2012 41st International Conference on Parallel Processing Workshops

On Virtualization of Reconfigurable Hardware in Distributed Systems

M. Faisal Nadeem, M. Nadeem, and Stephan Wong
Computer Engineering Laboratory, Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology, The Netherlands
{M.F.Nadeem, M.Nadeem, J.S.S.M.Wong}@TUDelft.nl

***Abstract*—In the design of next-generation distributed and high-performance computing systems, Reconfigurable Processing Elements (RPEs) such as FPGAs and multi-core heterogeneous computers will play an important role. FPGAs are well-known for their programmability, power efficiency, reasonably high performance, functional flexibility, and their scalability. However, proper virtualization schemes at higher software abstraction layers to utilize these RPEs are under research. In this paper, we propose a virtualization framework for distributed computing systems that supports RPEs. First, we present various scenarios in terms of use-cases to discuss the utilization of RPEs in distributed computing systems. Secondly, we propose a scheme to virtualize RPEs in distributed systems. Based on various virtualization levels, we provide a general model for a computing node which incorporates both General Purpose Processors (GPPs) and RPEs. Thirdly, we present a typical application task model. Finally, we present a case study of a large-scale application from the bioinformatics domain, which demands different types of processing elements in a distributed computing system.**

***Index Terms*—Resource virtualization; Reconfigurable processing elements; Distributed computing systems; Use-case scenarios; Case study.**

I. Introduction and Motivation

Distributed computing systems, such as grid networks utilize computing resources that are geographically distributed over the globe to perform computations for large-scale scientific applications that exceed the capabilities of clustered desktop computer or even a supercomputer [1][2]. Generally, the overall performance of a distributed computing system greatly depends on the processing power of the employed computing resources and till now, the main processing elements in these systems were (programmable) general-purpose (multi-/many-) core processors. However, due to growing demands of new scientific applications, more performance and power efficiency are required from the processing elements [3]. Therefore, new possibilities are opening up in order to utilize Reconfigurable Processing Elements (RPEs) such as FPGAs, in distributed systems [4][5].

In recent decades, FPGAs have obtained growing attention due to their flexibility, power efficiency, and ease of use [6]. Some of the characteristics of reconfigurable hardware include, adaptability and short design time, functional flexibility, ease of use, extensible (adding new functionality), power efficiency, reasonably high performance, hardware abstraction, and scalability by adding more cores. Most significantly, the reconfigurable architectures are utilized as hardware accelerators in order to increase performance. In addition, the (re-)configurability of these architectures means that they can be optimized for different applications without overhead operations that are required when using traditional architectures result in less resource waste and reduced energy consumption. Furthermore, they also provide the programmability of a general-purpose processor. Therefore, we propose a generic virtualization framework for a computing node in a grid system, that contains RPEs along with GPPs.

However, the design complexity and utilization of the RPEs in a distributed grid system lead to innovative research and open problems. One such problem in grid computing is the virtualization of hardware by hiding the details of the underlying hardware. Virtualization allows several application tasks to utilize resources by putting an abstraction layer between the tasks and resources.

The traditional grid systems are already virtualized for GPPs; hence, they are often termed as *Virtual Organization,* and there is a *hardware independent layer* between application developers and resources [2]. Although many previous attempts have been made to virtualize reconfigurable hardware in order to utilize them effectively for more than one application tasks [7][8][9], but with many limitations. Their main focus is on a single device [8][10] or a multi-FPGA High Performance Reconfigurable Computing (HPRC) system, and none of them discusses the virtualization for RPEs in a distributed grid computing system.

In this paper, we propose a general framework to virtualize a computing node in a grid system, which contains GPPs as well as RPEs. The proposed virtualization framework can be used to obtain and demonstrate the following objectives:

- More performance can be achieved by utilizing reconfigurable hardware, at lower power.
- Due to abstraction at a higher level, an application program can be directly mapped to any of the RPE or the GPP.
- Reconfigurable hardware is expected to support reconfigurability and different hardware implementations on the same RPE are possible due to reconfigurable nature of the fabric.
- The resources can be utilized in a more effective manner when the processing elements are both GPPs and RPEs. Those grid applications which contain more parallelism

can get more benefit if executed on the reconfigurable hardware.

The proposed virtualization framework is adaptive in adding/removing resources at runtime. Application tasks can be seamlessly submitted to the grid system. The main contributions of the paper are the following:

1) We present and describe a set of various use-case scenarios of applications that can benefit from our presented virtualization framework.
2) Based on the use-case scenarios and their virtualization levels, we provide a generic model for a grid node which incorporates both general-purpose and reconfigurable processing elements. Similarly, we present a typical task model representing an application task which requires a specific processing element for its execution.
3) Finally, we present a specific case study of a real world large-scale application that takes benefit of the proposed framework, and utilizes various processing elements in a distributed system.

The remainder of the paper is organized as follows. Section II presents related work. Section III presents possible use-case scenarios of applications utilizing various processing elements. It also discusses virtualization/abstraction levels on distributed network with reconfigurable elements. In Section IV, we explain our proposed virtualization framework. In Section V, we present a case study to show the utilization of reconfigurable processing elements in a distributed network. Finally, Section VI discusses the conclusions and future work.

II. Related Work

In this section, we provide some related work. In a survey [7], the authors identified three different approaches of hardware virtualization. In the first approach called *temporal partitioning*, a large-scale application is divided into smaller parts which can be mapped onto full FPGA. Then, each part is run sequentially, until the whole application is executed. In the second approach called *virtualized execution*, a certain level of device-independence — for a particular family of FPGAs — is achieved through a proposed programming model that defines an application task as an atomic unit of computation. In this way, a runtime system is developed to ensure that the application can be executed on any FPGA which is part of a device family. In the third approach, a hardware *virtual machine* is proposed which can execute an application on a general abstract FPGA architecture. This provides an even higher abstraction level and is useful for those reconfigurable systems which can be utilized in a network of FPGAs. Similarly, [11] specified general methodologies for the virtualization of reconfigurable devices in hardware and software systems. The authors adopted the concepts used in the operating systems, such as *partitioning* and *overlaying*.

In [8], two different hardware resource virtualization techniques have been proposed, for dynamically partially reconfigurable systems. In the first approach — *logic virtualization technique* — an FPGA is configured for a set of application tasks at runtime. It is a many-to-one mapping which supports many applications, dynamically. In the second approach — *hardware device virtualization* — an FPGA is configured with more than one hardware functions. In this one-to-many mapping, a single software application can utilize many functions on a single device. Similarly, [10] proposed a virtualization layer for dynamically allocating hardware functions on a reconfigurable system, to execute any software application. However, the techniques presented in these works are focused on a single FPGA device.

In [12], the authors proposed an approach to virtualize and share reconfigurable resources in High-Performance Reconfigurable Computers (HPRCs) that contain multi-node FPGAs. The main idea is to utilize the concept of *virtual* FPGA (VFPGA) by splitting the FPGA into smaller regions and executing different task functions on each region. The validation process is carried out by using an HPRC system, Cray XD1, and implementing a layer of OS virtualization to manage the virtual regions. Although, this approach utilizes a multi-FPGA system, but it does not focus on a distributed system containing FPGAs.

All the above-mentioned works attempt the FPGA virtualization for a single node or a multi-FPGA system. However, in [9], an abstract model of *hardware virtual machine* is proposed for the networked reconfiguration of FPGAs. The main focus of the work is to propose a design flow model by identifying the responsibilities of a *client* and a *service provider*. In this model, both the client and the service provider become aware of the FPGA mapping tools — which they are required to maintain — in order to reconfigure the device and execute the application tasks. But in their approach, it is not discussed, how to manage the FPGA resources in the resource management system.

Various different *Workflow Management Systems* in grid computing to match tasks with different computing resources, have been surveyed in [13]. Most important of such systems is the Condor project [14]. It supports a mix of various GPP resource and matches computing jobs accordingly. However, there is no previous work about the efficient utilization of RPEs in such system. It is expected that the future distributed systems will contain RPEs as their main processing elements, along with the GPPs. Therefore, the grid managers seek new methods for the virtualization of RPEs along with existing computing resources. In this paper, we propose a general virtualization approach for a computing node which contains both RPEs and GPPs.

III. Use-case Scenarios

In this section, we discuss all possible use-case scenarios of applications that utilize RPEs. Each of these scenarios have certain characteristics that need to be translated to specific requirements. Therefore, we present our virtualization framework that includes provisions to map the requirements of all possible scenarios to the required RPEs. Figure 1

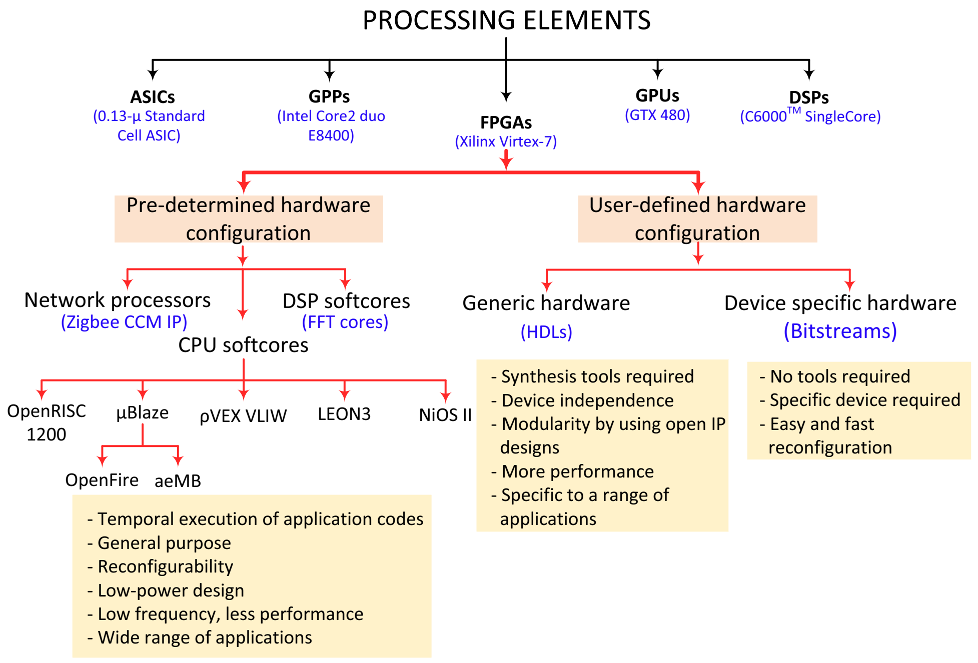


Figure 1: A taxonomy of enhanced processing elements.

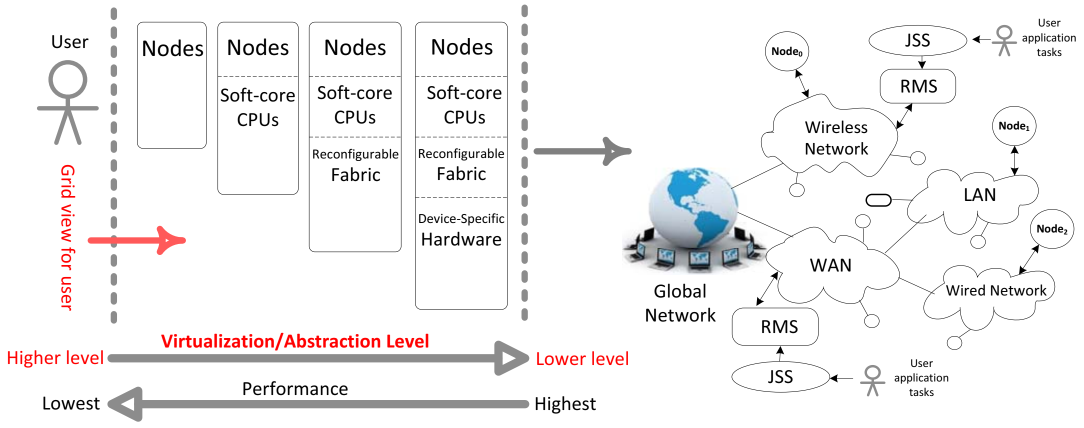


Figure 2: Different virtualization/abstraction levels on a reconfigurable grid system.

depicts a taxonomy of enhanced processing elements in high-performance domain. The framework focuses on virtualization of a computing node in a grid system, which contains GPPs as well as RPEs. However, it is extendable to add more types of processing elements. In the Section III-A, we discuss a scenario with already existing software-only applications, whereas in the Section III-B, we describe hybrid applications which can be executed on either GPPs or RPEs.

A. *Software-only applications*

Many existing applications are already developed for GPP-based grid networks. Therefore, there is a need to provide mechanism on the next-generation *polymorphic* grid networks to provide *backward compatibility* and support such applications. Their performance on the new grid network should be *similar if not better*. Therefore, this use-case scenario covers all the software-only applications already existing for the grid networks. These applications are executed on the GPP nodes and are not aware of the reconfigurable fabric in the grid. However, if the grid system can not provide the required GPP node to an application at some instance, it should be able to configure a soft-core CPU on a currently available RPE to obtain *similar if not better* performance. This use-case scenario is depicted in Figure 1 under the *pre-determined hardware configuration*.

Table I: Parameters of different processing elements.

Processing Element	Parameter	Description
FPGA	Logic cells, Slices, LUTs, Gates, Macrocells, ALMs	Designed to implement user-defined combinatorial and sequential functions.
	BRAM, Memory Blocks, Embedded Memory	Additional memory blocks available in terms of distributed RAM.
	DSP Slices	Pre-configured multiplier, adder, and accumulator required for high-speed filtering.
	Speed Grades	Maximum frequency at which a device can operate.
	Reconfiguration Bandwidth	Speed (in MB/s) to reconfigure a device
	IOBs	Support different I/O Standards
	Ethernet MAC	Embedded MAC for Ethernet applications
GPP	CPU Type/Model	Type of CPU
	MIPS ratings	Million Instructions per Second processing capability
	OS	Operating System
	RAM	Main Memory
	Cores	Total number of cores
Softcores (VLIW)	FU Type	Multipliers, ALUs
	Issue Width	Number of Issues
	Memory	Instruction and Data memory
	Register File	Register file size
	Pipeline	Number and Size of Pipelines
	Clusters	Number of Clusters
GPU	Model	GPU Model
	Shader Cores	Number of Data Parallel cores
	Warp Size	Number of SIMD threads grouped together
	SIMD Pipeline Width	Size of SIMD Pipeline
	Shared Memory/Core	Shared Memory per Core
	Memory frequency	Maximum clock rate of memory

B. Hybrid applications

In this scenario, the application is aware of the reconfigurable fabric on the grid network. Therefore, the application makes use of such computing nodes that contain both GPPs and RPEs. In this way, the performance of the application can be improved. This use-case scenario can be divided into further parts, explained as follows:

1) Pre-determined Hardware Configuration: There could be a situation where some of the *compute-intensive* tasks in a certain application, are implemented in an optimized way to improve the performance. These optimizations for speed are specific to some particular architecture, for example, software kernels (FFTs, filters, multipliers etc.) optimized for VLIW, RISC, μBLAZE, etc. Therefore, these tasks need to be executed on those particular architectures. One example of such architecture is a soft-core ρ-VEX VLIW processor (P_{type}) implemented on an FPGA [15]. Depending upon the requirements of an application, it can be adopted to several parameters such as, the number of issue slots, cluster cores, the number and types of functional units, or the number of memory units.

Such applications consist of generic and the user-selected soft-core specific tasks. The generic tasks are mapped onto the GPPs, whereas the other specialized tasks are executed on the corresponding user-defined soft-cores. In this scenario (see Figure 1), the architecture becomes more general purpose in nature, allowing temporal execution of wide range of applications. It is normally low-power and low-frequency design which is more flexible, but with less performance. Figure 2 depicts different virtualization/abstraction levels for a user. In this scenario, a grid user can view soft-core CPUs, along with the grid nodes.

2) User-defined Hardware Configuration: Open-source (e.g., the OpenCores IPs [16]) hardware are already available for specific tasks in many applications. Therefore, these designs — which are available in generic HDLs — can be reused, while developing the system applications. As depicted in Figure 1, this use-case scenario represents applications which are complex, and cycle (or performance) and/or data hungry. Hence, they consume a lot of time to finish their processing and provide results. For such applications, the performance can be significantly improved by application-specific accelerators which can greatly reduce the time overhead and therefore, an application designer would like to provide such hardware accelerator to be configured on the fabric in the grid. Here, the application developer provides the hardware accelerator specifications in generic hardware description languages, such as VHDL and Verilog. On one hand, this scenario creates opportunities for the grid managers to map the generic hardware accelerators on any of the available reconfigurable fabric in the grid to improve the utilization of its hardware resources. On the other hand, it also provides important grid *services*, such as mechanism and tools to generate device specific bitstreams for the user. In this use-case, the *service provider* is required to possess the synthesis CAD tools. A certain degree of device independence is achieved due to generic nature of user application specifications. However, the design is limited to a specific range of applications. On this virtualization level (see Figure 2), the *reconfigurable fabric* is visible to the grid user.

3) Device-specific Hardware: This is the lowest level of virtualization (see Figure 2) where the user is aware of the device specifications available in the grid. In case the user wants to make use of his own hardware design or IP for a particular device. Therefore, the grid should provide opportunity for such user to submit his/her applications. This scenario is suitable for the users with applications which require higher performance. In this case, the grid user can make use of a hardware of his/her own choice. The hardware is directly visible to the user. The cost of the high performance is long application development time. In this scenario, the *service providers* are not required to possess the CAD tools. However, they are expected to provide the specific device targeted by the application developer.

C. Different Virtualization/Abstraction levels

All these use-case scenarios lead to different virtualization/abstraction levels in a grid system, as depicted in Figure 2. It can be noted that, as we go to a lower abstraction level, the user should add more specifications along with his/her tasks and get more performance, and vice versa. For instance, at the lowest abstraction level, a user must provide device specific hardware along with application tasks.

IV. The Proposed Virtualization Framework: our approach

Based on the use-case scenarios discussed in the Section III, we discuss our proposed framework for reconfigurable hardware virtualization in distributed systems. We propose models for a general grid node and a generic application task which can incorporate both GPPs and RPEs in a grid network.

A. A typical node model

Figure 3 depicts our proposed grid node model to incorporate RPEs, along with GPPs. It contains a list of all processing elements (GPPs and RPEs) and their attribute, and can be defined as follows:

$$Node\,(NodeID, GPP\ Caps, RPE\ Caps, state) \quad (1)$$

A typical grid node contains a list of resources as depicted in Figure 3. Each resource consists of a null terminated list of GPPs, RPEs, and their current $state$. Each data member in the list of GPPs (or RPEs) is characterized by $GPP\ Caps$ (or $RPE\ Caps$), which represents a set of parameters.

These parameters provide information about the capabilities of the GPP (or RPE). A typical list of these parameters is given in Table I. Similarly, $state$ represents the current states of different elements. It is a dynamically changing attribute of the node. For instance, the $state$ can provide the current available reconfigurable area or maintains the information of current configuration(s) on an RPE.

The proposed node model is generic and adaptive in adding/removing resources at runtime. Furthermore, a grid manager can add more parameter specifications of a particular processing element.

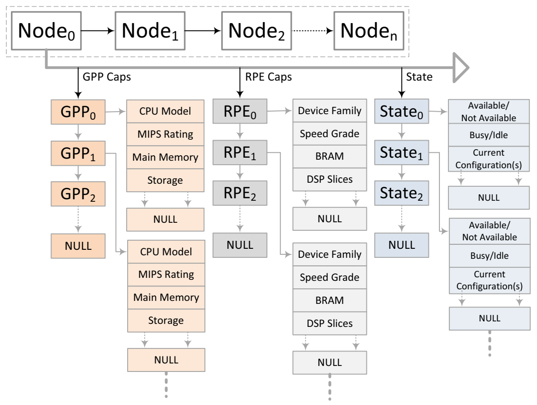


Figure 3: A typical grid node to virtualize RPE.

B. A typical application task model

Figure 4 depicts the proposed model for a typical application task. It is represented by the following tuple:

$$Task\,(TaskID, Data_{in}, Data_{out}, ExecReq, t_{estimated}) \quad (2)$$

In tuple (2), TaskID provides the ID of a task. $Data_{in}$ and $Data_{out}$ identify the input and output parameters of the task. Whereas, the *ExecReq* gives the execution requirements of the task. $Data_{in}$ is completely identified by the ID of the source task (TaskID), DataID, and data size (DSize). Similarly, $Data_{out}$ provides the information about the output generated by the current task and again, it is identified by the DataID and DSize. TaskID, $Data_{in}$ and $Data_{out}$ provide enough information to the scheduler in a grid system to assign this task to a node.

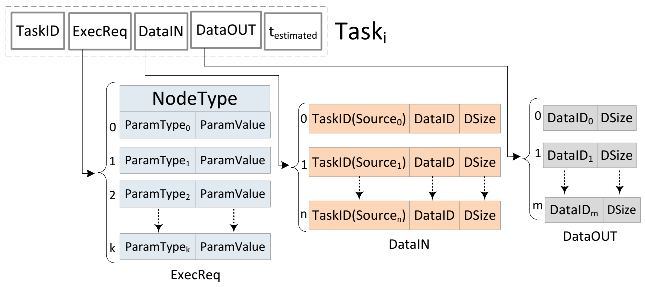
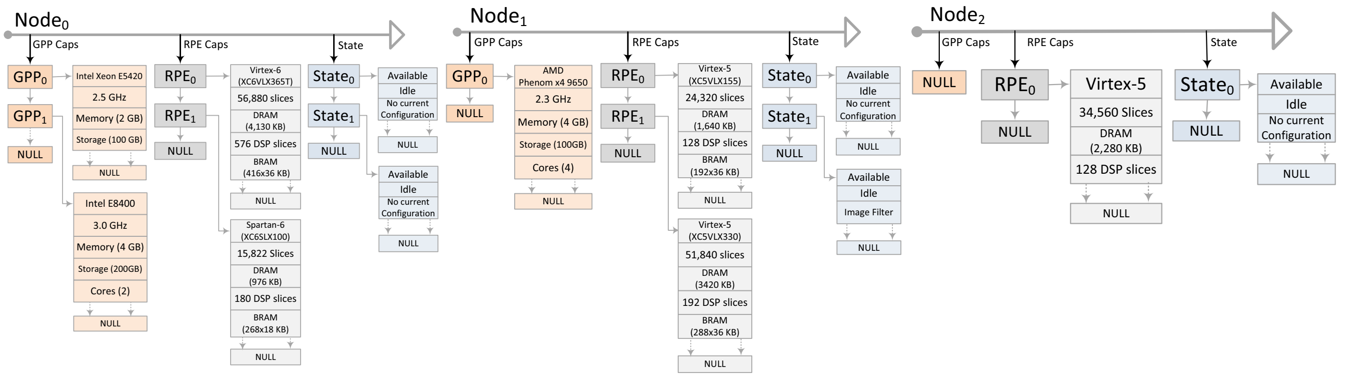


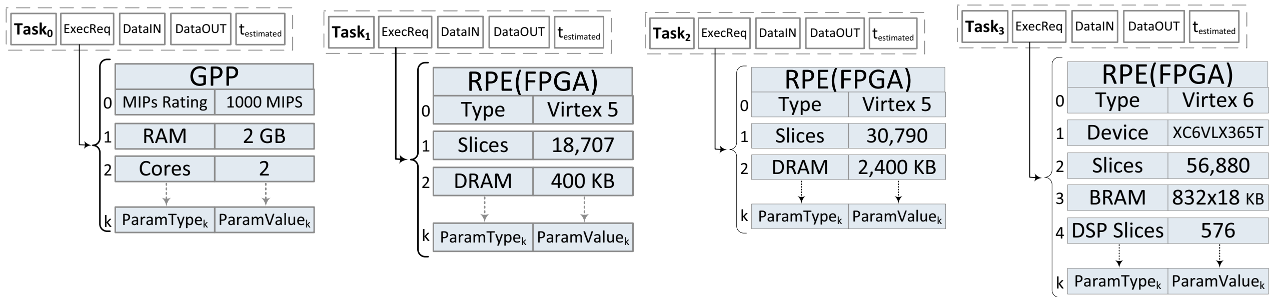
Figure 4: Application task virtualization for grid system with RPEs.

ExecReq provides the list of resources required by the task for its execution. This list is composed of the node type and its parameters. Each parameter is followed by its value. These parameters completely identify the architectural requirements by the current task. Finally, $t_{estimated}$ is the estimated time for completion of this task, when it is executed on a particular processing element, identified by its *ExecReq*.



(a) $Node_0$. (b) $Node_1$. (c) $Node_2$.

Figure 5: Specifications of 3 grid nodes in the case study.



(a) $Task_0$. (b) $Task_1$. (c) $Task_2$. (d) $Task_3$.

Figure 6: Execution requirements for task specifications in the case study.

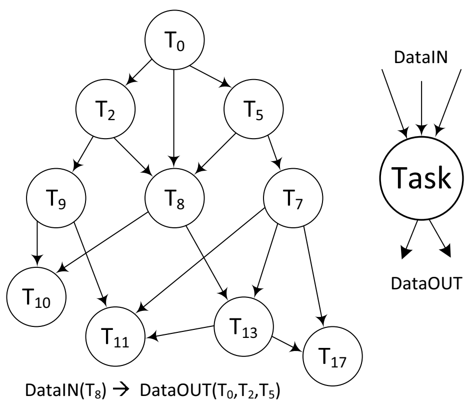


Figure 7: An application task graph.

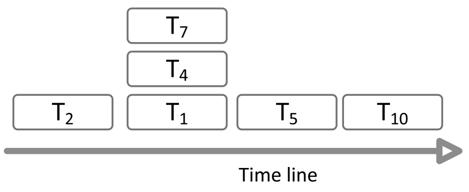


Figure 8: An example of application tasks execution given in tuple 4.

A typical task is illustrated by Figure 4. The input data is represented by DataIN and it can be initiated by n number of sources. Similarly, a task can produce m number of different outputs represented by DataID and DSize. Finally, the *ExecReq* provides list of k parameters which define a typical *NodeType* required to execute the task. The data dependencies among different tasks are represented by an application task graph in Figure 7. From example, it can be noticed that inputs to T_8 are the outputs of tasks T_0,T_2, and T_5. Similarly, $DataIN(T_{11}) \rightarrow DataOUT(T_7, T_9, T_{13})$, $DataIN(T_{13}) \rightarrow DataOUT(T_7, T_8)$, and $DataIN(T_{17}) \rightarrow DataOUT(T_7, T_{13})$.

The user provides its application in form of tasks and their dependencies. A typical application is represented by the following tuple.

$$Application_i\,(< Keyword >, Task\,list, < Keyword >) \tag{3}$$

Each application is identified a keyword followed by a task list. Whereas, a keyword shows whether the tasks can be executed in series or parallel. For example, a particular application is given as follows:

$$App\{Seq(T_2), Par(T_4, T_1, T_7), Seq, (T_5, T_{10})\} \qquad (4)$$

In this example, the task T_2 should be executed sequentially, along with the parallel execution of the task list T_1, T_4, T_7 followed by the task list T_5, T_{10}. The keywords *Seq* and *Par* show that the task lists followed by these keywords should be executed sequentially or in parallel manner, respectively. Each task list is terminated by next keyword. The sequence of task execution in this example is illustrated in Figure 8.

Figure 9 depicts different user *service* levels in a grid system. The minimum level of services required by a user is to submit his application tasks and get results. But more services can be added to satisfy the Quality of Service (QoS) requirements. These services include cost, monitoring, and other user constraints. With these services, a user is able to submit his/her queries and get a response as depicted in Figure 9.

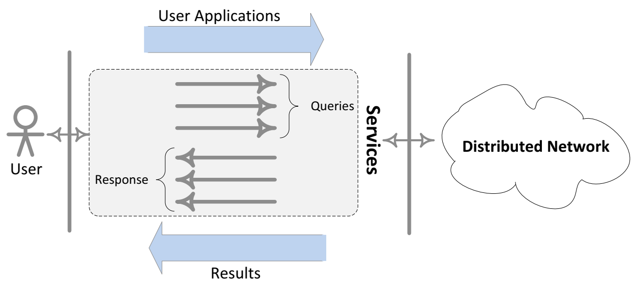


Figure 9: User *services* in a typical grid system.

V. A Case-Study

In this section, we describe a generic case study to provide the concept of virtualization of reconfigurable elements in a grid system. We give an example of a simple application and a grid network consisting of 3 different nodes. Furthermore, we elaborate on the role of resource management system and application task scheduling.

In Figure 2, a general view of a grid network is illustrated. It contains different computing resources in the form of nodes. Each node is characterized by different parameters as depicted in Figure 3. The grid network contains various *Resource Management Systems* (RMS) along with the *Job Submission System* (JSS). A grid user submits his application tasks through a JSS. Each application task is part of a large application and it is depicted by Figure 4. The RMS updates the statuses of all nodes in the grid. It also implements a task scheduler which assigns the user application tasks to different nodes in the network. The scheduling decisions are governed by a task scheduling algorithm and the availability of nodes.

For our case study, we consider a grid network which contains 3 different nodes, denoted by $Node_0$, $Node_1$, and $Node_2$, as depicted by Figure 2. Figures 5a, 5b, and 5c

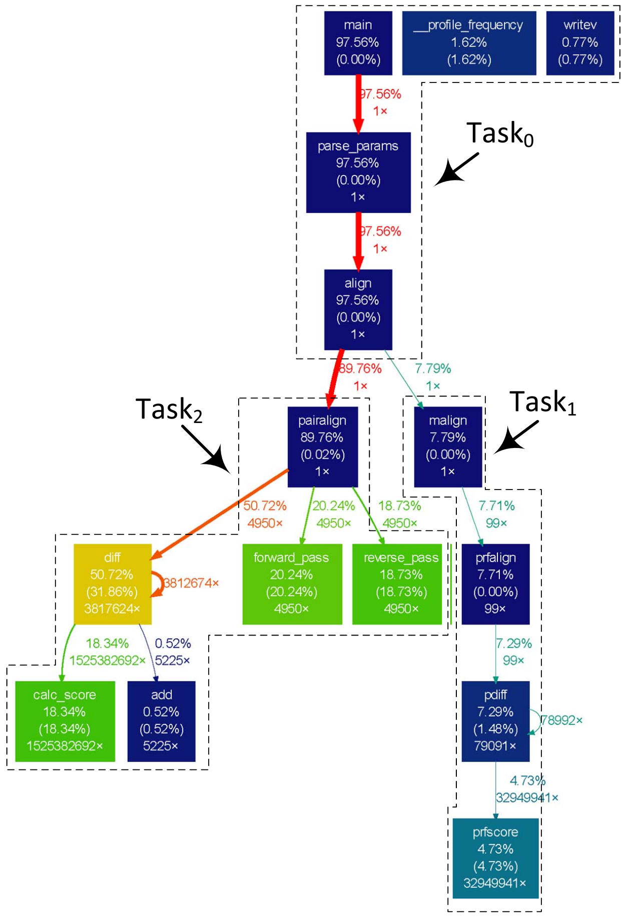

main 97.56% (0.00%)
__profile_frequency 1.62% (1.62%)
writev 0.77% (0.77%)
97.56% 1×
parse_params 97.56% (0.00%) 1×
Task$_0$
97.56% 1×
align 97.56% (0.00%) 1×
89.76% 1×
7.79% 1×
Task$_2$
pairalign 89.76% (0.02%) 1×
malign 7.79% (0.00%) 1×
Task$_1$
50.72% 4950×
20.24% 4950×
18.73% 4950×
7.71% 99×
diff 50.72% (31.86%) 3817624×
3812674×
forward_pass 20.24% (20.24%) 4950×
reverse_pass 18.73% (18.73%) 4950×
prfalign 7.71% (0.00%) 99×
18.34% 1525382692×
0.52% 5225×
7.29% 99×
calc_score 18.34% (18.34%) 1525382692×
add 0.52% (0.52%) 5225×
pdiff 7.29% (1.48%) 79091×
78992×
4.73% 32949941×
prfscore 4.73% (4.73%) 32949941×

Figure 10: Time profiling of the top 10 *compute-intensive* kernels in the ClustalW (BioBench) benchmark using gprof tool.

depict the specification of these nodes. It can be noticed that $Node_0$ contains 2 *GPPs* and 2 *RPEs*. The $State_0$ and $State_1$ provide information that both *RPEs* are currently *available* and *idle*. Moreover, they are not configured with any processor configuration. Each GPP and RPE is defined by a list of parameters which completely provide its characteristics. Similarly, $Node_1$ contains one *GPP* and 2 *RPEs*, and $Node_2$ consists of only one *RPE*.

We consider a large-scale application in the bioinformatics domain, from a famous benchmark suite, known as BioBench [17]. We choose and analyze ClustalW from the suite, which is a representative multiple-sequence alignment application. For an analysis of ClustalW, we first identified *compute-intensive* methods in the application using gprof [18], which is a GNU software tool for profiling. Figure 10 depicts gprof profiling graph of the top 10 most compute-intensive kernels in the ClustalW application. Secondly, for possible mapping of ClustalW application on an *RPE*, we used a tool a quantitative prediction model for hardware/software partitioning, called Quipu [19]. It is a linear model based on software com-

Table II: Possible node mappings for tasks $Task_0$, $Task_1$, $Task_2$, and $Task_3$.

Task	**Possible mappings**	**User-selected abstraction levels**
$Task_0$	$GPP_0 \leftrightarrow Node_0$, $GPP_1 \leftrightarrow Node_0$, $GPP_0 \leftrightarrow Node_1$.	*Software-only application* **OR** *Predetermined hardware configuration*
$Task_1$	$RPE_0 \leftrightarrow Node_1$, $RPE_1 \leftrightarrow Node_1$, $RPE_0 \leftrightarrow Node_2$.	*User-defined hardware configuration* **OR** *Device-specific hardware*
$Task_2$	$RPE_1 \leftrightarrow Node_1$, $RPE_0 \leftrightarrow Node_2$.	*User-defined hardware configuration* **OR** *Device-specific hardware*
$Task_3$	$RPE_0 \leftrightarrow Node_0$.	*Device-specific hardware*

plexity metrics (SCMs), and can estimate the number of slices, memory units, and look-up tables (LUTs) within reasonable bounds in an early design stage. Furthermore, such a model can make predictions in a relatively short time, as required in a hardware/software partitioning context. We identified that two main functions *pairalign* and *malign* contribute to the 89.76% and 7.79% of the total time consumption of the application. Using Quipu tool, we estimated that *pairalign* requires 30,790 slices, whereas, malign requires 18707 slices on Virtex 5 devices.

Based on the analysis using profiling information, we consider that the ClustalW application can be divided into three generic tasks, as depicted in Figure 10. The execution requirements *(ExecReq)* of each task are depicted in Figures 6a, 6b, and 6c. Similarly, we consider that an application developer implements a device-specific hardware of the whole ClustalW application as one hardware task and submits its *ExecReq* which are represented by Figure 6d. These tasks are submitted to a certain JSS which analyzes the requirements of each task and forwards it to the RMS. The RMS implements a task scheduler that takes decisions to assign each task onto a particular node specified by some task scheduling algorithm. In our case study, we analyze the mapping options for each task (given in Table II) in the following:

$\mathbf{Task_0}$**:** Since the profiling information in Figure 10 shows that this task only distributes data to the *malign* and *pairalign* functions, so it can be considered as a task requiring a GPP only. It can be noticed that any of the GPP_0 and GPP_1 in the $Node_0$ and GPP_0 in the $Node_1$ contain the minimum processing requirements by the $Task_0$. Consequently, the scheduler can assign it to any of $Node_0$ or $Node_1$. The user provides the application code and required input data. $Task_0$ is a typical example of use-case scenario mentioned in Section III-A.

$\mathbf{Task_1}$**:** requires a Virtex-5 FPGA device with minimum of 18,707 slices. It is evident from Figure 5 that RPE_0 and RPE_1 in $Node_1$ and RPE_0 in $Node_2$ all contain Virtex-5 type devices with more than 24,000 slices, so the $Task_1$ can be assigned to any of $Node_1$ or $Node_2$. The user provides the configuration details (HDL specifications or bitstream), along with application code and input data, to the RMS. $Task_1$ is an example of use-case scenarios given in the Sections III-B2 and III-B3.

$\mathbf{Task_2}$**:** is similar to $Task_1$ and requires at least 30,790 Virtex-5 slices. These requirements can only be met by the RPE_1 in the $Node_1$ and RPE_0 in the $Node_2$. Similar to $Task_1$, it also represents use-case scenarios given the Sections III-B2 and III-B3.

$\mathbf{Task_3}$**:** in this scenario, the task requires a particular device-specific hardware (Virtex XC6VLX365T). The application developer designs and implements its design on a particular targeted hardware device and provides its bitstream. The user submits the application along with the specific bitstream and data. In this case, $Task_3$ can be assigned to RPE_0 in the $Node_0$ only.

All possible mapping options and user-selected abstraction levels are given in Table II. The mapping decisions are based on a particular scheduling strategy implemented inside the scheduler in the RMS, that takes into account various parameters, such as area slices, reconfiguration delays, and the time required to send configuration bitstreams, the availability and current status of the nodes. By considering parameters as well as the right scheduling strategy, more performance gain can be achieved by utilizing reconfigurable fabric for particular applications. Moreover, the recent and future reconfigurable devices are expected to support dynamic partial reconfigurability, and different hardware implementations on the same RPE are possible. With the proposed virtualization framework, the resources can be managed in a more efficient manner when the available processing elements are both GPPs and RPEs. Those grid applications which contain more parallelism can get more benefit if executed on the reconfigurable hardware. For the purpose of testing task scheduling strategies and resource management for dynamic reconfigurable processing nodes in a distributed environment, we have developed a simulation framework, termed as **D**ynamic **Re**configurable **A**utonomous **M**any-task **Sim**ulator (DReAMSim) [20]. Moreover, we incorporated partial reconfigurable functionality to the nodes in the DReAMSim, in [21]. The design offers to model complex reconfigurable computing nodes, processor configurations, and tasks along with GPPs. A number of different reconfiguration parameters can be exploited to model the processor configurations. The DReAMSim can be used to investigate the desired system scenario(s) for a particular scheduling strategy and a given number of tasks, grid nodes, configurations, task arrival distributions, area ranges, and task required times etc.

VI. Conclusions and Future Work

This paper presented a virtualization framework for RPEs in distributed grid systems. We presented various scenarios in terms of use-cases to discuss the utilization of RPEs in grids. Based on different virtualization levels, we provided a general model for a computing node which incorporates both GPPs and RPEs in grids. We also presented a typical application task model. Finally, we presented a case study of a scientific application which can take benefit from our virtualization framework. The proposed framework is generic in nature and provides *backward compatibility* for existing software applications in grid network. Moreover, it is adaptive in dynamically adding or removing resources. Independent of the virtualized hardware, the framework can be extended different types of *services* to the user, such as monitoring and status updates etc. Currently, the framework does not support streaming applications. In our future work, we will propose a virtualization scenario for streaming applications. We will discuss a more case studies based on our virtualization approach.

References

[1] I. Foster and C. Kesselman, "Computational Grids," in *Selected Papers and Invited Talks from the 4th International Conference on Vector and Parallel Processing (VECPAR)*, pp. 3–37, 2001.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid - Enabling Scalable Virtual Organizations," *International Journal of Supercomputer Applications*, vol. 15, pp. 1–24, 2001.

[3] D. P. Anderson, "Boinc: A System for Public Resource Computing and Storage," in *5th IEEE/ACM International Workshop on Grid Computing*, pp. 4–10, 2004.

[4] TeraGrid, "FPGA Resources in Purdue University." http://www.rcac.purdue.edu/teragrid/userinfo/fpga.

[5] K. H. Tsoi and W. Luk, "Axel: a heterogeneous cluster with FPGAs and GPUs," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays (FPGA)*, pp. 115–124, 2010.

[6] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," *ACM Computer Survey*, vol. 34, no. 2, pp. 171–210, 2002.

[7] C. Plessl and M. Platzner, "Virtualization of Hardware - Introduction and Survey," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, pp. 63–69, 2004.

[8] C. H. Huang and P. A. Hsiung, "Hardware Resource Virtualization for Dynamically Partially Reconfigurable Systems," *IEEE Embedded Systems Letters*, vol. 1, no. 1, pp. 19–23, 2009.

[9] Yajun Ha, et. al, "A Hardware Virtual Machine for the Networked Reconfiguration," in *Proceedings of 11th IEEE International Workshop on Rapid System Prototyping(RSP)*, pp. 194–197, 2000.

[10] M. Sabeghi and K.L.M. Bertels, "Toward a Runtime System for Reconfigurable Computers: A Virtualization Approach," in *Design, Automation and Test in Europe (DATE)*, pp. 1576–1579, 2009.

[11] W. Fornaciari and V. Piuri, "General Methodologies to Virtualize FPGAs in Hw/Sw Systems," in *Proceedings of the 1998 Midwest Symposium on Systems and Circuits*, pp. 90–93, 1998.

[12] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Virtualizing and Sharing Reconfigurable Resources in High-Performance Reconfigurable Computing Systems," in *2nd International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA)*, pp. 1–8, 2008.

[13] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, no. 3, pp. 171–200, 2005.

[14] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor - A Distributed Job Scheduler," *Beowulf cluster computing with Linux*, pp. 307–350, 2002.

[15] S. Wong and F. Anjam, "The Delft Reconfigurable VLIW Processor," in *17th International Conference on Advanced Computing and Communications (ADCOM)*, pp. 244–251, 2009.

[16] Opencores, "The OpenCores Project Online Website." http://opencores.org/projects.

[17] K. Albayraktaroglu, et. al, "BioBench: A Benchmark Suite of Bioinformatics Applications," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 2–9, 2005.

[18] S. L. Graham et. al, "Gprof: A Call Graph Execution Profiler," *SIGPLAN Not.*, vol. 17, pp. 120–126, 1982.

[19] R. Meeuws, Y. Yankova, K. Bertels, G. Gaydadjiev, and S. Vassiliadis, "A Quantitative Prediction Model for Hardware/Software Partitioning," in *International Conference on Field Programmable Logic and Applications(FPL)*, pp. 735 –739, 2007.

[20] M. F. Nadeem et. al, "A simulation framework for reconfigurable processors in large-scale distributed systems," in *Proceedings of the 2011 40th International Conference on Parallel Processing Workshops (ICPPW)*, pp. 352–360, 2011.

[21] M. F. Nadeem et. al, "Task Scheduling in Large-scale Distributed Systems Utilizing Partial Reconfigurable Processing Elements," in *Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2012.