"Mircea cel Batran" Naval Academy Scientific Bulletin, Volume XIX – 2016 – Issue 2
The journal is indexed in: PROQUEST / DOAJ / Crossref / EBSCOhost / INDEX COPERNICUS / DRJI / OAJI / JOURNAL INDEX / I2OR / SCIENCE LIBRARY INDEX / Google Scholar / Academic Keys/ ROAD Open Access / Academic Resources / Scientific Indexing Services / SCIPIO / JIFACTOR

AUTOMATED FPGA FIRMWARE MANAGEMENT IN HPC CLUSTERS

Laurentiu Alexandru DUMITRU[1]
Stefania Loredana NITA[2]

[1]Eng., Ph.D. (c), Military Technical Academy, 39-49 George Cosbuc Bvd., Bucharest, Romania
[2]Ph.D. (c), Integrated Systems Department, Institute for Computers, Fabrica de Glucoză 11A, Bucharest, Romania

***Abstract**: FPGA-based accelerators are increasingly deployed on cluster and grid systems due to their highly flexible architecture. Given the generic nature of a high performance computing system, the firmware and software running on the FPGAs changes dynamically, according to the specifications requested by the launched application. Along with performance monitoring, this reconfiguration process can be automated in order to decrease idle-times on computing nodes and to have a centralized view of the system. Such an architecture would be centered around a client-server model in which the computing nodes run the client component, along with the batch agent. The server component would be located anywhere in the cluster as long as it has the appropriate permission to interact with the batch server. The paper explores the possibility of integrating this reconfiguration model with an existing batch system, without major changes in the way users operate the cluster.*

***Keywords:** FPGA, cluster, batch system, firmware management*

Introduction

FPGA-based accelerators can be installed in parallel computing environments, like computing grids, in cloud environments and, in special cases, in dedicated workstations. Whatever the situation, monitoring the available resources, the performance and the need for firmware reloading must be automatically managed in order to reduce idle time per computing node. Reprogramming an accelerator involves, in most cases, even rebooting and, therefore, should be done as seldom as possible.

Since a massive cluster has many computing nodes that execute user jobs by various algorithms, it is clear that the cluster administrator cannot manually trigger a reconfiguration, since it cannot cope with a high rate of change caused by the need of a different firmware. The current paper proposes a client-daemon approach that manages the reconfiguration process across accelerators in the entire cluster. Each computing node has a client that connects to a daemon which has an overview of all the available resources and requests. This application also interacts with the local resource manager and the cluster scheduler. By automating this life cycle of a specific FPGA firmware instance

the normal user does not need to know about any specific configuration details and the cluster administrator does not need to manually intervene.

The proposed solution is based on a shared library (FPGA Shared Acceleration Platform – FSAP) that is used by applications which make use of the accelerator modules. In the case of standalone workstations where there is no centralized management, the library assures the reconfiguration process, as it will be further described. It is also the library's task to bridge between the user space application and the acceleration module. Such a data transfer, which can be based on mapped memory as in [1], is usually mediated by a kernel module. The library wraps certain system functions in order to obtain the desired functionality. By mediating key function calls, it can signal if the current firmware needs to be replaced or not.

Another important part of the architecture is the actual firmware that is loaded on the accelerators. Since the firmware must be available on all nodes, the software uses a well-defined configuration scheme and storage location. This facilitates the exchange of modules between different working groups. By synchronizing to a central repository different users, for example, can benefit from a code addition, a bug fix or a performance enhancement simply by updating to the latest version. The control system takes care of re-instantiating the new firmware as soon as the operation can take place. The update process can cover non-firmware related code such as soft core applications or specific initialization values for memory regions.

Firmware management

As noted before, the firmware along with optional components form the reconfiguration package. The firmware is the core part which defines the internal architecture of the FPGA. Any acceleration modules are inside it. If the accelerator has a soft processor, a specific code

DOI: 10.21279/1454-864X-16-I2-060

must be also loaded to be executed by that processor. This configuration package can be used by standalone systems, systems that participate in a cloud or systems that are integrated in a computing cluster. Traditionally, the firmware is stored on a non-volatile memory and it is loaded on power up. Such a case is often seen on devices that have a relatively fixed configuration and do rarely require a change in it.

Standalone workstations can have dedicated accelerators (in the form of GPUs [5] or FPGAs) for various tasks. Although operating such a workstation requires an advanced degree of technical knowledge, the FSAP library tries to simplify day to day operations. In this case, firmware reprogramming is done either automatically or by an operator's request. A configuration file that is used by the library sets the work mode and various other parameters used internally, such as available firmwares, available modules and paths:

```
management {
        firmware_dir=/opt/fsap/bitfiles
        module_dir=/opt/fsap/modules
        auto_program=1
        program_script=xc3sprog
        program_args=""
}
firmware fw_name_1 {
        id=0x100
        bitfile=0x100.bin
        allowed_modules = { "mod_matrix1","smm3" }
        small_modules=3
        medium_modules=3
        large_modules=1
        …
}
```

The firmware section identifies available firmwares with their inherent properties. When manually triggered, the accelerator is reprogrammed with the firmware specified by the operator. In automatic mode, the library checks if the requested acceleration module by the userspace application is present on the current firmware. If not the systems notifies the operator if a reboot is necessary.

Reprogramming an accelerator requires a valid reconfiguration channel such as an USB link with an internal programmer or a JTAG adapter. An alternative method would be to have the firmwares stored on a non-volatile memory on the accelerator. In this case there is no need for a programming link but, the main disadvantage is that uploading a new firmware can be a more complicated operation. Usually a combination between these two options can prove to be a good compromise. For example – two main firmwares reside on a flash memory and can be loaded as needed and, if the situation requires it, an external firmware could be uploaded and used in a particular situation – such as testing a branch in the main application code.

Workstation-based environments are ideal for the development and testing phases of accelerator modules. Several debug tools and more verbose logging channels are available in this setup and allow an engineer to easily troubleshoot a particular module.

Cloud structures expose two type of resources: application middle-wares and virtual private servers. Middle-wares can be viewed as shared code that can be licensed through Software as a Service model. A middle-ware is rigid as the user cannot implement new functions or modify existing behavior. Virtual private servers are, from a software point of view, the same as physical servers. From a hardware point of view, multiple virtual servers share physical resources. If the host system has a FPGA accelerator installed, it can be exposed (passed-through) to a specific virtual server. If this is the case, the virtual server behaves just like a dedicated workstation with an FPGA installed. In particular cases, the accelerator can expose virtual functions that can be shared among multiple virtual servers. This approach is more technically complex and requires specific hardware that has support for technologies such as Single Root I/O Virtualization [9]. With regards to the most widely deployed scenarios, dedicated FPGAs are mostly used in academic and high performance computing clusters. Usually these systems do not use cloud architectures but cluster-like batch systems. If, however, a cloud must benefit from such acceleration methods, there are only some adjustments to be made in order to fully integrate the proposed solution. Another variant is to have an ad-hoc cluster based on virtual machines inside the cloud. Provided that the hardware exposes the appropriate integration method for an accelerator to a virtual machine, the cluster can be viewed as a traditional one, described in the following section.

Other solutions such as the DAC architecture [3] or the service oriented approach presented in [4] could be deployed if the jobs inside the cluster share most of the accelerated functions.

HPC Cluster integration

High performance clusters are composed of many computing nodes which can execute user submitted jobs. Usually, a cluster can be viewed as a non-interactive, submit – run — retrieve, resource, as opposed to interactive cloud resources. Multiple clusters can be connected to form a computing grid. Such an architecture can be regarded as a cluster of clusters, a super cluster, in which a user with the appropriate credentials can launch its job without knowing on which physical grid site it would execute. A typical example of a world-scale computing grid is the

DOI: 10.21279/1454-864X-16-I2-060

CERN's WLCG – Worldwide LHC Computing Grid.

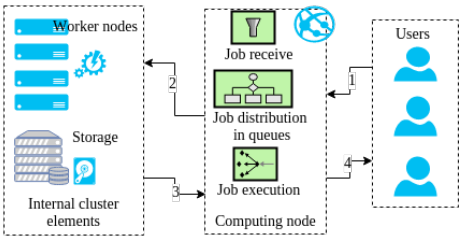

Figure 1- Cluster architecture

A generic cluster architecture, as in Fig. 1, illustrates the typical work-flow [2]. The user submits (1) a job, the computing node identifies a suitable run queue and transfers the job to that node (2) where it is executed (3). The result is sent back to the user (4) where it can be examined. Such a job has a specific definition in which a minimal set of required resources can be specified. Among the job's specification, an attribute that indicates a FPGA accelerator requirement can be set. The batch system on the Computing Element works with a resource scheduler in order to satisfy certain Quality of Service needs. Several scheduling approaches for high density heterogeneous clusters have been proposed in [6],[7] and [8].

When the system administrator configures the cluster, it can set specific properties for the worker nodes. The attributes declare a node's capabilities in terms of number of computing cores, available memory, dedicated graphic cards, specific software, and so on. In our FSAP context a new attribute such as “fsap-fw” can be set to identify hardware accelerated nodes. One of the most widely deployed batch system, PBS, can easily be configured in this way either by configuration files:

[/var/lib/torque/serv_priv/nodes]
node001np=24
nodeN np=X fsap-fw-0x100

or at runtime:

qmgr -c “set node node010 properties += “fsap-fw-0x200”

When a submitted job requires hardware acceleration all it must have is such an attribute as a requirement in the definition. Multiple attributes can be set in order to specify a set of compatible firmwares. As described above, each firmware can support multiple acceleration modules. When an application requests a particular module, the shared library identifies which firmwares are suitable for execution. Identification is done by searching all the available firmware for compatibility with the requested module. In the configuration file, each firmware section contains the *allowed_modules*parameters which is a list with all the module names that are supported by that firmware.

Given the fact that the shared library must be vendor-independent with regards to the batch system, the most convenient way to assure interoperability between the batch system and the FSAP platform is to use generic wrapper scripts for actions that have impact on the batch system. This method assures integration transparency and eliminates the need for multiple connectors which would increase the development and maintenance complexity.

In case of PBS the submission procedure must be altered in order to notify the FSAP daemon about a new job being ran. The simple wrapper easily does the job:

```
fsap_qsub.sh -s myscript.pbs -m 1234 -m 2345
#!/bin/bash
mods=""
whilegetopts ":s: :m:" opt; do
        case $opt in
        s) script="$OPTARG"; ;;
        m) mods="$mods$OPTARG "; ;;
        esac
done
fwid=`fsap_get_fw_by_mods $mods"
if (( fwid ==0 )); then
        echo "No firmware found!";
else
        fsap_enqueue_job $fwid
fi
```

The script takes as arguments the original batch job and the required modules, identified by their IDs as defined in the FSAP configuration files. Called functions inside the script are exposed by the shared library and are detailed in the following sections. Given the fact that such a framework requires additional attention, cluster users should be aware that for benefiting from accelerated functions, a specific submit method, different from the cluster's native one, must be used.

When the job is launched on the worker node, the cluster client launches an application, according to the job definition, after it has prepared the environment. The actual calls to the accelerator reside in this application and are entirely the user's responsibility. In a typical situation, each cluster node would have local libraries that facilitate the communication between the userspace application and the accelerator, most often with the help of a kernel module.

Communication inside the cluster-like

Inside a cluster that implements the proposed platform runs at least a daemon (FSAPD) to which clients (FSAPC) connect. The main roles of this additional communication system are: firmware reconfiguration, node monitoring and accelerator-related event logging. One or more daemons can exist inside a cluster in order to meet situations that need fine-tuning and have strict execution deadlines. For a more complex architecture, that requires different firmwares or has a tight schedule with regards to reboots, the platform can

DOI: 10.21279/1454-864X-16-I2-060

implement a partitioning scheme as described in Figure 2:

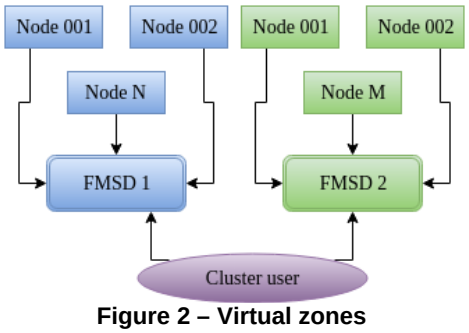

Figure 2 – Virtual zones

The user can manually choose a virtual execution zone which is controlled by a specific daemon. By selecting a particular zone, it can target specific systems. For example, if a cluster is divided into three zones – production, normal and development, a user can safely submit probe applications to the development zone without disrupting the operation of the production systems.

The communication protocol between clients and servers is stateless and has a restrictive set of commands with well-defined arguments. The main functions of the endpoints are: *Log(node)* – sends periodic information for logging purposes. By using this function, the daemon can detect certain conditions or errors and take action. *Reprogram()* signalizes the necessity for loading a new firmware. This event marks the node as temporary offline, until it is reprogrammed. *Isfree()* is used at the end of the reprogramming flow to signal that the node can execute jobs. *Reportfw()* acts as a keep-alive watchdog and also reports the current firmware. As with the case of the batch system, this functions can call wrapper scripts that are tailored for a specific environment.

Reconfiguration requests are queued into a First In First Out structure and are de-queued after the node is rebooted and has confirmed its status. This happens inside the FSMD. After the node is operational, jobs are not directly submitted with the cluster's native interface; instead the wrapper is used again. This method assures that the firmware version is checked again and the reprogramming was successful. Specific tuning might be needed in order to keep the rebooted node reserved until the original job that has triggered the reboot is queued to the node. Otherwise the node might start to run other waiting jobs and the original job would still be waiting.

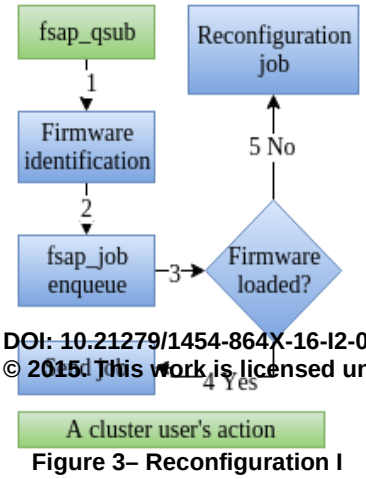

Figure 3– Reconfiguration I

Firmware reconfiguration on nodes

In the previous section, a call to *fsap_enqueue_job* was done inside the submit wrapper to start the execution for a specific job. This procedure has the critical role to check if there is at least one accelerator that runs the required firmware. If there is none, a worker node will be designated to reprogram its accelerator and reboot. This is done by a special job submitted to the cluster. The reconfiguration flow can be divided in: user actions, FSAP management system actions and FSAP client actions on a working node.

After a user executes the wrapper, the framework identifies, at steps 1 and 2, the firmware and version required and checks (3) if such a firmware is loaded anywhere in the cluster. If yes, the job is send to a node that matches the submit criteria. If the result is negative, the daemon must issue a reconfiguration request. From this point on, the management daemon takes control of the process and oversees the remaining steps. A cluster, based on its dimension, can only sustain a certain amount of different firmware at a given moment. In case of large clusters, the reprogramming probability decreases due to their large amount of resources. Otherwise, reprogramming can occur more often as the number of users (with specific needs) increases. If no firmware has been found, at step 5, the management system needs to send a reconfiguration job. The nod on which this job is executed will have to be rebooted in order for the FPGA firmware to properly load and interact with the host operating system.

The last part of the reconfiguration flow is handled without any user interaction and starts as soon as the job is transferred to the working node and starts executing, at step 7.

DOI: 10.21279/1454-864X-16-I2-060

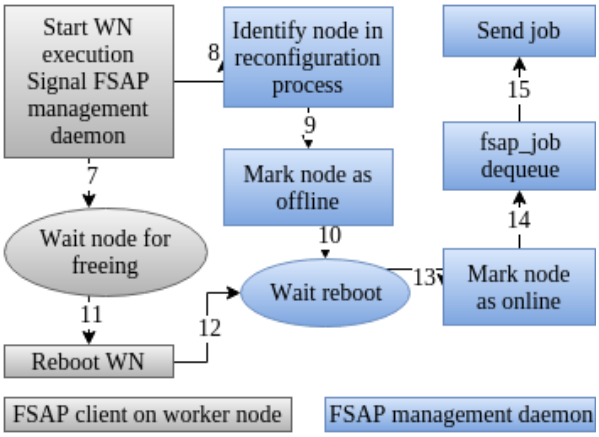

Figure 4 – Reconfiguration II

The job will signal the management node (step 8) and wait for any jobs that are still running – step 7. The FSAP daemon will communicate to the batch system to mark the node as offline, to prevent any new jobs from being submitted to it. After there are no more jobs left, at step 11, the client application reprograms the accelerator and reboots the node. Meanwhile, the daemon will detect through *Isfree() and Reportfw()* functions when the node is rebooted and checks if everything is in order. After the node comes back online, step 13, the daemon updates the node's attributes in the batch system because it has to reflect the new firmware version. The node is then marked as online (14), the original job is removed from the FIFO and is re-submitted. At this point, if everything went without any errors, the job should be executed on the node as soon as there is an open slot on it.

It is important to differentiate the need for integral FPGA reprogramming as opposed to partial reprogramming. Full reprogramming requires a reboot and is only needed when a desired module cannot be fitted into the current firmware. Partial reprogramming is the key element to minimize idle times caused by rebooting. In most of the cases, a firmware will attempt to accommodate a number of modules of a certain type. Each module, designated small, medium or large in the configuration file, uses a fixed number of FPGA resources and special elements, such as DSP blocks. If the requested module cannot be fitted onto the current firmware but a compatible firmware is available the middle-ware will trigger a full reprogramming request. If the required module is already loaded (best-case) or can be fitted in the current firmware then the middle-ware initiates a partial reconfiguration flow.

CONCLUSIONS AND FUTURE WORK

FPGA based accelerators are being more frequently used in high performance computing centers. Their reprogramming capabilities along with modern technologies, that enable them to run at high clock speeds, make them the ideal candidates for replacing the classic ASIC based hardware accelerators. Many of these accelerators come in the form of PCI Express expansion cards. Thus a single physical server can host multiple accelerators that can be used by local applications or forwarded to virtual machines by pass-through technologies.

In a typical large high performance cluster, the nature of the submitted jobs, in terms of called functions, is heterogeneous. This is the result of users from different fields working on the same infrastructure.

As the FPGAs are a finite resource, in terms of the number of hardware acceleration modules that can simultaneously exist on one accelerator, dynamic reconfiguration must take place to successfully load and execute a particular module.

The FPGA Shared Acceleration Platform, described in the current document, proposes a method of managing the reconfiguration requirements of the accelerators inside a cluster. It is minimally invasive, fully transparent and provides vendor-independent integration with both batch systems and accelerator providers. Its client-server architecture assures a consistent communication channel based on stable technologies such as web services or pure IP stack.

The platform seamlessly integrates the reconfiguration flow of an accelerator into the cluster's queue-based execution system. A well-defined configuration along with a code sharing and versioning system allows different working groups to easily exchange modules and firm wares.

As many batch systems have already included in their standard attribute sets the Graphic Processor capabilities for the worker nodes, FPGA capabilities will be present in the near future. A control system, as the one described here, would be integrated in the core of the batch system. In such a scenario, the normal user would not need any particular training for operating the cluster. Computing grids would have an ecosystem from which each participating cluster could just pull a particular FPGA image for its nodes and start running hardware-accelerated jobs. Such a case would have to address multiple constraints such as FPGA vendors, chip families and area constraints.

BIBLIOGRAPHY

DOI: 10.21279/1454-864X-16-I2-060

[1] Dumitru, Laurentiu Alexandru. "Leveragins FPGAS and SDNS for high speed ipc in high performance computing clusters."*Scientific Bulletin" Mircea cel Batran" Naval Academy* (2015)
[2] Orlandea, C. Coca and L. Dumitru, "High-performance computing system for high energy physics." ROMANIAN JOURNAL OF PHYSICS 56.3-4 (2011): 359-365.
[3] Prabhakaran, Suraj, et al. "A dynamic resource management system for network-attached accelerator clusters." *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013.
[4] Knodel, Oliver, et al. "Integration of a highly scalable, multi-FPGA-based hardware accelerator in common cluster infrastructures." *Parallel Processing (ICPP), 2013 42nd International Conference on*. IEEE, 2013.
[5] L. Shi, H. Chen, and J. Sun, "vCUDA: GPU accelerated high performance computing in virtual machines," in Proc. of the International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2009.
[6] L. Barsanti and A. C. Sodan, "Adaptive job scheduling via predictive job resource allocation," in Scheduling Strategies 782 for Parallel Processing. 140.
[7] S.-S. Boutammine, D. Millot, and C. Parrot, "A runtime scheduling method for dynamic and heterogeneous plat-forms," in Parallel Processing Workshops, 2006.ICPP 2006 Workshops. 2006 International Conference on, 2006,pp. 8 pp. –282.
[8] D. Kumar, Z.-Y. Shae, and H. Jamjoom, "Scheduling batch and heterogeneous jobs with runtime elasticity in a parallel processing environment," in Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International, may 2012, pp. 65 –78.
[9] Lockwood, Glenn K., MahidharTatineni, and Rick Wagner. "SR-IOV: Performance Benefits for Virtualized Interconnects." *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*.ACM, 2014.

DOI: 10.21279/1454-864X-16-I2-060