A Simulation Framework for Rapid Analysis of Reconfigurable Computing Systems

CASEY REARDON, ERIC GROBELNY, ALAN D. GEORGE, and GONGYU WANG
NSF Center for High-Performance Reconfigurable Computing (CHREC),
University of Florida

Reconfigurable computing (RC) is rapidly emerging as a promising technology for the future of high-performance and embedded computing, enabling systems with the computational density and power of custom-logic hardware and the versatility of software-driven hardware in an optimal mix. Novel methods for rapid virtual prototyping, performance prediction, and evaluation are of critical importance in the engineering of complex reconfigurable systems and applications. These techniques can yield insightful tradeoff analyses while saving valuable time and resources for researchers and engineers alike. The research described herein provides a methodology for mapping arbitrary applications to targeted reconfigurable platforms in a simulation environment called RCSE. By splitting the process into two domains, the application and simulation domains, characterization of each element can occur independently and in parallel, leading to fast and accurate performance prediction results for large and complex systems. This article presents the design of a novel framework for system-level simulative performance prediction of RC systems and applications. The article also presents a set of case studies analyzing two applications, Hyperspectral Imaging (HSI) and Molecular Dynamics (MD), across three disparate RC platforms within the simulation framework. The validation results using each of these applications and systems show that our framework can quickly obtain performance prediction results with reasonable accuracy on a variety of platforms. Finally, a set of simulative case studies are presented to illustrate the various capabilities of the framework to quickly obtain a wide range of performance prediction results and power consumption estimates.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Modeling Techniques; I.6.0 [**Simulation and Modeling**]: General

General Terms: Performance, Experimentation, Verification

Additional Key Words and Phrases: Reconfigurable computing, discrete-event simulation, performance prediction

ACM Reference Format:
Reardon, C., Grobelny, E., George, A. D., and Wang, G. 2010. A simulation framework for rapid analysis of reconfigurable computing systems. ACM Trans. Reconfig. Techn. Syst. 3, 4, Article 25 (November 2010), 29 pages. DOI: 10.1145/1862648.1862655.
http://doi.acm.org/10.1145/1862648.1862655.

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422.
Authors' address: Department of Electrical and Computer Engineering, University of Florida, PO Box 116200, 327 Larsen Hall, Gainesville, FL 32611-6200, Contact email: reardon@hcs.ufl.edu.
Permission to make digital or hard copies part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2010 ACM 1936-7406/2010/11-ART25 $10.00 DOI: 10.1145/1862648.1862655.
http://doi.acm.org/10.1145/1862648.1862655.

1. INTRODUCTION

As computing applications and platforms continue to increase in size and complexity, two reformations are taking place. The first is an architectural reformation, where improvements in the computational capabilities of devices is achieved through explicit parallelism, since it has become increasingly difficult to provide comparable performance increases through higher clock speeds and implicit instruction-level parallelism alone. In response, an application reformation is underway as well, requiring explicit parallelism in applications to exploit the parallelism of new hardware. Under these two reformations, Reconfigurable computing (RC) is becoming recognized as an increasingly important and viable paradigm for high-performance computing in times where the size and power consumption of clusters and traditional supercomputers have grown to alarming levels. With RC, the performance potential of underlying hardware resources in a system can be fully realized in a highly adaptive manner, extending the fields of large-scale and embedded high-performance computing. Hybrid systems of microprocessors and FPGAs can leverage system-level concepts from conventional high-performance computing while accommodating hardware reconfigurability. For these reasons, recent trends and research suggest RC systems will become common in fields such as computer vision [Bondalapati and Prasanna 2002], digital signal processing [Tessier and Burleson 2001], embedded computing [Garcia et al. 2006], and many others.

Application development typically follows an iterative four-stage process known as the FDTE model (Figure 1), whose name comes from the four stages of development: Formulation, Design, Translation, and Execution. The initial stage in the FDTE development model is formulation, where strategic planning and exploration is performed before coding a specific implementation. Early exploration during formulation can help ensure that the proposed implementation will meet project specifications before intensive coding has begun. To provide these services, methods for performance prediction and analysis of abstract designs are needed. Typically, this stage is often bypassed or performed haphazardly using quick and informal techniques, which can have dire consequences later in the development process. In the design stage, code is developed to implement the specific solution arrived at during formulation. The translation stage is then responsible for converting the design code into an executable format that can be fed to the target platform. In the execution stage, the program is executed, and runtime data is gathered to aid in the debugging, verification, and optimization processes. The data acquired during execution is typically fed back to the design stage, where changes to the code are made to address the issues observed during execution. Thus, the developer must continuously iterate through these three stages until the specifications of the project are met, which can be a costly process.

With overall development times and costs expanding, effective formulation techniques becomes increasingly critical to ensure timely development. Without effective formulation techniques and tools, costly iterations through the design, translation and execution (DTE) stages are performed, many of which could have been avoided through careful planning and analysis. The

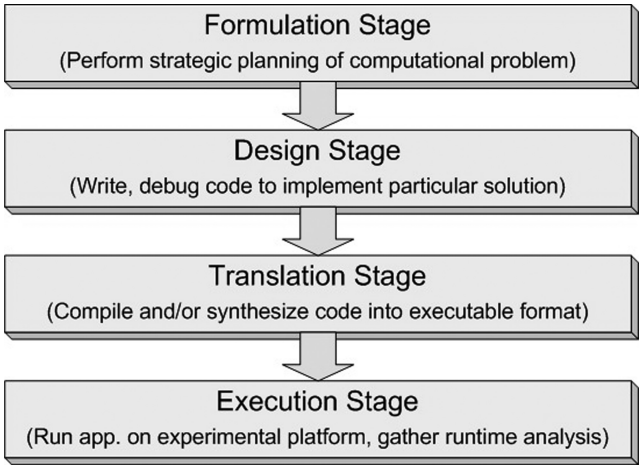


Fig. 1. FDTE model of development.

introduction of reconfigurable devices can further increase the design complexity of such systems as well. In addition to traditional design-space parameters such as processor speed, memory subsystem performance, and network interconnect, RC systems must also consider FPGA resources, IO subsystem performance, and reconfiguration capabilities. The large design space increases the need to conduct algorithm and architecture explorations early in the development process, that is, during formulation, to potentially save considerable amounts of development time. Computer simulation is a popular technique for performance prediction of complex systems when the application or platform are not readily available to the designer, whether due to cost or time constraints. For all of these reasons, a fast and efficient simulation environment is needed in order to tackle the expanded design space of RC while supporting the analysis of large-scale systems, otherwise prohibitively long simulation times will dissuade users from performing this important task.

In this article, we present a framework for simulating applications on RC systems called the RC Simulation Environment (RCSE), which focuses on balancing simulation speed and fidelity. Built atop the Fast and Accurate Simulation Environment (FASE) methodology [Grobelny et al. 2007], RCSE employs a process where reconfigurable computing applications are described by a sequence of high-level events, forming a script which stimulates discrete-event system models. Using a high-level of abstraction, this framework can rapidly provide a broad range of predictions for a given reconfigurable application and system when the performance of the application is data-independent. Furthermore, simulation results can be obtained for very large-scale and complex RC systems without requiring an inordinate amount of time or resources. These results can be obtained efficiently during the formulation stage, allowing designers to make informed strategic decisions early in the development process.

The remainder of this article is organized as follows. In Section 2, an overview of related research for performance prediction and simulation of reconfigurable computing systems is provided. Next, a detailed description of the RCSE framework is presented in Section 3. Section 4 provides a walkthrough with RCSE, including validation results to verify the accuracy of our approach. Results from a selected set of application case studies are provided in Section 5 demonstrating the effectiveness and usefulness of this framework. Finally, conclusions and future work are summarized in Section 6.

2. BACKGROUND AND RELATED RESEARCH

Typically, acquiring and building large-scale systems to determine how an application performs is simply not a viable option, due to the enormous costs associated with this approach. Therefore, modeling and simulation have been extensively employed to gain insight into the expected performance of these complex systems and applications. Classically, there are two types of computer simulation environments: execution-driven and trace-driven. Execution-driven simulations center around the execution of the program's code on simulation models designed to achieve near clock-cycle accuracy, such as the popular SimpleScalar toolset [Burger and Austin 1997]. While execution-driven simulation can provide very accurate and useful results, these simulations are often too slow and impractical when considering large-scale systems and applications. This problem becomes compounded when researchers need results from a series of simulations to complete their analysis. Therefore, methods for speeding up these simulations are desired. One approach to accelerate execution-driven simulations is to apply statistical sampling to select a representative subset of the benchmark application for execution-driven simulation [Wunderlich et al. 2006; Hamerly et al. 2005; Lafage and Seznec 2001]. This technique reduces simulation times by executing only a portion of the benchmark under consideration, which is then fed into a detailed processor simulator. One drawback of this approach is the need for detailed warming, where sections of code are executed in detail simply to estimate the state of the system at the beginning of a block of sampled code. Furthermore, it is unclear how such an approach would be applied to parallel systems, where the communication between processors could be omitted during sampling.

Trace-driven simulation offers another alternative by using a method of abstraction to formulate a representation of the application used as input into the simulation models. While using an abstract representation makes it impossible to replicate the exact behavior of the entire system, relatively accurate results can be gathered in significantly less time. The exact speed and accuracy of a trace-based simulator is dependant on the fidelity of the simulation models and the method of abstraction used for gathering the trace data. Varying approaches have been proposed for achieving accurate simulation results for high-performance computing systems without prohibitively long simulations [Uhlig and Mudge 1997; Snavely et al. 2002; Pllana and Fahringer 2005]. Unfortunately, none of the simulation projects listed thus far support the simulation of RC-based applications and systems.

While many research projects have investigated the domain of traditional computing simulation, comparatively few efforts have focused towards system-level performance prediction of RC systems. The RC Amenability Test (RAT) defines a set of analytic equations for predicting the potential speedup of a particular FPGA core design focusing on accurate throughput analysis, though the framework considers numerical precision and resource utilization as well [Holland et al. 2007]. A set of application-specific analytical equations for performance prediction of iterative synchronous programs running on heterogeneous clusters with RC devices have been proposed [Smith and Peterson 2002]. Results from case studies show these equations to be reasonably accurate for the applications under study, but the accuracy of these results diminishes as the cluster size increases. While the analytical models place great emphasis on load balancing, it can be hard to predict the precise effects of resource contention in complex systems with such models. Another set of analytical equations have been proposed to find potential performance bottlenecks imposed by the memory transfer system with FPGA applications [Steffen 2007]. These analytic models abstractly characterize algorithms based on their data-movement patterns and computational density, then identify the memory layer that will act as the largest bottleneck for the algorithm. Thus, the analytical models are limited to determining the best performance the hardware architecture is capable of supporting, as opposed to predicting the actual performance the architecture is expected to produce. Though analytic models are often valuable for providing useful data very quickly, their effectiveness is limited for performance prediction of large and complex RC systems. Therefore, our methodology focuses on using simulative techniques to provide performance prediction data.

Other projects have focused on using simulation for performance prediction of RC systems. One example is the Hybrid System Architecture Model (HySAM) and DRIVE simulation framework [Bondalapati 2001]. Within HySAM, architectures are defined by a set of attributes describing the capabilities of the system, while applications are partitioned into a series of tasks split between the CPU and RC device that are fed to DRIVE for visualization. A separate project created a co-simulation environment which integrated two existing cycle-accurate simulators [Enzler et al. 2005]. The authors employed the well-known SimpleScalar tool suite for processor emulation, while incorporating the ModelSim VHDL simulator for modeling the reconfigurable device. The reconfigurable device simulator assumes a specified FPGA design, targeted for the embedded domain. Another system-level RC simulator was built on top of the Simics platform, using the GEMS memory extension [Fu and Compton 2006]. The Simics-based simulator employs a cycle-accurate processor and memory simulator, spending significant time and effort capturing the precise pattern of memory accesses when RC device transfers are involved. Kernels executed by the reconfigurable device are simply performed by a software equivalent in the simulator to ensure functional correctness, while the hardware execution time is provided separately to the simulator so as to correctly advance simulation time. Results have shown both of the latter two simulation environments to be accurate in the case studies presented, which one would expect when using cycle-accurate models. However, in both cases, the results are

only reported for machines that contain a single processor chip and RC device. Also, no information is given describing the time and accuracy associated with each simulator when the system size is scaled to a large number of nodes. Since the RCSE framework presented in this article aims to provide timely prediction results for large-scale systems during formulation, a trace-driven approach is used.

A hierarchical model-based framework for FPGA development is presented by Mohanty and Prasanna [2007] intended to support evaluation of design alternatives early in the design process. The framework integrates a high-level performance estimator (HiPerE) and a design space exploration tool (DESERT) for efficient evaluation of candidate mappings against user-specified performance requirements onto architectures that can be described using the Generic Model (GenM) for System-on-Chip (SoC) architectures [Mohanty et al. 2002]. DESERT is used to prune the design space to eliminate any candidate design that fails to meet user-specified constraints. HiPerE is then employed to quickly evaluate the remaining designs using trace-based simulations, in terms of energy and latency. Both DESERT and HiPerE are integrated into the MILAN integrated simulation environment for embedded system design [Bakshi et al. 2001]. HiPerE is only designed to evaluate architectures that can be described by GenM, which is designed for SoC architectures. In contrast, RCSE supports analysis of any architecture that can be constructed using available component models and/or component models developed by the user.

Complimentary to performance prediction, a healthy amount of research exists in the field of power consumption modeling and prediction for hardware devices, including FPGAs. In addition to their potential for increasing the performance of certain applications, reconfigurable devices also offer the advantage of significant power savings over many current processors. Consequently, the ability to predict the power consumption of reconfigurable devices is important in areas where low-power solutions are critical to success, as in many embedded computing applications. Multiple projects have incorporated power estimation techniques into existing CAD tools for evaluating an FPGA design [Poon et al. 2005; Anderson and Najim 2004]. These tools apply information gathered during synthesis of the FPGA design to accurately estimate the power consumption of the circuit. Similarly, the power evaluation framework from Li et al. [2003] is designed for evaluating the power efficiency of the FPGA architecture based on attributes such as their LUT size, cluster size, and wiring scheme. At a higher level, power consumption can be estimated earlier in the design cycle using a model similar to the one described in [Weiss et al. 2000]. High-level parameters of the device and the design are combined to allow designers to gain a rough estimate of power consumption before the circuit design is complete and ready to be tested within the CAD tool environment. Given that RCSE is intended to provide analysis of potential designs during formulation, a high-level power model similar to the one presented by Weiss et al. [2000] is used. The remaining power estimation models require information describing the final core design, which often is not available in the formulation stage.

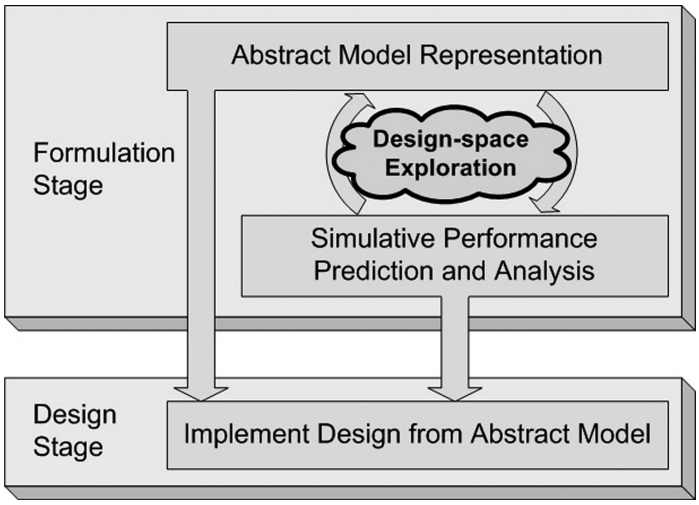


Fig. 2. Detailed diagram of formulation process.

3. SIMULATION FRAMEWORK OVERVIEW

The goal of the RCSE framework presented in this article is to augment the formulation stage of RC development with a methodology for providing efficient simulative performance prediction. Beginning with an abstract model that defines a candidate design, developers can perform design-space exploration by iterating between strategic designs and simulation before advancing beyond the formulation stage. Only until the analysis tools provide acceptable feedback for a candidate design will a developer advance to the remaining development stages with their abstract design. This process, illustrated in Figure 2, can greatly improve the overall productivity of RC projects by catching strategic errors and design flaws early in the development cycle.

To realize the formulation-based development flow described in the previous paragraph, a simulation framework was created to support efficient algorithmic and architectural exploration over many design parameters. This section details the RCSE framework for performance prediction of RC systems that balances speed, accuracy and flexibility. To achieve this goal, additional effort is performed within the application domain to characterize the behavior of the application so that instruction-level details of the program can be abstracted to reduce the complexity of the discrete-event simulation. This framework is split into two separate domains - the application domain and the simulation domain. This split allows users to characterize applications independently of the candidate system architectures while supporting concurrent model development that is independent of the potential applications. This independence offers a high level of data and model reusability and modularity which in turn facilitates rapid analyses of numerous virtually prototyped systems and applications. With this reusability, users can effectively iterate within the

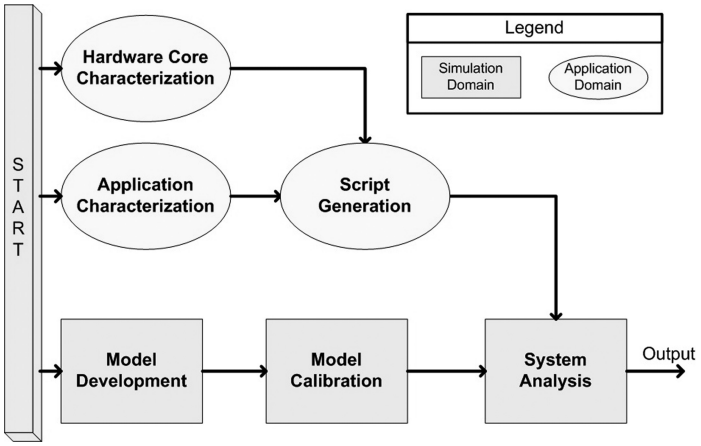


Fig. 3. RCSE framework diagram.

formulation stage by making changes within one domain and simply reusing the model from the second domain for the next simulation as the design space is explored. The overall structure of the RCSE framework, and the key steps within each domain, are illustrated in Figure 3. Sections 3.1 and 3.2 provide details and examples of the procedures employed in the application and simulation domains, respectively.

The Fast and Accurate Simulation Environment (FASE) is a simulation methodology that attempts to balance speed and accuracy for simulating message-passing parallel applications on HPC clusters using a two-stage simulation process [Grobelny et al. 2007]. The RCSE framework presented in this section leverages key concepts and models used in FASE, but also proposes many extensions and additions to support RC-based analyses. While FASE defines two primary stages in its simulation process, the RCSE framework defines six primary stages, including the hardware core characterization stage which is unique to RC. In addition, new sets of discrete-event models for RC-based architectural components have been developed, and the scripting language has been redefined and significantly expanded in order to effectively represent RC applications.

As previously mentioned, FASE is primarily designed for simulating message-passing parallel applications on HPC clusters. Since the RCSE framework incorporates many of the same approaches used by FASE, certain types of systems are not effectively modeled with RCSE. For example, FASE and the RCSE are currently designed to predict the performance of a single application on an unloaded system, therefore extensions are necessary to support analysis on a multi-user or loaded system. Streaming applications are not easily supported either since they cannot be naturally described with the current scripting language. Finally, RCSE assumes that actions on the RC device are initiated by commands from a host microprocessor, thus systems in which the RC device acts autonomously are not currently supported as well.

3.1 Application Domain

The purpose of the application domain is to collect characterization data on a selected application that captures its inherent behavior. The behavior of the application is captured in order to create an application model in the form of a script that acts as stimulus data for the simulation models. Algorithmic exploration can then be performed by making changes to the application model, producing a new script to stimulate further simulations. The application domain is divided into three stages; hardware core characterization, application characterization, and script generation. The remainder of this subsection describes each of the stages that make up the application domain, which are illustrated by the ovals in Figure 3.

Hardware core characterization, which is not a part of the original FASE framework, defines the behavior of the kernel or function to be performed within the RC device. The characterization of hardware cores is performed by defining a set of parameters for the core. These parameters include computational delay (in terms of latency and throughput), resource utilization, input data size, and output data size. The computation time for an RC core can be obtained from multiple sources. For cases where the hardware core design is complete and available to the user, this value may be measured experimentally, or similarly using delays supplied from cycle-accurate functional simulations of the hardware design from vendor-supplied tools. Both methods provide reasonably accurate results assuming deterministic behavior of the RC devices. While it is possible to integrate a cycle-accurate hardware simulator into our environment, it makes little sense to use one in cases where a deterministic hardware core is executed numerous times during a single run of an application. Instead of performing costly cycle-accurate simulations numerous times, it is more efficient to extract the performance data from one single run in this early stage. Furthermore, when a finished hardware core is not available to the user, cores can be simulated at the beginning of the development cycle by using parameter estimates based on initial core designs. By leveraging a process such as the RAT methodology presented by Holland et al. [2007], quick estimates of the core performance can be obtained by users with a fundamental understanding of their algorithm. Following the RAT methodology allows the user to estimate the full set of parameters needed to characterize a hardware core. The resource utilization parameters are important in order to consider performance gains when scaling up device size by squeezing more cores onto the fabric at once, thus executing on more data in parallel. The modeling of partial reconfiguration is supported as well by allowing cores to be exchanged within the RC device during the simulation. However, an in-depth study of this technique is not included in this paper. The final parameters, the input and output data sizes, allow accurate modeling of transactions with the RC device. These parameters are critical in order to accurately capture the communication delays incurred when passing data between various node components and RC device.

Another vital step in the application domain that can be conducted in parallel with characterizing the hardware core is the application characterization stage. This step involves identifying and gathering a sequence of key events

that defines the behavior and attributes of the target application. The events currently supported include computation conducted by host processors, computation performed by RC devices, along with inter- and intra-node communication. When the performance of the application is data-independent, a specific sequence of these events can be employed to accurately capture the behavior of any parallel or serial RC application. However, gathering these sequences of events is a key challenge when dealing with RC applications due to the large number of programming interfaces implemented to transfer data to and from RC devices. RCSE addresses this challenge by using a scripting language that allows the user to manually represent the behavior of an RC application. When using this approach, classic instrumentation tools can be employed to gather the data relevant to defining host computation and inter-node communication, such as the Performance Application Programming Interface (PAPI) from the University of Tennessee [Browne et al. 2000]. This framework currently uses PAPI to trace the number of instructions and memory accesses performed by the host processor during each computation block. This data is used to estimate the amount of time spent by the processor in that computation block, along with predicting the amount of contention that would be experienced with shared resources. Presently, the incorporation of RC events into scripts must be performed manually, since no common standard exists that defines interactions with RC devices. However, the adoption of a standard RC programming interface or abstract modeling language would remove the requirement of manual script creation, since RC-related events could be detected by an automated code or model parser. Combined with other standards such as the Message Passing Interface (MPI) [Walker 1994], a well defined and widely accepted programming interface for inter-node communication, the full automatic characterization of a large number of RC and parallel applications could be supported. However, until a common interface is finalized, manual script generation provides the most flexible alternative to cover the wide range of RC application and RC device combinations.

The final step in the application domain deals with generating scripts that represent the behavior of the target applications. The information collected during the application and core characterization steps define both the structure and values needed to construct an application script. Applications scripts are used in place of an application's code to stimulate the platform models during simulation. Application scripts represent the application as a sequence of key events, with each event represented as an individual script item. Script constructs currently include processor computation blocks, inter-node communication via MPI function calls, configuration of an RC device, and function calls to a configured RC device within the node. Blocking and nonblocking versions of MPI and RC device function calls are defined in the scripting language and supported by the platform models. Miscellaneous capabilities such as control loops are also included.

Figure 4 illustrates an example of an RC application script for a single node of a parallel application. The example script is a simplified version of the script for target detection in the HSI application defined and used in Sections 4 and 5. The script begins by configuring a single target detection (TD) core on the

```
#Sample RC Application Script

#Initiate configuration of FPGA
RC_CORECONFIG dev1 TD 3.84E6 200 1000 4500 8192 8192 0

#Begin application on node 1, receive data to be processed
MPI_Init
COMP 288.4  6858.9E6 10.9E6 3147.1E3
MPI_Recv 0 1 33554432 1 42

#Loop of processing with FPGA
STARTLOOP 100
RC_COREREQUEST dev1 TD 8192 0
COMP 151.4  3461.9E6 4973.4E3 1256.0E3
STOPLOOP
MPI_Send 1 0 2097152 1 91

#Wrap Up
COMP 9801.7  215991.0E6  602.0E6 152.2E6
MPI_Finalize
```

Fig. 4. Sample RC application script.

device named *dev1* while providing core details obtained during core characterization. In this example, the TD core is defined to occupy 4500 slices, operate at 200 MHz, and execute on 8192-byte input and output data chunks each in 1000 clock cycles. After the TD core has been configured, the script initializes the node for MPI-based communication via the MPI_Init call, followed by 288 μs of computation by the host processor. Afterwards, the node receives 32 MB of data from node 0, who must have a matching MPI_Send in its own script. The script then proceeds by defining 100 iterations of a loop, with each iteration containing an FPGA-based execution of TD followed by a block of host computation. Once the 100 iterations have completed, the node returns data to node 0 via the MPI_Send function. After the data has been returned, a block of host computation follows, after which the script is complete and the simulation terminates. It must be noted that the current scripting syntax is designed to support most of the functionalities exercised in current RC applications and is easily expandable to support other RC events that may arise as the technologies and programming interfaces mature.

3.2 Simulation Domain

The purpose of the simulation domain is to provide an environment for developing and simulating virtual prototypes of target platforms. Architectural exploration can then be performed by replacing or re-tuning components of the system model. The simulation domain is divided into three stages; model development, model calibration, and system analysis. The remainder of this subsection describes each of the stages in the simulation domain, which are illustrated by the rectangles in Figure 3.

The first step in the simulation domain is the model development stage. In this stage, models of key system components are constructed. The simulation environment used for building component and system models is Mission-Level Designer (MLD) from MLDesign Technologies [Schorcht et al. 2003]. MLD is

a graphical discrete-event simulation tool that supports modular, hierarchical designs of arbitrary systems allowing for quick development times of models with varying degrees of fidelity. Since the goal of RCSE is performance prediction of RC systems, component models do not fully incorporate mechanisms to manipulate data. In fact, the actual data is abstracted away by only considering how much data rather than what data. As a result, the models focus on the performance timing of the interactions between components exercised by the application. Focusing on system performance facilitates quick model design times (due to the reduced detail associated with each component model) and improved simulation speed (due to less processing needed to execute each component model).

An overview of key component models in the current RC model library is described here, all of which are new or significantly updated compared to the models in the original FASE library. The first component, the *RCScriptParser*, converts script commands into appropriate data structures processed by the models. The *RCMiddleware* model manages transactions between the host processor and the RC device, and also includes performance-critical overhead incurred by the device drivers. The *RCCore* model was developed as a generic black-box model, such that any hardware core could be represented. The black-box model uses the core size, input data size, output data size, and computational delay as characterized in the application domain to abstract away the data manipulation inside the core. This abstraction allows for faster simulations of systems while producing reasonably accurate results. Meanwhile, the parameters of the core that dictate performance can be scaled in order to predict the core's execution time on future generations of the RC device. *RCCore* typically resides inside an *RCFabric* model, which is used to represent the RC device and provide an interface between the RC core and the rest of the system. The *IOBus* model captures the communication delays of data transfers between hardware components sharing the bus through the application of simple bandwidth and latency calculations while also considering contention.

Once the component models have been developed, the next step in the simulation domain is the calibration of those models. In model calibration, the parameters of a component model are selected such that the model's performance matches that of the corresponding real-world technology. In general, the component models in the RC library have all been designed to be generic and parameterizable. Each of the primary component models accepts a parameter file as an input, used to tune a generic component model to match the particular device of interest. The set of parameters defined in a parameter file is unique for each type of component. For example, the parameter file for an instance of an *RCFabric* model includes parameters such as the size, reconfiguration time, number of I/O pins, and maximum clock speed of the FPGA board. By correctly setting the values of the parameter file, the generic *RCFabric* model can be calibrated to represent any number of different FPGA or other RC-based modules. In order to calibrate the component models in some cases, experimental measurements must be obtained from benchmarks that exercise the target component. Once the experimental data has been gathered, the

parameter file is filled out to match the measured data points. Various metrics such as average error or mean squared error can be applied to optimally match model behavior against the experimental data.

The current framework focuses on the components exercised when transferring data between the host processor and the RC device due to the common bottleneck that arises in this data path for many RC applications. As such, the corresponding component models often require an in-depth calibration process in order to accurately capture the performance of the data transfers. However, when attempting to calibrate the components exercised during FPGA data transfers, it can be very difficult to benchmark either the I/O bus or the drivers in isolation due to the complexity of the system and the proprietary nature of the device drivers, respectively. To resolve this problem, the performance of transactions between the host and RC device are measured as a whole (as seen by the top-level application), and the optimal set of parameters for the drivers is obtained using a parameter solver developed by the authors. The parameter solver, built in Matlab, defines bounds on the communication latency and bandwidth from the experimental data, then iterates over the bounded latency and bandwidth ranges to find the parameter combination that optimizes the communication model, within a definable granularity. By using this iterative approach, the user can choose the error metric used by the parameter solver to define the optimal parameter set. Calibration results obtained using the parameter solver, which is applied to calibrate both intra- and inter-node communication systems, are presented in Section 4.2.

The final stage of the simulation domain is system analysis. In this stage, the RC application script is processed by the system models producing performance results for each candidate system architecture. The performance results can be analyzed to identify bottlenecks in the virtually prototyped systems and conduct what-if scenarios and tradeoff analyses with respect to various design options such as algorithm decompositions and mappings and individual component specifications. The simulation results can also be used to perform multivariable analyses, where designers look to optimize their system over multiple parameters such as performance, power, cost, etc. In the next section, the steps outlined above are performed with two applications across three target systems to demonstrate the capabilities of this approach.

4. SIMULATION FRAMEWORK WALKTHROUGH

In this section, a set of case studies is presented which employs the RCSE framework described in this article. The goal of these case studies is to demonstrate the process of simulating an arbitrary reconfigurable system and application using RCSE. The results presented during these case studies are intended to serve as a validation of the framework, while illustrating its capabilities and features for modeling complex RC systems and applications. The first subsection provides details about the platforms and applications used in the following case studies. The second subsection describes details about how the applications were characterized, and shows results from model calibration for each RC platform. Finally, the results from validation experiments

are presented which serve to validate this framework, and serve as a basis for further simulative experiments in Section 5.

4.1 Experimental Setup

In this section, the applications and experimental platforms used throughout the case studies are described. The simulation results presented in this section are composed of case studies with two applications. The first set of experiments were conducted using a parameterizable benchmark which performs target detection and classification on a hyperspectral image (HSI) [Chang et al. 2004]. A hyperspectral image is a collection of 2-D images, all of the same scene but each containing a small unique portion of the overall spectrum picked up by the sensors. For the algorithm discussed in this article, HSI can be divided into three stages: calculation of the auto-correlation sample matrix (ACSM), weight computation, and target classification. The FPGA is employed to accelerate two different stages of the benchmark, ACSM calculation and target detection, which requires the FPGA to be completely reconfigured between stages. For calculation of the ACSM, the FPGA core receives a vector for each pixel in the HSI image with a number of elements equal to the number of spectral bands in the image data. The outer-product is calculated, and a running sum of the resulting matrices is constantly kept until all pixels have been processed, at which point the final matrix can be returned to the host processor. Meanwhile, target detection is primarily composed of correlating each pixel vector with a vector that represents the spectral signature of the target classification of interest. For each pixel vector sent to the FPGA, the core returns a floating-point detection value for each target classification specified at runtime.

The second application, Molecular Dynamics (MD), is the numerical simulation of the physical interactions of atoms and molecules over a given time interval. Along with standard Newtonian physics, properties such as Van Der Waals forces and electrostatic charge (among others) are calculated for each molecule at each time step with respect to the movement and the molecular structure of every particle in the system. The RC implementation of the parallel algorithm used for this case study was adapted from code provided by Oak Ridge National Laboratory (ORNL) [Alam et al. 2007]. The code implements the particle-mesh Ewald (PME) method, a biomolecular algorithm that is part of the popular Amber MD framework [Pearlman et al. 1995]. For each time step, the set of molecules in the MD system are sent to the FPGA, in order to determine the interactions between every pair of molecules. Each molecule is then passed through a set of parallel computational pipelines, where each computational pipeline determines a partial net acceleration for that molecule. As each molecule exits the computational pipelines, the partial accelerations are accumulated and immediately written to memory while the remaining molecules are still being processed. After all molecules have been processed, the results are returned by the FPGA to the host processor where the new location of each molecule is updated for the next time step. This process is repeated until the desired time interval has elapsed.

Table I. Summary of Experimental Platforms

System	Host Processor	RC Device (FPGA)	I/O Transport
H101	3.2 GHz P4 Xeon	H101-PCIXM board (Xilinx V4LX100)	133 MHz PCI-X
SRC-6	2.8 GHz P4 Xeon	MAP board (Xilinx V2-6000)	1.4 GB/s Hi-Bar
XD1000	2.2 GHz Opteron	XD1000 module (Altera EP2S180)	1.6 GB/s HT chain

The applications were simulated on three different RC experimental platforms. The three systems employed in these experiments each represent unique styles of RC system architectures, thus illustrating the versatility of RCSE. The H101 system is a cluster of Xeon servers each equipped with a Nallatech H100-series Application Accelerator connected via PCI-X. The SRC-6 system is a custom and scalable supercomputing platform from SRC Inc., which connects SRC-based CPU and FPGA modules via SRC's high-performance HiBar interconnect. Finally, the XD1000 system from Xtreme Data Inc. is a cluster of servers each featuring a dual-socket Opteron motherboard in which an FPGA-based XD1000 module is housed in one of the Opteron CPU sockets. High-speed HyperTransport (HT) links provide connections between the key components on the motherboard of the XD1000 system. Table I summarizes the key characteristics of the experimental platforms.

4.2 Characterization and Calibration Results

The first steps taken within this framework are to characterize the applications and hardware cores. The application characterization was performed using a pure software version of each application, with instrumentation code added to capture timing and memory data for the various sections of the application. As mentioned in Section 3.1, the PAPI library is employed for gathering memory access data. After running the application on each system, application scripts similar to the one illustrated in Figure 4 were manually generated from the instrumentation data for each application on each system. It should be noted that the generation of an application script for a system model does not require executing the application under study on the target system. While doing so will most likely produce the most accurate characterization data for a given scenario, there are many cases where this option is simply not available to the user. For cases where both the application and system are not available for instrumentation, an application script can be generated from scratch based on knowledge of the algorithm, or from instrumentation data gathered on a separate system.

In addition to characterizing the applications under study, the hardware cores used in each application must be characterized as well. This characterization data can come from multiple sources, such as experimental measurements, functional-level simulations of the FPGA core, or using early design calculations such as those presented by Holland et al. [2007]. In the case studies presented here, core characterization data was obtained through experimental measurements which in turn were applied to tune mathematical models for

the execution time of each core. For example, the execution time of the primary FPGA core for MD can be divided into three phases and is characterized as

$$t_{MD} = A \times N_{mol} + B \times \left(N_{mol}^2 - N_{mol}\right) + C \times N_{mol} \quad (1)$$

where N_{mol} is the number of molecules being processed. Experimental measurements were used to determine values for the constants A, B, and C on the XD1000 system by measuring the execution time for each phase of the core. The first product in Eq. (1) represents the time required to scatter each molecule from on-board memory to the computational pipelines, which must be performed before force calculations begin. The second product represents the force calculations performed between each pair of molecules, as each molecule is broadcasted one at a time to all of the computational pipelines. The final product represents the additional time required to write each result back to on-board memory after calculations of all of the molecular interactions have been completed. A similar approach is employed for the execution time for the two HSI cores, which were characterized simply as the product of an experimentally measured constant multiplied by the input image size.

For each system, a system model within the discrete-event simulation environment was constructed. Once completed, the system model parameters were then calibrated to match the performance characteristics of the target systems. For this article, the calibration results are limited to the calibration of the I/O interconnect between the RC device and host processor, and the system-area network used to connect nodes within each cluster. To obtain the experimental data used during calibration, a micro-benchmark executed on each system is used to measure transfer times for various sized data transfers between a host processor and its local FPGA, or between nodes across the data network. For all of the calibration results presented in this section, the model parameter values used to define the fitted curve were derived using the Matlab-based parameter solver cited in Section 3.2. In each case, the parameter solver is applied to minimize the mean percentage error between the modeled data and experimental performance.

Figures 5(a) and 5(b) show the experimental results versus simulation results in terms of throughput values for host processor reads from the FPGA (FPGA reads) and host processor transfers to the FPGA (FPGA writes), in an H101 node. For both of these cases, a penalty for large data transfers is automatically factored in by the parameter solver, due to the negative trend in average throughput for each set of experimental data as the transfer size grew beyond 256 KB. The modeling of this phenomenon was incorporated into the parameter solver and discrete-event models to represent the declining throughput performance sometimes experienced for extremely large data transfers, often caused by windowing or memory buffer overruns within the target device or operating system. Another behavior illustrated here is that for similar-sized transfers, the host processor in an H101 node is often able to write data to the FPGA at twice the rate the processor can read data from the FPGA. Such a disparity often has large implications on the performance of RC applications that

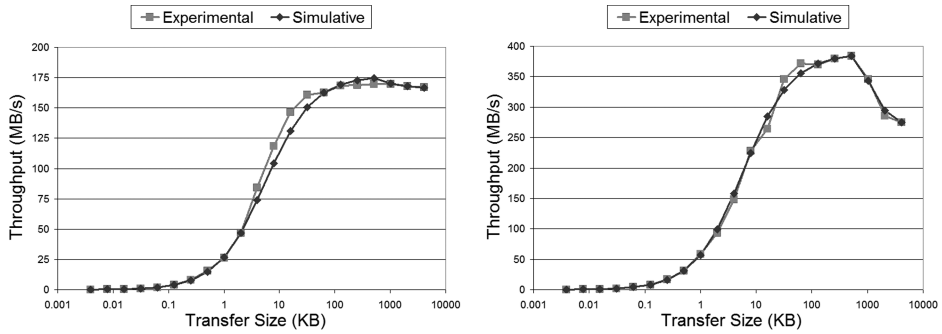


Fig. 5. Throughput for FPGA reads (a) and writes (b) on H101 system based on I/O benchmark.

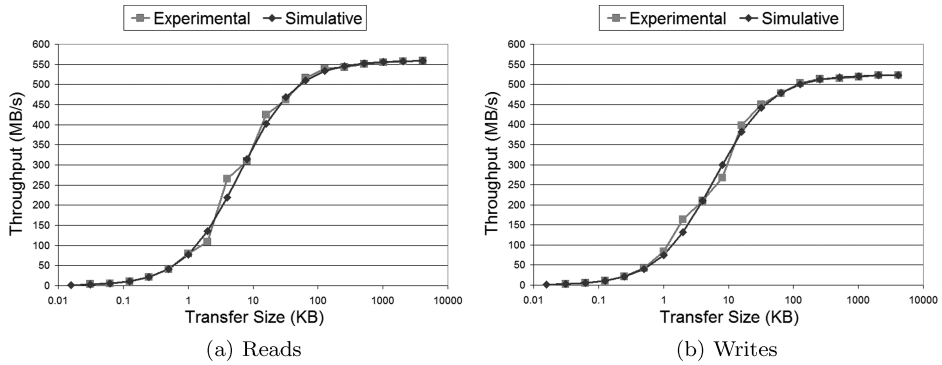
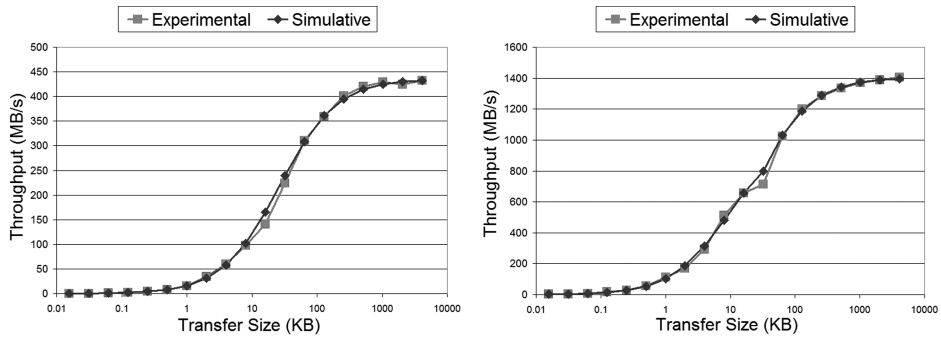


Fig. 6. Throughput for FPGA reads (a) and writes (b) on XD1000 system based on I/O benchmark.

are communication-bound, therefore it is important for the simulation framework to capture this behavior. The *RCMiddleware* models in the current framework allow the user to capture this behavior, by introducing additional delays to those experienced by simply traversing the system interconnect.

Next, Figures 6(a) and 6(b) show the calibration data for host processor reads and writes, respectively, with the FPGA's on-board memory on an XD1000 node. The I/O model curves for the XD1000 did not include a penalty for large transfers or a communication context switch. For the final system, Figures 7(a) and 7(b) show the calibration data for host processor reads and writes with the FPGA's on-board memory on the SRC-6 HiBar system, respectively. The SRC system exhibits much higher data throughput while writing to the FPGA as opposed to reading, similar to the behavior observed with the H101 node. Furthermore, unlike the first two systems, a communication context switch is used by the parameter solver and simulation models, with the switching point occurring at 16 KB on the SRC-6. A communication context switch is where the communication middleware attempts to increase efficiency by changing between internal transfer mechanisms or protocols, often based on the size of data to be transferred. A communication context switch should not to be confused



(a) Reads (b) Writes

Fig. 7. Throughput for FPGA reads (a) and writes (b) on SRC-6 system based on I/O benchmark.

with an FPGA context switch, which refers to a change in the configuration of an FPGA's reprogrammable fabric. When a communication context switch is applied, the performance of the transfer is modeled using two separate sets of latency and throughput parameters. The use of the appropriate set depends on which side of the switching point the size of the transfer data lies.

The modeled transfer times for the data sets across all three systems yielded a mean percentage error that ranged between 2.1% and 5.1% versus their experimental counterparts. These average values are encouraging, especially considering that they are mildly inflated in several cases by single erratic experimental data points that yield double-digit percent errors when compared to the calibrated model results. Additionally, it is important to note that the I/O behavior with the RC device varies significantly from system to system. In the three systems discussed here, one suffered performance penalties for large data transfers, another exhibited a communication context switch between small and large transfers, while two of the three systems allowed the host processor to write to the FPGA at a significantly higher rate than for FPGA reads of the same size. All of these factors illustrate the importance of a flexible calibration and modeling approach that can handle such disparate behaviors. Finally, the same process is employed to calibrate modeled data transfers between individual H101 or XD1000 nodes across a 20 Gb/s InfiniBand network. In this case, the model curve results in an average difference of 3.0% compared to the experimental data.

It is worth noting that users will not always be able to calibrate their component models with experimental data from benchmarking experiments, especially when considering notional platforms. In these cases, it is strongly suggested to use calibration data that can be obtained from a system most similar to the systems being simulated as a starting point. For example, a user might want to predict the performance gains expected by using a wider HyperTransport bus in the next-generation XD1000 platform. An educated approximation of the hypothetical XD1000 system is obtained by scaling the appropriate parameters of the calibrated XD1000 model to reflect the wider

Table II. Summary of Validation Results

App.	System	Exper. Runtime	Simulative Prediction	% Error	Simulation Time
HSI	H101 (1 Node)	116.12s	123.95s	6.74%	14.76s
HSI	H101 (4 Nodes)	44.16s	43.02s	2.58%	107.32s
HSI	SRC-6	86.49s	85.05s	1.66%	17.66s
MD	XD1000	4.41s	4.39s	0.45%	1.45s

bus. Delays for the middleware would be altered by the simulation models as well if the host processor or memory subsystem were updated too.

4.3 Validation Results

Once the application scripts have been generated and the system models are built and calibrated, the scripts can be fed into the system models for system analysis. In this section, the accuracy of the system models with respect to performance analysis is validated against the baseline applications. Table II summarizes the application validation results. Validation results were collected for each application/system combination where the corresponding FPGA core was available. For each of the HSI validation experiments, a 256×256 pixel input image with 1024 spectral bands was used. The simulative HSI predictions are validated against experimental executions of HSI on one and four H101 nodes, and using one CPU and one FPGA module on the SRC-6. For the MD validation experiment, a molecular system of 16,384 molecules is processed for five time steps on the XD1000 system. The maximum error between the experimental runtime and the predicted runtime across all of the validation experiments is 6.74%, where HSI is simulated on a single-node H101 system.

As previously mentioned, prediction accuracy is sacrificed in order to allow simulations to complete more quickly and without requiring a coded implementation. Validation errors under 10% produced by the RCSE are considered acceptable. Predictions within this bound provide designers with useful insight while performing strategic design-space exploration. Like many tools, the exact accuracy of the predictions will largely depend on the quality of the input provided by the user. Instances where characterization data comes from a system that is identical or very similar to the system being simulated will produce prediction results exhibiting error percentages similar to those observed in Table II. As the simulated system is varied, uncertainty is introduced into the process and the predictions become less reliable. In these cases, calibration and validation serve as an important starting point in design-space exploration, so that simulations involving hypothetical scenarios have a reliable grounding in reality. Given that the results obtained in each of the validation experiments yielded acceptable error rates for this framework, we can proceed to perform simulative analyses of these systems.

The final column of Table II reports the time measured for each simulation to execute on a 3.2-GHz Xeon processor. The simulation of MD on the XD1000 system is the quickest simulation by a substantial margin, largely because there are a very limited number of computational iterations and data transfers between system components during the application. Conversely, the simulation

of HSI on a 4-node H101 system required the longest simulation. The increased simulation time results from a number of factors, including the addition of the high-fidelity simulation of inter-node communication which is not present in single node simulations. For each of the single-node systems, the time required to simulate the application is less than the time required to execute the application by the system. These simulation times compare extremely well to traditional functional-level simulators, which typically simulate for periods of time that are orders of magnitude longer than the experimental execution times of the application.

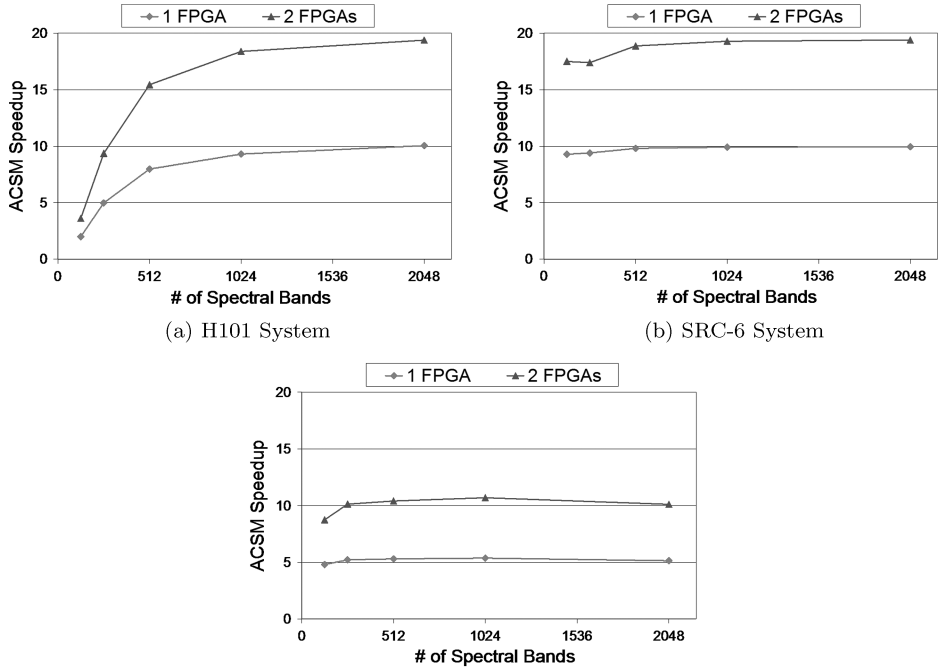
5. APPLICATION CASE STUDIES

In most cases, building large-scale or unique systems to measure performance or scalability is simply not a viable option due to time and cost constraints. In other cases, developing or porting an application to different platforms can be a costly process, and may provide disappointing results. Simulation is a very useful approach to address these issues and avoid potential productivity barriers by predicting an application's performance on existing or notional platforms that not physically available to the user. Furthermore, simulation tools are best suited for performance prediction on large-scale and complex systems where critical phenomena such as network congestion and resource contention are difficult to characterize using purely analytic or back-of-the-envelope approaches.

Using the platforms and applications previously introduced and validated in Section 4, this section presents design-space exploration case studies intended to demonstrate the usefulness of RCSE. These case studies illustrate the analysis and insight that RCSE can quickly provide to developers under the situations discussed in the previous paragraph. First, several performance studies are conducted using HSI, analyzing changes to the application parameters and device characteristics on multiple systems. Next, a set of case studies is presented analyzing the scalability of MD on a distributed XD1000 system. Finally, the RCSE framework is used to estimate the power consumption of the FPGA core design for MD.

5.1 HSI Performance Case Studies

The following simulative studies analyze impacts of system modifications to the performance of HSI. The first set of experiments investigates the performance of the RC designs for ACSM calculation as the number of spectral bands in the input image is varied. Figures. 8(a), 8(b), and 8(c) plot the speedup of the FPGA-based ACSM calculation on all three systems versus the number of spectral bands in the image, using one and two FPGA nodes. For all of the HSI case studies, speedup results are compared against the original software version of HSI running on a single 2.8-GHz Xeon processor, as found in the H101 system. On the H101 system (Figure 8(a)), the performance increases dramatically as the number of spectral bands increases from 128 to 1024. This performance improvement is primarily due to the minimization of the communication penalty for small data transfers. On the XD1000 and SRC-6 systems,



(a) H101 System

(b) SRC-6 System

(c) XD1000 System

Fig. 8. Predicted speedup of ACSM vs. spectral bands.

the predicted speedup is relatively constant for all numbers of spectral bands. Compared to the H101 system, the SRC-6 and XD1000 provide highly efficient high-speed communication between the host CPU and the FPGA, thus very little efficiency is gained by increasing the size of each data transfer.

Another observation is that the XD1000 system only obtains approximately half of the maximum speedup as the H101 and SRC-6 systems. This result is troubling considering the XD1000 contains the largest FPGA and the fastest host-FPGA interconnect of all three systems. After viewing these simulation results, a reexamination of the FPGA core design for the XD1000 was performed, which identified the single bank of SRAM on the XD1000 FPGA module as the performance bottleneck. In contrast, the FPGAs on the H101 and SRC-6 systems both contain multiple banks of SRAM, allowing multiple data values to be extracted from memory at the same time. Thus, the number of pixel vectors that can be processed in parallel is the same as the number of values that can be extracted from memory simultaneously. In this case, adding additional banks of memory to the FPGA could dramatically improve the performance of this FPGA core, and the simulation framework can be used to quickly estimate the expected performance change. In Figure 9, the performance of ACSM on the XD1000 is simulated as the number of SRAM banks on the FPGA is increased for an image with 1024 spectral bands. Approximately linear speedup is predicted as SRAM banks are added to the FPGA, since the FPGA on the XD1000 is large enough to hold the additional parallel kernels.

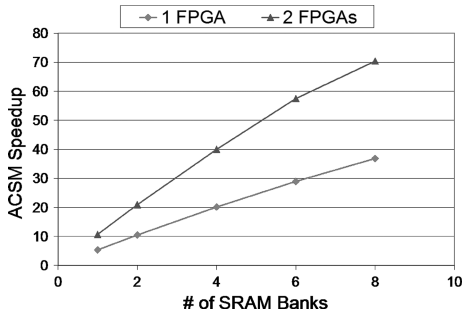
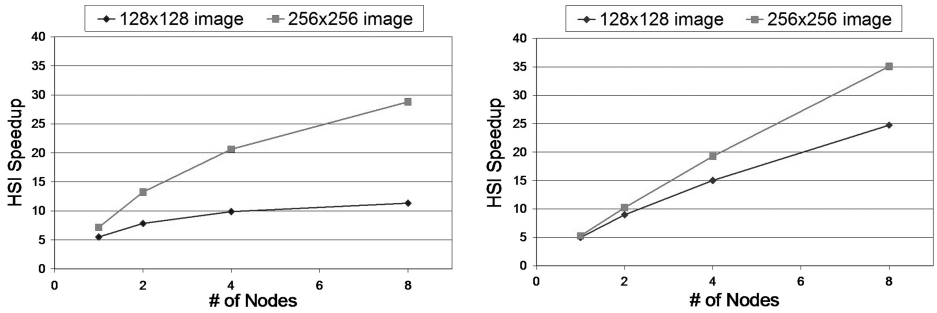


Fig. 9. Predicted speedup of ACSM vs. SRAM banks on XD1000 system.



(a) H101 system (b) XD1000 system

Fig. 10. Predicted speedup of HSI vs. system size.

The final HSI-based simulative study analyzes the performance of the entire HSI application for various system sizes with an image consisting of 1024 spectral bands. Case studies such as this one provide useful information to system designers investigating systems to efficiently perform the target application. Figures 10(a) and 10(b) show the speedup of the entire HSI application on the H101 and XD1000 systems respectively, ranging in size from one to eight nodes. For each system, the predicted speedup of the RC implementation is provided for two different input image sizes. Larger speedups are projected in every case when the image size is increased to 256×256 pixels. The larger image offers more opportunities to parallelize computations on the FPGA, while increasing the ratio of computation to communication for the entire application, causing the larger speedups over the software baseline. On the H101 system, the performance of processing the 128×128 image scales very poorly with system size. The poor scaling is due to a smaller ratio of computation to communication combined with the high cost of communication between the CPU and FPGA in each node.

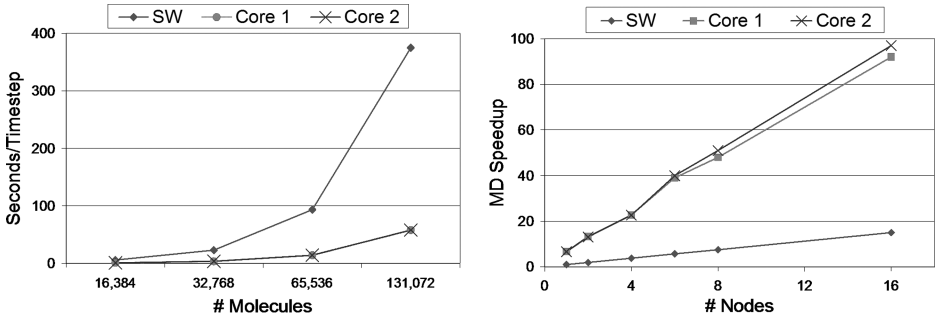
5.2 MD Performance Case Studies

For each of the simulative studies in this section, the RCSE framework is employed to analyze the scalability of MD in terms of the size of the application

(i.e., the number of molecules being processed by the algorithm) and the size of the system. Furthermore, the framework is used to analyze two different parallelization strategies for distributed execution of MD, which requires two slightly different FPGA core designs. The first strategy (Core 1) applies the baseline MD FPGA core design described in Section 4.1 and used for validation in Section 4.3. The application is distributed across multiple nodes by assigning each node to calculate partial accelerations for all molecules in the system, which are then returned to the root processing node and summed for each time step. Since the partial accelerations for each molecule are known once the molecule has finished passing through all of the computational pipelines within the FPGA, the value can be immediately written to on-board memory while the remaining molecules are passing through the pipelines. For this reason, we can assign a value of zero to the constant C in Eq. (1) used to characterize the execution time of the MD core. The second parallelization strategy (Core 2) computes the full accelerations for only a portion of the molecules at each node. The computational structure of Core 2 reduces the amount of data returned by each node at the end of the time step, since each node is only responsible for returning acceleration data for a fraction of the total molecules. Conversely, this design increases the execution time on the FPGA for each time step, since the total acceleration for each molecule is stored within the pipelines and is not known until after every molecule has passed through the computational pipeline. Since none of results may be written back to on-board memory until every force calculation has been completed, the constant C in Eq. (1) must be set equal to A for Core 2, as the time required to write every molecule back to SRAM is approximately equivalent to the time required to read each molecule from SRAM.

Figure 11(a) shows the simulation results for the execution time of MD on a single XD1000 node versus the number of molecules in the application. The results in Figure 11(a) illustrate that for any number of molecules, the relative speedup between the FPGA-accelerated versions and the software baseline is relatively constant. Furthermore, it is interesting to note the very close proximity in performance between Core 1 and Core 2. While Core 1 always executes slightly faster than Core 2, the difference is practically negligible for a single-node XD1000 system, which becomes important when considering distributed versions of this application in the following experiment.

The next experiment analyzes the projected speedup of MD parallelized across a distributed XD1000 system. A data set with 131,072 molecules is used here since larger data sets will benefit more greatly from distributed execution, while also illustrating the effectiveness of RCSE to analyze applications that have been scaled beyond what has been previously performed experimentally. Figure 11(b) shows the speedup of the distributed MD application on system sizes ranging from one to 16 nodes. For this case study, speedup results are compared against the original software version of MD running on a single 2.2 GHz Opteron processor, as found in the XD1000 system. Due to the very high ratio of computation vs. communication, the performance of the parallelized software and FPGA-acceleration applications scale almost linearly with the number of nodes. As the size of the system grows to 8 nodes and beyond,



(a) Execution time vs. molecules (b) Speedup vs. system size

Fig. 11. Predicted performance of MD on XD1000 system.

Core 2 begins to outperform Core 1 by a noticeable margin. For larger system sizes, Core 2 becomes more efficient due to the reduced amount of data returned to the root node at the end of each time step. The difference becomes more pronounced as the system size grows, since a larger number of nodes are simultaneously communicating with the root node at the beginning and end of each time step. Core 2 is designed to alleviate that network contention, at the expense of decreased performance within the FPGA. This data can be used before the design (or coding) stage to decide which core design is ideal for their scenario, depending on the size and type of system.

5.3 MD Power Case Study

The final case study illustrates the power modeling component of the RCSE framework. Due to the complexities of modeling and estimating power consumption, the approach employed here only attempts to roughly estimate the power consumption of RC systems. Therefore, higher error rates observed from power consumption estimates will be tolerated. In this case study, the power consumption of a single FPGA core for MD executing on the XD1000 is estimated. To begin, this framework assumes a common model for the total power consumption of FPGAs, which is

$$P_{est} = P_{IO} + P_{stat} + P_{int} \tag{2}$$

where P_{IO} is the power consumption of the FPGA from I/O activity, P_{stat} is the static power consumption caused by leakage current, and P_{int} is the internal (or dynamic) power consumption of the FPGA. The static power consumption for an FPGA is available from the device data sheet, and becomes a parameter of the FPGA device model. P_{IO} is the product of the number of I/O pins exercised by the core (N_{IO}) and the power of each pin (which can be derived from the capacitance and switching frequency of each pin). To estimate the internal power consumption of the FPGA, we apply the following equation [Weiss et al. 2000]:

$$P_{int} = V_{core} \times K_p \times f_{max} \times U_{res} \times T_{log} \tag{3}$$

Table III. Summary of Key Power Model Parameters

Parameter Name	Symbol	Value	Source of Value	Model Location
I/O Utilization	N_{IO}	485 pins	Core compilation report	Script (core param)
Resource Utilization	U_{res}	92%	Core compilation report	Script (core param)
Clock frequency	f_{max}	100 MHz	Core compilation report	Script (core param)
Core Voltage	V_{core}	2.5 V	FPGA data sheet	FPGA device model
Device Constant	K_p	1.67E-6	FPGA data sheet	FPGA device model

In Eq. (3), V_{core} represents the voltage level of the FPGA core, f_{max} is the maximum operating frequency of the core, U_{res} represents the resource utilization of the core, T_{log} is the average transistor toggle rate of the reconfigurable logic, and K_p is a device technology constant. Values for V_{core}, U_{res} and f_{max} are core-specific and obtained during core characterization. The toggle rate for T_{log} is difficult to determine without a functional-level simulation, but values ranging between 0.15 and 0.2 are commonly used as estimates of this parameter. Finally, K_p is a device constant that is specifically defined in datasheets for Xilinx FPGAs, while Altera datasheets typically provide multiple constants relating to individual resources on the chip that are averaged in RCSE to obtain a final estimate for K_p [Altera 2001].

Table III summarizes the key parameters used by the power estimation equations, including the source of the value and where it is specified within the RCSE framework. To test the accuracy of our power model, the estimated FPGA power consumption is compared to the value generated by Altera's PowerPlay tool. PowerPlay is a tool within Altera's Quartus-II FPGA development environment used to obtain accurate power estimates of an FPGA design based on a functional simulation of the final core (similar to the XPower Analyzer for Xilinx FPGAs). Using these values with Eq. (3), an estimated power consumption of 10.25 W is generated, compared to an estimate of 13.30 W from PowerPlay, resulting in a 3.05 W difference and a 22.9% error. In this case study, the power estimation error is primarily caused by the derived value of U_{res}. The value of U_{res} is calculated to be the average utilization of slices, BRAM, and DSP resources by the core. Unfortunately, this average does not provide an accurate representation of the core for power estimation purposes under scenarios where the utilization of one resource is very high compared to the other two (e.g., the MD core has a very high memory utilization compared to slice and DSP utilization).

Despite these limitations, results from early power prediction can provide useful data to RC designers. Designers can obtain rough determinations regarding the satisfaction of system power constraints from using their FPGA design, or whether significant power savings will be achieved versus an alternative device technology. Furthermore, this framework can track the total energy consumption of the FPGA throughout the lifetime of the application. The discrete-event device models track the total time the FPGA core spends performing active computation compared to the entire length of the application. By applying the total FPGA power consumption estimate to active computation periods and the static power consumption estimate to the remaining time, a prediction of the total energy consumption for the FPGA

can be calculated. Meanwhile, estimating the power consumption of other components such as microprocessors, memories [Hicks et al. 1997], etc. is well studied. Data sheet parameters or existing high-level models based on architectural parameters [Macii et al. 1998] can be used to obtain component power approximations. Combining each component power consumption estimate with the FPGA power consumption calculated by the simulation models will provide an approximation of total system power consumption.

6. CONCLUSIONS

Thanks to an architectural reformation in computing that favors the use of device technologies exploiting explicit parallelism to achieve greater performance, reconfigurable devices such as FPGAs are becoming an increasingly important option for accelerating applications in high-performance and/or embedded computing, from satellites to supercomputers. Unfortunately, RC devices cannot deliver desirable speedup with all applications and their mappings to a particular system. Applications need to be carefully designed to effectively exploit the parallelism of the underlying device technologies. As a result, an application reformation is taking place where explicit parallelism in applications is required to take advantage of emerging device technologies. To alleviate the cost and difficulty of RC application and system development under these two reformations, simulation early in the development process can provide a relatively quick and cost-effective means to perform design-space exploration involving applications and systems that incorporate RC devices. However, as the architectural and application reformations continue, and RC systems and applications scale in size and complexity, simulation times need to be managed in order to provide researchers and engineers a timely method of exploring the increasingly growing design space.

In this article, a framework for rapid simulations of applications on reconfigurable systems was introduced, called the Reconfigurable Computing Simulation Environment, or RCSE. This framework divides the modeling process into two domains, the application and simulation domains, which provides a methodology for mapping arbitrary applications to a variety of RC systems facilitating rapid in-depth performance projections and analyses. The complexity of simulating RC systems is managed by abstracting away the details of computation. For the case of the RC device, the execution of a core is replaced with an appropriate black-box model to represent the performance of the RC device. This methodology was illustrated with a set of simulative case studies using two applications and three different RC systems. Validation results showed our system models could predict the overall performance of the application within a modest range of error, ranging from 0.45% to 6.74% in the case studies presented in this article. After the application validation tests, a set of simulative studies were presented to demonstrate the ability of the RCSE framework to identify bottlenecks and trends in performance as various parameters pertaining to the application and system were varied. For example, the MD case studies showed how performance gains on a distributed XD1000 system with more than four nodes could be obtained using a slightly modified

FPGA core design. Finally, a case study was presented to illustrate the support for power modeling of RC devices in our methodology using the MD application core, which produced an FPGA power consumption estimate with a 22.9% error.

Future work in this area will include expanding the model library to support additional new and emerging RC systems. Case studies featuring systems and applications employing partial reconfigurability will be performed as well. Furthermore, a new formulation language is under development allowing scientists to efficiently create an abstract model of their application and system. The abstract model can then be fed to RCSE to complete the formulation process. Finally, a detailed core modeler is being designed to allow users to define the structure of their FPGA core design through a block-oriented interface within the simulation environment. The detailed core modeler would then analyze the core design after being mapped onto an RC device model. The device model interacts with the core design by providing micro-benchmark data that characterizes specific behaviors within the device, such as memory access delays and clock degradation as the chip is fully utilized.

ACKNOWLEDGMENTS

The authors gratefully acknowledge vendor equipment and/or tools provided by Xlinx, Altera, MLDesign Technologies, SRC, Nallatech, and the George Washington University that helped make this work possible.

REFERENCES

ALAM, S., AGRAWAL, P., SMITH, M., VETTER, J., AND CALIGA, D. 2007. Using FPGA devices to accelerate biomolecular simulations. *IEEE Computer 39,* 4, 66–73.

ALTERA. 2001. Evaluating power for altera devices. Application Note 74 version 3.1.

ANDERSON, J. H. AND NAJIM, F. N. 2004. Power estimation techniques for FPGAs. *IEEE Trans. VLSI Syst. 12,* 10, 1015–1027.

BAKSHI, A., PRASANNA, V. K., AND LEDECZI, A. 2001. Milan: A model based integrated simulation framework for design of embedded systems. In *Proceedings of the ACM SIGPLAN workshop on Languages, Compilers and Tools for Embedded Systems (LCTES'01)*. ACM, 82–93.

BONDALAPATI, K., AND PRASANNA, V. K. 2002. Reconfigurable computing systems. *Proc. IEEE 90,* 7, 1201–1217.

BONDALAPATI, K. K. 2001. Modeling and mapping for dynamically reconfigurable hybrid architectures. Ph.D. thesis, University of Southern California, Los Angeles, CA.

BROWNE, S., DONGARRA, J., GARNER, N., HO, G., AND MUCCI, P. 2000. A portable programming interface for performance evaluation on modern processors. *Int. J. High Perf. Appli. 14,* 3, 189–204.

BURGER, D., AND AUSTIN, T. M. 1997. The simpleScalar tool set, version 2.0. *ACM SIGARCH Comput. Architect. News 25,* 3, 13–25.

CHANG, C.-I., REN, H., AND CHIANG, S.-S. 2004. Real-time processing algorithm for target detection and classification in hyperspectral imagery. *IEEE Trans. Geosci. Remote Sensing 39,* 4, 760–768.

ENZLER, R., PLESSL, C., AND PLATZNER, M. 2005. System-level performance evaluation of reconfigurable processors. *Microprocess. Microsyst. 29,* 2-3, 63–75. (Special Issue on FPGA Tools and Techniques).

FU, W., AND COMPTON, K. 2006. A simulation platform for reconfigurable computing research. In *Proceedings of the International Conference on Field Programmable Logic and Applications. (FPL'06)*. 1–7.

GARCIA, P., COMPTON, K., SCHULTE, M., BLEM, E., AND FU, W. 2006. An overview of reconfigurable hardware in embedded systems. *EURASIP J. Embed. Syst.*, 1–19.

GROBELNY, E., BUENO, D., TROXEL, I., GEORGE, A., AND VETTER, J. 2007. FASE: A framework for scalable performance prediction of HPC systems and applications. *Simulation: Trans. Soc. Model. Simul. Int. 83,* 10, 721–745.

HAMERLY, G., PERELMAN, E., LAU, J., AND CALDER, B. 2005. Simpoint 3.0: Faster and more flexible program phase analysis. *J. Instruct.-Level Paral. 7*, 1–28.

HICKS, P., WALNOCK, M., AND OWENS, R. M. 1997. Analysis of power consumption in memory hierarchies. In *Proceedings of International Symposium on Low Power Electronics and Design*. ACM, 239–242.

HOLLAND, B., NAGARAJAN, K., CONGER, C., JACONS, A., AND GEORGE, A. 2007. RAT: A methodology for predicting performance in application design migration to FPGAs. In *Proceedings of High-Performance Reconfigurable Computing Technologies and Apps Workshop (HPRTCA)*. 1–10.

LAFAGE, T., AND SEZNEC, A. 2001. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In *Workload Characterization of Emerging Computer Applications*, Kluwer International Series in Engineering and Computer Science Series, Kluwer Academic Publishers, 145–163.

LI, F., CHEN, D., HE, L., AND CONG, J. 2003. Architecture evaluation for power-efficient FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 175–184.

MACII, E., PEDRAM, M., AND SOMENZI, F. 1998. High-level power modeling, estimation, and optimization. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst. 17,* 11, 1061–1079.

MOHANTY, S., AND PRASANNA, V. K. 2007. A model-based extensible framework for efficient applicatin design using FPGA. *ACM Trans. Des. Autom. Electr. Syst. 12,* 2, 13.

MOHANTY, S., PRASANNA, V. K., NEEMA, S., AND DAVIS, J. 2002. Rapid design-space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation. In *: Proceedings of the joint conference on Languages, Compilers and Tools for Embedded Systems (LCTES/SCOPES'02)*. ACM, New York, 18–27.

PEARLMAN, D. A., CASE, D. A., CALDWELL, J. W., ROSS, W. S., CHEATHAM III, T. E., DEBOLT, S., FERGUSON, D., SEIBEL, G., AND KOLLMAN, P. 1995. AMBER, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules. *Comput. Phys. Commun. 91,* 1-3, 1–41.

PLLANA, S., AND FAHRINGER, T. 2005. Performance prophet: A performance modeling and prediction tool for parallel and distributed programs. In *Proceedings of the Internation Conference on Parallel Processing*. 509–516.

POON, K. K., WILTON, S. J., AND YAN, A. 2005. A detailed power model for field-programmable gate arrays. *ACM Trans. Des. Autom. Elect. Syst. 10,* 2, 279–302.

SCHORCHT, G., TROXEL, I., FARHANIGAN, K., UNGER, P., ZINN, D., MICK, C., GEORGE, A. D., AND SALZWEDEL, H. 2003. System-level simulation modeling with mldesigner. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 207–212.

SMITH, M. C. AND PETERSON, G. D. 2002. Analytical modeling for high-peroformance recongifurable computers. In *Proceedings of the SCS International Symposium on Performance Evaluation of Computer and Telecommunications Systems (SPECTS)*.

SNAVELY, A., CARRINGTON, L., WOLTER, N., LABARTA, J., BADIA, R., AND PURKAYASTHA, A. 2002. A framework for performance modeling and prediction. In *Proceedings of the ACM/IEEE SC2002 Conference*. 21–21.

STEFFEN, C. P. 2007. Parametrization of algorithms and fpga accelerators to predict performance. In *Proceedings of the Reconfigurable System Summer Institute (RSSI)*. 17–20.

TESSIER, R., AND BURLESON, W. 2001. Recongifugrable computing for digital signal processing: A survey. *J. VLSI Signal Proces. 28,* 1-2, 7–27.

UHLIG, R. A., AND MUDGE, T. N. 1997. Trace-driven memory simulation: A survey. *ACM Comput. Surv. 29,* 2, 128–170.

WALKER, D. W. 1994. The design of a standard message passing interface for distributed memory concurrent computers. *Paral. Comput. 20,* 4, 657–673.

WEISS, K., OETKER, C., KATCHAN, I., STECKSTOR, T., AND ROSENSTIEL, W. 2000. Power estimation approach for SRAM-based FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 195–202.

WUNDERLICH, R. E., WENISCH, T. F., FALSAFI, B., AND HOE, J. C. 2006. Statistical sampling of microarchitecture simulation. *ACM Trans. Mod. Comput. Simul. 16,* 3, 197–224.

Received August 2008; revised February 2009; accepted April 2009