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$_h$ 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 XT5$_h$ 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

This work was supported in part by the I/UCRC Program of the National Science Foundation under the NSF Center for High-Performance Reconfigurable Computing (CHREC).

HPRCTA'08, November 17, 2008, Austin, Texas, USA. Copyright 2008. 978-1-4244-2826-7/08/$25.00 ©2008 IEEE.

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 are known previously and FRTR and/or PRTR are well supported on the system. Currently, this is true for FRTR while it is not the case for PRTR. Also, they do not take into consideration the architectural limitation of using partial reconfiguration on current HPRCs. To the end user, HPRC systems when compared to embedded systems are "closed black box" systems. Users do not have the possibility to modify the system nor have access to the FPGA configuration ports. They can only use the API functions provided by the vendor. With this regard, most of previous work is based on simulations rather than investigating such practical issues. Therefore, we approach the problem from a practical perspective by utilizing and building on the techniques and methodologies introduced in [16] by providing a virtualization infrastructure consisting of an OS Run-Time layer augmented with another layer of user APIs. The API layer abstracts the interactions between the user and the Run-Time layer in a transparent way. Furthermore, we extend the execution model to include this virtualization infrastructure for sharing the reconfigurable processors, and their communication channels.

III. EXECUTION MODEL FORMULATION

In order to investigate the performance potential of our techniques on HPRCs before conducting our experimental work, we will derive a formal analysis of the execution model. This analysis would provide us with theoretical expectations which would serve as a frame of reference against which we can project our experimental results. In addition, it will help us gain in-depth insight about the boundaries and/or conditions for performance gain. In achieving this objective, we will follow an approach in the derivation of the model similar to what has been proposed in [16, 19, 20, 21, 22, 23].

A. Analysis

In our analysis we assume an HPRC architecture with asymmetric heterogeneity at the node level [11] with 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.

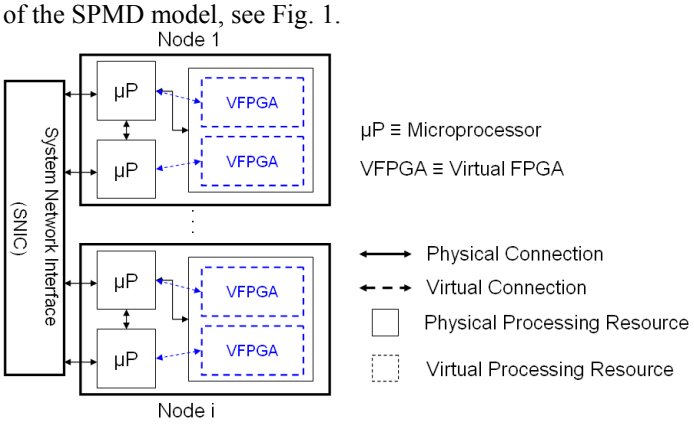


Fig. 1. SPMD view of reconfigurable resources on HPRCs

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.

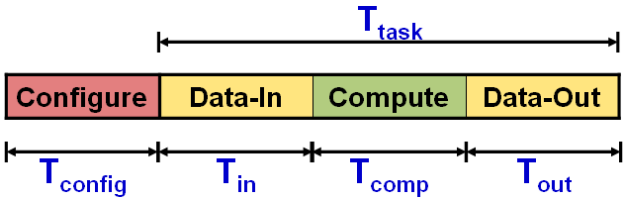


Fig. 2. Task execution time on an HPRC

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_{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:

- $N_{regions}$ is the maximum number of VFPGAs that can be provided based on the available microprocessors
- N_{tasks} is the total number of hardware tasks
- T_{in} is the average input transfer time from any microprocessor to its dedicated VFPGA
- T_{comp} is the average task computation time
- T_{out} is the average output transfer time from any VFPGA to its associated microprocessor
- $T_{config} = T_{FRTR}$ is the full configuration time for FRTR
- T_{PRTR} is the average partial configuration time for PRTR
- T^{FRTR}_{total} is the total execution time of FRTR
- T^{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:

$$T^{FRTR}_{total} = N_{tasks}\left(T_{FRTR} + T_{in} + T_{comp} + T_{out}\right) \quad (1)$$

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 VFPGAs. The exit/death process is when tasks at the end of their execution life-cycle request data transfer from the VFPGA back to the microprocessors. The computation process represents the actual processing performed by tasks on their VFPGAs. 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.

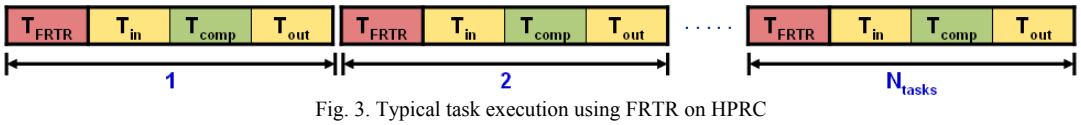
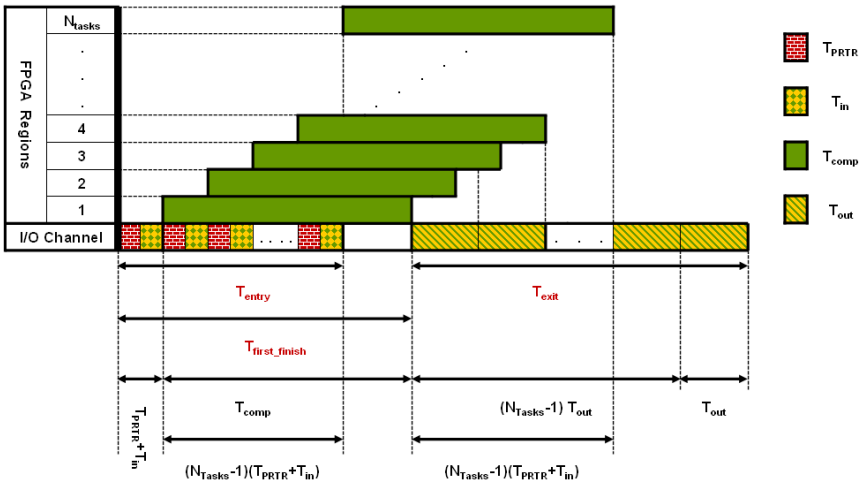
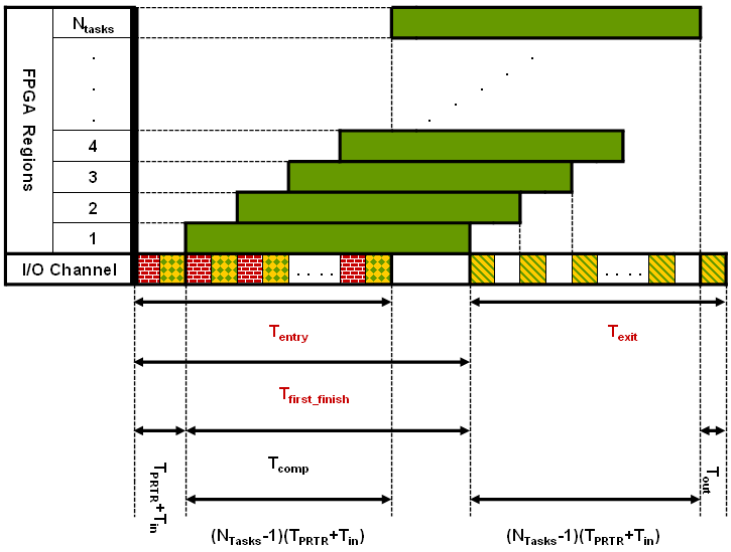


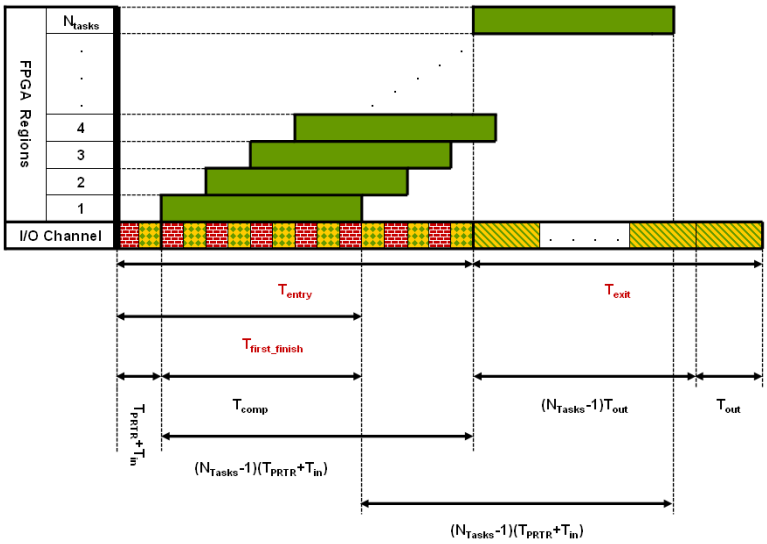
Fig. 3. Typical task execution using FRTR on HPRC



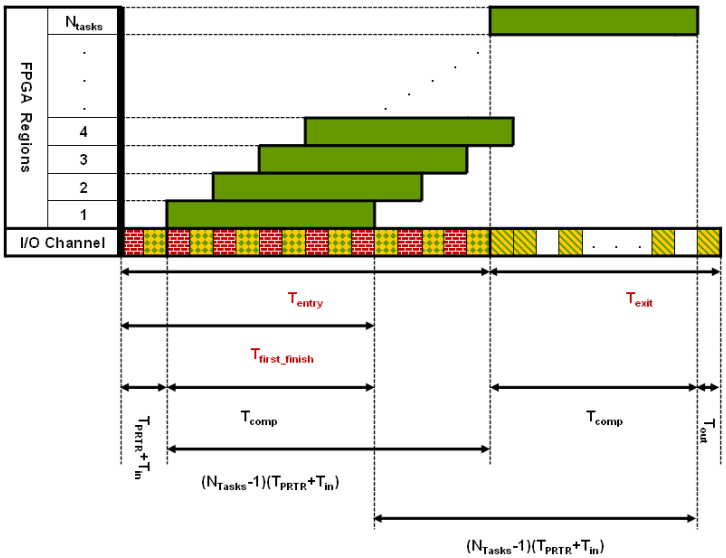
(4a) Faster entry than computation, with slower exit than entry



(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

Fig. 4. Execution profile for sharing virtual reconfigurable resources

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:

$$SPMDcondition: N_{tasks} \leq N_{regions}$$
$$T_{entry} = (T_{PRTR} + T_{in}) + MAX[(N_{tasks} - 1)(T_{PRTR} + T_{in}), T_{comp}]$$
$$T_{exit} = MAX[(N_{tasks} - 1)T_{out}, MIN\{(N_{tasks} - 1)(T_{PRTR} + T_{in}), T_{comp}\}] + T_{out}$$
$$T_{total}^{PRTR} = T_{entry} + T_{exit} \quad (2)$$

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

$$S \equiv Speedup = \frac{T_{total}^{FRTR}}{T_{total}^{PRTR}}$$
$$\Rightarrow S = \frac{N_{tasks}(T_{FRTR} + T_{in} + T_{comp} + T_{out})}{\left\{\begin{array}{l}(T_{PRTR} + T_{in}) + T_{out} + MAX[(N_{tasks} - 1)(T_{PRTR} + T_{in}), T_{comp}] + \\ MAX[(N_{tasks} - 1)T_{out}, MIN\{(N_{tasks} - 1)(T_{PRTR} + T_{in}), T_{comp}\}]\end{array}\right\}} \quad (3)$$

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. $(N_{tasks}=N_{regions}) \rightarrow \infty$. This will help us estimate the asymptotic behavior with respect to FRTR as follows:

$$S_{\infty} \equiv \lim_{N_{tasks} \to \infty} S$$
$$\Rightarrow S_{\infty} = \frac{T_{FRTR} + T_{in} + T_{comp} + T_{out}}{T_{PRTR} + T_{in} + T_{out}} = \frac{T_{FRTR} + T_{I/O} + T_{comp}}{T_{PRTR} + T_{I/O}} \quad (4)$$
$$where\ T_{I/O} = T_{in} + T_{out}$$

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. $T_{comp} \cong 0$, 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 work-around 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.

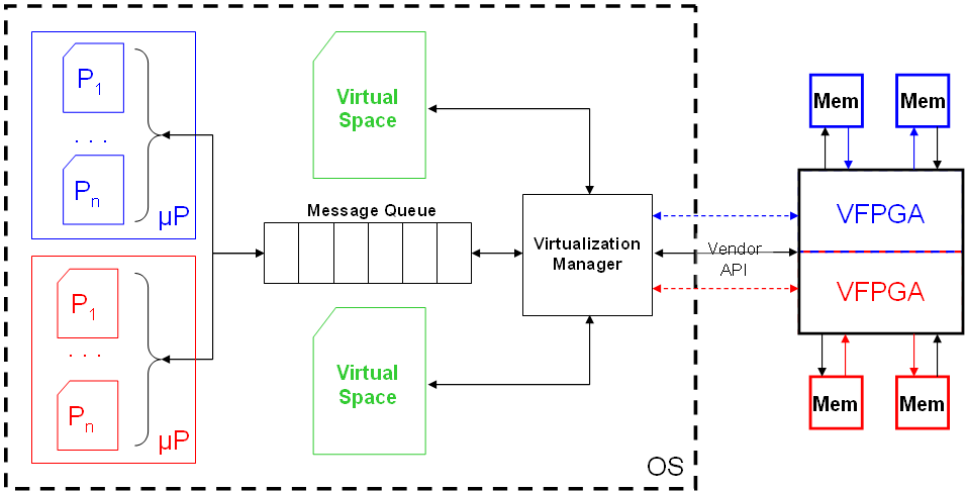


Fig. 5. Run-Time virtualization layer using PRTR

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).

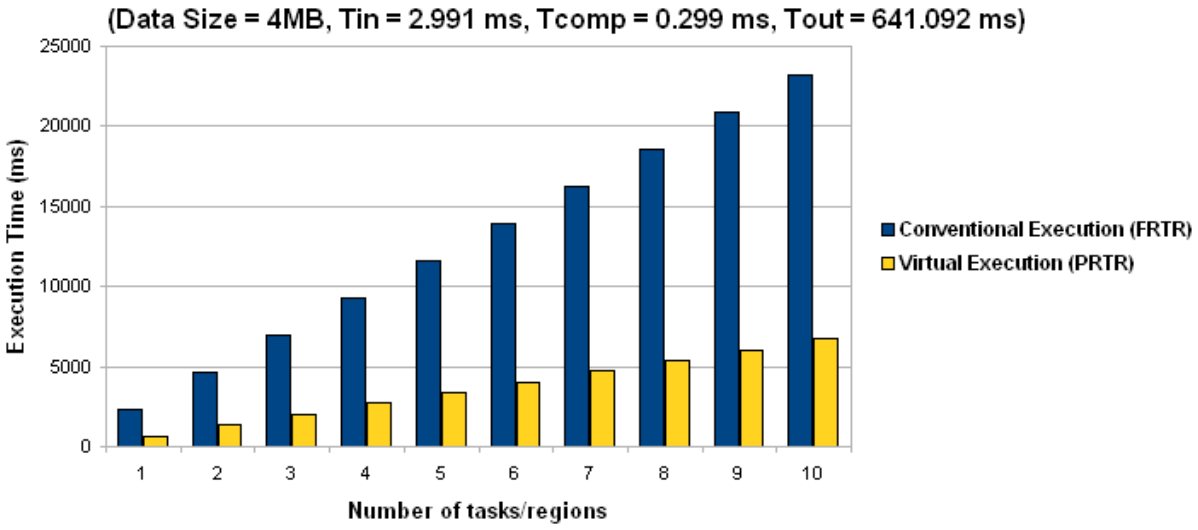
B. Experimental Results

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.

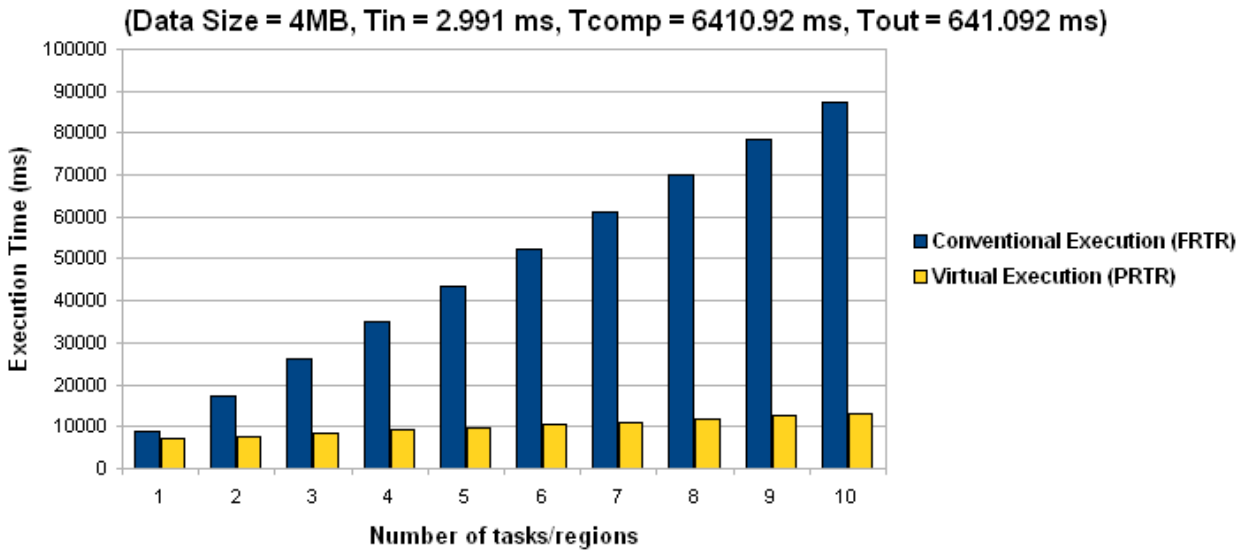
TABLE I EXPERIMENTAL VALUES FOR MODEL PARAMETERS

	Data Size (Bytes)	Time (msec)
Full Configuration	2381764	1678.04
Dual PRR	404168	19.77
Input Transfer	4194304	2.991
Output Transfer	4194304	641.092

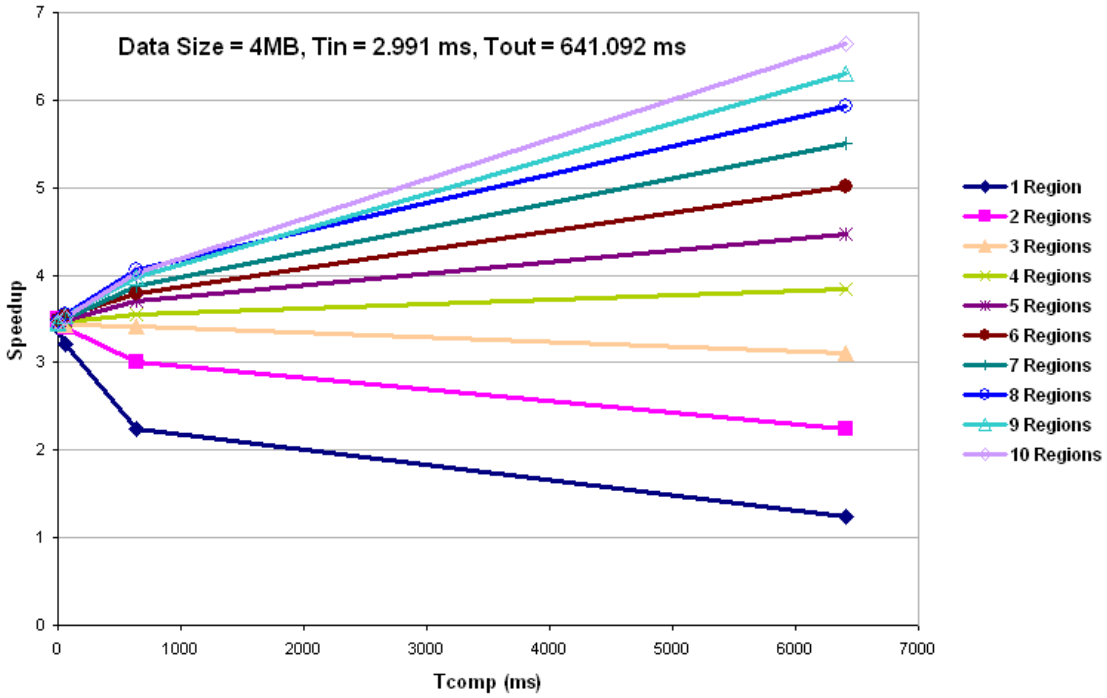
It is worth mentioning that the SPMD condition, i.e. $N_{tasks} \leq N_{regions}$, on Cray XD1 suggests that the maximum number of PRRs should not exceed the number of microprocessors per node which is two in this case. Therefore, we conducted the experiments on Cray XD1 using dual VFPGAs scenario. However, 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 bitsreams), I/O transfer bandwidth, and different computation time to emulate different tasks, etc.



(6a) Behavior of I/O intensive applications



(6b) Behavior of computational intensive applications



(6c) Speedup achieved using multiple PRRs (VFPGAs)

Fig. 6. Performance of applications using virtual resources

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. $T_{comp} \cong 0$, and comparing the experimental results shown in Fig. 6(c). The parameters collected from our experiments as shown in Table I are T_{FRTR}= 1678.040 ms, T_{PRTR}= 19.771 ms, T_{in}=2.991 ms, and T_{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

[1] SRC Computers, Inc., "SRC Carte™ C Programming Environment v2.2 Guide (SRC-007-18)", August 2006.

[2] Silicon Graphics Inc., "Reconfigurable Application-Specific Computing User's Guide (007-4718-005)", January 2007.

[3] Cray Inc., "Cray XD1™ FPGA Development (S-6400-14)", 2006.

[4] T. V. Court, and M. C. Herbordt, "Families of FPGA-Based Accelerators for Approximate String Matching", ACM Microprocessors & Microsystems, v. 31, Issue 2, March 2007, pp. 135-145.

[5] V. Kindratenko, and D. Pointer, "A case study in porting a production scientific supercomputing application to a reconfigurable computer", in Proc. IEEE Symposium on Field-Programmable Custom Computing Machines - FCCM'06, 2006. pp. 13-22.

[6] V. Aggarwal, A. D. George, K. C. Slatton, "Reconfigurable Computing with Multiscale Data Fusion for Remote Sensing", Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays (FPGA 2006), Monterey, California, USA.

[7] D. A. Buell, J. P. Davis, G. Quan, S. Akella, S. Devarkal, P. Kancharla, E. A. Michalski, and H. A. Wake, "Experiences with a reconfigurable computer," Proceedings, Engineering of Reconfigurable Systems and Algorithms, Las Vegas, Nevada, 21-24 June 2004.

[8] D. A. Buell and R. Sandhu, "Identity management," IEEE Internet Computing, v. 7, no. 6, November/December 2003, pp. 26-28 (guest editors' introduction).

[9] A. Michalski, K. Gaj, T. El-Ghazawi, "An Implementation Comparison of an IDEA Encryption Cryptosystem on Two General-Purpose Reconfigurable Computers", Proc. FPL 2003, Lisbon, Sept. 2003, pp. 204-219.

[10] O. O. Storaasli, "Scientific Applications on a NASA Reconfigurable Hypercomputer", 5th MAPLD International Conference, Washington, DC, USA, September, 2002.

[11] Tarek El-Ghazawi, Esam El-Araby, Miaoqing Huang, Kris Gaj, Volodymyr Kindratenko, and Duncan Buell, "The Promise of High-Performance Reconfigurable Computing," IEEE Computer, vol. 41, no. 2, pp. 69-76, February 2008.

[12] Algorithms and Theory of Computation Handbook, CRC Press LLC, 1999, "Single Program Multiple Data", in Dictionary of Algorithms and Data Structures, Paul E. Black, ed., U.S. National Institute of Standards and Technology. Available from: http://www.nist.gov/dads/HTML/singleprogrm.html

[13] F. Darema, SPMD model: past, present and future, Recent Advances in Parallel Virtual Machine and Message Passing Interface: 8th European PVM/MPI Users' Group Meeting, Santorini/Thera, Greece, September 23-26, 2001. Lecture Notes in Computer Science 2131, p. 1, 2001.

[14] Fornaciari, W., and Piuri, V., "General methodologies to virtualize FPGAs in Hw/Sw systems", Proc. of Midwest Symposium on Circuits and Systems, pp. 90-93, 1998.

[15] Ada Gavrilovska, Sanjay Kumar, Himanshu Raj, Karsten Schwan, Ripal Nathuji, Vishakha Gupta, Radhika Niranjan, Adit Randive, and Purav Saraiya, "High-Performance Hypervisor Architectures: Virtualization in HPC Systems", 1st Workshop on System-level Virtualization for High Performance Computing (HPCVirt 2007), Lisbon, Portugal, March 20, 2007.

[16] E. El-Araby, I. Gonzalez, and T. El-Ghazawi, "Performance Bounds of Partial Run-Time Reconfiguration in High-Performance Reconfigurable Computing", 1st International Workshop on High-Performance Reconfigurable Computing Technology and Applications (HPRCTA'07), held in conjunction with SC'07 Reno, NV, USA, November , 2007, pp. 11-20.

[17] Z. Li, and S. Hauck, "Configuration Prefetching Techniques for Partial Reconfigurable Coprocessor with Relocation and Defragmentation", Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA 2002), pp. 187-195.

[18] Z. Li, K. Compton, and S. Hauck, "Configuration Caching Management Techniques for Reconfigurable Computing", IEEE Symposium on FPGAs for Custom Computing Machines, 2000, pp. 87-96.

[19] E. El-Araby, M. Taher, K. Gaj, T. El-Ghazawi, D. Caliga, N. Alexandridis, "System-Level Parallelism and Concurrency Maximisation in Reconfigurable Computing Applications", International Journal of Embedded Systems (IJES) 2006, Vol. 2, No.1/2, pp. 62-72.

[20] E. El-Araby, "A System-Level Design Methodology For Reconfigurable Computing Applications", A Thesis for the Master of Science Degree in Computer Engineering, Department of Electrical and Computer Engineering, The George Washington University, January 2005.

[21] M. C. Smith, and G. D. Peterson, "Analytical Modeling for High Performance Reconfigurable Computers." In Proceedings of the SCS International Symposium on Performance Evaluation of Computer and Telecommunications Systems, July 2002.

[22] M. C. Smith, "Analytical Modeling of High Performance Reconfigurable Computers: Prediction and Analysis of System Performance", A Dissertation Proposal for the Doctor of Philosophy Degree in Electrical Engineering, The University of Tennessee, Knoxville, March 2002.

[23] J. D. Hadley, and B. L. Hutchings. Design methodologies for partially reconfigured systems. In P. Athanas and K. L. Pocek, editors, Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines, Napa, CA, April 1995, pp. 78-84.