Ubiquitous FPGA Access for Data Intensive Computing

Julio Dondo, Francisco Sanchez Molina, Fernando Rincon, Francisco Moya, Juan Carlos Lopez
University of Castilla-La Mancha - UCLM
Ciudad Real, Spain
Email: juliodaniel.dondo, francisco.smolina, fernando.rincon, francisco.moya, juancarlos.lopez @uclm.es

***Abstract*—The availability of ubiquitous distributed reconfigurable devices allows the implementation of different functionalities everywhere using remote resources. This work presents a development model introducing an efficient and secure manner of managing these remote and distributed reconfigurable resources. This model is a promising model to be used in scenarios where data intensive and high performance computing are needed, because it allows acceleration of applications using distributed hardware.**

***Index Terms*—reconfigurable logic; distributed architectures; FPGA; HPC;**

I. Introduction

A. Data Intensive Computing

Data-intensive computing capabilities are fundamental for data-intensive scientific research such as biological systems, weather prediction, simulation of statistical mechanics, as well as analyses of huge volumes of different data types to and from several distributed sources. Examples of these situations can be found in power grids (Smart Grid), or in a distributed-sensors network (cameras, sensors, etc.) used for security, and disaster prevention, among others. These networks may acquire large amounts of data and the data processing also may need to be done locally, and in real time, requiring some computation-intensive task, e.g. edge detection in images. Even more, in these networks each node can be power-constrained and then a data-intensive task can be limited by power consumption.

In a wireless sensor network, nodes are low-cost sensors operating in an environment with limited processing power and restricted battery autonomy. In this scenario data processing can be done locally, using nodes equipped with dedicated processors. Only the results of the processed data are transmitted to the base station. This approach has the advantage of consuming less bandwidth when sending data, but it has to deal with power restrictions. Another approach is to process data remotely. In this case a high volume of data is transferred through the network. One way to solve the disadvantages of both approaches is through the use of low power reconfigurable hardware (FPGAs) dedicated to performing data processing locally. Data Processing algorithms can be implemented in these FPGAs obtaining n improved time response with low power consumption. In a system with several sensors and actuators, FPGAs can be configured in order to perform different data-intensive processing, liberating the sensors from the data processing [7]

B. High Performance Computing

Currently there exists a great demand for high performance computing due largely to the requirements of scientific research. For these applications, the use of accelerators offers a qualitative improvement in terms of computing power and consumption, compared with the classical solutions of general purpose computing.

FPGA-based scientific computation is one of the solutions in the scientific community to improve the response time for numerically-intensive computation. FPGA-based systems are faster than a software-only approach in terms of computational power [5] [1].

Approaches for high performance reconfigurable computing, (HPRC), integrates both processors and FPGAs into a parallel architecture. HPRC can achieve an improvement in speed, size, and cost of several orders of magnitude over conventional supercomputers [6].

Another solution widely used to improve response time in intensive computation consists of the shared use of distributed resources [8]. Grid computing allows the connection of scattered computational resources enabling the use of their computer capacity, data, and storage space, regardless of their location. To deal with such diversity of resources and local administration policies virtualization technology has been used to abstract the demands and use of resources from the sujacent hardware platform [3].

Facilities to accelerate computational problem using reconfigurable hardware, and the availability of these reconfigurable resources as in a distributed grid system, can be combined to create a computing resource capable of making significant breakthroughs in data-intensive computing.

Distributed reconfigurable resources can be integrated according to the grid model, allowing the creation of a distributed reconfigurable platform for data-intensive computing as in sensor network or Smart Grids. Even more, this distributed reconfigurable platform is ideal also for high performance computing for the resolution of scientific computational problems. This platform is depicted in figure 1.

This platform that we call Reconfigurable Grid (R-GRID) is not just a set of spatially distributed FPGAs but also includes a set of functionalities to provide the ability of a transparent implementation of concurrent distributed applications, in order to obtain maximum benefits from reconfigurability and spatial distribution of resources. R-GRID platform was designed to

facilitate the implementation of multiple-users, and multiple-application-instances, in such a way that clients are unaware of the complexity inherent in the subjacent grid of FPGAs (thereby avoiding the need to provide implementation details), and to provide services that guarantee the security of deployed applications.

The R-GRID approach must deal with the complexity of distributed heterogeneous FPGAs. It is necessary to decouple the behaviour of the hardware resource from the physical implementation level. For this, R-GRID uses virtualization techniques allowing the sharing of the same resource to different users as if it were a local resource, and providing the same interface to geographically distributed users. Each user sees a virtual FPGA where it can deploy its components. R-GRID also provides the virtualization techniques to connect and communicate distributed components using a middleware approach.

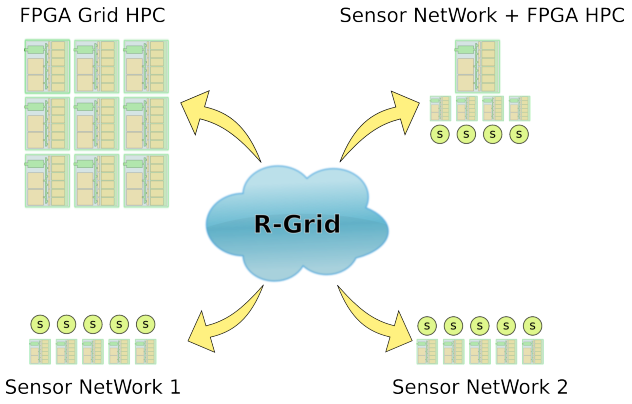

Figure 1. R-GRID

II. The R-GRID concept

The R-GRID main objective is to provide a safe, reliable and accessible platform where clients can implement and deploy their accelerated applications. For that it is necessary to provide a set of functionalities to facilitate the deployment of applications and the use of resources.

A. The R-GRID Logical Architecture

R-GRID platform is built upon a different set of services, with the purpose of providing a framework for the execution of high-performance distributed applications. Figure 2 describes the logical architecture of the platform, which has been divided into three levels: the user administration level, the platform management level and the node level.

User Administration Level and Platform Management Level are implemented in software and are part of the R-GRID Server. R-GRID Server is responsible for administration of the whole system, keeping information about descriptor of applications, registered users, correspondence between applications and owners, components and nodes, used resources, available resources, etc.

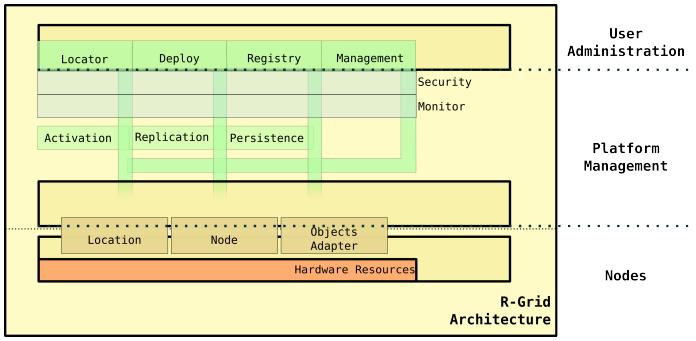

Figure 2. R-GRID Logical Architecture

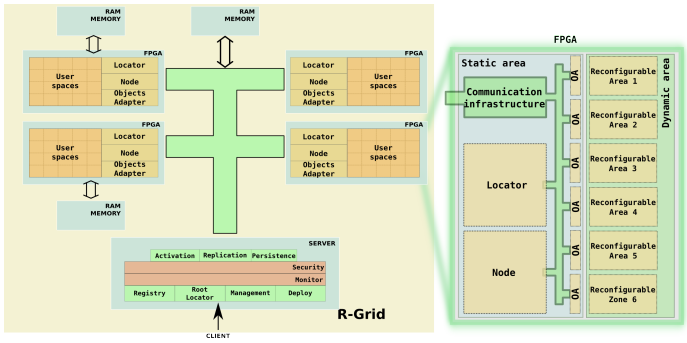

Figure 3. R-GRID Physical Architecture

1) User Administration Level: To start using R-GRID, the Server offers a user interface, first level, which is formed by the Registry, Deploy, Root Locator and Management objects: The Registry is an object with a public interface that allows clients with the proper access rights, to register a new application in R-GRID. A repository of applications is also offered that can be used anytime and from any location. The Deploy object is accessed by users to perform the deployment of registered application. Each application can be formed by one or several components and each component has an associated bitstream. Each bitstream has also associated a node. After deployment the Deploy object will update the Root Locator Object to register component address. The Root Locator object is a component that keeps information of the deployed components and their location. It is intended to provide access to the components.

Management Object gives to the administrator access to all system functions, to ensure the proper behaviour of the R-GRID. With the management service, the administrator can obtain information about the availability of resources, the state of FPGAs nodes, addresses of deployed objects and can modify this information to solve any problem that arise.

2) Platform Management Level: As a second level we found Platform Management. At this level transversal platform management decision are taken. Issues concerning security, monitoring, activation, component replication and persistence take place at this level.

a) Security: Security module is implemented taking into account two different points of view, user and application. In

the first aspect, security module will authenticate and authorize users, checking if they have privileges to perform required actions. From application point of view, due to the fact that in R-GRID different applications from different owners over shared resources are running at the same time, control of application access needs to be done at each computing node level of R-GRID. At this level, security functionality will ensure that each component of each application can contact and can be contacted only by components that belong to the same application. This functionality is performed by the Object Adapter on each FPGA.

b) Monitoring: Monitoring functionality collects all the information about use and availability of resources, status data, load and queue status, to provide information to the administrator to facilitate grid utilization and resource brokering.

c) Advanced functionalities: Activation, Replication and Persistence are advanced functionalities that are invoked by Root Locator and Deploy objects.

Activation: is a mechanism that instantiate a component when it is required by other component of the same application, if it is not in the system. This situation can occur if a component of the application was removed to liberate resources and it is required later.

Replication: is a functionality that allows component replication, if application and resources availability permit.

Persistence: In case that an instantiated component is removed or stopped from the grid, Persistence allows to saving its state in order to be later reused if component is reinserted. With this functionality, components can be removed from the grid and reinserted in another place, without lost of data consistency.

B. The R-GRID Physical Architecture

The NODE level in figure 2 is formed by heterogeneous FPGAs nodes, so this level is the physical architecture level. R-GRID intended to hide the implementation details of computing nodes to upper levels. Relationship between logical and physical architectures can be observed in the figure 3.

R-GRID physical architecture was implemented using dynamically reconfigurable FPGAs. Each FPGA node is divided in two parts: the static part, formed by three objects: Locator, Node and Object Adapter, and a dynamic part, which is formed by several dynamically reconfigurable areas. Each reconfigurable area can host a component, and the location of each instantiated component in the FPGA is solved by the local Locator object.

The partial reconfiguration is triggered by the Deploy object at User Administration level and performed by the Node object of the target FPGA. Node object takes the bitstream from memory and performs a partial reconfiguration of the FPGA. This object isolates the partial reconfiguration mechanism inherent to each FPGA from upper levels. Node object presents two methods: StartServer (/bitstream/path) to load the partial bitstream, and StopServer (int area) to stop the bitstream.

The Object adapter provides the mechanism to control and to avoid the access to instantiated components from those belonging to different applications, providing security at application level. Each object adapter has a unique global address within the system. These components are entirely hardware implemented.

This model is perfectly scalable to a set of FPGAs that are not partially reconfigurable. Each FPGA is registered in the Locator at the first level of architecture, the Deploy Object, instead of invocate the Deploy node to reconfigure the FPGA, proceed to reconfigure the target FPGA directly. Each FPGA is configured offering to Deploy object the same interface. From the registry point of view the hardware resources interface is the same.

C. Communication interface

Components can be locally or remotely connected. In the former case it is equivalent to a point to point connection from the client to the server, and implies no additional logic, while in the latter it involves the use of remote adapters, in order to translate messages into the transport protocol used by the communication link. R-GRID extends the functionality of the OOCE (Object Oriented Communication Engine) middleware [2] in order to enhance inter FPGA communication of data. The base configuration in OOCE isolates the computational aspect from communication details, mainly at chip level. In this scenario, software components can access to hardware components and vice versa. OOCE interface compilers generates the software and hardware adapters that automatically manage hardware-software interfacing. These adapter are in charge of translating bus request and module activation signals in one domain to conventional calls in the host processor. A more detailed description about the adapters and how they can be automatically generated is provided in [4].

An extra effort has been done in the abstraction of the memory interfaces and the memory hierarchy in R-GRID. For example, external memory to a component can be implemented as a shared memory in the same node or any other kind of memory in a different node. In any case, the middleware provides the component with the MemoryResource abstraction, through an adapter, so the physical access to such resource is decoupled from the components logic. Such interface represents a generic memory and provides methods for reading and writing data blocks. The addressing of the different memories is delegated to the proxies together with the global addressing system used in R-GRID.

D. Application Deployment

Once the set of components of the application have been defined, the next step will be the description of the application. For such purpose R-GRID uses an XML file, where all the components in the application are listed and the corresponding binaries are assigned. Let suppose an example of application with two cores to be placed in different resources. Figure 4 shows the XML description. In this simple case there are only two components, the controller of the application and the core processor. The task of the controller is to schedule the data distribution and execution of the core, so it has been mapped

```
<rgrid>
  <application name="Simple">
    <component id="controller"/>
    <component id="core"/>
  </application>
  <binaries>
    <binary component="controller" type="sw"
        bin="/opt/rgrid/controller.exe"   />
    <binary component="core" type="virtex4"
        bin="/opt/rgrid/core_v4.bit"      />
    <binary component="core" type="virtex5"
        bin="/opt/rgrid/core_v5.bit"      />
  </binaries>
</rgrid>
```

Figure 4. XML Description of the example application

to a software node, while the core is a hardware component. Both of them have at least an associated binary file: a software executable for the controller, and a two hardware partial bitstreams for the core, to deploy it over two different kind of hardware resources. Those binaries will be deployed at run-time from the R-GRID server to the corresponding nodes.

To deploy an application it is necessary to perform a few steps. First, the registry of an application using XML file as indicated before. With this configuration file R-GRID Server updates grid information. After registered, it is necessary to invoke Deploy object. Deploy takes information from R-GRID Server and send to the corresponding Node information about bitstream location to start reconfiguration. Finally, Deploy update address information to Root Locator and local Locator, in order to allow access to new component.

III. Implementation

To evaluate our approach a model of grid was implemented using Virtex4 vlx60 ff668-10 FPGA. In this FPGA a node and a locator were implemented. The resources used for both components are shown in the following tables:

Locator		
Number of	Usage	Percentage
Slices	281 out of 26624	1
Slice Flip Flops	159 out of 53248	0,2
4 input LUTs	528 out of 53248	0,9
IOs	104	
bonded IOBs	0 out of 448	0

Node		
Number of	Usage	Percentage
Slices	176 out of 26624	0,6
Slice Flip Flops	162 out of 53248	0,3
4 input LUTs	331 out of 53248	0,6
IOs	211	
bonded IOBs	0 out of 448	0

In this model were created 8 reconfigurable areas using locally independent clocks obtaining an easier and more efficient place and route. The defined areas are symmetrical allowing the interchange of bitstreams. The DDR memory controller alllows an independent access channel to memory for each area, prioritizing concurrent access using rotatory shifts. The DDR memory controller also allows independent memory partition for each implemented application.

IV. Conclusion

This article presents a new infrastructure for data intensive and high performance computing based on FPGAs. Ubiquitous reconfigurable resources can be integrated according to the grid model allowing the creation of a reconfigurable platform oriented to concurrent and decentralized exploitation of heterogeneous cluster of FPGAs. The combination of location transparency of components, (the capacity to invoke any object as it was a local invocation), access transparency (the capacity to reach any hw or sw component), and remote activation of the reconfiguration process, provide great benefits in the overall system design process. In environments where power constraints are mandatory, this approach allow the use, management and configuration of low power, low cost, FPGAs remotely and, in this way to take advantages of the reconfigurability of these devices.

V. Acknowledgement

This work has been founded by the Spanish Ministry of Science and Innovation under the project DADA (TEC2008-06553/TEC), and by the Regional Government of Castilla-La Mancha under project RGRID (PAI08-0234-8083)

References

[1] W. J. Gross A. J. Wong. Configurable flow models for fpga particle graphics engines. In *16th International Symposium on Field-Programmable Custom Computing Machines*, pages 283–284, Washington-USA, 2008.

[2] J. Barba, F. Rincon, F. Moya, F. Villanueva, D. Villa, J. Dondo, and J.C. Lopez. Ooce, object-oriented communication engine for soc design. In *Proc. X Euromicro Conf. on Digital System Design (DSD)*, Germany, 2007.

[3] Sumalatha Adabala et al. From virtualized resources to virtual computing grid: the in-vigo system. *Future Generation Computer Systems - Special Section: Complex problem-solving environments for Grid computing*, 21, 2005.

[4] F. Rincon, F. Moya. J. Barba F. Villanueva D. Villa J. Dondo and J.C. Lopez Transparent ip cores integration based on the distributed object paradigm. In *LNEE- Intelligent Technical Systems*, volume 38, pages 131–144. Springer Netherlands, February 2009.

[5] V. Rana. M. Santambrogio and D. Sciuto. Dynamic reconfigurability in embedded system. In *Proc. IEEE ISCASS 2007*, pages 2734–2737, 2007.

[6] M. Huang K. Gaj V. Krindatenko D. Buell T. El-Ghazawi, E. El-Araby. The promise of high-performace reconfigurable computing. *Computer*, 41(2):69 – 76, Feb 2008.

[7] Y. Kwok T. Kwok. Computation and image processing in wireless sensor network based on reconfigurable computing. In *Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06)*, pages 43–50, 2006.

[8] R. Sun Y. Ouyang, L. Yang. Research on grid platform oriented to intensive data processing. In *Seventh International Conference in Grid and Cooperative Computing*, pages 226–231, 2008.