Virtualizing and Sharing Reconfigurable Resources in High-Performance Reconfigurable Computing Systems

Esam El-Araby, Ivan Gonzalez, and Tarek El-Ghazawi

NSF Center for High-Performance Reconfigurable Computing (CHREC), ECE Department, The George Washington University
801 22nd Street NW, Washington, DC 20052, USA
{esam, ivangm, tarek}@gwu.edu

Abstract—High-Performance Reconfigurable Computers (HPRCs) are parallel computers but with added FPGA chips. Examples of such systems are the Cray XT5 and Cray XD1, the SRC-7 and SRC-6, and the SGI Altix/RASC. The execution of parallel applications on HPRCs mainly follows the Single-Program Multiple-Data (SPMD) model, which is largely the case in traditional High-Performance Computers (HPCs). In addition, the prevailing usage of FPGAs in such systems has been as co-processors. The overall system resources, however, are often underutilized because of the asymmetric distribution of the reconfigurable processors relative to the conventional processors. This asymmetry is often a challenge for using the SPMD programming model on these systems. In this work, we propose a resource virtualization solution based on Partial Run-Time Reconfiguration (PRTR). This technique will allow sharing the reconfigurable processors among the underutilized processors. We will present our virtualization infrastructure augmented with an analytical investigation. We will verify our proposed concepts with experimental implementations using the Cray XD1 as a testbed. It will be shown that this approach is quite promising and will allow full exploitation of the system resources with fair sharing of the reconfigurable processors among the microprocessors. Our approach is general and can be applied to any of the available HPRC systems.

Index Terms—High Performance Computing, Field Programmable Gate Arrays (FPGA), Reconfigurable Computing, Dynamic Partial Reconfiguration

I. INTRODUCTION

Reconfigurable Computers (RCs) have recently evolved from accelerator boards to stand-alone general purpose RCs and parallel (multi-node) reconfigurable supercomputers called High-Performance Reconfigurable Computers (HPRCs). Examples of such supercomputers are the SRC-7 and SRC-6 [1], the SGI Altix/RASC [2] and the Cray XT5n and Cray XD1 [3]. In these systems, FPGAs are used to implement coprocessors to accelerate in hardware the critical functions causing the poor performance of the general purpose processors, following HW/SW codesign approaches. Several efforts have proved the significant speedups obtained by these systems for many different applications [4 - 10].

The development of applications on HPRCs mainly follows the same programming model for HPC platforms, i.e. the Single-Program Multiple-Data (SPMD), which is the most common style of parallel programming [11]. In SPMD [12], multiple autonomous processors simultaneously execute the same program at independent points. In other words, tasks can be deployed and executed in parallel [12, 13] using either shared memory and/or message passing techniques such as MPI.

However, because current HPRC technology utilizes the reconfigurable processors as coprocessors to the main host processor, heterogeneity can be challenging to most accepted SPMD programming paradigms. In particular, when the ratio of microprocessors, reconfigurable processors, and their communication channels differs from unity, SPMD programs, which generally assume a unity ratio, might underutilize some of the system processing resources, e.g. microprocessors [11].

In this work, we propose to share the reconfigurable resources among the underutilized microprocessors by providing a virtual SPMD view and thus improving the overall system versatility. In other words, the pool of reconfigurable resources will be virtually increased to maintain the required symmetric view of SPMD, i.e. unity ratio among the microprocessors, reconfigurable processors, and their communication channels. The implementation of these concepts will be based on Partial Run-Time Reconfiguration (PRTR) from a practical perspective. We will provide a formal analysis of the execution model supported by
experimental work. Our work utilizes PRTR on one of the current HPRC systems, Cray XD1.

This paper is organized such that section II provides a discussion of related work in context of run-time reconfiguration and hardware virtualization. Section III describes our analytical model and explains the formulation steps of this model. Section IV shows the experimental work and presents the implementation of a partially reconfigurable architecture in Cray XD1. Section IV also includes the implementation of an Operating System (OS) virtualization infrastructure for sharing reconfigurable resources. Finally, section V summarizes and concludes the paper.

II. RELATED WORK

The objective of this work is to share the reconfigurable resources in HPRCs among all system microprocessors in a SPMD view regardless of the system physical limitations/configuration. In other words, we will try to maintain a virtual 1:1 correspondence among the microprocessors and the reconfigurable resources irrespective of the actual ratio in the system. In achieving this objective, our approach is based on leveraging previous work and concepts that were introduced for solving similar and related problems, namely hardware virtualization. For example, we will adopt the concept of virtual FPGA (VFPGA) as proposed in [14]. In addition, we will maintain ideas and considerations related to hardware virtualization on generic HPC architectures [15] and leverage them to HPRCs.

Most of the proposed solutions in many previous research work [17, 18] are to reproduce the same strategies adopted in Operating Systems to support virtual memory such as dynamic loading, partitioning, overlaying, segmentation, and paging, etc. The basic idea behind these techniques is to virtually enlarge the size of the FPGA from the point of view of the applications. Therefore, the concept of “virtual hardware” is an effective and efficient technique to increase the availability of hardware resources, implement larger circuits or reduce the costs by adopting smaller FPGA when the performance can still be satisfied. The possibility to apply this concept requires using special capabilities of the FPGAs namely Full Run-Time Reconfiguration (FRTR) and/or Partial Run-Time Reconfiguration (PRTR) [16]. However, all these proposed techniques assume that the applications and related hardware functions (tasks) that need to be executed on dedicated reconfigurable resources. These applications require on the average a few hardware resources in HPRCs among all system microprocessors in a SPMD view in which the system receives some applications as input. These applications require on the average a few hardware functions (tasks) that need to be executed on dedicated reconfigurable resources. The physical reconfigurable resources (FPGAs) will be virtualized and split into multiple virtual FPGAs (VFPGAs) to accommodate the requirements of the SPMD model, see Fig. 1.

Each VFPGA will be located in a separate partially reconfigured region (PRR) on the physical FPGA. The application tasks can then be distributed across the VFPGAs maintaining a 1:1 correspondence among the tasks and their dedicated resources (VFPGAs) and hence providing a SPMD view to the application. The required tasks by applications, $N_{tasks}$, are assumed to be equal to or less than the maximum number of VFPGAs/PRRs, $N_{regions}$. This condition, i.e. $N_{tasks} \leq N_{regions}$, is necessary for providing SPMD behavior. In other words, the maximum number of VFPGAs/PRRs should not exceed the number of microprocessors per node, see Fig. 1.
The execution cycle for any task on an HPRC consists of the computations time, the total I/O time and the configuration 
time [16, 19, 20], as shown in Fig. 2. The I/O time is the time 
necessary to transfer data between the microprocessor and the 
FPGA.

![Fig. 2. Task execution time on an HPRC](image)

The baseline for our analysis is FRTR. In other words, we 
will consider the performance gain (speedup) of the system 
when using our methodology based on PRTR compared to the 
performance using conventional techniques based on FRTR. 
This will focus our discussions on applications that are broken 
down into hardware tasks only. Software tasks are excluded 
from our analysis because, we think, that would add 
unnecessary complications to model the partitioning schemes 
as well as the profiles of scheduling among software and 
hardware tasks. In addition, we assume that each task is fully 
characterized by its time requirement, $T_{\text{task}}$, as shown in Fig. 2. 
The I/O and computations of each task can be overlapped to 
further enhance the overall execution time as proposed in [19, 
20].

The following notation will be used in our mathematical 
model:
- $N_{\text{regions}}$ is the maximum number of VFPGAs that can 
  be provided based on the available microprocessors
- $N_{\text{tasks}}$ is the total number of hardware tasks
- $T_{\text{in}}$ is the average input transfer time from any 
  microprocessor to its dedicated VFGPA
- $T_{\text{comp}}$ is the average task computation time

- $T_{\text{out}}$ is the average output transfer time from any 
  VFGPA to its associated microprocessor
- $T_{\text{config}} = T_{\text{FRTR}}$ is the full configuration time for FRTR
- $T_{\text{PRTR}}$ is the average partial configuration time for 
  PRTR
- $T_{\text{FRTR total}}$ is the total execution time of FRTR
- $T_{\text{PRTR total}}$ is the total execution time of PRTR
- $S$ is the speedup or performance gain of using PRTR relative to FRTR

The execution model of FRTR on each node, see Fig. 3, is 
sequential among tasks. This is because the reconfigurable 
resource, assuming one per node, is not sharable among the 
node microprocessors rendering some microprocessors 
unused. The total execution time for the case of FRTR, as 
shown in Fig. 3, can be derived as follows:

$$T_{\text{FRTR total}} = N_{\text{tasks}}(T_{\text{FRTR in}} + T_{\text{comp}} + T_{\text{out}})$$

(1)

The execution model of our proposed virtualization 
technique and sharing mechanism can be viewed as a 
combination of three traffic (queueing) processes, namely 
entry/birth, computation, and exit/death processes. The 
entry/birth process is when tasks at the beginning of their 
execution life-cycle request configuration and data transfer 
from the microprocessors into the VFGPAs. The exit/death 
process is when tasks at the end of their execution life-cycle 
request data transfer from the VFGPA back to the 
microprocessors. The computation process represents the 
actual processing performed by tasks on their VFGPAs. In our 
model tasks can continue their computations in parallel while 
others are entering into and/or exiting from the system. 
Several different traffic scenarios occur depending on the 
relative speed rates among the three different processes. Fig. 4 
shows the different execution profiles of tasks when sharing 
the reconfigurable resources.

![Fig. 3. Typical task execution using FRTR on HPRC](image)

(4a) Faster entry than computation, with slower exit than entry
Fig. 4. Execution profile for sharing virtual reconfigurable resources

(4b) Faster entry than computation, with faster exit than entry

(4c) Slower entry than computation, with slower exit than computation

(4d) Slower entry than computation, with faster exit than computation
Combining the necessary condition for SPMD behavior, i.e. \( N_{tasks} \leq N_{regions} \), with the entry and exit conditions as shown in Fig. 4, we can derive the following expression for the total execution time:

\[
SPMD\text{condition: } N_{tasks} \leq N_{regions},
\]

\[
T_{entry} = \left( T_{PRTR} + T_{in} \right) + \max \left( N_{tasks} - 1 \right) \left( T_{PRTR} + T_{in} + T_{comp} \right),
\]

\[
T_{exit} = \max \left( N_{tasks} - 1 \right) T_{out} + \min \left( N_{tasks} - 1 \right) \left( T_{PRTR} + T_{in} + T_{comp} \right) + T_{out},
\]

\[
T_{total}^{PRTR} = T_{entry} + T_{exit}
\]

(2)

The performance gain (speedup) of PRTR in reference to FRTR can be expressed as follows by combining equations (1) and (2):

\[
S = \text{Speedup} = \frac{T_{total}^{FRTR}}{T_{total}^{PRTR}}
\]

\[
\Rightarrow S = \frac{N_{tasks} \left( T_{PRTR} + T_{in} + T_{comp} + T_{out} \right)}{\max \left( N_{tasks} - 1 \right) \left( T_{PRTR} + T_{in} + T_{comp} \right) + \min \left( N_{tasks} - 1 \right) \left( T_{PRTR} + T_{in} + T_{comp} \right) + T_{out}}
\]

(3)

In order to estimate the upper bound for the performance gain (speedup) using our techniques, we take the limit of equation (3) as the number of tasks increases indefinitely, i.e. \( N_{tasks} \rightarrow \infty \). This will help us estimate the asymptotic behavior with respect to FRTR as follows:

\[
S_\infty = \lim_{N_{tasks} \rightarrow \infty} S
\]

\[
\Rightarrow S_\infty = \frac{T_{PRTR} + T_{in} + T_{comp} + T_{out}}{T_{PRTR} + T_{in} + T_{comp} + T_{out}}
\]

(4)

where \( T_{I/O} = T_{in} + T_{out} \)

It can be seen from equation (4) that the asymptotic performance gain increases linearly with the task computation requirement, i.e. \( T_{comp} \). This is due to the fact that our proposed technique overlaps the computation of tasks with other tasks entry and/or exit which significantly reduces the total execution time. It can also be seen from equation (4) that for I/O intensive applications characterized by minimal computational workloads, i.e. \( T_{comp} \approx 0 \), there is always a performance gain, i.e. \( S_\infty \geq 1 \). This is due to the fact that our techniques utilize PRTR rather than FRTR.

IV. EXPERIMENTAL WORK

Our experiments have been performed on one of the current HPRC systems, Cray XD1 [3]. The Cray XD1 is a multi-chassis system. Each chassis contains up to six nodes (blades). Each blade consists of two 64-bit AMD Opteron processors at 2.4 GHz, one Rapid Array Processor (RAP) that handles the communication, an optional second RAP, and an optional Application Accelerator Processor (AAP). The AAP is a Xilinx Virtex-II Pro XC2VP50-7 FPGA with a local memory of 16MB QDR-II SRAM [3].

A. Virtualization Infrastructure

In order to implement our concepts, we started by laying out an infrastructure based on two main techniques. The first technique, as proposed and explained in details in [16], is enabling the support for PRTR on HPRC systems. Because PRTR is not natively supported on Cray XD1, our workaround approach was to use the Internal Configuration Access Port (ICAP) and develop a new configuration API. Additionally, we define an FPGA layout that supports single and dual Partially Reconfigurable Regions (PRRs) in addition to the static region. In dual PRRs, each region has access to two memory banks, one for input and the other for output transfers as shown in Fig. 5. Finally, it is worth mentioning that the interface services block, i.e. RT core provided by Cray, and the reconfiguration control unit are included in the static region.

After establishing/enabling the low-level physical layer of partitioning/splitting the reconfigurable resources into multiple virtual reconfigurable resources through PRTR, we continue to provide a general infrastructure that manages these resources. This infrastructure, which is the main concern of this effort, is implemented as two layers. The first layer is an Operating System (OS) Run-Time Services layer on top of which lies the second layer which is an API layer.

As shown in Fig. 5, the Run-Time virtualization layer consists of three major components: a virtualization manager (VM), a request queue, and a virtual memory space. The VM is responsible for partitioning the physical resource, i.e.
FPGA, into multiple virtual resources, i.e. VFPGAs, and providing a coherent access to these resources as being physical in a balanced SPMD view. It also manages all interactions among applications and their required resources by handling the traffic of application requests for reconfigurable resources, memory, and/or I/O channels, as previously explained in section III and shown in Fig. 4. The queue helps in streaming the requests from the μPs to the VM as well as in providing synchronization, e.g. hand shaking, mechanisms. The queue has priorities to avoid contention problems, e.g. one task taking all the resources, and to ensure that all parallel tasks are executed as fast as possible. Finally, the virtual space allows a coherent access of the virtual resources to the user. This space is implemented as a Run-Time OS shared memory such that there is one memory space/region per VPGA. This space is used to exchange data between tasks and the VM, i.e. (re)configuration bitstreams, input data, and output data.

The API layer abstracts the interactions between the user and the Run-Time System in a transparent way. The APIs are designed in a way such that they cover all possible task profiles that were described earlier in Figs. 2, 3, and 4. More specifically, the APIs are categorized as Setup APIs, Configuration APIs, Transfer of Control APIs, and Execution APIs (where execution includes data transfer among μPs and VFPGAs).

B. Experimental Results

A set of experiments were conducted in order to verify the proposed techniques of our infrastructure. For our experiments we selected the application of image feature extraction. In this particular application object edges were of interest and were extracted after first reducing high-frequency noise components. Two different algorithms were used for noise reduction. The final images were transferred back to the microprocessor for quality checks. More specifically, this application required the execution of a sequence of image processing functions, namely median filtering followed by sobel edge detection as well as smoothing filtering also followed by sobel edge detection. From those experiments we extracted the needed parameters, see Table I, for our model explained in section III. Table I shows data transfer times, configuration times as well as the bitstream size associated with the layout configuration that we considered.

<table>
<thead>
<tr>
<th></th>
<th>Data Size (Bytes)</th>
<th>Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Configuration</td>
<td>2381764</td>
<td>1678.04</td>
</tr>
<tr>
<td>Dual PRR</td>
<td>404168</td>
<td>19.77</td>
</tr>
<tr>
<td>Input Transfer</td>
<td>4194304</td>
<td>2.991</td>
</tr>
<tr>
<td>Output Transfer</td>
<td>4194304</td>
<td>641.092</td>
</tr>
</tbody>
</table>

It is worth mentioning that the SPMD condition, i.e. $N_{tasks} \leq N_{regions}$, on Cray XD1 suggests that the maximum number of PRRs should not exceed the number of microprocessors per node which is two in this case. Therefore, we conducted the experiments on Cray XD1 using dual VFPGAs scenario. However, for the sake of completeness we developed an emulator that uses XD1 in as close to real setups as possible to emulate scenarios for larger number of VFPGAs (PRRs). Although the emulator accepts a minimum set of parameters for XD1 since it is running on the machine itself, it however can emulate any platform given its parameters. These parameters include full configuration time, partial reconfiguration time (calculated based on the size of bitstreams), I/O transfer bandwidth, and different computation time to emulate different tasks, etc.

(Data Size = 4MB, Tln = 2.991 ms, Tcomp = 0.299 ms, Tout = 641.092 ms)

(6a) Behavior of I/O intensive applications
Results collected were compared to the actual runs on XD1 as well as to the expected by the mathematical model presented in section III and were found in good agreement. Fig. 6 shows some of these experimental findings for different types of applications as well as for large number of VFPGAs versus conventional execution based on FRTR. Fig. 6(a) and 6(b) show the efficiency of our virtualization layer shown in Fig. 5. It introduces a minimal overhead to the total execution time as the number of PRRs increases. The accuracy of our analytical model can be seen by applying equation (4) to the case of I/O intensive applications, i.e. $T_{\text{comp}} \approx 0$, and comparing the experimental results shown in Fig. 6(c). The parameters collected from our experiments as shown in Table I are $T_{\text{FRTR}} = 1678.040$ ms, $T_{\text{PRTR}} = 19.771$ ms, $T_{\text{in}} = 2.991$ ms, and $T_{\text{out}} = 641.092$ ms. Equation (4) suggests that the speedup value should be 3.49, which is consistent with the value measured and shown in Fig. 6(c). Fig. 6(c) also proves the potential of virtualizing reconfigurable resources using our technique based on PRTR. It can be seen from Fig. 6(c), as also expected by equation (4) and discussed in section III, that the performance reaches a linear increase as the number of VFPGAs increases.
V. CONCLUSIONS

In this paper we presented an effort of virtualizing and sharing reconfigurable resources based on Partial Run-Time Reconfiguration (PRTR) for High-Performance Reconfigurable Computing (HPRC). We investigated the performance potential of our proposed virtualization techniques on HPRCs from both theoretical and practical perspectives. In doing so, we derived a formal and an analytical model of SPMD execution on HPRC systems relative to the baseline of Full Run-Time Reconfiguration (FRTR). The model provided us with theoretical expectations which served as a frame of reference against which we projected our experimental results. In addition, it helped us gain in-depth insight about the boundaries and/or conditions for possibilities of performance gain using PRTR for resource sharing and virtualization. In achieving this objective, our approach was based on leveraging previous work and concepts that were introduced for solving similar and related problems.

In conducting the experimental work, we utilized one of the current HPRC systems, Cray XD1. We also discussed the requirements and setups for PRTR-based resource virtualization on Cray XD1. Our setup included the design of a special configuration control unit managing the configuration of different layouts of Partially Reconfigured Regions (PRRs). In addition, we implemented an OS virtualization layer that manages the shared resources and the virtualization process. The experimental results showed good agreement with the analytical model expectations. Sharing reconfigurable resources among the underutilized microprocessors by providing a virtual SPMD view allows improving the overall system versatility and application performance. The approach we followed for Cray XD1 is general and can be applied to any of the available HPRC systems.

REFERENCES