Space and Time Sharing of Reconfigurable Hardware for Accelerated Parallel Processing

Esam El-Araby, Vikram K. Narayana, and Tarek El-Ghazawi

NSF Center for High-Performance Reconfigurable Computing (CHREC),
The George Washington University, Washington, DC 20052, USA
esam@gwmail.gwu.edu, {vikram, tarek}@gwu.edu

Abstract. High-Performance Reconfigurable Computers (HPRCs) are parallel machines consisting of FPGAs and microprocessors, with the FPGAs used as co-processors. The execution of parallel applications on such systems has mainly followed the Single-Program Multiple-Data (SPMD) model; however, overall system resources are often underutilized because of the asymmetric distribution of the reconfigurable (co-)processors relative to the (main) processors. Furthermore, with the introduction of HPRCs containing multi/many-core technologies, underutilization of system resources becomes more obvious especially for multi-tasking and multi-user usage. To address the asymmetry problem, we propose a resource virtualization solution based on Partial Run-Time Reconfiguration (PRTR). The proposed technique allows space, time, and/or space-time sharing of the reconfigurable (co-)processors among the (main) processors and thus increasing the overall system utilization. We show the effectiveness of the proposed concepts through a stochastic execution model verified with experimental implementations on the Cray XD1 platform. The results demonstrate favorable performance as well as scalability characteristics.

Keywords: Dynamic Partial Reconfiguration, Hardware Virtualization, High Performance Computing, Reconfigurable Computing

1 Introduction

Recent years have witnessed the introduction of stand-alone general purpose Reconfigurable Computers (RCs) as well as parallel 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]. These systems are capable of delivering high performance as well as maintaining flexibility, due to the use of FPGAs. The FPGAs are mainly used as co-processing element(s) (CPE) to the main processing element(s) (MPE) in order

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. IIP-0706352.

to accelerate critical functions in hardware. Several efforts have proved the significant performance speedups obtained by these systems for many different applications [4]. Applications for HPRCs are mainly developed using the Single-Program Multiple-Data (SPMD) programming model, which is the most common style of parallel programming used in HPC platforms. In SPMD [5], the participating processors simultaneously execute the same program at independent points, operating on different parts of the input data. Either shared memory or message passing techniques such as MPI may be adopted in order to deploy tasks and execute them in parallel [5], [6]. However, the use of SPMD programming paradigms for HPRCs can be challenging, due to the heterogeneity of the processing elements. This is primarily due to the fact that in HPRCs, the reconfigurable processors act as co-processing element(s) (CPE) to the main host processing element(s) (MPE). In particular, when the ratio of MPEs, CPEs, and their communication channels differs from unity, SPMD programs, which generally assume a unity ratio, might underutilize some of the system processing resources, for example microprocessors [4].

In this work, we propose to space, time, and/or space-time share the reconfigurable resources among the underutilized MPEs, namely microprocessor(s) and/or processor-cores by providing a virtual SPMD view and thus improving the overall system utilization for multi-user environments. In other words, the pool of reconfigurable resources will be virtually increased to maintain the symmetric view of SPMD, i.e. unity ratio among the MPEs, CPEs, and their communication channels. The implementation of these concepts is based on Partial Run-Time Reconfiguration (PRTR) from a practical perspective. We will provide a formal stochastic analysis of the execution model supported by experimental work. The execution model considers multi-user HPRCs equipped with multi-processor/multi-core technology. Our work utilizes PRTR on one of the current HPRC systems, Cray XD1. The results show near-linear scalability behavior for compute intensive applications.

This paper is organized such that section 2 provides a discussion of related work in context of run-time reconfiguration and hardware virtualization. Section 3 describes the space, time, and space-time techniques for sharing reconfigurable resources. Section 3 also includes our analytical model and explains the formulation steps of this model. Section 4 shows both the theoretical and the experimental results. Finally, Section 5 summarizes and concludes the paper with our findings.

2 Related Work

In this work, the primary objective is to share the reconfigurable resources (or CPEs) in HPRCs among all system microprocessors and/or processor-cores (or MPEs) in an SPMD view, irrespective of the system physical limitations/configuration, thereby providing support for true multi-user environments. In other words, regardless of the number of main processing elements (MPEs) and the co-processing elements (CPEs) in the system, we will try to provide a virtual 1:1 correspondence between MPEs and CPEs. To achieve the desired objective, we will leverage previous work and concepts that have been used for solving similar and

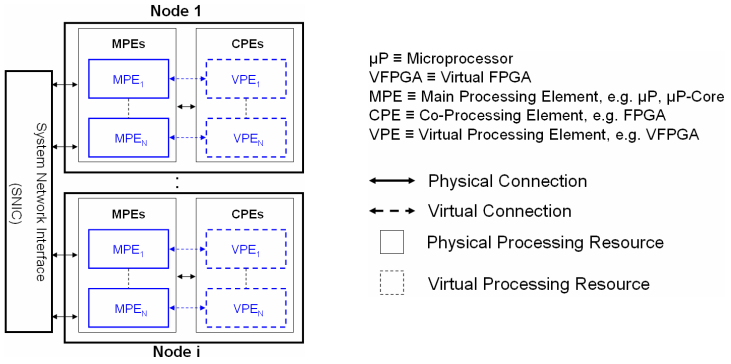


Fig. 1. Architectural assumptions (SPMD view of reconfigurable resources on HPRCs)

related problems, namely hardware virtualization. For example, we use the concept of virtual FPGA (VFPGA) proposed in [7].

Many of the proposed solutions in previous research [8], [9], are based on the strategies used in Operating Systems to support virtual memory – dynamic loading, partitioning, overlaying, segmentation, and paging, etc. All of these techniques strive to provide applications with the view of a larger FPGA, by virtually increasing the FPGA logic capacity. This concept of "virtual hardware" requires the use of special capabilities of the FPGAs, namely, Full Run-Time Reconfiguration (FRTR) and/or Partial Run-Time Reconfiguration (PRTR) [10],[11]. However, all of these proposed techniques are targeted towards embedded systems, with typically a single main processing element (MPE) and only one reconfigurable co-processing element (CPE). The multiplicity and imbalanced heterogeneity of the processing elements, common to HPRCs, is absent in embedded platforms. Furthermore, HPRC systems impose architectural constraints such as a shared configuration interface for the CPEs, as well as shared communication interfaces between the MPEs and CPEs. The unique nature of HPRCs adds a significant complexity to the virtualization problem, and therefore calls for a formal approach in order to solve it. Towards this end, we utilize and build on the techniques and methodologies introduced in [10],[11] by providing a virtualization infrastructure that allows space, time, and/or space-time sharing of the reconfigurable processors. Furthermore, we generalize the execution model based on stochastic Markov chains and queueing networks. The new model includes HPRCs equipped with multi-processor/multi-core technology utilizing the proposed virtualization infrastructure in a true multi-user environment.

3 Techniques for Sharing Reconfigurable Resources

Our methodology is based on the concept of "Computing in Time - Computing in Space" [12] for space, time, and/or space-time sharing of the reconfigurable resources. We first develop a formal analysis of the execution model based on our methodology, following an approach similar to what has been proposed in [10],[11]. In our analysis, we assume a multi-processor/multi-core HPRC architecture with

asymmetric heterogeneity at the node level [4]. All nodes are identical and in general each node is assumed to include a number of main processing elements (MPEs), e.g. microprocessors or processor cores, and a number of co-processing elements (CPEs), e.g. FPGAs. The number of MPEs, N_{MPE}, is not necessarily equal to the number of CPEs, N_{CPE}. In current HPRC systems N_{MPE} is typically larger than N_{CPE}, namely $N_{MPE} > N_{CPE}$. Additionally, each CPE is to be partitioned into a number of virtual processing elements (VPEs), N_{VPE}, such that each VPE is associated to a corresponding MPE maintaining a One-to-One correspondence among MPEs and their dedicated VPEs resulting in a balanced and symmetric distribution of system resources. In other words, the physical reconfigurable resources (FPGAs) will be virtualized and split into multiple virtual FPGAs (VFPGAs) such that $N_{VFPGA} = N_{VPE}$ in order to accommodate for the symmetry requirement of the SPMD execution. Each VFPGA will be located in a separate partially reconfigured region (PRR) on the physical FPGA. Finally, the number of necessary VPEs, N_{VPE}, for providing/guaranteeing the SPMD behavior can be given by equation (1) as follows:

$$N_{VPE} = \left\lceil \frac{N_{MPE}}{N_{CPE}} \right\rceil \times N_{CPE} \tag{1}$$

As the number of VPEs increases, the size of each VPE reduces; the task granularity α_{task} will determine the maximum number of VPEs, as seen below by rewriting (1):

$$N_{regions} = N_{VFPGA} = N_{VPE} = \min\left(N_{VPE}^{cores}, N_{VPE}^{tasks}\right) \times N_{CPE} \quad , \quad where \tag{2}$$

$$N_{VPE}^{cores} \equiv \left\lceil \frac{N_{MPE}}{N_{CPE}} \right\rceil, \quad N_{VPE}^{tasks} \equiv \left\lfloor \frac{1-\alpha_{static}}{\alpha_{task}} \right\rfloor, \quad \alpha_{static} \le 1 \ , \ \alpha_{task} \le 1$$

$$\alpha_{static} \equiv static \quad region \quad FPGA \quad resource \quad utilization$$

$$\alpha_{task} \equiv task \quad granularity \ (task \quad FPGA \quad resource \quad utilization)$$

Based on equation (2), space-time scheduling of task execution on VPEs is needed when $N_{VPE}^{tasks} < N_{VPE}^{cores}$ while space-only scheduling is needed when $N_{VPE}^{tasks} \geq N_{VPE}^{cores}$. In other words the execution of tasks needs to be performed through both space and time schedules when the task granularity is the governing bound on the number of VPEs while only space schedules are needed when there is a sufficient number of VPEs; at least equal to the number of MPEs. In later discussions, we will refer to equation (2) as the SPMD condition.

The usage model is SPMD in which the system receives some applications as input. These applications require on average a few independent hardware functions (tasks) that need to be executed on dedicated reconfigurable resources. The execution cycle for any task on an HPRC consists of the computation time, the total data input time, output time and the configuration time [10],[11], represented by T_{comp}, T_{in}, T_{out}, and T_{config} respectively. The I/O time T_{in} and T_{out} represent the time necessary to transfer data between the microprocessor and the FPGA. The baseline for our analysis is FRTR, where reconfigurable resources (CPEs) cannot be space shared among the node microprocessors/processor-cores (MPEs). We will focus our discussions on applications that are broken down into hardware tasks only. In addition, we assume

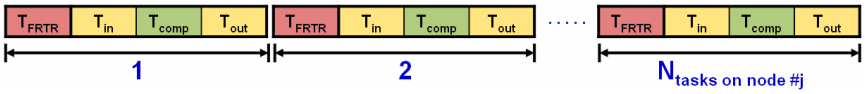


Fig. 2. Typical task execution per node using FRTR on a multi-user HPRC system

that each task is fully characterized by its time requirement, $T_{task}=T_{in}+T_{comp}+T_{out}$.

The execution model of FRTR on each node, see Fig. 2, is sequential among tasks and is independent from their owners (users). This is because the reconfigurable resource (CPE), assuming one per node, is not space sharable among the node MPEs rendering some MPEs unused. Considering different tasks to have similar average execution characteristics albeit different in functionalities, total execution time for the case of FRTR can be derived as follows:

$$T_{task} = T_{in} + T_{comp} + T_{out}$$

$$T_{task}^{FRTR} = T_{config}^{FRTR} + T_{task} = T_{FRTR} + T_{in} + T_{comp} + T_{out} \;,\; T_{user_i,node_j}^{FRTR} = \sum_{k=1}^{N_{tasks_{i,j}}} T_{task}^{FRTR} = N_{tasks_{i,j}} \cdot T_{task}^{FRTR}$$

$$T_{node_j}^{FRTR} = \sum_{i=1}^{N_{users}} T_{user_i,node_j}^{FRTR} = \sum_{i=1}^{N_{users}} N_{tasks_{i,j}} T_{task}^{FRTR} = T_{task}^{FRTR} \sum_{i=1}^{N_{users}} N_{tasks_{i,j}} \tag{3}$$

$$\Rightarrow T_{node_j}^{FRTR} = N_{tasks_{node_j}} T_{task}^{FRTR} \;,\; where$$

$T_{config}^{FRTR} \equiv T_{FRTR} \equiv Average\ \ full\ config.\ time\ of\ any\ task\ \ generated\ \ by\ \ any\ \ user\ \ on\ \ node\ \ \#\ j$

$T_{in} \equiv\ \ Average\ \ input\ \ transfer\ \ time\ \ of\ \ any\ \ task\ \ generated\ \ by\ \ any\ \ user\ \ on\ \ node\ \ \#\ j$

$T_{comp} \equiv\ \ Average\ \ computation\ \ time\ \ of\ \ any\ \ task\ \ generated\ \ by\ \ any\ \ user\ \ on\ \ node\ \ \#\ j$

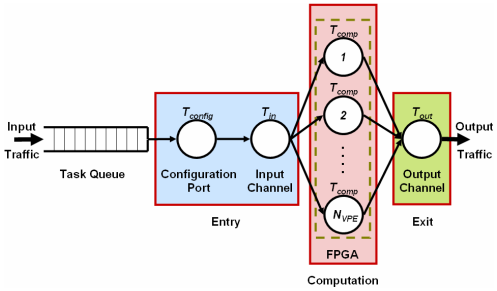
$T_{out} \equiv\ \ Average\ \ output\ \ transfer\ \ time\ \ of\ \ any\ \ task\ \ generated\ \ by\ \ any\ \ user\ \ on\ \ node\ \ \#\ j$

$T_{task} \equiv\ \ Average\ \ execution\ \ time\ \ of\ \ any\ \ task\ \ generated\ \ by\ \ any\ \ user\ \ on\ \ node\ \ \#\ j$

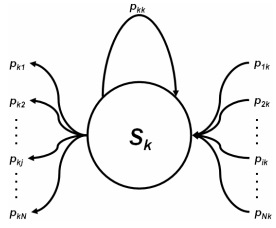
$T_{task}^{FRTR} \equiv Average\ \ execution\ \ time\ \ of\ \ any\ \ task\ \ generated\ \ by\ \ any\ \ user\ \ on\ \ node\ \ \#\ j\ \ for\ \ FRTR$

$T_{user_i,node_j}^{FRTR} \equiv Total\ \ execution\ \ time\ \ of\ \ all\ \ tasks\ \ generated\ \ by\ \ user\ \ \#i\ \ on\ \ node\ \ \#\ j\ \ for\ \ FRTR$

$T_{node_j}^{FRTR} \equiv Total\ \ execution\ \ time\ \ of\ \ all\ \ tasks\ \ generated\ \ by\ \ all\ \ users\ \ on\ \ node\ \ \#\ j\ \ for\ \ FRTR$



(3a) Traffic processes



(3b) Discrete-time Markov chains state diagram

Fig. 3. Queueing execution model

Queuing Analysis and Modeling

The execution model of our proposed virtualization technique and sharing mechanism can be viewed as a combination of three traffic (queueing) processes, namely entry, computation, and exit processes, see Fig. 3(a). The entry process is when tasks at the beginning of their execution life-cycle request configuration and data transfer from the MPEs into the VPEs/VFPGAs. The exit process is when tasks at the end of their execution life-cycle request data transfer from the VFPGA back to the MPEs. 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. In other words, we are considering that VFPGA reconfiguration, data transfers (in and out) and computations can be overlapped, sharing the I/O channel among all the MPEs in the node.

We base our analysis of the execution model on Markov processes. In particular, we will utilize the mathematical formulation of discrete-parameter (discrete-time) Markov chains [13]. Markov chains are described in general using a state diagram in which each state represents a case when the system contains a certain number of customers (tasks in our case). The system transitions from one state S_i to another state S_j with a probability p_{ij} known as the one-step transition probability, see Fig. 3(b). The probability distribution of a Markov chain is completely determined by the one-step transition probability matrix, $P=[p_{ij}]$, and the initial-state probability vector [13], see equation (4). Equation (5) shows some important and useful properties of P.

$$\vec{p}(n) = P^T \cdot \vec{p}(n-1) = \left(P^T\right)^n \cdot \vec{p}(0) = S \cdot \Lambda^n \cdot S^{-1} \cdot \vec{p}(0) \quad , \quad where \tag{4}$$

$$\vec{p}(n) = \begin{bmatrix} p_0(n) \\ p_1(n) \\ p_2(n) \\ . \\ p_k(n) \\ . \\ p_{N_{tasks}}(n) \end{bmatrix}, \quad \vec{p}(0) = \begin{bmatrix} p_0(0) \\ p_1(0) \\ p_2(0) \\ . \\ p_k(0) \\ . \\ p_{N_{tasks}}(0) \end{bmatrix}, \quad P = \begin{array}{c} \\ 0 \\ 1 \\ . \\ i \\ . \\ N_{tasks} \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & . & j & . & N_{tasks} \end{array} \\ \begin{bmatrix} p_{00} & p_{01} & . & p_{0j} & . & p_{0N_{tasks}} \\ p_{10} & p_{11} & . & p_{1j} & . & p_{1N_{tasks}} \\ . & . & . & . & . & . \\ p_{i0} & p_{i1} & . & p_{ij} & . & p_{iN_{tasks}} \\ . & . & . & . & . & . \\ p_{N_{tasks}0} & p_{N_{tasks}1} & . & p_{N_{tasks}j} & . & p_{N_{tasks}N_{tasks}} \end{bmatrix} \end{array}$$

$\vec{p}(n) \equiv state\ \ probability\ \ vector\ \ after\ \ n\ \ transitions$

$p_k(n) \equiv probability\ \ of\ \ the\ \ system\ \ being\ \ in\ \ state\ \ S_k\ \ (having\ \ k\ \ tasks)\ \ after\ \ n\ \ time\ \ steps$

$\vec{p}(0) \equiv initial-state\ \ probability\ \ vector$

$P \equiv Transition\ \ probability\ \ matrix$

$P^T \equiv Transposed\ \ Transition\ \ probability\ \ matrix$

$S \equiv Matrix\ \ of\ \ Eigenvectors\ \ of\ \ P^T$

$\Lambda \equiv Matrix\ \ of\ \ Eigenvalues\ \ of\ \ P^T$

$$\left.\begin{aligned} & 0 \le p_{ij} \le 1, \quad \sum_{j=0}^{N_{tasks}} p_{ij} = 1, \quad \sum_{k=0}^{N_{tasks}} p_k(n) = 1 \\ & p_{ij} = \begin{cases} r_{ij} \cdot \Delta t = \dfrac{r_{ij}}{r_{sampling}} \;, & j \neq i \\ 1 - \sum\limits_{\substack{k=0 \\ k \neq i}}^{N_{tasks}} p_{ik} = 1 - \sum\limits_{\substack{k=0 \\ k \neq i}}^{N_{tasks}} \dfrac{r_{ik}}{r_{sampling}} \;, & j = i \end{cases} \\ & r_{sampling} \ge \max_{i=0}^{N_{tasks}} \left(\sum_{\substack{k=0 \\ k \neq i}}^{N_{tasks}} r_{ik} \right) \equiv \max \left(\sum_{\substack{k=1 \\ k \neq 0}}^{N_{tasks}} r_{0k}, \sum_{\substack{k=0 \\ k \neq 1}}^{N_{tasks}} r_{1k}, \cdots, \sum_{\substack{k=0 \\ k \neq N_{tasks}}}^{N_{tasks}} r_{N_{tasks}k} \right) \end{aligned}\right\}, \; where \tag{5}$$

$r_{ij} \equiv Transition\ \ rate\ \ from\ \ state\ \ S_i\ \ to\ \ state\ \ S_j$

$\Delta t \equiv Time\ \ step\ \ duration$

$r_{sampling} \equiv \dfrac{1}{\Delta t} \equiv Time\ \ sampling\ \ rate$

The total execution time on node *j* can be determined/defined by the time step at which the instantaneous state probability vector becomes very close, within a certain error threshold ε, to its final steady state value. In other words, we define the total execution time as the minimum time step that is necessary for the system to reach as close as possible its final steady state where behavior transients become insignificant to a certain error threshold ε. This argument can be described as follows by equation (6):

$$\left.\begin{aligned} & T_{node_j}^{PRTR} = n_{\min} \cdot \Delta t = \frac{n_{\min}}{r_{sampling}} \\ & \lim_{n \to \infty} \left\| \vec{p}(n) - \vec{p}(n_{\min}) \right\| \equiv \left\| \vec{p}(\infty) - \vec{p}(n_{\min}) \right\| \le \varepsilon \\ & \Rightarrow \sqrt{\sum_{i=0}^{N_{tasks}} \left(p_i(\infty) - p_i(n_{\min}) \right)^2} \le \varepsilon \end{aligned}\right\}, \; where \tag{6}$$

$T_{node_j}^{PRTR} \equiv Total\ \ execution\ \ time\ \ of\ \ all\ \ tasks\ \ generated\ \ by\ \ all\ \ users\ \ on\ \ node\ \#\ j\ \ for\ PRTR$

$n_{\min} \equiv Minimum\ \ number\ \ of\ time\ \ steps\ \ necessary\ \ for\ \ close-to-steady-state\ behavior$

$\vec{p}(\infty) \equiv Steady\ \ state\ \ probability\ vector$

$\varepsilon \equiv Error\ threshold$

Taking into consideration that the final execution time on the system is determined by the longest execution time among all nodes, namely the slowest (critical) node, the performance gain (speedup) of PRTR in reference to FRTR can be expressed as follows by combining equations (3) and (6):

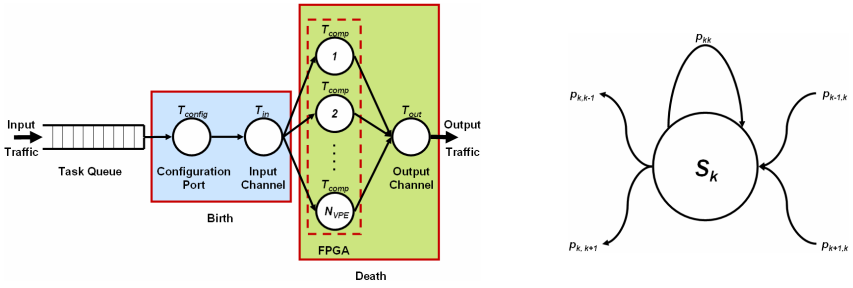
$$T_{total}^{FRTR} = \underset{j=1}{\overset{N_{nodes}}{MAX}}\left(T_{node_j}^{FRTR}\right) = T_{node_j}^{FRTR} \;,\; T_{total}^{PRTR} = \underset{j=1}{\overset{N_{nodes}}{MAX}}\left(T_{node_j}^{PRTR}\right) = T_{node_j}^{PRTR} \;,\; S \equiv \frac{T_{total}^{FRTR}}{T_{total}^{PRTR}}$$

$$\Rightarrow S = \frac{N_{tasks_{node_j}} \left(T_{FRTR} + T_{in} + T_{comp} + T_{out}\right)}{T_{node_j}^{PRTR}} \;,\; where \tag{7}$$

$T_{total}^{FRTR} \equiv$ *Total average execution time of all tasks generated by all users on all nodes for FRTR*

$T_{total}^{PRTR} \equiv$ *Total average execution time of all tasks generated by all users on all nodes for PRTR*

$S \equiv$ *Speedup or performance gain of PRTR relative to FRTR*



(4a) Traffic processes (4b) Birth-Death state model

Fig. 4. Simplified execution model

Due to the fact that typical HPRC architectures are designed with a single configuration port and a single communication channel between MPEs and CPEs, we will use a special class of Markov chains that is typically used to describe queueing systems. More specifically, we will simplify our model as a birth-death process in which transitions are allowed between only neighboring states. The simplified execution model is shown in Fig. 4. Equation (8) describes the simplified model.

$$\vec{p}(0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ . \\ 0 \\ . \\ 0 \end{bmatrix}, \quad P = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ . \\ i \\ . \\ N_{tasks} \end{array} \begin{array}{c} \begin{array}{cccccccccc} 0 & 1 & 2 & 3 & . & i-1 & i & i+1 & . & N_{tasks} \end{array} \\ \begin{bmatrix} p_{00} & p_{01} & 0 & 0 & . & 0 & 0 & 0 & . & 0 \\ p_{10} & p_{11} & p_{12} & 0 & . & 0 & 0 & 0 & . & 0 \\ 0 & p_{21} & p_{22} & p_{23} & . & 0 & 0 & 0 & . & 0 \\ 0 & 0 & p_{32} & p_{33} & . & 0 & 0 & 0 & . & 0 \\ . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & p_{i,i-1} & p_{ii} & p_{i,i+1} & . & 0 \\ . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & 0 & 0 & 0 & . & p_{N_{tasks}N_{tasks}} \end{bmatrix} \end{array}, \; where \tag{8}$$

$$r_{ij} = \begin{cases} \lambda \;,\; 0 \le i < N_{tasks} \;,\; j = i+1 \\ i\mu \;,\; 0 < i < N_{VPE} \;,\; j = i-1 \\ N_{VPE}\,\mu \;,\; N_{VPE} \le i \le N_{tasks} \;,\; j = i-1 \\ 0 \;,\; otherwise \end{cases}$$

$$\lambda = \frac{1}{T_{config} + T_{in}} \;,\; and \quad \mu = \frac{1}{T_{comp} + N_{VPE} \cdot T_{out}}$$

In order to investigate how scalable our approach, we will introduce what we call the scalability factor, η. The scalability factor, η, can be defined as the normalized speedup. In other words, the speedup achieved by a multiple of MPE-VPE pairs would be normalized with respect to the speedup achieved by one MPE-VPE pair. More specifically, η is defined as the ratio between two values of the speedup, namely $S(N_{VPE})$ and $S(1)$, as a function of N_{VPE}. This expression can be written as shown in equation (9). By taking the limit of equation (9) as the number of VPEs increases indefinitely, namely $N_{VPE} \rightarrow \infty$, the asymptotic scalability behavior can be obtained as given by equation (10).

$$\eta(N_{VPE}) \equiv \frac{S(N_{VPE})}{S(1)}$$

$$S(N_{VPE}) = \frac{N_{tasks_{node_j}} \left(T_{FRTR} + T_{in} + T_{comp} + T_{out}\right)}{T^{PRTR}_{node_j}(N_{VPE})} \quad , \quad S(1) = \frac{N_{tasks_{node_j}} \left(T_{FRTR} + T_{in} + T_{comp} + T_{ou}\right)}{T^{PRTR}_{node_j}(1)}$$

$$\Rightarrow \eta(N_{VPE}) = \frac{T^{PRTR}_{node_j}(1)}{T^{PRTR}_{node_j}(N_{VPE})} \tag{9}$$

$$\Rightarrow \eta_{\infty} \equiv \lim_{N_{VPE} \rightarrow \infty} \eta(N_{VPE}) = \frac{T^{PRTR}_{node_j}(1)}{T^{PRTR}_{node_j}(\infty)} \quad , \quad where \tag{10}$$

$\eta(N_{VPE}) \equiv Scalability\ \ factor\ \ as\ \ a\ \ function\ \ of\ \ the\ \ number\ \ of\ \ node\ \ VPEs,\ \ N_{VPE}$

$\eta_{\infty} \equiv Asymptotic\ \ Scalability\ \ factor\ \ as\ \ the\ \ number\ \ of\ \ VPEs\ \ increases\ \ indefinitely$

4 Results

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].

To verify the proposed virtualization techniques and the execution model, a set of experiments were conducted, starting with an application that carries out image feature extraction. In the chosen application, high frequency noise components were first removed from the images using two different algorithms, followed by some processing to extract the object edges of interest. Specifically, a sequence of image processing functions were executed, namely median filtering followed by Sobel edge detection, and smoothing filtering also followed by Sobel edge detection. The final images were then transferred back to the microprocessor memory for some quality checks.

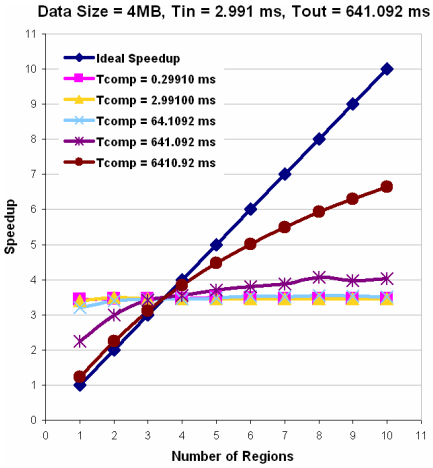
Table 1. Selected scenarios for Cray XD1

	Case 1 ($T_{comp}<T_{in}<T_{out}$)	Case 2 ($T_{comp}=T_{in}<T_{out}$)	Case 3 ($T_{in}<T_{comp}<T_{out}$)	Case 4 ($T_{in}<T_{comp}=T_{out}$)	Case 5 ($T_{in}< T_{out}<T_{comp}$)
T_{comp} (msec)	0.299	2.991	64.109	641.092	6410.92

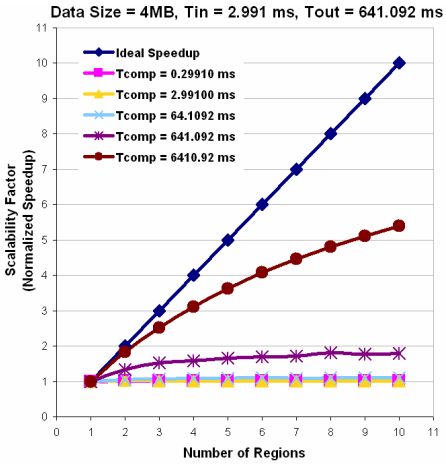
It may be noted that the SPMD condition as described by equation (2) suggests that the maximum number of PRRs should at least equal the number of microprocessors (MPEs) per node. For Cray XD1, the number of MPEs per node is two. We therefore conducted an initial set of experiments using dual VFPGAs (VPEs). In order to evaluate the proposed execution model for a larger number of cases, we added some features to the virtual infrastructure on Cray XD1 to emulate scenarios for a larger number of VFPGAs (PRRs). The emulation-based virtual infrastructure accepts a minimum set of parameters for XD1 since it is running on the machine itself. These parameters include the number of VFPGAs and different computation times to emulate different tasks, etc. Five scenarios were emulated to validate the model and the proposed infrastructure as shown in Table 1. These scenarios were selected to investigate different classes of applications starting from the least computational intensive, namely I/O intensive, in case 1 to the most computational intensive applications in case 5, see Table 1. A large (infinite) amount of task traffic was submitted to be executed on a variable number of VPEs from 1 to 10 VFPGAs.

Results for the described scenarios were obtained from actual runs on Cray XD1, and compared against the proposed execution model presented in Section 3. The measured results were found to be in good agreement with the mathematical model. Fig. 5 shows some of these experimental findings for the scenarios listed in Table 1, as a speedup over the conventional execution based on FRTR. The parameters collected from our experiments are T_{FRTR}= 1678.040 ms, T_{PRTR}= 19.771 ms, T_{in}=2.991 ms, and T_{out}= 641.092 ms. Equation (7) suggests that the speedup value should be 3.49, which is consistent with the value measured and shown in Fig. 5(a).

It is worth mentioning that for the measured parameters on Cray XD1 there is a region in Fig. 5(a) where the measured speedup is not upper bounded by the total number of processing elements, N_{MPE}. The upper bound is rather dictated by the ratio between T_{FRTR} and T_{PRTR}. This is true to a certain point, see Fig. 5(a), beyond which the situation reverses and the speedup would be upper bounded by the total number of processing elements, N_{VPE}. This is due to the fact that for a small number of VPEs the savings in the total execution time is not because of the parallel execution of tasks but rather because of the savings in (re)configuration overhead. On the other hand, for a large number of VPEs the savings in the total execution time because of the parallel execution of tasks become more significant than the savings in (re)configuration overhead.



(5a) Speedup achieved using multiple PRRs (VFPGAs) as a function of $N_{VPE} = N_{regions}$



(5b) Scalability factor, η

Fig. 5. Performance of applications using virtual resources

Finally, the scalability, as defined by equation (9), of our approach is shown in Fig. 5(b). In general, HPC applications with constant overhead show a similar scalability behavior to the one shown in Fig. 5(b). Such behavior is typically due to communication overhead between the system nodes. In our case, the overhead is due to (re)configuration and data transfer back and forth between the MPEs and VPEs, see equation (10). As shown in Fig. 5(b), when the task computation time, T_{comp}, becomes much larger than the associated overhead, the execution speedup, using our techniques, approaches linear behavior. In other words, the execution of highly compute intensive applications using our virtualization techniques becomes linearly scalable, which is a typical behavior on HPC supercomputers.

5 Conclusion

In this paper we presented an effort of virtualizing and space, time, and/or space-time sharing of reconfigurable resources based on Partial Run-Time Reconfiguration (PRTR) for High-Performance Reconfigurable Computing (HPRC) systems configured with multi-processor/multi-core technologies. 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 stochastic model of multi-user 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. The experimental results showed good agreement with the analytical model expectations. Sharing reconfigurable resources among the underutilized microprocessors/processor-cores by providing a virtual SPMD view allows improving the overall system versatility, resources utilization, and application performance in multi-user environments. The approach we followed for Cray XD1 has been proven to be scalable and general to be applied to any of the available HPRC systems.

6 References

1. SRC Computers, Inc.: SRC Carte™ C Programming Environment v2.2 Guide SRC-007-18. (2006)
2. Silicon Graphics Inc.: Reconfigurable Application-Specific Computing User's Guide 007-4718-005. (2007)
3. Cray Inc.: Cray XD1™ FPGA Development S-6400-14. (2006)
4. T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, D. A. Buell.: The Promise of High-Performance Reconfigurable Computing. IEEE Computer, vol. 41, no. 2, 69--76 (2008)
5. NIST: Single Program Multiple Data, http://www.nist.gov/dads/HTML/singleprogrm.html
6. F. Darema: SPMD Model: Past, Present and Future, Recent Advances in Parallel Virtual Machine and Message Passing Interface. In: 8th European PVM/MPI Users' Group Meeting, LNCS, vol. 2131, p. 1 (2001)
7. W. Fornaciari, V. Piuri: General Methodologies to Virtualize FPGAs in HW/SW Systems. In: Midwest Symposium on Circuits and Systems, pp. 90-93 (1998)
8. Z. Li, S. Hauck: Configuration Prefetching Techniques for Partial Reconfigurable Coprocessor with Relocation and Defragmentation. In: ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 187-195 (2002)
9. Z. Li, K. Compton, S. Hauck: Configuration Caching Management Techniques for Reconfigurable Computing. In: IEEE Symposium on FPGAs for Custom Computing Machines, pp. 87-96 (2000)
10. E. El-Araby, I. Gonzalez, T. El-Ghazawi.: Exploiting Partial Runtime Reconfiguration for High-Performance Reconfigurable Computing. ACM Transactions on Reconfigurable Technology and Systems, vol. 1, no. 4, 21:1--21:23 (2009).
11. E. El-Araby, I. Gonzalez, 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'08), pp.1-8 (2008)
12. C. Siemers: Reconfigurable Computing between Classifications and Metrics - The Approach of Space/Time-Scheduling. In: R.W. Hartenstein and H. Grünbacher (eds.) FPL 2000. LNCS, vol. 1896, pp. 769-772 (2000)
13. L.R. Lipsky.: Queueing Theory: A Linear Algebraic Approach. Macmillan, New York (1992).