with faster reconfiguration times and partial reconfiguration support, it is possible to use FPGAs in a dynamically reconfigurable environment. This technology makes possible the concept of unlimited hardware or "virtual hardware".

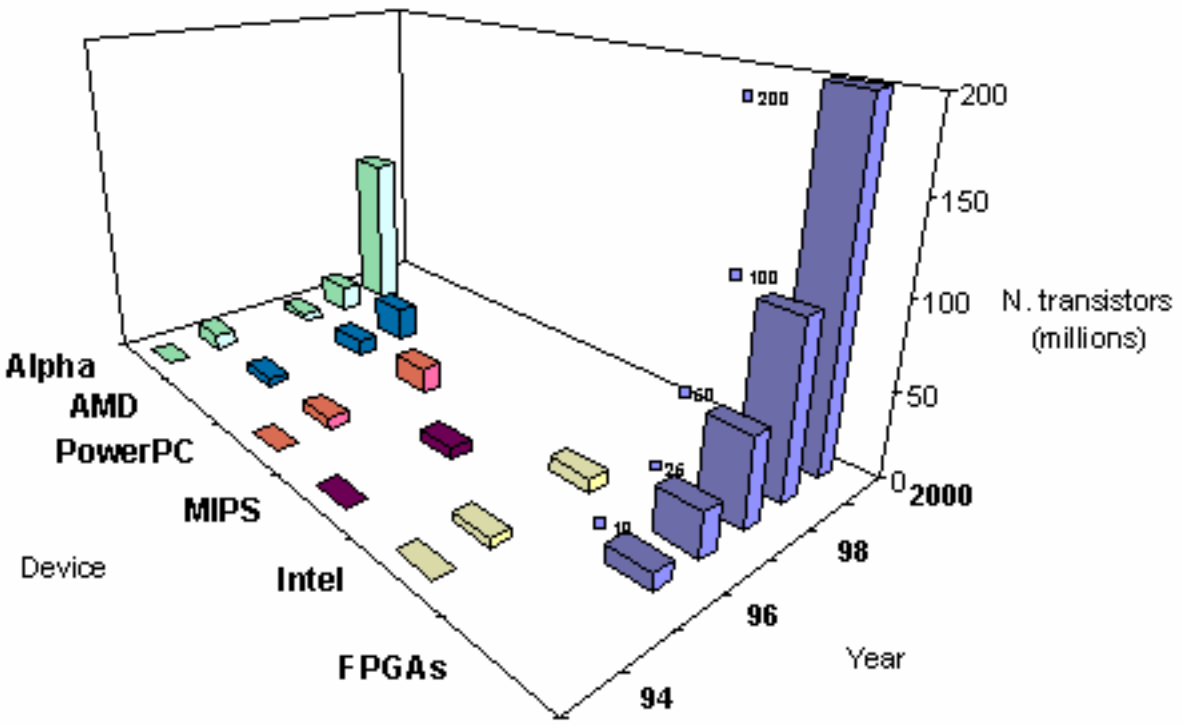



Figure 2. The growing of transistor density in processors [24] and FPGAs devices.

**Virtual Hardware**

One of the major disadvantages of RC is the impossibility of migrating only the portion of the application which does not exceed the total size of the used RPUs to hardware. Recently, a concept similar to virtual memory was adapted to allow multiple configuration RAM sets which are stored in an RPU to be used. The mapping of hardware units in RPUs without minimal hardware resources can be done with a temporal partition. This new concept is addressed in [21, 9]. However, since temporal partitioning techniques are not yet mature, the practical implications for hardware acceleration have not been found, and the algorithms proposed so far are not suitable for the acceleration of programs in an RC environment. There is more research effort on the concept of virtual hardware in the field of rapid prototyping of large hardware circuits. Researchers in this field do not consider memory accesses, because the input is a pure hardware description. This means that special effort must be made in order to partition and schedule large graphs obtained from high-level programming languages.

The virtual hardware concept is implemented by time-sharing a given RPU. It needs a scheduler that is responsible for the configurations, the execution, and the communications between temporal partitions.

RC systems and parallel computing systems have the same objective: to speed-up the execution of a given application. For parallel computations, acceleration is achieved through the exploitation of the parallelism in a program and its mapping onto an architecture of various processors, but for RC it is achieved through the migration of the most computationally-intensive parts of the program to RPUs. With RCs the exploitation of the parallelism of a given application is through the concurrent model of execution (hardware), instead of the sequential Von Neumann model of computation used by traditional parallel computation.

Scientific journals devoted to general research have already highlighted this field [25] and during the last few years, numerous approaches to RC have appeared in special topical conferences [1, 2, 3]. However, aside from new architectures that have been proposed and new concepts that have been presented, the utilization of these systems has been reported only for academic research. As discussed in [22], a methodology to automatically exploit the RC target architecture must be supported by compilers for RC systems to become widespread. Until effective support for these systems is available, software programmers will not be attracted to RC because the implementation of the hardware parts of the application is time consuming and needs hardware specialists.

The development of applications in an RC environment requires the use of one of the two following methodologies:

- the use of hardware objects (cores) bought from specific vendors or developed by hardware designers, with a specific interface to the software language used. This can be suitable for developing applications that need standard functionality such as FFT, JPEG/MPEG encoders/ decoders, etc.
- the use of a tool to compile from a high-level software programming language to the object code to run on a processor and the configuration files to program the RPUs. State-of-the-art techniques from the compiler and design automation worlds must be combined to achieve performance improvements. In particular, special care must be devoted to loop transformations, such as the exploitation of partial and total loop unrolling, to automatically achieve better acceleration factors. A hardware library of operators commonly used by high-level software programming languages must allow the specialized synthesis techniques to take advantage of the partial and on-the-fly reconfiguration characteristics of the target hardware.

The integration of HLS [10] techniques in the methodology to target the RC systems is the best known technique. This will permit the rapid generation of the hardware with the use of a library of basic hardware units. However, traditional HLS techniques, which target ASIC implementations, must be redefined because the target technology does not have the layout freedom of ASICs and the objective of accelerating the given application in the RC environment must prevail. The time spent in **HLS**, logic synthesis, placement, and routing, is unacceptable in an RC environment. To be adopted as a new

computing paradigm, it is necessary to decrease the overall compilation time, to provide efficient support for virtual hardware, and to provide the use of RPUs tightly coupled to the main processor.

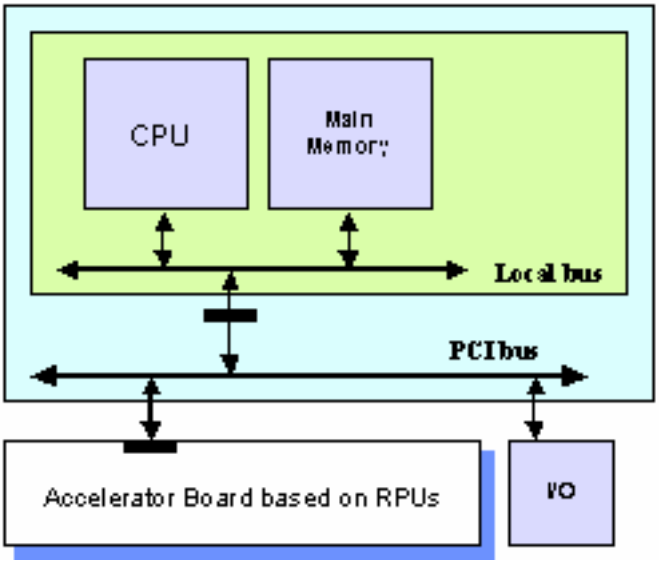
# RC Architectures Design Space

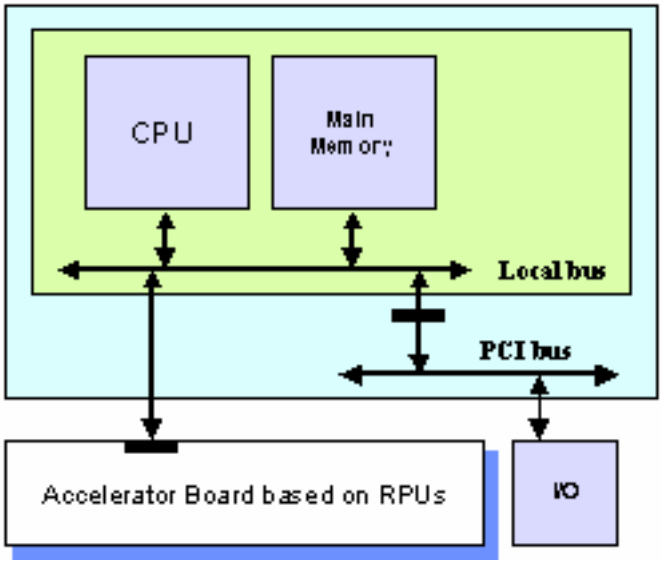
The type of interconnection between the RPU and the host system and the level of granularity of the RPU constitute a wide design space to be explored. Many points of this design space have already been explored and many more are left for further research. However, it is still not possible to identify a dominant solution for all kinds of applications. The next two sections will review some of the architectures used in RC systems.

## Coupling RPUs To The Host

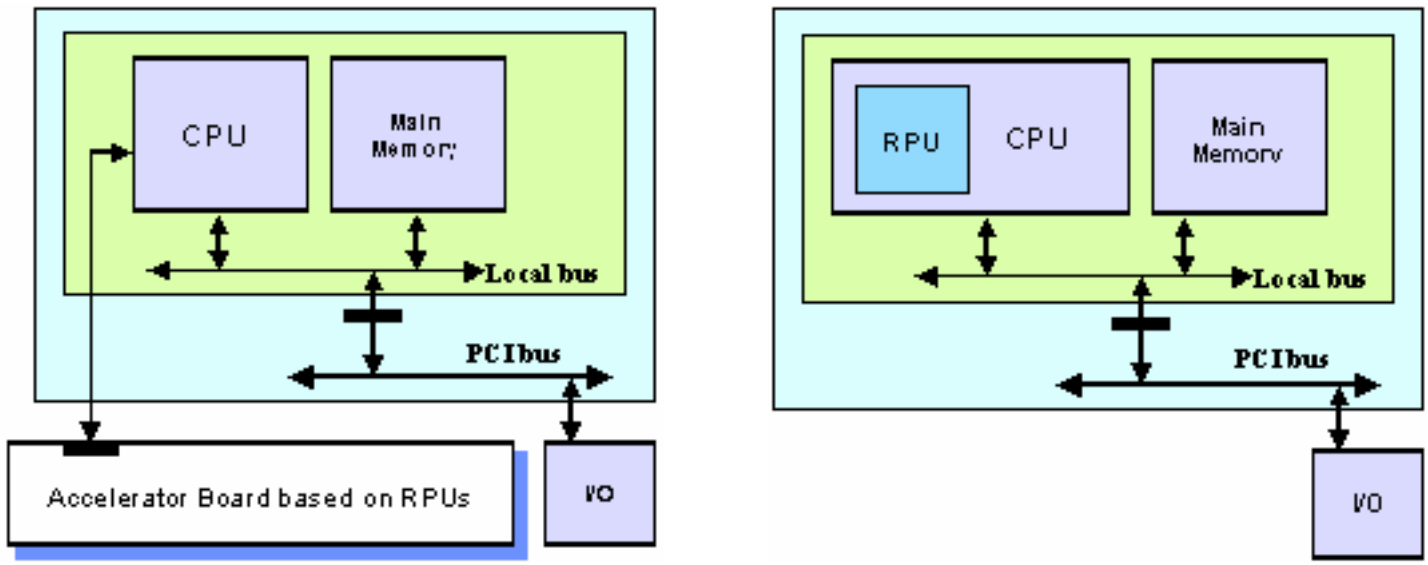
The type of coupling of the RPUs to the existing computing system has a big impact on the communication cost. The coupling can be classified into one of the four groups listed below, which are presented in order of decreasing communication costs:

- RPUs coupled to the I/O bus of the host (Figure 3.a). This group includes many commercial circuit boards. Some of them are connected to the PCI (**Peripheral Controller Interface**) bus of a PC or workstation.
- RPUs coupled to the local bus of the host (Figure 3.b)
- RPUs coupled like co-processors (such as the REMARC - **Reconfigurable Multimedia Array Coprocessor** - [18]) (Figure 3.c)
- RPUs acting like an extended data-path of the processor (such as the OneChip [27], the PRISC - **Programmable Reduced Instruction Set Computer** - [23], and the Chimaera [13]) (Figure 3.d);



**a) RPU coupled to the I/O system bus** **b) RPU coupled to the local bus**




**c) RPU coupled to the CPU** **d) RPU integrated in the process chip**

Figure 3. Organization of RC systems with respect to the coupling of the RPU to the host computer

## RPUs Granularity

Most current research efforts use FPGAs and directly manipulate their low-level processing elements. An FPGA cell typically consists of a flip-flop and a function generator that implements a boolean function of up to 4 variables. These elements can be used to implement nearly any digital circuit. This fine grain parallelism has been shown to be inefficient in time and area for certain classes of problems. Some ongoing research uses **Arithmetic-Logic Units** (ALUs) as the basic hardware primitives of the RC system. In some cases this approach can provide more efficient solutions, but somewhat limits the flexibility of the system. The approaches using an abstraction layer of coarse-grain granularity (operand level) over the physical fine-grain granularity of an FPGA have more flexibility but the compilation time is larger.

This abstraction level is provided with the use of a library of uni-functional **Relatively Placed Macros** (RPMs) with direct correspondence to the operators of the high-level software programming languages (arithmetic, logical, and memory access). This library can have more than one implementation of the same operator (different relations of area/latency/configuration time). Furthermore, the use of a library will decrease the overall RW compilation time, will make possible more accurate estimations, and are already a proven concept in HLS systems.

Various architecture concepts have been considered for use in RPUs, with differences in the flexibility and level of granularity used:

- The use of dynamic processor cores in the FPGA, such as the DISC (**Dynamic Instruction Set**

**Computer**). This approach permits the dynamic reconfiguration of dedicated hardware units when a new instruction appears whose corresponding unit has not yet been configured [26]. This processor reconfigures the new instructions at runtime. A more flexible solution is the use of a processor with a single MOVE instruction, such as the URISC (**Ultimate RISC**), as proposed in [7].

- The utilization of RPUs with medium granularity like the **Reconfigurable Data Path Array** (RDPA) [11]: These RPUs are designated as **Field-Programmable ALU Arrays** (FPAAs) and have ALUs as reconfigurable blocks (see Figure 4). These 32 bit ALUs can be reconfigured to execute some of the operators of the high-level language C. [6] presents a framework to deal with this type of RPUs, the CoDe-X. Its objective is to automatically map the more suitable portions of a C program to the RPDAs. Results achieved with this approach are shown in [11], and speed-ups (compared to the execution of the programs without CW support) of 13 for a JPEG compressor algorithm, and 70 for a two-dimension FIR (**Finite Impulse Response Filter**) are reported.

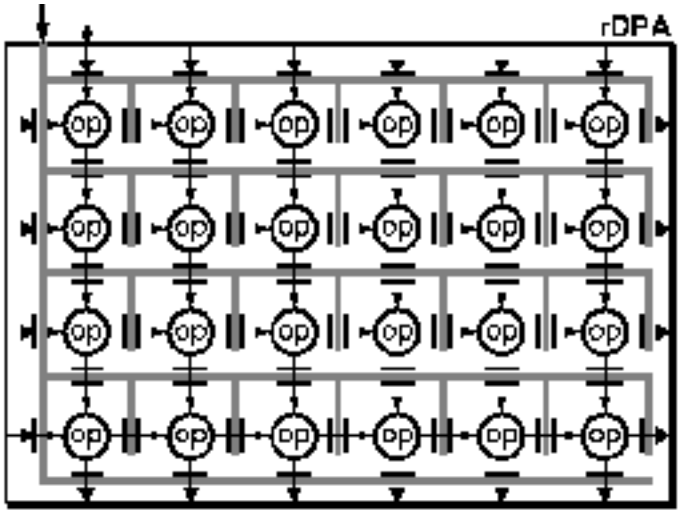



Figure 4. The matrix of reconfigurable datapaths of the Xputer system.

- A coarse-grain approach such as the MATRIX project [17], which is based on an array of identical 8-bit functional units and a reconfigurable network: Each functional unit contains memory, an ALU, a multiplier unit, and control logic.

With the existence of RPUs that permit partial reconfiguration, it is possible to reconfigure regions of the RPU while others are executing. These types of RPUs have many advantages over traditional FPGAs. One of the FPGAs especially suitable to RC is the XC6200 series of Xilinx(tm) [28]. It has very good properties such as a feature that allows the host processor to access the internal registers of the FPGA by address (like a typical memory access) without the need for any special routing or wasting of cells. It has a symmetrical internal structure, which allows units to be mapped independently of the position, and which configures 1,000 times faster than the traditional internal structures. A proposed reconfiguration based on layers of reconfiguration selected by a context switch has been addressed in [12].

# Compilation Techniques Suitable for RC

As detailed in [15], to develop an effective RW> compiler it is necessary to understand the state of the art in two research fields: compilers and ECAD (**Electronic Computer-Aided Design**). A number of data-flow techniques used by software compilers [4, 19] to generate object code can be used to generate hardware images with faster execution: operator strength reduction, dead code elimination, common sub-expression elimination, constant folding (constant propagation), variable propagation (copy propagation), tree-height reduction, memory accesses reduction (e.g., the elimination of redundant memory accesses over loop iterations), function inlining, etc.

---

## Example 1:

The operator strength reduction can be implemented on division and multiplication by constants. The simple cases occur when there is multiplication or division of an operand by a constant which is a power of two (accomplished by a simple shifting of the non-constant operand). In the general case, the multiplication by a constant can be transformed into a series of shifts and additions/subtractions (Figure 5). This produces very good results because of the smaller and faster characteristics of shifters and adders/subtracters than multipliers when implemented in the RPU.

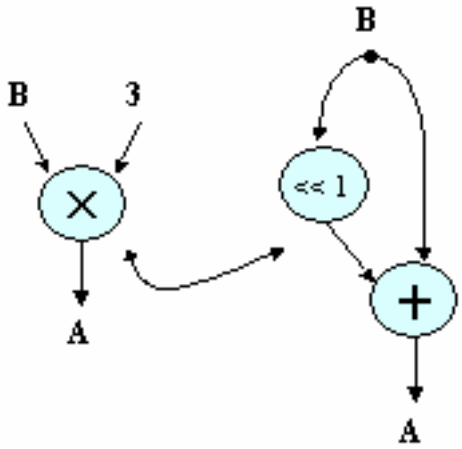



Figure 5. Operation strength reduction of the example: A = 3 x B (where A and B are integers).

---

These techniques produce better results when they are used with some type of loop transformation technique because of the repetitive nature of these structures. At the loop level, many transformations are used by software compiler designers [19]; they include strip mining or loop tiling, loop fusion, loop splitting or loop distribution, loop interchanging, loop permutation, loop reversal, loop skewing, loop peeling, and loop unrolling or loop expansion.

The loop unrolling of deterministic (statically bound) cyclic structures can achieve a large gain in performance, because of the elimination of the loop control and by making possible the use of the above data-flow techniques. However, loop unrolling, when mapped to hardware, can produce an unacceptable hardware area. Thus, it is important to exploit the loop unrolling factor together with the temporal partitioning of the large unrolled loops to make mapping to a single or multiple RPU(s) possible.

The exploitation of potential parallelism and scheduling algorithms in loop unrolling with different unrolling factors is not analyzed by any author. Furthermore, loop regions may contain many memory accesses which constrain the scheduling of operations because the system architecture usually has only one-port memories. Thus, the optimization of memory accesses can have great impact on the achieved performance.

---

## Example 2:

Figure 6.a)shows a simple example of a loop that sums 4 elements of an array. Illustrated in Figure 6.b)is the code after the unrolling of the loop. Shown in Figure 7is the scheduling of the **Data Flow Graph**(DFG) related to the unrolled loop. In this case, the tree-height transformation of the given DFG before scheduling produces worse results than the scheduling directly over the DFG. This shows that the compiler transformations must be performed with care to improve the obtained results and the majority of timings must be technology-driven.

```
int Sum = 0;

for(int i=0;i<4;i++)
{
   Sum += a[i];
}
```

a)

```
int Sum;

Sum = a[0] + a[1] + a[2]
+ a[3];
```

b)

Figure 6. A loop unrolling example which sums four array elements:
a) Initial code b) Final code

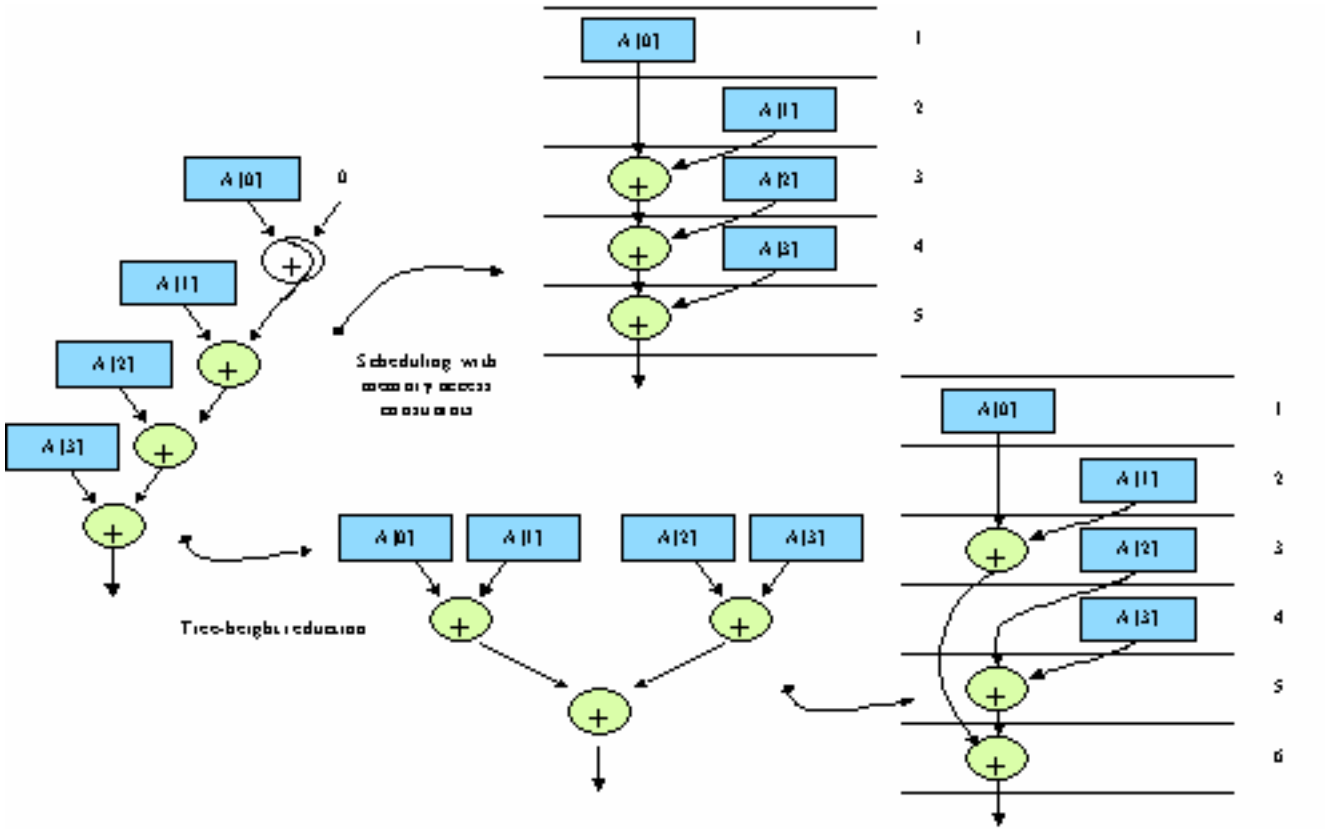



Figure 7. Scheduling operations from the unrolled loop under memory access constraints.

---

## Reconfigware/Software Compilation

The compiler for RC systems has as input the software program of a given application and produces two images: the executable model for the CPU and the configurations for the **FPU**s, as shown in the compilation flow of Figure 8.

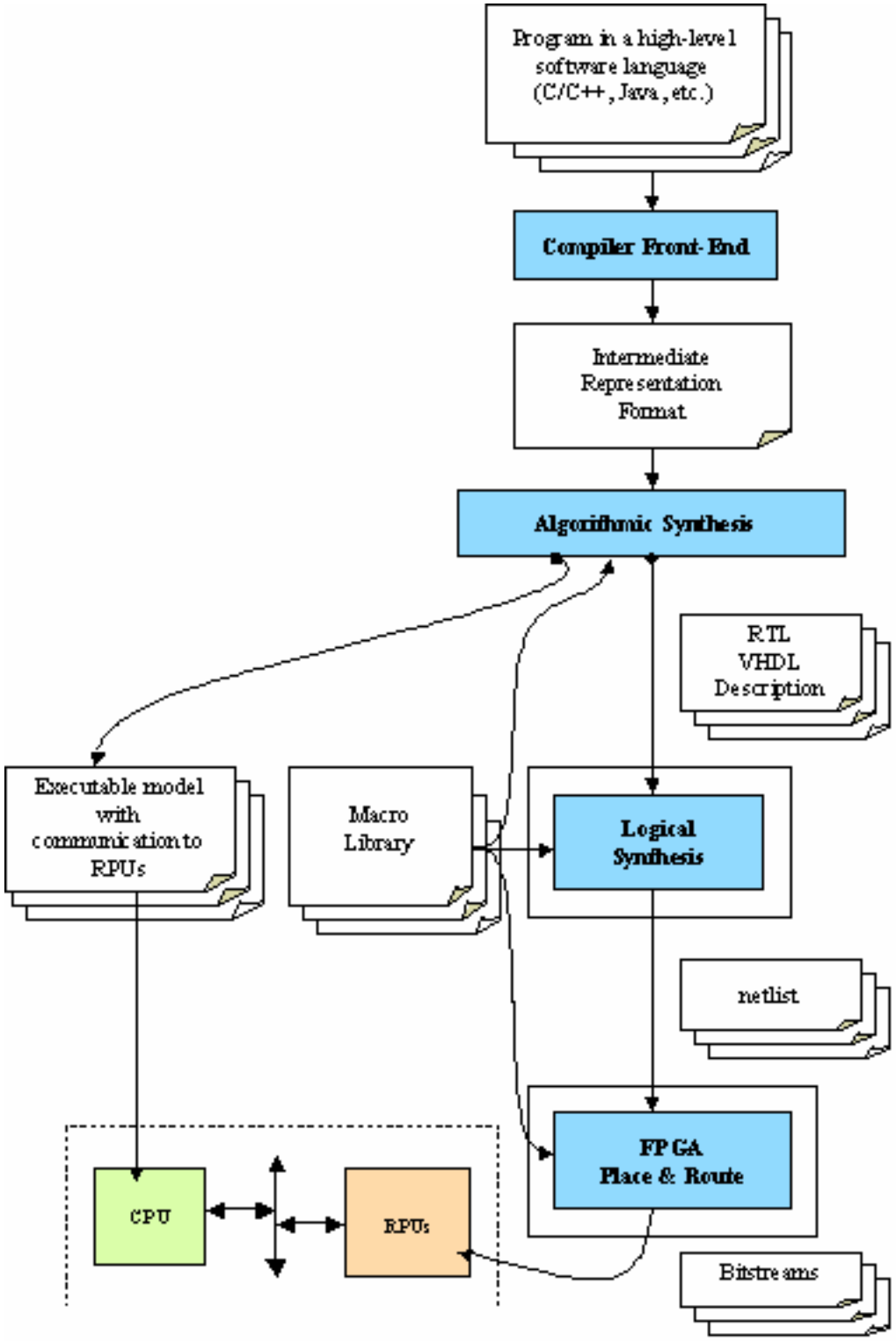



Figure 8. Compilation flow of a software program to run on a RC system.

The intermediate format can be obtained by front-end compilers of high-level software programming languages. These graph representations can also be obtained, for example, from the intermediate model used by the GNU C/C++ compiler. The compiler must be an integrated tool suitable to the programming of RC systems.

The compiler must generate a structural hardware representation (such as VHDL-RTL) that represents the connections between units contained in a library of RPMs, with direct correspondence to the

operators of high-level programming languages (such as Java(tm)). Unlike generic HLS techniques, allocation for partial hardware reconfiguration requires that different digital blocks have identical ``shapes'' and similar interfaces in order to enable temporal sharing. Therefore, the representation of the hardware operators will include specific attributes to specify the relative locations and shapes of the corresponding hardware which will be taken into account by the placement and routing tool. Moreover, the system will generate a description of the control unit of the runtime scheduler of reconfigurations to the same RPU. By using temporal partitioning, the host computer will view the set of RPUs as an unlimited hardware resource.

**The GALADRIEL Compiler**

Members of our research group are working on a front-end compiler called GALADRIEL[8] which analyzes the Java(tm) class files produced from the Java compiler and processes the information in order to exploit the implicit parallelism in each method so that it can be efficiently implemented by multiple hardware and/or software components.

After the programmer has identified the more critical functions in the program (with the help of profiling, for example) the compiler will find the code segments of the functions selected to migrate to the RPUs. The compiler will be able to evaluate different levels of granularity of the code specification, from basic block (definition used in [4]) to the overall function, so that the programmer does not need to reprogram the application to make it suitable for the compiler.

Bytecodes are produced for the software image. These bytecodes include calls to the communication library implemented in C/C++. Each hardware image related to a configuration is described in VHDL, which defines the units used from the library set, the interconnections between them, and when the scheduling of operations is necessary, the control part (sharing of resources such as operators or memory access units). Then these descriptions are passed to a VHDL-based logic synthesis tool (such as the Synopsys Design Compiler(tm)), which has access to the units library based on the **FPGA** used. The synthesis tool translates each hardware description into a list of primitive cells supported by the FPGA technology library in a format accepted by the placement and routing tools of the given FPGA family.

Generic **HLS** systems [10] that target ASIC implementations have been shown inadequate for RC systems. Technology driven architectural synthesis algorithms must be developed with the following specific objectives in mind:

- The generation of hardware images with the lowest execution time possible by exploiting the existence of the potential parallelism of the given application.
- Temporal partitioning of the graph segment that corresponds to hardware when there is not sufficient available area. This case implies the introduction of communication between partitions and the control of the scheduling of different configurations, as well as partial executions.
- The control of **Direct Memory Accesses** (DMA) on each partition. Since the memory accesses are quantitatively of great importance in the computer systems [14], alternative solutions for mapping arrays on the host computer or on local memory (board of RPUs) must be evaluated.
- Exploitation of loop unrolling techniques targeting theoretical unlimited hardware.

# Conclusion

One of the difficulties of research in this field is that each author uses his own examples, often publicly unavailable, so results from different implementations are impossible to compare. The majority of work starts from programming languages with little or no utilization in the real-world and frequently uses hardware description languages which prevent the use of the compiler tools by software programmers.

Since RC systems are partially handcrafted, it is very important to develop a Reconfigware/Software compiler tool that receives as input a software program described with a high-level of abstraction. To make this possible it is necessary to close the gap between the software and the hardware model with advanced compiler transformations.

A significant speed-up can be achieved only by using a compiler capable of extracting and exploiting the massive amounts of parallelism existent in the input application.

We have described briefly some types of architectures to support the RC concept. However, adopting a particular RPU granularity or abstraction level is not trivial because of the lack of comparable implementation results for a set of applications.

# Acknowledgements

We would like to acknowledge our research advisor Prof. Horácio Neto for his guidance and assistance, and Virginia Chu, who helped us to improve the English in this paper. We would also like to acknowledge the support given by: the Portuguese Ph.D. program of the Prodep 5.2 action, and the Portuguese Ph.D. program of the Praxis XXI.

# Glossary

**Reconfigurable Computing**
: A kind of computing based on custom computing machines, which combines high-density

FPGAs along with processors to achieve the best of both worlds.Definition by Danish Bathia: ``An ability for software to reach through the hardware layers and change the data-path for optimising the performance".

**Configuration**

Set-up the logic in the cells and the routing between them.

**Reconfigurable Devices**

Logic devices that can be customised once or as much as needed. In contrast to configurable devices which can be customised only once.

**Dynamic Reconfiguration**

The concept of customisation on-the-fly.

**High-Level Synthesis (HLS)**

High-Level Synthesis: also known as *behavioural synthesis* or *architectural synthesis*. Definition in [10]: ``A transformation of a behavioural description into a set of connected storage and functional units".

**Temporal Partition**

The ability to map a function which exceeds the available space of the reconfigurable device using time sharing.

**Spatial Partition**

The ability to partition a function onto a set a reconfigurable devices.

**Equivalent Gates**

It is the number of transistors of the considered circuit divided by the number of transistors of a 2-input NAND gate (usually four transistors in a CMOS technology).

**Hardware/Software Co-Design**

Concurrent design of hardware and software.

**VHDL**

**V**HSIC (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuit) **H**ardware **D**escription **L**anguage.

**SRAM FPGA**

A kind of FPGA based on static memory technology that is re-programmable and in-system programmable and requires external boot devices.

# References


**1**

___, Proceedings of the International Workshop on Field-Programmable Logic and Applications. 1991-1998. Printed by Springer-Verlag. See http://xputers.informatik.uni-kl.de/FPL/index_fpl.html

**2**

___, Proceedings of the Reconfigurable Architectures Workshop. 1994-1998. Printed by Springer-Verlag. See http://xputers.informatik.uni-kl.de/RAW/index_raw.html

**3**

___, Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines. 1992-1998. Printed by IEEE Computer Society Press, Los Alamitos, Calif. See http://www.fccm.org/.

**4**

Aho, A. V., Sethi, R., Ullman, J. D. *Compilers: Principles, Techniques and Tools*. Addison Wesley, 1986.

**5**

Athanas, P., and Silverman, H. Processor Reconfiguration through Instruction-Set Metamorphosis: Architecture and Compiler. *IEEE Computer*, vol. 26, n. 3, March 1993, pp. 11-18.

**6**

Becker, J., Hartenstein, R., Herz, M., Nageldinger, U. Parallelization in Co-Compilation for Configurable Accelerators. In *Proceedings of the Asia South Pacific Design Automation Conference (ASP-DAC'98)*, Yokohama, Japan, February 10-13, 1998.

**7**

Brebner, G., Donlin, A. Runtime Reconfigurable Routing. In *Proceedings of the Reconfigurable Architectures Workshop (RAW'98)*, Orlando, Florida, USA, March 30, 1998.

**8**

Cardoso, J. M. P., and Neto, H. C. Towards an Automatic Path from Java(tm) Bytecodes to Hardware Through High-Level Synthesis. In *Proceedings of the 5th IEEE International Conference on Electronics, Circuits and Systems (ICECS-98)*, Lisbon, Portugal, September 7-10, 1998, pp. 85-88.

**9**

GajjalaPurna, K. M., and Bhatia, D. Temporal Partitioning and Scheduling for reconfigurable Computing. In *Proceedings of the 6th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'98)*, Napa Valley, California, USA, April 15-17, 1998.

**10**

Gajski, D. D., Dutt, N. D., Wu, A. C.-H., Lin, S. Y.-L. *High-Level Synthesis, Introduction to Chip and System Design*. Kluwer Academic Publishers, Boston, Dordrecht, London, 1992.

**11**

Hartenstein, R. W., Becker, J., et al. High-Performance Computing Using a Reconfigurable Accelerator. In *CPE Journal*, Special Issue of Concurrency: Practice and Experience, John Wiley & Sons Ltd., 1996.

**12**

Hartenstein, R., Herz, M., Hoffmann, T., Nageldinger, U. On Reconfigurable Co-Processing Units. In *Proceedings of the Reconfigurable Architectures Workshop (RAW'98),* Orlando, Florida, USA, March 30, 1998.

**13**

Hauck, S., Fry, T. W., Hosler, M. M., Kao, J. P. The Chimaera Reconfigurable Functional Unit. In *Proceedings of the 5th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'97),* Napa Valley, California, USA, April, 1997, pp. 87-96.

**14**

Hennessy, J. L., Patterson, D. A. *Computer Architecture: A Quantative Approach.* Morgan Kaufman Publ., 1990.

**15**

Mangione-Smith, W. H., et al. Seeking Solutions in Configurable Computing. *IEEE Computer* 30,12 December 1997, pp. 38-43.

**16**

Micheli, G. De, Gupta, R. Hardware/Software Co-Design. In *Procedings of the IEEE,* vol. 85, no.3, March 1997, pp.349-365.

**17**

Mirsky, E., and DeHon, A. MATRIX: A Reconfigurable Computing Device with Configurable Instruction Deployable Resources. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines,* Napa Valley, California, USA, April 17-19, 1996, pp. 157-166.

**18**

Miyamori, T., and Olukotun, K. A Quantative Analysis of Reconfigurable Coprocessors for Multimedia Applications. In *Proceedings of the 6th IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'98),* Napa, California, USA, April 15-17, 1998.

**19**

Muchnick, S. *Advanced Compiler Design and Implementation.* Morgan Kaufmann Publishers, Inc., San Francisco, California, USA, 1997, ISBN 1-55860-320-4.

**20**

Olukotun, K. A., Helaihel, R., Levitt, J., and Ramirez, R. A Software-Hardware Cosynthesis Approach to Digital System Simulation. *IEEE Micro*, vol. 14, Nov. 1994, pp. 48-58.

**21**

Ouaiss, I., Govindarajan, S., Srinivasan, V., Kaul, M., and Vemuri, R. An Integrated Partioning and Synthesis System for Dynamically Reconfigurable Multi-FPGA Architectures. In *Proceedings of the Reconfigurable Architectures Workshop (RAW'98)*, Orlando, Florida, USA, March 30,1998.

**22**

Radunovic, B., Milutinovic, V. A Survey of Reconfigurable Computing Architectures. To appear as a tutorial in the *8th International Workshop on Field Programmable Logic and Applications (FPL'98)*, Talin, Estonia, 30 August - 2 September, 1998.

**23**

Razdan, R., Brace, K., and Smith, M. D. PRISC Software Acceleration Techniques. In *Proceedings of the IEEE International Conference on Computer Design*, Oct. 1994, pp. 145-149.

**24**

See General Processor Information, http://infopad.eecs.berkeley.edu/CIC/summary/local.

**25**

Villasenor, J., Mangione-Smith, W. H. Configurable Computing. *Scientific American*, June 1997, pp. 66-71. See http://www.sciam.com/0697issue/0697villasenor.html

**26**

Wirthlin, M. J., and Hutchings, B. L. A Dynamic Instruction Set Computer. In *Proceedings of the 4th IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'95)*, Napa Valley, California, USA, April 19-21, 1995, pp. 99-107.

**27**

Witting, R. D., and Chow, P. OneChip: An FPGA Processor with Reconfigurable Logic. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Napa Valley, California, USA, April 17-19, 1996, pp. 126-135.

**28**

Xilinx Inc. XC6000 Field Programmable Gate Arrays. Version 1.10, April 24, 1997. http://www.xilinx.com.

**29**

Xilinx, Inc., "The Virtex Series of FPGAs". See <http://www.xilinx.com/products/virtex>.

*João M. P. Cardoso (Joao.Cardoso@inesc.pt) is currently a Ph.D. student at the Instituto Superior Técnico in Lisbon, Portugal, working on the compilation of Java programs to custom computing machines. His research interests include system-level synthesis, high-level synthesis, hardware/software*

*co-design, and reconfigurable computing.*

*Mário P. Véstias (mpv@hobbes.inesc.pt) is currently a Ph.D. student at the Instituto Superior Técnico in Lisbon, Portugal, working on co-synthesis of embedded real-time systems. His research interests include hardware/software co-design, real-time systems, and reconfigurable computing.*