RAT: RC Amenability Test for Rapid Performance Prediction

BRIAN HOLLAND, KARTHIK NAGARAJAN, and ALAN D. GEORGE
NSF Center for High-Performance Reconfigurable Computing (CHREC),
University of Florida

While the promise of achieving speedup and additional benefits such as high performance per watt with FPGAs continues to expand, chief among the challenges with the emerging paradigm of reconfigurable computing is the complexity in application design and implementation. Before a lengthy development effort is undertaken to map a given application to hardware, it is important that a high-level parallel algorithm crafted for that application first be analyzed relative to the target platform, so as to ascertain the likelihood of success in terms of potential speedup. This article presents the RC Amenability Test, or RAT, a methodology and model developed for this purpose, supporting rapid exploration and prediction of strategic design tradeoffs during the formulation stage of application development.

Categories and Subject Descriptors: B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; I.6.m [**Simulation and Modeling**]: Miscellaneous; B.2.0 [**Arithmetic and Logic Structures**]: General

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: FPGA, reconfigurable computing, performance prediction, strategic design methodology, formulation methodology

ACM Reference Format:
Holland, B., Nagarajan, K., and George, A. D. 2009. RAT: RC amenability test for rapid performance prediction. ACM Trans. Reconfig. Techn. Syst. 1, 4, Article 22 (January 2009), 31 pages. DOI = 10.1145/1462586.1462591. http://doi.acm.org/10.1145/1462586.1462591.

1. INTRODUCTION

Computing is currently undergoing two reformations, one in device architecture and the other in application development. Using the growth in transistor density predicted by Moore's Law for increased clock rates and instruction-level parallelism has reached fundamental limits, and the nature of current

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422.
Author's address: B. Holland; email: Holland@hcs.ufl.edu.

and future device architectures is focused upon higher density in terms of multicore and many-core structures and more explicit forms of parallelism. Many such devices exist and are emerging on this path, some with a fixed structure (e.g., quad-core CPU, Cell Broadband Engine) and some reconfigurable (e.g., FPGAs). Concomitant with this reformation in device architecture, the complexity of application development for these fixed or reconfigurable devices is at the forefront of fundamental challenges in computing today.

The development of applications for complex architectures can be defined in terms of four stages: formulation, design, translation, and execution. The purpose of formulation is exploration of algorithms, architectures, and mappings, where strategic decisions are made prior to coding and implementation. The design, translation, and execution stages are where implementation occurs in an iterative fashion, in terms of programming, translation to executable codes and cores, debugging, verification, performance optimization, etc. As architecture complexity continues to increase, so too does the importance of the formulation stage, since productivity increases when design weaknesses are discovered and addressed early in the development process. FPGA applications are particularly noteworthy for the amount of effort needed with existing languages and tools to render a successful implementation, and thus productivity of application development for FPGA-based systems can greatly benefit from better concepts and tools in the formulation stage. This article presents a novel methodology and model to support the rapid formulation of applications for FPGA-based reconfigurable computing systems. Known as the RC Amenability Test, RAT provides a framework for rapid prediction of potential speedup for a given high-level parallel algorithm mapped to a selected hardware target, so that a variety of strategic tradeoffs in algorithm and architecture exploration can be evaluated before undertaking weeks or months of costly implementation.

The need for the RAT methodology stemmed from common difficulties encountered during several FPGA application development projects. Researchers would typically possess a software application but would be unsure about potential performance gains in hardware. The level of experience with FPGAs would vary greatly among the researchers and inexperienced designers were often unable to quantitatively project and compare possible algorithmic design and FPGA platform choices for their application. Many initial predictions were haphazardly formulated and performance estimation methods varied greatly. Consequently, RAT was created to consolidate and unify the performance prediction strategies for faster, simpler, and more effective analyses.

Three factors are considered for the amenability of an application to hardware: throughput, numerical precision, and resource usage. The authors believe that these issues dominate the overall effectiveness of an application's hardware mapping. Consequently, analyses for these three factors comprise the majority of the RAT methodology. The throughput analysis uses a series of simple equations to predict the performance of the application based upon known parameters (e.g., interconnect speed) and values estimated from the proposed design (e.g., volume of communicated data). Numerical precision

analysis is a subset of throughput encompassing the design trade-offs in performance associated with possible algorithm data formats and their associated error. Resource analysis involves estimating the application's hardware usage in order to detect designs that consume more than the available resources.

With the computing reformation increasing interest in parallel algorithms for the RC paradigm, achieving speedup with FPGAs over traditional CPUs is often a major performance goal. Consequently, accurate throughput analysis is the primary focus of the RAT methodology. While numerical precision, resource utilization, and other issues such as development time or power usage are not trivial, they are less likely to be the sole contributor to the failure of an application migration when speedup is the primary goal. Consequently, this article details the analytic model for estimating performance along with other considerations for effective usage of RAT. The complete methodology targets common algorithm and architecture features, primarily deterministic structures and communication between an FPGA accelerator and its host CPU. The model is meant to provide reasonable approximations of performance with minimal effort, and its intended usage is illustrated on several case studies. Errors less than 20% are generally considered acceptable for RAT predictions, though this paper attempts to provide greater accuracy where possible as further validation of the analytic model. RAT is intended to provide efficient and quantitative algorithm and platform exploration as a first step in uncovering suitable hardware mappings. It does not (and should not) replace design-time simulative and analytic models based on actual code but works synergistically to ensure the later stages of design focus on the most promising algorithm and platform choices.

The remainder of this article is structured as follows. Section 2 discusses background related to FPGA performance prediction and resource utilization. The fundamental analyse comprising the RAT methodology are detailed in Section 3. A detailed walkthrough illustrating the usage of RAT with a real application is in Section 4. Section 5 presents additional case studies using RAT to further explore the accuracy of the performance prediction methodology. Conclusions and future work are discussed in Section 6.

2. RELATED WORK

Efficient performance modeling for algorithms and systems is an ongoing area of research for traditional parallel computing. The Parallel Random Access Machine (PRAM) [Fortune and Wyllie 1978] attempts to model the critical (and hopefully small) set of algorithm and platform attributes necessary to achieve a better understanding of the greater computational interaction and ultimately the application performance. The LogP model [Culler et al. 1993] (one successor to PRAM) abstracts the computing bandwidth, communication bandwidth, communication delay, and efficiency of coupling communication and computation. LogGP [Alexandrov et al. 1997] and additional revisions such as heterogeneous LogGP (HLogGP) [Bosque and Perez 2004] provide further modeling fidelity to LogP by addressing specific issues such as

bandwidth constraints for long messages and heterogeneous system resources, respectively. The Bulk Synchronous Parallel (BSP) [Valiant 1990] model extends PRAM to include support for architecture-independent algorithm descriptions and analyses. However, these concepts are not limited to large-scale systems of general-purpose processors. The evolution towards heterogeneous many-core devices necessitates increased modeling research and usage due to rising development time and cost. In particular, RAT seeks to leverage these ideas of maximizing model flexibility (through parameterization) and fidelity (through analytic methods).

Understanding and improving algorithm design for FPGAs is an expanding area of RC research. A performance prediction technique presented in Steffen [2007] seeks to parameterize the computational algorithm and the FPGA system itself. The analytic methods have similarities to RAT but the emphasis is on projecting potential bottlenecks due to memory throughput, not on predicting total system performance. Dynamo [Quinn et al. 2007] involves performance prediction of image processing systems partitioned and compiled at runtime from existing pipelined kernels. The system provides dynamic optimization for application construction exclusively from existing modules and assumes that efficient algorithm design and analysis is completed prior to the use of Dynamo. Herbordt et al. [2007] present 12 design techniques for maximizing the performance of FPGA applications. This research is synergistic to RAT by potential reducing the number of algorithm and architecture iterations necessary to achieve suitable performance, however the RAT methodology is still required to quantitatively evaluate each design iteration.

Simulation is another common outlet for quantifying the performance of RC application models at a high level. One technique [Smith and Peterson 2005] focuses on analytic modeling of shared heterogeneous workstations containing reconfigurable computing devices. The methodology primarily emphasizes system-level, multi-FPGA architectures with variable computational loading due to the multi-user environment. In Grobelny et al. [2007], a framework for simulation of FPGA systems and applications is built on top of the Fast and Accurate Simulation Environment (FASE). Models are created for the Mission-Level Designer (MLD) tool based on scripts of algorithm behavior to rapidly explore large-scale FPGA systems. Another simulation framework is the Hybrid System Architecture Model (HySAM) coupled with DRIVE [Bondalapati 2001]. HySAM provides mechanisms for parameterizing architectures, defining algorithms, and simulating interactions, while DRIVE provides tools for visualizing results generated by HySAM. Enzler et al. [2005], SimpleScalar and ModelSIM are combined for system analysis through simultaneous processor emulation and VHDL simulation. Another tool [Fu and Compton 2006] uses a Simics-based simulator for capturing precise memory-access patterns while functionally verifying hardware kernels. While each of these methodologies provides high-level simulation fidelity, significant cost is associated with setting up the requisite models. Either actual hardware or software code is required or effort is spent on constructing custom simulation inputs distilled from algorithm and system behavior. In contrast, RAT seeks to render performance prediction of the application and FPGA platform prior to any significant

hardware or software coding. By using analytic models instead of simulation frameworks, prediction effort is minimized while maintaining reasonable accuracy. RAT is currently scoped for single-FPGA platforms, but insight about modeling larger systems can be leveraged from the multi-FPGA simulation frameworks.

Though prediction is quite common with FPGA technologies, it is not primarily used for system-level performance. Routing is a common target for device-level prediction due to the impact on development time and performance. In Fang and Rose [2008], a model of the algorithm routing demands is created early in the FPGA development cycle. In Manohararajah et al. [2006], prediction is used to mitigate the variability and long run times of commercial place and route tools for estimating interconnect delay. Other issues including timing [Xu and Kurdahi 1999], routability [Brown et al. 1993], interconnect planning [Singh and Marek-Sadowska 2002], and routing delay [Singh et al. 2005] are explored via prediction. Performance is also explored by modeling issues such as power [Degalahal and Tuan 2005] and wafer yield [Maidee and Bazargan 2006]. Many of these prediction techniques for lower-level issues migrated from application-specific integrated circuits into the RC domain to more efficiently model the growing complexity of FPGAs. Similarly, RAT and other methodologies are branching to RC from existing areas of parallel application modeling to bridge the growing need for efficient, high-level performance prediction.

3. RC AMENABILITY TEST

Figure 1 illustrates the basic methodology behind the RC amenability test. These throughput, numerical precision, and resource tests serve as a basis for determining the viability of an algorithm design on the FPGA platform prior to any FPGA programming. Again, RAT is intended to address the performance of a specific high-level parallel algorithm mapped to a particular FPGA platform, not a generic application. The results of the RAT tests must be compared against the designer's requirements to evaluate the success of the design. Though the throughput analysis is considered the most important step, the three tests are not necessarily used as a single, sequential procedure. Often, these tests are applied iteratively during the RAT analysis until a suitable version of the algorithm is formulated or all reasonable permutations are exhausted without a satisfactory solution. The throughput test is a suitable starting-point for an application wishing to match the numerical precision and general architecture of a legacy algorithm. However, starting with the numerical precision and resources tests to refine an application prior to throughput analysis is equally viable.

3.1 Throughput

For RAT, the predicted performance of an application is defined by two terms: communication time between the CPU and FPGA, and FPGA computation time. Reconfiguration and other setup times are ignored. These two terms encompass the rate at which data flows through the FPGA and rate at which

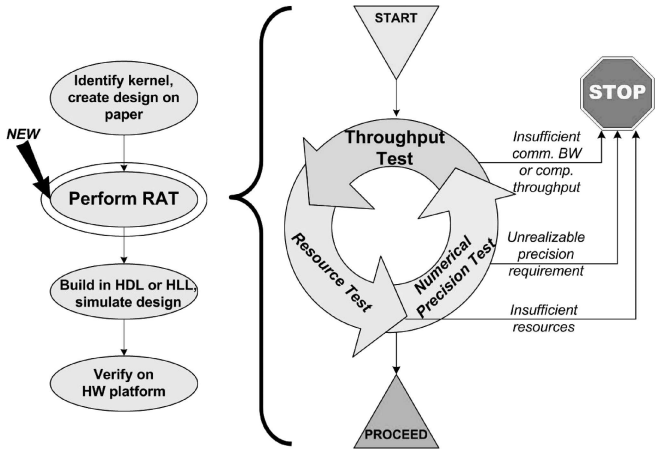

Fig. 1. Overview of RAT Methodology.

operations occur on that data, respectively. Because RAT seeks to analyze applications at the earliest stage of hardware mapping, these terms are reduced to the most generalized parameters. The RAT throughput test primarily models FPGAs as accelerators to general-purpose processors with burst communication but the framework can be adjusted for applications with streaming data.

Calculating the communication time is a relatively simplistic process given by Equations (1), (2), and (3). The overall communication time is defined as the summation of the read and write components. For the individual reads and writes, the problem size (i.e., number of data elements, $N_{elements}$) and the numerical precision (i.e., number of bytes per element, $N_{bytes/element}$) must be decided by the user with respect to the algorithm. Note that for these equations, the problem size only refers to a single block of data to be buffered by the FPGA system. All read or write communication for the application need not occur as a single transfer but can instead be partitioned into multiple blocks of data to be independently sent or received. Multiple transfers are considered in a subsequent equation. The hypothetical bandwidth of the FPGA/CPU interconnect on the target platform (e.g., 133MHz 64-bit PCI-X which has a documented maximum throughput of 1GB/s) is also necessary but is generally provided either with the FPGA subsystem documentation or as part of the interconnect standard. An additional parameter, α, represents the fraction of ideal throughput performing useful communication. The actual sustained performance of the FPGA interconnect is typically a fraction of the documented transfer rate.

$$t_{comm} = t_{read} + t_{write} \tag{1}$$

$$t_{read} = \frac{N_{elements} \cdot N_{bytes/element}}{\alpha_{read} \cdot throughput_{ideal}} \tag{2}$$

$$t_{write} = \frac{N_{elements} \cdot N_{bytes/element}}{\alpha_{write} \cdot throughput_{ideal}} . \tag{3}$$

Microbenchmarks composed of simple data transfers can be used to establish the true communication throughput. The communication times for these block transfers are measured and compared against the theoretical interconnect throughput to establish the α parameters. It is important for microbenchmarks to closely match the communication methods used by real applications on the target FPGA platform to accurately model the intended behavior. In general, microbenchmarks are performed over a wide range of possible data sizes. The resulting α values can be tabulated and used in future RAT analyses for that FPGA platform. By separating the effective throughput into the theoretical maximum and the α fraction, effects such as changing the interconnect type and efficiency can be explored separately. This fidelity is particularly useful for hypothetical or otherwise unavailable FPGA platforms.

Before further equations are discussed, it is important to clarify the concept of an "element." Until now, the expressions "problem size," "volume of communicated data," and "number of elements" have been used interchangeably. However, strictly speaking, the first two terms refer to a quantity of bytes whereas the last term has units of elements. RAT operates under the assumption that the computational workload of an algorithm is directly related to the size of the problem dataset. Because communication times are concerned with bytes and (as will be subsequently shown) computation times revolve around the number of operations, a common term is necessary to express this relationship. The element is meant to be the basic building block which governs both communication and computation. For example, an element could be a value in an array to be sorted, an atom in a molecular dynamics simulation, or a single character in a string-matching algorithm. In each of these cases, some number of bytes will be required to represent that element and some number of calculations will be necessary to complete all computations involving that element. The difficulty is establishing what subset of the data should constitute an element for a particular algorithm. Often an application must be analyzed in several separate stages, since each portion of the algorithm could interpret the input data in a different scope.

Estimating the computational component, as given in Equation (4), of the RC execution time is more complicated than communication due to the conversion factors. Whereas the number of bytes per element is ultimately a fixed, user-defined value, the number of operations (i.e., computations) per element must be manually measured from the algorithm structure. Generally, the number of operations will be a function of the overall computational complexity of the algorithm and the types of individual computations involved. Additionally, as with the communication equation, a throughput term, $throughput_{proc}$ is also included to establish the rate of execution. This parameter is meant to describe the number of operations completed per cycle. For fully pipelined designs, the number of operations per cycle will be a consistent portion of the number of operations per element, though the two terms are often equal for elements with linear computational complexity. Less optimized designs will have lower throughput, requiring multiple cycles to complete an element. Some designs may not use pipelining but process several elements per unit time via multiple parallel kernels. Again, note that computation time essentially refers

to the time required to operate on the data provided by one communication transfer. (Applications with multiple communication and computation blocks are resolved when the total FPGA execution time is computed later in this section.)

$$t_{comp} = \frac{N_{elements} \cdot N_{ops/element}}{f_{clock} \cdot throughput_{proc}} . \quad (4)$$

Despite the potential unpredictability of algorithm behavior, estimating a sufficiently precise number of operations is still possible for many types of applications. However, predicting the average rate of operation execution can be challenging even with detailed knowledge of the target hardware design. For applications with a highly deterministic pipeline, the procedure is straightforward. Throughput can be modeled as accurately as possible or adjustments can be made for optimistic or pessimistic predictions. But for interdependent or data-dependent operations, the problem is more complex. For these scenarios, a better approach would be to treat $throughput_{proc}$ as an independent variable and select a desired speedup value. Then one can solve for the particular $throughput_{proc}$ value required to achieve that desired speedup. This method provides the user with insight into the relative amount of parallelism that must be incorporated for a design to succeed. The molecular dynamics case study in Section 5 illustrates a complex algorithm where the throughput requirements are based on the desired speedup.

Similar to an element, one must also examine what is an "operation." Consider an example algorithm composed of a 32-bit addition followed by a 32-bit multiplication. The addition can be performed in a single clock cycle, but to save resources the 32-bit multiplier might be constructed using the Booth algorithm requiring 16 clock cycles. Arguments could be made that the addition and multiplication would count as either two operations (addition and multiplication) or 17 operations (addition plus 16 additions, the basis of the Booth multiplier algorithm). Either formulation is correct provided that $throughput_{proc}$ is formulated with the same assumption about the scope of an operation. Often, deterministic and highly structured algorithms are better viewed with the number of operations synonymous with the number of cycles. In contrast, complex or nondeterministic algorithms tend to be viewed as a number of abstract number of operations with an average rate of execution. Ultimately, either choice is viable and left to the preference of the user.

Figure 2 illustrates the types of communication and computation interaction to be modeled with the throughput test. Single buffering (SB) represents the most simplistic scenario with no overlapping tasks. However, a double-buffered (DB) system allows overlapping communication and computation by providing two independent buffers to keep both the processing and I/O elements occupied simultaneously. Since the first computation block cannot proceed until the first communication sequence has completed, steady-state behavior is not achievable until at least the second iteration. However, this startup cost is considered negligible for a sufficiently large number of iterations.

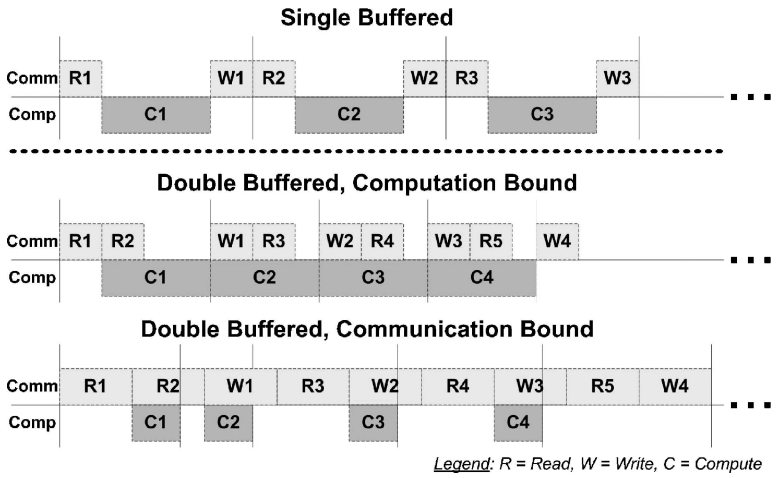

Fig. 2. Example Overlap Scenarios.

The FPGA execution time, t_{RC}, is a function not only of the t_{comm} and t_{comp} terms but also the amount of overlap between communication and computation. Equations (5) and (6) model both SB and DB scenarios. For SB, the execution time is simply the summation of the communication time, t_{comm}, and computation time, t_{comp}. With the DB case, either the communication or computation time completely overlaps the other term. The smaller latency essentially becomes hidden during steady state. The DB case is included for completeness of the RAT model, however the case studies focus on the SB scenario.

The RAT analysis for computing t_{comp} primarily assumes one algorithm "functional unit" operating on a single buffer's worth of transmitted information. The parameter N_{iter} is the number of iterations of communication and computation required to solve the entire problem.

$$t_{RC_{SB}} = N_{iter} \cdot (t_{comm} + t_{comp}) \tag{5}$$

$$t_{RC_{DB}} \approx N_{iter} \cdot Max(t_{comm}, t_{comp}) \, . \tag{6}$$

Assuming that the application design currently under analysis was based upon available sequential software code, a baseline execution time, t_{soft}, is available for comparison with the estimated FPGA execution time to predict the overall speedup. As given in Equation (7), speedup is a function of the total application execution time, not a single iteration.

$$speedup = \frac{t_{soft}}{t_{RC}} \, . \tag{7}$$

Related to the speedup is the computation and communication utilization given by Equations (8), (9), (10), and (11). These metrics determine the fraction of the total application execution time spent on computation and communication for the SB and DB cases. For SB, the computation utilization can provide additional insight about the application speedup. If utilization is high,

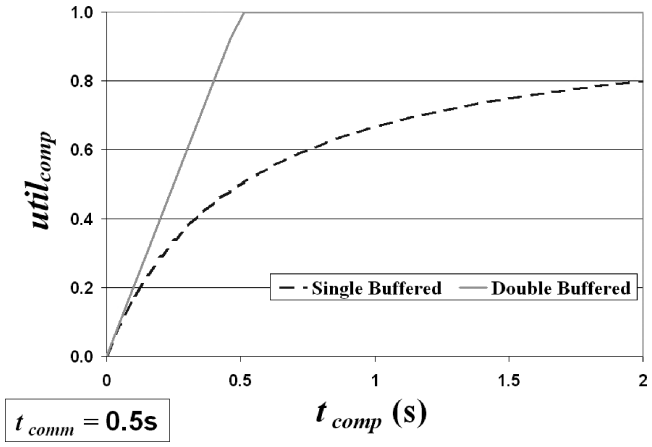

Fig. 3. Trends for computational utilization in SB and DB scenarios.

the FPGA is rarely idle thereby maximizing speedup. Low utilizations can indicate potential for increased speedups if the algorithm can be reformulated to have less (or more overlapped) communication. In contrast to computation that is effectively parallel for optimal FPGA processing, communication is serialized. Whereas computation utilization gives no indication about the overall resource usage, since additional FPGA logic could be added to operate in parallel without affecting the utilization, the communication utilization indicates the fraction of bandwidth remaining to facilitate additional transfers since the channel is only a single resource. For DB, assuming steady-state behavior, the implications of the utilization terms are slightly different. The larger value, whether communication or computation, will have a utilization of 1. If computation is the shorter (i.e., overlapped) time, utilization illustrates how starved the computation is for data. If communication is the shorter time, utilization is a measure of the available throughput to support additional parallel computation. An example of these utilization trends is shown in Figure 3.

$$util_{comp_{SB}} = \frac{t_{comp}}{t_{comm} + t_{comp}} \tag{8}$$

$$util_{comm_{SB}} = \frac{t_{comm}}{t_{comm} + t_{comp}} \tag{9}$$

$$util_{comp_{DB}} = \frac{t_{comp}}{Max(t_{comm}, t_{comp})} \tag{10}$$

$$util_{comm_{DB}} = \frac{t_{comm}}{Max(t_{comm}, t_{comp})} \; . \tag{11}$$

3.2 Numerical Precision

Application numerical precision is typically defined by the amount of fixed- or floating-point computation within a design. With FPGA devices, where

increased precision dictates higher resource utilization, it is important to use only as much precision as necessary to remain within acceptable tolerances. Because general-purpose processors have fixed-length data types and readily available floating-point resources, it is reasonable to assume that often a given software application will have at least some measure of wasted precision. Consequently, effective migration of applications to FPGAs requires a method to determine the minimum necessary precision before any translation begins.

While formal methods for numerical precision analysis of FPGA applications are important, they are outside the scope of this paper. A plethora of research exists on topics including maintaining precision with mixed data types [Buttari et al. 2007], automated conversion of floating-point software programs to fixed-point hardware designs [Banerjee et al. 2003], design-time precision analysis tools for RC [Chang and Hauck 2002], and custom or dynamic bitwidths for maximizing performance and area on FPGAs [Bondalapati and Prasanna 1999; Gaffar et al. 2002; Perri et al. 2004; Wang et al. 2006]. Application designs are meant to capitalize on these numerical precision techniques and then use the RAT methodology to evaluate the resulting algorithm performance. Numerical precision must also be balanced against the type and quantity of available FPGA resources to support the desired format. For example, 18-bit fixed point may be used in Xilinx FPGAs since it maximizes usage of single 18-bit embedded multipliers. As with parallel decomposition, numerical formulation is ultimately the decision of the application designer. RAT provides a quick and consistent procedure for evaluating these design choices.

3.3 Resources

By measuring resource utilization, RAT seeks to determine the scalability of an application design. Empirically, most FPGA designs will be limited in size by the availability of three common resources: on-chip memory, hardcore functional units (e.g., fixed multipliers), and basic logic elements (i.e., look-up tables and flip-flops).

On-chip RAM is readily measurable since some quantity of the memory will likely be used for I/O buffers of a known size. Additionally, intra-application buffering and storage must be considered. Vendor-provided wrappers for interfacing designs to FPGA platforms can also consume a significant number of memories but the quantity is generally constant and independent of the application design.

Although the types of dedicated functional units included in FPGAs can vary greatly, the hardware multiplier is a fairly common component. The demand for dedicated multiplier resources is highlighted by the availability of families of chips (e.g., Xilinx Virtex-4 and -5 SX series) with extra multipliers versus other comparably sized FPGAs. Quantifying the necessary number of hardware multipliers is dependent on the type and amount of parallel operations required. Multipliers, dividers, square roots, and floating-point units use hardware multipliers for fast execution. Varying levels of pipelining and other design choices can increase or decrease the overall demand for these

resources. With sufficient design planning, an accurate measure of resource utilization can be taken for a design given knowledge of the architecture of the basic computational kernels.

Measuring basic logic elements is the most common resource metric. High-level designs do not empirically translate into any discernible resource count. Qualitative assertions about the demand for logic elements can be made based upon approximate quantities of arithmetic or logical operations and registers. But a precise count is nearly impossible without an actual hardware description language (HDL) implementation. Above all other types of resources, routing strain increases exponentially as logic element utilization approaches maximum. Consequently, it is often unwise (if not impossible) to fill the entire FPGA.

Currently, RAT does not employ a database of statistics to facilitate resource analysis of an application for complete FPGA novices. The usage of RAT requires some vendor-specific knowledge (e.g., single-cycle 32-bit fixed-point multiplications with 64-bit resultants on Xilinx Virtex-4 FPGAs require four dedicated 18-bit multipliers). Additionally, the user must consider trade-offs such as using fixed resources versus logic elements and computational logic versus lookup tables. Resource analyses are meant to highlight general application trends and predict scalability. For example, the structure of the molecular dynamics case study in Section 5 is designed to minimize RAM usage and the parallelism is ultimately limited by the availability of multiplier resources.

3.4 Scope of RAT

The analytic model described in Section 3.1 establishes the basic scope of RAT as a strategic design methodology to formulate predictions about algorithm performance and RC amenability. RAT is intended to support a diverse collection of platforms and application fields because the methodology focuses on the common structures and determinism within the algorithm. Communication and computation are related to the number of data elements in the algorithm. Effective usage of the performance prediction models requires mitigation of variabilities in the algorithm structure such as data-driven computation. Based on the complexity of the algorithm and architecture, the RAT model may be used to directly predict performance or instead establish minimum throughput requirements based on the desired speedup. RAT currently targets systems with a single CPU and FPGA as a first step towards a broad RC methodology. The FPGA device is considered a coprocessor to the CPU but can initiate some operations independently such as DMA. Even for single-FPGA systems, a range of issues related to parallelism and scalable can be explored. RAT is scoped to make a convenient and impactful model that not only integrates broader issues such as numerical precision and resource utilization but also contribute to the larger goal of better parallel algorithm formulation and design-space exploration. Future research will expand the RAT methodology for larger scale prediction on multi-FPGA systems.

4. WALKTHROUGH

To simplify the RAT analysis in Section 3, a worksheet can be constructed based upon Equations (1) through (11). Users simply provide the input parameters and the resulting performance values are returned. This walkthrough further explains key concepts of the throughput test by performing a detailed analysis of a real application case study, one-dimensional probability density function (PDF) estimation. The goal is to provide a more complete description of how to use the RAT methodology in a practical setting.

4.1 Algorithm Architecture

The Parzen window technique [Nagarajan et al. 2008] is a generalized non-parametric approach to estimating probability density functions (PDFs) in a d-dimensional space. The common parametric forms of PDFs (e.g., Gaussian, Binomial, Rayleigh distributions) represent mathematical idealizations and, as such, are often not well matched to densities encountered in practice. Though more computationally intensive than using histograms, the Parzen window technique is mathematically advantageous. For example, the resulting probability density function is continuous therefore differentiable. The computational complexity of the algorithm is of order $O(Nn^d)$ where N is the total number of data samples (i.e., number of elements), n is the number of discrete points at which the PDF is estimated (comparable to the number of "bins" in a histogram), and d is the number of dimensions. A set of mathematical operations are performed on every data sample over n^d discrete points. Essentially, the algorithm computes the cumulative effect of every data sample at every discrete probability level. For simplicity, each discrete probability level is subsequently referred to as a bin.

In order to better understand the assumptions and choices made during the RAT analysis, the chosen algorithm for PDF estimation is highlighted in Figure 4. A total of 204,800 data samples are processed in batches of 512 elements against 256 bins. Eight separate pipelines are created to process data samples with respect to a particular subset of bins. Each data sample is an element with respect to the RAT analysis. The data elements are fed into the parallel pipelines sequentially. Each pipelined unit can process one element with respect to one bin per cycle. Internal registering for each bin keeps a running total of the impact of all processed elements. These cumulative totals comprise the final estimation of the PDF function.

4.2 RAT Input Parameters

Table I provides a list of all the input parameters necessary to perform a RAT analysis. The parameters are sorted into four distinct categories, each referring to a particular portion of the throughput analysis. Note that $N_{elements}$ is listed under a separate category when it is used by both communication and computation. It is assumed that the number of elements dictating the computation volume is also the number of elements that are input to the application (although the effective bit-widths may differ due to the fixed width of the communication channel). While applications can exhibit unusual computational

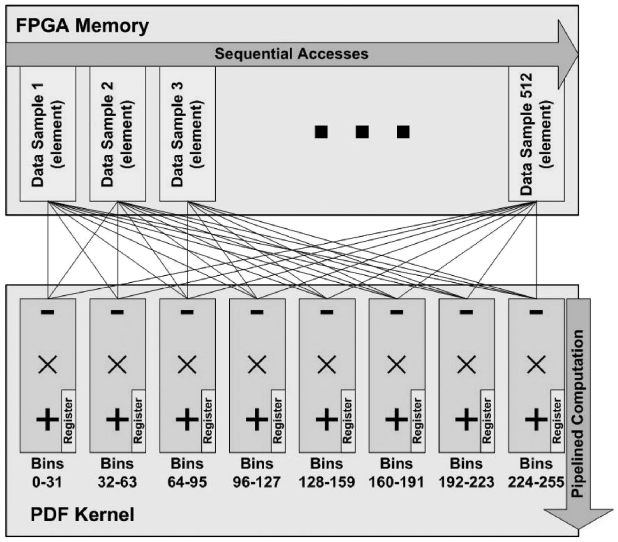

Fig. 4. Parallel algorithm and mapping for 1-D PDF.

Table I. Input Parameters for RAT

Dataset Parameters	
$N_{elements}$, input	(elements)
$N_{elements}$, output	(elements)
$N_{bytes/element}$	(bytes/element)
Communication Parameters	
$throughput_{ideal}$	(MB/s)
α_{write}	$0 < \alpha < 1$
α_{read}	$0 < \alpha < 1$
Computation Parameters	
$N_{ops/element}$	(ops/element)
$throughput_{proc}$	(ops/cycle)
f_{clock}	(MHz)
Software Parameters	
t_{soft}	(sec)
N_{iter}	(iterations)

trends or require significant amounts of additional data (e.g., constants, seed values, or lookup tables), these instances may be considered uncommon. Alterations can be made to account for uncorrelated communication and computation but such examples are not included in this paper.

Table II summarizes the input parameters for RAT analysis of the specified algorithm for 1-D PDF estimation. The dataset parameters are generally the first values supplied by the user, since the number of elements will ultimately govern the entire algorithm performance. Though the entire application involves 204,800 data samples, each iteration of the 1-D PDF estimation will involve only a portion, 512 data samples, or 1/400 of the total set. This algorithm effectively consumes all of the input values. Only one cumulative value is left after each iteration per bin but these results are retained on the FPGA. Values are only transferred back to the host after computation for all iterations

Table II. Input Parameters of 1-D PDF

Dataset Parameters		
$N_{elements}$, input	(elements)	512
$N_{elements}$, output	(elements)	1
$N_{bytes/element}$	(bytes/element)	4
Communication Parameters (Nallatech)		
$throughput_{ideal}$	(MB/s)	1000
α_{write}	$0 < \alpha < 1$	0.099
α_{read}	$0 < \alpha < 1$	0.001
Computation Parameters		
$N_{ops/element}$	(ops/element)	768
$throughput_{proc}$	(ops/cycle)	20
f_{clock}	(MHz)	75/100/150
Software Parameters		
t_{soft}	(sec)	0.578
N_{iter}	(iterations)	400

is complete. The final output communication must be represented as individual partial transfers, one per iteration, to correspond with the RAT model, but the throughput efficiency is adjusted to correspond with a single block of data.

The number of bytes per element, $N_{bytes/element}$, is rounded to four (i.e., 32 bits). Even though the PDF estimation algorithm only uses 18-bit fixed point, the interconnect uses 32-bit communication. The data was not byte-packed and the remaining 14 bits per word of communication are unused. During the algorithmic formulation, several formats including 18-bit fixed point, 32-bit fixed point, and 32-bit floating point were considered for use in the PDF algorithm. However, the maximum error percentage was found to be only 3.8% for 18-bit fixed point, which is satisfactory precision for the application. Ultimately 18-bit fixed point was chosen so that only one Xilinx 18×18 multiple-accumulate (MAC) unit would be needed per multiplication. Though slightly smaller bitwidths also had reasonable error constraints, no performance gains or appreciable resource savings would have been achieved.

The communication parameters are provided by the user since they are merely a function of the target RC platform, which is a Nallatech H101-PCIXM card containing a Virtex-4 LX100 user FPGA for this case study. The card is connected to the host CPU via a 133MHz PCI-X bus which has a theoretical maximum bandwidth of 1GB/s. The α parameters were computed using a microbenchmark consisting of a read and write for data sizes comparable to those used by the 1-D PDF algorithm. The resulting read and write times were measured, combined with the transfer size to compute the actual communicate rates, and finally used to calculate the α parameters by dividing by the theoretical maximum. The α parameters for the target FPGA platform are low due to communication protocols and middleware used by Nallatech atop PCI-X and high latencies associated with the small 2KB (512×4B) transfers.

The computation parameters are the more challenging portion of RAT performance prediction, but are still simplistic given the deterministic behavior of PDF estimation. As mentioned earlier, each element that comes into the PDF estimator is evaluated against each of the 256 bins. Each computation requires 3 operations: comparison (subtraction), multiplication, and addition.

Therefore, the number of operations per element totals 768 (i.e., 256 × 3). This particular algorithm structure has 8 pipelines that each perform 3 operations per cycle for a total of 24. However, this value is conservatively rounded down to 20 to account for implementation details such as pipeline latency and computation overhead. This conservative parameter was selected *prior* to the algorithm coding and has *not* (nor has any parameter for any case study) been adjusted for any "fudge factor" created from runtime data. Preimplementation adjustments to the RAT parameters such as reducing the throughput value are not required but are sometimes useful to create more optimistic or pessimistic predictions and account for application- or platform-specific behaviors not modeled by RAT. Similarly, a range of throughput values could be examined to explore the effect on performance when the implementation is better or worse than expected. However, this case study focuses on a specific value for each parameter to validate the RAT model.

While previous parameters could be reasonably inferred from the deterministic structure of the algorithm, a priori estimation of the required clock frequency is very difficult. Empirical knowledge of FPGA platforms and algorithm design practices provides some insight as to a range of likely values. However, attaining a single, accurate estimate of the maximum FPGA clock frequency achieved is generally impossible until after the entire application has been converted to a hardware design and analyzed by an FPGA vendor's layout and routing tools. Consequently, a number of clock values ranging from 75MHz to 150MHz for the LX100 are used to examine the scope of possible speedups.

The software parameters provide the last piece of information necessary to complete the speedup analysis. The software execution time of the algorithm is provided by the user. Often, software legacy code is the basis for the hardware migration initiative. FPGA development could be based directly on mathematical models, but there would be no baseline for evaluating speedup. The baseline software for the 1-D PDF estimation was written in C, compiled using gcc, and executed on a 3.2 GHz Xeon. Lastly, the number of iterations is deduced from the portion of the overall problem to reside in the FPGA at any one time. Since the user decided to only process 512 elements at a time from the set of 204800 element set, there must be 400 (i.e., 204800/512) iterations of the algorithm. The case study is implemented in VHDL.

4.3 Predicted and Actual Results

The RAT performance numbers are compared with the experimentally measured results in Table III. Each predicted value in the table is computed using the input parameters and equations listed in Section 3.1. For example, the predicted computation time when f_{clk} =150MHz is calculated as follows:

$$t_{comp} = \frac{512 \text{ elements} \cdot 768 \text{ ops/element}}{150 \text{ MHz} \cdot 20 \text{ ops/cycle}} = \frac{393216 \text{ ops}}{3\text{E+9 ops/sec}} = 1.31\text{E-4 secs}.$$

Table III. Performance Parameters of 1-D PDF (Nallatech)

	Predicted	Predicted	Predicted	Actual
f_{clk} (MHz)	75	100	150	150
t_{comm} (sec)	2.47E-5	2.47E-5	2.47E-5	2.50E-5
t_{comp} (sec)	2.62E-4	1.97E-4	1.31E-4	1.39E-4
$util_{comm_{SB}}$	9%	11%	16%	15%
$util_{comp_{SB}}$	91%	89%	84%	85%
$t_{RC_{SB}}$ (sec)	1.15E-1	8.85E-2	6.23E-2	7.45E-2
speedup	5.0	6.5	9.3	7.8

The communication time is computed using the corresponding equation. Because the application is single-buffered, the total RC execution time is simply:

$$\begin{aligned} t_{RC_{SB}} &= 400 \text{ iterations} \cdot (2.47\text{E-}5 \text{ secs} + 1.31\text{E-}4 \text{ secs}) \\ &= 6.23\text{E-}2 \text{ secs} . \end{aligned}$$

The speedup is simply the division of the software execution time by the RC execution time. The utilization is computed using the corresponding SB equations.

The communication and computation times for the actual FPGA code were measured using the wall-clock time of the CPU. The error in the prediction of the communication time was minimal, approximately 1%, due to detailed microbenchmarking for these exact transfer sizes. A relatively accurate t_{comp} prediction is expected given the deterministic structure of the parallel algorithm. However, the high degree of accuracy (two significant figures) between the predicted and actual computation times with f_{clk} =150MHz was unusual, since the computational throughput was a conservatively estimated parameter. Much of the 1-D PDF algorithm is pipelined but the lower effective throughput, due to the latency in the short 0.14ms of computation time, closely matched the conservatively estimated value for $throughput_{proc}$ (i.e., the 20 ops/cycle used in the RAT prediction instead of the theoretical 24). Also, this lower throughput accounted for extra overhead time involved with polling the FPGA for completion of computation.

The total execution time for the FPGA is also measured using the wall-clock time, rather that calculated from Equation (5), to ensure maximum accuracy. Additional factors may be present in the total time that are not accounted in the individual communication and computation. In this case study, the total error was 16% but the comunication and computation errors were only 1% and 6%, respectively. The discrepancy is due to overheads with managing and regulating data transfers by the host CPU that are not expressly part of the individual RAT models. The impact of these overheads and other synchronization issues will vary depending upon the particular FPGA platform and size of the overhead time relative to the total RC execution time. For 1-D PDF, the extra 8.5ms was significant compared to the 75ms of execution time. The relatively low resource usage in Table IV illustrates a potential for further speedup by including additional parallel kernels albeit at the risk of increasing the impact of the system overhead.

Table IV. Resource Usage of 1-D PDF (XC4VLX100)

FPGA Resource	Utilization
BRAMs	15%
48-bit DSPs	8%
Slices	16%

5. ADDITIONAL CASE STUDIES

Several case studies are presented as further analysis and validation of the RAT methodology: 2-D PDF estimation, coordinate calculation for LIDAR processing, the traveling salesman problem, and molecular dynamics. Two-dimensional PDF estimation continues to illustrate the accuracy of RAT for algorithms with a deterministic structure. Coordinate calculation uses prediction on a communication-bound algorithm. Traveling salesman explores a computation-bound searching algorithm with pipelined structure. However, the molecular dynamics application serves as a counterpoint given the relative difficulty of encapsulating its data-driven, non-deterministic computations. A diverse collection of vendor platforms, Nallatech, Cray, SRC, and XtremeData, is used for 2-D PDF, LIDAR, traveling salesman, and molecular dynamics, respectively. Each of these case studies has single-buffered communication and computation. As with one-dimensional PDF estimation, the design emphasis is placed on throughput analyses because the overall goal is to minimize execution time for these designs.

These RAT case studies represent a range of experiences with estimating computational throughput based on different user backgrounds and prediction emphases. Consequently, 2-D PDF estimation, LIDAR, and traveling salesman focus on more exact throughput parameterization in contrast to the conservative prediction in 1-D PDF. However, the performance of molecular dynamics could not be reliably estimated prior to implementation because of the difficulty of analyzing the complex and data-dependent algorithm structure as described by a high-level language. Instead, the target throughput is computed from the speedup requirements. While this prediction will be inaccurate if the minimum throughput is unrealizable, the RAT estimation provides a starting point for implementation and insight about the performance ramifications of a suboptimal architecture.

5.1 2-D PDF Estimation

As previously discussed, the Parzen window technique is applicable in an arbitrary number of dimensions [Nagarajan et al. 2008]. However, the two-dimensional case presents a significantly greater problem in terms of communication and computation volume than the original 1-D PDF estimate. Now 256×256 discrete bins are used for PDF estimation and the input data set is effectively doubled to account for the extra dimension. The basic computation per element grows from $(N-n)^2 + c$ to $((N_1 - n_1)^2 + (N_2 - n_2)^2 + c$ where N_1 and N_2 are the data sample values and n_1, n_2 are the probability levels for each dimension, and c is a probability scaling factor. But despite the added complexity, the increased quantity of parallelizable operations intuitively makes

Table V. Input Parameters of 2-D PDF

Dataset Parameters		
$N_{elements}$, input	(elements)	1024
$N_{elements}$, output	(elements)	65536
$N_{bytes/element}$	(bytes/element)	4
Communication Parameters (Nallatech)		
$throughput_{ideal}$	(MB/s)	1000
α_{write}	$0 < \alpha < 1$	0.147
α_{read}	$0 < \alpha < 1$	0.026
Computation Parameters		
$N_{ops/element}$	(ops/element)	196608
$throughput_{proc}$	(ops/cycle)	48
f_{clock}	(MHz)	75/100/150
Software Parameters		
t_{soft}	(sec)	158.8
N_{iter}	(iterations)	400

Table VI. Performance Parameters of 2-D PDF (Nallatech)

	Predicted	Predicted	Predicted	Actual
f_{clk} (MHz)	75	100	150	100
t_{comm} (sec)	1.01E-2	1.01E-2	1.01E-2	1.06E-2
t_{comp} (sec)	5.59E-2	4.19E-2	2.80E-2	4.46E-2
$util_{comm_{SB}}$	15%	19%	27%	19%
$util_{comp_{SB}}$	85%	81%	73%	81%
$t_{RC_{SB}}$ (sec)	2.64E+1	2.08E+1	1.52E+1	2.21E+1
speedup	6.0	7.6	10.4	7.2

this algorithm amendable to the RC paradigm, assuming sufficient quantities of hardware resources are available.

Table V summarizes the input parameters for the RAT analysis for our 2-D PDF estimation algorithm. Again, the computation is performed in a two-dimensional space, so twice the number of data samples are sent to the FPGA. In contrast to the 1-D case, the 65,535 (256×256) PDF values are sent back to the host processor after each iteration of computation due to memory size constraints on the FPGA. The same numerical precision of four bytes per element is used for the dataset. The interconnect parameters model the same Nallatech FPGA card as in the 1-D case study but for different transfer sizes. The α_{read} term is small for the relatively large output of 65,536 elements because data is transferred in 256 batches of 256 elements each, incurring a large latency overhead. Each of the 65,536 bins requires three operations for a total of 196,608 operations. Eight kernels, each containing two pipelines (one per dimension), perform three operations per pipeline per cycle for a total of 48 simultaneous computations per cycle. Again, the same range of clock frequencies is used for comparison. The software baseline for computing speedup values was written in C and executed on the same 3.2GHz Xeon processor. The algorithm requires the same 400 iterations to complete the computation and VHDL is also used.

The RAT performance predictions are listed with the experimentally measured results in Table VI. These three predictions are based on the range of clock frequency values listed in Table V, but the accuracy of the actual 100MHz design is only evaluated against the comparable 100MHz prediction. The

Table VII. Resource Usage of 2-D PDF (XC4VLX100)

FPGA Resource	Utilization
BRAMs	21%
48-bit DSPs	33%
Slices	22%

communication time is within 5% of the predicted value with a discrepancy of 0.5 milliseconds again due to accurate microbenchmarking of the Nallatech board's PCI-X interface. This difference is potentially significant given the 400 iterations required to perform this algorithm. The overall impact on speedup is further affected by variation in the computation time. An underestimation by approximately 2.7 milliseconds creates a total discrepancy just over 3 milliseconds per iteration. This larger error in the computational throughput parameter as compared to 1-D PDF is due to the more exact modeling of the pipeline behavior without adjustments for potential overhead. These overheads from pipeline latency and polling were assumed insignificant due to the length of the overall execution time but instead had noticeable effect each iteration. In total, the speedup was 6% less than the predicted speedup. This error margin is excellent given the fast and coarse-grained prediction approach of RAT compounded over hundreds of iterations. Greater attention to communication behavior and the nuances of the computation structure can further reduce this error if desired. To the extent possible while maintaining fast performance estimation, insight about shortcomings in previous RAT predictions can be factored into future projects to further boost accuracy. Comparing Table VII to the resource utilization from the 1-D algorithm, the hardware usage has increased but still has not nearly exhausted the resources of the FPGA. Additional parallelism could be exploited to improve the performance of the 2-D algorithm if additional revisions are desired.

5.2 Coordinate Calculation for LIDAR

Airborne light detection and ranging (LIDAR) [Shih et al. 2008] is emerging as an important remote sensing modality for providing high-resolution position information on targets of interest, primarily ground-based features such as terrain topology, from long distances. LIDAR processing begins with the calculation of the Cartesian coordinates of the LIDAR targets based upon the travel time of the laser beam (i.e., range ρ), angle of the scan (θ), GPS (Global Positioning System) coordinates of the airplane (X_{ac}, Y_{ac}, Z_{ac}), and aircraft attitude (roll ϕ_r, pitch ϕ_p, and yaw ϕ_y). For this case study, the laser continuously provides target range information at a rate of 33kHz while the GPS coordinates only update at 1Hz and the aircraft attitude at 5Hz. Interpolation is used to map each LIDAR return with a specific GPS coordinate and aircraft attitude.

For this algorithm, coordinate calculation is constructed as one pipeline on a single node of a Cray XD1 system consisting of a Xilinx XC2VP50 user FPGA connected to a host Opteron processor via the RapidArray interconnect. The algorithm is comprised of six steps for each LIDAR return. First, interpolation of the GPS coordinates and aircraft attitude is performed for the particular return value. Second, the unit vector between the laser source and the target

Table VIII. Input Parameters of LIDAR

Dataset Parameters		
$N_{elements}$, input	(elements)	33000
$N_{elements}$, output	(elements)	33000
$N_{bytes/element}$	(bytes/element)	8
Communication Parameters (Cray)		
$throughput_{ideal}$	(MB/s)	1600
α_{write}	$0 < \alpha < 1$	0.5
α_{read}	$0 < \alpha < 1$	0.5
Computation Parameters		
$N_{ops/element}$	(ops/element)	1
$throughput_{proc}$	(ops/cycle)	1
f_{clock}	(MHz)	100/125/150
Software Parameters		
t_{soft}	(sec)	0.011
N_{iter}	(iterations)	1

is computed from the scan angle. The third and fourth steps generate three rotation matrices to align the aircraft attitude with the GPS coordinate and apply the matrices to the unit vector, respectively. Fifth, a range vector is produced by scaling the rotated unit vector by the target's range. Sixth, the range vector is translated into the coordinate space relative to the aircraft GPS position. Equation (12) summarizes the mathematical computations of the steps where S and C abbreviate to sine and cosine operations, respectively.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \rho(C\phi_y C\phi_r S\theta - C\phi_y S\phi_r C\phi_p S\theta - S\phi_y S\phi_r S\theta)) + X_{ac} \\ \rho(S\phi_y C\phi_r S\theta - S\phi_y S\phi_r C\phi_p C\theta - C\phi_y S\phi_r C\theta)) + Y_{ac} \\ \rho(-S\phi_r S\theta - C\phi_r C\phi_p C\theta) + Z_{ac} \end{bmatrix}. \quad (12)$$

Table VIII summarizes the RAT input parameters of the algorithm for coordinate calculation. The input date size of 33,000 elements is based on one second of LIDAR returns (i.e., the time between GPS updates). A corresponding number of GPS coordinates is returned by the calculations. The X, Y, and Z dimensions of the LIDAR returns and GPS coordinates each use a 16-bit fixed-point format. A total of 48 bits is sent using the 64-bit (8-byte) RapidArray interconnect. This channel has a documented theoretical throughput of 1.6GB/s per direction but microbenchmarking indicates only half the rate is achievable for these data transfers. Because the computation is pipelined, the number of operations per element is synonymous with the number of elements. The pipeline can process one operation (i.e., element) per cycle. The exact depth of the pipeline is not known a priori but the extra latency is presumed negligible when compared to the size of the dataset. A range of clock frequencies is examined to predict the scope of the overall speedup. The software baseline was written in C and executed on a 2.4GHz Opteron processor, the host CPU for the Cray XD1 node. Only one iteration (i.e., GPS interval) is required for this case study and VHDL is used to implement the parallel algorithm in hardware.

Table IX compares the RAT performance predictions with the actual 125MHz experimental results. The structure of the particular Cray SRAM interface overlaps computation and DMA transfers back to the CPU (i.e., t_{read}).

Table IX. Performance Parameters of LIDAR (Cray)

	Predicted	Predicted	Predicted	Actual
f_{clk} (MHz)	100	125	150	125
t_{comm} (sec)	6.60E-4	6.60E-4	6.60E-4	5.65E-4
t_{comp} (sec)	3.30E-4	2.64E-4	2.20E-4	2.25E-4
$util_{comm_{SB}}$	33%	29%	25%	28%
$util_{comp_{SB}}$	67%	71%	75%	72%
$t_{RC_{SB}}$ (sec)	9.90E-4	9.24E-4	8.80E-4	7.90E-4
speedup	11.0	11.8	12.4	13.8

Table X. Resource Usage of LIDAR (XC2VP50)

FPGA Resource	Utilization
BRAMs	12%
18x18 Multipliers	5%
Slices	45%

Consequently, the total RC execution time was directly measurable but the individual computation and communication times for the actual result were estimated from the total execution time based on the expected latency of the computation. Two general conclusions were that both the computation and communication times were overestimated by RAT but that the utilization ratios were still fairly consistent with expectations. Unlike the previous case studies, the total speedup was *underestimated* by 16%. This discrepancy in speedup was primarily due to the difference in communication times. The actual computation pipeline is believed to correspond closely with the high-level algorithm. The communication and comptution times of 565μs and 225μs are likely comparable to the system overhead and measurement error causing noticable discrepancies and the unusual behavior of a pessimistic prediction even with the generalized analytic model. Though extra performance as compared to RAT projections may be an unexpected benefit for the algorithm, the goal of the methodology is precise prediction that considers all major factors to performance. Adjustments to the model for more accurate prediction of short communication and computation may be necessary. Table X highlights the availability of unused resources to expand the parallel computation but the benefit will be marginal because of the communication-bound algorithm.

5.3 Traveling Salesman Problem

The traveling salesman problem (TSP) [Tschoke et al. 1995] is a particular version of the NP-complete Hamiltonian path problem that locates the minimum length path through an undirected, weighted graph in which each vertex (i.e., city) is visited exactly once. (Other derivations of the Hamiltonian path problem include the snake-in-the-box, knight's tour, and the Lovász conjecture.) For this algorithm, any city may be the starting point and all cities are connected to every other city creating $N!$ potential Hamiltonian paths, where N is the number of cities. To accelerate the time to converge on a solution, heuristics are sometimes employed to systematically search a subset of the solution space. However, the algorithm for this case study performs an exhaustive search on all paths in the graph. The specific algorithm formulation

Table XI. Input Parameters of TSP

Dataset Parameters		
$N_{elements}$, input	(elements)	81
$N_{elements}$, output	(elements)	1
$N_{bytes/element}$	(bytes/element)	8
Communication Parameters (SRC)		
$throughput_{ideal}$	(MB/s)	1400
α_{write}	$0 < \alpha < 1$	0.03
α_{read}	$0 < \alpha < 1$	0.03
Computation Parameters		
$N_{ops/element}$	(ops/element)	4782969
$throughput_{proc}$	(ops/cycle)	9
f_{clock}	(MHz)	100
Software Parameters		
t_{soft}	(sec)	2.22
N_{iter}	(iterations)	1

has significant ramifications not only on the hardware performance but also on the prediction accuracy.

The case study targets SRC Computer's SRC-6 FPGA platform. Within the SRC-6, the algorithm uses one of the Xilinx XC2V6000 user FPGAs in a single MAP-B unit. The FPGA is connected to a host processor via the vendor's SNAP (memory DIMM slot) interconnect. Nine depth-first traversals of the graph occur simultaneously on a single FPGA starting from each of the nine different cities. Techniques such as branch and bound are not used because each step of the search would be dependent on the previous steps, thus preventing any pipelining. Instead, the algorithm starts with selecting N arbitrary cities (and their $N-1$ edges) all at once and is then followed by determination of path validity and length (i.e., if all cities were visited exactly once, report the total distance traveled). The individual steps are not interrelated and the examination of possible paths can be pipelined. However, unlike the branch-and-bound technique which backtracks in the middle of paths to avoid revisiting cities, the hardware pipeline operates on full N-length paths, even those invalid because of repeated cities. Extra computation is required but substantially more parallelism is exploitable.

Table XI lists the input parameters of the RAT performance prediction for TSP. The interconnect parameters model the proprietary SNAP interconnect of the SRC-6 system. The small fraction of throughput, α, represents the overhead associated with the extremely minimal communication in the algorithm, only $N \times N$ input elements. This information contains the distances between every pair of cities. The only output for this system is the minimal path length and this communication time is assumed to be negligible. Elements are 8 bytes, the width of the MAP-B's SRAM, but only 4 bytes (32 bits) per element are used to represent distances in fixed point. The information is not byte-packed for communication and consequently the other 32 bits are wasted. For this case study, the computational workload is exponentially related to the number of elements. While N^2 distances are need to compute path lengths, N^N total paths must be examined. For consistency with the other case studies, the number of operations per element is set to N^{N-2} (i.e., $9^7 = 4782969$), which

Table XII. Performance Parameters of TSP (SRC)

	Predicted	Actual
f_{clk} (MHz)	100	100
t_{comm} (sec)	1.54E-5	1.57E-5
t_{comp} (sec)	4.31E-1	4.30E-1
$util_{comm_{SB}}$	0.004%	0.003%
$util_{comp_{SB}}$	99.996%	86.2%
$t_{RC_{SB}}$ (sec)	4.31E-1	4.99E-1
speedup	5.16	4.45

Table XIII. Resource Usage of TSP (XC2V6000)

FPGA Resource	Utilization
BRAMs	56%
18x18 Multipliers	0%
Slices	73%

makes the RAT prediction computationally equivalent to the view of N^N path elements (for computation only) with one operation each. Since nine cities are examined in this case study using nine kernels, a total of nine potential paths are examined per clock cycle. The clock frequency of the MAP-B unit is fixed at 100 MHz and only one input/compute/output iteration is required for this algorithm. The C software baseline was executed on a 3.2GHz Xeon processor. The parallel algorithm is constructed in SRC's Carte C, a high-level language (HLL) for FPGA design.

The results of the hardware design are compared against the performance predictions in Table XII. The percent error in the predicted communication time was less than 2% due to microbenchmarking on the SRC-6 specifically to replicate the short communication transfers. The cycle-accurate timers of the SRC-6 system meant this discrepancy was not a measurement error but instead a function of the modeling and parameterization of the SNAP interconnect throughput. However, the actual communication time is only 16μs (versus 430ms for computation) and consequently its impact on speedup is negligible for this case study. The predicted and actual computation times were nearly identical due to deterministic, pipelined structure. For the SRC-6 system, the individual communication and computation times were measured on the FPGA via a vendor-provided counter function. Both operations are FPGA-controlled and initiated by a single function call which could not be separated from the perspective of the host microprocessor. However, the total RC execution as measured by the wall-clock time of the CPU is approximately 0.07 seconds longer than the sum of the computation and communication times measured on the FPGA. Consequently, extra system overhead not considered by RAT caused the actual speedup value to be 14% less than expected. The utilizations for the actual design reflect this overhead with only 86% of RC execution time comprising communication or computation. The discrepancy in total execution is large because the overhead is significant relative to the short time (less than 0.5s). If the overhead had been factored into the prediction, the total estimation error would be under 1%. Additionally, resource utilization is summarized in Table XIII. No multiplers are required for this type of

searching but the heavy usage of logic elements limits further scalability of the algorithm on a single FPGA of this size.

5.4 Molecular Dynamics

Molecular Dynamics (MD) is the numerical simulation of the physical interactions of atoms and molecules over a given time interval. Based on Newton's second law of motion, the acceleration (and subsequent velocity and position) of the atoms and molecules are calculated at each time step based on the particles' masses and the relevant subatomic forces. For this case study, the molecular dynamics simulation is primarily focused on the interaction of certain inert liquids such as neon or argon. These atoms do not form covalent bonds and consequently the subatomic interaction is limited to the Lennard-Jones potential (i.e., the attraction of distant particles by van der Waals force and the repulsion of close particles based on the Pauli exclusion principle) [Allen and Tildesley 1987]. Large-scale molecular dynamics simulators such as AMBER [Pearlman et al. 1995] and NAMD [Nelson et al. 1996] use these same classical physics principles but can calculate not only Lennard-Jones potential but also the nonbonded electrostatic energies and the forces of covalent bonds, their angles, and torsions making them applicable to not only inert atoms but also complex molecules such as proteins.

The parallel algorithm used for this case study was adapted from code provided by Oak Ridge National Labs (ORNL) and targets the XtremeData XD1000 FPGA platform. The most challenging aspect of performance prediction for MD is accurately measuring the number of operations per molecular interaction and the computational throughput. This particular algorithm's execution time is dependent on the locality of the molecules, which is a function of the dataset values. Sufficiently distant molecules are assumed to have negligible interaction and therefore require less computational effort. Attempts are made to mitigate the data-driven behavior through pipelined computations. However, the complexity of the algorithm and potential for suboptimal behavior during mapping requires some algorithm parameters to be estimated.

Table XIV summarizes the input parameters for the RAT analysis of the MD design. The data size of 16,384 molecules (i.e., elements) was chosen because it is a small but still scientifically interesting problem. Each element requires 36 bytes, 4 bytes each for position, velocity, and acceleration in each of the X, Y, and Z spatial directions. The interconnect parameters model an XtremeData XD1000 platform containing a Altera Stratix-II EP2S180 user FPGA connected to an Opteron processor over the HyperTransport fabric. The theoretical interconnect throughput is 1.6GB/s but only a fraction of the channel can be used for transferring data to the on-board SRAM as needed for the algorithm. The number of operations per element, approximately 16,400 interactions per molecule times 10 operations each, is estimated due to the length of the pipeline and data-driven behavior. Unlike the previous case studies, the computational throughput cannot be reliably measured due to the complex and nondeterministic algorithm structure. As discussed in Section 3.1, the number of operations per cycle is treated as a "tuning" parameter to compute the

Table XIV. Input Parameters of MD

Dataset Parameters		
$N_{elements}$, input	(elements)	16384
$N_{elements}$, output	(elements)	16384
$N_{bytes/element}$	(bytes/element)	36
Communication Parameters (XtremeData)		
$throughput_{ideal}$	(MB/s)	1600
α_{write}	$0 < \alpha < 1$	0.28
α_{read}	$0 < \alpha < 1$	0.28
Computation Parameters		
$N_{ops/element}$	(ops/element)	164000
$throughput_{proc}$	(ops/cycle)	50
f_{clock}	(MHz)	75/100/150
Software Parameters		
t_{soft}	(sec)	5.76
N_{iter}	(iterations)	1

Table XV. Performance Parameters of MD (XtremeData)

	Predicted	Predicted	Predicted	Actual
f_{clk} (MHz)	75	100	150	100
t_{comm} (sec)	8.77E-4	8.77E-4	8.77E-4	1.39E-3
t_{comp} (sec)	7.17E-1	5.37E-1	3.58E-1	8.79E-1
$util_{comm_{SB}}$	0.1%	0.2%	0.2%	0.2%
$util_{comp_{SB}}$	99.9%	99.8%	99.8%	99.8%
$t_{RC_{SB}}$ (sec)	7.19E-1	5.38E-1	3.59E-1	8.80E-1
speedup	8.0	10.7	16.0	6.6

throughput necessary to achieve the desired speedup based on the estimate of $N_{ops/element}$. Though 50 is the quantitative value computed by the equations to achieve the desired overall speedup of approximately 10, this value serves qualitatively as an indicator that substantial data parallelism and functional pipelining must be achieved in order to realize the desired speedup. The same range of clock frequencies was used as in PDF estimation. The serial software baseline was executed on a 2.2 GHz Opteron processor, the host processor of the XD1000 system. The entire dataset is processed in a single iteration and the algorithm is constructed in Impulse C, a cross-platform HLL for FPGAs.

Table XV outlines the predicted and actual results of the MD. Note that these results are unique to this specific algorithm and that different structures, target languages, and platforms will have varying prediction accuracy. The difference in predicted and actual communication time is 37%. The error itself is associated with the overhead of multiple I/O transfers between the CPU and on-board SRAM memory modeled as a single block of communication. While more accurate estimations are the goal of RAT, any further precision improvements for this parameter are inconsequential given the low communication utilization. Computation dominated the overall RC execution time and the actual time is 39% higher than the predicted value due to the data-driven operations and suboptimal pipelining performance. The total number of operations was higher than expected, coupled with relatively modest parallelism for the problem size. Consequently, the speedup error was also 39%, significantly less than desired. However, this case study is useful because the *qualitative*

Table XVI. Resource Usage of MD (EP2S180)

FPGA Resource	Utilization
BRAMs	24%
9-bit DSPs	100%
ALUTs	73%

Table XVII. Summary of Results

	1-D PDF	2-D PDF	LIDAR	TSP	MD
Predicted Comm. (s)	2.47E-5	1.01E-2	6.60E-4	1.54E-5	8.77E-4
Actual Comm. (s)	2.50E-5	1.06E-2	5.65E-4	1.57E-4	1.39E-3
Comm. Error	1%	5%	17%	2%	37%
Predicted Comp. (s)	1.31E-4	4.19E-2	2.64E-4	4.31E-1	5.37E-1
Actual Comp. (s)	1.39E-4	4.46E-2	2.25E-4	4.30E-1	8.79E-1
Comp. Error	6%	6%	17%	0.2%	39%
Predicted Speedup	6.5	7.6	11.8	5.2	10.7
Actual Speedup	7.8	7.2	13.8	4.5	6.6
Speedup Error	16%	6%	16%	14%	39%

need for significant parallelism is correctly predicted even though the algorithm cannot be fully analyzed at design time. As Table XVI illustrates, a large percentage of the combinatorial logic and all dedicated multiply-accumulators (DSPs) were required for the algorithm.

5.5 Summary of Case Studies

Table XVII outlines the predicted values, actual results, and error percentages for the communication time, computation time, and speedup. The magnitudes of the predicted and actual values are listed to compare the absolute impact of the relative error percentages. For example, molecular dynamics has comparable communication and computation error percentages but the magnitudes are different with communication having virtually no impact on total speedup.

For these case studies, communication had the lowest average error among the modeled times. The larger errors for LIDAR and MD were caused by minor discrepancies in the final communication setup versus the RAT analysis of the algorithm. These error percentages are acceptable for RAT because they still yield valid quantitative insight about the algorithm behavior. The cost of more precise prediction must be balanced with the impact of the communication time on performance. Again, the largest communication error, found in MD, did not significantly affect the speedup, because the communication was less than 1% of the overall execution time.

Even with the more complicated task of architecture and platform parameterization, the average prediction error for computation was only slightly higher than communication. PDF and TSP had low computation errors complementing the low communication errors. LIDAR had double-digit error, which was due to perceivable system overheads in the short 0.2ms computation time. The one outlier was the MD application. The difficulty of mitigating the data-driven computations was compounded by unknowns in the final algorithm mapping by the HLL tool. As with the communication predictions for

MD, the error was significant but RAT still provided a useful insight about what order of magnitude speedup should be achievable.

The prediction errors in the overall speedup were higher on average than the individual computation or communication times. Particularly with 1-D PDF, LIDAR, and TSP, overheads not part of the RAT computation and communication models were noticeable portions of the short RC execution times, 70ms, 0.2ms and 430ms, respectively. The 2-D PDF estimation was long enough to mitigate system overheads. The overall performance of molecular dynamics matched the computation time due to 99% utilization. While minimizing the prediction error was an important issue, rapidly achieving a reasonable projection was the ultimate goal.

The data summarized in Table XVII highlights the focus of RAT on performance approximation. Refinements to the RAT model for short communication and computation times, platform-specific overheads, and multiple data transfers per iteration are potentially useful and can help improve the overall accuracy of the performance estimations. Extra analysis during parameterization can also improve RAT predictions. While future research will involve some revisions to the model, the current errors in the 5% to 15% range have proven sufficient for the intent of RAT.

6. CONCLUSIONS

The promise of reconfigurable computing for achieving speedup and power savings versus traditional computing paradigms is expanding interest in FPGAs. Among the challenges for improving development of parallel algorithms for FPGAs, the lack of methods for strategic design is a key obstacle to efficient usage of RC devices. Better formulation methodologies are needed to explore algorithms, architectures and mappings to reduce FPGA development time and cost. Consequently, RAT is created as a simple and effective methodology for investigating the performance potential of the mapping of a given parallel algorithm for a given FPGA platform architecture. The methodology employs an analytic model to analyze FPGA designs prior to actual development. RAT is meant to work with empirical knowledge of RC devices to create more efficient and effective means for design-space exploration.

In this article, RAT demonstrated reasonable accuracy with predicting communication time, computation time, and speedup with the five case studies. Detailed microbenchmarking prior to the RAT analyses allowed for an average error of 12% (with individual errors as low as 1%) for the communication times of algorithms. For the deterministic case studies (i.e., all except MD), computation error peaked at 17%. Each algorithm's inclusion of pipelining allowed computational throughputs to be accurately projected even though the high-level parallel algorithms were not yet mapped to hardware. The total RC execution time had an average error of 18% for the case studies, slightly higher than the individual communication and computation components. Large system overhead versus short execution time was the main cause. Overall, the methodology performed well for the diverse collection of algorithm complexities, hardware languages, FPGA platforms, and total execution times. RAT

was designed to handled these issues in single CPU and FPGA systems where communication and computation are governed by the number of data elements.

Future work in the area of performance prediction will focus on balancing the scale and fidelity of the RAT methodology. The continuing trend toward designing scalable, explicitly parallel algorithms necessitates an expanded analytic model encompassing multi-FPGA platforms. Incorporating lessons learned from the current RAT methodology will help improve performance estimation for the current single-FPGA and future multi-FPGA models. Additionally, scaling the fidelity of algorithm and platform models with respect to the algorithm structure and behavior can increase model precision as necessary. A key issue is knowing when to move from the RAT methodology to a higher-fidelity, higher-cost model.

ACKNOWLEDGMENTS

The authors gratefully acknowledge vendor equipment and/or tools provided by Altera, Cray, Impulse Accelerated Technologies, Nallatech, SRC Computers, Xilinx, and XtremeData that helped make this work possible. Additional thanks go to George Washington University for the generous use of their SRC-6 platform, and Chris Conger, Adam Jacobs, Kuei-Tsung Shih, and Ann Gordon-Ross at the University of Florida for their assistance with the RAT methodology and case studies.

REFERENCES

ALEXANDROV, A., IONESCU, M. F., SCHAUSER, K. E., AND SCHEIMAN, C. 1997. Loggp: Incorporating long messages into the logp model for parallel computation. *J. Paral. Distrib. Comput. 44*, 1, 71–79.

ALLEN, M. P. AND TILDESLEY, D. J. 1987. *Computer Simulation of Liquids*. Oxford University Press, Oxford, UK.

BANERJEE, P., BAGCHI, D., HALDAR, M., NAYAK, A., KIM, V., AND URIBE, R. 2003. Automated conversion of floating point matlab programs into fixed point FPGA based hardware design. In *Proceedings of the 11th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 263–264.

BONDALAPATI, K. AND PRASANNA, V. 1999. Dynamic precision management for loop computations on reconfigurable architectures. In *Proceedings of the 7th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 249–258.

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

BOSQUE, J. L. AND PEREZ, L. P. 2004. HLogGP: A new parallel computational model for heterogeneous clusters. In *Proceedings of the IEEE Symposium on Cluster Computing and the Grid*.

BROWN, S. D., ROSE, J., AND VRANESIC, Z. G. 1993. A stochastic model to predict the routability of field-programmable gate arrays. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 12*, 12, 1827–1838.

BUTTARI, A., DONGARRA, J., KURZAK, J., LANGOU, J., LANGOU, J., LUSZCZEK, P., AND TOMOV, S. 2007. *High Performance Computing and Grids in Action*. IOS Press.

CHANG, M. AND HAUCK, S. 2002. Precis: A design-time precision analysis tool. In *Proceedings of the 10th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 229–238.

CULLER, D., KARP, R., PATTERSON, D., SAHAY, A., SCHAUSER, K. E., SANTOS, E., SUBRAMONIAN, R., AND VON EICKEN, T. 1993. Logp: Towards a realistic model of parallel

computation. In *Proceedings of the 4th ACM Symposium on Principles and Practice of Parallel Programming*. 1–12.

DEGALAHAL, V. AND TUAN, T. 2005. Methodology for high level estimation of FPGA power consumption. In *Proceedings of the ACM Conference on Asia South Pacific Design Automation (ASP-DAC)*. 657–660.

ENZLER, R., PLESSL, C., AND PLATZNER, M. 2005. System-level performance evaluation of reconfigurable processors. *Microprocess. Microsyst. 29*, 2-3, 63–75.

FANG, W. M. AND ROSE, J. 2008. Modeling routing demand for early-stage FPGA architecture development. In *Proceedings of the ACM Symposium on Field Programmable Gate Arrays (FPGA)*. 139–148.

FORTUNE, S. AND WYLLIE, J. 1978. Parallelism in random access machines. In *Proceedings of the 10th ACM Symposium on Theory of Computing*. 114–118.

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.

GAFFAR, A., MENCER, O., LUK, W., CHEUNG, P., AND SHIRAZI, N. 2002. Floating-point bitwidth analysis via automatic differentiation. In *Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT)*. 158–165.

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.

GROBELNY, E., REARDON, C., JACOBS, A., AND GEORGE, A. 2007. Simulation framework for performance prediction in the engineering of rc systems and applications. In *Proceedings of the Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*.

HERBORDT, M. C., VANCOURT, T., GU, Y., SUKHWANI, B., CONTI, A., MODEL, J., AND DISABELLO, D. 2007. Achieving high performance with FPGA-based computing. *IEEE Computer 40*, 3, 50–57.

MAIDEE, P. AND BAZARGAN, K. 2006. Defect-tolerant FPGA architecture exploration. In *Proceedings of the 13th IEEE Conference on Field Programmable Logic and Applications (FPL)*. 1–6.

MANOHARARAJAH, V., CHIU, G. R., SINGH, D. P., AND BROWN, S. D. 2006. Difficulty of predicting interconnect delay in a timing driven FPGA CAD flow. In *Proceedings of the ACM Workshop on System-Level Interconnect Prediction (SLIP)*. 3–8.

NAGARAJAN, K., HOLLAND, B., SLATTON, C., AND GEORGE, A. D. 2008. Scalable and por architecture for probability density function estimation on FPGAs. In *Proceedings of the 16th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

NELSON, M., HUMPHREY, W., GURSOY, A., DALKE, A., KAL, L., SKEEL, R. D., AND SCHULTEN, K. 1996. Namd—a parallel, object-oriented molecular dynamics program. *Int. J. Supercomp. Appl. High Perform. Comput. 10*, 4, 251–268.

PEARLMAN, D. A., CASE, D. A., CALDWELL, J. W., ROSS, W. S., THOMAS E. CHEATHAM, I., 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. Comm. 91*, 1-3, 1–41.

PERRI, S., CORSONELLO, P., IACHINO, M. A., LANUZZA, M., AND COCORULLO, G. 2004. Variable precision arithmetic circuits for FPGA-based multimedia processors. *IEEE Trans. (VLSI) 12*, 9, 995–999.

QUINN, H., LEESER, M., AND KING, L. S. 2007. Dynamo: A runtime partitioning system for FPGA-based hw/sw image processing systems. *J. Real-Time Image Process. 2*, 4, 179–190.

SHIH, K., BALACHANDRAN, A., NAGARAJAN, K., HOLLAND, B., SLATTON, C., AND GEORGE, A. 2008. Fast realtime lidar processing on FPGAs. In *Proceedings of the Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. Las Vegas, NV.

SINGH, A. AND MAREK-SADOWSKA, M. 2002. Fpga interconnect planning. In *Proceedings of the ACM Workshop on System-Level Interconnect Prediction (SLIP)*. 23–30.

SINGH, D. P., MANOHARARAJAH, V., AND BROWN, S. D. 2005. Two-stage physical synthsis for FPGAs. In *Proceedings of the 13th IEEE Conference on Custom Integrated Circuits*. 171–178.

SMITH, M. AND PETERSON, G. 2005. Parallel application performance on shared high performance reconfigurable computing resources. *Perform. Eval. 60*, 107–125.

STEFFEN, C. 2007. Parameterization of algorithms and FPGA accelerators to predict performance. *Reconfigurable System Summer Institute (RSSI)*. Urbana, IL.

TSCHOKE, S., LUBLING, R., AND MONIEN, B. 1995. Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network. In *Proceedings of the Symposium on Parallel Processing*.

VALIANT, L. G. 1990. A bridging model for parallel computation. *Comm. ACM 33*, 8, 103–111.

WANG, X., BRAGANZA, S., AND LEESER, M. 2006. Advanced components in the variable precision floating-point library. In *Proceedings of the Conference on Field-Programmable Custom Computing Machines (FCCM)*.

XU, M. AND KURDAHI, F. 1999. Accurate prediction of quality metrics for logic level design targeted towards lookup-table-based FPGA's. *IEEE Trans. VLSI Syst. 7*, 4, 411–418.

Received May 2008; revised September 2008; accepted October 2008