Simulation Framework for Performance Prediction in the Engineering of RC Systems and Applications

Eric Grobelny, Casey Reardon, Adam Jacobs, Alan D. George
NSF Center for High-Performance Reconfigurable Computing (CHREC)
HCS Research Lab, ECE Department, University of Florida
Telephone: (352) 392–5225 Fax: (352) 392–8671

Abstract*—Reconfigurable computing (RC) is rapidly emerging as a promising technology for the future of high-performance 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. This approach 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 systems in a simulation environment. 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. This paper presents the design of a novel new framework for simulative performance prediction, along with a single-node case study performed with a synthetic-aperture radar application to provide validation results and a component tradeoff analysis, illustrating the effectiveness of our approach at the node level.

I. Introduction

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. The underlying device technology enabling this new paradigm of computing is field-programmable hardware, such as the field-programmable gate array or FPGA. These programmable logic devices feature many thousands of logic cells as building blocks that can be quickly configured and interconnected to form application-specific custom logic. RC extends the fields of large-scale and embedded high-performance computing by incorporating and dynamically reconfiguring these devices at run-time to accelerate operations that would otherwise be performed in software. Hybrid systems of microprocessors and FPGAs can leverage system-level concepts from conventional high-performance computing while accommodating hardware reconfigurability.

While these hybrid systems offer the potential for large performance improvements over traditional systems, the introduction of reconfigurable devices can dramatically increase the design complexity of such systems. 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 can make targeting applications to a specific RC system difficult and daunting. Simulation provides a means of predicting the performance and bottlenecks of applications running on numerous system configurations, for the purpose of studying design tradeoffs. The resulting analyses can provide useful data before investing significant amounts of time and resources on the development of a particular solution.

In this paper, we present a framework for simulating applications on reconfigurable computing systems that balances both speed and fidelity. By providing this balance, our framework can provide a broad range of timely and meaningful prediction analyses for a given reconfigurable application or platform. With the appropriate models and calibration data, numerous existing and future systems and applications can be efficiently simulated and analyzed. The remainder of this paper is organized as follows. Section II presents related modeling and simulation research, for both reconfigurable and high-performance computing. Section III provides an overview of our simulation approach and methodology for performance prediction of reconfigurable computing systems. In Section IV, the results of our node-level architecture case study are presented to validate and help illustrate the capabilities of our simulation framework. Finally, the conclusions of this paper are summarized in Section V.

II. Related research

The modeling and performance prediction of RC devices can be broken into two primary categories: device-level and system-level. Device-level modeling is normally handled using electronic design toolkits such as ActiveHDL and ModelSim provided by vendors such as Aldec and Mentor Graphics, respectively. These tools use the HDL languages employed to design the corresponding functional cores and only target the performance of the specific configurable device or family of devices rather than the entire computational subsystem exercised when using the core with a real application. Although accurate, the tools are device-specific and have the potential to take hours or even days to execute and do not address critical performance issues in other components of a subsystem or system, such as the I/O bus to which the device attaches.

A number of research projects attempt to predict the performance of RC devices through the use of analytical models. In [1], analytical models were developed to analyze the performance of heterogeneous workstations outfitted with RC devices. The research specifically addresses performance issues dealing with load balancing using synchronous iterative applications at both the node and device level. The models showed reasonable accuracy, however significant effort is required for each application under study. In [2], RC models are developed to predict the performance of vision algorithms. The models incorporate the traditional configurable computing system with a configurable device being used as an offload engine by a host processor with implementation details abstracted away. The models in their project address performance prediction in a general sense with numerous architectural and core variations possible. Although a comprehensive model is presented, model validation is not provided and, again, the models are fairly complex and difficult to create for mapping applications to specific architectures.

III. Simulation Framework

This paper presents a novel new simulation framework for performance prediction of reconfigurable computing systems that balances speed and accuracy. The goal of the project is to develop a framework that supports the analyses of numerous variations of RC architectures at both the node and system levels and application mappings to them. These architectures include clusters of RC nodes, supercomputers with RC devices, embedded systems, and other current and emerging configurations. As a result of the wide design space possible for current and future RC systems, we assume a generic system model as shown in Fig. 1. The generic system consists of one or more nodes interconnected by some high-speed interconnect. Each RC node includes a host processor that typically handles general computing tasks while offloading specialized tasks to the corresponding RC device. The architecture is general enough, however, to support future systems that incorporate RC devices that act completely independent of the host processor. The RC devices can be attached to various local interconnect technologies within the node, including a peripheral bus or system bus or switching fabric. By supporting the generic architecture described in Fig. 1, this simulation framework allows for comprehensive analyses of arbitrary serial and parallel RC applications targeted for the wide range of reconfigurable systems. This node architecture can be tailored for prototyping of a variety of system platforms. For example, the Local Interconnect(s) cloud shown in Fig. 1 in some cases may represent one level of interconnect (e.g. a direct HyperTransport connection between CPUs, FPGAs, and main memory) and in other cases a hierarchy of interconnects (e.g. CPUs and main memory residing on the front-side interconnect such as HyperTransport, with FPGAs attached via a bridge to an I/O interconnect such as PCI-Express).

In order to tackle the problem of optimally mapping arbitrary applications to specific target architectures, the modeling framework is split into two separate domains - the application

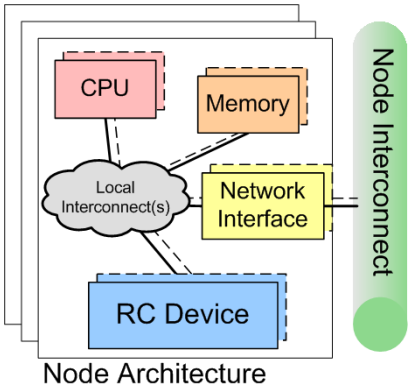


Fig. 1. Generic RC System Architecture

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. The overall structure of our simulation framework, and the key steps within each domain, are illustrated in Fig. 2. The remainder of this section provides details and examples on the procedures used in the application and simulation domains.

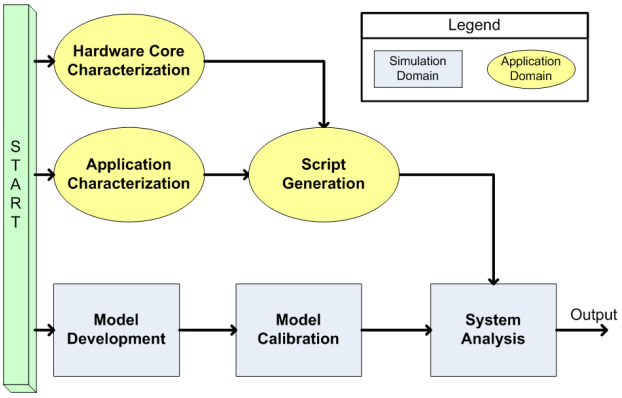


Fig. 2. RC framework diagram

A. Application domain

The purpose of the application domain is to collect characterization data on a selected application that captures its inherent behavior, with the intent of creating some form of stimulus data for the simulation models. The steps that make up the application domain are shown by the ovals in Fig. 2. Hardware core characterization defines the behavior of the kernel or function to be performed within the RC device. The computation time for an RC core can be obtained from two sources. The first source uses experimentally measured delays from a hardware core implementation, while the second uses delays supplied from simulations of the hardware design from vendor-supplied tools. Both methods provide reasonably

accurate results assuming deterministic behavior of the RC devices. Computation time is not the only parameter required to characterize a hardware core. Other key parameters include core size, input data size, and output data size. The core size parameter is important in order to manage how many cores can fit onto a single RC device. This management capability enables us 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. It also supports the modeling of partial reconfiguration by allowing cores to be exchanged within the RC device during the simulation. However, an in-depth study of this technique was 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 the 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 performance of the target application. The events currently supported within this framework include the computation conducted by the host processor, the computation executed in the RC device, and the communication between RC nodes. A specific sequence of these three events can be used to 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. The framework 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. Presently, the incorporation of RC events into scripts must be performed manually, since no common standard exists that defines interactions with RC devices. In the future, the adoption of a standard RC programming interface, such as that being explored by the USURP project [3] or the U-APPREQ work group in the OpenFPGA consortium [4] would allow the automatic characterization of these events. However, until a common standard is finalized, manual script generation provides the most flexible alternative to cover the wide range of RC application and RC device combinations. When a common standard is adopted, automatic generation of application scripts within our scripting syntax can be implemented. The proceeding paragraph presents specific details on the scripting language developed for stimulating RC simulation models.

The final step in the application domain deals with generating scripts that represent the behaviors of the target applications. The information collected during the application and core characterization steps define both the structure and the values needed to construct scripts that accurately exercise the computational subsystems as if the actual application was executing on the target system. The framework incorporates a custom scripting syntax to facilitate script construction. Fig. 3 illustrates a sample RC script. The script begins by initializing the RC device with key parameters such as device size and maximum clock frequency such that the necessary models can correctly manage the device. The next line configures a single FFT core on the device while providing core details obtained during the hardware core characterization step. In this case, the FFT core is defined to occupy 2500 slices of the 10000-slice device, operate at 150 MHz, and execute on 1024-byte data chunks in 650 clock cycles. After the fabric and the FFT core have been configured, the script starts executing the computation section of the application beginning with 1.12 seconds of computation at the host processor. The script then proceeds by defining 100 iterations of host computation followed by the execution of an FFT. Once the 100 iterations have completed, the script is complete and the simulation will terminate. It must be noted that the current scripting syntax was designed to support most of the functionalities used in current RC applications and is easily expandable to support other RC events that may arise as the technologies and programming interfaces mature.

```
#Sample RC Script

#Setup RC device
#RC_INITFABRIC <id> <total slices> <max frequency>
RC_INITFABRIC 1 10000 2000

#Configure RC device with FFT core
#RC_CORECONFIG <id> <name> <bitmap size> <clock frequency>
#         <clock cycles> <slices> <input data size> <output data size>
#         <overhead clock cycles> <delay>
RC_CORECONFIG 1 FFT 500 150 650 2500 1024 1024 50 25

#Host processor compute block – 1.12 seconds
COMP 1.12E6

#Define loop with 100 iterations
RC_STARTLOOP 100

#Compute at host processor for 450 microseconds
COMP 450

#Send 8192 bytes of data to the FFT core, blocking
RC_COREREQUEST 1 FFT 8192 0

#End loop
RC_STOPLOOP

#DONE
```

Fig. 3. Sample RC Script

B. Simulation domain

Now that the application domain has been described, we can transition to the simulation domain. The purpose of the simulation domain in our framework is to provide an environment for developing and simulating virtual prototypes of candidate systems. The steps that make up the simulation domain are illustrated by the rectangles in Fig. 2. The first step 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 [5]. 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 our framework is performance prediction of RC systems, component models do not incorporate the actual 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 the key component models in the current RC model library is described below. The first component, the *RCScriptParser*, converts script commands into appropriate data structures used 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. The *SimpleBus* model captures the communication delays of data transfers between hardware components sharing the bus through the use of simple bandwidth and latency calculations while also considering contention. From these models, higher-level complex components such as the host processor or RC device can be created.

Once the component models have been developed, the next step in the simulation domain is the calibration of those models. In the model calibration stage, the parameters of a component model are selected such that the model's performance matches that of the corresponding real-world technology. In order to calibrate the component models, experimental measurements must be obtained from benchmarks that exercise the target component. Once the experimental data has been gathered, the model parameters are tuned to match the measured data points. Various metrics such as average error or mean squared error can be used to match model results 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 at this data path in many RC applications. As such, the corresponding component models, specifically the local interconnect (e.g. a bus) and RC device drivers, require an in-depth calibration process in order to accurately capture the performance of the data transfers. However, in this instance, it is very difficult to benchmark either the 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, such that the measurements correspond to a interconnect technology (e.g. PCI-X, PCI-Express, HyperTransport) and the optimal set of parameters for the drivers is obtained using a parameter solver developed by the authors. The parameter solver defines bounds on the RC driver's latency and bandwidth from the experimental data, then iterates over each range to find the parameter combination that optimizes the chosen error metric, within a definable granularity. Furthermore, a penalty factor is included to represent the declining throughput performance sometimes experienced for extremely large data transfers, possibly caused by memory buffer overruns within the target device or operating system. The implementation details of the solver are outside the scope of this paper, but calibration results using the parameter solver are presented in Section IV.

The final stage of the simulation domain is the 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 used 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 performance. In the next section, the steps outlined above are performed with an application and two target systems to demonstrate the capabilities of this approach.

IV. Case study: Single-node system

In this section, the simulation framework described in the previous section is demonstrated and validated, using a node-level case study. The goal of this case study is to demonstrate the steps involved in the process of simulating an arbitrary reconfigurable system and application within this framework. The results presented during this case study are intended to serve as a validation of the framework, while later illustrating its capabilities and features.

A set of experiments were conducted using two variations of a single-node system. Table I summarizes the two systems used in these experiments. For the validation of each system, experiments were conducted using a synthetic aperture radar application (SAR). SAR is a high-resolution, broad-imaging application used for reconnaissance, surveillance, targeting, navigation, and other operations requiring highly detailed, terrain-structural information. The imaging process iterates over four stages: range compression, azimuth transform, range cell correction, and azimuth compression [6]. The SAR application under study uses data with 6,144 ranges and 4096 azimuths per iteration, or patch. Multiple patches can be combined to resolve larger images.

The 1D-FFT is the main computational component of the range compression, azimuth transform, and azimuth com-

TABLE I
SINGLE-NODE SYSTEM DETAILS

System	Host Processor	System Memory	RC Device	I/O Bus
Delta	P4 Xeon 3.2 GHz	2 GB PC2700 DDR-SDRAM	Nallatech H101-PCIXM board (Xilinx V4LX100)	133 MHz PCI-X
Kappa	P4 Xeon 2.4 GHz	1 GB PC2100 DDR-SDRAM	Nallatech BenNUEY board (Xilinx V2Pro-50)	66 MHz PCI

pression stages. In order to support all of these stages, the hardware implementation of the FFT uses a pipelined radix-2 structure that is capable of performing power-of-2 FFTs (or inverse FFT) up to 8,192 elements in length. Each element processed by the FFT is a 16-bit fixed-point complex number (32 bits total). Transforms performed along the range dimension use 8,192-point FFTs, while those performed along the azimuth dimension use 4,096-point FFTs. SAR was chosen for this case study because it includes a well-defined computational kernel (i.e. the FFT) that is sometimes a prime candidate for implementation inside the FPGA. The remainder of this section illustrates each step of the framework for validating and predicting the performance of SAR.

A. SAR characterization

The first steps taken to validate the framework were to characterize the application and hardware core. The application characterization was originally performed using a pure software version of SAR, with instrumentation code added to capture timing data for the various sections of the application. From the timing data, as well as an understanding of the algorithm, we identified the FFT as a repeatedly used function that consumes a significant portion of the total execution time. Thus, corresponding code was added to the SAR code in order to produce an RC version of the algorithm and an FFT hardware core was obtained and characterized through experimental measurements.

Once the characterization data was obtained, a script of the SAR algorithm was manually written to incorporate measured host processor computation as well as the necessary events that offload the FFT tasks to the RC device. For each system, two versions of the SAR script were generated, the *SW-Based Timing* script and the *RC-Based Timing* script. Both of these scripts represent an implementation of the SAR algorithm that uses an FPGA to perform the FFT operations. The overall structure of each script version is the same, since they contain the same sequence of computation blocks in the host processor and FFT operations using the FPGA. Where these two script versions differ is in the length of time spent within each host processor computation block. For the *SW-Based Timing* script the timing data for each of the host processor compute blocks is measured from a pure software version of SAR. The *RC-Based Timing* script derives host processor computation block timing data from the RC version of SAR that incorporates code to offload FFT operations to the system's FPGA device. A comparison between both script versions was performed to study possible variations in performance predictions due to differences in host computation times between the two versions of the SAR application.

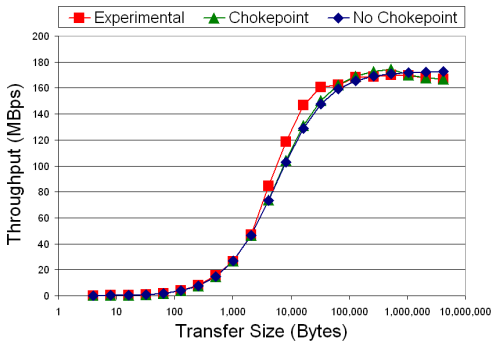
B. Model development and calibration

In parallel with the characterization processes, system models for the single-node testbeds were built. The primary components of the RC node model include a host CPU, RC device, and bus model. The processor model incorporates the *RCScriptParser* and *RCMiddleware* models described in Section III. The RC device model uses the *RCCore* model which was configured to capture the behavior of the FFT used in the SAR application. Finally, the *SimpleBus* model was used as the interconnect between the host CPU and the RC device.

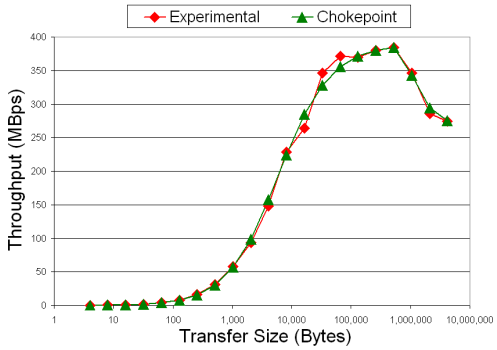
Once completed, the system model parameters were then calibrated to match the performance characteristics of the target systems. For this case study, we focused our calibration efforts on the interconnect between the host processor and RC device. Figs. 4a and 4b show the experimental results versus simulation results in terms of throughput values for device reads and writes, respectively, on the Delta system, while Figs. 5a and 5b show the same on the Kappa system. The *No Chokepoint* curve represents the resulting calibration data points from the parameter solver when it does not incorporate the detection of transfer penalties for large transfers, while the *Chokepoint* data does consider these penalties. A *No Chokepoint* curve was not included in the Delta write results (Fig. 4b), since the existence of a hardware overflow was obvious and severe, thus the *Chokepoint* curve was the only logical fit to the experimental data. Similarly, a *Chokepoint* curve was not included in the Kappa write results (Fig. 5b), since a hardware threshold was not observed in the experimental data. The simulation results for the data sets in these four charts yielded a mean percent error that ranged between 2.1% and 5.1% versus their experimental counterparts. Those average values are very encouraging, especially considering that they are mildly inflated by single erratic experimental data points that yield double-digit percent errors when compared to the calibrated model results. Additionally, the data from the Kappa varies widely from the Delta system, illustrating the importance of a flexible calibration approach that can handle such disparate behaviors. The accurate and flexible calibration approach enables accurate modeling of various systems and applications as described the following paragraphs.

C. System validation with SAR

After calibrating the models, the manually generated SAR scripts are fed into the system models for system analysis. The results from the simulation of the SAR script can be compared to the execution times of SAR on the two experimental testbeds with the intent of validating the simulation framework. The experimental run-times were gathered using

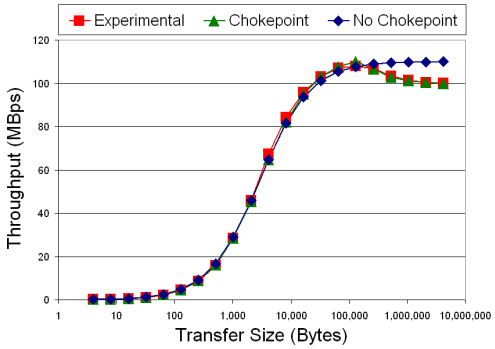


(a) Reads

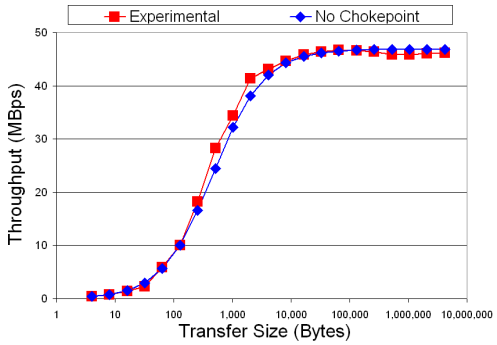


(b) Writes

Fig. 4. Throughput for device reads (a) and writes (b) on the Delta system



(a) Reads



(b) Writes

Fig. 5. Throughput for device reads (a) and writes (b) on the Kappa system

the version of SAR with code to offload the FFTs to the RC device. For each system, the two different versions of the SAR script, the *SW-Based Timing* script and *RC-Based Timing* script, were used as inputs to the system models, and the performance prediction results were recorded.

Table II summarizes the results of the SAR validation experiments. On the Kappa system, our simulation-based predictions of execution time were within 6.8% and 4.6% of the actual experimental application results for the two script versions, respectively. On the Delta system, the accuracies of the simulations improved to 2.6% and 2.1% for the two scripts. These results show that our system models have accurately predicted the performance of the entire SAR application for each system, thus providing a basis that demonstrates our prediction results will be accurate and insightful when exploring various system modifications. Meanwhile, the runtimes for each simulation ranged between 12 and 14 seconds, thus providing a fast and accurate simulative analysis.

It should also be noted that for each system, the *RC-Based Timing* script results provided a smaller margin of error versus the *SW-Based Timing* script. Thus, in both cases using the RC version of the application for script generation led to more accurate predictions. This observation is due to the differences in timing of the host processor computation blocks. Since the length of these computation blocks differed when the FPGA was introduced into the application, only the *RC-Based Timing* script captures those differences. Thus, the *RC-Based Timing* script can account for various nuances that take place within the system when a reconfigurable device is utilized. One possible reason for the observed differences in performance is cache-fetching patterns, which can change the number of cache hits in a host processor computation block.

TABLE II
SAR VALIDATION RESULTS

System	Exp.	SW-Based Timing	Diff.	RC-Based Timing	Diff.
Delta	42.63s	41.53s	2.58%	41.75s	2.06%
Kappa	77.11s	82.32s	6.76%	80.68s	4.63%

D. SAR simulative study

Now that we have validated the accuracy of our system models for SAR, we can now predict and analyze the effects of system modifications on the application's performance. The simulations conducted in this section observe the impact on overall system performance due to varying the characteristics of the local interconnect and middleware models for the Delta system. Four parameter sets are examined for the local

interconnect with each set representing a specific generation of the PCI interconnect technology, including 64-bit, 133 MHz PCI-X, x1 PCIe, x8 PCIe, and x16 PCIe. For each PCI technology, four parameter sets for the middleware capabilities are considered. The *Base MW* set uses the baseline values obtained during the calibration of the experimental Delta system. The *1/2L* set uses latency values that are one-half that of the baseline values. The *2xBW* set employs throughput values that are twice that of the baseline values, while the *Combo* set combines the halved latency values from the *1/2L* set and the doubled throughput values in the *2xBW* set.

TABLE III
SAR EXECUTION TIME VS. I/O PARAMETERS

	PCI-X	x1 PCIe	x8 PCIe	x16 PCIe
Base MW	41.75s	48.82s	40.50s	39.91s
1/2L	41.56s	48.63s	40.31s	39.72s
2xBW	38.30s	45.37s	37.05s	36.45s
Combo	38.11s	45.18s	36.86s	36.26s

The results from the I/O study are presented in Table III. As expected, the improved interconnect technologies yield faster SAR execution times than the original PCI-X configuration. However, the improvements were modest, as scaling from PCI-X in the baseline case to a 16-lane PCIe interface only led to a 2.24 second decrease in run-time. Meanwhile, halving the middleware latency had little effect on the overall system performance, although simply doubling the throughput of the device middleware (2*xBW*) led to a 3.45 second drop in run-time from the baseline. Thus, it appears that the middleware, namely the throughput of the middleware, is the primary bottleneck of those evaluated with this system for SAR. Despite the high number of transfers that occur during each application run, the time gained during each transfer from the latency improvement was far outweighed by the gains from the bandwidth increases for the large data transfers.

In this study, major increases in the capabilities of the local interconnect led to modest gains in the results due to other bottlenecks - insight which illustrates the effectiveness of simulation in predicting performance. Of course, with other system architectures, such as multiple-FPGA nodes, use of a more powerful technology such as PCI-Express may be expected to achieve a pronounced improvement. This study serves as a simple yet valuable example of the potential provided by this simulation framework. Numerous other tradeoff analyses can easily be performed within this framework, such as scaling the size and performance of the RC device and scaling the number of nodes in the system. These analyses can provide important insight into design decisions regarding current and future RC system architectures.

V. CONCLUSIONS

Reconfigurable devices such as FPGAs are becoming an increasingly important option for accelerating applications in high-performance computing, from satellites to supercomputers. Unfortunately, RC devices cannot achieve good speedup with all applications and their mappings to a particular platform, and developing reconfigurable applications and platforms can be a costly and time-consuming process. Meanwhile, simulation can be a relatively quick and cost-effective means to evaluate an application's performance on platforms that incorporate reconfigurable computing devices.

In this paper, a framework for fast and accurate simulations of applications on reconfigurable systems was introduced. First, the framework and methodology of our modeling approach for reconfigurable systems was presented. This framework divides its process into two domains, the application and simulation domain, which provides a methodology for mapping arbitrary applications to a variety of RC systems facilitating rapid in-depth performance projections and analyses. Finally, an application case study using the SAR application was performed. Validation results showed our system models could predict the overall performance of the application within a modest range of error. After the application validation tests, a simulation study was performed to demonstrate the capabilities of the framework to identify bottlenecks when running the SAR application on various system configurations. Future work in this area will include expanding the model library to support parallel and large-scale reconfigurable systems and applications, and developing characterization and calibration techniques for additional resources such as the memory hierarchy.

ACKNOWLEDGMENTS

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. The authors gratefully acknowledge vendor equipment and/or tools provided by Xilinx, MLDesign Technologies, and Nallatech that helped make this work possible.

REFERENCES

[1] M. Smith and G. Patterson, "Parallel Application Performance on Shared High Performance Reconfigurable Computing Resources," *Performance Evaluation,* Vol. 60, No. 1-4, May 2005, pp. 107-125

[2] K. Bondalapati, and V. Prasanna, "Reconfigurable Computing Systems," *Proc. IEEE,* Vol. 90, No. 7, July 2002, pp. 1201-1217.

[3] B. Holland, J. Greco, I. Troxel, G. Barfield, V. Aggarwal, and A. George, "Compile- and Run-time Services for Distributed Heterogeneous Reconfigurable Computing" *Proc. International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA),* Las Vegas, NV, June 26-29, 2006.

[4] OpenFPGA, http://www.openfpga.org.

[5] G. Schorcht, I. Troxel, K. Farhangian, P. Unger, D. Zinn, C. Mick, A. George, and H. Salzwedel, "System-Level Simulation Modeling with MLDesigner," *Proc. 11th IEEE/ACM International Symposium on Computer and Telecommunication Systems (MASCOTS),* Orlando, FL, Oct. 12-15, 2003, pp. 207-212.

[6] P. Meisl, M. Ito, and I. Cumming, "Parallel Synthetic Aperture Radar Processing on Workstation Networks," *Proc. Of 10th International Parallel Processing Symp. (IPPS)*, Washington, DC, Apr. 15-19, 1996.