Using Reconfigurable Supercomputers and C-to-hardware Synthesis for CNN Emulation

J. Javier Martínez-Álvarez, F. Javier Garrigós-Guerrero, F. Javier Toledo-Moreo, J. Manuel Ferrández-Vicente

Dpto. Electrónica, Tecnología de Computadoras y Proyectos,
Universidad Politécnica de Cartagena, 30202 Cartagena, Spain
jjavier.martinez@upct.es

Abstract. The complexity of hardware design methodologies represents a significant difficulty for non hardware focused scientists working on CNN-based applications. An emerging generation of Electronic System Level (ESL) design tools is been developed, which allow software-hardware codesign and partitioning of complex algorithms from High Level Language (HLL) descriptions. These tools, together with High Performance Reconfigurable Computer (HPRC) systems consisting of standard microprocessors coupled with application specific FPGA chips, provide a new approach for rapid emulation and acceleration of CNN-based applications. In this article CoDeveloper, and ESL IDE from Impulse Accelerated Technologies, is analyzed. A sequential CNN architecture, suitable for FPGA implementation, proposed by the authors in a previous paper, is implemented using CoDeveloper tools and the DS1002 HPRC platform from DRC Computers. Results for a typical edge detection algorithm shown that, with a minimum development time, a 10x acceleration, when compared to the software emulation, can be obtained.

1 Introduction

Cellular Neural Networks (CNNs) based on analogue cells are very efficient for real-time image processing applications. Analogue CNN chips present however a complex implementation and a high development cost. This facts make them appropriate just for applications where few layers are sufficient and great amounts of data must be processed in real time. Another disadvantage is their low precision, due to undesirable noise effects, or flaws and tolerances in their components derived from the manufacturing processes. This has favored the search for new approaches for the implementation of CNN architectures.

One important step in this evolution has been the development of reprogrammable neural networks. This kind of network, known as CNN-UM (CNN universal machine) was conceived as a bidimensional array of processing elements that increase the functionality of the standard CNN model adding new analogue and digital blocks to the cells and making them reprogramable. The ACE16K[1] is the last generation of devices with the functionality of the CNN-UM model.

These mixed signal chips were designed using standard 0.35u technology and integrate a net of 128×128 cells providing a total processing power of 330GOPS.

During the last decade, the tendency of using reconfigurable hardware (FPGA) has taken an increasing interest. These devices improve precision and design flexibility, while simultaneously reducing cost and developing time due to the nature of the devices and development tools provided. With clock frequencies an order of magnitude lower than that of typical microprocessors, FPGAs can provide greater performance when executing real-time video or image processing algorithms as they take advantage of their fine-grained parallelism. Thus, FPGAs has been commonly used as platforms for CNN emulation and acceleration[2–5]. However, the design process is not exempt of difficulties as traditional methodologies, based on hardware description languages (VHDL, Verilog, etc.) still require deep hardware skills from the designer.

Recently, a new generation of tools for highly complex circuit design is been developed. This new methodology, known as ESL (Electronic System Level), aims to target the problem of hardware-software co-design from system level, untimed descriptions, using different flavors of high level programming languages, such as C, C++ or matlab. An exhaustive taxonomy of the design methodologies and ESL design environments commercially or educationally available can be found in [6]. Also, a new generation of hybrid supercomputers, called HPRCs, is been developed to take full advantage of the new co-design tools. These HPRC systems provide, the standard microprocessor nodes, plus new closely-coupled reconfigurable nodes, based on FPGAs chips.

In this paper, we propose a discrete sequential CNN architecture for easy prototyping and hardware acceleration of CNN applications on programmable devices. We show the results obtained when accelerating a typical CNN-based edge detection algorithm on a DS1002, a HPRC platform from DRC Computers[7]. We then analyze CoDeveloper$^{\mathrm{TM}}$, an ESL IDE from Impulse Accelerated Technologies, Inc.[8] used for hardware-software co-design, to evaluate its suitability for the non-hardware specialist scientist, and provide some keys to get better results with these kind of tools. Finally conclusions and future work is exposed.

2 ImpulseC programming model

ImpulseC uses the communicating sequential process (CSP) model. An algorithm is described using ANSI C code and a library of specific functions. Communication between processes is performed mainly by data streams or shared memories. Some signals can be transfered also to other processes like flags, for non continuous communication. The API provided contains the necessary functions to express process parallelization and communication, as standard C language does not support concurrent programming.

Once the algorithm has been coded, it can be compiled using any standard C compiler. Each of the processes defined is translated to a software thread if the

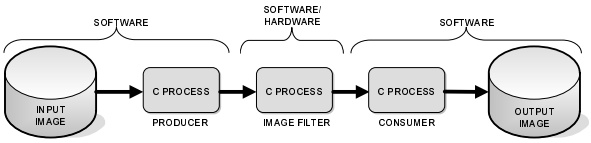

Fig. 1. Typical ImpulseC model

operating system supports them. Other tools do not have this key characteristic, and can only compile to hardware.

The entire application can then be executed and tested for correctness. Debugging and profiling the algorithm is thus strait forward, using standard tools. Then computing intensive processes can be selected for hardware synthesis, and the included compiler will generate the appropriate VHDL or Verilog code for them, but also for the communication channels and synchronization mechanisms. The code can be generic of optimized for a growing number of commercially available platforms. Several *pragmas* are also provided that can be introduced in the C code to configure the hardware generation, for example, to force loop unrolling, pipelining or primitive instantiation.

The versatility of their model allows for different uses of the tool. Let's consider a simple example, with 3 processes working in a dataflow scheme, as shown in Figure 1. In this case, Producer and Consumer processes undertake just the tasks of extracting the data, send them to be processed, receive the results and store them. The computing intensive part resides in the central process, that applies a given image processing algorithm. A first use of the tool would consist in generating application specific hardware for the filtering process, that would be used as a primitive of a larger hardware system. The Producer and Consumer would then be "disposable", and used just as a testbench to check first, the correct behavior of the filtering algorithm, and second, the filtering hardware once generated.

A different way of using the tool could consist in generating an embedded CPU accelerated by specific hardware. In this case, Producer and Consumer would be used during the normal operation of the system, and reside in an embedded microprocessor. The filter would work as its coprocessor, accelerating the kernel of the algorithm. CoDeveloper generates the hardware, and resolves the software-to-software and hardware-to-hardware, communication mechanisms, but also the software-to-hardware and hardware-to-software interfaces, for a number of platforms and standard buses. This is a great help for the designer that gets free of dealing with the time-consuming task of interface design and synchronization.

Finally, the objective can be accelerating an external CPU by means of a FPGA board. In this case, the software processes would reside on the host microprocessors, that would communicate to the application specific hardware on the board by means of a high performance buses (HyperTransport, PCI, Gigabit Ethernet, etc.).

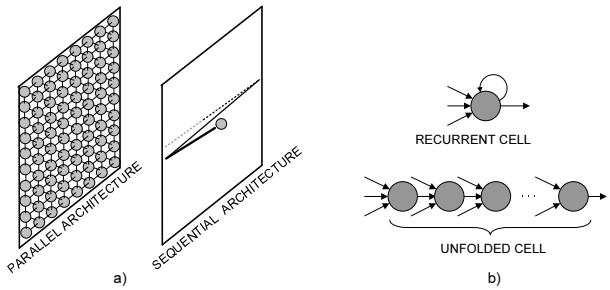

Fig. 2. a) Different architectures used to implement a CNN. The parallel architecture implements a complete CNN, while the sequential uses a single cell which moves on the input array to process the information. b) different types of cell: recurrent and unrolled

3 Proposed discrete sequential CNN architecture

Regardless of the language used, implementing a CNN on an FPGA requires to take into account a number of considerations. Conceived as a massively parallel array of analogue processors [9], the original CNN model must be transformed for digital implementation on an FPGA. First, it is necessary to translate their continuous nature into the discrete domain, providing an approximation with sufficient accuracy that minimizes the hardware resources. The implementation will depend on the application type, size of the data array and processing speed restrictions. Two different architectures, sequential or parallel, can be used to emulate CNN-based systems.

Parallel architectures devote specific hardware resources to implement each of the CNN cells. As in analog chips, these architectures provide a complete implementation of the CNN, which gives them the highest degree of parallelism and processing speed. On the other hand, they require larger amount of hardware resources, which limits their implementation to a few tens of cells per FPGA. Larger CNNs would require several FPGAs, rising orders of magnitude the cost and complexity of the system.

Sequential architectures are the solution when the network is too large or it is not feasible to use multi-FPGA platforms. These architectures include just one or several functional units, multiplexed in time, to emulate the full processing of the CNN. The computation effect is equivalent to a single cell which shifts, from left to right and from top to bottom, in a similar way as an image is generated by a video camera. This sequential process allows using small buffers, instead of the large memories required by parallel architectures, reducing the cost of the system. Moreover, sequential architectures simplify the I/O interface, providing a serial communication, which simplifies the connection with other circuits, and allows for the development of multi-layer and multi-FPGA systems. Figure 2-a shows main differences between sequential and parallel architectures.

Given that CNN discrete models are recurrent algorithms, cells can be implemented as recursive or iterative, regardless of the network architecture (parallel or sequential). The recursive implementation uses a closed-loop circuit, while

in the iterative approach, the cell is unfolded in a number of sequential stages that can be implemented as separate circuits. Recurrent cells use less hardware, however, its processing speed will be lower because they have to run more recursive iterations per data. Iterative cells will be faster, as cell stages can work as pipelined circuits, but also consume more resources. Figure 2-b shows both types of cells.

The use of iterative cells in a sequential architecture establish a trade-off between area and speed that optimizes both, resource utilization and processing time. With respect to the parallel architecture, its sequentiality allow to emulate complex systems using simple I/O interfaces and smaller memory buffers. On the other hand, the implicit parallelism of the cells improves the processing speed, which enables their use in real-time applications.

An implementation of a CNN model suitable for hardware projection, using the proposed approach, was introduced in a previous paper[5]. This architecture is based on a discrete model of the CNN obtained from the method of Euler equations, whose dynamics are shown in equations 1 and 2.

$$X_{ij}[n] = \sum_{k,l \in Nr(ij)} A_{kl}[n-1]Y_{kl}[n-1] \; + \sum_{k,l \in Nr(ij)} B_{kl}[n-1]U_{kl} \; + \; I_{ij} \; , \tag{1}$$

$$Y_{ij}[n] = \frac{1}{2}\left(|X_{ij}[n]+1| - |X_{ij}[n]-1|\right) \tag{2}$$

A single cell is used to sequentially process full image information. The cell can be unfolded in a number of stages which depend on the application requirements. Each stage will have two sequential inputs and outputs used for connection with previous and following stages. Figure 3 shows the stage structure, formed by two 3×3 convolutions, a three-input adder and a comparator to resolve the activation function. The fixed-point convolution kernel has been efficiently designed using circular memories, three multipliers and the logic to resolve the network contour conditions (Dirichlet conditions). This architecture can emulate a complete CNN up to 1024×1024 cells processing grayscale images. Its correctiveness and efficiency has been validated in other studies [10, 11].

Next section shows the results obtained when implementing our model using a high level description on a HPRC. Our objective is to evaluate the performance of both, the new C-to-hardware synthesizers, and the software-hardware co-execution platforms, when accelerating CNN applications, by non hardware-focused scientists.

4 Evaluation platform

Traditional platforms, that use commodity FPGA boards that communicate with a host workstation using high speed interfaces (like USB, PCI or Ethernet), are the preferred solution for standalone or not highly-coupled applications. However, when accelerating algorithms using FPGA as coprocessors, the main

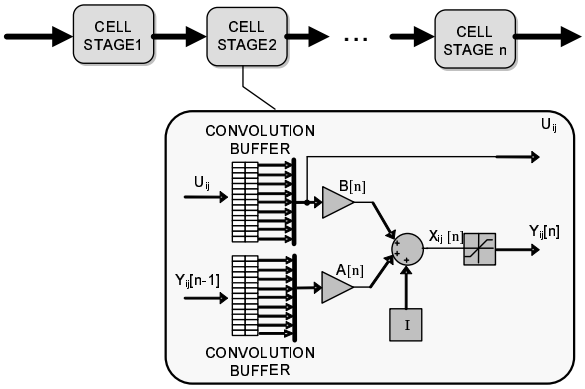

Fig. 3. Proposed architecture. Sequential unfolded cells, with details on stage structure

bottleneck usually comes from the communication between the software and hardware stages of the algorithm.

A new generation of High Performance Reconfigurable Computers (HPRC) are addressing this fact providing tightly coupled standard microprocessors and reconfigurable devices. Examples of these reconfigurable supercomputers are the SRC-7, the SGI Altix 350 and the Cray XD1[12–14]. These platforms have the potential to exploit coarse-grained parallelism, as well as fine-grained (known as instruction-level) parallelism, showing orders of magnitude improvement in performance, power and cost over conventional high performance computers (HPCs)[16–20].

In our case, the development platform DS1002 from DRC Computer Corporation was used to benchmark the proposed CNN architecture. This is a single server system that includes a standard PC workstation enhanced with a DRC Reconfigurable Processor Unit (RPU^{TM}). The DS1002 is a 2-way system with an AMD Opteron^{TM}Model 275 on one socket and a RPU110-L200 on the other. The RPU includes a Virtex-4 LX200, 2GB of DDR2 RAM and 128MB of low latency RLDRAM. Communication between the main processor and the FPGA board is carried out by 3 HyperTransport^{TM}(HT) links. The current HT interface is limited to 8bits $\times$ 400MHz (double data rate) providing a theoretical throughput of 800MB/s per direction, or agregated 1.6GB/s, for a total bandwidth of 9.6GB/s.

The testbed designed for the CNN implementation is shown in Figure 4. Every cell stage has been implemented as a single process. Producer and consumer processes were merged in a single process to maximize efficiency, as it just has to read image data, send pixels to the hardware processes, receive processed pixels and write images back to disk. Images were sized 640 $\times$ 480 pixels, coded 8 bits grey-scale.

The whole system was coded using standard ANSI C syntax and specific functions from the ImpulseC API for process intercommunication. Different versions

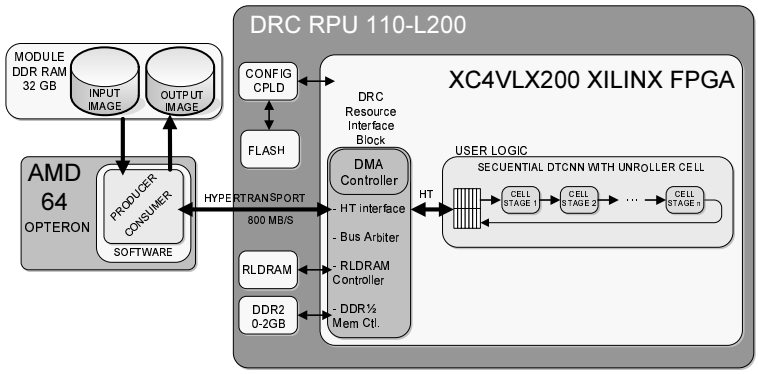

Fig. 4. Block diagram of the DRC Development System 1002 as used for the CNN implementation

of the cell, with 1, 2, 4, 8 y 16 cascaded stages (processes), were implemented to observe the effect on the precision and the processing speed.

The entire system was compiled using the standard *gcc* compiler[21] to executable software for both a conventional Windows XP^{TM}-based PC, and the Linux-based DS1002 (Kubuntu 6.06 LTS). Subsequently, the system was separated in software and hardware processes. The Producer-Consumer was compiled to be executed in the Opteron. The cell stage processes, however, were compiled to VHDL using ImpulseC tools. Finally, the RPU was programmed with the circuit synthesized from the VHDL description and combined with the Producer-Consumer for software-hardware co-execution. The obtained results are depicted in section 4.1.

4.1 Results

Table 1 summarizes the hardware resources consumed by each cell version, that spans from the 760 slices of the single-stage cell, to the 11201 of the most complex. The critical resource resulted to be multipliers, that in the case of the 16-stages cell (with 6 multipliers per stage) ends up 100% of the DSP48 blocks availables in this device.

Another important parameter, the number of clock cycles necessary to process a pixel, shown a perfect correspondence between the CoDeveloper debugger estimations and the real execution of the algorithm. Using 3 multipliers per convolution, each stage takes 27 cycles/pixel, a number that is maintained when the stage number increases, due to the pipelined behavior of the cell. We used a 133MHz clock for the CNN, as we met that, for this system, the HT provided enough bandwidth between microprocessor and RPU for clock frequencies under 200MHz.

Figure 5 shows processing time for CNN execution on three different platforms: all software Core2Duo^{TM}Windows XP^{TM}based, all software Opteron^{TM}Linux based, and hardware-software co-execution. Working with 640 $\times$ 480 pixel

Table 1. Summary of timing information and used resources for different stages. Percentages are referred to DRC-RPU FPGA (V4LX200-11).

Area(%used)	1 stage	2 stages	4 stages	8 stages	16 stages
Slices	760(0)	1455(1)	2847(3)	5657(6)	11201(12)
Flip flops	784(0)	1482(0)	2878(1)	5670(3)	11254(6)
DSP48	6(6)	12(12)	24(25)	48(50)	96(100)
BlockRAM	6(1)	10(2)	18(5)	34(10)	66(19)

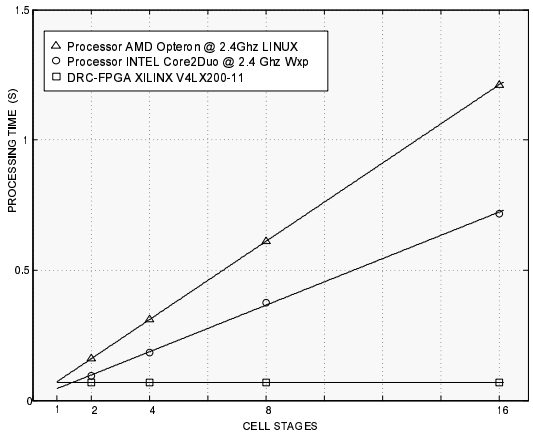

Fig. 5. processing time for the CNN applied to one 640×480 pixel image and diferent stage per cell on the three platforms.

images, the 2.4GHz Core2Duo is the fastest for the single-stage cell, with just 0.05s. Beginning with 2 stages cells and following, the FPGA accelerated DS1002 platform is faster, getting a constant mark of 0.07s. The linear behavior shown by the three platforms allows extrapolating the acceleration provided by the FPGA for any number of stages/cells.

This results show that, the proposed CNN architecture, applied to 640×480 pixel images, would process a theoretical maximum of $(133e6/27)/(640 * 480) = 16.03$frames/s. This is 10.25 times faster than a standard PC (Core2Duo, 2.4GHz), that takes 0.718s in processing an image with 16-stages cell.

5 Discussion

The results obtained in our first experiments with different CNN architectures show that this kind of algorithms can benefit from custom hardware coprocessors for accelerating execution, as well as for rapid prototyping from C-to-hardware compilers. However, to obtain any advantage, both, an algorithm profiling and a careful design are mandatory. These are the key aspects we have found to be useful:

– The algorithm should make an intensive use of data in different processing flows, to make up for the time spent in the transfer to/from the accelerator.

- The algorithm can make use of several data flows, taking advantage of the massive bandwidth provided by the several hundred o I/O bits that FPGA devices include.
- The working data set can be limited to 1-2MB, so that it may be stored in the internal FPGA memory, minimizing access to external memory.
- The algorithm should use integer or fixed point arithmetic when possible, minimizing the inference of floating point units that reduce the processing speed and devour FPGA resources.
- The algorithm must be profiled to identify and isolate the computational intensive processes. All parallelizing opportunities must be identified and explicitly marked for concurrent execution. Isolation of hardware processes means identifying the process boundaries that maximize concurrency and minimize data dependencies between processes, to optimize the use of on-chip memory.
- Maximize the data-flow working mode. Insert FIFO buffers if necessary to adjust clock speeds and/or data widths. This makes automatic pipelining easier for the tools, resulting in dramatic performance improvement.
- Array partitioning and scalarizing. Array variables usually translate to typical sequential access memories in hardware, thus if the algorithm should use several data in parallel, they must be allocated in different C variables, to grant the concurrent availability of data in the same clock cycle.
- Avoiding excessive nested loops. This could difficult or avoid correct pipelining of the process. Instead, try partitioning the algorithm in a greater number of flattened processes.

6 Conclusions

HPRC systems are showing greater performance with respect to other HPC approaches, particularly taking into account that they provide also increments of several orders of magnitude in the GFlops/euro and GFlops/watio ratios.

Our first experiments have demonstrated the viability of applying HPRC platforms and ESL tools to rapid prototyping of CNN-based image processing algorithms, provided that some requisites comply. Our first results, still under refinement, have shown a 10x acceleration for the hardware-software co-execution on a HPRC DS1002 from DRC Computers, with regards to the algorithm executed on the same machine as pure software.

Future work will be directed to the development of more complex algorithms, based on CNNs and standard DSP processing stages, for on-line stellar image acquisition and preprocessing, as part of our collaboration with the FastCam[22] initiative.

Acknowledgements

This work has been partially supported by the Fundacin Séneca de la Región de Murcia through the research projects 08801/PI/08 and 08788/PI/08, and by the Spanish Government through project TIN2008-06893-C03.

References

1. A. Rodriguez-Vazquez, G. Linan-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galan, F. Jimenez-Garrido, R. Dominguez-Castro, S. EMeana "ACE16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs", IEEE Transactions on Circuits and Systems I, Vol. 51, Issue 5, pp.851-863, 2004.
2. Z. Nagy, P. Szolgay, "Configurable multilayer CNN-UM emulator on FPGA", IEEE Trans. on Circuits and Systems I, vol 50, Issue 6, pp.774-778, 2003.
3. M. Perko, I. Fajfar, T. Tuma, J. Puhan, J, "Low-cost, high-performance CNN simulator implemented in FPGA", IEEE Int. Work. on Cellular Neural Networks and Their Applications, CNNA, pp. 277-282, 2000.
4. S. Malki, L. Spaanenburg, " CNN Image Processing on a Xilinx Virtex-II 6000, Proceedings ECCTD03 (Krakow) pp. 261-264, 2003.
5. J.J. Martínez, F.J. Garrigós , F.J. Toledo, J.M. Ferrández, "High Performance Implementation of an FPGA-Based Sequential DT-CNN", Int. Work. on the Interplay between Natural and Artificial Computation IWINAC, pp. 1-9, 2007.
6. D. Densmore, R. Passerone, "A Platform-Based Taxonomy for ESL Design", IEEE Design & Test of Computers, pp. 359-374, vol. 23-5. 2006. ISSN: 0740-7475.
7. DRC Computers, Web page,`http://www.drccomputer.com`, 2008.
8. Impulse Accelerated Technologies Inc., Web page, `http://www.impulsec.com`, 2003-2009.
9. L. O. Chua, L. Yang, "Cellular neural networks: theory", IEEE Trans. Circuits and Systems, CAS-35, 1988.
10. J.J. Martínez, F.J. Toledo, E. Fernndez, J.M. Ferrández "A retinomorphic architecture based on discrete-time cellular neural networks using reconfigurable computing", Neurocomputing, vol 71, Issue 4-6, pp. 766-775, 2008.
11. J.J. Martínez, F.J. Toledo, E. Fernndez, J.M. Ferrández "Study of the contrast processing in the early visual system using a neuromorphic retinal architecture", Neurocomputing, vol 72, Issue 4-6, pp. 928-935, 2009.
12. SRC Computers, LLC., Web page, `http://www.srccomp.com/`,2009.
13. Silicon Graphics, Inc., Web page, `http://www.sgi.com`,2009.
14. Cray Inc., Web page, `http://www.cray.com/`,2009.
15. Xilinx Inc, "Virtex-4 User Guide", data sheet(ug070), `www.xilinx.com`, 2004.
16. T. V. Court, and M. C. Herbordt, "Families of FPGA-Based Accelerators for Approximate String Matching", ACM Microprocessors & Microsystems, vol 31, Issue 2, pp. 135-145, 2007.
17. V. Kindratenko, and D. Pointer, "A case study in porting a production scientific supercomputing application to a reconfigurable computer", in Proc. IEEE Symposium on Field-Programmable Custom Computing Machines - FCCM'06, pp. 13-22, 2006.
18. E. El-Araby, T. El-Ghazawi, J. Le Moigne, and K. Gaj, "Wavelet Spectral Dimension Reduction of Hyperspectral Imagery on a Reconfigurable Computer", IEEE Int. Conference On Field-Programmable Technology (FPT 2004), Brisbane, Australia, December, 2004.
19. A. Michalski, K. Gaj, T. El-Ghazawi, "An Implementation Comparison of an IDEA Encryption Cryptosystem on Two General-Purpose Reconfigurable Computers", Proc. FPL 2003, pp. 204-219, 2003.
20. O. O. Storaasli, "Scientific Applications on a NASA Reconfigurable Hypercomputer", 5th MAPLD Int. Conference, Washington, DC, USA, September, 2002.
21. GCC, the GNU Compiler Collection, Web page, `http://gcc.gnu.org/`, 2009.
22. FastCam, Web page, `http://www.iac.es/proyecto/fastcam/`, 2009.