Analytical Modeling for High Performance Reconfigurable Computers

Melissa C. Smith and Gregory D. Peterson
The University of Tennessee, Department of Electrical and Computer Engineering
414 Ferris Hall
Knoxville, TN, USA 37996-2100
{smithmc, gdp}@utk.edu

Keywords*:* Analytical Performance Models, Performance Analysis, Field Programmable Gate Arrays (FPGA), Reconfigurable Computing, High Performance Computing, Beowolf Clusters

Abstract

The integration of methodologies and techniques from parallel processing or High Performance Computing (HPC) with those of Reconfigurable Computing (RC) systems offers great potential for increased performance and flexibility for a wide range of computing problems. High Performance Computing architectures and Reconfigurable Computing systems have independently demonstrated performance advantages for applications such as digital signal processing, circuit simulation, and pattern recognition. By exploiting the near "hardware specific" speed of Reconfigurable Computing systems in a Beowolf cluster there is potential for significant performance advantages over other software-only or uniprocessor solutions. In this paper we present our initial results for an analytical modeling framework for High Performance Reconfigurable Computing systems.

INTRODUCTION

High Performance Computing or HPC encompasses vector supercomputers, massively parallel processors (MPPs), networks of workstations (NOWs), and other architectures configured to work collectively on a common problem. For our purposes, we will narrow our focus of HPC topologies to the distributed memory, MIMD class. Reconfigurable Computing or RC is the combination of reconfigurable logic with a general-purpose microprocessor. The architectural intention is to achieve hardware-like performance with software-like flexibility. In RC architectures, the microprocessor performs those operations that cannot be done efficiently in the reconfigurable logic such as loops, branches, and memory accesses, while computational cores are mapped to the reconfigurable hardware for better performance [13].

High Performance Reconfigurable Computing or HPRC is the combination of architecture ideas from HPC and RC. As shown in Figure 1, HPRC consists of a number of distributed computing nodes connected by some interconnection network (the network can be a switch, hypercube, systolic array, etc.), with some or all of the computing nodes having RC element(s) associated with them. The HPRC platform will potentially allow users to exploit the performance speedups commonly achieved in parallel systems in addition to the speedup offered by reconfigurable hardware coprocessors. Many applications such as image or signal processing algorithms and various simulation algorithms stand to benefit from the potential performance of this architecture.

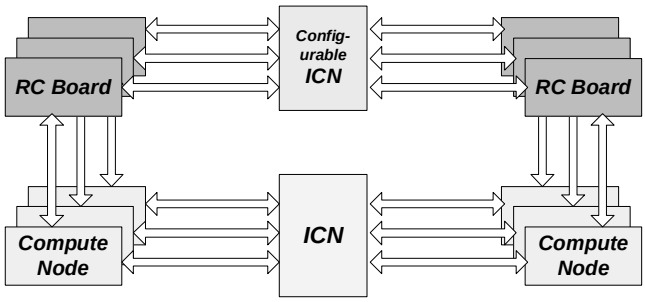


Figure 1 High Performance Reconfigurable Computer (HPRC) Architecture

An additional configurable network connecting the RC elements offers further performance advantages for many applications such as discrete event simulation. The additional network would provide a less inhibited route for synchronization, data exchange, and other communications between processing nodes. Research by Chamberlain [10, 27, 28], Reynolds et al. [32, 33, 34], and Underwood et al. [36] all confirm the performance benefits of a specialized configurable network for applications with barrier synchronization events or applications requiring the exchange of large amounts of data.

As individual platforms for computing, HPC and RC are challenging enough to program and utilize effectively. The combination of these powerful domains will require the development of new analysis and design tools. A performance modeling framework with models describing this new architecture will help not only in understanding and exploiting the design space but will also serve as a building block for many of these tools. HPRC system performance is affected by architectural decisions such as number of nodes, number of FPGAs, FPGA size, heterogeneity, and network performance. With all the potential permutations, the design

space for HPRC is extremely large. Without a modeling framework to assist with the analysis of these issues, trade-offs cannot be effectively analyzed potentially resulting in grossly inefficient use of the resources.

Given these observations, it is evident that a mathematical modeling framework is a necessary tool for analyzing the complex interaction of design issues and performance metrics for HPRC. Although substantial performance analysis research exists in the literature with regard to High Performance Computing (HPC) architectures [7, 8, 11, 17, 20, 24, 27, 28, 29, 30, 31, 35], the analysis of these architectures working together has received little attention to date.

The development of a modeling framework for a complex architecture such as HPRC presents several challenges and questions which will need to be addressed: *modeling communication time* (node-to-node, processor-to-RC unit, and RC unit-to-RC unit), *modeling computation time* (software in processor and firmware in RC unit), *modeling setup costs* (application setup, RC unit configuration, and network configuration), and *modeling load imbalance* (application tasks or data imbalance and hardware versus software imbalance).

ARCHITECTURE BACKGROUND AND PERFORMANCE METRICS

Reconfigurable Computing

Research has shown that many of today's computationally intensive applications can benefit from the speed offered by application specific hardware co-processors, but for applications with multiple specialized needs, it is not feasible to have a different co-processor for every specialized function. Such diverse applications stand to benefit the most from the flexibility of Reconfigurable Computing (RC) architectures since one RC unit can provide the functionality of several ASIC co-processors in a single device. Several research groups have demonstrated successful RC architectures [9, 14, 15, 18, 19, 21, 22, 23, 25, 37, 38].

There are many RC systems available from companies such as Annapolis Microsystems [1], Nallatech [5], Virtual Computer Corporation [6], and research organizations such as University of Southern California's Information Sciences Institute (ISI) [21], The Chinese University of Hong Kong [23], and Carnegie Mellon University [16, 26]. The Wildforce and Firebird units from Annapolis Microsystems and the SLAAC units from ISI are all PCI-bus cards with onboard memory. The Pilchard architecture developed by The Chinese University of Hong Kong interfaces through the memory bus for closer coupling with the processor. The PipeRench reconfigurable fabric from Carnegie Mellon is an interconnected network of configurable logic and storage elements which uses pipeline reconfiguration to reduce overhead which is one of the primary sources of inefficiency in other RC systems.

High Performance Reconfigurable Computing

The proposed HPRC platform consists of a system of RC nodes connected by some interconnection network (switch, hypercube, array, etc.). Our studies will focus on Beowolf Clusters of workstations populated with RC units. Each of the RC nodes may have one or more reconfigurable units associated with them. This architecture as stated before provides the user with the potential for more computational performance than traditional parallel computers or reconfigurable coprocessor systems alone.

The HPRC architecture offers many architecture options. Starting with the roots of HPC, there are many network and processor considerations to choose from which alone make performance analysis complicated and interesting. With the additional options available for RC such as the coupling of reconfigurable hardware to the processor, number of reconfigurable units, size of FPGA(s), dedicated interconnection network, and others, the analysis problem becomes enormous. Understanding these issues and how they affect the overall system performance is vital in exploiting this potentially powerful architecture.

Speedup and Efficiency

HPC performance is commonly measured in terms of speedup and efficiency. The basic definition for *speedup* in HPC is the ratio of the execution time (R) of the best possible serial algorithm on a single processor to the parallel execution time of the parallel algorithm on an m-processor parallel system:

$$S(m) = \frac{R(1)}{R(m)} \qquad \textit{(EQ 1)}$$

Efficiency is defined as the ratio of speedup to the number of processors, m:

$$Eff(m) = \frac{S(m)}{m} = \frac{R(1)}{m \cdot R(m)} \qquad \textit{(EQ 2)}$$

PERFORMANCE EVALUATION, ANALYSIS AND MODELING

Overview

To effectively use the proposed HPRC architecture, we must be able to analyze design trade-offs and evaluate the performance of applications as they are mapped onto the architecture. Performance models are commonly used tools for analyzing and exploiting available computational resources in HPC environments. Some commonly used modeling techniques in the analysis of computing systems are analytic models, simulations, and measurements. The best suitable modeling approach depends on the required accuracy, level of complexity, and analysis goal of the model. We can employ one or more of these modeling approaches to

better understand the trade-offs in mapping applications to HPRC resources as well as the most effective ways of doing so.

To develop a representative modeling framework for HPRC we will begin by investigating and characterizing the RC architecture and expanding this model to multiple nodes representative of an HPRC platform. In the RC environment, the focus will be on FPGA configuration, processor to FPGA communication, data distribution between FPGA and processor, memory access time, computation time in hardware and software, and other RC application setup costs. Next, we apply this knowledge to the multi node environment building on the earlier load balance work by Peterson [29]. We will develop an analytic modeling methodology for determining the execution time of a synchronous iterative algorithm and the potential speedup. *Synchronous iterative algorithms*, present in a large class of parallel applications, are iterative in nature and each iteration is separated from the previous and subsequent iterations by a synchronization operation. Examples of synchronous iterative algorithms include synchronous simulations, Gaussian elimination, and many image processing and data classification algorithms.

What follows below is the first iteration of the model using our available hardware and firmware for parameter measurement and model validation. Table 1 lists the symbols and definitions that will be used in the following sections.

Reconfigurable Computing Node Analysis

Our performance model analysis will begin with a single RC node running a synchronous iterative algorithm.

Table 1 Symbols and Definitions

Symbol	Definition
$T_{comm}(c)$	Communication time
N_C	Total number of messages per processor
t	Message latency
B_i	Size of message i
η	Network bandwidth
ρ	Load Imbalance factor between host nodes in a multi-node system
n	Number of hardware tasks
r	Number of hardware tasks not requiring new configuration
d	Number of hardware tasks not requiring new data set
I	Number of iterations
m	Number of workstations
σ	Hardware acceleration factor for node k in multi-node system
β	RC node background load factor

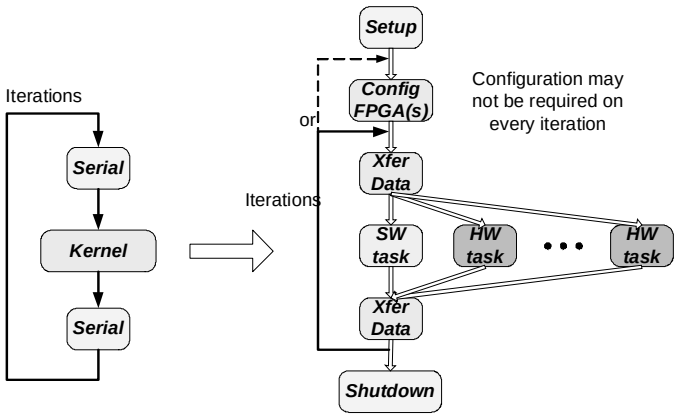


Figure 2 Synchronous Iterative Algorithm

These restrictions will allow us to investigate the interaction between the processor and RC unit.

First, we will assume we have a segment of an application that has I iterations and all iterations are roughly the same as shown in Figure 2. The RC unit has at least one FPGA (there may be other reconfigurable devices which provide control functions) and tasks can potentially execute in parallel in hardware (RC unit(s)) and in software on the processor. We should point out that the hardware and software task trees can be arbitrarily complex, however Figure 2 shows a simple hardware/software tree structure. Additionally, hardware can be reused within a given iteration if the number of tasks or size of the task exceeds that of the available FPGAs.

For a synchronous iterative algorithm, the time to complete a given iteration is equal to the time for the last task to complete either in hardware or software as shown in Figure 3. For each iteration of the algorithm, there are some operations which are not part of the kernel to be accelerated and are denoted $t_{serial,i}$. Other overhead processes that must occur such as configurations and exchange of data are denoted $t_{overhead,i}$. The time to complete the kernel tasks executing in software and hardware are $t_{SW,i}$ and $t_{HW,i}$. respectively. For I iterations of the algorithm where n is the number of hardware tasks, the runtime, R_{RC}, can be represented as [29]:

$$R_{RC} = \sum_{i=1}^{I} \left[t_{serial,i} + \max\left(t_{SW,i}, \max_{1 \le j \le n} [t_{HW,i,j}] \right) + t_{overhead,i} \right] \qquad (EQ\ 3)$$

To simplify the math analysis, we will make a couple of reasonable assumptions. First, we will assume that each iteration requires roughly the same amount of computation allowing us to remove the reference to individual iterations in Eq. 3. Second, we will model each term as a random variable and use their expected values. Thus we define t_{serial} as the expected value of $t_{serial,i}$ and $t_{overhead}$ as the expected

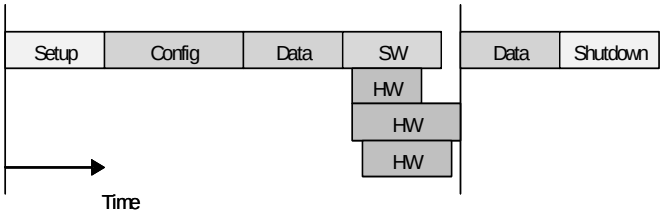


Figure 3 Reconfigurable Computing Task Completion

value of $t_{overhead,i}$. The mean time required for the completion of the parallel hardware/software tasks is represented by the expected value of the max(t_{SW}, t_{HW}). Finally, we will assume that each of the random variables are independent and identically distributed (iid). We can then write the run time as:

$$R_{RC} = \sum_{i=1}^{I} Et_{serial,i} + E[t_{overhead,i}] + E\left[\max\left(t_{SW,i}, \max_{1 \le j \le n}[t_{HW,i,j}]\right)\right] \quad (EQ\ 4)$$

$$= I\left(t_{serial} + E\left[\max\left(t_{SW}, \max_{1 \le j \le n}[t_{HW,j}]\right)\right] + t_{overhead}\right)$$

If the iterations are not iid, we must retain the first form of Eq. 4 and the math analysis is more cumbersome but the same general approach used in the following analysis can be utilized.

The execution time for hardware tasks should be deterministic and related to the clock frequency of the hardware. We will assume that all concurrent or parallel hardware tasks are the same. Also, we will initially assume that the hardware and software tasks do not overlap (we will relax this restriction in later versions of the model). Therefore we can represent the expected hardware execution time with the mean value, t_{HW}, and simplify the equation. The execution time of the software tasks will depend not only on the speed of the processor but also on the background load of the system. We will represent this background load as β and represent the expected completion time of the software on a dedicated processor as t_{SW}. The expected execution or runtime on the RC system then becomes:

$$R_{RC} = I(t_{serial} + (\beta \cdot t_{SW}) + (t_{HW}) + t_{overhead}) \quad (EQ\ 5)$$

If there is no background load on the processor, $\beta = 1$, and the equation reduces to a dedicated system. The background load is normally greater than or equal to 1. If there are multiple sequential hardware tasks, g, the expected value becomes:

$$R_{RC} = I\left(t_{serial} + (\beta \cdot t_{SW}) + \sum_{j=1}^{g} t_{HW,j} + t_{overhead}\right) \quad (EQ\ 6)$$

Noting that the total work measured in time for a software-only solution is not equivalent to the total work measured in time on an RC system solution, we introduce a hardware acceleration factor σ to account for the difference. The acceleration factor is less than or equal to 1. Since the goal of RC systems is to speed up an application, only tasks that would be faster in hardware are implemented in hardware. For example, an FFT in software may take longer to execute than an equivalent implementation in the hardware. Given the total work that will be completed in hardware and software on an RC system, we can represent the software-only run time on a single processor as:

$$R_1 = I \cdot \left(t_{serial} + t_{SW} + \frac{1}{\sigma} \cdot \sum_{n} t_{HW}\right) \quad (EQ\ 7)$$

The overhead for an RC system consists of the FPGA configuration time and data transfer time. The configuration time for the FPGA(s) is (n-r) x t_{config}, where r is the number of hardware tasks not requiring a new configuration. The time to transfer data to and from the RC unit is (n-d) x t_{data}, where d is the number of hardware tasks not requiring a new data set.

The speedup, S_{RC}, is defined as the ratio of the run time on a single processor to the run time on the RC node:

$$S_{RC} = \frac{R_1}{R_{RC}} \quad (EQ\ 8)$$

$$= \frac{t_{serial} + t_{SW} + \frac{1}{\sigma} \cdot \sum_{n} t_{HW}}{t_{serial} + (\beta \cdot t_{SW}) + (t_{HW}) + [(n-d) \cdot t_{data} + (n-r) \cdot t_{config}]}$$

Using Eq. 8 we can investigate the impact of load imbalance and various overhead issues on the algorithm performance by varying β, t_{config}, t_{data}, t_{HW}, t_{SW}, r, d, and n.

High Performance Computing Communication Analysis

Communication delay between workstations in a network is affected by the network topology, communication volume, and communication patterns. Other research on network performance models report that a simple communication model that accounts for message startup time and network bandwidth is adequate [12]. For the total number of messages per processor, N_C, the message latency τ, network bandwidth η, and size of message B_i, the communication time can be modeled as [12]:

$$T_{comm}(c) = \sum_{i=1}^{N_c} \left(\tau + \frac{B_i}{\eta}\right) \quad (EQ\ 9)$$

Both τ and η can be approximated from measured values. It should be noted that in practice, η may not be a constant. The model represented in Eq. 9 is non-preemptive

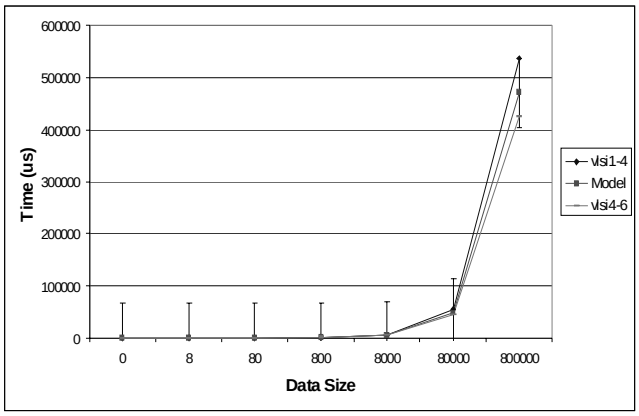


Figure 4 Communication Model and Measurements

(messages are serviced one-by-one) and useful for modeling clusters connected with contention free networks. We have conducted some measurements on communication time of messages between several processors and compared those measurements to the values predicted by Eq. 9. Results for two of the measurements are given with the model prediction in Figure 4.

High Performance Reconfigurable Computing Multi-Node Analysis

Now that we have a model for a single RC node and an understanding of the basic HPC issues involved in a set of distributed nodes, we will turn our focus to expanding the model for multi-node analysis. An example of the HPRC architecture was shown in Figure 1. For now, we will not consider the optional configurable interconnection network between the RC units in our modeling analysis.

We will again start our performance model analysis using a synchronous iterative algorithm this time running on a platform consisting of multiple RC nodes. The restriction of synchronous iterative algorithms will allow us to investigate the communication and synchronization that occurs among nodes between iterations. We will begin our model by restricting our network to a dedicated homogeneous system where there is no background load (i.e. all nodes are identical, same processor and same RC system configuration). We will relax this restriction in future editions of the model.

Again, we will assume we have a segment of an application having *I* iterations that will execute on parallel nodes with hardware acceleration. Additionally, we will assume that all iterations are roughly the same as is shown in Figure 5. Software tasks can be distributed across computing nodes in parallel and hardware tasks are distributed to the RC unit(s) at each individual node.

For a synchronous iterative algorithm, the time to complete an iteration is equal to the time for the last task to complete on the slowest node whether it be hardware or software. For each iteration of the algorithm, there are some calculations which cannot be executed in parallel or accelerated in hardware and are denoted $t_{mserial,i}$. There are other serial operations required by the RC hardware and they are denoted $t_{nserial,i}$. Other overhead processes that must occur such as synchronization and exchange of data are denoted $t_{movhd,i}$ and $t_{novhd,i}$ for the host and RC systems respectively. The time to complete the tasks executing in parallel on the processor and RC unit are $t_{SW,i,k}$ and $t_{HW,i,j,k}$. respectively. For *I* iterations of the algorithm where *n* is the number of hardware tasks at node *k* and *m* is the number of processing nodes, the runtime, R_P, can be represented as [29]**:**

$$R_p = \sum_{i=1}^{I} \left[t_{mserial,i} + t_{nserial,i} + \max_{1 \le k \le m} \left(t_{SW,i,k}, \max_{1 \le j \le n} [t_{HW,i,k,j}] \right) + t_{movhd,i} + t_{novhd,i} \right] \quad \textit{(EQ 10)}$$

$$= \sum_{i=1}^{I} \left[t_{mserial,i} + t_{nserial,i} + \max_{1 \le k \le m} (t_{node,i,k}) + t_{movhd,i} + t_{novhd,i} \right]$$

Again, to make the math analysis more resonable, we will make a couple of reasonable assumptions. First, we will assume that each iteration requires roughly the same amount of computation thus we can remove the reference to individual iterations in Eq. 10. Second, we will also assume that each node has the same hardware tasks and configuration making the configuration overhead for each node the same. Third, we will model each term as a random variable and use their expected values. Thus we define $t_{mserial}$ and $t_{nserial}$ as the expected value of $t_{mserial,i}$ and $t_{nserial,i}$. Similarly, we define t_{movhd} and t_{novhd} as the expected value of $t_{movhd,i}$ and $t_{novhd,i}$. The mean time required for the completion of the RC hardware/software tasks is represented by the expected value of the maximum $t_{node,k}$ ($1 \le k \le m$). Finally, assuming that the random variables are each independent and identically distributed (iid), the run time can then be expressed as:

$$R_p = \sum_{i=1}^{I} \left[E[t_{mserial,i}] + E[t_{movhd,i}] + E\left[\max_{1 \le k \le m} \{ t_{RC,i,k} + t_{nserial,i} + t_{novhd,i} \} \right] \right] \quad \textit{(EQ 11)}$$

$$= I \cdot \left(t_{mserial} + t_{movhd} + E\left[\max_{1 \le k \le m} (t_{node,k}) \right] \right)$$

We can rewrite the total work at node *k* in terms of the average task completion time rather than the maximum and later multiply by an imbalance factor to account for application and background load imbalances. Again assuming the random variables are iid, we can express the total work across all *m* nodes in the HPRC platform as:

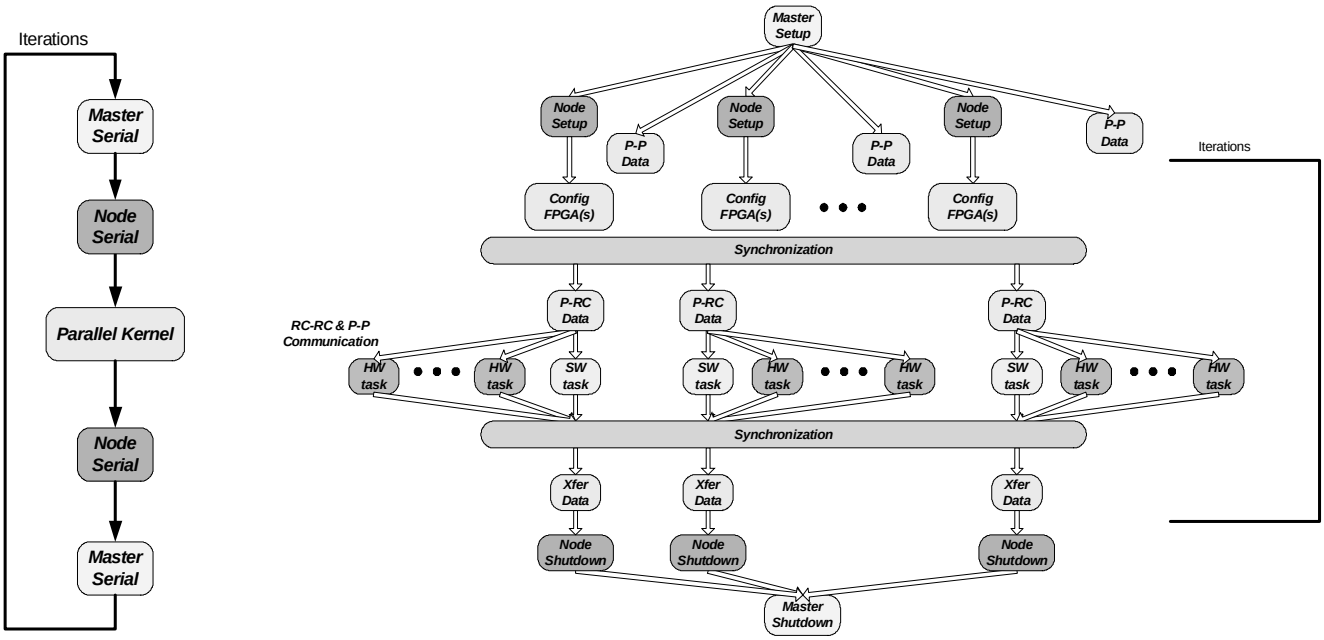


Figure 5 Flow of Synchronous Iterative Algorithm for Multi Node

$$t_{work} = m \cdot E[t_{node,k}] \qquad \textit{(EQ 12)}$$

When tasks are divided across the nodes, a load imbalance due to application workload distribution, background or network heterogeneity exists. We will represent this load imbalance as ρ. We will assume that the RC system load imbalance at any node is independent of the others. The completion time can then be expressed as the average task completion time within an iteration multiplied by the load imbalance factor:

$$E\left[\max_{1 \le k \le m}(t_{node,k})\right] = \rho \cdot E[t_{node,k}] \qquad \textit{(EQ 13)}$$

Combining Eq. 12 and Eq. 13 we can rewrite the maximum task completion time as,

$$E\left[\max_{1 \le k \le m}(t_{node,k})\right] = \frac{\rho \cdot t_{work}}{m} \qquad \textit{(EQ 14)}$$

Note that if the load is perfectly balanced or if the algorithm runs entirely in software (n=0), ρ is the ideal value of 1. As the load imbalance becomes worse, ρ increases. If the algorithm runs entirely on a single node, m=1, ρ is the ideal value of 1 and the model reduces to a single processor.

Noting that the total work measured in time for a software-only solution is not equivalent to the total work measured in time on an HPRC platform solution, we introduce a hardware acceleration factor σ_k to account for the difference at each node k. Given the total work that will be completed in hardware and software on an HPRC platform, we can represent the software only run time on a single processor as:

$$R_1 = I \cdot \left[t_{mserial} + \sum_{k=1}^{m} \left(t_{SW} + \frac{1}{\sigma_k} \cdot \sum_{n} t_{HW} \right) \right] \qquad \textit{(EQ 15)}$$

The overhead for the HPRC platform consists of the FPGA configuration and data transfers as discussed earlier and the synchronization between the nodes. We will initially model the time required for synchronization as a logarithmic growth with the number of nodes [29]. The communication between nodes can be modeled using Eqn. 9. The speedup, S_P, for the HPRC platform is defined as the ratio of the run time on a single processor to the run time on m RC nodes:

$$S_P = \frac{t_{mserial} + \sum_{k=1}^{m} \left(t_{SW} + \frac{1}{\sigma_k} \cdot \sum_{n} t_{HW} \right)}{t_{mserial} + \frac{\rho \cdot t_{work}}{m} + [t_{synch} \cdot \log_2(m)] + T_{comm}(c)} \qquad \textit{(EQ 16)}$$

Using Eq. 16 we can investigate the impact of load imbalance and various other overhead issues on the algorithm performance by varying σ_k, ρ, t_{synch}, n, and the variables of $T_{comm}(c)$.

High Performance Reconfigurable Computing Development Hardware

There are two HPRC clusters that will be available for development and validation of the model. The Air Force Research Laboratory in Rome, NY is assembling a "heterogeneous HPC" which is a cluster of four Pentium nodes populated with Firebird boards [1]**.** Future plans include expansion to more nodes. The HPRC cluster at UT consists of eight Pentium nodes populated with Pilchard boards [23]. Other available hardware for parameter measurements includes a cluster of Sun workstations, Wildforce, Firebird, and SLAAC RC boards [21].

Reconfigurable Computing Model Validation

We have made basic model parameter measurements using a sample application for the Wildforce board as a benchmark. Figure 6 shows plots of the model with the measured parameters. To validate the execution time prediction of the model, the benchmark measurements are used with the developed model to predict the runtime for three signal processing demos: a high pass filter, and two version of an automatic target recognition algorithm. From the benchmark, we have determined the model parameters as shown in Table 2. The configuration values for CPE0 and PE1 are significantly different because they are two different Xilinx devices and are therefore accounted for separately in the model calculations. For this application the only part of the algorithm considered as serial is the board configuration and setup. There is only one iteration therefore *I* is set to 1. The remaining unknowns are the values for the total work and the application load imbalance.

The total work can be determined from the amount of work completed by the software task plus the amount completed by the hardware task. This can be represented in terms of the number of events to be processed multiplied by the execution time per event:

$$t_{RCwork} = N_e \cdot t_{hwexe} + t_{swexe} \qquad \textit{(EQ 17)}$$

Where N_e is the total number of events, t_{hwexe} is the hardware execution time per event, and t_{swexe} is the software execution time. In this particular application, the software and hardware tasks do not overlap.

Using the denominator of Eq. 8, we can predict the runtime of the three demo algorithms. The average runtime of fifty trials on the Wildforce RC system is shown in Table 3 and Figure 7 along with the model predictions. The number following the algorithm name indicates the input data size. The value 128 indicates an input data set of 128x128 and similarly the value 256 indicates a 256x256 input data set.

One possible candidate for the error in the model prediction is our measurement techniques for the model parameters. We believe that the measurements have enough accuracy and the problem with over estimation of the runtime could be in the overhead introduced by the probes.

Another possible error contribution is from the model methodology and assumptions. Being the first pass at modeling the performance of an RC system, the representation for the total work and application workload balance may be inaccurate. More studies of different algorithms and systems will be required to make a final determination on the accuracy of this representation.

Finally, issues not represented in the model may be contributing to the error. System issues such as caching, optimum data packet size, and other optimization techniques used in operating systems and possibly the RC board API will need to be investigated.

Table 2 Model Parameters for Wildforce from Benchmark Application

	benchmark (usec)
CPE0	535274.96
PE1	257232.82
HW	1250.52
Data	33282.08
Setup (tsw)	68892.34
Serial	40750.46

Table 3 Runtime Predictions and Measurements (time in seconds)

	model prediction	average
hipass_128	0.911342	1.313353
hipass_256	1.773769	1.907098
START_128	5.166674	4.597426
START_256	6.175542	6.121883
START20_128	7.253404	8.134971
START20_256	8.292768	8.855299

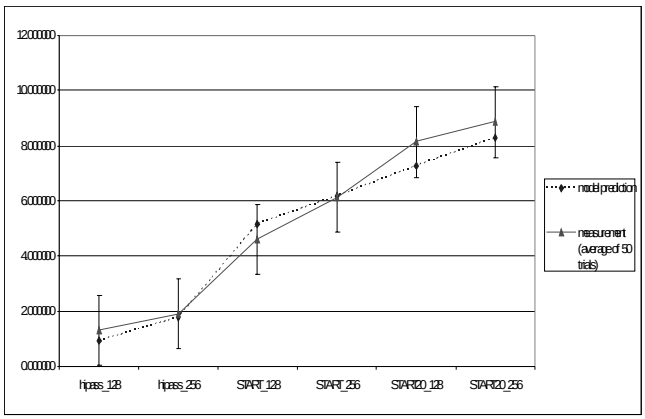
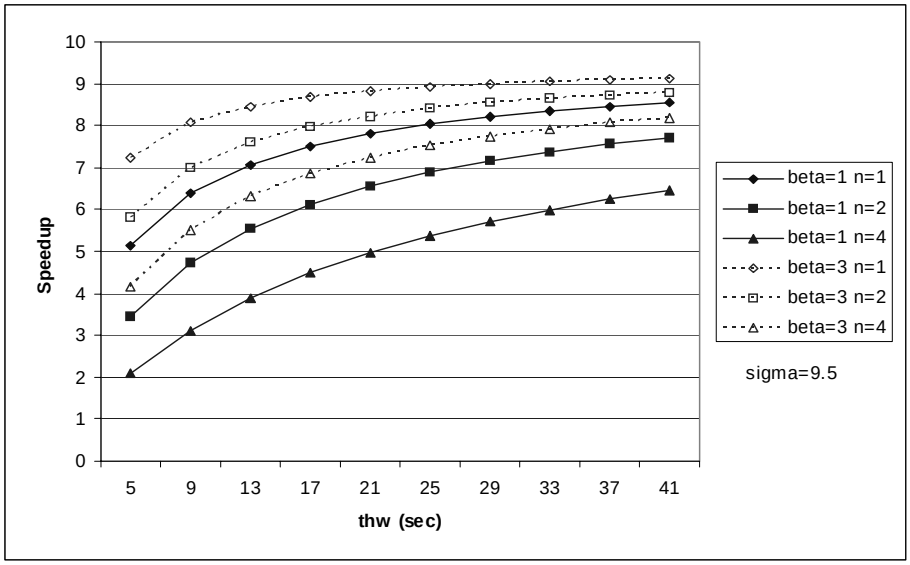


Figure 7 Comparison of RC Model Prediction with Measurement Results

High Performance Reconfigurable Computing Model

Again we will use the measured values from the Wildforce and Firebird experiments for the RC parameters. Figure 8 shows the speedup curves for the model. As seen in the figures, the speedup improves with increasing workload for lower load imbalance values but is inversely impacted by increasing load imbalance. Model vaildation work for the complete HPRC model is in progress.

(a)



(b)

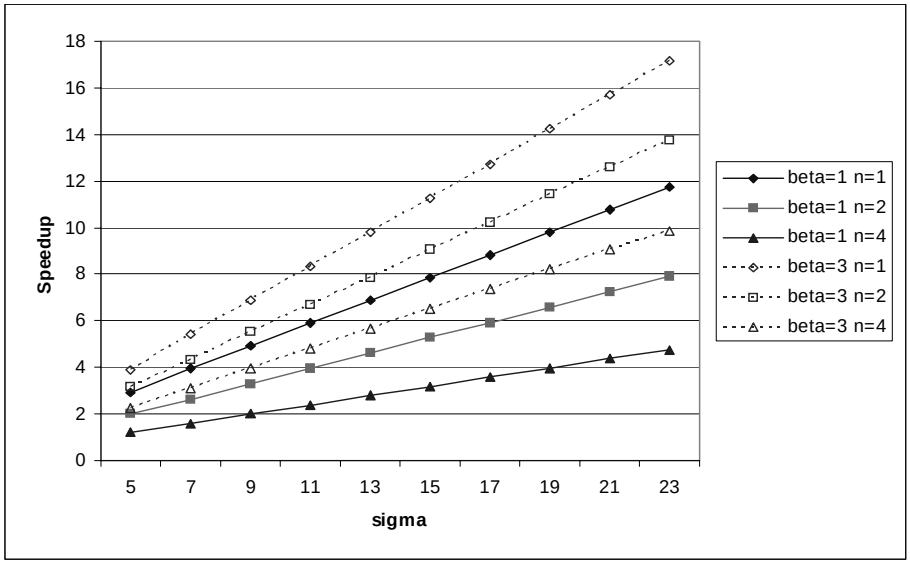


Figure 6 Speedup Curves: a) Vary total work by hardware and b) Vary hardware acceleration factor

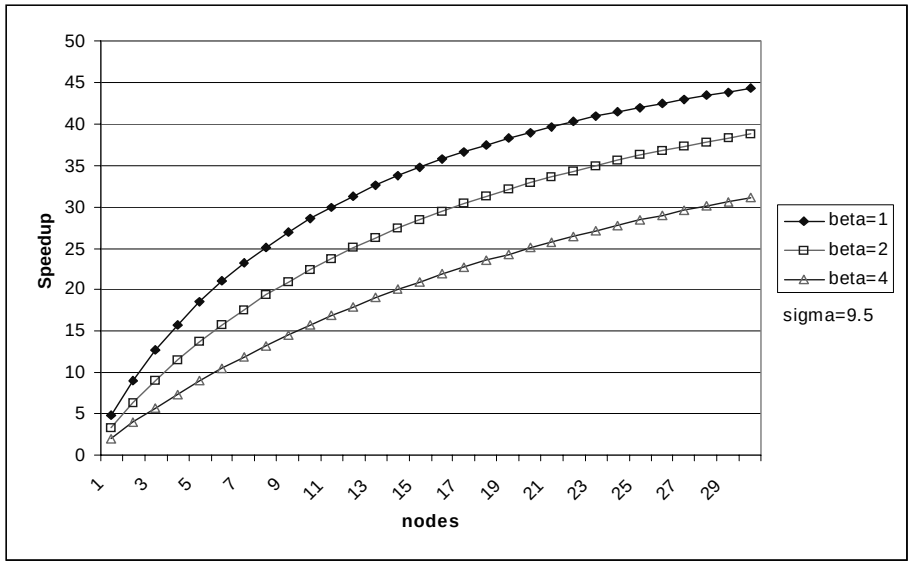
CONCLUSIONS AND FUTURE WORK

The first editions of the RC and HPRC models are presented including initial parameter analysis and measurements. The parameter measurements for the RC platform need to be expanded to include tests for determining the background load parameter. Future work includes parameter and validation measurements for the multi node environment as well as further model development for node-to-node communications in networks with contention. Models to predict the load imbalance factors are currently under development expanding on the results from Peterson's work [29]. Also, cost functions for power and total cost of the system need to be fully modeled and verified. Finally, as the HPRC platform applications are developed and studied, the use of the model for scheduling and load balancing will be demonstrated as a manual exercise.

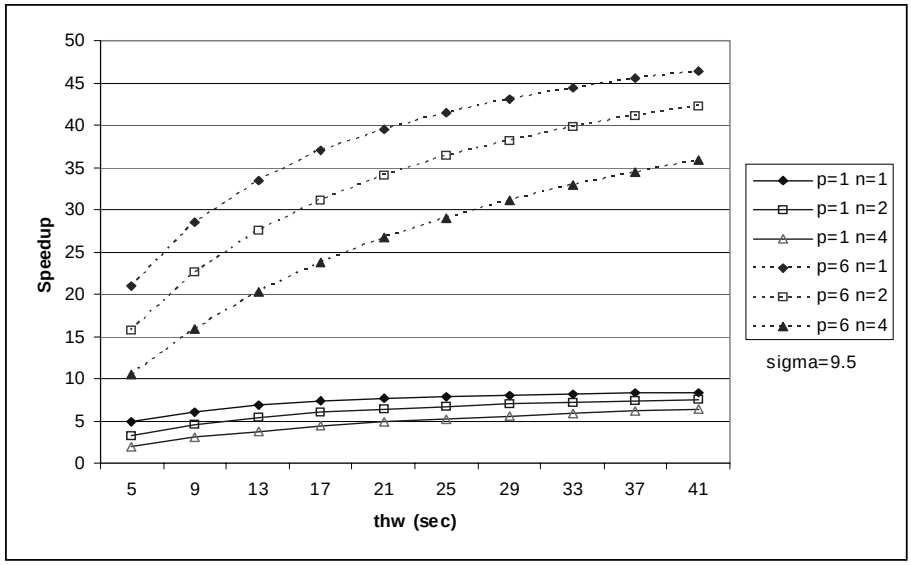
While the primary motivation for the development of the HPRC modeling framework is for performance modeling of the HPRC platform, the model can also serve as the foundation for task scheduling and load balancing CAD tools. A specific RC performance related use of the model is for computing performance classification of an RC node. The NetSolve [3] and SinRG [4] programs at the University of Tennessee or other similar projects such as Condor [2] would potentially find the performace classification capabilities useful for their tools.

Finally, many parallels can be drawn between the architecture and design issues encountered in Systems on a Chip (SoC) design and the HPRC architecture. SoC is a self-contained electronic system residing on a single piece of silicon as shown in Figure 9. The HPRC modeling framework could provide vital performance analysis information to the SoC designer during intital phases of the design process. SoC processors and memory modules are very similar to the nodes in an HPRC system and both architectures can also incorporate reconfigurable units. Also, both architectures

(a)



(b)



(c)

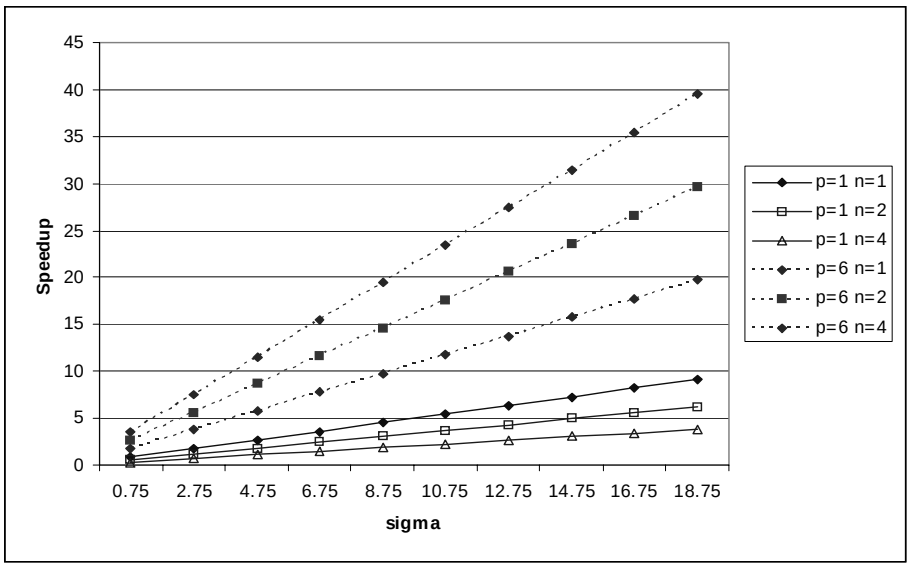


Figure 8 Speedup Curves: a) Vary number of nodes b) Vary total hardware work and c) Vary hardware acceleration factor

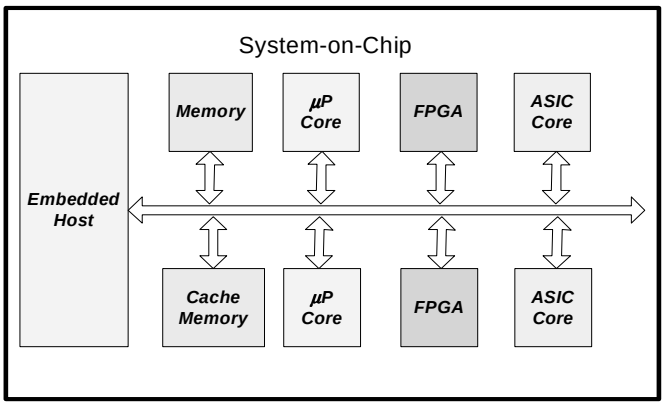


Figure 9 SoC Architecture Example

have an interconnection network between nodes. Considering these similarities, the HPRC modeling framework would also be useful in the area of SoC design.

REFERENCES

[1] *Annapolis Microsystems*, Available from http://www.annapmicro.com.

[2] *Condor*, Available from http://www.cs.wisc.edu/condor/.

[3] *NetSolve*, Available from http://icl.cs.utk.edu/netsolve/.

[4] *SInRG: Scalable Intracampus Research Grid*, Available from http://www.cs.utk.edu/sinrg/index.html.

[5] *Nallatech FPGA-centric Systems & Design Services*, Available from http://www.nallatech.com/ .

[6] *Virtual Computer Corporation*, Available from http://www.vcc.com/index.html.

[7] Amdahl, G. M. 1967. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities." In *AFIPS Conference Proceedings*, 483-485.

[8] Atallah, Mikhail J. et al. 1992. "Models and Algorithms for Coscheduling Compute-Intensive Tasks on a Network of Workstations." *Journal of Parallel and Distributed Computing*, No. 16: 319-327.

[9] Bondalapati, Kiran et al. 1999. "DEFACTO: A Design Environment for Adaptive Computing Technology." In *Proceedings of the 6th Reconfigurable Architectures Workshop (RAW99)*, Springer-Verlag.

[10] Chamberlain, Roger D. 1995. "Parallel Logic Simulation of VLSI Systems." In *Proc. of 32nd Design Automation Conf.*, 139-143.

[11] Clement, Mark J. and Quinn, Michael J. 1993. "Analytical Performance Prediction on Multicomputers." In *Proceedings of Supercomputing '93*.

[12] Clement, Mark J., Steed, Michael R., and Crandall, Phyllis E. 1996. "Network Performance Modeling for PVM Clusters." In *Proceedings of Supercomputing '96*.

[13] Compton, K. and Hauck, S. 1999. "Configurable Computing: A Survey of Systems and Software." Northwestern University, Dept. of ECE Technical Report. Northwestern University.

[14] DeHon, Andre. 1998. "Comparing Computing Machines." *Proceedings of SPIE*, No. 3526(Configurable Computing: Technology and Applications): 124.

[15] Gokhale, M. et al. 1991. "Building and Using a Highly Parallel Programmable Logic Array." *IEEE Computer*, No. 24(1): 81-89.

[16] Goldstein, Seth Copen et al. 2000. "PipeRench: A Reconfigurable Architecture and Compiler." *IEEE Computer*, 70-77.

[17] Gustafson, J. L. 1988. "Reevaluating Amdahl's Law." *Communications of the ACM*, No. 31(5): 532-533.

[18] Hauck, Scott et al. 1997. "The Chimaera Reconfigurable Functional Unit." *IEEE Symposium on Field-Programmable Custom Computing Machines*, -10.

[19] Hauser, John R. and Wawrzynek, John. 1997. "Garp: A MIPS Processor With a Reconfigurable Coprocessor." *IEEE Symposium on Field-Programmable Custom Computing Machines*.

[20] Hu, Lei and Gorton, Ian. 1997. "Performance Evaluation for Parallel Systems: A Survey." UNSW-CSE-TR-9707. University of NSW, School of Computer Science and Engineering, Sydney, Australia.

[21] I.S.I.East, *SLAAC: System-Level Applications of Adaptive Computing*, Available from http://www.east.isi.edu/projects/SLAAC/.

[22] Jones, Mark T., Langston, Michael A., and Raghavan, Padma. 1998. "Tools for Mapping Applications to CCMs." In *SPIE Photonics East '98.*

[23] Leong, P. H. W. et al. 2001. "Pilchard - A Reconfigurable Computing Platform With Memory Slot Interface." In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, (California USA, IEEE).

[24] Mohapatra, Prasant, Das, Chita R., and Feng, Tse-yun. 1994. "Performance Analysis of Cluster-Based Multiprocessors." *IEEE Transactions on Computers*, 109-115.

[25] Moll, L., Vuillemin, J., and Boucard, P. 1995. "High-Energy Physics on DECPeRLe-1 Programmable Active Memory." In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 47-52.

[26] Myers, M., Jaget, K., Cadambi, Srihari, Weener, J., Moe, Matt, Schmit, Herman, Goldstein, Seth Copen, and Bowersox, D. 1998. "PipeRench Manual." Carnegie Mellon University.

[27] Noble, Bradley L. and Chamberlain, Roger D. 1999. "Performance Model for Speculative Simulation Using Predictive Optimism." In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, 1-8.

[28] Noble, Bradley L. and Chamberlain, Roger D. 2000. "Analytic Performance Model for Speculative, Synchronous, Discrete-Event Simulation." In *Proc.of 14th Workshop on Parallel and Distributed Simulation*,

[29] Peterson, Gregory D. 1994. "Parallel Application Performance on Shared, Heterogeneous Workstations." Doctor of Science, Washington University Sever Institute of Technology, Saint Louis, Missouri.

[30] Peterson, Gregory D. and Chamberlain, Roger D. 1996. "Parallel Application Performance in a Shared Resource Environment." *Distributed Systems Engineering*, No. 3: 9-19.

[31] Peterson, J. L. 1981. *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Englewood Cliffs, NJ.

[32] Reynolds, Paul F., Jr. and Pancerella, Carmen M. 1992. "Hardware Support for Parallel Discrete Event Simulations." TR-92-08. Computer Science Dept.

[33] Reynolds, Paul F., Jr., Pancerella, Carmen M., and Srinivasan, Sudhir. 1992. "Making Parallel Simulations Go Fast." In *1992 ACM Winter Simulation Conference.*

[34] Reynolds, Paul F., Jr., Pancerella, Carmen M., and Srinivasan, Sudhir. 1993. "Design and Performance Analysis of Hardware Support for Parallel Simulations." *Journal of Parallel and Distributed Computing*.

[35] Thomasian, A. and Bay, P. F. 1986. "Analytic Queueing Network Models for Parallel Processing of Task Systems." *IEEE Transactions on Computers*, No. C-35 (12): 1045-1054.

[36] Underwood, Keith D., Sass, Ron R., and Ligon, Walter B., III. 2001. "A Reconfigurable Extension to the Network Interface of Beowulf Clusters." In *Proc. of the 2001 IEEE International Conference on Cluster Computing*, IEEE Computer Society, -10.

[37] Vuillemin, J. et al. 1996. "Programmable Active Memories: Reconfigurable Systems Come of Age." *IEEE Transactions on VLSI Systems*, No. 4 (1): 56-69.

[38] Ye, Zhi Alex et al. 2000. "CHIMAERA: A High-Performance Architecture With A Tightly-Coupled Reconfigurable Functional Unit." In *Proc. of International Symposium on Computer Architecture*, (Toronto, Canada).

Melissa C. Smith is a doctoral candidate in electrical engineering at the University of Tennessee and a research and development staff member at Oak Ridge National Laboratory. Her research interests include reconfigurable computing, parallel and distributed computing, and performance modeling and analysis. She received her MS in electrical engineering in 1994 and her BS in electrical engineering in 1993, all from Florida State University. She is a member of IEEE, ACM, Eta Kappa Nu, Tau Beta Pi and Phi Kappa Phi.

Gregory D. Peterson is an assistant professor of electrical engineering at the University of Tennessee. He conducts research and teaches in the areas of computer architecture, digital systems, reconfigurable and parallel computing. He received his DSc in electrical engineering in 1994, his MS in electrical engineering in 1992 and his BS in electrical engineering in 1990, all from Washington University in St. Louis, Missouri. He is a member of IEEE, ACM, Eta Kappa Nu, and Tau Beta Pi.