# Invited Paper: Acceleration of Computationally-Intensive Kernels in the Reconfigurable Era

Kyprianos Papadimitriou[1,2], Charalampos Vatsolakis[2], and Dionisios Pnevmatikatos[1,2]

[1]Foundation for Research and Technology - Hellas
[2]Technical University of Crete

***Abstract*—One of the major topics that attracts constantly the interest of research community is the acceleration of computationally-intensive applications. Towards this direction, different technologies are competing each other and are all characterized by a common tendency; they evolve continuously towards improving their products so as to serve better their customers and attract new ones. Each company stresses the stronger advantages of its technology to convince people from academia and industry, and either individuals to use it. However, none of the existing technologies monopolizes all domains yet. This is because each one has its own benefits and drawbacks and cannot serve well all application domains. Another reason is that different people prefer to acquire skills in a specific technology and avoid switching between different technologies, even though this could benefit the application under development. Hence, from the one side a technology itself imposes performance margins in terms of factors such as speed, power, energy, and cost, while on the other side stands the willingness of the user to adopt a technology. This adoption can be affected heavily by the flexibility and the "friendliness" (i.e. easy-to-use) of the technology.**

**Present paper discusses problems in using reconfigurable technology and suggests some research directions. We delve into the details of two systems equipped with specific reconfigurable platforms and explore the capabilities offered to the user for accessing this technology. We point out some problems in adopting reconfigurable technology and we study some performance characteristics. Aim of the paper is to signify issues that could gain the attention of the research community, and trigger the exploration of solutions that will foster the adoption of reconfigurable technology by a wider range of people.**

## I. Introduction

Several technologies exist which are continuously evolving in order to adjust to the user and application needs. Some of the prevalent ones are traditional microprocessors, multi-core and many-core systems, General Purpose GPU, DSP, ASIC and reconfigurable computing. Although each of these technologies dominates certain area of the market, there are cases in which all or some of them are competing each other in serving effectively a domain, especially when this domain is promising and profitable. Moreover, technological advancements tend to be application-driven. Hence, once a new application domain appears, or when the requirements of a domain are changing, companies would consider altering their products, either their hardware platforms or software tools.

The parameters affecting the decision of selecting a technology platform vary and are case dependent, and might not be driven by the application. For example, some scientists are interested mainly in execution speed regardless of factors such as the effort and cost to develop a system, or the energy consumption. On the opposite, there are scientists concerned mainly about the energy consumption and cost of the system and they do not place emphasis on execution speed. Looking from a different aspect, a major issue is the flexibility and friendliness of a technology that allows for shortening the development time by even compromising performance. From this perspective, traditional programming in software and technologies such as microprocessors and DSP tend to be the dominant technological solutions even in case the target application is characterized by parallelism and pipelining. In order for the reconfigurable technology to compete with the other technologies, the former should be easily accessible by the user. This is of great importance, especially when it comes to serve people that are unaware of the details of a new technology but they consider trying it.

We envision systems in which the user will be able to access the hardware without controlling explicitly the FPGA, such as knowing when and how to establish communication between the host and the FPGA, when and what kind of data to transmit, and when to perform reconfiguration. This can result to a system in which the access to reconfigurable hardware is transparent to the user; at the same time the performance should not be compromised for transparency. Certainly, such a system is justified for applications characterized by heavy parallelism and pipelining. This concept allows to formulate systems combining software with reconfigurable hardware, which will be self-reconfigured to adjust automatically to the user and application needs. The user will not be bothered with tasks such as manual injection of configuration data through the execution of reconfiguration commands, or, data transferring to the FPGA; these functionalities will not be exposed to the user and they will be undertaken by the host software.

The paper is structured as follows. Initially, we identify the problem given the current state of the reconfigurable technology. Next, we discuss our experiences with two FPGA-based platforms plugged on a PCI by focusing on the user's convenience. Finally we conclude the paper.

## II. Problem Identification

Depending on the technology, different features can be stressed by a company to make it more attractable to the potential customers. Execution speed, customization level, cost,

power and energy consumption are some of them. Another factor is the easiness in adopting and using a technology. All these factors can be considered as metrics constituting the performance criterion. From the user perspective, the decision of selecting a technology can differ even amongst two individuals that target the same application. This is because the selection criterion with the highest priority can be different for every individual. For example, the decision can rely mainly on the comfort of the individual and less on the application demands and the advantages of a technology. The reason someone chooses a technology can be irrational and contradictory, e.g. someone who will develop an application but does not intend to put effort in learning a technology even though it would provide remarkable results in the specific application domain; however, this scenario is very realistic.

Many users and developers prefer completing the development cycle without experiencing the hassle of optimizing a system, or passing through the learning curve for acquiring new skills. At the same time they prefer keeping a system at a reasonable level of performance, and not putting effort in order to elevate performance at the highest achievable level. The concept of integrating reconfigurable technology for accelerating kernels in certain cases seems appealing; this applies perfectly in the generic domain of desktop computing. In order for this to happen, two main requirements should be satisfied; first, reconfigurable technology should be accessed effectively by the software host, and second, the access to the reconfigurable hardware should remain transparent to the user.

Different people differ in the way they develop the same application due to the different habits and style in programming. There are experienced programmers skilled in a hardware programming language (HDL) but the vast amount of people prefers software-like programming. For example, there are people programming in Fortran that do not want to switch to a different language as they will experience a learning curve. Also, there are people preferring to program in C, or in a simpler language like Matlab, or even users wanting to use scripting languages in order to get the results of their computation. The level into which each people of the aforesaid categories would be bothered and change its programming habits in order to accelerate a kernel is a personal matter, and does not fall within the scope of the present work. Furthermore, there are simple users that need to execute heavy tasks such as real-time video processing, 3D rendering, or large FIR filters, not willing to program in any language.

To overcome the above issues reconfigurable hardware should be accessed in a transparent manner by the host. Efficient IP cores residing in an repository can be available for configuring the FPGA when needed. The set of IP cores that will reside in a desktop computer depend on the user needs. For example, if the user works extensively on image processing then the repository should house relevant IP cores. Once a heavy task is called the corresponding IP core will be loaded, e.g. the user clicks an option on the toolbar of an image processing program. This way the FPGA acts as a coprocessor, while the system adjusts itself to the changing performance demands. At the same time, the user is not bothered controlling the FPGA or being aware that processing has been transferred to the FPGA.

In order to build such systems, device drivers and APIs are needed, which allow the host computer to interface with the FPGA platform. High-speed interfaces should connect the host with the FPGA platforms, such as PCI, Ethernet or USB, for serving data transfer and reconfiguration. The connection setup needs to be efficient enough so as to avoid hampering system performance due to high communication delays.

To enable such functionality runtime system support is needed, which is a software component for supporting execution of application workloads in OS based systems. It will undertake low-level operations so as to offload the programmer from manually configuring the FPGA and controlling the data transfers. The runtime system should reside always in memory and can be implemented as a standard library. It contains subroutines that realize functions by accessing the OS with system calls.

## III. The FPGA-based Systems

We are studying the capabilities offered by reconfigurable technology through experimentation with two FPGA-based platforms equipped with a Virtex-II and a Virtex-5 respectively. Both platforms were plugged on the PCI slot of a PC running a linux operating system. In both systems we executed the same software application. This application awaits from the user to make a selection through a command line interface. Three options are available: i) addition, ii) subtraction and iii) LED toggling. These kernels have been designed as IP cores for execution in the FPGA; the bitstreams are stored in the HDD of the host PC. Once the user enters a choice, a user level program triggers reconfiguration of the FPGA by loading the corresponding bitstream; if the choice matches the kernel already placed in the FPGA, reconfiguration is not triggered. The reconfiguration process remains completely transparent to the user entering the kernel choice. The user receives only the result of kernel execution.

Both FPGA platforms are used widely by the research community. The kernels we developed are simple but serve well our scope of demonstrating the concept. Below we discuss each setup separately, we include measurements of the throughput and the latencies of certain operations, and then we briefly report the differences of the two setups. For each setup, we describe the system architecture in terms of hardware and software and we provide a description of the path an I/O request has to follow from the user application to the FPGA.

### A. NetFPGA platform on PCI

The NetFPGA houses a Spartan-2 which is responsible for PCI transactions and a Virtex-II which is available to be programmed with a user design. It targets mainly the network domain, but we experimented with a non-network related application. The system configures the Virtex-II FPGA according to the user choice using the default NetFPGA driver

and the default PCI design interface loaded into the Spartan-2 FPGA. To reprogram the Virtex-II FPGA and transfer the data to it from the host we used the NetFPGA API. Two are the main modules that configure the FPGAs; the Configurable PCI module (CPCI) loaded into the Spartan-2 device, and the Configurable Network module (CNET) loaded into the Virtex-II device.

The communication between the Spartan-2 and the Virtex-II concerns the latter's reconfiguration and the data I/O transfers. The CPCI module drives the reprogramming signals that are attached to the 8-bit SelectMAP configuration port of the Virtex-II in order to control the reconfiguration process. In particular, reconfiguration is triggered by a software program in the host that opens a bitstream and sends quantities of 32-bit words to a register in the CPCI module. The latter inserts the data to a FIFO in order to address synchronization issues. The configuration data are then read from the FIFO and sent to the SelectMAP configuration port of the Virtex-II FPGA.

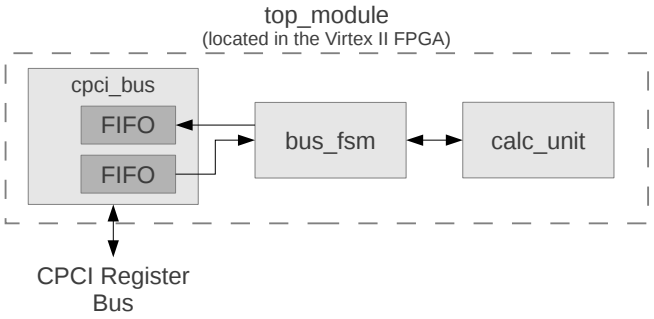


Figure 1. CNET module placed in the Virtex-II FPGA

Data transactions between the CPCI and the CNET are carried out through a register I/O interface. We used the standard cpci_bus module of the CNET design for the communication between the two FPGAs, which is shown in Figure 1. This module contains two FIFOs and an FSM for control purposes. The two FIFOs are used for storing temporarily the requests and the responses, which pass through a single bidirectional port connecting the two FPGAs. The bus_fsm module is an FSM responsible for dispatching the incoming requests and passing the appropriate arguments to the calculation unit. The calc_unit is the only module that differs amongst the three designs, while the interface with the bus_fsm is the same for all designs.

The interface of the software with the hardware is achieved by a user level API combined with a kernel level driver provided by the NetFPGA. This provides access to a virtual register file in the CNET module. Figure 2 shows the stages for each request starting from the user level application until it reaches the hardware. The user application sends/receives data to/from the device by calling the proper writeReg/readReg functions. These functions are implemented in the user level API and depending on the driver they may either perform a transaction through a register operation or through a network socket; we used the register operation. In both cases, there

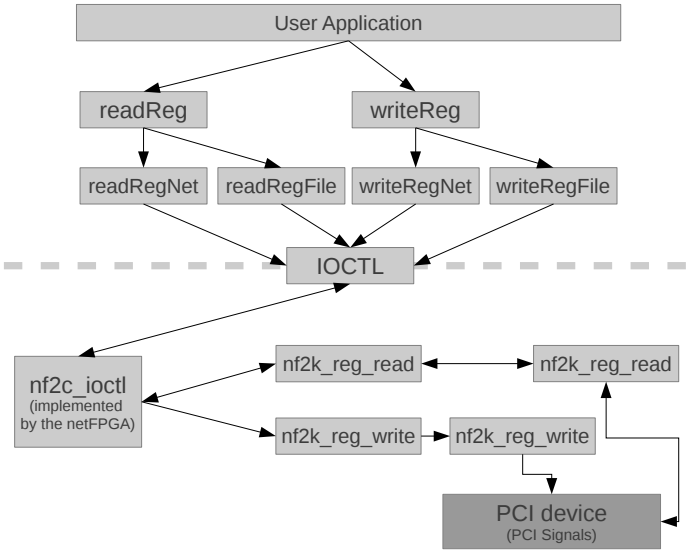


Figure 2. Software components needed for the communication between software and hardware


Figure 3. Overall system connection

is an IOCTL system call which is invoked in order to pass the appropriate parameters to the kernel level. The IOCTL system call passes the control to the device driver where the device specific IOCTL function is implemented. Then the driver performs the appropriate copies of the parameters from user memory space to kernel memory space. The last step is to perform the appropriate copy to/from the device memory space. Figure 3 illustrates the connection between the software and the hardware blocks of the system.

Table I has the time needed for each distinct operation. Reconfiguration time includes the time for opening/closing the device. A noticeable fact is that in case two sequential writes are executed, the second write takes less time than the first one.

### *B. XUPV5 platform on PCI*

The same system was implemented using the XUPV5 platform which was plugged on the PCI express interface. Transactions were carried out through register I/O. The software components of this setup include the necessary driver and a simple user level program. The latter sends/receives the appropriate arguments to/from the kernel driver by issuing the IOCTL system call. The driver is then responsible for transmitting the data and receiving the results.

This setup does not support reconfiguration of the FPGA through the PCI interface. Instead, reconfiguration is performed through the JTAG interface using the vendor's USB programmer. When the system is in running state, there is a sequence of states that need to be performed in order to

Table I
OPERATIONS FOR ACCESSING THE FPGA IN THE NETFPGA

| Operation | Time |
|---|---|
| Reconfiguration | 3,806.0 ms |
| Open | 37.8 us |
| Read | 5.3 us |
| Write | 6.75 , 1.75 us |
| Close | 10.5 us |

Table II
ACTIONS NEEDED TO RECONFIGURE THE FPGA IN THE XUPV5

| Action | Time (ms) |
|---|---|
| Driver Removal, Device Disable | 27.1 |
| Reconfiguration | 18,513.5 |
| PCI Rescan, PCI Device Enable, Driver Insertion | 10.1 |

reconfigure the FPGA. First, we need to disable the PCI express device by removing the driver and disabling the PCI express port on which the FPGA platform is placed. At this point the FPGA can be reconfigured. Once it is completed, we need to rescan all the PCI express interfaces for new devices. Finally, we need to enable the device and insert the driver to the kernel. Table II shows the distinct actions that should take place along with the time needed per action. These actions should occur in the given sequence every time the FPGA is reconfigured.

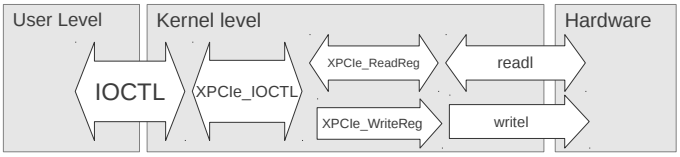


Figure 4. Route for I/O request from software to hardware

Once the device is configured, our user level application can communicate with it. The route of a request from the user application to the hardware and vice versa is shown in Figure 4. A user application that wishes to send data to the FPGA opens the device and issues an IOCTL request. The IOCTL request requires a number of parameters, such as the address of the register the user wishes to access and the data the user wishes to write. The data are then passed to the XPCIe_IOCTL which is a function implemented by the driver provided by the vendor. This function calls the XPCIe_ReadReg in case of a register read or the XPCIe_WriteReg in case of a register write. These functions compute the real hardware address of the memory mapped register and then they issue a readl or a writel request accordingly. Finally, in case the request is a read, the result is led back to the user application as a return value. Table III has the time required for each operation.

In this setup we measured the throughput for transactions through the register I/O and also for DMA. A reference application provided by the vendor "as it is" was used to collect throughput measurements for both device access modes

Table III
OPERATIONS FOR ACCESSING THE FPGA IN THE XUPV5

| Operation | Time (us) |
|---|---|
| Open | 13.0 |
| Read | 15.8 |
| Write | 3.6 , 2.9 |
| Close | 3.8 |

Table IV
THROUGHPUT FOR THE XUPV5 FOR REGISTER I/O AND DMA ACCESSES

| Operation | Throughput (Mbps) |
|---|---|
| Register Read | 8.8 |
| Register Write | 5.6 |
| DMA Write | 784.0 |
| DMA Read | 496.0 |

shown in Table IV. It should be noted that we didn't use the DMA access mode for our software application.

### C. Analysis and Evaluation

Two functional systems have been demonstrated with an FPGA acting as coprocessor to the host CPU. The FPGA platforms are inexpensive and can be used in desktop computing by plugging them into the PCI slot. Two basic operations should be supported; FPGA reconfiguration and data transfer between the host and the FPGA. The whole process is transparent to the user as the latter is not aware when the host software initiates the injection of a bitstream to the FPGA.

From the performance aspect the two systems do not perform well both in terms of reconfiguration time and data transfer time. The performance barriers can be due to improper choices by the designer or due to the available solutions provided by the vendor. Table V consolidates some characteristics of the two systems. The rates for reconfiguration and data transfer are not even close to the theoretical maximum. Also, in both systems we have not used the more efficient DMA access mode, which can offer remarkable performance as shown in Table IV. Still, even though we used the reference application provided by the vendor, the theoretical throughput of PCIe with single lane width (1x), i.e. 2 Gbps, was not achieved. The same result was obtained in [1]. Moreover, configuration either through the SelectMAP or JTAG is not close to the theoretical maximum. Configuration through JTAG is inherently slow [2], and alternative solutions would offer faster reconfiguration.

The main difference between the two setups is that in the NetFPGA the PCI interface is implemented on a different FPGA than the one configured with the user design, while in the XUPV5 the same FPGA holds the PCI and the user design. Thus in the former case the Virtex-II is configured by a dedicated FPGA controlling the signals of the SelectMAP port through the PCI, but in the latter case Virtex-5 cannot be programmed directly through the PCI.

In the same way the FPGAs are fully reconfigured they can also be reconfigured in part. Each of the three simple cores occupies less than 1% of the FPGA resources, either

Table V
CHARACTERISTICS OF NETFPGA- AND XUPV5-BASED SETUPS

| Platform Characteristics | NetFPGA | XUPV5 |
|---|---|---|
| Device | XC2VP50 | XC5VLX110T |
| Configuration bits | 19,021,344 | 31,118,848 |
| PCI characteristics | PCI 32-bit@33MHz | PCIe 1x@62.5MHz |
| Programming interface | PCI Spartan-2 | Platform cable USB |
| Reconfiguration port | SMAP 8-bit@33MHz | JTAG@6MHz |
| Reconfiguration time | 3.80 sec | 18.51 sec |
| Reconfiguration rate | 4.98 Mbps | 1.68 Mbps |
| Host/FPGA comm. | Register I/O | Register I/O |

in the Virtex-II and the Virtex-5 device, thus it would be of great benefit to isolate and configure only the respective core. Also, in order to reduce the reconfiguration time, the partial bitstreams could be cached in a memory located on the FPGA board.

## IV. DISCUSSION SUMMARY

This paper discusses the feasibility for keeping transparent the acceleration of certain kernels to the user by injecting automatically configuration bitstreams into the FPGA coprocessor. To overcome the problem of efficiency in the generic desktop computing, different issues should be addressed that relate with the development of efficient device drivers and APIs. Also, the interfaces for reconfiguration and data transfer should be studied and selected wisely. Choosing a separate path for each of these operations, e.g. PCI for data transferring and USB for reconfiguration through JTAG, can be an option but the efficiency relies heavily on the device drivers and the controllers besides the interfaces themselves.

Reconfigurable computing should exhibit its more advanced features in order to attract people from different domains and with varying habits. Development of efficient IP cores that also should be ported to a wide range of FPGA platforms is needed. Availability of efficient components for systems such as the ones discussed above is pervasive in desktop computing. Also, it is essential to define clearly the interfaces and their control. Run-time system support is needed which can be developed as a standard library lying above the host operating system. Finally, partial reconfiguration can play an important role in kernel acceleration [3]. Towards this direction, the development of runtime system support for controlling the injection of partial bitstreams and their route to the configuration memory remains a challenging research area [4].

## V. ACKNOWLEDGEMENT

This work is supported by the research project "Facilitating Analysis and Synthesis Technologies for Effective Reconfiguration" (FASTER, #287804), financed by the European Commission in the context of Seventh Framework Programme. Also, it is partly funded by the "Increasing EU citizen security by utilising innovative intelligent signal processing systems for euro-coin validation and metal quality testing" research project (SAFEMETAL, #262558), implemented within the Seventh Framework Programme and financed by Community Funds.

## REFERENCES

[1] R. Bittner, "Bus Mastering PCI Express In An FPGA," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, February 2009, pp. 273–276.

[2] H. Tan, R. F. DeMara, A. J. Thakkar, A. Ejnioui, and J. D. Sattler, "Complexity and Performance Tradeoffs of Two Partial Reconfiguration Interfaces on FPGAs: a Case Study," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, 2006.

[3] D. Koch, C. Beckhoff, and J. Torresen, "How Partial Reconfiguration Can Help HPC," in *Proceedings of the Design, Automation and Test in Europe (DATE)*, March 2011.

[4] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Exploiting Partial Run-Time Reconfiguration for High-Performance Reconfigurable Computing," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 1, no. 4, January 2009.