Energy-Aware Distributed Simulation for Mobile Platforms: An Empirical Study

By

Fahad Maqbool

00000117243

Supervisor

Dr. Asad W. Malik

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of MSCS

In

School of Electrical Engineering and Computer Science,

National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(May 2018)

Approval

It is certified that the contents and form of the thesis entitled "**Energy-Aware Distributed Simulation for Mobile Platforms: An Empirical Study**" submitted by **Fahad Maqbool** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Asad W. Malik**

Signature: _____

Date: _____

Committee Member 1: **Dr. Mian Muhammad Hamayun**

Signature: _____

Date: _____

Committee Member 2: **Dr. Imran Mahmood**

Signature: _____

Date: _____

Committee Member 3: **Dr. Anis ur Rahman**

Signature: _____

Date: _____

Acceptance Certificate

Certified that final copy of MS/MPhil thesis written by Mr. **Fahad Maqbool**, Reg no. **00000117243**, of SEECS has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: _____

Name of Supervisor: _____

Date: _____

Signature (HOD): _____

Date: _____

Signature (Dean/Principal): _____

Date: _____

Abstract

The rapid increase of the computing power on embedded and handheld devices has made these devices attractive for many applications including simulation systems. There are a number of Parallel Discrete Event Simulation (PDES) frameworks that exists but most of these are designed for traditional cluster systems and are not suitable for battery operated devices where energy and power consumption are among the major concerns. A new PDES framework is thus required that takes into account the typical constraints of the mobile devices. However, before designing a new PDES framework that is specifically aimed for mobile devices, it is helpful to analyze the performance of existing frameworks. In this work, well-known Rensselaer's Optimistic Simulation System (ROSS) framework has been instrumented for a detailed analysis of its performance in terms of CPU usage, memory consumption, and energy and power requirements. This profiling helps in many ways. For example, one can select the most appropriate synchronizations algorithm for running the PDES frameworks on the mobile devices. Additionally, identification of resource intensive modules within the framework can be extremely useful in redesign/optimization of these frameworks while being ported to the heterogeneous environments. Based on these observations, a new simulation framework is proposed that is specifically designed for running on handheld

devices. The simulation framework, that is called SEECSSim[1], is the first one designed keeping in mind the characteristics and the constraints that are typical of mobile devices. SEECSSim includes the support for a number of state-of-the-art synchronization protocols and, thanks to its flexible design, the users can easily integrate any other simulation model/synchronization algorithm of their choice. The performance of SEECSSim has been studied using a well-known simulation model (i.e. PHOLD) for different synchronization algorithms.

Keywords — Parallel and Distributed Computing, Discrete Event Simulation (DES), Performance Analysis, Simulations, PHOLD, Mobile Computing

[1]Acronym for School of Electrical Engineering and Computer Science Simulator

Dedication

Dedicated to my mother, Zahida Maqbool, who taught me to persevere and prepared me to face the challenges with faith and humility. She was constant source of inspiration to my life. Although she is not here to give me strength and support, I always feel her presence that used to urge me to strive to achieve my goals in life.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Fahad Maqbool**

Signature: _____

Acknowledgment

First of all, I am very thankful to Allah for His countless blessings and His Prophet Muhammad (S.A.W), who taught us true meaning of life. Secondly, I would like to thank my father for his silent support that always inspired me to achieve my goals in life. And I would like to specially thank and pay my regards to my advisor, Dr. Asad W. Malik, for his invaluable guidance and immense help in every possible way.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Field of Parallel and Discrete Event Simulation (PDES) system has evolved significantly since its birth in 1970s and 80s [1]. However, it still remains an active area of research because of its applications in domains ranging from military, manufacturing, communication networks, computer systems, VLSI design, design automation, to air traffic and road traffic systems. Simulation of a system may have several objectives, including: (i) understanding behavior of a system; (ii) obtaining estimates of performance of a system; (iii) guiding the selection of design parameters; (iv) validation of a model. The availability of low cost microcomputers has introduced simulation to many real life applications.

One kind of discrete simulation is the fixed time increment, or the time-stepped approach, the other kind is discrete-event method. At the basic level, a Discrete Event Simulation (DES) consists of an ordered list of events called event-list, event execution function and state variables that represent the state of the system being modeled. DES Algorithm repeatedly performs the following steps: (1) removes the event with the minimum simulation time from the event-list; (2) evaluates the event message and executes the event

execution function accordingly; (3) it modifies the state variables and (4) generates further events.

Traditional simulation systems are sequential, however, many practical simulations, e.g. in engineering applications, consume several of hours (and even days) on a sequential machine. Parallel and distributed computing environments can be used to reduce execution time of such simulation programs. Parallel and Discrete Event Simulation (PDES) reduces the overall execution time of a simulation by executing it on a parallel or distributed computers. Thus, Parallel and distributed simulation systems have emerged with the concept of utilizing high-performance computing infrastructure to efficiently execute the complex simulations models [2]. PDES is a collection of processes running in parallel that interact through messages. These messages are used to encapsulate the events and they are used to drive the simulation among different processes. Events that simultaneously run on multiple computing systems need to be synchronized. Synchronization (or Time management) algorithms are used to ensure that events are processed in correct order adhering local causality constraint. Local causality constraint ensures that a parallel or distributed simulation produces the same results as a sequential execution.

Fig. 1.1, shows the classification of traditional synchronization algorithms. There are two basic categories of time flow management:

- Time stepped approach, where the simulation time is evenly spaced along a sequence of equal sized time steps or intervals.

- Event driven approach, where the simulation time does not progress in time steps but only when something interesting happens that is referred to as an "event."

Time stepped approaches are only limited to specific applications. PDES community widely adopted event driven approaches that are categorized as two famous synchronization mechanisms, (1) conservative and (2) optimistic. In conservative approach causality error is strictly avoided by applying strategies to determine when it is safe to process an event. Whereas, in optimistic approach simulation continues until causality error is detected and then the error is handled by applying rollback mechanism for recovery and later re-execution of rollback events. There are two types of conservative mechanisms that are synchronous and asynchronous. Synchronous algorithms require global synchronization to compute Lower Bound on Timestamp (LBTS) thus all LPs proceed in synchronous fashion. On the other hand, asynchronous algorithms do not use global synchronization mechanism and LPs proceeds asynchronously.

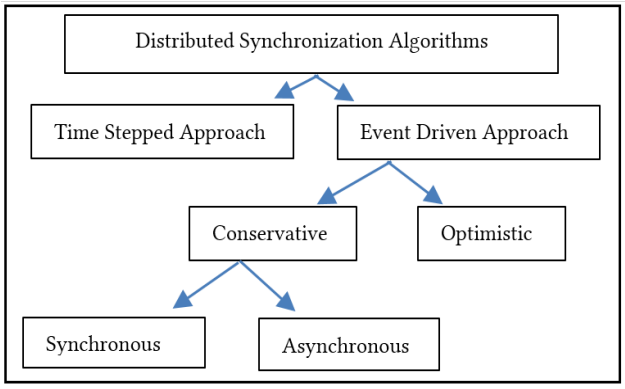

Figure 1.1: Types of Synchronization Algorithms

The field of parallel & distributed simulations have been strongly influenced by emerging technologies. These technologies include massive parallel systems, Cloud computing, GPU computing, embedded computing systems and sensor networks. With the development of technology, use of electronic devices is growing rapidly. Applications and services are hosted inside the

Cloud and can be accessed through any device connected to the internet. In addition, with the inception of internet-of-things (IoT), heterogeneous devices can become part of grid network. Consequently, available computation platforms have changed as compared to the conventional cluster environment. This advancement in technology introduced new challenges for research community. In Cloud computing environment, the workload on a single node can affect the whole simulation process executing on the system. This can cause longer wait time for event execution that can give rise to the straggler message problem [3–5].

Similarly, with IoT, where any computing device can become a part of a network to share its computing resources and store data. Conventional simulation protocols fail to perform efficiently on such network or devices. Most of the sensors and smart-phones are resource constrained, cannot keep a large amount of data and perform large computations. Furthermore, the sending and receiving of data also costs in terms of energy and data transfer rate. Because of ever growing demand for mobile devices and embedded systems in microelectronics market, many modern applications are particularly designed to execute on mobile platforms. Overall performance in terms of energy consumption is the primary design factor for such applications. Therefore, to improve the performance and energy efficiency of modern computing applications and platforms, it is important to accurately estimate their power and energy consumption and identify critical parameters and modules for further optimizations.

Execution of large-scale distributed simulation over mobile and embedded devices opens new research areas that need to be explored. Such research fields include energy-aware distributed simulation and dynamic data-driven applications. Many aspects of such applications have already been focused in

research community. These systems can be useful in many applications such as manufacturing, telecommunications, preparation for inclement weather, defense, intelligent transportation systems and crisis management systems.

In battery operated systems, energy consumption is a major concern; however, minimizing power consumption may not always result in low energy consumption. Decreasing the frequency at which the CPU operates results in low power consumption but it increases the total time required to complete the task. Thus, computations require more time to complete. The problem becomes more complex in an environment where heterogeneous devices are participating in distributed simulations.

In traditional parallel and distributed simulation, logical processes (LPs) are mapped onto different systems or processing cores. These LPs communicate with each other by exchanging timestamped messages. However, process mapping on heterogeneous devices or resource constraint devices can affect the performance of entire simulation. Some simulation algorithms need significantly large storage capacity to store the history of processed events and more computation is required to undo out-of-order execution. Therefore, it increases the number of memory accesses, sending and receiving new event messages and execution of events; that requires significant amount of energy. Thus, resource constraint devices are not suitable for traditional simulation algorithms.

Traditional PDES protocols are very well tested and designed for distributed systems and Cloud infrastructure but still they need to be tested for handheld, mobile and IoT devices. Moreover, traditional frameworks are not designed to support mobile or handheld devices that have memory and energy constraints. Therefore, thorough analysis of traditional PDES frameworks are required to migrate resource hungry modules to the cloud and to

be accessed through well-defined services.

As a case study, an initial instrumentation of a PDES framework (Rensselaer's Optimistic Simulation System - ROSS) is performed on a desktop computing environment to measure power, CPU usage energy and memory consumption using PHOLD benchmark. This case study includes the results of serial, parallel conservative and parallel optimistic approaches. This allows us to understand resource utilization of simulation approaches before adopting the best synchronization model for handheld embedded and IOT devices. The objective is to precisely identify the modules that are resource hungry, so those can be executed on cloudlets and traditional frameworks. Accurately identifying these modules allows simulations to be adapted to handheld devices with little modification.

Using these results, *SEECSSim* – a distributed simulation suite designed to work on mobile and embedded devices is proposed. It includes the core synchronization algorithms as classified in fig. 1.1. The proposed suite includes Chandy-Misra-Bryant CMB NULL message algorithm, Time-Stepped, Tree Barrier, Time Warp and Time Warp with Wolf algorithm (wolf calls). *SEECSSim*, will help researchers in selecting suitable algorithms for mobile and embedded systems. A correctly selected energy efficient algorithm can exploit the true potential of embedded systems for highly scalable parallel and distributed simulation.

1.1 Problem Statement

In battery operated systems, energy consumption is a major concern; however, minimizing power consumption may not always result in low energy consumption. Decreasing the frequency at which the CPU operates results

in low power consumption but it increases the total time required to complete the task. Thus, computations require more time to complete. The problem becomes more complex in an environment where heterogeneous devices are participating in distributed simulations.

Traditional PDES protocols are very well tested and designed for distributed systems and Cloud infrastructure but still they need to be tested for handheld, mobile and IoT devices. Moreover, traditional frameworks are not designed to support mobile or handheld devices that have memory and energy constraints. Therefore, thorough analysis of traditional PDES frameworks are required to migrate resource hungry modules to the cloud and to be accessed through well-defined services.

1.2 Thesis Outline

Chapter 2, presents a brief background about the parallel and distributed discrete event simulation and provides a brief about different existing frameworks while specially focusing on ROSS simulation framework, it also lists various tools and techniques that are used to perform this study. Chapter 3 is about the literature review of energy aware computing in the domain of High Performance Computing (HPC), mobile computing and sensor networks. It also includes the literature about different profiling methods, techniques and tools. Chapter 4 presents the instrumentation results of ROSS and discusses about various aspects of the performance analysis. Chapter 5 presents the main contribution of this thesis. It includes the discussion about the proposed simulation framework for mobile devices and different algorithms that are included in this framework. Chapter 6 discusses the results of PHOLD benchmark application over mobile device and investigates the performance

of different algorithms. Finally, chapter 7 concludes this thesis and presents possible future directions.

Chapter 2

Background

A brief description of Parallel and Distributed Simulation platforms specially Rensselaer's Optimistic Simulation System (ROSS) is presented in this section. ROSS simulation framework is selected to present a case study for the instrumentation of PDES systems with different synchronization algorithms. Some Important features of ROSS framework are discussed and explained. Moreover, the software tools and the main techniques used for performing instrumentation and power consumption analysis are also discussed in this section. In Table 2.1, different PDES frameworks are briefly explained to have an understanding of how some of these frameworks have be designed previously.

2.1 Parallel Discrete Event Simulation (PDES)

A Discrete Event Framework is a system where state changes (events) happen at discrete occurrences in time, and events take zero time to happen. It is accepted that nothing (interesting) happens between two continuous events, that means, no state change happened in the system between the events.

Such frameworks that can be categorized as Discrete Event Frameworks can be modeled using Discrete Event Simulation (DES) Systems.

A Discrete Event Simulation (DES) comprises of events and Logical Processes (LPs). LPs are agent entities in a simulation system and store the simulation system states using state variables. A DES continues the execution by sending event messages between LPs to communicate state changes. It is important to execute events while adhering local causality constraint. It means if the simulation system does not respect causal ordering, causality error occurs. This causality error is a logical error that occurs if an event with higher value of time-stamp is executed before an event with smaller time-stamp [6]. In a discrete event simulation, a scheduler is the core part of the simulation engine. Hence, performance of a discrete event simulation is directly influenced by the working of scheduler. The DES can be implemented using sequential and parallel techniques. in a sequential approach events are executed in time-stamp order in a serial manner. In this simplest approach main focus is to execute a simulation model without concerning the performance in terms of fast execution. While there can be many Logical Processes (LPs) in sequential DES, all events are stored in a single priority queue ordered by their virtual timestamp and executed one by one like an ordered sequence [7].

Similarly, Distributed and Parallel Discrete Event Simulation (PDES), provides the capability of executing a single discrete event simulation program on multiple cores using parallel computing [8]. There are two main categories of time management/scheduling mechanisms that PDES widely follow; conservative and optimistic. In conservative approach causality errors are strictly avoided by applying strategies to determine when it is safe to process an event. While in optimistic approach simulation continues until

a causality error is detected and this situation is recovered by means of a rollback mechanism and later re-execution of rolled-back events.

Table 2.1: Commonly Used PDES Frameworks

S. No.	OS PDES Frameworks	Language	Simulation Model	Description
1	GloMoSim (Qualnet)	C-based PARSEC	Hybrid	Developed at the University of California, Los Angeles, Global Mobile system Simulator (GloMoSim) is a set of library modules, developed for parallel execution of wireless network simulations [9]. It is an extensible simulator implemented on shared memory and distributed memory system.
2	DaSSF	C++ Java	Conservative	DaSSF (Dartmouth Scalable Simulation Framework) is a Scalable Simulation Framework (SSF) for discrete-event simulation [10].The SSF is designed to achieve interoperability with other SSF compliant frameworks. DaSSF is capable of simulating very large-scale network with tens of thousands of complex nodes.
3	ARTIS+ GAIA	C with Java bindings	Hybrid	ARTIS (Advanced RTI System) is a middleware for Parallel and Distributed Simulation (PADS) that can simulate complex systems [11]. It provides a set of simple services to simulate massively populated system models. ARTIS uses an adaptive approach and makes use of physical allocation of LPs for efficient execution and communication. It supports different communication systems such as shared memory, MPI and network communication. GAIA (Generic Adaptive Interaction Architecture), built on th top of ARTIS, is also an adaptive middleware that dynamically reallocates the LPs to optimize the simulation execution [12]. GAIA uses a migration policy to partition and allocate the interacting components over many LPs dynamically.
4	LUNES	C	Conservative	LUNES (Large Unstructured NEtwork Simulator) is an agent-based simulator to model complex large-scale networks [13]. Its modular approach separates the phases of topology creation, protocol simulation and performance analysis. LUNES uses dynamic model partitioning and simulation middle-ware services provided by GAIA and ARTIS frameworks respectively.
5	ScipySim	Python	Conservative	ScipySim is a distributed simulator to simulate heterogeneous systems developed using SciPy scientific computing platform [14]. It is based on the generalized Kahn theory of heterogeneous system semantics. It was designed to provide basic simulation capability to develop simulations using Python.
6	ROOT-Sim	C	Optimistic	ROOT-Sim (The ROme OpTimistic Simulator) is a MPI based parallel simulation platform developed using C/POSIX technology [15]. To achieve high scalability and performance it uses a set of optimized protocols to minimize the run-time overhead.
7	Spades/JAVA	Java	Optimistic	SPaDES/Java (Structured Parallel Discrete-event Simulation in Java) is a process oriented parallel simulation [16]. It was designed to isolate the synchronization and parallelization implementation details. It supports both sequential and parallel simulations.
8	ErlangTW	Erlang	Optimistic	ErlangTW is a parallel and distributed simulator based on Time Warp synchronization [17]. Erlang is a concurrent programming language specifically designed to build distributed systems. ErlangTW simulation model can be executed on single-core processors, shared memory multiprocessors and distributed memory clusters.
9	GO-Warp	GO	Optimistic	GO-Warp simulator, implemented using GO programming language, is also based on Time Warp synchronization [18]. It uses Samadi's algorithm for Global Virtual Time (GVT) computation. Using concurrent execution and inter-process communication mechanisms of GO, LPs are allowed to proceed execution without blocking.

2.2 Rensselaer's Optimistic Simulation System (ROSS)

In this study, ROSS, a PDES framework is used that is based on Message Passing Interface (MPI). ROSS is a high performance and extremely modular PDES system that uses small (and constant) amount of memory to keep state variables and execute events [19]. Its modular implementation, use of reverse computation, Kernel Processes and Fujimoto's GVT (Global Virtual Time) algorithm makes it a state of the art simulation system to perform experimental studies. Continuous analysis shows that ROSS outperforms the famous GTW (Georgia-Tech Time Warp) System. In this study, performance and power consumption of the ROSS simulation implementation is analyzed under classical PHOLD simulation model benchmark. While ROSS is based on optimistic scheduling approach it also provides implementation for both conservative and sequential approaches. Our study includes performance and power consumption analysis of all sequential, conservative and optimistic approaches.

2.3 PHOLD Benchmark Model

PHOLD is one of the commonly used benchmark models used to analyze the performance of synchronization algorithms designed for parallel and distributed simulations. It is the parallel version of HOLD model, initially used for the performance analysis of sequential event list algorithms. PHOLD model consists of N connected logical processes (LPs). On receiving and processing an event message, LPs sends an event to its neighboring LP, thus a fixed message population circulates throughout the model. The message

size, message population size, timestamp increment and the message routing probabilities can be varied to test the simulation.

2.4 Instrumentation Tools

Various tools have been employed for carrying out an extensive analysis of the PDES frameworks. These include tools developed by Intel such as PIN-based tool, Vtune Amplifier and SoC Watch [20] as well as profiling tool Allinea Forge Map (renamed as Arm Map) from Arm [21] and Trepn. A more detailed description of each tool along with its usage follows.

2.4.1 Performance Profiling – Intel PIN tool

PIN-tools provide a dynamic memory instrumentation framework to perform instrumentation on both IA-32 and x86-64 architectures. The PIN framework can be used to analyze applications running in the user-space and to perform instrumentation on compiled binary files.

2.4.2 CPU Usage – Intel Vtune Amplifier

Intel® VTune™ Amplifier is a profiling tool used for analyzing the code for better performance by profiling the system CPU usage. It provides a user-friendly interface to analyze and obtain results using enriched performance insights. It helps the application developers to write code that is more threaded, scalable, vectorized and tuned. VTune™ amplifier is used to check the average CPU usage and the amount of time that ROSS spends on locks and waits in a parallel setup. The results are used to estimate the speedup that can be obtained by parallel version with respect to the serial execution.

2.4.3 CPU Cores Temperature – Intel® SoC Watch

Intel SoC is a command-line utility designed by Intel [20] to study the temperature profiles of CPU cores. SoC watch is used to study the behavior of ROSS with an increasing load on each LP.

2.4.4 Energy, Memory Consumption – Allinea MAP

Allinea MAP is a profiling tool designed for a wide range of applications including parallel, single threaded and multi-threaded (Pthread, OpenMP, and MPI) applications that are based on Fortran, C, and C++ [21]. It performs a thorough analysis of any target application, pinpoints the bottlenecks in the execution code and keeps logs for power, energy and memory consumption traces. The Allinea forge MAP tool is used to get insight of the power, energy and memory consumption of ROSS while running the PHOLD benchmarking model [22, 23] with a variable number of logical processes (LPs).

2.4.5 Energy, Memory Consumption for Android – Trepn Profiler

Trepn profiler[1], a product of Qualcomm, is a diagnostic tool that lets the developers profile the performance of Android applications running on mobile devices. It is a hardware sensor-based power profiler that can help in optimizing code for CPU usage, frequency, memory statistics, energy consumption and network usage. It can display data in real-time or store it in a log file for a later off-line analysis. Trepn can analyze one particular application, or the device as a whole.

[1] https://developer.qualcomm.com/software/trepn-power-profiler

Chapter 3

Literature Review

Despite the need of profiling power and performance of simulation protocols and creating energy-aware PDES platforms, there are only few articles that address this issue. There exists substantial literature work related to energy-aware computing in the domain of High-Performance Computing (HPC), mobile computing platforms and Wireless Sensor Networks (WSN). Moreover, different profiling methods, techniques, and tools have been proposed over the years. In this section, brief discussion of some related contributions is presented for performance analysis and energy-aware simulation for HPC systems as well as mobile and embedded systems.

Many tools have been developed over the years for profiling performance and power consumption to generate, analyze and visualize the data of systems and applications from functional components. Tuning and analysis utilities [24], [25], [26], [27] and [28] provide support for instrumentation and performance visualization of parallel applications. Isci et al. [29] presented an approach to estimate power consumption using performance logs. Similarly, authors in [30] provided a framework called PowerPack that can be used for energy profiling and analysis of parallel applications on multi core proces-

sors. Authors in [31] indicated that power profiles always correspond to the characteristics of the application and increasing number of nodes results in more power consumption and not always results in better performance. Authors in [32] analyzed the power consumption in relationship with software components. Their analysis shows that in different software scenarios, power consumption on a general-purpose computer system can vary from 12% to 20%. Authors in [33] and [34] focused on low power embedded systems to analyze and benchmark HPC applications for their energy consumption.

There is substantial work available in power-aware computing and various techniques has been deployed to reduce the power consumption [35], [36] and [37]. Computing energy consumed by each machine instructions is one way to profile the energy consumption of all functional components. Tiwari et al. [38] discussed that functional level profiles for energy consumption obtained through computing energy consumed by each machine instruction. However, the proposed work is only limited to function level energy marking. Whereas, communication between processes is also holding a major portion is parallel and distributed simulation. In a distributed simulation, different techniques are used to reduce the communication delay between processes [39]. One such approach is to utilize the different cores available in a physical system [40]. However, the synchronization algorithms incurred overhead which is difficult to reduce.

Most of the existing work on energy-efficient computing has been done for HPC environment [41–44]; where different techniques are used to optimize the use of energy such as DVFS, process migration, task consolidation and Dynamic Power Management (DPM). R. Child et al. in [45] explored the features of Dynamic Voltage and Frequency Scaling (DVFS) to enhance the performance and reduce power consumption by repeatedly reducing the op-

erating frequencies of the cores. The authors investigated energy efficiency through DVFS while for Time Warp simulation algorithm. The proposed study is conducted over physical systems using MPI version of wrapped TW simulator.

Similarly, G. Tom et al. [46] described the integration of energy-aware module to simulate the energy consumption of distributed systems. Authors have provided an overview of energy-aware simulations and described DVFS simulation tools required for obtaining accurate simulations in terms of power consumptions. This work is mostly related to the energy of cloud systems; therefore, authors have explored the DVFS and cloud simulators in detail. Moreover, they have also added DVFS features in one of the cloud simulator.

Communication is a common performance bottleneck for fine grained parallel applications like ROSS,. Authors in [47] and [48] have discussed the techniques for improving network performance by reducing lock contention and overlapping communication. Jagtap et al. [49] analyzed the performance of ROSS simulation framework on two different platforms and compared multithreaded implementation with MPI based implementation. Erazo et al. [50] presented a case study to profile the energy consumption of distributed simulation tested it on their PRIME simulator [51]. The authors of PRIME Sim concluded that using more nodes to achieve parallelism results in significant increase in energy consumption.

In [52], Fujimoto shared future research challenges for PDES. These challenges include *large-scale simulation of complex networks*, *exploiting GPU's*, *Cloud computing exploitation*, *composable simulation* and *energy consumption of PDES*. In PDES energy consumption is less explored with respect to other aspects. The minimization of power consumption through change in clock rate (DVFS) can eventually increase the overall energy required to com-

plete the task. The schemes such as DVFS are more suited for data centers and super computers. Therefore, for IoT, the power-aware and energy-aware techniques are more important for design consideration of PDES over handheld devices as energy consumption is based on many factors such as network communication, memory usage etc.

Traditional simulations are designed for cluster environment. With the advancement of technology, easy availability of infrastructure-as-a-service offers flexible computing environment on a pay-as-you-go model. New PDES techniques are proposed for such cloud architectures. In [4] authors studied the execution of conservative algorithm over various configurations of Amazon EC2. The objective is to see the suitability of cloud platform for distributed discrete event simulation. For conservative algorithm, null messages play a significant role in the performance of whole simulation system. Therefore, the authors tested various variations of synchronization algorithms such as Chandy−Misra−Bryant, time-out based null message sending, deadlock avoidance based null message, on-demand null message, timeout protocols etc. The results showed that timeout and blocking protocol performed better in a cloud environment.

Cloud provides a multi-tenant paradigm, therefore, execution of optimistic PDES over cloud results in a large number of rollbacks. This is because systems are not equally loaded in terms of number of jobs. Moreover, some tasks require more computation whereas other need more communication. Similarly, Asad et al. [3] proposed a PDES model for cloud environment to improve the performance of optimistic parallel simulation through dynamically defining the barrier points, to reduce the total number of rollbacks. The experimental section shows the significant gain in terms of performance over traditional optimistic simulation.

Similarly, Yihua Wu et al. [53] presented a BOINC based system for Cloud environment to execute parallel and distributed simulation over private cloud infrastructures. BOINC is a middle-ware developed for volunteer and grid computing. It is an open-source framework designed to support task distribution and result gathering in client server model. However, use of private cloud is not a recommended option due to its cost and other management factors. The main reason for adopting private clouds is due to the sensitive nature of simulations results.

In the context of distributed simulation over mobile/embedded devices, Biswas et al. [54] discussed the techniques to create the power profiles. Energy consumption of simulation model, engine, computations and communications is separated to understand the energy consumption of each aspect of simulation. They have also presented the comparative analysis of energy consumed by Chandy-Misra-Bryant and YAWNS algorithms. Similarly, Malik et al. [55] have analyzed the energy consumption of Time Warp protocol over smart phones. Moreover, on-line distributed simulation such as traffic prediction system requires significant amount of energy. Neal et al. [56] analyzed the energy consumption of data driven traffic simulations on mobile devices. Online traffic prediction requires significant amount of energy therefore, understanding the energy consumption at various levels, helps in optimizing the use of resources. The authors presented the empirical investigation of modules such as data transmission, gathering and traffic computations. Fujimoto et al. [57] has presented a detailed work on power efficient distributed simulation. The authors have covered few conservative and optimistic synchronization algorithms along with discussion on energy efficient distributed simulations. The main objective of their work is to analyze the power consumption of various distributed simulation techniques along with profiling

of simulation engine, application, and communication. The experiments are conducted on multiple configurations, such as Jetson TK1 development board and quad-core LG Nexus 5 cellular phone.

In this study, a detailed study of existing traditional algorithms on hand held devices is presented. The analysis includes the execution time, CPU, memory usage, energy consumption and event rate. This article will serve as a guideline for PDES community especially the new researchers, to help them in selecting the right protocols for embedded systems keeping in view the resource constraints.

Chapter 4

Instrumentation of ROSS

This chapter presents an in-depth discussions of the results obtained from the analysis of the ROSS framework. The results reported in this section are obtained averaging multiple independent simulations runs over the specified system while using the profiling tools discussed in Section **??**. The PHOLD simulation model is used to benchmark the ROSS framework. In order to determine the adaptability of the different synchronization algorithms (serial, conservative and parallel) to mobile devices, these algorithms are compared in terms of their CPU usage, memory consumption, total execution time, and energy and power consumption.

For this study, a 4th generation Intel® Core™ i7-4790 processor (Intel Haswell family) 3.6 GHz (Hyper-Threading) with 4 cores, 8 threads, 8 MB Cache, 8 GB RAM, 5 GT/s DMI2 and Ubuntu 14.04.3 operating system with kernel version 3.19 is used.

4.1 Execution Time

Results for the PHOLD benchmark for serial/sequential, conservative and optimistic approaches running on top of the ROSS framework are presented in Tables 4.1, 4.2 and 4.3 respectively. In all simulations, a linear mapping between the Logical Processes (LPs) and the physical processors has been used. The total number of events is kept constant while also works as stopping condition for the simulation. Results show that as expected, the serial execution takes more time as compared with parallel conservative and parallel optimistic approaches. Using 1024 LPs, the sequential simulation took *34.827* seconds to complete whereas the conservative and optimistic parallel approaches took *21.4* and *24.7* seconds respectively. Further increasing the number of LPs to 524288, the sequential execution took *8.72* hours whereas parallel conservative took *4.04* hours and parallel optimistic execution *5.99* hours. Results shows that the conservative simulation execution outperforms the other techniques across the range of LPs. As discussed earlier, in optimistic simulation, there are out of order event executions that cause some events to rollback and then re-executed. Thus the total number of events is larger than the number of committed events and that results in performance reduction when compared to the conservative approach. Moreover, functions such as GVT computations, fossil collection and reverse computation that are required by the optimistic simulator are among the reasons of the increased execution time.

4.2 Memory Consumption

In this section, the memory usage results for the PHOLD execution on the ROSS framework are presented. Using 1024 LPs, the parallel conservative

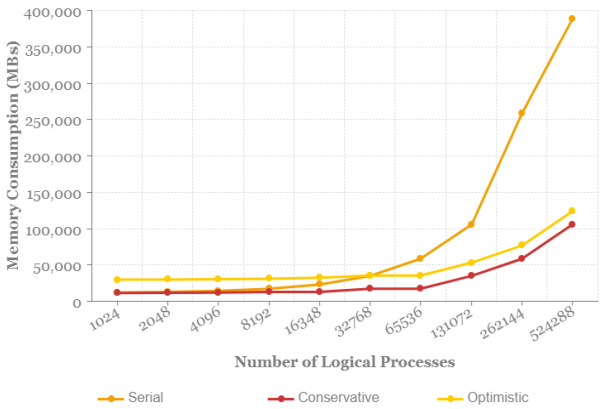

Figure 4.1: Memory usage analysis

used *11,528* MBs and the optimistic used *29,768* MBs. The memory usage with 524,288 LPs, the parallel conservative used *105,552* MBs and the optimistic *123,792* MBs to complete a simulation run. Comparing the conservative and optimistic approaches in Figure 4.1, a similar trend is observed in memory usage with conservative approach requiring the least amount of memory. Optimistic approach requires more memory as every LP must implement the rollback mechanism, therefore, the LPs need to maintain the history of all the processed events to be able to handle transient and anti-messages. The memory usage is also based on execution time. The serial execution is taking the maximum execution time as reported in Table 4.1, thus, the memory usage reported for serial is more for large number of LPs e.g. for 524288 LPs the memory usage is around 388176 MBs.

4.3 Efficiency, GVT computation, Fossil Collection and Rollbacks

These important metrics for simulation synchronization schemes are reported in Tables 4.1, 4.2 and 4.3. The term *efficiency* is defined as the ratio between the number of committed events to the total number of events. Since there is no rollback mechanism in serial and parallel conservative approaches, both approaches show a 100% efficiency. On the other hand, in the optimistic approach, the committed events are always less than the total number of events (due to the rollback mechanism); therefore, the efficiency of parallel optimistic synchronization is always lower than 100%. The total amount of events processed in all three approaches ranged from 0.102 to 52,432 billion events with an increasing number of LPs. Another important factor to measure the performance efficiency of the simulation system is the processing rate of events. As specified by the PHOLD model, increasing number of LPs leads to an increase in the number of events to be processed and thus the communication overhead increases, resulting in decreased event rate. The event rate (measured as number of events per second) is the lowest in serial execution (ranging from 2.94 to 1.67 millions events/second) and the highest for the conservative approach (ranging from 4.78 to 3.60 millions events/second), while optimistic lies in the middle (ranging from 4.14 to 2.42 millions events/second). The reason behind this decreasing of the event rate while increasing the total number of LPs is due to the scheduling and communication overhead between the LPs. Comparison results for above-mentioned parameters suggests that the conservative approach outperforms the other options.

In parallel optimistic simulation, the Global Virtual Time (GVT) acts

Table 4.1: Results - Serial Execution of PHOLD with Varying Number of LPs

	LPs									
	1024	**2048**	**4096**	**8192**	**16348**	**32768**	**65536**	**131072**	**262144**	**524288**
Running Time (seconds)	34.827	72.272	148.343	303.874	658.453	1556.235	3516.398	7768.777	3730.151	31381.415
Event Rate (million events/sec)	2.94	2.83	2.76	2.70	2.49	2.11	1.86	1.69	1.76	1.67
Memory Allocated (MB)	12080	12816	14288	17232	23120	34896	58448	105552	258448	388176
Memory Wasted (MB)	533	341	469	213	213	213	213	212	213	210
Total Events Processed (billions)	0.102	0.205	0.410	0.819	1.638	3.277	6.554	13.107	6.554	52.428
Efficiency (%)	100	100	100	100	100	100	100	100	100	100

Table 4.2: Results - Parallel Conservative Execution of PHOLD with Varying Number of LPs

	LPs									
	1024	**2048**	**4096**	**8192**	**16348**	**32768**	**65536**	**131072**	**262144**	**524288**
Running Time (seconds)	21.432	42.233	83.687	169.071	375.403	770.323	1598.350	3513.903	7212.069	14542.773
Event Rate (millions events/sec)	4.78	4.85	4.89	4.85	4.36	4.25	4.100	3.73	3.63	3.60
Memory Allocated (MB)	11528	11712	12080	12816	12873	17232	23120	34896	58448	105552
Memory Wasted (MB)	677	629	533	341	469	213	213	213	213	212
Total Events Processed (billions)	0.102	0.205	0.410	0.819	1.638	3.277	6.554	13.107	26.214	52.428
Total LBTS Computations (millions)	0.200	0.300	0.500	0.900	1.700	3.300	6.504	12.920	25.751	51.394
Efficiency (%)	100	100	100	100	100	100	100	100	100	100

as a barrier point in the past for rollback guaranteeing that no process can rollback to a timestamp that is smaller than the current GVT value. The GVT used in optimistic simulation is similar to the Lower Bound Timestamp (LBTS) in the conservative approach [58]. The results show that the total number of GVT computations in optimistic (ranging from 0.10 up to 51.20 millions of computations) are slightly less than in the conservative approach (ranging from 0.20 up to 51.40 millions of computations). This due to the

Table 4.3: Results - Parallel Optimistic Execution of PHOLD with Varying Number of LPs

	LPs									
	1024	**2048**	**4096**	**8192**	**16348**	**32768**	**65536**	**131072**	**262144**	**524288**
Running Time (seconds)	24.727	50.468	100.051	216.820	510.869	1197.870	2287.070	5065.395	10401.267	21598.115
Event Rate (millions events/sec)	4.141	4.058	4.094	3.778	3.207	2.735	2.760	2.588	2.520	2.427
Memory Allocated (MB)	29768	29952	30320	31056	32528	35472	35472	53136	76688	123792
Memory Wasted (MB)	869	821	725	533	149	405	405	405	405	404
Fossil Collect Attempts (millions)	0.408	0.808	1.609	3.210	6.410	12.811	25.611	51.212	102.410	204.811
Total Events Processed (billions)	0.104	0.207	0.412	0.822	1.641	3.280	3.280	13.110	26.217	52.432
Total GVT Computations (millions)	0.102	0.202	0.402	0.802	1.603	3.203	3.203	12.803	25.603	51.203
Total Roll Backs	133218	79114	45681	23980	12143	5983	5970	1665	716	421
PrimaryRoll Backs	105890	63101	35860	19571	10561	5541	5448	1578	681	398
SecondaryRoll Backs	27328	16013	9821	4409	1582	442	522	87	35	23
Efficiency (%)	98.13	98.96	99.43	99.69	99.84	99.90	99.92	99.98	99.99	99.99

fact that in the conservative approach a large number of LBTS computations is performed to avoid causality errors. Moreover, for optimistic synchronization, the simulation frameworks typically store event histories to proactively resolve issues due to the causality errors. Storing the event histories increases the memory usage over time and therefore the memory must be reclaimed periodically (to reduce the runtime memory requirements of the simulation framework). This reclamation process is commonly known as fossil collection and add a relevant overhead to the simulation execution. The high memory wastage is also made evident by a large number of fossil collection attempts in optimistic execution to reclaim memory.

A mechanism called rollback is necessary in the optimistic simulation approach whenever a causality error is detected. Extensive rollbacks affect the efficiency of the simulation engine. More in detail, there are two different types of rollbacks, the primary rollbacks and secondary ones. A primary rollback occurs whenever a LP receives an event with a time-stamp that is lower than its local time. The primary rollbacks transitively propagate secondary rollbacks to other LPs to revert the effects of the previously sent messages [59]. The results of the optimistic setup show that as the number of LPs increases there is a considerable decrease in rollbacks (ranging from 133,218 to only 421 rollbacks for 1024 LPs and 524,288 LPs respectively). This is due to the fact that the executions slow down caused by the increased number of LPs also decreases the number of causality errors and therefore the number of rollback invocations.

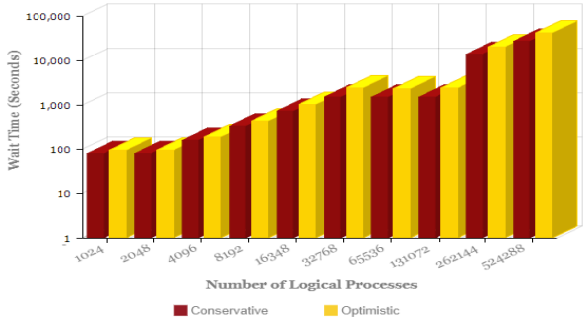

Figure 4.2: Wait Time Analysis for PHOLD model

4.4 Wait Time

The wait time is another interesting metric that can be used for studying the performance of a system. It is defined as the time spent on locks and waits in a parallel execution. Wait time for the serial simulation execution is negligible as the maximum wait time is *0.007* secs. This is due to the fact that the sequential execution do not have to wait for the completion of other processes. For parallel approaches, the results show that the optimistic execution spends more time on locks and waits than the conservative techniques as shown in the Figure 4.2. This increase in the wait time for the optimistic execution (as the number of LPs is increased), is caused by the rollbacks that increase with the number of total events to be processed. Greater is the number of pending events, higher is the synchronization and scheduling overhead. It is worth noting that the difference between the wait time for parallel conservative and parallel optimistic approaches is less than 80 secs.

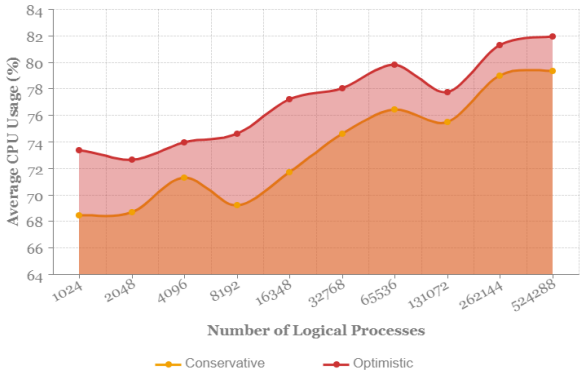

Figure 4.3: Average CPU usage for PHOLD model

4.5 Average CPU Usage

The average CPU usage defines the CPU utilization that depends on the total number of processor cores being used for running the simulation. Moreover, it gives some information on the concurrency level of the code being executed. In the case of the serial execution the average CPU usage is constant as only one processor core is used; whereas the results for parallel versions are presented in Figure 4.3. The results indicate that the average CPU usage for optimistic simulation is higher as compared with the conservative approach. This is exactly what the optimistic approach aims for, due to the optimism that allows the LPs to execute events on availability. Moreover, the optimistic approach also utilizes more CPU due to the fossil collection mechanism and the need of dealing with straggler and anti-messages.

The results in terms of CPU usage, energy and power consumption are reported in Figure 4.3-4.5. The power and energy are related concepts; however, the energy consumption and power consumption are not the same. It is possible to simply define the power as the energy consumed per unit of time (rate of energy consumption) as given in Equation 4.1.

$$Power = Energy/Time \tag{4.1}$$

Desktop systems and similar devices have a constant power supply, while the mobile devices and the other battery operated devices are energy constraint. For this reason, the results for both energy and power consumption have been collected and reported with a varying number of LPs.

4.6 Energy Consumption

The results collected by the Allinea forge's Map utility in terms of CPU energy consumption are reported while the PHOLD model was running. Figure 4.4 shows the energy consumption results for sequential, parallel conservative and optimistic executions (varying number of LPs ranging from 1024 to 524,288). The optimistic execution results show the highest energy consumption due to its usage of all the available physical cores and the extra computations that are needed with respect to the conservative approach. Interestingly, the results show that the sequential execution uses a single CPU but still consumes a large amount of energy due to its longer execution time (with respect to the other approaches).

4.7 Power Consumption

The CPU power consumption (measured in Watts) is a significant portion of the overall power consumed. It is a combination of the electrical energy used by CPU while performing various tasks per unit time and the energy dissipated in the form of heat during the course of execution. Figure 4.5 shows that for low number of LPs in the simulation, the CPU power consumption

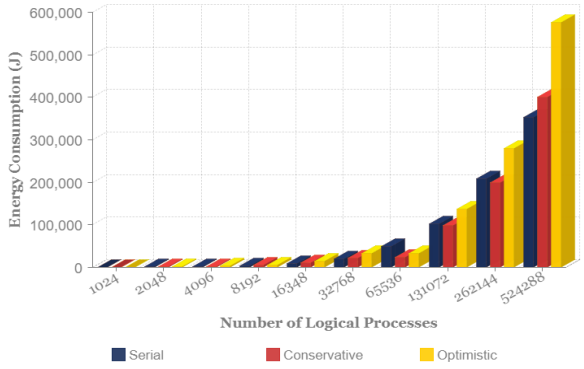

Figure 4.4: Energy consumption analysis

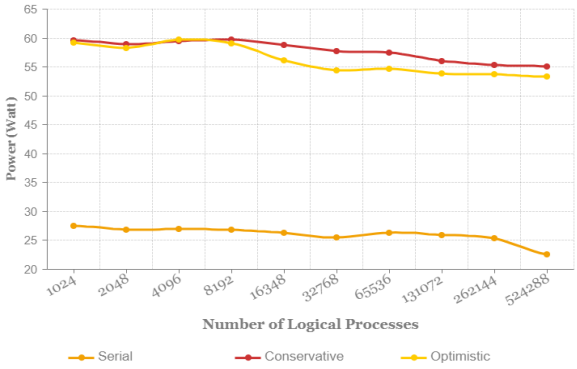

Figure 4.5: Power consumption analysis

for both conservative and optimistic approaches are almost similar while the optimistic approach showing slightly better performance for higher numbers of LP. On the other hand, the CPU power consumption of serial is very low as compared with both parallel versions. This is due to the fact it uses a single CPU core instead of multiple cores.

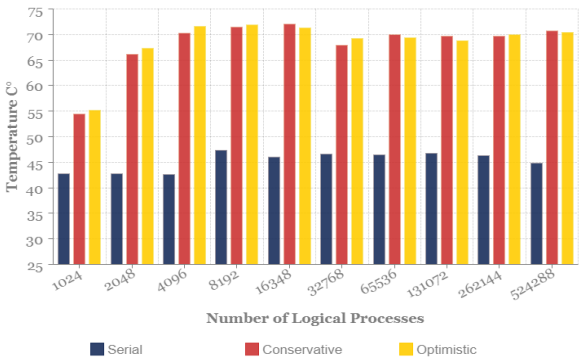

Figure 4.6: CPU temperature statistics

4.8 CPU Temperature

The average temperature statistics for each CPU core, while the PHOLD simulation model is being executed, are presented in Figure 4.6. For serial execution, the temperature of the specific CPU core is higher with respect to the another cores. This is obvious, since the serial approach is able to use only a single core. In the parallel executions (both conservative and optimistic) the temperature increases on all the four CPU cores that are available. This is due to the parallelism in conservative and optimistic execution since both approaches are able to utilize all the available CPU cores. In conservative, the minimum and maximum average temperature are 52.0°C and 73.2°C respectively. Similarly, in optimistic they are 52.5°C and 73.2°C.

It is evident from the temperature results that increasing the number of CPU cores in use, more energy is dissipated as heat and thus the average CPU temperature increases. The maximum average temperature of the serial version is comparable to the minimum average temperature of parallel versions. The maximum average temperature of parallel versions was 18.8°C higher than the serial one. The reason behind the high energy consumption

Table 4.4: Functional Level Execution Time for the Serial Version of ROSS

	LP's				
Functions (% Time)	**1024**	**2048**	**32768**	**262144**	**524288**
tw_run	100	99.9	99.9	100	99.9
tw_scheduler_sequential	100	99.9	99.9	100	99.9
phold_event_handler	89	86	76	77	80
tw_rand_exponential	35	33	23	22	21
tw_event_new	21	21	20	19	20
rng_gen_val	14	14	15	19	18
tw_event_send	12	11	13	11	15
Others	7	7	5	6	6
Others	11	14	24	23	20
Others	0	0.1	0.1	0	0.1
Execution Time (sec)	**34.8**	**72.3**	**1556.2**	**3730.2**	**31381.4**

for optimistic and conservative execution is that a significant part of the energy is dissipated in the form of heat.

4.9 Functional Level Execution Time

The execution time of the core functions for the serial, parallel conservative and optimistic simulation execution has been measured and is reported in Tables 4.4, 4.5 and 4.6. These tables list the functional hierarchy and the time spent on the execution of the main functions and their corresponding individual sub-functions. In all three simulation approaches, the total number of events is kept constant.

The functional level percentage time for each sequential simulation execution (for different numbers of LPs) is reported in Table 4.4. The results show that the serial simulation with 1024 LPs took *34.8* seconds while increasing the number of LPs to 524,288 it required *8.72* hours to complete. The texitittw_scheduler_sequential is the main executing function since it responsible for the event processing, memory management and virtual time

Table 4.5: Functional Level Execution Time for the Parallel Conservative Version of ROSS

	LP's									
	1024		2048		32768		262144		524288	
Functions (% Time)	**Total**	**MPI**	**Total**	**MPI**	**Total**	**MPI**	**Total**	**MPI**	**Total**	**MPI**
tw_run	99.7	43	99.8	40	99.9	32	100	27	100	27
tw_scheduler_conservative	99.5	43	99.8	40	99.9	32	100	27	100	27
phold_event_handler	59	14	64	14	55	12	58	10	58	11
tw_event_send	21	14	22	14	18	12	19	10	21	11
tw_rand_exponential	15	-	16	-	13	-	12	-	11	-
tw_event_new	14	-	15	-	14	-	11	-	12	-
rng_gen_val	6	-	8	-	8	-	13	-	12	-
Others	3	-	3	-	2	-	3	-	2	-
tw_net_read	17	15	17	14	16	12	16	11	16	11
service_queues	17	15	17	14	16	12	16	11	16	11
Others	0	-	0	0	0	0	0	0	0	0
tw_gvt_step2	14	14	9	12	10	8	6	6	6	5
MPI_Allreduce	13	13	8	11	10	8	5	5	5	5
Others	1	1	1	1	0	-	1	1	1	-
Others	9.5	-	9.8	-	18.9	-	20	-	20	-
Others	0.3	-	0.2	-	0.1	-	0	-	0	-
Execution Time (sec)	**21.4**		**42.2**		**770.3**		**7212.1**		**14542.8**	

computation.

Table 4.5 reports the execution time for the simulation functions in case of parallel conservative approach. The functional time reported in Table is in percentage of total execution time. The total execution time of the simulation run with 1024 LPs was 21.4 seconds out of which the parallel execution part was of 9.2 seconds. Similarly, when the number of LPs increases, the total execution time was about 4.04 hours with a parallel execution part of 1.09 hours. This gives an idea on the degree of parallelism that MPI based ROSS provides when compared to the sequential execution – the execution time is decreased to half with the use of a parallel conservative executions. It is worth noting that for the parallel version, the degree of parallelism tends to decrease as the number of LPs is increased. This can be attributed to the earliest time tag GVT or (LBTS in the case of conservative synchronization) associated to the unprocessed pending events. The GVT/LBTS computations need to

Table 4.6: Functional Level Execution Time for the Parallel Optimistic Version of ROSS

	LP's									
	1024		2048		32768		262144		524288	
Functions (% Time)	**Total**	**MPI**	**Total**	**MPI**	**Total**	**MPI**	**Total**	**MPI**	**Total**	**MPI**
tw_run	99.8	37	99.9	38	99.9	26	99.9	23	99.9	23
tw_scheduler_optimistic	99.6	37	99.7	38	99.9	26	99.9	23	99.9	23
tw_sched_batch	60	13	58	13	54	8	52	7	50	7
phold_event_handler	52	13	51	13	39	8	37	7	38	7
tw_event_send	18	13	18	13	13	8	13	7	13	7
tw_event_new	11	-	11	-	9	-	8	-	8	-
rng_gen_val	7	-	5	-	7	-	7	-	8	-
tw_rand_exponential	13	-	14	-	9	-	8	-	7	-
Others	3	-	3	-	1	-	0	-	1	-
tw_gvt_step2	18	11	20	12	28	8	28	7	29	8
tw_pe_fossil_collect	7	0	8	0	19	-	20	0	20	0
MPI_Allreduce	10	10	11	11	8	8	7	7	7	7
Others	1	1	1	1	1	0	0	0	2	1
tw_net_read	20	12	21	13	17	9	20	8	21	9
service_queues	20	12	21	13	17	9	20	8	21	9
test_q	9	2	9	2	9	0.8	12	1.2	13	1
recv_begin	11	10	12	11	9	8	8	7.1	8	7
Others	0	-	0	-	0	0.2	0	0	0	1
tw_kp_rollback_to	1.4	0.6	0.8	0.2	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
tw_event_rollback	1.2	0.6	0.8	0.2	<0.1	<0.1	<0.1	<0.1	<0.1	<0.1
Others	0.2	-	0	-	0	0	0	0	0	0
Others	0	-	0	-	0	0	0	0	0	1
Others	1.6	-	0.7	-	0.9	1	0	1	0	0
Others	0.2	0	0.1	0	0.1	0	0.1	0	0.1	0
Execution Time (sec)	**24.7**		**50.5**		**1197.9**		**10401.3**		**21598.1**	

be done in a sequential fashion essentially for rollbacks, as no process can rollback to a timestamp smaller than the GVT value [58]. This trend can be seen in the Tables 4.5 and 4.6, as the number of LPs increases there is decrease in the parallel execution time of GVT calculation function thus spending more time in sequential GVT calculation due to rollbacks.

Table 4.6 contains the functional level percentage execution time results for the parallel optimistic simulation with a different numbers of LPs. The total execution time of the simulation model with 1024 LPs was 24.7 seconds out of which the parallel execution part was 9.14 seconds. Similarly, for 524,288 LPs, the total execution time was about 6 hours with parallel exe-

cution part near to 1.38 hours. Similarly, the GVT computations (that need to be performed in a sequential manner) are used to find a time in the past for which it is guaranteed that there will be no rollbacks. For this reason, in Tables 4.5 – 4.6, it can be seen that there is a considerable increase in the computation time of the tw_gvt_step function as the number of LPs increases. Increasing the frequency of rollbacks increases the amount of overhead due to reverse computation. In parallel discrete event simulation, the reverse computation is used for reducing the amount of state saving (that is very memory consuming). For energy constrained systems, the excessive amount of rollbacks can cause a longer execution time. Sometimes, it results in a cascading effect – the primary rollbacks cause secondary rollbacks transitively, to reverse the effect of previously sent messages [59]. On the other hand, in conservative simulation, the causality errors are avoided by performing more LBTS computations and this is the reason for a better execution time in the conservative approach.

4.10 Discussion - ROSS Framework

The in-depth instrumentation results for the serial, parallel conservative and optimistic approaches have been presented in this section. Table 4.7 summarize the average results obtained for all the LPs for each technique. The conservative approach is shown to perform better in most of the parameters. The results can help the research community to determine what are the critical parameters that need to be focused on while designing PDES frameworks for mobile platforms. A serial execution of the simulation model consumed fewer resources than the other approaches but it has a longer execution time. Thus, it is possible to conclude that this technique is not efficient and ef-

Table 4.7: Summary of Results (average) for Serial, Parallel Conservative and Parallel Optimistic Algorithms

Simulation Algorithm	**Execution Time (Seconds)**	**Memory Consumption (MBs)**	**Efficiency (%)**	**Wait time (Seconds)**	**Average CPU Usage (%)**	**Power Consumption (Watt)**	**Energy Consumption (J)**	**CPU Temperature (C)**
Serial Approach	4917.07	25.75	100	0.005	25	25.97	74859	45.23
Conservative Approach	2756.09	28.63	100	796.05	73.25	57.77	76785	68.21
Optimistic Approach	4035.26	29.12	99.58	1572.75	77.00	56.18	108241	68.51

fective for usage on mobile devices. The serial execution model can be used effectively only for models that are less computation intensive. For parallel conservative and optimistic simulation models, a good strategy can be to migrate resource extensive functions (or modules) to cloudlets for their execution. On this aspects, the execution time of core functions are reported in Tables 4.4, 4.5 and 4.6. Parallel optimistic approach provides the most opportunities in this regard as there are more modules consuming higher execution times as compared with conservative approach.

In our view, the detailed analysis of the various PDES execution models is the first step towards the design and implementation of new simulation frameworks for mobile handheld devices. Moreover, it is also necessary to determine the migration cost of each simulation module before moving the compute-intensive code at cloudlets. In fact, the decision of migrating a whole or a partial module is based on a number of factors and some of them can be unknown or unpredictable before running the simulation or at the initial stage. For this reason, heuristic approaches for the dynamic (and adaptive) re-allocation of these modules is an area that require further exploration.

Chapter 5

SEECSSim – Proposed Simulation Suite

In this section, a new simulation framework is presented that is specifically aimed for mobile devices that include support for various synchronization algorithms. *SEECSSim* is designed for running efficiently on handheld devices, more specifically SEECSSim version 1.0 supports Android devices.

5.1 Architecture of SEECSSim

The SEECSSim architecture is shown in Figure 5.1. At the top, there is the application layer, that supports users in building their own applications. The application layer communicates with the core functionalities of SEECSSim through an Application Programming Interface (APIs). To benchmark the proposed simulation framework, we rely on the PHOLD simulation model implemented at the application layer. The SEECSSim Simulation Engine (SSE) is the main module that manages all the PDES related tasks. These include data distribution, scheduling and synchronization algorithms

(e.g. time-stepped, synchronous conservative, asynchronous conservative and optimistic). The SSE includes the serial, conservative and optimistic synchronization approaches. The users can select one of these approaches through a configuration file before launching their application. Moreover, the modular design of SEECSSim allows users to easily incorporate their own synchronization algorithms. The data distribution module is designed to facilitate the data distribution format and communication between LPs that are executing on same or different mobile devices. One of the main characteristics of SSE is that it is resource-aware, this means that it keeps track of the available resources and its usage. The reported usage of the resources is taken in account by SSE to tune the simulation parameters such as aggregate communication, frequency of GVT computation, thus, it balance the usage. The communication layer provides abstraction to underlying available communication protocols such as TCP and UDP transmission protocols. Proposed simulation suite covers different types of distributed simulation algorithms that are; Time-stepped approach, ii). Synchronous conservative, iii). Asynchronous conservative and iv). Optimistic simulation. A brief description of the synchronization algorithms and techniques supported by SSE follows in this section.

5.2 Time-Stepped Model

In a time-stepped simulation model, time advances in fixed intervals, that is provided to all the logical processes or to the one requesting for the time interval value. At any interval in wall clock time all the logical processes are at the same logical time in the simulation. Typically, the entire simulation time is divided into time steps to equal size. This simulation model also requires

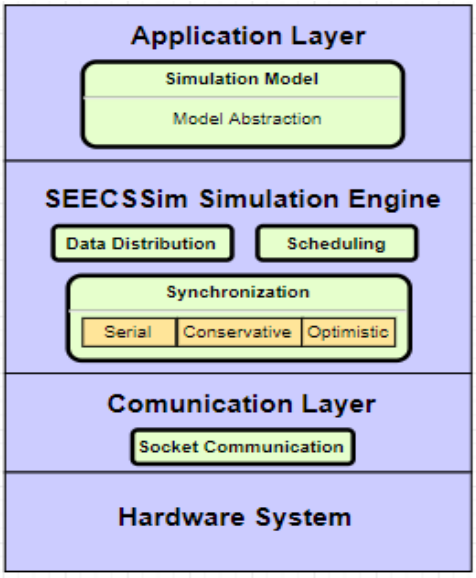

Figure 5.1: The SEECSSim Architecture

a barrier synchronization mechanism to ensure that all processes complete their execution in a time step before going forward to the next time time step by fixed interval Δt. Time-stepped algorithm is presented as Algorithm 5.1. In this algorithm, each process maintains a process logical time T_p, the current simulation time. This synchronization approach is most appropriate for models where simulation events are frequent and dense. However, in simulation models where simulation events are less frequent, performance may suffer as it might be difficult to define a correct time-step. For real-time interactive systems, the optimization is achieved by maintaining some information about the future events. The future event list is shared with the destination node which can generate time advancement request based on its future event list [60].

Algorithm 5.1 Time-Stepped algorithm

initialization $T_p = 0$
while $T_p < T_{end}$ **do**
  calculate state at T_p send self-generated output for T_p send output complete marker request advance to T_p update time T_p repeat;

  receive(message) **if** *message* $\neq$ *grant_advance*(T_{end}) **then**
    receive all messages receive input complete marker process input send output in response to this new input send output complete marker
  else
    Simulation end time is reached
  end
  $T_p \leftarrow T_p + \Delta t$
end

5.3 Synchronous Conservative Model

There are multiple centralized and decentralized algorithms in synchronous conservative approach that implement global synchronization mechanism. These algorithms include Distributed snapshot, Grid-based approach, Tree barrier, Broadcast and Centralized barrier algorithms. A centralized tree barrier approach is used, in which logical processes are organized as the nodes of a binary tree. Each LP processes events until it reaches a barrier point. Once an LP reaches a barrier point it sends a signal to its parent LP only if it has received signal from both child processes (if they exist). The processes continues until the root node receives the signal message. Once root node receives the signal, it knows that each LP has reached at barrier point. Once all LPs sync, root node (centralized node) broadcasts a message to release the barrier [61] . In this way, a centralized control is achieved. The algorithm for centralized tree barrier approach is presented as Algorithm 5.2.

Algorithm 5.2 Tree Barrier Algorithm

initialization $T_p = 0$ N_i = time of next event in LP_i
LA_i = lookahead of LP_i **while** $T_p < T_{end}$ **do**
| Enqueue NewMessages(InQ) **if** *IsMsg(InQ)* $\leq$*LBTS* **then**
| | // process safe events Msg = Dequeue NextMsg(InQ) Process Msg Barrier synchronization
| **else**
| | No safe Event, Compute LBTS
| **end**
end
Compute LBTS (Tree Barrier) LBTS = min(N_i + LA_i) **if** *Node* = *Root* **then**
| send new LBTS to all LPs
else
| send min(NextEventTime + LA)
| to Parent Process
end

5.4 Asynchronous Conservative Model

Chandy-Misra-Bryant (CMB) algorithm is implemented as an asynchronous conservative approach. In an asynchronous conservative model, LPs communicates through messages with increasing time-stamp. Each process maintains a separate queue for all of its incoming channels (C). Time-stamped messages guarantee that at each LP, the time-stamp of the last message received on an incoming link is the lower bound of any event message (E) that can be received later. However, deadlock can occur if the time-stamp of unprocessed events is greater than lower bound of an empty queue. In this situation, there is no surety that either an event is safe to be processed or not. However, null messages (special control messages) can be sent to other LPs in order to recover from deadlock condition. On receiving a null message, an LP can advance its local clock time, but null message will not cause an LP to change state variables or generate new events [62]. Chandy−Misra−Bryant (CMB) algorithm is presented as Algorithm 5.3. Each LP maintains a process

logical time T_p and a lower bound on every incoming channel T_{ch}. Receiving a null message with T_{nm} time-stamp from an LP, receiving LP is assured that there will be no messages with time-stamp smaller than the time-stamp of null message.

Algorithm 5.3 CMB Null Message Algorithm

initialization $T_p = 0$ **while** $T_p < T_{end}$ **do**
  forall C_{in} **do**
    Enqueue NewMessages(InQ) Update channel time T_{ch}
  end
  Select the incoming queue InQ with
  smallest channel time T_{ch}

  if *IsMsg(InQ)* **then**
    E_{im} = Dequeue NextMsg(InQ) $T_p = T_{im}$ process E_{im} Enqueue NewMessages(OutQ) **forall** C_{out} **do**
      ReleaseMesssagesUpTo($T_p + T_{cl}$) SendNullMessage($T_p + T_{cl}$)
    end
  else
    Simulation end time is reached
  end
end

5.5 Optimistic Model

Famous optimistic Time Warp (TW) synchronization mechanism is included in this proposed suite. In time warp mechanism, each LP starts event execution independently without coordinating with other other LPs. Thus, processes start to execute events without looking to synchronize initially until they detect that there is out of order execution or causality error. Once it is determined that there is some causality error, it is recovered using a roll-back mechanism. Processes re-execute the rolled back events if thet are not annihilated. On detecting the error, a process sends anti message to cancel or

roll back the execution of event message that is caused by out of order execution. Time Warp mechanism needs to keep the list of processed messages to send anti messages, this uses extravagant amount of memory. To reclaim the memory TW uses Global Virtual Time (GVT) mechanism to serve as a floor for the virtual times to which any process can ever again roll back. Every process reports its minimum time-stamp among all unprocessed events, partially processed events and anti-messages to the coordinating process. After GVT calculation, process reclaim the memory used to store processed events having time-stamp smaller than the time-stamp of GVT [63]. Algorithm for Time Warp mechanism is presented as Algorithm 5.4.

Algorithm 5.4 Time Warp Algorithm

```
initialization  T_p = 0
while T_p < T_end do
    unProcessedMsgQ.Enqueue ( buffer.outAll ) incomingMsg = unProcessedMsgQ.Dequeue
    switch incomingMsg.type do
        case GVT_Message: do
            / / GVT: Global Virtual Time  / / LVT: Local Virtual Time  submit LVT and wait For GVT Computation  submit LVT and wait For GVT Computation  do FossileCollection
        case anti_Message: do
            do process anti_Message
        end
        case normal_Message: do
            if anti_Message has already arrived then
                do Annihilation
            end
            if incomingMsg is straggler then
                do Rollback
            end
            else
                set the LVT to incomingMsg.getTimeStamp
                processedMsgQ.add(incomingMsg)
                LP.execute(incomingMsg.getEvent) do StateSaving foreach event in model.out do
                    outMsgQueue.Enqeue(newMessage(event)) sendMessages
                end
            end
        end
    end
    Simulation end time is reached
end
```

Chapter 6

Result & Discussions

6.1 Results – Benchmark Application Over Mobile Device

In order to benchmark the proposed SEECSSim framework, PHOLD has been implemented. The simulation topology of the PHOLD model is shown in Figure 6.1. The system specification of the mobile device used for running the benchmark tests is listed in Table 6.1.

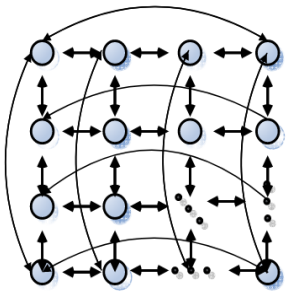

Figure 6.1: Simulation topology of the PHOLD benchmark

The results of Time-Stepped synchronization algorithm on the mobile

Table 6.1: Embedded System Specification

Parameters	Values
CPU	Quad-core 2.7 GHz
RAM	3GB
Storage	32GB
Operating System	Android
OS version	Marshmallow
Manufacturer	Samsung
Chipset	Qualcomm Snapdragon 805
Battery	Li-lon - 3220 mAh

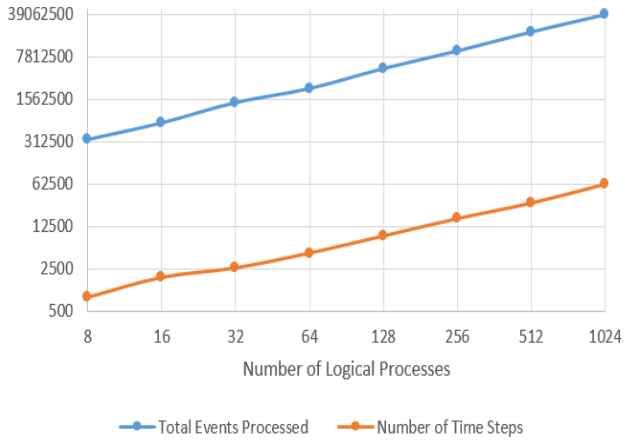

Figure 6.2: PHOLD with Time-Stepped synchronization on the mobile platform

platform with a variable number of LPs are shown in Figure 6.2. The timestep size is kept fixed for all the simulation runs. The figure shows the total number of events processed along with the total number of timesteps (Δt) that each simulation is able to complete.

The Tree Barrier synchronization algorithm work in a different way with respect to the Time-Stepped approach as it computes a new barrier point each time the LPs reach a barrier point. This means that the Tree Barrier takes slightly more time to complete execution due to all these LBTS computations that needs to be performed at each step. It is observed that the trend shown

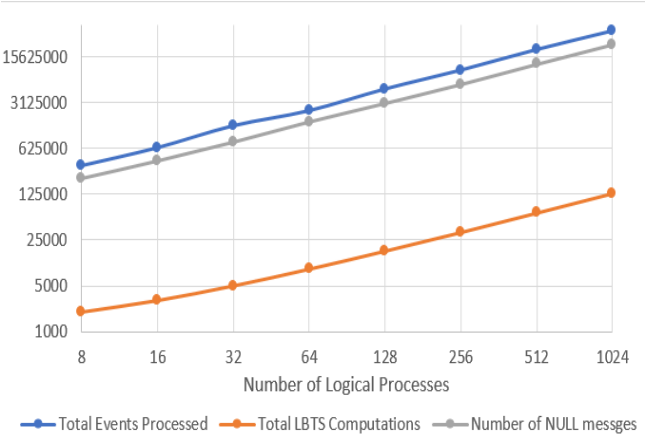

Figure 6.3: PHOLD with CMB NULL messages synchronization on the mobile platform

by the Tree Barrier (results not shown here) with an increasing number of LPs is almost the same as the Time-Stepped approach.

The results of PHOLD with Chandy−Misra−Bryant NULL message synchronization with a varying number of LPs are reported in Figure 6.3. The figure shows the total number of events processed, total number of LBTS computations along with the total number of NULL messages. The number of NULL messages is almost equal to the total number of events processed in the PHOLD execution. The total number of LBTS computations is very close to the LBTS computations in the Tree Barrier. The CMB performed better than the Tree Barrier and the Time Stepped approaches in terms of execution time.

The results of PHOLD with with Time Warp (TW) synchronization and a varying number of LPs are shown in Figure 6.4. The total number of events processed, the number of rollback events and total number of GVT computations are plotted for a varying number of LPs. With the increase of the LPs, the total number of rollback events increases gradually. This is

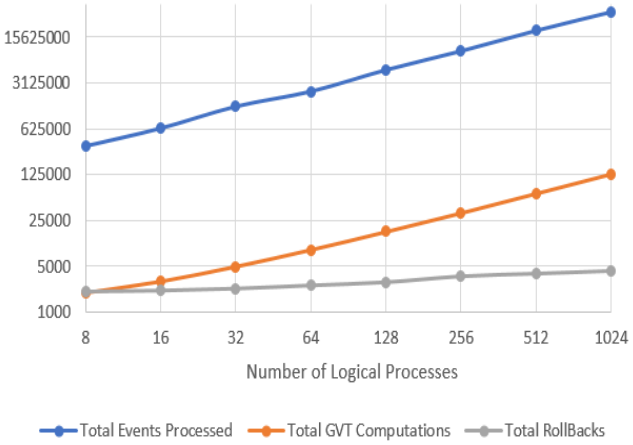

Figure 6.4: PHOLD with Time Warp synchronization on the mobile platform

due to the fact that large number of LPs generates an increasing number of events and the LP execution rate is slower than event input rate. Thus, the rollback rate is slowed down.

The results comparison in terms of event-rate for the PHOLD benchmark when run with all of the above-mentioned synchronization algorithms is reported in Figure 6.5. The event rate is defined as the total number of events processed in a unit time. Figure 6.5 shows that TW and CMB *NULL* message algorithm shows a better event rate as compared with the Tree Barrier and the Time-Stepped approaches. In TW, it is due to the optimistic behavior that, for most of the execution time, continues the processing of events without any need of synchronization. Similarly, the CMB keeps on executing the available events except when it needs to exchanges NULL message to get the value of the LBTS of an unknown link. In the case of Tree Barrier and Time-Stepped, most of the computing time is consumed in defining the LBTS or processing synchronization respectively.

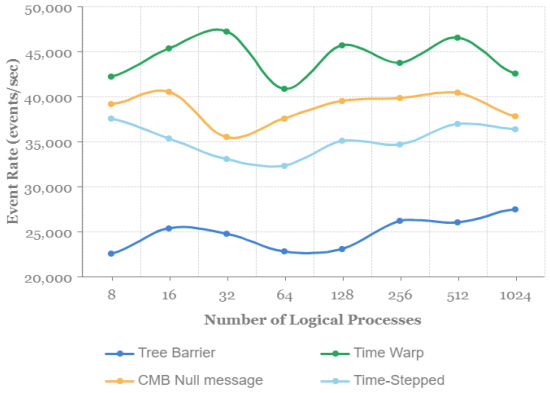

Figure 6.5: Event rate for different synchronization algorithms

6.2 Results – Mobile Device Resource Utilization

This section reports the resource utilization of proposed SEECSSim framework when run on a mobile platform. In comparison with traditional desktop or server systems, the handheld devices provide a limited amount of computational resources. For this reason, the completion time of simulations run on handheld devices usually increases as compared with traditional systems. It is thus not fair to compare the traditional execution architectures with handheld devices considering only the amount of time that is necessary to complete the simulation runs. A more comprehensive approach needs to take in account both the execution time and the energy consumptions of the simulations. Our goal is to thus analyze the CPU usage, memory consumption, energy consumption and the amount of time spent to complete each simulation while using different supported synchronization algorithms. The results also include the total number of events processed, the number of LBTS/GVT computations and the total execution time. The Trepn profiler tool is used

for measuring the power consumption and the performance of the different synchronization algorithms.

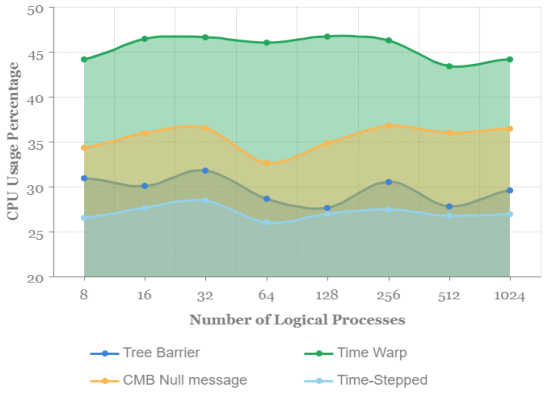

Figure 6.6: Average CPU usage for different synchronization algorithms

6.2.1 CPU Usage

The average CPU usage of the different simulation synchronization algorithms is reported in Figure 6.6. The Time-Stepped approach consumed the least CPU resources as compared with the other algorithms available in SEECSSim. On the other hand, the TW consumed a significant amount of CPU. The reason for the excessive CPU utilization in TW is that it needs to process a larger amount of events than the other approaches. Moreover, during the rollbacks, many events are pilled up in the input queues and thus more CPU work is required.

The GVT computations and fossil collect attempts also contribute to higher CPU utilization. The CPU usage for the CMB algorithm is lower as compared to TW but higher than Tree Barrier. This is caused by the number of *NULL* messages that are generated for processing each single event. Similarly, the look-ahead value, also has an impact on the performance of

CMB. The Time-Stepped approach performed better compared to others. However, it is important to note that the concept of CPU utilization for mobile devices is different than for desktop systems. For example, in a desktop computer a high CPU utilization can be termed as *better* CPU utilization since a consistent power supply is always available. This does not happen in mobile devices in which a high CPU usage means more energy consumption and therefore a faster battery depletion.

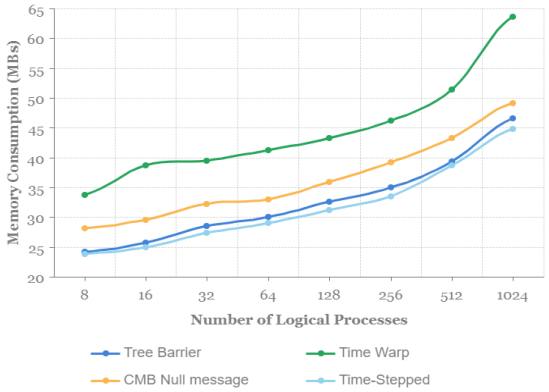

Figure 6.7: Memory consumption for different synchronization algorithms

6.2.2 Memory Consumption

Memory consumption is another important parameter that needs to be considered in embedded or mobile devices. As discussed in the previous section, the TW algorithm executes events using an optimistic behavior (that is without time synchronization). In order to perform rollbacks, each LP saves state variables and processed events. This state saving mechanism requires a significant amount of memory as compared with the other techniques discussed in this work. In fact, the other approaches consume a lower amount of energy and are very close to each other in terms of memory consumption. The

memory usage comparison for all the algorithms is shown in Figure 6.7.

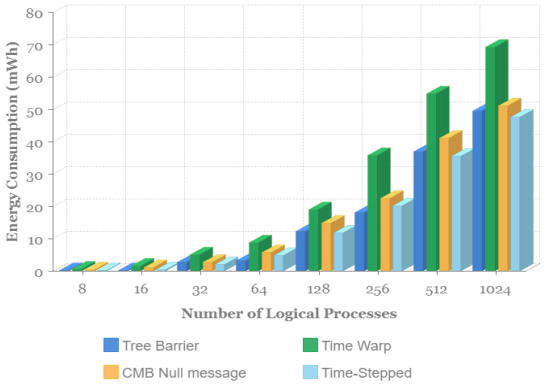

Figure 6.8: Energy consumption for different synchronization algorithms

6.2.3 Energy Consumption

Handheld devices are usually based on ARM processors that are designed for optimizing the energy consumption ([64] [65]) instead of the peak performance. These battery operated mobile devices are energy constrained, therefore one of the main design requirements is to minimize the total amount of energy consumption to complete a given computation task. The energy [1] consumed by executing the PHOLD benchmark with the different synchronization algorithms is presented in Figure 6.8.

In the figure, the amount of consumed energy is plotted using a logarithmic scale. It is important to note that reported energy consumption of the algorithms is relative to each other rather than their individual energy consumption. The energy is computed by multiplying power with time, therefore, if the execution time increases then its energy consumption also

[1] Here energy is given in milliwatt-hour, the watt-hour (Wh) is a unit of energy equivalent to one watt (1W) of power expended for one hour (1h) of time, thus, a milliwatt hour is 1/1000 Wh (symbolized mWh).

increases.

Time Warp is shown to be energy intensive as compared to the other algorithms. On the other hand, Time Stepped and Tree Barrier consume lower amount of energy while the energy consumption of CMB is nearly the same as the Time Stepped and Tree Barrier approaches.

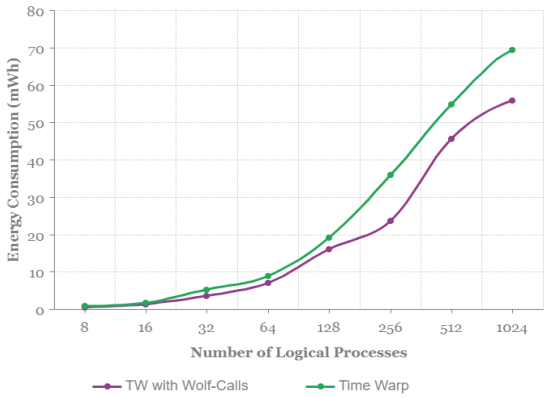

Figure 6.9: Energy Consumption – Time Warp vs. Time Warp with Wolf Calls

The energy consumption of the TW algorithm can be improved using specific techniques that help limit the number of rollbacks. One of such techniques is called Wolf Calls [66] whereby when a LP detects a straggler message, it sends a control message (Wolf Call) to all other LPs causing them to stop their message processing until the error is removed. An even better way for improving the performance of the Wolf Calls algorithm is to stop the processing only in LPs that would have been affected by the propagation of the causality error. Other, more advanced techniques such as lazy, re-lazy cancellation, and reverse computation can be employed. In the current version, the *SEECSSim* simulation framework includes support for Wolf Calls only.

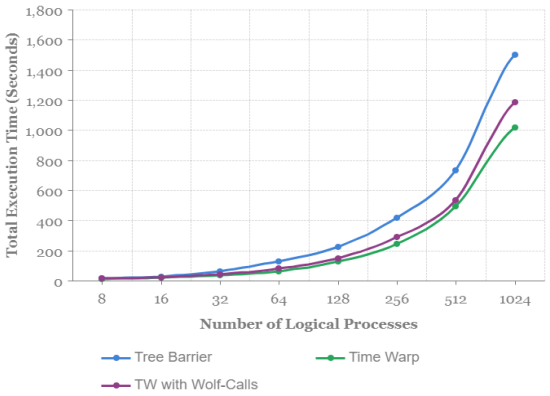

Figure 6.10: Total Execution Time – Tree Barrier, Time Warp and Time Warp with Wolf Calls

The results presented in Figure 6.9 suggest that the energy consumption of Time Warp with Wolf Calls enabled is improved considerably with an increasing number of LPs. It is pertinent to note that, in this case, the improvement in terms of energy consumption is achieved at the expense of the execution time (Figure 6.10). The execution time has increased when using the Wolf Calls but it is still better than the Tree Barrier approach.

6.2.4 Total Execution Time

The total execution time for different synchronization algorithms is shown in Figure 6.11. Results show that Time Barrier takes the most time to execute followed by Time Stepped while CMB NULL takes the least execution time. The total execution time for TW is less as compared to Tree Barrier and Time Stepped Algorithms due to its optimistic approach.

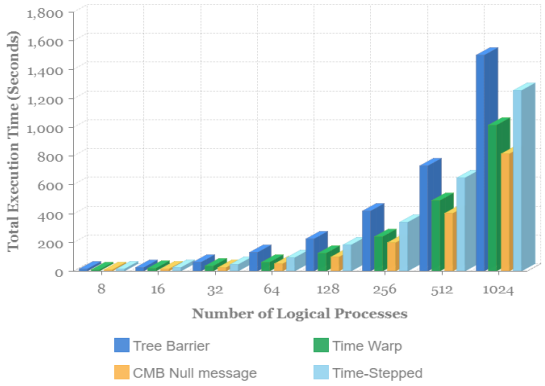

Figure 6.11: Total Execution Time for different synchronization algorithms

6.2.5 Function Level Execution Time

The objective of function level benchmark is to identify the modules that are taking most of the execution time. The execution time of every major module for all the mentioned algorithms is reported in Table 6.2–6.5. It can be seen that the Event-Handler (EH) module is taking the bulk of the execution time. Specifically, in Time-Stepped, the EH takes on average the 72.05 % whereas the same function in TW takes the minimum amount of time (58.375%) of all the approaches discussed in this section. Inside the EH implementation, there are a number of submodules taking more than 10% of the total execution time.

Looking at the execution time at the submodule level, it can be identified that TS_Event_Send for Time Stepped, TRB_Event_Send for Tree Barrier, CMB_Event_Send for CMB NULL and TW_Event_Send for Time Warp consumes the most percentage of the total execution time.

Table 6.2: Functional Level Execution Time for the Time Stepped Synchronization Algorithm

Functions (% Time)	**LP's**							
	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
main_run	97	97	98	97	99	99	98	97
TS_Init	0.3	0.4	0.5	0.5	0.7	0.8	0.4	0.5
TS_Scheduler	96.2	96	97	96	97.5	97.4	97	96
Event_Handler	73	71	72	71	73	73.4	72	71
TS_Event_Send	19	18	20	18	19	18	19	18
TS_Rand_SEEDS	12	13	13	12	12	13.3	12	12
TS_Event_New	13	13	11	13	13	13	13	13
RNG_Gen_Val	5	4	6	5	5	4.1	6	5
Others	9	10	9	9	8	10	9	9
TS_Net_Read	14	15	15	15	14	14	15	15
TS_Service_Queue	13	14	13	14	12	13	14	14
Others	1	1	2	1	2	1	1	1
TS_Event_Receive	15	14	13	15	16	16	13	15
Others	9.2	10	10	10	10.5	10	10	10
TS_Finalize	0.5	0.6	0.5	0.5	0.8	0.8	0.6	0.5
Initializations	3	3	2	3	1	1	2	3
Execution Time (sec)	**14**	**24**	**48**	**94**	**185**	**339**	**651**	**1255**

Table 6.3: Functional Level Execution Time for the Tree Barrier Synchronization Algorithm

Functions (% Time)	**LP's**							
	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
main_run	96	97	98	97	98	98	97	97
TRB_Initialize	0.7	0.3	1.2	0.4	0.4	0.5	0.5	0.4
TRB_Scheduler	94.5	96.2	97	96	97	97	96	96
Event_handler	70	73	73	71	72	72	71	71
TRB_Event_Send	17	19	18	18	19	20	18	18
TRB_Rand_SEEDS	12	12	13	13	12	13	12	13
TRB_Event_New	13	13	13	13	13	11	13	13
RNG_en_val	5	5	4	4	6	6	5	4
Others	8	9	10	10	9	9	9	10
TRB_net_read	14	14	14	15	15	15	15	15
TRB_Dqueue	12	13	13	14	14	13	14	14
Others	2	1	1	1	1	2	1	1
TRB_event_receive	16	15	16	14	13	13	15	14
Others	10.5	9.2	10	10	10	10	10	10
TRB_Finalize	0.8	0.5	1.8	0.6	0.6	0.5	0.5	0.6
Initializations	4	3	2	3	2	2	3	3
Execution Time (sec)	**16**	**27**	**63**	**129**	**224**	**418**	**734**	**1501**

Table 6.4: Functional Level Execution Time for the CMB NULL Messages Synchronization Algorithm

Functions (% Time)	**LP's**							
	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
main_run	99	98	99	98	99	99	99	98
CMB_init	1.2	1.5	2	1	1	1	2	1.2
CMB_scheduler	96	95	95	94	96	97	96	97
Event_handler	59	58	56	60	58	56	59	57
CMB_event_send	20	19	18	20	18	19	19	18
CMB_rand_SEEDS	13	14	15	14	12	14	14	13
CMB_event_new	12	11	10	10	13	9	11	13
rng_gen_val	5	4	5	6	6	6	6	4
Others	9	10	8	10	9	8	9	10
CMB_net_read	14	12	15	14	15	16	13	14
CMB_Dqueue	12	10	12	11	12	11	11	13
Others	2	2	3	3	3	4	2	1
CMB_LBTS_step	12	14	15	14	11	13	13	16
CMB_event_receive	7	9	8	10	8	9	7	16
Others	11	11	9	6	12	12	11	10
CMB_Finalize	1.8	1.5	2	2	2	1	2	1.8
Initializations	1	2	1	2	1	1	1	2
Execution Time (sec)	**8**	**14**	**27**	**51**	**98**	**201**	**403**	**817**

Table 6.5: Functional Level Execution Time for the Time Warp Synchronization Algorithm

Functions (% Time)	**LP's**							
	8	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
main_run	99	99	98	98	96	98	97	99
TW_Init	2	1	0.5	0.4	0.7	1.2	0.3	2
TW_Scheduler	96	96	97	97	94.5	97	96.2	95
Event_Handler	59	58	59	58	59	60	58	56
TW_Event_Send	19	18	20	19	17	18	19	18
TW_Rand_SEEDS	14	12	13	12	12	13	12	15
TW_Event_New	11	13	11	13	13	13	13	10
RNG_Gen_Val	6	6	6	6	5	4	5	5
Others	9	9	9	9	8	10	9	8
TW_Net_Read	13	15	15	15	14	14	14	15
TW_Dqueue	11	12	13	14	12	13	13	12
Others	2	3	2	1	2	1	1	3
TW_Gvt_Step	13	11	13	14	11	13	15	15
TW_Event_Receive	7	8	13	13	16	16	15	8
Others	11	12	10	10	10.5	10	9.2	9
TW_Finalize	2	2	0.5	0.6	0.8	1.8	0.5	2
Initializations	1	1	2	2	4	2	3	1
Execution Time (sec)	**12**	**20**	**35**	**62**	**126**	**243**	**493**	**1015**

6.3 Discussion

Table 6.6 shows comparison between different synchronization approaches supported in SEECSSim. The Time Warp is one of the most extensively used algorithms in distributed simulations. However, it is evident from summary results that it consumes the most of memory and energy as compared with other approaches, thus, it is not suitable for mobile and embedded devices. Moreover, CMB NULL approach performed better than TW on all parameters evaluated.

Table 6.6: Summary of the Average Resource Utilization for synchronization algorithms in SEECSSim

Simulation Algorithm	**Execution Time (Seconds)**	**Memory Consumption (MBs)**	**Average CPU Usage (%)**	**Energy Consumption (mWh)**
Synchronous Execution (Tree Barrier)	389.00	32.71	30.13	15.44
Time-Stepped	326.25	31.65	27.22	15.41
Conservative Approach (CMB NULL Message)	202.37	36.30	34.93	17.47
Optimistic Approach (Time Warp)	250.72	44.69	45.47	24.42

On the other hand, the Time-Stepped approach is the best among all the synchronization algorithms (except for the execution time where the CMB Null outperforms all others) but with a significant issue: due to its time-advancement mechanism, it cannot exploit true parallelism. It concludes that the CMB conservative algorithm is adequate in terms of execution time as well as energy consumption for adoption over mobile devices.

Chapter 7

Conclusion

It is worth noting that, in comparison with traditional systems, the handheld devices provide a limited amount of computational resources. Therefore, the completion time of the simulations on handheld devices usually increases in comparison to traditional systems. This is due to the fact that handheld devices uses ARM based processors that are designed for optimizing the energy consumption ([64] [65]). In other words, it is not fair to compare traditional execution architectures (e.g. desktops, servers) with handheld devices considering only the amount of time that is necessary to complete the simulation runs. A more comprehensive approach is to take in account both the execution time and the energy consumption of the simulation runs.

This work describes the instrumentation of a traditional distributed simulation framework, ROSS. The profiling results have provided valuable insight into the design and implementation of simulation frameworks for embedded and mobile devices. Based on our findings, A new distributed simulation framework is proposed called SEECSSim specifically designed for resource constraint devices. The simulator framework supports a number of synchronization algorithms, out of which CMB NULL conservative algorithm is

shown to perform adequately both in terms of execution time and energy consumption.

To the best of our knowledge, SEECSSim is the first open-source simulator framework that can help researchers to build simulations that can be efficiently executed on mobile devices. The flexible design of SEECSSim allows the researchers to incorporate their own simulation models and synchronization algorithms. As a future work, plans are to extend the support of SEECSSim for distributed simulations running on top of interconnected heterogeneous devices. In our view, this will open many new research direction for the PDES community.

7.1 Future Work

This thesis work has provided a full-fledged working PDES framework that includes all famous types of synchronization algorithms. It also provides an brief about how the resource utilization of an algorithm can be improved using techniques to reduce computations or overall execution time. This study will be able to help the researcher in selecting the most appropriate synchronization algorithms for simulation applications designed and running on embedded or mobile devices. As a future work, plan is to add other synchronization approaches to our simulation framework and to enhance the already available ones with other techniques that could be able to improve the cost of the algorithms when run on mobile and embedded systems. Moreover, formal verification and validation of the framework also needs be to carried out before making it available to the research community.

Bibliography

[1] Richard M Fujimoto, Rajive Bagrodia, Randal E Bryant, K Mani Chandy, David Jefferson, Jayadev Misra, David Nicol, and Brian Unger. Parallel discrete event simulation: The making of a field. 2017.

[2] Richard M Fujimoto. *Parallel and distributed simulation systems*, volume 300. Wiley New York, 2000.

[3] Asad Waqar Malik, Alfred J Park, and Richard M Fujimoto. An optimistic parallel simulation protocol for cloud computing environments. *SCS M&S Magazine*, 4:1–9, 2010.

[4] Kurt Vanmechelen, Silas De Munck, and Jan Broeckhove. Conservative distributed discrete event simulation on amazon ec2. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 853–860. IEEE Computer Society, 2012.

[5] Yihua Wu, Jian Cao, and Minglu Li. Private cloud system based on boinc with support for parallel and distributed simulation. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 1172–1178. IEEE, 2011.

[6] Eric Mikida, Nikhil Jain, Laxmikant Kale, Elsa Gonsiorowski, Christopher D Carothers, Peter D Barnes Jr, and David Jefferson. Towards pdes in a message-driven paradigm: A preliminary case study using charm++. In *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, pages 99–110. ACM, 2016.

[7] ROSS. Rensselaer's optimistic simulation system. `https://carothersc.github.io/ROSS`, 2017. Accessed March 20, 2017.

[8] Richard M Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[9] Lokesh Bajaj, Mineo Takai, Rajat Ahuja, Ken Tang, Rajive Bagrodia, and Mario Gerla. Glomosim: A scalable network simulation environment. *UCLA Computer Science Department Technical Report*, 990027(1999):213, 1999.

[10] James H Cowie, David M Nicol, and Andrew T Ogielski. Modeling the global internet. *Computing in Science & Engineering*, 1(1):42–50, 1999.

[11] Luciano Bononi, Michele Bracuto, Gabriele DAngelo, and Lorenzo Donatiello. Artis: a parallel and distributed simulation middleware for performance evaluation. In *International Symposium on Computer and Information Sciences*, pages 627–637. Springer, 2004.

[12] Gabriele DAngelo. The simulation model partitioning problem: an adaptive solution based on self-clustering. *Simulation Modelling Practice and Theory (SIMPAT)*, 70:1 – 20, 2017.

[13] Gabriele D'Angelo and Stefano Ferretti. Lunes: Agent-based simulation of p2p systems. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 593–599. IEEE, 2011.

[14] Allan I McInnes and Brian R Thorne. Scipysim: towards distributed heterogeneous system simulation for the scipy platform (work-in-progress). In *Proceedings of the 2011 Symposium on Theory of Modeling & Simulation: DEVS Integrative M&S Symposium*, pages 89–94. Society for Computer Simulation International, 2011.

[15] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia. The rome optimistic simulator: core internals and programming model. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 96–98. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[16] Yong Meng Teo and Yew Kwong Ng. Spades/java: object-oriented parallel discrete-event simulation. In *Simulation Symposium, 2002. Proceedings. 35th Annual*, pages 245–252. IEEE, 2002.

[17] Luca Toscano, Gabriele D'Angelo, and Moreno Marzolla. Parallel discrete event simulation with erlang. In *Proceedings of the 1st ACM SIGPLAN workshop on Functional high-performance computing*, FHPC'12, pages 83–92, New York, NY, USA, 2012. ACM.

[18] Gabriele D'Angelo, Stefano Ferretti, and Moreno Marzolla. Time warp on the go. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 242–248, ICST, Brussels, Belgium, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[19] Christopher D Carothers, David Bauer, and Shawn Pearce. Ross: A high-performance, low-memory, modular time warp system. *Journal of Parallel and Distributed Computing*, 62(11):1648–1669, 2002.

[20] Intel. Intel® soc watch. `https://software.intel.com/en-us/node/589913`, 2017. Accessed March 29, 2017.

[21] Allinea. Allinea-map. `http://www.allinea.com/products/map`, 2017. Accessed April. 2, 2017.

[22] Kalyan S Perumalla. Scaling time warp-based discrete event execution to 104 processors on a blue gene supercomputer. In *Proceedings of the 4th international conference on Computing frontiers*, pages 69–76. ACM, 2007.

[23] David W Bauer Jr, Christopher D Carothers, and Akintayo Holder. Scalable time warp on blue gene supercomputers. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, pages 35–44. IEEE Computer Society, 2009.

[24] Ilya Zhukov, Christian Feld, Markus Geimer, Michael Knobloch, Bernd Mohr, and Pavel Saviankou. Scalasca v2: Back to the future. In *Tools for High Performance Computing 2014*, pages 1–24. Springer, 2015.

[25] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S Müller, and Wolfgang E Nagel. The vampir performance analysis tool-set. In *Tools for High Performance Computing*, pages 139–155. Springer, 2008.

[26] Allen D Malony and Sameer Shende. Performance technology for complex parallel and distributed systems. In *Distributed and parallel systems*, pages 37–46. Springer, 2000.

[27] Allen D Malony, Sameer Shende, Robert Bell, Kai Li, Li Li, and Nick Trebon. Advances in the tau performance system. In *Performance analysis and grid computing*, pages 129–144. Springer, 2004.

[28] Sameer S Shende and Allen D Malony. The tau parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[29] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93. IEEE Computer Society, 2003.

[30] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, 2010.

[31] Xixhou Feng, Rong Ge, and Kirk W Cameron. Power and energy profiling of scientific applications on distributed systems. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 34–34. IEEE, 2005.

[32] Giuseppe Procaccianti, Luca Ardito, Maurizio Morisio, et al. Profiling power consumption on desktop computer systems. In *International Conference on Information and Communication on Technology*, pages 110–123. Springer, 2011.

[33] Luka Stanisic, Brice Videau, Johan Cronsioe, Augustin Degomme, Vania Marangozova-Martin, Arnaud Legrand, and Jean-François Méhaut. Performance analysis of hpc applications on low-power embedded platforms. In *Proceedings of the conference on design, automation and test in Europe*, pages 475–480. EDA Consortium, 2013.

[34] Nikola Rajovic, Alejandro Rico, James Vipond, Isaac Gelado, Nikola Puzovic, and Alex Ramirez. Experiences with mobile processors for energy efficient hpc. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 464–468. EDA Consortium, 2013.

[35] Shaoxiong Hua and Gang Qu. Approaching the maximum energy saving on embedded systems with multiple voltages. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 26. IEEE Computer Society, 2003.

[36] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S Nikolopoulos, Bronis R De Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 250–259. ACM, 2008.

[37] Charles Lively, Valerie Taylor, Xingfu Wu, Hung-Ching Chang, Chun-Yi Su, Kirk Cameron, Shirley Moore, and Dan Terpstra. E-amom: an energy-aware modeling and optimization methodology for scientific applications. *Computer Science-Research and Development*, 29(3-4):197–210, 2014.

[38] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and MT-C Lee. Instruction level power analysis and optimization of software. In *VLSI Design, 1996. Proceedings., Ninth International Conference on*, pages 326–328. IEEE, 1996.

[39] Richard M Yoo, Christopher J Hughes, Konrad Lai, and Ravi Rajwar. Performance evaluation of intel® transactional synchronization extensions for high-performance computing. In *High Performance Computing, Networking, Storage and Analysis (SC), 2013 International Conference for*, pages 1–11. IEEE, 2013.

[40] Kishore Kumar, Pusukuri Rajiv, Gupta Laxmi, and N Bhuyan. Shuffling: a framework for lock contention aware thread scheduling for multicore multiprocessor systems. In *Parallel Architecture and Compilation Techniques (PACT), 2014 23rd International Conference on*, pages 289–300. IEEE, 2014.

[41] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S Nikolopoulos, Bronis R De Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 250–259. ACM, 2008.

[42] Xixhou Feng, Rong Ge, and Kirk W Cameron. Power and energy profiling of scientific applications on distributed systems. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 10–pp. IEEE, 2005.

[43] Shaoxiong Hua and Gang Qu. Approaching the maximum energy saving on embedded systems with multiple voltages. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 26. IEEE Computer Society, 2003.

[44] Charles Lively, Valerie Taylor, Xingfu Wu, Hung-Ching Chang, Chun-Yi Su, Kirk Cameron, Shirley Moore, and Dan Terpstra. E-amom: an energy-aware modeling and optimization methodology for scientific applications. *Computer Science-Research and Development*, 29(3-4):197–210, 2014.

[45] Ryan Child and Philip A Wilsey. Using dvfs to optimize time warp simulations. In *Proceedings of the Winter Simulation Conference*, page 288. Winter Simulation Conference, 2012.

[46] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, and Mihai Alexandru. Energy-aware simulation with dvfs. *Simulation Modelling Practice and Theory*, 39:76–91, 2013.

[47] Richard M Yoo, Christopher J Hughes, Konrad Lai, and Ravi Rajwar. Performance evaluation of intel® transactional synchronization extensions for high-performance computing. In *2013 SC-International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11. IEEE, 2013.

[48] Kishore Kumar Pusukuri, Rajiv Gupta, and Laxmi N Bhuyan. Shuffling: a framework for lock contention aware thread scheduling for multicore multiprocessor systems. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 289–300. ACM, 2014.

[49] Deepak Jagtap, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Optimization of parallel discrete event simulator for multi-core systems. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 520–531. IEEE, 2012.

[50] Miguel A Erazo and Roberto Pereira. On profiling the energy consumption of distributed simulations: A case study. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 133–138. IEEE Computer Society, 2010.

[51] Jason Liu. The prime research, 2007.

[52] Richard M Fujimoto. Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 26(4):22, 2016.

[53] Yihua Wu, Jian Cao, and Minglu Li. Private cloud system based on boinc with support for parallel and distributed simulation. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 1172–1178. IEEE, 2011.

[54] Aradhya Biswas and Richard Fujimoto. Profiling energy consumption in distributed simulations. In *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, pages 201–209. ACM, 2016.

[55] Asad W Malik, Imran Mahmood, and Aakash Parkash. Energy consumption of traditional simulation protocol over smartphones: an empirical study (wip). In *Proceedings of the Summer Computer Simulation Conference*, page 23. Society for Computer Simulation International, 2016.

[56] SaBra Neal, Richard Fujimoto, and Michael Hunter. Energy consumption of data driven traffic simulations. In *Winter Simulation Conference (WSC), 2016*, pages 1119–1130. IEEE, 2016.

[57] Richard M Fujimoto, Michael Hunter, Aradhya Biswas, Mark Jackson, and SaBra Neal. Power efficient distributed simulation. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 77–88. ACM, 2017.

[58] Kenneth R Rouse, William Boff. *Organizational simulation.* Wily-Interscience, 2005.

[59] Kalyan S Perumalla. *Introduction to reversible computing.* Chapman and Hall/CRC, 2013.

[60] Karthik Shenoy. Techniques for optimizing time-stepped simulations. 2004.

[61] Rahul Garg, Vijay K Garg, and Yogish Sabharwal. Efficient algorithms for global snapshots in large distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):620–630, 2010.

[62] K. Mani Chandy and Jayadev Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on software engineering*, (5):440–452, 1979.

[63] David R Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3):404–425, 1985.

[64] Joshua Wyatt Smith and A Hamilton. Massive affordable computing using arm processors in high energy physics. In *Journal of Physics: Conference Series*, volume 608, page 012001. IOP Publishing, 2015.

[65] S Ryu and G Ganis. The proof benchmark suite measuring proof performance. In *Journal of Physics: Conference Series*, volume 368, page 012020. IOP Publishing, 2012.

[66] Vijay Madisetti, Jean Walrand, and David Messerschmitt. Wolf: A rollback algorithm for optimistic distributed simulation systems. In *Simulation Conference Proceedings, 1988 Winter*, pages 296–305. IEEE, 1988.

[67] Richard M. Fujimoto. Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 26(4):22, 2016.

[68] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, 2010.

[69] Canturk Isci and Margaret Martonosi. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 93. IEEE Computer Society, 2003.

[70] Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S. Mller, and Wolfgang E. Nagel. The vampir performance analysis tool-set. *Tools for High Performance Computing*, pages 139–155, 2008.

[71] Allen D. Malony, Sameer Shende, Robert Bell, Kai Li, Li Li, and Nick Trebon. *Advances in the TAU performance system*, pages 129–144. Springer, 2004.

[72] Sameer S. Shende and Allen D. Malony. The tau parallel performance system. *The International Journal of High Performance Computing Applications*, 20(2):287–311, 2006.

[73] Ilya Zhukov, Christian Feld, Markus Geimer, Michael Knobloch, Bernd Mohr, and Pavel Saviankou. *Scalasca v2: Back to the future*, pages 1–24. Springer, 2015.

[74] Richard M Fujimoto. Performance of time warp under synthetic workloads. 1990.

[75] Fahad Maqbool, Syed Meesam Raza Naqvi, and Asad W. Malik. Why to redesign pdes framework for smart devices: An empirical study. In *Proceedings of the Summer Simulation Multi-Conference*, SummerSim '17, pages 20:1–20:11, San Diego, CA, USA, 2017. Society for Computer Simulation International.

[76] A. W. Malik, R. Fujimoto, and A. Park. An optimistic parallel simulation protocol for cloud computing environments. *Simulation Magazine, Society for Modeling and Simulation, International*, 1(3):1–9, Oct 2010.

[77] Miguel A. Erazo and Roberto Pereira. On profiling the energy consumption of distributed simulations: A case study. In IEEE, editor, *IEEE/ACM Intl Conference on Cyber, Physcial and Social Computing*, pages 133 – 138, Hangzhou, 2010. IEEE.

[78] Asad W. Malik, Imran Mahmood, and Aakash Parkash. Energy consumption of traditional simulation protocol over smartphones: an empirical study. In *Proceedings of the summer Computer Simulation Conference- SCSC 2016, ISBN: 978-1-5108-2424-9*, number 23, pages 23:1 – 23:4. ACM, 2016.

[79] Ryan Elmore, Kenny Gruchalla, Caleb Phillips, Avi Purkayastha, and Nick Wunder. Analysis of application power and schedule composition in a high performance computing environment. Technical report, NREL (National Renewable Energy Laboratory (NREL), Golden, CO (United States)), 2016.

[80] Osman S Unsal. *System-level power-aware computing in complex real-time and multimedia systems*. PhD thesis, University of Massachusetts Amherst, 2003.

[81] Richard Biswas, AradhyaFujimoto. Energy consumption of synchronization algorithms in distributed simulations. *Journal of Simulation*, 2016.

[82] Michael J Quinn, Michael JQuinn. *Parallel computing*. McGraw-Hill, 1994.

[83] Aradhya Biswas and Richard Fujimoto. Profiling energy consumption in distributed simulations. In *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, pages 201–209. ACM, 2016.

[84] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. In *Technologies for wireless computing*, pages 139–154. Springer, 1996.

[85] Ryan Child and Philip A Wilsey. Using dvfs to optimize time warp simulations. In *Proceedings of the Winter Simulation Conference*, page 288. Winter Simulation Conference, 2012.

[86] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, and Mihai Alexandru. Energy-aware simulation with dvfs. *Simulation Modelling Practice and Theory*, 39:76–91, 2013.

[87] Miguel A Erazo and Roberto Pereira. On profiling the energy consumption of distributed simulations: A case study. In *Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 133–138. IEEE Computer Society, 2010.

[88] Jason Liu. The prime research, 2007.

[89] Yihua Wu, Jian Cao, and Minglu Li. Private cloud system based on boinc with sfupport for parallel and distributed simulation. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 1172–1178. IEEE, 2011.

[90] Kurt Vanmechelen, Silas De Munck, and Jan Broeckhove. Conservative distributed discrete event simulation on amazon ec2. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 853–860. IEEE Computer Society, 2012.

[91] SaBra Neal, Richard Fujimoto, and Michael Hunter. Energy consumption of data driven traffic simulations. In *Winter Simulation Conference (WSC), 2016*, pages 1119–1130. IEEE, 2016.

[92] Deepak Jagtap, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Optimization of parallel discrete event simulator for multi-core systems. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 520–531. IEEE, 2012.

[93] Richard M Fujimoto. {Performance of Time Warp under synthetic workloads}. 1990.

[94] Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu-an Chen, Xiang Zeng, Jay Martin, and Ha Yoon Song. Parsec: A parallel simulation environment for complex systems. *Computer*, 31(10):77–85, 1998.

[95] Intel. Intel® vtune amplifier. `https://software.intel.com/en-us/intel-vtune-amplifier-xe`, 2017. Accessed April. 2, 2017.