COMPUTATIONAL STEERING FOR SPIKE-COUPLED NEURONAL NETWORK SIMULATIONS ON HIGH-PERFORMANCE COMPUTING RESOURCES

by

Sean McDaniel-Gray

A dissertation submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

Summer 2018

COMPUTATIONAL STEERING FOR SPIKE-COUPLED NEURONAL NETWORK SIMULATIONS ON HIGH-PERFORMANCE COMPUTING RESOURCES

by

Sean McDaniel-Gray

Approved: _____

Kathleen McCoy, Ph.D.
Chair of the Department of Computer & Information Sciences

Approved: _____

Babatunde Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____

Douglas J. Doren, Ph.D.
Interim Vice Provost for Graduate and Professional Education

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Michela Taufer, Ph.D.
Professor in charge of dissertation

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Sunita Chandrasekaran, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Li Liao, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

Alfred B. Yu, Ph.D.
Member of dissertation committee

I certify that I have read this dissertation and that in my opinion it meets the academic and professional standard required by the University as a dissertation for the degree of Doctor of Philosophy.

Signed: _____

David L. Boothe, Ph.D.
Member of dissertation committee

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Michela Taufer and my other committee members: Dr. David Boothe, Dr. Sunita Chandrasekaran, Dr. Li Liao, and Dr. Alfred Yu. Additionally, many thanks to the members of the Global Computing Lab for their collaboration and friendship.

Moreover, I would like to thank Josh Crone for his mentorship and support. Thank you for allowing me to share your simulation hours on DOD HPC resources. Your support made this thesis possible. Also, thank you to Dale Shires and Song Park for giving me the opportunity to work at ARL and pursue my research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

One significant challenge in neuroscience is understanding the cooperative behavior of large numbers of neurons. Neuroscientists have long postulated that information processing is a result of the synchronicity of spike trains of large groups of connected neurons. How this synchronous behavior produces complex behavior such as planning, language, and emotion is not well understood. Models of brains neuronal networks allow scientists to explore the impact of differential neuronal connectivity using analysis techniques and information not available experimentally. However, encoding more and more biophysically realistic neurobiological models into computational simulations combine increasing computing and data requirements. High-performance computing (HPC) has enabled the simulation of brain at increasing levels of fidelity; the amount of data produced by simulations on HPC systems is becoming increasingly difficult to save and transform into scientific insights because of the memory/storage characteristics of the systems.

The ability to process, analyze, and produce substantive scientific inferences on data is becoming increasingly difficult. Furthermore, the growth in complexity of brain models makes finding optimal input variable configurations hard, as performing efficient parameter sweeps become impractical due to the increasing number of input parameters and the need of narrowing down the parameters values. Coupling simulation with analysis has been widely investigated in applications such as computational fluid dynamics and other numerical simulations codes, but little work has been done in integrating real-time analysis and interactive steering with brains neural network simulations. Analysis of data produced by brain models is still relegated to traditional post-processing and sequential scientific workflows.

In this thesis, we claim that new workflows that leverage real-time analysis and computational steering methods are required to overcome the increasing divide between computational and I/O subsystem performance. We claim that there exists a need to transform the end-to-end scientific HPC workflow from one that is non-transparent, trial-and-error based, and static to one that is investigative, hypothesis-driven, and adaptive. To this end, we design, implement, and evaluate a distributed computational steering environment (CSE) prototype for the GEneral NEural SImulation System (GENESIS) on HPC systems. First, we investigate how the diversification of HPC hardware along with increasing model complexity impact performance and data generation. We show that increasingly biophysically realistic brain models are computationally feasible but require HPC resources to mitigate the burgeoning computing and data requests of the associated simulations. Second, we investigate the integration of an *in situ* analysis approach with a simulated multi-compartmental model of neocortex. We assess our approach using both qualitative and quantitative methods. We show that by eliminating the need for data movement, using only a local view of our data, results in comparable scientific insights to those that are obtained via a global post-simulation analysis. Third, we design, implement and demonstrate a working prototype of a CSE that leverages data collected at run-time via *in situ* analysis, to steer a GENESIS simulation in real time. We assess and quantify simulation perturbation (i.e., overhead) and presentation latency of our prototype when steering a simulated neuronal networks models on GENESIS.

The primary goal of this thesis is to show how the use of an environment such as our CSE can support dynamic workflows in brains neural network simulation via computational steering and can mitigate resource usage associated with more naive trial-and-error based approaches. Our work is the first step towards a change in thinking from traditional static, post-simulation to dynamical, adaptive steering simulations of realistic neurobiological models.

Chapter 1

THESIS OVERVIEW

1.1 Problem, Motivation, and Proposed Solution

Numerical simulations of neuronal networks are critical in addressing a significant challenge in neuroscience, which is understanding the cooperative behavior of large, diverse groups of neurons [1]. At present, little is known about how neurons work together to form complex behavior such as speech, memory, and emotion. To gain knowledge and better insight into the fundamentals of information processing in the brain, scientists rely on large neuronal network models that simulate more than 10^5 neurons and 10^9 synaptic connections [2, 3, 4]. However, current networks are far from approaching adequate representations of the biological complexity of the human brain concerning the number of neurons and connections. Present HPC hardware fails to keep pace with the increasing computational requirements of these simulations [5, 6].

Post-petascale computing is imperative, in meeting the computational resource requirements of brain scale networks. New advancements in simulation technology must be achieved that allows for numerical models to leverage the maximum compute capability of new hardware efficiently. Also, increasingly sophisticated models demand more from I/O subsystems and, as a consequence, these systems are overburdened by the sheer amount of data being generated. The increasing compute-I/O gap has caused traditional domain-specific analysis methods to cease to scale and efficiently meet these demands [7, 8, 9]. New interactive and adaptive analysis workflows are needed to analyze the data at its locale of generation. Resulting in a reduction of data movement and granting the ability to perform "hypothesis-driven" data discovery.

In this thesis, we propose a working prototype of a computational steering environment (CSE) for simulated neuronal network models on the GEneral NEural SImulation System (GENESIS). Our CSE enables the integration of flexible high performance computing applications and analysis frameworks in a turnkey manner, facilitating dynamic and adaptable analytical workflows. Specifically, our approach moves in three directions: we assess the impact of running simulations of neuronal networks on HPC systems; we integrate *in situ* analysis into the simulations; and we use scientific knowledge, captured with the *in situ* analysis, to steer the simulations.

1.1.1 Simulating Neuronal Networks on HPC Resources

Simulation of neuronal networks is fast becoming the primary vehicle for enabling scientific discovery in the neuroscience research community. The increasing use of simulation for science is possible due to advances in computer hardware and simulation technologies for the modeling of small sub-cellular processes up to brain-scale networks [10]. Neuroscientists use simulated models of neurobiological processes to validate theory with observation and investigate the relationship between structure and function. Additionally, these models are used to study highly dynamical systems where *in vitro* and *in vivo* experimental data is not always readily available. The human brain is complex and contains over 100 billion interconnected cells. However, with current technology, it is impossible to achieve simulated networks at this scale.

The primary challenge for an application implementing neuronal network models based on scaled down models of neurons and synapses is the number of network elements that have to reside in memory at any one time [6]. Research in simulation technology has focused on the design of data structures to enable the simulation of network models at increasing sizes [5, 10, 11, 12, 13]. In addition to the challenge of representing large numbers of network elements in memory, the time needed to instantiate a network is of concern [10]. As models become larger so do the computational requirements and time required for instantiation, which has the potential to affect the simulation performance negatively.

In this thesis, we analyze the impact of HPC resources (i.e., single fat nodes and high-end clusters) on performance, data generation, and science delivered by GENESIS for increasingly large and complex models of the brain's neocortex. We explore the relationship between model fidelity (i.e., the level to which our model successfully reproduces the behavior of neuronal networks in nature) and performance, by increasing the number of cells simulated and the amount of synaptic connections between the cells in more advanced models and we run the simulations on high-end clusters.

1.1.2 Integrating *In situ* Analysis into Simulated Neuronal Networks

Advancements in technology have enabled increases in the fidelity and resolution of neuronal network models [14]. A consequence of this has been a steep hike in the rate at which neuronal network models generate data [15]. Traditionally, analysis of neuronal network models is accomplished through the use of well-known, widely-used post-simulation analysis workflows. Conventional workflows in high performance computing involve the preparation of an application's input, configuration of model parameters, execution of the simulation, and concludes with the analysis and/or visualization of the simulation's results. This is performed as a post-processing step, usually done in a sequential fashion [16, 17, 18]. However, the large amount of data produced by high fidelity models is becoming increasingly difficult to save and transform into scientific insights for runtime simulations. One attractive solution is to modify the data analysis workflow from one that is accomplished post-simulation to one that is completed *in situ* at the cost of confining the analysis to the level of local data rather than a single global view. The missing global view of data has the potential to introduce inaccuracies in the simulation's findings.

In this thesis, we present the integration of *in situ* analysis into a simulation of the electrical activity of a brain's neuronal network. We evaluate the accuracy of our *in situ* analysis versus the traditional post-simulation analysis using both qualitative and quantitative methods. We show that we gain meaningful global insights from local data that are comparable to those that are obtained post-simulation using global data.

1.1.3 Steering Simulated Neuronal Networks on HPC Resources

Having the ability to monitor model behavior and attributes at runtime brings many advantages: tuning model parameters can be accomplished more efficiently, allowing combinations of model input parameters that yield nonsensical behavior to be identified sooner; "what if" studies are possible leading to additional hypotheses being formed instead of being generated from scratch [19]; early termination of simulation if desired behavior is not observed. By integrating *in situ* analysis into our simulated neuronal network model, we improve upon the traditional method of analysis that is growing in impracticality. However, permitting the ability to monitor neuronal network simulation behavior at runtime is only a partial solution to the impending compute-I/O gap that is being exacerbated by every generation of new hardware [20, 14]. We extend a step further by augmenting our *in situ* analysis with the ability to computationally steer the simulation at runtime. Computational steering enables one the ability to direct and re-direct the execution flow of an HPC application on-line by changing application-defined control parameters using a steering agent.

In this thesis, we propose, develop, and implement a computational steering environment (CSE) for simulated neuronal networks executed on GENESIS. Our CSE performs three primary functions. First, it monitors, extracts, and manages simulation output at runtime. Second, it allows for the distributed analysis of intermediate simulation results. Third, it enables the modification of model parameters at execution time. We demonstrate our CSE by steering the electrical activity (i.e., excitation/inhibition balance) of a simulated neuronal network at runtime through the manipulation of excitatory and inhibitory neuronal connections. To assess our CSE, we examine it on two points. First, we evaluate its' ability to govern desired model behavior. Second, we evaluate simulation perturbation as it relates to the overhead introduced by the steering of our model. The results show that our method successfully directs model execution flow while minimally impacting simulation performance.

1.2 Thesis Statement

In this thesis, we claim that neuronal network simulations must integrate *in situ* analysis and simulation steering workflows to increase scientific throughput and improve utilization of current and next-generation HPC systems. To this end, there is the need to transform the end-to-end simulation workflow from one that is non-transparent, trial-and-error based, and static to one that is investigative, hypothesis-driven, and adaptive. To validate the thesis statement, we:

- study the impact of HPC resources on performance, data generation, and science delivered by GENESIS for increasingly complex neuronal network models;
- integrate *in situ* analysis into a simulated neuronal network model, evaluating our analysis using both qualitative and quantitative methods; and
- demonstrate the effectiveness of a distributed computational steering environment prototype designed to perform analysis and steer neurobiological simulations on GENESIS.

1.3 Contributions

The contributions of this dissertation are as follows: In regards to the investigation of the impact of HPC resources on performance, data generation, and science delivered by GENESIS this thesis makes two contributions, we:

- study the impact of platforms (i.e., single fat node versus high-end cluster) and their features on the performance and data generation for a small neuronal network model; and
- assess the impact of model complexity (i.e., number of simulated neurons and level of neuronal connectivity) on the performance and data generation for increasingly larger neuronal network models.

In reference to the integration of *in situ* analysis into a simulated model of the neocortex. In this thesis we make two contributions, we:

- describe the integration of our *in situ* analysis method for spectral analysis on top of Dataspaces [21], a high performance middle-ware for general *in situ* analysis; and
- demonstrate the accuracy of our method by comparing the results of the analysis performed *in situ* to the analysis performed after the simulation (post-simulation) using both qualitative and quantitative methods.

Lastly, in this thesis we make three contributions for our computational steering environment, we:

- describe a distributed computational steering environment for simulated neuronal networks on GENESIS
- integrate a working computational steering prototype into a simulated neuronal network model; and
- evaluate the performance our computational steering environment for simulated neuronal networks on GENESIS.

1.4 Organization

The remainder of the thesis is organized as follows: Chapter 2 describes the necessity of brain simulation in neuroscience, introduces our simulation platform GENESIS, reports on the current state of the art in *in situ* analysis frameworks and tools, and closes with a description of computational steering. Chapter 3 presents our experimental study of the impact of platform and model complexity to the performance and data generation of a simulated model of neocortex. Chapter 4 presents our work in integrating *in situ* analysis into a numerical model of the neocortex simulated on high performance computing resources. Chapter 5 describes the design, and implementation of our CSE prototype, along with a performance evaluation of our working prototype of a computational steering environment. Additionally, we provide two successful case studies of steering of our model. In Chapter 6 we summarize and discuss the contributions of this thesis along with a description of future work.

Chapter 2

BACKGROUND

In this chapter, we describe the necessity of brain simulation in neuroscience. We explain the basics of brain modeling and simulation. We start off with a description of how biological neuronal networks are represented as computational models. We continue with an introduction of our simulation platform GENESIS, which we use to execute the model on high-performance computing resources. We follow with a characterization of the limitations of the traditional workflow. We then report on the current state of the art in real-time analysis frameworks and tools. Finally, we conclude with a description of computational steering and its general challenges.

The rest of this chapter is organized as follows. Section 2.1 explains the need for computational neuronal networks and how the brain's and other neuronal networks are represented as simulated computational models. Section 2.2 introduces the principal neurobiological simulation framework that we use in this thesis, GENESIS. Section 2.3 describes the limitations of the traditional approach of simulation and analysis. Section 2.4 describes frameworks and methods supporting real-time analysis of large-scale simulations. Section 2.5 introduces the concept of computational steering to drive "what-if" scenario experimentation. Section 2.6 provides a discussion of this chapter.

2.1 Neuronal Network Modeling

2.1.1 State of Practice

Cognitive functions are the results of neuronal connectivity and the associated neurons' electrical activity. It is theorized that cortical processing is accomplished by the synchronicity of the spike trains of large assemblages of neurons [22, 23]. Brain-wide electrical activity can be observed using electroencephlography (EEG), where

electrodes are placed non-invasively on the outside of the head/skull to record billions of neurons simultaneously. EEG has been used successfully to study neuropathology [24]. EEG records fluctuations that result from ionic current with the neurons of the brain [25, 26]. An EEG contains a great deal of information about the functions of the brain. This leads to a multitude of approaches for performing analysis on the EEG using various spectral analysis methods.

EEG recordings consist of the spontaneous electrical activity of the brain over a period of time [25]. Brain-wide electrical activity is often oscillatory with frequency ranges from 1 to 100 Hz and has been shown to be associated with human cognition [27]. EEG signals are non-deterministic and are not restricted to special formations such as the electrocardiogram sign (ECG). Due to these non-restrictions the parametric and statistical analysis approaches are used (e.g., time-frequency analysis) [28]. The ability to directly observe how fine-grained neuronal connectivity impacts the activity of the brain is not feasible given the current state of *in vivo* human neuroimaging [3]. To address this limitation large-scale models of the brain with high biophysical detail have been developed. Accurate brain models are inherently computationally intensive both in numbers of neurons (the cerebral cortex contains 14 to 20 billion neurons [29]) and regarding connectivity (the cerebral cortex contains trillions of synapses).

2.1.2 The Neocortex

The neocortex is a thin layered convoluted structure that surrounds the brain of mammals with a surface area of approximately $2600cm^2$ and with a thickness of 3-4 mm [30]. Neurons in the neocortex are connected to one another by synapses. The Neocortex consists of folds and ridges and comprises 76% of the volume in the human brain [31]. The neocortex is the newest evolutionary development in the brain, and because of this, gets its name "neo". The neocortex is composed of different layers each with specialized functionality. Additionally, on each layer, neurons are organized into vertical structures called microcolumns. Neurons contained in the same microcolumn share the same static and physiological dynamic properties [30].

It has been long believed that connectivity within the brain is responsible for the cognition and behavior of mammals such as perception, decision-making, and language [32, 33, 34]. The neocortex is comprised of 80% excitatory and 20% inhibitory neurons, which are named after the effect that they have on other neurons when they spike [31]. Excitatory neurons spikes or action potentials cause other neurons to be more likely to fire, where in contrast inhibitory neurons cause neighboring connected neurons to be less likely to fire.

2.1.3 Representation of Simulated Neuronal Networks

Neuronal networks can be described as distance weighted, directed graphs where graph nodes are neurons and the edges of the graph are the synaptic connections [35]. Neurons in the network interact with one another by communicating through the use of point events called action potentials or spikes [35]. Spikes are propagated unidirectionally with a specified delay from the neuron initiating the spike to the neuron receiving the spike. If enough synaptic input is received from connected neurons, the receiving neuron's threshold will be exceeded, and the neuron will be caused to fire continuing the propagation of the spike across the network [36].

At present how neuronal connectivity produces complex behaviors such as speaking, understanding emotion, planning, and performing complicated motor tasks is poorly understood. Understanding the brain not only helps us understand its biological functions but also contributes to the design of new emerging computing architectures such as neuromorphic computing [37]. Numerous initiatives have focused on gaining a deeper understanding of how the brain functions. The Blue Brain Project (BPP) [38] and the Brain Research through Advancing Innovative Neurotechnologies (BRAIN) [39], have both achieved success in the simulation of important brain regions. Both projects leverage cheap commodity high-performance hardware to simulate thousands of neurons simultaneously. Currently, it is understood that there is a strong relationship between neuronal connectivity and functionally relevant brain activity [40].

Directly observing neuronal activity of individual neurons is not possible given the current state of in-vivo human neuroimaging [3]. To achieve brain-scale simulations (i.e., 10^9 neurons and 10^{12} synaptic connections), larger neuronal networks models are needed that are not currently available. These future networks, currently exceed the capacity of small-to-medium sized clusters that are traditionally available to most scientists in their own research facilities. To meet these growing computational demands researchers have relied on massively parallel computing architectures installed at high performance computing research centers which have fortunately increased in availability.

To study the fundamental principles of information processing in the brain, simulated neuronal network models are used consisting of more than 10^5 neurons and 10^9 synaptic connections [10]. Simulation models of neuronal networks allow scientists to explore the impact of differential neuronal connectivity, using analysis and simulation environments. There has been increased development in the performance and capability of neuronal network simulations in the past decade [11, 12, 13, 41, 42]. The focus of these improvements have mainly targeted models that are less than the size of a cubic millimeter of cortex (approximately 10^5 neurons) executed on which has become the current dominant paradigm of high-performance computing (i.e., small to medium-sized clusters). A major challenge to the simulation of neuronal networks is the large number of synaptic inputs to a single neuron [6]. The number of synaptic connections in these networks has been shown to exceed the current memory capacity of hardware available to researchers [5]. Even if there were sufficient memory resources the large amount of time need to construct/instantiate and simulate the network makes behavioral optimizations to the model through parameter searches difficult. Balancing the need for model fidelity and performance becomes increasingly important as model complexity grows. To achieve neuron counts in the billions, we need a clearer understanding of how model features impact computational performance and data generation.

2.2 Simulation Frameworks and GENESIS

2.2.1 State of Practice

Simulation of neurobiological processes has established itself as a significant part of the scientific analysis method in neuroscience. It is used to study the how anatomical data relates to the physiological data available. Also, it is used to study non-static systems that are difficult to study with the current state of the art of analytical methods. Simulation frameworks and models used to study this behavior came into prominence in the late 90's and early 2000's. Most analyses on brain networks were done on datasets that described the large connection patterns of the rat [43], cat [44], and monkey [45]. However, these were all partial network models. Scientist started to see that only collecting and studying experimental data was not enough, to be able to answer the fundamental questions of how neuronal circuits work. From this conclusion, experimental neuroscientist saw the need to go from an observational to a more quantitative approach for their study of neuronal networks. This has led to scientist migrating to computational models to study network dynamics and is what we now call computational neuroscience [46]. One of the first full simulated network models was of the Caenorhabditis elegans (C. elegans) [47]. The C. elegans is a simple multi-cellular organism consisting of approximately 1,000 cells and 302 neurons. "Due to the simplicity of the organism C. Elegans were ideal for studying information processing in neural networks. The next jump in network size came from the successful simulation of a full cortical column of rat brain [43]. The most recent notable advancement in the simulation neural networks has come from the Blue Brain Project and their simulation of the entire rat brain [2].

2.2.2 General Neural Simulation System

GENESIS [48, 49, 50] is an object-oriented multi-function simulation package that allows scientists to build scientifically rigorous neurobiological models of the brains functions. Models in GENESIS simulate brain activity from the level of small sub-cellular processes to sophisticated large neuronal networks. Individual neurons are

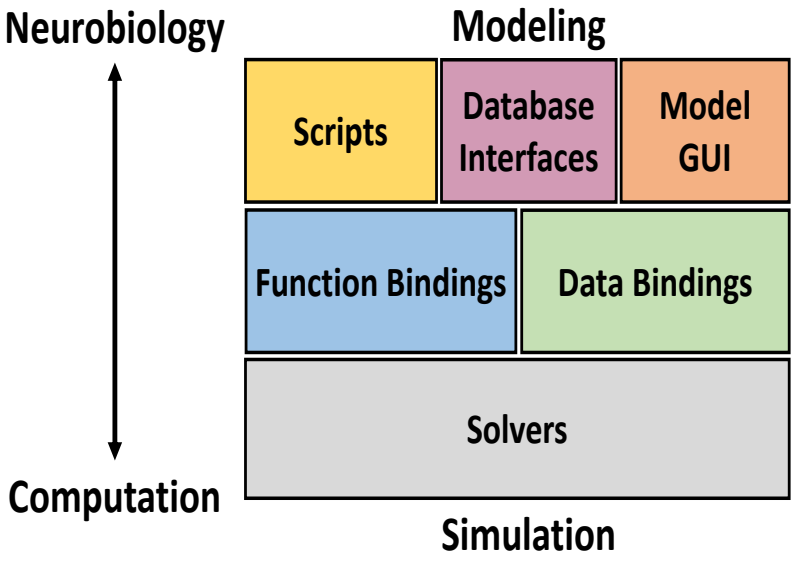


Figure 2.1: GENESIS' module components.

modeled using multiple compartments, generating realistic electrical activity in the 1-100 Hz frequency range. GENESIS was originally developed in the 1980's by Dr. James M. Bower's in his laboratory at Caltech. GENESIS is designed to be easily extensible and adaptable to run on large-scale clusters [51].

PGENESIS is the parallel extension of GENESIS and runs on a wide range of hardware from single or networked workstations to supercomputers using the Message Passing Interface (MPI) and the Parallel Virtual Machine (PVM) [52]. GENESIS simulations are built from "objects" that are fed inputs, perform some type of mathematical operation on them, and then based on the result of the operation, generates outputs which are input to other "objects". Neurons in GENESIS models are built from these basic "objects" in a compartmental fashion [53]. These compartments are linked to their ion channels and these channels are linked together to form multi-compartmental

neurons which can form any required complexity needed.

GENESIS employs a high-level simulation language to configure and construct neurons and their networks. GENESIS can be used both inter- and non-interactively [53]. GENESIS is written in C, using the X Window System, and the Portable Operating System Interface (POSIX) for its I/O needs. Figure 2.1 shows an overview of the modular structure of the GENESIS simulation from computation (i.e., solvers that govern the gate channels of the simulated neurons) to the neurobiology (i.e., scripting language and module graphical user interface used to configure and construct the neuronal networks). This overview has been recreated from the Book of GENESIS [48].

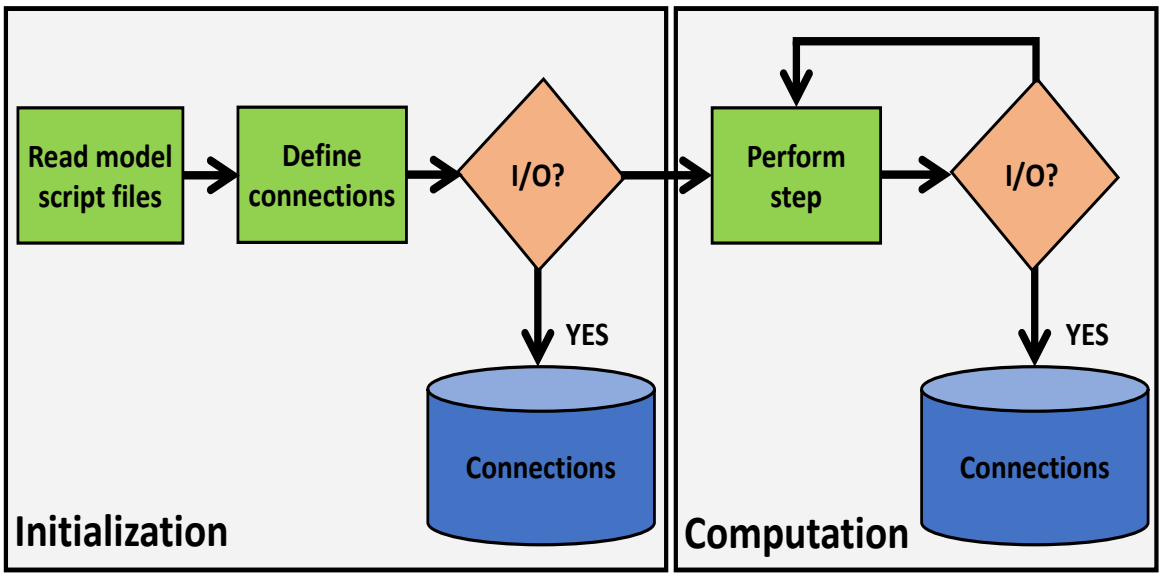


Figure 2.2: Workflow of GENESIS broken up into its initialization phase and its iterative computation phase.

The workflow of a GENESIS simulation is shown in Figure 2.2 and can be broken into two main phases: model initialization and model computation time. During the initialization phase, GENESIS launches the model's input scripts. The scripts hold the initial status and values of the model. After parameters values have been initialized

the model proceeds to establish connections between neurons according to probabilities outlined in the initialization scripts. Once all connections have been completed, the entire neural network is written to disk, if the I/O setting in GENESES is set to on. This can be very time consuming, as we show later in this chapter. Following writing the network out to disk, GENESIS enters the simulation phase. GENESIS runs for a predefined maximum amount of steps; solving underlying model differential equations and updating simulation variables at every time step. If I/O is on, GENESIS writes out performance and brain-behavioral data to disk at each time step.

2.2.3 Simulation Frameworks Beyond GENESIS

Several simulators exist for large-scale neural modeling with parallel execution. Parallel simulation of spike-coupled neural networks is supported in NEURON [54], GENESIS [48], NEST [35], NCS [55]. and SPLIT [55], the research and development in parallel simulation have been modest. Furthermore, I/O effects were not analyzed previously for parallel simulations. There has been sparse research in regards to the study of the impact of neural network parameterizations on performance and data generation. Specifically scientists who have done work in this space have focused on profiling communication but have not considered input/output. Hines et al. look at how interprocessor spike communication affected total simulation time [56]. They focus on the spike exchange methods used such as MPI_All_Gather, and several methods for non-blocking Mult-Send. Shehzad et al. addressed inter-processor spike communication. They improve MPI_All_Gather to remote memory access for NEURON. Our work analyzes the impact of high-performance computing resources on compute performance, data generation, and science delivered by the General Neural Simulation System (GENESIS) for increasingly complex models of a regions of the brain's neocortex.

2.3 Traditional Analysis Approach

2.3.1 State of Practice

The onset of extreme-scale computing brings with it the promise of allowing neuroscientists to examine the brain *in silico* at unprecedented fidelity and resolutions not possible using traditional methods such as the electroencephalogram. The large amount of data produced by high fidelity simulations is becoming increasingly difficult to save and transform into scientific insights for runtime simulations. The ability to process, analyze, and make substantive inferences on the simulation output is growing in difficulty. Traditional workflows in high-performance computing involve the preparation of an application's input, execution of the simulation, and concludes with the analysis and visualization of the simulation's results as a post-processing step, usually done in a sequential fashion. Growth in compute power, memory, and storage in high-performance computing has enabled increasingly computationally demanding and complex simulations. The input/output subsystems are struggling to meet the application data management demands that are being exacerbated by the increasing compute capability of current high-performance hardware. This trend is predicted to continue into the foreseeable future.

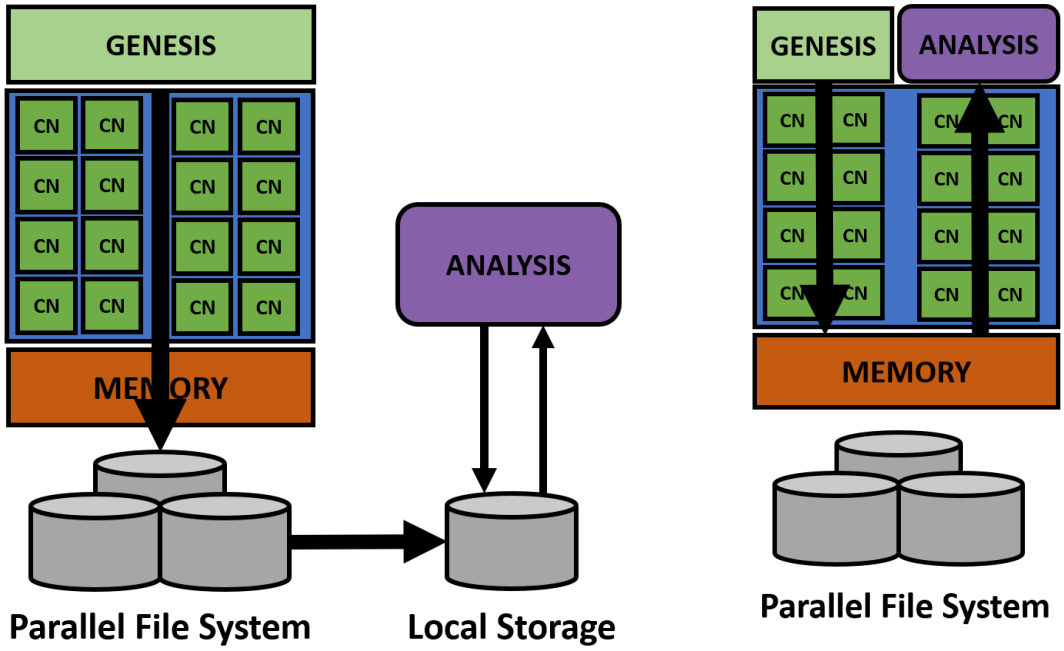
The complexity of numerical codes and explosion of input parameters make finding optimal configurations hard. Additionally, the data deluge brought forth by highly anatomically detailed models are placing increased pressure on input/output subsystems of current and next generation high-performance computing systems. Simulated spike-coupled neuronal networks have been of high interest to the scientific community for quite some time. Simulated neural networks produce massive amounts of data, have a high degree of dimensionality, and are highly dynamic. Currently there are very few simulation frameworks that allow for the analysis of simulation data *in situ*, with there being none to our knowledge with the ability to be steered and directed interactively at runtime.

2.3.2 Limitations of Traditional Approach

Traditionally, analysis of simulated local field potentials are completed in a centralized manner. This is performed once the simulation of the brain's electrical activity is completed (post-simulation analysis), where voltage potentials along with spike data are moved to a centralized node where it is processed post-simulation. This approach leads to three inherent drawbacks. First, analysis of data post-simulation assumes that the chosen recurrence of data output is high enough to capture all relevant model behavior. Second, knowledge is needed a priori regarding when desired electrical activity occurs and the simulation can be terminated. Third, post-simulation analysis eliminates the ability to use knowledge gleaned from data generated at runtime to drive computational steering. To address these challenges, scientist are attempting to bypass the I/O bottleneck through the use of *in situ*, where the analysis occurs at the locale of data generation, thus eliminating the need to move raw simulation data from persistent storage. The schematics of the two analysis workflows that we focus on (i.e., the current post-simulation analysis and our new *in situ* analysis) are shown in Figure 2.3.

2.4 Frameworks and Methods for *In situ* analysis

A solution to the above limitations is to modify the data analysis workflow from one that is accomplished post-simulation to one that is completed *in situ* at the cost of confining the analysis to the level of local data rather than a single global view. The missing global view of data has the potential to introduce inaccuracies in analysis results. Due to its local nature both spatially and locationally. *In situ* analysis results are not necessarily identical to results obtained post-simulation. Successful *in situ* analysis must identify a suitable window of data that is large enough to assure the analysis scientific meaningfulness and, at the same time, small enough to fit in memory, eliminating the need to swap data to storage. *In situ* analysis needs to meet the following requirements: execute relatively fast, avoid moving simulation output,



(a) Post-simulation Analysis (b) In-situ Analysis

Figure 2.3: Schematics of analysis workflows, post-simulation and *in situ* analysis. In post-simulation analysis, output is stored on persistent storage (e.g., parallel file system) then moved to a centralized node for post-processing. With an *in situ* approach analysis and simulation occur simultaneously on the same primary resources, eliminating the need for data movement.

limit memory usage, and be accurate within some tolerance to results obtained from a full global view of the data.

2.4.1 State of Practice

Increases in compute capability have far out-paced I/O bandwidth capacity [57]. Over the last decade, the compute performance of leadership class machines have increased over several folds from teraflops to petaflops. Leadership class machines' I/O subsystems have failed to keep up with the increases in compute. This inequality combined with the high level of data being generated by high fidelity simulations has lead to a major bottleneck in the performance of leadership-class systems. There have been efforts in addressing this imbalance. One of the primary solutions has been to increase I/O performance by adding faster disk such as solid state disk drives set up as

burst buffers [58]. However, adding faster disks will inevitably fail to keep up with the increases in compute capability due to the associated cost. In addition to the cost, the I/O challenge is worsened by the fact that large-scale applications fail to attain a significant fraction of the peak input/output performance of high-performance computing systems [59].

The necessity for novel *in situ* analysis techniques at the current and next-generation systems has been documented in a number of workshop reports from organizations such as the Department of Energy as well as the National Science Foundation [14, 20]. The obstacle this poses to traditional data-parallel post-simulation analysis such as visualization and analysis workflows have been widely examined [60, 61]. This call to action has prompted a surge in the developments of *in situ* and co-analysis data processing approaches [62]. A major need in the analysis of the large amounts of data produced by scientific applications is that it can be indexed, organized, and annotated [63].

The use of data-staging, in which dedicated nodes are used as a buffer where data is moved from compute to closely connected staging nodes before migration to persistent storage, has been shown to assuage the I/O challenges of high-performance machines. Data-staging nodes allow advantages over the traditional dedicated I/O nodes of large-scale systems. Staging nodes allows the development of user-defined data processing scripts to leverage their compute power. Projects such as DataStager [64], and AcitiveSpaces [65], have studied the benefits of using data-staging to add value to the traditional I/O pipeline from compute to storage. In each work, they demonstrate how data-staging can be used with large-scale production level codes and their associated tradeoffs. The drawback of data-staging is that it only supports limited operations on the staged data such as transformations, curations, and pre-processing. This leads to an underutilization of the dedicated staging nodes.

Many recent efforts have looked at using staging nodes to facilitate *in situ* analysis [66, 67]. Work in [62] explores integrating a combination of *in situ* and co-analysis of S3D a massively parallel turbulent combustion code. They show how three common

analysis algorithms can be transformed into massively parallel *in situ* and in-transit workflows on high-performance computing. This work has shown how data staging and *in situ* is successful at enabling analysis at execution time for a specific set of scientific analyses; it does not include a comparison of post-simulation versus *in situ* results.

2.5 Computational Steering and Scientific Workflows

2.5.1 State of Practice

Computational steering has been identified as an important challenge since the late 1990's. A report published by the United States National Science foundation on scientific visualization, determined in 1987 that computational steering will be an important tool for scientific discovery [68]. This early work targeted efforts in developing computational visualization techniques that have the potential to increase scientific productivity. Brooks argued in [69] that there is a "need to design generalized interfaces for visualizing, exploring, and steering scientific computations". Brooks' work was some of the first in asking whether computer graphic technologies could be leveraged for scientific discovery. Additionally, early distinctions were made between the interactive visualization stage of post-simulation analysis, it was argued by [70, 71] that computational steering allows for both users of the application and the analysis algorithms themselves to be "in the loop" of the actions that are taken by the application in response to state changes.

More relatively recent work in [72] supported computational steering of large-scale computational fluid dynamics (CFD) codes on large-scale clusters. In this work, they focused on MPI-based CFD codes, optimizing them to be used with a computational steering environment. Ruess et al. in [73] described a computational steering environment that enabled a "Human-in-the-loop" dynamic that allowed for a user's intuition and interpretation to be part of the analysis pipeline. Their work focused on finite element models (FEM). There are a number of computational steering environments available. The Visualization and Applications Steering Environment (VASE) [74]

is one of the earliest developed by the University of Illinois. VASE is designed for optimizing algorithms and model exploration, VASE allows for users to perform parameter explorations easily in addition to enabling portions of the application's code to be swapped out or replaced. SCIRun [75] developed in C++ out of the University of Utah, provides algorithm and parameter exploration. SCIRun does not provide support for multiple users to steer the application. SCIRun's intended purpose is to act as a workbench for users to be able to debug and synchronously or asynchronously steer large-scale applications.

The Program and Resource Steering System (Progress) [76] and its successor Magellan [77] are developed out of Georgia Institute of Technology. Like most computational steering environments listed above Progress and Magellan specialize in behavioral optimization and model exploration. Legacy applications can have interactivity added with a simple annotation of their source code. This annotation can be very focused, only revealing the most important steering parameters that affect model behavior. Lastly, CUMULVS [78] developed out of Oak Ridge National Laboratory is broken up into two parts one for the application itself and the other for the visualization and user-facing steering front end. CUMULVS allows for the dynamic coupling of multiple independent visualization and steering front ends to be attached to an application as it is running. The work in this thesis differentiates itself from the above mentioned computational steering platforms in two ways. First, simulation data that is used to make interpretations of the simulation's behavior resides solely in memory. By operating on data solely in memory, users can perform fast analysis on simulation output. Second, it enables users to leverage powerful Python packages such has SciPy and Numpy, in addition to popular R packages.

2.5.2 Scientific Workflows

Simulation through the use of large-scale computing systems is becoming a larger and larger part of how today's scientific advances are being achieved in both neuroscience and the greater physical science community. These scientific advances are

made possible by complex computations on data that consist of several steps, including the use of coupling different models, data sources, that can span across a distributed computing resources [79]. An example of such a scientific workflow is a user retrieving data from a distributed data source or model simulation, reformatting and transforming the data, indexing and labeling the data, and then running several analyses on the data. Each workflow is highly dependent on the application and the desired output from the analysis. Workflow automation has been shown to increase productivity when performing simulation-based experiment across many scientific disciplines [80]. This automation frees a user from being required to manage the analysis pipeline from data generation to scientific dissemination. Additionally, this automation allows for the workflow to be adaptable to the changing requirements of a simulation's resource needs (e.g., compute, memory) [81]. There is little activity in the computational neuroscience community for automated scientific workflows, as increasingly realistic simulations resource requirements grow the case for new analysis workflows becomes stronger.

2.5.3 Interactive Data Visualization

Today, the majority of large-scale applications are executed in a batch processing mode. Batch processing is when a group of tasks are organized together in a collection called a job and several jobs are executed without any kind of manual intervention from the user. The user controls the execution of the job through the use of submission scripts where the user defines the required resources for the application as well as how the output data should be managed once the simulation has been terminated (e.g., synching the data to archival resources for long-term storage). Since there is not a method for interacting with the simulation while it is running the user must wait for the simulation to terminate before being able to observe whether the simulation was fruitful by analyzing it's post-simulation output.

Traditionally, analysis of simulation output has been performed through the utilization of interactive data visualization. Interactive data visualization is the interactive navigation and visualization of large amounts of data produced by high-performance

computing applications. Interactive data visualization is the task that everyone envisions when one thinks about the traditionally scientific analysis workflow. In this workflow a simulation is first configured with any input data and is configured using input scripts which determine the size of the simulation, the size of the problem domain, and the amount of time that the simulation runs. Once the simulation has been executed to termination, the raw output data is moved to dedicated visualization resource. Here, static datasets produced by the simulation are visualized in an way in which the user has full control over what data is presented at the time of analysis. The user is able to navigate freely through the data, transforming it ways that are needed for the type of analysis that needs to be performed. Most large interactive data visualization methods use offline post-processing steps to reduce data size.

2.5.4 Computational Steering

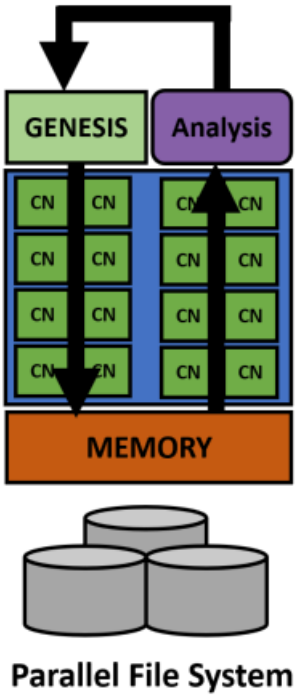


Figure 2.4: Schematics of computational steering workflow, the simulation is interactively controlled by the user. Inferences from analysis and visualization are directed back into the simulation enabling "what-if" scenarios.

Computational steering is the on-line monitoring of intermediate simulation results and interactive modification of model parameters of long-running high-performance computing jobs for model exploration or performance improvement. Computational steering has many applications. First, it makes possible the adjustment of algorithm parameters for which performance randomly varies, is highly dependent on input data, or for which the behavior at execution time is not well understood. Another application is in modeling and simulation codes having the ability to monitor and modify model parameters at runtime enabling scientists to terminate low-yielding or unproductive simulations pre-maturely and to perform efficient parameter sweeps across parameter configurations.

One of the primary uses of computational steering is the interactive visualization of simulation data. However, this can be confused with interactive data visualization in which the user manipulates and produces visualizations of data post-simulation. The primary distinguishing feature of computational steering and interactive data visualization is that the former allows the user to make inferences and knowledge gained from the data to drive the simulation's behavior towards more desirable problem spaces. Figure 2.5 depicts how information is used to steer an application (i.e., GENESIS) on large-scale clusters. Note that data is not written to persistent storage but is instead contained only in memory. The user can interact with the memory-resident data and use knowledge gained at simulation time to steer GENESIS.

Figure 2.5 shows the two points in the parameter space (P_1, P_2) and $(P'1, P'2)$ and their solutions and corresponding visualizations respectively. If time is one of the parameters in the parameter space then changes to one of the parameter will affect the direction that the simulation will take. Steering the application through time facilitates hypothesis-driven experimentation through "what if" interactions and fast debugging. This means that the user does not have to go back to the start of the simulation to make changes to input parameters, enabling the user to decide at runtime to modify a parameter at a particular moment to study tangential behavior.

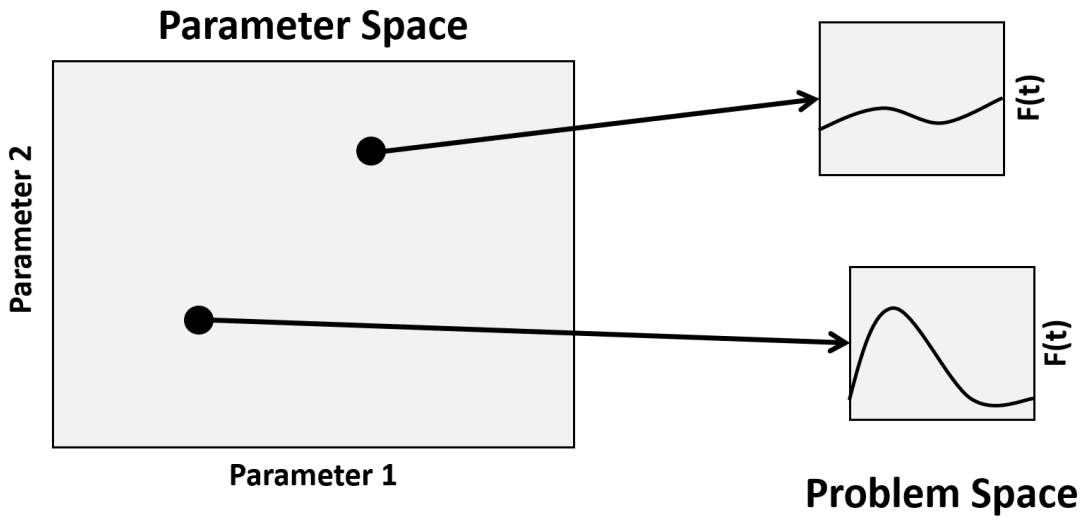


Figure 2.5: Parameter space resulting from two steering variables, with two of the solutions corresponding from two points in the parameter space with their visual representations.

2.5.5 Steering Challenges

There are several challenges when integrating a computational steering environment for numerical simulations such as the simulated neuronal network which is the focus of this thesis. One major challenge in the implementation of a steering environment is providing a way for steering actions to be distributed to the simulations that are being monitored. A computational steering environment must address this challenge by making an effort to maintain the integrity of the computations of a simulation. Safeguarding integrity is done through a devoted effort to ensure minimal latency and computational perturbation and consistency [82]. We define latency as the difference in time between the event taking place and when the event is observed or acted upon by the user. Understanding the degree of latency in our CSE will provide a quantifiable measure of our CSE's ability to keep pace with model events and provide a method for users to conduct science in an intuitive and familiar to which is traditionally accomplished in a wet lab.

Monitoring an application is essential in the study of dynamic models that are

distributed across diverse resources. Domain scientists who leverage numerical models of physical processes rely on simulation monitoring to perform behavior optimization on the model. In addition to enabling the study of application behavior, monitoring is also used for performance optimization. Understanding how different algorithms and data structures perform under different conditions is a very common scenario for performance optimization. However, to monitor an application, the application must be instrumented with additional source code. Injecting additional source code into an application can have unforeseen consequences on simulation performance. We label this negative effect on simulation performance, instrumentation overhead. We define instrumentation overhead as the increase in execution time of an application due to instrumentation. Simulation perturbation can be defined the degree to which a numerical computation is slowed or negatively affected (i.e., overhead) due to the presence of steering instrumentation.

Finally, consistency presents a major challenge in the implementation of a computational steering environment. When presenting information to the user its of the utmost importance to present the user with up-to-date information. In a distributed setting, collecting data from multiple sources might result in inconsistent views that give an erroneous picture of the model's behavior [83]. These inconsistencies are a consequence of receiving data from multiple sources and integrating these data sources into a cohesive data stream to the user. Due to the distributed nature of HPC applications, data might come in at different times resulting in a stream of events that are not representative of the true model behavior. This stream, when presented to the user could possibly be misleading and result in incorrect interpretations of model behavior.

2.6 Discussion

To investigate the origin and functional role of the brains electrical activity in specific frequency ranges, neuroscientists are leveraging biologically accurate simulated models of the brain. Models of neuronal networks allow scientists to explore the impact of differential neuronal connectivity using analysis techniques and information

not available experimentally. These models are inherently computationally intensive requiring more resources from both memory and I/O subsystems than what is currently available to most small research teams. Recently research in the simulation of neuronal network models has undergone a multitude of advances. Computational neuronal network models of are approaching sizes on the order of 10^5 neurons and 10^9 synaptic connections. However, this size is nowhere near what is needed to fully simulate the human brain which is theorized to be comprised of 10^10 neurons and 10^12 synaptic connections which are 1,000 times more than the number of stars in the Milky Way galaxy.

Traditional approaches for analysis of data-intensive applications are reaching their limit in providing practical methods for scientific discovery. It's no longer effective to rely on I/O subsystem to provide suitable bandwidth for off-site data analysis of simulation output. We have to move away from approaches that require the data to be moved to the analysis, but leverage approaches that allow for the analysis at the site of the data. To meet the demands of these data-intensive simulations, we need to design, implement, and utilize novel scientific analysis workflows that utilize modern analysis techniques such as *in situ*, *in transit*, and computational steering.

The rise in the complexity of simulated models have increased the need for analysis tools that offer a higher level of interactivity. Computational steering has been an active research area since the early 1980's. One of the earliest applications was to computational fluid dynamic codes simulated on high-performance computing resources. However, computational steering has yet to be integrated in the computational neuronscience scientific workflow. The computational neuroscience community is still relegated to traditional analysis approaches that fail to scale to meet the demands of increasingly realistic brain models. By transitioning to richer interactive data analysis pipelines via computational steering, neuroscientist will be able to go from one experiment per execution of a simulated model to multiple experiments per execution. Increasing analysis throughput and reducing the end-to-end scientific discovery pipeline.

Chapter 3

NEURONAL NETWORK MODELS ON HPC RESOURCES

In this chapter, we study the performance of neocortex simulations using GENESIS. Specifically, we study the impact of platforms (i.e., single fat nodes versus high-end clusters) and their features on the performance and data generation for a small scale model of neocortex. We assess the impact of the neocortex model's complexity (i.e., number of neurons and neuronal connectivity) on the performance and data generation for increasingly large versions of a simulated neocortex model on high-end clusters. Additionally, we use this chapter to highlight the relationship between initialization cost of simulating our neuronal networks and model complexity.

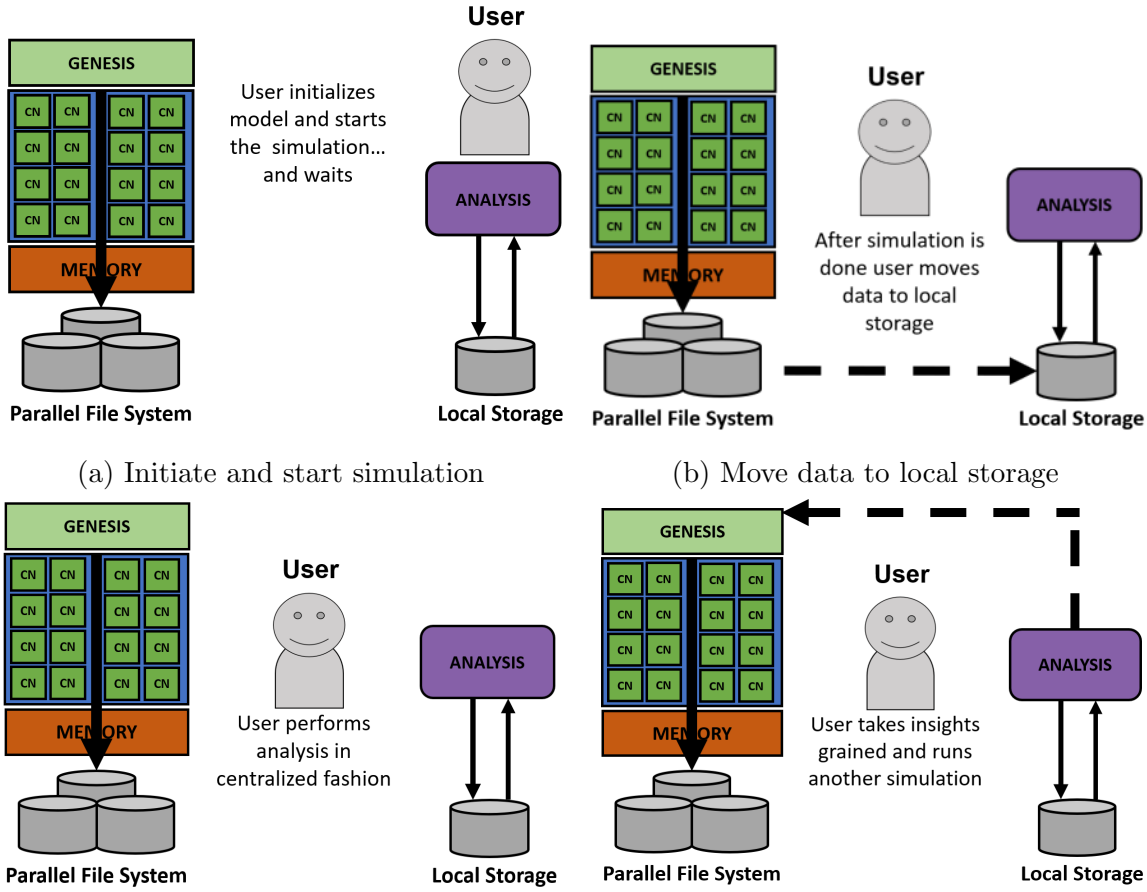
The rest of this chapter is organized as follows. Section 3.1 we describe the neocortex model and associated output files used in this thesis. Section 3.2 presents tests we use to evaluate the impact of simulation platform and their features along with model complexity on performance and data generation. Section 3.3 presents the results of our tests that show that realistic modeling of neocortex functions are feasible but requires high-performance computing to mitigate the growing computing and data demands of the associated simulations. Section 3.4 we conclude and with a discussion of lessons learned and a description of future work.

3.1 Neocortex Model and Associated Output Files

The brain model we target in this thesis simulates the neocortex which comprises the outer gray matter of the brain [84]. The neocortex is organized into six layers and is home to roughly 14 - 20 billion neurons. In mammals the neocortex underlies close to all sensory and cognitive processing [84]. The model consists of 23 different neuronal cell types, each with its own set of parameters and dynamics. Our baseline model

represents a 150 x 150 x 2871 μm patch of cortex. The simulated cortical patch is broken into multiple regions. Each region contains a N by N matrix of micro-columns. Micro-columns are vertical columns which extend through the six layers of the brain and are believed to be the smallest functional unit of the neocortex. Each micro-column contains an identical set of neurons. Micro-columns are typically arranged in a square and separated by $25e-5$ μm along either the x or y axes. There are two kinds of connections made between neurons within the model, short and long range connections. Short-range connections are intra-regional connections. Long range connections are connections made across regions to other neurons in neighboring regions. Memory consumption in our neuronal network model are dependent on two aspects: the first being the size of the brain model (e.g., the size of the simulated patch in terms of neurons) and the second being the number of processes used for the simulation.

Figure 3.1 shows the typical workflow of an execution of the model which is done in a sequential fashion. The user starts by initializing the model. Examples of model initialization is setting the total amount of biological time and number of neurons to be simulated. After the model is fully configured the user then starts the simulation in batch-processing mode. During this time the user has zero ability to interact with the model, waiting until the model has reach the pre-defined biological time. Upon the completion of the simulation the user transports all raw simulation output from a large parallel file system (PFS) to a smaller cluster with access to its local storage. This can be costly due to the fact that the local storage does not have the capabilities of a PFS. After all the raw simulation data has been moved, the user performs post-hoc analysis on the output gaining insights into model behavior that occurred during the simulation. Lastly, after the analysis is complete the user takes insights gained during the analysis phase and moves that knowledge back into the next parameterization and execution of the model. The user can now a priori to execution, determine where he would like the simulation to go. To perform I/O, the model relies on the Portable Operating System Interface for Unix (POSIX) calls (e.g., fwrite) to create, read from,



(a) Initiate and start simulation

(b) Move data to local storage

(c) Perform analysis on local storage

(d) Apply knowledge back into simulation

Figure 3.1: Sequential workflow for a single execution of GENESIS. (a) The user parameterizes the model and starts execution in non-interactive match mode (b) After simulation is complete the user moves all raw simulation from a PFS to dedicated local storage, (c) user performs analysis on model, (d) user takes lessons learned and apply back to model.

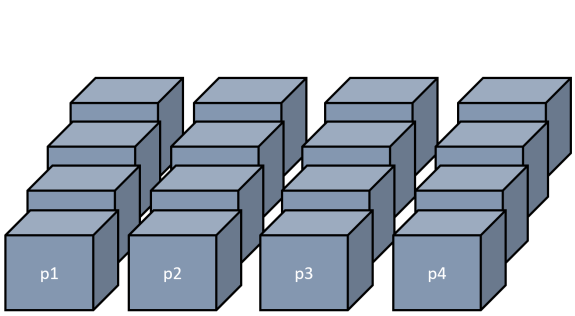
and write files to disk. POSIX provides an application programming interface standard in addition to some interfaces to shells and utilities. If the I/O is turned on, a simulation of the neocortex subsystem creates seven main file types. During the initialization, GENESIS writes a set of files which contain biologically relevant information useful to interpret model results. These files are: a list of neurons and times receiving Poisson distributed input spikes (randomspikehist files), a file describing the connectivity (randomspike connections), and a file describing the long-range connections, and short-range connections. At the end of each iterative step, GENESIS writes spikehist files, local field potential (LFP) files, and membrane files. Spikehist files contain all the normal spikes fired by neurons within a column. Each line describes the spiking cell and the time of the spike. This file records an important aspect of information transmission in the model – a line in the file means that MPI messages were sent from the cell to its connected children at the stated time. Randomspikehist files are similar to spikehist, however the difference is that this file records spikes not from normal neurons but from Poisson processes (with a mean fire rate). These represent input to the model from background activity in other non-modeled parts of the brain. Short-range and long-range connection files record incoming and outgoing messages from the cell. Randomspike connections record messages from Poisson processes. LFP files are the summed extracellular electrical activity of neurons recored at fixed electrode positions. Membrane files record the membrane potential (voltage) for all neurons of a given type in the column.

3.2 Evaluation Methodology

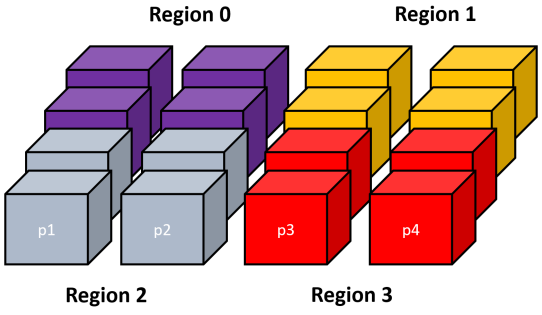
3.2.1 Hardware Platform Impact on Model Performance

To understand the impact of platform selection on performance and data generation, we use a simple model of the neocortex as our baseline model and we call it M(1, 2x2). The model consists of 16 GENESIS processes (nodes) as shown in Figure 3.2a. The 16 processes are partitioned into four regions of the model as shown in Figure 3.2b. Each process simulates 2 by 2 micro-columns, separated by $25e-6$ on

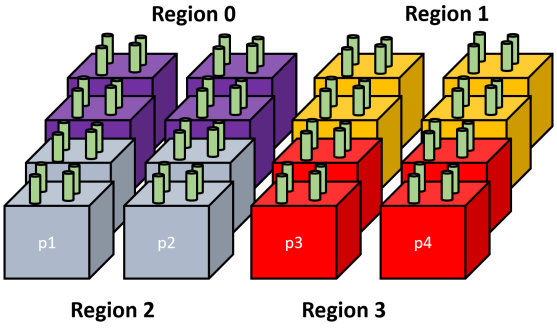
both the x and y axis as shown in Figure 3.2c. A unit of time within the simulation is 50 μs in length. In our model each micro-column holds roughly 23 neurons. Thus each GENESIS process in the baseline model is simulating approximately 92 neurons. Each cell has a probability of establishing a long-range and short-range connection with another cell as shown in Figure 3.2d. Short-range connections consists of those connections that can occur within a node or within a region. Long-range connections are those connections that occur inter regionally.



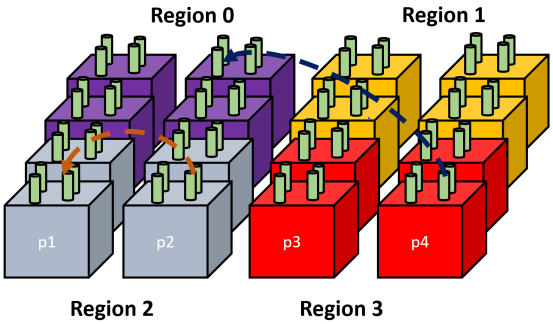
(a) 16 processes simulating 23 neurons



(b) Processes broken into four regions



(c) Each process simulates four micro-columns



(d) Inter-region long range and intra-region short range connections

Figure 3.2: Baseline model simulated among 16 GENESIS processes (a) with its 16 processes partitioned into four regions of the model (b), each process dealing with 2 x 2 micro columns (c) and with a connectivity level to control the amount of short-range and long-range connections (d).

We focus our study on three platforms which exemplify the landscape of high performance hardware available: Unitank, Spirit, and Excalibur. Our first machine, Unitank, is a single fat node, consisting of a large amount of cores, disk, and memory

in a single machine. Fat nodes such as Unitank, represent commodity Linux machines that are available to single researchers. It is a two-socket Dell R710, with Intel(R) Xeon(R) CPU E7-8837 processors clocked at 2.67 GHz. Each socket has a 16-core processor. In total there is 512 GB of memory. Our second machine is Spirit, representing a traditional high-end commodity cluster available to research groups at most universities. Spirit is a small scale SGI ICE X cluster of $54,368$ cores with a peak performance of 1.5 PFLOPS, and 3,398 compute nodes with 16 cores per node. Each core is an Intel Xeon E5-2699v3 clocked at 2.3 GHz with 32 GB of memory per node. Spirit runs Suse Linux, has a FDR 14x InfiniBand, and supports Enhanced LX Hypercube for each interconnect. The third machine, Excalibur, represents a high-end cluster commissioned by an organization such as the Department of Defense or Department of Energy. It is a Cray XC40 supercomputer with a peak performance of 3.7 PFLOPS. Excalibur has $100,160$ cores and $3,130$ nodes. Each node has a total of 32 Intel Xeon E5-2698 cores clocked at 2.3 GHz; each core has 128 GB of memory.

We measure execution time to initialize the model and time spent in computation performing iterative solver steps. We consider two types of I/O settings: with I/O when GENESIS writes to disk verbosely at the end of the initialization and at each iteration step, and without I/O when GENESIS writes no data to disk for the entire simulation. With I/O, for each type of files generated, we measure the number of files, the total size of files, the files' mean size, standard deviation, and number of lines. For the version of GENESIS used in our work, each write operation generates a new line in the output files and thus the number of lines is synonymous with the number of writes. We consider two types of simulations: time-bounded simulations where we set a fix amount of total execution time for GENESIS (i.e., 1 hour or 2 hours), and iteration-bounded simulations where we set GENESIS to run for a fix number of iterative steps (i.e., 500 steps or 0.025 seconds of neocortex simulated time). We run both types of simulations with and without I/O. Each test is repeated 10 times on each one of the three platforms. On Unitank, the processes share the platform share memory. On Spirit and Excalibur, each GENESIS process runs on its own compute node and

has full access to the node's resources.

3.2.2 Neuronal Network Model Impact on Performance and I/O

The base model M(1, 2x2) is simple and thus, it provides limited insights in the neocortex activities. The model's I/O is modest and thus, it does not substantially impact performance. These conclusions are not true for a model whose complexity has been increased (i.e., with a larger number of neurons and cell connectivity). An increase in model complexity is synonymous with an incrementally detailed representation of neocortex activities and consequently, a decrease in performance along with an increase data generation (or I/O). The level of details in the model and its required computing resources (e.g., memory) restricts our work to the high-end cluster Excalibur.

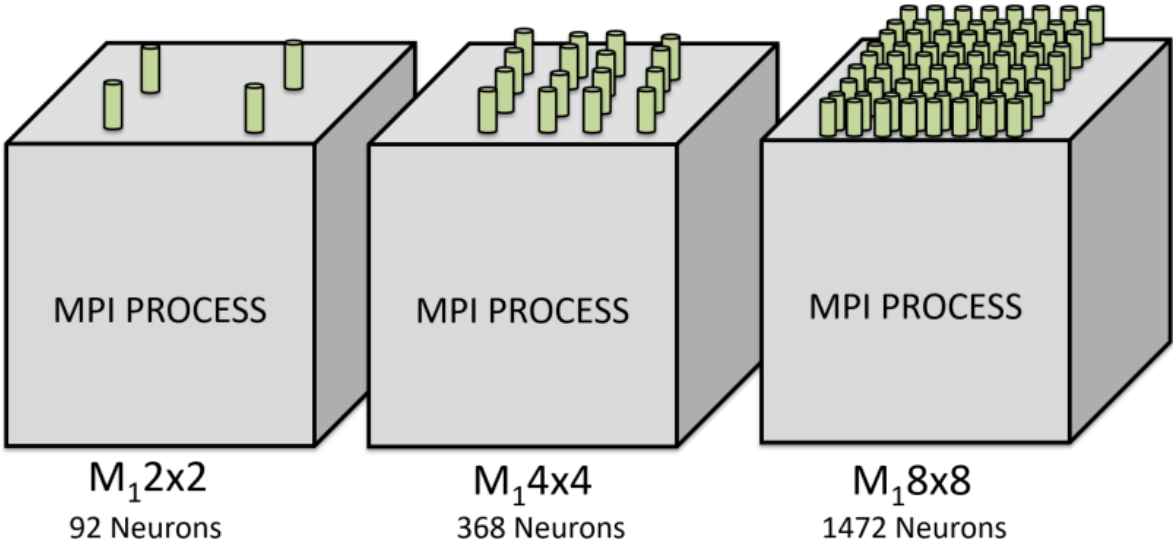


Figure 3.3: First model extension: Increasing the number of neurons

Figure 3.3 shows our first model extension. We first increase the number of neurons that are simulated within the model and measure performance. In our first set of tests, we configure the model to simulate 4, 16, and 64 micro-columns per process. The configurations are called M(1, 2x2), M(1, 4x4), and M(1, 8x8) respectively. Note that M(1, 2x2) is our baseline model. Each micro-column holds 23 simulated neurons; therefore our three models simulate $1,472$, $5,888$, and $23,552$ neurons respectively. We

configure the tests to run in a time-bound setting and in a step-bound setting. As in Section 3.3.1, for our time-bound tests we examine how the increase of neurons impact initialization time. For step-bound tests, we measure how computation times change, as we increase the number of neurons in the model. We measure the impact of I/O on each of these configurations by simulating the model with and without I/O.

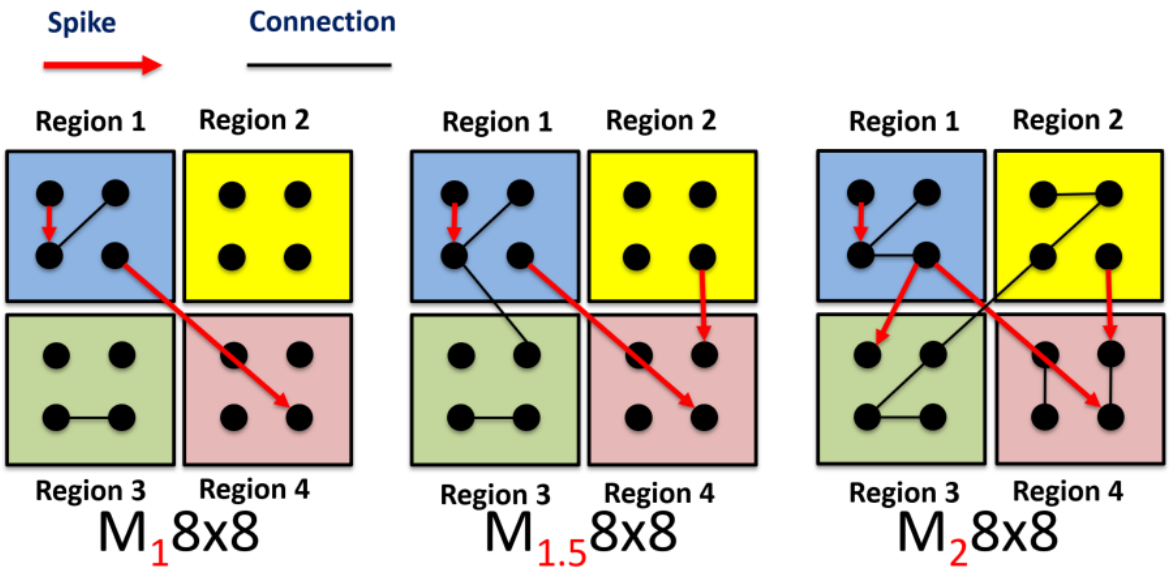


Figure 3.4: Second model extension: Increasing the number of connections

Figure 3.4 shows our first model extension. In a second set of tests, we keep the number of micro-columns and number of neurons constant (i.e., 64 micro-columns and the 23,552 neurons respectively) but increase the level of connectivity between neurons from the connectivity factor equal to 1 in M(1, 8x8), to 1.5 in M(1.5, 8x8), and to 2 in M(2, 8x8), respectively. In terms of synaptic connections, the models consist of 2,000, 3,000, and 4,000 synaptic connections correspondingly. By increasing the connectivity factor to 1.5 in M(1.5, 8x8), the number of connections increases by 50%, compared to M(1, 8x8), whereas M(2, 8x8) increases its connections between neurons by 100%. Additionally, we increase the total time GENESIS runs from our original time limit of 1 hour to 2 hours because for such a complex model, the initialization time for this model can approach and exceed 1 hour. Each micro-column is composed of roughly 23

neurons and the number of neurons is held constant across the three models but their degree of connectivity increases.

3.3 Results

3.3.1 Platform Impact on Performance and I/O Results

We run simulations of the baseline model M(1, 2x2) in parallel by using 16 processes. On Unitank, the 16 processes are hosted on the single server and share the same shared memory. On Excalibur and Spirit, each process is hosted on a node.

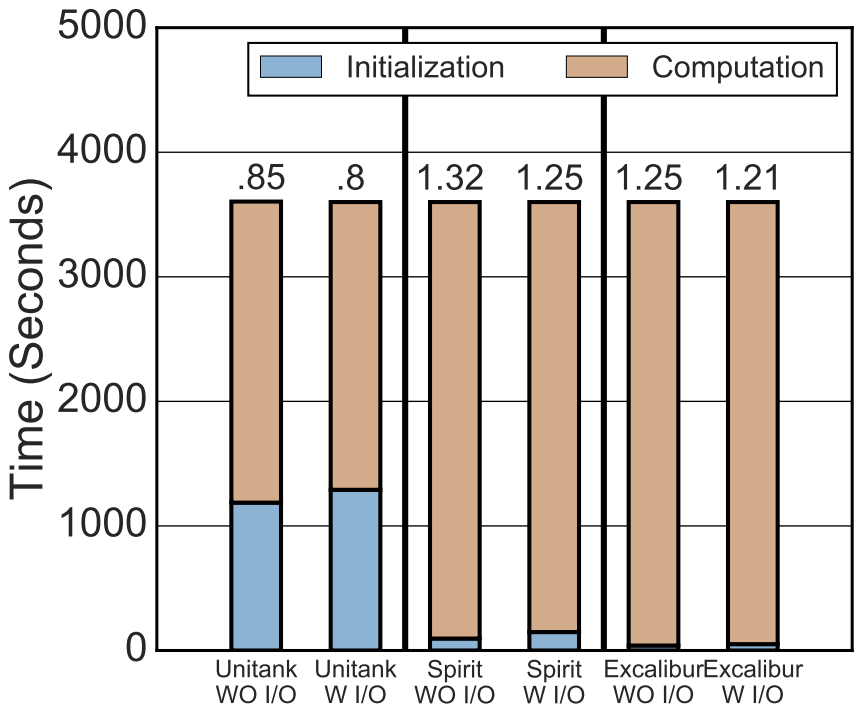


Figure 3.5: Time-bounded execution times and neocortex simulated times of the M(1, 2x2) model on Unitank, Spirit, and Excalibur, without I/O (WO) and with I/O (W).

Figures 3.5 shows the performance of the baseline model M(1, 2x2) on Unitank, Spirit, and Excalibur as bar charts for time-bounded simulations of 1 hour, while I/O is turned off and on. The blue bars represent the amount of time spent in initialization. The one-time step consists of reading the model scripts, parameterization of the model based on the values outlined in the script, establishing the neural network, and writing

the network out to disk, if the I/O is on. The brown bars represent the time to perform the iterative solver steps (i.e., simulating the neocortex functions). Without I/O, the simulation does not write any scientific and statistical information to disk. With I/O, after each compute step, descriptive behavioral statistics are written out to disk (e.g., number of spikes in a micro-column). The number above each bar is the total amount of simulated time of neuronal network activity that the model achieves in 1 hour of simulation time. This value can be considered a measurement of the amount of science performed by the model.

In Figure 3.5, we see that Spirit and Excalibur have small initialization times and deliver over 1 second of simulated brain activity. Excalibur simulates in average 1.25 seconds without I/O and 1.21 with I/O, whereas Spirit simulates in average 1.32 seconds without I/O and 1.25 with I/O. Spirit narrowly outperforms Excalibur in the amount of science delivered because of its faster processor with a larger L3 cache. On the other hand, Unitank takes a significantly large amount of time to initialize the model, and thus it only simulates close to 0.85 seconds and 0.80 seconds without and with I/O respectively. The higher cost in initialization for Unitank is due to contention of the 16 processes to access the shared memory; the contention results in a larger initialization time and less time for the iterative steps and thus less science delivered to the scientists. The time to initialize the model is a function of the amount and type of resources available for the simulations. The comparison of overall performance without and with I/O outlines how the baseline model M(1, 2x2) is not substantially impacted by the write operations. With I/O, Excalibur science delivered decreases of 3.5%, whereas Spirit and Unitank show a decrease in delivered science of 5.1% and 5.7% respectively.

The simulations in Figure 3.5 reaches different simulated time (amount of iterative steps performed) but all simulations perform the same initialization and write to disk the same amount of data, if I/O is on. Therefore, a comparison of the performance for the initialization times can provide us with valuable insights on the impact

of I/O on initialization across the three platforms. Data written during the initialization are stored in files of three types (i.e., randomspike, long-range, and short-range). Table 3.1 describes the average sizes of these files (in Byte), number of writes per files, and number of files performed during the initialization of the time-bound simulation of M(1, 2x2). The small amount of writing is consistent with the small impact of I/O on initialization performance in Figure 3.5.

	File size (Bytes)		Number of Writes		Quantity
File Type	μ	σ	μ	σ	
Randomspike	253	7	8	0	1,040
Long-range	4,028	50	51	51	592
Short-range	73,066	71,172	50	51	1,041

Table 3.1: Amount and type of writing performed by a time-bound simulation of the baseline model M(1, 2x2) during the initialization phase (i.e., mean values μ and standard deviation σ).

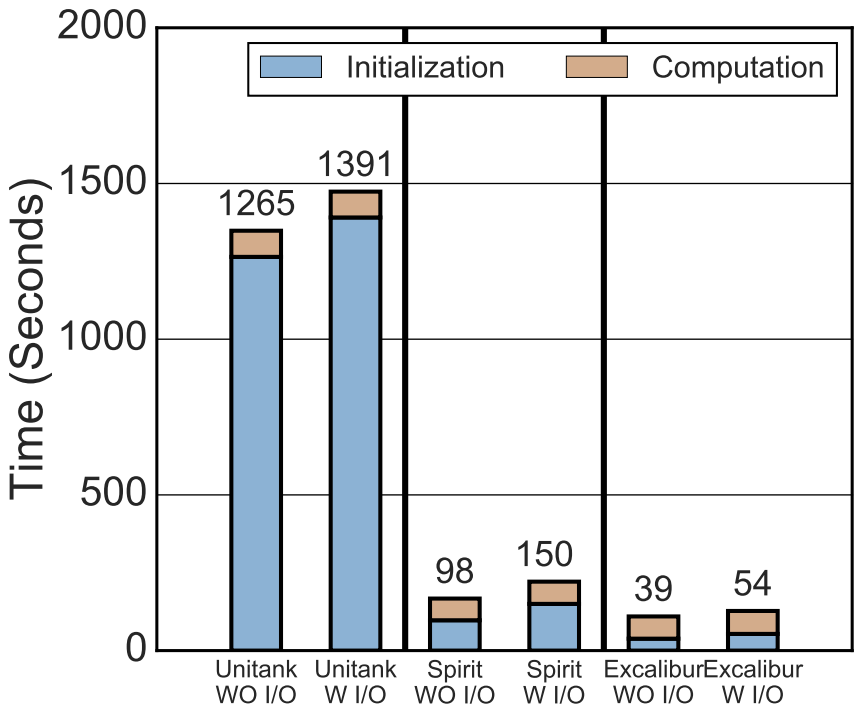


Figure 3.6: Step-bounded execution times for 500 steps (0.25 seconds of neocortex simulated time) of the M(1, 2x2) model on Unitank, Spirit, and Excalibur, without I/O (W) and with I/O (W).

Figure 3.6 shows the performance on each platform without and with I/O for iteration-bound simulations (i.e., when a fix number of 500 iterative steps is executed). When comparing the wall-clock time of 500 iteration steps, we see how Excalibur offers the best performance. Excalibur is able to simulate 500 steps in roughly 110 seconds without I/O and 127 seconds with I/O. Spirit is able to simulate 500 steps in roughly 167 seconds without I/O and 221 seconds with I/O. Excalibur outperforms Spirit because of its faster interconnect which allows the supercomputer to establish connections between neurons quicker, and to begin computation earlier. Finally, Unitank is able to simulate 500 steps in roughly 1,348 seconds without I/O and 1,474 seconds with I/O. The comparison of the times for the iterative steps outline how for a small model such as M(1, 2x2) the simulation time is very similar across platforms and the performance gains is mainly related to the cost of the initialization.

Table 3.2 shows the amount and type of writing performed by the step-bound simulations of the baseline model M(1, 2x2) during the fix number of iteration steps (i.e., 500). Data written during the iterative steps are stored in four types of files (i.e., spikehist, randomspikehist, LFP, and membrane). The amount of data written over the 500 steps of the step-bound simulations of model M(1, 2x2) and the associated writing frequencies are quite modest for a large-scale scientific simulation. The comparisons of simulation times without and with I/O outline a slight increase of time associated to the I/O operations; the increase is modest as the amount of data written to disk is.

	File size (Bytes)		Number of Writes		Quantity
File Type	μ	σ	μ	σ	
Spikehist	1,091	21	26	0.5	16
Randomspikehist	262	88	8	3	16
LFP	11,836	523	500	0	80
Membrane	25,714	7,391	500	0	336

Table 3.2: Amount and type of writing performed by a 500 step-bound simulation of the baseline model M(1, 2x2) during the computation phase (i.e., mean values μ and standard deviation σ).

3.3.2 Model Impact on Performance and I/O Results

Figure 3.7 presents the results of time-bound simulations of the three models (i.e., M(1, 2x2) M(1, 4x4), and M(1, 8x8)) on Excalibur. The number of compute nodes does not change for the three models and in each test, we run one GENESIS process per compute node. GENESIS is configured to run for 1 hour of wall-clock time. We measure the total simulated time of the neocortex functions achieved, the time spent during the model initialization, and the iterative computations or steps. Simulated time achieved within the hour is shown above each of the respective bars. Figure 3.7 shows each of the model's performance without I/O and with I/O is on.

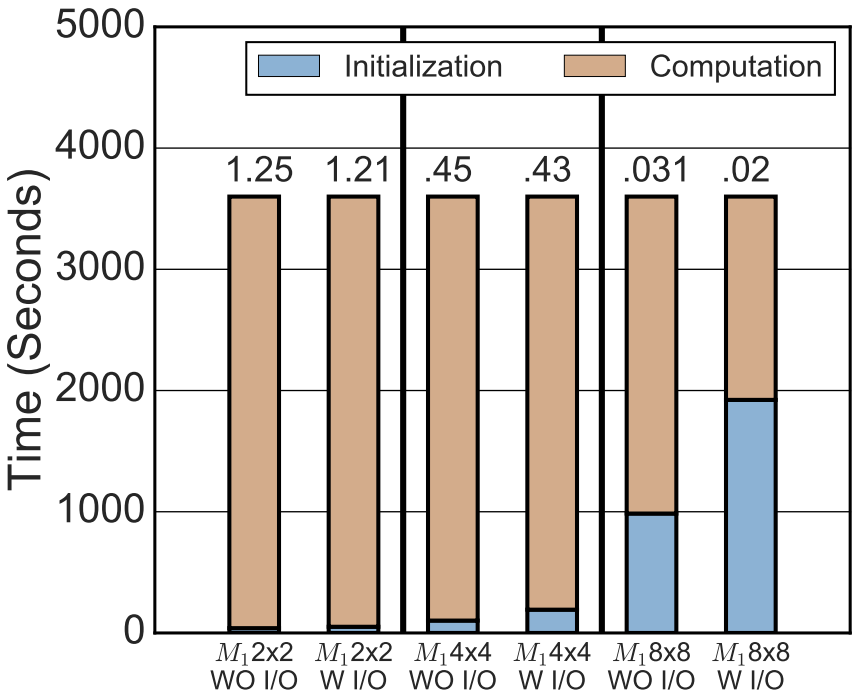


Figure 3.7: Impact of increasing the number of neurons on performance and neocortex simulated times for Excalibur's time-bound simulations without and with I/O.

When comparing the simulated time of the neocortex functions without I/O across the three models, we observe that the simulated time decreases by $\approx 2/3$ when moving from 4 to 8 micro-columns per node (i.e., from 1.25 seconds to 0.45 seconds) and becomes $\approx 1/15$ when moving from 4 to 64 micro-columns per node (i.e., from 1.25

seconds to 0.03 seconds). Simulations of M(1, 2x2) with I/O exhibit a slight decrease in simulated time of 3.3% compared to the same simulations with I/O; simulations of M(1, 4x4) with I/O exhibit a decrease in simulated time of 4.6%. However, when increasing the amount of micro-columns to 64 as it is in M(1, 8x8) the simulated time substantially decreases by $\approx$ 60% compared to the simulations with I/O. In the neocortex models considered in our work, connections among neurons are established through the passing of synchronous buffered MPI messages in the initialization phase, prior to the computation. At the end of the initialization phase, the entire network of interconnected neurons is written to disk. As the number of neurons increase, the number of connecting neurons and the associated MPI-based communications increase; at the same time, the number of networks of interconnected neurons written to disk increases.

Figure 3.7 outlines how the increasing number of connecting neurons and networks of interconnected neurons can cause an increase in initialization times, and ultimately, a decrease in science delivered when the amount of neurons is very large. When the models are small in number of connecting neurons, such as for M(1, 2x2) and M(1,4x4), the number of connecting neurons is still modest. Hence, without I/O, M(1, 2x2) simulations spend an average of 40 seconds in the initialization phase; and M(1, 4x4) simulations spend an average of 102 seconds in the same initialization phase. With I/O, the initialization times only slightly increase. For M(1, 2x2) simulations and M(1, 4x4) simulations both experience a slight increase in initialization time (i.e., M(1, 2x2) spends an average of 51 seconds in the same initialization phase and M(1, 4x4) spends an average of 192 seconds).

On the other hand, when the models have a larger number of connecting neurons, for example with 64 micro-columns in M(1, 8x8), we observe that the initialization time increases substantially even without I/O. Without I/O the increase in initialization time is mainly due to the increase in connections among the 23,552 neurons. With I/O, the further increase of 100% compared with the same initialization without I/O from 985 seconds to 1,922 seconds can be explained by measuring the volume of files

produced during the initialization as the number of cell increases. Table 3.3 shows the amount of writing for the files randomspike, long-range, and short-range when the number of neurons increases. As we increase the number of neurons, the files that are written to disk during the initialization phase of the simulation grow in size. While models M(1, 2x2) and M(1, 4x4) generate limited amount of data that explains the slight increase of initialization time and decrease in simulated time when I/O is on, M(1, 8x8) with its 64 micro-columns data generates gigabytes of data during the initialization alone, causing the further increase in initialization time and decrease of simulated time observed in Figure 3.7.

	File size (Bytes)		Number of Writes		Quantity
File Type	μ	σ	μ	σ	
M(1, 2x2)					
Randomspike	253	7	8	0	1,040
Long-range	4,028	50	51	51	592
Short-range	73,066	71,172	50	51	1,040
M(1, 4x4)					
Randomspike	257	6	8	0	4,160
Long-range	16,298	16,298	177	202	2,368
Short-range	294,830	287,697	177	202	4,160
M(1, 8x8)					
Randomspike	259	6	8	0	16,640
Long-range	55,134	65,681	684	809	9,472
Short-range	1.19e+03	1.16e+03	684	809	16,640

Table 3.3: Amount and type of writing performed by a time-bound simulation of models M(1, 2x2), M(1, 4x4), and M(1, 8x8) during the initialization phase (i.e., mean values μ and standard deviation σ).

Figure 3.8 presents the results of step-bound simulations of the three models M(1, 2x2), M(1, 4x4), and M(1, 8x8) on Excalibur. Each model runs the same number of iterative steps. Therefore, we focus on how time spent in computation is affected by the number of neurons simulated. In Figure 3.8, we observe that by increasing the number of neurons, the computation time increases. From M(1, 2x2) to M(1, 4x4), the computation time increases of 75%; from M(1, 2x2) to M(1, 4x4), the computation time is 10x larger. In the step-bound tests, we observe no major differences in time

when I/O is on or off. Table 3.4 shows the size, quantity, and frequency of writes for

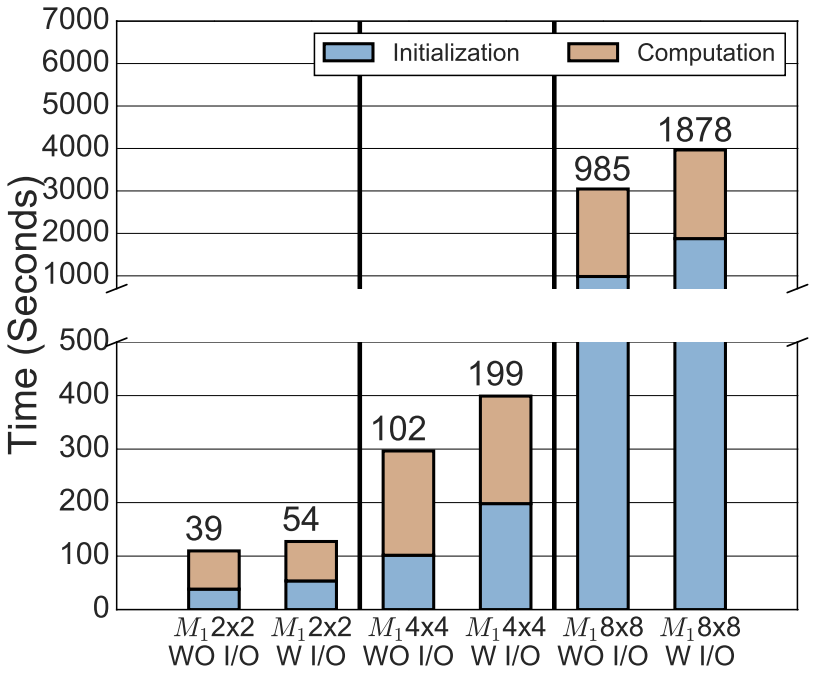


Figure 3.8: Impact of increasing number of neurons on performance for Excalibur's step-bound simulations of 500 steps (0.25 seconds of neocortex simulated time) without and with I/O.

the files that are written after each step of the computation phase. Although the size of files produced at each step increases as we increase the number neurons, the volume of these files are not large enough to affect performance in a significant way. We can thus deduce that it is feasible continue to increase the amount of neurons within the model without severely impacting the amount of time it takes to simulate the model with I/O on.

In a second set of tests, we keep the number of micro-columns and number of the neurons constant (i.e., 64 micro-columns and the 23,552 neurons respectively) but increase the level of connectivity between neurons from the connectivity factor equal to 1 in M(1, 8x8), to 1.5 in M(1.5, 8x8), and to 2 in M(2, 8x8), respectively. In terms of synaptic connections, the models consist of 2000, 3000, and 4000 synaptic connections correspondingly. By increasing the connectivity factor to 1.5 in M(1.5, 8x8), the

	File size (Bytes)		Number of Writes		Quantity
File Type	μ	σ	μ	σ	
M(1, 2x2)					
Spikehist	1091	21	8	0.5	16
Randomspikehist	262	88	8	3	16
LFP	11,836	523	500	0	80
Membrane	25,714	7,391	500	0	336
M(1, 4x4)					
Spikehist	4,476	62	106	1	16
Randomspikehist	1,116	263	34	8	16
LFP	11,882	34	500	0	80
Membrane	90,200	29,560	500	0	336
M(1, 8x8)					
Spikehist	26,016	6,163	594	132	16
Randomspikehist	4,648	575	141	17	16
LFP	11,841	47	500	0	80
Membrane	348,177	118,239	500	0	336

Table 3.4: Amount and type of writing performed by a step-bound simulation of models M(1, 2x2), M(1, 4x4), and M(1, 8x8) during the computation phase (i.e., mean values μ and standard deviation σ).

number of connections increases by 50%, compared to M(1, 8x8), whereas M(2, 8x8) increases its connections between neurons by 100%. Additionally, we increase the total time GENESIS runs to 2 hours because for such a complex model, the initialization time for this model can approach and exceed 1 hour. Each micro-column is composed of roughly 23 neurons and the number of neurons is held constant across the three models but their degree of connectivity increases.

Figure 3.9 shows the time-bound simulations of the models. With I/O turned off, the simulated time decreases by 50% and the initialization time increases by 50%, with an increase of 50% in cell connections. The slowdown is associated to the increase in connectivity that proportionally increases both the synchronous buffered MPI messages exchanged in the initialization phase. With I/O tuned on, the simulated time further decreases in average of roughly 21% and the initialization time increases in average of 41%, as the number of synaptic connections written to disk increases. Table 3.5 motivates the performance lost in terms of amount and type of writing performed by

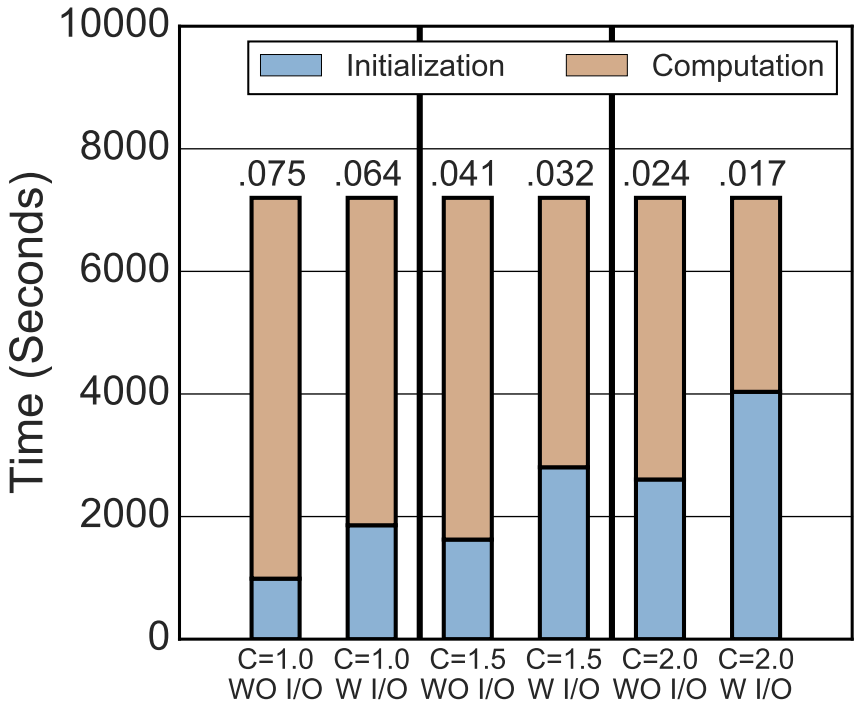


Figure 3.9: Impact of increasing cellular connectivity on performance and neocortex simulated times for Excalibur's time-bound simulations without and with I/O.

the time-bound simulations with I/O. From the table we observe that as we increase the amount of connectivity in the model the size of the files also increased. Moreover, the amount of writes to the files associated with the initialization phase increases.

Figure 3.10 shows the step-bound simulations for the three models. As we increase connectivity, computation time during the simulation stays relatively flat. Moreover, increasing connectivity does not significantly increase the effect of I/O during computation phase (i.e., within 1% with and without I/O). The impact of connectivity on performance during the computation phase is not tangible due to the fact that the files that are influenced by connectivity are written in the initialization phase. Furthermore, we observe that by increasing the connectivity, no increase in the level of spiking within the model is observed and the number of files associated with the iterative step is too small to be a bottleneck to the simulation performance.

	File size (Bytes)		Number of Writes		Quantity
File Type	μ	σ	μ	σ	
M(1, 2x2)					
Randomspike	259	6	8	0	4,160
Long-range	55,134	65,681	684	809	2,368
Short-range	1.2e+03	1.2e+06	684	809	4,160
M(1, 4x4)					
Randomspike	259	6	8	0	16,640
Long-range	82,258	95,936	1,019	1,182	9,472
Short-range	1.7e+03	1.7e+03	1,019	1,182	16,640
M(1, 8x8)					
Randomspike	259	6	8	0	16,640
Long-range	109,623	127,045	1,355	1,565	9,472
Short-range	2.3e+06	2.3e+06	1,355	1,565	16,640

Table 3.5: Amount and type of writing performed by a time-bound simulation of models M(1, 2x2), M(1, 4x4), and M(1, 8x8) during the initialization phase (i.e., mean μ and standard deviation σ).

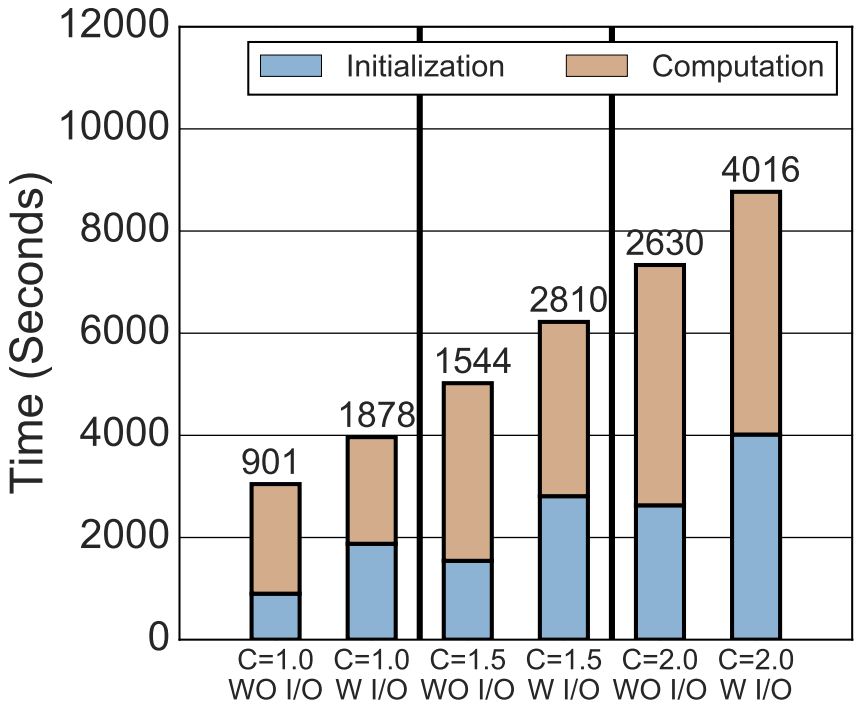


Figure 3.10: Impact of increasing cellular connectivity on performance for Excalibur's step-bound simulations of 500 steps (0.25 seconds of cortex simulated time) without and with I/O.

3.4 Discussion

Advances in hardware technology and simulation methods have enabled scientist to analyze data and behavior that is not available *in vitro* or *in vivo* at unprecedented scales. However, the resolution of data available via simulation is far from the resolution and detail to that of real-life experimentation. At present, current HPC hardware fail to meet the computational and data demands of higher fidelity models, forcing scientists to use scaled-down models. To increase the quality of science achievable via simulation more work is needed in researching new data structures and simulation technologies which efficiently leverage present and next-generation HPC hardware. Investigating the impact of model complexity on simulation performance is important if we are to be able to accurately predict the needs of future increasingly complex models. Current models are orders of magnitude less than what is needed to study brain-level behavior. Being able to predict within ballpark range the computational needs of future models will eliminate the possibility of under procurement of hardware.

In this chapter, we analyze the impact of HPC resources (i.e., single fat nodes and high-end clusters) on performance, data generation, and science delivered by GEneral NEural SImulation System (GENESIS) for increasingly complex models of the brain's neocortex. Subsection 3.3.1 showed that there is a discrepancy between the diverse platforms used for neural network simulation. Initialization time we the main factor for the loss in simulated biological time. We saw that by migrating the model to HPC we were able to increase our scientific throughput by 57%. In subsection 3.3.2 we explored the effects of model complexity on scientific throughput and data generation. As we increased the complexity of the models we witnessed super linear growth in initialization time. And, as a consequence of this, a loss in simulated biological time. The cost of initializing the model is a challenge to larger simulations in the model, in the following chapters we present our solutions for overcoming this obstacle.

At present, most domain scientists perform their simulation studies on simple fat nodes such as the one used in this study. The principal reason for this is cost, larger clusters are just not economically practical. By migrating the model from a single fast

node to an HPC cluster we were able to increase the amount of parallel hardware at our disposal and as a result were able to increase our scientific throughput. Additionally, we are able to identify the propagation of events via the synaptic connections as the primary culprit in why the execution time is increasing, and therefore more focus should be placed on how to better accomplish this task in hardware.

Lastly, we list the limitations of this and directions for future work. In this, we focused on the GENESIS simulation system and a single model of a neuronal patch. Due to the narrow focus of this work, it is possible that our observations are not ones that can be made in general for other neuronal networks. However, we are strongly confident that our generalizations are applicable to most simulated spike-coupled neuronal networks executed on HPC. In addition, this worked focused on observing the impact of model complexity to overall execution time. Future work will consist of an analysis of how model complexity impacts resource utilization (i.e., memory and CPU). Finally, GENESIS is a CPU only code, understanding how accelerators can be leveraged for the simulation of simulated neuronal networks is a direction of future work.

Chapter 4

INTEGRATION OF *IN SITU* ANALYSIS & TRADEOFFS

In the previous chapter we studied the effects of model complexity on scientific throughput. When increasing the complexity of the model we witnessed a large growth in the amount of time spent in initialization the model. This presents a challenge to larger more complex simulations. In this chapter, we look to address the above mentioned challenge, we explore the integration of *in situ* analysis. We evaluate the *in situ* analysis using both statistical and empirical methods and present a study of the analysis' resource consumption and scaling ability. We show how our *in situ* analysis can generate substantive scientific insights comparable to post-simulation analysis, despite its local data view.

The rest of this chapter is organized as follows. Section 4.1 describes the integration of in situ analysis with a model of neocortex, along with the analyses performed in detail. Section 4.2 presents tests we use to evaluate the validation of our in situ analysis method using both empirical and statistical techniques. Section 4.3 summarizes the current status of the research for this thesis.

4.1 *In Situ* Analysis

In chapter 3 we introduce the sequential workflow for the simulation of neuronal networks models executed on GENESIS. We highlight one of the main impediments to scientific throughput, which is the movement of data from the parallel file system on the large-scale cluster to the local storage of a dedicated analysis machine. In this chapter we address another pitfall of the sequential workflow, which is the underutilization of computational resources. Figure 4.1 demonstrates what we mean by the underutilization of resources. When the user performs analysis on the data, HPC resources

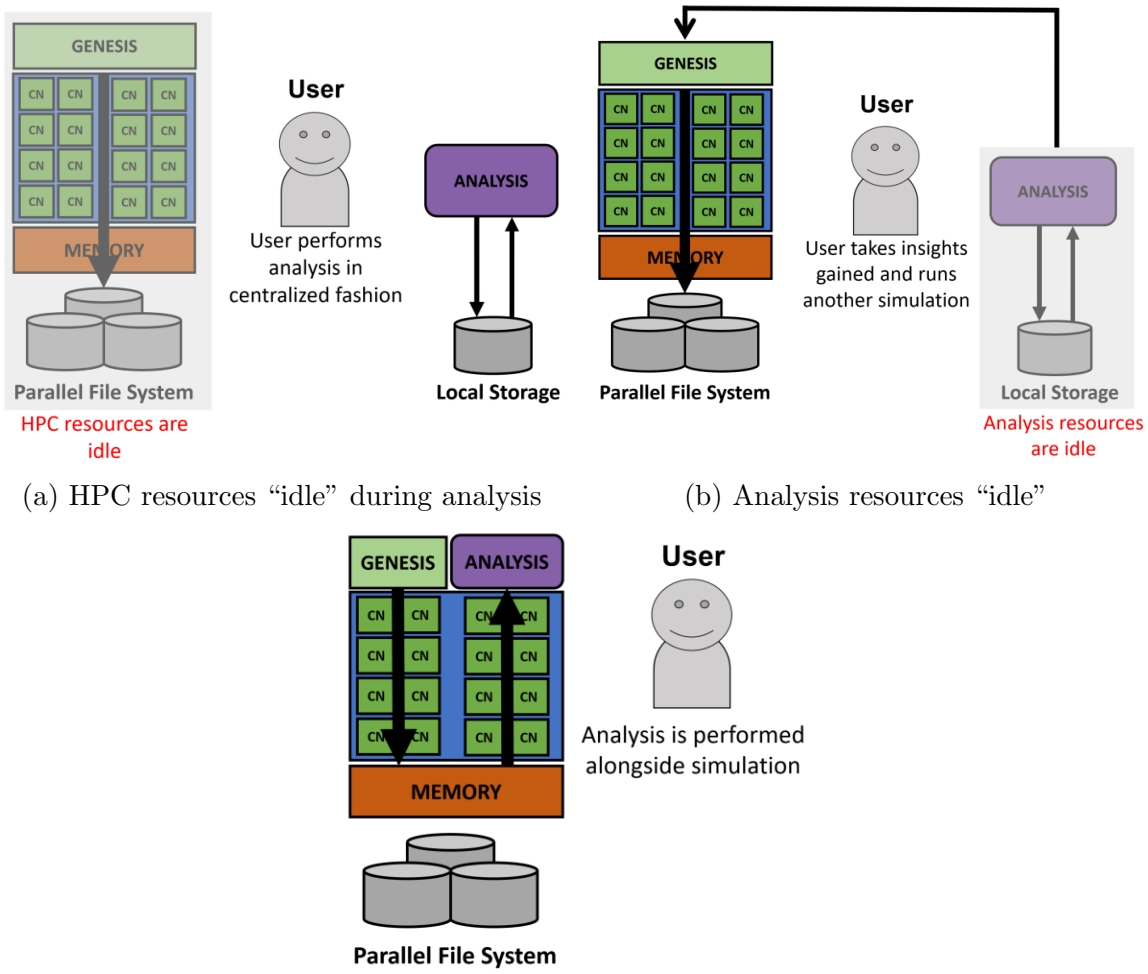
are unused and thus idol. In contrast, during simulation time the analysis resources are not used. Our solution to improve utilization and as a consequence improving scientific throughput is to couple analysis with simulation. By coupling simulation and analysis we enable the user to increase scientific yield, allowing more analysis per execution, reducing the impact of initialization cost. Figure 4.1c shows the evolution of our methodology we now couple the analysis with the simulation on the HPC resources.

4.1.1 GENESIS and Analysis

Simulating neuronal network models assists in the study of the functional connectivity of the brain. The type of simulations that we augment with our *in situ* analysis method is a simulation of a patch of neurons in the neocortex using the General Neural Simulation System (GENESIS) [49] software suite.

The neocortex comprises the outer gray matter of the brain [84] and is organized into six layers and is home to roughly 14-20 billion neurons, responsible for the majority of sensory and cognitive processing in mammals [84]. The human neocortex measures at approximately 2.4 mm in thickness and with its folds (sulci and gyri) make up 76% of the brains volume. All mammals have the same number of the layers. It has been estimated that there are 100,000,000 cortical minicolumns each comprising about 110 neurons each [85]. The neocortex is comprised of different "compartments" that each perform different functions.

Our model of the neocortex is composed of 14 biophysically detailed neuron types, organized into six layers representing a 150 x 150 x 2871 μm patch of cortex. The simulated domain of the model is comprised of a matrix of micro-columns, which are vertical columns that extend through the six layers of the brain. In our simulations, each column contains 16 neurons. Individual neurons are modeled using multiple compartments to generate realistic electrical activity in the 1-100 Hz frequency range as an emergent property of the underlying physics. In this paper, our analyses focus on two main components in the model which are the local field potential (LFP) and the membrane potential (MP). LFP are the distance weighted sum of the simulated



(a) HPC resources "idle" during analysis

(b) Analysis resources "idle"

(c) Analysis occurs with simulation

Figure 4.1: Underutilization of HPC resources occur at two different times. First is when the user is performing analysis on the simulation output after the simulation has terminated. During this time the resources that are dedicated to running the simulation are left unused while the user is performing analysis. Second is when the user is taking intuitions and knowledge gained from the analysis and using those to configure and re-run the simulation. During this time the dedicated analysis resources are left unused.

extracellular electrical activity as measured at fixed electrode positions. MP are the electrical potential difference between the interior and exterior of a single compartment of a cell. In our analysis approach, we focus on the membrane potential at the main body of the neuron, known as the soma.

GENESIS is the general software suite that we augment with our *in situ* analysis

method. GENESIS enables scientists to build biologically accurate neuronal network models including our neocortex model. GENESIS is very powerful and its models can range from the simple level of small sub-cellular processes to sophisticated large neuronal networks. GENESIS is written in C++ and utilizes the Message Passage Interface (MPI) for communication. Simulations executed using GENESIS are solved in discretized timesteps in which, ordinary differential equations governing individual compartments of the model are solved. A significant challenge to the scalability of neuronal simulators on high-end clusters is the limited amount of memory available per core. Memory consumption in our brain simulations are dependent on two aspects: the size of the brain model (e.g., the size of the simulated patch in terms of neurons) and the number of processes used for the simulation. Figure 4.2 demonstrates the memory requirements of GENESIS as we increase the number of processes from 16 to 4,096 processes when simulating a patch of neurons in the neocortex of approximately 256 to 70,000 neurons. The results are collected on the Air Force Research Laboratory's Thunder supercomputer. Thunder is an SGI ICE X system, that is comprised of over 3,000 compute nodes. Thunder is configured with 128 GB (118 GB available to user) of memory per node and has 36 Intel E5-2699 cores clocked at 2.3 GHz. With 16 processes, GENESIS requires 3 GB of RAM. As we increase our simulation size to 4,096 processes (i.e., 288 nodes), GENESIS memory requirements greatly increases to 115 of the 118 GB of user available memory. Model scales higher than 4,096 MPI processes are not possible due to memory being completely saturated and GENESIS being terminated prematurely due to lack of free memory and is killed by the operating system kernel.

4.1.2 Scientific Analysis and In Situ Approach

We perform our primary analysis on the output of the GENESIS simulations mimicking the neocortex behavior. We use both a traditionally used post-simulation approach and our *in situ* analysis approach.

From the scientific point of view, we use power spectral density estimation to

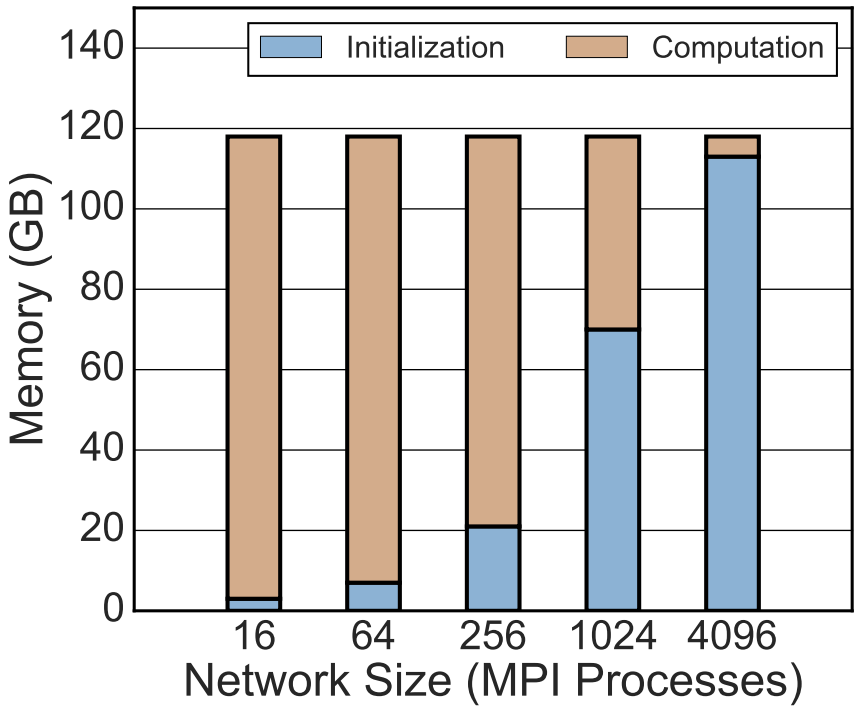


Figure 4.2: GENESIS memory usage when scaling the simulated neuronal network from a network size of 16 (256 neurons) to 4,096 (70,000 neurons) MPI processes running on AFRL Thunder supercomputer.

estimate the power of the extracellular voltages surrounding neurons. Spectral analysis is a sub-form of time series analysis which is concerned with quantitatively characterizing relationships between series of samples ordered in time. Here this analysis allows us to understand the frequency content of the extracellular voltages and determine how samples are different or are related to one another.

Implementation-wise, we perform spectral analysis by transforming voltages from the time domain (i.e., local field potential or LFP activity) to the frequency domain (i.e., power spectral density or PSD estimates) and by observing which frequencies exhibit the highest power/energy. Local field potential activities are a continuous process and are comprised of a series of continuously varying voltages in time. We employ Belch's method [86] to calculate the power spectral density estimate of the LFP and perform our PSD estimation as a Short-time Fourier transform (SIFT). Pads show what frequencies exhibit the strongest oscillatory activity in the model, and are

the primary way of visualizing electrical activity in the brain.

To electrical activity in specific frequency ranges, neuroscientists are leveraging biologically accurate simulated models of the brain. Models of neuronal networks allow scientists to explore the impact of differential neuronal connectivity using analysis techniques and information not available experimentally. These models are inherently computationally intensive requiring more resources from both memory and I/O subsystems than what are currently available to most small research teams. Recently research in the simulation of neuronal network models have undergone a multitude of advances. Computational neuronal network models of are approaching sizes on the order of 10^5 neurons and 10^9 synaptic connections. However, this size is nowhere near what is needed to fully simulate the human brain which is theorized to be comprised of 10^10 neurons and 10^12 synaptic connections which is 1,000 times more than the number of stars in the Milky Way galaxy.

In order to meet the demands of these compute intensive simulations we need to design, implement, and utilize novel scientific analysis workflows that utilize modern analysis techniques such as *in situ*, *in transit*, and computational steering.

However, modeling realistic neurobiological processes and encoding them in computer simulations is challenging, as increasing computing and data requirements are all of concern. The large amount of data produced by high fidelity simulations are becoming increasingly difficult to save and transform into scientific insights for runtime simulations. One attractive solution is to modify the data analysis workflow from one that is accomplished post-simulation to one that is completed *in situ*. The integration of *in situ* analysis enables interactive methods for analyzing neuronal network models such as computational steering allowing for scientists to transform their their workflow one that is static to one that is dynamic and hypothesis driven.move from a global view (i.e., post-simulation analysis) to a local view (i.e., *in situ* analysis) of the data, we transform the STFT to work on a reduced subset of the local field potential signal by introducing sliding windows that shift over the local field potential signal with a sliding size of 10,000 timesteps. Choosing the length of the window for the STFT is

non-trivial. The selection is dependent on the analysis and data at hand. Increasing the window length increases accuracy in frequency but negatively effects time resolution and resource usage (i.e., specifically memory usage) and vice versa. Our approach performs analysis on one window of memory resident data at the time, meaning we are only concerned at the time of analysis with the data that fits within a single window. In other words, once a simulation begins, at each timestep of the simulation we are analyzing data that is present in memory (i.e., spatial locality) and contained within the current window (i.e., temporal locality). We have no view of future states of the model nor past collected knowledge before the window of interest. As we will show in Chapter 5 we use this local knowledge to steer the on-going brain simulation.

4.1.3 Integration of In situ Analysis

We build our *in situ* analysis on top of a general framework such as DataSpaces [21], a light-weight, scalable, and flexible abstraction to an in-memory staging area. DataSpaces enables the coordination of multiple components and services in addition to supporting dynamic coupling of applications. DataSpaces provides a set of query operators for the asynchronous insertion and retrieval of data to the storage space via RDMA for direct memory-to-memory data transfers. While DataSpaces provides a set of operators for the asynchronous insertion and retrieval of data to the storage space via RDMA, we still need to mold the framework to plug into the GENESIS simulations and to integrate the specific application analyses described in section 4.1. For each simulation process, we allocate a dedicated storage area from the global data domain of DataSpaces' shared abstraction space. Each analysis process ingests, indexes, and stores its respective column's data up to the defined window length into its allocated sub-domain. For each compute node we reserve two cores for the analysis while the rest are fully available for our simulations. All analysis is performed on data held in memory eliminating the need to write data to persistent storage.

To perform the *in situ* analysis (i.e., the transformation from the time to the frequency analysis and the extraction of knowledge on the ongoing simulation in terms

of extracellular voltages surrounding neurons), we write scripts using popular Python libraries SciPy and NumPy and plug the scripts into the adapted DataSpaces framework. SciPy contains modules for popular scientific and engineering tasks and signal processing. We heavily leverage the signal processing module of SciPy, which provides a method for estimating the spectral density of a signal using the periodogram method. NumPy supports the computation of large multi-dimensional arrays along with matrices. It provides an extensive selection of mathematical methods for the processing of arrays. The Python NumPy packages provides similar functionality to that which can be found in MATLAB.

While our work in this paper targets a specific simulated neuronal network model (i.e., the simulation of a patch of connected neurons in the neocortex), our overall workflow that combines simulations and *in situ* analysis can be used with other types of simulations using GENESIS or other software suites (i.e., NEST, NEURON, and Brian) with the same input models and output (i.e., local field potentials, membrane potentials, and spike output patterns).

4.2 Qualitative and Quantitative Validation

As outlined in section 4.1, we observe electrical activity through the use of spectral analysis of electrical potentials. Time-frequency analysis has been used successfully to study EEG signals. The STFT is one of the most fundamental methods of time-frequency analysis. STFT analysis allows for the simultaneous presentation of both the time and spectral content of the signal, which has been shown to improve feature extraction in EEG signals [87]. A STFT is implemented by introducing a fixed-sized sliding window to an input signal. We move this fixed-size sliding window by 10,000 timesteps at a time so that we have overlapping windows. Choosing the length of the window or the number of samples for the STFT is non-trivial. The selection of window size is highly dependent on the analysis, with the choice of window size raising a trade-off between frequency and time resolution. Increasing the window size provides a more

accurate view of changes in frequency. However, lengthening the analysis window negatively impacts the time resolution and monitoring changes in power over time becomes difficult. Vice versa, decreasing the window size improves time resolution but decreases the frequency resolution thus reducing the detail in the power spectrum. The major challenge is to find a length that can satisfy the requirements on both domains satisfying the necessary balance needed to facilitate computational steering. We perform both qualitative and quantitative comparisons of the spectral estimates (a) when using post-simulation estimates once the simulation is completed and all the simulation data is available versus (b) when using runtime, *in situ* estimates with data window lengths ranging from smaller-in-size 60,000 simulation timesteps to larger 200,000 simulation timesteps at different simulation phases (i.e., the initialization, in the middle, and the final phase of a simulation). Our simulations are performed on the Air Force Research Laboratory's Thunder supercomputer described in the previous section.

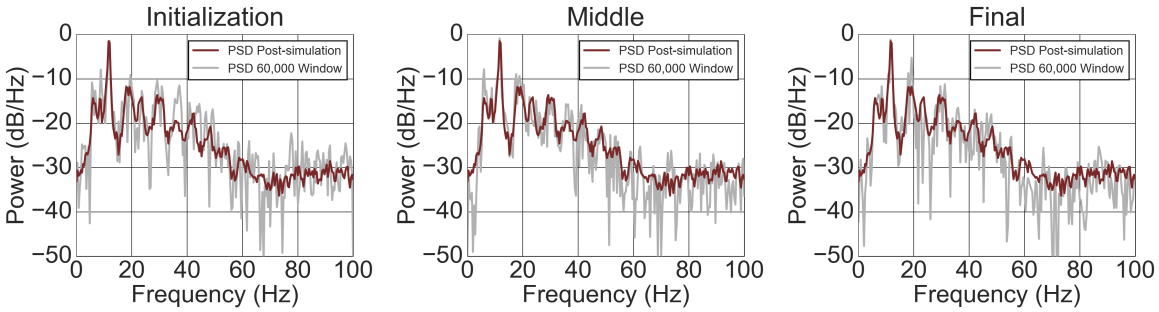
4.2.1 Qualitative Analysis

Qualitatively, we visually compare the post-simulation estimates to the *in situ* estimates by examining differences in the estimates' spectral content. We present the resulting scenarios from the three window lengths of 60,000, 100,000, and 200,000 timesteps using the *in situ* estimates in Figures 4.3a, 4.3b, and 4.3c, respectively. In each figure, the *in situ* estimates are in light-gray and the post-simulation estimates are in red. To ensure consistency across phases of a simulation, results are shown for each simulation in its initialization phase (i.e., between 90,000 and 150,000 timesteps for the smallest window of 60,000 timesteps; between 90,000 and 190,000 timesteps for the middle-size window of 100,000 timesteps; and between 90,000 and 290,000 timesteps for the largest window of 200,000 timesteps) and two following phases that we called middle phase (i.e., between 130,000 and 190,000 timesteps for the smallest window of 60,000 timesteps; between 170,000 and 270,000 timesteps for the middle-size window of 100,000 timesteps; and between 130,000 and 330,000 timesteps for the largest window of 200,000 timesteps) and final phase (i.e., between 190,000 and 250,000 timesteps for

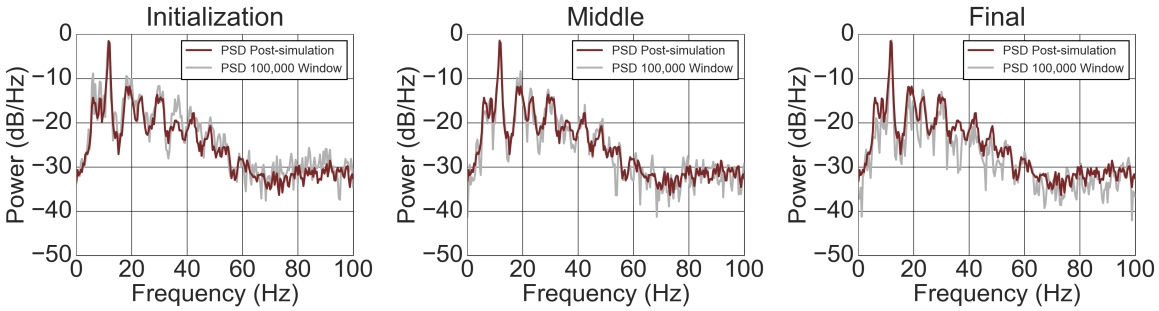
the smallest window of 60,000 timesteps; between 270,000 and 370,000 timesteps for the middle-size window of 100,000 timesteps; and between 170,000 and 370,000 timesteps for the largest window of 200,000 timesteps). By choosing the smallest window length of 60,000 timesteps or a slightly larger window length of 100,000 timesteps, we observe results in our *in situ* estimates that suffer from visible fluctuations compared with the observed post-simulation estimates. The fluctuations substantially reduce with a larger window length of 200,000 timesteps, suggesting the need for such a window length. At a window length of 200,000 timesteps our *in situ* results that are generated at runtime are visually comparable to analysis results obtained post-simulation (i.e., once the simulation has terminated). In other words, with a window of 200,000 timesteps, we are gaining meaningful scientific insights as the simulation evolves that were traditionally observable only once the simulation had terminated. Another empirical observation is that the fluctuation behavior described above does not depend on the simulation phase. In other words, window length is the factor driving accuracy and, when too small of a window is selected, its impact is tangible across the different simulation's phases.

4.2.2 Quantitative Analysis

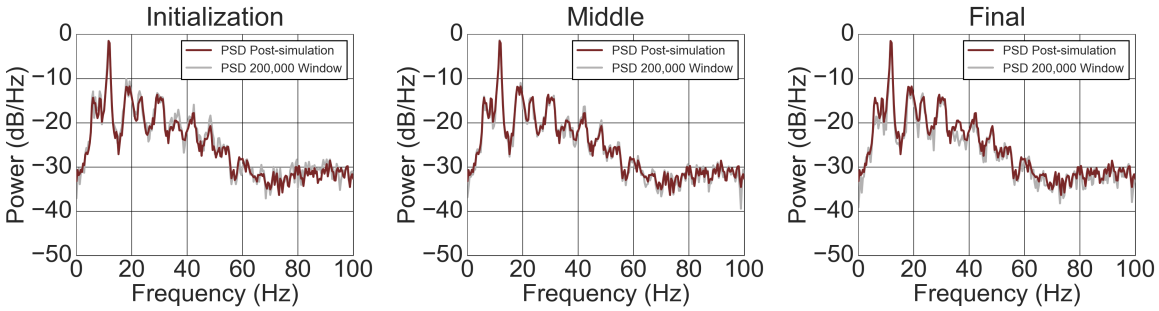
While the qualitative assessment confirms that as the window's length increases the amount of variability in the STFT decreases, its use cannot be extended beyond the visualization of the information. For steering our simulations at runtime, we need to quantitatively define a suitable window length that provides accurate scientific insights and, at the same time, does not result in memory spilling over to disk in an automatic way. Note that the memory and not the CPU is the principle contended resource between simulations and analysis [10]. To quantify the suitable window length for simulations' steering, we perform three statistical tests: (1) we measure the mean absolute spectral difference of estimates across frequency spectra; (2) we compare error versus window length and variation versus window length; and last (3) we perform the Wilcoxon Rank Sums Test for different window lengths and relate the accuracy to the



(a) Power spectral estimate of single column of simulated local field potential using window length of 60,000 samples.



(b) Power spectral estimate of single column of simulated local field potential using window length of 100,000 samples.



(c) Power spectral estimate of single column of simulated local field potential using window length of 200,000 samples.

Figure 4.3: Power spectral density estimate of simulated local field potential using window lengths 60,000, 100,000, 200,000.

memory usage. The three tests cross-validate each other outcomes.

Mean absolute spectral difference of estimates across frequency spectra: We first, evaluate the accuracy of computing the STFT over the varying field potentials during the simulation time using the spectral difference of the *in situ* estimates and post-simulation estimates. To this end, we measure the mean absolute spectral difference

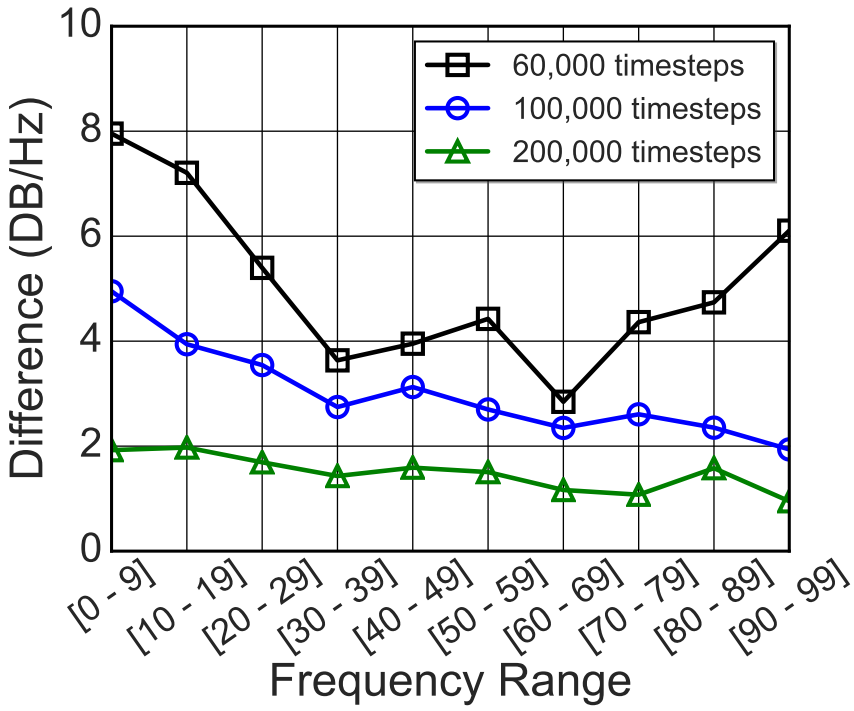


Figure 4.4: Spectral difference using windows 60,000, 100,000, and 200,000 timestep window lengths. The difference is taken from the analysis results obtained post-simulation in terms of the mean absolute spectral difference of estimates across the entire frequency spectrum.

of estimates across the frequency spectrum for the three windows lengths used in the qualitative assessment (i.e., 60,000, 100,000, and 200,000 timesteps). We consider a simulation of at least 800,000 timesteps that is evolving and has passed its initialization phase. Figure 4.4 shows the mean spectral difference between the windows and the baseline across 10 frequency bands over the range of 0-99 Hz. Our smallest window of 60,000 timesteps has the greatest spectral difference from the baseline through all frequency bands, confirming the difficulty that the shorter windows have with estimating the power spectra. In particular, the smallest window struggles the most at both edges of the spectrum: the power is mis-estimated by almost 10×10^8 dB/Hz at its worst. The longest window length of 200,000 samples produces the greatest accuracy, with the smallest recorded mean difference of 1 dB/Hz. In other words, our largest window length of 200,000 timesteps does the greatest job at consistently representing the

spectral content and features across spectrum with a lower, less variable mean absolute spectral difference. The observation confirms what we observed in the qualitative assessment (in section 4.2.1); here our findings rely on statistical metrics that can be computed automatically and thus can be used immediately at runtime, for example, in simulation steering.

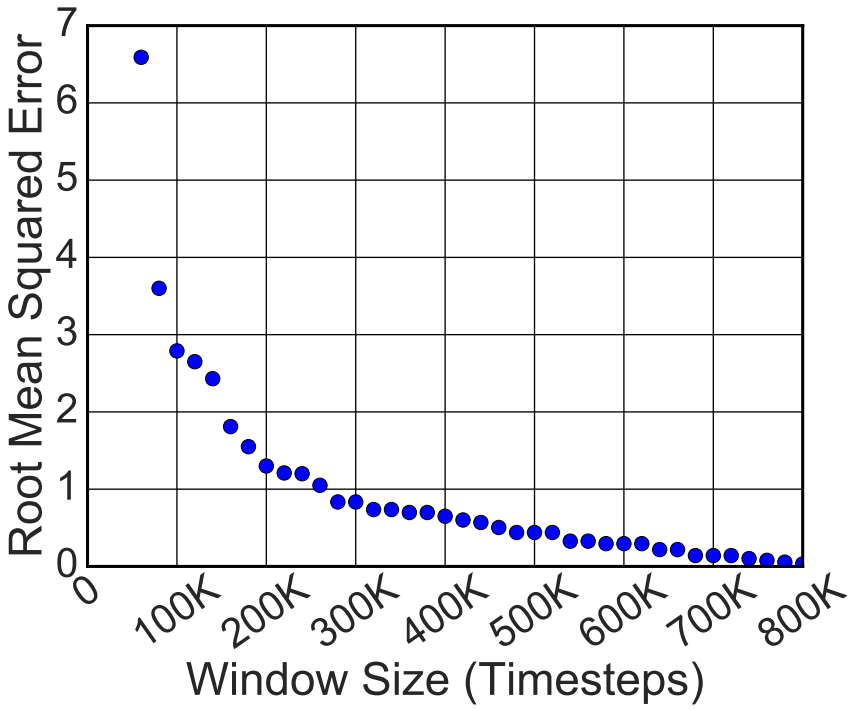


Figure 4.5: Error vs. window length

Error versus window length and variation versus window length: To expand our quantitative assessment beyond the three window lengths considered above, we measure the error and variation for different window lengths with a finer granularity ranging from 60,000 timesteps to 800,000 timesteps. To quantify the error between our *in situ* analysis spectral estimates and the post-simulation spectral estimates, we use the root mean squared error (RMSE) which is the standard deviation of the residuals or prediction errors. RMSE is a negatively slanted score thus, lower values are better. We choose RSME over other measurements of error because it has the benefit of associating greater weight and penalizes large errors, making it a better indicator of performance

differences. For our computation of RMSE we use the mean RMSE of the overlapping windows of the STFT from the post-simulation signal, averaging the errors to a single value. Figure 4.5, shows the RMSE for window length 60,000 to 800,000. We utilize the traditional "elbow criterion" to evaluate the window length's contribution to the reduction in error. Here we increase the window's length until we observe that further increasing the window's length does not significantly decrease the error. We observe the largest reduction in error occurring when the window length is increased from 60,000 to 200,000 timesteps. After 200,000 timesteps further increasing our window's length no longer substantially decreases the observable error.

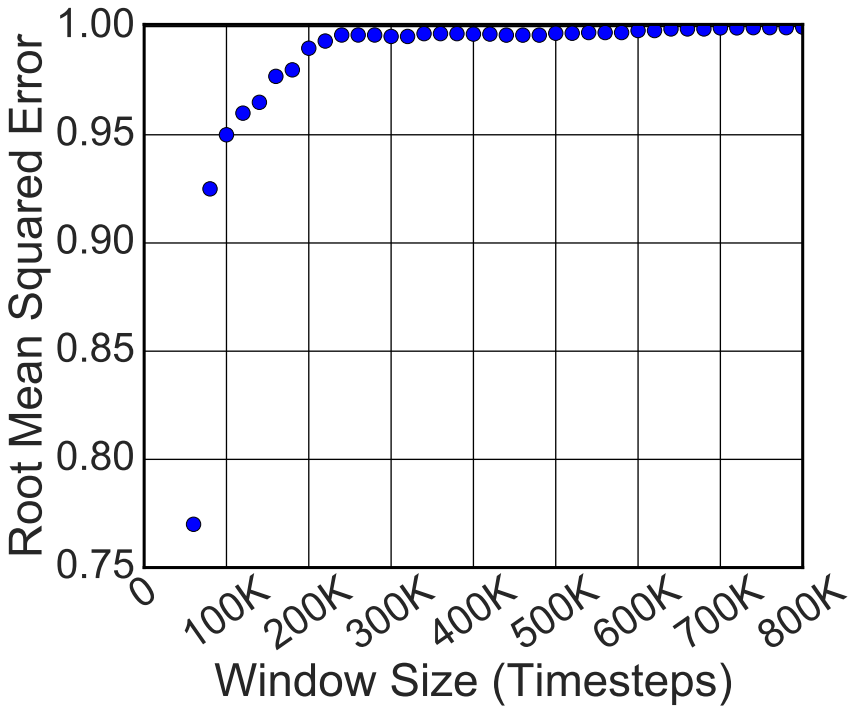


Figure 4.6: Variation vs. window length

To quantify the variation of the *in situ* and post-simulation analysis estimates, we use the coefficient of determination. The coefficient of determination also known as the R^2 score provides a measure of how well the *in situ* estimates represent the "true" power provided by the post-simulation estimate. It is a measure that is indicative of the level of explained variability in a data set. We use this measure to explain the level

of difference in variation among the post-simulation and *in situ* analysis results. The coefficient of determination in Figure 4.6 explains the difference in variation among post-simulation and *in situ* estimates by ranging between 0 and 1. A coefficient value closer to the value of one means that our *in situ* estimates are similar to the post-simulation estimates, and vice versa, a coefficient value closer to zero means a large dissimilarity. Figure 4.6 indicates that by increasing the window length, we decrease the variation among the estimates but the variation reaches a value of 0.98 as we reach the 200,000 timesteps. Once again, further increasing the window length does not result in a significant increase in the coefficient.

Evaluating descriptive statistics we see the same trend. We focus on the mean and variance. We compare their ability to accurately represent the overall post-simulation mean and variance for our window lengths. Computing the mean post-simulation and for each of the respective windows gives values of $-1.046e{-}07$, $-1.042e{-}07$, $-9.947e-08$, $-9.763e-08$ db/Hz. Using a window length of 200,000 provides very accurate estimation of the mean. This holds true for all window sizes greater than 120,000. Any window size smaller than 120,000 no longer accurately captures the mean. All window lengths are able to capture the variance accurately, this is due to the fact that the data produced by the model is stationary.

Wilcoxon Rank Sums Test for different window lengths: Last, we compare power spectral estimates from post-simulation with estimates from *in situ* analysis by using a Wilcox on Rank Sums test (WEST) and relate the test values achieved with the memory usage for each window length. Specifically, the two inputs to our WEST are: (1) the PSD that results from post-simulation processing and (2) the PSD computed *in situ* using the specified window length. The null hypothesis is that the post-simulation and the *in situ* estimates are statistically identical populations. We compute the test using 95% confidence which results in an α of 0.05. If the results of our WRST are greater than our α, then we fail to find any evidence that the two signals are not from the same statistical distribution and thus conclude that the *in situ* results can be used for substantive scientific observations. Table 4.1 shows results of our WRST

together with the associated memory usage for window lengths ranging from 60,000 to 300,000. In the table we observe that we need to increase the window length to increase accuracy in our *in situ* results. However, accuracy is not a "free" comodity because neurobiological simulations are memory bound. While providing the simulation with as much accuracy as possible is desirable, by arbitrarily increasing the window length we introduce the potential of negatively impacting simulation performance by using more memory than is available. Its vital to find the minimum window length that provides an accurate representation of simulation behavior at runtime but does not cause memory spill. Our WRST reinforces our observations from the qualitative comparison of the PSD of the field potentials. Window lengths under 200,000 timesteps fail to exceed our expected α and thus do not accurately represent the correct behavior of the model. At 200,000 timesteps, we receive a p-value of .06 exceeding our α of .05, and thus we fail to find evidence that the *in situ* results are different than the post-simulation analysis results and conclude that the estimates are usable for making substantive observations about the model behavior. Increasing the window's length further, increases our p-value but at the cost of increasing the memory requirement of storing the varying field potential data in our memory resident staging area. As we showed in 4.1.2, memory is the primary contended resource when combining simulations and anlayses of neuronal network models. Thus, we determine that 200,000 timestep window provides sufficient accuracy for analysis while minimize memory usage.

We have demonstrated through both qualitative and quantitative methods that using a window length of 200,000 timesteps is the minimum length window that provides sufficient accuracy for our *in situ* analyses while also minimizing memory usage. Qualitatively, a 200,000 timestep window provides spectral estimates that retain important features in the frequency content while greatly reducing fluctuations. Additionally, we showed that using a window of length 200,000 timesteps results in spectral estimates that are statistically indistinguishable from estimates obtained post-simulation. By choosing the minimal window length that provides statistically comparable estimates we are able to reduce the memory footprint of the analysis. Through our ability to

Wilcoxon Rank Sums Test		
Window Size (timesteps)	P-value	Memory Per Process (MB)
60,000	<0.01	48
80,000	<0.01	64
100,000	<0.01	80
120,000	<0.01	96
140,000	<0.01	112
160,000	<0.01	128
180,000	<0.01	144
200,000	0.06	160
220,000	0.06	176
240,000	0.27	192
260,000	0.51	208
280,000	0.91	224
300,000	0.95	240

Table 4.1: Wilcoxon Rank Sums Test (WRST) and memory usage results across window lengths ranging from 60,000 to 300,000 steps.

locally collect meaningful information about the simulation we eliminate data movement while maintaining our capacity to make substantive scientific insights at execution time.

4.3 Discussion

Coupling analysis alongside computation will be imperative in mitigating the impending challenges to data-intensive workflows in future generation computing. In future generations of high-performance computing, it will no longer remain practical to transport data to the analysis. Instead, analysis will have to occur at the location the data is generated decreasing the impact of I/O bottlenecks to the end-to-end scientific analysis pipeline. There is a need for increased attention to the design and implementation of new analytical frameworks that allow scientist to perform pre-processing, analysis, and interactive data visualization with minimal data movement. In addition, there is a need to understand efficient ways in analyzing the diverse landscape of

HPC applications. There is no one-size-fit-all method, and designing optimal analysis workflows will require a deeper understanding of the applications themselves.

In this chapter, we looked to address the challenge of the increasing cost of the instantiation of the model (i.e., initialization cost). Initialization cost is unavoidable in the current iteration of our model. Because of this, we proposed the integration of *in situ* analysis. By integrating *in situ* we allow for more science to be done per execution of the model. We evaluate the accuracy of in-situ analysis results using three qualitative measures the Wilcoxon Rank Sums test statistic, Root Mean Squared error, and the Coefficient of Determination. We compare the resulting spectral estimates of both the *in-situ* approach with the post-simulation approach. We observe that we can reduce the amount of data needed to perform analysis, while also achieving qualitatively and quantitatively comparable scientific insights, by using smaller windows of data than our post-simulation approach. In this chapter, we also highlight the fact that increasing the window size for better accuracy results in an increase in memory. This shows that increasing the window size is not "free" and memory use must be of a concern. Increasing the amount of science that can be accomplished per execution by coupling analysis with simulation reduces the cost of initialization. However, it does not remove the cost. The need to re-initiate the model every time parameter modifications are needed is still a problem. In the next chapter we look to take things a step forward by integrating computational steering into our model.

Our work solely focused on evaluating analysis performed through the use of in-memory data staging and custom tailored analysis methods. We are aware that there are other methods for enabling runtime analysis of HPC applications. In this workm our goal was to understand how temporality of our analysis impacts the degree of science achievable *in situ*. Because of this, it was important that we build on the light-weight framework DataSpaces, which allows for a higher level of control over how data is managed. This level of control is not easily possible with other I/O frameworks. Additionally, in our implementation of *in situ* analysis, the analysis was not fully integrated into the course code of the monitored application. Instead our *in*

situ analysis ran concurrently with the application extracting simulation output and performing analysis solely in memory. The major benefit of having the analysis no reside in application source code is that our framework remains application agnostic and is easily modifiable to be used with an entirely different simulation framework.

Chapter 5

COMPUTATIONAL STEERING IN SIMULATED NEURAL NETWORKS

We leverage knowledge gleaned from *in situ* analysis to steer a neuronal network simulation. Specifically, we first implement then integrate a working prototype of a computational steering environment into our simulation environment supporting neuronal network models on GENESIS. Our computational steering environment allows for the dynamic modification of model attributes and parameters at runtime, enabling the ability to perform hypothesis-driven simulation. Having the ability to modify parameters at runtime transforms our scientific analysis workflow from one in which we are restricted to one experiment per execution, which was the case with the traditional workflow, to multiple experiments per execution. This results in our ability to study the highly complex internal structure of large-scale simulated neuronal networks models on GENESIS.

The rest of this chapter is organized as follows. Section 5.1 describes our implementation of a prototype to computationally steer a GENESIS simulation and the challenges associated with doing so. Section 5.2 demonstrates the steering capability of our CSE in two scenarios excitatory/inhibitory modulation and pulse stimulus injection. Section 5.3 provides a performance evaluation of our steering method. Section 5.4 concludes with a discussion for this chapter.

5.1 Steering Simulated Neural Networks with GENESIS

5.1.1 Necessity

Neurons in the brain communicate through point events called spikes [88]. A single neuron receives input from many different sources which influence firing activity.

Complex cognitive behavior is believed to be generated by the coordinated activity of large groups of neurons. For the brain to be able to form sophisticated signal patterns needed for complex behavior the balance between excitation and inhibition (E/I balance) activity must be maintained [89]. Inhibitory signals in the brain decrease the likelihood that a receiving neuron will fire. On the other hand, an excitatory signal in the brain makes a neuron more likely to fire. Inhibitory and excitatory signals are used in collaboration to keep excitation in the brain in check. It is postulated that real life neuronal networks operate in a sort of "equilibrium". Excitatory and inhibitory neurons manage correlated levels of activity [90]. Brain activity dominated by excitatory signals would only be capable of exciting itself in repeated bursts of activity (i.e. epileptic seizure). Brain activity dominated by inhibitory signals will only be capable of low activity not enough to facilitate synchronization among brain diverse areas [91]. For healthy brain activity, you need to reside in the middle which generates complex patterns of activity. Studying the balance of E/I is becoming more important due to the belief that it plays a big role in the complex patterns of the brain. Low E/I ratios are associated with Schizophrenia, whereas high E/I ratios are associated with Autism. Even greater excesses of E/I are associated with epileptic seizures [92].

Neuronal network models are known to have very complex dynamics and to switch fluidly between different biologically relevant functional states. The models presented in this paper have been tuned to produce electrical activity with mathematical properties that are associated with the local field potentials thought to underlie activity recorded using electroencephalogram. The local field potential is the distance weighted sum of simulated voltages in all neuronal compartments. LFP activity contributing to different frequency ranges in electroencephalogram are associated with task performance in behavioral experiments. Activity in the alpha range (8 to 13 Hz) is associated with background brain activity, while activity in the gamma range (40 to 100 Hz) has been associated with perceptual binding [93]. Computational neuroscientists explore potential neuronal mechanisms underlying functional behavior by

manipulating model parameters.

For this thesis, our computational steering focuses on enabling the investigation of network dynamics under different connectivity configurations, specifically we investigate the role of inhibitory and excitatory synaptic connections in the neocortex, through the manipulation of excitatory/inhibitory modulation. Additionally, we use knowledge extracted from *in situ* analysis with a window length of 200,000 timesteps to drive our neocortex model's oscillatory activity. Our initial setting targets two of the four major EEG frequency bands: alpha (8-13Hz) and beta (13-40Hz).

5.1.2 Components

Mulder et al., in a survey of computational steering environments [94], defined three components that constitute a computational steering environment for high-performance computing applications. Listed were a component for monitoring and interacting with the HPC application, a communication and data transfer layer for handling steering request and managing data, and the monitored application. We leverage these three components and integrate them into our steering-based computational environment which also consists of the user interface (UI), interaction manager (IM), and data manager (DM). We show an overview of all three in Figure 5.1, where black arrows show the flow of control in our CSE.

The first component of our CSE is the DM. The DM leverages our previous work in integrating *in situ* analysis in Chapter 4. We build our DM on top of DataSpaces, which facilitates distributed analysis and steering. DataSpaces provides a light-weight, scalable, as well as flexible shared abstraction to a memory-resident data staging area where in-memory storage space is allocated locally at each node. The DM provides the primary method for which users perform interactive analysis and data visualization on memory resident model output. Data in the model is handled by the data manager which is responsible for presenting data to the user. The DM enables users to index, label, and perform fast in-memory transformations on the data and other pre-processing.

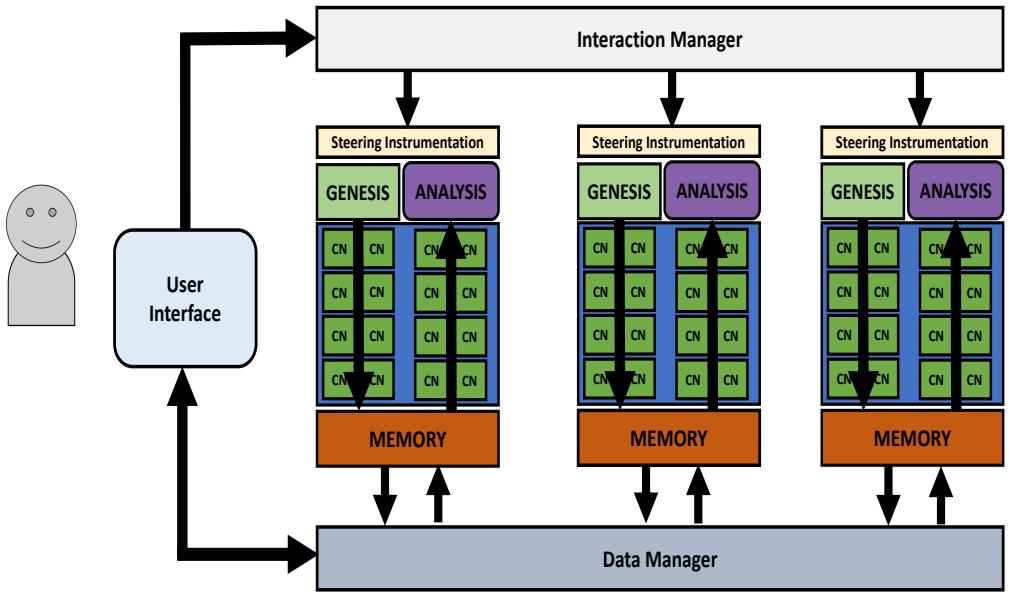


Figure 5.1: A adaptation of the Mulder et al. [94] definition of steering environment for our neural networks simulations.

The DM is responsible for extracting data from the application and placing it within the virtual dataspace.

The second component is our interaction manager. The interaction manager interacts with the steering instrumentation that resides in the script files that control the GENESIS execution flow. Our interaction manager is responsible for the modification and management of the neuronal network model script files that are used to steer the model. The interaction manager is also responsible for the modification of the environment variables that the steering instrumentation uses to know when to enact the steering changes. The interaction manager sends user steering actions to the steering instrumentation that resides alongside the simulation and analysis. Each of the three components are executed by separate processes, and as a consequence, the components do not all have to reside on the same node. This allows for the components to be

distributed.

The last component of our CSE, is a text-based user interface (TUI) written in Python. A user of our CSE asserts control over the environment through the use of our TUI . Our TUI allows a user to select modifiable application parameters and define their properties. Once one or more changes have been applied, a flag is set locally at the compute node. This flag is used by the steering instrumentation to know when new steering actions are pending and need to be applied to the monitored application. Steering actions are essentially the modification of parameters that are predefined by the CSE. Users navigate a hierarchical menu to select and modify parameters at run time. In addition to the modification of parameters, our TUI acts as the primary method for interacting with the data manager and interaction manager. The TUI is the intermediary between the DM and IM and allows them to communicate with one another.

The challenge of addressing the high volume and velocity of data produced by numerical simulations can be dealt with in many ways from a software engineering point of view. Using strongly typed compiled languages like C offer performance. However this comes at the cost of a loss of flexibility and extensibility of the code. More modern popular languages like Python, R, and MATLAB offer straight forward interfaces but do not offer the same level of performance as traditional HPC languages C and C++. For our method of interactive computational steering, we use a hybrid of both approaches. For tasks where performance is necessary, such as data ingestion of the model's output, we use C. For task where extensibility and flexibility are important, such as analysis of simulation output we use Python. We use Python for all analysis and steering functionality. Python provides excellent debugging, expressive syntax, and support for modularity. We integrate the two languages through the use of the Python/C API [95].

5.1.3 Implementation

For this thesis, we enable two kinds of steering. The first kind of steering is steering a pulse input square wave injection. The injection current pulses on and off for a duration of time injecting electrical current into the simulated neurons. The second kind of steering that we implement is the manipulation of synaptic weights which control the inhibitory and excitatory balance of the model. Traditionally, after network parameter configuration and simulated element creation, the neuronal network model executes uninterrupted until it reaches a completion time pre-set by the user, which is routinely on the order of several simulated seconds. The user is not able to direct execution flow by modifying parameters during this time. Conventionally, to modify network parameters, the user must stop and restart the simulation which we demonstrated in Chapter 3 is very costly.

To enable steering, we modify the primary script file of the neuronal network model, known as the Neocortex.g file. The Neocortex.g file is ingested by every process and is responsible for instantiating all simulated model elements either directly or indirectly. It controls network instantiation and drives model execution flow. We replace the non-interactive method of driving execution flow with our method which is based on a centralized control loop (i.e., main loop). By augmenting the Neocortex.g file we enable interactive event based steering. Our control loops detect events (i.e., user steering actions, flags set by *in situ* analysis) by actively sampling the state of global environment variables that are set by the user and automatically by the *in situ* analysis. We determine the frequency of environment variable polling, by controlling the length of the steering step. Based on the status of the environment variables the control loops drive execution flow of the simulation through the use of conditional expressions (i.e., if-else statements) that determine whether the parameters of simulated elements need updating or should remain unchanged. Our control loop can be as involved as we need, enabling the execution of dynamic simulation scenarios. Polling of events continues throughout model execution until the simulation reaches completion.

When a steering action is initiated the TUI creates a very small output text

file with a list of modified application parameters. An environment variable flag is set signaling to the control loop that a steering action is pending. Upon the beginning of the next steering step, the IM sets the specified parameters to their new values. However, these changes do not take effect until the IM calls on the model to call a routine that sets the value of field(s) in the data structure of a specified element. Once the routine has been called, changes to the parameters of simulated elements are no longer pending and take effect before the next simulation step. The IM then runs a check routine which verifies the consistency and well being of the simulated elements and reports any issues to the user. If needed, a reset routine will be initiated which returns the simulation to its original conditions. A barrier is administered to make sure that all processes complete their respective consistency checks successfully before the model can progress further in time. If there are no consistency issues reported then the simulation continues execution until it reaches the next steering step when then the process repeats itself until reaching the pre-defined maximum amount of simulated time.

5.1.4 Synaptic Connections

We now discuss how we enable steering of synaptic connections within our model. In addition to augmenting the Neocortex.g file to allow for event-based steering, we add the ability for the dynamic modification of the synaptic weights. During model instantiation, synaptic connections between neurons are statically created. The creation of connections between neurons is controlled by probabilities which are pre-defined likelihoods that one neuron type will establish connections to another neuron type. By increasing this probability a neuron of type A is more likely to connect to a neuron of type B and so on. Once the simulation begins there is no way to change the connectivity in the model. However, we are able to modify the weights and scales of the connections between neurons. Modifying a neuron's influence over other neurons through manipulation of synaptic weights, allows us to get around the restriction of the statically defined connections. By controlling the weights of the synaptic connections

we can essentially "turn off" connections between neurons by reducing their weight to zero. This is equivalent to removing a connection since a weight of zero does not influence on the firing activity of neighboring neurons it is connected to.

In this thesis, we focus on enabling the ability to dynamically govern the level of excitation (i.e., firing activity) within our model. To accomplish this we leverage existing model parameters which influence the synaptic weights of inhibitory and excitatory neurons. By enabling the ability to modify these model parameters at runtime we create the ability to define the minimum and maximum ranges of synaptic weights. We define the maximum and minimum weights on a neuron type by neuron type basis. Equation 5.1 shows how the maximum weights are calculated in our model for each neuron type. Each neuron type has a base weight which is represented by $Cell_{basewgt}$, this weight is modified by a multiplier which scales the weight with the addition of an offset. We avoid changing the base weights and focus solely on the weight multiplier and offset. Equation 5.2 demonstrates how the minimum weight of synaptic connections are defined.

$$Cell_{MAXwgt} = Cell_{basewgt} * weightMultiplier + offset \tag{5.1}$$

$$Cell_{MINwgt} = 0 + offset \tag{5.2}$$

Making the multiplier large will result in a very high ceiling for the synaptic weight and will increase its maximum potential influence on other neurons. Making the multiplier small results in a maximum weight with a low ceiling reducing its maximum potential influence on neighboring neurons. The minimum weight is controlled by an offset value which we can be increased to make the floor weight higher and prevent the weight from taking on to less of a value. When the user wants to make a modification to the synaptic weights of the model, the user submits a steering request with new weight multipliers. This request gets pushed to the IM which initiates the model to compute new ranges for the synaptic weights. Upon the completion of this, the IM

forces the model to re-instantiate the weights by calling a routine that instantiates the weight field of the synapses, setting the synapses maximum and minimum weights. The final step is the propagation of the new weights throughout the network. The final weight is computed by $final_{wgt} = wgt + (wgt * randomNumber)$. Synaptic weights are not allowed to go negative even if the user specifies a negative weight multiplier, instead the weight will be set to zero.

5.1.5 Current Injection

We now discuss how we enable steering of current injection within our model. Steering stimulus current injection is relatively more simple than steering the strength of connections. Just like the modification of the synaptic connections, we control the stimulus current within the central control loop located in the main input file to the model. Initially, upon creation of the network a pulse injection object is created. A pulse injection object accepts three arguments; these are the length, delay, and level of the current injection. The length corresponds to the how long the pulse injection occurs. The delay corresponds to how long the current is turned off between injections. Lastly, the level corresponds to the strength or amplitude of the current injection. The injection objects are then propagated to the simulated elements (i.e., neurons). When a user initiates a current injection steering action, the user specifies the length, amplitude, and delay of the current injection. This is reassigned to the existing current injections objects; a consistency check is made to ensure there are no issues with the reassigned values. If not the simulation proceeds and current will be injected at the conclusion of the specified delay time.

To steer the pulse stimulus injection we create multiple pre-defined stimulus configurations that are increasing in amplitude of their injection. We leverage our *in situ* analysis which monitors the level of activity in the model using spectral analysis enabling automated steering. If the intensity of activity is not at our desired level we automatically initiate a steering action which assigns a greater amplitude multiplied by some scalar to the stimulus current. The scalar acts as similar to how momentum

is used in gradient decent search problems. At the start of steering the scalar is very large and causes large large increases and decreases in the amplitude. However, with each successive steering action the size of the scalar is reduced thus reducing the effect of the change to the amplitude.

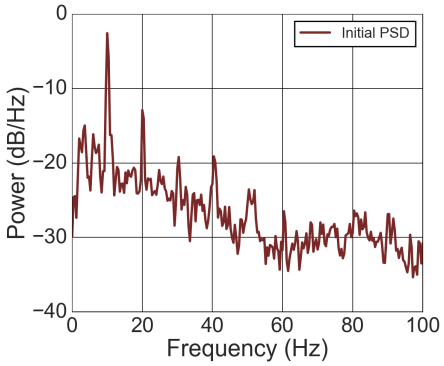
5.2 Examples of Successful Steering in GENESIS

5.2.1 Pulse Stimulus Current Injection

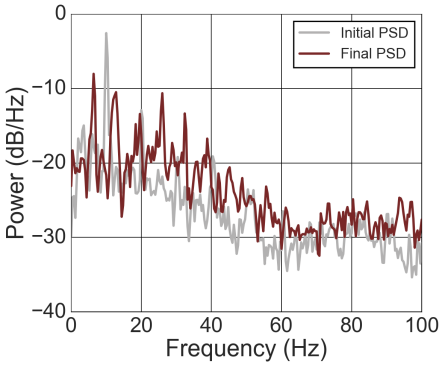
In the initial phase of our simulation, the power (or energy density) of our neocortex system is concentrated in the alpha range around 11Hz: 60% of the power in the simulation is coming from the model oscillating in the alpha's frequency range and 20% from the beta's frequency range (in Figure 5.4b). The other two bands (i.e., delta 0-4Hz and theta 4-8Hz) have less impact on the simulation's power (i.e., less than 20% combined) and thus are not represented in the figure. Under these circumstances, the intensity of the spectral estimates of our model exhibit a power peak that is concentrated in the alpha range around 11Hz (in Figure 5.4a).

Our workflow combining the GENESIS simulation and *in situ* analysis (in Figure 2.3) steers the power, initially concentrated in the alpha range, to move and concentrate it in the beta range. To this end, we set up our simulation into six epochs, with each epoch consisting of 200,000 timesteps (i.e., coinciding with the length of our analysis window). At each epoch, our *in situ* analysis monitors the relative contribution to the power (in Figure 5.4b) and injects a stimulus current by modifying the square wave pulse injection in our neuronal network model's input files. The stimulus' level is automatically controlled and adjusted based on the relative power.

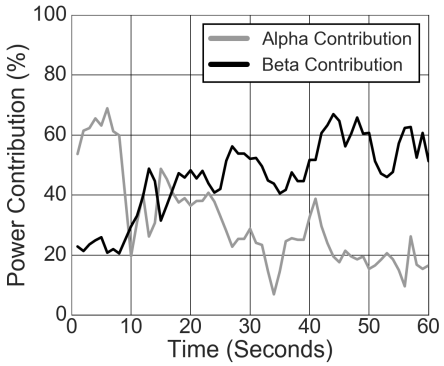
As we move across time epochs, our workflow initially injects a higher stimulus level (see Figure 5.4d). For example, at the second epoch, the intensity of the stimulus increase from 0.5 to 15. This increase impacts the contribution of power in the beta range that consequently increases to narrowly overtake the power contribution of the frequencies in the alpha range. By observing the substantial change in power contributions, our workflow reduces the current intensity rate of the stimulus level by 66%,



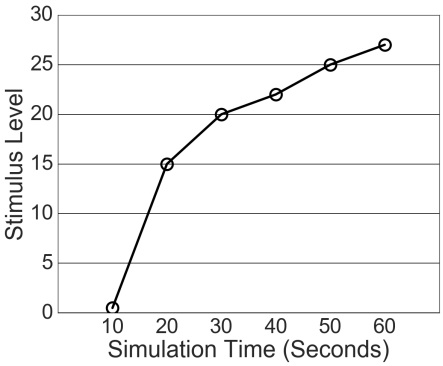
(a) Initial PSD



(c) Final PSD



(b) Relative power in alpha and beta bands



(d) Stimuli levels

Figure 5.2: Example of steering our neocortex simulation by automatically tuning the intensity of a stimulus current injection to increase the contribution to the power in 13-40Hz frequency range (i.e., figure 5.4b). Figure 5.4d shows the stimulus level across the six time epochs occurring every 10 seconds during the simulation. Figures 5.4a and 5.4c show the initial and final spectral power estimates of the simulation as a result of our tuning.

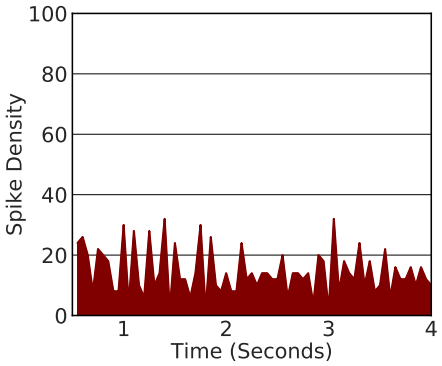
setting the intensity from 15 to 20. At the third epoch, the contribution of the beta range is approximately half of the power, while the alpha range is providing around a third of power. As we are approaching the end of the simulation, the intensity rate of the stimulus level diminishes as the alpha and beta contributions become flat around 20% and 60% of the relative power respectively, inverting the initial conditions of the neocortex system. In other words, the change in the relative power results in a shifting of our model from a lower frequency in the 10Hz range to a higher frequency in a range between 13-40Hz. Figure 5.4c shows the resulting spectral estimate at the end of the sixth epoch. We can see an increase in the power above 16Hz affirming that we

accomplished our task of driving the model's oscillatory activity, from predominantly existing in the alpha (power spectral density shown in gray) to predominantly existing in the beta wave range while running the simulation. In a traditional scenario based on post-simulation analysis, a scientist would have been required to run the simulation, learn about the initial relative power, stop the simulation to adjust the square wave pulse injection, and re-run the simulation. The unknown estimate of a suitable value for the square wave pulse injection would have required the manual stop and restart of one or more simulation, incurring into the heavy initialization costs of GENESIS simulations shown in Chapter 3. By effectively combining the simulation and *in situ* analysis in a single workflow, we can adjust the simulation power and identify the proper setting parameter for such an adjustment at runtime.

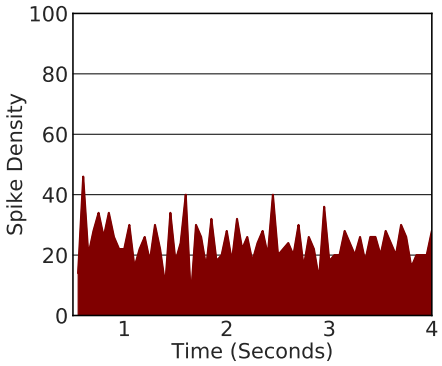
5.2.2 Excitatory/Inhibitory Modulation

The objective of this case study is to demonstrate our ability to steer the balance of excitation and inhibition in our neuronal network model. We demonstrate the success of this by showing a simple example in which we increase the excitation of the model while decreasing the inhibition. Our goal is to increase the activity in the model which we observe visually through the change in spike density and the time between spikes known as the inter-spike interval time. We compare a baseline model (no steering) to a model in which we increase the excitation.

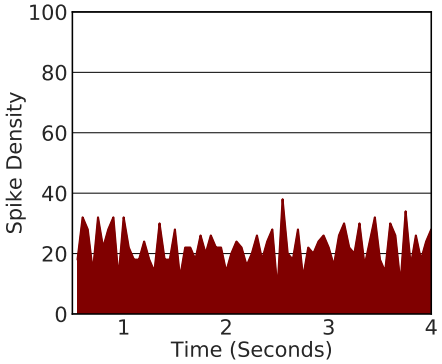
Figure 5.3, and Figure 5.4 shows the spike density and inter-spike interval times for the P23RS neuron respectively. The spike density is computed by counting the number of spikes that occur every 50 milliseconds. The mean inter-spike interval time is calculated by computing the difference between two successive spikes. We execute the model for 80,000 timesteps or 4 seconds of simulated brain activity. We observe that the spike density is pretty level with increases in the density of spikes occurring intermittently throughout the simulation. The mean spike density for the P23RS neurons for our baseline model are 14, 22, 24, and 25 respectively. This results in a mean of 21 spikes per 50 milliseconds with a standard deviation of 5 spikes. Additionally,



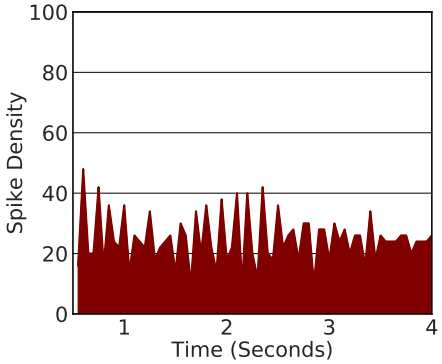
(a) Spike density of P23RSa cell.



(c) Spike density of P23RSc cell.



(b) Spike density of P23RSb cell.

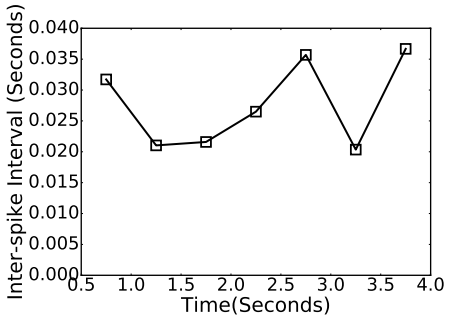


(d) Spike density of P23RSd cell.

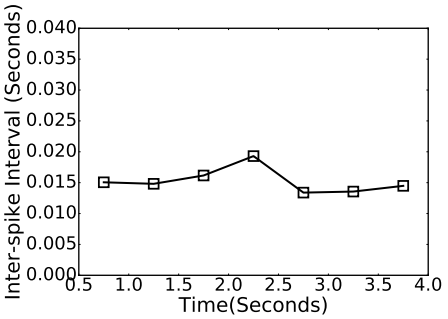
Figure 5.3: Spike density of P23RS neurons for a baseline model. Our baseline model represents the default behavior of the model when we do not apply any steering actions. The spike density remains relative constant during model execution with intermittent increases in spiking. For our baseline model we observe a mean spike density of of 21 spikes per 50 milliseconds.

we observe that the majority of the inter-spike interval times are relatively steady. We observe the average mean inter-spike interval time is .027, .017, .015, and .016. This results in the P23RS neuron having an average mean inter-spike interval time of .019 with a standard deviation of $5e-3$.

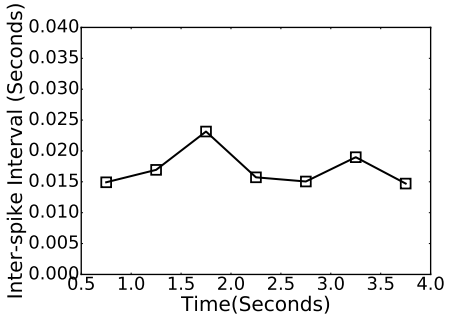
Figures 5.5 and 5.6 show the results of our steering when we attempt to increase the excitation in the model by manipulating the synaptic weights our model. At 40,000 timesteps we increase the scale at which excitatory neuronal activity impact neighboring neurons, and we decrease the scale at which inhibitory neuronal activity impact neighbors. This results in an overall increase in synaptic activity which we



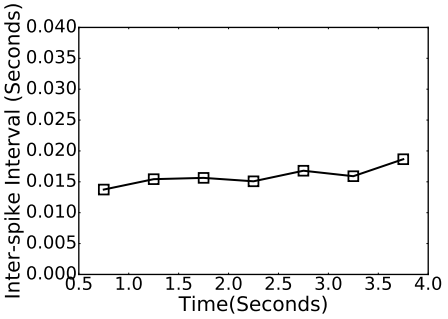
(a) Inter-spike interval of P23RSa cell.



(c) Inter-spike interval of P23RSc cell.



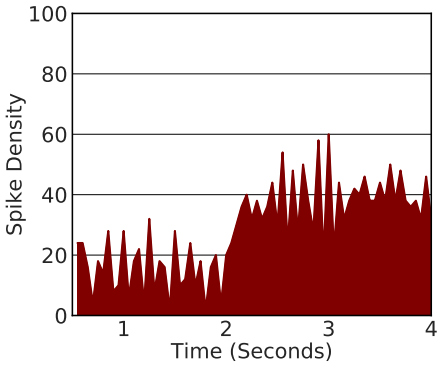
(b) Inter-spike interval of P23RSb cell.



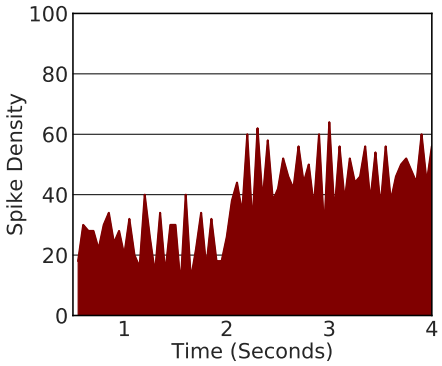
(d) Inter-spike interval of P23RSd cell.

Figure 5.4: Inter-spike interval time for of P23RS neurons for a baseline model. For our baseline model we observe a mean inter-spike interval time of 19 milliseconds.

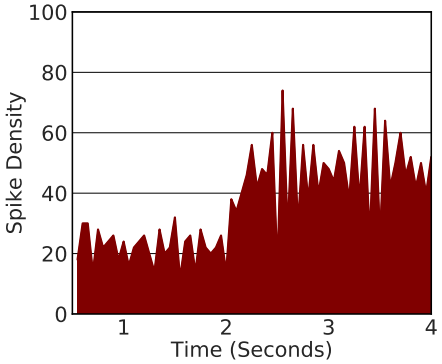
observe in the spike density. Each of our examples displays a relatively steep increase in the spike density around the two-second mark. This is where we modify the synaptic weights through our steering instrumentation. The mean spike density is 29, 36, 37, and 36 per 50 milliseconds respectively. Resulting in an overall mean spike density of 34 spikes per milliseconds, which is a 61% increase in activity. As a consequence of the increasing spike density, we observe a relatively steep reduction in the inter-spike interval time for the P23RS cell. The average mean inter-spike interval times for each of the P23RS subtypes are .018, .011, .011, .012 respectively, which results in an overall mean inter-spike interval time of .013, and a standard deviation of $3e-3$ a reduction of 46% in the mean inter-spike interval time. At the start of the simulation, we observe a mean inter-spike interval time of 19 milliseconds after our steering action of increasing the level of excitation in the model, the mean inter-spike interval time decreases to a mean of 8 milliseconds, a mean decrease of 10 milliseconds.



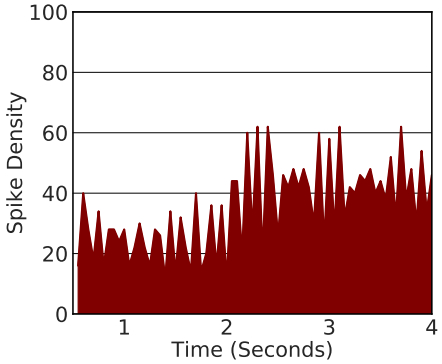
(a) Spike density of P23RSa cell.



(c) Spike density of P23RSc cell.



(b) Spike density of P23RSb cell.



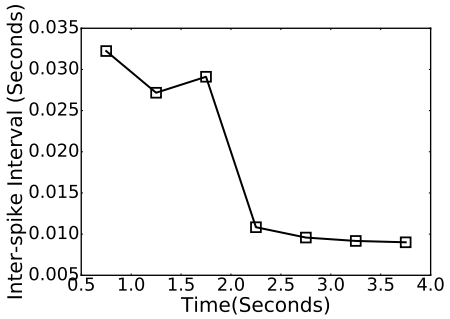
(d) Spike density of P23RSd cell.

Figure 5.5: Spike density of P23RS neurons for our steered model. At two seconds we see a sharp increase in the spike density going from 21 spikes per 50 milliseconds to 36 per 50 milliseconds.

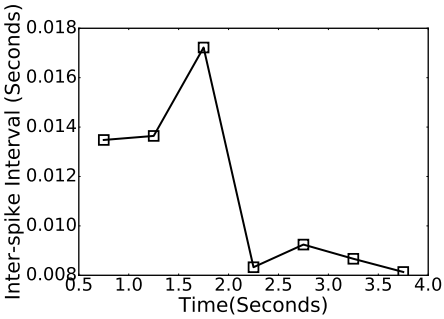
5.3 Performance Evaluation

In this section, we present a performance evaluation of our CSE. We look at two different measurements of performance, presentation latency and simulation perturbation. In 5.3.1, we evaluate presentation latency with respect to the user interacting with the CSE with the goal of monitoring the simulation. In 5.3.2 we focus on simulation perturbation measuring instrumentation overhead and how it affects overall execution time of the application.

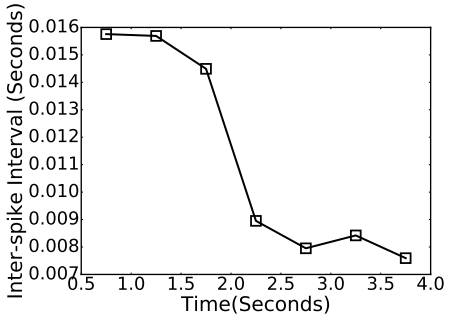
We perform our experiments on an SGI Ice X Thunder system at the Air Force Research Laboratory with a peak performance of 5.2 PFLOP. Thunder has 3,216 standard memory compute nodes each with 128 GB of memory and 36 cores clocked at



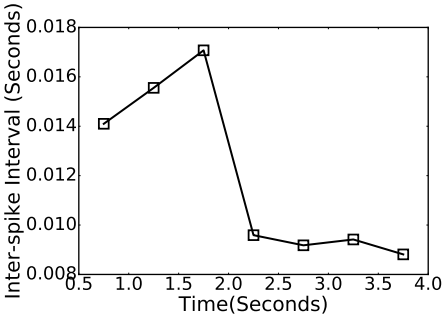
(a) Inter-spike interval of P23RSa cell.



(c) Inter-spike interval of P23RSc cell.



(b) Inter-spike interval of P23RSb cell.



(d) Inter-spike interval of P23RSd cell.

Figure 5.6: Inter-spike interval time for of P23RS neurons for our steered model. At two seconds into the simulation we see an abrupt decrease in the time between spikes. This coincides with the sharp increase in spike density shown in Figure 5.5

2.3 GHz. Thunder uses the Lustre file system to manage its Infiniband FDR 14X file system. In our *in situ* analysis experiments, the simulation is executed on 16 of the 36 cores of a node. We execute an analysis process for each simulation process resulting in the 16 of the remaining 20 cores being used for our analysis.

5.3.1 Latency

Presentation latency is the difference in time between and event occurring in the monitored application and the presentation of the event through some visualization to the user. If the latency of visualizations presented to the user is too high, the results shown will not be representative of current model behavior [96]. The user's interpretations and decisions will be inconsistent with what is actually occurring in the system [96]. The user will initiate steering interactions on model behavior that is no longer present. One of the primary tenets of *in situ* analysis is that analysis

should be executed sufficiently fast. Increasing the size of analysis window increases the accuracy results but at the cost of an increase in the latency between when behavior is occurring in the simulation and when it is presented to the user for interpretation. It also decreases the time resolution of the analysis which is important for the "human-in-the-loop" dynamic of computational steering.

With the selected window length of 200,000 timesteps for our runtime analysis, we measure the presentation latency for computing spectral analysis of the monitored application. We compare this latency with the primary method of performing analysis post-simulation. We observe the presentation latency as we increase the number of processes and the size of brain model (i.e., the neuronal network). Figure 5.7 shows the presentation latency when performing spectral analysis of the neuronal network as the size of the network increases from 16 to 4,096 MPI processes. As we increase the number of processes, the post-simulation analysis' presentation latency grows linearly: a larger number of processes have to read their data from the centralized file system sequentially. On the other hand, the time to perform the analysis and present the visualizations to the user via our CSE stays flat at under 14 seconds: the increasing-in-size analysis is distributed across the increasing number of nodes. The result is a large time saving in comparison to the traditional sequential analysis.

We also compare the presentation latency of analysis for a data window of 200,000 timesteps with the time needed to generate the data at runtime. Figure 5.8 shows the average execution time for performing both spectral analysis (visualization) and descriptive statistics along with the duration of the GENESIS simulation generating data over a window of 200,000 timesteps. The execution time of the spectral analysis and statistics on the window are 1.3 and 14 seconds respectively. These are dwarfed by the GENESIS' duration time which approaches 485,000 seconds. This demonstrates that our analysis approach meets one of the primary tenets of successful *in situ* analysis: its execution time is fast relative to the simulation's time by several orders of magnitude. Allowing for low latency presentation of model behavior to the user.

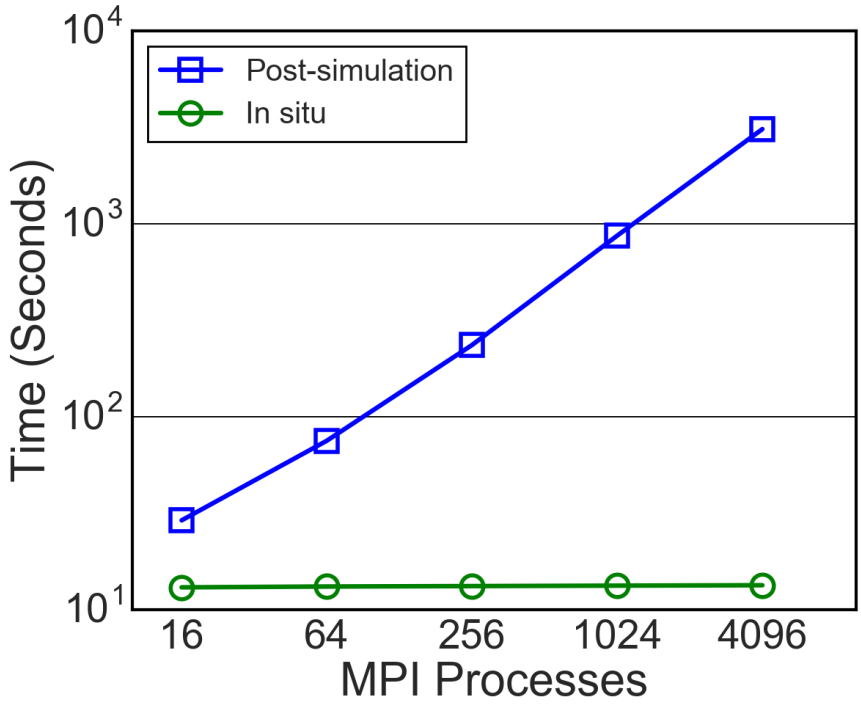


Figure 5.7: Presentation latency for performing spectral analysis in situ and post-simulation analysis time on simulation sizes 16 to 4,096 MPI processes. As we increase the size of the network the post-simulation analysis method greatly increases in size. At 4,096 MPI processes the post-simulation method needs approximately 50 min to complete. This is opposed to our *in situ* method which only needs 13 seconds to complete.

5.3.2 Simulation Perturbation

For this thesis we quantify simulation perturbation by measuring the observed difference in the execution time of our application with and without steering instrumentation.

Its important to investigate how the model performance is affected under the CSE when we scale to larger network sizes. Increasing network size increases the potential for inconsistencies in the computations and also the presentation of data to the user for interpretation. To mitigate the risks of introducing additional inconsistency in the model its imperative to make sure the components of the model and the CSE itself are properly synchronized through the used of barriers and other synchronization methods. These barriers increase the potential of additional overhead. Figure 5.9

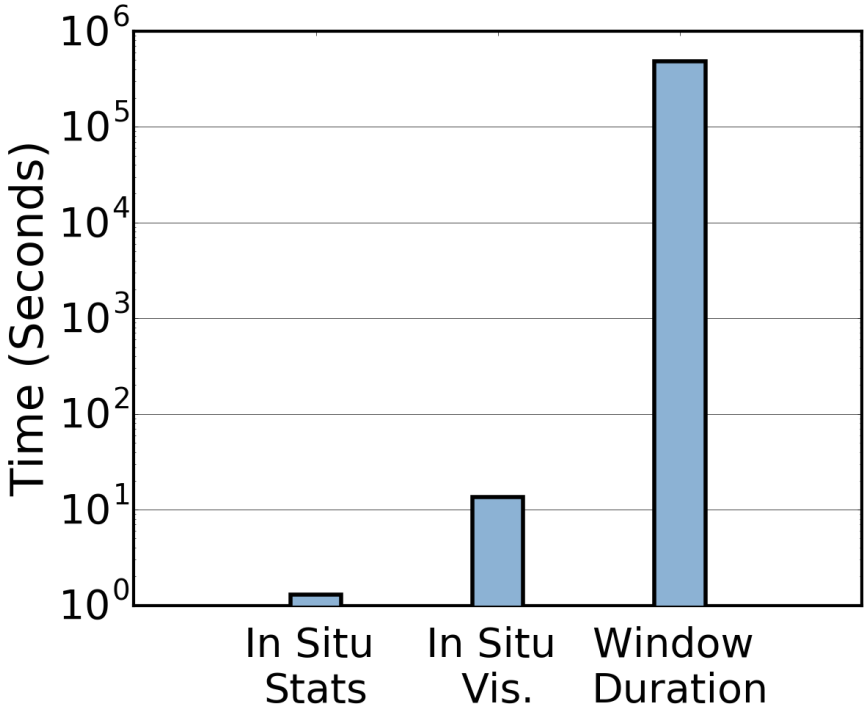


Figure 5.8: Comparison of our *in situ* analysis execution time in relation to the duration of GENESIS simulation generating a window of data over 200,000 timesteps when using 4,096 MPI processes.

shows the execution time of our model with and without steering instrumentation as we increase the network size from 16 processes to 256 processes. Over top of each bar group we place the percentage of overhead incurred due to instrumentation. At 16 processes on a single node we see approximately 2% overhead which amounts to the simulation running for an additional minute when instrumented for steering. As we increase the network size further we see an increase in overhead. This is a result of more neurons and synaptic connections being simulated. As the network is increased it is distributed across more nodes this leads to spike events having to travel further across the network to neurons that are not within the same region as the neuron that generated the spike. This leads to non-uniform simulation step times. Processes who complete a simulation step fast have to wait for slow processes to complete their step before moving to the next step.

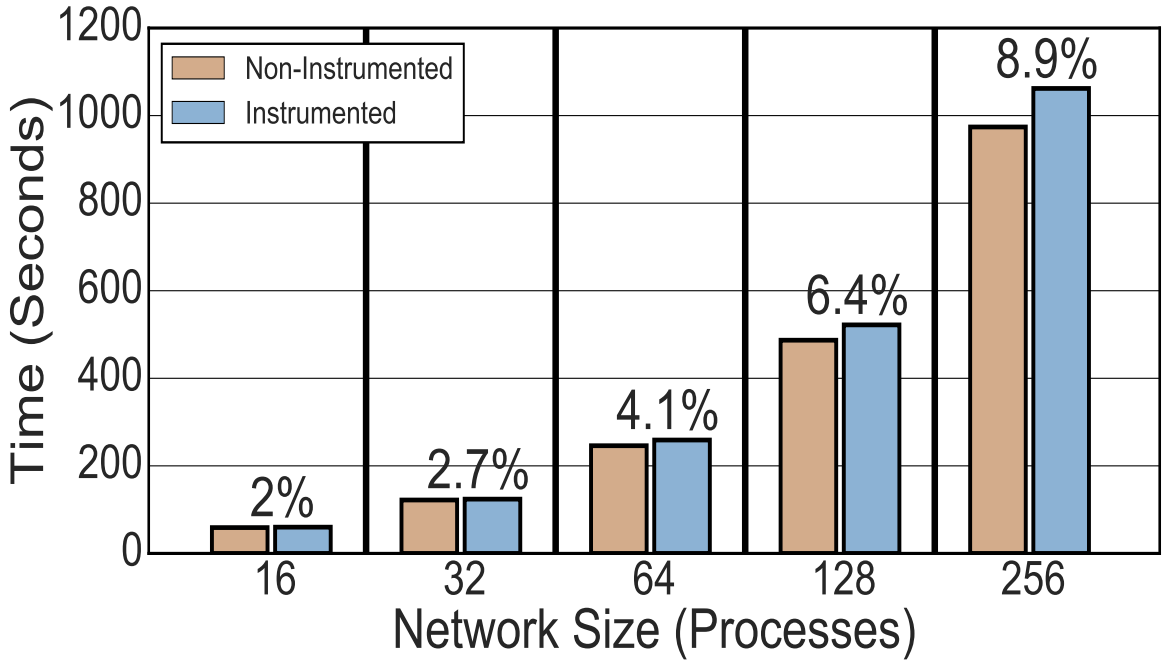


Figure 5.9: Execution overhead of steering instrumentation when increasing the number of neurons from 16 MPI processes on a single node to 256 MPI processes on 16 nodes. At 256 processes the steering instrumentation incurs approximately 9% overhead increasing the execution time of the simulation by 84 minutes.

5.4 Discussion

Simulation will continue to play a major role in the study of physical phenomena. At present, computational neuroscience simulations are performed on a trial-and-error basis. Computational models are simulated then stopped multiple times. With each start and stop the input parameters to the model are changed. Changing the overall behavior of the model in an iterative fashion until the desired model behavior is achieved. This start-stop method is extraordinarily costly and continues to be a primary bottleneck in the end-to-end scientific discovery pipeline. As the complexity of models increases so does the need for tools that enable rich interactive steering and analysis. There is a significant need for new methods that allow the user to participate in the simulation. In this thesis, we attempt to address this need by developing and integrating a working prototype of a computational steering environment for simulated neuronal networks on GENESIS.

In this chapter, we define and describe the major components of our CSE that allow for the steering of model behavior. We detail how these components operate together to steer both the synaptic weights and simulated external stimulus current that is injected into the model. We illustrate our ability to steer by showing two successful case studies of steering. In our first case study, we demonstrate our ability to steer the simulation by driving the model from the alpha wave range to the beta wave range. Enabling the ability to influence model behavior in this way allows scientist to better understand the spiking nature of neurons under external stimuli. The second case study demonstrated our ability to manipulate the level of excitation in the model through the steering of the synaptic weights. Through our efforts, we were able to increase the level of spiking activity in the model resulting in a decrease in the inter-spike interval time. Enabling the control of the excitation/inhibition balance in the model allows scientist to better understand how this balance plays a role in epilepsy and autism. Finally, we present a characterization of the performance of our CSE prototype. We show its ability to provide presentations of model behavior with reasonable latency. We also show the performance overhead incurred by instrumenting the model with the ability to steer. We demonstrate that we can achieve less than 9% overhead when scaling to 256 MPI processes.

Instrumenting our model with the ability to be steered introduced additional overhead. We believe that the additional overhead is a cost users are willing to pay if it means they have increased control and interaction with their simulations. It is still unknown of whether computational steering of simulated neuronal networks is practical at large scales. We envision our CSE being used in smaller use-cases such as debugging for performance and iterative behavior optimization.

Chapter 6

DISCUSSION AND FUTURE WORK

6.1 Summary

Advancements in simulation methods have enabled scientist to study physical phenomena at greater resolutions. Scientists are relying more on simulation to be one of their primary means of scientific discovery and knowledge acquisition. Neuroscientists have leveraged simulated neuronal networks to understand the fundamentals of information processing in the brain. Traditionally access to HPC hardware has been limited to large universities and governmental groups. However, Moore's Law has resulted in computing technology increasing in speed and at the same time decreasing in cost. The adherence of the computing industry to Moore's Law has resulted in small research groups the opportunity to procure small to medium-in-size clusters that traditionally they could not afford.

The democratization of HPC hardware has produced unprecedented heterogeneity in both hardware and its use. This hardware diversity poses a significant challenge to many domain-specific scientists, due to their lack of expertise in high-performance computing. Most researchers in the physical sciences do not have the necessary expertise to address high-performance computing bottlenecks and challenges. This lack of expertise results in scientists restricting their simulations to scaled down models that do not reflect the actual behavior of the natural processes they are modeled after. Most domain-specific applications are not written for HPC and do not scale well past a few nodes. Understanding the resource requirements and performance of models of increasing fidelity is extremely important if we are to create simulations that accurately portray their real-life counterparts.

We address this challenge in Chapter 3, where we analyzed the influence of HPC resources (i.e., single fat nodes and high-end clusters) on performance and data generation. Additionally, we focused on the science delivered by GEneral NEural SImulation System (GENESIS) for increasingly complex models of the brain's neocortex. Subsection 3.3.1 showed that there is a discrepancy between the diverse platforms used for neuronal network simulation. We observed that scientific throughput is greatly increased by migrating the model to HPC hardware from a network of workstations (NOW). Traditionally most domain-specific researchers perform their simulation on single fat nodes or NOW that have access to large quantities of memory and compute cores. This is because medium and large-scale clusters are not monetarily practical. Also, most researchers are still reluctant to the idea of remote distributed computing.

Furthermore, in subsection 3.3.2 we explored the effects of model complexity on scientific throughput and data generation. We observed that model instantiation is the primary area of concern in simulating neuronal networks, due to the sheer amount of connections that are generated and propagated by, and to, all neurons in the simulation during the creation of the network. Increasing the complexity of the models resulted in super-linear growth in the time needed to instantiate the network. This "cost" resulted in a decrease in scientific throughput. The cost of instantiating the model poses a major challenge in the simulation of larger networks. Long network instantiations make performing systematic parameter optimization on large networks impractical. More attention needs to be paid to the design of new data structure and algorithms that enable faster model instantiation.

We believe network instantiation will be exacerbated by the increase in concurrency promised by next-generation computing. This challenge will be combined with the increasing difference in performance between the compute and I/O subsystems. In next-generation simulation, it will no longer be practical to move huge amounts of data from the parallel file system of a large-scale HPC cluster to the dedicated resources of an analysis machine. Instead, analysis will have to occur at the location the data is generated. Resulting in a lessening of the impact of I/O bottlenecks to the end-to-end

scientific analysis pipeline. Increased attention must be paid to the design and implementation of new analysis frameworks. These new frameworks will allow scientist to perform pre-processing, analysis, and interactive data visualization with minimal to zero data movement.

In Chapter 4, we proposed the integration of *in situ* analysis as a solution for mitigating the impact of network instantiation to scientific throughput. By enabling monitoring of model behavior and analysis at runtime we allow more science to be accomplished per execution of the model. This "piggybacking" of more science per execution increase in scientific throughput. However, transforming an analysis workflow that is accomplished primarily post-simulation to one that is performed *in situ* is non-trivial. In moving to an *in situ* workflow, we give up our global view of data for one that is local in both time and space. This reduction in data scope increases the probability of inaccuracies in analysis results. To address this, we evaluated the accuracy of in-situ analysis results using three qualitative measures the Wilcoxon Rank Sums test statistic; Root Mean Squared error and the Coefficient of Determination. We compared the resulting spectral estimates of both the *in-situ* approach with the post-simulation approach. We observed that reducing the amount of data needed to perform analysis, does not eliminate our ability to achieve qualitatively and quantitatively comparable scientific insights.

We also study the impact of our analysis in regards to memory consumption. We conclude that increasing the window size is not "free", and attention must be paid to the memory consumption of analyses. For neuronal networks, memory is the primary resource of contention. It is important that analyses minimize memory use. Our work highlights the tradeoff between memory utilization and accuracy of the analyses. The choice of the level of accuracy should be made on a case by case basis. We are well aware that there will be scenarios in which accuracy will be sacrificed for low memory overhead and vice versa.

In Chapter 5 we enabled the ability to leverage *in situ* analysis results to steer our simulated neuronal network. To this end, we described, developed, and integrated

a working prototype of a computational steering environment for simulated models on GENESIS. We demonstrate results that show that our efforts bring us closer to achieving our goal of transforming the analysis workflow from that is opaque and trial-and-error based, to one that is transparent and hypothesis-driven.

We defined and describe the major components of our CSE that enables the steering of model behavior. We explain how our CSE components operate to steer two major factors in the network, synaptic weights, and simulated external stimulus current. We show our ability to steer the model by demonstrating two simple, successful case studies. These case studies are used to show in a straightforward way our ability to make changes at runtime to influence model behavior. In our first case study, we show our ability to drive model activity from the alpha wave range (which is the default frequency range of the model) to the beta wave range. We accomplish this by enabling the modification of a pulse stimulus current which we inject into the model. We show the success of our efforts by visualizing the spectral estimate prior to our steering and after our steering. In our second case study, we demonstrate our ability to steer by controlling the balance of excitation and inhibition through the modification of the influence of excitatory and inhibitory synaptic weights. Our goal in this case study was to demonstrate our ability to increase and decrease excitation in the model. We depict our success by showing the spike density and inter-spike interval time for a single neuron type in our model before and after steering. We show that through our steering efforts we can increase the spike rate of our model decreasing the inter-spike interval time. Finally, we present a performance characterization of the impact of the computational steering instrumentation to execution time. We observe approximately less than 10% overhead when we scale to 256 MPI processes.

6.2 Limitations and Opportunities

This thesis focused on the design of a CSE for GENESIS-based models. We focused solely on GENESIS, and this might place limitations on the overall impact of this work. However, GENESIS presents a great example of existing legacy applications. We

are strongly confident that our generalizations apply to most simulated spike-coupled neuronal networks and legacy applications executed on HPC. Additionally, our observations may not be reflective of what would be observed when integrating a CSE into newer simulation frameworks. The number of steerable parameters in our CSE is small. Our CSE only allows for two different types of steering. There are numerous other parameters that we would like to have the ability to steer. Furthermore, in Chapter 4 we only showed the impact of temporality on the accuracy of performing spectral analysis on our model. We showed that we were able to get both qualitatively and quantitatively comparable analysis results to those obtain post-simulation, but these findings may not translate to other types of analyses. More data-intense analyses might display higher sensitivity to temporality. Lastly, our study of how model complexity impacts performance only concerned itself with the growth in execution time. Though this is an important metric, a finer grain study of how complexity impacts compute and memory usage would make our study stronger. Understanding how design choice impacts resource utilization for both the GENESIS framework and the model would be very valuable.

The landscape of HPC hardware is becoming increasingly heterogeneous. Additionally, memory hierarchies are also increasing in complexity. New technology such as field-programmable gate arrays (FPGAs) and general purpose graphical processing units (GPGPUs) are becoming cheaper. As a consequence of the decreasing cost of accelerators, accelerators are growing in prevalence in high-end clusters and single fat node machines. The increasing penetration of accelerators has presented opportunities for the co-location of analysis on the accelerator alleviating the computational burden on the CPU. Users are requesting richer analysis pipelines in addition to faster throughput on their analysis making it imperative to perform analysis on the resources that yield the fastest results. Studies have shown how accelerators can speed up scientific simulation and analysis [97, 98, 99]. However, the co-location and coordination of analysis with scientific simulation is non-trivial, additional research is needed to address

these challenges. Methods for performing analysis described above may not be applicable to new hardware architectures and memory hierarchies. Future architectures will require new methods and workflows that are tuned to exploit the hardware.

Opportunities for future work consists of an analysis of how model complexity impacts resource utilization (i.e., memory and CPU). GENESIS is a CPU only code, understanding how accelerators (e.g, GPGPUs and Xeon Phi) can be leveraged for the simulation of neuronal network models is of high value. Additionally, we would like to understand to what extent does having only a local view of data impact the scope of analysis we can perform. Also, ideally we would like to integrate data extraction to in-memory staging, directly into GENESIS. This would avoid the need to write any data to disk. At the moment our implementation ingests data from disk into our staging area for analysis. We would also like for collaborative steering. Theoretically this is possible with our current prototype. However, it has not been confirmed. Currently analysis performed is "home-grown", future work will consist of integrating popular neuroinformatic packages and applications from the community to create a richer analysis and steering experience. Moreover, we would like to integrate finer grained steering behavior and study the impact this has on simulation performance. Lastly, we would like to integrate basic checkpoint-management functionality. Through this work, it has become clearer that this functionality is a large part of many scientific discovery methods.

6.3 Broader Impact

This thesis has sought to address the challenge of enabling hypothesis-driven simulation of neuronal network models through the employment of a working prototype of a computational steering environment. Next-generation computing will bring unprecedented concurrency. This degree of parallelism will heighten the need for tools that enable collaborative science to be accomplished on HPC. Also, it will increase the need for tools that allow for users to link several existing domain-specific applications ad-hoc to create dynamic workflows. To our knowledge, the works presented in this

thesis are the first of its kind in the design and implementation of a prototype to steer neuronal network models on GENESIS. Furthermore, in this thesis, we advance the ongoing discussion of how to efficiently progress the state of knowledge acquisition in high-performance computing. Our work presents a blueprint for the design of a computational steering environment for neuronal networks. We highlight and attempt to tackle key challenges in the integration of monitoring and steering.

BIBLIOGRAPHY

[1] Andrea Becchetti, Francesca Gullo, Giuseppe Bruno, Elena Dossi, Marzia Lecchi, and Enzo Wanke. Exact distinction of excitatory and inhibitory neurons in neural networks: a study with gfp-gad67 neurons optically and electrophysiologically recognized on multielectrode arrays. *Frontiers in neural circuits*, 6:63, 2012.

[2] Henry Markram, Eilif Muller, Srikanth Ramaswamy, Michael W Reimann, Marwan Abdellah, Carlos Aguado Sanchez, Anastasia Ailamaki, Lidia Alonso-Nanclares, Nicolas Antille, Selim Arsever, et al. Reconstruction and Simulation of Neocortical Microcircuitry. *Cell*, 163(2):456–492, 2015.

[3] A Paul Alivisatos, Miyoung Chun, George M Church, Ralph J Greenspan, Michael L Roukes, and Rafael Yuste. The Brain Activity Map Project and the Challenge of Functional Connectomics. *Neuron*, 74(6):970–974, 2012.

[4] Michael Beierlein, Jay R Gibson, and Barry W Connors. A network of electrically coupled interneurons drives synchronized inhibition in neocortex. *Nature neuroscience*, 3(9):904, 2000.

[5] Susanne Kunkel, Tobias C Potjans, Jochen Martin Eppler, Hans Ekkehard Ekkehard Plesser, Abigail Morrison, and Markus Diesmann. Meeting the memory challenges of brain-scale network simulation. *Frontiers in neuroinformatics*, 5:35, 2012.

[6] Jakob Jordan, Tammo Ippen, Moritz Helias, Itaru Kitayama, Mitsuhisa Sato, Jun Igarashi, Markus Diesmann, and Susanne Kunkel. Extremely scalable spiking neural network simulation code: from laptops to exascale computers. *Frontiers in neuroinformatics*, 12:2, 2018.

[7] Giovanni Aloisio and Sandro Fiore. Towards exascale distributed data management. *The International Journal of High Performance Computing Applications*, 23(4):398–400, 2009.

[8] G Aloisioa, S Fiorea, Ian Foster, and D Williams. Scientific big data analytics challenges at large scale. *Proceedings of Big Data and Extreme-scale Computing (BDEC)*, 2013.

[9] Salman Habib, Robert Roser, Richard Gerber, Katie Antypas, Katherine Riley, Tim Williams, Jack Wells, Tjerk Straatsma, A Almgren, J Amundson, et al. Ascr/hep exascale requirements review report. *arXiv preprint arXiv:1603.09303*, 2016.

[10] Tammo Ippen, Jochen M Eppler, Hans E Plesser, and Markus Diesmann. Constructing neuronal network models in massively parallel environments. *Frontiers in neuroinformatics*, 11:30, 2017.

[11] Nicholas T Carnevale and Michael L Hines. *The NEURON book*. Cambridge University Press, 2006.

[12] Marc De Kamps, Volker Baier, Johannes Drever, Melanie Dietz, Lorenz Mösenlechner, and Frank Van Der Velde. The state of miind. *Neural Networks*, 21(8):1164–1181, 2008.

[13] Moritz Helias, Susanne Kunkel, Gen Masumoto, Jun Igarashi, Jochen Martin Eppler, Shin Ishii, Tomoki Fukai, Abigail Morrison, and Markus Diesmann. Supercomputers ready for use as discovery machines for neuroscience. *Frontiers in neuroinformatics*, 6:26, 2012.

[14] S. Ahern, A. Shoshani, K. Ma, A. Choudhary, T. Critchlow, S. Klasky, V. Pascucci, J. Ahrens, W. Bethel, H. Childs, J. Huang, K. Joy, Q. Koziol, G. Lofstead, J. S. Meredith, K. Moreland, G. Ostrouchov, M. Papka, V. Vishwanath, M. Wolf, N. Wright, and K. Wu. Scientific discovery at the exascale: report from the doe ascr 2011 workshop on exascale data management, analysis, and visualization. *Dept. of Energy Office of Advanced Scientific Computing Research*, 2011.

[15] Esther Landhuis. Neuroscience: Big brain, big data, 2017.

[16] Mukeshwar Dhamala, Govindan Rangarajan, and Mingzhou Ding. Analyzing information flow in brain networks with nonparametric granger causality. *Neuroimage*, 41(2):354–362, 2008.

[17] Zhenyu Zhou, Yonghong Chen, Mingzhou Ding, Paul Wright, Zuhong Lu, and Yijun Liu. Analyzing brain networks with pca and conditional granger causality. *Human brain mapping*, 30(7):2197–2206, 2009.

[18] Sean L Simpson, F DuBois Bowman, and Paul J Laurienti. Analyzing complex functional brain networks: fusing statistics and network science to understand the brain. *Statistics surveys*, 7:1, 2013.

[19] Lars Kegel, Martin Hahmann, and Wolfgang Lehner. Generating what-if scenarios for time series data. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, page 3. ACM, 2017.

[20] D. A. Reed and J. Dongarra. Exascale computing and big data. *Communications of the ACM*, 58(7):56–68, 2015.

[21] C. Docan, M. Parashar, and S. Klasky. Dataspaces: an interaction and coordination framework for coupled simulation workflows. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 25–36. ACM, 2010.

[22] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[23] Valentino Braitenberg. Cell assemblies in the cerebral cortex. In *Theoretical approaches to complex systems*, pages 171–188. Springer, 1978.

[24] Timo Kirschstein and Rüdiger Köhling. What is the source of the eeg? *Clinical EEG and neuroscience*, 40(3):146–149, 2009.

[25] George W Fenton. The eeg, epilepsy and psychiatry. *What is epilepsy*, pages 139–161, 1986.

[26] Ernst Niedermeyer and FH Lopes da Silva. *Electroencephalography: basic principles, clinical applications, and related fields*. Lippincott Williams & Wilkins, 2005.

[27] Rodolfo Llinas and Urs Ribary. Coherent 40-hz oscillation characterizes dream state in humans. *Proceedings of the National Academy of Sciences*, 90(5):2078–2081, 1993.

[28] Pari Jahankhani, Kenneth Revett, and Vassilis Kodogiannis. Detecting clinically relevant eeg anomalies using discrete wavelet transforms. In *Proceedings of the 5th WSEAS International Conference on Wavelet Analysis and Multirate Systems*, pages 8–11. World Scientific and Engineering Academy and Society (WSEAS), 2005.

[29] S. Herculano-Houzel. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in human neuroscience*, 3, 2009.

[30] Vernon B Mountcastle. The columnar organization of the neocortex. *Brain: a journal of neurology*, 120(4):701–722, 1997.

[31] Charles R Noback, Norman L Strominger, Robert J Demarest, and David A Ruggiero. *The human nervous system: structure and function*. Number 744. Springer Science & Business Media, 2005.

[32] Steven L Bressler and Vinod Menon. Large-scale brain networks in cognition: emerging methods and principles. *Trends in cognitive sciences*, 14(6):277–290, 2010.

[33] Mikail Rubinov and Olaf Sporns. Complex network measures of brain connectivity: uses and interpretations. *Neuroimage*, 52(3):1059–1069, 2010.

[34] Larry W Swanson and Mihail Bota. Foundational model of structural connectivity in the nervous system with a schema for wiring diagrams, connectome, and basic plan architecture. *Proceedings of the National Academy of Sciences*, 107(48):20610–20617, 2010.

[35] Hans E Plesser, Jochen M Eppler, Abigail Morrison, Markus Diesmann, and Marc-Oliver Gewaltig. Efficient Parallel Simulation of Large-scale Neuronal Networks on Clusters of Multiprocessor Computers. In *Proceedigns of the Euro-Par 2007 Parallel Processing Conference*, pages 672–681. 2007.

[36] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[37] Yiran Chen and Qing Wu. Neuromorphic Computing: A SoC Scaling Path for the Next dececades. In *Proceedings 2012 IEEE International SOC Conference (SOCC)*, pages 290–291, 2012.

[38] S. Hill and H. Markram. The Blue Brain Project. In *Proceedings of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2008.

[39] Brain Research through Advancing Innovative Neurotechnologies (BRAIN) - National Institutes of Health (NIH). http://www.braininitiative.nih.gov/. Accessed: April 28, 2016.

[40] Aleena Garner and Mark Mayford. New Approaches to Neural Circuits in Behavior. *Learning & Memory*, 19(9):385–390, 2012.

[41] Iain Hepburn, Weiliang Chen, Stefan Wils, and Erik De Schutter. Steps: efficient simulation of stochastic reaction–diffusion models in realistic morphologies. *BMC systems biology*, 6(1):36, 2012.

[42] Petra Ritter, Michael Schirner, Anthony R McIntosh, and Viktor K Jirsa. The virtual brain integrates computational modeling and multimodal neuroimaging. *Brain connectivity*, 3(2):121–145, 2013.

[43] Gully APC Burns and Malcolm P Young. Analysis of the connectional organization of neural systems associated with the hippocampus in rats. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 355(1393):55–70, 2000.

[44] Jack W Scannell, Colin Blakemore, and Malcolm P Young. Analysis of connectivity in the cat cerebral cortex. *Journal of Neuroscience*, 15(2):1463–1483, 1995.

[45] Daniel J Felleman and David C Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex (New York, NY: 1991)*, 1(1):1–47, 1991.

[46] Frank H Eeckman. *Computation in neurons and neural systems*. Springer Science & Business Media, 2012.

[47] Olaf Sporns, Dante R Chialvo, Marcus Kaiser, and Claus C Hilgetag. Organization, development and function of complex brain networks. *Trends in cognitive sciences*, 8(9):418–425, 2004.

[48] J. M. Bower and D. Beeman. *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System*. Springer-Verlag, 2 edition, 1998.

[49] Nigel H Goddard and Greg Hood. Parallel Genesis for Large-scale Modeling. In *Computational Neuroscience*, pages 911–917. Springer, 1997.

[50] Research Publications Using GENESIS. http://www.genesis-sim.org/pubs.html. Accessed: April 28, 2016.

[51] James M Bower and David Beeman. Exploring realistic neural models with the general neural simulation system. *The Book of GENESIS, New York*, 1998.

[52] James M Bower. *Computational Neuroscience: Trends in Research 2000*. Elsevier, 2000.

[53] David Beeman. Genesis modeling tutorial. *Brains, Minds, and Media*, 1:1–44, 2005.

[54] Michele Migliore, C Cannia, William W Lytton, Henry Markram, and Michael L Hines. Parallel Network Simulations With NEURON. *Journal of Computational Neuroscience*, 21(2):119–129, 2006.

[55] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris Jr, et al. Simulation of Networks of Spiking Neurons: a Review of Tools and Strategies. *Journal of Computational Neuroscience*, 23(3):349–398, 2007.

[56] Michael Hines, Sameer Kumar, and Felix Schürmann. Comparison of Neuronal Spike Exchange Methods On a Blue Gene/P Supercomputer. *Front. Comput. Neurosci*, 5(49):10–3389, 2011.

[57] J. Kress, R. M. Churchill, S. Klasky, M. Kim, H. Childs, and D. Pugmire. Preparing for in situ processing on upcoming leading-edge supercomputers. *Supercomputing Frontiers and Innovations*, 3(4):49–65, 2016.

[58] Teng Wang, Sarp Oral, Yandong Wang, Brad Settlemyer, Scott Atchley, and Weikuan Yu. Burstmem: A high-performance burst buffer system for scientific applications. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 71–79. IEEE, 2014.

[59] Matthew Wolf, Fang Zheng, Scott Klasky, Karsten Schwan, Ron A Oldfield, Gerald Fredrick Lofstead, Qing Liu, and Todd Kordenbrock. Managing variability in the io performance of petascale storage systems. Technical report, Sandia National Laboratories, 2010.

[60] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi. Enabling in-situ execution of coupled scientific workflow on multi-core platform. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 1352–1363. IEEE, 2012.

[61] Q. Sun, M. Romanus, T. Jin, H. Yu, P. Bremer, S. Petruzza, S. Klasky, and M. Parashar. In-staging data placement for asynchronous coupling of task-based scientific workflows. In *Extreme Scale Programming Models and Middlewar (ESPM2), International Workshop on*, pages 2–9. IEEE, 2016.

[62] J. C. Bennett, H. Abbasi, P. T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, P. Pebay, D. Thompson, H. Yu, F. Zhang, and J. Chen. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–9, Nov 2012.

[63] Fang Zheng, Hasan Abbasi, Ciprian Docan, Jay Lofstead, Qing Liu, Scott Klasky, Manish Parashar, Norbert Podhorszki, Karsten Schwan, and Matthew Wolf. Predata–preparatory data analytics on peta-scale machines. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, 2010.

[64] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng. Datastager: scalable data staging services for petascale applications. *Cluster Computing*, 13(3):277–290, 2010.

[65] C. Docan, M. Parashar, J. Cummings, and S. Klasky. Moving the code to the data-dynamic code deployment using activespaces. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 758–769. IEEE, 2011.

[66] C. Docan, M. Parashar, and S. Klasky. High speed asynchronous data transfers on the cray xt3. In *Cray User Group Conference*, volume 5, page 2007, 2007.

[67] H. M. Monti, A. R. Butt, and S. S. Vazhkudai. Timely offloading of result-data in hpc centers. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 124–133. ACM, 2008.

[68] Bruce H McCormick. Visualization in scientific computing. *ACM SIGBIO Newsletter*, 10(1):15–21, 1988.

[69] Frederick P Brooks. Grasping reality through illusioninteractive graphics serving science. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–11. ACM, 1988.

[70] Bodhisattwa C Mukherjee and Karsten Schwan. Improving performance by use of adaptive objects: Experimentation with a configurable multiprocessor thread package. In *High Performance Distributed Computing, 1993., Proceedings the 2nd International Symposium on*, pages 59–66. IEEE, 1993.

[71] Thomas E Bihari and Karsten Schwan. Dynamic adaptation of real-time software. *ACM Transactions on Computer Systems (TOCS)*, 9(2):143–174, 1991.

[72] Petra Wenisch, Oliver Wenisch, and Ernst Rank. Harnessing high-performance computers for computational steering. In *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*, pages 536–543. Springer, 2005.

[73] Martin Ruess, Ralf-Peter Mundani, and Ernst Rank. Computational steering in dynamic structure simulation: An improved communication concept. In *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference*, pages 497–502. IEEE, 2009.

[74] RLISS Haber, B Bliss, D Jablonowski, and C Jog. A distributed environment for run-time visualization and application steering in computational mechanics. *Computing Systems in Engineering*, 3(1-4):501–515, 1992.

[75] Steven G Parker and Christopher R Johnson. Scirun: a scientific programming environment for computational steering. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, page 52. ACM, 1995.

[76] Jeffrey Scott Vetter and Karsten Schwan. Progress: A toolkit for interactive program steering. Technical report, Georgia Institute of Technology, 1995.

[77] Jeffrey Vetter and Karsten Schwan. High performance computational steering of physical simulations. In *Parallel Processing Symposium, 1997. Proceedings., 11th International*, pages 128–132. IEEE, 1997.

[78] GA Geist, James Arthur Kohl, and Philip M Papadopoulos. Cumulvs: Providing fault toler. ance, visualization, and steer ing of parallel applications. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3):224–235, 1997.

[79] H Lin and J Wooley. Computational modeling and simulation as enablers for biological discovery. *Catalyzing inquiry at the interface of computing and biology*, page 468, 2005.

[80] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[81] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the challenges of scientific workflows. *Computer*, 40(12), 2007.

[82] Eileen Kraemer, Delbert Hart, and G-C Roman. Balancing consistency and lag in transaction-based computational steering. In *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on*, volume 7, pages 137–146. IEEE, 1998.

[83] Xin Wang, Linpeng Huang, Xiaohui Xu, Yi Zhang, and Jun-Qing Chen. A solution for data inconsistency in data integration. *J. Inf. Sci. Eng.*, 27(2):681–695, 2011.

[84] Jan H Lui, David V Hansen, and Arnold R Kriegstein. Development and Evolution of the Human Neocortex. *Cell*, 146(1):18–36, 2011.

[85] James M Krueger, David M Rector, Sandip Roy, Hans PA Van Dongen, Gregory Belenky, and Jaak Panksepp. Sleep as a fundamental property of neuronal assemblies. *Nature Reviews Neuroscience*, 9(12):910, 2008.

[86] P. Welch. The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. *IEEE Transactions on audio and electroacoustics*, 15(2):70–73, 1967.

[87] W. Ting, Y. Guo-zheng, Y. Bang-hua, and S. Hong. Eeg feature extraction based on wavelet packet decomposition for brain computer interface. *Measurement*, 41(6):618–625, 2008.

[88] Fu Ming Zhou and JOHN J Hablitz. Layer i neurons of rat neocortex. i. action potential and repetitive firing properties. *Journal of Neurophysiology*, 76(2):651–667, 1996.

[89] Ofer Yizhar, Lief E Fenno, Matthias Prigge, Franziska Schneider, Thomas J Davidson, Daniel J OShea, Vikaas S Sohal, Inbal Goshen, Joel Finkelstein, Jeanne T Paz, et al. Neocortical excitation/inhibition balance in information processing and social dysfunction. *Nature*, 477(7363):171, 2011.

[90] Nima Dehghani, Adrien Peyrache, Bartosz Telenczuk, Michel Le Van Quyen, Eric Halgren, Sydney S Cash, Nicholas G Hatsopoulos, and Alain Destexhe. Dynamic balance of excitation and inhibition in human and monkey neocortex. *Scientific reports*, 6:23176, 2016.

[91] Jean-Marc Fritschy. Epilepsy, e/i balance and gabaa receptor plasticity. *Frontiers in molecular neuroscience*, 1:5, 2008.

[92] R Gao and P Penzes. Common mechanisms of excitatory and inhibitory imbalance in schizophrenia and autism spectrum disorders. *Current molecular medicine*, 15(2):146–167, 2015.

[93] Rufin VanRullen, Leila Reddy, and Christof Koch. Visual Search and Dual Tasks Reveal two Distinct Attentional Resources. *Journal of Cognitive Neuroscience*, 16(1):4–14, 2004.

[94] Jurriaan D Mulder, Jarke J Van Wijk, and Robert Van Liere. A survey of computational steering environments. *Future generation computer systems*, 15(1):119–129, 1999.

[95] E. Jones, T. Oliphant, and P. Peterson. SciPy: Open source scientific tools for Python. =http://www.scipy.org/, 2001–. [Online; accessed ¡today¿].

[96] Zhicheng Liu and Jeffrey Heer. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics*, 20(12):2122–2131, 2014.

[97] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, and Kevin Skadron. A performance study of general-purpose applications on graphics processors using cuda. *Journal of parallel and distributed computing*, 68(10):1370–1380, 2008.

[98] Simon J Pennycook, Simon D Hammond, Stephen A Jarvis, and Gihan R Mudalige. Performance analysis of a hybrid mpi/cuda implementation of the naslu benchmark. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):23–29, 2011.

[99] Rajesh Bordawekar, Uday Bondhugula, and Ravi Rao. Can cpus match gpus on performance with productivity?: experiences with optimizing a flop-intensive application on cpus and gpu. Technical report, IBM Research Report, RC25033, 2010.

[100] DL Boothe, AB Yu, P Kudela, JM Vettel, WS Anderson, and Franaszczuk PJ. Impact of Blast-dependent Cellular Damage on the Local Field Potential (LFP) in a Large Scale Simulation of the Cortex. *Society for Neuroscience*, 2015.

[101] Roger D Traub, Eberhard H Buhl, Tengis Gloveli, and Miles A Whittington. Fast Rhythmic Bursting Can Be Induced in Layer 2/3 Cortical Neurons By Enhancing Persistent Na+ Conductance or By Blocking BK Channels. *Journal of Neurophysiology*, 89(2):909–921, 2003.

[102] Roger D Traub, Diego Contreras, Mark O Cunningham, Hilary Murray, Fiona EN LeBeau, Anita Roopun, Andrea Bibbig, W Bryan Wilent, Michael J Higley, and Miles A Whittington. Single-column Thalamocortical Network Model Exhibiting Gamma Oscillations, Sleep Spindles, and Epileptogenic Bursts. *Journal of Neurophysiology*, 93(4):2194–2232, 2005.

[103] Danish Shehzad and Zeki Bozkus. Optimizing NEURON Brain Simulator With Remote Memory Access on Distributed Memory Systems. In *Proceedigns of the 2015 IEEE International Conference on Emerging Technologies (ICET)*, pages 1–5, 2015.

[104] D. L. Boothe, A. B. Yu, P. Kudela, W. S. Anderson, J. M. Vettel, and P. J. Franaszczuk. Impact of neuronal membrane damage on the local field potential in a large-scale simulation of cerebral cortex. *Frontiers in Neurology*, 8, 2017.

[105] M. Kemal Kiymik, Inan Gler, Alper Dizibyk, and Mehmet Akin. Comparison of stft and wavelet transform methods in determining epileptic seizure activity in eeg signals for real-time application. *Comp. in Bio. and Med.*, 35:603–616, 2005.

[106] M. Soleymani, S. Asghari-Esfeden, Y. Fu, and M. Pantic. Analysis of eeg signals and facial expressions for continuous emotion detection. *IEEE Transactions on Affective Computing*, 7(1):17–28, 2016.

[107] H. Behnam, A. Sheikhani, M. R. Mohammadi, M. Noroozian, and P. Golabi. Analyses of eeg background activity in autism disorders with fast fourier transform and short time fourier measure. In *2007 International Conference on Intelligent and Advanced Systems*, pages 1240–1244, Nov 2007.

[108] Y. Li, X. Wang, L. Luo, K. Li, X. Yang, and Q. Guo. Epileptic seizure classification of eegs using time-frequency analysis based multiscale radial basis functions. *IEEE Journal of Biomedical and Health Informatics*, 2017.

[109] S. McDaniel, D. L. Boothe, J. C. Crone, S. J. Park, D. R. Shires, A. B. Yu, and M. Taufer. Study of neocortex simulations with genesis on high performance computing resources. In *2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 924–931, Dec 2016.

[110] S. McDaniel, D. L. Boothe, A. B. Yu, and M. Taufer. Evaluating tradeoffs for in situ analysis in simulations of brain models on supercomputers. In Preperation.

[111] Michael N Shadlen and William T Newsome. Noise, neural codes and cortical organization. *Current opinion in neurobiology*, 4(4):569–579, 1994.

[112] Yousheng Shu, Andrea Hasenstaub, and David A McCormick. Turning on and off recurrent balanced cortical activity. *Nature*, 423(6937):288, 2003.

[113] Bilal Haider, Alvaro Duque, Andrea R Hasenstaub, and David A McCormick. Neocortical network activity in vivo is generated through a dynamic balance of excitation and inhibition. *Journal of Neuroscience*, 26(17):4535–4545, 2006.

[114] Carl Van Vreeswijk and Haim Sompolinsky. Chaos in neuronal networks with balanced excitatory and inhibitory activity. *Science*, 274(5293):1724–1726, 1996.

[115] Javier DeFelipe. The evolution of the brain, the human nature of cortical circuits, and intellectual creativity. *Frontiers in neuroanatomy*, 5:29, 2011.

[116] Ian Gorton, Paul Greenfield, Alex Szalay, and Roy Williams. Data-intensive computing in the 21st century. *Computer*, 41(4):30–32, 2008.

[117] Jung-Wei Chen and Jiajie Zhang. Comparing text-based and graphic user interfaces for novice and expert users. In *AMIA annual symposium proceedings*, volume 2007, page 125. American Medical Informatics Association, 2007.