RAT: A Methodology for Predicting Performance in Application Design Migration to FPGAs

Brian Holland, Karthik Nagarajan, Chris Conger, Adam Jacobs, Alan D. George
NSF Center for High-Performance Reconfigurable Computing (CHREC)
ECE Department, University of Florida
{holland,nagarajan,conger,jacobs,george}@chrec.org

ABSTRACT

Before any application is migrated to a reconfigurable computer (RC), it is important to consider its amenability to the hardware paradigm. In order to maximize the probability of success for an application's migration to an FPGA, one must quickly and with a reasonable degree of accuracy analyze not only the performance of the system but also the required precision and necessary resources to support a particular design. This extra preparation is meant to reduce the risk of failure to achieve the application's design requirements (e.g. speed or area) by quantitatively predicting the expected performance and system utilization. This paper presents the RC Amenability Test (RAT), a methodology for rapidly analyzing an application's design compatibility to a specific FPGA platform.

1. INTRODUCTION

FPGAs continue to grow as a viable option for increasing the performance of many applications over traditional CPUs without the need for ASICs. Because no standardized rules exist for FPGA amenability, it is important for a designer to consider the likely performance of an application in hardware before undergoing a lengthy migration process. Ultimately, the designer must know what order of magnitude speedup (or potentially slowdown) will be encountered.

Some researchers have suggested [4] that a 50x to 100x speedup is required to gain the attention and approval of "middle management." Other scenarios might place the break-even point (time of development versus time saved at execution) at a more conservative factor of ten or less. The high-performance embedded community might simply want FPGA performance to parallel a traditional processor since savings could come in the form of reduced power usage. Ultimately, the success or failure of an application's RC migration will be judged against some metric of performance. It is critical to consider whether the chosen application architecture and FPGA platform will meet the speed, area, and power requirements of the project. The RC Amenability Test (RAT) is a combination of algorithm and software legacy code analyses along with 'pencil and paper' computations that seeks to determine the likelihood of success for a specific algorithm's migration to a particular RC platform before any (or at least significant) hardware coding is begun.

The need for the RAT methodology stemmed from common difficulties encountered during several FPGA application migration 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 platforms 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, more simple, 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 migration. 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.

Many research projects as discussed in [11] emphasize the usage of FPGAs to achieve speedup over traditional CPUs. 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, RAT's throughput analysis is the most detailed performance test and the focus of the application case studies in this paper.

The remainder of this paper is structured as follows. Section 2 discusses background related to FPGA performance

.

prediction and resource utilization. The fundamental analyses 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

Researchers have studied the area of hardware amenability but their approaches vary. A Performance Prediction Model (PPM) is suggested in [12] for determining the optimal mapping of an algorithm to an FPGA. Their methodology consists of four steps: choice and modification of the implementation, classification, feature extraction, and performance matrix computation (for frequency, latency, throughput, and area requirements). The concept of a detailed classification of the internal operations of an application design is very practical. However, the performance estimation method incorporates quantitative area, IO pins, latency, and throughput into a large system of platform-dependent equations which is impractical for RAT. The goal of the research presented in [14] is to determine the optimal function design with respect to area, latency and throughput. Ultimately, the project seeks to create a tool or library for identifying the best version among many alternatives for a particular scenario. The work provides valuable insight, especially into the domain of application precision, but its focus on automated design of single kernels is overly specific for a high-level RAT methodology for applications.

A performance prediction technique presented in [15] seeks to parameterize not only the computational algorithm but also the FPGA system. Applications are decomposed and analyzed to determine their total size and computational density. Computational platforms are characterized by their memory size, bandwidth, and latency. By comparing the algorithm's computational requirement with the memory bottleneck of the FPGA platform, a worst-case computational throughput (in operations per second) can be quantified. However, the author asserts that "the performance analysis in this paper is not real performance prediction; rather it targets the general concern of whether or not an algorithm will fit within the memory subsystem that is designed to feed it." The RAT methodology differs because it seeks not only to quantify the number of operations but also the expected number of operations executed per clock cycle yielding performance predictions strictly in units of time.

A research project into molecular dynamics at the University of Illinois [13] proposes a framework for application design in the hardware paradigm. The project asserts that the most frequently accessed block of code is not the only indicator for RC amenability. The types of computations and the volume of communication will increase or decrease the recommended quantity of FPGA functions. Although the general framework stresses resource utilization tests for case studies, the researchers "postpone finding out about space requirements for the design until [they] actually map it to the FPGA." A priori measurements of resource requirements may be inexact, but they are still necessary to avoid creating initial designs that are physically unrealizable.

Conceptually, the RAT methodology is meant to resemble the approach behind the Parallel Random Access Machine (PRAM) [8] model for traditional supercomputing. Both

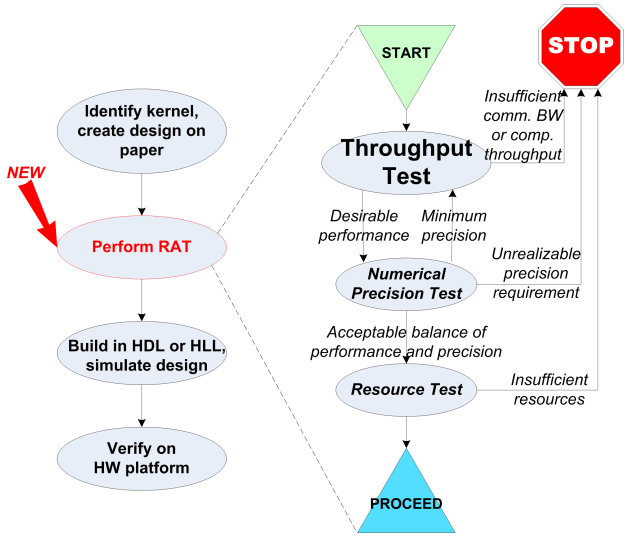

Figure 1: Overview of RAT Methodology

RAT and PRAM attempt 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. PRAM focuses on modes of concurrent memory accesses whereas RAT examines the communication and computation interaction on an FPGA. These critical attributes of RAT also resemble the LogP model [7] (a successor to PRAM) which seeks to abstract "the computing bandwidth, the communication bandwidth, the communication delay, and the efficiency of coupling communication and computation." RAT does not claim to be the ultimate solution to RC performance prediction but instead encourages the FPGA community to consider more structured and standardized ways for algorithm analysis using established concepts inherited from prior successes in traditional parallel computing modeling.

3. RC AMENABILITY TEST

Figure 1 illustrates the basic methodology behind the RC amenability test. This simple set of tests serves 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 design, not a generic algorithm. The results of the RAT tests must be compared against the designer's requirements to evaluate the success of the application design. Though the throughput analysis is considered the most important step, the three tests are not necessarily used as a single, sequential procedure. Often, RAT is applied iteratively during the design process until a suitable version of the algorithm is formulated or all reasonable permutations are exhausted without a satisfactory solution.

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 operations occur on that data, respectively. Because RAT seeks to analyze applications at the earliest stage of hardware migration, these terms are reduced to the most generalized parameters.

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 problem size only refers to a single block of data to be buffered by the FPGA system. An application's data communication may be divided into multiple discrete transfers, which is accounted for in a subsequent equation. The hypothetical bandwidth of the FPGA/processor 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 system 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 will only be a fraction of the documented transfer rate. Microbenchmarks composed of simple data transfers can be used to establish the true communication bandwidth.

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

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

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

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 the ubiquitous unit "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 equal the number of operations per element. Less optimized designs will only have a fraction of the capacity requiring multiple cycles to complete an element. 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.)

$$t_{comp} = \frac{N_{elements} \cdot N_{ops/element}}{f_{clock} \cdot throughput_{proc}} \tag{4}$$

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. 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.

Similar to an element, one must also examine what is an "operation." Consider an example application 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 the $throughput_{proc}$ is formulated with same assumption about the scope of an operation. For this example, 2/17 and 1 operation per second, respectively, yield the correct computation time of 17 cycles.

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,

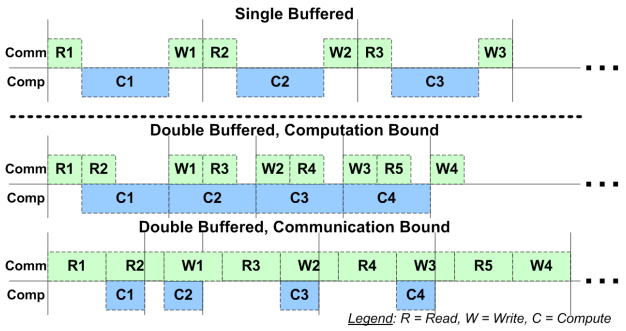

Figure 2: Example Overlap Scenarios

this startup cost is considered negligible for a sufficiently large number of iterations.

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 single- and double-buffered scenarios. For single buffered, the execution time is simply the summation of the communication time, t_{comm}, and computation time, t_{comp}. With the double-buffered case, either the communication or computation time completely overlaps the other term. The smaller latency essentially becomes hidden during steady-state.

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 single- and double-buffered cases. Note that the double-buffered case is only applicable to applications with a sufficient number of iterations so as to achieve a steady-state behavior throughout most of the execution time. The computation utilization can provide additional insight about the application speedup. If utilization is high, 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 which 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.

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

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

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

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

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 time-efficient 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 the RAT methodology. A plethora of research exists on topics including automated conversion of floating-point software programs to fixed-point hardware designs [2], design-time precision analysis tools for RC [5], and custom or dynamic bit-widths for maximizing performance and area on FPGAs [3, 9]. Application designs are meant to capitalize on these numerical precision techniques and then use the RAT methodology to evaluate the resulting algorithm performance. 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, dedicated hardware functional units (e.g. 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 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 ubiquitous 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. 32-bit fixed-point multiplications on Xilinx V4 FPGAs require two dedicated 18-bit multipliers). 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 was ultimately limited by the availability of multiplier resources.

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 not only expounds upon key concepts of the throughput test but also offers a detailed analysis of a real application, 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 is a generalized nonparametric approach to estimating probability density functions (PDFs) in a d-dimensional space. 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 discrete probability levels (comparable to the number of "bins" in a histogram), n is the number of discrete points at which the PDF is estimated (i.e. number of elements), 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.

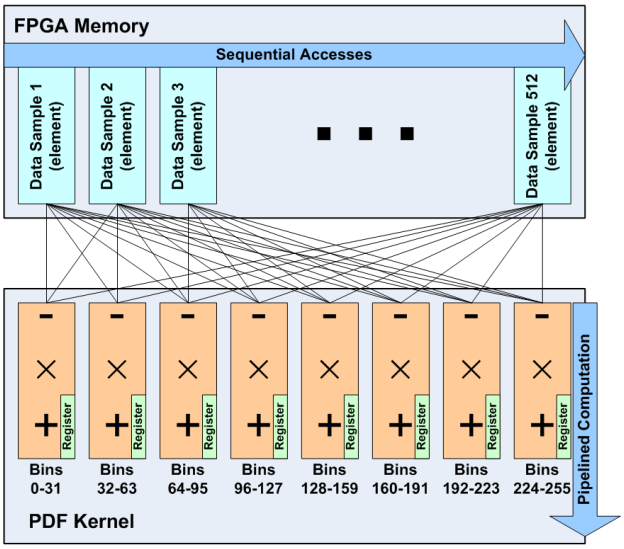

Figure 3: Architecture of 1-D PDF Algorithm

Table 1: Input parameters for RAT analysis

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

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, one general architecture of the PDF estimation algorithm is highlighted in Figure 3. A total of 204,800 data samples are processed in batches of 512 elements at a time against 256 bins. Eight separate pipelines are created to process a data sample 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 1 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

Table 2: 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		
$throughput_{ideal}$	(MB/s)	1000
α_{write}	$0 < \alpha < 1$	0.37
α_{read}	$0 < \alpha < 1$	0.16
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

listed under a separate category when it is actually 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. While there are cases where applications exhibit unusual computational trends or require significant amounts of additional data (e.g. constants, seed values, or lookup tables), the current RAT user base has found these instances to be uncommon. Alterations can be made to account for these cases but such examples are not included in this paper.

Table 2 summarizes the input parameters for the RAT analysis for our 1-D PDF estimation algorithm using Gaussian kernels. 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 value is retained after each iteration per bin. Each iteration's result is retained on the FPGA and all values are transferred back to the host in a single block after the algorithm has completed. However, the output transfer time, regardless of how it is modeled, remains negligible with respect to the overall execution time.

The number of bytes per element is rounded to four (i.e. 32 bits). Even though the PDF estimation algorithm only uses 18-bit fixed point, the communication channel uses 32-bit communication. During the algorithmic formulation, 18-bit and 32-bit fixed point along with 32-bit floating point were considered for use in the PDF algorithm. However, the maximum error percentage was 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 18x18 multiple-accumulate (MAC) unit would be needed per multiplication. Though slightly smaller bitwidths would have also possessed reasonable error constraints, no performance gains or appreciable resource savings would have been achieved.

Next, the communication parameters are provided by the user since they are merely a function of the target RC platform, which in this case is a Nallatech H101-PCIXM card containing a Virtex-4 LX100 user PFGA. The card is connected to the host CPU via a 133MHz PCI-X bus which has a theoretical maximum bandwidth of 1000MB/s. The α parameters were computed using a microbenchmark consisting of a read and write for a data size comparable to one 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 calculate the α parameters by dividing by the theoretical maximum. The α parameters for the FPGA platforms are low due to communication protocols used by Nallatech atop PCI-X. In general, the microbenchmark is performed on an FPGA 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.

The computation parameters are perhaps the most challenging portion of the performance analysis, but are still simplistic given the deterministic behavior of the PDF estimation algorithm. 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 effectively perform 3 operations per cycle for a total of 24. However, this value is conservatively rounded down to 20 to account for pipeline latency and other overheads that are not otherwise considered. The 1-D PDF algorithm is constructed in VHDL to allow explicit, cycle-accurate construction of the intended design.

While previous parameters could be reasonably inferred from the deterministic structure of the 1-D PDF 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. But one could equally generate this code from a mathematically defined algorithm strictly as a baseline for performance analysis. 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.

4.3 Predicted and Actual Results

The RAT performance numbers are compared with the experimentally measured results in Table 3. 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 computed as follows:

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

The communication time is computed using the corresponding equation. Because the application is single buffered, the total RC execution time is a simple summation.

$$t_{RC_{SB}} = 400 \text{ iterations} \cdot (5.56\text{E-6 secs} + 1.31\text{E-4 secs}) = 5.46\text{E-2 secs}$$

The speedup is simply the division of the software execution time by the RC execution time. The utilization equations are computed using the corresponding single-buffered equations.

The communication and computation times for the actual FPGA code were measured from the hardware. The total execution time for the hardware can be computed from Equation (5) or, as with this case study, measured from the FPGA to ensure maximum accuracy. The speedup and utilization values were computed from this information using the same equations as the predicted values. From the table, the predicted speedup when f_{clk} =150 is reasonably close to the actual value. The discrepancy in speed in this case is due to the inaccuracies in the t_{comm} estimation. Although communication predictions are based on experimentally gathered data from microbenchmarks, the true behavior was not encapsulated for this algorithm. Variability in the communication time with the small data sizes (i.e 2KB for one iteration of the 1-D PDF algorithm) and additional delays introduced by 800 (400 read, 400 write) repetitive transfers are considered to be the source of the error.

A relatively accurate t_{comp} prediction is not surprising given the deterministic nature of the algorithm. However, the two significant figures of accuracy between the predicted and actual computation times with f_{clk} =150MHz was unexpected given that computational throughput was conservatively estimated. Much of the 1-D PDF algorithm is pipelined but enough latency and pipeline stalls existed to genuinely warrant a 17% reduction in the throughput estimate (i.e. 20 ops/cycle instead of 24). Unfortunately, the inaccuracies in the communication time predictions reduced the accuracy of the overall speedup. Had the communication been double buffered, the inaccuracies in the communication time could have been masked behind the more stable computation time for a more accurate (and higher) speedup. The relatively low resource usage in Table 4 also illustrates a potential for further speedup by including additional parallel kernels.

Table 3: Performance parameters of 1-D PDF

	Predicted	Predicted	Predicted	Actual
f_{clk} (MHz)	75	100	150	150
t_{comm} (sec)	5.56E-6	5.56E-6	5.56E-6	2.50E-5
t_{comp} (sec)	2.62E-4	1.97E-4	1.31E-4	1.39E-4
$util_{comm_{SB}}$	2%	3%	4%	15%
$util_{comp_{SB}}$	98%	97%	96%	85%
$t_{RC_{SB}}$ (sec)	1.07E-1	8.09E-2	5.46E-2	7.45E-2
speedup	5.4	7.2	10.6	7.8

Table 4: Resource usage of 1-D PDF (LX100)

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

Table 5: Input parameters of 2-D PDF (LX100)

Dataset Parameters		
$N_{elements}$, input	(elements)	1024
$N_{elements}$, output	(elements)	65536
$N_{bytes/element}$	(bytes/element)	4
Communication Parameters		
$throughput_{ideal}$	(MB/s)	1000
α_{write}	$0 < \alpha < 1$	0.37
α_{read}	$0 < \alpha < 1$	0.16
Computation Parameters		
$N_{ops/element}$	(ops/element)	393216
$throughput_{proc}$	(ops/cycle)	48
f_{clock}	(MHz)	75/100/150
Software Parameters		
t_{soft}	(sec)	158.8
N_{iter}	(iterations)	400

5. ADDITIONAL CASE STUDIES

These case studies are presented as further analysis and validation of the RAT methodology, 2-D PDF estimation and molecular dynamics. Two-dimensional PDF estimation continues to illustrate the accuracy of RAT for algorithms with a deterministic structure. However, the molecular dynamics application serves as an interesting counterpoint given the relative difficulty of encapsulating its computational behavior. As with the one-dimensional PDF estimation algorithm, the design emphasis is placed on throughput analyses because the overall goal was to minimize execution time for these designs.

5.1 2-D PDF Estimation

As previously discussed, the Parzen window technique is applicable in an arbitrary number of dimensions. 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 histogram generation 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 probability level values and n_1, n_2 are the data sample values for each dimension, and c is a probability scaling factor. But despite the added complexity, the increased quantity of parallelizable operations intuitively makes this algorithm more amendable to the RC paradigm, assuming sufficient quantities of hardware resources are available.

Table 5 summarizes the input parameters for the RAT analysis for our 2-D PDF estimation algorithm. Again,

Table 6: Performance parameters of 2-D PDF

	Predicted	Predicted	Predicted	Actual
f_{clk} (MHz)	75	100	150	150
t_{comm} (sec)	1.65E-3	1.65E-3	1.65E-3	1.06E-2
t_{comp} (sec)	1.12E-1	8.39E-2	5.59E-2	4.46E-2
$util_{comm_{SB}}$	1%	2%	3%	19%
$util_{comp_{SB}}$	99%	98%	97%	81%
$t_{RC_{SB}}$ (sec)	4.54E+1	3.42E+1	2.30E+1	2.21E+1
speedup	3.5	4.6	6.9	7.2

Table 7: Resource usage of 2-D PDF (LX100)

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

Table 8: Input parameters of MD

Dataset Parameters		
$N_{elements}$, input	(elements)	16384
$N_{elements}$, output	(elements)	16384
$N_{bytes/element}$	(bytes/element)	36
Communication Parameters		
$throughput_{ideal}$	(MB/s)	500
α_{write}	$0 < \alpha < 1$	0.9
α_{read}	$0 < \alpha < 1$	0.9
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

the computation is performed in a two-dimensional space so twice the number of data samples (in blocks of 512 words for each dimension) are sent to the FPGA. In contrast to the 1-D case, the PDF values computed over each iteration are sent back to the host processor. The same numerical precision of four bytes per element is used for the data set. The interconnect parameters model the same Nallatech FPGA card as in the 1-D case study. The higher order of computational complexity is reflected in the larger number of computations per element, approximately three orders of magnitude greater. However, the number of parallel operations is only increased by a factor of two. VHDL is also used for the 2-D PDF algorithm to create the cycle-accurate pipeline. Again, the same range of clock frequencies are used for comparison. The software baseline for computing speedup values was written in C and executed on the same 3.2GHz Xeon processor. The same 400 iterations are required to complete the algorithm.

The RAT performance predictions are compared with the experimentally measured results in Table 6. The predicted speedup at 150MHz is closer to the experimental 150MHz value than the one-dimensional case. Though the percent error is greater for both communication and computation, the important difference is that the predicted computation time was significantly overestimated which balanced out the underestimated communication time. However, the problem is not with the computation parameters. These values are often conservatively estimated to account for unforeseen problems and to avoid promising unattainable results. Unfortunately, the 'unforeseen problem' for this algorithm turned out to be communication six times larger than predicted, comprising 19% of the total execution instead of the originally estimated 3%. The authors acknowledge this case study as a victory in contingency planning but highlight the precariousness of applications where relatively small shifts in the communication time can greatly effect the utilization of the FPGA.

Ultimately, the reduced number of parallel operations ($throughput_{proc}$) with respect to the quantity of operations ($N_{ops/element}$) resulted in an effective speedup less than the one-dimensional algorithm. The qualitative RC 'amenability' of the two-dimensional PDF application could only be capitalized to the extent permissible by the designer's algorithm structure and physical device limitations, specifically the communication bandwidth. The performance decrease with respect to the 1-D PDF is highlighted by the computation utilizations. Though more parallelizable operations occur in the 2-D PDF algorithm, the increased communication demands of the higher order reduced the speedup of this design for this platform. Comparing Table 7 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.

5.2 Molecular Dynamics

Molecular Dynamics (MD) is the numerical simulation of the physical interactions of atoms and molecules over a given time interval. Along with standard Newtonian physics, properties such as Van Der Waals forces and electrostatic charge (among others) are calculated for each molecule at each time step with respect to the movement and the molecular structure of every particle in the system. The tremendous number of options for physically modeled phenomena and computational methods makes MD a perfect example of how various designs for an application can have radically different execution times. Three different versions of the molecular dynamics algorithm [1], [6], and [10] report speedup values of 0.29x, 2x, and 46x respectively. These designs make use of various algorithm optimizations, precision choices, and FPGA platform selections. Consequently, RAT can offer insight about a particular design, but it cannot guarantee that a better solution does not exist.

The version of the molecular dynamics application used for this case study was adapted from code provided by Oak Ridge National Lab (ORNL). Table 8 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.

Table 9: Performance parameters of MD

	Predicted	Predicted	Predicted	Actual
f_{clk} (MHz)	75	100	150	100
t_{comm} (sec)	2.62E-3	2.62E-3	2.62E-3	1.39E-3
t_{comp} (sec)	7.17E-1	5.37E-1	3.58E-1	8.79E-1
$util_{comm_{SB}}$	0.4%	0.5%	0.7%	0.2%
$util_{comp_{SB}}$	99.6%	99.5%	99.3%	99.8%
$t_{RC_{SB}}$ (sec)	7.19E-1	5.40E-1	3.61E-1	8.80E-1
speedup	8.0	10.7	16.0	6.6

Table 10: Resource usage of MD (EP2S180)

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

Ultimately, the most challenging aspect of performance prediction for the molecular dynamics application is accurately measuring the number of operations per element (i.e. molecule) and operations per clock cycle. This particular algorithm's execution time is dependent on the locality of the molecules, which is a function of the dataset values. Distant molecules are assumed to have negligible interaction and therefore require less computational effort. Consequently, the number of operations per element can only be estimated for this circumstance and as previously discussed, the number of operations per element is treated as a "tuning" parameter. Though 50 is the quantitative value computed by the equations to achieve the desired overall speedup of approximately 10x, this value serves qualitatively to the user as an indicator that substantial data parallelism and functional pipelining must be achieved in order to realize the desired speedup. Several major architectural design revisions were explored, based upon the RAT findings, in order to facilitate the necessary parallelism. Additionally, the same clock frequencies were used as in the previous case studies, even though the FPGA platform has changed, because the parameters are empirically reasonable. The serial software baseline was performed on a 2.2 GHz Opteron processor, the host processor of the XD1000 system. The entire dataset is processed in a single iteration.

The application was constructed in the HLL language Impulse C in contrast to the PDF algorithms. Normally the usage of an HLL would have made it more difficult (if not impossible) to ensure that the particular cycle-by-cycle structure desired by the researcher was utilized in the final design. Even slight variabilities in algorithm structure introduced by HLLs could make accurate RAT computational analyses challenging. For the molecular dynamics application, which possesses data dependent operations and irregular computation structure independent of its linguistical representation, performance prediction is already handicapped in terms of accuracy. Consequently, the usage of HLLs is beneficial for this MD application for significantly reducing development time because the variability of the high-level paradigm will not significantly impact the already complex design structure.

Table 9 outlines the predicted and actual results of the molecular dynamics algorithm. The actual communication times is the same order of magnitude as the predicted value. While more accurate estimations are always 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 was also the same order of magnitude as the predicted value. Again, what eventually allowed the molecular dynamics algorithm to succeed in the RC paradigm was a qualitative interpretation of the prediction parameters which highlighted the need for scalable parallelism (specifically the ability to work on several molecules simultaneously). After major high-level design revisions, the designer successfully created a molecular dynamics algorithm that met the predicted criteria with moderate success. However, as Table 10 illustrates, a large percentage of the combinatorial logic and dedicated multiply-accumulators (DSPs) were required.

6. CONCLUSIONS

RAT is a simple and effective method for investigating the performance potential of the mapping of a given application design for a given FPGA platform architecture. The methodology employs a simple approach to planning an FPGA design. RAT is meant to work with empirical knowledge of RC devices to create more efficient and effective means for formulating an application design.

RAT succeeds in its goal of simple and reasonably accurate prediction for application designs. For deterministic algorithms such as the PDF estimation, RAT can accurately estimate the computational loads expected of the FPGA. Coupled with an accurate measurement of the FPGA platform interconnect throughput, a suitable prediction as to the 'RC amenability' was formulated before any hardware code was composed. In contrast, the molecular dynamics experiment grapples with a situation where data dependencies in the algorithm create uncertainty about the overall runtime. However, RAT was able to qualitatively highlight the large volume of parallelism that would be required to achieve even a 10x speedup. Consequently, extra algorithm restructuring was incorporated to increase performance and obtain a speedup somewhat near the 10x goal.

Though RAT has thus far proven effective and useful to the endeavors of its designers, there are several additional areas under consideration. The current methodology was designed to support applications involving several algorithms, each with their own separate RAT analysis. Further experimentation and usage with such design projects is necessary, especially with systems containing multiple FPGAs being increasingly deployed. Several strategies for improving the throughput test's analysis are also under consideration.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] N. Azizi, I. Kuon, A. Egier, A. Darabiha, and P. Chow. Reconfigurable molecular dynamics simulator. In *Proc IEEE 12th Symp. Field-Programmable Custom Computing Machines (FCCM)*, pages 197–206, Napa, CA, Apr 20-23 2004.

[2] P. Banerjee, D. Bagchi, M. Haldar, A. Nayak, V. Kim, and R. Uribe. Automated conversion of floating point matlab programs into fixed point fpga based hardware design. In *Proc IEEE 11th Symp. Field-Programmable Custom Computing Machines (FCCM)*, pages 263–264, Napa, CA, Apr 8-11 2003.

[3] K. Bondalapati and V. Prasanna. Dynamic precision management for loop computations on reconfigurable architectures. In *Proc IEEE 7th Symp. Field-Programmable Custom Computing Machines (FCCM)*, pages 249–258, Napa, CA, Apr 21-23 1999.

[4] D. Buell. Programming reconfigurable computers: Language lessons learned. In *Reconfigurable System Summer Institute (RSSI)*, Urbana, IL, Jul 12-13 2006.

[5] M. Chang and S. Hauck. Precis: A design-time precision analysis tool. In *Proc IEEE 10th Symp. Field-Programmable Custom Computing Machines (FCCM)*, pages 229–238, Napa, CA, Apr 22-24 2002.

[6] L. Cordova and D. Buell. An approach to scalable molecular dynamics simulation using supercomputing adaptive processing elements. In *Proc. IEEE Int. Conf. Field Programmable Logic and Applications (FPL)*, pages 711–712, Aug 24-26 2005.

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. Logp: Towards a realistic model of parallel computation. In *Proc ACM 4th Symp. Principles and Practice of Parallel Programming*, pages 1–12, San Diego, CA, May 19-22 1993.

[8] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proc ACM 10th Symp. Theory of Computing*, pages 114–118, San Diego, CA, May 01-03 1978.

[9] A. Gaffar, O. Mencer, W. Luk, P. Cheung, and N. Shirazi. Floating-point bitwidth analysis via automatic differentiation. In *Proc IEEE Int. Conf. Field-Programmable Technology (FPT)*, pages 158–165, Hong Kong, China, Dec 16-18 2002.

[10] Y. Gu, T. VanCourt, and M. Herbordt. Accelerating molecular dynamics simulations with configurable circuits. In *Proc. IEE Computers and Digital Techniques*, volume 153, pages 189–195, May 2 2006.

[11] Z. Guo, W. Najjar, F. Vahid, and K. Vissers. A quantitative analysis of the speedup factors of fpgas over processors. In *Proc ACM 16th Symp. Field-Programmable Gate Arrays (FPGA)*, pages 162–170, Monterey, CA, Feb 22-24 2004.

[12] T. Jeger, R. Enzler, D. Cottet, and G. Troster. The performance prediction model - a methodology for estimating the performance of an fpga implementation of an algorithm. technical report, Electronics Lab, Swiss Federal Inst. of Technology (ETH) Zurich, 2000.

[13] V. Kindratenko and D. Pointer. A case study in porting a production scientific supercomputing application to a reconfigurable computer. In *Proc IEEE 14th Symp. Field-Programmable Custom Computing Machines (FCCM)*, pages 13–22, Napa, CA, Apr 24-26 2006.

[14] D.-U. Lee, A. Gaffar, O. Mencer, and W. Luk. Optimizing hardware function evaluation. *IEEE Trans. Computers*, 54(12):1520–1531, Dec. 2005.

[15] C. Steffen. Parameterization of algorithms and fpga accelerators to predict performance. In *Reconfigurable System Summer Institute (RSSI)*, Urbana, IL, Jul 17-20 2007.