SEECSSim: A toolkit for Parallel and Distributed Simulations for Mobile Devices

Fahad Maqbool[1], Asad W. Malik[1,2], Syed Meesam Raza Naqvi[1], Nadeem Ahmed[1], Gabriele D'Angelo[3] and Imran Mahmood[1]

[1]School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad, Pakistan.
[2]Department of Information Systems, Faculty of Computer Science and Information Technology, University of Malaya, Malaysia.
[3]Department of Computer Science and Engineering (DISI), University of Bologna, Italy

ARTICLE HISTORY
Preprint: This article has been accepted for publication in Journal of Simulation, published by Taylor Francis. Acceptance date: Nov 29, 2019.

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 paper, the 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, we 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, we propose a new simulation framework 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 proposed framework dynamically manages simulation on devices and also perform process migration to optimize the use of resources. The performance of SEECSSim has been studied using a well-known simulation model (i.e. PHOLD) for different synchronization algorithms.

KEYWORDS
Distributed computing; Mobile computing; Performance analysis; Simulations.

CONTACT Asad W. Malik Email: asad.malik@seecs.edu.pk

[1]available on request

1. Introduction

The field of Parallel Discrete Event Simulation (PDES) has evolved significantly since its inception (Fujimoto et al., 2017a). It still remains an active area of research due to its employability in numerous applications in domains such as military, manufacturing, communication networks, computer systems, and road traffic systems etc. Traditional sequential simulations have the drawback of requiring a significant amount of time for completing their execution making these unsuitable for simulating large complex scenarios. On the other hand, the evolution of parallel and distributed simulation systems has permitted the usage of high-performance computing infrastructures to efficiently execute complex simulations models (Fujimoto, 2000). In this case, the simulation processes are executed on a parallel/distributed architecture whereby each simulation process executes a part of the simulation model while communicating through time-stamped messages. These messages are used for transporting the events that drive the simulation execution. The parallel/distributed execution obtained using this kind of architectures enables multiple simulation events to be run at the same time on multiple computing systems.

As with many other fields, the parallel and distributed simulations has been greatly influenced by emerging technologies. These technologies include massive parallel systems, Cloud computing, GPU computing, embedded computing systems and sensor networks. In addition, with the inception of Internet-of-things (IoT), heterogeneous devices have become an integral part of the grid network. Any computing device can thus become a part of a network, share its computing resources and store data. Consequently, the available computation platforms have changed as compared to the conventional cluster environment. This advancement in available platforms introduces new challenges for the PDES research community. For example, the workload on a single node in the cloud environment, where PDES process is competing with other CPU intensive processes, can significantly slowdown the whole simulation process that is running on top of the distributed architecture. Moreover, the context switching and OS process scheduling policies can also affect the performance of PDES over the Cloud environment (Malik et al., 2010; Vanmechelen et al., 2012; Wu et al., 2011a).

The existing PDES protocols/frameworks are very well tested but designed for cluster-based system configuration; whereas, there is a need to ascertain the suitability of the existing frameworks/protocols on hand-held mobile devices. Most of the sensors and smart phones being resource constrained, cannot store a large amount of data and perform heavy computations. Further, sending and receiving data also incurs a cost in terms of energy and network usage. The execution of large-scale distributed simulations over mobile and embedded devices thus opens new research challenges that need to be explored. In our view, in order to design an efficient simulation framework for mobile devices, the best approach is to start with the analysis of existing frameworks. It is imperative to accurately estimate the power and energy consumption of such frameworks to identify the critical parameters and modules that needs further redesign/optimizations. This profiling can help identify modules that require a large amount of resources to be executed. These resource hungry modules, for example can be hosted on Cloud/cloudlets and accessed through simulation-as-a-service model. Therefore, a careful analysis of the traditional PDES frameworks is required before attempting their porting on hand-held devices. This research work is divided in two distinct parts. Firstly, we have performed profiling of a commonly used PDES framework the Rensselaer's Optimistic Simulation System - ROSS, with the aim to measure its resource utilization in terms of power, CPU usage, energy and memory consumption while using a benchmark application. To obtain simulation results that are causally correct i.e. exactly the same as a sequential execution, synchronization algorithms must be used. In order to synchronize the processes involved in a PDES, the existing state-of-the-art synchronization mechanisms are categorized as (1) conservative or (2) optimistic approach. In the conservative approach, the causality errors are strictly avoided by applying strategies to determine when it is safe to process an event. Whereas, in the optimistic approach, the simulation continues until causality error is detected and then the error is handled by using a rollback mechanism that implements a state recovery and later re-execution of the rollback events. The instrumentation of ROSS framework is performed and results are obtained from serial, parallel conservative and parallel optimistic approaches when running on a desktop computing environment. The results allow us to better understand the resource utilization of the different simulation approaches. This would eventually help in deciding what synchronization approach is more appropriate for running PDES on hand-held embedded devices. Moreover, another objective is to precisely identify the resource hungry modules that are candidates to be ported on cloudlets while the traditional frameworks are still responsible for the core functionalities. Accurately identifying these modules allows simulations to be adapted to hand-held devices with little modifications.

In second part of this paper, we present *SEECSSim* – a distributed simulation framework designed to work on mobile and embedded devices as an extension of PDES framework instrumentation (SEECSSim can be made available on request). The proposed SEECSSim framework includes support for the state-of-the-art synchronization algorithms such as Chandy−Misra−Bryant (CMB) NULL message algorithm, Time-Stepped (TS), Tree Barrier (TB), Time Warp (TW) and Time Warp with Wolf (TWW) algorithm. We analyze the efficiency of the proposed framework for these algorithm in terms of execution time, CPU utilization, memory usage, energy consumption and event rate. We show that a properly selected synchronization algorithm can exploit the true potential of PDES over heterogeneous networks comprising of mobile and embedded devices. This paper will serve as a guideline for the PDES community to help them in selecting the right algorithms for running simulations on embedded systems while keeping in mind the resource constraints of such devices.
The rest of the paper is organized as follows. Section II covers the PDES synchronization models, literature review and background information are provided in Section III and IV. Section V covers the Instrumentation of the Rensselaer's Optimistic Simulation System. The motivation behind SEECSSim and proposed design is presented in Section VI and VII. Further, Section VIII covers the results and discussions. Finally, Section IX and X presents the limitations and conclusion.

2. PDES Synchronization Models

A brief description of the synchronization algorithms and techniques is covered in this section.

Time-Stepped Model – In a time-stepped simulation model, the simulated time advances at fixed time intervals (Δt). At any interval in the wall clock time, all the logical processes are at the same logical time T_p in the simulation. This simulation model requires a synchronization barrier mechanism to ensure that all processes complete their execution in a specific timestep before going to the next one. This approach is most appropriate where simulated events are frequent and dense. On the other hand, in simulation models in which the simulation events are less frequent, the simulator

performance may suffer as it might be difficult to define a correct timestep size. For real-time interactive systems, a possible optimization is achieved by maintaining some information about the future events. The future event list can be used to generate time advancement requests (Shenoy, 2004).
Synchronous Conservative Model – The synchronous conservative approach can be implemented using many different centralized and decentralized algorithms that implement a global synchronization mechanism e.g., distributed snapshot, grid-based approach, tree barrier, broadcast and centralized barrier algorithms. SEECSSim includes a centralized tree barrier approach, in which the LPs are organized as a binary tree. Each LP processes the events until it reaches a barrier point. At the barrier, it sends a signal to its parent process. At this point, the parent process forwards the signal only if it has received a signal from both children processes (if they exist). This processes continues until the root node receives the signal message. Once the root node receives the signal then it knows that every LPs has reached a barrier point. Once all the LPs are synchronized then the root node (that is a centralization point) broadcasts a message to release the barrier (Garg et al., 2010). In this way, a centralized control over the parallel/distributed architecture is achieved.
Asynchronous Conservative Model – In an asynchronous model, each LP maintains a local time and separate queue for all of its incoming channels. The 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 that can be received in future. 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 regarding the safe events. Therefore, NULL messages (i.e. special control messages) need to be sent to other LPs in order to determine which are the safe events. Receiving a NULL message with time-stamp from a LP, the receiving LP is assured that there are no messages with time-stamp smaller than the time-stamp of the NULL message. On receiving a NULL message, a LP can advance its local clock time but the NULL message cannot cause a LP to change its state variables or generate new events (Chandy and Misra, 1979). To support the Asynchronous Conservative model, SEECSSim includes the Chandy−Misra−Bryant (CMB) algorithm.
Optimistic Model – The SEECSSim framework also includes an implementation of the optimistic Time Warp (TW) synchronization. In the Time Warp mechanism, each LP starts the event execution independently without any coordination with other LPs. The LP are able to recover from causality errors using a rollback mechanism. After the rollback to safe state, each LP re-executes all the rolled back events if these have not been annihilated. On detecting a causality error, a LP sends the anti-messages that are necessary to cancel or rollback the execution of event messages sent during the out of order execution. For this reason, every LP needs to keep a list of all the processed messages (the anti-messages that could be necessary in case of roll back). The storage of this list can consume an excessive amount of memory. To reclaim that memory, the Time Warp algorithm uses a Global Virtual Time (GVT) mechanism. The goal of the GVT computation is to compute a floor to the virtual times. After each GVT calculation, each LP reclaims the memory used to store processed events having a time-stamp that is smaller than the time-stamp of GVT (Jefferson, 1985).

3. Literature Review

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 we discuss some related contributions for performance analysis and energy-aware simulation on HPC systems as well as mobile and embedded systems.

Over the years, many profiling tools have been developed to gauge the power consumption at a functional level. Tuning and analysis utilities, such as (Zhukov et al., 2015), (Knüpfer et al., 2008), (Malony and Shende, 2000), (Malony et al., 2004) and (Shende and Malony, 2006) provide support for instrumentation and performance visualization of parallel applications. Isci et al. (Isci and Martonosi, 2003) presented an approach to estimate the power consumption using performance logs. Similarly, the authors of (Ge et al., 2010) provided a framework called PowerPack that can be used for energy profiling and analysis of parallel applications on multi-core processors. The authors of (Feng et al., 2005a) show that power profiles always correspond to the characteristics of the application and increasing number of nodes results in more power consumption; however, it does not always result in better performance. The authors of (Procaccianti et al., 2011) analyze the power consumption related to software components. Their analysis shows that in different software scenarios, power consumption on a general-purpose computer system can vary from 12% to 20%. The authors of (Stanisic et al., 2013) and (Rajovic et al., 2013) focus on low power embedded systems to analyze and benchmark HPC applications for their energy consumption.

There is substantial amount of work that is available in power-aware computing; various techniques have been deployed to reduce the power consumption such as (Hua and Qu, 2003a), (Curtis-Maury et al., 2008a) and (Lively et al., 2014a). Computing the energy consumed by every machine instruction is an approach that can be followed to profile the energy consumption of all the functional components. For example, (Tiwari et al., 1996) discuss the functional level profiles for energy consumption that can be obtained through computing the energy consumed by each machine instruction. However, the proposed work is only limited to function level energy marking. Whereas, communication between processes is a major contributor in a parallel and distributed simulation. In a distributed simulation, different techniques are used to reduce the communication delay between processes (Yoo et al., 2013a). One of such approaches is to utilize the different cores available in a physical system (Kumar et al., 2014). However, the synchronization algorithms generate an amount of overhead that is difficult to reduce.

Most of the existing work on energy-efficient computing is related to the HPC environment (Curtis-Maury et al., 2008b; Feng et al., 2005b; Hua and Qu, 2003b; Lively et al., 2014b); where different techniques are used for optimizing the energy usage, e.g. Dynamic Voltage and Frequency Scaling (DVFS), process migration, task consolidation and Dynamic Power Management (DPM). R. Child et al. (Child and Wilsey, 2012) explored the features of DVFS to enhance the the performance while reducing the power consumption by repeatedly reducing the operating frequencies of the CPU cores. The authors investigate energy efficiency through DVFS while the Time Warp simulation algorithm is being executed. The proposed study is conducted over physical systems using a MPI version of the TW simulator.

Similarly, G. Tom et al. (Guérout et al., 2013) describe the integration of an energy-aware module to simulate the energy consumption of distributed systems. The authors provide an overview on energy-aware simulation and described the DVFS simulation tools used for obtaining simulations results 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 integrated the DVFS features in one of the cloud simulators.

The communication is a common performance bottleneck for fine-grained parallel applications (Yoo et al., 2013b). The authors in (Pusukuri et al., 2014) discuss the techniques used for improving the network performance by reducing lock contention and overlapping communications. In (Jagtap et al., 2012) are analyzed the performance of the ROSS simulation framework on different platforms and the multi-threaded implementation is compared with the MPI-based version. (Erazo and Pereira, 2010) present a case study used for profiling the energy consumption of distributed simulation, in this case tested on the PRIME simulator (Liu, 2007). The authors conclude that using more nodes to achieve parallelism results in a significant increase in the energy consumption.

In (Fujimoto, 2016), Fujimoto describes the main future research challenges in PDES. Among them large-scale simulation of complex networks, exploiting GPU's, Cloud computing exploitation, composable simulation and energy consumption of PDES are the latest research areas. In PDES, the energy consumption has not been widely investigated. The minimization of power consumption through a change in clock rate (DVFS) can eventually increase the overall energy required to complete the task. In fact, schemes such as DVFS are best for data centers and supercomputers. For embedded systems, the power-aware and energy-aware techniques are more important for design consideration.

Traditional simulations are designed to run on cluster environments. With the advancement of technology, the availability of infrastructure-as-a-service has provided flexible computing environments following a pay-as-you-go model. As a consequence, new PDES techniques have been proposed for such cloud architectures. In (Vanmechelen et al., 2012), the authors studied the execution of conservative synchronization algorithms over various configurations of Amazon EC2. The objective is to verify the suitability of cloud platforms for running distributed discrete event simulations. More specifically, the NULL messages that are part of the conservative algorithms play a significant role in the performance of the whole simulation system. For this reason, the authors tested various variations of synchronization algorithms such as Chandy−Misra−Bryant, time-out based NULL message sending, deadlock avoidance based on NULL messages, on-demand NULL messages, timeout protocols etc. The results show that the timeout and blocking protocol has the best performance in a cloud environment.

The Cloud is based on a multi-tenant paradigm, the effect on the execution of optimistic PDES is often a large number of rollbacks. This happens because the different nodes that are running the PDES are not equally loaded (e.g. number of running jobs). Moreover, some tasks require more computation whereas other need a large amount of communication. On these aspects, (Malik et al., 2010) proposed a PDES model for cloud environment for improving the performance of optimistic parallel simulation through dynamically defining the barrier points and there reducing the total number of rollbacks. The reported results show a significant gain in terms of performance over traditional optimistic simulation. Similarly, (Wu et al., 2011b) presented a BOINC based system for Cloud environment to execute parallel and distributed simulation over private cloud infrastructures. BOINC is a middleware developed for volunteer and grid computing. It is an open-source framework designed to support task distribution and result gathering in a client-server model.

In the context of distributed simulation over mobile and embedded devices, (Biswas and Fujimoto, 2016) discuss an approach based on power profiles. The energy consumption of simulation model, engine, computations, and communications is separated with the aim to measure the energy consumption of each aspect of the simulation. More in detail, the authors have presented a comparative analysis of the energy consumed by the Chandy-Misra-Bryant and the YAWNS algorithms. Similarly, (Malik et al., 2016) have analyzed the energy consumption of the Time Warp protocol when run on smartphones. Online distributed simulations (e.g. traffic prediction systems) requires a significant amount of energy. (Neal et al., 2016) analyze the energy consumption of data-driven traffic simulations on mobile devices. Since the online traffic prediction requires a significant amount of energy then understanding the energy consumption at various levels, helps in optimizing the use of resources. On this aspect, the authors presented an empirical investigation of modules such as data transmission, gathering and traffic computations. In (Fujimoto et al., 2017b), Fujimoto et al. presented a detailed work on power-efficient distributed simulation. The authors covered few conservative and optimistic synchronization algorithms along with a 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 have been conducted on multiple configurations such as the Jetson TK1 development board and a quad-core LG Nexus 5 smartphone.

In this paper, we present the instrumentation of an existing simulation framework ROSS. Based on the results obtained from this instrumentation, we have designed a new framework for distributed simulation running on mobile devices. The next section provides some background on our work, the description of the experimental setup for both the desktop and mobile configurations, the results that have been obtained and their discussion.

4. Background

In Table 1, we have listed some of the commonly used PDES frameworks. For this work, we have chosen to instrument Rensselaer's Optimistic Simulation System (ROSS) since it is well-known and widely used in the PDES community. A brief description of ROSS is provided in this section. Moreover, the software tools and the main techniques used for performing instrumentation and power consumption analysis are also discussed in this section.

Rensselaer's Optimistic Simulation System (ROSS) – ROSS is a high performance and extremely modular PDES system (Carothers et al., 2002) that is implemented using the Message Passing Interface (MPI). Its modular implementation, ability to use reverse computation, Kernel processes and implementation of Fujimoto's GVT (Global Virtual Time) algorithm makes it a state-of-the-art simulation system to perform experimental studies. In this study, we analyze the performance and the power consumption of the sequential, optimistic, and conservative synchronization approaches supported in ROSS simulations.

PHOLD: 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

Table 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 (Bajaj et al., 1999). 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 (Cowie et al., 1999).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 middle-ware for Parallel and Distributed Simulation (PADS) that can simulate complex systems (Bononi et al., 2004). 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 (D'Angelo, 2017). 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 (D'Angelo and Ferretti, 2011). 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 (McInnes and Thorne, 2011). 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 (Pellegrini et al., 2011). 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 (Teo and Ng, 2002). 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 (Toscano et al., 2012). 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 (D'Angelo et al., 2012). 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.

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.

Instrumentation Tools: We have employed various tools 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 (Intel, 2017) as well as profiling tool Allinea Forge Map (renamed as Arm Map) from Arm (Allinea, 2017) and Trepn. A more detailed description of each tool along with its usage follows.

4.0.1. 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.

4.0.2. 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. We have used VTune™ amplifier 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.

4.0.3. CPU Cores Temperature – Intel® SoC Watch

Intel SoC is a command-line utility designed by Intel (Intel, 2017) to study the temperature profiles of CPU cores. We used SoC to study the behavior of ROSS with an increasing load on each LP.

4.0.4. 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++ (Allinea, 2017). 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 bench-marking model (Bauer Jr et al., 2009; Perumalla, 2007) with a variable number of logical processes (LPs).

4.0.5. Energy, Memory Consumption for Android – Trepn Profiler

Profiler[2], 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,

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

Table 2. 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

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.

Table 3. 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

5. Instrumentation of the Rensselaer's Optimistic Simulation System (ROSS)

This section 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 4. 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, we have compared these algorithms 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.

Table 4. 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

5.0.1. 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 2, 3 and 4 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.

5.0.2. Memory Consumption

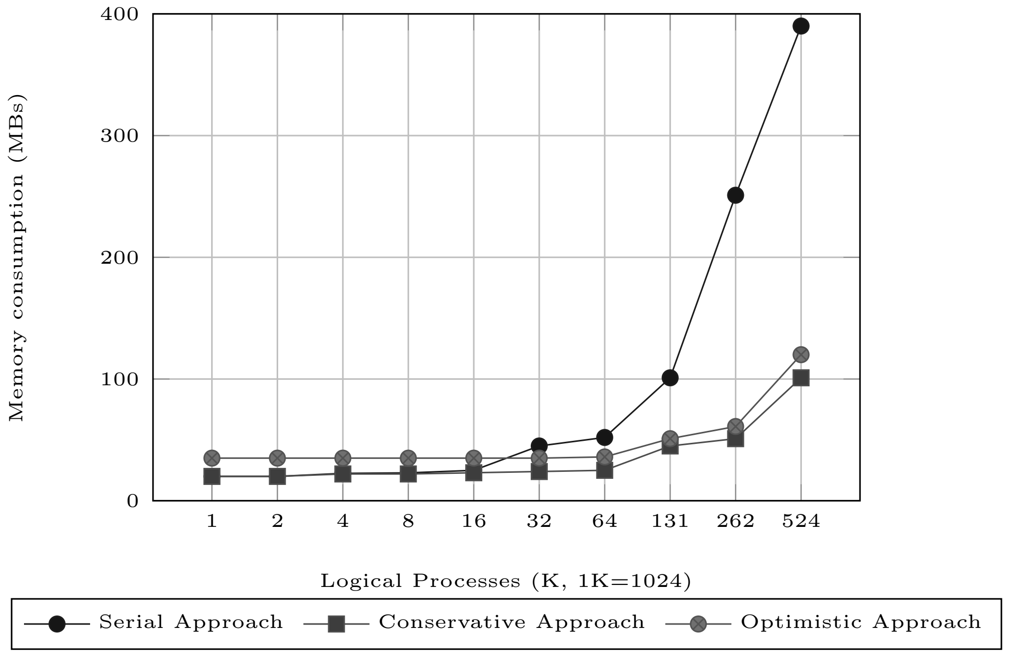


Figure 1.: Memory usage analysis

In this section, the memory usage results for the PHOLD execution on the ROSS framework are presented. Using 1024 LPs, the parallel conservative 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 1, we observe a similar trend 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 2, 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.

5.0.3. Efficiency, GVT computation, Fossil Collection and Rollbacks

These important metrics for simulation synchronization schemes are reported in Tables 2, 3 and 4. 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 the 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 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 (Rouse, 2005). 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 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 its 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 (Perumalla, 2013). 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.

5.0.4. 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 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.

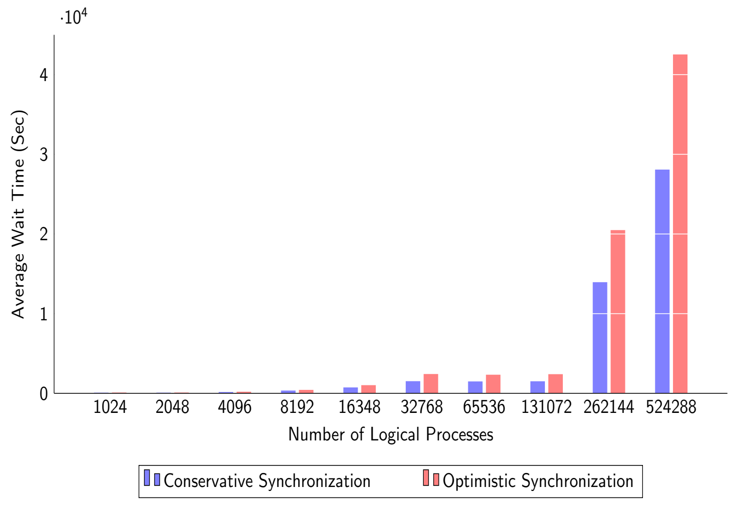


Figure 2.: Wait Time Analysis for PHOLD model

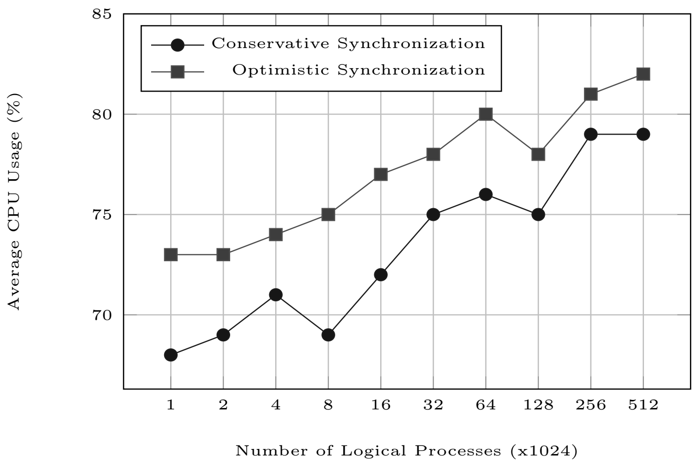


Figure 3.: Average CPU usage for PHOLD model

5.0.5. 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 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 of power and energy consumption are reported in Figure 4 and 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 1.

$$Power = Energy/Time \tag{1}$$

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

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 5 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

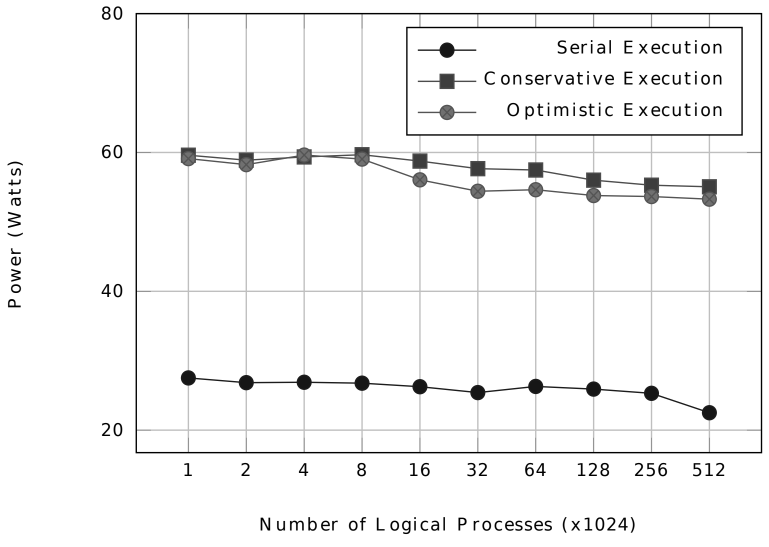


Figure 4.: Power consumption analysis

CPU but still consumes a large amount of energy due to its longer execution time (with respect to the other approaches).

5.0.6. 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 shows that for a low number of LPs in the simulation, the CPU power consumption 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.

5.0.7. CPU Temperature

The average temperature statistics for each CPU core, while the PHOLD simulation model is being executed, are presented in Figure 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 54.4°C and 71.8°C respectively. Similarly, in optimistic they are 55.5°C and 79.9°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

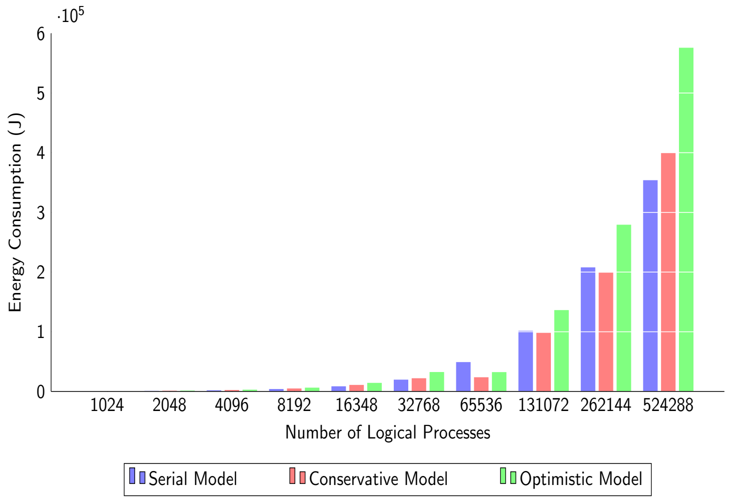


Figure 5.: Energy consumption analysis

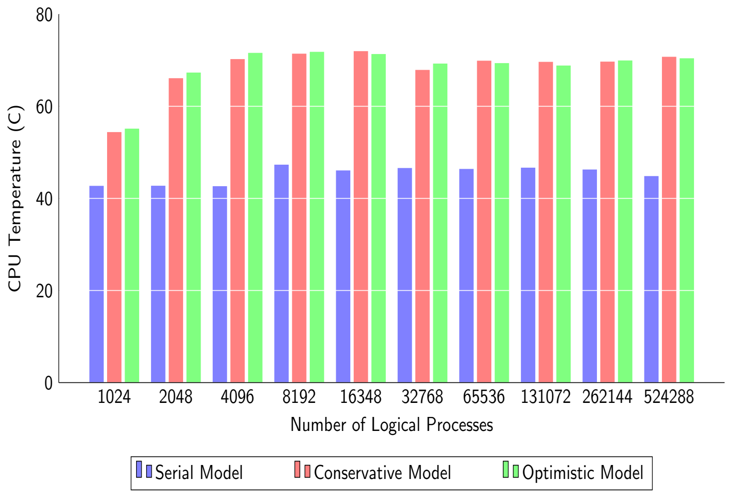


Figure 6.: CPU temperature statistics

Table 5. 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**

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

5.0.8. 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 5, 6 and 7. 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 5. 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 computation.

Table 6 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 be done in a sequential fashion essentially for rollbacks, as no process can rollback to a timestamp smaller than the GVT value (Rouse, 2005). This trend can be seen in the Tables 6 and 7, 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 7 contains the functional level percentage execution time results for the par-

Table 6. 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**	

Table 7. 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**	

allel 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 execution 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 6 – 7, 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 (Perumalla, 2013). 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.

Discussion - ROSS Framework – The in-depth instrumentation results for the serial, parallel conservative and optimistic approaches are presented in this section. Table 8 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 effective 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 5, 6 and 7. 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

Table 8. 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

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 requires further exploration. The different techniques used a different number of resources, after profiling ROSS, we have carefully designed the core modules of the proposed framework. The modular design of core modules allows us to migrate the module to nearby cloudlets. In the next section, we explain our motivation for adopting mobile devices for parallel and distributed simulations. So far no such framework exists that support parallel and distributed simulation over mobile devices.

6. Motivation behind SEECSSim

In embedded systems, the adoption of highly programmable microprocessors, easy network connectivity features and user-friendly interface supported by the operating system makes the embedded devices smarter compared to traditional fixed function and disconnected systems. The embedded system market has generated over 2 trillion dollers in revenue in 2015 which is expected to increase in next few years. Moreover, 14.5 billion microprocessor cores have been utilized by 2015 with compound annual growth rate around 15%. Moreover, in real life, the role of embedded systems has been increased. Embedded devices are used in crisis management systems, traffic prediction and etc. Further, a large number of devices available in every home, and office. Thus, the available connected devices can be used to share the resources.

Distributed simulations over embedded devices can be used to monitor physical systems. In real life, the use of unmanned aerial vehicles (UAVs) has been increased. The UAVs can be used to monitor or track vehicles, flood monitoring, situation after earthquake and spread of forest fire. Usually, the UAVs are equipped with sensors and have a wireless communication link. Let's assume that each UAV is assigned a particular geographical area to collect information. Through communication link, the UAVs can form a grid where based on the received information they can execute the distributed simulation to project the future state of the system. e.g. distributed simulation over UAVs can be used to predict the direction of a forest fire, and location of the fire in future. Moreover, it can also be used to reassign the areas to UAVs for better coverage and resource optimization.

Likewise, in case of traffic congestion, more UAVs can be assigned to particular intersection based on the future states. Moreover, there is a number of application where embedded systems can be utilized to predict the future state based on information received in real-time. Traditionally, sensors like UAVs are used to send the received information to a centralized computing facility, where the states are projected. However, moving the simulation over embedded devices is still emerging area and it offers several advantages. Such as it reduces the use of central computing resources and thus, removes the single point of failure. The communication delay between connected UAVs is lower if compared to sending the information to the central location. Thus, such schemes are more suited to delay-tolerant applications. Moreover, distributed implementation using embedded devices provides great scalability. The above mentioned near real-time systems are referred to as dynamic data-driven systems (DDDAS) (Darema, 2004). The DDDAS gather information, predict future state and reconfigure the system to optimize the resources. The DDDAS has been discussed for a number of applications (Madey et al., 2012). However, from the above discussion, it is clear that for such simulations, energy consumption is the most important concern for executing simulation over battery operated devices. Here our focus is on energy, and memory consumption over embedded mobile devices. SEECSSim (Maqbool et al. (2018)) is a novel framework designed to support distributed simulation on embedded devices. More detail on SEECSSim are presented in next section.

Algorithm 1 SEECSSim Framework ▷ Dynamically maintain resources and peer information

```
$PeerTable_i \leftarrow null$
$Res_i \leftarrow null$
$JoinNet_i \leftarrow null$
$syn_{algo} \leftarrow ConfigFile\ (CMB, TS, TW, TB)$
while ($Simulation != End$) do
    if ($JoinNet_i == False$) then
        $JoinNet_i \leftarrow call\ JoinNetwork()$
        $JoinNet_i \leftarrow True$
        $PeerTable_i \leftarrow build$
    end if
    if ($Res_i.UpdateInterval\ expired$) then
        $Res_i \leftarrow ResourceManager(..)$
        $Reset \leftarrow timer$
    end if
    if ($Res_i.usage \geq threshold.value\ OR\ null$) then
        $Simulation \leftarrow pause$
        $Peer_i \leftarrow Inform$
        $Generate \leftarrow Sharing\ Request$
        if ($Generate\ Request_i.accepted == true$) then
            $Send \leftarrow simulation_{dump}$
            $Peer_i \leftarrow NULL$
            $Res_i \leftarrow reset$
        end if
    end if
    if ($Generate\ Request_i.received == true$) then
        $Res_i \leftarrow check$
        if ($Res_i.isEnough == true$) then
            $send \leftarrow accept_{call}$ to $Peer_i$
        end if
    end if
    if ($Simulation \neq pause$) then
        $SynchronizationAlgo\ (syn_{algo}, e_i)$
    end if
end while
function SynchronizationAlgo($syn_{algo}$, $e_i$)
    $Process\ event \leftarrow e_i$
    $sim_t \leftarrow update\ time$
    $send \leftarrow newe_i$ to $Peer_i$
end function
```

In this section, we present a new simulation framework 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. The SEECSSim architecture is shown in Figure 7. 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). The SEECSSim framework acts as a middleware between the application and communication layer. At present, communication through UDP and TCP is supported. The user can select the communication mechanism through the configuration module available at the application layer. The core modules of the SEECSSim frameworks are the Resource

Algorithm 2 Memory Manager ▷ Manage and allocate memory

```
1: $Mem_{pool} \leftarrow null$
2: $timer \leftarrow \Delta$t
3: $Mem_{pool} \leftarrow GetMemory\ (Block_S, num_b)$
4: **function** MemoryManager($Req_i$, $Mem_{handle}$)
5:     **if** ($Req_i$ == memory request And $Mem_{pool} \neq null$) **then**
6:         $returnMem_{handle} \leftarrow Mem_{pool}$
7:     **else**
8:         return null
9:     **end if**
10:    **if** ($Req_i$ == release memory request ($Mem_{handle}$) ) **then**
11:        Add $Mem_{pool} \leftarrow Mem_{handle}.getMemory$
12:    **end if**
13:    **if** (timer expired or $Req_i$ == status) **then**
14:        $U_{ri} \leftarrow$ Update $Resource_{info}$
15:        $timer \leftarrow \Delta$t
16:        **if** ($U_{ri} \geq threshold.value$) **then**
17:            $return \leftarrow Exception$
18:        **end if**
19:    **end if**
20:    **if** ($Res_i$.usage request) **then**
21:        return $U_{ri}$
22:    **end if**
23: **end function**
```

Algorithm 3 Resource Manager ▷ Monitor CPU and battery usage

```
1: $U_{ri} \leftarrow null$
2: $timer \leftarrow \Delta$t
3: **function** ResourceManager($Req_i$)
4:     **if** (timer expired OR $Req_i$ == resource status) **then**
5:         $U_{ri} \leftarrow$ Acquire $battery_info$
6:         $U_{ri} \leftarrow$ Acquire $CPU_info$
7:         $U_{ri} \leftarrow$ Acquire $MemoryManager(Req_i.stat)$
8:         $timer \leftarrow$ reset $\Delta$t
9:         $return \leftarrow U_{ri}$
10:    **end if**
11: **end function**
```

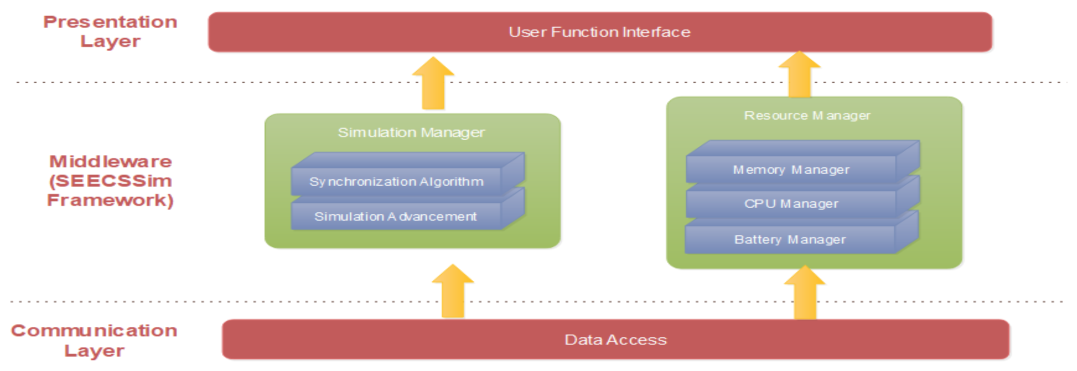


Figure 7.: The SEECSSim Architecture

Manager (RM) and the Simulation Manager (SM). Algorithm 1 shows the operations of the proposed framework. The simulation starts with processes on embedded systems join the simulation network, build the communication table. The communication table maintains the addresses of all the processes to send and receive messages. The resource manager keeps track of the available resources on embedded devices i.e. memory, CPU and battery. In case of any resource usage goes beyond the threshold value, the framework pauses the simulation and informs all the other processes. At the same time the framework sends a process migration request to all the processes. The device with maximum resources can accept the request and thus simulation resumes. Moreover, the other devices can join the simulation and they become the candidate host for the migration. The complete execution of the Memory and Resource Manager is shown in Algorithm 3 and Algorithm 2. To benchmark the proposed simulation framework, we rely on the PHOLD simulation model implemented at the application layer. The Simulation Manager (SM) provides the synchronization algorithms (e.g. time-stepped, synchronous conservative, asynchronous conservative and optimistic). The SM 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.

Causality constraint – In parallel and discrete event simulation, causality constraint is a highly critical factor to produce correct the results. However, using devices to execute without a proper mechanism to ensure causality can lead to incorrect results. In the proposed framework, SEECSSimn ensures that during processes migration from a resource constraint device to another must fulfill the causality constraint. In SEECSSim, when the device generates a request for process migration due to insufficient resources at the same time it includes all the process related parameters in its request. These parameters include remaining simulation execution time, GVT computation interval (for optimistic model), the memory usage of state variables, event history (to support rollback) and pending event list. These variables help the devices in decision making. Thus, to obey causality, on acceptance, all the pending event list along with the event history since last GVT is transferred. Similarly, in Algorithm 1 shows the process attributes sharing mechanism. These parameters help the re-execution from

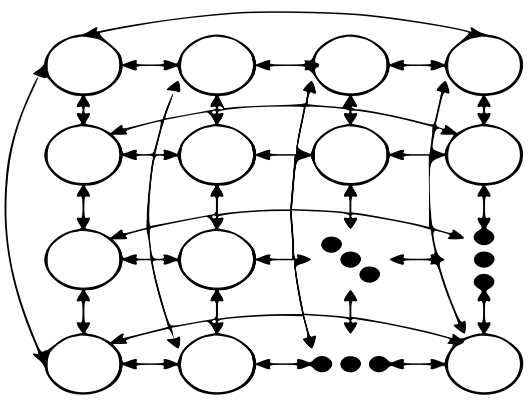


Figure 8.: Simulation topology of the PHOLD benchmark

Table 9. Embedded System Specification used for benckmark

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
# of devices used	03

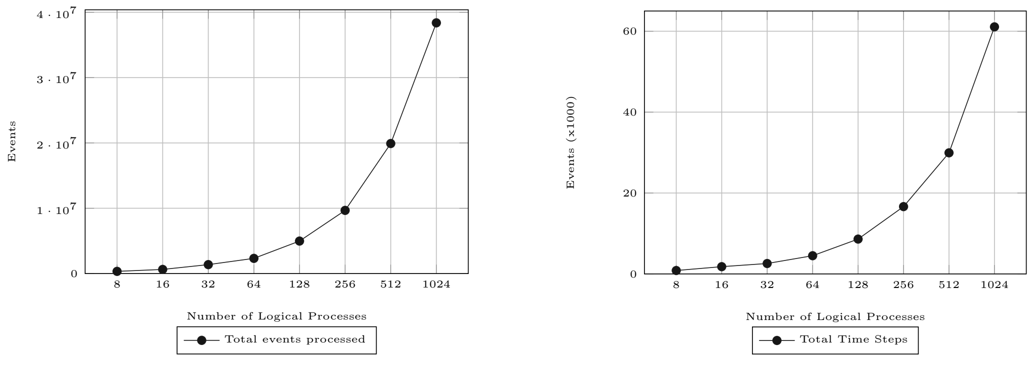
the same state, without effecting processes on other devices.

8. Result & Discussions

8.0.1. Experimental Setup & Benchmark Application Over Mobile Devices

For the experimental setup, three mobile devices are used with specifications listed in Table 9. All the experiments are performed five times and average results are reported in this section. For the result comparison, the total number of LPs are varied to measure the total events, processed events, event rate, CPU, memory usage, energy consumption, and execution time. The result section shows the comparison among tree barriers, time warp, CMB Null message, and time-stepped algorithms. In this experimental setup, devices are connected through a one-hop distance i.e. WiFi router; therefore, the communication delay is not covered due to the controlled environment. In order to benchmark the proposed SEECSSim framework, PHOLD has been implemented. The simulation topology of the PHOLD model is shown in Figure 8. All the devices have the same specifications during the benchmark. It is also worth pointing out that the margin of error (computed with 95% confidence interval) is very small compared with the average values; therefore, it is not reported in the Figures for better readability.

The results of Time-Stepped synchronization algorithm on the mobile platform with

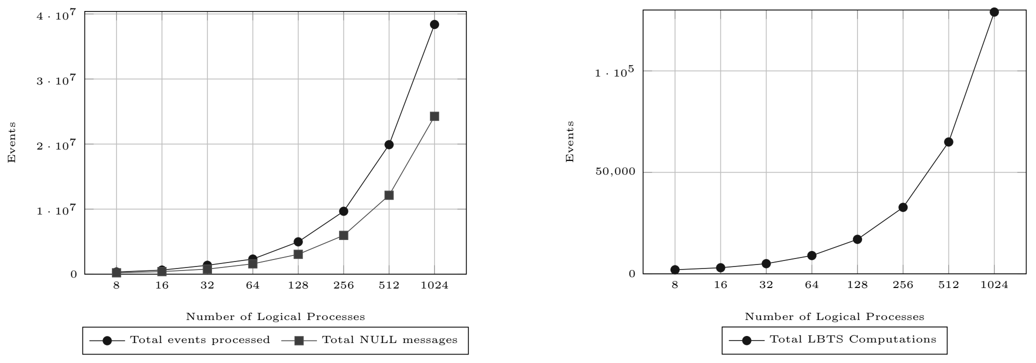


(a) (b)

Figure 9.: PHOLD with Time-Stepped synchronization on the mobile platform (a) Total events processed, and (b) Total time-stepped with increasing LPs.

a variable number of LPs are shown in Figure 9a and Figure 9b. 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 works 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. We observed that the trend shown by the Tree Barrier (results not shown here) with an increasing number of LPs is almost the same as the Time-Stepped approach.



(a) (b)

Figure 10.: PHOLD with CMB NULL messages synchronization on the mobile platform (a) Total events processed and NULL messages generated with varying number of logical processes, and (b) LBTS computations with increasing LPs.

The results of PHOLD with Chandy−Misra−Bryant NULL message synchronization with a varying number of LPs are reported in Figure 10a and Figure 10b. 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.

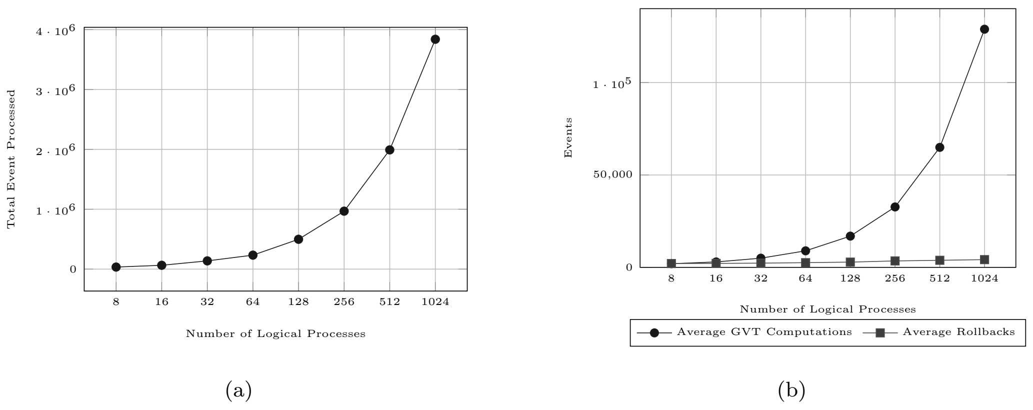


Figure 11.: PHOLD with Time Warp synchronization on the mobile platform (a) Total events processed, and (b) shows the average GVT and Rollbacks with varying logical processes

The results of PHOLD with Time Warp (TW) synchronization and a varying number of LPs are shown in Figure 11a and Figure 11b. 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 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 12. The event rate is defined as the total number of events processed in a unit time. Figure 12 shows that TW and CMB *NULL* message algorithm show 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.

8.0.2. 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

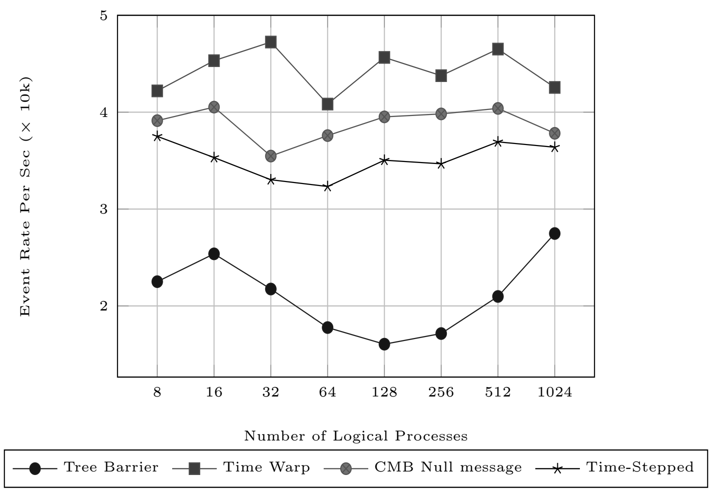


Figure 12.: Event rate for different synchronization algorithms

the simulations. Our goal is to 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.

CPU Usage – The average CPU usage of the different simulation synchronization algorithms is reported in Figure 13. 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.

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 paper. In fact, the other approaches consume a lower amount of energy and are very close to each other in

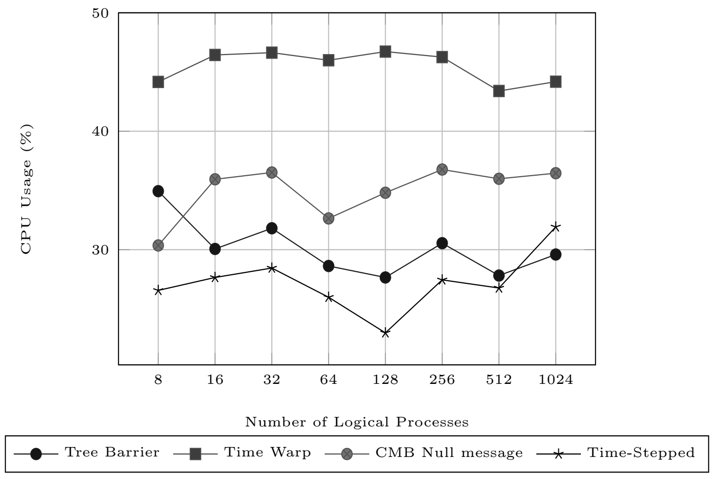


Figure 13.: Average CPU usage for different synchronization algorithms

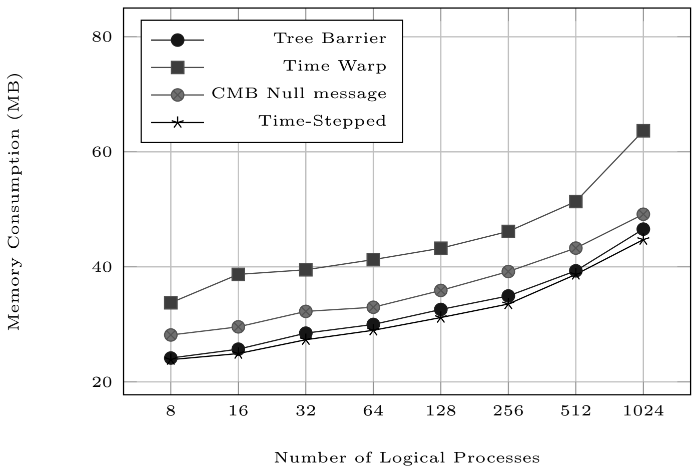


Figure 14.: Memory consumption for different synchronization algorithms

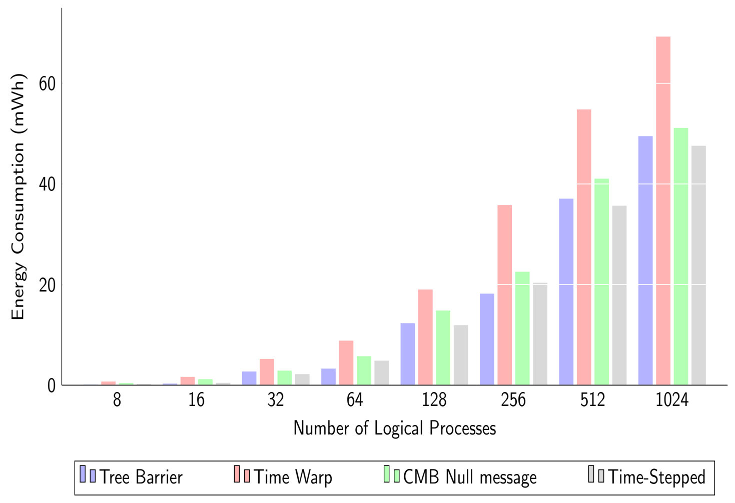


Figure 15.: Energy consumption for different synchronization algorithms

terms of memory consumption. The memory usage comparison for all the algorithms is shown in Figure 14.

Energy Consumption – Handheld devices are usually based on ARM processors that are designed for optimizing the energy consumption ((Smith and Hamilton, 2015) (Ryu and Ganis, 2012)) 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 [3] consumed by executing the PHOLD benchmark with the different synchronization algorithms is presented in Figure 15. 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 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 a lower amount of energy while the energy consumption of CMB is nearly the same as the Time Stepped and Tree Barrier approaches.

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 (Madisetti et al., 1988) 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.

The results presented in Figure 16 suggest that the energy consumption of Time Warp with Wolf Calls enabled is improved considerably with an increasing number

[3]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).

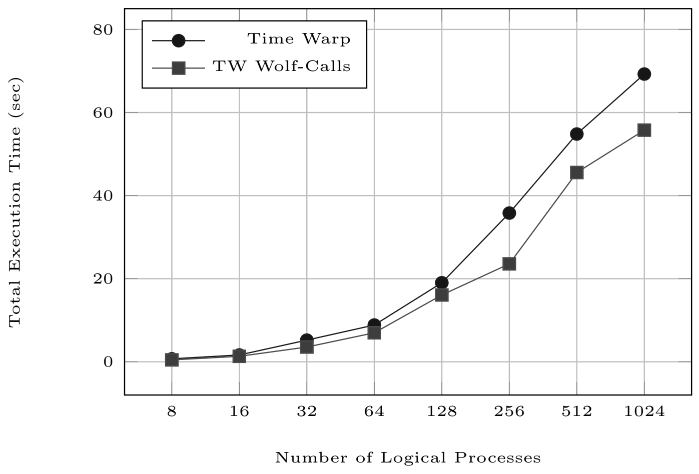


Figure 16.: Energy Consumption – Time Warp vs. Time Warp with Wolf Calls

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 17). The execution time has increased when using the Wolf Calls but it is still better than the Tree Barrier approach.

Total Execution Time – The total execution time for the different synchronization algorithms is shown in Figure 18. 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.

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 10–13. 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, we can identify 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

Discussion – Table 14 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.

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. We conclude that the CMB conservative algorithm is

Table 10. 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 11. 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 12. 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 13. 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**

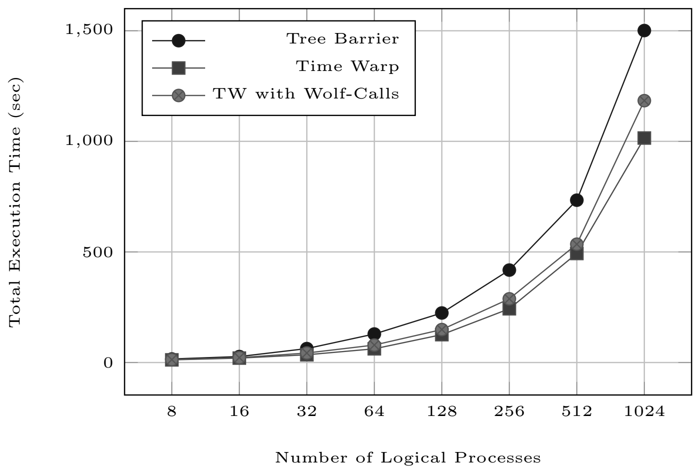


Figure 17.: Total Execution Time – Tree Barrier, Time Warp and Time Warp with Wolf Calls

Table 14. 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

adequate in terms of execution time as well as energy consumption for adoption over mobile devices.

9. SEECSSim Limitations and Future Directions

The initial version of SEECSSim v1.0 is designed to work on Andriod devices only. The framework establishes a peer-to-peer connection between devices where the devices maintain a communication table. The main objective of this study is to observe the behavior of well-known distributed simulation protocols over handheld devices due to resource constraints. In this version, the LPs are assigned in a sequential fashion i.e. every device executes the same number of LPs; however, in the future, we are interested to develop a more sophisticated model for LPs distributions while considering the specification of the devices. In this version, fault tolerance is not considered. Whereas, devices can join a simulation network at any time; however, the device departure should be planned in order to migrate the LPs to other devices to support causality constraints. In the future, we are interested to incorporate the fault tolerance scenario to handle the more unreliable execution environment.

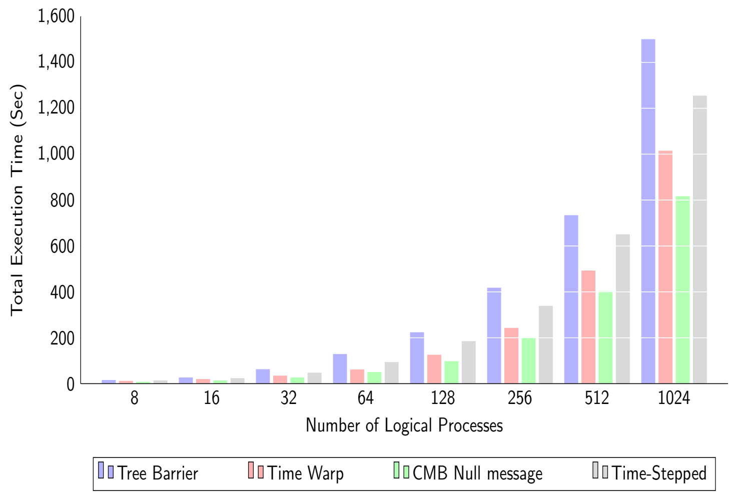


Figure 18.: Total Execution Time for different synchronization algorithms

10. Conclusions

In this paper, we have analyzed through instrumentation 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, we proposed a new distributed simulation framework 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.

References

Allinea (2017). Allinea-map. `http://www.allinea.com/products/map`. Accessed April. 2, 2017.

Bajaj, L., Takai, M., Ahuja, R., Tang, K., Bagrodia, R., and Gerla, M. (1999). Glomosim: A scalable network simulation environment. *UCLA Computer Science Department Technical Report*, 990027(1999):213.

Bauer Jr, D. W., Carothers, C. D., and Holder, A. (2009). 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.

Biswas, A. and Fujimoto, R. (2016). 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.

Bononi, L., Bracuto, M., D'Angelo, G., and Donatiello, L. (2004). Artis: a parallel and distributed simulation middleware for performance evaluation. In *International Symposium on Computer and Information Sciences*, pages 627–637. Springer.

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

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

Child, R. and Wilsey, P. A. (2012). Using dvfs to optimize time warp simulations. In *Proceedings of the Winter Simulation Conference*, page 288. Winter Simulation Conference.

Cowie, J. H., Nicol, D. M., and Ogielski, A. T. (1999). Modeling the global internet. *Computing in Science & Engineering*, 1(1):42–50.

Curtis-Maury, M., Shah, A., Blagojevic, F., Nikolopoulos, D. S., De Supinski, B. R., and Schulz, M. (2008a). 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.

Curtis-Maury, M., Shah, A., Blagojevic, F., Nikolopoulos, D. S., De Supinski, B. R., and Schulz, M. (2008b). 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.

D'Angelo, G. (2017). The simulation model partitioning problem: an adaptive solution based on self-clustering. *Simulation Modelling Practice and Theory (SIMPAT)*, 70:1 – 20.

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

D'Angelo, G., Ferretti, S., and Marzolla, M. (2012). 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. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Darema, F. (2004). Dynamic data driven applications systems: A new paradigm for application simulations and measurements. In Bubak, M., van Albada, G. D., Sloot, P. M. A., and Dongarra, J., editors, *Computational Science - ICCS 2004*, pages 662–669, Berlin, Heidelberg. Springer Berlin Heidelberg.

Erazo, M. A. and Pereira, R. (2010). 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.

Feng, X., Ge, R., and Cameron, K. W. (2005a). Power and energy profiling of scientific applications on distributed systems. In *19th IEEE International Parallel and Distributed Processing Symposium*, pages 34–34. IEEE.

Feng, X., Ge, R., and Cameron, K. W. (2005b). 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.

Fujimoto, R. M. (2000). *Parallel and distributed simulation systems*, volume 300. Wiley New York.

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

Fujimoto, R. M., Bagrodia, R., Bryant, R. E., Chandy, K. M., Jefferson, D., Misra, J., Nicol, D., and Unger, B. (2017a). Parallel discrete event simulation: The making of a field.

Fujimoto, R. M., Hunter, M., Biswas, A., Jackson, M., and Neal, S. (2017b). Power efficient distributed simulation. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 77–88. ACM.

Garg, R., Garg, V. K., and Sabharwal, Y. (2010). Efficient algorithms for global snapshots in large distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):620–630.

Ge, R., Feng, X., Song, S., Chang, H.-C., Li, D., and Cameron, K. W. (2010). Powerpack:

Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671.

Guérout, T., Monteil, T., Da Costa, G., Calheiros, R. N., Buyya, R., and Alexandru, M. (2013). Energy-aware simulation with dvfs. *Simulation Modelling Practice and Theory*, 39:76–91.

Hua, S. and Qu, G. (2003a). 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.

Hua, S. and Qu, G. (2003b). 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.

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

Isci, C. and Martonosi, M. (2003). 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.

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

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

Knüpfer, A., Brunst, H., Doleschal, J., Jurenz, M., Lieber, M., Mickler, H., Müller, M. S., and Nagel, W. E. (2008). The vampir performance analysis tool-set. In *Tools for High Performance Computing*, pages 139–155. Springer.

Kumar, K., Rajiv, P., Laxmi, G., and Bhuyan, N. (2014). 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.

Liu, J. (2007). The prime research.

Lively, C., Taylor, V., Wu, X., Chang, H.-C., Su, C.-Y., Cameron, K., Moore, S., and Terpstra, D. (2014a). E-amom: an energy-aware modeling and optimization methodology for scientific applications. *Computer Science-Research and Development*, 29(3-4):197–210.

Lively, C., Taylor, V., Wu, X., Chang, H.-C., Su, C.-Y., Cameron, K., Moore, S., and Terpstra, D. (2014b). E-amom: an energy-aware modeling and optimization methodology for scientific applications. *Computer Science-Research and Development*, 29(3-4):197–210.

Madey, G. R., Blake, M. B., Poellabauer, C., Lu, H., McCune, R. R., and Wei, Y. (2012). Applying dddas principles to command, control and mission planning for uav swarms. *Procedia Computer Science*, 9:1177 – 1186. Proceedings of the International Conference on Computational Science, ICCS 2012.

Madisetti, V., Walrand, J., and Messerschmitt, D. (1988). Wolf: A rollback algorithm for optimistic distributed simulation systems. In *Simulation Conference Proceedings, 1988 Winter*, pages 296–305. IEEE.

Malik, A. W., Mahmood, I., and Parkash, A. (2016). 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.

Malik, A. W., Park, A. J., and Fujimoto, R. M. (2010). An optimistic parallel simulation protocol for cloud computing environments. *SCS M&S Magazine*, 4:1–9.

Malony, A. D. and Shende, S. (2000). Performance technology for complex parallel and distributed systems. In *Distributed and parallel systems*, pages 37–46. Springer.

Malony, A. D., Shende, S., Bell, R., Li, K., Li, L., and Trebon, N. (2004). Advances in the tau performance system. In *Performance analysis and grid computing*, pages 129–144. Springer.

Maqbool, F., Malik, A. W., Mahmood, I., and D'Angelo, G. (2018). Seecssim - a parallel and distributed simulation framework for mobile devices. In *2018 IEEE/ACM 22nd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–7.

McInnes, A. I. and Thorne, B. R. (2011). 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.

Neal, S., Fujimoto, R., and Hunter, M. (2016). Energy consumption of data driven traffic simulations. In *Winter Simulation Conference (WSC), 2016*, pages 1119–1130. IEEE.

Pellegrini, A., Vitali, R., and Quaglia, F. (2011). 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).

Perumalla, K. S. (2007). 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.

Perumalla, K. S. (2013). *Introduction to reversible computing*. Chapman and Hall/CRC.

Procaccianti, G., Ardito, L., Morisio, M., et al. (2011). Profiling power consumption on desktop computer systems. In *International Conference on Information and Communication on Technology*, pages 110–123. Springer.

Pusukuri, K. K., Gupta, R., and Bhuyan, L. N. (2014). 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.

Rajovic, N., Rico, A., Vipond, J., Gelado, I., Puzovic, N., and Ramirez, A. (2013). 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.

Rouse, William Boff, K. R. (2005). *Organizational simulation*. Wily-Interscience.

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

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

Shenoy, K. (2004). Techniques for optimizing time-stepped simulations.

Smith, J. W. and Hamilton, A. (2015). Massive affordable computing using arm processors in high energy physics. In *Journal of Physics: Conference Series*, volume 608, page 012001. IOP Publishing.

Stanisic, L., Videau, B., Cronsioe, J., Degomme, A., Marangozova-Martin, V., Legrand, A., and Méhaut, J.-F. (2013). 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.

Teo, Y. M. and Ng, Y. K. (2002). Spades/java: object-oriented parallel discrete-event simulation. In *Simulation Symposium, 2002. Proceedings. 35th Annual*, pages 245–252. IEEE.

Tiwari, V., Malik, S., Wolfe, A., and Lee, M.-C. (1996). Instruction level power analysis and optimization of software. In *VLSI Design, 1996. Proceedings., Ninth International Conference on*, pages 326–328. IEEE.

Toscano, L., D'Angelo, G., and Marzolla, M. (2012). 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. ACM.

Vanmechelen, K., De Munck, S., and Broeckhove, J. (2012). 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.

Wu, Y., Cao, J., and Li, M. (2011a). 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.

Wu, Y., Cao, J., and Li, M. (2011b). 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.

Yoo, R. M., Hughes, C. J., Lai, K., and Rajwar, R. (2013a). 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.

Yoo, R. M., Hughes, C. J., Lai, K., and Rajwar, R. (2013b). 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.

Zhukov, I., Feld, C., Geimer, M., Knobloch, M., Mohr, B., and Saviankou, P. (2015). Scalasca v2: Back to the future. In *Tools for High Performance Computing 2014*, pages 1–24. Springer.