CGRA HARP: Virtualization of a Reconfigurable Architecture on the Intel HARP Platform

Lucas Bragança da Silva, Fredy Alves and José A. Nacif
Department of Informatics
Federal University of Viçosa
Campus Florestal
Florestal-MG 35690-000, Brazil
Email:{lucas.braganca, fredy.alves, jnacif}@ufv.br

Fernando Passe, Vanessa C. R. Vasconcelos and Ricardo Ferreira
Department of Informatics
Federal University of Viçosa
Campus Viçosa
Viçosa-MG 36570-900, Brazil
Email:{ricardo, fernando.passe, vanessa.vasconcelos}@ufv.br

***Abstract*—Researches in heterogeneous computing have shown that the usage of more than one computational node in the same system increases the performance and decreases the energy consumption. There are some types of architectures that have many advantages, both in performance increasing and in energy efficiency, such as architectures that possess a FPGA as an accelerating unit. An example of this type of architecture is the HARP, recently launched by Intel. Currently, in order to use the HARP's FPGA, the developer must implement an accelerating functional unit (AFU) and perform the synthesis of this unity in the FPGA, but this synthesis may demand a considerable time, making the architecure unfeasible for a real time system that requires the FPGA reconfiguration. Thus, in this work it is presented an abstract layer for the HARP's FPGA, which allows the FPGA reconfiguration with no need to perform a new synthesis, making this architecture feasible for a real time system.**

I. Indroduction

Nowadays, with the rise in the quantity of produced data, comes the problem of processing these information in the least time possible [1]. Researches in technologies capable of increasing the processing power of modern computers are turning to the usage of parallel processing.

The great challenge for computer scientists is to develop architectures that allow the processing of parallel data in an efficient way. An example of technology that explores this kind of processing are the GPUs (Graphics Processing Units), which are capable of performinf operations over a large quantity of data [2]. When regarding problems that revolve around parallel processing at a data level, GPUs are a great alternative. These kind of problems can be modelled and processed directly at the GPU, thus optimizing the processing time. There are other classes of problems which can be solved by using parallelism, but not in a data level. When this is the case, an interesting strategy is the usage of reconfigurable hardware as FPGAs (Field Programmable Gate Arrays). The FPGAs can be reconfigured in order to behave as any kind of digital circuit. A new technology presented by Intel is called HARP (Heterogeneous Architecture Research Platform) and has the goal to ease the researches on heterogeneous computing, by providing an architecture with an API (Application Programming Interface) for its usage. This architecture has a central processing unit (CPU) and a FPGA in the same system, which makes it possible to program new hardwares in the FPGA with the goal of aiding the data processing, thus obtaining a reconfigurable hardware acceleration.

In order to obtain a proper operation of the system, an application must be divided in two parts: one that will be programmed to the CPU by using the Intel API in C++, a high level object oriented language, and another that should be programmed for the FPGA in a hardware description language such as Verilog. This configuration shows to be functional when a solution is implemented in order to use both architectures in an efficient way, being that the C++ code is compiled for the CPU and the other part of the system is implemented in Verilog and synthesized in the FPGA. However, the process of modeling and programming the system for the FPGA is difficult and unintuitive for non experienced programmers. Moreover, the process of synthesising and configuring the FPGA may take a considerable time, depending on the application size, what makes it unfeasable to change the context for real time systems. One way to avoid this problem is the development of an abstraction layer. By using this layer, it will be possible reconfigure the FPGA from the HARP platform in execution time. Additionally, it will provide a way of programming the FPGA without the requisite to use a hardware description language, making it easier and more intuitive, and discarding the need of synthesising it again.

Thus, this work has the objective to implement an abstraction layer for the HARP platform, in order to provide a simple and efficient programming interface while keeping all the processing power of this architecture, without the need for a programmer who is familiarized on hardware description languages and allowing system reconfigurations in real time, which are not possible in the current context.

This paper is divided as following: the types of heterogeneous computing and the description of the problems faced when using HARP were approached in the introduction; in the second section a general vision in heterogeneous computing is presented, with emphasis on its two main types and a description of the Intel platform; in the third section there is a discussion on dataflow graphs and how those can be used for

obtaining parallelism in coarse-grained reconfigurable architectures (CGRA); in the fourth section there is a description of the CGRA implemented to be virtualized in the HARP's FPGA; in the fifth section the results and future works are discussed; lastly, the sixth section presents this work's final considerations.

II. Heterogeneous Computing

Heterogeneous computing can be defined as any system that uses different types of architectures in a single computational node. The current main used architectures in heterogeneous computing are the systems with a central processing unit (CPU) together with a graphics processing unit (GPU) or a more common reprogrammable architecture such as FPGA. In this section we present the two types of computational nodes, showing an architecture developed by Intel as an example.

A. CPU and GPU Architecture

This is the most frequently used architecture, due to most of the personal computers and workstations containing a GPU, whereas GPUs are accelerator devices that act together with the CPU and whose only function is graphics processing. However, in 2001 the GPUs started to use 32 bits floating point values when dealing with pixels, what made it possible for programmers to perform computation on graphics cards, thus obtaining a better performance in their applications, as in [3].

In 2007, Nvidia launched the first graphics card as an accelerator, boosting the execution of applications of deep learning, analysis and engineering [2]. The computing that gets accelerated by graphics cards performs the most intense computations of a program in the thousands of present cores in a GPU. Thus, a part of the code is executed by the CPU, and another part is executed by the GPU, causing the execution time for these applications to get shorter.

Figure 1 shows a system with CPU and GPU, exemplifying how GPUs accelerate workloads. Yet, parallelizable chunks of a code represent solely 5% of the whole application code, which still leaves most of the work for the CPU, [2]. This is due to the way the GPU executes the instructions in each of its cores. Because of its usage of an architecture model where the same instruction is executed by all the cores concurrently (SIMD), it loses performance to algorithms that cannot be implemented with this kind of parallelism.

B. CPU and FPGA Architecture

This architecture for heterogeneous computing is more recent. In this one, a system is composed by a CPU and a FPGA, connected via high speed bus. Although FPGAs have been invented a long time ago, with the new technological advancements in the microelectronics field, these devices have been gaining noticeability recently. An interesting characteristic of a FPGA is its reconfigurable capacity, making it possible to implement various types of circuits [4]. Other important properties from the FPGAs are their energy efficiency and the excelent cost/performance relation in comparison with a regular hardware, they can also offer high density of parallel computing, both in spatial terms as well as in temporal parallelism terms acquired by means of pipeline [5].

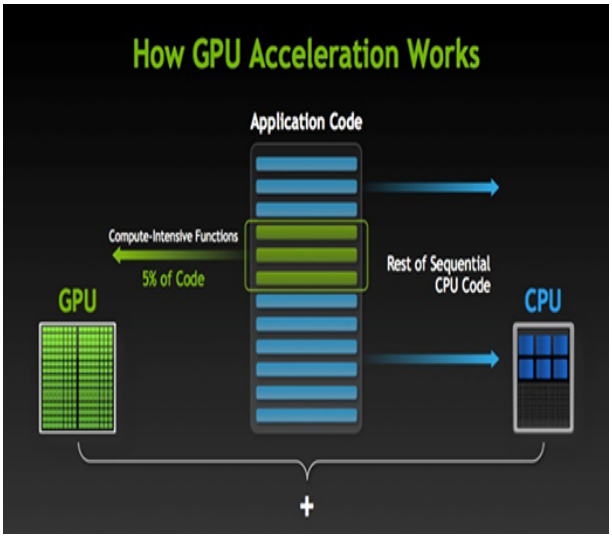

Fig. 1. System with CPU and GPU (source: [2])

There are various CPU/FPGA heterogeneous platforms such as: Alpha data [6], which can be easily adapted in any work station by using the PCI slot; Microsoft Catapult [7] and Intel HARP platform [8], which will be discussed in this paper.

C. HARP: Heterogeneous Architecture Research Platform

The purchase of Altera by Intel shows that the researches in reconfigurable architectures are promising [9]. The recently launched HARP platform is the result of the combination from two of the most important technology companies from the current scenario.

HARP is a server for datacenters that possess a CPU Intel Xeon E5-2680v2@2.80GHz and a FPGA from Altera Stratix V@200MHz in its system, they are connected via a QPI bus of 8 GT/s, developed by Intel itself. In order to optimize the data transfer speed between the CPU and the FPGA, Intel has used two sockets in HARP's motherboard, being that one is used by the processor Xeon and the other is used by the Stratix V. The communication between these two entities happens via a 64kb cache, which produces a better data transfer performance [10]. Figure 2 shows the HARP architecture and the QPI communication bus.

Currently, the development process for a HARP platform is slow and difficult. The developer must map its solution to the FPGA by developing an acceleration functional unit (AFU) and suiting this functional unit to a communication structure that is already implemented by Intel. A host application must also be implemented by using the HARP's API, which is written in the C++ programming language, this application is responsible for allocating and using the AFU. Figure 3 illustrates the execution flow from an application in the HARP Intel platform.

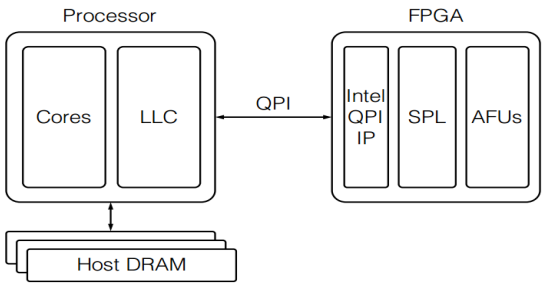

Fig. 2. HARP Intel Communication Scheme (Source: [10])

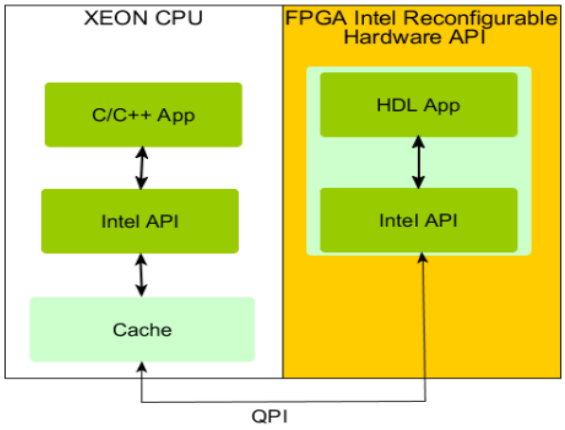

Fig. 3. HARP development flow (Source: [11])

This process may demand much of the project's time, since hardware programming is not a trivial task. Another important observation regarding this development environment and the platform's usage is that once the AFU is implemented it must be compiled and synthethized in the HARP's FPGA, this process may take time depending on the complexity of the solution. Once the accelerator (AFU) is synthethized in the FPGA, it can be used in computations in which it was designed to perform. However, if it is necessary to perform a new type of computing, a new AFU will have to be implemented and synthethized, thus causing the context change to be unfeasible, which makes the system limited on its usage.

A feasible way to avoid the HARP's FPGA reconfiguration problem is to implement an abstraction layer for the FPGA, where once this layer is implemented and synthethized, the developer should worry solely on modeling the solution for this new architecture, spending less time on development and allowing configuration change without the need to execute a new synthesis, thus being able to perform reconfigurations in execution time. In this work the abstraction implemented in HARP's FPGA is a coarse-grained reconfigurable architecture (CGRA), in other words, it allows reconfiguration to a word level.

III. CGRA

CGRA is a coarsed-grained reconfigurable architecture that possesses logical components, which are connected by an interconnection network and may be configured [4].

CGRA was designed with the goal to reduce the configuration time and complexity when compared to a FPGA. In order to reach this goal, the logical and arithmetic operations of a CGRA are executed by processing units named UPs, which perform more complex operations than the logical and arithmetic units of an FPGA. This is due to the FPGAs operations being performed in a bit level while a CGRA performs operations in a word level [4].

The basic composition of a CGRA is given by the following elements:

- Processing Unit (UP): Element that performs the logical and arithmetic operations;
- Interconnection Network: Unit responsible for connecting the UPs outputs to each of the UPs own inputs;
- Configuration Memory: A memory that works as a program memory. In this element the operations which the UPs should perform and the configurations of the CGRA interconnection network are stored.

In a CGRA, the processing units (UPs) can be classified in two ways: Heterogeneous or homogeneous. The heterogeneous UPs can perform different operations between them, in other words, each UP has a different instruction set; on the other hand, the homogeneous UPs perform the same set of instructions. The operations that the UPs are capable of performing may be logical, arithmetic or input and output data. Another important characteristic in a CGRA is the type of interconnection network it adopts. There are two main types of networks in CGRAs, the multistage network and the crossbar network. The multistage networks are smaller in terms of logical elements in comparison to a crossbar network, but its mapping is more complex and sometimes it may not be possible. As of the crossbar networks allow any mapping from the inputs to any outputs, what facilitates the mapping process, yet they consume a high number of logical elements.

Thus, a basic CGRA allows a form of data computing similar to a dataflow machine. Dataflow machines are a type of programmable computer in which the hardware is optimized for fine-grained parallel computing, where fine-grained means the processes are executed in parallel and have the same size as a conventional machine instruction code. This type of architecture allows the execution of dataflow graphs. Due to its properties, this machine was widely studied by Von Neumann in his researches on neural networks, for it is possible to model a neural network by using dataflow graphs [12].

IV. Dataflow graphs

Dataflow graphs (GFD) are structures that allow the modeling of any conventional algorithm in the form of a graph. In this structure the nodes of the graph are the operations that are going to be executed and the edges are the data. A GFD may be used to map an algorithm in CGRA, if we take the

nodes of the graph as an UP and its edges as the input data and their outputs. In this way, we can obtain what is named as a temporal parallelism, in which each level of the graph can be seen as a pipeline stage. Figure 4 shows an example of a GFD for the second degree equation $ax^2 + bx + c$, which may be mapped for a CGRA. In this example, we have a latency in the calculation of the 3 input equation, but after the third input, for each new input data, the result of a new equation is calculated.

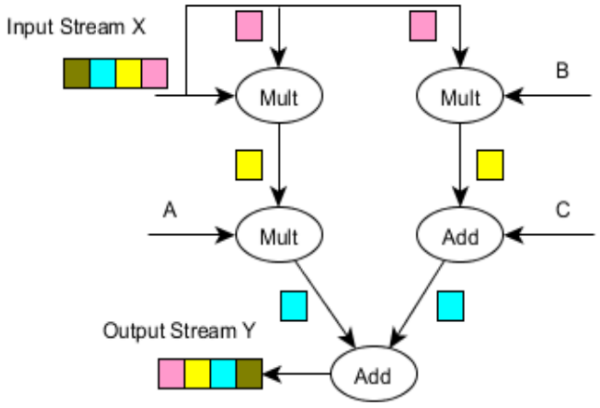

Fig. 4. GFD for the equation $ax^2 + bx + c$ (Source: [11])

A. CGRA virtualized in HARP

The virtualization of a CGRA in the HARP's FPGA allows the configuration of the system without the need of performing a new synthesis, thus the developers may map algorithms in the form of a GFD and execute them in HARP in a fast and simple way, taking advantage of the natural parallelism of a GFD. In this work a 16 bits word CGRA and 8 UPs connected by a interconnection network of the crossbar type was implemented. Besides its basic elements, this CGRA has an array of constants, which allows their uses as inputs for the UPs, easing the mapping of mathematic equations such as the example of Figure 4.

Figura 5 illustrates the CGRA datapath implemented for the HARP's FPGA.

This CGRA is classified as heterogeneous solely because of two different UPs performing the data loading and writing, respectively, while the rest of the UPs can perform basic arithmetic and logical operations. Each UP possess 3 configuration bits and 1 enabling bit, also, only the UPs for data loading and writing have 2 more bits, what makes a total of 34 bits for all UPs. The crossbar network from the UPs is 8x16, in other words, it is capable of connecting the eight UPs outputs to any of the sixteen UPs inputs, spending a total of 48 configuration bits. The crossbar network from the constants is also 8x16, what makes it possible for mapping any constant to any of the UPs inputs. Lastly, 16 bits are used for multiplexer control, making a configuration word with a total of 146 bits.

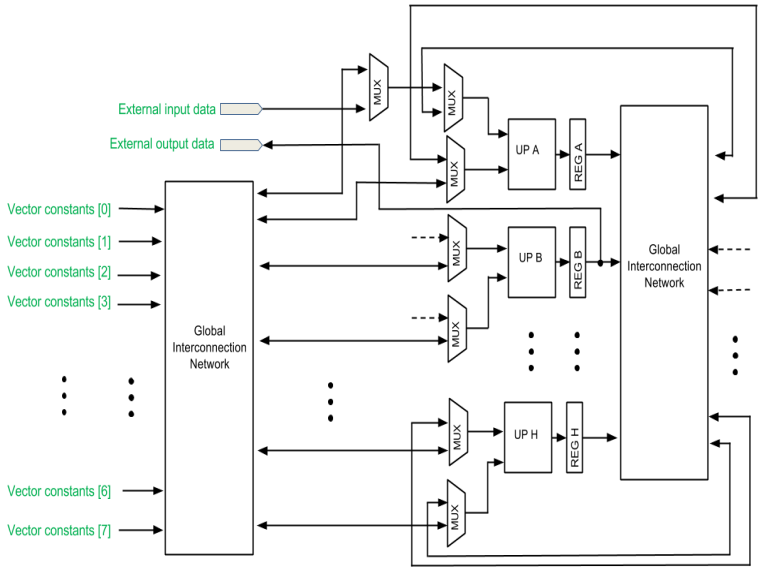

Fig. 5. CGRA datapath (Source: The author himself)

V. Results and Future Work

As a result, it is obtained an architecture that makes it possible to execute GFD for mathematic calculations in the HARP platform, without the need of performing a new synthesis, being this the main focus of this work. Thus, there is no need for results regarding the execution time.

Besides that, this architecture is capable of performing calculations of various types of GFDs in a fast and simple way, without the need for the developer to directly program for the HARP's FPGA.

In future works, we plan to expand the architecture making it faster and more efficient. For that, we would need to: rise the number of UPs in order to obtain better performance through parallelism; make a homogeneous CGRA, which would allow any of the UPs to perform all the operations, including data input and output; and rise the number of operations from the UPs, in order to make floating point operations possible.

VI. Conclusion

This work presented the HARP platform and the main problem found by its developers.

The reconfiguration time of reprogrammable devices such as the FPGA is an obstacle that needs alternatives in the current context. The quantity of generated data tends to rise even more in the upcoming years, and extracting information from these data with conventional architectures might not be possible. Due to that, researches of new architectures are of utmost importance in the current scenario. CGRA HARP constitutes an architecture that solves the configuration time for the FPGA, increasing the efficiency of applications that require to change its context quickly.

References

[1] IBM. (2013) Curso big data analytics. [Online]. Available: https://www.ibm.com/developerworks/community/blogs/bigdata/entry/curso_-big_data_analytics_na_fgv_management_rio_de_janeiro

[2] NVidia. (2016) Computação acelerada. [Online]. Available: http://www.nvidia.com.br/object/what-is-gpu-computing-br.html
[3] A. Moravánszky, "Dense Matrix Algebra on the GPU." [Online]. Available: http://www.shaderx2.com/shaderx.PDF
[4] W. D. M. M. FILHO, "Virtualização e execução de algoritmos em fpga: Um algoritmo de modulo scheduling para arranjos reconfiguravéis de grão grosso," Master's thesis, Universidade Federal de Viçosa, Viçosa-MG 36570-900, Brasil, 2014.
[5] R. Ferreira and J. A. M. Nacif, "Computação heterogênea com gpus e fpgas," *Livro dos Minicursos do WSCAD 2016*.
[6] A. DATA. (2016) Alpha data datasheet:adm-pcie-7v3. [Online]. Available: http://www.alpha-data.com/pdfs/adm-pcie-7v3.pdf
[7] A. C. Andrew Putnam, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA)*. IEEE Press, June 2014, pp. 13–24. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-reconfigurable-fabric-for-accelerating-large-scale-datacenter-services/
[8] N. Carter. (2016) Intel-altera heterogeneous architecture research platform (harp) program. [Online]. Available: https://www.sigarch.org/2015/01/17/call-for-proposals-intel-altera-heterogeneous-architecture-research-platform-program/
[9] ComputerWorld. (2015) Intel anuncia a conclusão da compra da altera por us$ 16,7 bilhões. [Online]. Available: http://computerworld.com.br/intel-anuncia-conclusao-da-compra-da-altera-por-us-167-bilhoes
[10] Y.-k. Choi, J. Cong, Z. Fang, Y. Hao, G. Reinman, and P. Wei, "A quantitative analysis on microarchitectures of modern cpu-fpga platforms," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 109.
[11] R. F. Fernando Passe, Vanessa C. R. Vasconcelos, L. B. Silva, and J. A. Miranda, "Virtual reconfigurable functional units on shared-memory processor-fpga systems," 2016.
[12] A. H. Veen, "Dataflow machine architecture," *ACM Comput. Surv.*, vol. 18, no. 4, pp. 365–396, Dec. 1986. [Online]. Available: http://doi.acm.org/10.1145/27633.28055