12-2013

EXPLORING MULTIPLE LEVELS OF PERFORMANCE MODELING FOR HETEROGENEOUS SYSTEMS

Venkittaraman Vivek Pallipuram Krishnamani
Clemson University, kpallip@clemson.edu

EXPLORING MULTIPLE LEVELS OF PERFORMANCE MODELING FOR HETEROGENEOUS SYSTEMS

A Dissertation
Presented to
The Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Engineering

By
Venkittaraman Vivek Pallipuram Krishnamani
December 2013

Accepted by:
Dr. Melissa C. Smith, Committee Chair
Dr. Haiying (Helen) Shen
Dr. Walter Ligon III
Dr. Amy Apon

ABSTRACT

The current trend in High-Performance Computing (HPC) is to extract concurrency from clusters that include heterogeneous resources such as General Purpose Graphical Processing Units (GPGPUs) and Field Programmable Gate Array (FPGAs). Although these heterogeneous systems can provide substantial performance for massively parallel applications, much of the available computing resources are often under-utilized due to inefficient application mapping, load balancing, and tuning. While several performance prediction models exist to efficiently tune applications, they often require significant computing architecture knowledge for reliable prediction. In addition, they do not address multiple levels of design space abstraction and it is often difficult to choose a reliable prediction model for a given design.

In this research, we develop a multi-level suite of performance prediction models for heterogeneous systems that primarily targets *Synchronous Iterative Algorithms* (SIAs). The modeling suite aims to produce accurate and straightforward application runtime prediction prior to the actual large-scale implementation. This suite addresses two levels of system abstraction: 1) *low-level* where partial knowledge of the application implementation is present along with the system specifications and 2) *high-level* where the implementation details are minimum and only high-level computing system specifications are given. The performance prediction modeling suite is developed using our proposed *Synchronous Iterative GPGPU Execution* (SIGE) *model* for GPGPU clusters, motivated by the *RC Amenability Test for Scalable Systems* (RATSS) *model* for FPGA clusters.

The low-level abstraction for GPGPU clusters consists of a regression-based performance prediction framework that statistically abstracts system architecture characteristics, enabling performance prediction without detailed architecture knowledge. In this framework, the overall execution time of an application is predicted using regression models developed for host-device computations and network-level communications performed in the algorithm. We have used a family of Spiking Neural Network (SNN) models and an Anisotropic Diffusion Filter (ADF) algorithm as SIA case studies for verification of the regression-based framework and achieved over 90% prediction accuracy compared to the actual implementations for several GPGPU cluster configurations tested. The results establish the adequacy of the low-level abstraction model for advanced, fine-grained performance prediction and design space exploration (DSE). The high-level abstraction consists of the following two primary modeling approaches: *qualitative modeling* that uses existing *subjective-analytical models* for computation and communication; and *quantitative modeling* that predicts computation and communication performance by measuring hardware events associated with *objective-analytical models* using micro-benchmarks. The performance prediction provided by the high-level abstraction approaches, albeit coarse-grained, delivers useful insight into application performance on the chosen heterogeneous system. A blend of the two high-level modeling approaches, labeled as *hybrid modeling*, is explored for insightful preliminary performance prediction.

The performance prediction models in the multi-level suite are verified and compared for their accuracy and ease-of-use, allowing developers to choose a model that best

satisfies their design space abstraction. We also construct a roadmap that guides user from optimal Application-to-Accelerator (A2A) mapping to fine-grained performance prediction, thereby providing a hierarchical approach to optimal application porting on the target heterogeneous system. The end goal of this dissertation research is to offer the HPC community a thorough, non-architecture specific, performance prediction framework in the form of a hierarchical modeling suite that enables them to optimally utilize the heterogeneous resources.

DEDICATION

I dedicate this dissertation to my mother, father, academic advisor, and all teachers that shaped my life. I also dedicate this dissertation to my younger brother, cousins, close family members, and friends; they are the source of my happiness. Special dedications to three symbols in my life: Saraswati (Knowledge), Ganesh (Success), and Hanuman (Strength).

ACKNOWLEDGMENTS

Five years of graduate school at Clemson were the golden years of my life. I was able to advance myself intellectually, physically, and spiritually. I owe heartfelt thanks to my advisor Dr. Melissa C. Smith for making my graduate life an enriching experience; I could not have asked for a better mentor. She encouraged me at each turn with her wisdom and insightful suggestions that made research, writing, and teaching enjoyable. I sincerely hope to follow in her footsteps. I would like to thank my committee members: Dr. Helen Shen, Dr. Walter Ligon, and Dr. Amy Apon for reviewing my dissertation work. They taught me several aspects of parallel computing and provided me with necessary resources that contributed significantly to this dissertation research. I would also like to thank Dr. Smith, Dr. Ligon and Dr. Apon for providing me with several opportunities to represent Clemson at top-tier conferences including Super-Computing (SC), which contributed significantly to my professional development.

Many thanks to the National Science Foundation (NSF) for their support with research grant (NSF Career Award #1149644), resources provided by XSEDE, Clemson University CITI, and NSF MRI Grant #1228312 that made this dissertation possible.

Special thanks to my professional family at Future Computing Technologies (FCTLab) group here at Clemson University for all the jovial and fun-filled discussions.

Finally, I would like to thank my parents for their unwavering support and teaching me valuable life lessons. My friends Abhishek, Koushik, Chinmay, Sudershan, Jyoti, Vijay, CM, Raj, and many others that I undoubtedly missed, I can’t thank enough for having you in my life. Go Tigers!

TABLE OF CONTENTS

Page

Table of Contents (Continued) Page

Table of Contents (Continued) Page

LIST OF TABLES

List of Tables (Continued)

Table Page

List of Tables (Continued)

List of Tables (Continued)

Table Page

List of Tables (Continued)

Table Page

List of Tables (Continued)

Table Page

LIST OF FIGURES

List of Figures (Continued)

Figure Page

List of Figures (Continued)

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

There is widespread speculation that the principles of the Moore's law for increasing the single-core processor performance will no longer hold [1]. Because of power and memory clock limitations, the industrial trend has shifted to multi-core and many-core processors. Many vendors including IBM, AMD, and Intel are demonstrating many-core processor prototypes that can theoretically achieve performance over 1 Teraflops. Intel's Many Integrated Core (MIC) architecture is one such initiative that claims to surpass the Exascale performance barrier using a combination of several MICs [2]. However, amongst these advancements, hybrid accelerators such as the General Purpose Graphical Processing Units (GPGPUs) and Field Programmable Gate Arrays (FPGAs) continue to remain effective and popular in the High-Performance Computing (HPC) community. These architectures have been reported to provide several orders of magnitude higher performance compared to traditional sequential processors. Furthermore, the aforementioned architectures provide high floating-point operations per second per watt (FLOPS/watt) performance, an increasingly important parameter in *green super-computing* [3]. With the advent of GPGPUs and FPGAs in HPC, the conventional methods of seeking concurrency in a homogeneous environment no longer apply. The current trend is to extract concurrency from heterogeneous clusters that include GPGPU and FPGA clusters [4 and 5]. Current state-of-the-art heterogeneous systems are composed of several thousand compute nodes

where each node consists of multiple CPU-cores in conjunction with one or more hybrid accelerators.

Although these heterogeneous systems can provide substantial performance for massively parallel applications, much of their computing resources are often under-utilized due to inefficient application mapping, load-balancing, and tuning, ultimately leading to poor application speed-up and sub-optimal scaling efficiency. This inefficiency further leads to secondary effects such as long job queue delays and increased power consumption [6]. To achieve optimal utilization of heterogeneous resources, it is important to perform efficient load-balancing between the CPU-cores and accelerators. Several performance prediction models exist that enable developers to efficiently tune applications via design space exploration [6, 7, and 8]. Typically, the performance prediction models are used to predict application runtime prior to the actual execution, allowing developers to further fine-tune their applications. Although existing performance prediction models are sufficiently accurate, they *do not address* multiple levels of design space abstraction and it is often *difficult* to choose a *reliable* prediction model for the *given design goals*. Additionally, the existing performance prediction models often require intricate knowledge of the underlying computing architecture for accurate prediction, making the modeling task difficult. With the above as motivation, we formally introduce the problem statement:

> *Design a straightforward and accurate performance prediction framework for heterogeneous clusters that addresses multiple levels of design space abstraction, allowing developers to choose an effective performance model that best fits their design needs and goals.*

1.2 DISSERTATION RESEARCH

We develop a multi-level suite of performance prediction models for heterogeneous systems that primarily targets *Synchronous Iterative Algorithms* (SIAs). The modeling suite aims to accurately predict application runtime with a user-friendly approach prior to actual large-scale implementation. The application runtime prediction is also employed to perform Design Space Exploration (DSE) that enables researchers to ultimately map an optimal implementation to the target heterogeneous cluster, thereby facilitating high application performance. The modeling suite addresses two levels of system abstraction: 1) *low-level* where partial knowledge of the implementation is present along with the target system specifications and 2) *high-level* where the implementation details are minimum and only high-level computing system specifications are given. The multi-level performance modeling suite is developed using our proposed *Synchronous Iterative GPGPU Execution* (SIGE) *model* for GPGPU clusters, motivated by the existing *RC Amenability Test for Scalable Systems* (RATSS) *model* [9] for FPGA clusters. These execution models describe the execution flow of SIAs on GPGPU and FPGA clusters, respectively.

The low-level abstraction of the modeling suite consists of a regression-based performance prediction framework that statistically abstracts the system architecture characteristics, thereby enabling performance and scalability prediction without detailed system architecture knowledge. The regression-based framework is broken into two primary components: the *computation component* that models the hybrid accelerator and host computations; and the *communication component* that models the network-level communications. The regression models for the computation component use algorithm characteristics such as the number of floating-point operations (FLOPs) performed and total number of bytes required as predictor variables. It is worth mentioning that FLOPS and FLOPs are two distinct parameters; FLOPS (floating-point

operations per second) is a measure of computer performance, whereas FLOPs is the number of floating-point operations performed in an algorithm. The regression models are trained using several small instrumented executions of an SIA set with a range of communication-to-computation requirements. The communication component of the regression-based framework is broken into two sub-components: 1) inter-processor communication over Infiniband [10] and 2) CPU-host/GPGPU-device (host-device) communication over Peripheral Interconnect Express (PCI-Ex) bus [11]. The regression models for the communication component are developed using micro-benchmarks and employ data transfer size and processor count as predictor variables.

The high-level abstraction of the modeling suite relies on minimum implementation details and high-level system specifications to model the computations and communications. The high-level abstraction consists of the following two primary modeling approaches: *Qualitative Modeling* and *Quantitative Modeling*. The qualitative modeling uses *subjective-analytical models* for the computation and communication components. The quantitative modeling approach predicts computation and communication performance by measuring hardware events associated with *objective-analytical models* using micro-benchmarks. The measurement of hardware events such as arithmetic operation throughput, device memory bandwidth, latency and bandwidth of the network (and interconnects), etc. in conjunction with algorithm characteristics enables the developer to estimate the application execution time. The qualitative and quantitative approaches are combined to yield an intermediate *hybrid* approach where a few performance components are estimated analytically, while the remaining components are estimated by employing micro-benchmarks. In this dissertation research, we show that amongst the high-level

abstraction approaches, the *hybrid* approach is a viable paradigm to perform high quality performance prediction on the chosen computing platform.

The two levels of the modeling suite are verified with large-scale SNN simulations and a non-linear anisotropic diffusion filter (ADF) algorithm for massive images as SIA case studies. We implemented both applications on the National Center for Supercomputing Applications (NCSA) Forge GPGPU cluster [12] and achieved significantly high performance versus the Message Passing Interface (MPI)-only implementations. The multi-GPGPU based large-scale SNN simulations scale up to 200 million neurons using a 32-node cluster configuration and achieves speed-up as high as 253x compared to an equivalent MPI-only implementation [13]. The multi-GPGPU implementation of the ADF is capable of processing images as large as 156 mega-pixels and achieves 11.5x speed-up using a 32-node GPGPU cluster configuration when compared to an equivalent MPI-only implementation [14].

The multi-level performance prediction models are compared for their accuracy and ease-of-use, thereby providing *model selection criteria* that allow developers to choose a prediction model that best satisfies their design space abstraction. The verification of the low-level abstraction reports average prediction accuracy over 90% compared to the actual implementations for several tested GPGPU cluster configurations, making it practicable for advanced, fine-grained performance prediction and design space exploration. Predictions with the two high-level abstraction approaches were found to be coarse-grained; however the *hybrid approach*, a suitable combination of these two modeling strategies, is an efficacious paradigm that provides significant insight into application performance, ergo highly suitable for preliminary performance prediction on the chosen or potential heterogeneous systems.

The dissertation research also provides a roadmap for users to perform optimal Application-to-Accelerator (A2A) mapping by means of appropriate architecture identification and performance prediction (preliminary and advanced). In this roadmap, the first milestone is A2A mapping that identifies an optimal accelerator for the application. The next milestone is preliminary performance prediction, facilitated by the high-level abstraction approach, to obtain an insight into application performance on the selected accelerator platform. This task also enables the identification of plausible optimization techniques for high application performance. The last milestone is constituted by the low-level abstraction that determines the best implementation for the target system via DSE. The A2A roadmap facilitates a hierarchical approach to optimal application porting on the heterogeneous system. It is worth mentioning that we follow a bottom-up approach to construct the performance modeling suite (low-level abstraction to high-level abstraction). However, the A2A roadmap seeks a top-down approach (high-level abstraction to low-level abstraction) for application performance prediction that is most useful for developers.

The end goal of this dissertation research is to offer the HPC community a thorough performance prediction framework in the form of a hierarchical modeling suite that enables them to optimally utilize the heterogeneous resources without requiring intricate knowledge of the low level architectures or restricting the specific architectures or accelerators used. The outcomes and contributions of this doctoral dissertation research are summarized below.

1) Development of synchronous iterative execution model (SIGE) for GPGPU clusters.
2) Development of a multi-level performance modeling suite for heterogeneous systems encompassing multiple levels of system abstraction.

3) Verification of the modeling suite using Synchronous Iterative Algorithms (SIAs) with a range of computation-to-communication requirements.
4) Application of the low-level abstraction for Design Space Exploration (DSE).
5) Performance analysis of SIAs on the chosen heterogeneous systems (to confirm the implementations achieve sufficient efficiency and scaling).
6) Tying-it-all-Together: A roadmap for users to perform optimal A2A mapping.

1.3 METHOD OF STUDY

The set of highly biologically accurate SNN models and ADF algorithm, both SIAs, offer a range of communication and computation requirements, making them valuable case studies to verify the hierarchical performance model for this algorithm domain; these algorithms are used to perform large-scale SNN and image filtering simulations, respectively.

The planned experiments are conducted on available heterogeneous clusters by varying the problem size (neural network size, image size, etc.) and scaling the number of nodes in the cluster. The heterogeneous resources include NCSA Forge GPGPU cluster [12] and GPGPU-augmented Palmetto cluster [15]. In addition to verifying the performance prediction models for accurate runtime prediction, performance and scalability studies are also conducted on the NCSA Forge cluster to confirm the implementations achieve sufficient efficiency and scaling. Initial verification of the regression-based framework (low-level abstraction) for GPGPU clusters using the SNN models and ADF algorithm [16] is completed on the NCSA Forge GPGPU cluster. The GPGPU DSE using low-level abstraction and high-level abstraction studies are performed on the GPGPU-augmented Palmetto cluster.

1.4 DISSERTATION OUTLINE

Chapter 2 provides a literature review of important work done in the field of heterogeneous performance modeling. Following the literature survey, Chapter 3 provides the background on the base GPGPU architectures and the SNN and ADF algorithms. Chapter 4 details the experimental set-up, SIA mapping methodology and multi-node orchestration. This chapter also provides the performance analysis study of SNN-ADF SIAs on the NCSA Forge cluster. The development of SIGE model and multi-level performance prediction suite is explained in Chapter 5. The low-level abstraction approach is elaborated in Chapter 6 followed by the verification results provided in Chapter 7. The high-level abstraction approach is elucidated in Chapter 8 and verified in Chapter 9. The dissertation is concluded in Chapter 10 with conclusions and directions for future research. Appendix A provides the A2A roadmap.

CHAPTER 2

LITERATURE REVIEW

Systematic architecture studies conducted on heterogeneous systems including GPGPU- and FPGA-based clusters are widely documented in the literature. Several research activities have focused on important performance modeling aspects that include runtime prediction, architecture parameter study, load-balancing, programming models for HPC, and network-level modeling; making them relevant to this dissertation research. The two SIA case studies discussed, namely Spiking Neural Networks (SNNs) and Anisotropic Diffusion Filtering (ADF) have been implemented on several leading architectures. In this chapter, we examine some of the prominent heterogeneous performance modeling efforts targeting GPGPU- and FPGA-based systems and several architecture studies using SNNs and ADF. The chapter is structured as follows. Section 2.1 examines performance modeling studies conducted on GPGPU-based systems, the primary heterogeneous platform investigated in this research. We also review load-balancing studies, performance tuning for applications, and programming models for GPGPU architectures. The discussion of performance models for FPGA-based systems, influential in this research, follows in Section 2.2. Section 2.3 reviews some of the important network-level modeling research. Section 2.4 highlights the architecture studies conducted using SNNs and ADF. The chapter is concluded with a summary in Section 2.5.

2.1 PERFORMANCE MODELING: GPGPU-BASED SYSTEMS

In [17], the authors proposed an analytical model that estimates the execution time of GPGPU kernels for massively parallel applications by estimating the number of memory requests (memory-warp parallelism) and the number of computations (computation-warp parallelism). Based on these warp-level parallelisms, the analytical model estimates the costs of memory requests and computations, thereby estimating the overall execution time of the application. The authors achieved geometric mean error rate of 5.4% for micro-benchmarks and 13.3% for other GPGPU applications. Although sufficiently accurate, the model proposed in [17] requires meticulous evaluation of the warp-level parallelism for accurate runtime prediction. Additionally, their analytical model is tightly-coupled to the Nvidia Tesla architecture used in the GeForce-8 series, which is significantly different from subsequent GPGPU architectures.

In [7], the authors designed an analytical model to provide performance information to an auto-tuning compiler, thereby assisting the fine-tuning of GPGPU implementations. The analytical model interprets the GPGPU kernel as an abstract work-flow graph to estimate the execution time. The authors used micro-benchmarks to characterize GPGPU micro-architecture events such as incoherent memory accesses, shared memory bank conflicts, and control flow divergence. The authors validated their model using commonly used benchmarks and observed good agreement between the predicted and observed measurements. Similar to the research work presented in [17], the model requires significant GPGPU micro-architecture knowledge for accurate runtime prediction for complex applications.

In [18], the authors proposed a performance model for the Nvidia GeForce 200-series GPGPUs using micro-benchmarks. The proposed model targets three major components of the GPGPU execution time: instruction pipeline, shared memory accesses, and global memory

accesses. Using real-world matrix problems, the authors achieved prediction performance with 5-15% error rate. While the approach is expected to satisfactorily predict the aforementioned architecture components, quantitative modeling of other micro-architecture events such as thread block synchronization may not be trivial. A similar quantitative approach is presented in [19] where the authors developed a micro-benchmark suite that measures CUDA-visible architectural characteristics of the Nvidia GTX 280. The suite also measures several undisclosed architectural features that impact program performance and correctness. Although the proposed suite is very thorough with respect to the Nvidia GTX 280 architecture, continual revision of the micro-benchmark suite is required to accommodate new architectural features as the GPGPU architecture evolves.

In [8], the authors developed a methodology to predict the execution time of GPGPU applications using runtime information from a single GPGPU implementation while varying the number and configuration of GPGPU devices. The authors define *per-element average* as the average time taken by the reference GPGPU device to execute a single computational entity in a given algorithm. The authors then use the per-element average information to extrapolate the algorithm execution time on M GPGPU devices, where M is the number of devices. The authors estimate the performance of the PCI-Ex bus and network-level transactions using micro-benchmarked throughput values and peak theoretical network bandwidth, respectively. The authors used their prediction framework on six applications and achieved 11% average error rate. Although straightforward, this approach to predicting the GPGPU execution time lacks statistical rigor. Several algorithm parameters, including but not limited to floating-point operations (FLOPs) and computational bytes, affect the GPGPU execution time. Therefore, it is extremely important to characterize the relationship between GPGPU execution time and algorithm

parameters. A similar argument can be used for predicting the performance of the PCI-Ex bus and network-level transactions, where the data transaction behavior can be characterized using statistical analysis.

Regression-based methods have been previously investigated for GPGPU design space exploration. In [20], the authors proposed an automated tool developed using step-wise regression modeling to evaluate the GPGPU performance. The tool randomly samples parameter values from the GPGPU design space and simulates regression designs. The tool then selects the most significant architecture parameters and their interactions to construct an estimator. The authors reported less than 1.1% error rate for 11 GPGPU applications. Unlike the statistical approach described in [20], the low-level abstraction of our proposed multi-level performance modeling suite relies on easily accessible algorithm parameters such as FLOPs and computational bytes for runtime prediction, thereby statistically capturing the architectural behavior.

In [21], the authors developed an analytical tool called TEG (Timing Estimation tool for GPU) to estimate the GPGPU device performance. The inputs to TEG are constituted by kernel binary code and instruction trace obtained using *cuobjdump* [22] and Barra simulator [23], respectively. TEG analyzes the binary code and instruction trace to generate information regarding the type of instructions and operands used in the GPGPU kernel. The analytical tool then uses instruction latency information obtained from micro-benchmarks [19] to evaluate the total number of execution cycles. The authors used dense matrix multiplication as a case study and achieved less than 10% error rate in execution cycle prediction. The authors admit that TEG does not model other important parameters such as instruction pipeline stages and memory

behavior. Additionally, their modeling methodology only supports a specific Nvidia GPGPU device.

Similar to the work described in [21], Parallel Thread eXecution (PTX) kernels [24] have been analyzed to solidify the understanding of GPGPU architectures. As mentioned in [24], PTX defines a virtual machine and instruction set architecture (ISA) for parallel thread execution on GPGPU devices. In [25], the authors proposed a set of metrics for GPGPU workloads to analyze the behavior of GPGPU programs. The authors analyzed over 50 CUDA kernels from Nvidia CUDA SDK [26] and UIUC's Parboil benchmark suite [27]. The analysis was conducted to study control flow, data flow, and memory behavior of CUDA programs using a PTX functional emulator developed by the same authors. The authors also used the PTX functional emulator to quantify the effects of common CUDA optimizations such as branch divergence reduction, synchronization, etc. However, as mentioned in [21], direct PTX analysis is not always desirable since resource allocations occur at the compiling stage from PTX to binary code. In [21], the authors claim that since binary code is the native code that executes on the GPGPU device, this level of analysis is more suitable for performance modeling and related studies.

In [28], the authors proposed a performance prediction model for GPGPU-based systems that incorporates various components of the GPGPU architecture including warp scheduling, memory hierarchy, and pipelining. The model is developed with a combination of the BSP model of Valiant [29], the PRAM model of Fortune and Wyllie [30], and the extension to the PRAM model proposed by Gibbons et al. called the QRQW model [31]. The proposed model derives a relationship among the various components of the GPGPU architecture including the number of cores, effects of memory latency, memory access conflicts, computing cost, scheduling, and pipelining to analyze pseudo-code for a CUDA kernel and finally predicts the performance of an

application. Unlike the regression-based framework developed in this research, the model in [28] does not consider the performance of texture memory along with global and shared memories, thereby providing limited insight into the GPGPU design space exploration (DSE).

In [32], the *GPGPURoofline* model was proposed to empirically guide the optimizations on GPGPU devices with limited knowledge of the GPGPU architecture. The model explores the potential performance bottlenecks and evaluates the impact of specific optimization techniques on the overall kernel performance. The authors optimized representative applications, namely matrix transpose, Laplace transform, and face detection on NVIDIA and AMD GPGPU devices and achieved 3.74 to 14.8 times speed-up compared to the naïve implementations. The modeling approach, similar to the popular Roofline model by Williams et al. [33] for multi-core architectures, is primarily intended to evaluate the GPGPU performance optimizations. Unlike the low-level abstraction methods developed in this dissertation research, the performance prediction facilitated by the *GPURoofline* model is expected to be coarse-grained, hence of limited value for accurate runtime and scalability predictions.

In [34], the authors introduced a metric that accurately estimates the effect of control flow divergence on application performance. The metric targets computation-bound GPGPU kernels with control flow divergence and is used as a value function for thread re-grouping algorithms to eliminate the divergence. The authors claim that their metric enables performance modeling more efficiently versus the previous control flow divergence metrics such as divergent warps and divergent branches. The authors tested the proposed metric on CUDA SDK examples [26] and two real-world applications including 3D sound localization [35] and stereo-matching [36]. The authors reported application performance improvement up to 3.19x using thread re-grouping [37]

guided by the proposed metric. Similar to the study presented in [32], the research in [34] solely aims at guiding users to perform kernel optimizations that improve the overall performance.

In [38], the authors presented an approach to analytical modeling by constructing a domain specific language (DSL) called *Aspen*. *Aspen* includes a formal specification of an application's performance behavior and an abstract machine model. The DSL allows scientists to write structured models of their applications and architecture, thereby describing the application behavior and abstract machine model. The authors demonstrated the use of *Aspen* to express a performance model for 3D Fast Fourier Transform (FFT), in addition, showed how *Aspen* allows model composition by incorporating 3D FFT model for use in molecular dynamics. Although an efficient tool for quick performance estimation, the proposed DSL is based on analytical models that often provide coarse-grained predictions.

Recently, application specific performance models have been proposed to predict the application execution time on GPGPU devices. In [39], the authors proposed an integrated analytical and profile-based performance model to predict the CUDA kernel execution time for Sparse Matrix Vector Multiplication (SpMV). The modeling approach involves two phases. In the first phase, benchmark matrices are generated based on the GPGPU architectural features. These benchmark matrices are then executed on the target GPGPU device to obtain the execution time. In the second phase, the authors derive an analytical model that establishes a relationship between the maximum number of rows that the target GPGPU device can execute at a time, the number of non-zero elements per row in the target matrix, and execution times of the benchmark matrices. Although the authors report less than 10% error rate for 32 test cases, the prediction approach is tightly coupled to the SpMV application and must be revised as the GPGPU architecture changes.

The literature also reports multi-GPGPU studies that assist in the characterization and performance modeling of GPGPU clusters. In [40], the authors studied Non-Uniform Memory Access (NUMA) contention effects for shared system resources, quantified the contention effects, and presented guidelines to maximize the performance. The authors conducted their tests using the Scalable Heterogeneous Computing (SHOC) benchmark suite [41] and High-Performance Linpack (HPL) [42] and concluded that significant NUMA contention effects prevail in dual-IO hub multi-GPGPU systems. The authors claim that the severity of the contention penalty depends on several factors such as computational density, number of kernel executions per PCI-Ex transfer, and the fraction of the application ported to the GPGPU devices. The authors suggest that sharing GPGPU devices among a small number of MPI tasks or threads can increase GPGPU device utilization. The authors also suggest splitting MPI communication and GPGPU traffic into different threads to alleviate the contention penalties and promote maximum GPGPU bandwidth. The SHOC benchmark [41] proposed by the same authors is a valuable tool to measure throughput values for several device related operations such as arithmetic computations, host-device transfers, and hierarchical memory transactions (global, shared, texture, and constant). The throughput values of the above mentioned parameters in turn assist with the quantitative analysis of GPGPU performance.

In addition to performance modeling and GPGPU architecture studies, several research activities have focused on load-balancing issues for GPGPU systems. These studies are interesting since optimal performance is achieved only with efficient application tuning that further aides in consolidated performance analysis. In [43], the authors presented a task-based dynamic load-balancing solution in the form of a task queue scheme for single- and multi-GPGPU systems. The authors assert that their scheme provides a load-balancing solution at a

finer granularity compared to the Nvidia CUDA SDK [26]. The authors verified their scheme using micro-benchmarks and a molecular dynamics application and achieved significant performance improvement over other implementations. In [44], the authors proposed a technique that distributes iso-surfacing load (used for scientific visualization) to GPGPU devices in a cluster. The load-balanced implementation by the same authors is reported to exhibit strong scalability and yield performance as high as 250 million triangles per second on 24 GPGPUs. In [45], the authors studied different load-balancing schemes including: static task list, blocking dynamic task queue, lock-free dynamic task queue, and task stealing to improve the performance of GPGPU quicksort algorithm. The authors concluded that lock-free methods achieve better scaling and higher performance over blocking methods for the quicksort algorithm on GPGPUs.

The programming models for GPGPU devices, such as the Compute Unified Device Architecture (CUDA) [22] and Open Computing Language (OpenCL) [46], are integral for high application performance. Although programming models are not commonly incorporated into performance modeling, their study provides useful insight into application-on-accelerator behavior. In what follows, we mention important programming model comparison studies and recent programming paradigms developed for GPGPU devices. In [47], the authors accelerated an EMRI modeling application using Nvidia's C1060 as one of the accelerators and achieved similar performance for both CUDA and OpenCL. In [48], the authors used the Adiabatic Quantum Algorithms (AQUA), which are Monte Carlo simulations, to compare CUDA and OpenCL on Nvidia's GTX-260 (Compute capability 1.3). They compared the programming models for data transfer time, kernel execution time and end-to-end runtime. They concluded that CUDA implementations perform consistently better than the OpenCL implementations. In [49], the authors studied the performance portability of OpenCL and concluded that the

performance is not portable. They implemented TRSM and GEMM (both SGEMM and DGEMM) from the BLAS library [50] for their studies on both Nvidia Fermi [51] and AMD Radeon [52] architectures. Based on the above literature review for CUDA and OpenCL programming paradigms, we conclude that CUDA programming model is an optimal choice for high application performance on Nvidia GPGPU architectures.

Recently, directive-based programming models have emerged that provide different levels of abstraction and require different levels of programming effort to port and optimize applications on GPGPU devices. The examples of directive-based programming models include Hicuda [53], OpenMPC [54], PGI Accelerator [55], and OpenACC [56]. In [57], the authors evaluated these directive-based programming models by porting thirteen application kernels from various scientific fields on CUDA GPGPU devices. Their evaluation reported that the directive-based models can achieve reasonable performance versus the traditional hand-written GPGPU kernel codes. They also concluded that the high-level abstraction provided by the directive-based programming models will better assist in code portability for future architectures that combine GPUs and CPUs onto the same die [58].

In this sub-section, some of the prominent GPGPU performance modeling and architecture studies documented in the literature were discussed. Although the performance modeling schemes discussed are sufficiently accurate, they present a number of shortcomings. Both the analytical and quantitative models discussed require intricate GPGPU architecture knowledge for viable performance prediction. The accuracy of qualitative models is highly sensitive to the precise evaluation of model parameters. The quantitative models are prone to miss non-measurable architecture parameters, leading to imprecise predictions. Additionally, the quantitative approach is often tightly coupled to a specific GPGPU architecture, rendering them

invalid/incomplete for future generations. Therefore, it is not always a clear choice for developers to select a reliable prediction model for a given application. Unlike the performance modeling approaches discussed in this sub-section, our research aims to provide a user-friendly performance prediction framework that addresses multiple levels of design space abstraction, thereby allowing developers to choose the best model for the given design goals and the level of knowledge regarding the algorithm and architecture(s).

2.2 PERFORMANCE MODELING: FPGA-BASED SYSTEMS

Several research activities have focused on performance modeling of High-Performance Reconfigurable Computing (HPRC) systems. Although our research does not include FPGA-based systems, we mention relevant HPRC modeling studies that inspired the research work in this dissertation. In [59], the author proposed a model for shared resource load imbalance, dedicated resource imbalance, and communications in distributed applications utilizing shared resources. The author validated the model using four implementations: Boolean Satisfiability, Matrix-Vector Multiplication, Encryption, and CHAMPION demo algorithms. In [60], the authors proposed the RC Amenability Test (RAT) model that provides a framework to predict speed-up of applications on single-node FPGA-based systems. In [9], the authors extended the RAT model for multi-node FPGA systems. The RATSS (RC Amenability Test for Scalable Systems) model proposed in [9] predicts the application runtime by separately modeling the node computations using the RAT model and inter-node communications using LogGP model [61]. The authors validated the RATSS model using 2D Probability Density Function (PDF) estimation and image processing algorithms. The research presented in this dissertation is motivated by the multi-FPGA-based system modeling studies presented in [9].

2.3 NETWORK-LEVEL MODELING

In addition to performance analysis of node-level computations (device kernels, host computations, and host-device transactions), it is imperative to perform efficient analysis of the network-level transactions to accurately predict the application runtime on heterogeneous clusters. In this sub-section, we discuss some of the important network-level modeling techniques documented in the literature. In [62], the authors proposed the *logP* model that attempts to capture important bottlenecks in parallel computing with a limited number of parameters that include latency, overhead, bandwidth of communication, and the number of processors. The authors claim that the *logP* model can sufficiently describe the performance characteristics of several parallel machines. An extension to the *logP* model, *parameterized logP* (*plogP*), provided in [63], incorporates the message size for measurements. The *plogP* model defines five parameters, namely the number of processors, end-to-end latency, sender overhead, receiver overhead, and bandwidth for a given message size. Although *logP* and *plogP* models are state-of-the-art parallel machine models, the *logGP* model [61] is currently the most popular and widely used parallel machine model. The *logGP* model adds the gap term, *G* for long messages to the *logP* model. The experimental data collected by the authors in [61] shows that the *logGP* model can accurately predict the communication performance for both long and short messages. In [64], the authors derived a new *logGP* parameter assessment technique, *netgauge* that does not saturate the network for measurements. The authors also proposed a methodology to detect network protocol changes in the underlying communication system.

While the *logP*, *PlogP*, and *logGP* models constitute the foundation of any network-level performance analysis, several other derivatives of the *logP* model exist that explain the secondary network characteristics. The *logGPG* model [65] adds a network contention parameter

to the traditional *logGP* model. The *logGPS* model [66] captures the synchronization needed prior to sending long messages by high-level communication libraries. As mentioned in [66], the *logGPS* model adds the parameter *S* that defines the threshold for message length above which the synchronous messages are sent. In [67], the authors developed the *logfP* model that characterizes the small message performance over Infiniband. The *logfP* model adds the parameter *f* to the *logP* model, which indicates the number of messages where a small message gap has not been accounted.

In our research, we develop a variant of the above mentioned performance models for network communication, specifically in the high-level abstraction of the modeling suite. Although the above mentioned models adequately describe the network characteristics, communication transactions in heterogeneous systems often exhibit randomness in their behavior as explored ahead in Chapter 7. Therefore, regression analysis of communications (both PCI-Ex and network-level) enables us to capture the data transaction behavior statistically, thereby abstracting high-level architecture details. Regression-based techniques for modeling the communications using Michaelis-Menten kinetics [68] are expounded in Chapter 7.

2.4 SNNs and ADF

2.4.1 SNNs

Spiking Neural Networks (SNNs) are very popular in the neuroscience community for modeling the mammalian brain to understand its functional and operational principles. The ability of spiking neurons to reproduce most of the neuronal properties with high accuracy makes them amenable for brain related studies [69]. Biologically inspired SNNs are popular in other fields such as pattern recognition [70], artificial intelligence [71], and smart control of power

grids [72]. In this section, we discuss some of the prominent architecture studies conducted using large-scale SNN simulations.

In [73], the authors studied the mammalian brain neo-cortex and simulated a rat-size cortex in 42% of real-time and a cat-size cortex in 23% of real-time on a 442-node Dell Xeon cluster. In [74], the authors successfully utilized the Izhikevich SNN model to simulate a cat-size cortical model with 10^9 neurons and 10^{13} synapses using the BlueGene/P machine [75] with 147,456 processors and 144 TB of main memory. The authors claim their simulation scale is roughly 1-2 orders of magnitude smaller than the human cortex and 2-3 orders of magnitude slower than real-time.

Heterogeneous architectures such as GPGPUs are now being investigated for biologically realistic simulations. In [76], the authors implemented Izhikevich's random network on Nvidia's GTX-280 with 1 GB memory and achieved a speed-up for a 100K neuron network simulation. They also discussed mapping strategies on the GPGPU to efficiently utilize the memory bandwidth and parallelism. In [77], the authors investigated GPGPU cluster-based implementations of the Hodgkin-Huxley (HH) and Izhikevich SNN models using a two-level character recognition network. They reported GPGPU speed-ups of 24.6x and 177x for the Izhikevich and HH models, respectively. Their 16 GPGPU-based MPI implementation on a 32-node Tesla S1070 NCSA cluster was successful in scaling the network up to 150 million neurons and achieved 17910 millisecond runtime for the HH model.

2.4.2 ADF

The non-linear anisotropic diffusion filter (ADF) investigated in this research belongs to the class of stencil-based algorithms for image processing. Several research activities have been motivated by the cluster and grid computing paradigms for stencil-based image processing

applications. In [78], the authors implemented an anisotropic diffusion filter for parallel and distributed systems. Their implementation was parallelized with point-to-point and collective communications using LAM-MPI [79] on a heterogeneous cluster of workstations. The anisotropic filtering technique adopted by the authors used 30 iterations and a neighborhood factor of 15. Their point-to-point and collective communication implementations achieved performance gains of 81.9% and 93.8%, respectively, when compared to the execution time on a single computing node. The authors observed that their collective communication implementation was 21% more efficient when compared to the point-to-point communication implementation.

The CUDA and hybrid CUDA/MPI paradigms have recently gained interest for stencil-based image processing applications. In [80], the authors proposed a new method to remove Rician noise from magnetic resonance images using GPGPU devices. The authors designed an anisotropic diffusion filter that characterizes the direction of diffusion and pixel properties using Eigen-values and Eigen-vectors. To preserve the edges, the authors coupled the proposed anisotropic diffusion filter with a shock filter based on fuzzy sets. The authors compared their filter implementation with the traditional anisotropic diffusion filter and wavelet based methods and reported an average gain of 0.01 dB in PSNR values. Additionally, their GPGPU implementation (kernel computation only) performed approximately 9 times faster than the CPU-only implementation.

In [81], the authors implemented the gradient domain processing technique for massive images using MPI, threading, and a GPGPU-based component. The authors successfully stitched giga-pixel size panoramas and demonstrated performance and scalability on two GPGPU

clusters. The authors achieved over 60% scaling efficiency for both clusters even when scaled beyond 60 nodes.

2.5 SUMMARY

In this chapter, we discussed some of the recent performance modeling studies targeting GPGPU- and FPGA-based heterogeneous systems. While the GPGPU performance prediction models discussed are accurate, they require significant knowledge of the underlying system architecture. In addition, they do not address the multiple levels of design space abstraction, making the model selection and implementation task difficult. Unlike the modeling efforts discussed in this chapter, our research addresses two levels of design space abstraction in the form of a multi-level performance modeling suite: *low-level* where some implementation details are present along with the system specifications; and *high-level* where the implementation details are minimum and only high-level system specifications are available. The proposed multi-level suite aims to provide straightforward and accurate runtime prediction, allowing developers to choose a performance prediction model that best satisfies their design space.

In addition to performance models for heterogeneous systems, we also discussed several architecture studies conducted using SNNs and ADF. Since our current research focuses on GPGPU-based systems, the next chapter provides additional details on the base GPGPU architectures and SNN-ADF SIA case studies.

CHAPTER 3

BACKGROUND

In this chapter, we provide background on Nvidia's Fermi and Kepler GPGPU architectures and the Compute Unified Device Architecture (CUDA) framework, and discuss the algorithmic details of the SNN-ADF SIAs studied in this research. The chapter is structured as follows. Section 3.1 describes the Fermi and Kepler GPGPU architectures and the CUDA framework for general purpose graphics computing. Section 3.2 provides background on the Spiking Neural Network (SNN) models along with the large-scale SNN simulation performed in the form of a two-level character recognition network. The non-linear anisotropic diffusion filtering (ADF) is described in Section 3.3. The chapter is concluded in Section 3.4 with a summary.

3.1 GPGPU ARCHITECTURE

The GPU architecture, initially intended as a fixed many-core processor dedicated to transforming 3D scenes to a 2D image composed of pixels, has undergone several innovations to meet the computationally demanding needs of the supercomputing research community. The traditional GPU pipeline came with several disadvantages for HPC including limited data reuse in the pipeline, excessive variations in hardware usage, and lack of integer instructions coupled with weak floating-point precision. In November 2006 [82], NVIDIA introduced the GeForce 8800 GTX with a novel unified pipeline and shader architecture. In addition to overcoming the limitations of the traditional GPU pipeline, the GeForce 8800 GTX architecture added the concept of a *streaming processor (SMP) architecture* that is highly pertinent to current GPGPU

programming. SMPs can work together in close proximity with extremely high parallel processing power. The outputs produced can be stored in fast cache and used by other SMPs. SMPs have instruction decoder units and execution logic performing similar operations on the data. This architecture allows SIMD instructions to be efficiently mapped across groups of SMPs. The streaming processors are accompanied by units for texture fetch (TF), texture addressing (TA), and caches. The structure is maintained and scaled up to 128 SMPs in the GeForce 8800 GTX. The SMPs operate at 2.35 GHz in the GeForce 8800 GTX, which is separate from core clock operating at 575 MHz. Several GPGPUs used thus far for HPC applications have architectures that are concurrent with the GeForce 8800 GTX. However, introduction of the Fermi architecture by Nvidia in September 2009 [51] has radically changed the contours of the GPGPU architecture, as discussed in this section.

3.1.1 Nvidia Fermi GPGPU Architecture

The Compute Unified Device Architecture (CUDA) programming framework [22] views the GPGPU architecture as an array of streaming multi-processors (SMPs), each containing a set of scalar processors (referred to as CUDA cores), a double-precision (DP) unit, shared memory for thread cooperation, and texture addressing and texture fetch units. The GPGPU functionality in CUDA is expressed by writing GPGPU user-defined functions, referred to as *kernels*, that are executed by all threads created in an application. While a single thread is executed on a CUDA core, a group of threads called a thread block is executed on the SMPs. The thread blocks are further divided into warps (a group of 32 concurrent threads) and half-warps (a group of 16 concurrent threads). Threads in a thread block can synchronize with each other using shared memory.

The 20-series architecture, codenamed Fermi [51], has brought numerous innovations versus previous architectures. The 512 CUDA cores are organized as 16 SMPs with 32 cores each gathered around an L2 cache. A Gigathread scheduler dispatches thread blocks to the SMP thread schedulers. The GPGPU has the capability of supporting 6 GB of GDDR 5 DRAM memory. SMPs in Fermi have an instruction cache, dual warp schedulers and dispatch units, two sets of 16 CUDA cores, 4 special function units for transcendental functions, 16 load/store units, a hefty register file, and most importantly, a configurable 64 KB of shared memory/L1 cache. The SMPs share a second level L2 cache. More information about the architecture can be found in [51]. The Fermi-based Tesla M2070 used for this research can theoretically offer 1.03 Teraflops of single-precision floating-point performance and 515 Gigaflops of double-precision floating-point performance. This GPGPU architecture is used for the verification and Strengths, Weaknesses, and Opportunities (SWO) analysis of the regression-based framework (low-level abstraction).

3.1.2 Kepler GK110 (K20) Architecture

The GK110 Kepler GPGPU devices [83] have 5 GB of GDDR5 memory, 64 KB L1 cache/shared memory, 48KB read-only cache, 1536 KB L2 cache, and a quad warp scheduler. The Kepler GPGPU device family introduces new features such as the *Next Generation Streaming Multiprocessor* (SMX) that includes 192 CUDA cores, for a total of 1536 cores in the entire GPGPU, providing tremendous performance boost at lower power consumption when compared to the earlier GPGPUs. The Kepler GPGPU devices also feature *Dynamic Parallelism* that enables dynamic spawning of new threads from the device kernel without returning to the host CPU. Furthermore, the Hyper-Q technology enables multiple CPU-cores to launch work on a single GPGPU device simultaneously, thereby increasing the GPGPU device utilization and

reducing the CPU idle time. Figure 3.1 shows the SMX of the Kepler GK110 GPGPU architecture [83]. We use the Kepler architecture for SWO analysis of the regression-based framework and high-level abstraction studies.

Figure 3.1 An SMX in Kepler GK110 Architecture [83]

3.1.3 Compute Unified Device Architecture (CUDA) Framework

In CUDA for C [22], the GPGPU functionality is defined by writing device functions, which are called *kernels*. A *thread*, which is a sequence of instructions, is instantiated several thousands of times. When a kernel is called, *N* threads execute the kernel in parallel. Threads are accessed inside kernels using built-in variables: *threadIdx*, *blockIdx*, and *blockDim*. Collections of threads

called thread blocks are executed on the SMPs. The blocks are further divided into SIMD groups of 32 threads called *warps,* which are further divided into groups of 16 threads called *half-warps.* The memory hierarchy in CUDA is comprised of a set of *registers* (on-chip) and *local memory* (residing in an off-chip DRAM) for each thread, private shared memory for thread blocks, *global memory* for all threads created, and read-only *texture cache* and *constant memory.* CUDA offers three primary optimization strategies, namely the *Memory Optimization, Execution Configuration Optimization,* and *Instruction Optimization*.

Several memory optimization strategies can be found in [22]; here we discuss the prominent ones used in this research. One memory optimization strategy is to reduce the frequent transfers between the host and the device since the host-to-device bandwidth is usually an order of magnitude lower than the device-to-device bandwidth. It is highly beneficial to transfer all of the relevant data to the device memory for processing and later transfer the data back to the host memory once all of the operations are finished. The device-host bandwidth can be most efficiently utilized by overlapping the kernel execution with data transfers using *Zero Copy* (*Z*). This feature is available only in devices with compute capability greater than or equal to 1.1. In this technique, the data transfers are performed implicitly as needed by the device kernel code. For the operation described, it is required that the device should support the host mapped memory.

Compute capability devices 2.0 and beyond introduce L1 and L2 caches for improving the global memory performance. These architectures allow the user to configure the amount of L1 cache and shared memory used. From the 64 KB of on-chip memory, 48 KB can be configured either as L1 cache or shared memory. The user is also allowed to cache the global memory either in L2 cache alone, or both in L1 and L2 caches [22]. Caching the intermediate data can promote

performance improvement in applications that involve frequent global memory data accesses or those that suffer from register pressure.

Software Pre-fetching (SP) is another useful memory optimization technique for avoiding frequent accesses to the device global memory. The technique involves the use of on-chip *Registers* and/or *Shared Memory* (SM) to cache and operate on the data. Once all of the operations are finished, the data is transferred back to the device memory. *Registers* are more commonly used for such scenarios since they do not involve *bank conflicts* that can occur with shared memory accesses. Bank conflicts occur when threads in a half-warp access the same shared memory bank. These conflicting accesses are serialized and therefore negatively impact the performance.

Execution Configuration Optimization is an effective method for hiding latency on the memory bound kernels. Execution configuration is related to the number of threads per block. Varying the number of threads per block changes the *multiprocessor occupancy*: the ratio of the number of warps running on the multiprocessor to the maximum number of warps that can physically run on the multiprocessor. The CUDA profiler [22] provides information about the multiprocessor occupancy. The number of threads per block should also remain a multiple of 32 and sufficiently large, typically greater than or equal to 192. Keeping the number of threads per block a multiple of 32 facilitates coalescing, meaning all threads in a warp complete the data access in one or more transactions.

The *Instruction-level Optimization* technique utilized in this research with CUDA involves the use of fast math functions and *Reduced Conditional Statements* (RCS). The use of fast math results in fewer clock cycles for the instruction at the expense of reduced accuracy. The compiler optimization *–use_fast_math* forces compiling arithmetic functions as fast math functions. RCS

reduces divergent paths taken within a warp. Divergent paths are serialized, which results in reduced performance.

3.2 SPIKING NEURAL NETWORKS (SNNs) AND LARGE-SCALE SNN SIMULATIONS

SNNs constitute the *third generation* of neural networks and are considered highly biologically accurate. A *spiking* neuron fires an electric pulse, commonly referred to as *spike*, at certain time intervals. The amplitude of the spike is irrespective of the input, but the timing of the spike is a function of the input. This type of time encoding is useful for many signal-processing applications. Several models have been proposed for SNNs, ranging from very computationally efficient and moderately accurate, to compute intensive and highly accurate. In [69], Izhikevich lists the 20 most prominent features of biological neurons and ranks several models based on their ability to mimic these neuron features. Four models, namely, the Hodgkin-Huxley (HH) model [84], Morris-Lecar model [85], Wilson model [86], and Izhikevich model [87] were found to satisfy the requirements of accurately modeling the neuron dynamics, and hence were used in this research not only for their validity, but also for their range of computation and communication requirements. In what follows, we provide a brief chronological overview of these four SNN models.

3.2.1. Four SNN Models

The Hodgkin-Huxley (HH) model is considered to be the most accurate and the most important model in the neuroscience community till date. As mentioned in [69], the model involves four equations and ten parameters describing neuron current activation and deactivation.

The model takes 1200 FLOPs per millisecond to the complete neuron update. In our research, we have used a 0.01 milliseconds time-step for the neuron update.

The Morris-Lecar (ML) model is another biophysically meaningful model, replicating almost all of the spiking neuron properties. The relevant equations found in [85] include hyperbolic functions, making this model the second most complex SNN model used in this study. The model takes 600 FLOPs per millisecond time-step for the neuron update. For our experiments, we have used a plausible 0.01 milliseconds time-step for the neuron update.

Wilson [86] attempted to model cortical neurons with a system of polynomial equations. This model introduces a few additional conduction channels compared to the HH model as reported in [86]. With proper tuning of the channel parameters, the Wilson model can mimic all characteristics of spiking neurons. A time-step of 0.01 milliseconds was used to evaluate the polynomial equations describing neuron dynamics. The model in general takes 180 FLOPs per millisecond for the neuron update.

In [87], Izhikevich developed a simple and very computationally efficient spiking neuron model that is almost as plausible as the most accurate HH model. Izhikevich was successful in reducing the complex HH model equations to a 2D system of ordinary equations. Izhikevich's model requires only 13 FLOPs per neuron update and still sufficiently reproduces a majority of the neuronal properties with the equations found in [87]. In our research, we have used a 1 millisecond time-step (13 FLOPS per millisecond) for neuronal dynamics update.

The time-step values used in our research for the SNN models discussed are in the range deemed sufficient for reproducing biologically relevant neuron dynamics [69]. More detailed description of the four SNN models can be found in [88]. In Table 3.1, we summarize the FLOPs/Byte ratio for the four SNN models, which provides an algorithmic analysis of the

aforementioned SNN models used in this study. The FLOPs/Byte ratio is an algorithm specific value and is defined as the ratio of the number of floating-point operations required for a complete neuron update (level-1 and level-2 of the two-level network) to the overall bytes requested (all model parameters and supporting data structures) for all of the neuron updates [88].

Table 3.1 FLOPs/Byte Ratio for SNN Models

Model	**FLOPs required for the complete neuron update**	**Bytes required for the complete neuron update**	**FLOPs/Byte Ratio**
HH	246	25	9.84
ML	147	17	8.65
Wilson	38	25	1.52
Izhikevich	13	13	1

3.2.2. The Two-Level Network

We use the SNN models discussed in the previous section for the large-scale SNN simulations. These simulations are performed using a two-level character recognition network based on [89] shown in Figure 3.2. The task of the network is to identify images from a training data set of 48 images. The level-1 neurons act as an input collection layer and the level-2 neurons act as output collection layer. Each neuron in level-1 corresponds to a pixel in the input image; hence the number of neurons in the input level is equal to the total number of pixels in the test image (*image-size*^{2}), making it the most compute-intensive layer of the two-level network. The number of neurons in the output layer, level-2, is equal to the number of images in the database, making it less computationally dense. When an input image is presented to level-1, each neuron evaluates its membrane potential based on the pixel level presented and the neuron model chosen. This process is referred to as the *evaluation of neuron dynamics*. If the pixel is "on," a constant current is supplied to the neuron for membrane potential evaluation. The input current equation for a level-2 neuron is:

$$I_j = \sum w_{ij} * f_i \tag{3.1}$$

In Equation 3.1, I_j is the net input current to the neuron j in level-2, w_{ij} is the weight of the synapse connecting neuron i in level-1 with the neuron j in level-2. A neuron in any level is said to have "fired" if its membrane potential crosses the threshold value for the selected neuron model. In our research, we accelerate the recognition phase of the network by implementing all of the level-1 neurons on the GPGPU devices since they are highly compute-intensive, while the less computationally dense level-2 neurons (input current accumulation and dynamics) are implemented on the host processors.

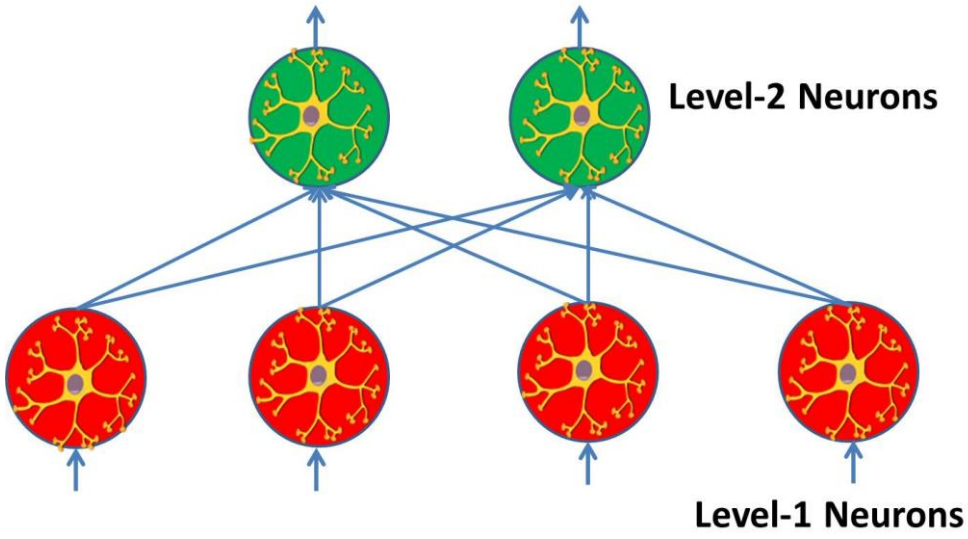

Figure 3.2 Two-level Character Recognition Network

3.3 NON LINEAR ANISOTROPIC DIFFUSION FILTER (ADF)

The quality of an image is highly critical for image processing applications such as machine vision, surveillance, medical imaging, etc. Even the most sophisticated image capturing devices are prone to noise signals from the surroundings including but not limited to Gaussian noise, Poisson noise, and Salt-and-Pepper noise. The literature reports the existence of several noise removal schemes, some of which are computationally efficient but prone to boundary errors [90],

while others require an excessively large number of iterations [91]. Some of the proposed filtering schemes such as the median filtering and hybrid median filtering (bidirectional linear median filter) preserve the edge information at the expense of fine image details ultimately leading to streak and blotched effects in the final image [92]. Out of several proposed noise removal schemes, non-linear anisotropic diffusion filtering has been reported to yield superior results [78, 93, and 94]. The anisotropic diffusion filtering scheme effectively improves the quality of noised images via piecewise smoothing and immediate localization. In piecewise smoothing of an image, the intra-region smoothing is preferred over inter-region smoothing at all scales. The immediate localization property requires the region boundaries to be sharp and coincide with the "semantically meaningful" boundaries at a given resolution. These properties of anisotropic diffusion filtering preserve the inter-region edges and fine details of the image. Therefore, it is widely used in real-time video processing [95].

The theoretical aspects of anisotropic diffusion filtering are well documented in the literature [96]. In this research, we discuss and implement a novel non-linear anisotropic diffusion filter based on the statistic-local open system proposed in [97]. In the proposed filtering scheme, only the estimated noised pixels are processed to reduce any unnecessary blurring caused by pure pixel energy diffusion. The filtering scheme also incorporates a newly designed conduction coefficient to avoid energy flow from neighboring noised pixels.

In [97], the authors assert that the traditional order-statistic filter has two shortcomings. First, the order-statistic filter tends to ignore the texture information in edges. Second, the order-statistic filter cannot efficiently filter the impulse noise in high-level noised images. In what follows, we describe the steps proposed in [97] to alleviate these problems. To address the first problem, the proposed filter only processes the estimated noised pixels in a single iteration,

thereby only allowing for local diffusion. The proposed scheme then compares the real value of the center pixel with the pixel value after the order-statistic filtering. If the difference in the values is above a threshold level K_{noise}, only then will the pixel be declared a noised pixel, otherwise it is declared a pure pixel.

To address the second problem associated with the traditional order-statistic filter, the authors in [97] propose an anisotropic diffusion system based on a local open system, where part of the pixels are labeled as *convergences* and others as *origins*. The *convergence* pixels represent the energy flowing in, whereas the *origin* pixels represent the energy flowing out. The neighbors of noised pixels are declared as either *convergences* or *origin*s and their values remain unchanged. The authors claim if the above two labels are properly chosen, the image details can be well preserved. The authors also propose a new conduction coefficient $sgn_i\ (med(u_i))*c_i$, to avoid the effects of neighboring noise energy as shown in Equation 3.2.

$$\begin{aligned} \mathrm{sgn_i}(med(u_i))*c_i = 0 \qquad & \text{if } K_{noise} \le |\, med(u_i) - u_i| \\ & g(||\nabla_i u||) \text{ Otherwise} \end{aligned} \tag{3.2}$$

where, u_i represents the pixel in the i^{th} direction (i = N, S, E, W), $med(u_i)$ represents the median filter pixel value in the i^{th} direction, and c_i represents the conduction coefficient in the i^{th} direction. As suggested in [97], the conduction coefficient c_i can be selected as the gradient of the image in the i^{th} direction. The new conduction coefficient in Equation 3.2 is zero if u_i is estimated as a noised pixel; otherwise the conduction coefficient follows the gradient of u_i. The proposed filter is then modeled as shown in Equation 3.3.

$$\begin{aligned} &\frac{\delta u}{\delta t} = div(\mathrm{sgn}(med(u)).c.\nabla u) \\ &u(x, y; t = 0) = u_0 \end{aligned} \tag{3.3}$$

To achieve sufficiently accurate filter output, we have chosen 30 iterations for the implementation as suggested in [98]. The quality of the filter is evaluated using the Peak Signal-to-Noise Ratio (PSNR) criteria as shown in Equation 3.4 where, MSE stands for mean squared error, *u* represents the original noise-free image, and *v* represents the filtered output image.

$$PSNR = 10 * \log(255 * 255 / MSE)$$
$$MSE = [\sum_{i} \sum_{j} (u(i, j) - v(i, j)) \wedge 2] / 256 \wedge 2 \quad (3.4)$$

The anisotropic diffusion filtering scheme used in this research is summarized as follows:

1) Estimate the noised pixels. If the difference between the real center pixel value and the value of pixel after the order-statistic filtering is above a threshold K_{noise}, the pixel is declared as a noised pixel and will be processed. The threshold K_{noise} for our implementation is 40.
2) Evaluate the new conduction coefficient using Equation 3.2.
3) Perform the anisotropic diffusion filtering using Equation 3.3.
4) Repeat steps 1 through 3 for 30 iterations.

3.4 SUMMARY

In this chapter, we discussed the base GPGPU architectures utilized in this research, namely Nvidia's Fermi architecture and Kepler K20 architecture and the CUDA framework for general purpose graphics computing. We also discussed the four SNN models, the two-level character recognition network for large-scale simulations, and the anisotropic diffusion filter (ADF) for massive images. In the next chapter, we describe the experimental set-up, mapping and orchestration of the SIA algorithms on GPGPU clusters. We also provide the performance

analysis study for the SNN-ADF SIA implementations to confirm their applicability for the verification of the multi-level modeling suite.

CHAPTER 4

EXPERIMENTAL SET-UP, MAPPING, ORCHESTRATION, AND PERFORMANCE ANALYSIS STUDY

In this chapter, we provide the details of the experimental set-up, SNN-ADF SIA mapping, and multi-GPGPU orchestration. We also provide a performance analysis study for the SNN-ADF SIA implementations conducted on the NCSA Forge cluster. The contents of this chapter are focused toward the verification of the performance modeling suite. Section 4.1 details the layout of the NCSA's Forge GPGPU cluster and GPGPU-augmented Palmetto cluster. Sections 4.2 and 4.3 describe the mapping and orchestration of SNN and ADF simulations, respectively. The performance analysis study for the SNN-ADF SIAs follows in Sections 4.4 and 4.5. The chapter is summarized in Section 4.6.

4.1 EXPERIMENTAL SET-UP

4.1.1 NCSA Forge Cluster

Our research uses the Forge GPGPU cluster at the National Center for Super-Computing Applications (NCSA) [12] for the large-scale SNN and ADF simulations. The 153 Teraflop cluster is composed of 36 Dell PowerEdge C6145 servers; each server is connected to six Fermi-based Tesla M2070 GPGPUs via three PCI-Ex Gen2x16 slots. Each server is equipped with two 2.4 GHz AMD Opteron Magny-Cours 6136 processors, eight cores each. The network interconnect is comprised of Infiniband QDR. Our implementations were developed using CUDA 4.0 and OpenMPI version 1.4.3 [99] on Red Hat Enterprise Linux 6. More information

on the Forge GPGPU cluster can be obtained from [12]. Figure 4.1 provides an example layout of a server in the Forge cluster.

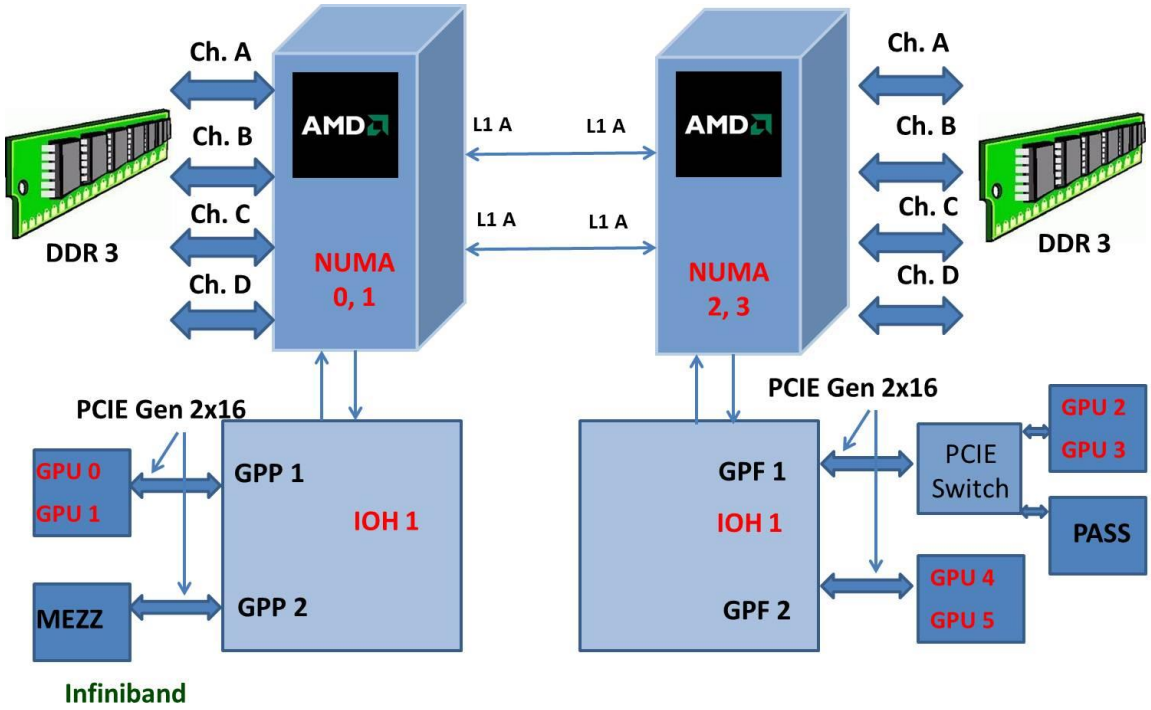

Figure 4.1 An Example Layout of a server in the NCSA Forge Cluster [12]

4.1.2 GPGPU-augmented Palmetto Cluster

The research also uses the GPGPU-augmented Palmetto cluster at Clemson University [15] for the SNN-ADF SIA implementations, GPGPU Design Space Exploration (DSE) study using low-level abstraction, and the development of the high-level abstraction prediction models. The Palmetto Cluster includes 12 GPGPU HP SL250 servers, with each server connected to two Fermi-based Nvidia Tesla M2075 [51] GPGPU devices via Peripheral Component Interconnect Express (PCI-Ex) bus. Recently, the cluster acquired an additional 96 nodes equipped with Nvidia Kepler GK110 (K20) GPGPU devices [83]. Each server is composed of two 2.4 GHz Intel E5-2665 processors with 8 cores each and 64 GB RAM. The servers are connected via Infiniband [10]. For our implementations, we used CUDA 4.2 [26] and MPI version 2.2 [100] on

Scientific Linux 6. Additional details on the Palmetto Cluster can be found in [15]. The low-level abstraction DSE studies were performed on the Kepler devices. Both the Fermi and Kepler GPGPU architectures were employed for the Strengths, Weaknesses, and Opportunities (SWO) analysis of the regression-based framework (low-level abstraction). This analysis shows the ability of the framework to span generations of the GPGPU architecture. The high-level abstraction modeling and analysis were completed using Kepler devices.

4.2 SNN MAPPING AND ORCHESTRATION

In this sub-section, we first provide details of the network mapping for the single-GPGPU implementation that is subsequently extended to a multi-GPGPU implementation.

As discussed in Chapter 3, level-1 is the most compute-intensive layer of the network since the number of neurons is equal to the number of pixels in the input image; therefore these operations are performed on the GPGPU device. Each GPGPU thread evaluates the dynamics of a single level-1 neuron. Therefore, the number of GPGPU threads created is equal to the number of level-1 neurons. The GPGPU device then provides the host processor with the level-1 neuron firing information, the *global firing vector*, which is used by the host processor to obtain the level-2 neuron currents and dynamics. The level-2 computations (current accumulation and dynamics) are implemented on the host processor since the level-2 neuron computations constitute less than 5% of the total computation overhead and, implementing the level-2 dynamics on the GPGPU would require transfer of the weight matrix (matrix-size = level-2 neurons * level-1 neurons) to the GPGPU device memory. Hence any computational improvement obtained by implementing level-2 neuron dynamics will be insufficient to amortize the communication overhead involved in transferring the large weight matrix to the GPGPU

device. The single-GPGPU implementation was optimized with memory-level, instruction-level, and execution configuration level optimizations as mentioned in [101].

The host-device bandwidth was further optimized using a *block firing vector* concept introduced in [88]. The block firing vector is implemented in the device shared memory to avoid transferring the global firing vector in each algorithmic time-step. The block firing vector is similar to the global firing vector but instead acts as a collection of flags for thread blocks. Since the threads are collected in thread blocks of size: *blocksize*, the block firing vector is *blocksize* magnitude smaller than the global firing vector, and hence can be transferred from the device to host in each time-step with minimal overhead. If at any time-step the block firing vector contains information of a firing event, only then will the entire global firing vector be transferred from the device to host and then read by the host. Figure 4.2 illustrates the block firing vector concept.

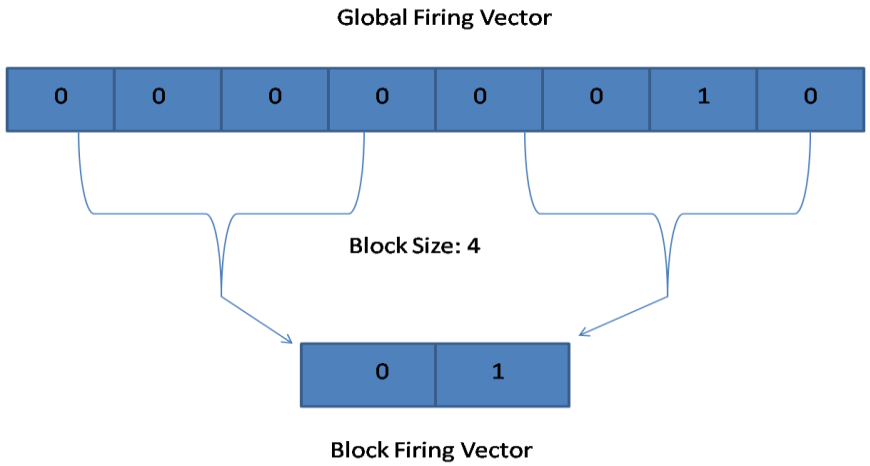

Figure 4.2 The Concept of Block Firing Vector

The single-GPGPU implementation is then extended to a multi-GPGPU implementation. The MPI ranks were assigned in node-packing fashion, meaning the ranks are packed into nodes. The nodes were configured with a maximum of six MPI processes per node allowing for a 1:1 CPU-core/GPGPU-device ratio at each node and potentially reducing long distance inter-node communication. The GPGPU devices were allotted to the CPU cores using modulo rule where an

MPI process with rank *n* is coupled with the GPGPU device number, *n modulo 6* [4]. Future work beyond this dissertation will investigate the impact of other CPU-core/GPGPU-device ratios on application performance.

The multi-GPGPU orchestration follows the Master-Worker Paradigm as shown in Figure 4.3. MPI rank 0 acts as the master process that scatters the level-1 neuron inputs to all other processes. The level-1 neuron parameters are initialized to the SNN model specific constant values at each MPI process, and hence require no MPI communication. Each CPU-GPGPU pair works as an independent unit where the GPGPU device evaluates the partial level-1 neuron dynamics and the host processor evaluates the partial level-2 currents using the firing vector obtained from its designated GPGPU device. The partial level-2 currents from each MPI process are then accumulated at MPI rank 0 where the complete level-2 neuron dynamics are evaluated and the image detection decision is made. The level-2 neuron computations on the hosts were accelerated using OpenMP.

As discussed later in this chapter, we successfully scaled the neural network size from 5.7 million to over 200 million neurons.

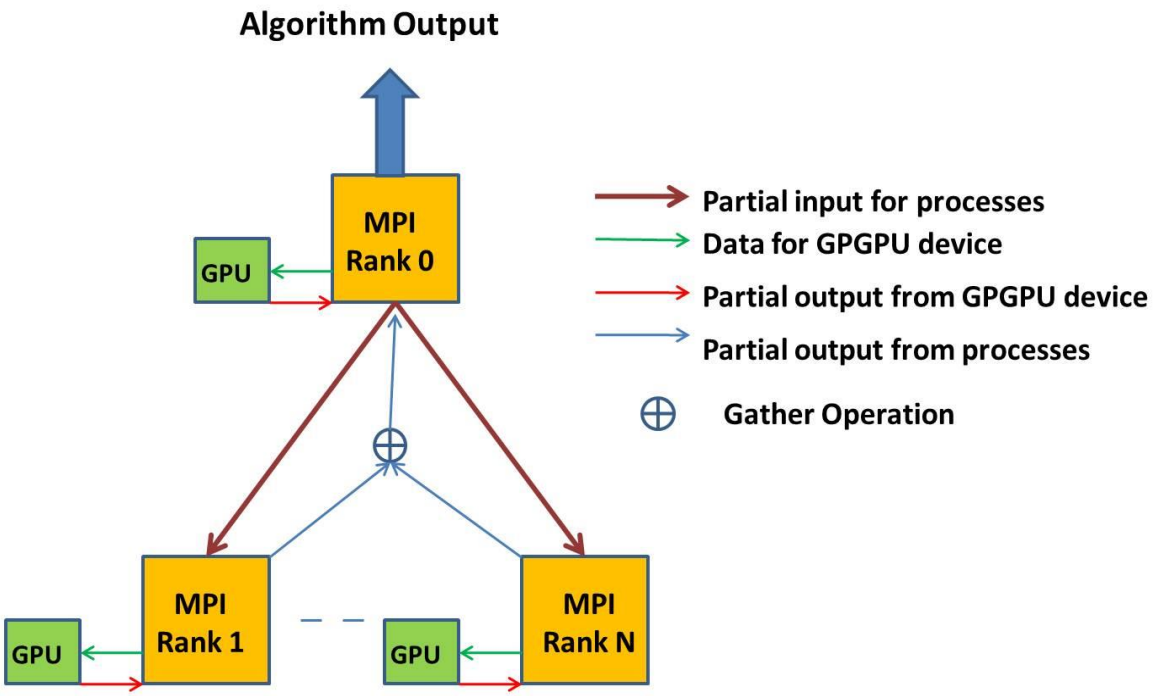

Figure 4.3 Multi-GPGPU Orchestration using Master-Worker Paradigm

4.3 ADF MAPPING AND ORCHESTRATION

4.3.1 ADF Mapping Methodology

The steps involved in the anisotropic diffusion scheme are described in Chapter 3. The algorithm involves the evaluation of two computationally intensive tasks: 1) median filtering of the input image to evaluate the conduction coefficient as shown in Equation 3.2; and 2) evaluation of the partial differential equation (PDE) to perform the anisotropic diffusion filtering as shown in Equation 3.3. Since these operations are highly data-parallel, they are performed on the GPGPU devices using two separate GPGPU kernels, namely the *median_kernel* and *PDE_kernel*, whereas the CPU host processor(s) only perform communication operations (row exchange) and serial processing (image padding).

In each of the GPGPU kernels, a single CUDA thread operates on a single pixel. Therefore, the number of threads created for each kernel is equal to the number of pixels in the input image.

The GPGPU kernels were optimized with CUDA optimization techniques including execution configuration optimization, memory optimization, and branch divergence reduction. The execution configuration optimization involves the selection of an optimal thread-block configuration to maximize the *multiprocessor occupancy*: the ratio of the number of warps (a group of 32 concurrent threads) running on the multi-processor to the maximum number of warps that can physically run on the multi-processor. In our implementation, we chose a thread-block configuration of 256 threads per block to maximize the multiprocessor occupancy. The *Software-Prefetching* (SP) memory optimization technique was used to fetch the neighboring pixel values into the GPGPU registers, reducing frequent incoherent accesses to the device global memory and promoting performance. Divergent branches, due to conditional statements, lead to warp serialization and low execution unit utilization, ultimately impeding performance [102]. The conditional statements were replaced with ternary operators to reduce divergent branches. Detailed information on CUDA optimization techniques used in this research is presented in Chapter 3.

4.3.2 Multi-GPGPU Orchestration

The network set-up and multi-GPGPU orchestration for ADF is similar to that of the SNN simulations described in Section 4.2.

The orchestration for ADF is divided into four stages. In the first stage, the master process MPI rank 0 reads the input image and scatters the image tiles in row-wise fashion to all other processes. In the second stage, each of the individual processes pads its respective image tile to avoid any out-of-bound conditions. The adjacent processes then exchange the boundary rows, labeled as *ghost rows*, to avoid any boundary errors. The MPI point-to-point routine *Sendrecv* is used to accomplish the exchange operation. Once the above serial processing and

communication operations are completed, the implementation proceeds to the third stage where each CPU-GPGPU pair works as an independent unit. The CPU host transfers the image tile to the GPGPU device memory and the GPGPU device performs the filtering iterations on the image tile as described in Chapter 3. Once the GPGPU device completes the iterations, it transfers the output image tile to its respective CPU host. The CPU host then un-pads the output image tile to remove unnecessary ghost rows and pad-boundaries. In the fourth and final stage, the master process (MPI rank 0) gathers the output image tiles from all other processes, constructs the final output image, and performs the PSNR check using Equation 3.4. Figure 4.4 elucidates the four stages of the multi-GPGPU implementation.

As will be discussed in detail in Section 4.5, our ADF implementation successfully scaled up to 156 mega-pixels. In the next section, we present the SIA performance analysis study conducted on the Forge GPGPU cluster. We investigate the scaling behavior of the SIAs by varying configuration from 2- to 32-nodes. As elaborated in Chapter 5, a *node* consists of a single CPU-host tightly coupled with a GPGPU device to perform computations and data exchange. For a few specific SIAs, we provide the speed-up achieved by the multi-GPGPU implementations over equivalent MPI-only implementations.

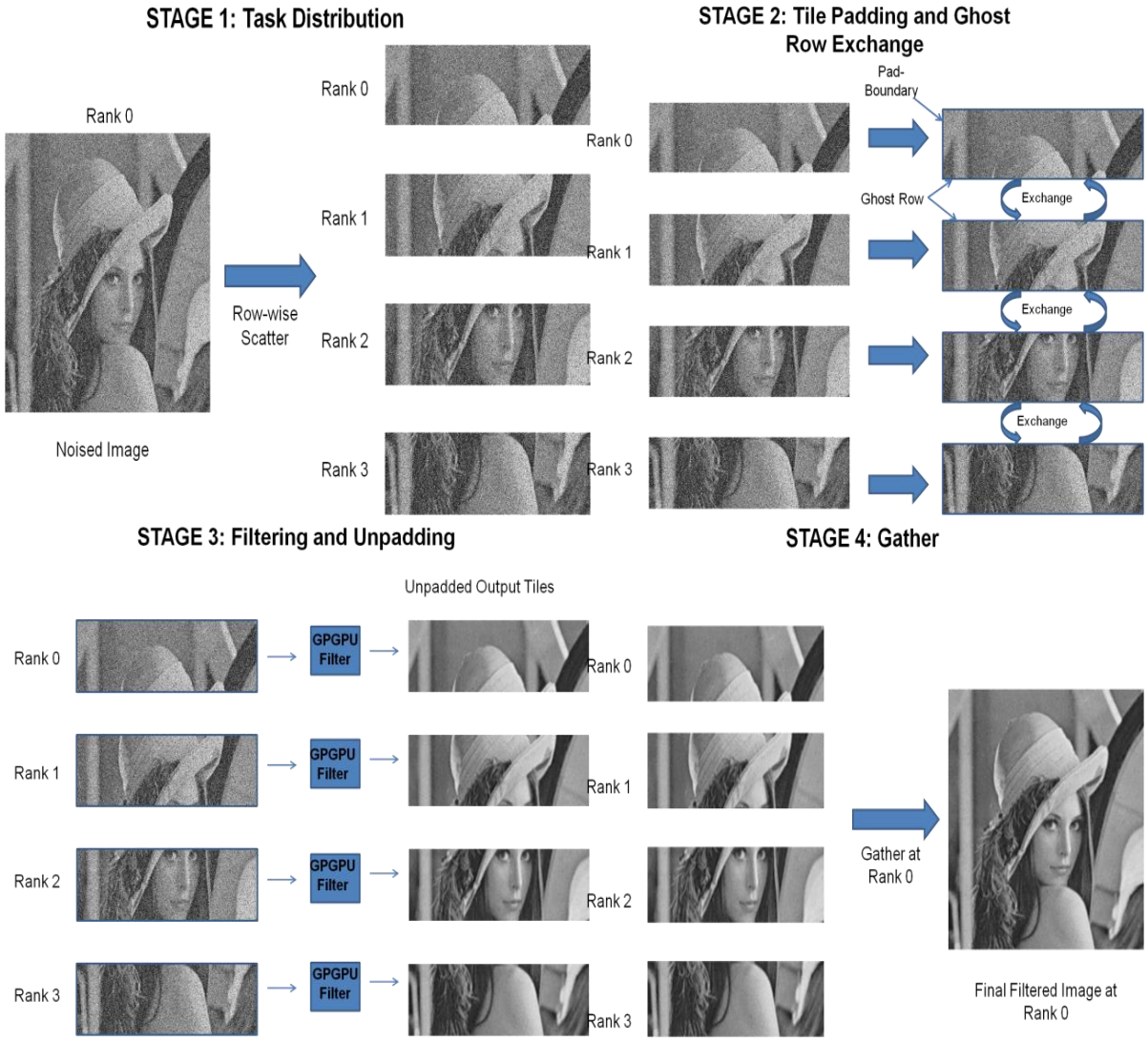

Figure 4.4 Four Stages in Multi-GPGPU Implementation

4.4 PERFORMANCE ANALYSIS STUDY: SNNs

In this section, we present the performance analysis study of the four SNN models conducted on the Forge GPGPU cluster. We discuss the application runtime values for all of the tested node configurations and show the overall runtime breakdown in terms of GPGPU time, CPU time, and communication time for a 32-node configuration. For the HH and Izhikevich models, we compare the multi-GPGPU implementation with an equivalent MPI-only implementation. The HH and Izhikevich models are particularly interesting since they represent the two ends of the

computation-to-communication spectrum for the SNN models. First, we present the results for the compute-intensive HH model and then proceed to the compute-efficient Izhikevich model.

4.4.1 Performance Analysis Study: HH Model

The statistical-average runtime values for different node configurations versus the neural network size are given in Table 4.1. These runtimes correspond to those measured by the master process, MPI rank 0, which distributes the tasks and makes the final image detection decision. The implementation for the HH model successfully scaled the two-level network to 200 million neurons using a 32-node configuration with a statistical-average runtime of 3315.4 milliseconds. The dashes in the table indicate problem sizes that do not fit in the GPGPU device memory, resulting in a configuration failure for that particular neural network size.

Table 4.1 HH model: Statistical-Average Runtime Values (in milliseconds)

Node Configuration	**Network Size (in millions)**			
	12.96	**51.8**	**92.16**	**207.36**
2	1946.99	-	-	-
4	1123.4	4172.82	-	-
8	725.8	2492.45	4443.04	-
16	512.68	1568.03	2663.6	-
32	360.63	922.37	1529.23	3315.4

As seen in Table 4.1, the scalability of the implementation generally improves with an increase in network size. We define the runtime *improvement ratio* as the ratio of runtimes of two successive node configurations for a given network size. For a network size of 12.96 million neurons, the runtime improvement ratio is 1.8 for 2- vs. 4-node, 1.63 for 4- vs. 8-node, 1.5 for 8- vs. 16-node, and 1.6 for 16- vs. 32-node configuration. However, for a larger network size, 51.8 million neurons, the improvement ratios are better with values 1.67, 1.6, and 1.7 for 4 vs. 8, 8 vs. 16, and 16 vs. 32-node configuration, respectively. The above scaling behavior is expected since the amount of computations per GPGPU device decreases with the CPU-host/GPGPU-device

pair (node) scaling. Consequently, for smaller network sizes, the GPGPU computations are not sufficient to amortize the necessary CPU computations and MPI communications.

Figure 4.5 further supports the observed scaling. The figure provides the runtime broken into: GPGPU time (kernel time and host-device transfer time), CPU time (level-2 currents and dynamics), and MPI communication time for a 32-node configuration versus the network size. As the network size increases, the number of computations per GPGPU device increases significantly, thereby making the computations highly dominant with respect to the overall runtime. Because GPGPU computations generally scale well, their dominance with respect to the application runtime is highly amenable to the overall scalability.

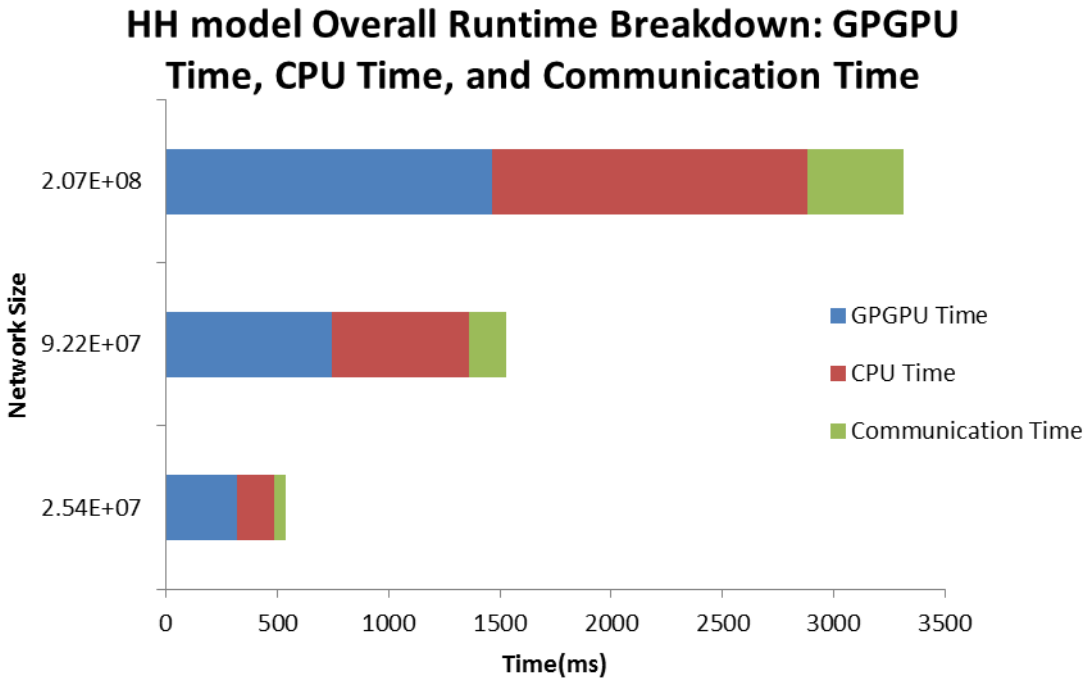

Figure 4.5 HH model: Runtime Breakdown for 32-node Configuration

Table 4.2 provides the speed-up of the multi-GPGPU implementation over an equivalent MPI-only implementation for many of the intermediate network sizes tested. As shown in Table 4.2, the speed-up over the equivalent MPI implementation increases with the increase in network

size for all of the node configurations. The increased speed-up is due to the amortization of MPI communication by GPGPU computations due to the increased number of GPGPU computations required by the increasing network size. The speed-up values are particularly large for the HH model due to its high FLOPs/Byte ratio requirements (see Table 3.1). This data supports the claim that applications with high FLOPs/Byte ratios are particularly suited for GPGPU-based implementations [88]. Further inspection of Table 4.2 reveals that for a fixed network size, the speed-up of the multi-GPGPU implementation over the equivalent MPI-only implementation declines with the node scaling due to fewer computations per GPGPU device. As explained previously, a significant number of computations are required to fully utilize the compute capabilities of the GPGPU device; hence large node configurations observe lower speed-up values for smaller network sizes.

Table 4.2 HH model: multi-GPGPU vs. MPI-only Implementation

Node Configuration	**Network Size (in millions)**			
	1.44	**9.73**	**25.4**	**92.2**
2	187x	340x	-	-
4	146x	288x	374x	-
8	75x	220x	264x	355x
16	44x	162x	233x	306x
32	20x	90x	120x	253x

4.4.2 Performance Analysis Study: ML Model

The statistical average runtime values for the ML model are given in Table 4.3. As seen in the same table, for a given network size, the improvement ratio drops with node scaling due to decreasing GPGPU device computations. For the network size 25.4 million neurons, the improvement ratios are 1.76, 1.70, 1.46, and 1.40 for 2- vs. 4-node, 4- vs. 8-node, 8- vs. 16-node, and 16- vs. 32-node configurations, respectively. For a large network size, 51.8 million for instance, the improvement ratios are better with values: 1.75, 1.58, and 1.54 for 4- vs. 8-node, 8-

vs. 16-node, 16-node vs. 32-node configurations, attributed to the increased GPGPU device computations. Additionally, for a given node configuration, the improvement ratio improves with the network size due to increasing computations that amortize the communication overhead. As seen in the same table, the improvement ratios for a 32-node configuration are 1.40, 1.54, and 1.61 for network sizes 25.4, 51.8, and 92.16 million, respectively. Figure 4.6 provides the runtime broken into: GPGPU time, CPU time, and MPI communication time.

Table 4.3 ML model: Statistical-Average Runtime Values (in milliseconds)

Node Configuration	**Network Size (in millions)**			
	25.4	**51.8**	**92.16**	**207.36**
2	2064	-	-	-
4	1169	2309	-	-
8	691	1319	2316	-
16	472	831	1383	-
32	340	540	859	1768

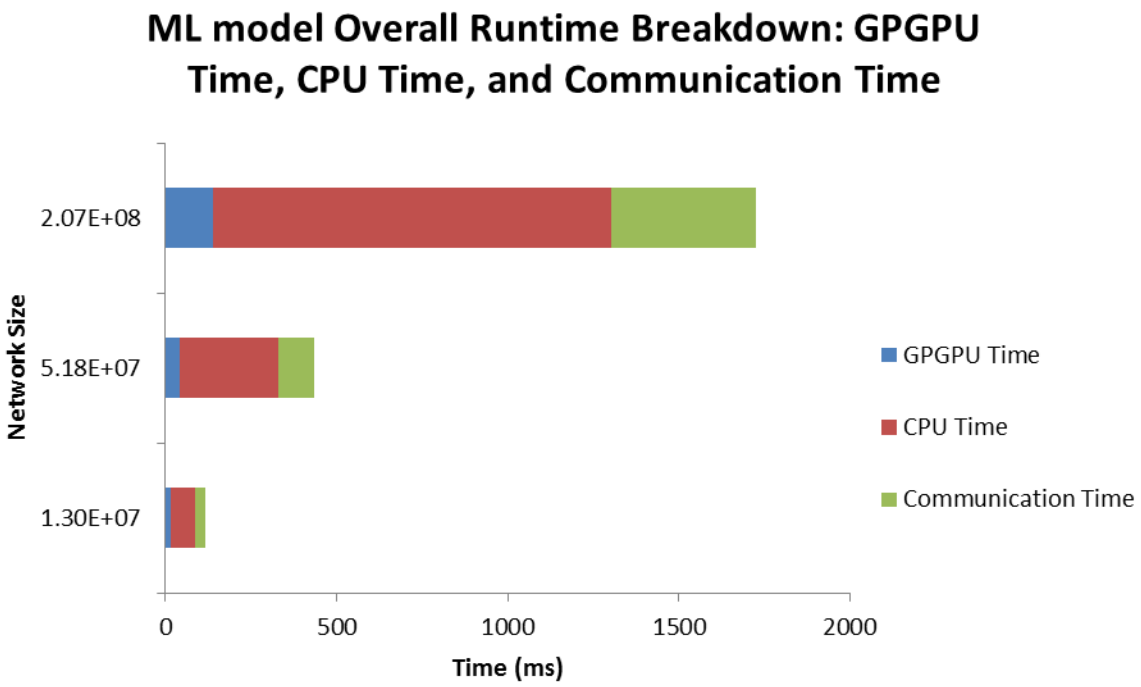

Figure 4.6 ML model: Runtime Breakdown for 32-node Configuration

Unlike the HH model, the ML model with low FLOPs, bytes, and FLOPs/Byte ratio requirement (see Table 3.1) exhibits relatively short GPGPU execution time, while the CPU time and MPI communication time dominate the overall runtime. Consequently, the improvement ratios are relatively weak for the ML model compared to the HH model. However, the scalability is preserved due to dominant CPU computations that scale better compared to the MPI communications.

4.4.3 Performance Analysis Study: Wilson Model

The statistical-average runtime values for the Wilson model are presented in Table 4.4. Similar to the previously discussed SNN models, the improvement ratio drops for a given neural network size with the node scaling. Also seen in Table 4.4, the improvement ratio is slightly weaker compared to the ML model. For 16- vs. 32-node configuration, the improvement ratios are 1.27, 1.44, and 1.53 versus 1.40, 1.54, and 1.61 for the ML model. As seen in Table 3.1, the FLOPs/Byte ratio for the Wilson model is low compared to the ML model, thereby resulting in relatively weak scaling behavior. Figure 4.7 provides the overall runtime breakdown for a 32-node configuration. As seen in the same figure, the Wilson model is less computationally dense compared to the previously discussed SNN models. Consequently, the MPI communication time contributes significantly to the overall runtime, leading to relatively weak scaling behavior for the Wilson model.

Table 4.4 Wilson model: Statistical-Average Runtime Values (in milliseconds)

Node Configuration	**Network Size (in millions)**			
	25.4	**51.8**	**92.16**	**207.36**
2	1827	-	-	-
4	1200	2334	-	-
8	679	1256	2152	-
16	485	815	1328	-
32	381	564	865	1735

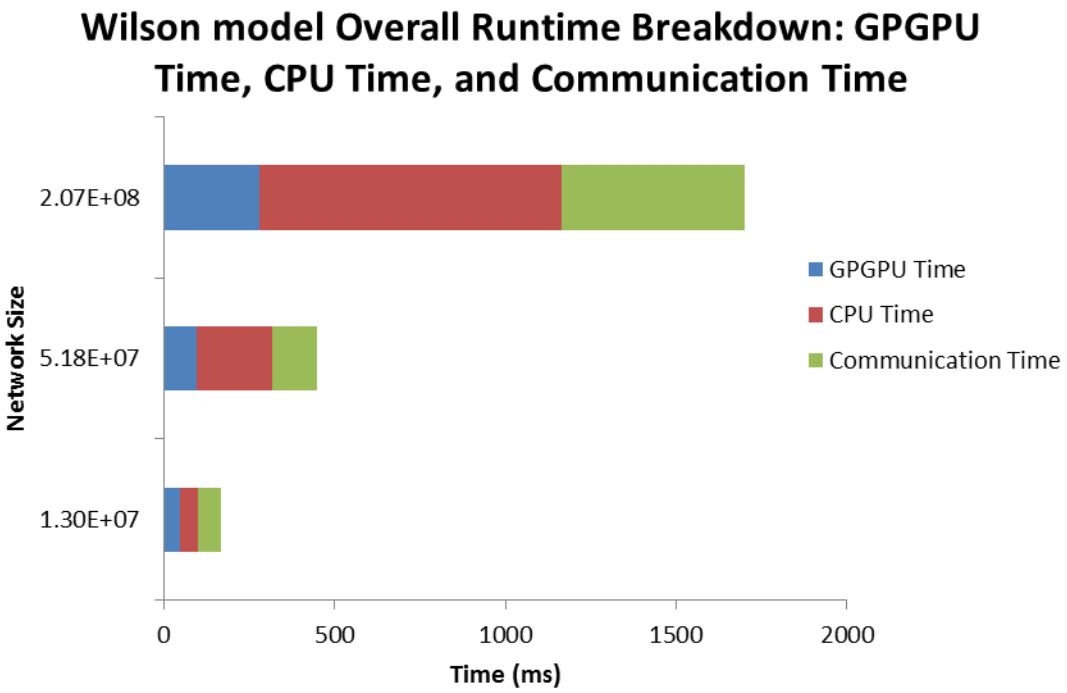

Figure 4.7 Wilson model: Runtime Breakdown for 32-node Configuration

4.4.4 Performance Analysis Study: Izhikevich Model

The statistical-average runtime values for different node configurations versus the network size using the Izhikevich model are given in Table 4.5. Unlike the high FLOPs/Byte ratio models, strong scaling is not observed for the low FLOPs/Byte ratio Izhikevich model as seen in Table 4.5. In addition to the lower number of computations in the Izhikevich model (see Table 3.1), the lower number of computations per GPGPU device further impedes the scaling performance. Figure 4.8 provides the overall runtime breakdown for the 32-node configuration in terms of CPU time, GPGPU time, and communication time.

Table 4.5 Izhikevich model: Statistical-Average Runtime Values (in milliseconds)

Node Configuration	**Network Size (in millions)**			
	25.4	**51.8**	**92.16**	**207.36**
2	1425			
4	829	1637		
8	499	945	1669	-
16	332	583	963	-
32	254	392	614	1260

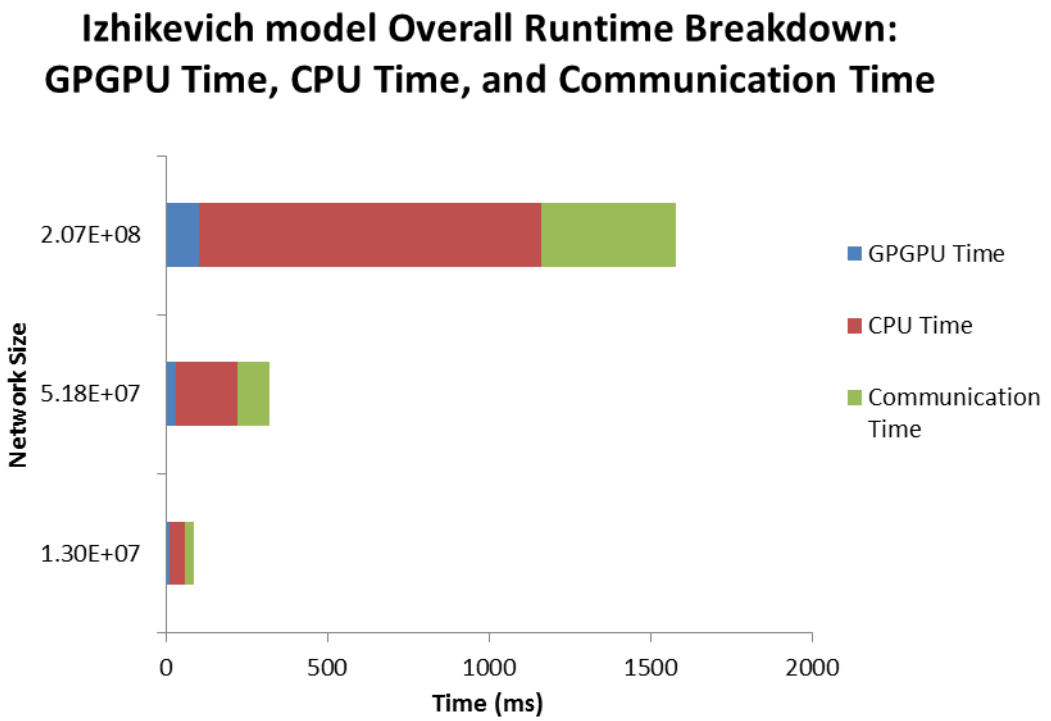

Figure 4.8 Izhikevich model: Runtime Breakdown for 32-node Configuration

As seen in Figure 4.8, the MPI communication time continues to dominate the GPGPU time as the network size increases, leading to sub-optimal performance for the Izhikevich model. Although computations per GPGPU device also increase with an increase in network size, the increase is marginal due to nominal number of computations in the Izhikevich model.

Table 4.6 presents the performance comparison of the multi-GPGPU implementation and MPI-only implementation. The 32-node configuration attained a speed-up of 2.87x versus the 32-processor MPI-only implementation. As seen in Table 4.6, the increase in speed-up with the increase in network size is marginal for the node configurations examined. The explanation for the decline in the speed-up with the increase in node configuration for fixed network size is the same as was given for the HH model.

Table 4.6 Izhikevich model: multi-GPGPU vs. MPI-only Implementation

Node Configuration	Network Size (in millions)			
	1.44	**9.73**	**25.4**	**92.2**
2	3.9x	4.0x	4.0x	-
4	2.8x	3.0x	3.2x	-
8	2.3x	2.0x	2.4x	-
16	1.5x	2.5x	1.6x	2.7x
32	1.2x	1.1x	2.4x	2.5x

The Izhikevich model is an interesting case for multi-GPGPU implementation. Although the application itself is massively-parallel, it involves only a nominal amount of computations per byte accessed. Therefore, the GPGPU computations cannot amortize the increased CPU computation and MPI communication overhead as the SNN network size increases. The Izhikevich model explored in this research serves well to highlight the importance of an optimal application-to-accelerator cluster match. It is claimed that applications should not only expose sufficient parallelism, but should also yield enough computations to fully utilize the compute capabilities of heterogeneous clusters. Nonetheless, our multi-GPGPU implementations produced performance advantages versus the equivalent MPI-only implementations as shown in this section. A thorough analysis of the impact of GPGPU kernel optimizations on SNN implementations is given in [88].

In this section, we presented the performance analysis study of the four SNN models conducted on the Forge GPGPU cluster. The two-level character recognition network (see Figure 3.2) based on the four SNN models successfully scaled to 200 million neurons using a 32-node (CPU-host/GPGPU-device pairs) configuration. In addition to providing significant speed-ups, as high as 282x over an equivalent MPI-only implementation, the multi-GPGPU implementation for the HH model scaled well with the SNN network size. Although the scaling behavior was found to be satisfactory for other SNN models, the runtime improvement ratios were found to fall with the decrease in FLOPs/Byte ratio requirements (HH to Izhikevich models). The

implementation for the Izhikevich model highlighted the importance of an optimal application-to-accelerator cluster match for maximum application performance. It is claimed that applications should not only expose sufficient parallelism, but should also yield enough computations to fully utilize the compute capabilities of heterogeneous clusters.

4.5 PERFORMANCE ANALYSIS STUDY: ADF

In this section, we present the performance results for the multi-GPGPU implementation of the non-linear anisotropic diffusion (ADF) filter. First, we compare the runtime performance of the optimized and un-optimized versions of the GPGPU kernels. Second, we present the filter output quality evaluation using the PSNR criteria as discussed in Section 3.3. Third, we discuss the application runtime for different node configurations and the scalability analysis. Fourth, to assist with the scalability analysis, we provide the application runtime breakdown in terms of GPGPU time, CPU time, and communication time for intermediate node configurations. The section concludes by comparing the multi-GPGPU implementation with an equivalent MPI-only implementation.

As mentioned in Section 4.3, the ADF scheme requires two separate GPGPU kernels, namely the *median_kernel* and *PDE_kernel*. Table 4.7 provides the statistical-average runtimes of the optimized and un-optimized kernel versions versus the test image size. The optimized kernel version employs all of the CUDA optimization techniques mentioned in Section 4.3 and performs approximately 4.5 times faster than the un-optimized version for all the test image sizes as shown in the same table. The un-optimized kernel version lags in runtime performance due to frequent incoherent global memory accesses and divergent branches resulting from conditional statements. The frequent incoherent global memory accesses waste the GPGPU device's memory

bandwidth and the divergent branches lead to warp serialization, both of which are highly detrimental to performance.

Table 4.7 ADF: Statistical-Average Kernel Runtimes (ms)

	Image Size				
Kernel Version	**4096x4096**	**5120x5120**	**7680x7680**	**10240x10240**	**12800x12800**
Optimized	669.23	1073.8	2468.031	4363.16	7083.01
Un-Optimized	3030.50	4973.13	11285.24	19873.33	31957.10

The multi-GPGPU implementation of the ADF algorithm was tested using multi-GPGPU node configurations varying from 2- to 32-nodes. The standard Lenna test images were used to evaluate the filter operation. The following Lenna test image sizes were obtained using the MATLAB *imresize* command: 4096x4096, 5120x5120, 7680x7680, 10240x10240, and 12800x12800. The Salt-and-Pepper noise was added to each of the above Lenna test image sizes with 30% noise density using the MATLAB *imnoise* command. More information on the MATLAB commands is available in [103].

Table 4.8 provides the PSNR values for noised test images of varying sizes used for the filter implementation. Table 4.9 provides the final output PSNR values for different node configurations versus the image size. A careful inspection of Equation 3.4 in Chapter 3 suggests that since PSNR is inversely related with the mean square error (MSE), a high value of PSNR implies a good quality output image. As seen in Table 4.9, the final output images attain high PSNR, thereby indicating good noise removal quality of the implemented filter. The output PSNR values are also consistent across all node configurations. The output PSNR value for any node configuration is observed to decrease with the test image size due to different initial PSNR values for the test images as seen in Table 4.9.

Table 4.8 PSNR Values (in dB) for Varying Test Image Sizes

Noised Image Size	PSNR Value
4096 x 4096	10.67
5120 x 5120	10.67
76280 x 7680	12.633
10240 x 10240	12.74
12800 x 12800	12.70

Table 4.9 Final output PSNR Values (in dB) for Varying Images Sizes and Node Configurations

Node Configuration	Image Size				
	4096x4096	**5120x5120**	**7680x7680**	**10240x10240**	**12800x12800**
1	37.01	37.10	27.25	24.62	22.71
2	37.007	37.09	27.25	24.62	22.71
4	36.99	37.08	27.25	24.61	22.71
8	36.96	37.06	27.24	24.61	22.70
16	36.91	37.02	27.22	24.60	22.70
32	36.79	36.93	27.20	24.58	22.68

Table 4.10 provides the statistical-average runtime values for different node configurations versus the test image size. These values correspond to those measured by the master process, MPI rank 0, which distributes the tasks and gathers the final filtered output image. As seen in the table, a 32-node configuration achieves a statistical-average runtime of 1404.34 milliseconds for the image size, 12800 x 12800, which corresponds to 156 mega-pixels. Table 4.11 presents the scaling efficiency values (η) for successive host-device pair configurations. The scaling efficiency is calculated using:

$$\eta = \frac{0.5 * T_a}{T_{2a}} * 100\% \quad a \geq 1 \tag{4.1}$$

where T_a and T_{2a} represent the time required to complete a unit of work on *a* and *2a* processors, respectively.

Table 4.10 ADF: Statistical-Average Runtime Values (ms)

Node Configuration	Image Size				
	4096x4096	**5120x5120**	**7680x7680**	**10240x10240**	**12800x12800**
1	1153.3	1776.02	4114.62	7181.89	12118.54
2	734.90	1145.92	2568.81	4570.64	7121.68
4	430.35	661.38	1505.34	2673.46	4146.87
8	316.77	486.96	1028.70	1979.80	3147.98
16	207.76	318.64	643.78	1238.33	2104.50
32	154.26	230.52	515.26	904.85	1404.34

Table 4.11 ADF: Scaling Efficiency Values, η (%)

Node Configuration	Image Size				
	4096x4096	**5120x5120**	**7680x7680**	**10240x10240**	**12800x12800**
1	-	-	-	-	-
2	78.5	77.49	80.08	78.56	85.08
4	85.4	86.63	85.32	85.48	85.86
8	67.92	67.90	73.16	67.51	65.86
16	76.23	76.41	79.90	79.94	74.8
32	67.34	69.11	62.47	68.42	74.93

As seen in Table 4.11, the scaling efficiency for all node configurations generally improves with the increase in test image size. Additionally, for a given test image size, the scaling efficiency decreases with node scaling. This behavior is expected since the amount of computations per GPGPU device decreases with node scaling. Consequently, for smaller test image sizes, the GPGPU computations are not sufficient to amortize the necessary CPU computations and MPI communications. Also seen in Table 4.11, the scaling efficiency values do not reach the maximum value of 100%, which is largely due to the MPI communications overhead.

Figure 4.9 further supports the scalability explanation given above, justifying the low scaling efficiency values for a 32-node configuration. The figure provides the overall runtime broken into: GPGPU time (kernel time and host-device transfer time), CPU time, and MPI communication time for a 32-node configuration versus the test image size. The figure highlights

that the application is largely communication bound for a 32-node configuration due to the expensive scatter and gather operations. While the MPI communication time dominates the overall runtime for all of the test image sizes, for small test image sizes, the GPGPU time is insignificant due to a small number of computations per GPGPU device. As the test image size increases, the GPGPU time increases due to increased computations per GPGPU device, providing marginal improvement in scaling efficiency. Nonetheless, the dominating MPI communication overhead results in low scaling efficiency for the 32-node configuration.

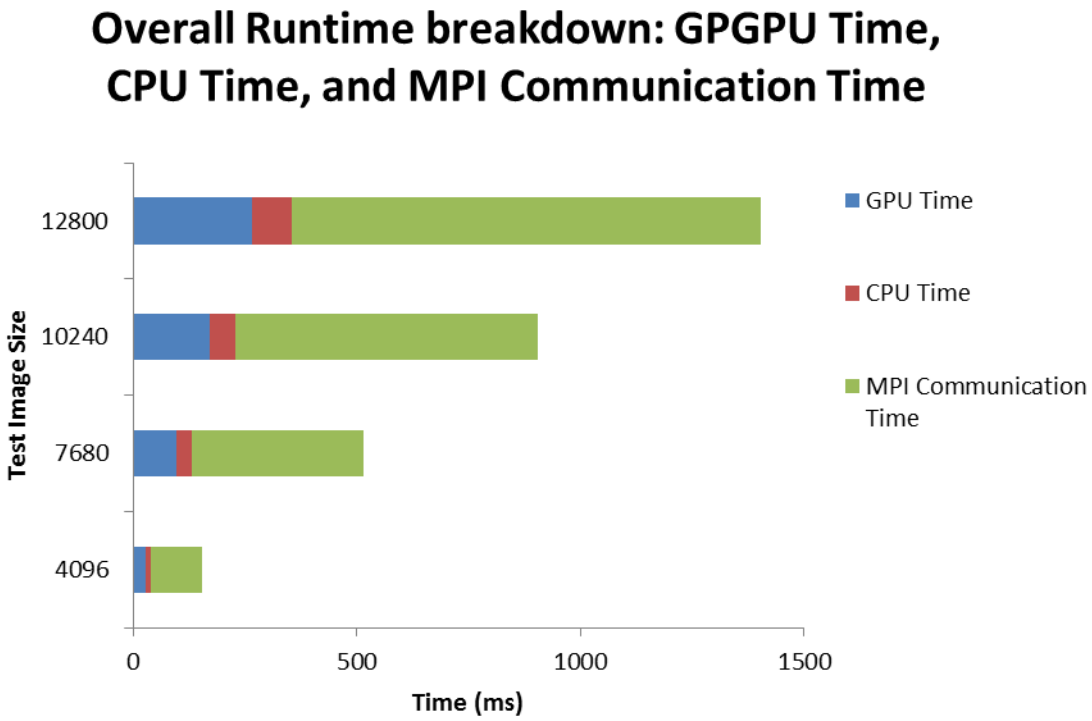

Figure 4.9 Overall Runtime Breakdown for 32-node Configuration

Figure 4.10 provides the overall runtime breakdown for a 4-node configuration. In this case, unlike the 32-node configuration, the CPU and GPGPU computation times dominate the overall runtime. Since the amount of computation generally scales well with the number of processors, dominant CPU-GPGPU computation times are highly amenable to strong-scaling behavior as shown in Table 4.11. Unlike computation, the amount of MPI communication scales differently

and depends on the application [6]. Since the GPGPU and CPU computation times significantly influence the overall runtime, high scaling efficiency values are observed for the 4-node configuration.

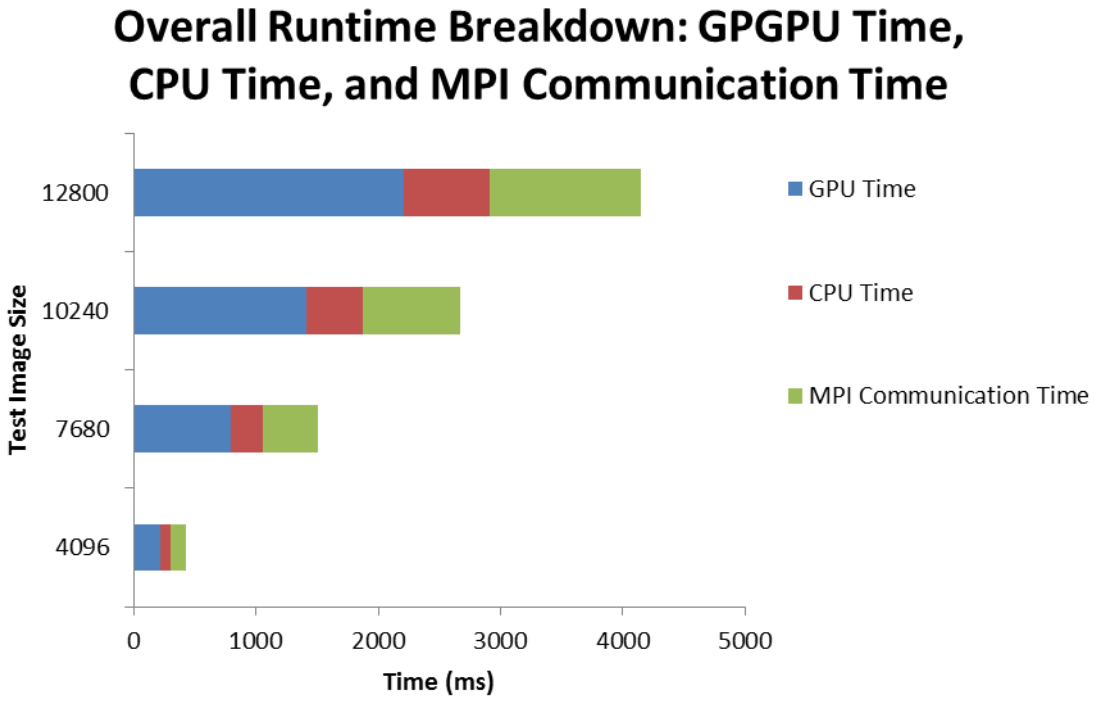

Figure 4.10 Overall Runtime Breakdown for 4-node Configuration

Table 4.12 provides the speed-up values for all node configurations versus the test image size. As seen in the table, the 32-node configuration achieves a speed-up of 11.5x over the equivalent MPI-only implementation. The speed-up values reach a plateau for all node configurations and fall with the node scaling for a given test image size. This behavior confirms the previously provided scalability explanation. With the node scaling, GPGPU and CPU communications do not amortize the dominant MPI communication overhead, which ultimately degrades the overall performance.

Table 4.12 Speed-up Values: Multi-GPGPU Implementation vs. MPI-only Implementation

Node Configuration	Image Size				
	4096x4096	**5120x5120**	**7680x7680**	**10240x10240**	**12800x12800**
1	32.02x	33.80x	32.26x	32.27x	26.73x
2	27.51x	28.01x	27.44x	28.07x	29.14x
4	16.67x	17.04x	17.02x	15.75x	15.36x
8	15.13x	15.91x	16.76x	16.10x	15.10x
16	10.07x	10.80x	11.80x	11.50x	11.50x
32	32.02x	33.80x	32.26x	32.27x	26.73x

In this section, we provided the performance analysis study for the ADF algorithm on the Forge GPGPU cluster. Our implementation of the ADF algorithm was successful in processing images as large as 156 mega-pixels and achieved a speed-up, as high as 29x, over an equivalent MPI-only implementation for the same test image size. The multi-GPGPU implementation demonstrated reasonable scaling behavior with nearly 86% scaling efficiency for a 4-node configuration. The scaling efficiency for all node configurations generally improved with the test image sizes. However, the scaling efficiency dropped with the node (CPU-host/GPGPU-device pair) scaling. Analysis of the application runtime broken down in terms of GPGPU time, CPU time, and MPI communication time for intermediate node configurations demonstrated the dominance of MPI communication overhead in the application runtime for large node configurations. Subsequently, large node configurations observed low scaling efficiency values. Conversely, smaller node configurations observed higher scaling efficiency values since GPGPU and CPU computations dominate the application runtime. The multi-node GPGPU implementation speed-up over the equivalent MPI implementation followed the scaling behavior.

4.6 SUMMARY

In this chapter, we discussed the Forge and Palmetto GPGPU clusters used for the verification of the multi-level performance modeling suite. We also discussed in detail the

mapping methodology and orchestration of the large-scale SNN simulations and ADF algorithm for massive images. The scaling behavior of the SIA case studies was studied to ensure that the implementations used for the preliminary verification of the multi-level suite were optimal. In the next chapter, we describe the development of the *Synchronous Iterative GPGPU Execution* (SIGE) *model* and multi-level performance modeling suite.

CHAPTER 5

SIGE MODEL AND MULTI-LEVEL PERFORMANCE MODELING SUITE

In this chapter, we define and describe the *Synchronous Iterative GPGPU Execution* (SIGE) *model* that serves as the backbone for the multi-level performance modeling suite. The SIGE model describes the execution flow of *synchronous iterative algorithms* (SIAs) on multi-GPGPU systems by providing a set of equations for estimating the total runtime; these equations are evaluated using the multi-level suite. This chapter also provides a prelude to the proposed multi-level performance modeling suite. The individual performance modeling methodologies (low-level and high-level abstractions) are discussed in detail in Chapters 6 and 8, respectively. The rest of this chapter is structured as follows. Section 5.1 describes the SIGE model in detail. The multi-level performance modeling suite is discussed in Section 5.2, where we introduce the low-level and high-level abstraction approaches. The chapter concludes in Section 5.3 with a summary.

5.1 SYNCHRONOUS ITERATIVE GPGPU EXECUTION (SIGE) MODEL

5.1.1 The Definition and Description

Figure 5.1 (a) elucidates the *Synchronous Iterative GPGPU Execution* (SIGE) *model* that serves as the backbone for the proposed multi-level performance modeling suite. The SIGE model describes the execution flow of a special class of deterministic algorithms on multi-GPGPU systems: synchronous iterative algorithms (SIAs). Some examples that fall in the

category of SIAs include: neural network simulations (SNNs), stencil-based image processing (e.g. ADF), 2D Probability Density Function (PDF) estimation [104], and bio-molecular dynamics [105]. Prior to describing the SIGE model operation in detail, we first define the following important terms pertinent to the model: *node*, *network*, *stage*, *synchronous*, and *iterative*.

A *node* in the SIGE model consists of a single CPU-host tightly coupled with a GPGPU device to perform computations and data exchange. The CPU-host/GPGPU-device coupling is referred to as a *host-device* pair and is shown in Figure 5.1 (b). The nodes communicate data and synchronize with each other using the communication medium: *network*. It should be noted that both Infiniband and PCI-Ex bus constitute *communication mediums*; they serve as *channels* to perform data communication.

A *stage* in the SIGE model is a collection of hardware operations pertinent to the algorithm. Some examples that constitute a stage include: inter-node synchronizing data transfers, pre-/post-processing, intra-node computations and communications, etc. A stage is executed by either one node or a combination of nodes.

The *synchronous* property of the SIGE model implies that computations occur *concurrently* on the nodes. The synchronizing inter-node communications occur prior to and after the node computations as shown in Figure 5.1 (a).

The *iterative* property of the SIGE model implies that a single stage or a combination of stages can be repeated multiple times as required by the algorithm.

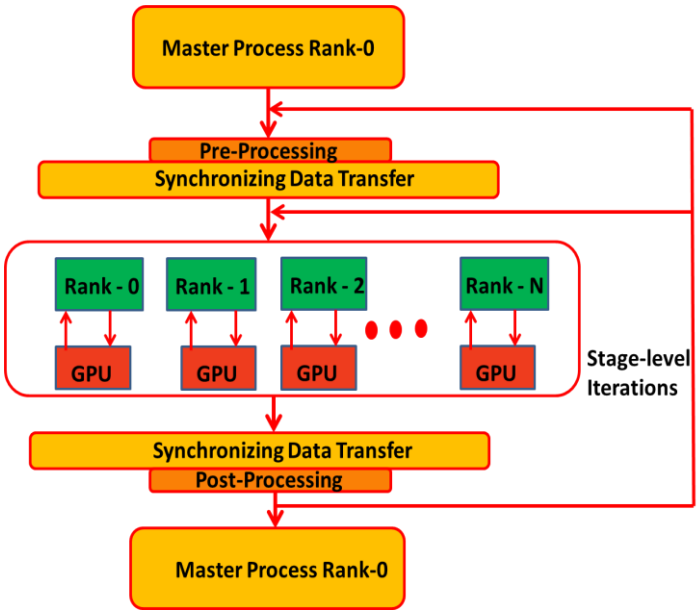

Figure 5.1 (a) SIGE Model

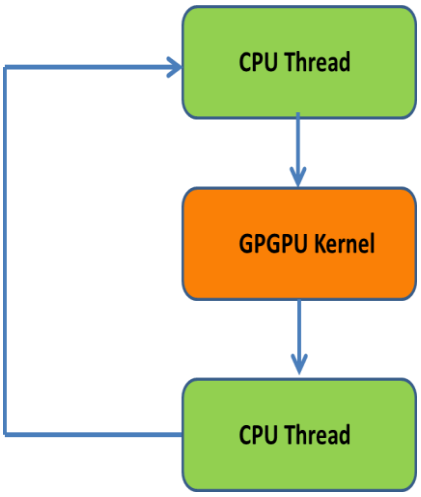

Figure 5.1 (b) 1:1 Host-Device Pairing

In what follows, we describe the SIGE model operation used to develop the multi-level suite. The SIGE model assumes deterministic SIA execution flow, meaning the algorithm behavior is predictable. Unless specified otherwise, the SIAs studied are assumed deterministic. The SIA execution flow begins with the master MPI process rank 0 disseminating tasks to all other MPI processes via a synchronizing data transfer. Once the tasks are distributed, the MPI processes act as independent nodes and perform the assigned computations. At each node, the CPU-host transfers the data to the GPGPU device for computationally intensive operations. The CPU-host performs serial processing operations and waits for the GPGPU device operations to complete. Once the GPGPU device operations are completed, the GPGPU device transfers the data back to the CPU-host. The host-device computations and communications constitute an algorithm stage that can be iterated several times as required by the algorithm. Once the host-device stage finishes, the MPI processes synchronize in the form of data transfer, typically at MPI rank 0, to evaluate the final/partial result or to terminate the SIA with post-processing.

The overall execution time of a deterministic SIA executing on the SIGE model is the summation of runtimes of all the stages. Mathematically, the execution time of a SIA is expressed as shown in Equation 5.1:

$$T_{execution-time} = \sum_{i=1}^{stage-iter.} T_{stage-1} + \sum_{i=1}^{stage-iter.} T_{stage-2} + \ldots + \sum_{i=1}^{stage-iter.} T_{stage-N} \quad (5.1)$$

where, the individual summation terms represent the accumulation of the longest completion times (T_{stage}) for that particular stage over the given stage iterations. Equation 5.1 assumes that none of the stages overlap during the course of SIA execution. However, recent GPGPU architectures allow for concurrent stages including but not limited to asynchronous data transfer(s) from CPU-host(s) to the GPGPU device(s), host-device computation overlap, and host-device communication overlap with the kernel computation. These overlapping stages are accommodated by introducing the *max* function as elucidated by Equation 5.2. The *max* function returns the largest value amongst the parameters in the function's list.

$$T_{execution-time} = \sum \max(\sum T_{stage-a}, \sum T_{stage-b}, \ldots) \quad (5.2)$$

In Equation 5.2, the parameters in the *max* function represent the overlapping SIA stages and the total execution time is equal to the sum of disjoint *max* functions. The overall execution time evaluation involves identification of appropriate stages pertinent to the SIA. As mentioned previously, these stages represent the hardware operations required by the algorithm. In our research, we perform a two-level stage classification for straightforward execution time evaluation: 1) *computation-level* stage that includes pre-/post-processing ($T_{pre-proc.}$ and $T_{post-proc.}$), CPU-host/GPGPU-device computations ($T_{CPU-Host}$ and $T_{GPU-Kernel}$), host-to-device and device-to-host communications (T_{H2D} and T_{D2H}), and 2) *network- or communication-level* transfers that include all of the inter-node network-level transactions (scatter, gather, send-receive, etc. denoted by $\sum T_{Transactions}$) pertaining to the algorithm. Throughout the rest of the dissertation, we

use the terms execution time and runtime interchangeably. Equation 5.3 summarizes the SIGE model:

$$T_{execution-time} = \sum_{i=1}^{computation-iter.} T_{computation} + \sum_{i=1}^{communication-iter.} T_{communication}$$

$$T_{computation} = T_{pre-proc.} + T_{post-proc.} + T_{comp.}$$

$$T_{comp.} = T_{CPU-Host} + T_{GPU} \quad (5.3)$$

$$T_{GPU} = T_{GPU-Kernel} + T_{H2D} + T_{D2H}$$

$$T_{communication} = \sum T_{Transactions}$$

Using the SIGE model explained in this sub-section, we construct the multi-level performance modeling suite to predict the overall execution time of SIAs on multi-GPGPU systems.

5.1.2 SIGE Model Usefulness

Several parallel machine models have been proposed such as the Bulk Synchronous Parallel (BSP) model [29] and Heterogeneous Bulk Synchronous Parallel (HBSP) model [106] that aim to guide the design of applications for optimal performance on a given machine. Unlike these parallel models, the goal of the SIGE model is to generalize the execution flow of deterministic synchronous iterative algorithms (SIAs) on multi-GPGPU systems. Although the SIGE model does not provide guidelines for optimal performance, it is useful for straightforward design space exploration (DSE) via runtime prediction. The SIGE model breaks the application runtime into a number of stages (see Equations 5.1 and 5.2) that are dependent on the SIA studied and the corresponding application mapping. The stages, classified as a computation- or communication-stage, are estimated either using statistical techniques provided by the low-level abstraction or the high-level abstraction models (existing qualitative models, quantitative models, or hybrid

models). The overall runtime breakdown into multiple stages allows the developer to weigh the effects of optimizations on the overall application behavior, enabling a thorough survey of the design space. For instance, optimizing the CUDA kernel (labeled stage-k for instance) may lead to increased host execution time (labeled stage-n) or device-host communications (labeled stage-m). Our framework allows developers to identify such problems and take preventative measures.

Using the SIGE model explained in this sub-section, we construct the multi-level performance modeling suite to predict the overall execution time of the SNN-ADF SIAs on multi-GPGPU systems. In the following sections, we introduce the low-level and high-level abstractions of the modeling suite. As mentioned previously, the low-level abstraction is constituted by the regression-based framework that is broken into two primary components: computation that models the computation-level stage of the SIA and communication that seeks to model the network- or communication-level stage of the SIA. The high-level abstraction uses the qualitative, quantitative, or hybrid approach to evaluate the components of the SIGE model.

5.2 MULTI-LEVEL MODELING SUITE: LOW-LEVEL ABSTRACTION

The *low-level* abstraction of the modeling suite uses limited implementation details and system information for the application runtime prediction. Therefore, partial details of the implementation such as the legacy code, preliminary device kernel, and system specifications must be available. The regression-based analysis best fits the low-level abstraction since it enables the determination of mathematical models that describe the application behavior on the given computing system with a certain degree of confidence [107]. In performance modeling studies, such as the one conducted in this dissertation research, application runtime adequately represents the dependent variable for the statistical regression analysis. Furthermore, to assist with the user-friendly analysis, the application runtime can be further broken into host-device

runtime, host-device data transfer time, and inter-processor data transfer time. Selection of the independent variables depends on analysis of the algorithm. For SIAs such as those mentioned previously, common algorithm parameters that can adequately represent the set of independent variables to characterize the application runtime with a high degree of confidence include but are not limited to the number of floating-point operations (FLOPs) performed, number of bytes required for computation, data transfer size, number of data transactions, and processor count. It is worth reiterating that FLOPS and FLOPs are two distinct parameters; FLOPS (floating-point operations per second) is a measure of computer performance, whereas FLOPs is the number of floating-point operations performed in an algorithm. In addition, one can adjust the independent variable space by adding/removing the parameters based on their statistical significance (contribution to the overall regression model).

The primary goal of the low-level abstraction of the multi-level performance modeling suite is to statistically abstract the system architecture characteristics, thereby enabling performance prediction without detailed knowledge of the underlying computing architecture. The low-level abstraction constituted by the regression-based framework is broken into two components: *computation* and *communication.* The computation component models the CPU-host and GPGPU device computations using algorithm characteristics such as the number of FLOPs and computational bytes as predictor variables. The regression models for the computation component are trained using several small, instrumented executions of an SIA set with a range of computation-to-communication requirements. These instrumented executions are conducted using a set of selected problem sizes (neural network size, image size, etc.) that constitute the *sample* for the regression analysis. For any statistical study, it is imperative to choose a sample large enough to satisfactorily estimate/model the behavior of the entire *population.* In our

research, we choose a set of problem sizes that *adequately fit* on a single GPGPU device as the sample to typify the behavior of the entire population (other problem sizes including those executing on larger GPGPU cluster configurations). The communication component of the regression-based framework is further divided into two sub-components: 1) inter-processor communication over the network (Infiniband) and 2) CPU-host/GPGPU-device (host-device) communication over the PCI-Ex bus. The regression models for the communication component are developed using micro-benchmarks that measure transaction throughput and employ data transfer size and processor count as predictor variables. The sample for the communication component is constituted by a set of representative data transfer sizes (e.g. 8 KB – 128 MB).

We assert that the low-level abstraction is expected to provide fine-grained runtime predictions because the performance models are developed using instrumented executions of the SIA on the chosen system. Consequently, it is a viable approach to DSE where the goal is to identify an optimal implementation from the design space for the target heterogeneous system. We substantiate the above claim in Chapter 7 by verifying the low-level abstraction for accurate runtime prediction and productive GPGPU DSE. In the roadmap for optimal A2A mapping (Appendix A), the low-level abstraction is the last milestone that identifies the best implementation for the target system through DSE.

5.3 MULTI-LEVEL MODELING SUITE: HIGH-LEVEL ABSTRACTION

The high-level abstraction of the performance modeling suite aims to predict the runtime of SIAs on multi-GPGPU systems using minimum implementation details and high-level system specifications. The high-level abstraction *does not* assume existence of significant implementation knowledge and largely relies on the algorithm characteristics (floating-point operations, bytes consumed, number of computational elements, etc.) and system specifications

(device computation bandwidth, PCI-Ex bandwidth, network bandwidth, etc.). The SIGE model described in Section 5.1 is applicable to the high-level abstraction modeling approach where the computation and communication components are estimated either analytically or using micro-benchmarks (or augmented micro-benchmarks). Consequently, the high-level abstraction is broken into two primary components: *Qualitative Modeling* and *Quantitative Modeling.* The qualitative modeling approach uses existing *subjective-analytical models* for device computations, host-device communications, and network-level communications. The subjective-analytical models describe the system using simple mathematical analytic functions, avoiding minute estimation of the large number of parameters pertaining to the system. These analytical models are developed based on those discussed in Chapter 2. The quantitative modeling approach predicts computation and communication performance by measuring hardware-specific events associated with *objective-analytical models* using micro-benchmarks. A hybrid modeling approach is derived using the above two high-level approaches where some of the SIGE model components are estimated analytically, and the remaining components are analyzed quantitatively. We assert that the predictions enabled by the high-level abstraction models are expected to be coarse-grained; accordingly the models are better suited for preliminary performance prediction. As elaborated by the A2A roadmap (Chapter 10), the high-level abstraction is an intermediate milestone that provides an initial insight into the application performance.

Figure 5.2 summarizes the multi-level performance modeling suite and highlights the performance modeling space. Based on the given design goals and the level of knowledge regarding the algorithm and architecture(s), the multi-level performance modeling suite provides

an appropriate modeling strategy from the modeling space that enables straightforward and accurate application runtime prediction.

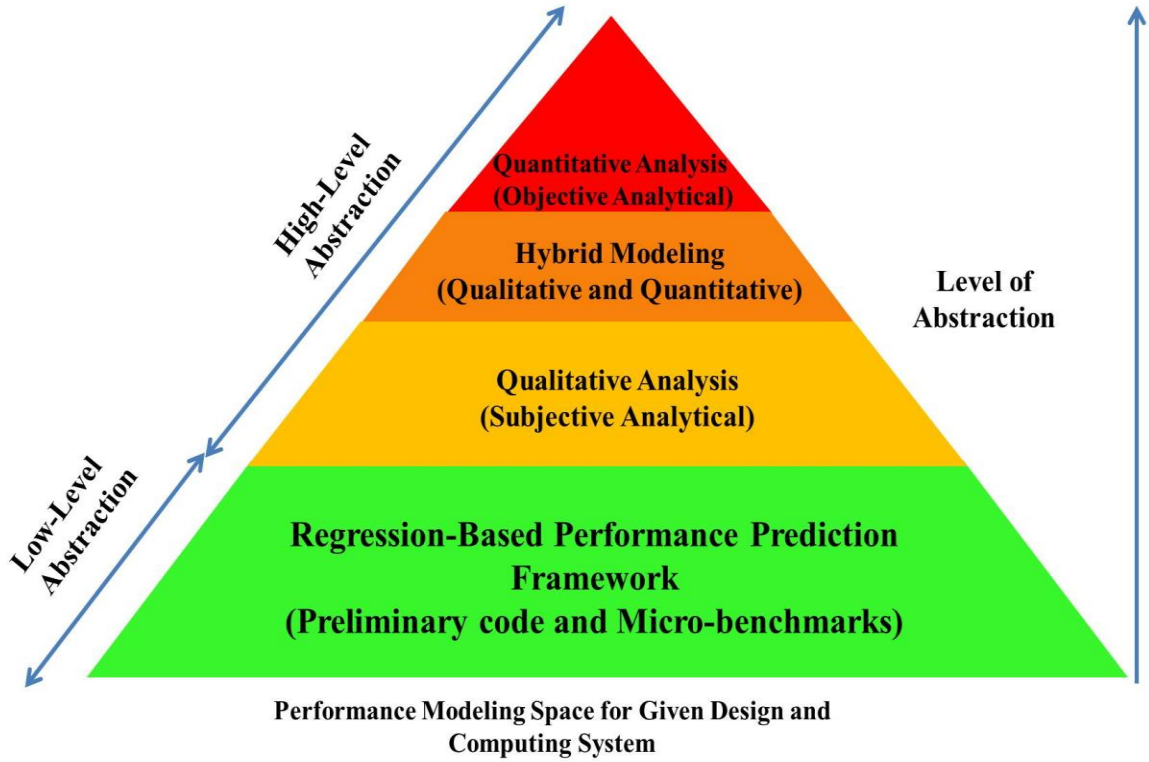

Figure 5.2 The Multi-level Performance Modeling Suite

5.4 SUMMARY

In this chapter, we introduced the multi-level performance prediction modeling suite proposed in the dissertation research. We explained the development of the SIGE model and described the SIA execution flow on the SIGE model. We also provided a prelude to the multi-level performance modeling suite and summarized the performance modeling space in Figure 5.2. The next chapter details the low-level abstraction of the performance modeling suite. We elucidate the development of regression models for the computation and communication components that are ultimately used to estimate the overall SIA execution time (Equations 5.1 -

5.3). It is re-iterated that we follow a bottom-up approach to construct the performance modeling suite (low-level abstraction to high-level abstraction); whereas the A2A roadmap seeks a top-down approach (high-level abstraction to low-level abstraction) for application performance prediction.

CHAPTER 6

THE LOW-LEVEL ABSTRACTION

In the foregoing chapter, we explained the SIGE model that describes the execution flow of SIAs on multi-GPGPU systems. We also provided an overview of the multi-level performance modeling suite that includes two-levels of abstraction: *low-level* and *high-level*. In this chapter, we discuss the low-level abstraction in detail. The low-level abstraction consists of the regression-based framework that is broken into two primary components: *computation* and *communication*. In Section 6.1, we provide a brief background on the multiple regression analysis theory and mention pertinent mathematical terms used throughout the analysis. The low-level abstraction is expounded in Section 6.2 where we construct the regression models for SNN-ADF SIAs. In Section 6.3, we demonstrate the application of the low-level abstraction for GPGPU DSE for the chosen SIAs. Section 6.4 concludes the chapter with a summary.

6.1 MULTIPLE REGRESSION ANALYSIS

Multiple regression analysis is a popular statistical tool used to obtain a relationship between the dependent variable and a set of independent variables with a certain degree of confidence [107 and 108]. Mathematically, the regression analysis is concerned with relating the response, y, with a set of independent variables, x_i. The mathematical literature defines *population* as an entire set of data-points that may be collected for a given problem. The size of the population is usually very large and it is difficult to draw any statistical inference using all of the data-points in that population. Instead, a valid statistical inference is made by selecting a subset of data-points, termed as *sample*, from the population. Multiple regression analysis is concerned with obtaining

a *predictor equation* or *regression model* using a sample that adequately represents the entire population. A multiple regression model can take several mathematical forms, either it can be linear with respect to the independent variables or may involve interaction and higher-order terms. An example multiple regression model is elucidated by Equation 6.1:

$$y = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_1 x_2 + \varepsilon \tag{6.1}$$

where the coefficients α_i represent the estimates of the model parameters, ε represents the error due to the difference between the actual response and the estimated response, and the term $x_1 x_2$ represents the interaction between independent variables x_1 and x_2, respectively. The commonly used model estimation criterion is the *least square method*, which must satisfy two important conditions: 1) The sum of errors must be zero and 2) the sum of the squares of errors is the minimum. Additionally, as described in [107], the error ε must satisfy the following four conditions for reliable prediction. First, the mean of the probability distribution (PD) of ε is zero. Second, the variance of PD is constant irrespective of x_i. Third, the PD of ε is normal and lastly, the errors associated with any two observations are independent.

Once an initial model is constructed, it is important to evaluate the validity of the model. Several criteria exist to evaluate the model's validity, in this research we rely on the R-squared and p-values of the regression model, p-values of the individual estimates, and visual inspection of the residual plots. Typically, a model is considered reliable if the R-squared value is greater than 0.95 and p-values are less than 0.05. A detailed background on the regression theory can be found in [107]. In this research, we use the statistical package *R* [109] to perform all regression analysis.

6.2 LOW-LEVEL ABSTRACTION: REGRESSION-BASED FRAMEWORK

In this section, we describe the regression model construction for the computation and communication components of the regression-based framework using two SIA case studies: large-scale SNN simulations based on the four SNN models and ADF for massive images. These SIA implementations were performed on the NCSA Forge GPGPU cluster, subsequently the regression models developed correspond to this computing system.

6.2.1 The Computation Component

The computation component of the regression-based framework models the CPU-host and GPGPU device computations. The regression model for CPU-host computations is trained using instrumented runs of the legacy code on a small set of processors. This method has been adapted from [6] where the authors develop the regression model for CPU computations using a set of processors to predict the performance of large cluster configurations. In our research, we profile the sections of code executing on the CPU-host and develop the regression model for the CPU execution time, $T_{CPU\text{-}Host}$, with the following predictor variables: the number of processors P, number of floating-point operations *FLOPs*, and the total number of bytes involved in computations, *BYTES*.

The regression model for CPU computations can take several forms depending on the number of FLOPs performed (computation-bound) and bytes accessed (communication-bound). However, for our chosen SIA case studies, the CPU-host computations are predominantly communication-bound (data structure access/ look-up); therefore P and *bytes* constitute the significant parameters in the regression model for CPU computations. The regression models for $T_{CPU\text{-}Host}$ for the four SNN models and ADF are elucidated by equations 6.2 – 6.6. These

regression models were selected based on their high R^2 values (greater than 0.95) and low p-values (less than 0.05).

HH model:

$$T_{CPU\text{-}Host} = -126.35 + 256.72 * GBYTES + 25.016 * P + 12.19 * (GBYTES - 2.55) * (P - 4.6) \quad (6.2)$$

ML model:

$$T_{CPU\text{-}Host} = -147.85 + 1486.12 * GBYTES + 28.8 * P + 51.14 * (GBYTES - 0.581) * (P - 4.57) \quad (6.3)$$

Wilson model:

$$T_{CPU\text{-}Host} = -62.65 + 944.8 * GBYTES + 11.86 * P + 23.56 * (GBYTES - 0.70) * (P - 4.5) \quad (6.4)$$

Izhikevich model:

$$T_{CPU\text{-}Host} = -100.832 + 10240.5 * GBYTES + 18.76 * P + 484.97 * (GBYTES - 0.0581) * (P - 4.48) \quad (6.5)$$

ADF:

$$T_{CPU\text{-}Host} = -36.57 + 4.28 * MBYTES + 5.11 * P + 0.206 * (MBYTES - 27.215) * (P - 7.13) \quad (6.6)$$

Unlike the CPU computations, the GPGPU computations for SNN-ADF SIAs significantly depend on the *FLOPs* and *BYTES* variables, which increase with the problem size (SNN size and image size). First, we describe the derivation of the regression models for the SNN SIAs. To obtain reliable regression models (high R^2 and low p-values) for the SNN SIAs, the four SNN models are grouped either as computation-bound or communication-bound models based on the FLOPs/Byte ratio values in Table 3.1. The regression models are then developed separately for

the computation-bound or communication-bound SNN models. As seen in Table 3.1, the HH and ML models have high FLOPs/Bytes ratio, hence they are grouped as computation-bound models, whereas the Izhikevich and Wilson models have low FLOPs/Byte ratio, consequently they are grouped as communication-bound models. For each of the SNN models, we perform instrumented executions of the GPGPU kernel using several network sizes to construct the regression models. These network sizes adequately fit on a single GPGPU device, hence fittingly constitute the sample for regression testing. The regression models for computation- and communication-bound SNN models are shown in Equations 6.7 and 6.8.

Computation-Bound:

$$T_{GPU-Kernel} = 85.25 + 19.2 * GFLOPs - 177.6 * GBYTES - 0.0028 * (GFLOPs - 363.34) * (GBYTES - 35.9) \quad (6.7)$$

Communication-Bound:

$$T_{GPU-Kernel} = 8.3 - 23.53 * GFLOPs + 42.6 * GBYTES - 0.0133 * (GFLOPs - 13.35) * (GBYTES - 8.54) \quad (6.8)$$

The ML and Wilson models present an interesting situation where both models are moderately computation-bound and communication-bound with moderate FLOPs and bytes requirement as shown in Table 3.1. In addition to the above regression models for computation-bound and communication-bound SNNs, we also develop regression models for the special case of moderately computation- and communication-bound SNN models as shown in Equation 6.9.

Moderately Computation- and Communication-Bound:

$$T_{GPU-Kernel} = 10.083 - 0.275 * GFLOPs + 5.43 * GBYTES \quad (6.9)$$

To demonstrate the cost of constructing the regression models for GPGPU device computations, Table 6.1 shows the GPGPU kernel execution time for selected neural network

sizes from the chosen test sample. As seen in Table 6.1, the execution times are fairly short and easily obtainable based on the system/device availability. The regression models are derived using the sample data fed to a regression engine, *R* [109] for instance.

Table 6.1 GPGPU Kernel Execution Time for SNN Models

Network Size (in millions)	**GPGPU Kernel Execution Time (milliseconds)**			
	HH Model	**ML Model**	**Wilson Model**	**Izhikevich Model**
12.7	2315.31	70.79	183.1	32.6
10.5	1868.85	57.41	148.38	26.56
8.1	1499.56	46.54	119.01	21.67
4.8	934.5	29.29	74.25	13.97
2.88	588.97	18.78	46.71	9.29
0.72	206.1	7.34	16.41	4.23

To obtain the GPGPU computation regression model for the ADF algorithm, we paired the ADF algorithm with the Izhikevich SNN model. Table 6.2 shows the FLOPs-to-Byte and FLOPs/Byte ratio information per data element for the two algorithms. For the ADF-Izhikevich SIA pair, we define FLOPs/Byte ratio as the ratio of the number of floating-point operations performed in the algorithm to the overall bytes requested by the algorithm for computations. As seen in Table 6.2, both Izhikevich SNN and ADF algorithms have similar FLOPs-to-Byte requirements with FLOPs/Byte ratio close to 1, therefore we classify them together as communication-bound algorithms with a common regression model for GPGPU device computations. Similar to the SNN case studies, we perform several small, instrumented executions of the GPGPU kernels for different problem sizes to construct the ADF-Izhikevich GPGPU regression model given by Equation 6.10.

$$T_{GPU\text{-}Kernel} = 2.212 + 490.63 * GFLOPs - 509.7012 * GBYTES + 0.246 * (GFLOPs - 1.53) * (GBYTES - 1.09) \quad (6.10)$$

Table 6.2 FLOPs, Bytes, and FLOPs/Byte ratio per Data Element

Algorithm	FLOPs	Bytes	FLOPs/Byte ratio
Izhikevich SNN	13	13	1.00
ADF	16	12	1.33

6.2.2 The Communication Component

The communication component of the regression-based framework is broken into two sub-components: 1) Inter-node communication over Infiniband and 2) CPU-host/GPGPU-device communication over PCI-Ex bus. Although mentioned here as a part of the communication component, we also include the host-device communications over PCI-Ex bus in the computation stage of the SIGE model for straightforward analysis. First, we develop the regression models for the inter-node communication.

A. Inter-node Communications

The inter-node communication over Infiniband can be comprised of several network-level transactions such as scatter, gather, reduce, etc. We separately model the network-level operations as a function of the message size, *MBYTES* (message size in megabytes) and the number of processors, *P*. We perform micro-benchmarks on the aforementioned network-level transactions using typical data-size range (8 KB - 128 MB) to obtain an initial sketch of the transaction throughput. Figures 6.1 and 6.2 show the scatter and gather throughputs for different node configurations versus the message size. As seen in the same figures, the scatter and gather throughput curves saturate at different levels for different node configurations and resemble the Michaelis-Menten kinetics [68]. The development of a single regression model for transaction throughput with this behavior is non-trivial; therefore we choose to perform a separate regression analysis for the network-level transactions at all node configurations. The equation for the Michaelis-Menten kinetics adapted to model the scatter/gather throughput is:

$$v = \frac{V_{max}[S]}{K_m + [S]} \tag{6.11}$$

where, v represents the reaction rate, V_{max} represents the maximum rate achieved by the system, and K_m represents the substrate concentration where the reaction rate is half of V_{max} [68]. Correspondingly, for the scatter/gather throughput over Infiniband, v and [S] correspond to the scatter/gather throughput and message size in megabytes, respectively. The terms K_m and V_{max} for the scatter/gather throughput, expressed in megabytes and MB/sec respectively, are obtained by performing non-linear regression analysis (using the *nls* command in *R* [109 and 110]) on the training dataset. Table 6.3 provides the K_m and V_{max} values corresponding to the Michaelis-Menten kinetics (Equation 6.11) for the scatter and gather network-level operations. For the reduce operation performed in the SNN multi-GPGPU orchestration, we use the micro-benchmark throughput values, since data size is constant (48 neurons x 4 bytes = 192 bytes) and is reduced at MPI rank 0 irrespective of the neural network size and cluster configuration. The regression models for scatter/gather throughput presented in Table 6.3 have satisfactory R^2 and p-values, making them reliable for prediction.

In Chapter 8, we explain this intuitive mapping of the network-level transaction problem onto the Michaelis-Menten kinetics with a perspective of *subjective-analytical models.*

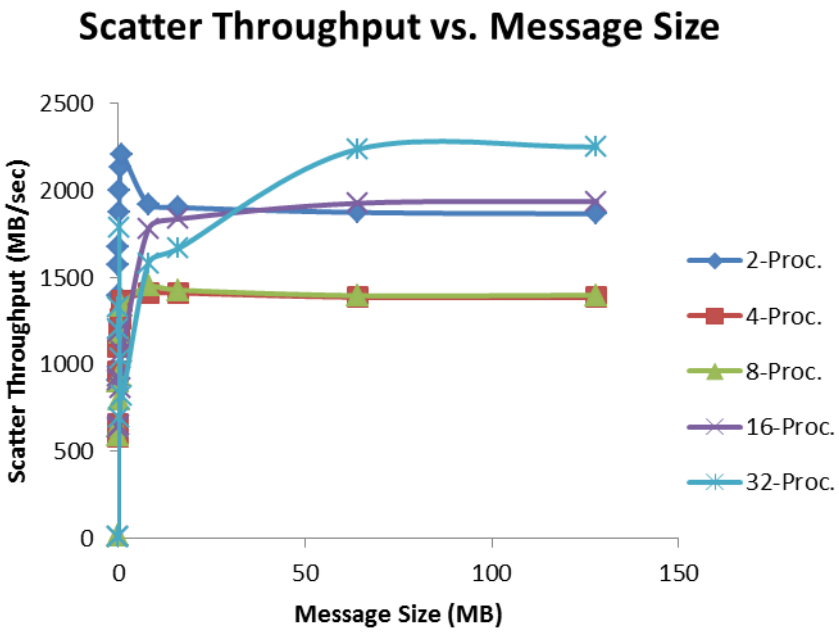

Figure 6.1 Scatter Throughput vs. Message Size

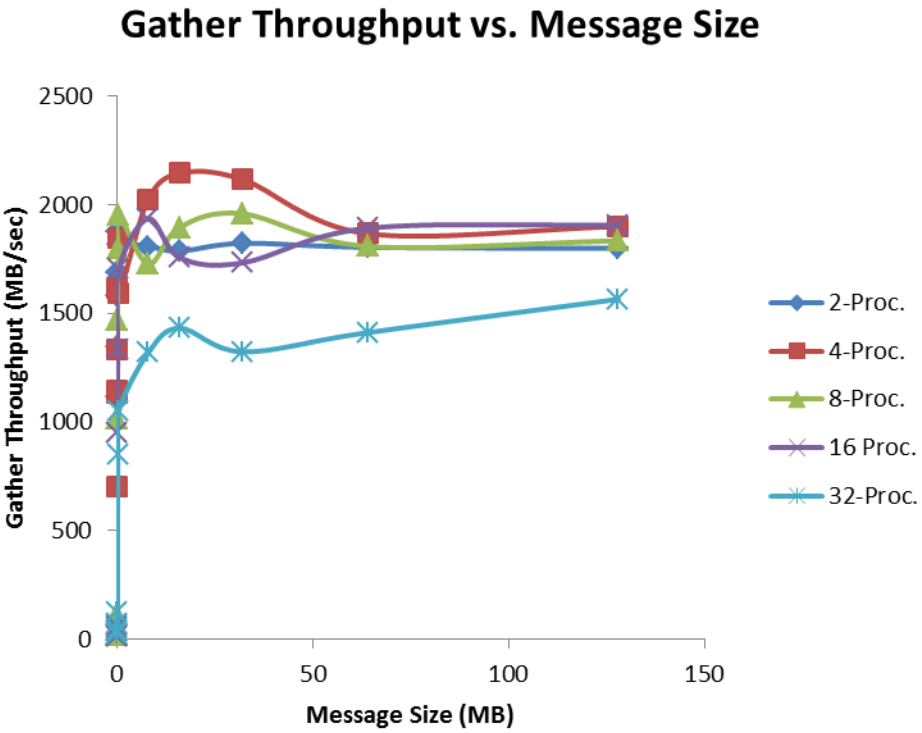

Figure 6.2 Gather Throughput vs. Message Size

Table 6.3 V_{max} (MB/sec) and K_m (MB) for Scatter and Gather Operations

Network Operation	**2 Proc.**		**4 Proc.**		**8 Proc.**		**16 Proc.**		**32 Proc.**	
Scatter	V_{max}	K_m	V_{max}	K_m	V_{max}	K_m	V_{max}	K_m	V_{max}	K_m
	1867	-0.14	1386	-0.03	1399	0.03	1947	0.65	2253	2.42
Gather	V_{max}	K_m	V_{max}	K_m	V_{max}	K_m	V_{max}	K_m	V_{max}	K_m
	1801	-0.06	1953.9	0.43	1788.5	-0.34	1774.5	-0.22	1669.7	-1.4

To obtain the regression models for the *sendrecv* operation, we perform micro-benchmarks on configurations ranging from 4- to 32-nodes. The *sendrecv* times obtained for the 2-node configuration were very short (in fractions of milliseconds) for any reasonable data size compared to the other node configurations; therefore we do not show the regression analysis for the 2-node case. The *sendrecv* micro-benchmark replicates the s*endrecv* communication pattern used in the ADF algorithm for different test image sizes. Figure 6.3 shows the *sendrecv* throughput values versus data exchange size for different node configurations.

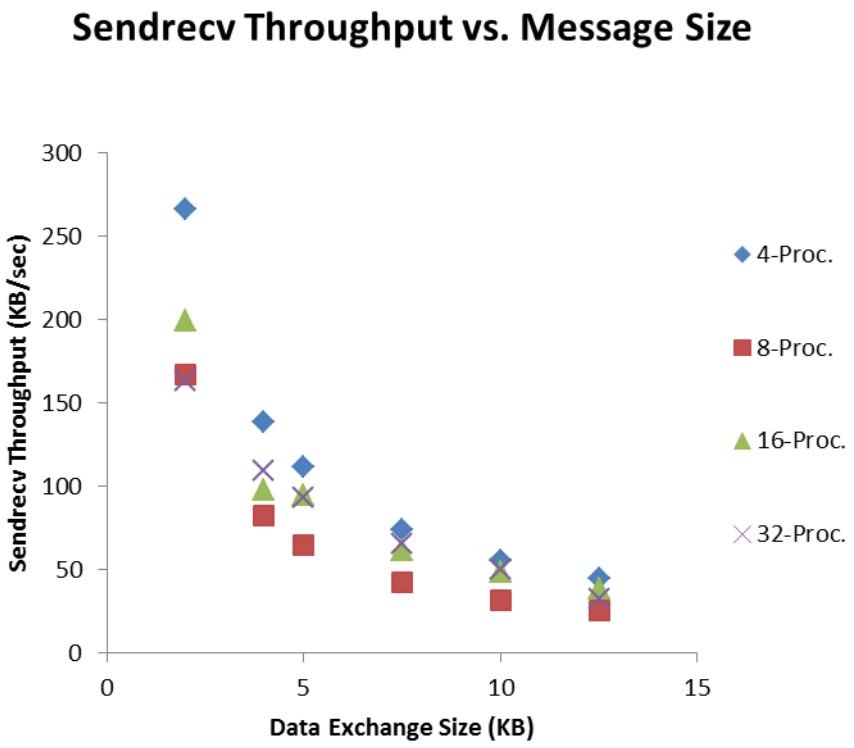

Figure 6.3 Sendrecv Throughput vs. Data Exchange Size

As seen in Figure 6.3, the *sendrecv* throughput exponentially decays with the data exchange size for all of the specified node configurations. A visual inspection of the same figure suggests regression of the logarithm of throughput on the data exchange size to obtain a linear model. Table 6.4 summarizes the regression models for the *sendrecv* operation. In the equations shown below, the *sendrecv* variable corresponds to the *sendrecv* throughput and *Kbytes* represents the data exchange size in KB.

Table 6.4 Regression Models for *sendrecv* Operation in ADF Algorithm

Node Configuration	**Regression Model**
4	$\log(Sendrecv) = 6.98 - 0.039 * Kbytes$
8	$\log(Sendrecv) = 6.90 - 0.049 * Kbytes$
16	$\log(Sendrecv) = 7.01 - 0.045 * Kbytes$
32	$\log(Sendrecv) = 6.86 - 0.035 * Kbytes$

B. PCI-Ex Bus Communications

As discussed in Section 4.1, each server in the Forge GPGPU cluster consists of 6 GPGPU devices interfaced with the NUMA nodes via PCI-Ex bus using PCI-Ex switches (see Figure 4.1). As mentioned in Section 4.2, the MPI ranks are assigned in node packing fashion with 1:1 CPU-host/GPGPU-device ratio at each server. Consequently, at node configurations greater than 4 host-device pairs, up to 6 host-device pairs may be packed in a single server leading to PCI-Ex bus congestion in that server. Therefore, the regression models for PCI-Ex download (host-to-device) and read-back (device-to-host) throughputs are developed for different host-device pairings in a single server.

We perform micro-benchmarks for download and read-back throughputs using typical message sizes (8 KB to 32 MB) for 2, 4, and 6 host-device pairs in a single server. The intermediate host-device pairs (1, 3, and 5 host-device pairs) are not included since our test node configurations are multiples of 2. Figures 6.4 and 6.5 show the download and read-back

throughput curves for different per-server host-device pair configurations. Similar to the Infiniband performance, the PCI-Ex bus performance resembles the Michaelis-Menten kinetics. Also seen in the figures, the throughput values drop with host-device pair scaling, confirming the hypothesis that host-device pair scaling in a server leads to PCI-Ex traffic congestion, leading to reduced download and read-back throughput values. Table 6.5 provides the V_{max} and K_m values corresponding to Equation 6.11 for download and read-back throughput. The *subjective-analytical modeling* perspective of this analysis is elaborated in Chapter 8.

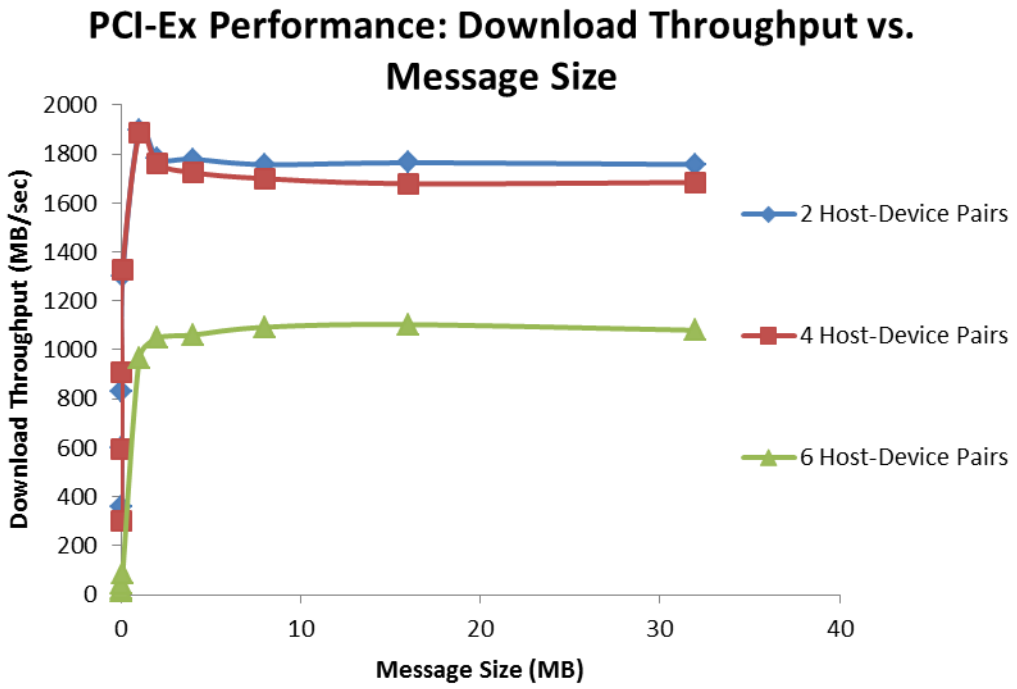

Figure 6.4 Download Throughput vs. Message Size

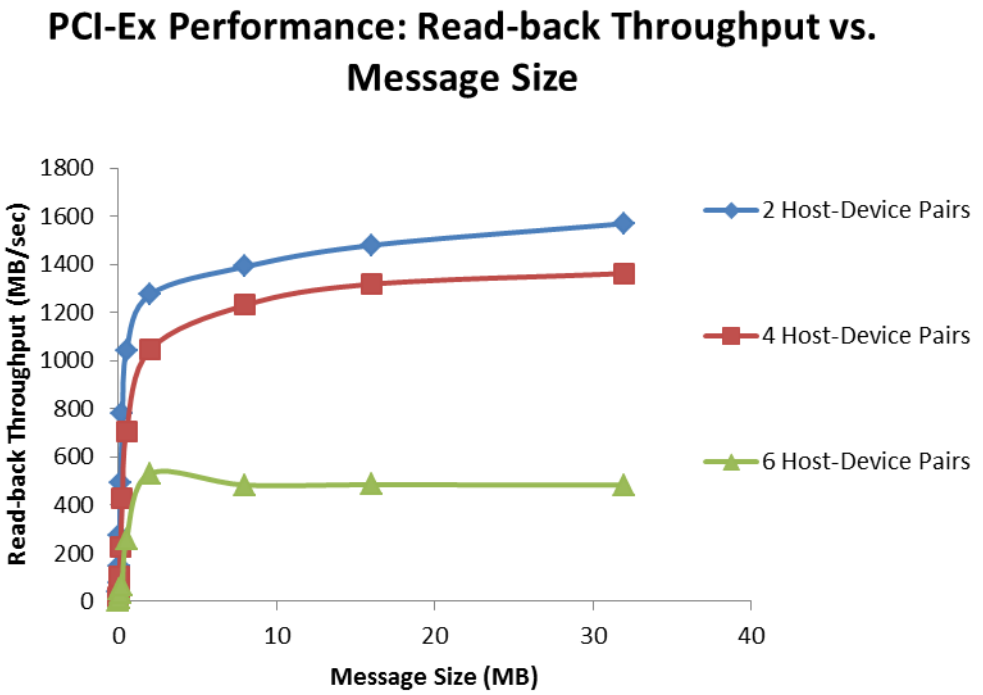

Figure 6.5 Read-back Throughput vs. Message Size

Table 6.5 V_{max} (MB/sec) and K_m (MB) for PCI-Ex Download and Read-back

PCI-Ex Operation	**2 Proc.**		**4 Proc.**		**6 Proc.**	
Download	V_{max}	K_m	V_{max}	K_m	V_{max}	K_m
	1759	0.0012	1682.8	-0.02	1108.9	0.48
Readback	V_{max}	K_m	V_{max}	K_m	V_{max}	K_m
	1567.86	0.43	1385.12	0.7	501.12	0.8

In the foregoing discussion, the Michaelis-Menten kinetics equation was intuitively applied to model the download and read-back operations over the PCI-Ex bus. However, additional mathematical techniques can be employed to fit regression models that may provide higher prediction accuracy. As seen in Figures 6.4 and 6.5, the relationship between the throughput values and message size is highly non-linear, thereby requiring a variable transformation. We apply a logarithm transformation, henceforth log-transformation, on the message size and perform regression of the PCI-Ex throughput on log-transformed message size to obtain a simple linear relation. Table 6.6 provides the regression models for the download and read-back

operations obtained using the log-transformation. These regression models were selected based on high R^2 and low p-values.

Table 6.6 Regression Models for Download and Read-back Throughput (MB/sec)

Host-Device Pair	PCI-Ex Download	PCI-Ex Read-back
2	$Download = 1269.34 + 284.11 * \log(Mbytes)$	$Read\text{-}back = 1022.21 + 218.81 * \log(Mbytes)$
4	$Download = 1021.36 + 255.06 * \log(Mbytes)$	$Read\text{-}back = 794.12 + 193.43 * \log(Mbytes)$
6	$Download = 720.73 + 179.63 * \log(Mbytes)$	$Read\text{-}back = 290.25 + 74.24 * \log(Mbytes)$

In this section, we elucidated the low-level abstraction constituted by the regression-based framework that aims to provide analysis of the following components of the SIGE model: *computation* and *communication.* The computation components were developed using algorithm characteristics such FLOPs and bytes, whereas the communication component regression models were developed with micro-benchmarks of the Infiniband and PCI-Ex bus performance. In addition to intuitively applying the Michaelis-Menten kinetics for PCI-Ex bus performance modeling, the variable transformation technique was also applied to develop alternate regression models. We performed log-transformations on the message size to obtain a simple linear relation between the PCI-Ex throughput (download and read-back) and log-transformed message size. The resulting simple linear models for download and read-back throughputs were accepted based on their high R^2 and low p-values. In simple linear regression analysis, a high R^2 value signifies that the chosen regression model adequately explains the variation of the independent variable with respect to the dependent variable, whereas a low p-value signifies the validity of the simple linear model. Although the variable transformation analysis can be applied for the network-level transactions, our log-transformation analysis for the network-level yielded regression models with low R^2 values, hence not suitable for predictions. We surmise that a larger sample for the network-level can better aide the regression analysis (both Michaelis-Menten and log-

transformation). To justify this claim, Figures 6.6 and 6.7 show the scatter throughput prediction capability of the Michaelis-Menten and log-transformation methods when a large sample is chosen. These figures show the predicted and actual scatter throughput values for an 8-node configuration on the Palmetto multi-GPGPU cluster [15]. As seen in the same figures, the scatter throughput is approximated reasonably by both Michaelis-Menten and log-transformation methods due to the selection of a large sample for analysis. The Michaelis-Menten kinetics better approximates the scatter throughput compared to the log-transformation method given its high R^2 value (0.99 vs. 0.93). In the next chapter, we employ the regression models developed in this section to perform runtime predictions for SNN-ADF SIAs.

The authors assert that these regression-based techniques can be extended to other computing systems as well. In the next section, we present the GPGPU DSE leveraged by the low-level abstraction. This analysis was conducted on the GPGPU-augmented Palmetto cluster with latest Kepler K20 devices.

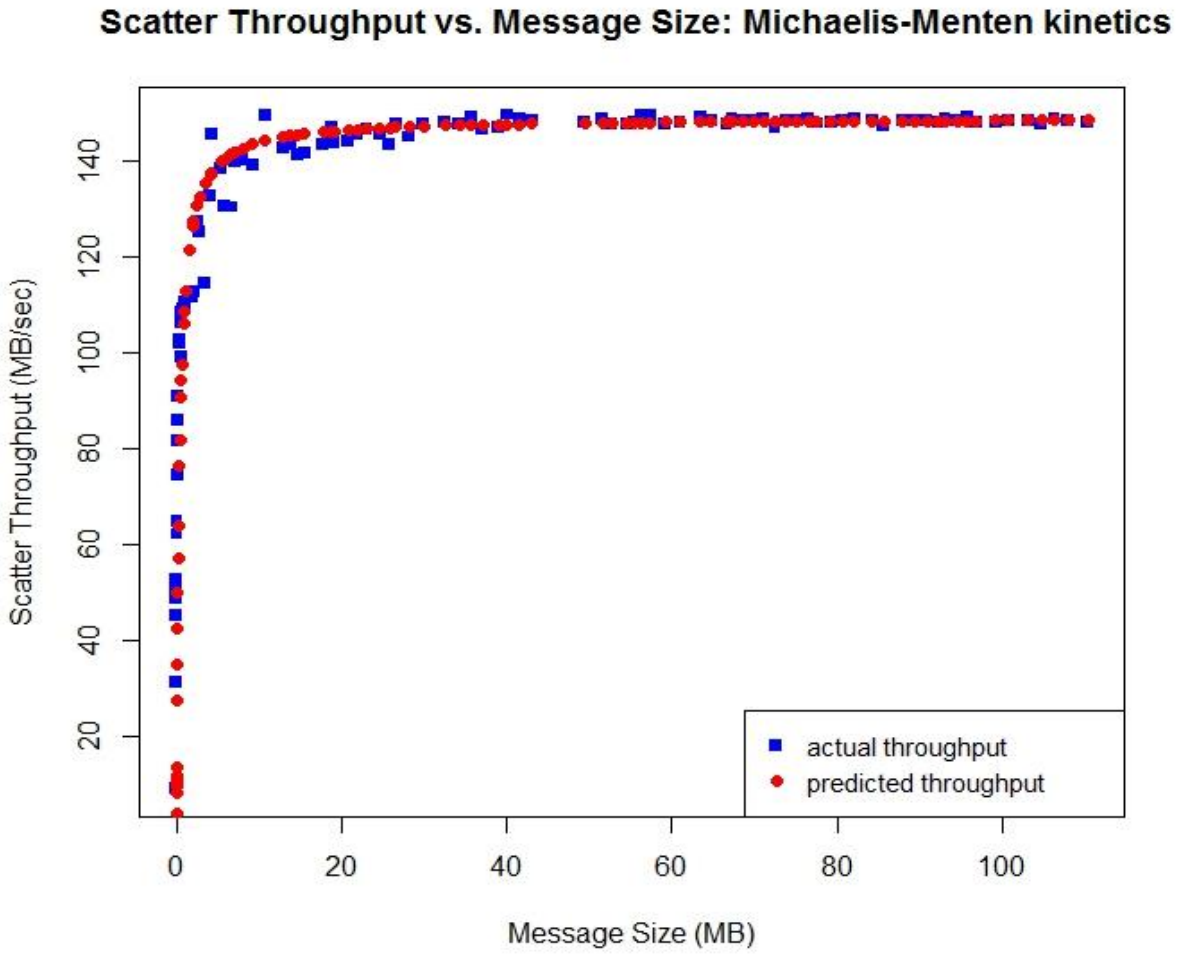

Figure 6.6 Scatter Throughput Prediction for 8-node Configuration using Michaelis-Menten Kinetics

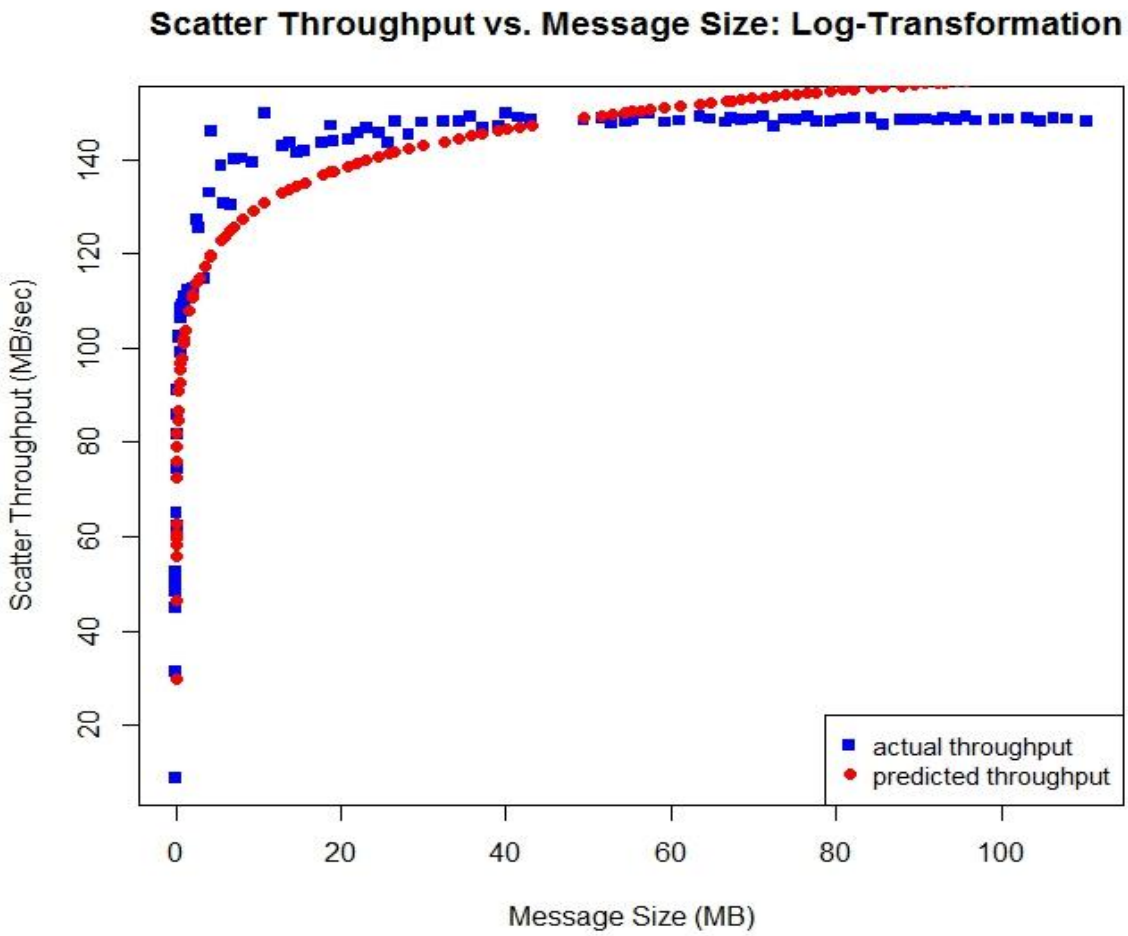

Figure 6.7 Scatter Throughput Prediction for 8-node Configuration using Log-Transformation

6.3 GPGPU DSE USING LOW-LEVEL ABSTRACTION

Design Space Exploration (DSE) studies offer an interesting way to perform application tuning and mapping by exploring several possible implementations (the design space) of an application on the target computing system. The GPGPU DSE aims to analyze the runtime performance of several functionally equivalent implementations of an algorithm, thereby ranking the GPGPU design space. This ranking enables developers to choose the best implementation for optimal algorithm performance on GPGPU-based systems. The GPGPU devices have a

specialized architecture with a memory hierarchy comprising of global, local, shared, constant, and texture memories, each with distinct properties that influence the application performance, thereby requiring prudent use of these memories. An application can employ several plausible optimizations pertaining to the GPGPU memory hierarchy, creating a large design space. As mentioned in Chapter 5, the low-level abstraction (regression-based framework) is anticipated to provide fine-grained runtime predictions, providing a viable approach to GPGPU DSE. Using the regression-based framework, we explore the GPGPU design space featuring optimizations of the GPGPU memory hierarchy for optimal application performance. The regression-based framework models the GPGPU kernel performance using minimum application and accelerator details such as the number of floating-point operations (FLOPs), number of bytes consumed, and parameters pertaining to the GPGPU memory hierarchy including global, texture, and shared memories. Additional algorithm parameters that influence the runtime performance can also be included in the regression analysis. For instance, the number of non-zero rows in a sparse matrix problem can be used as an independent variable for the analysis. The kernel runtime predictor equations are developed with the kernel runtime data collected using several small, instrumented executions of SIAs with a range of computation-to-communication requirements. The kernel runtime predictions for candidate implementations are then compared to ultimately rank the GPGPU design space for a given application. In Section 6.3.1, we discuss the three GPGPU design space implementations for the SNN-ADF SIAs studied in this research. These implementations employ GPGPU-CPU task division identical to the one described in Section 4.2. The GPGPU kernels for implementations differ with respect to the type of memory optimizations employed. These functionally identical implementations are executed on the GPGPU-augmented Palmetto cluster with Kepler K20 devices. The development of regression

equations for evaluating the GPGPU design space is given in Section 6.3.2. The verification of the low-level abstraction for GPGPU DSE is performed in the next chapter.

6.3.1 Design Space Implementations

A. Global Memory

Implementation 1 uses the GPGPU device DRAM (the largest memory), the *global memory*, to store the entire input data pertaining to an application. The GPGPU device fetches the data from the global memory for computations; once all of the computations are finished, the GPGPU device writes the output back to the global memory for reading by the host processor. As the global memory is off-chip memory, frequent accesses result in higher memory latency, thereby impeding the overall application performance. All memory accesses for the SNN and ADF implementations use the global memory. We chose a constant thread block configuration of 256 threads per block to maximize the multiprocessor occupancy for the SNN and ADF implementations using the global memory.

B. Shared Memory

Implementation 2 uses the shared memory, which is an on-chip read/write memory local to a given thread block. All the threads in a thread block have access to the same shared memory, thereby enabling synchronization of the threads within a thread block. Additionally, being an on-chip memory, the use of shared memory reduces the frequent accesses to the off-chip global memory, improving the application performance. For our chosen SIAs, the size of the shared memory depends on the BLOCKSIZE (number of threads in a block). Therefore, to obtain the kernel runtimes using various BLOCKSIZES, we vary the BLOCKSIZE parameter in the kernel from 32 threads to 1024 threads. Additionally, for the SNN models, Implementation 1 is

equivalent to Implementation 2 using a BLOCKSIZE of 256, as they have same number of global memory accesses; whereas for the ADF algorithm, the neighboring pixels in the noised image are fetched from the shared memory, making Implementation 2 distinct from Implementation 1.

C. Texture Memory

For Implementation 3, we use the texture memory designed for high-speed data reading. The texture memory is cached and therefore allows for faster accesses to the data, reducing the frequent high latency accesses to the global memory. The CUDA framework provides techniques for using 1D, 2D, or 3D textures. We use the read-only 1D texture memory to read the level-1 currents for the SNN implementation. For the ADF implementation, we use the read-only 2D texture memory to fetch the neighboring pixels in the noised image.

The next section discusses the low-level design space abstraction where we develop the kernel runtime regression models for these implementations.

6.3.2 Regression-Based Framework for GPGPU DSE

In this section, we explain the regression-based framework for GPGPU design space exploration. The regression-based framework constitutes the low-level abstraction of the design space where partial knowledge of the implementation is present along with the system specifications. We first explain the low-level design space abstraction, followed by the development of regression equations for the three GPGPU design space implementations of the SNN-ADF SIAs.

A. Low-Level Design Space Abstraction

As mentioned previously, the GPGPU design space consists of a specialized memory hierarchy comprising of global, local, shared, constant, and texture memories, each with distinct properties that influence the application performance. Motivated by the modeling concepts developed in [16], we introduce the low-level design space abstraction that aims to statistically encapsulate the characteristics of the aforementioned GPGPU device memories, enabling DSE via kernel runtime prediction using limited implementation details and system information. The regression-based framework, which constitutes the low-level design space abstraction, enables the formulation of mathematical models that assist in the kernel runtime prediction for the given GPGPU architecture with a certain degree of confidence [107]. In this framework, the GPGPU kernel runtime satisfactorily typifies the dependent variable for the regression analysis. The choice of independent variables depends on the algorithm studied and the implementation selected from the design space. For the SIAs used in this research, parameters that can adequately represent the set of independent variables include: the number of floating-point operations (FLOPs), number of bytes required for computation, and memory types employed from the GPGPU device memory hierarchy.

The regression models for GPGPU computations are trained using several instrumented executions of an SIA set with a range of computation-to-communication requirements. To perform the regression analysis, we choose a set of nominal test sizes as samples to characterize the behavior of the entire population that includes larger input sizes. The regression models were selected based on their high R^2 values (greater than 0.95) and low p-values of the regression coefficients and overall model (less than 0.05).

B. Regression Models for Implementation 1

For Implementation 1, we group the four SNN models either as computation-bound or communication-bound SNN models based on the FLOPs/Byte ratio values mentioned in Table 3.1. Therefore, the HH and ML models are grouped as computation-bound models, whereas the Izhikevich and Wilson models are grouped as communication-bound models. Additionally, to obtain the prediction models for algorithms that have FLOPs/Byte ratios between the ML and Wilson models, we present a case where both the models are moderately computation-bound and communication-bound with moderate FLOPs and bytes requirements. The GPGPU kernel regression models are developed separately for the computation-bound, communication-bound, and moderately computation-bound and communication-bound SNN models. These regression models use algorithm characteristics such as the number of floating-point operations, *MFLOPs* (in megaflops) and the number of computational bytes, *MBYTES* (in megabytes) as predictor variables. For each of the SNN models, we perform several instrumented executions of the GPGPU kernel using several network sizes to construct the regression models for the aforementioned bounds. The SNN regression models for all of the aforementioned bounds are shown in Equations 6.12, 6.13, and 6.14.

Computation-Bound:

$$T_{GPU-Kernel} = -4.821375 + 0.008194 * MFLOPs - 0.065055 * MBYTES \quad (6.12)$$

Communication-Bound:

$$T_{GPU-Kernel} = 2.2410263 - 0.0405150 * MFLOPs + 0.0678999 * MBYTES \quad (6.13)$$

Moderately Computation- and Communication-Bound:

$$T_{GPU-Kernel} = 3.449 - 3.649e-04 * MFLOPs + 6.669e-03 * MBYTES \quad (6.14)$$

We now explain the development of the GPGPU kernel runtime regression model for the ADF algorithm. Table 6.2 shows the FLOPs, Bytes, and FLOPs/Byte ratio information per data element for the ADF algorithm and the Izhikevich SNN model. As seen in Table 6.2, both the Izhikevich SNN and ADF algorithms have similar FLOPs-to-Byte requirements with FLOPs/Byte ratio close to 1, therefore we group them together as communication-bound algorithms with a common regression model for the GPGPU device computations, given by Equation 6.15.

Communication-Bound (ADF and Izhikevich):

$$T_{GPU-Kernel} = -5.304158 + 0.126048 * MFLOPs - 0.107107 * MBYTES \quad (6.15)$$

C. Regression Models for Implementation 2

As mentioned in Section 6.3.1, shared memory utilizes locality to reduce the frequent accesses to the global memory. As shared memory is allocated per thread block and all threads in the block have access to the same shared memory, we consider the hardware parameter *BLOCKSIZE* (number of threads in a thread block), as one of the independent variables for developing the GPGPU kernel runtime regression model, in addition to the parameters *MFLOPs* and *MBYTES*. Due to hardware constraints on the algorithm correctness, the SNN implementations were limited to *BLOCKSIZES*: 128, 256, and 512. Consequently, we define two indicator variables, *A* and *B*, to index the above *BLOCKSIZES* and analyze each of the four SNN models individually. The indicator variables are commonly used to incorporate the categorical

effects of independent variables in the regression analysis [107]. The indexing of *BLOCKSIZES* is elucidated as:

BLOCKSIZE 128: A=1, B=0
BLOCKSIZE 256: A=0, B=1
BLOCKSIZE 512: A=0, B=0

The regression models for the four SNN models are shown in Equations 6.16, 6.17, 6.18, and 6.19. It should be noted that *MBYTES* is not included in the regression models due to its weak statistical significance. Unlike the SNN models, the shared memory implementation of the ADF algorithm was not limited by the choice of *BLOCKSIZE*. Consequently for the ADF Implementation 2, we consider the *BLOCKSIZE* parameter as a quantitative variable along with *MFLOPs* and *MBYTES* for developing the regression model given in Equation 6.20. These regression models statistically capture the effects of shared memory usage on the GPGPU kernel runtime, in addition to the *FLOPs* performed and *BYTES* consumed by the GPGPU kernel.

HH:

$$T_{GPU-Kernel} = -129 + 0.001796 * MFLOPs + 120.3 * A + 107.2 * B \quad (6.16)$$

ML:

$$T_{GPU-Kernel} = 2.502 + 0.0004477 * MFLOPs - 0.1879 * A - 0.2645 * B \quad (6.17)$$

Wilson:

$$T_{GPU-Kernel} = 4.320 + 0.003955 * MFLOPs + 0.02133 * A + 0.2126 * B \quad (6.18)$$

Izhikevich:

$$T_{GPU-Kernel} = 1.800584 + 0.0287466 * MFLOPs - 0.9955 * A - 0.567 * B \quad (6.19)$$

ADF:

$$T_{GPU-Kernel} = -244.25560 + 208.47496 * MFLOPs + 580.82102 * MBYTES - 0.15345 * BLOCKSIZE \quad (6.20)$$

D. Regression Models for Implementation 3

Texture memory is a fast, read-only cache between the GPGPU Streaming Multiprocessors (SMPs) and device memory that provides high bandwidth by reducing memory requests to the off-chip global memory. The four SNN models represent a wide-range of computation requirements; therefore the amount of texture memory and global memory accessed varies for each of the four SNN models. Unlike Implementation 1, we model the kernel runtime of the four SNN models algorithm individually. The kernel runtime regression models for the four SNN models observed significant collinearity between the predictor variables: global memory (*GLOBAL*) and the texture memory (*TEXTURE*). To mitigate the collinearity between the predictor variables, we use the texture memory as an indicator variable for developing the kernel runtime regression models. The predictor variables used for the kernel runtime regression models are the number of floating-point operations (*MFLOPs*) and the number of bytes accessed from the global memory (*GLOBAL*) as quantitative variables, and the texture memory (*TEXTURE*) as an indicator variable. The regression models for the SNN models are shown in Equations 6.21, 6.22, 6.23, and 6.24. The texture memory implementation of the ADF algorithm did not observe any collinearity amongst the predictor variables. Consequently, *GLOBAL*, *MFLOPS*, and *TEXTURE* are used as quantitative variables. Equation 6.25 gives the regression equation for the ADF algorithm.

HH:

$$T_{GPU-Kernel} = -57.02 + 7.589e-03 * MFLOPs - 2.383e-01 * GLOBAL - 56.66 * TEXTURE \quad (6.21)$$

ML:

$$T_{GPU-Kernel} = 1.775 + 6.655e-04 * MFLOPs - 7.221e-03 * GLOBAL - 2.138e-01 * TEXTURE \quad (6.22)$$

Wilson:

$$T_{GPU-Kernel} = 3.5580964 + 6.4678e-03 * MFLOPs - 1.48080e-02 * GLOBAL - 3.98392e-02 * TEXTURE \quad (6.23)$$

Izhikevich:

$$T_{GPU-Kernel} = 1.1830696 + 0.0316368 * MFLOPs - 0.0016329 * GLOBAL - 0.0144303 * TEXTURE \quad (6.24)$$

ADF:

$$T_{GPU-Kernel} = 65.90 + 57.08 * MFLOPs - 3415.26 * TEXTURE \quad (6.25)$$

6.4 SUMMARY

In this chapter, we discussed the low-level abstraction of the multi-level performance modeling suite in detail. We explained the development of regression models to estimate the computation and communication components of the SNN-ADF SIAs using the NCSA Forge GPGPU cluster. Profiles of the CPU sections of the parallel algorithm were used to develop the CPU computation regression models. These regression models were constructed using the number of processors (*P*) and data accessed (*BYTES*) as predictor variables. Unlike the CPU-host computations, the GPGPU device computation regression models were developed using the number of floating-point operations (*FLOPs*) and bytes consumed (*BYTES*) as predictor variables. The SIAs were grouped either as computation-bound, communication-bound, or moderately computation- and communication-bound models to obtain reliable predictor

equations. For the communication component of the SIAs, micro-benchmarks were used to train the transaction throughput regression equations. The throughput equations were developed using the Michaelis-Menten kinetics equation and log-transformation method. We also demonstrated the use of low-level abstraction for design space exploration. We discussed three design space implementations of the SNN-ADF SIAs, namely *global memory*, *shared memory*, and *texture memory,* and developed the kernel runtime regression equations for these implementations. The construction of the kernel runtime regression equations included parameters pertaining to the GPGPU device memory hierarchy, in addition to *FLOPs* and *BYTES*. In the next chapter, we employ the regression equations developed in this chapter to verify the low-level abstraction for fine-grained runtime prediction and GPGPU DSE.

CHAPTER 7

VERIFICATION OF THE LOW-LEVEL ABSTRACTION

In this chapter, we present the verification results for the low-level abstraction using all of the SIA case studies employed in this dissertation research. We report error rates for the computation component, communication component, and the overall application runtime. We also verify the use of low-level abstraction for optimal design space exploration. A Strengths, Weaknesses, and Opportunities (SWO) analysis study is also conducted to identify the merits and demerits of the low-level abstraction methodology, identifying avenues for further improvement. The verification of the low-level abstraction for accurate runtime prediction of SIAs on multi-GPGPU systems is provided in Section 7.1. As mentioned in the previous chapters, this analysis is performed on the NCSA Forge GPGPU cluster. Section 7.2 presents the results and analysis of GPGPU DSE using the low-level abstraction. This study was conducted on the GPGPU-augmented Palmetto cluster with Kepler K20 devices. The SWO analysis is performed in Section 7.3 with both the Fermi and Kepler architectures, highlighting the framework's ability to span GPGPU architecture generations. The chapter concludes in Section 7.4 with a summary.

7.1 VERIFICATION RESULTS: SNNs

In this section, we present the verification results for the regression-based framework using the four SNN models. As mentioned in Chapter 5, we scaled the two-level network from 5.7 million neurons to 207 million neurons and varied the node configuration from 2- to 32-nodes. We present the prediction errors for the computation and communication components of the regression-based framework for all of the node configurations using a set of selected SNN

network sizes at each node configuration. First, we discuss the computationally intensive HH model, followed by the ML model, Wilson model, and the Izhikevich model.

7.1.1 HH Model

Table 7.1 shows the total estimated and experimental computation times for the computation component of the node configurations varying from 2- to 32-nodes. As shown in Equation 5.3, the computation time, $T_{comp.}$, is the sum of CPU computation time, $T_{CPU\text{-}Host}$ and GPGPU computation time, T_{GPU}. The GPGPU computation time includes GPGPU kernel time and host-device transfer times as shown by the same equation. In our experiments, we do not account for pre-/post-processing operations since they are only data structure initializations. Consequently, the equation for the computation component takes the form:

$$\sum_{i=1}^{computation-iter.} T_{computation} = \sum_{i=1}^{computation-iter.} T_{comp.} \tag{7.1}$$

Equations 6.2 and 6.7 give the regression models for the computation component of the HH model. As seen in Table 7.1, the computation component regression models provide good prediction results for the tested node configurations and SNN network sizes with maximum error rate of 8.3%.

Table 7.1 HH model: Estimated and Experimental Time Values for Computation Component

Configuration	**Computation Component ($\sum T_{comp.} = \sum T_{CPU\text{-}Host} + \sum T_{GPU}$) (in ms)**					
2-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	4800x4800	1144.16	2393.84	3538	3402.96	-3.96%
	5040x5040	1260.96	2581.88	3842.84	3708.05	-3.64%
4-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	5040x5040	688.62	1467.47	2156.1	2031.89	-6.11%
	7200x7200	1413.27	2629.52	4042.8	4026.05	-0.42%
8-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200x7200	817.27	1541.29	2358.57	2342.29	-0.69%
	9600x9600	1477.86	2484.35	3962.22	4100.45	3.37%
16-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200x7200	482.6	847.75	1330.35	1450.3	8.27%
	9600x9600	920.92	1393.08	2313.99	2455.62	5.76%
32-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	12480 x12480	1085.12	1204.602	2289.72	2165.78	-5.72%
	14400x14400	1503.92	1541.3	3045.21	2883.80	-5.59%

Table 7.2 shows the communication times involved in a single scatter operation and multiple reduction operations. As discussed in Chapter 4, the input image is scattered by the master MPI process rank 0 to all the other MPI processes at the beginning of the algorithm. Once the algorithm begins, at each time-step (472 times-steps for the HH model), the MPI processes synchronize at the master process to accumulate the partial level-2 currents (reduce at MPI rank 0) required for the level-2 neuron dynamics computation. Consequently, the equation for the communication component reduces to:

$$T_{communication} = T_{scatter} + \sum_{i=1}^{communication-iter.} T_{reduce} \qquad (7.2)$$

The regression models for the communication component yield satisfactory results with few outliers for large node configurations. The error rates are approximately 24% for 8-node and 16% for 32-node configurations at the respective largest SNN network sizes. Table 7.3 provides

the estimated runtime, experimental runtime, and the error rate in overall runtime prediction, where the largest error is 8.44%.

Table 7.2 HH model: Estimated and Experimental Time Values for Communication Component

Configuration	**Communication Component ($\sum T_{comm.} = T_{Scatter} + \sum T_{Reduce}$) (in ms)**					
2-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	4800x4800	46.98	1.156	48.14	49.25	2.25%
	5040x5040	51.81	1.156	52.96	53.98	1.87%
4-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	5040x5040	69.88	2.675	72.56	74.207	2.21%
	7200x7200	142.65	2.675	145.33	149.381	2.71%
8-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200x7200	141.3447	11.68	153.028	154.965	1.25%
	9600x9600	251.26	11.68	262.94	347.6	24.35%
16-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200x7200	101.88	9.87	111.76	124.8	10.43%
	9600x9600	180.87	9.87	190.75	214.771	11.2%
32-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	12480 x12480	264.7	14.65	279.35	290.5	3.82%
	14400x14400	352.05	14.65	366.70	439.86	16.63%

Table 7.3 HH model: Estimated Runtime, Experimental Runtime, and Error Rate

Configuration	**$T_{Execution} = \sum T_{computation} + \sum T_{communication}$ (in ms)**			
2-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	4800 x 4800	3586.15	3452.207	-3.87%
	5040 x 5040	3895.81	3762.025	-3.55%
4-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	5040 x 5040	2228.66	2106.1	-5.81%
	7200 x 7200	4188.12	4175.43	-0.30%
8-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	7200 x 7200	2511.6	2497.25	-0.57%
	9600 x 9600	4225.164	4448.044	5.07%
16-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	7200 x 7200	1442.11	1575.1	8.44%
	9600 x 9600	2504.741	2670.4	6.2%
32-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	12480 x 12480	2554.417	2447.751	-4.35%
	14400 x 14400	3397.265	3315.4	-2.46%

7.1.2 ML Model

Tables 7.4 and 7.5 provide the experimental and estimated runtime values for the computation and communication components, respectively. Equations 7.1 and 7.2 apply for the evaluation of computation and communication components. Since the ML model is moderately computation- and communication-bound, we use Equation 6.9 for the GPGPU kernel time estimation. Equation 6.3 applies for the CPU-host computation time estimation. The estimations for the computation component are observed to be generally satisfactory; however the prediction errors are high for 32-node configuration for large SNN network sizes. Although our results achieve high prediction accuracy for the GPGPU time estimation (3-5%), the CPU-host estimation time observed high error rates. The authors attribute the high error rate to variability in the level-1 firing event. The slightly non-deterministic nature of the level-1 firing leads to imprecise CPU-host time estimation. Additionally, unlike the HH model, the ML model is CPU computation-bound as seen in Table 7.4. The regression models for the communication components yield satisfactory results with high prediction accuracy (error rate < 11%) as seen in Table 7.5.

Table 7.6 shows the estimated runtime, experimental runtime, and overall error rate in the runtime prediction. While the error estimates for most of the node configurations are in acceptable ranges, the 32-node configuration observes only about 80% prediction accuracy due to inaccurate CPU-host time predictions as previously explained in this sub-section.

Table 7.4 ML model: Estimated and Experimental Time Values for Computation Component

Configuration	**Computation Component ($\sum T_{comp.} = \sum T_{CPU\text{-}Host} + \sum T_{GPU}$) (in ms)**					
2-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	4800x4800	1657.55	183.6	1841.15	1734.43	-6.15%
	5040x5040	1828.87	200.51	2029.38	1911.42	-6.17%
4-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	5040x5040	975.22	112.05	1087.26	1231.92	-11.74%
	7200x7200	2006.61	209.3	2215.89	2498.84	-11.32%
8-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200x7200	545.64	82.03	627.67	625	-0.42%
	9600x9600	1133.75	146.15	1279.9	1244.67	-2.83%
16-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200x7200	691.82	83.3	775.1	703.88	-10.12%
	9600x9600	1250.68	132.19	1382.86	1246.85	-10.9%
32-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	9600 x9600	849.71	77.44	927.15	742.24	-25%
	12480x12480	1464.56	116.07	1580.63	1250.83	-26.4%

Table 7.5 ML model: Estimated and Experimental Time Values for Communication Component

Configuration	**Communication Component ($\sum T_{comm.} = T_{Scatter} + \sum T_{Reduce}$) (in ms)**					
2-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	4800x4800	46.98	0.27	47.26	48.63	2.81%
	5040x5040	51.81	0.27	52.08	53.04	1.8%
4-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	5040x5040	69.88	0.63	70.52	77.95	9.53%
	7200x7200	142.65	0.63	143.28	153.772	6.82%
8-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200x7200	69.27	2.76	72.02	74.15	2.86%
	9600x9600	141.35	2.76	144.1	144.84	0.51%
16-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200x7200	101.88	2.33	104.22	113.6	8.26%
	9600x9600	180.86	2.33	183.2	200.85	8.78%
32-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	9600x9600	157.06	3.46	160.52	176.33	8.96%
	12480x12480	264.7	3.46	268.15	299.1	10.35%

Table 7.6 ML model: Estimated Runtime, Experimental Runtime, and Error Rate

Configuration	**$T_{Execution} = \sum T_{computation} + \sum T_{communication}$ (in ms)**			
2-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	4800 x 4800	1888.412	1783.06	-5.9%
	5040 x 5040	2081.46	1964.45	-5.96%
4-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	5040 x 5040	1157.78	1309.87	11.6%
	7200 x 7200	2359.17	2652.6	11.06%
8-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	7200 x 7200	1424	1389.5	-2.5%
	9600 x 9600	2528.6	2549.85	-0.83%
16-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	7200 x 7200	879.32	817.5	-7.56%
	9600 x 9600	1566.06	1447.7	-8.17%
32-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	9600x9600	1087.67	918.56	-18.41%
	12480x12480	1848.78	1549.94	-19.28%

7.1.3 Wilson Model

Table 7.7 provides the experimental and estimated times for the computation components. To predict the computation performance, we use Equation 6.4 for the CPU-host computations and Equation 6.8 for the GPGPU kernel time estimation. The prediction error values for most of the test cases are less than 10% as seen in the same table.

Table 7.8 provides the estimated time, experimental time, and prediction error values for the communication component of the regression-based framework. The equations for the communication components are given in Tables 6.3 and 6.6. For the Wilson model, the communication component prediction models yielded slightly higher error values that are between 10-15%. Table 7.9 provides the estimated runtime, experimental runtime, and overall prediction error values for all of the node configurations versus the SNN network size where the maximum error is 12.2%.

Table 7.7 Wilson model: Estimated and Experimental Time Values for Computation Component

Configuration	**Computation Component ($\sum T_{comp.} = \sum T_{CPU\text{-}Host} + \sum T_{GPU}$) (in ms)**					
2-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	4800x4800	1218.6	340.24	1558.84	1444.03	-7.95%
	5040x5040	1343.3	370.8	1714.14	1587.4	-7.98%
4-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	5040x5040	699.3	204.26	903.56	976.46	7.46%
	7200x7200	1434.45	385.08	1819.52	1972.15	7.74%
8-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200x7200	768.45	256.25	1024.71	959.921	-6.74%
	9600x9600	1385.65	430.75	1816.4	1687.07	-7.66%
16-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200x7200	408.36	139.46	547.82	617.165	11.23%
	9600x9600	773.58	235.4	1004.20	1085.95	7.52%
32-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	9600x9600	413.377	126.25	539.63	550.1	1.91%
	12480x12480	790.70	198.57	989.25	899.4	-9.98%

Table 7.8 Wilson model: Estimated and Experimental Time Values for Communication Component

Configuration	**Communication Component ($\sum T_{comm.} = T_{Scatter} + \sum T_{Reduce}$) (in ms)**					
2-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	4800x4800	46.98	0.306	47.29	48.643	2.77%
	5040x5040	51.81	0.306	52.11	52.93	1.53%
4-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	5040x5040	69.88	0.708	70.59	76.76	8.02%
	7200x7200	142.65	0.708	143.36	161.673	11.32%
8-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200x7200	69.27	3.09	72.36	83.31	13.13%
	9600x9600	141.34	3.09	144.44	166.06	13.02%
16-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200x7200	101.88	2.61	104.5	125.56	16.76%
	9600x9600	180.87	2.61	183.5	217.41	15.6%
32-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	9600x9600	157.06	3.88	160.95	181.90	11.52%
	12480x12480	264.7	3.88	268.57	301.86	11.02%

Table 7.9 Wilson model: Estimated Runtime, Experimental Runtime, and Error Rate

Configuration	$\mathbf{T_{Execution} = \sum T_{computation} + \sum T_{communication}}$			
2-Node	**Network Size**	$\mathbf{T_{Execution}}$ **Est.**	$\mathbf{T_{Execution}}$ **Exp.**	**Error (%)**
	4800 x 4800	1606.131	1492.675	-7.6%
	5040 x 5040	1766.26	1640.307	-7.67%
4-Node	**Network Size**	$\mathbf{T_{Execution}}$ **Est.**	$\mathbf{T_{Execution}}$ **Exp.**	**Error (%)**
	5040 x 5040	974.155	1053.214	7.5%
	7200 x 7200	1962.883	2133.82	8.01%
8-Node	**Network Size**	$\mathbf{T_{Execution}}$ **Est.**	$\mathbf{T_{Execution}}$ **Exp.**	**Error (%)**
	7200 x 7200	1169.15	1125.98	-3.83%
	9600 x 9600	2070.75	2057.14	-0.66%
16-Node	**Network Size**	$\mathbf{T_{Execution}}$ **Est.**	$\mathbf{T_{Execution}}$ **Exp.**	**Error (%)**
	7200 x 7200	652.33	741.8	12.17%
	9600 x 9600	1187.69	1303.36	8.87%
32-Node	**Network Size**	$\mathbf{T_{Execution}}$ **Est.**	$\mathbf{T_{Execution}}$ **Exp.**	**Error (%)**
	9600x9600	700.57	732.07	4.3%
	12480x12480	1257.83	1201.27	-6.82%

7.1.4 Izhikevich Model

Tables 7.10 and 7.11 provide the estimated and experimental time values for the computation and communication components, respectively, along with the prediction error values.

The prediction error values for the computation component are high compared to previously studied SNN models. As mentioned in Section 6.2.1, the Wilson and Izhikevich models are communication-bound SNN models; therefore the communication-bound regression model is trained using execution times from both SNN models. However, any deviation produced by the resulting regression model may cause large errors for short execution times. The Izhikevich model, with its nominal FLOPs and bytes requirements (see Table 3.1), has a relatively short execution time and consequently results in high prediction error rates [111] compared to the more complex SNN models with longer execution times. The regression models for the communication component yielded satisfactory results with one outlier (15% error) for the 32-node configuration. Table 7.12 provides the estimated and experimental runtime values along

with the overall prediction error values for all of the node configurations where the maximum error is 14.8%.

Table 7.10 Izhikevich model: Estimated and Experimental Time Values for Computation Component

Configuration	**Computation Component ($\sum T_{comp.} = \sum T_{CPU\text{-}Host} + \sum T_{GPU}$) (in ms)**					
2-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	4800x4800	1073.16	112.31	1185.5	1201.75	1.35%
	5040x5040	1182.48	123.03	1305.52	1296.83	-0.67%
4-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	5040x5040	638.76	65.38	704.14	735.15	4.21%
	7200x7200	1316.41	125.38	1441.8	1491.53	3.33%
8-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200x7200	743.12	66.55	809.7	857.23	5.55%
	9600x9600	1360.1	112.31	1472.32	1557.75	5.5%
16-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200x7200	399.92	58.21	458.13	545.42	16%
	9600x9600	808.52	97.5	906	963.91	6%
32-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	9600x9600	419.65	33.86	453.52	390.46	-16.15%
	14400x14400	1289.55	66.55	1356.10	1162.63	-16.64%

Table 7.11 Izhikevich model: Estimated and Experimental Time Values for Communication Component

Configuration	**Communication Component ($\sum T_{comm.} = T_{Scatter} + \sum T_{Reduce}$) (in ms)**					
2-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	4800x4800	46.98	0.026	47.01	48.58	3.22%
	5040x5040	51.81	0.026	51.83	53.90	3.82%
4-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	5040x5040	69.88	0.06	69.95	77.65	9.92%
	7200x7200	142.65	0.06	142.71	148.66	4%
8-Node	**Network Size**	**$T_{Scatter}$ Est.**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**
	7200x7200	69.27	0.27	69.54	69.93	0.57%
	9600x9600	141.35	0.27	141.61	142.35	0.53%
16-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200x7200	101.89	0.25	102.11	111.85	8.7%
	9600x9600	180.86	0.25	181.1	198.02	8.56%
32-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	9600x9600	157.06	0.33	157.4	176.75	8.93%
	14400x14400	352.05	0.33	352.4	418.327	15.76%

Table 7.12 Izhikevich model: Estimated Runtime, Experimental Runtime, and Error Rate

Configuration	**$T_{Execution} = \sum T_{computation} + \sum T_{communication}$**			
2-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	4800 x 4800	1232.5	1250.3	1.42%
	5040 x 5040	1357.35	1350.32	-0.5%
4-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	5040 x 5040	774.1	812.8	4.76%
	7200 x 7200	1584.5	1640.2	3.4%
8-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	7200 x 7200	951.29	999.59	4.83%
	9600 x 9600	1723.85	1872.995	7.96%
16-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	7200 x 7200	560.24	657.27	14.76%
	9600 x 9600	1087.1	1161.9	6.44%
32-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	9600x9600	610.6	567.2	-7.8%
	14400x14400	1708.16	1580.23	-8.1%

In this section, we presented the preliminary verification results for the regression-based framework (low-level abstraction) using the four SNN models as SIA case studies. The

regression models for the computation and communication components demonstrated high prediction accuracy (over 90%), barring a few test cases. It was observed that the regression models yielded better results for the complex SNN models, HH model for instance, which have longer execution times. The complex SNN models with longer execution times have relatively small deviations from the predicted values compared to the deviations observed for simple SNN models with shorter execution times. Additionally, the regression models for the computation components were generally more accurate compared to the communication component models. One theory to explain these deviations is that additional unaccounted for network characteristics, such as change in the protocol, can affect the network-level transactions and hence the prediction accuracy. Additionally, implicit synchronization in collective operations including scatter and reduce may affect the prediction accuracy. Future work beyond this dissertation research will include exploring these network effects on communication performance and prediction.

7.2 VERIFICATION RESULTS: ADF

As mentioned in Section 6.2.1, we paired the ADF algorithm with the Izhikevich SNN model to obtain a common GPGPU computation regression model, given their similar FLOPs, bytes, and FLOPs/Byte ratio requirements (see Table 6.2). First, we provide the prediction error rates for the Izhikevich SNN model followed by the discussion of the ADF algorithm.

7.2.1 Izhikevich Model

The computation component of the Izhikevich model follows Equation 7.1. While Equation 6.5 applies for the CPU computations, we use the combined GPGPU computation regression model given by Equation 6.10 for the GPGPU device computations. Table 7.13 shows the total estimated and experimental computation times for the computation component with node

configurations varying from 2- to 32-nodes. The lower FLOPs/Byte ratio requirements of the Izhikevich SNN (see Table 6.2) and small number of algorithm iterations (12 vs. 30 in ADF) results in shorter execution times, which ultimately leads to higher prediction errors (small execution time deviations result in larger errors for shorter execution times). Table 7.14 shows the communication times involved in a single scatter operation and multiple reduction operations. As discussed in Chapter 4, the input image is scattered by the master MPI process rank 0 to all the other MPI processes at the beginning of the algorithm. Once the algorithm begins, at each time-step (12 times-steps for the Izhikevich SNN model), the MPI processes synchronize at the master process to accumulate the partial level-2 currents (reduce at MPI rank 0) required for the level-2 neuron dynamics computation. The reduced equation for the communication component is given by Equation 7.2.

Table 7.13 Izhikevich model: Estimated and Experimental Time Values for Computation Component

Configuration	**Computation Component ($\sum T_{comp.} = \sum T_{CPU\text{-}Host} + \sum T_{GPU}$)**					
2-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	4800 x 4800	1073.16	113.97	1187.15	1201.75	1.21
	5040 x 5040	1182.48	125.35	1307.85	1296.83	-0.85
4-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	5040 x 5040	638.76	64.28	703.09	735.15	4.36%
	7200 x 7200	1316.41	127.85	1444.3	1491.53	3.16%
8-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200 x 7200	743.12	66.52	808.73	857.23	5.65%
	9600 x 9600	1360.1	113.96	1474.05	1557.75	5.37%
16-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	7200 x 7200	399.92	55.57	458.65	545.42	16.45%
	9600 x 9600	808.52	96.1	904.7	963.91	6.13%
32-Node	**Network Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	9600 x 9600	419.65	31.05	451.03	390.46	-15.51%
	14400 x 14400	1289.55	65.52	1355.4	1162.63	-16.6%

Table 7.14 Izhikevich model: Estimated and Experimental Time Values for Communication Component

Configuration	**Communication Component ($\sum T_{comm.} = T_{Scatter} + \sum T_{Reduce}$)**					
2-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	4800 x 4800	46.98	0.026	47.01	48.58	3.22%
	5040 x 5040	51.81	0.026	51.83	53.90	3.82%
4-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	5040 x 5040	69.88	0.06	69.95	77.65	9.92%
	7200 x 7200	142.65	0.06	142.71	148.66	4%
8-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200 x 7200	69.27	0.27	69.54	69.93	0.57%
	9600 x 9600	141.35	0.27	141.61	142.35	0.53%
16-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	7200 x 7200	101.89	0.25	102.11	111.85	8.7%
	9600 x 9600	180.86	0.25	181.1	198.02	8.56%
32-Node	**Network Size**	**$T_{Scatter}$ Est.**	**T_{Reduce} Est.**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error in $T_{comm.}$**
	12480 x 12480	157.06	0.33	157.4	176.75	8.93%
	14400 x 14400	352.05	0.33	352.4	418.327	15.76%

Table 7.15 provides the estimated and experimental runtime values along with the overall prediction error values for all of the node configurations for maximum image size tested at that configuration.

Table 7.15 Izhikevich model: Estimated Runtime, Experimental Runtime, and Error Rate (%)

Configuration	**$T_{Execution} = \sum T_{computation} + \sum T_{communication}$**			
2-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	5040 x 5040	1359.67	1350.32	-0.67
4-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	7200 x 7200	1586.95	1640.2	3.23
8-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	9600 x 9600	1725.32	1872.995	7.86
16-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	9600 x 9600	1085.65	1161.9	6.52
32-Node	**Network Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	14400x14400	1707.45	1580.23	-8.1

7.2.2 ADF

As mentioned previously, the test images for ADF were scaled up to 156 mega-pixels and the node configurations varied from 2- to 32-nodes. Equation 7.1 also applies for the ADF computation component since pre-processing only involves image read operations at rank 0. As described in Chapter 4, the network-level operations (scatter, gather, and sendrecv) occur only once in the algorithm. Consequently, the communication component for ADF algorithm reduces to:

$$\sum_{i=1}^{communication-iter.} T_{communication} = T_{scatter} + T_{sendrecv} + T_{gather} \tag{7.3}$$

Tables 7.16 and 7.17 (a-b) provide the experimental and estimated values for the computation and communication components, respectively for selected image sizes. As seen in these tables, the error rates for the predictions are less than 10% for the computation component for several of the test cases, whereas the communication component observes slightly higher error rates, contributing to higher error rates in the overall execution time prediction. Table 7.18 provides the estimated and experimental runtime values along with the prediction error rates.

Table 7.16 ADF: Estimated and Experimental Time Values for Computation Component

Configuration	**Computation Component ($\sum T_{comp.} = \sum T_{CPU\text{-}Host} + \sum T_{GPU}$) (in ms)**					
2-Node	**Image Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	10240 x 10240	969.5	2854.1	3802.5	4069.95	6.57%
	12800 x 12800	1513.5	4547	5954.5	6575.52	9.45%
4-Node	**Image Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	10240 x 10240	546.91	1411.86	1975.38	1901.3	-3.9%
	12800 x 12800	853.7	2226.74	3094.02	2970.6	-4.15%
8-Node	**Image Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	10240 x 10240	334.13	738.8	1194.74	1258.1	5.03%
	12800 x 12800	522.32	1158.17	1871.7	1957.34	4.37%
16-Node	**Image Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	10240 x 10240	224.8	369.4	638.11	618.01	-3.25%
	12800 x 12800	353.8	576.8	990.82	957.21	-3.51%
32-Node	**Image Size**	**$T_{CPU\text{-}Host}$ Est.**	**T_{GPU} Est.**	**$T_{comp.}$ Est.**	**$T_{comp.}$ Exp.**	**Error in $T_{comp.}$**
	10240 x 10240	164.4	185.87	371.52	307.9	17.1%
	12800 x 12800	263.7	289.03	616.35	661.7	6.85%

Table 7.17 (a) ADF: Estimated and Experimental Time Values for Communication Component

2-Node	**Image Size**	**$T_{Scatter}$ Est.**	**T_{Gather} Est.**	**$T_{sendrecv}$ Est.**
	10240 x 10240	428.3	222.02	0
	12800 x 12800	669.24	346.95	0
4-Node	**Image Size**	**$T_{Scatter}$ Est.**	**T_{Gather} Est.**	**$T_{sendrecv}$ Est.**
	10240 x 10240	577.16	204.94	177.17
	12800 x 12800	901.83	320.1	327.1
8-Node	**Image Size**	**$T_{Scatter}$ Est.**	**T_{Gather} Est.**	**$T_{sendrecv}$ Est.**
	10240 x 10240	571.73	223.46	286.35
	12800 x 12800	893.32	349.26	584.24
16-Node	**Image Size**	**$T_{Scatter}$ Est.**	**T_{Gather} Est.**	**$T_{sendrecv}$ Est.**
	10240 x 10240	411.23	225.3	218.6
	12800 x 12800	642.36	352.1	428.5
32-Node	**Image Size**	**$T_{Scatter}$ Est.**	**T_{Gather} Est.**	**$T_{sendrecv}$ Est.**
	10240 x 10240	356.03	234.51	170.2
	12800 x 12800	555.7	366.89	201.93

Table 7.17 (b) ADF: Prediction Error in Communication Component

Node Configuration	**Communication Component ($\sum T_{comm.} = T_{scatter} + T_{gather} + T_{sendrecv}$) (in ms)**			
2-Node	**Image Size**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error (%)**
	10240 x 10240	650.31	732.53	11.22
	12800 x 12800	1016.17	1131.4	10.2
4-Node	**Image Size**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error (%)**
	10240 x 10240	959.28	981.93	2.3
	12800 x 12800	1549.02	1529.32	-1.28
8-Node	**Image Size**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error (%)**
	10240 x 10240	1081.54	1225.37	11.73
	12800 x 12800	1826.83	2118.93	13.8
16-Node	**Image Size**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error (%)**
	10240 x 10240	855.1	907.98	5.8
	12800 x 12800	1422.95	1421.1	-0.133
32-Node	**Image Size**	**$T_{comm.}$ Est.**	**$T_{comm.}$ Exp.**	**Error (%)**
	10240 x 10240	760.77	854.6	10.97
	12800 x 12800	1224.53	1482.4	17.4

Table 7.18 ADF: Estimated Runtime, Experimental Runtime, and Error Rate

Configuration	**$T_{Execution} = \sum T_{computation} + \sum T_{communication}$ (in ms)**			
2-Node	**Image Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	10240 x 10240	4494.9	4802.5	6.4
	12800 x 12800	7111.8	7706.9	7.7
4-Node	**Image Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	10240 x 10240	2935.4	2883.3	-1.8
	12800 x 12800	4657.13	4499.9	-3.5
8-Node	**Image Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	10240 x 10240	2214.96	2483.5	10.8
	12800 x 12800	3598.5	4076.3	11.72
16-Node	**Image Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	10240 x 10240	1497.8	1526	1.85
	12800 x 12800	2419.3	2378.3	-1.7
32-Node	**Image Size**	**$T_{Execution}$ Est.**	**$T_{Execution}$ Exp.**	**Error (%)**
	10240 x 10240	1170.85	1162.55	-0.71
	12800 x 12800	1847.8	2144.04	13.81

In this section, we provided the preliminary verification results for the regression-based framework using the Izhikevich-ADF SIA pair. The regression models for the computation and communication components demonstrated high prediction accuracy (over 90%), discounting a few test cases. It was observed that the regression models for computation yielded better results for the computationally intensive ADF algorithm. The ADF algorithm with its longer execution time observes relatively small deviations from the predicted values compared to the deviations observed for relatively less computationally intensive Izhikevich SNN. The regression models for the computation components were generally more accurate compared to the communication component models, a similar behavior was also observed for the SNN-SIA case studies.

7.3 RESULTS AND ANALYSIS FOR DSE

We present the results and analysis for GPGPU DSE study using the regression-based performance prediction framework. The study was conducted on the GPGPU-augmented Palmetto cluster with Kepler K20 devices. Section 7.3.1 provides the design space exploration results using the SNN models and ADF algorithm.

7.3.1 Design Space Exploration

First, we discuss the kernel runtime values and the prediction error rates for the four SNN models and ADF algorithm to further consolidate the efficacy of the prediction framework and facilitate the DSE analysis. Second, the GPGPU design space for the chosen SIAs is explored using the intermediate SNN network sizes ranging from 3120x3120 to 4800x4800. Similarly, we use the image sizes ranging from 8960x8960 to 10240x10240 for the ADF algorithm.

A. Prediction Results for Implementation 1

Implementation 1 relies on global memory for all of the input data accesses and uses a fixed thread BLOCKSIZE equal to 256. Table 7.19 presents the observed statistical-average kernel runtime values, predicted kernel runtime values, and the prediction error rates obtained using Equations 6.12, 6.13, 6.14, and 6.15 for the four SNN models and ADF algorithm. For the compute-intensive HH model, the regression-based framework predicts the kernel runtime with error rate 7.59% for the largest test data size, with overall prediction error rates less than 10% for all the other test data sizes. The ML, Wilson, and the Izhikevich models observe error rates of 9.27%, 3.2%, and 4.48%, respectively for their largest test input size. The ADF algorithm also observes less than 10% prediction error rate for all of the test input sizes.

Table 7.19 Observed and Predicted Runtime Values (in ms) for Implementation 1

Algorithms	**Test Data Size**	**Observed Time**	**Predicted Time**	**Error Rate (%)**
HH	3360x3360	960.5499	958.4288	0.22
	3840x3840	1361.244	1253.415	7.92
	4800x4800	2184.25	2018.444	7.59
ML	3120x3120	39.04656	35.33421	9.50
	3240x3240	41.37167	37.83408	8.55
	3360x3360	44.56183	40.4283	9.27
Wilson	3120x3120	96.72402	94.97815	1.80
	3240x3240	104.8952	102.1543	2.61
	3360x3360	113.2308	109.6012	3.20
Izhikevich	3120x3120	32.84	32.27	1.73
	3240x3240	39.19	38.17	2.60
	3360x3360	51.48	49.17	4.48
ADF	8960x8960	1804.638	1674.823	7.20
	9728x9728	2078.855	1975.189	4.98
	10240x10240	2218.426	2189.148	1.32

B. Prediction Results for Implementation 2

For Implementation 2, we use the best performing BLOCKSIZE for the four SNN models and the ADF algorithm: 512 for the HH model, 256 for the ML model, and 128 for the Wilson,

Izhikevich, and ADF algorithms, respectively. The observed statistical-average kernel runtime values, predicted kernel runtime values, and prediction error rates are given in Table 7.20. The predicted kernel runtime values are obtained using Equations 6.16 through 6.20. All case studies observe error rates below 10%, barring the HH model where the highest error rate of 11% is observed for an intermediate test data size.

Table 7.20 Observed and Predicted Runtime Values (in ms) for Implementation 2

Algorithms	Test Data Size	Observed Time	Predicted Time	Error Rate (%)
HH	3840x3840	1267.877	1340.383	-5.72
	4080x4080	1467.17	1628.807	-11.02
	4200x4200	2237.872	2166.912	3.17
ML	3120x3120	39.12896	37.36777	4.50
	3240x3240	40.73593	40.12207	1.50
	3360x3360	44.99906	42.9803	4.48
Wilson	3120x3120	96.87	94.311	3.04
	3240x3240	103.89	101.3665	2.62
	3360x3360	110.97	108.6883	2.78
Izhikevich	3120x3120	32.57	32.33	0.73
	3240x3240	38.44	38.52	-0.21
	3360x3360	49.85	50.07	-0.44
ADF	8192x8192	2190.834	2151.443	1.80
	8488x8488	2359.41	2238.716	5.12
	8704x8704	2426.996	2302.402	5.13

C. Prediction Results for Implementation 3

Implementation 3 uses the texture memory as discussed in Section 6.3.2. Table 7.21 presents the observed statistical-average kernel runtime values, predicted kernel runtime values and the error rate obtained using the Equations 6.21 through 6.25 for the SNN-ADF SIAs. The prediction error rates are below 5% for all of the SNN models and less than 11% for the ADF algorithm. The largest data size used to verify the prediction framework for the HH model is 4800x4800 with an observed error rate of 1.61%. For the ML model, the largest data size used to verify the framework is 3360x3360 with error rate 0.2%. The Wilson and Izhikevich models observe error

rates 0.97% and 2.1%, respectively for the largest data size as seen in Table 7.21. Finally, for the ADF algorithm the largest image size used for verification is 10240x10240 with an error rate of 10.78%.

Table 7.21 Observed and Predicted Values for Implementation 3

Algorithms	Test Data Size	Observed Time (ms)	Predicted Time (ms)	Error Rate (%)
HH	3360x3360	1671.249	1602.441	4.11
	3840x3840	2114.976	2127.784	-0.60
	4800x4800	3444.07	3388.607	1.61
ML	3120x3120	41.55399	41.86467	-0.75
	3240x3240	44.51993	45.02456	-1.13
	3360x3360	48.4023	48.30368	0.20
Wilson	3120x3120	105.8983	105.5538	0.32
	3240x3240	114.7781	113.5537	1.06
	3360x3360	123.0514	121.8554	0.97
Izhikevich	3120x3120	32.64	32.72	-0.24
	3240x3240	39.23	38.92	-0.79
	3360x3360	51.58	50.48	2.1
ADF	8960x8960	5115.432	4975.186	2.74
	9728x9728	6306.859	5939.77	5.82
	10240x10240	7432.2	6630.527	10.80

D. Design Space Exploration: Comparing Implementations

Sections 7.3.1.A, 7.3.1.B, and 7.3.1.C provided the kernel runtime values for the three design space implementations. In this sub-section, we first compare the observed kernel runtime values of the implementations in Table 7.22, followed by the predicted kernel runtime values comparison in Table 7.23. We discuss the comparison results for the four SNN models first and then discuss the results for the ADF algorithm.

As mentioned in Section 7.3.1.B, the SNN Implementation 2 employs a BLOCKSIZE of 512 for the HH model, 256 for the ML model, and 128 for the Wilson and Izhikevich SNN models based on the best observed kernel runtime values. Based on the test input sizes given in Table 7.22 and other inspected inputs, the design space Implementations 1 and 2 perform similarly for

the HH, ML, and Wilson models. Implementation 2 however, outperforms the rest in the case of the Izhikevich model. As mentioned in Section 6.3.2, Implementation 1 uses the global memory for all the data accesses. Since the latest GPGPU devices including Tesla M2075 and Kepler K20 have cached global memory, the memory access latencies are reduced, improving performance. Implementation 2 is identical to Implementation 1, except for the choice of thread BLOCKSIZE. For our experiments with SNN SIAs on the Kepler K20, a significant difference in performance was not observed across the BLOCKSIZES. As seen in Table 7.22, for the HH, ML, and Wilson SNN models, the difference in the kernel timing between Implementations 1 and 2 is nominal. Therefore either of the two implementations can be a viable candidate for the GPGPU device. The use of texture memory did not provide performance improvement versus the use of cached global memory as seen in Table 7.22.

Table 7.22 Observed Kernel Runtime Values for Three Design Space Implementations

Algorithms	**Data Size**	**Impl. 1 (ms)**	**Impl. 2 (ms)**	**Impl. 3 (ms)**	**Best Implementation**
HH	3840x3840	1361.244	1267.877	2114.976	Impl. 1 Impl. 2
	4200x4200	1778.395	1467.17	2514.61	
	4800x4800	2184.25	2237.872	3444.07	
ML	3120x3120	39.04656	39.12896	41.55399	Impl. 1 Impl. 2
	3240x3240	41.37167	40.73593	44.51993	
	3360x3360	44.56183	44.99906	48.4023	
Wilson	3120x3120	96.72402	96.87869	105.8983	Impl. 1 Impl. 2
	3240x3240	104.8952	103.8904	114.7781	
	3360x3360	113.2308	110.9705	123.0514	
Izhikevich	3840x3840	32.84272	32.57381	32.63963	Impl. 2
	4200x4200	39.19225	38.44291	39.23024	
	4800x4800	51.48266	49.85226	51.57866	
ADF	7680x7680	1378.762	1980.202	3468.152	Impl. 1
	8192x8192	1581.421	2190.834	4169.742	
	8704x8704	1657.688	2426.996	4964.6	

Table 7.23 shows that the regression-based framework predicts Implementation 1 of the four SNN models as the best candidate for the GPGPU device. Except for the Izhikevich model, the design space ranking matches for all of the other SNN models. Additionally for the Izhikevich

model, the difference in the kernel runtime values of the observed design space Implementation 2 and the predicted design space Implementation 1 is small (less than 3% difference) for the tested data sizes. Therefore, the prediction framework satisfactorily maps the appropriate design space implementations and gives expected prediction results for all of the SNN models.

Unlike the SNN implementations, Implementations 1 and 2 for the ADF algorithm are distinct as they use the global memory and shared memory, respectively for fetching the neighboring pixels in an image. Additionally, we use 2D read-only texture memory for fetching the neighboring pixels for Implementation 3. As seen in Tables 7.22 and 7.23, Implementation 1 decisively outperforms Implementations 2 and 3. Since the Kepler GPGPUs are equipped with global memory cache, accesses to the global memory are optimized.

Table 7.23 Predicted Kernel Runtime Values for Three Design Space Implementations

Algorithms	Test Data Size	Impl. 1 (ms)	Impl. 2 (ms)	Impl. 3 (ms)	Best Implementation
HH	3840x3840	1290.068	1340.383	2127.784	Impl. 1
	4200x4200	1544.241	1628.807	2567.759	
	4800x4800	2018.444	2166.912	3388.607	
ML	3120x3120	35.33421	37.36777	41.86467	Impl. 1
	3240x3240	37.83408	40.12207	45.02456	
	3360x3360	40.4283	42.9803	48.30368	
Wilson	3120x3120	94.97815	94.311	105.5538	Impl. 1
	3240x3240	102.1543	101.365	113.5537	
	3360x3360	109.6012	108.688	121.8554	
Izhikevich	3840x3840	32.27884	32.33651	32.72586	Impl. 1
	4200x4200	38.17493	38.52579	38.9202	
	4800x4800	49.17511	50.07294	50.4768	
ADF	7680x7680	1229.075	2000.484	3558.344	Impl. 1
	8192x8192	1399.145	2151.443	4096.461	
	8704x8704	1580.187	2302.402	4672.737	

In this section, we performed the GPGPU Design Space Exploration (DSE) study to map an optimal implementation to the target GPGPU architecture, promoting high application performance. We explored the GPGPU design space for Synchronous Iterative Algorithms

(SIAs) featuring optimizations of the GPGPU memory hierarchy using a regression-based performance prediction framework. The implementations were ranked based on application runtime predictions that were facilitated by the regression-based framework.

From the design space exploration results based on the observed kernel runtime, we conclude that the global memory implementation performs the best for most of the case studies used in this research. In recent GPGPU architectures such as the Tesla M2075 and Kepler K20, the device global memory is cached, which aids in faster data accesses and promoting performance. The predicted kernel runtime also ranks the global memory implementation as the best implementation for the four SNN models and ADF algorithm. The regression-based framework appropriately ranks the design space implementations for 4 out of 5 case studies, although there is a deviation in the predicted and observed design space ranking for the Izhikevich SNN case study. The difference in the kernel runtime values of the observed design space Implementation 2 and the predicted design space Implementation 1 is small (less than 3%) for the tested data sizes. Therefore, our prediction framework ranks the best design space implementation for an application as expected for 4 out 5 cases and provides acceptable results for the Izhikevich SNN case study. Future work includes extension of the GPGPU design space by including other GPGPU memories such as the local memory and constant memory.

7.4 SWO ANALYSIS OF THE REGRESSION-BASED FRAMEWORK

In this sub-section, we perform the Strengths, Weaknesses, and Opportunities (SWO) analysis of the regression-based framework for multi-GPGPU systems proposed in [16]. This study is conducted on the GPGPU-augmented Palmetto cluster with multiple Tesla M2075 and Kepler K20 devices. The host-device pairs are varied from 2-node up to 16-node configuration. The SWO analysis enables one to study a framework or model, discussing its strengths and

weaknesses for further improvements. To perform the SWO analysis, we provide the predicted overall runtime, observed runtime, and overall error rate for the HH, ML, Wilson, and Izhikevich models in Tables 7.24-7.27 (Fermi) and Tables 7.28-7.31 (Kepler). An in-depth SWO analysis of the regression-based framework can be found in [112 and 113].

Table 7.24 HH Model on Fermi: Observed and Predicted Values for Total Execution Time (ms)

Configuration	**$T_{execution-time}=T_{computation}+T_{communication}$**			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
2-Node	3360x3360	2377.918	2375.387	-0.10
	3600x3600	2722.044	2688.843	-1.23
4-Node	4940x4940	2979.75	2998.08	0.61
	5040x5040	3098.522	3118.508	0.64
8-Node	5200x5200	2163.648	2218.991	2.49
	5280x5280	2227.842	2251.59	1.05
16-Node	5040x5040	1519.719	1518.377	-0.08
	5200x5200	1609.402	1606.104	-0.20

Table 7.25 ML Model on Fermi: Observed and Predicted Values for Total Execution Time (ms)

Configuration	**$T_{execution-time}=T_{computation}+T_{communication}$**			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
2-Node	3360x3360	377.1141	363.7835	-3.66
	3600x3600	432.8445	414.3471	-4.46
4-Node	4800x4800	766.701	803.5798	4.59
	5040x5040	843.8928	879.417	4.04
8-Node	6960x6960	1659.5	1678.35	1.12
	7200x7200	1772.969	1792.154	1.07
16-Node	10080x10080	3463.262	3487.739	0.70
	10120x10120	3490.021	3722.942	6.26

Table 7.26 Wilson Model on Fermi: Observed and Predicted Values for Total Execution Time (ms)

Configuration	$T_{execution-time}=T_{computation}+T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
2-Node	3360x3360	491.8691	478.0189	-2.90
	3600x3600	563.9396	549.4184	-2.64
4-Node	4800x4800	882.9638	914.9215	3.50
	5040x5040	971.7063	1004.55	3.27
8-Node	7140x7140	1870.148	1891.293	1.12
	7200x7200	1900.98	1931.365	1.57
16-Node	9840x9840	3419.897	3444.12	0.70
	10080x10080	3584.07	3621.075	1.02

Table7.27 Izhikevich Model on Fermi: Observed and Predicted Values for Total Execution Time (ms)

Configuration	$T_{execution-time}=T_{computation}+T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
2-Node	3360x3360	315.995	296.9556	-6.41
	3600x3600	362.2265	346.1223	-4.65
4-Node	4940x4940	735.2246	769.96	4.51
	5040x5040	765.0101	807.8265	5.30
8-Node	6960x6960	1564.886	1574.407	0.60
	7200x7200	1673.528	1692.217	1.10
16-Node	10080x10080	3327.271	3338.222	0.33
	10120x10120	3353.455	3362.771	0.28

Table 7.28 HH Model on Kepler: Observed and Predicted Values for Total Execution Time (ms)

Configuration	$T_{execution-time}=T_{computation}+T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
2-Node	3360x3360	975.905	1020.044	-4.52
	3720X3720	1191.36	1250.501	-4.96
4-Node	4200X4200	1723.866	171.609	0.362
	4800X4800	2220.83	2228.33	-0.34
8-Node	5040x5040	1713.93	1696.42	1.02
	6840x6840	2765.74	2753.45	0.44
16-Node	7200x7200	2860.74	2832.107	1
	8400x8400	3911	3786.04	3.2

Table 7.29 ML Model on Kepler: Observed and Predicted Values for Total Execution Time (ms)

Configuration	$T_{execution-time}=T_{computation}+T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
2-Node	5040x5040	1903.7	1555.6	18.3
	5420x5420	2218.85	1791.33	19.3
4-Node	4080x4080	1712.8	1367.98	20.13
	5040x5040	2447.1	2099.95	14.2
8-Node	6840x6840	2227.8	2233.115	-0.24
	7140x7140	2599.8	2435.1	6.31
16-Node	4800x4800	1209.22	1164.033	3.73
	6840x6840	2555.6	2303.842	9.85

Table 7.30 Wilson Model on Kepler: Observed and Predicted Values for Total Execution Time (ms)

Configuration	$T_{execution-time}=T_{computation}+T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
2-Node	3720x3720	614.05	637.99	-3.89
	4800x4800	1032.1	1050.723	-1.81
4-Node	3600x3600	516.7	448.2	13.26
	4080x4080	630.26	573.445	9.01
8-Node	5040x5040	893.67	877.1	1.86
	6840x6840	1640.26	1592.4	2.92
16-Node	6840x6840	2050.63	1797.4	12.35
	7200x7200	1792.633	1978.45	-10.36

Table 7.31 Izhikevich Model on Kepler: Observed and Predicted Values for Total Execution Time (ms)

Configuration	$T_{execution-time}=T_{computation}+T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
2-Node	4480x4480	458.9	453.3	1.22
	4800x4800	528.18	521.42	1.28
4-Node	3600x3600	369.32	365.95	0.91
	4080x4080	530.52	469.422	11.52
8-Node	6840x6840	1441.5	1439.13	0.17
	7200x7200	1589.87	1567.023	1.44
16-Node	6840x6840	1493.1	1694.7	-13.5
	7200x7200	1677.44	1868.31	11.4

Strengths – In [16], the authors proposed the SIGE model for developing the regression-based framework for predicting the runtime of Synchronous Iterative Algorithms (SIAs) on multi-GPGPU systems. The authors used the Forge GPGPU cluster at the National Center for Super-Computing Applications (NCSA) [12], which consists of the Fermi-based Tesla M2070 GPGPUs for implementing the SIAs. For the SWO analysis, we use the Palmetto cluster that has a mix of Fermi-enabled and Kepler-enabled GPGPU nodes; each of these nodes is equipped with two GPGPU devices. From Tables 7.24 – 7.27 for the Fermi GPGPU device, we observe that the prediction framework developed using the SIGE model gives good prediction results with very low error rates. The HH model yields a prediction error rate below 3% for all test data sizes and all node configurations. The ML model provides an overall prediction error rate below 5%. The Wilson model also yields a prediction error rate below 5%. The Izhikevich model gives error rates up to 6.5% for the given test data sizes and all node configurations. Similarly for the Kepler architecture, we observe less than 5% error rates for the HH model (see Table 7.28). The prediction error rates for the ML model (Table 7.29) agree with those mentioned in [16]. Although the framework provides high accuracy for GPGPU time estimation, the CPU-host estimation time observed high error rates due to variability in the level-1 firing event. Both the Wilson and Izhikevich models (Tables 7.30 and 7.31) observe satisfactory prediction error rates that are less than 12%. The high prediction error rates for the low complexity SNN models are attributed to low prediction accuracy in the communication component, which significantly contributes to the overall execution time [16 and 112]. The regression-based framework is deemed satisfactory for runtime prediction for other clusters consisting of other GPGPU architectures, thereby establishing its efficacy to span architecture generations. The regression-based framework enables runtime prediction for SIAs without actual large-scale

implementations; therefore the framework can be used for obtaining runtime values for larger-node configurations and larger data sizes.

The regression-based framework for DSE targets researchers and developers that lack the expertise to use complex analytical models, which require architecture knowledge beyond CUDA programming paradigm. The framework allows for quick and straightforward evaluation of the SIA design space with limited architecture knowledge. We expect the framework to be independent of application regularity. The authors assert that the regression-based framework will also work for other complex algorithms where the algorithm complexity is accounted for by the regression coefficients.

Weaknesses – The regression-based framework is broken into two components: computation and communication. Although this component division provides sufficient insight into the algorithm performance, the behavior of the individual components may vary across computing systems. Albeit the regression-based framework provides satisfactory prediction results for the communication component [112], we observed a few outliers that are attributed to the missing predictor variables in the regression equations, for instance, network protocol changes and implicit synchronization effects. In addition to the above mentioned shortcomings, the regression-based framework requires a preliminary GPGPU kernel implementation; therefore it is imperative to possess knowledge of the sections of algorithm appropriate for implementation on GPGPU devices.

Opportunities – Considering the weaknesses mentioned above, other predictor variables, in addition to the ones used in this research, can be employed to obtain better prediction results. The synchronous iterative model and the regression-based framework should be verified with other accelerators and non-regular algorithms to broaden the scope of performance modeling.

Additional features of the Kepler architecture, dynamic parallelism for instance, should also be explored.

In this section, we performed a SWO analysis study of the regression-based framework for multi-GPGPU systems. In research beyond this dissertation, other predictor variables such as network protocols will be explored for the communication component performance modeling. The synchronous iterative scheme coupled with regression-based framework will also be verified using non-regular algorithms and other accelerators to broaden the scope of performance modeling. The regression-based framework employed for GPGPU DSE constitutes the low-level abstraction of the design space, where partial knowledge of the implementation is present along with system specifications. The next step is to address the high-level abstraction of the design space where the implementation knowledge is less and only high-level system specifications are known. The high-level design space abstraction consists of qualitative, quantitative, and hybrid (mix of qualitative and quantitative approaches) performance modeling approaches. The two levels of design space abstractions will be compared for the ease-of-use and accuracy, allowing the developers to select a suitable DSE method that best satisfies their design goals.

6.5 SUMMARY

In this chapter, we presented the verification results for the low-level abstraction (regression-based framework) of the modeling suite using the four SNN models and ADF algorithm as SIA case studies. The regression models for the computation and communication components demonstrated satisfactory prediction accuracy (less than 10-12%), barring a few test cases. The computation component yielded high prediction accuracy, given the high reproducible nature of the computing devices in general. The communication component (network-level) observed larger errors compared to the computation component. The authors assert that additional network

characteristics such as change in network protocol can affect the network-level transactions and hence the prediction accuracy.

We also performed the GPGPU DSE to map an optimal implementation to the target GPGPU architecture. The design space was explored for SIAs featuring optimizations of the GPGPU memory hierarchy including global, shared, and texture memories. The implementations in the design space were ranked based on the runtime predictions facilitated by the low-level abstraction (regression-based framework). The SWO analysis was conducted that enunciates the strengths and weaknesses of the prediction framework. Additionally, the analysis identifies the scope for further improvement. In the next chapter, we discuss the high-level abstraction of the modeling suite.

CHAPTER 8

THE HIGH-LEVEL ABSTRACTION

In this chapter, we discuss the high-level abstraction that consists of two principal approaches: *Qualitative Modeling* and *Quantitative Modeling.* The former employs *subjective-analytical models* to estimate the computation and communication components of the SIGE model; whereas, the latter predicts these components by measuring hardware events associated with the *objective-analytical models* using micro-benchmarks. The classification of analytical models into subjective and objective categories is explained in this chapter. These two modeling techniques are coupled to yield an intermediate *hybrid approach* where some SIGE model components are estimated qualitatively, while the remaining components are analyzed quantitatively. This analysis is demonstrated in the next chapter. The high-level abstraction study is conducted on the GPGPU-augmented Palmetto cluster with Kepler GPGPU devices. It should be noted that we follow the same CPU computation modeling strategies given by [6 and 9], which resulted in the construction of CPU regression equations in Chapter 6. Therefore, the emphasis is on modeling the GPGPU computations, network-level and PCI-Ex bus communications. Sections 8.1 and 8.2 discuss the qualitative and quantitative modeling approaches, respectively. The chapter concludes with a summary in Section 8.3.

8.1 QUALITATIVE MODELING

In [114], the authors claim that the accuracy of quantitative models largely depends on the precise estimation of several parameters pertaining to the system under investigation. They also assert that the determination of precise parameter values is not always feasible; however it is

usually possible to state some qualitative relations that sufficiently describe the system behavior. Qualitative models avoid numerical complexities by specifying minimum essential qualitative relations amongst the system parameters, thereby providing straightforward insight into the system characteristics. To facilitate qualitative modeling, we study *subjective-analytical models* that describe the system behavior using simple analytical equations. For the heterogeneous systems studied, these analytical models relate the target variables (GPGPU kernel runtime and communication throughput) to algorithm characteristics (computation elements, data size, etc.) and system specifications (computation throughput, peak communication bandwidth, etc.). The following sections illustrate how the *subjective-analytical models* are developed for estimating the SIGE model components.

8.1.1 Qualitative Modeling of GPGPU Computations

We study the *subjective-analytical modeling* for GPGPU computations by adapting the analytical model proposed by Schaa et al. [8], which predicts the application execution time on multi-GPGPU systems using runtime information from a reference GPGPU implementation while varying the number and configuration of GPGPU devices. The authors define *per-element average* ($T_{per\_element\_average}$) as the average time taken by the reference GPGPU device to execute a single computational element (total $N_{elements}$) in the given algorithm. This information is used to extrapolate the algorithm execution time on M GPGPU devices, where M is the number of devices. The *per-element average* evaluation and execution time extrapolation is elucidated by Equations 8.1 and 8.2, respectively.

$$T_{per\_element\_average} = \frac{T_{ref-GPGPU}}{N_{elements}} \tag{8.1}$$

$$T_{M\ -\ GPGPU} = T_{per\ \_\ element\ \_\ average} * \frac{N_{elements}}{M} \tag{8.2}$$

As highlighted in Chapter 2, the performance modeling approach in [8] lacks statistical rigor. Several algorithm parameters, including but not limited to floating-point operations (FLOPs), computational bytes, and the number of computational entities affect the *per-element average* time. To verify this claim, we define *element-throughput* as the number of elements processed by the GPGPU device per unit time (mathematical inverse of *per-element average*). For the chosen SIA case studies, this throughput corresponds to either the number of neurons (SNN models) or pixels (ADF) processed per unit time by the GPGPU device. Figures 8.1 through 8.4 show the non-linear variation of *element-throughput* with respect to the number of elements (SNN network size) for the SNN models using a 4-node configuration. It is worth mentioning that 1- and 2-node configurations yielded substantially different results that do not reflect the application behavior at larger configurations, therefore we chose the 4-node configuration as the reference in this analytical modeling. As seen in these figures, the GPGPU device utilization increases with the SNN network size, thereby resulting in an initial rise of *element-throughput* values. Beyond a threshold SNN network size, the GPGPU device is fully occupied with computations, ultimately leading to *element-throughput* saturation. This observation confirms the claim that *per-element average* should be expressed as a function of algorithm parameters (number of elements in this case) for accurate runtime estimation.

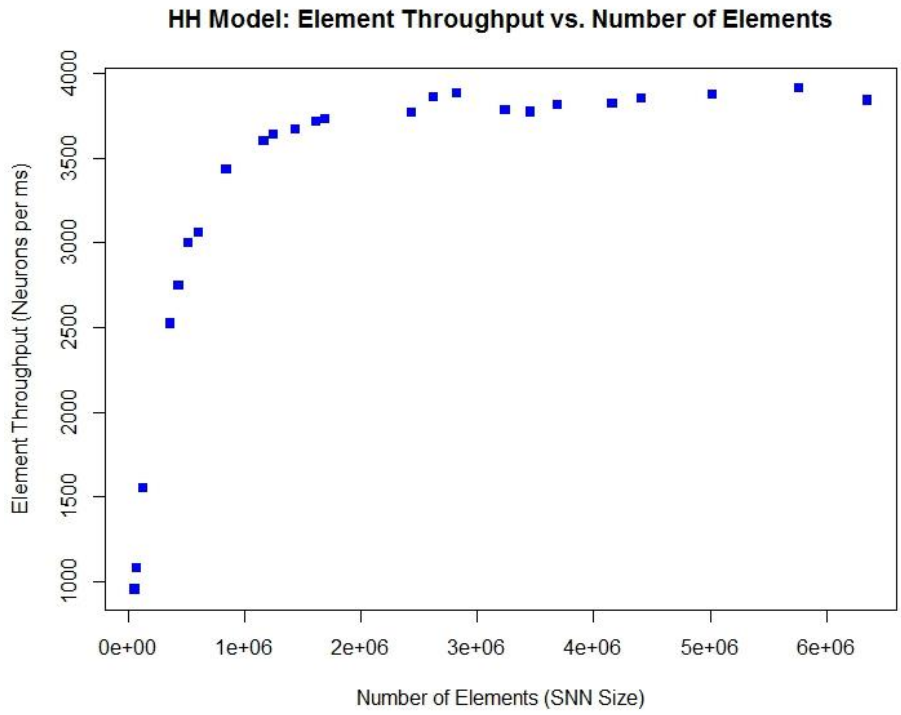

Figure 8.1 HH Model: Element Throughput vs. Number of Elements

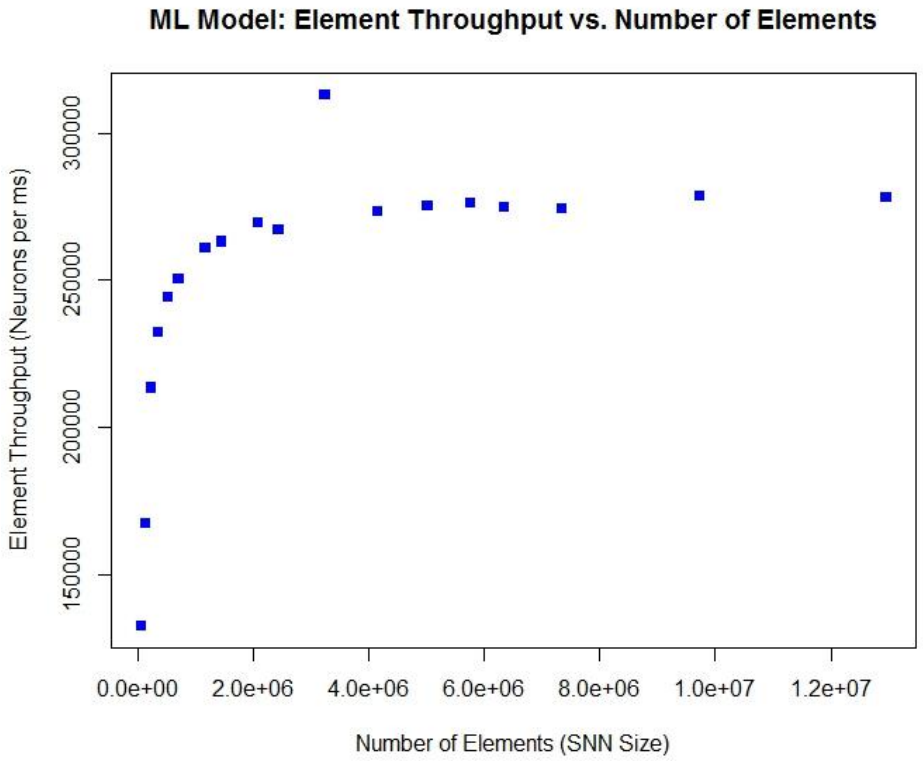

Figure 8.2 ML Model: Element Throughput vs. Number of Elements

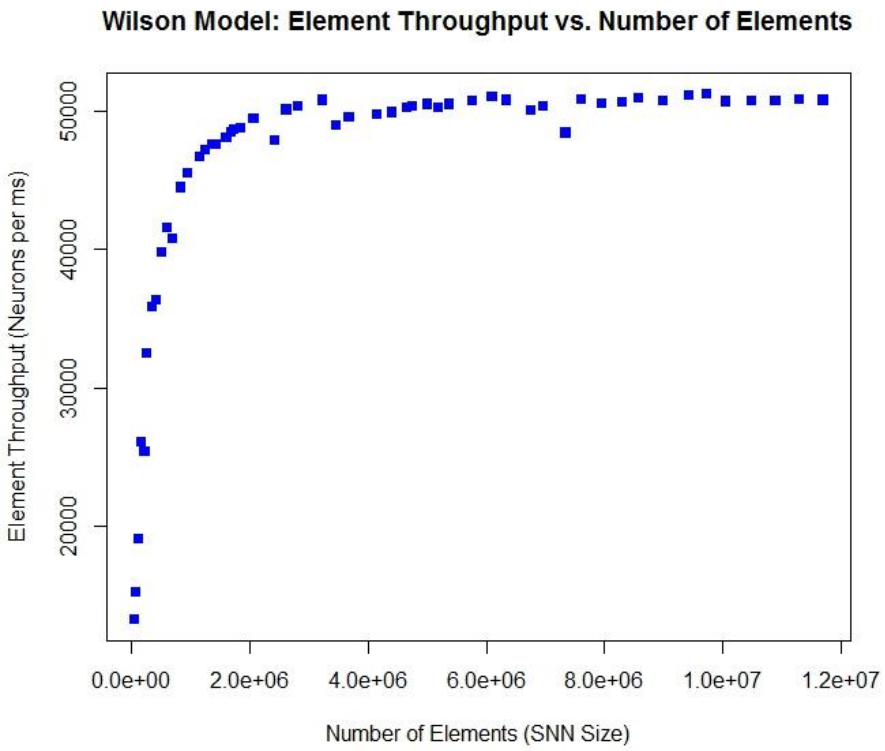

Figure 8.3 Wilson Model: Element Throughput vs. Number of Elements

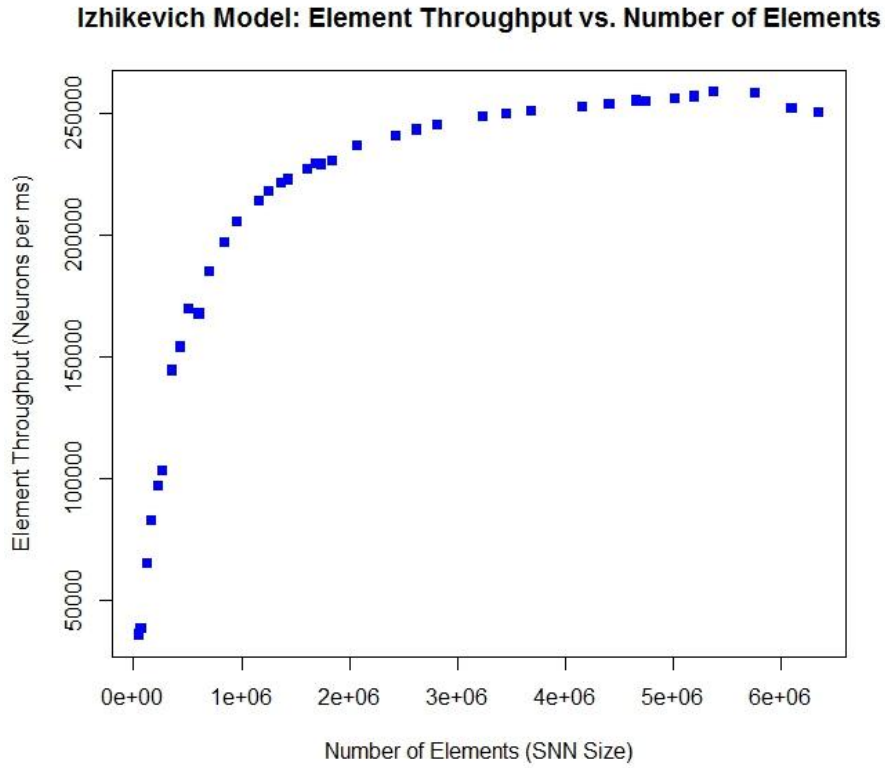

Figure 8.4 Izhikevich Model: Element Throughput vs. Number of Elements

To address the non-linear relationship between *element-throughput* and the number of elements processed by the GPGPU device, we provide a new set of equations for the multi-GPGPU *subjective-analytical model* shown below.

$$Element\_Throughput(N_{elements}) = \frac{N_{elements}}{T_{ref\ -\ GPGPU}} \tag{8.3}$$

$$T_{M\ -\ GPGPU} = \frac{(\frac{N_{elements}}{M})}{Element\_Throughput((\frac{N_{elements}}{M}))} \tag{8.4}$$

The accurate runtime estimation on M GPGPU devices highly depends on the precise description of *element-throughput* as a function of the number of elements ($N_{elements}$). The initial sketches of *element-throughput* (Figures 8.1 – 8.4) strongly resemble the Michaelis-Menten kinetics [68]; therefore, we apply the regression technique developed in Chapter 6 to obtain a relation between *element-throughput* and $N_{elements}$. Equations 8.5 through 8.8 provide the *element-throughput* equations for the HH, ML, Wilson, and the Izhikevich SNN models. The terms K_m and V_{max} expressed in *elements* and *elements/millisecond*, respectively are obtained using non-linear regression analysis explained in Chapter 6.

HH Model:

$$\begin{aligned} element\_throughput &= \frac{3993.23 * N_{elements}}{N_{elements} + 160014.3} \\ V_{max} &= 3993.23\ elements / ms \\ K_m &= 160014.3\ elements \end{aligned} \tag{8.5}$$

ML Model:

$$\begin{aligned} element\_throughput &= \frac{279435.6 * N_{elements}}{N_{elements} + 58294.5} \\ V_{max} &= 279435.6\ elements / ms \\ K_m &= 58294.5\ elements \end{aligned} \tag{8.6}$$

Wilson Model:

$$element\_throughput = \frac{51490.83 * N_{elements}}{N_{elements} + 140027.1}$$
$$V_{max} = 51490.83\, elements / ms \tag{8.7}$$
$$K_m = 140027.1\, elements$$

Izhikevich Model:

$$element\_throughput = \frac{272022.6 * N_{elements}}{N_{elements} + 343499.3}$$
$$V_{max} = 272022.6\, elements / ms \tag{8.8}$$
$$K_m = 343499.3\, elements$$

For the ADF algorithm, we follow the same approach and plot *element-throughput* with respect to the number of elements shown in Figure 8.5. Unlike the SNN models, the throughput sketch initially resembles the Michaelis-Menten kinetics, however after a particular image size, the throughput values drop and saturate to a distinct level. Consequently, the mathematical equation for *element-throughput* takes the form shown by Equation 8.9.

ADF:

$$element - throughput = \frac{14682.8 * N_{elements}}{N_{elements} + 8766.999} * (u(N_{elements}) - u(N_{elements} - 802816))$$
$$+(9.254e-5 * N_{elements} + 8247) * u(N_{elements} - 802816)$$

$$u(N_{elements} - a) = 1 \text{ if } N_{elements} \geq a \tag{8.9}$$
$$= 0 \text{ elsewhere}$$

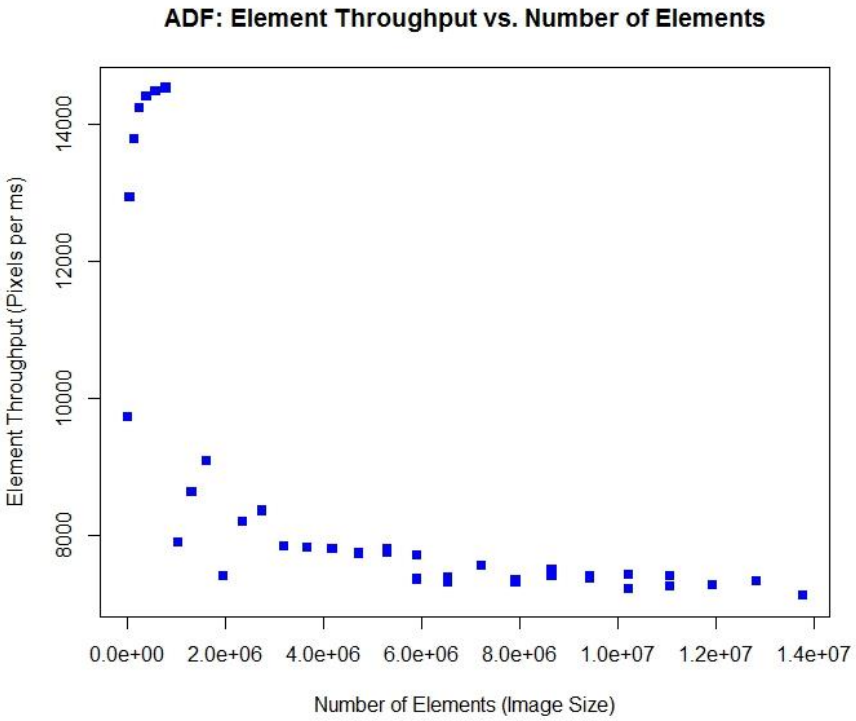

Figure 8.5 ADF: Element Throughput vs. Number of Elements

For the chosen SIAs, we elucidated the multi-GPGPU *subjective-analytical modeling* by relating the kernel execution time on *M* GPGPU devices with *element-throughput* using simple analytical functions. We established that *element-throughput* largely depends on the number of computational entities and provided mathematical equations for *element-throughput* using the regression analysis developed in Chapter 6. This level of modeling avoided complex numerical estimations of architecture specific parameters and facilitated the development of intuitive and simple qualitative relations that explain the application behavior on GPGPU devices. The next section describes the qualitative modeling of communications.

8.1.2 Qualitative Modeling of Communications (Network-level and PCI-Ex Bus)

In Chapter 2, we discussed some of the important network-level modeling techniques including *logP*, *plogP*, and *logGP* models that provide significant insight into the network

characteristics. However, these models require accurate measurement of network parameters such as *latency*, *overhead*, *small* and *large message gaps*; a task that may not be straightforward on all heterogeneous systems. Additionally, the network simulators [64] that estimate these parameters provide overly elaborate numerical output, making the analysis more complex. As discussed in the previous section, *subjective-analytical models* evade numerical complexities by describing the system behavior intuitively. In Chapter 6, we mapped the data transfer problem onto the well-known Michaelis-Menten enzyme kinetics [68], which relates the reaction rate, v, with the substrate concentration, [S], using a first-order equation (see Equation 6.11). Mapping the data transfer problem onto the enzyme kinetics problem is highly intuitive because the data transfer throughput (MB/sec) corresponds to the reaction rate and the data transfer size (MB) corresponds to the substrate concentration, [S]. In Chapters 6 and 7, this qualitative mapping is demonstrated for high prediction accuracy whilst avoiding any complex network parameter estimation. The readers are referred to Sections 6.2.2, 7.1, 7.2, and 7.4 for the qualitative analysis of communication performance.

8.2 QUANTITATIVE MODELING

In the foregoing section, we discussed the *subjective-analytical models* that derive qualitative relations amongst the parameters to represent the system behavior. The quantitative methods also provide an interesting route to performance modeling where the performance/behavior of the target system is estimated by measuring several associated parameters using micro-benchmarks. For instance, one can measure the number of execution cycles involved in computations and DRAM communications to estimate the overall GPGPU kernel execution time [17]. Similarly, the performance of data transfer operations over Infiniband and PCI-Ex bus (henceforth referred to as *communication mediums* or simply *mediums*) can be estimated by measuring *overhead*,

latency, and *message gap* on these *mediums* [61]. These system parameters often constitute the variables of analytical equations, which we refer to as *objective-analytical models.* Formally, the *objective-analytical models* are defined as a class of analytical models that express target variable(s) as function(s) of hardware events estimated using micro-benchmarks. The following sections elucidate the *objective-analytical models* for GPGPU computations and *medium* communications.

8.2.1 Quantitative Modeling of GPGPU Computations

Amongst the several GPGPU analytical models discussed in Chapter 2, the model proposed by Hong and Kim [17] aligns well with our definition of *objective-analytical models*, which we describe in this section. The authors claim that their analytical model is the first for the GPGPU architecture that can also be extended to other multi-threaded architectures. The analytical model estimates the total number of execution cycles in an application by estimating the number of parallel memory requests (*memory warp parallelism*) and computation requests (*computation warp parallelism*). These metrics are evaluated by measuring GPGPU-specific hardware events such as the number of coalesced/uncoalesced accesses, memory access latency, global memory bandwidth, number of memory and computation instructions, and the number of warps (groups of 32 concurrent threads) active on a streaming multiprocessor (SM).

In [17], Hong and Kim assert that active warps execute on SMs in a time sharing fashion; when a warp issues a memory request, the computations from *ready* warps are serviced. This warp-level parallelism is expressed using two metrics: *memory warp parallelism* (*MWP*) and *computation warp parallelism* (*CWP*). The authors define *MWP* as the maximum number of warps that can simultaneously access the memory during the period when a memory request has been issued from a warp. The waiting warp is referred to as the *memory warp* and the waiting

period is labeled as the *memory waiting period. CWP*, a parameter of less significance in this model, is defined as the number of warps that are ready for computations during the *memory waiting period. MWP* is strongly related to the peak DRAM (global) memory bandwidth and number of active warps per SM. The authors express peak *MWP* as shown by Equation 8.10.

$$MWP_{peak} = \frac{Memory\_Bandwidth}{Bandwidth\_per\_warp * Active\_SMs}$$

$$Bandwidth\_per\_warp = \frac{Frequency * load\_bytes\_per\_warp}{Mem\_Latency} \tag{8.10}$$

The variables in this equation are defined as follows.

- *Memory_Bandwidth*: Peak DRAM (global memory) bandwidth
- *Bandwidth_per_warp*: Peak DRAM bandwidth for a single warp
- *Active_SMs*: Number of active SMs in the device
- *Frequency*: Operation frequency of a SM
- *Load_bytes_per_warp*: Number of bytes loaded/stored by the warp
- *Mem_Latency*: The round-trip time to the device DRAM

Unlike MWP_{peak}, $MWP_{not\text{-}peak}$ (*MWP* without the peak device bandwidth) is related to the number of coalesced and uncoalesced accesses in an application. Hong and Kim claim that active warps accessing the global memory concurrently are skewed with respect to each other by *departure_delay* time. Because $MWP_{not\text{-}peak}$ is the number of warps that can access memory in a *memory warp period* (*Mem_Latency*) simultaneously, this metric is evaluated as:

$$MWP_{not-peak} = \frac{Mem\_Latency}{departure\_delay} \tag{8.11}$$

The actual *MWP* is the minimum of MWP_{peak}, $MWP_{not\text{-}peak}$, and the number of active warps on SMs, *N*. Because recent GPGPU devices have relaxed memory access coalescing rules, in

most cases the *MWP* is equal to *N* as discussed in the next chapter. The authors provide several equations in [17] that evaluate the total number of execution cycles in the given application. Equation 8.12 shows the two most commonly occurring scenarios including: 1) *MWP* is equal to the number of active warps, *N* and 2) computation cycles are greater than memory cycles. This equation also includes any execution costs associated with the thread synchronization. The equation parameters are evaluated using micro-benchmarks and PTX assembly [24] inspection.

$$
\begin{aligned}
&if\ (MWP = N)\\
&Exec\_cycles\_app = (Mem\_cycles + Comp\_cycles + \frac{Comp\_cycles}{\#Memory\_Insts} * (MWP - 1)) * Reps\\
&if\ (Comp\_cycles > Mem\_cycles)\\
&Exec\_cycles\_app = (Mem\_cycles * \frac{N}{MWP} + \frac{Comp\_cycles}{\#Memory\_Insts} * (MWP - 1)) * Reps\\
&\\
&Thread\_Sync\_cost = departure\_delay * (MWP - 1) * sync\_insts * Active\_blocks\_per\_SM * Reps\\
&Exec_{Total} = Exec\_cycles\_app + Thread\_Sync\_cost
\end{aligned}
\tag{8.12}
$$

The variables in the above equation are summarized as follows.

- *Mem_cycles*: Execution cycles per thread to execute memory instructions
- *Comp_cycles*: Execution cycles per thread to execute computation instructions
- *#Memory_Insts*: The number of memory instructions
- *Reps*: The number of repetitions for SMs to execute all of the assigned warps in the application
- *Thread_Sync_cost*: Execution cycles due to synchronizing threads in a block
- *Sync_insts*: Number of *__syncthreads()* calls
- *Active_blocks_per_SM*: Total number of active blocks assigned to a single SM

The *objective-analytical model* described in this section is used in the next chapter to predict the GPGPU kernel execution time for the SNN-ADF SIAs, highlighting the potential merits, challenges, and pitfalls associated with this modeling paradigm.

8.2.2 Qualitative Modeling of Communications (Network-level and PCI-Ex Bus)

To study the *objective-analytical modeling* for communications, we develop a variant of the communication models discussed in Chapter 2. We propose *piecewise analytical model* that describes the performance of communication operations (*scatter, gather, sendrecv, device-to-host, host-to-device,* etc.) over different data regions (e.g. 1 KB – 256 KB, 256 KB – 512 KB, etc.) using two *medium* parameters: *overhead* (o_T) and *message gap* (G). Any two data regions are separated by the cut-off message size, k_{cutoff}. The parameters pertaining to the *piecewise analytical model* are summarized below:

- *Overhead* (o_T): The estimated time taken by the processor to initiate the operation
- *Message gap* (G): The estimated transfer time per byte for a message in a given data region; consequently, G varies across data regions
- *Message cut-off* ($k_{cutoff\text{-}n}$): The message size that separates data regions, n and $n+1$

The runtime performance of data operations over Infiniband and PCI-Ex bus is given by Equation 8.13. The numbers in the subscript denote the data regions.

$$
\begin{aligned}
&k < k_{cutoff\,-1} && T = o_T + k * G_1 \\
&k_{cutoff\,-1} \le k < k_{cutoff\,-2} && T = o_T + k_{cutoff\,-1} * G_1 + (k - k_{cutoff\,-1}) * G_2 \\
&k_{cutoff\,-2} \le k < k_{cutoff\,-3} && T = o_T + k_{cutoff\,-1} * G_1 + (k_{cutoff\,-2} - k_{cutoff\,-1}) * G_2 + (k - k_{cutoff\,-2}) * G_3 \\
&k_{cutoff\,-(n)} \le k < k_{cutoff\,-(n+1)} && T = o_T + \sum_{i=1}^{n} (k_{cutoff\,-i} - k_{cutoff\,-i-1}) * G_{i+1} + (k - k_{cutoff\,-n}) * G_{n+1}
\end{aligned}
\tag{8.13}
$$

In what follows, we illustrate the *piecewise-analytical modeling* for the two *communication mediums*.

A. Infiniband Operations: Scatter, Gather, and Sendrecv

As mentioned previously, SIAs fit well with the Master-Worker paradigm where the Master process disseminates tasks to all the other processes and gathers the final result when all of the computations are finished. Albeit not recommended, the processes may also engage in intermediate data exchange during the course of SIA execution using the point-to-point *Sendrecv* routine. We elaborate our communication modeling methodology for the two most commonly used and runtime intensive message passing routines namely, *scatter* and *gather*. We briefly discuss the *Sendrecv* routine and provide the relevant model parameters. The *piecewise analytical modeling* approach can be easily extended to other communication routines as well.

We perform micro-benchmarks for the *communication medium* operations at different node configurations and select the data regions based on the initial sketches of data transfer time. Typically, these data regions can be classified into *short*, *medium*, and *long message regions*. In our experiments for Infiniband operations, message sizes 1 B – 512 KB constitute the *short message region*, 512 KB – 1024 KB constitute the *medium message region* and lastly, 1 MB and above belong to the *long message region.* Thereafter, the *message gap* (G) parameter is determined for each of these regions via curve fitting. It is worth mentioning that *overhead* (o_T) is the one-time cost required to initiate the operation and is relevant in region 1 only. Equation 8.13 implicitly accounts for the *overhead* parameter in all the other data regions. Figures 8.6 – 8.8 show the scatter operation time and corresponding *message gap* (G) in different data regions for the 4-node configuration, justifying the piecewise modeling approach. This technique also overcomes any inaccuracies introduced by the *subjective-analytical model*, which considers all messages in a single data region and fits a single curve for communication throughput.

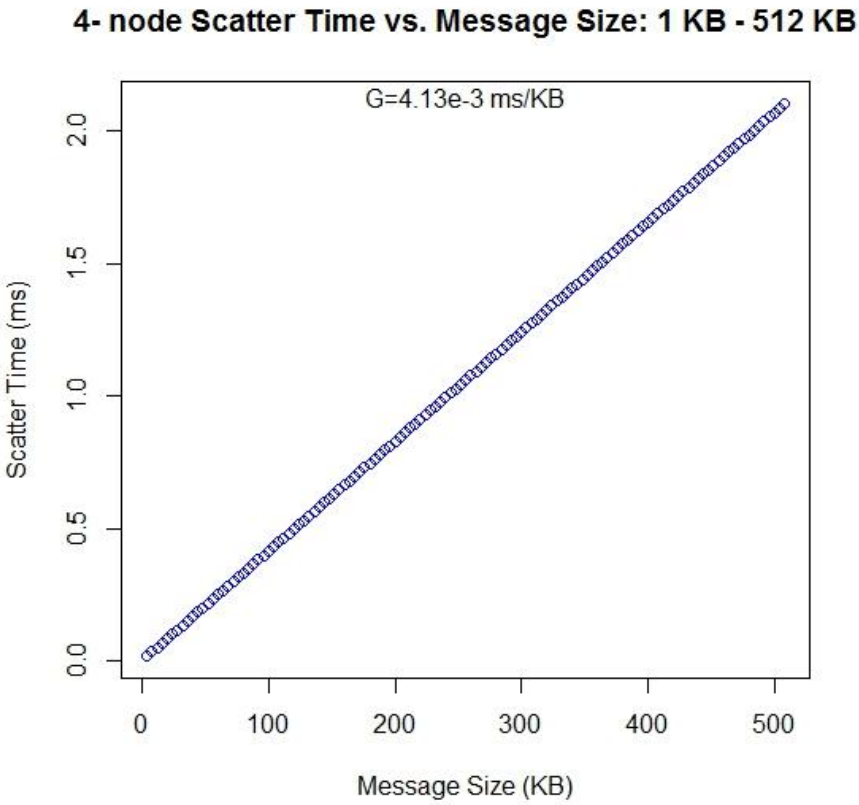

Figure 8.6 4-node Scatter Time vs. Message Size: Data Region 1 KB – 512 KB

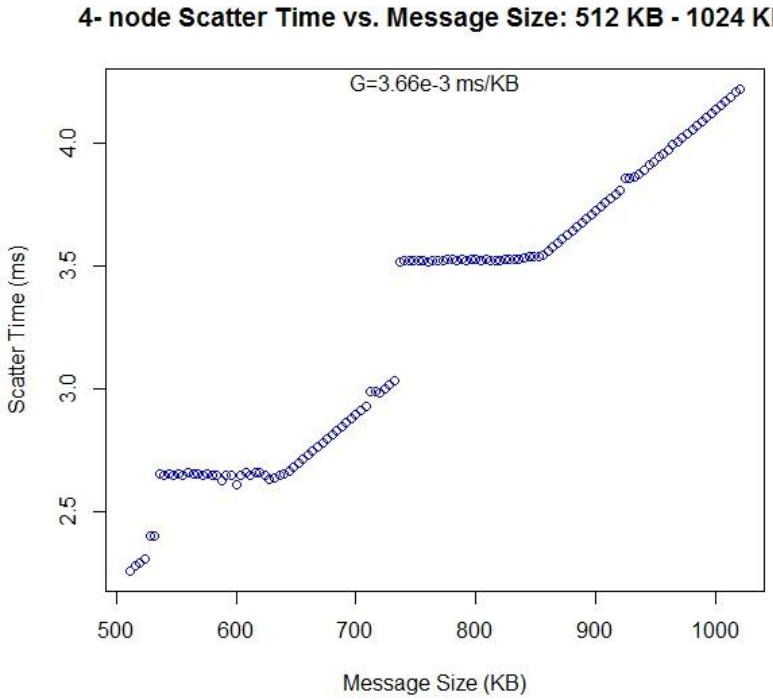

Figure 8.7 4-node Scatter Time vs. Message Size: Data Region 512 KB-1024 KB

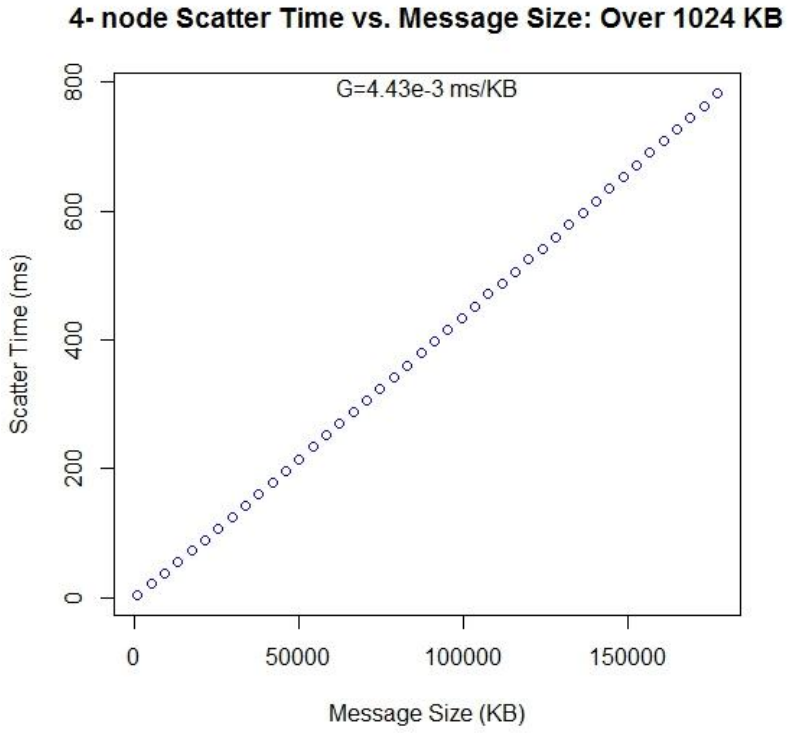

Figure 8.8 4-node Scatter Time vs. Message Size: Data Region Over 1024 KB

Tables 8.1 and 8.2 show the *overhead* and *message gap* parameters for *scatter* and *gather* operations at 2-node, 4-node, 8-node, and 16-node configurations. The dashes in the table signify that the parameter is either irrelevant or statistically insignificant in that data region. Table 8.3 provides the model parameters for the *Sendrecv* routine at different node configurations. The training data-set for the *Sendrecv* routine is obtained using micro-benchmarks that resemble the ADF operations. As seen in the same table, the *overhead* and *message gap* parameters for this point-to-point routine are large when compared to the collective *scatter* and *gather* routines. Therefore, programmers are advised to avoid frequent point-to-point communications and instead use collective operations for optimal performance.

Table 8.1 Overhead (ms) and Message Gap (ms/KB) for Scatter Time

Node Configuration	Region 1 (1 KB – 512 KB)		Region 2 (512 KB – 1024 KB)		Region 3 (over 1024 KB)	
	o_T	G	o_T	G	o_T	G
2-Node	-	9.57e-5	-	9.39e-5	-	2.6e-4
4-Node	2.771e-3	4.13e-3	-	3.66e-3	-	4.43e-3
8-Node	6.86e-3	6.21e-3	-	6.03e-3	-	6.43e-3
16-Node	5.6e-3	7.27e-3	-	7.03e-3	-	7.5e-3

Table 8.2 Overhead (ms) and Message Gap (ms/KB) for Gather Time

Node Configuration	Region 1 (1 KB – 512 KB)		Region 2 (512 KB – 1024 KB)		Region 3 (over 1024 KB)	
	o_T	G	o_T	G	o_T	G
2-Node	9e-4	6.15e-5	-	1.53e-5	-	2.65e-4
4-Node	4.52e-3	4.13e-3	-	4.2e-3	-	4.26e-3
8-Node	2.93e-3	6.20e-3	-	6.17e-3	-	6.32e-3
16-Node	0.59	1e-2	-	1.03e-2	-	7.36e-3

Table 8.3 Overhead (ms) and Message Gap (ms/KB) for Sendrecv Time

Node Configuration	Model Parameters	
	o_T	G
2-Node	0.58	0.02
4-Node	1.1	0.086
8-Node	1.98	0.186
16-Node	3.77	0.43

B. PCI-Ex Bus Operations: Download and Read-back

As discussed in Chapter 4, the Palmetto cluster [15] includes GPGPU-enabled servers equipped with two Nvidia Kepler GK110 devices each. Consequently, up to two MPI ranks (two host-device pairs) can be packed in a single server for node configurations greater than two nodes. Therefore, we perform micro-benchmarks for *download* (host-to-device) and *read-bac*k (device-to-host) operations using two host-device pairs in a single server. We define the following data regions: 1 B – 8 KB (*small message region)*, 8 KB – 512 KB and 512 KB – 1024 KB (*medium message region*), and 1024 KB – 8 MB and 8 MB – 256 MB (*long message regions*). Figures 8.9 through 8.13 provide the initial sketches of download time and corresponding *message gap* (*G*), justifying the constructed data regions.

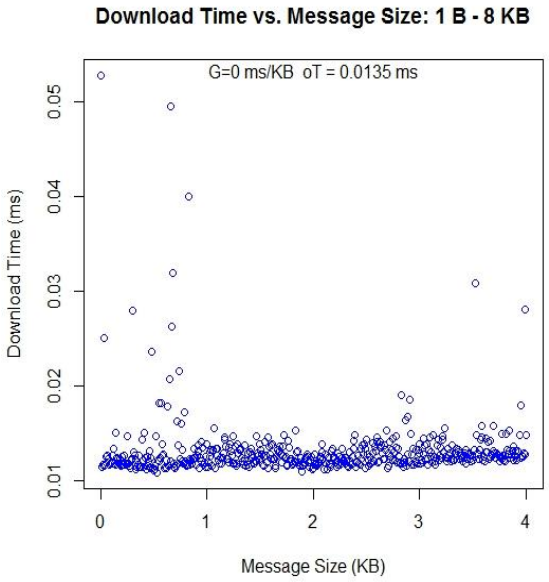

Figure 8.9 Download Time vs. Message Size 1 B – 8 KB

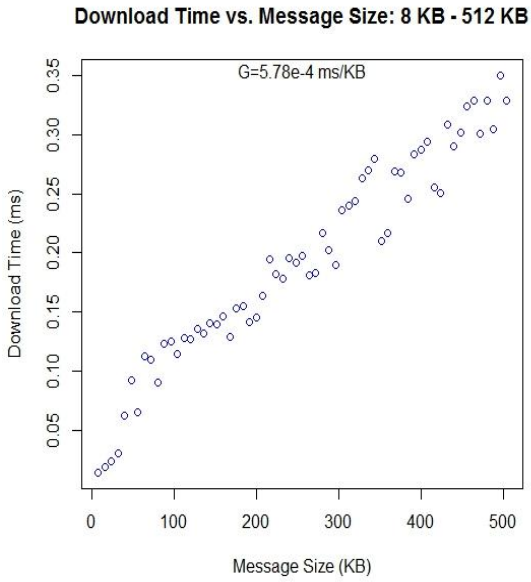

Figure 8.10 Download Time vs. Message Size 8 KB – 512 KB

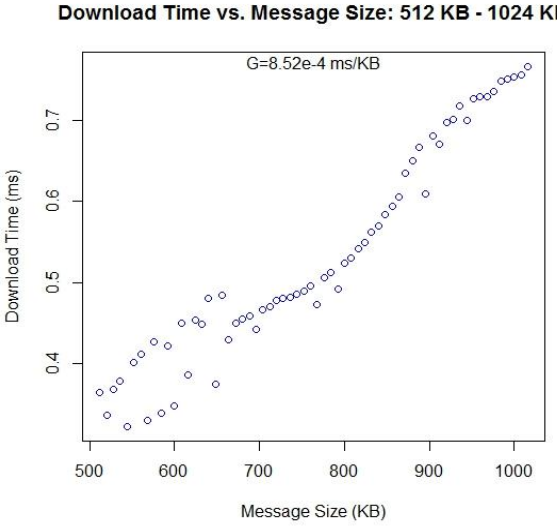

Figure 8.11 Download Time vs. Message Size 512 KB – 1024 KB

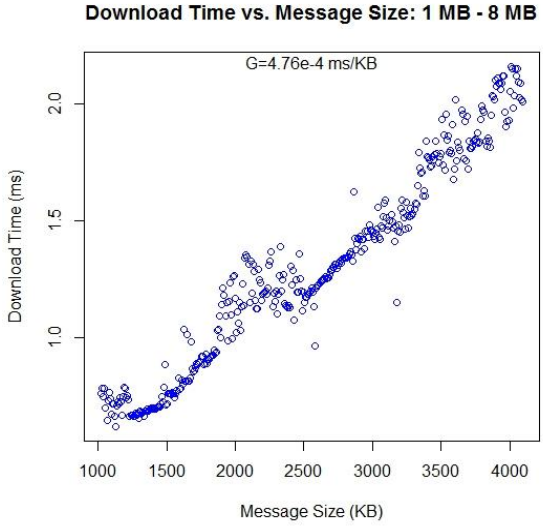

Figure 8.12 Download Time vs. Message Size 1 MB – 8 MB

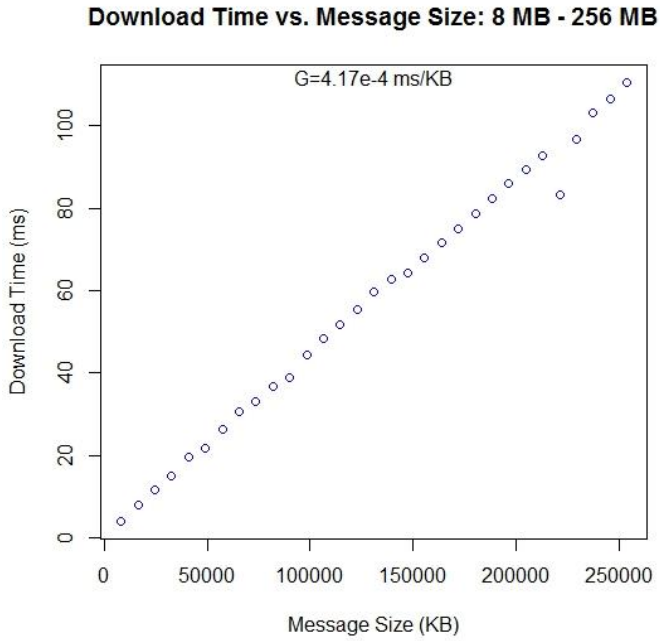

Figure 8.13 Download Time vs. Message Size 8 MB – 256 MB

Table 8.4 gives the *overhead* and *message gap* parameters for *download* and *read-back* operations. Similar to the Infiniband operations, the dashes in the table are due to irrelevance or statistical insignificance of the parameter values.

Table 8.4 Overhead (ms) and Message Gap (ms/KB) for Download and Read-back Time

PCI-Ex Bus Operation	**Region 1 (1 KB – 8 KB)**		**Region 2 (8 KB – 512 KB)**		**Region 3 (512 KB – 1024 MB)**		**Region 4 (1 MB – 8 MB)**		**Region 5 (8 MB – 256 MB)**	
	o_T	G	o_T	G	o_T	G	o_T	G	o_T	G
Download	0.014	-	-	5.8e-4	-	8.5e-4	-	4.76e-4	-	4.2e-4
Readback	0.017	-	-	4.95e-4	-	5.4e-4	-	3.41e-4	-	3.6e-4

8.3 SUMMARY

In this chapter, we discussed the high-level abstraction for modeling the GPGPU computations and *medium* communications. The high-level abstraction approaches, namely qualitative and quantitative methods, were described using *subjective-analytical* and *objective-analytical models*, respectively. The *subjective-analytical models* avoid numerical complexities

by describing the system using minimum qualitative relations amongst the system parameters, providing user-friendly approach to performance modeling. To model the GPGPU computations, we derived simple mathematical relations between *element-throughput*, number of computational entities, and the execution time on *M* GPGPU devices. For the *communication medium* modeling, we explained the Michaelis-Menten kinetics approach with a *subjective-analytic* perspective.

Unlike qualitative methods, the quantitative approach is described by *objective-analytical models* that employ micro-benchmarks to measure system parameters, thereby estimating the target variable. We adapted the GPGPU analytical model proposed by Hong and Kim [17] and provided a sub-set of relevant equations for measuring the GPGPU architecture performance. The parameters associated with this *objective-analytical model* are estimated using micro-benchmarks and PTX assembly inspection. For the communication component, we proposed a variant of the analytical models described in Chapter 2 called the *piecewise-analytical model*. This approach estimates the data transfer time by defining data regions determined by *cut-off messages* and using two *medium* parameters: *overhead* (o_T) and *message gap* (G). The *overhead* parameter is the estimated time taken by the processor to initiate the operation; whereas the *message gap* parameter is the transfer time per byte for a message in a given data region. We elucidated the *piecewise-analytical model* construction for the two most commonly used, runtime intensive network-level routines (*scatter* and *gather*) and interconnect operations (*download* and *read-back*). A brief discussion on the *Sendrecv* routine was provided along with the model parameter values. This point-to-point routine observed significantly high *overhead* and *message gap* parameter values, suggesting the avoidance of this point-to-point routine. In the next chapter, we verify the high-level abstraction models for their prediction efficacy, keeping the emphasis on

GPGPU computations and *medium* communications. We also explore a suitable combination of qualitative and quantitative methods, the *hybrid approach,* for performance predictions on heterogeneous system.

CHAPTER 9

VERIFICATION OF THE HIGH-LEVEL ABSTRACTION

This chapter presents the verification results for the high-level abstraction using the SNN-ADF SIAs studied in this dissertation research. The primary focus is verifying the prediction models for GPGPU computations and *medium* communications; we report prediction error rates for these SIGE model components. A Strengths, Weaknesses, and Opportunities (SWO) study for the high-level abstraction is also presented. The rest of the chapter is structured as follows. Section 9.1 verifies the *subjective-analytical model* for GPGPU computations. Because qualitative models for communications were comprehensively studied in Chapters 6 and 7, we do not show this analysis. Section 9.2 verifies the *objective-analytical models* for GPGPU computations and *medium* communications (Infiniband and PCI-Ex bus) including *scatter, gather, sendrecv, download*, and *read-back.* A combination of effective qualitative and quantitative methods, the *hybrid approach*, is discussed in Section 9.3. The SWO analysis for the high-level abstraction models follows in Section 9.4. The chapter concludes with a summary in Section 9.5.

9.1 VERIFICATION RESULTS: QUALITATIVE MODELING

9.1.1 GPGPU Computations

This section provides the verification results for the GPGPU *subjective-analytical model* using the SNN-ADF SIA case studies. We present the values for observed kernel runtime, estimated kernel runtime, and prediction error rate using selected SNN network and noised

image sizes. The SIAs were executed on the Kepler GPGPU-augmented Palmetto cluster using node configurations varying from 4- to 16-nodes.

We use Equations 8.3 through 8.8 for estimating the GPGPU kernel execution time for the four SNN models. Tables 9.1 through 9.4 provide the observed and estimated kernel runtime values along with the prediction error rates for the HH, ML, Wilson, and Izhikevich models. All of the SNN models observe high prediction accuracy (less than 10%) for several SNN network sizes across the tested node configurations. The ML model, however observes a few outliers with error rates distinctively high compared to the other SNN models. As explained in Chapter 8, the *subjective-analytical model* extrapolates the execution time on *M* GPGPU devices using runtime information from the reference GPGPU device (see Equation 8.3). Because GPGPU computations usually scale well with the number of processors, the analytical approach is expected to yield highly accurate predictions as shown by these tables.

Table 9.5 provides the values of observed kernel runtime, estimated kernel runtime, and corresponding error rate for the ADF algorithm. Similar to the SNN models, the ADF algorithm also observes high prediction accuracy (error less than 5%).

Table 9.1 HH Model: Observed and Estimated Kernel Runtime Values (ms)

Node Configuration	Input Size	Observed Kernel Time	Estimated Kernel Time	Error Rate (%)
4-Node	4200x4200	1144.58	1144.44	0.01
	4800x4800	1472.32	1482.5	-0.7
8-Node	6480x6480	1349.07	1354.5	-0.41
	7200x7200	1681.24	1662.82	1.1
16-Node	8400x8400	1145.26	1144.44	0.072
	9600x9600	1474.214	1482.51	-0.6

Table 9.2 ML Model: Observed and Estimated Kernel Runtime Values (ms)

Node Configuration	Input Size	Observed Kernel Time	Estimated Kernel Time	Error Rate (%)
4-Node	4080x4080	18.24	18.16	0.41
	4800x4800	20.87	20.82	0.22
8-Node	4800x4800	11.78	10.51	10.73
	5040x5040	12.91	11.57	10.4
16-Node	6840x6840	11.97	10.67	10.83
	7140x7140	12.81	11.61	9.36

Table 9.3 Wilson Model: Observed and Estimated Kernel Runtime Values (ms)

Node Configuration	Input Size	Observed Kernel Time	Estimated Kernel Time	Error Rate (%)
4-Node	3600x3600	63.83	65.64	-2.83
	4080x4080	83.7	83.5	0.2
8-Node	4800x4800	57.06	58.7	-2.8
	5040x5040	63.53	64.4	-1.35
16-Node	4800x4800	30.11	30.68	-1.9
	7200x7200	63.83	65.64	-2.85

Table 9.4 Izhikevich Model: Observed and Estimated Kernel Runtime Values (ms)

Node Configuration	Input Size	Observed Kernel Time	Estimated Kernel Time	Error Rate (%)
4-Node	3600x3600	13.04	13.17	-1.04
	4080x4080	16.50	16.56	-0.41
8-Node	4800x4800	11.72	11.85	-1.14
	5040x5040	12.72	12.93	-1.7
16-Node	4800x4800	6.45	6.56	-1.7
	7200x7200	13.01	13.17	-1.26

Table 9.5 ADF: Observed and Estimated Kernel Runtime Values (ms)

Node Configuration	Input Size	Observed Kernel Time	Estimated Kernel Time	Error Rate (%)
4-Node	5120x5120	897	857.74	4.38
	6400x6400	1380.18	1402.86	-1.65
8-Node	6400x6400	686.5	658.7	4.06
	7168x7168	852.3	839.26	1.53
16-Node	7168x7168	391.18	374.62	4.23
	7680x7680	468.6	466.3	0.5

9.2 VERIFICATION RESULTS: QUANTITATIVE MODELING

In this section, we verify the *objective-analytical models* for GPGPU computations and *medium* communications. For the GPGPU analysis, we only show the prediction results for the SNN models using a 4-node configuration; the kernel runtime can be extrapolated for predictions at larger node configurations. The SNN models, with their wide range of computation-to-communication requirements, are highly suitable case studies for the model verification. The communication component prediction models are verified across a range of data sizes; these models are later included in the *hybrid approach* where we investigate their efficacy for overall application runtime prediction.

9.2.1 GPGPU Computations

The *objective-analytical model* for GPGPU computations is discussed in Chapter 8 along with the relevant equations. We performed micro-benchmarks on the Kepler GPGPU device to estimate the values of global memory bandwidth, memory access latency, and departure delays. We specifically adapted the micro-benchmarks given in the SHOC suite [41] to obtain these values. Additional hardware parameters including multi-processor (SMX) frequency, instruction issue cycles, and the number of SMXs were obtained using CUDA programming guide [22] and *deviceQuery* routine from the CUDA SDK [26]. The hardware parameters relevant to the Kepler architecture are summarized in Table 9.6. The application specific parameters including the number of active warps per SMX (related to occupancy), number of load/store bytes per warp, and the number of computation and memory instructions were obtained via PTX assembly analysis [24] and CUDA profile generation [22]. These parameter values are given in Table 9.7

Table 9.6 Kepler (K20) GPGPU Device Parameter Values

Parameter	**Value**		**Method**
Global Memory Bandwidth	144 GB/sec		SHOC Benchmark
Memory Access Latency	Coalesced	Uncoalesced	Adapted SHOC Benchmark
	133 cycles	572 cycles	
Departure Delays	Coalesced	Uncoalesced	Adapted SHOC Benchmark
	1 cycle	38 cycles	
# SMX	13		*deviceQuery*
SMX Frequency	0.71 GHz		*deviceQuery*
Instruction Issue Cycles	SP*	Trans.*	Device Specifications
	$\frac{32}{192}$ cycles	$\frac{32}{32}$ cycles	

*SP: Single-Precision Floating Point; Trans.: Transcendental Functions; cycles: SMX cycles

.

Table 9.7 SNN Models: Application Specific Parameters

SNN Model	**Application Specific Parameters**	
HH	*#Active_Warps*	24
	#load/store_bytes_per Warp	56
	#Comp_Insts	71 SP + 13 Trans.
	#Mem_Insts	14
ML	*#Active_Warps*	32
	#load/store_bytes_per Warp	40
	#Comp_Insts	58 SP + 9 Trans.
	#Mem_Insts	10
Wilson	*#Active_Warps*	32
	#load/store_bytes_per Warp	52
	#Comp_Insts	49 SP + 5 Trans.
	#Mem_Insts	13
Izhikevich	*#Active_Warps*	32
	#load/store_bytes_per Warp	32
	#Comp_Insts	19 SP
	#Mem_Insts	8

Prior to providing the prediction results for the four SNN models, we first demonstrate the *objective-analytical model* for HH model kernel runtime prediction using the 4-node configuration and a SNN network size of 4200 x 4200. The runtime analysis is based on the

multi-GPGPU orchestration for the SNN models given in Chapter 4. The parameters pertaining to Equations 8.10 through 8.12 are summarized in Table 9.8.

Table 9.8 HH Model: Objective-Analytical Model Parameter Values; 4-Node Configuration

Parameter	Value	Obtained Using
Input Size Per GPGPU	$\frac{4200*4200}{4} = 4.41e6$	Algorithm Specification
Threads per Block	256	CUDA User-Defined Specification
#Blocks	17227	$\frac{\text{Input Size}}{\text{Threads per Block}}$
#Active_blocks_per_SM	3	$\frac{\#Active\_Warps}{(\frac{\text{Threads per Block}}{\text{Warp Size}})}$
Reps	442	Equation 8.12
Mem_cycles (cycles)	1862	$Mem\_Insts * Mem\_Latency$
Comp_cycles (cycles)	28	$Comp\_insts * instruction\_issue\_cycles$
Exec_cycles_app (cycles)	8.56e5	Equation 8.12
Thread_Sync_cost (cycles)	6.1e4	Equation 8.12
$Exec_{Total}$	9.17e5	Equation 8.12
Execution time per kernel (milliseconds)	1.29	$\frac{Exec_{Total}}{Frequency}$
Algorithm Iterations	472	Algorithm Specification
Total Execution Time (milliseconds)	608.8	Execution time per kernel * Algorithm Iterations

The kernel runtime predictions for the four SNN models at the 4-node configuration using selected SNN sizes are given in Table 9.9. The error rates for all test cases are high (40-60%), suggesting several missing components in the *objective-analytical model.* The prediction model yielded significantly high error values for the ML model that are beyond 100%; this observation is under investigation. Although the model provides significant insight into the GPGPU architecture, a comprehensive study of several device parameters pertaining to instruction caches, quad warp schedulers, and multi-level L1/L2 caches should to be incorporated in the modeling approach in future work. The future work also includes the use/development of

effective PTX assembly parsing software to obtain precise counts of memory and computation instructions.

Table 9.9 SNN Models: Observed and Estimated Kernel Runtime Values (ms)

SNN Model	Network Size	Observed Kernel Time	Estimated Kernel Time	Error
HH	4200x4200	1144.58	608.88	-47%
	4800x4800	1472.32	791.1	-47%
ML	4080x4080	18.24	83.22	356%
	5040x5040	12.91	127.7	889%
Wilson	3600x3600	63.83	89.22	-40%
	4080x4080	83.7	114.56	37%
Izhikevich	3600x3600	13.04	5.03	-61%
	4080x4080	16.5	6.5	-61%

The values marked in red are under investigation

9.2.2 Medium Communications: Infiniband and PCI-Ex Bus

A. Infiniband: Scatter, Gather, and Sendrecv

The *piecewise analytical models* for *medium communications* were discussed in the previous chapter. Figures 9.1 through 9.3 provide the bar graph representation of the observed and predicted *scatter* time values versus the data size for 4-node, 8-node, and 16-node configurations. The prediction analysis is performed using Equation 8.13 and model parameters given in Table 8.1. The predicted *scatter* time values match the observed time values closely for multiple test cases. The 4-node and 8-node configurations observed satisfactory predictions with error rates of 2.06% and 0.9% for their respective largest test data size. Although the prediction model yielded acceptable predictions for the 16-node configuration using several test cases, a few outliers with over 15% error rate were observed. Overall, the *scatter* time predictions were found to be satisfactory; the *objective-analytical model* captures the network behavior effectively by analyzing the data regions separately.

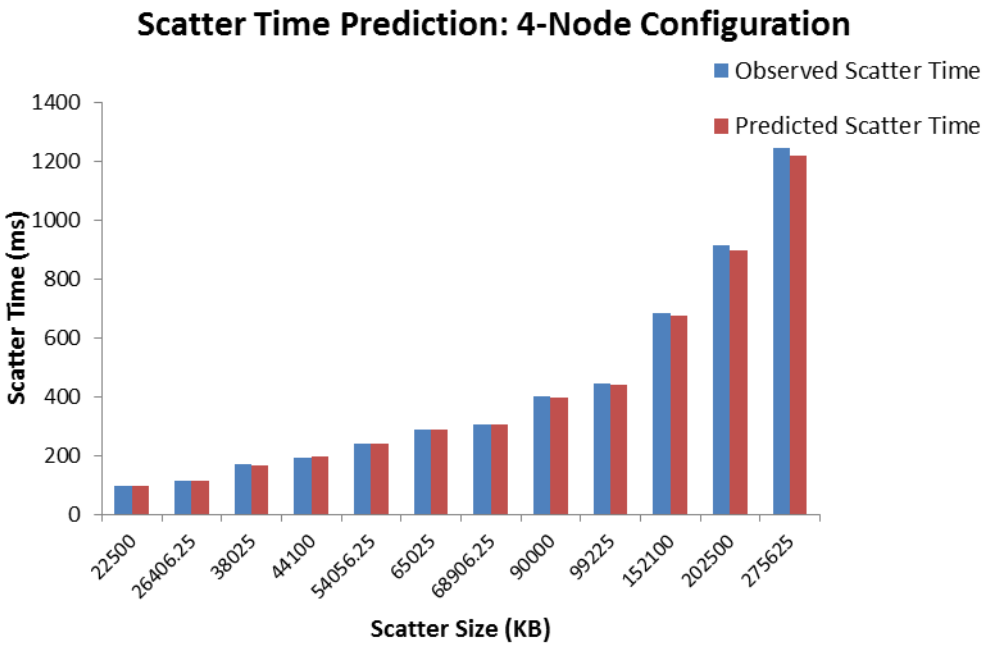

Figure 9.1 Scatter Time Prediction for 4-Node Configuration

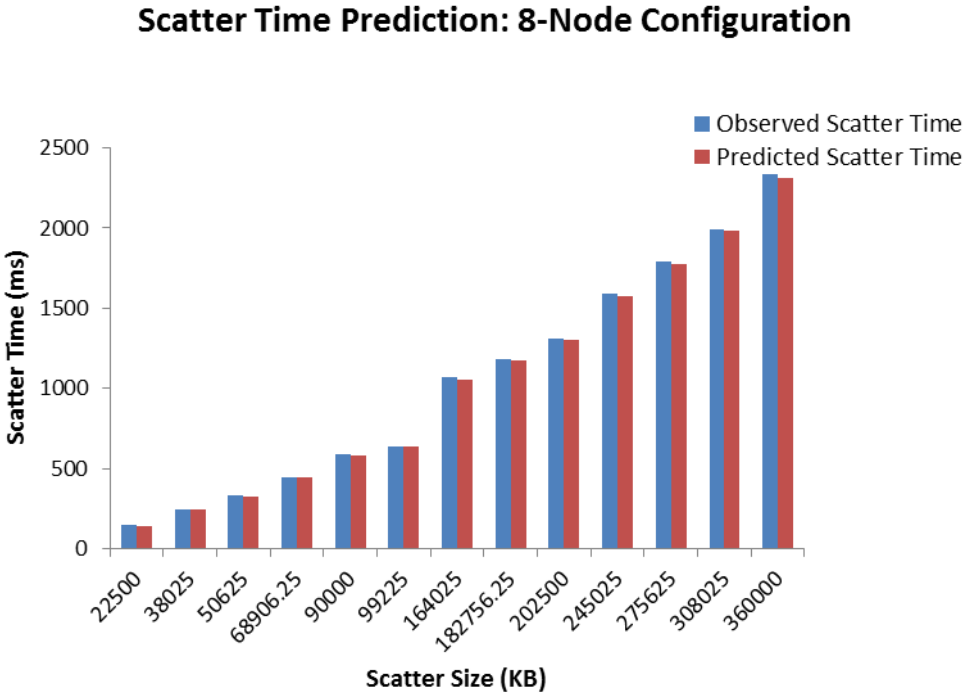

Figure 9.2 Scatter Time Prediction for 8-Node Configuration

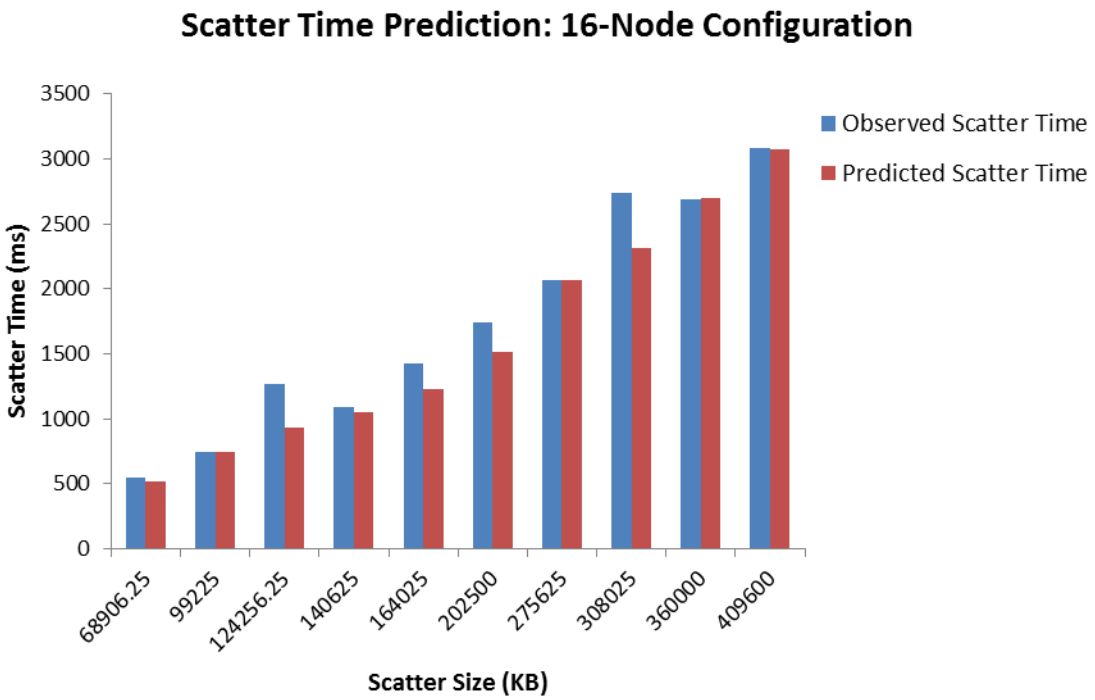

Figure 9.3 Scatter Time Prediction for 16-Node Configuration

Figures 9.4 through 9.6 provide the observed and predicted *gather* time bar graphs using data sizes typically employed by the ADF algorithm. The predictions are performed using Equation 8.13 and model parameter values given in Table 8.2. All node configurations observed satisfactory *gather* time predictions with error rate less than 6% for several tested data sizes. The *sendrecv* time predictions were also acceptable as shown in Figures 9.7 – 9.9, verifying the adequacy of *objective-analytical models* for productive communication component prediction.

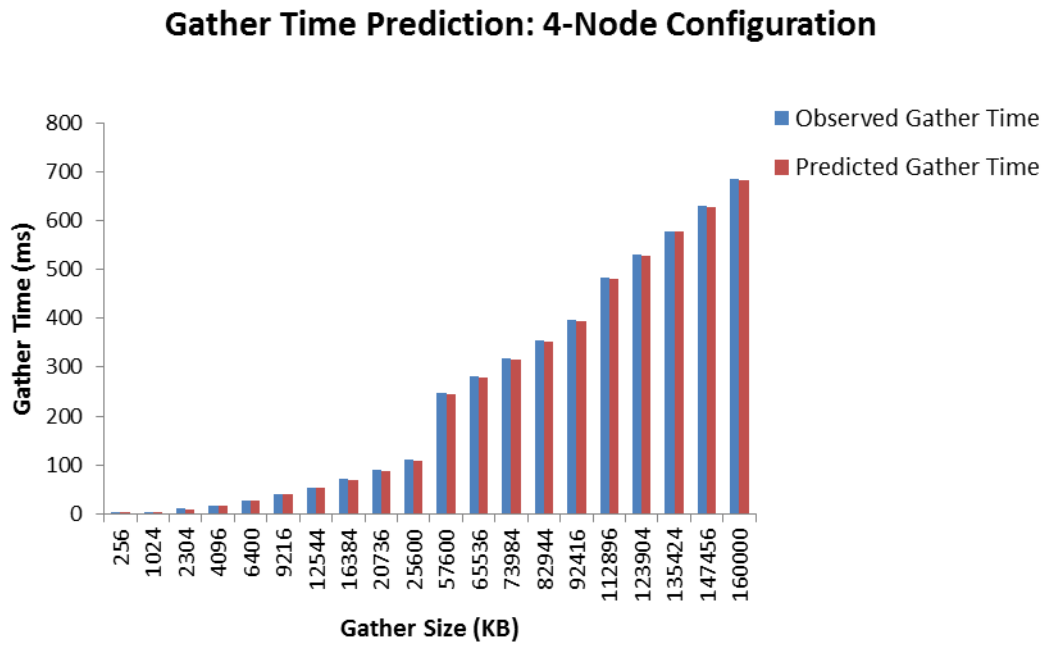

Figure 9.4 Gather Time Prediction for 4-Node Configuration

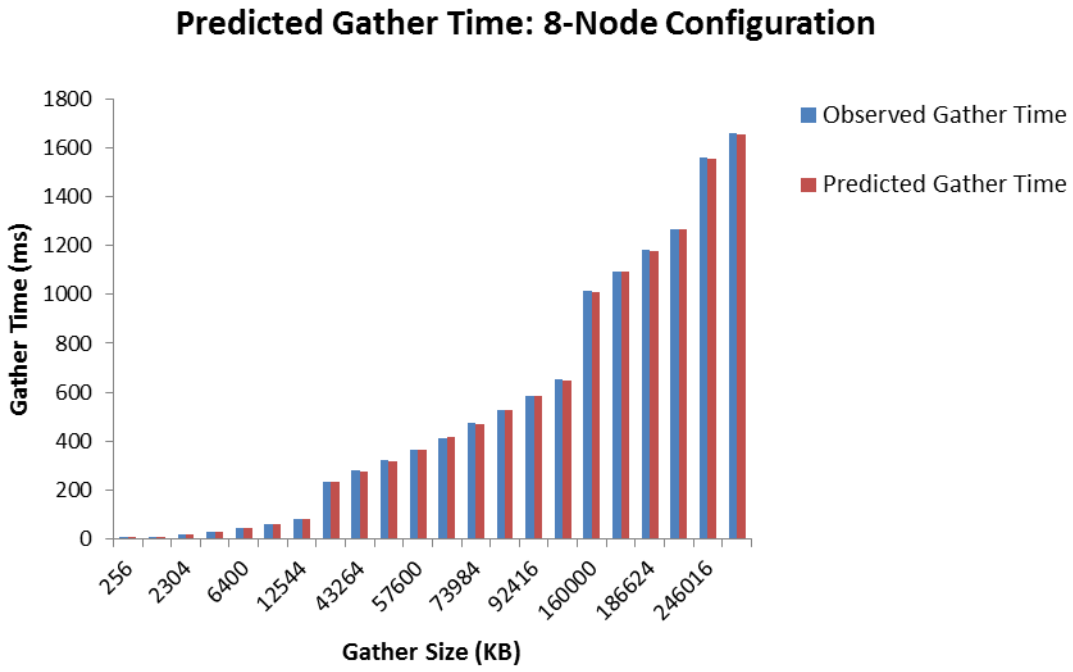

Figure 9.5 Gather Time Prediction for 8-Node Configuration

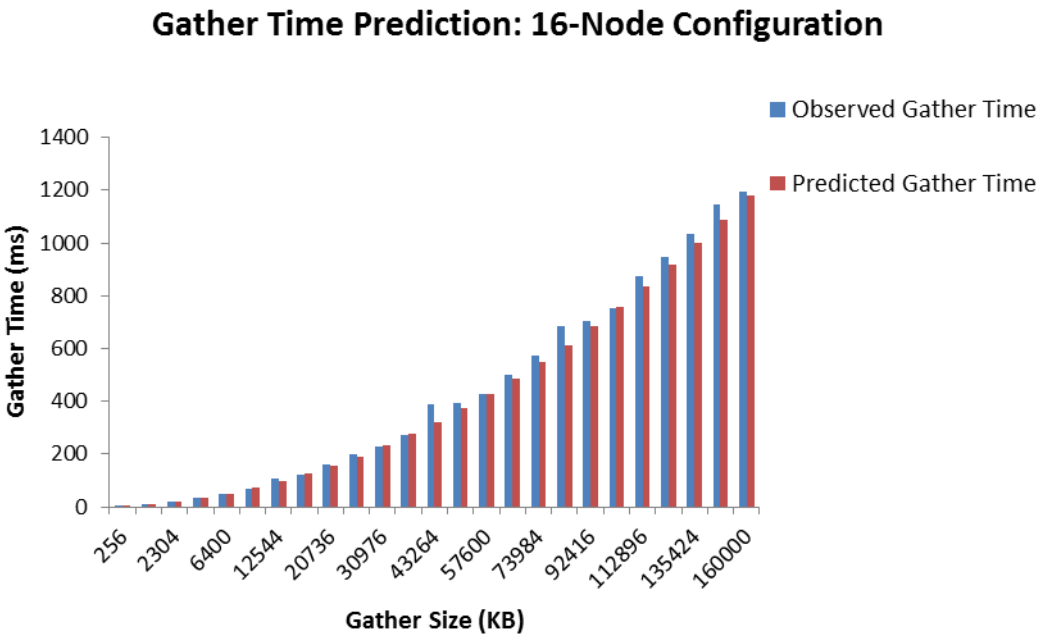

Figure 9.6 Gather Time Prediction for 16-Node Configuration

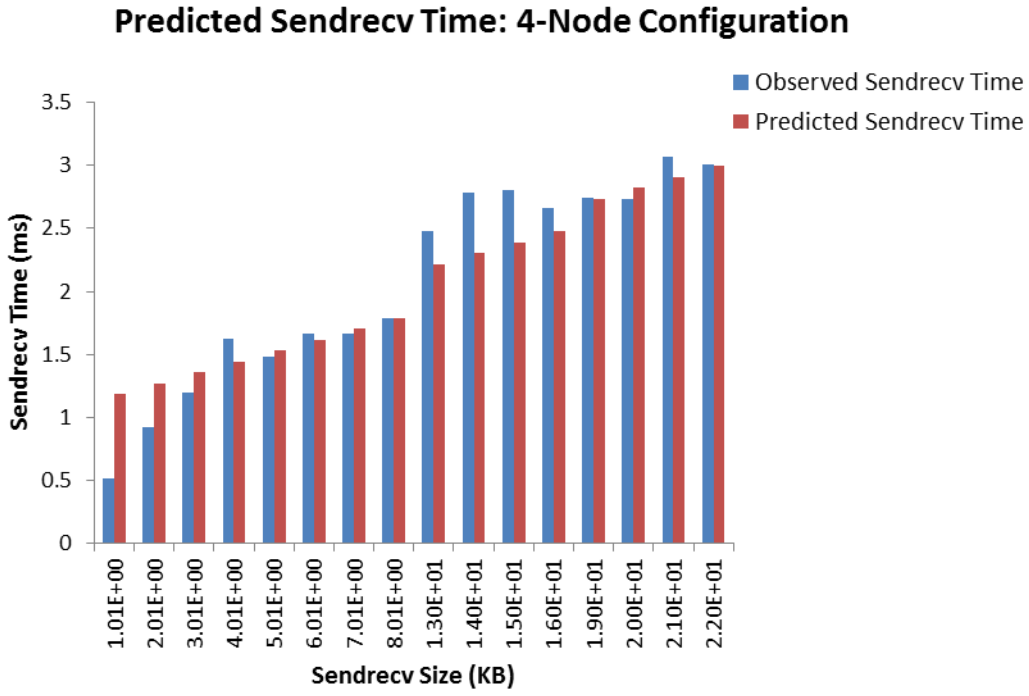

Figure 9.7 Sendrecv Time Prediction for 4-Node Configuration

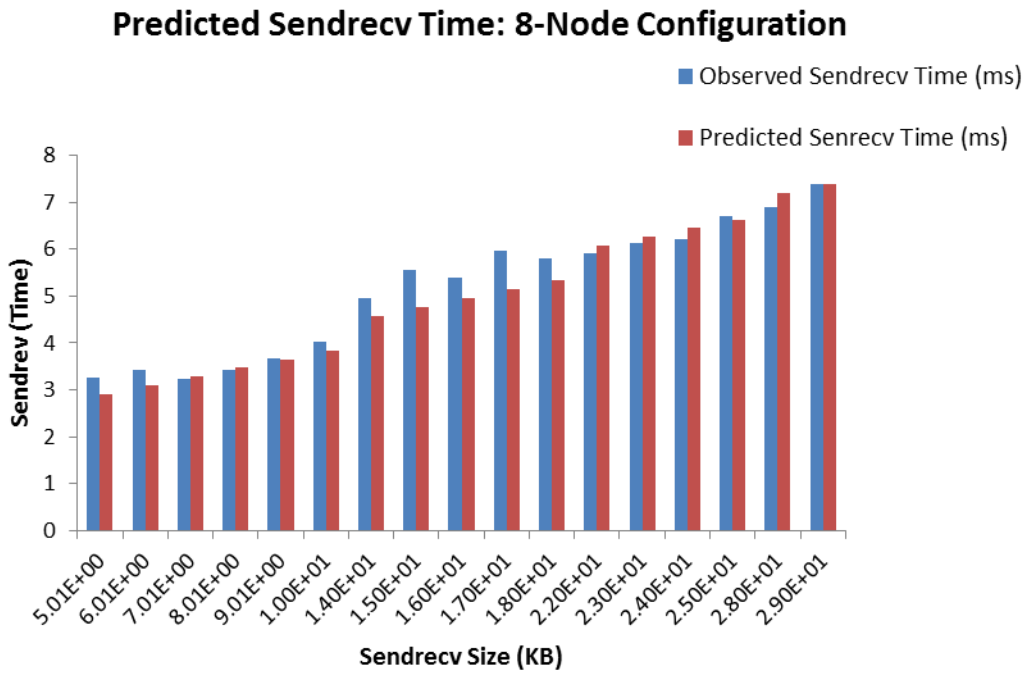

Figure 9.8 Sendrecv Time Prediction for 8-Node Configuration

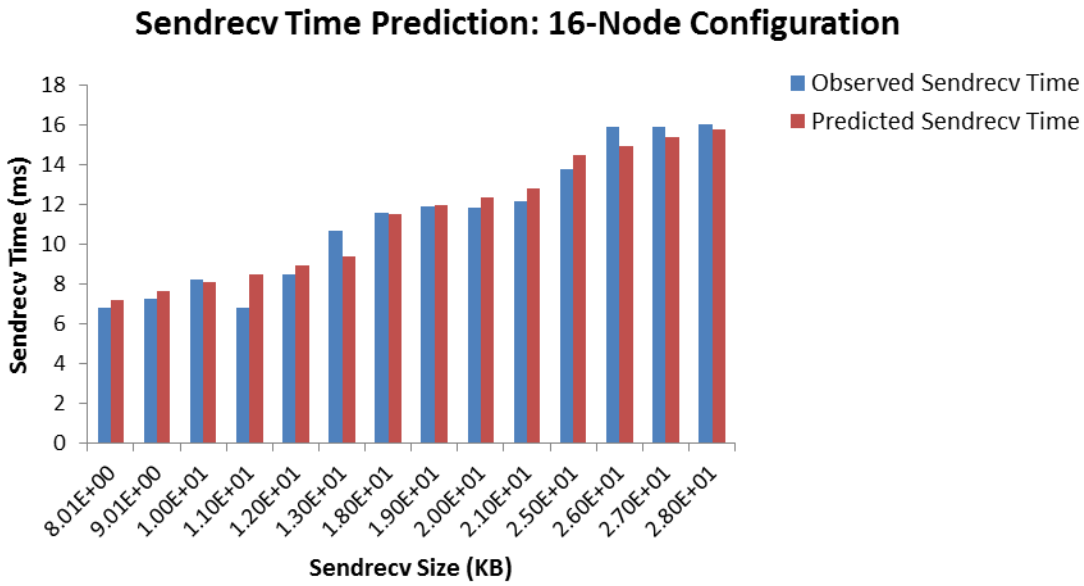

Figure 9.9 Sendrecv Time Prediction for 16-Node Configuration

B. PCI-Ex Bus: Download and Read-back

We assert that the performance impact of *download* and *read-back* operations are most suitably studied in conjunction with the GPGPU kernel execution time as elucidated in Chapters

6 and 7. Equation 5.3 combines the GPGPU kernel time with host-to-device (*download*) and device-to-host (*read-back*) transfer times to facilitate user-friendly analysis. To demonstrate the prediction capabilities of the PCI-Ex bus *objective-analytical model*, we only show the selected cases of the HH model and ADF algorithm at 8-node configuration for the *download* and *read-back* operations, respectively. These two case studies sufficiently represent the chosen SIA set for inter-connect *medium* communications. The analytical model parameters are given in Table 8.4. Figures 9.10 and 9.11 show the *download* and *read-back* prediction performances, respectively.

The *download* prediction accuracy was found to be satisfactory with less than 10% error rate for most of the test cases. Moderately high prediction errors were observed only for smaller data sizes; small deviations in predictions result in high error rates for numerically small runtimes. Unlike the *download* operation, the *read-back* predictions were imprecise with few test cases yielding error rates between 20 to 25%. The authors surmise that the GPGPU device may require additional time to service the data request from the host processor, which may vary across applications. The additional time may be attributed to the inter-connect protocol execution [11]. Consequently, the model parameters generated using micro-benchmarks may not completely represent the *read-back* characteristics in an application. These claims require additional investigation and are left for future work beyond this dissertation.

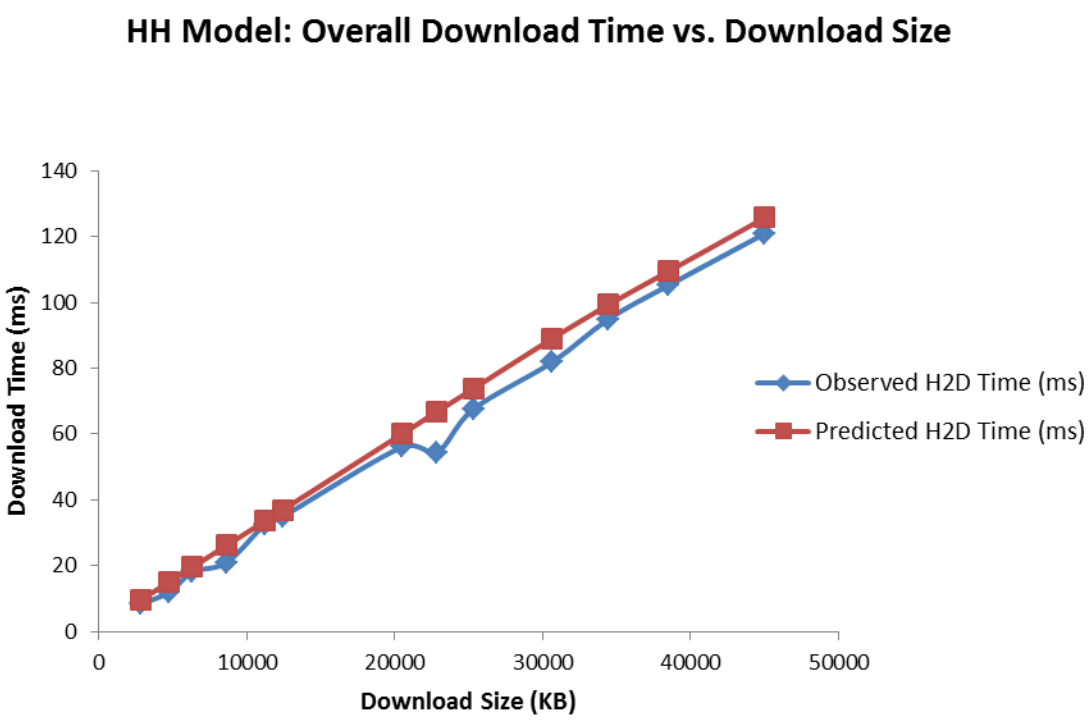

Figure 9.10 HH model: Overall Download Time Prediction for 8-Node Configuration

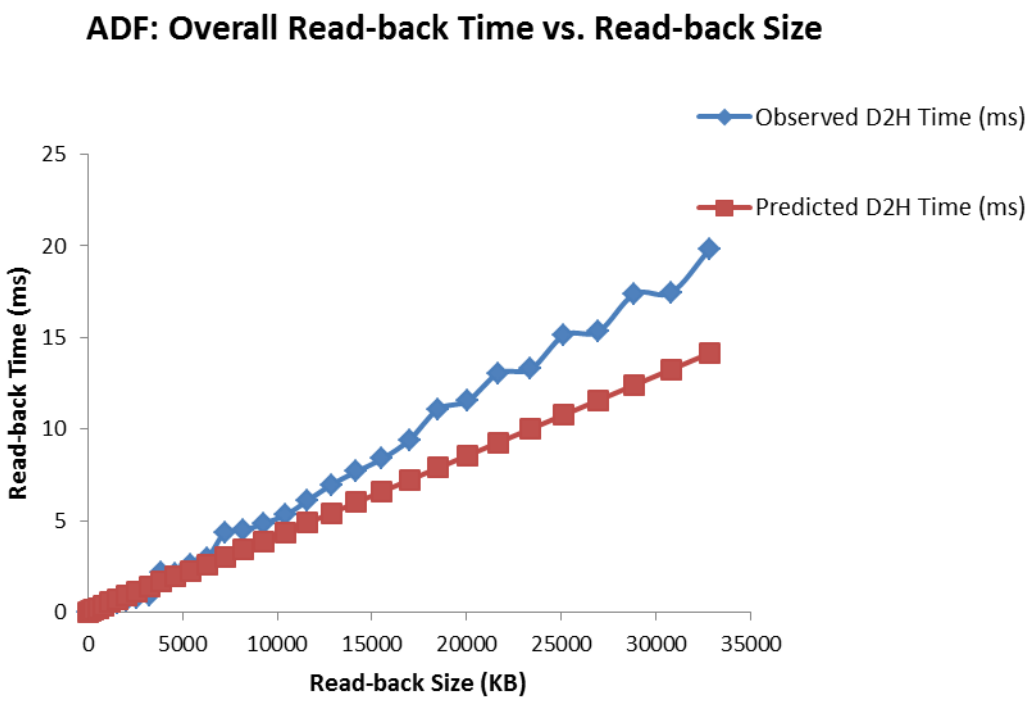

Figure 9.11 ADF: Overall Read-back Time Prediction for 8-Node Configuration

9.3 HYBRID MODELING

The high-level abstraction verification results in the preceding sections suggest that these models, when used alone, are likely to yield coarse-grained application runtime prediction, thereby necessitating a mixed approach. While several combinations of qualitative and quantitative methods can be explored to yield an optimal *hybrid modeling approach*, our selection of the high-level abstraction model framework is as follows. To perform the GPGPU kernel runtime predictions, we employ the *subjective-analytical model* because it is user-friendly and offers high prediction accuracy. Unlike the *objective-analytical models*, the *subjective modeling* approach avoids complex numerical estimations of hardware events by determining simple and intuitive relations amongst the system variables. Additionally, since the GPGPU computations are highly reproducible and generally scale well with data and the number of processors, the *subjective modeling* approach is expected to yield superior results. However for the *medium* communications (Infiniband and PCI-Ex bus), we select the *objective-analytical models* for performance prediction. As discussed in the previous chapter, the proposed *piecewise analytical model* describes the communication performance across different data regions using *medium* specific parameters namely, *overhead* and *message gap.* This *objective-analytical model* overcomes any inaccuracies introduced by the *subjective modeling* approach, which fits a single qualitative relation for the data transfer throughput regardless of varying *medium* performance over multiple data regions.

In what follows, we present the preliminary prediction results for the computation and communication components of the SIGE model for the SNN-ADF SIAs. The predictions are performed using a set of selected input sizes and node configurations varying from 4- to 16-nodes. We also provide error rates for the overall runtime prediction. It should be noted that the

high-level abstraction analysis for the computation component only comprises the GPGPU computations, which includes kernel time, host-to-device time, and device-to-host time (see Equation 5.3). Similar to the low-level abstraction modeling, the communication component analysis comprises of all the network-level transactions performed in the algorithm.

9.3.1 HH Model

Tables 9.10 and 9.11 provide the observed and predicted runtime values for computations and communication components, respectively. All of the SIGE model equations employed to verify the regression-based framework (Chapter 7) also apply for this analysis. The computation component yielded satisfactory prediction results with error rates less than 2%, barring a single test case where 11% error rate was observed. The communication component observed satisfactory prediction results using the *piecewise-analytical model* with error rates less than 2%, owing to the highly accurate *scatter* time predictions. Given the high prediction accuracies of the computation and communication components, error rates for the overall execution time predictions were also low as seen in Table 9.12.

Table 9.10 HH Model: Observed and Predicted Time Values for Computation Component

Configuration	$\mathbf{T_{computation}=T_{GPGPU\text{-}kernel}+T_{H2D}+T_{D2H}}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	4200x4200	1229.25	1249.03	-1.6
	4800x4800	1574.52	1401.8	10.96
8-Node	6480x6480	1444.5	1453.7	-0.64
	7200x7200	1788.72	1776.4	0.7
16-Node	8400x8400	1232.4	1233.9	-0.13
	9600x9600	1575.5	1587.73	-0.77

Table 9.11 HH Model: Observed and Predicted Time Values for Communication Component

Configuration	$T_{communicatation}=\sum T_{Transactions}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	4200x4200	307.3	304.7	0.84
	4800x4800	401.5	398.16	0.83
8-Node	6480x6480	1066.35	1054.37	1.12
	7200x7200	1312.6	1301.8	0.83
16-Node	8400x8400	2066.4	2066.84	-0.02
	9600x9600	2692.07	2699.65	-0.28

Table 9.12 HH Model: Observed and Predicted Execution Time Values

Configuration	$T_{Execution} = T_{computatiom} + T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	4200x4200	1536.53	1553.74	-1.12
	4800x4800	1976.02	1799.97	8.9
8-Node	6480x6480	2630.53	2508.14	4.65
	7200x7200	3101.32	3078.16	0.75
16-Node	8400x8400	3298.8	3300.807	-0.06
	9600x9600	4267.61	4287.37	-0.46

9.3.2 ML Model

The computation and communication component predictions are given in Tables 9.13 and 9.14, respectively. The computation predictions were not as accurate as the HH model; error rate values as high as 7% and 11% were observed. Although the GPGPU computation predictions were acceptable, the *download* and *read-back* operations yielded large error values. Because the ML model is moderately computationally intensive, the host-device transfer times match the GPGPU kernel time, thereby significantly contributing to the overall prediction error. Similar to the HH model, the communication component predictions were satisfactory as seen in Table 9.14. The overall execution time prediction results are given in Table 9.15.

Table 9.13 ML Model: Observed and Predicted Time Values for Computation Component

Configuration	$T_{computation}=T_{GPGPU\text{-}kernel}+T_{H2D}+T_{D2H}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	4080x4080	44.99	48.22	-7.17
	4800x4800	59.33	66.17	-11.55
8-Node	4800x4800	33.87	35.02	-3.4
	5040x5040	39.3	38.33	2.46
16-Node	6840x6840	33.922	35.5	-4.65
	7140x7140	37.47	38.44	-2.6

Table 9.14 ML Model: Observed and Predicted Time Values for Communication Component

Configuration	$T_{computation}=\sum T_{Transactions}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	4080x4080	283.52	287.52	-1.41
	4800x4800	393.95	398.16	-1.07
8-Node	4800x4800	581.67	578.38	0.6
	5040x5040	637.12	637.7	-0.1
16-Node	6840x6840	1370.65	1370.32	0.024
	7140x7140	1491.74	1493.2	-0.1

Table 9.15 ML Model: Observed and Predicted Execution Time Values

Configuration	$T_{Execution} = T_{computatiom} + T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	4080x4080	328.52	335.75	-2.2
	4800x4800	453.28	464.33	-2.44
8-Node	4800x4800	615.54	613.411	0.35
	5040x5040	676.43	676.04	0.06
16-Node	6840x6840	1404.57	1405.82	-0.1
	7140x7140	1529.209	1531.632	-0.16

9.3.3 Wilson Model

Similar to the previous SNN model, the Wilson model is also moderately computationally intensive. Consequently, error rates between 5-10% were observed for the computation component shown in Table 9.16. The communication component predictions were fair with error

rates less than 2%, albeit with an outlier at SNN network size 4080x4080 for the 4-node configuration (see Table 9.17). Given the high prediction accuracies for the computations and communications, the overall execution time predictions observed error values less than 6% (see Table 9.18).

Table 9.16 Wilson Model: Observed and Predicted Time Values for Computation Component

Configuration	$\mathbf{T_{computation}=T_{GPGPU\text{-}kernel}+T_{H2D}+T_{D2H}}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	3600x3600	100.76	103.94	-3.15
	4080x4080	129.83	132.34	-1.93
8-Node	4800x4800	90.22	95.45	-5.81
	5040x5040	100.47	104.55	-4.1
16-Node	6840x6840	101.1	94.25	6.78
	7200x7200	111.27	103.97	6.55

Table 9.17 Wilson Model: Observed and Predicted Time Values for Communication Component

Configuration	$\mathbf{T_{computation}=\sum T_{Transactions}}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	3600x3600	220.95	223.72	-1.25
	4080x4080	315.31	287.52	8.81
8-Node	4800x4800	586.06	578.4	1.31
	5040x5040	645.84	637.7	1.26
16-Node	6840x6840	1376.14	1370.32	0.42
	7200x7200	1515.67	1521.99	-0.42

Table 9.18 Wilson Model: Observed and Predicted Execution Time Values

Configuration	$\mathbf{T_{Execution} = T_{computatiom} + T_{communication}}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	3600x3600	321.7	327.66	-1.85
	4080x4080	445.14	419.85	5.7
8-Node	4800x4800	676.27	673.85	0.36
	5040x5040	746.3	742.26	0.54
16-Node	6840x6840	1477.24	1464.56	0.86
	7200x7200	1626.94	1625.97	0.06

9.3.4 Izhikevich Model

The prediction results for the computation component, communication component, and overall execution time are given in Tables 9.19 – 9.21. The *subjective-analytical model* for the GPGPU kernel computations coupled with the *objective-analytical models* for the *medium* communications yielded high prediction accuracy as seen in these tables.

Table 9.19 Izhikevich Model: Observed and Predicted Time Values for Computation Component

Configuration	**$T_{computation}=T_{GPGPU\text{-}kernel}+T_{H2D}+T_{D2H}$**			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	3600x3600	30.15	31.33	-3.9
	4080x4080	37.43	39.88	-6.52
8-Node	4800x4800	27.834	27.99	-0.57
	5040x5040	30.26	30.73	-1.55
16-Node	6840x6840	28.61	28.40	0.74
	7200x7200	32	31.33	2.1

Table 9.20 Izhikevich Model: Observed and Predicted Time Values for Communication Component

Configuration	**$T_{computation}=\sum T_{Transactions}$**			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	3600x3600	218.45	223.73	-2.41
	4080x4080	338.435	287.52	15.04
8-Node	4800x4800	576.2	578.4	-0.4
	5040x5040	638.55	637.7	0.132
16-Node	6840x6840	1356.84	1370.32	-1
	7200x7200	1527.26	1518.4	0.6

Table 9.21 Izhikevich Model: Observed and Predicted Execution Time Values

Configuration	$T_{Execution} = T_{computatiom} + T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	3600x3600	248.6	255.056	-2.6
	4080x4080	375.87	327.4	12.89
8-Node	4800x4800	604.034	606.4	-0.4
	5040x5040	668.81	668.44	0.056
16-Node	6840x6840	1385.45	1398.72	-0.95
	7200x7200	1559.27	1549.73	0.61

9.3.5 ADF

Similar to the SNN models, the ADF algorithm also observed high prediction accuracy for all of the SIGE model components. The prediction values are provided in Tables 9.22 through 9.24. The computation component observed slightly high prediction error rates (values up to 4.5% versus 1.6% for the HH model) due to the error-prone *download* and *read-back* predictions. Owing to the high prediction accuracy of *objective-analytical models* for *scatter, gather,* and *sendrecv* operations, the communication component for the ADF algorithm yielded error rates less than 1.5%. The results confirm the applicability of the *piecewise analytical models* for highly accurate communication performance prediction. The overall execution time predictions are nearly 98% accurate, verifying the viability of the *hybrid approach* for satisfactory performance prediction.

Table 9.22 ADF: Observed and Predicted Time Values for Computation Component

Configuration	$T_{computation}=T_{GPGPU\text{-}kernel}+T_{H2D} + T_{D2H}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	5120x5120	918.83	881.22	4.1
	6400x6400	1463.5	1439.1	1.67
8-Node	6400x6400	708.83	677.05	4.48
	7168x7168	881.98	862.3	2.23
16-Node	7168x7168	406.52	415.75	-2.3
	7680x7680	486.4	479.8	1.36

Table 9.23 ADF: Observed and Predicted Time Values for Communication Component

Configuration	$T_{computation}=\sum T_{Transactions}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	5120x5120	1330.78	1345.13	-1.08
	6400x6400	2085.56	2101.27	-0.74
8-Node	6400x6400	3082.7	3074.7	0.26
	7168x7168	3867.05	3855.94	0.29
16-Node	7168x7168	4471.5	4506.3	-0.78
	7680x7680	5136.7	5171.16	-0.7

Table 9.24 ADF: Observed and Predicted Execution Time Values

Configuration	$T_{Execution} = T_{computatiom} + T_{communication}$			
	Test Data Size	**Observed Time (ms)**	**Predicted Time (ms)**	**Error Rate (%)**
4-Node	5120x5120	2255.31	2226.77	1.27
	6400x6400	3510.26	3540.32	-0.85
8-Node	6400x6400	3791.51	3751.73	1.05
	7168x7168	4749.02	4718.22	0.65
16-Node	7168x7168	4878.02	4922.05	-1
	7680x7680	5623.11	5650.95	-0.5

In the foregoing section, we discussed the *hybrid approach* as a suitable combination of qualitative and quantitative models for highly accurate application performance prediction. The initial verification of this approach with the SNN-ADF SIAs yielded high quality prediction results for both the computation and communication components of the SIGE model. The presented results satisfactorily establish the *hybrid approach* as a viable paradigm for precise runtime estimations. In the next section, we provide the Strengths, Weaknesses, and Opportunities (SWO) analysis of the high-level abstraction models, based on the verification results given in Sections 9.1 – 9.3.

9.4 STRENGTHS, WEAKNESSES, AND OPPORTUNITIES (SWO) ANALYSIS

Strengths – The *qualitative models* are described using *subjective-analytical models* that define qualitative relations amongst the system variables to describe the overall system behavior. This intuitive approach is easy to understand and straightforward to apply; consequently, these models can also be used by developers and researchers with limited computer architecture knowledge. In this dissertation research, the *subjective-analytical models* were shown to provide quality performance predictions for GPGPU computations. Similar to the regression-based framework, the *subjective-analytical modeling* approach is expected to span generations of GPGPU architectures.

Unlike qualitative modeling, the quantitative approach is described by *objective-analytical models* that estimate the system behavior by measuring hardware-specific events using micro-benchmarks. We developed *piecewise analytical models* for the medium communications that leveraged accurate communication time predictions. These models also avoid any inaccuracies introduced by the *subjective-analytical models* that provide a single qualitative relation for communications regardless of varying system behavior across message sizes.

Weaknesses – The simplified *qualitative methods* are prone to overlooking additional system features, ultimately leading to imprecise performance predictions. The proposed *subjective-analytical model* (Michaelis-Menten approach) for *medium* communications does not include additional network parameters, such as change in network protocol for instance. Consequently, these models observed high error rates for the communication component as elucidated in Chapter 7. We assert that the *qualitative models* are better suited for systems with reproducible characteristics, GPGPU devices computations for instance. Owing to the reproducible scalability of GPGPU computations, the *subjective-analytical model* was shown to effectively extrapolate

the execution time on *M* GPGPU devices using runtime information from the reference device. Unlike computations, the *medium* communications are prone to randomness in the system; consequently, simple qualitative relations with minimum parameters may not suffice for accurate performance predictions.

The *quantitative methods* provide an elaborate route to performance prediction via system parameter estimation using micro-benchmarks. Depending on the system complexity, this approach offers varied user-friendliness and accuracy. Complex systems including CPU and GPGPU architectures necessitate precise parameter estimation for accurate performance prediction. The GPGPU architecture, expounded in Chapter 3, has significantly developed since the introduction of programmable-shader architecture in GeForce 8800 device [82]. The computer architects have unfailingly addressed the ever-growing demands of HPC programmers by offering performance enhancing features including relaxed memory access coalescing rules, L1/L2 caches, large shared memory, dual-warp and quad-warp schedulers, and increased number of double-precision (DP) units. Consequently, the quantitative methods require comprehensive micro-benchmark suites that address these architecture features for precise performance predictions. The accuracy of micro-benchmarks is also highly critical because even the slightest miscalculations may lead to ineffective predictions. We claim that the quantitative methods are better suited for less complex systems, *communication mediums* for instance, which can be represented using a small set of measurable system parameters. The *piecewise analytical models* proposed in Chapter 8 were shown to effectively represent the *medium* behavior using parameters including *overhead, message gap,* and *cut-off messages.*

Opportunities – The *qualitative models* can include additional parameters to obtain a better insight into the system behavior. The quantitative methods also provide significant research

opportunities; architecture specific micro-benchmarks need continual revision with the evolving architecture. The *piecewise analytical models* proposed in this dissertation require *communication medium* characterization at each node configuration, thereby yielding varying model parameter values across the node configurations. The proposed future work includes the development of generic model(s) that relate(s) the system parameters across the node configurations. The high-level abstraction studies conducted in this dissertation research were limited to a single computing system with limited number of host-device pairs. To broaden the scope of performance modeling, the future work also includes comprehensive verification using other computing systems with larger node configurations. The high-level abstraction approach can also be explored for effective design space exploration (DSE).

9.5 SUMMARY

In this chapter, we verified the high-level abstraction of the multi-level performance modeling suite using the SNN-ADF SIA case studies. The qualitative and quantitative modeling approaches were verified for satisfactory estimation of computation and communication components of the SIGE model. The qualitative approach, described by the *subjective-analytical models*, provided highly accurate predictions for the GPGPU device computations. However in Chapter 7, this approach was shown to be error-prone for communication component modeling, given their inability to accommodate additional *medium* parameters.

Unlike qualitative models, the quantitative approach described by *objective-analytical models* yielded high prediction error rates for the GPGPU computations. Because the GPGPU device architecture is rapidly evolving, these prediction methods often miss several hardware parameters that ultimately lead to imprecise predictions. The quantitative modeling approach provided satisfactory prediction results for the communication component. Relatively less

complex systems, including Infiniband and PCI-Ex bus, can be effectively characterized using limited number of measurable parameters.

We suitably combined the *subjective-analytical model* for GPGPU computations and *objective-analytical models* for communications to produce the *hybrid approach*, which provided high quality predictions as discussed in Section 9.3. With this chapter, we conclude the construction and verification of the multi-level performance modeling suite for heterogeneous systems with GPGPU devices. In the next chapter, we summarize the research findings and provide *model selection criteria* based on the performance modeling efforts presented in this dissertation research. We also highlight the dissertation research contributions and discuss future work directions. The A2A roadmap construction is given in Appendix A.

CHAPTER 10

CONCLUSIONS AND FUTURE RESEARCH

In this final chapter, we summarize the dissertation findings, provide *model selection criteria* for effective performance modeling, highlight the important contributions made, and provide directions for future research. The chapter organization is as follows. Section 1 provides chapter-wise summaries, highlighting the key dissertation research findings. This section also serves as an epilogue that connects all of the major developments in this doctoral dissertation research. Based on our performance modeling efforts, we outline the *model selection criteria* in Section 2. The primary contributions and research outcomes are listed in Section 3. The chapter closes with directions for future work in Section 4.

10.1 DISSERTATION SUMMARY

The research presented in this doctoral dissertation seeks to address one of the major challenges faced by the HPC community today: *user-friendly and accurate heterogeneous performance modeling*. Chapter 1 highlights the widespread popularity of heterogeneous architectures such as GPGPU- and FPGA-based clusters in HPC. As asserted in Chapter 1, although these heterogeneous systems offer tremendous performance gains for highly parallel applications, their resources may be under-utilized due to inefficient application mapping, load-balancing, and tuning. These inefficiencies lead to secondary effects including long job queue delays and increased power consumption. Although performance prediction models exist to fine-tune applications, they are seldom easy-to-use and do not address multiple levels of design space abstraction. Due to the above mentioned factors, application developers ultimately face difficulty

in choosing a reliable model for the given design goals. This dissertation research aims to bridge the gap between reliable performance model selection and user-friendly performance analysis. More formally, the doctoral dissertation research goal is to *design a straightforward and accurate performance prediction framework for heterogeneous systems that addresses multiple levels of design space abstraction*, thereby allowing developers to choose an optimal performance model that best fits their design goals. The dissertation research also provides a roadmap for users to perform optimal Application-to-Accelerator (A2A) mapping via appropriate architecture selection and performance prediction (preliminary and advanced). This roadmap is given in Appendix A.

Chapter 2 surveyed the literature, discussing several performance prediction modeling efforts, GPGPU architecture studies, and load-balancing issues. Several qualitative and quantitative performance models were discussed that provide reasonable runtime prediction accuracy. However, it was asserted that these modeling approaches are accompanied with numerous shortcomings. First, the qualitative models require significant knowledge of the computing architecture for accurate runtime prediction. Consequently, this approach can potentially be inaccessible to developers or researchers with limited knowledge of the computer architecture. Second, the quantitative approach relies heavily on micro-benchmarks that measure hardware events, making them prone to miss non-measurable architecture features. Third, the quantitative approach is often tied to a specific GPGPU device. The aim of this dissertation research is to address the above mentioned issues in the form of a *multi-level performance modeling suite that provides an optimal performance modeling strategy for the given design goals and architecture knowledge*. Chapter 2 also examines some of the important analytical models that characterize the network-level behavior. It was highlighted that communication

transactions in heterogeneous systems often exhibit randomness in their behavior, making them non-compliant with the network-level analytical models. To address this problem, we recommended the use of regression-based approaches to model the network-level transactions. The regression analysis of the network-level transactions can either be performed intuitively by mapping the transaction problem to the well-known Michaelis-Menten enzyme kinetics or by employing traditional regression methods such as the log-transformation. We also alluded to the proposal of a simple quantitative model motivated by the existing analytical models for communications.

Chapter 3 acquaints readers with Nvidia's GPGPU architecture (Fermi and Kepler) and CUDA framework used in this dissertation research. The chapter also provides background on the case studies, namely the spiking neural networks (SNNs), large-scale SNN simulations, and non-linear anisotropic diffusion filter (ADF) for massive images. Chapter 4 describes the tested GPGPU clusters: NCSA Forge and GPGPU-augmented Palmetto cluster. The chapter also provides a detailed discussion of the SNN-ADF mapping methodology and orchestration on these clusters. To verify the applicability of SNN-ADF implementations, a thorough performance analysis study was conducted on the Forge GPGPU cluster. This performance analysis was supplemented with the application runtime values, speed-up versus the equivalent MPI-only implementations, and overall runtime breakdown into CPU time, GPGPU time, and MPI communication time for intermediate node configurations. The scalability of the SNN models correlated with their FLOPs/Byte ratio requirements. The most compute-intensive HH model scaled well compared to the lower FLOPs/Byte ratio models. The performance of the SNN models was found to improve generally with both problem size and node scaling. A similar scaling characteristic was observed for the ADF implementation. The performance analysis

exercise establishes *high-data parallelism* as necessary but not sufficient condition for GPGPU system usage. The applications should also yield enough computations to the amortize communication latency for optimal performance.

Chapter 5 describes the *Synchronous Iterative GPGPU Execution* (SIGE) model that serves as the backbone for the proposed modeling suite. The SIGE model describes the execution flow of the *synchronous iterative algorithms* (SIAs) on multi-GPGPU systems by providing a set of equations for estimating the total application runtime. These equations are evaluated using modeling techniques provided by the multi-level suite. The chapter also highlights the goals and usefulness of the SIGE model. The aim of the SIGE model is to generalize the execution flow of deterministic SIAs on multi-GPGPU systems. We asserted that although the SIGE model does not provide explicit optimization guidelines, it is useful for straightforward and insightful design space exploration (DSE). The SIGE model breaks the SIA execution flow into a number of stages, which allows developers to selectively and progressively optimize their applications. In addition to discussing the SIGE model, the chapter also provides a prelude to the multi-level performance modeling suite that is broken into two levels of abstraction, namely the *low-level abstraction* and *high-level abstraction.* The low-level abstraction uses limited implementation details and system information for the application runtime prediction; therefore, partial details of the implementation such as the legacy code, preliminary device kernel, and system specifications must be available. The regression-based analysis best fits the low-level abstraction since it enables the determination of mathematical models that describe the application behavior on the given computing system with a certain degree of confidence. On the contrary, the high-level abstraction seeks to predict the application runtime using algorithm characteristics and system specifications whilst minimizing the reliance on implementation details. This level of abstraction

predicts the computation and communication components of the SIGE model using qualitative and quantitative modeling approaches. The qualitative approach estimates the SIGE model components using *subjective-analytical models* that employ simple analytical functions, thereby avoiding meticulous evaluation of parameters pertaining to the system; whereas the quantitative approach is based on *objective-analytical models*, which predicts these components by measuring system parameters using micro-benchmarks. These two approaches are expounded in Chapters 6 through 9.

Chapter 6 elaborates on the low-level abstraction of the modeling suite. This level of abstraction is composed of the regression-based framework, which aims to model the computations (host and device) and *medium* communications (network-level and PCI-Ex). The regression model development for the SNN-ADF SIAs was described in detail. It was highlighted that simple algorithm parameters, including but not limited to the number of floating-point operations (FLOPs) and computational bytes, can be used to model the host-device computations with a high degree of confidence. We elucidated two regression-based approaches for the network-level and PCI-Ex bus performance modeling: 1) intuitive mapping of the transaction problem to the well-known Michaelis-Menten enzymatic kinetics and 2) log-transformation method. To demonstrate their prediction efficacy, we presented the prediction results for an 8-node scatter throughput problem on the Palmetto cluster using these approaches. It was observed that the Michaelis-Menten kinetics approach better approximates the scatter throughput versus the log-transformation method given its high R^2 value (0.99 vs. 0.93).

We also demonstrated the use of a low-level abstraction approach to perform straightforward and productive GPGPU design space exploration (DSE). This exercise offers an interesting method to perform application tuning and mapping by exploring several possible

implementations (the design space) of an application on the target or potential computing systems. The GPGPU DSE analyzes the runtime performance of several functionally equivalent implementations of an algorithm, thereby ranking the GPGPU design space. This ranking enables developers to select the best implementation for optimal algorithm performance on GPGPU-based systems. Using the low-level abstraction, we exemplified the GPGPU DSE for SNN-ADF SIAs by developing kernel runtime regression equations for three design space implementations; each implementation features an optimization of the GPGPU memory hierarchy.

Chapter 7 provides the preliminary verification results for the low-level abstraction using the SNN-ADF SIAs. This analysis was conducted on the NCSA Forge GPGPU cluster. The regression models for the SNN computation and communication components demonstrated reasonable prediction accuracies (10-12% error rate), discounting a few test cases. Analysis of the results revealed that the complex SNN models with longer execution times have relatively small deviations from the predicted values compared to the deviations observed for simple SNN models with shorter execution times. The computation component regression models were found to be more accurate compared to the communication component models, given the high reproducibility of computations versus the communications. Additional network-level characteristics, such as change in network protocols, may affect the network-level transactions and hence the prediction accuracy. Future work beyond this dissertation includes expansion of the independent variables space for the network-level transactions and inclusion of the protocol parameters for superior performance modeling. The joint regression analysis of the Izhikevich-ADF pair yielded results similar to the four SNN SIAs. We observed high prediction accuracies for the computation component, communication component, and overall runtime prediction.

Chapter 7 also presents the results and analysis for GPGPU DSE using the regression-based framework. This study was conducted on the GPGPU-augmented Palmetto cluster with Kepler K20 devices using the four SNN models and ADF algorithm as case studies. The design space included implementations that feature optimizations of the GPGPU memory hierarchy including global, shared, and texture memories. These implementations were ranked based on the runtime predictions facilitated by the regression-based framework. The GPGPU DSE for the Kepler K20 devices ranked the global memory implementation as the best implementation for the SNN-ADF SIA set. The regression-based framework ranked the design space implementations appropriately for the HH, ML, Wilson, and ADF algorithms, while providing acceptable results for the Izhikevich SNN model.

The Strengths, Weaknesses, and Opportunities (SWO) study follows the comprehensive verification of the low-level abstraction. This level of analysis cogently identifies the merits and demerits of any structured methodology (heterogeneous performance modeling in this research), opening avenues for further refinement and inquiry. We highly recommend SWO studies to the academic community for effective strengths and limitation analysis of any recently developed methodology/theory. The SWO analysis was conducted on the Palmetto cluster using multiple Tesla 2075 and Kepler K20 devices (two GPGPU generations) with host-device pairs varying from 2-node up to 16-node configuration. The ability to provide highly accurate computation component predictions was identified as one of the strengths of the low-level abstraction paradigm. Because the low-level abstraction was tested across computing systems and GPGPU architectures, this modeling approach is expected to span architecture generations. The regression-framework is also expected to be independent of application regularity. We asserted that this framework will also work for complex algorithms where the algorithm complexity is

accounted for by the regression coefficients. As elucidated in Chapter 7, the low-level abstraction also allows for quick and straightforward evaluation of the GPGPU design space. Consequently, the approach extends well to developers and researchers with limited computer architecture knowledge. The slightly error-prone communication component predictions constitute one of the weaknesses of the low-level abstraction paradigm. However, including additional parameters, for example change in network protocol and implicit synchronization, can alleviate this weakness. Additionally, the regression-based framework requires a preliminary GPGPU device implementation and partial access to the computing systems to enable performance predictions, an inherent weakness of this approach. The opportunities for the low-level abstraction modeling paradigm include exploration of additional system parameters for quality predictions and verification with other accelerators and non-regular algorithms.

Chapter 8 elucidates the high-level abstraction that consists of two primary approaches namely, *qualitative modeling* and *quantitative modeling.* The qualitative approach employs *subjective-analytical models* that define simple qualitative relations amongst the parameters to describe the system behavior. On the contrary, the quantitative approach uses *objective-analytical models* that estimate the system performance by measuring hardware-specific events. Using these two approaches, we demonstrated the construction of prediction models for the SIGE model computation and communication components. For the GPGPU *subjective-analytical modeling*, we adapted the analytical model proposed by Schaa et al. [8] that extrapolates the runtime on *M* GPGPU devices using the runtime information from a reference device. We highlighted that this modeling approach lacks statistical rigor because it does not consider several application features (FLOPs, bytes, the number of computational entities, etc.) that affect the GPGPU runtime. To address this issue, we derived simple mathematical relations between

element-throughput, number of computational entities, and execution time on *M* GPGPU devices. For the communication component, we described the Michaelis-Menten enzyme kinetics approach with a *subjective-analytical* perspective. We emphasized that mapping the data transfer problem onto the enzyme kinetics problem is highly intuitive because the data transfer throughput (MB/sec) corresponds to the reaction rate and the data transfer size (MB) corresponds to substrate concentration. Using this qualitative mapping, we developed throughput equations for the *medium* communications in Chapter 6.

Following the qualitative modeling approach, the quantitative modeling approach was discussed that provides an interesting route to performance modeling via system parameter estimation using micro-benchmarks. We discussed the GPGPU analytical model proposed by Hong and Kim [17] that matches our definition of *objective-analytical models* and provided a sub-set of analytical equations given in [17]. To study the *objective-analytical modeling* for communications, we developed a variant of common communication models (*logP, plogP, logGP,* etc.) called the *piecewise analytical model.* This approach describes the *medium* communications using *medium* parameters including *overhead, message gap,* and *cut-off messages.* We elucidated that *medium communication* performance varies across data regions, thereby requiring piecewise modeling for each data region. Using simple micro-benchmarks, we estimated the model parameters for Infiniband (*scatter, gather, Sendrecv*) and PCI-Ex bus (*download* and *read-back*) operations. We observed that the model parameters for *Sendrecv* routine were large when compared to the collective *scatter* and *gather* routines, suggesting the avoidance of this point-to-point routine. In addition to discussing the two primary high-level approaches, we alluded to the *hybrid approach,* a suitable combination of effective qualitative and quantitative methods for high quality performance prediction.

Chapter 9 provides the initial verification of the high-level abstraction models using the SNN-ADF SIAs; this analysis was conducted on the GPGPU-augmented Palmetto cluster with the Kepler K20 devices. Because CPU modeling is suitably performed using modeling strategies given by [6 and 9] that resulted in the development of CPU regression equations; we emphasized modeling the GPGPU computations and *medium* communications. We evaluated each of the primary high-level abstraction approaches for acceptable performance predictions. The *subjective-analytical model* for GPGPU computations yielded superior results for all of the SIA case studies; we reported error rates less than 5% for several tested input sizes and node configurations. Because GPGPU computations usually scale well with the number of processors, the analytical approach is expected to provide satisfactory predictions. The *objective-analytical modeling* for GPGPU computations yielded significant prediction errors. We attributed the high error rates to the missing GPGPU parameters pertaining to instruction caches, L1/L2 caches, shared memory, and warp schedulers. Unlike computations, the communication component predictions were favorable with the *piecewise analytical models.* The Infiniband operations observed satisfactory predictions (less than 10%) at all node configurations, barring a few outliers. The predictions for PCI-Ex bus operations were also acceptable; however, the *read-back* operation yielded error rates over 20% for a few test cases. We attributed this anomaly to the un-measured GPGPU wait time required to service the CPU-host data request, which varies across applications.

Based on the verification results for the high-level abstraction, we asserted that the two primary approaches, when operated alone, are likely to yield coarse-grained application runtime predictions, necessitating a *hybrid approach.* We suitably combined the *subjective-analytical model* for GPGPU computations and *objective-analytical models* for *medium* communications to

perform satisfactory fruitful performance predictions. The initial verification of the *hybrid approach* with SNN-ADF SIAs yielded prediction error rates less than 5%, thereby establishing the viability of this approach for precise predictions.

The SWO analysis for the high-level abstraction approach follows the initial verification. The strengths of qualitative methods include ease-of-use and high accuracy for the computation component. Additionally, this approach is expected to span generations of GPGPU architectures and can also be extended to other computing architectures. However, these methods are prone to overlooking additional system features and variations that may lead to imprecise performance predictions. The error-prone communication component models reinforce this claim. Unlike qualitative methods, the quantitative methods leverage highly accurate predictions for the communication component. This approach also provides significant insight into the computing architecture by measuring the parameters using micro-benchmarks. The quantitative models are expected to offer varied user-friendliness and accuracy depending on the system complexity, an in-built weakness of this modeling paradigm. We asserted that complex systems, including GPGPU devices and CPU hosts, require precise parameter measurements for meaningful predictions. Therefore, erroneous measurements may lead to counterproductive predictions. Given the strengths and weaknesses of these two approaches, we asserted that the qualitative modeling approach is highly suitable for complex systems with reproducible characteristics, GPGPU computations for instance. On the other hand, quantitative methods are more appropriate for less complex systems, communications for instance, which can be described using a small set of measurable parameters. These two assertions were supported by superior performance predictions facilitated by the *hybrid approach*. We discussed several opportunities to improve the high-level abstraction paradigm that includes the use of additional parameters for qualitative

models, continual revision of high-fidelity micro-benchmarks for quantitative models, and comprehensive verification using other computing systems with larger node configurations.

Based on the performance modeling experiences shared in this section, we provide performance model selection criteria that enable effective predictions on heterogeneous systems.

10.2 MODEL SELECTION CRITERIA

As discussed in Chapter 5, the multi-level performance modeling suite is designed with respect to the levels of system abstraction. Given the preliminary implementation knowledge and access to the target system, we assert that the regression-based framework (low-level abstraction) is the most suitable performance modeling approach. This paradigm enables the formulation of mathematical equations using statistically significant system and algorithm parameters, enabling productive performance predictions and fined-tuned DSE. Given the relative simplicity of the regression-based framework, we claim that it is highly suitable for non-Computer Science researchers. Several scientific fields including but not limited to physical and life sciences often use legacy codes to perform large-scale simulations. Because the data used by these codes is ever-growing, constantly updated genome banks [115] for instance, these simulations necessitate code adaptation for HPC systems including GPGPU clusters. Given the knowledge of parallelizable code sections, performance prediction at large node configurations is reliably facilitated by the regression-based framework. We present the first criterion as follows:

Criterion #1: Use the regression-based framework for existing codes to estimate performance at production-scale node configurations.

Unlike low-level abstraction, the high-level abstraction models enable performance modeling with minimum implementation knowledge and system availability. The *objective-analytical*

model for GPGPU computations provides insight into the architecture resource usage by measuring parameter values using micro-benchmarks; this task also enables code optimization for optimal GPGPU resource utilization. Once an initial implementation is identified, the kernel execution time on large computing systems can be predicted using the runtime information from a reference device, for instance the target GPGPU device installed in a desktop machine. The *medium* communication modeling however is most reliably performed using micro-benchmarks on the target system. The second criterion follows as:

Criterion #2: Use the high-level abstraction models when the implementation details and target system availability are limited.

Chapters 6 – 9 comprehensively study the multi-level performance modeling suite, targeting the computation and communication components individually. The following two criteria enable the model selection to address these components.

Criterion #3: Use legacy codes and regression-based framework to model the CPU computations. Either the subjective-analytical model or the regression-based framework can be used for GPGPU computations. The regression-based framework offers additional advantages by statistically incorporating the effects of several algorithm and architecture specific parameters.

Criterion #4: Use the objective-analytical models (piecewise analytical) for medium communications. Although, the subjective-analytical models may also provide satisfactory results, they may not effectively capture the system performance variation with respect to the message size.

10.3 CONTRIBUTIONS AND OUTCOMES

With the preceding discussions as summary, the key objectives addressed by this dissertation research can be summarized as:

1) Development of the *Synchronous Iterative GPGPU Execution* (SIGE) model for multi-GPGPU systems that describes the execution flow of SIAs and provides a foundation for SIA performance analysis on multi-GPGPU systems.
2) Development of a hierarchical, multi-level performance modeling suite for heterogeneous systems that addresses multiple levels of design space abstraction. The multi-level suite allows developers to select a performance model that best fits their design goals. This task is accomplished by presenting the *model selection criteria.*
3) Thorough verification of the performance modeling suite using SIAs with a range of computation-to-communication requirements.
4) The demonstration of the low-level abstraction for well-rounded GPGPU design space exploration (DSE).
5) Presentation of conclusive SWO analysis for each levels of abstraction.
6) Performance analysis of SIAs on the chosen heterogeneous systems to provide insight into the application behavior, thereby assisting in runtime prediction. This exercise also confirms that implementations achieve sufficient efficiency and scaling.
7) A roadmap for users to perform optimal A2A mapping (see Appendix A).

In addition to the above primary contributions, we also include our earlier research achievements that supported this doctoral dissertation research.

1) The two highly important SNN models, namely the Hodgkin-Huxley and Izhikevich models were implemented on several leading multi-core and GPGPU architectures. A

performance analysis study was conducted that highlights the impact of optimizations on the architecture performance for a given application. The contribution was in the form of a conference paper [116]. A subsequent performance analysis study on single- and multi-GPU systems culminated in the form of a Master's Thesis [88].

2) A systematic and exhaustive performance comparison study of the two leading GPGPU programming models, namely the CUDA framework and Open Computing Language (OpenCL) was conducted using the four SNN models as the case studies. The contribution, in the form of a journal paper [101], enables the scientific community to choose the best GPGPU programming paradigm for the given application characteristics.
3) A thorough evaluation of the two leading GPGPU architectures, namely Nvidia's Fermi and AMD's Radeon was performed using the OpenCL programming paradigm. The four SNN models were used as the case studies and several inferences were drawn based on the application-to-accelerator-to-programming model coupling. The contribution studies the effect of the chosen programming model on architecture performance, thereby establishing a tight accelerator-to-programming model coupling for the given application characteristics. The contribution was in the form of a conference paper [117].
4) The above mentioned contributions assisted in the proposal of the fitness model [84 and 118] that ranks the accelerator performance for a given application prior to the actual implementation.

10.4 FUTURE WORK

The research presented in this doctoral dissertation opens several potential research avenues as categorized and discussed below.

Performance Analysis – The SNN-ADF SIAs studied in this research were implemented on GPGPU clusters with 1:1 host-device pairing (see Chapter 4). One area of future work includes the exploration of other cluster configurations with different CPU core-to-GPGPU device ratios per server and investigation of application performance at such configurations. The ADF SIA case study was implemented using the Master-Worker paradigm; it would be interesting to investigate the adequacy of other data partitioning strategies such as the dynamic work pool model for massive image processing applications. Future research can also emphasize further optimization of these SIA implementations, for instance mitigating the large communication overhead associated with large cluster configurations. Specifically for the ADF algorithm, one possible improvement is to require that all processes read their respective image tiles and boundaries from the file, thereby obviating the expensive *scatter* and *Sendrecv* operations. These new performance analysis opportunities favor further improvements in our performance modeling approach.

Enhancing the low-level abstraction – Suggested future work for this level of modeling includes exhaustive analysis of the network-level communications by modeling additional network-level events such as a change in the network protocol and implicit synchronization in collective operations. The GPGPU design space can be extended to include other GPGPU memories such as the local memory and constant memory. The synchronous iterative model and the regression-based framework should be verified with other accelerators and non-regular algorithms to broaden the scope of performance modeling. New GPGPU architecture features, dynamic parallelism in Kepler devices for instance, should be explored with the low-level abstraction. Given the ease-of-use and generic nature of the low-level abstraction, it would be interesting to investigate this approach with other accelerators and computing architectures.

Enhancing the high-level abstraction – The micro-benchmarks used by the high-level abstraction models to describe the GPGPU computations and *medium* communications require frequent revisions. Potential future research efforts should target continual amendment of these micro-benchmarks (*objective models* included) to accommodate new system features. The *piecewise analytical models* developed in this research require communication modeling at each node configuration. Future work includes the development of generic models that relate the system parameters across the node configurations. The micro-benchmarks pertaining to the PCI-Ex bus communications can also include estimation of the GPGPU wait time required to service the CPU-host data request. The high-level abstraction studies can further be consolidated via comprehensive verification using computing systems with larger node configurations. Future work should also address GPGPU DSE facilitated by the high-level abstraction.

BIBLIOGRAPHY

[1] F. T. Ulaby (2006). The Legacy of Moore's Law. Proceedings of the IEEE, Vol. 94, No. 7, July 2006. DOI: 10.1109/JPROC.2006.876941

[2] Many Integrated Core (MIC) Architecture – Advanced. http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html

[3] W.C. Feng, X. Feng, R. Ce (2008). Green Supercomputing Comes of Age. IT Professional, Vol. 10, Issue 1, pp. 17-23

[4] V. Kindratenko, J. Enos, G. Shi, M. Showerman, G. Arnold, J. Stone, J. Phillips, W. Hwu (2009). GPU Clusters for High-Performance Computing. In: Proceedings of the Workshop on Parallel Programming on Accelerator Clusters (PPAC 2009) held in conjunction with Cluster 2009, New Orleans, LA, pp. 1-8, August 31st – September 4th , 2009

[5] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, D. Buell (2008). The Promise of High-Performance Reconfigurable Computing. Computer, Vol. 41, Issue 2, pp. 69-76, 2008

[6] B.J. Barnes, B. Rountree, D.K. Lowenthal, J. Reeves, B.D. Supinski, M. Schulz (2008). A Regression-Based Approach to Scalability Prediction (2008). In: Proceedings of the 22nd Annual International Conference on Supercomputing (ICS 2008), pp. 368-377, June 2008

[7] S.S. Baghsorkhi, M. Delhaye, S.J. Patel, W.D. Gropp, W.W. Hwu (2011). Adaptive Performance Modeling Tool for GPU Architectures. In: Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Vol. 45, Issue 5, pp. 105-114, May 2011

[8] D. Schaa, D. Kaeli (2009). Exploring the Multiple-GPU Design Space (2009). In: Proceedings of the International Symposium on Parallel and Distributed Processing (IPDPS 2009), pp. 1-12, 23rd May – 29th May, 2009

[9] B. Holland, A.D. George, H. Lam, M.C. Smith (2011). An Analytical Model for Multilevel Performance Prediction of Multi-FPGA Systems. ACM Transactions on Reconfigurable Technology and Systems, Vol. 4, Issue 3, Article 27, 28 pages, 2011

[10] Infiniband. http://www.infinibandta.org/

[11] PCI-Express.
http://www.nvidia.com/page/pci_express.html

[12] Dell NVIDIA Linux Cluster Forge.
http://www.ncsa.illinois.edu/UserInfo/Resources/Hardware/DellNVIDIACluster/

[13] V.K. Pallipuram, M.C. Smith, N. Raut, X. Ren (2012). Exploring Multi-Level Parallelism for Large-Scale Spiking Neural Networks. In: Proceedings of the International Conference on Parallel and Distributed Techniques and Applications (PDPTA 2012) held in conjunction with WORLDCOMP 2012, Las Vegas, NV, Vol. 2, pp. 773-779, July 2012

[14] V.K. Pallipuram, N. Raut, X. Ren, M.C. Smith, S. Naik (2012). A Multi-Node GPGPU Implementation of Non-Linear Anisotropic Diffusion Filter. In: Proceedings of the Symposium on Application Accelerators for High-Performance Computing (SAAHPC 2012), Argonne, IL, pp. 11 – 18, 10th July – 11th July 2012

[15] MRI: Acquisition of high-performance computing instrument for collaborative data-enabled science.
http://nsf.gov/awardsearch/showAward.do?AwardNumber=1228312

[16] V.K. Pallipuram, M.C. Smith, N. Raut, X. Ren (2012). A Regression-Based Performance Prediction Framework for Synchronous Iterative Algorithms on GPGPU Clusters. Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.3017

[17] S. Hong, H. Kim (2009). An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness. In: Proceedings of the 36th International Symposium on Computer Architecture, Vol. 37, Issue 3, pp. 152-163, June 2009

[18] Y. Zhang, J.D. Owens (2011). A Quantitative Performance Analysis Model for GPU Architectures. In: Proceedings of the 17th International Symposium on High Performance Computer Architecture (HPCA 2011), pp. 383-393, 12th February – 16th February, 2011

[19] H. Wong, M.M. Papadopoulou, M.S. Alvandi, A. Moshovos (2010). Demystifying GPU Microarchitecture through Microbenchmarking. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2010), pp. 235-246, 28th March – 30th March, 2010

[20] W. Jia, K.A. Shaw, M. Martonosi (2012). Stargazer: Automated Regression-Based GPU Design Space Exploration. In: Proceedings of the IEEE International Symposium on

Performance Analysis of Systems and Software (ISPASS 2012), New Brunswick, NJ, April 1st – April 3rd, 2012

[21] J. Lai, A. Seznec (2012). Break Down GPGPU Execution Time with an Analytical Method. In: Proceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (Rapido 2012), Paris, France, pp. 33- 39, January 2012

[22] NVIDIA CUDA Programming Guide. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf

[23] S. Collange, M. Daumas, D. Defour, D. Parello (2010). Barra: A Parallel Functional Simulator for GPGPU. In: Proceedings of the IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2010), pp. 351 – 360, August 2010

[24] PTX: Parallel Thread Execution ISA version 2.3. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/ptx_isa_2.3.pdf

[25] A. Kerr, G. Diamos, S. Yalamanchili (2009). A Characterization and Analysis of PTX Kernels. In: Proceedings of the IEEE Symposium on Workload Characterization (IISWC 2009), pp. 3 -12, October 2009

[26] CUDA Downloads. http://developer.nvidia.com/cuda/cuda-downloads

[27] UIUC Parboil Benchmarks. http://impact.crhc.illinois.edu/parboil.aspx

[28] K. Kothapalli, R. Mukherjee, M. S. Rehman, S. Patidar, P. J. Narayanan, K. Srinathan (2009). A Performance Prediction Model for the CUDA GPGPU Platform. International Conference on High-Performance Computing (HiPC 2009), pp. 463 – 472, December 16th – December 19th 2009, Kochi, India

[29] L.G. Valiant (1990). A Bridging Model for Parallel Computation, Communications of the ACM 33, 9 (1990), 103 – 111

[30] S. Fortune, J. Wyllie (1978). Parallelism in Random Access Machines. In Proceedings of the ACM STOC (1978), pp. 114 – 118

[31] P.B. Gibbons, Y. Matthias, V. Ramachandran (1999). The Queue-Read Queue-Write PRAM Model: Accounting for Contention in Parallel Algorithms. SIAM J. Comp. 29, 2 (1999), 733 – 769

[32] H. Jia, Y. Zhang, G. Long, J. Xu, S. Yan, Y. Li (2012). GPURoofline: A Model for Guiding Optimizations on GPUs. In: Proceedings of the 18th International Conference on Parallel Processing (Euro-Par 2012), pp. 920 – 932, Rhodes Island, Greece, August 2012

[33] S. Williams, A. Waterman, D. Patterson (2009). Roofline: An Insightful Visual Performance Model for Multi-Core Architectures. Communications of the ACM, Vol. 52, Issue 4, pp. 65 – 76, DOI: 10.1145/1498765.1498785

[34] Z. Cui, Y. Liang, K. Rupnow, D. Chen (2012). An Accurate GPU Performance Model for Effective Control Flow Divergence. In: Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2012), pp. 83 – 94, Shanghai, China, May 2012

[35] Y. Liang, Z. Cui, S. Zhao, K. Rupnow, Y. Zhang, D.L. Jones, D. Chen (2012). Real-time Implementation and Performance Optimization of 3D Sound Localization on GPUs. In: Proceedings of Design, Automation, and Testing in Europe Conference and Exhibition (DATE) 2012, pp. 832 – 835, DOI: 10.1109/DATE.2012.6176610

[36] D. Min, J. Lu, M. Do (2011). A Revisit to Cost Aggregation in Stereo Matching: How Far Can We Reduce its Computational Redundancy? In: Proceedings of IEEE Conference on Computer Vision (ICCV) 2011, pp. 1567 – 1574, DOI: 10.1109/ICCV.2011.6126416

[37] E. Z. Zhang, Y. Jiang, Z. Guo, X. Shen (2010). Streamlining GPU Applications on the Fly: Thread Divergence Elimination through Runtime Thread-Data Mapping. In: Proceedings of the 24th ACM International Conference on Supercomputing (ICS 2010), pp. 115 – 126, DOI: 10.1145/1810085.1810104

[38] K. L. Spafford, J.S. Vetter (2012). Aspen: A Domain Specific Language for Performance Modeling. In: Proceedings of the International Conference for High-Performance Computing, Networking, Storage and Analysis (SC 2012), pp. 1 – 11, Salt Lake City, Utah

[39] P. Guo, L. Wang (2012). Accurate CUDA Performance Modeling for Sparse Matrix-Vector Multiplication. In: Proceedings of the 2012 International Conference on High Performance Computing and Simulation (HPCS 2012), Madrid, Spain, 2012

[40] K. Spafford, J.S. Meredith, J.S. Vetter (2011). Quantifying NUMA Effects in Multi-GPU Systems. In: Proceedings of the 4th Workshop on General Purpose Processing on Graphical Processing Units, Article No. 11, 2011

[41] A. Danalis, G. Marin, C. McCurdy, J.S. Meredith, P.C. Roth, K. Spafford, V. Tipparaju, J.S. Vetter (2010). The Scalable Heterogeneous Computing (SHOC) Benchmark Suite. In: Proceedings of the 3rd Workshop on General Purpose Computation on Graphical Processing Units (GPGPU 2010), pp. 63 – 74, 2010

[42] HPL – A portal implementation of the high-performance linpack benchmark for distributed-memory computers.
www.netlib.org/benchmark/hpl/

[43] L. Chen, O. Villa, S. Krishnamoorthy, G.R. Gao (2010). Dynamic Load Balancing on Single- and Multi-GPU Systems. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2010), pp. 1 – 12, April 2010

[44] S. Martin, H.W. Shen, P. McCormick (2010). Load-Balanced Isosurfacing on Multi-GPU Clusters. In: Proceedings of Eurographics Symposium on Parallel Graphics and Visualization (EGPGV 2010), pp. 91 – 100, May 2010

[45] D. Cederman, P. Tsigas (2008). On Sorting and Load Balancing on GPUs. ACM SIGARCH Computer Architecture News, Vol. 36, Issue 5, pp. 11 – 18, December 2008

[46] OpenCL Khronos Group (June 2013).
http://www.khronos.org/opencl/

[47] G. Khanna, J. McKennon (2010). Numerical Modeling of Gravitational Wave Sources Accelerated by OpenCL. Comput Phys Commun 181(9), 1605 – 1611

[48] K. Karimi, N.G. Dickson, F. Hamze (2010. A Performance Comparison of CUDA and OpenCL. The Computing Research Repository (CoRR), arXiv:1005.2581

[49] P. Du, R. Weber, S. Tomov, G. Peterson, J. Dongarra (2010). From CUDA to OpenCL: Towards a Performance –Portable Solution for Multi-Platform GPU Programming. Journal of Parallel Computing, pp. 391 – 407, Vol. 38, Issue 8, DOI: 10.1016/j.parco.2011.10.002

[50] Basic Linear Algebra Sub-Programs (BLAS).
http://www.netlib.org/blas/

[51] NVIDIA's Next Generation CUDA Compute Architecture: Fermi.
http://www.nvidia.com/object/fermi-architecture.html

[52] ATI Radeon HD 5870 Graphics.
www.amd.com

[53] T.D. Han, T.S. Abdelrahman (2011). Hicuda: High-Level GPGPU Programming. IEEE Transactions on Parallel and Distributed Systems, Vol. 22, no. 1, 78 – 90

[54] S. Lee, R. Eigenmann (2010). OpenMPC: Extended OpenMP Programming and Tuning for GPUs. In: Proceedings of the 2010 ACM/IEEE Conference for High-Performance Computing, Storage, Networking and Analysis, pp. 1 – 11, DOI: 10.1109/SC.2010.36

[55] PGI Accelerator.
www.pgroup.com/resources/accel.htm

[56] OpenACC Home.
http://www.openacc-standard.org/

[57] S. Lee, J.S. Vetter (2012). Early Evaluation of Directive-Based GPU Programming Models for Productive Exascale Computing. In: Proceedings of the IEEE/ACM International Conference on High-Performance Computing, Networking, Storage and Analysis. Article no. 23

[58] K. Spafford, J.S. Meredith, S. Lee, D. Li, P.C. Roth, J.S. Vetter (2012). The Tradeoffs of Fused Memory Hierarchies in Heterogeneous Architectures. In: Proceedings of the ACM Computing Frontiers (CF), Cagliari, Italy, 2012

[59] Melissa C. Smith (2003). Analytical Modeling of High Performance Reconfigurable Computers: Prediction and analysis of system performance. Ph.D. Dissertation, The University of Tennessee, Knoxville, 2003

[60] B. Holland, K. Nagarajan, A.D. George (2009). RAT: RC Amenability Test for Rapid Performance Prediction. ACM Transactions on Reconfigurable Technology and Systems, Vol. 1, Issue 4, Article 22, pp. 1 – 30, 2009

[61] A. Alexandrov, M.F. Ionescu, K.E. Schauser, C. Scheiman (1995). LogGP: Incorporating Long Messages into the LogP Model: One Step Closer towards a Realistic Model for Parallel Computation. Proceedings of the 7th Annual ACM Symposium on Parallel Algorithms and Architectures, DOI: 10.1145/215399.215426, pp. 95 – 105, 1995

[62] D. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, and T. von Eicken (1993). LogP: Towards a Realistic Model of Parallel Computation. In:

Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 1 – 12, DOI: 10.1145/155332.155333, 1993

[63] T. Kielman, H.E. Bal, K. Verstoep (2000). Fast Measurement of LogP Parameters for Message Passing Platforms. In: Proceedings of the 15th Worskshop on Parallel and Distributed Processing (IPDPS 2000), pp. 1176 – 1183, 2000

[64] T. Hoefler, A. Lichei, W. Rehm (2007). Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks. In: Proceedings of the Parallel and Distributed Processing Symposium (IPDPS 2007), pp. 1 – 8, March 2007

[65] C.A. Moritz, M.I. Frank (2001). LogGPG: Modeling Network Contention in Message-Passing Programs. IEEE Transactions on Parallel and Distributed Systems, Vol. 2, Issue 4, pp. 404 – 415, 2001

[66] F. Ino, N. Fujimoto, K. Hagihara (2001). LogGPS: A Parallel Computational Model for Synchronization Analysis. In: Proceedings of the 8th ACM SIGPLAN symposium on Principles and Practices in Parallel Programming, pp. 133 – 142, 2001

[67] T. Hoefler, T. Mehlan, F. Mietke, W. Rehm (2006). LogfP: A Model for Small Messages in Infiniband. In: 20th International Symposium on Parallel and Distributed Processing (IPDPS 2006), April 2006

[68] L. Michaelis, M.L. Menten (1913). Die kinetic der invertinwirkung, Biochem. Z, Vol. 49, pp. 333 – 369, 1913

[69] E.M. Izhikevich (2004). Which Model to Use for Cortical Spiking Neurons? IEEE Transactions on Neural Networks. Vol. 15, Issue 5, pp. 1063-1070

[70] J.W Sohn, B.T. Zhang, B.K. Kaang (1999). Temporal Pattern Recognition Using a Spiking Neural Network with Delays. In: Proceedings on the International Joint Conference on Neural Networks (IJCNN 1999), Vol. 4, pp. 2590 – 2593, 1999

[71] D. Surdilovic, J. Radojicic, M. Schulze, M. Dembek (2008). Modular Hybrid Robots with Actuators and Joint Stiffness Control. In: Proceedings of the 2nd IEEE RAS & EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob 2008), pp. 289 – 294, October 2008

[72] C.E. Johnson (2011). Spiking Neural Networks and their Applications. Ph.D. dissertation, Missouri Institute of Science and Technology, 2011

[73] C. Johansson, A. Lansner (2007).Towards Cortex Sized Artificial Neural Systems. Neural Networks, 20(1), pp. 48-61, 2007

[74] R. Ananthanarayanan, S.K. Esser, H.D. Simon, D.S. Modha (2009). The Cat is Out of the Bag: Cortical Simulations with 10^9 Neurons, 10^{13} Synapses. In: Proceedings of Super-Computing 2009, Portland OR, 2009

[75] Powerful Blue Gene/P supercomputer at Argonne to address most challenging science problems.
www.anl.gov

[76] J.M. Nageswaran, N. Dutt, J.L. Krihmar, A. Nicolau, A.V. Veidenbauma (2009). A Configurable Simulation Environment for the Efficient Simulation of Large-Scale Spiking Neural Networks on Graphics Processors. Special issue of Neural Network, Elsevier, Vol. 22 (5-6), pp. 791 – 800, 2009

[77] B. Han and T.M. Taha (2010). Neuromorphic models on GPGPU cluster. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2010), pp. 1-8, 2010

[78] A.C. Sobieranski, L. Coser, M.A.R. Dantas, A. Wangenheim, E. Comunello (2008). An Anisotropic Diffusion Filtering Implementation to Execute in Parallel Distributed Systems. In: Proceedings of the 11th International Conference on Computational Science and Engineering Workshops, 2008

[79] G. Burns, R. Daoud, J. Vaigl (1994). LAM: An Open Cluster Environment for MPI. In: Proceedings of Supercomputing Symposium, pp. 379-386, 1994

[80] L. Yuangfeng, Z. Yan (2011). Accelerating Fuzzy Adaptive Anisotropic Diffusion on GPU. In: Proceedings of the 10th International Conference on Electronic Measurement & Instruments, 2011

[81] S. Philip, B. Summa, V. Pascucci, P.-T. Bremer (2011). Hybrid CPU-GPU Solver for Gradient Domain Processing of Massive Images. In: Proceedings of the 17th International Conference of Parallel and Distributed Systems (ICPADS 2011), 2011

[82] GeForce 8800 Technical Briefs.
http://www.nvidia.com/page/8800_tech_briefs.html

[83] Nvidia's Next Generation CUDA Compute Architecture: Kepler GK110 – Whitepaper http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf

[84] A.L. Hodgkin, A.F. Huxley (1952). A Quantitative Description of Membrane Current and Application to Conduction and Excitation in Nerve. Journal of Physiology, Vol. 117, pp. 500-544, 1952

[85] C. Morris, H. Lecar (1981). Voltage Oscillations in the Barnacle Giant Muscle Fiber. Biophysical Journal, Vol. 35, Issue 1, pp. 193-213, 1981

[86] H.R. Wilson (1999). Simplified Dynamics of Human and Mammalian Neocortical Neurons. Journal of Theoretical Biology, Vol. 200, Issue 4, pp. 375-388, 1999

[87] E.M. Izhikevich (2003). Simple Model to Use for Cortical Spiking Neurons. IEEE Transactions on Neural Networks, Vol. 14, Issue 5, pp. 1569-1572, 2003

[88] V.K. Pallipuram (2010). Acceleration of Spiking Neural Networks on Single-GPU and Multi-GPU Systems. Master's Thesis, Clemson University, May 2010

[89] A. Gupta, L. Long (2007). Character Recognition Using Spiking Neural Networks. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN 2007), pp. 53 – 58, August 2007

[90] R.C. Gonzales, R.E. Woods. Digital Image Processing. 2nd Edition, ISBN – 10: 0201180758

[91] H. Romeny, M. Bart. Geometry-Driven Diffusion in Computer Vision. Vol. 1, ISBN 978-0-7923-3087-5

[92] D.M. Tsai, W.Y. Chiu, W.C. Li (2010). Anisotropic Diffusion-Based Detail-Preserving Smoothing for Image Restoration. In: Proceedings of the 17th IEEE International Conference on Image Processing (ICIP 2010), September 2010

[93] H. Hildebrandt, K. Polthier (2004). Anisotropic Diffusion Filtering of Non-Linear Surface Features. Computer Graphics Forum, Vol. 23, Issue 3, pp. 391 – 400, 2004

[94] J. Weickert (1998). Anisotropic Diffusion in Image Processing. B.G. Teubner, Stuttgart, 1998

[95] A. Dumitras (2004). An Automatic Method for Unequal and Omni-Directional Diffusion Filtering of Video Sequences. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2004), Vol. 3, pp. 317 – 320, May 2004

[96] J. Weickert (1996).Theoretical Foundations of Anisotropic Diffusion in Image Processing. In: Proceedings of the 7th Theoretical Foundations of Computer Vision, 1996, ISBN: 3-211-82730-7, 1996

[97] W. Wu, H. Liu (2008). Noise Removal using Nonlinear Diffusion Filtering Based on Statistic-Local Open System. In: Proceedings of the Congress on Image and Signal Processing (CISP), Vol. 3, pp. 372 – 378, May 2008

[98] P. Perona, J. Malik (1990). Scale Space and Edge Detection using Anisotropic Diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 2, Issue 7, pp. 629 – 639, July 1990

[99] Open MPI: Open Source High-Performance Computing. http://www.open-mpi.org/

[100] Message Passing Interface (MPI) Standard. http://www.mcs.anl.gov/research/projects/mpi/

[101] V.K. Pallipuram, M.A. Bhuiyan, M.C. Smith (2011). A Comparative Study of GPU Programming Models and Architectures using Neural Networks. Journal of Supercomputing, pp. 46, DOI: 10.1007/s11227-011-0631-3, 2011

[102] T.D. Han, T.S. Abdelrehman (2011). Reducing Branch Divergence in GPU Programs. In: Proceedings of the 4th Workshop on General Purpose Processing on Graphical Processing Units. 2011

[103] MathWorks – MATLAB and Simulink for Technical Computing. www.mathworks.com

[104] E. Parzen (1962). On Estimation of a Probability Density Function and Mode. Annals of Mathematical Statistics, Vol. 33, No. 3, pp. 1065 – 1076, 1962

[105] S.S. Hampton, S.R. Alam, P.S. Crozier, P.K. Agarwal (2010). Optimal Utilization of Heterogeneous Resources for Biomolecular Simulations. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1 – 11, November 2010

[106] T.L. Williams, R.J. Parsons (2000). The Heterogeneous Bulk Synchronous Parallel Model. In: Proceedings of the 15th IPDPS 2000 Workshops on Parallel and Distributed Processing (IPDPS 2000), pp. 102 – 108, 2000

[107] D.G. Kleinbaum, L.L. Kupper, K.E. Muller, A. Nizam (1998). Applied Regression Analysis and Other Multivariable Methods. 3rd Edition, Duxbury Press, 1998

[108] W. Mendenhall, T. Sincich (2003). A Second Course in Statistics: Regression Analysis, 6th Edition, Pearson Education, New Jersey, 2003

[109] R core team (2012). R: A language and environment for statistical computing. R foundation for statistical computing, Vienna, Austria, ISBN 3-900051-07-0. http://www.R-project.org/

[110] The R manual. http://cran.r-project.org/manuals.html

[111] M.A. Bhuiyan (2011). Performance Analysis and Fitness of GPGPU and Multi-Core Architectures for Scientific Applications. Ph.D. Dissertation, Clemson University, December 2011

[112] N. S. Raut (2013). Statistical Regression Methods for GPGPU Design Space Exploration. Master's Thesis, Clemson University, August 2013

[113]V.K. Pallipuram, N. Raut, M.C. Smith (2013). Regression-Based Framework for GPGPU Design Space Exploration. *Under Preparation*

[114] I. Bratko, D. Suc (2003). Learning Qualitative Models. AI Magazine. Vol. 24, No. 4

[115] National Center for Biotechnology Information (NCBI). http://www.ncbi.nlm.nih.gov/

[116] M.A. Bhuiyan, V.K. Pallipuram, M.C. Smith (2010). Acceleration of Spiking Neural Networks in Emerging Multi-core and GPU Architectures. In: Proceedings of the IEEE International Workshop on High Performance Computational Biology in conjunction with IEEE International Conference on Parallel and Distributed Systems (IPDPS 2010), pp. 1 – 8, 19th April – 23rd April 2010

[117] V.K. Pallipuram, M.A. Bhuiyan, M.C. Smith (2011). Evaluation of GPU Architectures Using Spiking Neural Networks. In: Proceedings of the Symposium on Application

Accelerators in High-Performance Computing (SAAHPC 2011), pp. 93 – 102, 19th July – 21st July 2011

[118] M.A. Bhuiyan, M.C. Smith, V.K. Pallipuram (2010). Performance, Optimization, and Fitness: Connecting Applications to Architectures. Concurrency and Computation: Practice and Experience. Vol. 23, Issue 10, pp. 1066 – 1100, July 2011

[119] TOP500 Supercomputer Sites. http://www.top500.org/

[120] CAREER: Harnessing Hybrid Computing Resources in PetaScale Computing and Beyond. http://www.nsf.gov/awardsearch/showAward?AWD_ID=1149644&HistoricalAwards=false

APPENDIX A

TYING-IT-ALL-TOGETHER: APPLICATION-TO-ACCELERATOR ROADMAP

Heterogeneous systems continue to exhibit several hundred thousands of computing nodes, each equipped with multiple accelerators and powerful host processors. Each year, the Top500 list [119] showcases new HPC systems that persistently strive to push the computational limits. However, inefficiencies including application-to-accelerator mismatch, improper application tuning and load-balancing result in counterproductive resource utilization, ultimately leading to economic loss. Deployment of an optimal application on the computing system is a challenge continuously presented to the HPC community. The common users of these HPC systems include scientists and researchers that often require guidelines for an optimal application-to-accelerator (A2A) mapping. The research presented in this dissertation addresses some of the stated goals in the NSF Career Award #1149644 [120]; these research goals include coarse-grained architecture selection, fine-grained performance prediction, and taxonomy of application and architecture characteristics. The ultimate goal is to enable researchers and scientists to productively optimize and maintain their codes. To address the above stated tasks, we provide a preliminary A2A roadmap that serves as an outline for further research. Although the roadmap is constructed with respect to the heterogeneous systems including GPGPU devices, we assert that this philosophy can be also extended to other current and future HPC systems. Figure A.1 provides the constructed roadmap; we discuss each of the listed milestones.

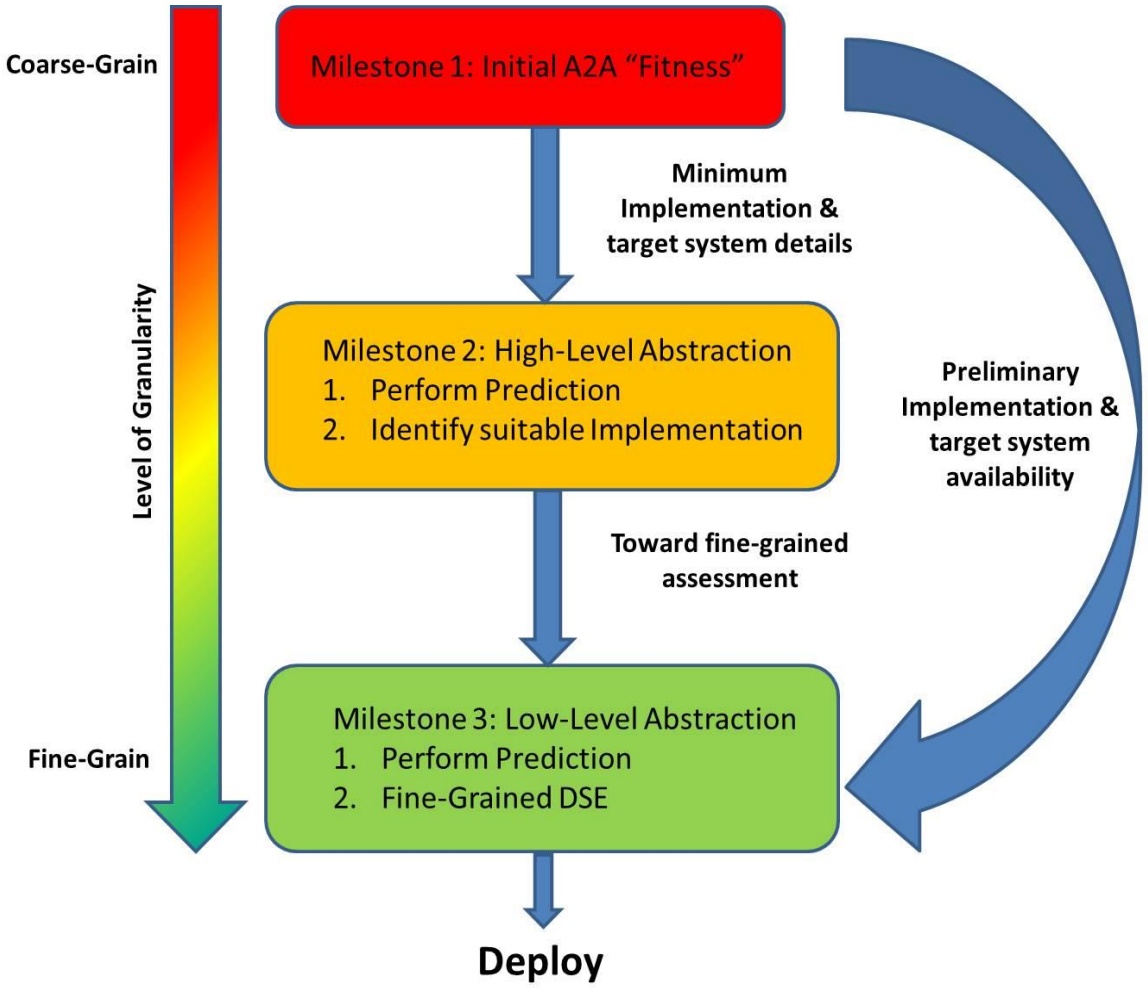

Figure A.1 Application-to-Accelerator Roadmap

Milestone 1 Initial A2A "Fitness" – The aim of this milestone is to identify an initial application-to-accelerator mapping facilitated by the *Fitness Model* proposed by Bhuiyan [111]. This model finds an optimal architecture match for the given algorithm by predicting the coarse-grained application runtime. This exercise is performed by evaluating the scalar product of two vectors: *application vector* and *accelerator vector*. The components of *application vector* include application-specific parameters such as the number of single-precision floating point operations (FLOPs), number of double-precision FLOPs, bytes required by the processing cores from the device memory, and host-device transfer bytes. The corresponding components of *accelerator*

vector include single-precision FLOP time, double-precision FLOP time, per byte device-to-device transfer time, and per byte host-device transfer time. The accelerator with the minimum scalar product value is deemed to be the best fit for the chosen algorithm. This coarse-grained application-to-accelerator mapping is straightforward; the *vector* components are easily obtained via algorithm study and accelerator specifications.

We recommend this A2A mapping prior to the algorithm testing and subsequent performance modeling. As asserted in this dissertation, the highly-parallel nature of an algorithm is a necessary but not sufficient condition to justify the use of massively-parallel computing systems including GPGPU architectures. Because GPGPU devices are throughput oriented architectures, the applications should also yield significant amount of computations to amortize the communication latency. Using the SNN-SIA as case studies, the authors in [118] matched the highly computation- and communication-intensive HH model with the GPGPU architecture; whereas, the computationally-efficient Izhikevich model was appropriately mapped to the multi-core architectures. This finding supports our claim that the massively-parallel and computationally-intensive nature of algorithms appropriately justify the use of GPGPU-based systems. Since these algorithm features vary across applications, the *Fitness Model* offers a reliable metric to assess their impact on A2A mapping. Future work includes expansion of the *application* and *architecture vector space* to further consolidate the initial A2A mapping.

Milestone 2 High-Level Abstraction – The research presented in this dissertation details several high-level abstraction approaches to model the computations and communications in the given algorithm. This level of analysis is highly recommended when knowledge of the initial implementation and target system availability are limited. Using the appropriate qualitative and quantitative approaches, significant performance insight can be obtained that enables developers

to design an optimal implementation for the potential computing system. The readers are referred to Chapters 8 and 9 for this detailed study.

Milestone 3 Low-Level Abstraction – Commonly, users and developers often possess significant knowledge of their legacy codes along with the code sections that could benefit from improved compute performance. Therefore, following an initial A2A mapping, we recommend the use of a regression-based framework (low-level abstraction) for straightforward runtime prediction and fine-tuned DSE. This level of analysis also follows the high-level abstraction for fine-grained performance assessment. The low-level abstraction studies are provided in Chapters 6 and 7. Future work pertaining to these two milestones is elaborated in Chapter 10. Finally, enhancement of the A2A roadmap to accommodate other computing architectures and classes of algorithms is of significant interest that creates lucrative research opportunities.

APPENDIX B

LIST OF FREQUENTLY USED ACRONYMS

A2A	Application-to-Accelerator
ADF	Anisotropic Diffusion Filter
AMD	Advanced Micro Devices
BSP	Bulk Synchronous Parallel
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CWP	Computation Warp Parallelism
D2H	Device-to-Host
DP	Double-Precision
DSE	Design Space Exploration
DSL	Domain Specific Language
FFT	Fast Fourier Transform
FLOPs	Floating-Point Operations
FLOPS	Floating-Point Operations per Second
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
GPGPU	General Purpose Graphical Processing Unit
H2D	Host-to-Device
HBSP	Heterogeneous Bulk Synchronous Parallel
HH	Hodgkin-Huxley
HPC	High-Performance Computing
HPL	High-Performance Linpack
HPRC	High-Performance Reconfigurable Computing
MIC	Many Integrated Core
ML	Morris-Lecar
MPI	Message Passing Interface
ms	Milliseconds
MWP	Memory Warp Parallelism
NCSA	National Center for Supercomputing Applications
NUMA	Non-Uniform Memory Access
OpenCL	Open Computing Language
PCI-Ex	Peripheral Component Interconnect Express
PD	Probability Distribution

PDE	Partial Differentiation Equation
PDF	Probability Density Function
PSNR	Peak Signal-to-Noise Ratio
PTX	Parallel Thread eXeution
RC	Reconfigurable Computing
RAT	RC Amenability Test
RATSS	RC Amenability Test for Scalable Systems
RCS	Reduced Conditional Statement
SHOC	Scalable Heterogeneous Computing Benchmark Suite
SIA	Synchronous Iterative Algorithm
SIGE	Synchronous Iterative GPGPU Execution
SIMD	Single Instruction Multiple Data
SM	Shared Memory
SMP	Streaming Multi-Processor
SMX	Next Generation Streaming Multi-Processor
SNN	Spiking Neural Network
SP	Software Prefetching
SWO	Strengths, Weaknesses, and Opportunities
TA	Texture Addressing
TEG	Timing Estimation Tool
TF	Texture Fetch