An Analytical Model for Multilevel Performance Prediction of Multi-FPGA Systems

BRIAN HOLLAND, ALAN D. GEORGE, and HERMAN LAM, University of Florida
MELISSA C. SMITH, Clemson University

Power limitations in semiconductors have made explicitly parallel device architectures such as Field-Programmable Gate Arrays (FPGAs) increasingly attractive for use in scalable systems. However, mitigating the significant cost of FPGA development requires efficient design-space exploration to plan and evaluate a range of potential algorithm and platform choices *prior* to implementation. The authors propose the RC Amenability Test for Scalable Systems (RATSS), an analytical model which enables straightforward, fast, and reasonably accurate performance prediction prior to implementation by extending current modeling concepts to multi-FPGA designs. RATSS provides a comprehensive strategic model to evaluate applications based on the computation and communication requirements of the algorithm and capabilities of the FPGA platform. The RATSS model targets data-parallel applications on current scalable FPGA systems. Three case studies with RATSS demonstrate nearly 90% prediction accuracy as compared to corresponding implementations.

Categories and Subject Descriptors: B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; I.6.m [**Computing Methodologies**]: 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 space exploration, formulation methodology

ACM Reference Format:
Holland, B., George, A. D., Lam, H., and Smith, M. C. 2011. An analytical model for multilevel performance prediction of multi-FPGA systems. ACM Trans. Reconfig. Technol. Syst. 4, 3, Article 27 (August 2011), 28 pages.
DOI = 10.1145/2000832.2000839 http://doi.acm.org/10.1145/2000832.2000839

1. INTRODUCTION

Power limitations have changed computing trends towards explicitly parallel architectures such as Field-Programmable Gate Arrays (FPGAs). This reformation is accompanied by increased emphasis on multidevice systems for achieving additional performance benefits. However, exploiting explicit parallelism for scalable FPGA systems requires expensive development cycles, which limits widespread adoption. Current design approaches focus on faster coding paths to the hardware implementation (e.g.,

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, Nallatech, SRC Computers, and Xilinx that helped make this work possible, and also George Washington University for use of their SRC-6 system.
Authors' addresses: B. Holland (corresponding author), A. D. George, H. Lam, NSF Center for High-Performance Reconfigurable Computing (CHREC), University of Florida, Gainesville, FL 32611; email: Holland@hcs.ufl.edu; M. C. Smith, Clemson University, Clemson, SC.

DOI 10.1145/2000832.2000839 http://doi.acm.org/10.1145/2000832.2000839

high-level synthesis), which address only one symptom of the greater productivity challenge.

In contrast to low-level, iterative development practices, strategic Design-Space Exploration (DSE) is needed to improve productivity with scalable FPGA systems. Applications should be planned and performance issues analyzed prior to implementation, narrowing the range of possible algorithm and platform mappings based on application performance requirements. The authors' prior work with the RC Amenability Test (RAT) has introduced strategic DSE for single-FPGA systems. Traditional computing models such as BSP [Valiant 1990] and LogP [Culler et al. 1993] can also be leveraged to help facilitate strategic DSE for multi-FPGA, scalable systems. The authors propose the RC Amenability Test for Scalable Systems (RATSS), which extends RAT to multi-FPGA systems by incorporating key concepts from traditional analytical modeling. Specifically, this article introduces comprehensive performance prediction for multi-FPGA systems by agglomerating and analyzing the results of the underlying models based on the system hierarchy. RAT provides a significant basis for the computation and communication models with additional scoping for the modeling of scalable systems in RATSS. Multi-FPGA versions of two of the original case studies for RAT, 2D PDF estimation and molecular dynamics, assist in validation of RATSS. RATSS prediction is fast, usually requiring only minutes to perform, and reasonably accurate thereby supporting the overall strategic DSE goal.

RAT, LogP, and many other analytical models seek to express the behavior of the computation and communication within a target component, device, subsystem, etc. "Nodes" and "networks" are used for simple, abstract representation of computation and communication, respectively. RATSS systematically characterizes an algorithm mapped to an FPGA platform using comparable node- and network-level analysis, which combine to form a complete model for the system. This model emphasizes *multilevel* analysis: performance prediction for parallel computation connected with multiple levels of communication infrastructure. RATSS is tractable for scalable systems because the scope and constraints of the underlying models are adapted for strategic, multi-FPGA performance characterization. This article discusses RATSS modeling for data-parallel algorithms structured as SIMD-style pipelines targeting modern high-performance FPGA systems [El-Ghazawi et al. 2008]. While the authors believe that RATSS has usage beyond this scope, these algorithm and platform assumptions allow for concise and consistent identification of requirements and capabilities, analysis of individual computation and communication components, and agglomeration of the node and network models for complete performance prediction.

The remainder of this article is structured as follows. Section 2 discusses background and related research. The assumptions, attributes, analytical model, and scope of RATSS are discussed in Section 3. In Section 4, a detailed walkthrough of a reasonably complex application, 2D PDF estimation, is presented. Two additional case studies, image filtering and molecular dynamics, are discussed in Section 5. Conclusions are given in Section 6.

2. BACKGROUND AND RELATED RESEARCH

One challenge to application productivity for FPGA-based systems is faster generation of more abstract FPGA design codes. Raising the design focus from traditional hardware description languages, high-level languages such as Impulse C [Pellerin and Thibault 2005], Carte C [SRC Computers 2007], Mitrion C [Mitrionics 2008], and Handel C [Agility Design Solutions 2007] provide a software-like infrastructure for a more efficient and familiar programming model for FPGA applications. Similarly, research in hardware/software codesign enables a faster bridge between application

specification and hardware implementation and a brief history of this research trend can be found in Wolf [2003]. Design environments such Ptolemy [Buck et al. 1994], Metropolis [Balarin et al. 2003], and Artemis [Pimentel et al. 2001] provide rich frameworks for evaluating heterogeneous systems with some extensions for FPGA systems. Though these languages and environments attempt to bridge more abstract design entry to the low-level implementation challenges, the entry point can require significant effort prior to analysis of a target application design.

As part of the ongoing trend towards increasing productivity with explicitly parallel computing devices, significant emphasis is given to application design, optimization, and documentation. Herbordt et al. [2007] propose concepts and general guildlines for achieving high performance in FPGA application designs, effectively boosting productivity by summarizing best practices for developers. Design patterns [DeHon et al. 2004] and loop analysis [Bednara and Teich 2001; Kaul et al. 1999] are two additional research areas benefiting both application performance and development cost. Such methodologies are important considerations even during high-level formulation, as they help ensure design-space exploration accurately reflects the resulting implementation capabilities. However, this body of research only describes a portion of the application design and must be used within a larger performance prediction model to fully quantify the entire system behavior.

Simulation is a common technique for measuring the performance of RC applications, particularly at system level. In Reardon et al. [2009], a framework for simulation of FPGA systems and applications is built on top of the Fast and Accurate Simulation Environment (FASE) [Grobelny et al. 2007], which uses scripts of algorithm and platform behavior to rapidly explore large-scale FPGA systems. Similarly, Bondalapati [2001] combines the Hybrid System Architecture Model (HySAM) with the Dynamically Reconfigurable system Interpretive simulation and Visualization Environment (DRIVE) to parameterize algorithms and platforms, simulate interactions, and visualize results. In Enzler et al. [2005], SimpleScalar and ModelSIM combine for simultaneous software emulation and VHDL simulation. Another tool [Fu and Compton 2006] captures precise memory-access patterns and functionally verifies hardware kernels using a Simics-based simulator. While these simulation environments can provide accurate performance modeling for multi-FPGA systems, they require either actual codes or custom inputs distilled from algorithm and platform behavior. These inputs can be prohibitively expensive to obtain for strategic design-space exploration.

Some analytical models have been proposed to help address development productivity through rapid distillation of quantitative attributes. In Enzler et al. [2000], a model for area and timing is described and while the general concept of design-space exploration is proposed, the focus is on low-level hardware "building blocks." In Steffen [2007], a performance prediction technique seeks to parameterize the computational algorithm and FPGA system, albeit with emphasis primarily on bottleneck detection. Dynamo [Quinn et al. 2007] defines performance analysis for image processing applications constructed at runtime from existing pipelined kernels. Smith and Peterson [2005] propose an analytical model for synchronous, iterative applications on clusters of shared heterogeneous workstations containing reconfigurable computing devices. While these models address some of the challenges for productive application development, they do not directly address the need for analytical, system-level modeling *prior* to any detailed and potentially costly implementation of FPGA kernels or applications. The authors' RC Amenability Test (RAT) [Holland et al. 2009] defines an analytical model for performance estimation of a specific algorithm on a specific platform prior to implementation, albeit for single-FPGA designs. Independent of the primary RAT development, Jacobs et al. [2008] and Nagarajan et al. [2009] further demonstrated the capabilities of RAT for strategic application planning for single-FPGA systems. For

RATSS, the RAT methodology must be paired with system-level modeling concepts to provide a complete model for scalable, multi-FPGA systems.

Existing research with microprocessor-based algorithms and parallel platforms can help bridge the gap between analytical modeling for FPGA devices and the design challenges of FPGA-based scalable systems. The Parallel Random Access Machine (PRAM) [Fortune and Wyllie 1978] is one of the first widely studied models that attempts to reduce complex system behavior into a few key attributes, but the model neglects issues such as synchronization and communication, which can greatly affect the accuracy of the performance estimations. The Bulk Synchronous Parallel (BSP) Model [Valiant 1990] extends the modeling concepts of PRAM by defining an application in terms of a series of global supersteps each consisting of local computation, communication, and synchronization. The LogP [Culler et al. 1993] model attempts to define networks of complete computing nodes (i.e., microprocessors and local memory) by latency, L; overhead, o; gap between (short) messages, g; and number of processing units, P. The LogGP model [Alexandrov et al. 1997] extends the LogP concept with support for a long-message gap, G. Other extensions to LogP and LogGP include support for contention, LoPC [Frank et al. 1997], and parameterization of the L, o, g, G, and P attributes (PlogP) to support dynamically changing values in wide-area networks [Kielmann et al. 1999]. Additionally, benchmarks have been created to assist in the measurement of these attributes [Kielmann et al. 2000].

Prior work has leveraged system-level modeling concepts beyond homogeneous microprocessors. Heterogeneous LogGP (HLogGP) [Bosque and Perez 2004; Bosque and Pastor 2006] considers extensions for multiple processor speeds and communication networks within a cluster. In Lastovetsky et al. [2006], system-level modeling concepts form the basis for a proposed model for heterogeneous clusters. Collective communication modeling and scheduling for node-heterogeneous Networks Of Workstations (NOWs) [Kesavan et al. 1997; Bhat et al. 1999] and clusters of clusters with hierarchical networks [Cappello et al. 2001] are further extensions to traditional system-level modeling. These modifications for heterogeneous computing provide useful insight towards the proposed RATSS model.

3. RATSS MODEL

This section provides a detailed discussion of the structure and contributions of the RATSS model for fast and reasonably accurate performance prediction. This prediction (and consequently, design-space exploration) begins with the designer's specifications of the FPGA platform and algorithm pairing for RATSS analysis. The FPGA platform specification defines the performance capabilities of each component in the system, specifically the computation and communication metrics such as latency, bandwidth, and clock frequency. The algorithm specification defines the computation requirements of every specific task and the resulting communication between devices, which depends on the algorithm/platform mapping. Quantitative attributes are provided for every unique computation and communication task in the FPGA system and these values feed the component-level analytical equations. The RATSS model agglomerates the individual computation and communication predictions based on the system-level schedule defined by the application specification and subsequently provides a quantitative performance estimate for the platform/algorithm pairing. This prediction is used by the designer for further design-space exploration, revising (and reanalyzing) as necessary until the application meets their performance requirements.

Again, RATSS adapts existing computation and communication models to provide a complete performance prediction for an FPGA application (i.e., a *specific* algorithm mapped to a *specific* FPGA platform). RATSS performance prediction is based on efficient quantitative characterization of the key attributes of this algorithm/platform

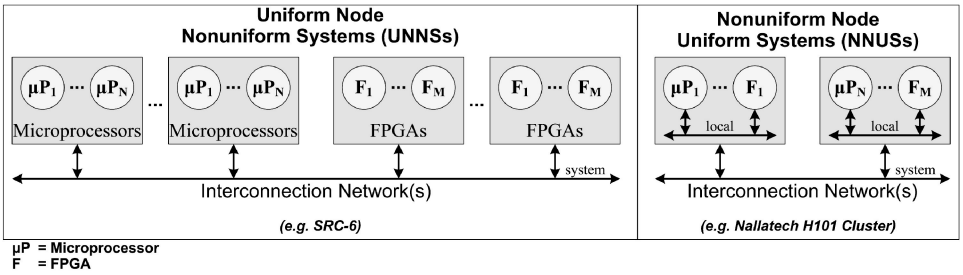


Fig. 1. Adapted from El-Ghazawi et al. [2008], modern high-performance FPGA systems comprise two classes: Uniform Node Nonuniform Systems (UNNSs) and Nonuniform Node Uniform Systems (NNUSs).

pairing. These attributes are explicitly defined by RATSS from adaptations of the underlying models for use in the computation, communication, and ultimately the full application-level models. Section 3.1 discusses how the scope of the platform, algorithm, and their mapping reveal the computation and communication attributes of an application. Section 3.2 discusses the usage of the attributes to form equations that model the performance of the target application.

3.1. RATSS Scope

Analytical models such as LogP are effective because a few key characteristics can describe the performance of a diverse set of application structures. Similarly, FPGA platform and application descriptions, specified by the designer within the model's intended scope, can exploit inherent commonalities for efficient performance prediction. (Conversely, platforms and applications outside the intended scope will not be precisely characterizable by the model attributes.) The following sections discuss how the scope of platform and application features affects the amenability of RATSS performance characterization and ultimately the accuracy of the RATSS performance model.

3.1.1. FPGA Platform Scope. Defining the range of the FPGA platforms applicable to RATSS is necessary for clear and consistent characterization of the performance attributes that form the comprehensive prediction. Even with a reduced scope for FPGA platforms, a key challenge for RATSS is concise modeling of the architectural diversities. Efficient characterization of FPGA platforms provides the necessary insight into the computation and communication capabilities for executing a specified application. As an extension to traditional HPC modeling, RATSS abstracts FPGA platform architectures as compute "nodes" connected by the communication "networks."

Node. One or more devices, tightly coupled, for computation (e.g., microprocessors or FPGAs).
Network. Communication medium connecting two or more nodes.

Figure 1, adapted from El-Ghazawi et al. [2008], illustrates the two major classes of modern high-performance FPGA systems. RATSS supports comprehensive performance prediction for these two classes of systems through node and network models addressing the computation and communication abstractions, respectively. In contrast to traditional homogeneous HPC systems, the presence of FPGAs as application accelerators, orchestrated by microprocessors, can create heterogeneity not only among adjacent devices but also at the system level. For RATSS, nodes are not defined as fixed arrangements of FPGAs and microprocessors, but as an abstraction for one or more devices that can be accurately modeled as a single computational unit (e.g., two FPGAs

with a dedicated interconnect). Conversely, collections of devices are separate nodes if their interconnect requires an explicit communication model.

More simply, RATSS agglomerates computation devices into a single node model if the interaction between the devices does not require a full network model. This simplification is potentially application dependent, but the general assumption is that directly connected FPGAs can be modeled as a single compute node. Similarly, multicore microprocessors or other locally connected SMPs are modeled as single compute nodes. RATSS focuses on explicit network models for system-wide multimicroprocessor interconnects (e.g., Ethernet) and microprocessor/FPGA interconnects (e.g., PCI-based FPGA accelerator cards). From Figure 1, the UNNS architecture requires one system-wide network model and two node models: the microprocessor and FPGA. However, the NNUS architecture will involve two networks: a local interconnect between the FPGA and microprocessor and a system-wide network between the microprocessors. Defining nodes based on their adjoining networks creates a consistent abstraction of computation and communication for both prevailing system classes. This distinction becomes increasingly important as the hierarchy of the FPGA platform increases in depth.

Ultimately, the collection of node and network models provides small, separable descriptions of the complete computation capabilities and communication performance of the FPGA platform. For each piece of computation in the application, the clock frequency attribute for the FPGA defines the overall rate of execution. For each network communication, quantitative attributes include the delay through the interconnect medium (i.e., latency) and bandwidth for message transmission.

3.1.2. Application Scope. Strategic performance prediction requires application characteristics amenable to quantification. An application encompasses an algorithm and its mapping to an FPGA platform.

Algorithm. Finite number of hardware-independent tasks with explicitly defined parallelism and ordering used to solve a problem.

Mapping. Algorithm's computation tasks assigned to nodes and data movement between nodes assigned to one or more communication networks.

A complete description of an algorithm and its mapping must be provided by the designer for effective performance modeling. The composition and parallelization of algorithm tasks defines the computational load for each node and the required communication for each network to support the application. Algorithm and mapping features must be scoped to ensure quantitative characterization of computation and communication interaction that is tractable for analytical modeling. Specifically, the computation and communication of the FPGA application should conform to the synchronous, iterative model.

Synchronous. Computation occurs simultaneously on all nodes and is preceded and proceeded by communication, which collectively define a "stage" of the application.

Iterative. Individual or multiple stages collectively may be repeated as part of the execution.

The synchronous, iterative model requires general application behavior where each computational resource (e.g., microprocessor or FPGA) performs a portion of the required computation during each iteration with synchronizing communication between iterations [Peterson and Chamberlain 1994]. Figure 2, extended from Smith and Peterson [2005], summarizes analytical modeling for synchronous, iterative applications for multi-FPGA systems as scoped by RATSS. The characteristics of the synchronous, iterative model reduce the complexity of computation and communication

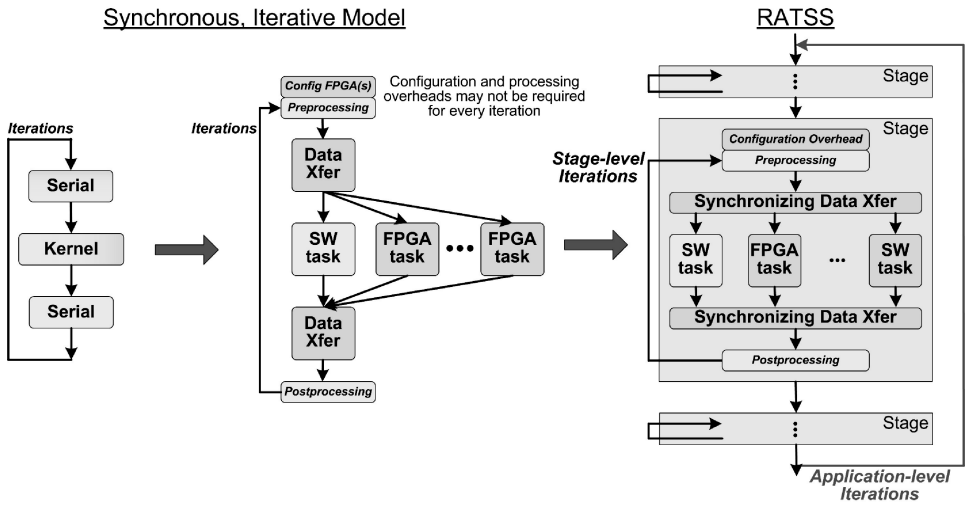


Fig. 2. Adapted from Smith and Peterson [2005], the synchronous, iterative model is expanded for multistage applications. Synchronous communication can occur over one or more networks. Iterative behavior can occur within a stage, over one stage, or over many stages.

overlap and allow straightforward agglomeration of node and network performance by RATSS. Regardless of the platform architecture, problem requirements, and the locality of data, the synchronous behavior of computation and communication defines the total application performance as either the collective summation of the stage times or the single slowest component (assuming steady state). FPGA resources are assumed devoted either all to each stage sequentially (i.e., execution time is related to the summation of stage times) or divided among all stages for pipelined execution (i.e., execution time is related to the slowest stage time). Arbitrary overlap of application stages is outside the scope of this article. Iterative behavior can occur over one stage of execution or the full application. Implicitly, the analytical model for synchronous, iterative behavior also requires deterministic behavior.

Deterministic. Algorithm tasks and data movement between tasks are predictable prior to implementation, either as a constant or an average performance of typical datasets.

Precise characterization of application task scheduling is insufficient for design-space exploration if the underlying computation and communication times cannot be precisely quantified. Randomness in computation and communication behavior requires quantification of application characteristics as averages of expected behavior. The RATSS assumption for deterministic behavior is reasonable as many applications targeting the FPGA paradigm are SIMD-style algorithms implemented as pipelines.

Ultimately, synchronous, iterative, and deterministic behavior allows efficient characterization of computation needs and communication requirements of the FPGA application. Pipelined, SIMD-style algorithms involve data transformations and both the communication and computation are characterized by the quantity of data associated with the particular platform/algorithm mapping. The computational demands of the application are quantified by the number of operations per input data element and the rate of execution (i.e., amount of deep and wide parallelism). Similarly, the attributes for the communication requirements define the amount of data for each network transaction in terms of bytes.

3.1.3. Model Usage and Design-Space Exploration. Again, RATSS involves quantifying the key attributes of the FPGA platform and application for use in the underlying analytical models for performance prediction. These quantitative characteristics are provided largely *by the designer*. Platform-intrinsic attributes such as network latency and throughput are gathered from microbenchmarks that specifically mirror algorithm operations, such as a DMA read and write. Ideally, a database of microbenchmark results is referenced by the designer for the platform attributes, else the benchmarks must be performed prior to any performance prediction. Note that accurate microbenchmarking can be a nontrivial process, albeit with nonrecurring cost due to potential reuse for analysis of future applications with similar platform mapping. Application-specific attributes such as the quantity of data and amount of computational parallelism are explicitly specified by the user based on the algorithm and platform mapping. These attributes feed the equations described in Section 3.2 that compute the performance estimate. The accuracy of the RATSS predictions is dependent on the correspondence between the quantitative attributes and true implementation behavior. Possible discrepancies due to unforeseen issues, such as complex synthesis and translation, can impact application predictability and may require explicit consideration in future predictions. However, these challenges are not unique to strategic DSE with RATSS.

Strategic DSE is a methodology for application design and implementation that involves analysis of one or more platform/algorithm mappings for suitable performance based on provided criteria. RATSS provides suitable analysis capabilities for efficient DSE of multi-FPGA systems. Based on the results of RATSS analysis of an initial application specification, the designer may further refine the target algorithm and/or platform characteristics, proceeding to low-level implementation only when satisfied with the predicted performance. The authors expect that strategic DSE will involve multiple variations on algorithm structure and platform mapping, which would require several repetitions of RATSS analysis with comparison of the predicted performance values against the designer's requirements. However, the accompanying case studies presented in Sections 4 and 5 validate performance prediction accuracy for only the *final* configuration of an application prior to implementation. While detailed strategies for exploration of application designs are outside the scope of this article, these case studies involved revisions to the algorithm structure for maximizing performance of the available hardware resources. The key performance criterion explored in this article is execution time, but issues of application scalability, resource utilization (e.g., load balancing), power-delay product, etc., can also be inferred from the RATSS analysis. Analyses of these issues are not limited to physically realizable systems and can project capabilities of future system configurations.

3.2. Model Attributes and Equations

This section discusses the attributes, equations, and general approach of the node and network models along with their arrangement into stage- and application-level models for RATSS multilevel performance prediction. The attributes and equations for the node and network models leverage existing research from RAT [Holland et al. 2009] and LogGP [Alexandrov et al. 1997] to construct computation and communications models. The platform and algorithm scope provide efficient quantification of performance features of the computation and communication, which serves as input to the analytical models. Essentially, both computation and communication represent the time cost of data movement through a component (e.g., FPGA or interconnect). Eq. (1) defines the general structure for node and network time as the delay overhead through medium/architecture, *delay*; quantity of data, *quantity*; and rate of service, *throughput* (i.e., gap^{-1}). Sections 3.2.1 and 3.2.2 expound on this general equation for the node and network models, respectively. From Eq. (2), this article considers time inversely

proportional to performance (i.e., maximizing performance is minimizing execution time).

$$time = delay + \frac{quantity}{throughput} = delay + quantity \times gap \tag{1}$$

$$time \propto \frac{1}{performance} \tag{2}$$

Sections 3.2.3 and 3.2.4 discuss the multilevel agglomeration of the individual node and network components to model the performance of the individual algorithm stages and subsequently the total application. The synchronous, iterative behavior described in the node and network defines the computation and communication scheduling for each algorithm stage and the overlap of these stages defines the total application performance. The RATSS model uses this multilevel approach to agglomerate the individual performance estimates for the components into a single, quantitative prediction for the application.

3.2.1. Compute Node Model. The goal of the node model is to estimate the performance of each computational task based on the user-provided platform and application attributes. Depending on the application requirements, each of the devices performing a given algorithm task may have different computational demands, each requiring a separate node-level analysis. Similarly, the application will likely have different computational loads for each task (i.e., stage) of the algorithm. Again, the node model is used to describe each unique portion of computation and the individual performance estimates are combined in the RATSS stage-level model.

As summarized in Eq. (3), each node at each stage of execution can have a unique set of attribute values, $\boldsymbol{S}_{fpga}$, which includes the pipeline latency, PL_{fpga}; number of data elements, $N_{fpga\_element}$; number of computational operations per element, $N_{ops\_element}$; FPGA clock frequency, F_{clock}; and computation throughput, R_{fpga}, for the specific algorithm task. Adapted from RAT, the computation time (Eq. (4)) is analogous to Eq. (1) where the pipeline latency is the *delay* term, the number of data elements and number of operations per element are the *quantity*, and the computation throughput and clock frequency define the effective *throughput*.

$$\boldsymbol{S}_{fpga} = \left\{PL_{fpga}, N_{fpga\_elements}, N_{ops/element}, F_{clock}, R_{fpga}\right\} \tag{3}$$

$\boldsymbol{S}_{fpga}$: set of attribute values for a specific computation unit
PL_{fpga}: pipeline latency of the computation (cycles)
$N_{fpga\_element}$: number of computation elements (elements)
$N_{ops/element}$: number of operations per element (ops/element)
F_{clock}: FPGA clock frequency (MHz)
R_{fpga}: computation throughput (ops/cycle)

$$t_{fpga}(\boldsymbol{S}_{fpga}) = \frac{PL_{fpga}}{F_{clock}} + \frac{N_{fpga\_elements} \times N_{ops/element}}{F_{clock} \times R_{fpga}} \tag{4}$$

t_{fpga}: execution time for the fpga compute node (s)

Microprocessor nodes can also impact FPGA application performance with computation coinciding with FPGA execution (from Figure 2). The execution for a microprocessor, $t_{\mu P}$, is defined by the software time, which must be measured from legacy code or estimated using a traditional model. Note that this microprocessor performance attribute is only intended for application-related computation occurring in parallel with

FPGA execution. FPGA setup, configuration, and other software-involved overheads are considered in the stage-level model.

3.2.2. Network Model. The goal of the network model is to estimate the performance of a communication transaction based on the provided platform and application attributes. Analogous to the node model, each unique communication transaction will require a separate network-level analysis, which then combine in the RATSS stage-level model. Based on LogP and its derivatives (e.g., LogGP and the RAT communication model, to an extent), the network model consists of parameters for the latency (i.e., physical interconnect delay), L; overhead, o; message gap, g; number of "processors", P; and message size, k. These attributes are analogous to the general terms from Eq. (1) in that L and o determine *delay*; P and k determine data *quantity*; and g is the *gap*. The key distinction between LogP, LogGP, PLogP, and the RAT communication models is the *gap* parameter, which is defined by the expected behavior of the particular network. Approximations of the short-message gap, g, and long-message gap, G, of LogGP are often sufficient for microprocessor networks such as Ethernet. However, PLogP and RAT define the gap as a function of the message size, $g(k)$.

Determining message size is a key issue for accurate performance prediction. The general assumption is that each node, P, will contribute k bytes of data for the network transaction. However, the message gap, $g(k)$, is highly dependent not only on the volume of data per node but also any subdivision of that data into multiple smaller transfers. Typically, a large message will have less performance overhead than several smaller messages. Ensuring the gap attribute accurately reflects the performance of the actual message segmentation size will reduce modeling errors.

Although specific communication transactions (e.g., MPICH2 implementation of an MPI scatter over Ethernet) have detailed, potentially application-specific performance, the network model introduces general equations for the two types of network communication used in the case studies. Eq. (5) illustrates the individual set of attributes, $\boldsymbol{S}_{transaction}$, for multinode network transactions such as InfiniBand or Ethernet, which includes the L, o, g, G, P, and k attributes along with a γ cost value for any additional required computations (e.g., reduce operations). Eq. (6) defines the performance of the communication transaction, $t_{transaction}$, by the delay as a function, f_{delay}, of the latency and overhead attributes; the volume of data as a function, $f_{quatity}$, of the message size and number of nodes; the short- or long-message gap; and the additional computation, if any, as a function, f_{cost}, of the amount of data and γ cost value.

$$\boldsymbol{S}_{transaction} = \{L, o, g, G, P, k, \gamma\} \tag{5}$$

$\boldsymbol{S}_{transaction}$: set of attribute values for the specific transaction
L, o: LogGP latency and overhead attributes, respectively
g, G: LogGP short and long-message gap, respectively
P, k: LogGP number of nodes and message size, respectively
γ: additional computation for operations such as reduce

$$t_{transaction}(\boldsymbol{S}_{transaction}) = f_{delay}(L, o) + f_{quantity}(P, k) \times [g \text{ or } G] + f_{cost}(P, k, \gamma) \tag{6}$$

$t_{transaction}$: total time for the network transaction
$f_{delay}()$: function defining delay w.r.t. L and o
$f_{quantity}()$: function defining total data quantity w.r.t P and k
$f_{cost}()$: function defining additional computation cost (e.g., reduce)

In contrast to the multinode network model, I/O interconnects between microprocessors and FPGA accelerator cards often exhibit a highly variable gap over a range of message sizes. However, the different gap values for a range of data sizes and transfer types (i.e., DMA to BRAM or read from registers) can be collected prior to application analysis using microbenchmarks and reused on future applications with similar I/O communication. These attribute values are either collected into a table for reference or used to construct an explicit $g(k)$ function. From Eq. (7), the original RAT model separated individual gap values into the theoretical interconnect throughput, R_{IO}, and the efficiency of the interconnect for the message size, Eff_{IO}. Similarly, the message size was decomposed into the number of data elements, $N_{elements}$, and number of bytes per element, $N_{bytes/element}$ (Eq. (8)). Expressing data in terms of elements allowed more direct correlation between the volume of computation and the amount of communication. However, the RAT I/O model is adjusted to coincide with the LogGP formulation for consistency within the network model.

$$g(k) = (R_{IO} \times Eff_{IO})^{-1} \tag{7}$$

R_{IO}: theoretical throughput rate of I/O channel (from RAT)

Eff_{IO}: efficiency of I/O channel (from RAT)

$$k = N_{IO\_elements} \times N_{bytes/element} \tag{8}$$

$N_{IO\_elements}$: number of I/O elements (from RAT)

$N_{bytes/element}$: number of bytes per element (from RAT)

Eq. (9) defines the set of attributes, $\boldsymbol{S}_{IO}$, for the revised I/O transaction model, which consists of the latency and overhead delays, L and o; size-dependent message gap, $g(k)$; number of nodes, P; message size, k; and the additional computation cost, γ, for the communication, if any. Though the I/O model represents a point-to-point interconnect, the P value remains to represent unidirectional (i.e., $P = 1$) or bidirectional (i.e., $P = 2$) behavior. Eq. (10) defines the communication for the I/O transaction, t_{IO}, by the delay function, f_{delay}, for latency and overhead; number of nodes; message size; gap; and additional computation as a function, f_{cost}, of the message size and γ cost value.

$$\boldsymbol{S}_{IO} = \{L, o, g(k), P, k, \gamma\} \tag{9}$$

$\boldsymbol{S}_{IO}$: set of attribute values for the I/O transaction

L, o: latency and overhead attributes, respectively

$g(k)$: gap as a function of message size, k

k: message size

γ: additional computation cost

$$t_{IO}(\boldsymbol{S}_{IO}) = f_{delay}(L, o) + P \times k \times g(k) + f_{cost}(k, \gamma) \tag{10}$$

t_{IO}: total time for the I/O transaction

$f_{delay}()$: function defining delay w.r.t. L and o

$f_{cost}()$: function defining additional computation cost

3.2.3. Stage Model. As discussed in Section 3.1.2, the RATSS stage model represents the collective scheduling of the one or more repetitions of computation and communication for an algorithm task. The set of computation times, $\boldsymbol{S}_{comp}$, contains the individual FPGA execution times, t_{fpga}, for each of the n nodes involved in the application stage (Eq. (11)). Often, all available nodes are performing the specific task, but some stages may require less computation time and use only n nodes where $n < P$. Thus, the total computation time, t_{comp}, for a stage of an application is the sum of the software preprocessing overhead, $t_{preprocessing}$, the maximum (longest) individual execution times

from the FPGA nodes and microprocessor, and the software postprocessing overhead, $t_{postprocessing}$ (Eq. (12)). As illustrated in Figure 2, the software processing overheads can share resources with the actual computation and occur in serial with the node-level computation. Software processing includes overheads such as file reading and writing, but can be considered negligible depending on the application implementation characteristics. The software processing overheads for the case studies in Sections 4 and 5 are considered insignificant relative to the computation and communication times, and are not explicitly modeled.

$$\boldsymbol{S}_{comp} = \left\{t_{fpga_1} \cdots t_{fpga_n}\right\} \tag{11}$$

$\boldsymbol{S}_{comp}$: set of FPGA execution times for the stage

n: total number of FPGA nodes

$$t_{comp} = t_{preprocessing} + Max(Max(\boldsymbol{S}_{comp}), t_{\mu P}) + t_{postprocessing} \tag{12}$$

t_{comp} total computation time for the stage

$t_{preprocessing}$ any software overhead prior to computation

$t_{postprocessing}$ any software overhead after computation

Similarly, the set of communication times, $\boldsymbol{S}_{comm}$, contains the performance estimates for each of the τ network transaction times, $t_{transaction}$ (Eq. (13)). Typically, this set will contain one or more input and output transactions for each level of network communication in the platform, though some applications will instead accumulate partial results within the FPGAs over multiple stages with cumulative output after the last computation iteration. From Eq. (14), the communication time for the stage, t_{comm}, is composed of the sum of the τ transaction times, $t_{transaction}$. Multiple levels of communication within an application stage are assumed nonoverlapping due to blocking. However, nonblocking transactions can be modeled by the total network delay and longest (i.e., maximum) throughput time.

$$\boldsymbol{S}_{comm} = \left\{t_{transaction_1} \cdots t_{transaction_\tau}\right\} \tag{13}$$

$\boldsymbol{S}_{comm}$: set of transaction times for the stage

τ: total number of communication transactions

$$t_{comm}(\boldsymbol{S}_{comm}) = \sum S_{comm} \tag{14}$$

t_{comm}: total communication time for the stage

Depending on the FPGA platform architecture and mapping, computation and communication within a stage is either serialized or overlapping with the total execution time of the stage, t_{stage}, defined as the configuration overhead, $t_{configuration}$, plus the number of iterations, $N_{stage\_iterations}$, of either the sum or maximum of the t_{comp} and t_{comm} terms (Eq. (15)). Note that performance estimates should be modeled for each unique stage of the application execution with attention to any special cases, such as initial and final stages possessing more or less communication.

$$t_{stage} = t_{configuration} + N_{stage\_iterations} \times \begin{cases} t_{comp} + t_{comm} \\ Max(t_{comp}, t_{comm}) \end{cases} \tag{15}$$

t_{stage}: total execution time of the application stage

$t_{configuration}$: any FPGA configuration overhead for the stage

$N_{stage\_iterations}$: number of stage-level iterations

3.2.4. Application Model. The RATSS application-level model describes the scheduling of the individual stages of task execution to estimate the full system performance.

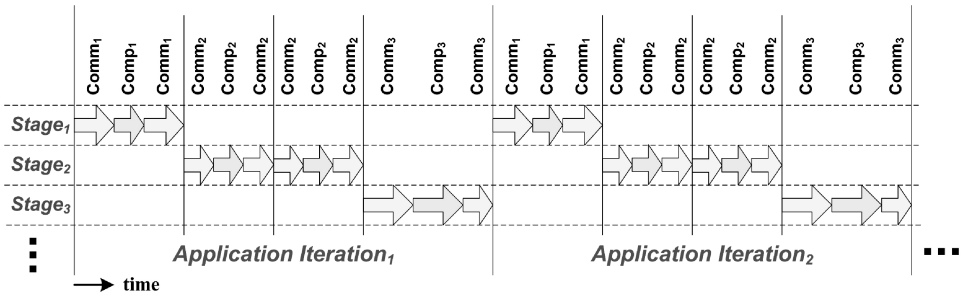


Fig. 3. Example timing diagram illustrating stage- and application-level scheduling of computation and communication.

Applications consist of one or more distinct stages of execution, which may be collectively repeated for one or more of iterations. Analogous to the computation and communication scheduling, application stages can either be serialized or overlapping. Figure 3 provides an example timing diagram for potential iterative behavior at application level. The example consists of three stages collectively repeated twice, reinforcing the multilevel iterative behavior of the stage and application models as first described in Figure 2. This sample application does not illustrate all potential platform and algorithm features such as a complex network hierarchy, but instead reinforces the ability of RATSS to organize execution paths into multilevel models.

Eq. (16) defines the set, $\boldsymbol{S}_{stage}$, of s stage times, t_{stage}, for the application. The total execution time for the application, $t_{application}$, is the number of iterations, $N_{app\_iterations}$, of either the sum or maximum (longest) of the stage times. From Section 3.1.2, FPGA resources are either collectively used on stages serially (i.e., execution time is the summation of stage times) or divided among stages for pipelined application execution at stage level (i.e., execution time is the slowest stage time). Arbitrary assignment of FPGA resources and overlap of stage execution is possible, but outside the scope of this article. Again, this model generalizes high-level iterative behavior. Applications can contain repetitive but irregular behavior (e.g., 3 iterations of stage one, 7 iterations of stage two, 5 iterations of stage three, etc.), which is simple to calculate but not explicitly considered by the model.

$$\boldsymbol{S}_{stage} = \left\{ t_{stage_1} \cdots t_{stage_s} \right\} \tag{16}$$

$\boldsymbol{S}_{stage}$: set of stage times for the application
s: total number of stages for the application

$$t_{application} = N_{app\_iterations} \times \begin{cases} \sum \boldsymbol{S}_{stage} \\ Max(\boldsymbol{S}_{stage}) \end{cases} \tag{17}$$

$t_{application}$: total execution time of the application
$N_{app\_iterations}$: number of application-level iterations

4. DETAILED WALKTHROUGH: 2D PDF ESTIMATION

This section presents a detailed walkthrough of RATSS performance prediction with a reasonably complex case study, 2D PDF estimation. The intended algorithm and platform structure along with the feature characterization and performance calculations for the node, network, stage, and application models are discussed. The results of the performance estimation are compared against a subsequent hardware implementation to evaluate the accuracy of the RATSS model.

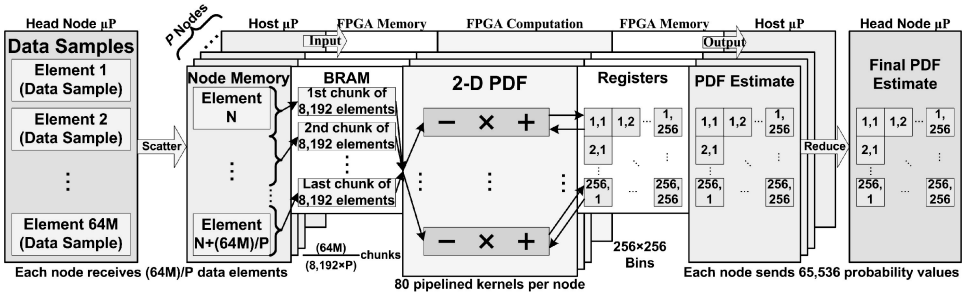


Fig. 4. Application structure for 2D PDF estimation case study.

4.1. Algorithm and Platform Structure

The 2D PDF estimation algorithm for this case study uses the Parzen window technique [Parzen 1962], a generalized nonparametric approach to estimating PDFs in a d-dimensional space. Despite the increased computational complexity versus traditional histograms, the Parzen window technique is mathematically advantageous because of the rapid convergence to a continuous function. This algorithm is amenable for FPGA acceleration because of the high degree of computational parallelism and large computation effort relative to the amount of data consumed (i.e., input) and produced (i.e., output). The computational complexity of a d-dimensional PDF algorithm is $O(Nn^d)$, where N is the total number of samples of the random variable, n is the number of levels where the PDF is estimated, and d is the number of dimensions. This 2D PDF estimation algorithm accumulates the statistical likelihood of every sample occurring within every probability level. Each sample/level combination is independent, thereby making the algorithm embarrassingly parallel. The data input consists of $O(N)$ samples whereas the output is the resulting $O(n^2)$ probability levels.

A general overview of the algorithm structure for this case study is presented in Figure 4. A total of 67,108,684 (i.e., 64M) data samples, originating on one microprocessor, are scattered equally among the P microprocessors. The number of samples is large to fully stress the communication and memory capabilities of the target FPGA platform. The microprocessors transfer the data to their respective FPGA node in chunks of 8,192 data samples, limited by the available on-chip block RAM. A total of 80 pipelined kernels per node perform the necessary computations (comparison, scaling, and accumulation) to analyze each data sample against the 256×256 probability levels. The 80 parallel kernels maximize the available multiple accumulators (MACs) on the target FPGA with some leeway. The numerical precision for the computation is 18-bit fixed point. The results are accumulated in 256×256 registers and periodically read back by the host microprocessor. The resulting 256×256 partial sums on each of the P nodes are collected with a reduce operation. More discussion on this 2D PDF estimation architecture along with general issues related to FPGA implementation can be found in Nagarajan et al. [2009].

The intended platform for this case study is illustrated in Figure 5. The full platform consists of eight 3.2GHz Xeon microprocessor nodes each connected to one Xilinx XC4VLX100 (Nallatech H101 card) via a PCI-X bus. The processing nodes are organized as a cluster of traditional computers each augmented with FPGA hardware (i.e., an NNUS system). Each XC4VLX100 FPGA contains 96 MACs providing sufficient hardware resources for the 80 parallel kernels (i.e., one MAC per kernel). The on-chip Block RAMs (BRAMs) are explicitly illustrated since they are used to store the input and output data for the 2D PDF case study. The microprocessor nodes are connected via Gigabit Ethernet. Network-level communication uses the MPICH2 implementation of

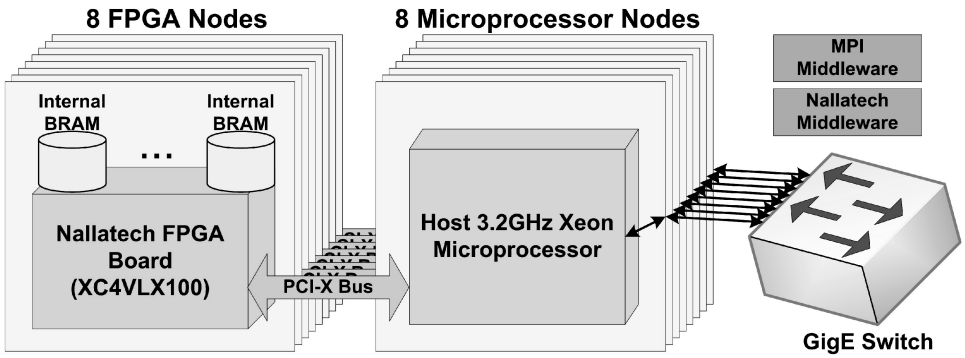


Fig. 5. Platform structure for 2D PDF estimation case study.

Table I. Node Attributes for 2D PDF Estimation

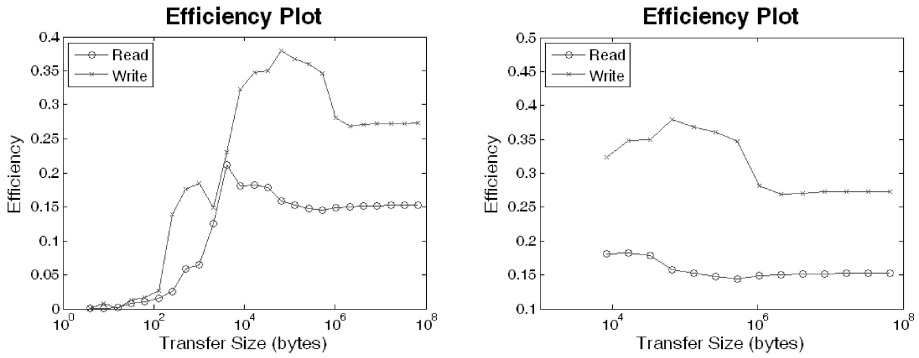
Attribute	*Units*	*2 Nodes*	*4 Nodes*	*8 Nodes*
PL_{comp}	(cycles)	11		
R_{comp}	(ops/cycle)	240		
F_{clock}	(MHz)	195		
$N_{comp\_elements}$	(elements)	33554432	16777216	8388608
$N_{ops/element}$	(ops/element)	196,608		
t_{fpga}	(s)	1.41E+2	7.05E+1	3.52E+1

the Message Passing Interface (MPI). The case study is modeled and the implementation is tested using 2, 4, and 8 FPGA nodes.

The Nallatech H101 card allows user-specified clock frequencies for applications, which can necessitate further DSE. Typically, clock frequency is difficult to estimate at a strategic level and RATSS should be performed for a range of potential clock frequencies to help ensure satisfactory performance for even more pessimistic timing. RATSS predictions based on revisions to a single attribute value, such as clock frequency, should require trivial additional efforts. This case study describes only the RATSS prediction matching the actual clock frequency of the subsequent FPGA implementation. However, many frequencies were explored as part of the strategic DSE.

4.2. Compute Node Modeling

The node-level model consists of estimating the computation for the 2, 4, and 8 Nallatech-augmented compute nodes. The values in Table I consist of the computation attributes, which are distilled from the structure of the 2D PDF estimation algorithm as mapped to the architecture of the FPGA node. For computation, accurate parameterization of the pipeline latency, PL_{comp}, requires detailed knowledge of the final algorithm structure. The pipeline for the 2D PDF estimation has a straightforward computational structure of three operations (subtraction, multiplication, and addition from Figure 4) requiring 11 total cycles. The relatively deep pipeline helps ensure a higher clock frequency. The computational throughput, R_{comp}, of 240 operations per cycle comes from the 80 pipelined kernels, each with 3 simultaneous operations per pipeline. Predictions are generated for a large range of possible frequencies. The prediction for the clock frequency, F_{clock}, of 195MHz is shown since it ultimately matched the maximum frequency for later implementation. The number of computation elements, $N_{comp\_elements}$, is 33,554,432 (64M $\div$ 2); 16,777,216 (64M $\div$ 4); and 8,388,608 (64M $\div$ 8) for the 2- 4- and 8-node configurations, respectively, due to the balanced data decomposition. The number of operations per element, $N_{ops/element}$, is based on the 256 $\times$ 256



(a) range of transfer sizes (b) large transfer sizes

Fig. 6. Results of efficiency microbenchmarks for Nallatech BRAM I/O.

comparisons per data element times 3 operations per element for a total of 196,608 operations.

The last attribute in Table I, t_{fpga}, summarizes the computation time for the 2-, 4-, and 8-node cases. Each node for a particular system size (i.e., number of nodes, P) will have an identical execution time because of the equal data decomposition. Due to the increasing number of node resources, FPGA computation time, t_{fpga}, decreases approximately linearly. Two FPGAs require twice the time as four FPGAs which need twice the time of eight FPGAs. This behavior is consistent with the embarrassingly parallel nature of the 2D PDF estimation algorithm.

4.3. Network Modeling

For the FPGA platform used in this case study, two communication network models are necessary: PCI-X I/O Bus and Ethernet. The PCI-X bus model describes the point-to-point interconnect between a host microprocessor and its Nallatech FPGA node. The Ethernet model describes the MPICH2 communication over the Gigabit Ethernet network. Assembling the attribute values for these models involves not only analysis of the algorithm structure and mapping but also microbenchmarking of the underlying platform behavior for typical communication transactions.

4.3.1. PCI-X Bus Modeling. The I/O operations for 2D PDF estimation involve transfers between the host CPU and the onboard FPGA block RAM. Microbenchmarks were performed on common transfer sizes (i.e., powers of two from 4B to 64MB). Figure 6 summarizes the results of these transfers, which can be referenced for all future I/O performance estimations. Smaller transfers, Figure 6(a), have erratic but steadily increasing efficiency whereas larger transfers, Figure 4.3.1, could be approximated by a single value. For the transfer sizes used in this case study (writing 8,192 elements and reading 65,536 elements, discussed later), the I/O efficiencies are 0.31 and 0.10, respectively.

Table II summarizes the delay and throughput attributes, gathered from microbenchmarks of the PCI-X I/O bus, along with the quantity of data transmitted for the 2D PDF estimation case study. The microbenchmarks measure the total time of a data transfer, which defines the effective throughput for a given transfer size. The I/O latency and overhead, $L_{IO} + o_{IO}$, is assumed to be the total transfer time for a very small transfer (i.e., 4B of data), which is dominated by the channel delay of the PCI-X bus. For writes and reads with the FPGA block RAM, the measured performance is 1.60E-5(s) and 3.20E-5(s), respectively. The gap, $g(k)$, for the write and read I/O transactions is the

Table II. PCI-X Bus Attributes for 2D PDF Estimation

Attribute		*Units*	*2 Nodes*	*4 Nodes*	*8 Nodes*
$L_{IO} + o_{IO}$	write	(s)	1.60E–5		
	read		3.20E–5		
$g(k)_{IO}$ $(R_{IO} \times Eff_{IO})$	write	$(s)^{-1}$	3.03E–3 $(1{,}064 \times 0.31)^{-1}$		
	read		9.40E–3 $(1{,}064 \times 0.10)^{-1}$		
k $(N_{IO\_elements} \times N_{bytes/element})$	write X	(elements)	128M (32M×4)	64M (16M×4)	32M (8M×4)
	write Y		128M (32M×4)	64M (16M×4)	32M (8M×4)
	read		1024M (256M×4)	512M (128M×4)	256M (64M×4)
$t_{transaction}$	write X	(s)	4.07E–1	2.03E–1	1.02E–1
	write Y		4.07E–1	2.03E–1	1.02E–1
	read		1.01E+1	5.05E+0	2.52E+0

Table III. Ethernet Network Attributes for 2D PDF Estimation

Attribute		*Units*	*2 Nodes*	*4 Nodes*	*8 Nodes*
L		(s)	1.08E–4		
o		(s)	6.75E–6		
g		(s)	1.64E–5		
G		(s/Byte)	9.56E–9		
γ		(s/Byte)	1.90E–8		
P		(nodes)	2	4	8
k	scatter X	(Bytes)	128M	64M	32M
	scatter Y		128M	64M	32M
	reduce		256K	256K	256K
$t_{transaction}$	scatter X	(s)	1.28E+0	1.92E+0	2.25E+0
	scatter Y		1.28E+0	1.92E+0	2.25E+0
	reduce		3.89E–3	7.78E–3	1.17E–2

multiplicative inverse of the 1,064MB/s (i.e., 33MHz, 64-bit PCI-X) theoretical throughput, R_{IO}, times the I/O efficiency, Eff_{IO}. These particular $g(k)$ values are determined by the message size, k, which is defined by the number of I/O elements, $N_{IO\_elements}$, and number of bytes per element, $N_{bytes/element}$. For the write I/O, the 64M input data elements for each of the X and Y dimension are divided among the 2, 4, and 8 nodes for the $N_{IO\_elements}$ term. Again, these write transfers are divided into blocks of 8,192 elements meaning 64M $\div$ 8,192 $\div$ P distinct transfers. The output (i.e., read I/O) involves collecting the 65,536 (256×256) elements storing the partial PDF estimates for each of the 64M $\div$ 8,192 $\div$ P iterations. Though the computation is 18-bit fixed point, the data format for the I/O transfers is 32-bit integer and consequently the number of bytes per element, $N_{bytes/element}$, is 4.

Eq. (18) defines the performance for PCI-X write and read transaction, $t_{transaction_{write,read}}$, by the I/O latency and overhead, $L_{IO} + o_{IO}$, gap value for the message size, $g(k)$, and message size, k. The I/O results of the two writes and one read for this case study are summarized in the second block of Table II.

$$t_{transaction_{write,read}} = L_{IO} + o_{IO} + g(k) \times k \tag{18}$$

4.3.2. Ethernet Network Modeling. The attributes in the first block of Table III were gathered using the LogGP benchmarking tool described in Kielmann et al. [2000]. This tool computes the parameterized LogP functions, which are converted to the fixed latency, L, overhead, o, short-message gap g, and long-message gap, G, values. Much of the tool's methodology is outside the scope of the article, but essentially a progressively

increasing sequence of messages is used to calculate the gap. The latency and overhead are deduced from the delays between these messages. The computational overhead, γ, consists of the addition operations for the reduce transaction used in the 2D PDF estimation and is benchmarked on the Xeon microprocessor. Using these parameters, precise analytical models can be constructed for the binomial tree scatter and reduce used in this case study.

The analytical model for the binomial tree scatter used in the MPICH2 implementation of MPI is defined in Eq. (19). As opposed to a naive scatter, which requires $P-1$ messages to complete a P-node scatter, a binomial tree scatter only requires $log_2(P)$ messages as each node that has received data subsequently scatters to other remaining nodes. However, these messages for the binomial tree scatter begin as half the total data to be scattered and decrease by half with each transmission because nodes must be supplied with not only their data but also the data they must pass on. In actuality, the binomial tree scatter uses more network bandwidth (i.e., many transmissions in parallel) and the throughput component is the same as a naive scatter. The advantage of the binomial tree scatter is the reduction in latency.

$$\begin{aligned} t_{transaction_{scatter}} &= log_2(P) \times L + 2o + G \sum_{X=1}^{log_2 P} \frac{P}{2^X} k \\ &= log_2(P) \times L + 2o + G\left(\frac{P}{2} + \frac{P}{4} + \cdots + \frac{P}{P}\right)k \\ &= log_2(P) \times L + 2o + G(P-1)k \end{aligned} \tag{19}$$

For the shorter message sizes necessary for data collection in the 2D PDF estimation algorithm, the MPICH2 implementation of the MPI_Reduce function uses a binomial tree similar to the scatter. However, unlike scatter (or gather) the amount of data during each transmission does not increase because the data is reduced at every node. Consequently, the reduce has $log_2(P)$ latency, L, and transmission time, Gk, as defined in Eq. (20). Additionally, each transmission requires an addition operation, γ, for each of the k data values in the message. Note that Eqs. (19) and (20) assume P is a power of 2.

$$t_{transaction_{reduce}} = log_2(P) \times (L + 2o + Gk + \gamma k) \tag{20}$$

The second block of Table III lists the two application-dependent attributes defined by the user based on the 2D PDF estimation case study. Again, system configurations of 2, 4, and 8 nodes, P, are used for this case study. The 2D PDF application requires two distinct transactions: distribution of the input data for the X and Y dimensions (i.e., MPI_Scatter) and reduction of the partial PDFs (i.e., MPI_Reduce). The 2, 4, and 8 FPGA platform configurations will involve message sizes, k, of 128MB, 64MB, and 32MB of data, respectively for the scatter. For the reduce, every node will contribute the 256KB ($256 \times 256 \times 4$B) partial results (regardless of the number of nodes) that are ultimately accumulated on the head node.

The third block of Table III summarizes the results of the network model. The individual times for the scatter and reduce transactions, $t_{transaction}$, are listed. These times increase logarithmically for the 2-, 4-, and 8-node platforms due to the increasing number of messages (i.e., $log_2(P)$) required for the transaction.

4.4. Stage/Application Modeling

For this analysis, the stage and application models are combined due to the single stage of execution in the application. Eq. (21) summarizes the set, $\boldsymbol{S}_{comp}$, of the execution times, t_{fpga}, for the 2, 4 and 8 (i.e., P) FPGA nodes used in this case study. From Eq. (22), the computation time, t_{comp}, is determined by the maximum (longest) node

Table IV. System Model Attributes for 2D PDF Estimation

Attribute		*Units*	*2 Nodes*	*4 Nodes*	*8 Nodes*
$N_{stage\_iterations}$		(iterations)	1		
$\boldsymbol{S}_{comp}$		(s)	1.41E+2	7.05E+1	3.52E+1
$\boldsymbol{S}_{comm}$	scatter X	(s)	1.28E+0	1.92E+0	2.25E+0
	scatter Y		1.28E+0	1.92E+0	2.25E+0
	write X		4.07E–1	2.03E–1	1.02E–1
	write Y		4.07E–1	2.03E–1	1.02E–1
	read		1.01E+1	5.05E+0	2.52E+0
	gather		3.89E–3	7.78E–3	1.17E–2
t_{comp}		(s)	1.41E+2	7.05E+1	3.52E+1
t_{comm}		(s)	1.35E+1	9.31E+0	7.25E+0
t_{stage}		(s)	1.54E+2	7.98E+1	4.24E+1

time. Because of the equal load balancing among the nodes, the total computation time is equivalent to the performance of any i-th node in the system.

$$\boldsymbol{S}_{comp} = \{t_{fpga_1} \cdots t_{fpga_P}\} \tag{21}$$

$$t_{comp}(\boldsymbol{S}_{comp}) = Max(\boldsymbol{S}_{comp}) = t_{fpga_i} \tag{22}$$

The set of communication times, $\boldsymbol{S}_{comm}$, summarizes network transactions involved in the single stage of the case study. The scatter and write times, $t_{scatter}$ and t_{write}, represent the X and Y dimensions of the input data and the read and reduce times, t_{read} and t_{reduce}, define the data collected after computation. From Eq. (24), the total communication performance, t_{comm}, is the summation of the individual, nonoverlapping network transactions.

$$\boldsymbol{S}_{comm} = \{t_{scatter_X}, t_{scatter_Y}, t_{write_X}, t_{write_Y}, t_{read}, t_{reduce}\} \tag{23}$$

$$t_{comm}(\boldsymbol{S}_{comm}) = t_{scatter_X} + t_{scatter_Y} + t_{write_X} + t_{write_Y} + t_{read} + t_{reduce} \tag{24}$$

The inputs to the RATSS system-level model are summarized in the first block of Table IV. The only user-provided attribute is the number of stage-level iterations, $N_{iterations}$. The 2D PDF application only requires one iteration of node and network interaction (i.e., internode data distribution, node-level computation, internode data collection). The individual node and network transaction times comprise the majority of the input to the system model. The $\boldsymbol{S}_{comp}$ and $\boldsymbol{S}_{comm}$ attribute sets contain the compute node and network transaction times from the respective models. The second block of Table IV summarizes the estimated performance of the total computation and communication times, t_{comp} and t_{comm} from Eqs. (22) and (24). As expected, the computation time decreases as the number of FPGAs increases. The communication time also decreases as the number of FPGAs increases due to the shorter I/O transactions (i.e., $\frac{N_{IO\_elements}}{P}$) that dominate the total communication time.

The 2D PDF estimation does not use an elaborate buffering scheme so the total performance of the application, $t_{application}$, as defined by the stage time, t_{stage}, is the summation of the number of iterations, $N_{stage\_iterations}$, of I/O communication and FPGA computation for the node, t_{comm} and t_{comp} (Eq. (25)). Single buffering maximizes the available memory bandwidth to the computation units. The computation time, as shown in Table IV, dominates the total application execution time and would not greatly benefit from double buffering.

$$t_{application} = t_{stage} = N_{stage\_iterations} \times (t_{comm} + t_{comp}) \tag{25}$$

The model output is summarized in the third block of Table IV. As the number of nodes doubles, the node time reduces by half but the network time increases slightly. Consequently, the total execution time, t_{total}, decreases by slightly less than half as the

Table V. Modeling Error for 2D PDF Estimation (Nallatech, XC4VLX100, 195MHz)

		Predicted (s)	Experimental (s)	Error
2 FPGAs	t_{comp}	1.41E+2	1.56E+2	–9.6%
	t_{comm}	1.35E+1	1.51E+1	–10.7%
	t_{total}	1.54E+2	1.71E+2	–9.7%
	Speedup	146	132	10.6%
4 FPGAs	t_{comp}	7.05E+1	7.84E+1	–10.1%
	t_{comm}	9.31E+0	9.93E+0	–6.2%
	t_{total}	7.98E+1	8.84E+1	–9.7%
	Speedup	283	255	11.0%
8 FPGAs	t_{comp}	3.52E+1	3.95E+1	–10.9%
	t_{comm}	7.25E+0	7.70E+0	–5.9%
	t_{total}	4.24E+1	4.72E+1	–10.1%
	Speedup	532	478	11.3%

number of nodes doubles. This trend of nearly linear performance improvement with increasing platform size is reasonable given the embarrassingly parallel nature of the computation and relatively low impact of the PCI-X and Ethernet communication.

4.5. Results and Verification

As previously discussed, the performance prediction is calculated *prior* to low-level design and was not adjusted based on implementation details. Predictions were generated for a range of clock frequency values, though only the results of the 195MHz estimation are shown. Major revisions to the target algorithm or platform architecture during implementation can significantly alter the application performance affecting the validity of the prediction. Thus, RATSS can be used iteratively throughout the design process, recomputing predictions whenever significant revisions are considered or become necessary to ensure the subsequent implementation will still meet performance requirements and thereby prevent further reductions in productivity. However, such modifications to the application structure were not necessary for the 2D PDF estimation case study.

In Table V, the results of the performance prediction for the 2D PDF estimation case study are compared against a subsequent implementation of the target algorithm. The node and network models underestimated the actual implementation times and subsequently overestimated the total application speedup over the software baseline. The node times represented the majority of the execution time (over 90% of the physical implementation) thereby having the greatest impact on prediction accuracy. The prediction errors for the 2, 4, and 8 FPGA configurations are under 11%, which is considered reasonably accurate given the focus on high-level design-space exploration prior to implementation. Most of the discrepancy is due to additional cycles of overhead related to data movement during the FPGA computation. In contrast to the node modeling error, which remained relatively constant for 2, 4, and 8 nodes, the network modeling error had significantly more variability. The error for the 2-node platform was just under 12%. Though not ideal, the error had minimal impact on the overall prediction accuracy and was due to extra overhead associated with a 2-node scatter and reduce versus the single message predicted by the model. The network error was 6.2% and 5.9% for the 4- and 8-node platforms, respectively. This network error, lower than the respective node error, reduced the overall execution error. Consequently, the predicted speedup over a sequential software baseline executed on one 3.2GHz Xeon microprocessor in the platform was approximately 90% accurate (i.e., under 12% error) for all three system sizes. The speedup values are substantial due to the large problem size and high computation-to-communication ratio.

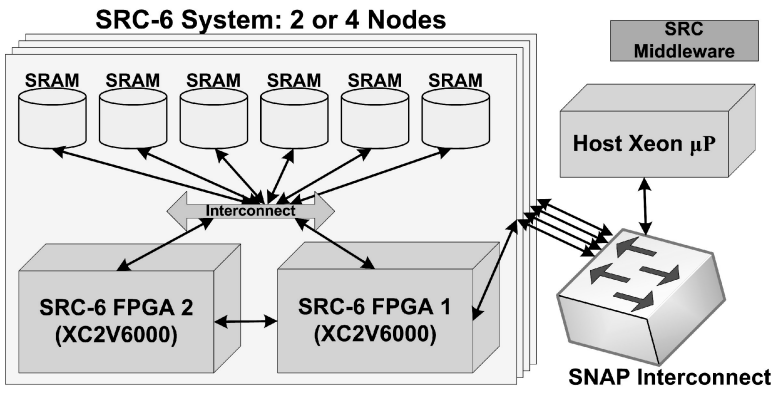


Fig. 7. Platform structure for image filtering case study.

Table VI. SNAP Network Attributes for SRC-6 System

Attributes	*Units*	*Value*
L	(s)	1.01E–5
o	(s)	o(P,k) [negligible]
g	(s)	6.40E–7
G	(s/Byte)	1.25E–9

A key lesson learned from this case study relates to modeling effort versus accuracy. The node characterization and performance estimation required significantly less effort as compared to microbenchmarking and modeling the two communication networks. However, precise node modeling is important as the case study consists of over 90% computation and more effort related to modeling unforeseen computational overheads could benefit future applications targeting this platform. The need for greater accuracy in communication modeling is situational, as illustrated in the next two case studies, image filtering and MD, which have significant and trivial communication, respectively.

5. ADDITIONAL CASE STUDIES

This section discusses two additional case studies for further validation of the RATSS model: image filtering and molecular dynamics. Both case studies use an SRC-6 system (i.e., an UNNS system), which incorporates up to four MAP-B nodes each consisting of two Xilinx XC2V6000 FPGAs (Figure 7). In contrast to the Nallatech-based platform, the application clock frequency for the SRC-6 system is fixed at 100MHz. All four MAP-B nodes are connected to a dual-Xeon microprocessor workstation via a single, proprietary interconnect called SNAP. In contrast to the Nallatech-augmented cluster from the 2D PDF estimation case study, initiation of communication is performed locally by each node as a DMA transfer and arbitrated by the interconnect controller. Network transactions are still described using collective communication terminology but the physical operations are independently initiated point-to-point messages. Configuration and other overheads are considered negligible. Additionally, case study implementations are written in Carte C.

Common to both case studies, Table VI defines the network attributes for the SNAP interconnect, which are measured from microbenchmarks. Similar to the Nallatech system, latency, L, is the transmission time of a single-word transfer, which is dominated by the network delay. However, overhead, o, the time between successive messages, is a function of the number of nodes, P, and message size, k, not a constant parameter

due to the decentralized communication requests. More extensive microbenchmarking can determine approximate network overhead values although variability in message ordering inhibits detailed analysis. For these case studies, the effect of overhead is considered negligible and not part of the network model. The short-message gap, g, is measured as the transmission time for short messages minus latency. No short messages are used in these case studies but the value is included for completeness. The long-message gap, G, is one 8-byte word every 10ns clock cycle based on the fixed 100MHz clock.

The equations for the SNAP communication, broadcast, scatter, and gather, are defined in Eqs. (26) and (27). In general, SRC-6 transactions will be a series of serialized messages, albeit of different sizes. However, the independently initiated DMA transfers are designed to allow node computation to proceed before all communication has completed. For an accurate yet simple model of this partial overlap, the output transaction (i.e., gather or another collection-type operation) can be represented by just the final message of the communication, which cannot be hidden because computation has completed, instead of the total P messages.

$$t_{transaction_{broadcast,scatter}} = L + G \times P \times k \tag{26}$$

$$t_{transaction_{gather}} = \begin{cases} L + G \times k, & \text{overlapping communication} \\ L + G \times P \times k, & \text{nonoverlapping communication} \end{cases} \tag{27}$$

For brevity, the performance model for the image filtering and MD case studies (and many other SRC-6 applications) is summarized in Eqs. (28)–(30). Again, application execution is defined by FPGA computation with synchronizing DMA communication to a central microprocessor node. Distributed data movement incorporating global common memories can help overlap communication and computation but is not considered for the following case studies. The longest node time defines the computation time (Eq. (28)) and the communication consists of broadcast or scatter, and gather (Eq. (29)). The total application performance is the summation of the computation and communication times (Eq. (30)).

$$t_{comp} = Max(\{t_{fpga_1} \cdots t_{fpga_P}\}) \tag{28}$$

$$t_{comm} = t_{broadcast,scatter} + t_{gather} \tag{29}$$

$$t_{application} = t_{stage} = N_{stage\_iterations} \times (t_{comp} + t_{comm}) \tag{30}$$

Although these two additional case studies involve a different FPGA platform from the 2D PDF application, the sequential software baselines are measured from the same 3.2GHz Xeon microprocessor for consistency. While speedup is often an advantageous performance metric, the specific speedup value must be compared with the problem size and computation-to-communication ratio for the application. The image filtering and molecular dynamics case studies illustrate communication- and computation-bound problems, respectively, with correspondingly lower and higher speedups. The accurate modeling of both computation and communication demonstrated in these case studies is important for RATSS as high or low data parallelism in an algorithm does not guarantee or exclude suitability for FPGA implementation, respectively. Strategic DSE encompasses not only significant design revisions but also maximizing available parallelism for the platform capabilities and application requirements.

5.1. Image Filtering

The particular image filter used in this case study is a discrete 2D convolution of a 3×3 image segment (i.e., a pixel and its 8 neighbors) with a user-specified filter. Example usages of this application include Sobel or Canny edge detection and high, low, or

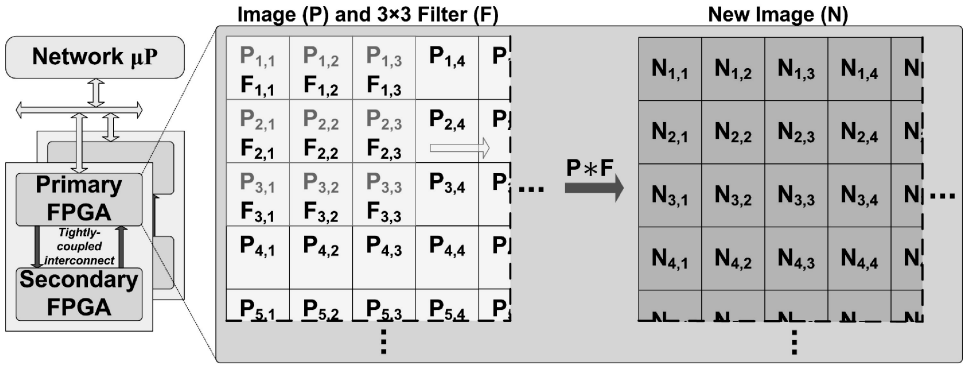


Fig. 8. Algorithm structure for image filtering case study.

Table VII. Node Attributes for Image Filtering

Attribute	*Units*	*Value*
PL_{comp}	(cycles)	0
R_{comp}	(operations/cycle)	34
F_{clock}	(MHz)	100
$N_{comp\_elements}$	(elements)	349,448
$N_{ops/element}$	(operations/element)	17

band-pass filtering for noise reduction. Figure 8 provides an illustration of this algorithm. The same 418 × 418 image is streamed (i.e., written) to the primary FPGA of two nodes of the SRC-6 system. As part of the computation, the primary FPGAs stream the image data to their respective secondary FPGAs. Each FPGA performs the convolution of the image data with respect to different filter values. The resulting images on the secondary FPGAs are streamed back to their respective primary FPGA which DMAs the two new images from the node back to the network-attached microprocessor. A more general overview of convolution for image filtering can be found in Gonzalez and Woods [2002].

Table VII summarizes the compute node attributes for the RATSS model. Because of the double-precision operations, the overall pipeline will be fairly deep and too complex to quickly and accurately determine a priori. However, the pipeline latency, RL_{comp}, should be negligible with respect to the volume of data. Both FPGAs contain a fully pipelined kernel for filtering that calculates the nine multiplications and eight additions for the convolution (i.e., the $9 + 8 = 17$ operations per pixel element) for a total computational throughput, R_{comp}, of 34 (2 FPGAs × 17 operations). The clock frequency, F_{clock}, for the MAP-B node is fixed at 100MHz. An image size of 418 × 418 pixels, limited by the size of the MAP-B SRAM, is used for this case study though larger sizes can be simulated by repeatedly looping through the memory. In contrast to the previous case study, each FPGA of each node needs the complete dataset (i.e., image) because each kernel convolves a different filter. Consequently, the effective number of data elements, $N_{elements}$, per node is 349,448 (418 pixels × 418 pixels × 2 FPGAs). Again, each element requires nine multiplications with the 3×3 filter and eight subsequent summations for a total of 17 operations per element, $N_{ops/element}$.

Table VIII defines the application-specific network attributes for the SNAP network model. Two nodes, P, with four total FPGAs are used for this case study. The pipelined (streaming) computation is structured using shift-registers and requires three new inputs each cycle (i.e., a pixel and its upper and lower neighbors). Consequently, the streaming communication of the 418×418 images requires a total of 4,193,376 bytes

Table VIII. Additional Network Attributes for Image Filtering

Attribute		*Units*	*Value*
P		(nodes)	2
k	broadcast	(Bytes)	4,193,376
	gather		2,795,584

Table IX. Modeling Error for Image Filtering (SRC-6, XC2V6000)

		Predicted (s)	Experimental (s)	Error
2 Nodes (4 FPGAs)	t_{comp}	5.24E–3	5.24E–3	–0.04%
	t_{comm}	1.40E–2	1.41E–2	–1.08%
	$t_{application}$	1.92E–2	1.98E–2	–3.13%
	Speedup	1.39	1.35	2.96%

(418×418×3×8B) for the broadcast message, k. Similarly, the output communication will involve two filtered images (one per FPGA) of 418×418 pixels each, for a total gather message size, k, of 2,795,584 bytes (418×418×2×8B).

Table IX highlights the sources of error for the image filtering case study. The computation time is comparable to the $O(N)$ write time. The deterministic structure of the computational pipelines allowed for highly accurate modeling of the total execution time for the nodes, t_{comp}, which is off by only 204 cycles, nearly the 127-cycle pipeline latency reported by the Carte tool during implementation. The network communication time, t_{comm}, comprised the majority of the total execution time, $t_{application}$, and the error remained just over 1%. Consequently, the total error remained just over 3%. The accuracy of the RATSS communication model for the SNAP interconnect ensured only a small error (under 3%) in the speedup prediction. The speedup value is relatively modest due to the lower computation-to-communication ratio as compared to the other case studies.

5.2. 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 MD simulation is 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 MD 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 Lab (ORNL).

Figure 9 provides an overview of the MD case study. In slight contrast to the image-filtering case study, four MAP-B nodes, one FPGA each, are used for MD. In order to compare two molecules every clock cycle, two copies of the molecular data are sent by the network-attached microprocessor to the primary FPGA of each node. (The secondary FPGA is not used for this case study). Each copy contains the X, Y, and Z dimensions of the molecular position data, requiring 2 SRAMs per copy for a total of 4 banks per node. Each MD kernel checks the distance of $N/4$ molecules against the other $N-1$ molecules, where $N/4$ is the number of molecules for each of the four nodes. If the molecules

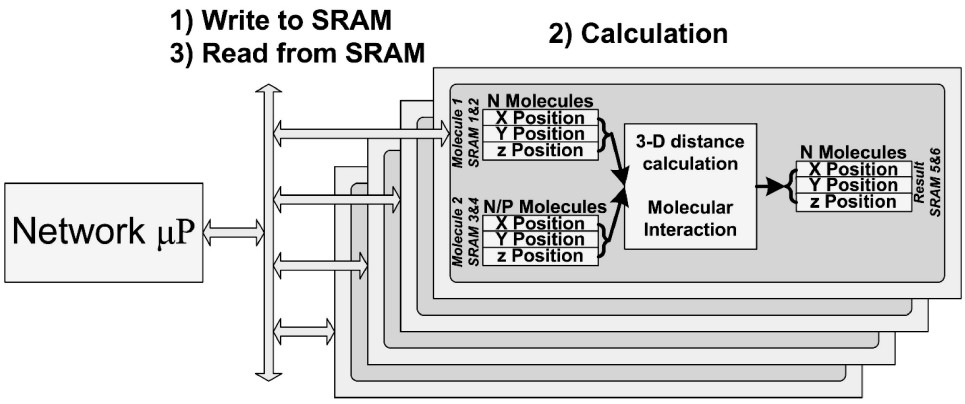


Fig. 9. Algorithm structure for molecular dynamics case study.

Table X. Node Attributes for Molecular Dynamics

Attribute	*Units*	*Value*
PL_{comp}	(cycles)	0
R_{comp}	(operations/cycle)	1
F_{clock}	(MHz)	100
$N_{comp\_elements}$	(elements)	8,192
$N_{ops/element}$	(operations/element)	32,767

Table XI. Additional Network Attributes for Molecular Dynamics

Attribute		*Units*	*Value*
P		(nodes)	4
k	scatter	(Bytes)	1,048,576
	gather		524,228

are sufficiently close, the MD kernel calculates the molecular forces (and subsequent acceleration) imparted on each other. The acceleration effects are accumulated in the last two SRAM banks and transferred back to the network-attached microprocessor.

The node-level attributes for the MD case study are defined in Table X. The pipeline latency, PL_{comp}, is considered negligible for this case study due to the $O(N^2)$ computational complexity. One pipeline per node allows for molecular iteration (i.e., operation) per cycle, $N_{ops/cycle}$. Again, the clock frequency, F_{clock}, for the MAP-B nodes is fixed at 100MHz. For this case study, the number of data elements (i.e., molecules) per node, $N_{elements}$, is 8,192 (32,768/4). Each molecule's interaction is computed against every other molecule for a total of 32,767 operations, $N_{ops/element}$.

Table XI defines the application-specific attributes for the SNAP network model. A total of four nodes, P, are used for the case study. The scatter message size is twice the gather message size due to two copies of input data required to compute a molecular interaction in a single cycle (i.e., two memory access per cycle). Also, the 4-byte, single-precision x, y, and z, dimensions of the molecule data are packed into two 8-byte words. Thus, the scatter and gather message sizes, k, are 1,048,576B (32,768×4×8B) and 524,228B (32,768×2×8B) respectively. Because of the single time step, only one system-level iteration, $N_{system\_iterations}$, is required for this case study.

Table XII compares the results of the RATSS model with the subsequent implementation of MD. Over 99% of the execution time is dominated by the FPGA computation, which is highly deterministic. The model error for the node-level time, t_{nodes}, is –0.0001%, an underestimation of 304 cycles, roughly the pipeline latency of

Table XII. Modeling Error for Molecular Dynamics (SRC-6, XC2V6000)

		Predicted (s)	Experimental (s)	Error
4 Nodes (4 FPGAs)	t_{comp}	2.68E+0	2.68E+0	–0.0001%
	t_{comm}	5.90E–3	4.84E-3	21.8%
	$t_{application}$	2.69E+0	2.69E+0	0.03%
	Speedup	5.12	5.12	–0.03%

271 cycles reported by the Carte tool during implementation. However, the total network time, $t_{network}$, is difficult to measure accurately due to the independently initiated communication for the four nodes, which resulted in a 22% error. However, the overall impact on prediction accuracy is negligible as the model discrepancy for the total application execution time, t_{total}, was overestimated by 0.03%. Consequently, the speedup versus the sequential software baseline was underestimated by 0.03%. This problem size was an advantageous case study due to the nontrivial hardware execution time (i.e., greater than 1s) yet highly tractable software baseline (i.e., less than 14s). Larger problem sizes would have slightly higher speedup due to higher FPGA computation-to-communication ratios and marginally slower software baselines due to cache misses.

6. CONCLUSIONS

Based on current parallel computing trends, scalable systems with FPGAs are increasingly desirable for their performance and power benefits. However, the associated cost of application development has inhibited greater adoption of FPGAs. Insufficient attention has been given to strategic planning for applications, particularly as applications scale. Simply providing faster implementation paths for FPGA devices only addresses one symptom of the productivity challenge. Effective DSE involves not only rapid design but also efficient performance evaluation. Analytical models can estimate the performance of application designers by distilling and evaluating key performance characteristics from the designer's specification. Such models prevent wasted implementation effort by identifying unrealizable designs and reducing the revisions necessary to achieve performance requirements.

RATSS provides an efficient and reasonably accurate analytical model for evaluating a scalable FPGA application prior to implementation. RATSS boosts designer productivity by extending concepts from component-level models to allow efficient abstraction and estimation of the computation and communication features of FPGA applications. The RATSS model contributes a multilevel approach for agglomerating component descriptions into a full performance estimate. RATSS performance prediction remains tractable by focusing on synchronous, iterative computation models for the two major classes of modern high-performance FPGA platforms.

The 2D PDF, image filter, and MD case studies illustrate performance modeling for a range of problem sizes and ratios of computation-to-communication. These case studies demonstrated nearly 90% prediction accuracy, which is considered sufficient given the focus of RATSS on strategic application planning. The accuracy of both the computation and communication models allows not only individual performance estimates but also accurate predictions across a range of potential application configurations including wide variations in problems sizes and computation-to-communication ratios. Specifically, important performance tradeoffs such as increasing parallelism or decreasing the communication rate can be efficiently evaluated with reasonable accuracy. These case studies serve as motivation for broad design-space exploration with RATSS as predictions are efficiently generated and reasonably accurate, which help ensure the eventual implementation is the most desirable design configuration. Future work includes pairing RATSS with a graphical design environment to provide an integrated, end-to-end framework for further efficiency improvements in application specification and

performance estimation with the overall goal of wider usage of the RATSS methodology by the reconfigurable computing community.

ACKNOWLEDGMENTS

Thanks go to Seth Koehler, Karthik Nagarajan, and Casey Reardon in our center for their contributions to the methodology and case studies.

REFERENCES

Agility Design Solutions. 2007. *Handel-C Language Reference Manual*. Agility Design Solutions, http://www.agilityds.com/literature/HandelC_Language_Reference_Manual.pdf.

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.

BALARIN, F., WATANABE, Y., HSIEH, H., LAVAGNO, L., PASSERONE, C., AND SANGIOVANNI-VINCENTELLI, A. 2003. Metropolis: An integrated electronic system design environment. *Comput. 36,* 4, 45–52.

BEDNARA, M. AND TEICH, J. 2001. Synthesis of FPGA implementations from loop algorithms. In *Proceedings of the 1st International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*. 1–7.

BHAT, P. B., PRASANNA, V. K., AND RAGHAVENDRA, C. S. 1999. Adaptive communication algorithms for distributed heterogeneous systems. *J. Parall. Distrib. Comput. 59,* 2, 252–279.

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 PASTOR, L. 2006. A parallel computation model for heterogenous clusters. *IEEE Trans. Paral. Distrib. Syst. 17,* 13.

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*. 403–410.

BUCK, J., HA, S., LEE, E. A., AND MESSERSCHMITT, D. G. 1994. Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. J. Comp. Simul. 4*, 152–184.

CAPPELLO, F., FRAIGNIAUD, P., MANS, B., AND ROSENBERG, A. L. 2001. HiHCoHP: Toward a realistic communication model for hierarchical hyperclusters of heterogeneous processors. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE Computer Society, Los Alamitos, CA, 42.

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

DEHON, A., ADAMS, J., DELORIMIER, M., KAPRE, N., MATSUDA, Y., NAEIMI, H., VANIER, M., AND WRIGHTON, M. 2004. Design patterns for reconfigurable computing. In *Proceeding of the 12th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

EL-GHAZAWI, T., EL-ARABY, E., HUANG, M., GAJ, K., KINDRATENKO, V., AND BUELL, D. 2008. The promise of high-performance reconfigurable computing. *Comput. 41,* 2, 69–76.

ENZLER, R., JEGER, T., COTTET, D., AND TRÖSTER, G. 2000. High-Level area and performance estimation of hardware building blocks on fpgas. In *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications*. Springer, 525–534.

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

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

FRANK, M. I., AGARWAL, A., AND VERNON, M. K. 1997. LoPC: Modeling contention in parallel algorithms. In *Proceedings of the 6th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP)*. ACM, 276–287.

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)*. 1–7.

GONZALEZ, R. C. AND WOODS, R. E. 2002. *Digital Image Processing,* 2nd ed. Prentice-Hall, Upper Saddle River, NJ.

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

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 Comput. 40,* 3, 50–57.

HOLLAND, B., NAGARAJAN, K., AND GEORGE, A. D. 2009. RAT: RC amenability test for rapid performance prediction. *ACM Trans. Reconfig. Tech. Syst. 1,* 4, 22:1–22:31.

JACOBS, A., CONGER, C., AND GEORGE, A. D. 2008. Multiparadigm space processing for hyperspectral imaging. In *Proceedings of the IEEE Aerospace Conference*.

KAUL, M., VEMURI, R., GOVINDARAJAN, S., AND OUAISS, I. 1999. An automated temporal partitioning and loop fission approach for FPGA based reconfigurable synthesis of DSP applications. In *Proceedings of the 36th ACM/IEEE Design Automation Conference (DAC)*. ACM, New York, 616–622.

KESAVAN, R., BONDALAPATI, K., PANDA, D., AND P, D. K. 1997. Multicast on irregular switch-based networks with wormhole routing. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. 48–57.

KIELMANN, T., BAL, H. E., AND GORLATCH, S. 1999. Bandwidth-Efficient collective communication for clustered wide area systems. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. 492–499.

KIELMANN, T., BAL, H. E., AND VERSTOEP, K. 2000. Fast measurement of LogP parameters for message passing platforms. In *Proceedings of the 15th IPDPS Workshop on Parallel and Distributed Processing*. 1176–1183.

LASTOVETSKY, A., MKWAWA, I.-H., AND O'FLYNN, M. 2006. An accurate communication model of a heterogenous cluster based on a switch-enabled ethernet network. In *Proceedings of the 12th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*.

MITRIONICS. 2008. Low power hybrid computing for efficient software acceleration. http://www.mitrion.com/?document=Hybrid-Computing-Whitepaper.pdf.

NAGARAJAN, K., HOLLAND, B., GEORGE, A., SLATTON, K. C., AND LAM, H. 2009. Accelerating machine-learning algorithms on FPGAs using pattern-based decomposition. *J. Sig. Process. Syst.*

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

PARZEN, E. 1962. On estimation of a probability density function and mode. *Ann. Math. Statist. 33,* 3, 1065–1076.

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.

PELLERIN, D. AND THIBAULT, S. 2005. *Practical FPGA Programming in C*. Prentice Hall Press.

PETERSON, G. D. AND CHAMBERLAIN, R. D. 1994. Beyond execution time: Expanding the use of performance models. *IEEE Concurr. 2*, 37–49.

PIMENTEL, A. D., HERTZBETGER, L. O., LIEVERSE, P., VAN DER WOLF, P., AND DEPRETTERE, E. F. 2001. Exploring embedded-systems architectures with artemis. *Comput. 34,* 11, 57–63.

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.

REARDON, C., GROBELNY, E., GEORGE, A., AND WANG, G. 2009. A simulation framework for rapid analysis of reconfigurable computing systems. *ACM Trans. Reconfig. Tech. Syst.* to appear.

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

SRC Computers 2007. *SRC Carte C Programming Environment*. SRC Computers.

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

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

WOLF, W. 2003. A decade of hardware/software codesign. *Comput. 36,* 4, 38–43.

Received March 2010; revised July 2010; accepted October 2010