Parallel Simulation Techniques for Large-Scale Discrete-Event Models

By

Shafagh Jafer, B. Eng., M.A.Sc.

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering (OCIECE)
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6

August 2011

© Copyright 2011, Shafagh Jafer

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-87762-3

Our file Notre référence
ISBN: 978-0-494-87762-3

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distrbute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protege cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

To my parents,

making me who I am now with their love and support,

and to my loving husband, Mohammad,

for his constant love, support, and encouragement.

Abstract

The Discrete Event System Specification (DEVS) provides a general methodology for hierarchical construction of reusable models in a modular way and has been used to simulate complex systems in a variety of domains. This dissertation addresses software design and performance issues that arise in parallel simulation of large-scale DEVS-based models on multiprocessor cluster architecture.

Parallel simulation of complex DEVS-based models requires a robust simulator with low synchronization overhead. Recent researches focused on optimistic parallel simulation of DEVS-based systems. In this research three conservative parallel DEVS protocols (Lower-Bound-Time-Stamp (LBTS), Chandy-Misra-Bryant (CMB), and Global-Lookahead-Management (GLM)) are proposed, allowing pure conservative simulation of DEVS-based systems. The protocols are based on the classical Chandy-Misra-Bryant synchronization mechanism, and they extend the DEVS abstract simulator, providing means for lookahead computation and null message distribution. A purely conservative simulator, called CCD++, is presented designed for running large-scale DEVS and Cell-DEVS models in parallel and distributed fashion.

An extensive comparative performance analysis is presented, analyzing the performance of CCD++ compared to an optimistic DEVS simulator. Several DEVS-based environmental models with different characteristics are studied. The experiments indicate that the conservative simulator improves performance in terms of execution time, memory usage, operational cost, and system stability for large models.

Acknowledgements

I want to express my sincere gratitude toward my adviser and my mentor, Professor Gabriel Wainer, for his support, guidance, and trust that helped me through my graduate studies. His diligence and commitment to science have been and will be a great influence on me for many years to come. I am grateful for having the opportunity to learn from him and work with him.

I would also like to thank the members of the ARS Laboratory and the Department of Systems and Computer Engineering at Carleton University. Special thanks to Qi Liu and Narendra Mehta for all kinds of technical assistance and support.

Table of Contents

Chapter 1: Introduction 1

1.1 Research Motivations and Objectives ... 3

A) DEVS Simulation with Conservative Approach ... 3

B) Comparative Study: Conservative vs. Optimistic DEVS ... 4

1.2 Organization ... 5

Chapter 2: Background 6

2.1 Conceptual Modeling and Simulation Framework ... 6

2.2 Classical DEVS Formalism ... 7

2.3 Parallel DEVS Formalism ... 10

2.4 Timed Cell-DEVS Formalism ... 12

2.5 Parallel Cell-DEVS Formalism ... 15

2.6 Parallel Discrete-Event Simulation ... 16

2.6.1 Conservative Synchronization Algorithms ... 17

2.6.2 Optimistic Synchronization Algorithms ... 27

2.6.3 Parallel and Distributed Environments ... 30

2.7 Parallel DEVS and Cell-DEVS Simulation in CD++ ... 31

2.7.1 Event-Processing Algorithms ... 34

2.8 DEVS-based Parallel and Distributed Simulation ... 42

2.9 Performance Evaluation of PDES Environments ... 44

Chapter 3: Contributions 46

3.1 Research Publications ... 48

Chapter 4: Conservative DEVS Protocols 51

4.1 Problem Statement and Design Methodologies ... 51

4.2 The Lower-Bound-Time-Stamp Protocol ... 52

A) Lookahead and LVT Computations ... 55

B) Scheduling 61
C) Resuming a Blocked LP 62
D) Null Message Handling 63
E) Deadlock Avoidance 63
F) Simulation Termination 64
G) Simulation Scenario in CCD++ with LBTS Protocol 65
4.3 The CMB DEVS Protocol 67
4.4 The Global Lookahead Management Protocol 67
A) Phase-Based Simulation with GLM Protocol 68
B) Lookahead and LVT Computation Strategy 69
C) Null Message Distribution 69
D) LP Block and Resume Mechanism 70
E) Deadlock Avoidance 70
F) I/O Operation 70
G) Termination 70
4.5 Comparison of the Protocols 71
4.6 The Classical P-DEVS Protocol 72
4.7 Zero-Lookahead in P-DEVS 75

Chapter 5: Comparative Study: Optimistic VS. Conservative Simulation **76**

5.1 Model-based Sensitivity Analysis 76
5.2 Protocol-based Sensitivity Analysis 79

Chapter 6: Performance Analysis **81**

6.1 Introduction to Benchmark Models 81
6.1.1 Definition of a Wildfire Model 82
6.1.2 Definition of a Watershed Model 84
6.1.3 Definition of the Synthetic Model 84
6.2 Experimental Configurations and Performance Metrics 85
6.3 Evaluation of the Conservative and Optimistic Protocols 86
6.4 Evaluation of the Conservative Protocols 99

6.5 Sensitivity Analysis of the Conservative Protocols 105

Chapter 7: Conclusion and Future Work **113**

7.1 Review of Key Contributions 114

7.2 Suggestions for Future Research 116

References **117**

List of Tables

Table 1. Comparison of PDES Approaches for DEVS 44
Table 2. Performance Metrics 86

List of Figures

Figure 1. Entities and Relationships of a System M&S Framework [8] 6
Figure 2. DEVS Semantics of an Atomic Model [8] 8
Figure 3. Sketch of a Cellular Automaton [Wai00] 12
Figure 4. A Timed Cell-DEVS Atomic Model [100] 13
Figure 5. Causality Error Scenario 17
Figure 6. Layered Architecture of CCD++ Simulator [21] 32
Figure 7. CCD++ Flat Architecture and Message-Passing Paradigm 33
Figure 8. Simulator Algorithm for (*I*, *0*) 34
Figure 9. Simulator Algorithm for (@, *t*) 35
Figure 10. Simulator Algorithm for (*, *t*) 35
Figure 11. Simulator Algorithm for (*x*, *t*) 36
Figure 12. FC Algorithm for (*I*, *t*) 37
Figure 13. FC Algorithm for (@, *t*) 37
Figure 14. FC Algorithm for (*y*, *t*) 38
Figure 15. FC Algorithm for (*x*, *t*) 38
Figure 16. FC Algorithm for (*, *t*) 39
Figure 17. FC algorithm for (*D*, *t*) 39
Figure 18. NC algorithm for (*I*, *0*) 41
Figure 19. NC algorithm for (*x*, *t*) 41
Figure 20. NC algorithm for (*y*, *t*) 41
Figure 21. Conservative Architecture of CCD++ 54
Figure 22. Flow Chart of the Conservative Algorithm on each LP 55
Figure 23. Conservative NC Algorithm for *Done* Message 60
Figure 24. Scheduler Mechanism of LBTS Protocol 62
Figure 25. Suspension-Event Execution Algorithm 63
Figure 26. NC Null Message Handling Algorithm 64
Figure 27. Simulation Termination Algorithm 64

Figure 28. Sample Simulation Scenario in CCD++ 65
Figure 29. Null Message Distribution Strategy in LBTS, CMB, and GLM Protocol 71
Figure 30. Parallel DEVS Simulation Protocol [82] 73
Figure 31. Sample Activity Patterns in Cell-DEVS Models 77
Figure 32. A: Horizontal, B: Vertical Partitioning of an 8x8 Cell-DEVS Model on Four Nodes 78
Figure 33. A: Small, and B: Large Cell Neighborhood 78
Figure 34. Initial-Load Distribution by Setting Multiple Initial Points 79
Figure 35. Predetermined Spread Rates for the *Fire1* Model [116] 82
Figure 36. A Skeleton of the *Fire1* Model Definition in CD++ [116] 83
Figure 37. A Skeleton of the *Fire2* Model Definition in CD++ 83
Figure 38. A Skeleton of the *Watershed* Model Definition in CD++ [Wai06] 84
Figure 39. A Skeleton of the *Synthetic* Model Definition in CD++ 85
Figure 40. *Fire1* Results for Various Sizes 88
Figure 41. BT Results of *Fire1* 90
Figure 42. NMR Results of *Fire1* 91
Figure 43. Memory Consumption for *Fire1* Model 92
Figure 44. *Fire2* Results for Various Sizes 93
Figure 45. BT Results of *Fire2* 94
Figure 46. NMR Results of *Fire2* 94
Figure 47. Memory Consumption for *Fire2* Model 95
Figure 48. *Watershed* Results for Various Sizes 96
Figure 49. BT Results of *Watershed* 97
Figure 50. NMR Results of *Watershed* 98
Figure 51. Memory consumption for *Watershed* model 98
Figure 52. *Fire1* Model T and BT Results 100
Figure 53. *Fire1* Model Memory Consumption Results 100
Figure 54. *Watershed* Model T and BT Results 101
Figure 55. *Watershed* Model Memory Consumption Results 102
Figure 56. *Synth* Model Results 102
Figure 57. *Fire1* Model NMR Results 103

Figure 58. *Watershed* Model NMR Results 104
Figure 59. *Synth* Model NMR and NEV Results 105
Figure 60. Initial States Analysis of *Fire1* Model (100x100) 107
Figure 61. Initial States Analysis of *Fire1* Model (300x300) 107
Figure 62. Initial States Analysis of *Fire1* Model (500x500) 107
Figure 63. Partitioning Experiments of *Fire1* Model (100x100) 110
Figure 64. Partitioning Experiments of *Fire1* Model (300x300) 110
Figure 65. Partitioning Experiments of *Fire1* Model (500x500) 110
Figure 66. Partitioning Experiments of *Fire2* Model (100x100) 111
Figure 67. Partitioning Experiments of *Fire2* Model (300x300) 111
Figure 68. Partitioning Experiments of *Fire2* Model (500x500) 111
Figure 69. Partitioning Experiments of *Watershed* Model (25x25x2) 112
Figure 70. Partitioning Experiments of *Watershed* Model (50x50x2) 112
Figure 71. Partitioning Experiments of *Watershed* Model (100x100x2) 112

List of Acronyms

CMB	Chandy-Misra-Bryant
CTW	Conservative Time Window
DEVS	Discrete Event System Specification
EIT	Earliest Input Time
EOT	Earliest Output Time
FC	Flat Coordinator
GLM	Global Lookahead Management
GVT	Global Virtual Time
LBTS	Lower Band Time Stamp
LM	Lookahead Manager
LP	Logical Process
LTW	Lightweight Time Warp
LVT	Local Virtual Time
M&S	Modeling and Simulation
MPI	Message Passing Interface
MTW	Moving Time Windows
NC	Node Coordinator
PDES	Parallel Discrete Event Simulation
TW	Time Warp

Chapter 1: Introduction

Recent advances in computer technology have influenced modeling and simulation (M&S) techniques to become an effective approach for analyzing and designing a broad array of complex systems where a mathematical analysis is intractable. The simulation process begins with a problem to solve. First, the real system is observed, its entities are identified, and a model is constructed. Then, the model is executed using a simulator consisting of a computer system, which executes the model's instructions and generates relevant output. These outputs are compared with the real system to verify the correctness of the model.

As models become larger and more complex, the problems of limited resources within a single processor arise. In order to improve the performance of discrete-event simulations, Parallel Discrete-Event Simulation (PDES) techniques were proposed. These methods allow for executing a single discrete-event simulation program on a parallel computer with multiple processors (or nodes). A PDES system is typically constructed as a set of Logical Processes (LPs), each representing a different portion of the physical system and potentially executing on a different processor in event-driven fashion. The execution of an event at a LP may modify the state of the LP and generate new events that will be sent to other LPs. During a simulation, the LPs interact with each other solely by exchanging time-stamped event messages. To ensure correct simulation results, the LPs must be synchronized properly to comply with the *local causality constraint* [26], which restricts each LP to process events in nondecreasing time stamp order. Errors resulting from out-of-order event execution are referred to as *causality errors*. Synchronization, as the key to parallel and distributed simulation, requires a robust mechanism to handle communication among concurrent processes. Synchronization techniques for PDES systems are broadly classified into two categories, namely *conservative* and *optimistic*. The conservative approach, as the first synchronization algorithm that was proposed in the late 1970s by Bryant [16], Chandy and Misra [17] and known as the Chandy-Misra-Bryant (CMB) algorithm, strictly avoids the possibility of processing events out of time stamp order. In contrast, the optimistic approaches, introduced by Jefferson's Time Warp (TW) protocol [15], allow causality errors to happen temporarily, but provide mechanisms to recover from them during execution. Both approaches have their own merits and are being used in different applications. An extensive survey of existing PDES techniques can be found in [14].

Among the existing modeling and simulation techniques, the DEVS (Discrete Event System Specification) formalism [1][2][3][8] is regarded as one of the most developed general-purpose M&S frameworks for Discrete Event Dynamic Systems (DEDS) [10]. DEVS not only allows for hierarchical construction of reusable discrete-event models in a modular way, but also provides an abstract *simulation engine architecture* that can be realized on diverse computing platforms [5]. The term simulation engine architecture refers to a hierarchy of simulation entities and their associated algorithms that can be used to execute DEVS-representable models correctly. It is considered as abstract in the sense that the conceptual simulation entities may not necessarily be mapped to physical processors in a one-to-one relation [4]. In the past four decades of research, many extensions to DEVS have been proposed in the literature. For instance, the Parallel DEVS (or P-DEVS) formalism [9] eliminates the serialization constraints existed in the original DEVS by allowing adequate handling of simultaneous events, which is needed for efficient execution of models in parallel and distributed environments. The Cell-DEVS [12] formalism is an extension to DEVS that allows defining an n-dimensional cell space to represent complex discrete event spatial models, where each cell is a DEVS atomic model, allowing for specifying both temporal and spatial relations between model components. Aside from these theoretical developments, various DEVS-based simulation tools have been implemented, such as DEVS-C++ [6], RTDEVS/CORBA [13], DEVSCluster [73], and DEVS/SOA [74], just to mention a few. In particular, the CD++ toolkit [18] is an open-source, object-oriented M&S environment that implements both P-DEVS and Cell-DEVS formalisms using different middleware technologies on varied platforms (see, e.g., [19][20][21][22][23][24][25]).

The Parallel simulation of complex DEVS-based models requires a robust simulator with low synchronization overhead. This dissertation combines advanced parallel simulation algorithms for large scale DEVS-based simulations. The goal was to integrate the formal advantages of the DEVS formal modeling and simulation framework with parallel simulation techniques, specifically the conservative synchronization approaches. Although PCD++ [21], an optimistic simulator for DEVS and Cell-DEVS , improves the overhead of optimistic parallel simulation, the issue of memory consumption due to state savings and rollbacks still remains. In order to experiment with both the conservative and optimistic methods within the DEVS modeling framework, this research work introduces three conservative DEVS protocols and a purely conservative simulator [9] for DEVS and Cell-DEVS. The resulting simulator,

called CCD++ (Conservative CD++), incorporated various strategies and was successfully able to execute large Cell-DEVS models, and in many cases out-performed the optimistic PCD++ simulator. This dissertation also provides a comparative study of the performance of conservative versus optimistic simulation of DEVS-based large-scale models using a number of Cell-DEVS models.

1.1 Research Motivations and Objectives

This research is motivated by two complementary and interrelated objectives. The first one is to address the challenges of large-scale conservative parallel simulation of P-DEVS and Cell-DEVS models on distributed-memory multiprocessor clusters using conservative null message-based protocols. The second one is to achieve a thorough comparative study of optimistic versus conservative DEVS-based simulation by conducting extensive experiments and performing precise sensitivity analyses at both model- and underlying synchronization protocol-level.

A) DEVS Simulation with Conservative Approach

Parallel simulation of complex models requires a robust simulator with low synchronization overhead. There has been a number of research efforts focused on the optimistic parallel simulation of DEVS-based models (see, e.g., [27][28][8][29][30][31]). For instance, the CD++ toolkit was extended to support Time Warp (TW) [15] simulation of P-DEVS and Cell-DEVS models on distributed-memory multiprocessors using the WARPED [67] simulation kernel as a middleware layer [21][34]. The resulting optimistic parallel simulator, referred to as PCD++, addressed several important issues raised in DEVS-based TW simulations, including asynchronous state transition, messaging anomalies, and rollbacks at virtual time zero [34].

Although optimistic protocols allow higher degree of parallelism, the issue of memory consumption due to state savings and rollbacks still remains. This is especially apparent when the number of participating nodes increases; resulting in cascaded rollbacks, and further memory and computation overhead. In contrast, conservative approaches overcome these issues by determining when it is safe to process an event for all the LPs in the system. In general, conservative synchronization algorithms are classified into two categories, namely *synchronous* and *asynchronous*. Synchronous conservative algorithms use global barrier synchronization and reduction at specific points in the simulation process

to iteratively determine which events are safe to process (see, e.g., [55] [56][57][58]), making them best suited for shared-memory computers where the overhead of global synchronization can be minimized. On the other hand, asynchronous conservative protocols discard the global barrier computation by imposing a locking mechanism where a LP is blocked when it does not have enough information to process its next event safely. However, deadlocks can occur if the blocked LPs form a cycle [59], requiring the use of either deadlock-avoidance or deadlock-recovery techniques to ensure the progress of the simulation.

With many LPs allocated on each available processor in a typical large-scale simulation, saving historical data in the event and state queues not only consumes an excessive amount of memory, but also raises the cost of queue operation, fossil collection, and dynamic process migration. More importantly, these problems are worse when a large number of simultaneous events (i.e., events with exactly the same time stamp) need to be executed at each virtual time, as commonly found in large-scale, densely-interconnected, and highly-active DEVS-based models.

Aside from optimistic DEVS-based protocols, a number of conservative DEVS parallel approaches have been proposed in the literature (see, e.g., [75][76][77][1]). Nevertheless, most of these approaches are in the High Level Architecture (HLA) [68] domain [64][78][79][80][81][82], leaving the challenges of *purely conservative* DEVS-based simulations unaddressed.

The issues related to performance, scalability, and complexity of optimistic-based parallel simulations and the need for a purely conservative DEVS/Cell-DEVS simulator motivated this research to investigate conservative approaches for efficient conservative parallel simulation of P-DEVS and Cell-DEVS models.

B) Comparative Study: Conservative vs. Optimistic DEVS

To analyze the effect of the underlying synchronization protocol on the overall simulation performance, a comparative study is required that carefully investigates different metrics using the same benchmark. Deciding whether to use a conservative simulator or an optimistic one is only possible if large number of experiments have been conducted under the same hardware/software infrastructure. To evaluate the optimistic DEVS simulator (i.e., PCD++) versus conservative versions, the DEVS or Cell-DEVS model has to be executed on both simulators given the exact same conditions (such as initial values, size of the

model, hardware system configurations, etc.,). Towards this goal, the research presented in this dissertation takes a comparative approach to analyze the efficiency of different parallel DEVS synchronization mechanisms. In addition, this dissertation also attempts to take into account several methods to provide a detailed analysis that could be used for further expansion and development of the proposed synchronization techniques.

1.2 Organization

The rest of this dissertation is organized as follows. Chapter 2 provides background information on DEVS-based simulation and also it presents a literature review of parallel synchronization techniques and their application to DEVS. Chapter 3 provides the contributions of this dissertation. Chapter 4 presents three conservative DEVS protocols and discusses their implementation details in CCD++ simulator. Chapter 5 describes the various sensitivity analyses metrics that can be used to conduct experiments on PCD++ and CCD++. The experimental platform and metrics for performance evaluation are provided in Chapter 6, followed by the results of this dissertation and a comparative performance evaluation of CCD++ and PCD++ simulators in simulating DEVS-based cellular models. Finally, the concluding remarks and future research directions are reported in Chapter 7.

Chapter 2: Background

This chapter first presents the DEVS M&S framework and its implementation in the CD++ environment, and then it provides the state-of-the-art research efforts in PDES field. Section 2.1 introduces the basic concepts and the software architecture of the DEVS M&S framework. Section 2.2 reviews the classical DEVS formalism. Section 2.3 covers the Parallel DEVS (or P-DEVS) formalism. The Timed Cell-DEVS and Parallel Cell-DEVS formalisms are presented in Section 2.4 and 2.5 respectively. Section 2.6 describes the simulation algorithms and computational properties in the context of the CD++ environment. The literature review of various parallel synchronization algorithms and the DEVS-based parallel simulation techniques are presented in Section 2.7 and 2.8 respectively.

2.1 Conceptual Modeling and Simulation Framework

Conceptual M&S framework defines the system under study as basic entities and their relationships. Zeigler *et al.* proposed a conceptual M&S framework that strictly separates modelling from simulation framework by introducing four basic entities and two types of relationships [8], as illustrated in Figure 1.

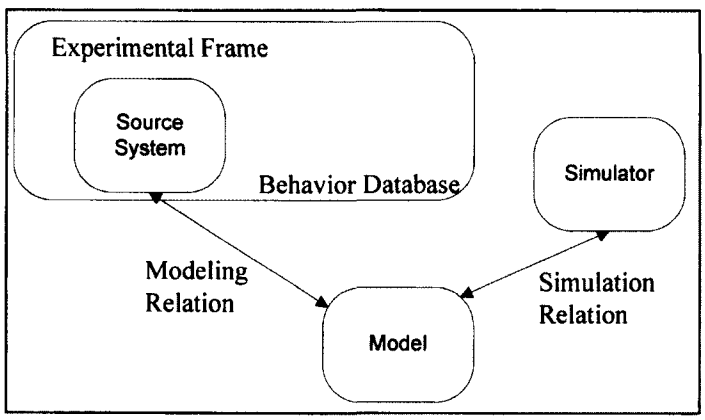


Figure 1. Entities and Relationships of a System M&S Framework [8]

The entities include *source system*, *experimental frame*, *model*, and *simulator*. The source system entity defines the real or virtual environment under analysis. This entity, which is viewed as the data source, together with the behavior database forms the Experimental Frame. The experimental frame

specifies the conditions under which the source system is observed or experimented with. A model entity represents an abstraction of the source system represented by a set of instructions, rules, mathematical equations, or a set of constraints to approximate the behavior of the real system. The simulator entity is a computer-based entity which is in charge of executing the model's instructions.

The two fundamental relationships among the entities are the *modeling relation* (or *validity*) and the *simulation relation* (or *simulator correctness*) [8]. The modeling relation links the model and the source system to validate the results generated by the model. In general, the model is considered valid if the data it generates agree with the data generated by the source system in the experimental frame in use. On the other hand, the simulation relation lies between the simulator and the model to indicate how reliable is the simulator in terms of being capable to execute the model's instructions.

The separation between model and simulator significantly simplifies the model validation and simulator verification [8]. Furthermore, it gives the opportunity to use different simulation algorithms within the simulator or even different simulators. In addition, the separation of concerns involved in this architecture allows model reusability as well as later extension of the model.

2.2 Classical DEVS Formalism

Based on the above M&S framework concepts, the **Discrete Event System Specification** (DEVS) formalism supports hierarchical construction of reusable discrete-event models in a modular way [8]. In DEVS, a real system is decomposed into behavioral (*atomic*) and structural (*coupled*) components. DEVS theory provides a rigorous methodology for representing models, while presenting an abstract way of thinking about the world with independence of the simulation mechanisms and the underlying hardware and middleware. A DEVS atomic model is formally defined by [8]:

$$M = <X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta>,$$

where

$X = \{(p,v) \mid p \in IPorts, v \in X_p\}$	is the set of input ports and values;
$Y = \{(p,v) \mid p \in OPorts, v \in Y_p\}$	is the set of output ports and values;
S	is the set of sequential states;
$\delta_{int}: S \rightarrow S$	is the *internal state transition function*;
$\delta_{ext}: Q \times X \rightarrow S$	is the *external state transition function, where*

$Q = \{(s,e) \mid s \in S, 0 < \in < ta(s)\}$ is the total state set, *e* is the time elapsed since the last state transition;

$\lambda: S \rightarrow Y$ is the *output function*;

$ta: S \rightarrow R^{+}_{0,\infty}$ is the *time advance function.*

Figure 2 shows the description of states and variables in DEVS models. At any time, a DEVS atomic model is in a state $s \in S$. In the absence of external events, the model stays at this state for the duration specified by ta(*s*). When the elapsed time *e*, is equal to ta(*s*), the state duration expires and the atomic model outputs the value give by $\lambda(s)$, and changes to a new state $\delta_{int}(s)$. An atomic model that has a due internal state transition at the current simulation time is referred to as an *imminent* model component. Notice that output is only generated by an imminent model and occurs right before the scheduled internal state transition. Transitions that occur due to the expiration of ta(*s*) are called *internal transitions*. On the other hand, state transition can also happen due to arrival of an external event which places the model into a new state specified by $\delta_{ext}(s,e,x)$; where *s* is the current state, *e* is the elapsed time, and *x* is the input value. Note that the life time of a state can take any real value from *zero* to *infinity*. A state with zero duration is called *transient* state, while when ta(s) is equal to *infinity* the state is said to be *passive*, in which the system will remain in this state until receiving an external event.

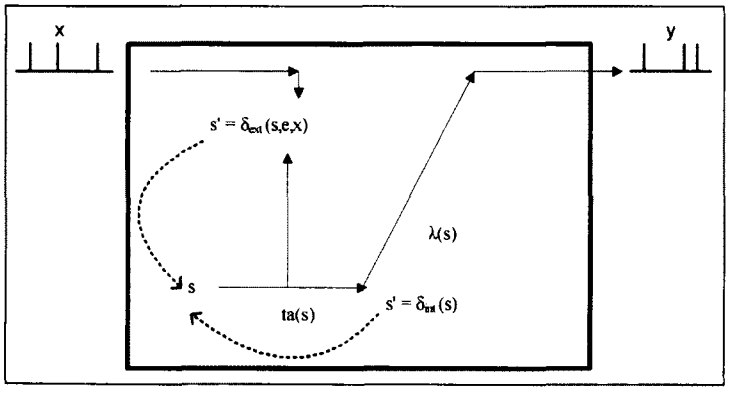


Figure 2. DEVS Semantics of an Atomic Model [8]

The DEVS formalism provides a well-defined concept of system modularity and component coupling allowing for construction of hierarchical models. A DEVS *coupled model* is composed of several atomic or coupled sub-models that are connected with each other and with the external environment, as shown in the following formal definition [8].

$$CM = \langle X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC, \textit{Select} \rangle,$$

Where

$X = \{(p,v) \mid p \in IPorts, v \in X_p\}$ is the set of input ports and values;

$Y = \{(p,v) \mid p \in OPorts, v \in Y_p\}$ is the set of output ports and values;

D is the set of the component names;

The following requirements are imposed on each component *d* that is included in D:

$M_d = (X_d, Y_d, S_d, \delta_{int}, \delta_{ext}, \lambda, ta)$ is a DEVS model with

$X_d = \{(p,v) \mid p \in IPorts_d, v \in X_p\}$, and $Y_d = \{(p,v) \mid p \in OPorts_d, v \in Y_p\}$.

The component couplings are subject to the following requirements:

External input coupling (EIC) connects external inputs to component inputs,

$EIC \subseteq \{((N, ip_N), (d, ip_d)) \mid ip_N e\ IPorts, d \in D, ip_d \in IPorts_d\}$;

External output coupling (EOC) connects component outputs to external outputs,

$EOC \subseteq \{((d, op_d), (N, op_N)) \mid op_N \in OPorts, d \in D, op_d \in OPorts_d\}$;

Internal coupling (IC) connects component outputs to component inputs,

$IC \subseteq \{((a, op_a), (b, ip_b)) \mid a, b \in D, op_a \in OPorts_a, ip_b \in IPorts_b\}$;

Select: $2^D - \{\} \rightarrow D$ is the tie-breaking function for imminent components.

Direct feedback loops are not allowed, i.e., an output port of a component may not be connected to an input port of the same component which can be formally specified as $((d, op_d), (e, ip_d)) \in IC$ implies $d \neq e$. In addition, the values sent from a source port must follow the range inclusion constraint of a destination port, formally expressed as:

$$\forall ((N, ip_N), (d, ip_d)) \in EIC : X_{ipN} \subseteq X_{ipd}$$

$$\forall ((a, op_a), (N, op_N)) \in EOC : Y_{opa} \subseteq Y_{opN}$$

$$\forall ((a, op_a), (b, ip_b)) \in IC : Y_{opa} \subseteq X_{ipb}.$$

From the coupled DEVS formalism it can be observed that due to the *closure under coupling* property, a coupled model is regarded as a new DEVS model [96]. This property ensures that the overall behavior of a coupled model is equivalent to a basic atomic model, and therefore, allows for hierarchical model construction. The X and Y sets describe the input and output events of the coupled model. Upon reception of an input event, it has to be redirected to the corresponding atomic component. Similarly,

when an output is generated by a component, it must be mapped as an input to another component or sent out as an output of the coupled model. The mapping mechanism is defined by the Z function.

In coupled DEVS models, when multiple imminent components are scheduled for an internal transition at the same time, this can lead to ambiguity. For example, let's consider a case where we have two imminent components: A, and B. When component A executes its internal transition, it produces an output that maps to an external event for component B. However, at this moment, component B is already scheduled for an internal transition. This will cause an ambiguity for component B, not knowing which transition to execute first. The DEVS formalism suggests two alternatives for this scenario: 1) execute the external transition first with e being equal to $ta(s)$ and then the internal transition, or 2) execute the internal transition first and then the external transition with e being equal to zero. DEVS resolves this ambiguity by introducing the *select* tie-breaking function. This function gives order to the imminent components of a coupled model so that only one component has $e = 0$. Then the rest of imminent components are divided into two groups: 1) a set of components that receive an external output from this model, 2) the rest of components. The first group will then execute their external transition functions with $e = ta(s)$, and the second group will be imminent during the next simulation cycle which may further require the use of the *select* function to decide which component will be the first. The use of tie-breaking mechanism adds overhead to the simulation and, in addition, decreases the level of parallelism and forces the simulation to have a serialized manner. Since the *select* mechanism associates priorities with imminent components, it will cause a potential bottleneck in the simulation system when many interconnected atomic models are imminent at the same time. These problems have been addressed by Chow and Zeigler in a DEVS extension, known as the Parallel DEVS formalism [11], which will be presented in the next section.

2.3 Parallel DEVS Formalism

The **Parallel DEVS** or **P-DEVS** [11] formalism is an extension to DEVS that eliminates all the serialization constraints and provides an environment for executing simultaneous DEVS models in parallel. P-DEVS implements *confluent* function to deal with collision scenarios at which events are scheduled simultaneously [8]. This function allows a modeler to explicitly define the collision behavior for individual atomic models. In addition, each atomic model maintains a *bag* structure to collect all

external events received from other model components at a given simulation time so that these events can be processed as a group in the state transition, combining the execution of multiple external transitions into a single one. As a result, many imminent components can be activated simultaneously to send output to other components all at the same simulation time [9]. The receiver is responsible for examining the input external events and interpreting them properly. The P-DEVS formalism allows for increased parallelism to be exploited in a simulation [11].

An atomic P-DEVS model is specified by [11]:

$$M = < X_M , Y_M , S, \delta_{ext} , \delta_{int}, \delta_{con}, \lambda, ta)$$

where

$X_M = \{(p,v) \mid p \in IPorts, v \in X_p \}$	is the set of input ports and values;
$Y_M = \{(p,v) \mid p \in OPorts, v \in Y_p \}$	is the set of output ports and values;
S	is the set of sequential states;
$\delta_{ext}: Q \times X_M^b \rightarrow S$	is the external state transition function;
$\delta_{int}: S \rightarrow S$	is the internal state transition function;
$\delta_{con}: Q \times X_M^b \rightarrow S$	is the confluent transition function;
$\lambda : S \rightarrow Y_M^b$	is the output function;
$ta : S \rightarrow R_0^+ \cup \infty$	is the time advance function;

with $Q := \{(s, e) \mid s \in S , 0 \leq e \leq ta(s)\}$ the set of total states.

The elimination of the sequential *Select* function and its replacement with the *confluent transition function* gives all the imminent components equal priority and the permission to be activated and to send their output to other components at the same time. On the other hand, the receiver component is only responsible for identifying the type of the received input event and taking the required actions. A P-DEVS coupled model is similar to DEVS, except for the omission of the *Select* function. Formally, a coupled model is defined as [11]:

$$CM = <X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC>$$

Therefore, the set of input and output events (*X* and *Y*), components (*D* and M_d), and couplings (*EIC*, *EOC*, and *IC*) are identically the same as of DEVS. Since in P-DEVS there is no serialization among imminent components, in case of having multiple imminent components within a coupled P-

DEVS model, firstly, all the outputs are collected and redirected to the corresponding influences, secondly, the transition function is executed [8].

2.4 Timed Cell-DEVS Formalism

The **Cellular Automata** (CA) theory was first introduced by John von Neumann in his study of self-replicating systems [97] . A cellular automaton, as presented in Figure 3, is an infinite regular n-dimensional lattice of finite state machines interconnected locally with each other. The lattice consists of cells that change their states synchronously and in parallel at discrete time steps based on the states of a finite set of neighboring cells, referred to as the neighborhood, by evaluating a local update rule. This is performed by using the current cell's state and those of a finite set of nearby cells. Despite its widespread application, the CA approach has two major limitations making it computationally inefficient [99]. First, due to its discrete time nature, simulation precision and execution efficiency is greatly restricted. Secondly, at each time step, all the cells are evaluated synchronously, incurring an unnecessarily high computational cost when only a small fraction of the cells needs to be updated. The **Timed Cell-DEVS** formalism [100] overcomes these issues by integrating DEVS and CA to present each cell as an atomic DEVS model.

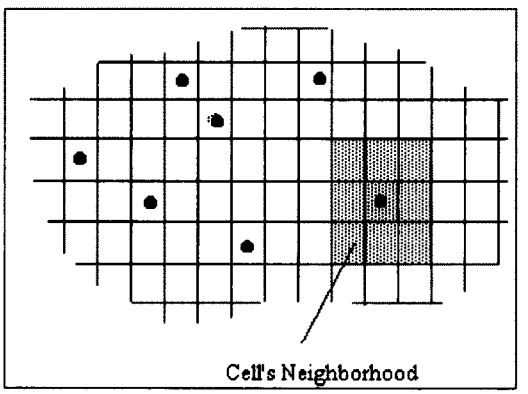


Figure 3. Sketch of a Cellular Automaton [Wai00]

Cell-DEVS formalism defines n-dimensional cell spaces as discrete-event models, allowing for more efficient asynchronous execution using a continuous time base without losing simulation accuracy. Each cell is represented as a DEVS atomic model that changes state in response to the occurrence of events in an event-driven fashion. Moreover, Cell-DEVS allows the implementation of cellular models with timing delays. Two types of timing delays can be used, namely *transport* and *inertial*. When

transport delay is used, the future value is added to a queue sorted by output time, allowing the previous values that were scheduled for output but have not yet been sent to be kept. On the other hand, inertial delays allow a pre-emptive policy at which any previous scheduled output value will be deleted and the new value will be scheduled. Figure 4 illustrates a timed Cell-DEVS atomic model.

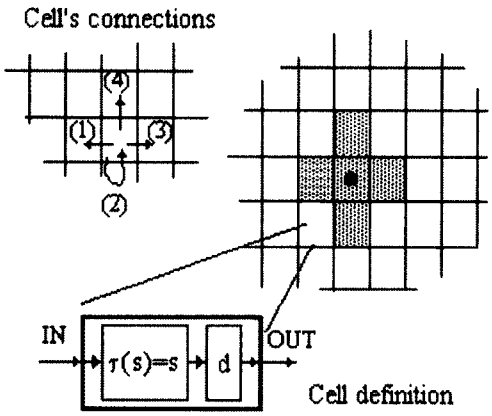


Figure 4. A Timed Cell-DEVS Atomic Model [100]

A Cell-DEVS atomic model is defined by [99]:

$$TDC = < X, Y, I, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >,$$

where

X	is a set of external input events;
Y	is a set of external output events;
I	represents the model's modular interface;
S	is the set of sequential states for the cell;
θ	is the cell state definition;
N	is the set of states for the input events;
d	is the delay for the cell;
δ_{int}	is the internal transition function;
δ_{ext}	is the external transition function;
τ	is the local computation function;
λ	is the output function; and
D	is the state's duration function.

The modular interface (I) represents the input/output ports of the cell and their connection to the neighbor cell. Communications among cells are performed through these ports. The values inserted through input ports are used to compute the future state of the cell by evaluating the local computation function τ. Once τ is computed, if the result is different than the current cell's state, this new state value must be sent out to all neighboring cells informing the state change. Otherwise, the cell remains in its current state and therefore no output will be propagated to other cells. This will happen when the time given by the delay function expires. Finally, the internal, external transition functions and output functions (λ) define this behavior. Cell-DEVS improves execution performance of cellular models by using a discrete-event approach. It also enhances the cell's timing definition by making it more expressive. Cell-DEVS coupled models represent the cell space as follows [100]:

$$GCC = < X_{list}, Y_{list}, I, X, Y, n, \{t_1,...,t_n\}, N, C, B, Z, select >,$$

where

X_{list}	is the input coupling list;
Y_{list}	is the output coupling list;
I	represents the definition of the model's interface;
X	is the set of external input events;
Y	is the set of external output events;
n	is the dimension of the cell space;
$\{t_1,...,t_n\}$	is the number of cells in each of the dimensions;
N	is the neighborhood set;
C	is the cell space;
B	is the set of border cells;
Z	is the translation function; and
select	is the tie-breaking function for simultaneous events.

A coupled model is composed of an array of atomic cells (C) with given size and dimensions where each cell is connected through standard DEVS input/output ports to the cells defined in the neighborhood (N). Since the cell space is finite, the borders of the cells are either connected to a different neighborhood than the rest of the space, or they are "wrapped" (i.e. $B = \{\varnothing\}$) in which they are connected to those in the opposite one using the inverse neighborhood relationship. However, border

cells have a different behavior due to their particular locations, which result in a non-uniform neighborhood. The Z function defines the internal and external coupling of cells in the model. It translates the outputs of the i^{th} output port in cell C_a into values for the i^{th} input port in cell C_b. *Select* function has similar functionality as in the basic DEVS, where it is the tie-breaking function for the imminent components.

As in coupled DEVS models, the use of *Select* function produces serialization, and therefore similar limitations when the Cell-DEVS models are considered to be executed in parallel. These limitations would lead to lack of parallelism exploitation and a probable inconsistency with the real system [100]. Moreover, since the timed Cell-DEVS allows only one input from each input port, zero-delay transitions are not possible and also the external DEVS models are not allowed to send two simultaneous events to the same cell.

2.5 Parallel Cell-DEVS Formalism

In order to resolve transition collisions without using the *Select* function, a new version of the Timed Cell-DEVS formalism, referred to as **Parallel Cell-DEVS** [98], has been proposed based on the P-DEVS concepts. The Parallel Cell-DEVS formalism overcomes these restrictions by revising and extending Cell-DEVS to allow a higher degree of parallelism and allowing zero-delay transitions as well as multiple simultaneous events per external model. Parallel Cell-DEVS models are equivalent to parallel DEVS models and closure under coupling holds for parallel Cell-DEVS models as well. That is a coupled Cell-DEVS model is equivalent to an atomic Cell-DEVS model.

The formal definition of a Parallel Cell-DEVS atomic model is given as follows [98].

$$PCM = < X_b, Y_b, I, S, \theta, N, d, \delta_{int}, \delta_{ext}, \delta_{con}, \tau, \tau_{con}, \lambda, D >.$$

Most of the components in the definition remain unchanged as in the Timed Cell-DEVS specification. However, two exceptions exist: first, the external state transition function and the output function maintain *bags* of inputs and outputs (X_b and Y_b) for each cell. Secondly, two additional confluent state transition functions (δ_{con} and τ_{con}) are introduced in the definition. When collisions between internal and external events happen at a cell, the confluent function δ_{con} is invoked as in the P-DEVS formalism and it activates the *confluent local transition function* τ_{con}, which in turn analyzes the current values in the input bags and presents a unique set of inputs for the cell to compute the next state. Hence, allowing the

modeler to precisely control the behavior of each cell under collision situations by implementing the confluent local transition function.

By eliminating the *Select* function, the Parallel Cell-DEVS coupled model definition is given as follows [98]:

$$GCC = <X_{list}, Y_{list}, I, X, Y, \eta, \{t_1, ..., t_n\}, N, C, B, Z >.$$

While each cell in the cell space (C) is a Parallel Cell-DEVS atomic model, all the other components are defined in the same way as presented in the previous section.

The Parallel DEVS and Cell-DEVS formalisms not only provide a unified M&S framework, but also allow exploiting higher degree of parallelism in parallel and distributed simulations. Together, these two formalisms serve as the theoretical foundation for this research.

2.6 Parallel Discrete-Event Simulation

A **Parallel Discrete-Event Simulation (PDES)** consists of **Logical Processes (LPs)** acting as the simulation entities, which do *not* share any state variables, and interact with each other merely through exchanging time-stamped event messages [14]. In general, each LP is mapped to a physical processor of a parallel computing system, but if the number of LPs exceeds the number of available processors, multiple LPs are mapped to a single physical processor. The LPs that are allocated on the same processor maintain a single **Future Event List (FEL)** to schedule events execution in a *sequential* manner. The major challenge in PDES is being able to produce exactly the same results as in a sequential execution of the simulation program. This requires a precise and accurate synchronization of all the LPs in the system since data and computation distribution may result in different errors related to the concurrent processing of the simulation messages. Synchronization among these LPs is violated when one of the LPs receives an out of order event. This violation is referred to as *causality error*. Such a scenario is represented by Figure 5 where two LPs, each with one event in its input queue, process their events simultaneously. When LP1 executes event *e1* (whose timestamp is = 1), it generates and sends a new event message *e2* to LP2 (with timestamp = 2). However, at this time, LP2 has already processed *e6* and therefore its local clock has already advanced to 6. Consequently, the arrival of *e2* at LP2 violates causality, and an error occurs.

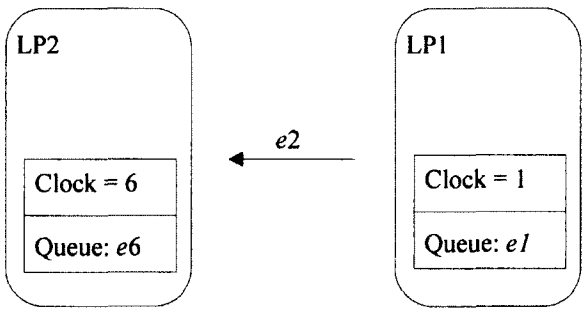


Figure 5. Causality Error Scenario

In order to prevent causality error, there is a synchronization requirement expressed as the following necessary and sufficient condition [14].

Local Causality Constraint: A discrete-event simulation, consisting of LPs that interact exclusively by exchanging timestamped messages obeys the local causality constraint if and only if each LP processes events in non-decreasing timestamp order.

To satisfy the local causality constraint, different synchronization techniques have been proposed for PDES systems which generally fall into two major classes of synchronization: *conservative*, which strictly avoid causality violations ; and *optimistic*, which allow violations and recover from them. In the past three decades, numerous approaches have been proposed by different researches in this field. A number of surveys can be found in the literature which summarize both conservative and optimistic techniques [14][26][117][118][119][95]. The following subsections introduce the basic concepts behind these two approaches.

2.6.1 Conservative Synchronization Algorithms

As discussed in the previous section, conservative synchronization algorithms strictly avoid any occurrence of causality errors. To do so, the LP is blocked from further processing of events until it can make sure that the next event in the local Future Event List is safe from future event arrivals from other LPs with smaller timestamps. The basic problem for a conservative parallel simulator is how to determine if it is safe for a processor to execute events. To deal with this issue, several techniques have been proposed which are further classified into four categories: methods with deadlock avoidance, deadlock detection and recovery, synchronous operation, and conservative time windows.

- **Synchronous Operation**

The first techniques developed for solving these problems proposed different centralized and decentralized mechanisms for implementing global clocks, and they used synchronous operations for the parallel discrete-event simulations. In [49] the authors proposed a centralized mechanism with one dedicated processor controlling a global clock (which represents the global virtual time of the simulation). Under that scheme, all the LPs' local clocks are kept at the same value at every point in real time, and the simulation proceeds according to this global clock, which is advanced based on the minimum timestamp of all possible next events. Peacock *et al.* [50] introduced the distributed implementation of such a global clock, which was used by [51] [51] on a hierarchical LP structure to determine the minimum next event time. The *min-reduction operation* [52] used a hierarchical LP organization. In this method, the minimum timestamp is moved to the root of a process tree, and it is then propagated down the tree. The *Distributed Snapshot Algorithm* [53][110] proposed a method to avoid the bottleneck of a centralized global clock coordinator by enabling the processes to record their own states and the states of the communication channels. In this way, the set of process and channel states recorded conform a global system state.

Three efficient algorithms for global snapshots in large distributed systems are presented in [120][121]. The proposed algorithms (a grid-based, a tree-based, and a centralized) overcome the issue of scalability of other existing global snapshot algorithms. Experiments showed that the proposed mechanisms significantly reduce the message and space complexity of a global snapshot.

In general, synchronous protocols decompose the simulation into two phases: (1) processing safe events, (2) performing global computations to determine such events. Unlike the detection and recovery methods that will be discussed in the following sections, synchronous mechanisms are deadlock-free. However, they continuously suspend and restart the simulation. In contrast, a major disadvantage of detection and recovery method is that during the period leading up to a deadlock, the execution may be largely sequential, leading to limited speedup.

- **Deadlock Avoidance**

The first existing asynchronous parallel simulation protocol was a conservative technique developed independently by Chandy and Misra [17], and Bryant [16]. In the *CMB Chandy-Misra-Bryant* (CMB) algorithm, LPs are assumed to be connected statically via directional links. LPs communicate through timestamped messages, also called *event messages*, which are transmitted from one LP to another, in

non-decreasing timestamp order. This guarantees that the timestamp of the last message received on an incoming link is a lower bound of any future event messages that will be received later. At each LP, there is a queue associated with each incoming link that is used to store incoming messages in FIFO order. Each link has its own clock which is equal to the timestamp of the first message in the queue (if there is one), or the timestamp of the last received message (if the queue is empty). The LP repeatedly selects the queue with the smallest clock and, if the queue is not empty, processes the first message available. If the queue is otherwise empty, the LP blocks until a message arrives at the queue, which updates its clock value of the incoming link. Afterwards, the LP selects a new queue with the smallest clock and the procedure is repeated.

Since an LP may block on an empty link, deadlocks may occur in the case of a waiting cycle. The CMB mechanism avoids deadlocks by introducing the notion of *null messages*, which are for synchronization purposes only and do not represent real activities in the model. A null message is a promise about the earliest message that will arrive in the future. When an LP receives such a null message, it advances the clock value associated with the link, and, if possible, it progresses by processing events that are waiting in other queues. If processing is not possible, it propagates the time carried by the null message and other time advancements to its successors by sending out more null messages through its outgoing links. The essential part of this mechanism is determining the null message timestamp. This *lookahead* value defines the degree to which LPs can look ahead and predict future events.

Since each incoming link defines the lower bound for the next unprocessed event, a good measure for the lookahead value can be the minimum among all incoming links' clocks plus the LP service time. In fact, the lookahead represents a lower bound on the timestamp of the next outgoing message. Every time the LP finishes processing an event, it sends a null message on each of its outgoing links to report this bound. When an LP receives a null message, it calculates a new bound based on the information it receives and passes it to its neighbors, and so on. The lookahead can also be determined by the programmer statically. It has been shown that the larger the lookahead, the better is the performance of the algorithm [122] [123] [119]. In order to avoid deadlock, there is a constraint on the value of lookahead; it cannot be zero [50]. This restriction implies that certain types of simulations that require

zero lookahead cannot be performed by the CMB algorithm (e.g. queuing networks with service time of zero).

- **Variations of CMB**

The CMB algorithm can produce many null messages, degrading the performance of the simulation. Since its original implementation, numerous approaches have been proposed aiming at reducing the number of null messages. Here, some of the variations on the CMB approach that deal with this issue are presented.

The *demand-driven* null message protocol [35] avoids the aggressive distribution of null messages by enforcing LPs to send null messages only when they are asked to. All synchronization messages are of fixed size and independent of the number of processors. When an LP needs to process an event with timestamp t, but cannot do it due to timing constraints, it sends a timing request, reporting the sender's *id* and the requested time t to the neighboring LPs. The receiving LPs then inform to the sender LP if they can guarantee that they will not emit an event at a time earlier than the requested time t. There are three types of replies (which can be used to avoid repeated polling in the presence of cycles). The *yes* message indicates that the receiver LP has advanced to the requested time; the *no* message indicates that it is still lagging behind (resulting in another request to be made by the sender LP), and the *ryes* (reflected *yes*) message indicates that the receiver LP has conditionally reached t. The *ryes* are used to detect possible cycles and minimize the number of subsequent requests sent to the neighboring LPs.

Misra [36] and Peacock *et al.* [37] also revisited the CMB mechanism by imposing the idea of sending null messages on demand rather than after each event. Nicol and Reynolds [38] used a variation of this approach for distributed simulations with shared resources. Su and Seitz [39] investigated a family of variants of the basic CMB algorithm to speedup gate-level simulations on an Intel iPSC computer. They focused on reducing the volume of null messages by deferring sending outputs and packing the information into fewer messages.

Other approaches to null message generation, including *generation after a time-out* and *generation using stimulus nulls* were introduced in [40]. The purpose of the null message after a time-out algorithm is to reduce the system overhead of processing null messages by reducing the actual number of null messages transmitted between LPs. The null messages are transmitted only after a specified amount of real clock time, the *time-out* value. It was shown that when the time-out value increases, fewer null

messages are generated, thus reducing overhead. In contrast to the null message with time-out algorithm, the stimulus null variation added, rather than eliminated, null messages. Stimulus null messages are generated and transmitted after the execution of a given number of internal events, specified as a ratio between the events and the stimulus nulls. These nulls are in addition to any nulls normally generated, and they give the receiving LPs an earlier indication of time progression (when compared to the original CMB null message algorithm). Consequently, there is a greater potential to execute the simulation faster.

Although demand-driven protocols reduce the amount of null message distribution, in return, the delay associated with receiving null messages increases because two messages are required. The *carrier-null message* algorithm introduced by Cai and Turner [41] reduces the number of CMB null messages and it increases the lookahead ability by exploring the simulation network topology. A carrier-null message includes extra information, in particular, the message route. This carrier information is a record of all LPs visited by the carrier-null message since its creation. This information allows individual LPs to advance their simulation clocks while keeping the null message traffic low. The carrier-null message scheme only supports simulations with certain communication graphs such as those with nested cycles. Wood and Turner [42] extended the carrier-null message approach by proposing a generalized carrier-null method to support arbitrary graphs. In [43] the *null message cancellation* protocol was investigated, and the impact of the cancellation of spare null messages was examined. Under this protocol, a null message is discarded before being receipt if it is overrun by a message with a larger timestamp. The empirical results showed how the impact of null message cancellation is affected by the lookahead of the LP. Porras *et al.* [124] improved the CMB algorithm by using null message cancellation, simulation loop optimization, and multicasting techniques. Their algorithm, named *Simloop* reduces the number of null messages and improves the execution of messages by allowing simulation of multiple messages instead of a single message.

The *Critical Channel Traversing* (CCT) algorithm [125] extended the CMB algorithm with the addition of rules that determine when to schedule an LP for event execution. CCT attempts to schedule the LPs with the largest number of events that are ready to execute. This is accomplished through identifying critical channels. The CCT algorithm was implemented along with a simulation kernel called *TasKit*, designed for high performance simulation on small to medium sized shared memory multi-

processors. The algorithm provides multi-level scheduling by allowing scheduling large grains of computation even in very low granularity models. In [126], two new versions of the CCT algorithm were presented. The first one, called *simple sender side CCT*, differs from the original in the elimination of busy waiting. Consequently, it avoids the performance problems that can be caused by busy waiting. The second algorithm, called *receive side CCT,* uses a different strategy for updating channel clocks and scheduling objects connected to critical channels. Receive side CCT reported better scaling with respect to the connectivity of the model, but at the cost of additional overhead for low connectivity models.

Boukerche and Das [127] proposed a null message algorithm that reduced the overhead of null messages using load balancing. The synchronization protocol is a variation of CMB null messages combined with a load-balancing algorithm that assumes no compile time knowledge about the workload parameters. The algorithm is based on a process migration mechanism, and the notion of the *CPU-queue length*, which indicates the workload at each processor. In addition, they presented two variations of the algorithm: a centralized, and a multi-level hierarchical method.

Other null message reduction algorithms that have been proposed use a generic mathematical model to approximate the optimal values of the parameters that are directly involved in the performance of a time management algorithm [128][129]. Thomas *et al.* [130] proposed another null message reduction algorithm based on grouping and status retrieval by determining an optimum value of the lookahead.

There have been varied efforts trying to improve the lookahead computation. For example, in [131] the authors presented a method where the compiler automatically extracted information about the lookahead present in the application. The *lock-free* algorithm [132] is another conservative scheduling technique implemented for shared-memory multiprocessor machines, which uses *fetch&add* operations to avoid the inefficiencies associated with using locks. The authors show that compared with lock-based scheduling algorithms, the lock-free algorithm exhibits better performance when the number of logical processes assigned to each processor is small or when the workload becomes significant. However, due to the overhead spent for extra bookkeeping, only modest performance gain is achieved for a large number of logical processes. Solcany and Safarik [133] presented a user-transparent conservative parallel simulator that allows users to build simulation models with lookahead transparently. To do so, they analyze the conditions for cumulating the lookahead of entities inside the same LP, and using this information they derived a mechanism to calculate such cumulated lookahead based on the Dijkstra's

shortest path first algorithm. Chung *et al.* [134] proposed a scheme for the prediction of the software execution path in order to extend the lookahead computation for parallel multiprocessor simulation. They use templates for predicting the program execution path, which are generated by software analysis. Then, a processor model obtains the lookahead by evaluating the templates at simulation time. The proposed method aggressively extends the lookahead of null messages by executing the path prediction of the software application dynamically.

Other studies have devised and compared variants to the CMB algorithm by evaluating the performance of the algorithms for inefficiencies and overhead. Park *et al.* [135] compared the performance and scalability of a *lazy null message* algorithm with global reduction approaches. They suggested that, for scenarios simulating scaled network models with constant number of input and output channels per LP, the lazy null message algorithm offers better scalability than efficient global reduction based synchronous protocols. Bagrodia and Takai [140] studied the performance of three diverse conservative algorithms implemented in Maisie: a synchronous algorithm (conditional event), an asynchronous algorithm (with null messages), and a hybrid algorithm (ANM - *Accelerated Null Message*) that combines features from the preceding algorithms. Maisie models were developed for standard queuing network benchmarks, and various configurations of the model such as model connectivity, computation granularity, load balance, and lookahead were executed using the three different algorithms. Song *et al.* [141] discussed an empirical study of conservative scheduling by examining several heuristics that help identifying critical events. They presented a performance study comparing several scheduling algorithms based on LP's next event timestamp, safe time, or local simulation clock. In [142], a performance evaluation of a CMB protocol was investigated. They analyzed the performance and behavior of each logical process, and showed that, in the same simulation, different LPs can show different performance. The analyses were performed by adding software monitors to the simulation code. The monitors computed some metrics whose values were used to estimate the performance of each logical process in execution time.

- **Deadlock Detection and Recovery**

Another approach for conservative synchronization is to allow deadlocks to occur, and to provide a mechanism to detect and recover from them. This approach eliminates the use of null messages and the overhead associated with their communication traffic. Deadlock is broken by allowing to processing the

event with the smallest timestamp. Chandy and Misra [46] proposed an asynchronous distributed simulation approach via a sequence of parallel computations. Their approach did not use a global clock nor did they use a single process to drive the simulation. Rather, to avoid bottlenecks, they use a special process, called the *controller*, which synchronizes the LPs when the simulation deadlocks. Under that scheme, the simulation is divided into a sequence of computations: the *parallel phase* and the *phase interface*. The controller is only responsible for detecting the termination of one phase and initiating the next one. Other seminal deadlock detection mechanisms were discussed in [44][45][36].

One approach to determine safe events is to perform a set of distributed computations across all the LPs. The *Critical Path Analysis* algorithm (CPA) [143][144] generates an acyclic event-dependency graph by tracing the events in the simulation. The critical path is calculated as the longest event path in the event graph, and its related time is considered as the lower bound on the execution time of the simulation. Srinivasan and Reynolds [146] mention that the conservatism of the CPA is relaxed in the sense that it only considers the dependency among events, requiring each LP to know exactly what the next event is, and when does it arrive. This is not ideal since events in a parallel simulation are unpredictable at runtime. The *State Causality Analysis* algorithm (SCA) [149] overcomes the limitation of CPA by focusing on the dependency of the logical process states, rather than on unpredictable events. This technique takes into consideration the effect of many algorithm independent factors, such as lookahead, I/O overhead, physical transfer delay, processor speed, and event distribution.

Groselj and Tropper [47] proposed the *time-of-next-event* (TNE) algorithm for situations where multiple LPs reside on a single processor. TNE relies upon a shortest-path algorithm and increases parallelism by computing the largest lower bound of all LPs independently on every processor. The advantage of this approach is that it does not rely on message passing to distribute the lookahead information; rather, the algorithm is executed on each LP independently. A deadlock recovery algorithm is used to resolve inter-process deadlocks. Boukerche and Tropper [48] presented an extension to TNE, namely SGTNE (Semi Global TNE), whose goal was to exploit lookahead information from both the local and the neighbor LPs (unlike TNE where only LPs within a process are used to unblock an LP). Consequently, SGTNE outperforms TNE, as it allows a higher degree of parallelism as well as avoiding inter-process deadlocks.

In order to increase the set of safe events, a global reduction computation can be used to derive a *Lower Bound on the Timestamp* (LBTS) among the events that can be received by a LP in the future (i.e., the minimum timestamp of the next future event in the entire simulation system). With such information, each LP can safely process any pending events with a timestamp smaller than the LBTS value [150][118][152][151]. Curry *et al.* [153] studied the performance of asynchronous conservative PDES algorithms by examining the performance of systems based on a sequential Centralized Event List (CEL) and compared those with that of CMB. In their experiments the performance of a CMB-based system was compared with three CEL implementations namely the heap, splay tree, and calendar queue for a particular workload model. The results showed that both the number of instructions executed and the cache behavior have significant impact on the performance, and the superior cache performance was able to make up for a larger number of instructions executed.

- **Conservative Time Windows**

Lubachevsky [55] was the first to introduce the idea of a *moving time window* to determine the set of safe events that can be executed in parallel. Using this approach, a lower edge is defined for the window, based on the minimum timestamp of all the unprocessed events, and a window size. Any event whose timestamp is within the window size is eligible for processing. Although this mechanism eliminates the overhead associated to the search for safe events, an important success factor is the window size. A small window size would decrease parallelism while a large window size would result as if there was no time window at all. An appropriate window size can be obtained either from the programmer, the compiler, or at runtime by monitoring the simulation [14]. The *Moving Time Windows* (MTW) protocol [63] is a relaxed version of Lubachevsky's approach where global windows are adjusted dynamically and the events within a window are assumed to be parallel. When an event with a timestamp earlier than the LP's clock is received, an anomaly occurs. Ayani and Rajaei [64] show that better parallelism can be achieved using the *Conservative Time Window* (CTW), where the global ceiling of the window is eliminated, and allowing different windows to have different sizes.

Lemeire and Dirkx [154] proposed a hybrid synchronization technique that combined the asynchronous CMB algorithm with CTW to maximize the lookahead capabilities of a model by using lookahead accumulation. The algorithm tries to maximize the performance by optimally tuning two attributes of the model: granularity and lookahead. Granularity is defined as the amount of computations

between two synchronization points. The mechanism tried to improve the performance by maximizing granularity and thus reducing the communication overhead. This is done aggregating all the dedicated LPs in a processor, and forming a multiprocess, which can be simulated sequentially on each processor. The algorithm exploits maximum performance by accumulating lookahead information, and computing and using the global lookahead of the multiprocess. Lin *et al.* [148] presented a method called *micro-synchronization* to exploit the parallelism inside each LP. Unlike the methods of lookahead accumulation [154] and local time warp [155][156], this technique keeps the traditional use of lookahead among LPs unchanged, while imposing a relaxed sequential event scheduling inside each LP, which can statistically increase the lookahead.

- **Other Conservative Protocols**

Numerous conservative protocols have been proposed, and some of them are presented here. Several of these protocols were defined as a combination of synchronous approaches with event-driven clock progression. The idea is to divide the computation into cycles, in which one first determines the *safe events*, and then processes all those events. Ayani [54] used the concept of distance between LPs to determine the safe events. Under this scheme, the distance gives the minimum time it takes for an event in one LP to directly or indirectly affect another LP (similar to a shortest-path algorithm). This draws a bound on when should an LP expect an event from its neighbors. Likewise, Lubachevsky's *bounded-lag* algorithm [55] took advantage of the propagation delay between LPs to exploit lookahead [60]. The algorithm uses a time interval (called the time lag) in order to compute a set of LPs that can affect a given LP within the lag interval.

The *conditional event* protocol [61] categorized events into two types: *definite*, and *conditional*. Definite events are scheduled locally, while conditional events require communication among all LPs to determine the earliest conditional event globally (which is then converted into a definite event). Nicol [62] used a similar idea based on *synchronization barriers* and introducing time windows with the restriction that all events within a window are safe to process. Similar to the conditional event approach, global computations are conducted to determine the time of next synchronization point.

Nicol and Liu [157] proposed a composition strategy by combining the synchronous barrier synchronization (global) with channel scanning (local) synchronization protocols to allow tailoring the synchronization mechanism to the model being simulated. Their attempt to combining synchronous and

asynchronous approaches allows using one method in part of the model where the other method is weak. Using this approach, the effect of high connectivity is limited by making most of a node's channels synchronous. On the other hand, by making channels with low lookahead asynchronous, the effect of unusually low lookahead is limited.

The success of all the conservative synchronization algorithms presented in this section largely depends upon the ability to predict the future, in terms of the lookahead or LBTS [14][117]. In order to achieve acceptable performance, this, in turn, requires an effective use of *application-specific* information such as the topological structure of the network of LPs, the characteristics of the communication network, and the underlying model behavior. A side effect of this requirement is that a seemingly minor changes to the model could affect the simulation performance dramatically, hindering the robustness of the application [26]. Perhaps the most prominent drawback of conservative approaches is that they often cannot fully exploit the potential parallelism available in a simulation [26], especially when the estimated lookahead or LBTS values are overly pessimistic and when global synchronizations are performed too frequently in the synchronous execution mode. The optimistic synchronization algorithms introduced in the following sections do not have any of these problems. Nonetheless, when the application characteristics are favourable, conservative approaches can reduce the execution time significantly with moderate memory consumption (see, e.g., [158][159][160][134]).

2.6.2 Optimistic Synchronization Algorithms

Jefferson's Time Warp (TW) mechanism [15] was the first (and remains the best known) optimistic synchronization protocol. A TW simulation uses virtual time to model the passage of time in a simulation, and it is driven by a set of Time Warp Logical Processes (TWLPs), each of which has its own Local Virtual Time (LVT) and processes events autonomously without explicit synchronization. Aside from LVT, another fundamental synchronization concept in optimistic simulations is the notion of Global Virtual Time (GVT) which is defined as the earliest time tag within the set of unprocessed pending event in the entire simulation. TWLPs differ from ordinary LPs, (such as those used in sequential and conservative simulations), in the way in which the states and events are managed. Specifically, an ordinary LP maintains only one copy of its state (i.e., its current state), which is updated repeatedly during the event execution. Furthermore, an ordinary LP does not need to keep a record of

past input and output events, allowing the events to be reclaimed immediately after execution. In contrast, each TWLP needs to manage a history of its past events (both input and output) and states. This includes three data structures: an input queue that contains the recently arrived input events (sorted in receive timestamp order), an output queue that holds *anti-messages* that are negative copies of the recently sent output events (sorted in send timestamp order), and a state queue that stores the recent states of the TWLP. The data of these queues are kept until it is guaranteed that no event with a smaller timestamp can ever be received by any TWLP in the system.

A *causality error* is detected when an event with a timestamp earlier than the LVT of the receiving TWLP is received. Such an event is referred to as a *straggler* event. TWLP recovers from the causality error by undoing the effects caused by a straggler event. This recovery operation is known as *rollback.* As a result, the state of the TWLP is restored to the last one that was saved prior to the arrival of the straggler event. Since incorrect messages may have spread to other TWLPs, they must be cancelled as well. Cancellation of such messages is performed by sending anti-messages, which are negative copies of those output messages that were saved in the output queue. Arrival of anti-messages at a TWLP causes further rollback if the timestamp of the anti-message is less than the LP's LVT. Therefore, anti-messages (just as positive stragglers) would cause rollbacks and further propagation of anti-messages. These are referred to as secondary rollbacks which result in cascaded rollbacks flooding the simulation system with anti-messages.

Jefferson's original Time Warp has been revised and optimized several times to reduce operational overhead and improve performance of optimistic simulations. A wide variety of techniques and optimization strategies have been proposed in the literature to deal with the well-known challenges of optimistic TW-based PDES. Issues like memory management, fossil collection, memory stall recovery, checkpointing, cascaded rollback, and event cancellation remain to be top challenges of this field. In the following points, a brief state-of-the-art is presented for each category to summarize the most relevant previous contributions made towards tackling these challenges.

- **Memory Management**

Due to date and state saving and rollback operations, optimistic parallel simulation requires much higher memory space compared to sequential simulation. Different memory-conserving techniques have been proposed to reduce memory consumption in optimistic simulations [14][161][162][163][164].

- **Fossil Collection**

Fossil collection frees up memory occupied by historical data (input/output events, LP states), thus reducing memory stalls in optimistic simulations. Memory stall occurs when the simulation is halted due to memory exhaustion. Different techniques to achieve efficient fossil collection have been proposed. In [165][166] an optimistic mechanism is introduced where fossil collection decisions are made based on locally predicted information without estimating the global state of the simulation. Chetlur and Wilsey [167] propose a fossil identification mechanism that uses an extended time stamp structure, known as plausible total clock, instead of using the global time of the simulation. The PAL fossil collector technique is an enhanced mechanism that reduces the fossil collection cost by prioritizing the LPs based on the amount of fossil they carry [168].

- **Memory Stall Recovery**

Different approaches have been explored aiming at recovering from memory stalls. Among these, techniques such as *cancelback* [161][169][170], *artificial rollback* [163][171], and *pruneback* [172] have shown promising results. Both cancelback and artificial rollback require a globally shared pool of memory that is accessible by all LPS in the system. However, they differ in the way they deal with situations where the pool runs out of memory. With cancelback mechanism, memory-acquiring requests are returned to their originating senders, thus forcing sender LPs to rollback and release memory. In artificial rollback, those LPs with the greatest LVTs are forced to artificially rollback, releasing memory. Under the pruneback mechanism, memory reclamation occurs by targeting past states, thus forcing LPs to release portions of memory occupied by the state queues.

- **Checkpointing**

An alternative approach to memory stalls is checkpointing. Such techniques reduce state-saving overhead by enforcing LPs to save fewer historical data. There are two major types of checkpointing algorithms: *infrequent state-saving* or *periodic state-saving* techniques [173][174][175][176][177][178][179], which focus on reducing the number of states saved in a simulation [180][181][182][183]; and *incremental state-saving* techniques, which attempt to reduce the amount of data that need to be saved in each state (see, e.g.,). Techniques that combine different state-saving mechanisms are proposed as well [184][185].

- **Event Cancellation**

Rollback operation significantly affects the performance of optimistic simulations. An efficient cancellation mechanism can improve rollback overhead. The original TW protocol adapts an *aggressive cancellation* scheme where anti-messages are sent immediately when a LP rolls back. The *lazy cancellation* technique [186][187] improves rollback efficiency by reducing the communication overhead of event cancellation. Under such mechanism, anti-messages are only sent when the necessity is verified by the LP. Another technique, referred to as *throttled lazy cancellation* [188], slows down the spread of potentially incorrect computation resulted from re-evaluation during lazy cancellation operations. On the other hand, the *early cancellation* scheme [189] cancels false messages in place in the buffer of a programmable network interface controller, which in return would require a specialized hardware. The *proactive cancellation* and the *batch-based cancellation* algorithms [190][191][192] improve cancellation performance by capturing the causal relationship between events.

2.6.3 Parallel and Distributed Environments

A number of environments have been developed in the past, which provide numerous services to building parallel/distributed simulation systems by supporting optimistic, conservative, or hybrid synchronization strategies. Examples of such environments are: **YADDES** (Yet Another Distributed Discrete Event Simulator) [65], **SPEEDES** (Synchronous Parallel Environment for Emulation and Discrete Event Simulation) [66], **WARPED** [67], **HLA** (High-Level Architecture) [67], **WarpIV** [69], and **μsik** [70]. Park and Fujimoto [2006] proposed a **Master/Worker** (MW) paradigm for executing large-scale parallel discrete event simulation programs over network enabled computational resources. The MW depicts a client/server architecture where clients repeatedly download state vectors of logical processes and associated message data from a server (master), perform simulation computations locally at the client, and then return the results back to the server. The advantages of such approach over conventional PDES include support for execution over heterogeneous distributed computing platforms, load balancing, efficient execution on shared platforms, easy addition or removal of client machines during execution, simpler fault tolerance, and improved portability. The *Aurora Parallel and Distributed Simulation System* (Aurora) [136] is a prototype implementation of the MW. Several extensions and improvements to Aurora were presented later on including a scalable version for parallel

discrete event simulations on desktop girds [137], an optimistic time management compliant for public-resource computing infrastructures and desktop grids [138], and a version implemented for metacomputing systems [139].

2.7 Parallel DEVS and Cell-DEVS Simulation in CD++

CD++ [Wai02], a M&S toolkit originally developed by Wainer [101], is an open-source, object-oriented environment that implements both P-DEVS and Cell-DEVS theories in C++. The tool includes facilities to build DEVS and Cell-DEVS models. DEVS atomic models can be programmed and incorporated into a class hierarchy. Furthermore, coupled models can be defined using a built-in specification language. Therefore, coupled and Cell-DEVS models need not to be programmed, rather the tool provides a specification language that defines the model's coupling, the initial values, the external events, and the local transition rules for Cell-DEVS models. CD++ also includes an interpreter for Cell-DEVS models. The language is based on the formal specifications of Cell-DEVS. The model specification includes the definition of the size and dimension of the cell space, the shape of the neighborhood and the border. The cell's local computing function is defined using a set of rules with the form ***POSTCONDITION DELAY { PRECONDITION }***. These indicate that when the *PRECONDITION* is met, the state of the cell changes to the designated *POSTCONDITION* after the duration specified by *DELAY*. If the precondition is not met, then the next rule is evaluated until a rule is satisfied or there are no more rules.

Over years, CD++ has been evolved and extended using different middleware technologies to support simulation on varied platforms [19] [109] [20] [21][34][22][23][24][25] [12]. In particular, **Parallel CD++** (or **PCD++** for short) simulation engine allows optimistic TW simulation of P-DEVS and Cell-DEVS models on Linux-based distributed-memory multiprocessor cluster systems [21]. It is built on top of the WARPED simulation kernel [67] [111] and relies on the Message Passing Interface (MPI) libraries [112] for inter-node communication. The optimistic synchronization protocol of PCD++ was revised in [34] and a new protocol, namely, **Lightweight Time Warp (LTW)** was integrated into the tool which significantly improved the performance [32]. In PCD++, a model is partitioned at the atomic level, and each abstract DEVS processor is implemented as a LP. The resulting LPs are then mapped to a set of physical processors (or nodes) for parallel execution [21]. The optimistic synchronization of PCD++ simulator was replaced with the conservative protocols proposed in this

research resulting in a purely conservative simulator, called **Conservative CD++** (**CCD++** for short) [87] addressing several important issues arising in DEVS-based conservative simulations, providing a testbed for the research presented in this dissertation.

CCD++ consists of a layered architecture where each layer only depends on the layers below it, as illustrated in Figure 6.

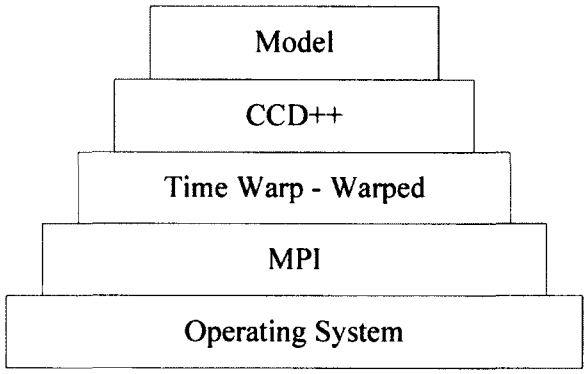


Figure 6. Layered Architecture of CCD++ Simulator [21]

The operating system resides on the bottom of the architecture. CCD++ uses Linux Operating System as the underlying platform for high-performance parallel and distributed computing. Above the Operating System lays the MPI, the standard specification of message-passing library for high-performance communications on parallel machines and workstations clusters. The Operating System with the use of MPI provides the communication infrastructure for the CCD++ simulator. The WARPED [67][111] kernel serves as a configuration middleware that provides services for defining different types of processes (simulation objects), memory management, I/O, and file handling. Simulation objects mapped on a physical processor are grouped by an LP. On top of the WARPED kernel resides the CCD++ simulation engine source code. Finally, the top most layer is the DEVS or Cell-DEVS model created in CD++.

To reduce communication overhead [73] [109], similar to PCD++, CCD++ adopts a flat structure that creates three types of DEVS processors on each node: a *Node Coordinator* (NC), a *Flat Coordinator* (FC), and a set of *Simulators*. Doing so eliminates intermediate Coordinators in the LP hierarchy, reducing the communication cost. The *Simulator* represents an atomic DEVS model, where the *Coordinator* is paired with a coupled model. The *Simulator* is in charge of invoking the atomic model's *transition* and *external event* functions. On the other hand, the *Coordinator* has the

responsibility of translating its children's output events and estimating the time of the next imminent dependent(s). At the beginning of the simulation, one LP is created on each machine (physical process). Then, each LP will host one or more DEVS processors. Only one *NC* is created on each machine and acts as the local controller on its hosting LP. The *NC* is the parent coordinator for the *FC* and routes remote messages received from other remote *NC*s to the *FC*. The *Simulators* are the child processors of the local *FC* representing the atomic components of DEVS and Cell-DEVS models. The *NC* is a local central controller and the final destination of inter-node messages, whereas the *FC* routes messages between its child *Simulators* and the parent *NC*. The DEVS processors exchange two categories of messages: *content* and *control*. The first category includes the *external* (*x*) and the *output* (*y*) messages, and the second includes the *initialization* (*I*), *collect* (@), *internal* (*), and *done* (*D*) messages. *External* and *output* messages exchange simulation data between the models, *collect* and *internal* messages trigger the output and the state transition functions respectively (in atomic DEVS models), and *done* messages handle scheduling by carrying the model timing information. The simulation is executed in a message-driven manner. Figure 7 illustrates CCD++ processors and the messaging paradigm.

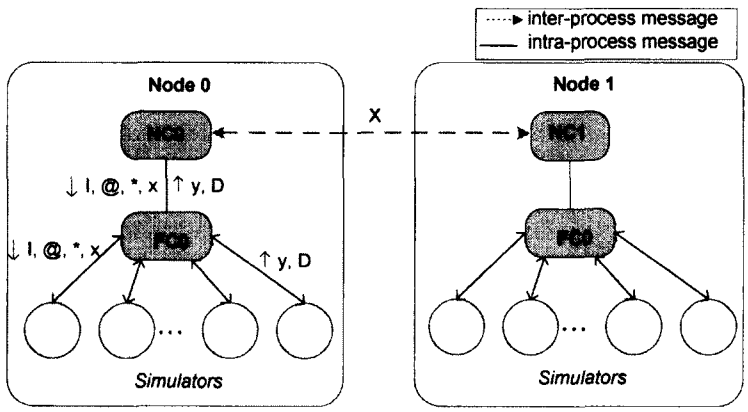


Figure 7. CCD++ Flat Architecture and Message-Passing Paradigm

The simulation starts by *NCs* sending an (*I*, *t*) message to their child *FCs*. At any virtual time, the message flow among the LPs is organized into a multi-phased structure that includes an optional *collect* phase and a mandatory *transition* phase, which in turn may involve multiple rounds of computation to execute state transitions incrementally. The *collect* phase starts with a *collect* message sent from the *NC* to the *FC* and ends with the following *done* message received by the *NC*. The *transition* phase begins with the first *internal* message sent from the *NC* to the *FC* and ends at the last *done* message received by

the *NC* at that time. The *transition* phase is mandatory for each individual simulation time. The *output* functions in the imminent atomic models are invoked during *collect* phases, while the state transitions for the atomic models are performed in the *transition* phases (as defined in P-DEVS formalism).

2.7.1 Event-Processing Algorithms

Based on the flat LP structure, this section briefly describes the parallel CD++ event-processing algorithms defined for the *Simulators*, the *FC*, and the *NC* respectively. In the following discussion, the form *(type, t)* is used to denote a message of *type* that has a receive time of *t*. The send time stamp of an event is by default the current virtual time.

- **Simulator event-processing algorithm**

The Simulator algorithm for initialization message is defined as follows:

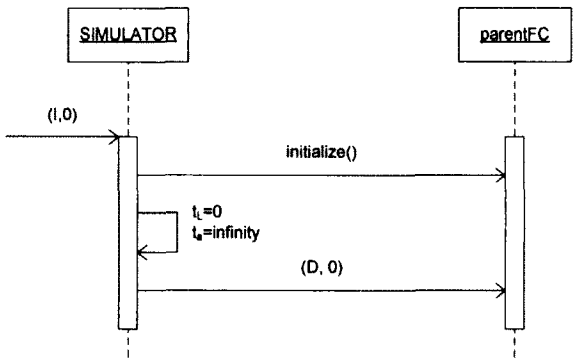


Figure 8. Simulator Algorithm for (*I*, *0*)

As defined in DEVS formalism, two variables are used in the simulator to record its current simulation time (t_L) and the value of *sigma* (ta). Using these two values, the value of *absolute next time* (denoted as t_N) is calculated as t_L + ta. Upon receiving the initialization message, (*I*, 0), the Simulator resets t_L to the timestamp of the message, therefore the Simulator's virtual time now is equal to zero. Then, the simulator initializes the variables defined in its associated atomic model, and after that, it informs its parent FC of the value of t_a by sending a *done* message stamped with time 0.

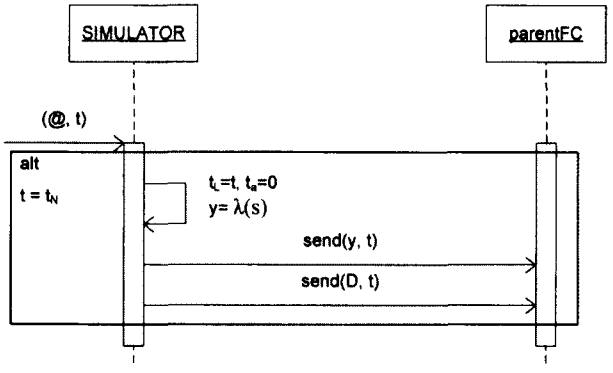


Figure 9. Simulator Algorithm for (@, *t*)

When a (@, *t*) message is received, the Simulator invokes the output function (λ) of the atomic model and as a result an output message (*y*, *t*) is sent to the FC. After this, the Simulator will send (*D*, *t*) to the FC with $t_a = 0$ to indicate that it is imminent.

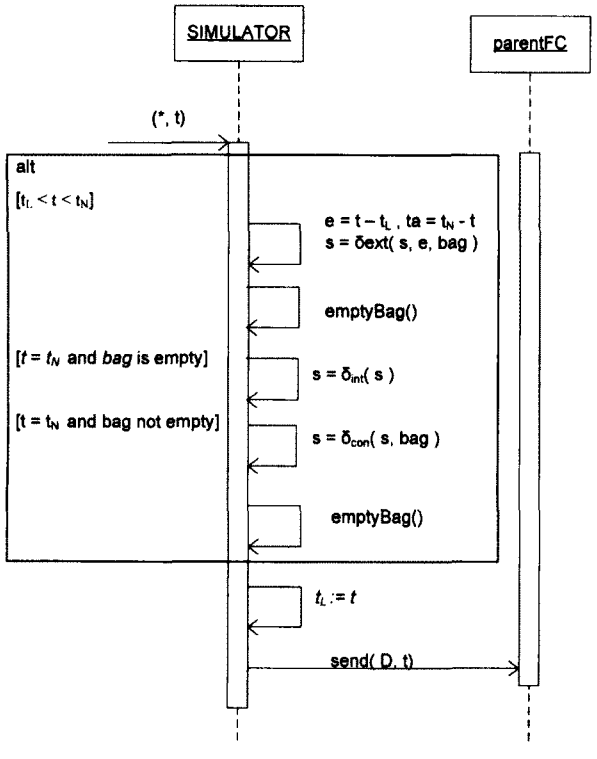


Figure 10. Simulator Algorithm for (*, *t*)

Following the collect message, a (*, t) will arrive to trigger internal/external/confluent function of the atomic model depending on the timing of the message and the status of the Simulator's message bag.

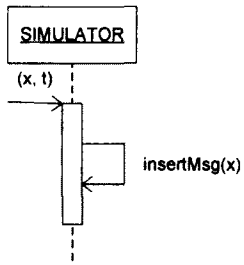


Figure 11. Simulator Algorithm for (x, t)

The last message that may arrive at the Simulator is (x, t) which is simply inserted into the Simulator's message bag. Note that, only external messages with identical timestamp can be inserted into the message bag at a given simulation time. Before adding further messages with a different timestamp, the existing messages must be processed and the bag be cleared in the receive function for internal message. In other words, an internal message will always arrive in between two consecutive batches of external messages.

- **Flat coordinator event-processing algorithm**

The FC, sitting in between the NC and the Simulators, performs three tasks: synchronizing the execution of all child Simulators, routing messages exchanged among its children, and delivering to its parent NC those messages that are sent from its children to the environment or to other remote Simulators. To accomplish the first task, the FC finds its imminent children with the minimum absolute next time and records them in a structure called *synchronize set*. It also uses a variable, *doneCount*, to keep track of the number of done messages it should receive from its children. This variable is used to implement a simple barrier. The FC only passes control to its parent NC after these children (the number is given by *doneCount*) have finished their previous computation. The other two tasks rely on the model coupling information that is loaded into the *main administrator* of the simulation administration facility during the bootstrap operation.

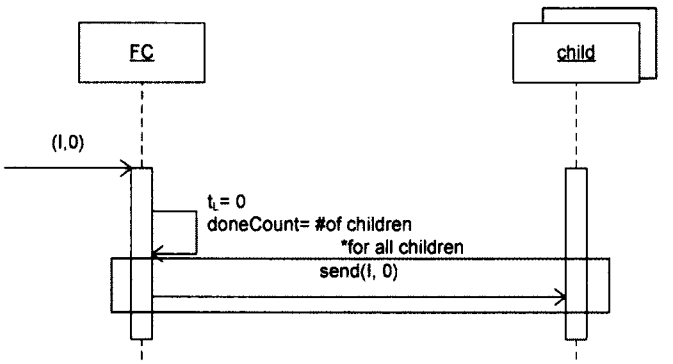


Figure 12. FC Algorithm for (*I*, *t*)

When (*I*, 0) is received, the FC records the total number of its children in a variable named as *doneCount* then forwards the (*I*, 0) message to each child. After this, the FC waits for all its children to respond to this initialization by sending back a (*D*, 0). The FC will only pass the control over to the NC if all its children have finished their previous computation and have sent done messages as notification messages.

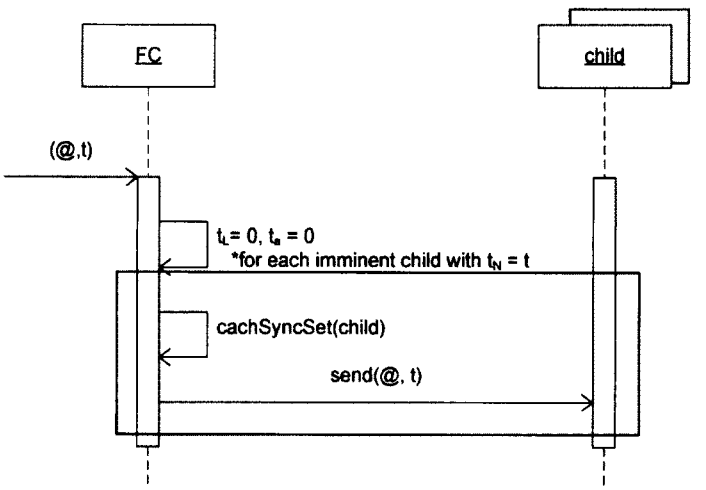


Figure 13. FC Algorithm for (@, *t*)

Upon receiving a (@, *t*) message, the FC forwards it to all imminent Simulators and will keep a record of this for later use (to know which children need to do state transitions when (*, *t*) is received).

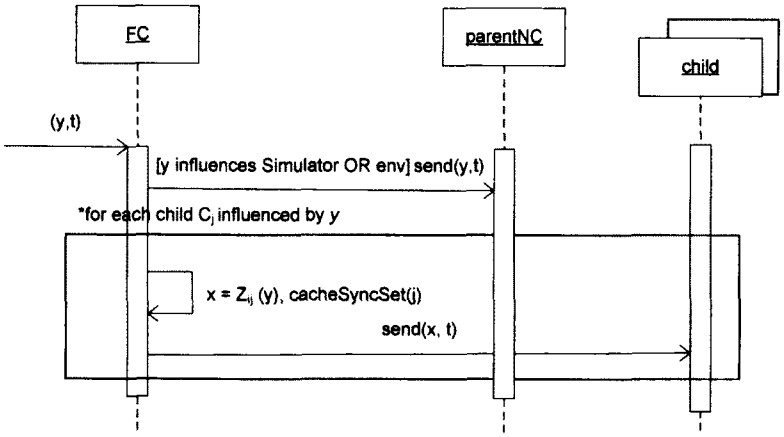


Figure 14. FC Algorithm for (y, t)

Moreover, when (y, t) is received, the FC searches the model coupling information to find out the correct destination. The destination is either an input port on an atomic model, or an output port on the topmost coupled model.

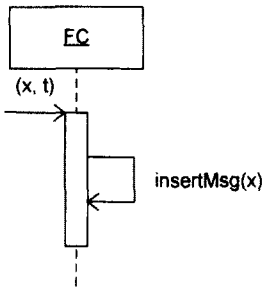


Figure 15. FC Algorithm for (x, t)

In case of receiving (x, t) message, the FC will simply insert the message into its message bag.

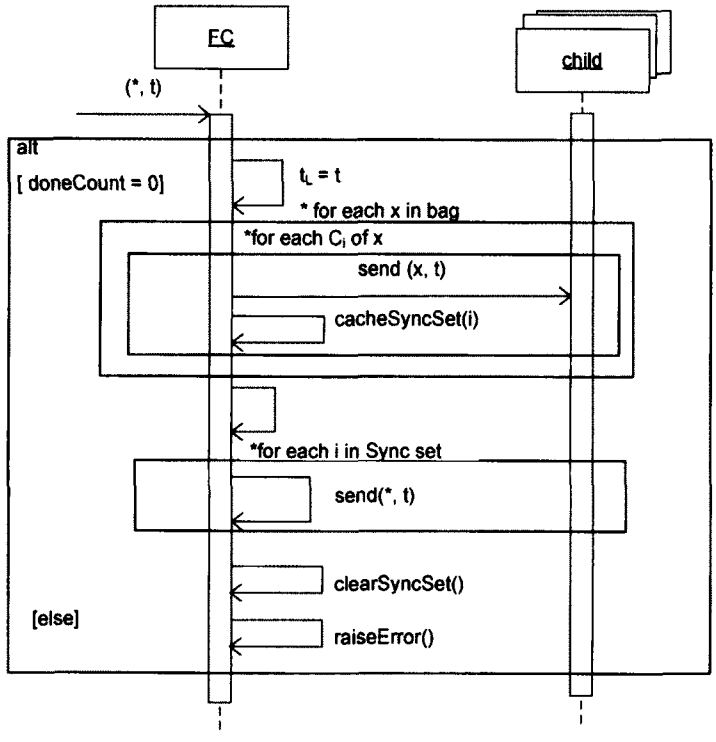


Figure 16. FC Algorithm for (*, *t*)

Upon receiving (*, *t*) message, the external messages inside the FC's message bag are flushed to the local receiving Simulators. This will trigger the imminent Simulators to perform a state transition.

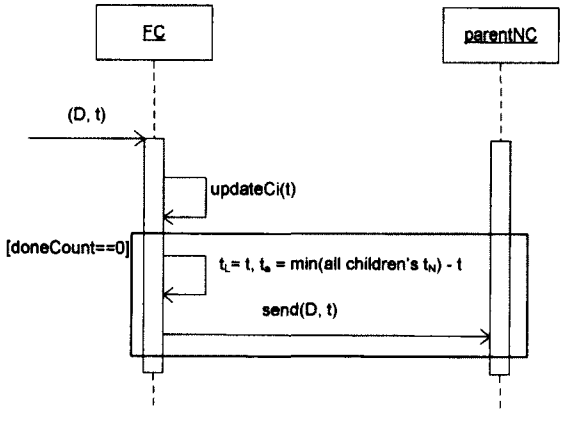


Figure 17. FC algorithm for (*D*, *t*)

Finally, when a (*D, t*) message is received from a child Simulator, the FC updates the child's t_N to the sum of the current simulation time and the *sigma* value carried by the received (*D, t*) message.

- **Node coordinator event-processing algorithm**

Each LP has one NC that acts as the local central controller in charge of the sequential simulation on the hosting machine. It has a single child, the FC underneath. The NC plays a very important role in the simulation as summarized below:

1) It takes care of the inter-LP communication among the Simulators. The messages exchanged between the NCs is handled using a special structure, the *NC Message Bag*.
2) It is responsible for handling the external events from the environment that are known prior to the start of the simulation and are scheduled by the modeler using a text file, namely *EV file*. These external events are loaded into the NCs during the bootstrap operations by the *main administrator*. Each NC uses a structure called *Event List* to hold those external events it needs to handle during the simulation. Events in the structure are sorted so that they can be processed in increasing timestamp order. The NC uses a pointer called *event-pointer* to reference the first event that has not yet been sent out. Initially, this pointer points to the first event in the list.
3) It synchronizes the activities of all local processors and drives the simulation on the hosting LP. The local simulation time is advanced by the NC based on three factors: the external events in its Event List, the external messages received in its NC Message Bag, and the closest state transition time provided by the FC.
4) It manages the flow of control messages for the local Simulators in line with the Parallel DEVS formalism. For example, the formalism requires that the output operation must take place just before the state transition in imminent Simulators. Hence, the NC must ensure that the collect message, which triggers the output operation, will be received by imminent Simulators *before* the internal message, which results in the state transition. The correct sequence of these control messages is manipulated using a flag, namely *next-message-type*, which is defined in the state of the NC. It may have a value of collect (@) or internal (*),

corresponding to the type of the control message that will be sent out by the NC in the next simulation cycle. The initial value of the flag is set to @.

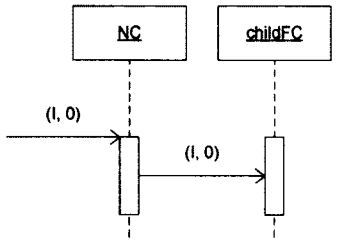


Figure 18. NC algorithm for (*I*, *0*)

Upon receiving (*I*, 0), the NC simply forwards it to the child FC.

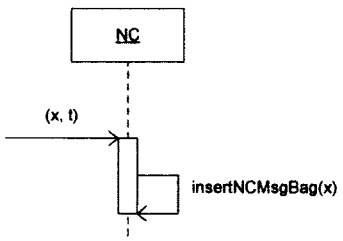


Figure 19. NC algorithm for (*x*, *t*)

In case of receiving (*x*, *t*), NC will insert this message into the *NC Message Bag*. These external messages contain values sent from remote Simulators to local ones.

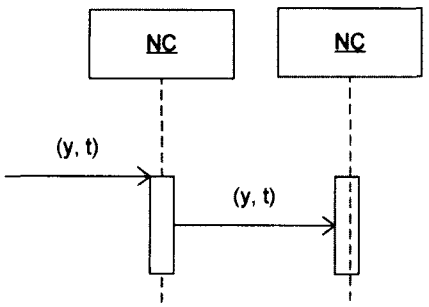


Figure 20. NC algorithm for (*y*, *t*)

When (y, t) is received, the NC simply forward it to the destination processor which happens to be a remote NC. Finally, reception of a (D, t) message by the NC from a child FC indicates that this is a response to a control message that was previously sent out by the NC.

2.8 DEVS-based Parallel and Distributed Simulation

A number of studies have been devoted to DEVS-based parallel simulations including: DEVS-C++ [71], DEVS/CORBA [72], DEVSCluster [73], DEVS/P2P [75], DEVS/RMI [76], DEVSim++ [77], and P-DEVSim++ [78]. In [79] a distributed simulation strategy for DEVS is presented which combines conservative and risk-free optimistic strategies. Nutaro [80] presented an implementation of the parallel DEVS simulation protocol that uses a modified Time Warp optimistic algorithm for a shared memory multiprocessor [81]. In terms of conservative DEVS-based simulations, there has been a large body of work proposed in the literature by integrating DEVS with HLA [68], allowing DEVS tools to use the synchronization services provided by HLA. Based on the specifications of HLA, DEVS atomic components are defined as HLA federates communicating by exchanging messages through the Run Time Infrastructure (RTI). In [82] the first integrating algorithm of DEVS models into a HLA-compliant environment was proposed, which was based on the classical CMB synchronization mechanism using the conservative algorithm provided by HLA. However, this approach was prone to deadlock which was later resolved in [103]. Zacharewicz *et al.* investigated several approaches to improve lookahead computation in the G-DEVS/HLA environment. In [104][105], they developed an algorithm for G-DEVS federation execution with a conservative synchronization mechanism using a positive lookahead value gained from the HLA **Least Incoming Time Stamp** (LITS) value. In [106], they present a new HLA lookahead computation algorithm which uses the *Dijkstra* path search in a graph to compute the different values of state variables and mathematical function analysis to determine the lookahead for the model states.

HLA requires complex system's configuration which needs considerable efforts. In order to fully benefit from HLA, the user needs to have an invasive knowledge which requires a considerable amount of time. More importantly, one of the major issues of HLA-based parallel simulation is the extensive performance overheads incurred due to the runtime infrastructure (RTI) software that links the simulators, especially for DEVS-based simulators that have fine-grained event computations. These

issues and the many advantages of WARPED kernel (such as ease of use, simple configuration, light-weight oriented middleware, and many more) were the motivation to use WARPED as the middleware in both PCD++ (optimistic CD++) and CCD++ (conservative CD++) simulators.

There has been some research done outside the HLA domain. For instance, Zeigler [8] introduced conservative parallel simulation of DEVS models based on the classical CMB approach with deadlock avoidance and the Yaddes [108] algorithm. The principal idea behind this method is to maintain a network of correlated *Earliest Output Time* (EOT) and *Earliest Input Time* (EIT) estimates, which matches the output-to-input coupling structure of the DEVS coupled model. The EOT/EIT estimates represent the time information distributed via null messages. Under this scheme, the lookahead calculation is performed at each DEVS *Simulator*, by looking at input and output ports. However, there are two limitations associated with this technique: a) a large number of EIT and EOT computations are required (since the algorithm is implemented at the *Simulator* level, the overhead increases as the model size grows; one *Simulator* is needed per atomic component); b) a large number of null messages are sent among processors, since both EIT and EOT must be distributed, as opposed to sending only one type of information (i.e. only lookahead). Besides, Himmelspach *et al.* [193] introduced a different conservative simulation algorithm for efficient distributed simulation of P-DEVS models. The algorithm makes use of Java threads and performs sequential execution among the entities on each computing node while the simulation is distributed over remote nodes.

There are three approaches to mapping the DEVS formalism into PDES protocols as illustrated in Table 1. Two of these approaches are specializations of the more generic protocols (i.e., conservative and optimistic schemes). The third is the direct mapping into a simulation algorithm (e.g., the original DEVS [8] and the classical P-DEVS protocols [11]).

Table 1. Comparison of PDES Approaches for DEVS

Scheme	Approach	Overhead	Advantages	Disadvantages
Chandy-Misra Conservative (LBTS, GLM, and CMB protocols implemented in CCD++)	- Process events in strict time stamped order - Causality preservation with deadlock avoidance	- Null Messages	- Relatively simple to implement - Dynamic and low-cost lookahead and LVT computation - Low memory usage	- Does not exploit all parallelism - Not intrinsically load balancing
Original DEVS	- Propagation delay for lookahead	- Global Minimum time synchronization	- Easy to implement	Simultaneous Events are Problematic
Time Warp Optimistic (LTW protocol implemented in PCD++)	- Permits Causality Violation - Detects violations and recovers using rollback	- State, Message Saving, Fossil Collection -Anti-messages -GVT Computation	- Can exploit higher level of parallelism	- Complex logic is difficult to implement and verify - High memory consumption
P-DEVS	Riskfree and strict causality adherence	- Global Minimum time synchronization - Simultaneous output collection	- Easy to implement - Exploits Simultaneous Events	- Does not exploit all parallelism - Not intrinsically load balancing

2.9 Performance Evaluation of PDES Environments

Performance evaluation of PDES protocols is a complex task. PDES environments are commonly tested and compared using synthetic or real models. There have been many empirical and analytical studies on the performance of PDES algorithms. Most of them generally try to evaluate a particular algorithm or an implementation of it. For instance, conservative time synchronization protocols have been evaluated using synthetic application benchmarks simulating a variety of topics including: queuing networks [140][14][202], communication networks [201], or electronic circuits [203][201]. Similarly, parallel discrete event simulation performance evaluations using an optimistic synchronization protocol have

been conducted using synthetic benchmarks simulating communication networks [201][204], or electronic circuits [201].

Parallel DEVS-based simulators also mostly use synthetic models by conducting some experiments. For instance, the work presented in [71] used a number of benchmarking models to analyze the total execution time and speed up. In [72] the performance of the DEVS/CORBA environment was evaluated by testing a number of Supply Chain Models. The simulator was also tested by conducting a benchmark simulation for a large-scale logistics system [73]. A distributed DEVS simulator was proposed in [75] which measured the performances of the peer-to-peer network systems. The DEVS/RMI system used a test case of a large-scale dynamic 2D-Cell space model to analyze the performance of the simulator in terms of dynamic re-configuration capabilities [76]. Zacharewicz *et al.* [107] used an example of a microelectronic production workflow to test the performance of the G-DEVS/HLA environment.

From the above discussion it is clear that most of the PDES research efforts analyze their simulator by investigating only a few real or synthetic models. Moreover, performance evaluation studies for both conservative and optimistic time synchronization have been limited to an evaluation of parameters specific to the simulation model. In contrast, the performance analyses presented in this dissertation (Chapter 6) analyze the impact of more general discrete event simulation parameters such as total number of null messages, null message ratio, blocked time, and memory consumption. In addition, the performance evaluation conducted in this research uses not only synthetic models, but also various real-world scenarios. The performance results of this research quantify the impact of various simulation model parameters on the attained speedup. As such, the results can be used to assess to what extent a particular simulation can benefit from parallel execution under a conservative or optimistic time synchronization protocol.

Chapter 3: Contributions

The central themes of this dissertation are to tackle DEVS-based conservative simulation on distributed-memory multiprocessor clusters and to provide a comparative study analyzing the effect of different parallel synchronization strategies on the simulation performance. This section summarizes the key contributions made in pursuit of each of these two research objectives.

As the first contribution of this research, three conservative DEVS protocols were proposed, based on the classical Chandy-Misra-Bryant synchronization mechanism with deadlock avoidance, extending DEVS abstract simulator to provide means for lookahead computation and null message distribution. These protocols have been integrated into the CD++ simulation toolkit, providing a purely conservative simulator, called CCD++, for running large-scale DEVS and Cell-DEVS models. The three algorithms were implemented on a revised DEVS abstract simulator to reduce the frequency of lookahead computation. They also replace time information estimations with a single lookahead computation, reducing the number of null messages. The dynamic lookahead values of the proposed algorithms are extracted directly from the model specification, obviating the need for the modeler in providing predefined values. In addition, the low-cost lookahead computation feature of the algorithms provides a fast and efficient method and reduces the underlying protocol's overhead.

The first protocol is referred to as the **Lower Bound Time Stamp** mechanism (LBTS), the way used to compute the next global virtual time. Under LBTS, processes communicate only through messages with their neighbors; there are no shared variables and no central process for message routing or process scheduling. Although each LP has its own Local Virtual Time (LVT), no event is received at the virtual past time. The null messages carry lookahead information. The protocol is deadlock-free, as null message cycles cannot occur. At the start of every synchronization phase, each LP computes its lookahead value, which is extracted dynamically from the model specifications, and forwards it to all other LPs. Then, the LP suspends and waits for all remote null messages to arrive from other LPs.

The main issue of the LBTS protocol is the large number of null messages that are distributed throughout the simulation. Each LP not only sends null messages to its direct neighbors, but also to every other LP to ensure correct computation of the LBTS value. This limitation degrades the

performance when the processors are fully connected, overloading the simulation with excessive synchronization messages. In order to reduce the overhead of the LBTS conservative DEVS algorithm, the **Global Lookahead Management** (GLM) protocol was proposed. The GLM mechanism significantly reduces the number of null messages by organizing the conservative execution in such a way that every LP reports its lookahead only to the global manager rather than to every neighboring LP. The proposed protocol implemented an asynchronous strategy in the sense that there is no global clock (every process maintains its own local clock). A central lookahead manager is in charge of receiving every LP's lookahead, identifying the global minimum lookahead of the system, and broadcasting it via null messages to all LPs. The sole function of the central manager is to detect the suspension phase, and to initiate the resume phase by broadcasting the global minimum lookahead. The simulation is divided into cycles of two phases: *parallel* and *broadcast*.

The third protocol, referred to as **Chandy-Misra-Bryant** (CMB) **DEVS**, is a variation of the LBTS mechanism aiming to reduce the number of null messages by introducing multiple rounds of null messages. The protocol changes the way conservative synchronization is maintained by focusing on null message distribution among the neighboring LPs. An LP only forwards null messages to its direct neighbors as defined by the DEVS translation function. Under this scheme, at the start of every synchronization phase, the LP computes its lookahead similarly to the way it is calculated in LBTS, but the null message is only sent to its direct neighbors. Each null message distribution and reception continues in multiple rounds until there is a guarantee that no smaller lookahead value can be received from neighboring LPs later in time. Once the LP has received the smallest possible lookahead value, it computes the new LVT and resumes the simulation. With the CMB protocol, the overall number of null messages is reduced, but the multiple-round of lookahead computation and null message distribution could have a negative effect on the overall performance of the simulation.

The conservative DEVS algorithms presented in this dissertation overcome the limitations of the original conservative Parallel DEVS [11] by: (i) implementing the mechanism at the top most level of the DEVS abstract simulator hierarchy (i.e. the *Coordinator*), thus reducing the frequency of information computation, and (ii) performing a single lookahead computation rather than two types of calculations (i.e. EIT and EOT), which results in a significant reduction of number of null messages.

The key focus of these approaches is on how to compute lookahead values and distribute them via null messages, and when to suspend/resume a processor.

The second contribution of this research is a comparative study of optimistic versus conservative DEVS-based simulation by conducting experiments with variety of Cell-DEVS models. To achieve this goal, precise sensitivity analyses at both model- and underlying synchronization protocol-level were carried out on both of the conservative simulator (CCD++) and the optimistic one (PCD++), studying the performance in terms of execution time, memory usage, operational cost, and system stability for very large models.

3.1 Research Publications

Some of the results derived from this research have been already published thus far, including those directly related to the two central research themes and those relevant to DEVS-based M&S in general. Following is a list of manuscripts that have been published or submitted awaiting acceptance.

- Jafer, S., Wainer, G. "Conservative Synchronization Methods for Parallel DEVS and Cell-DEVS". Proceedings of Summersim'11, Netherlands. 2011 [83]. This paper evaluates the performance of the three conservative protocols by presenting the results of running three Cell-DEVS models on a cluster of 12 nodes. The metrics used to evaluate these protocols are based on the synchronization protocol and the nature of the model.
- Jafer, S., Wainer, G. "A Performance Evaluation of the Conservative DEVS Protocol in Parallel Simulation of DEVS-based Models ". Proceedings of Springsim'11, 2011 [84]. This paper provides the performance gained using the LBTS protocol in running large-scale Cell-DEVS environmental models on a cluster of 26 machines.
- Jafer, S., Wainer, G. "Global Lookahead Management (GLM) Protocol for Conservative DEVS Simulation". Proceedings of DS-RT 2010, Virginia, USA. 2010 [86]. This paper proposes the GLM protocol by outlining the implementation details and the integration of the protocol into CD++. Several Cell-DEVS models are examined to evaluate the performance of the protocol.
- Jafer, S., Wainer, G. "Conservative DEVS - A Novel Protocol for Parallel Conservative Simulation of DEVS and Cell-DEVS Models ", Proceedings of SpringSim'10, Orlando, USA.

2010 [87]. This paper introduces the LBTS protocol and discusses the challenges with regards to computing lookahead and LVT for DEVS-based parallel simulation.

- Jafer, S., Wainer, G. "Conservative vs. Optimistic Parallel Simulation of DEVS and Cell-DEVS: A Comparative Study ", SummerSim'10, Canada. 2010 [88]. This paper provides a comparative study by conducting thorough experiments running large-scale Cell-DEVS model on a conservative CD++ and an optimistic version of the tool. Performance analyses are performed highlighting the strength of each protocol under varied scenarios.
- Wainer, G., Liu, Q., and Jafer, S. "Parallel Simulation of DEVS and Cell-DEVS models in CD++", In Discrete-Event Modeling and Simulation: Theory and Applications, Boca Raton, FL: CRC Press, pp. 226-272, 2010 [89]. This book chapter presents optimistic and hybrid synchronization approaches for running large-scale DEVS and Cell-DEVS models by evaluating the protocols through performance analyses of environmental Cell-DEVS models.
- Jafer, S., Wainer, G. "Flattened Conservative Parallel Simulator for DEVS and CELL-DEVS", Proceedings of International Conferences on Computational Science and Engineering, Vancouver, 2009 [90]. This paper proposes flat architecture for conservative simulation of parallel DEVS and Cell-DEVS using WARPED as a simulation middleware.
- Jafer, S., Wainer, G. "Advanced Parallel/Distributed Simulation Benchmark for Cellular Models". Poster Proceedings of AI/ GI/ CRV/ IS Annual Conference, Windsor, May 2008 [93]. This poster-paper outlines different simulation approaches for running cellular models in parallel and distributed fashion.
- Jafer, S., Wainer, G. "Synchronization Strategies for Parallel Simulation of Large-Scale DEVS-based Models ". Submitted to SIMULATION: Transactions of the Society for Modeling and Simulation International, 2011 [94]. This journal paper summarizes the thesis and discusses the different conservative protocols that are proposed in this research. It also provides a thorough performance analysis of these protocols and compares their performance in terms of a sent of metrics.
- Jafer, S., Liu, Q., and Wainer, G. "Synchronization Methods in Parallel Discrete-Event Simulation ". In-preparation for Journal of SIMULATION: Transactions of the Society for Modeling and Simulation International, 2011 [95]. This journal paper is a literature survey of

existing synchronization methods in parallel discrete-event simulation. It surveys different approaches that have been proposed in the last three decades.

Other contributions published during this research include:

- Moallemi, M., Jafer, S., Seyed, A., and Wainer, G. "Interfacing DEVS and Visualization Models for Emergency Management". Proceedings of Springsim'11, 2011 [85]. This paper proposes a collaborative framework for integrating real-time DEVS simulation with real-time 3D visualization in an emergency planning scenario using robotic agents.
- Sanz, V., Jafer, S., Wainer, G., Nicolescu, G., Urquia, A., and Dormido, S. "Hybrid Modeling of OptoElectrical Interfaces Using DEVS and Modelica". Proceedings of the DEVS Integrative M&S Symposium, Springsim'09. San Diego, CA, USA. 2009 [91]. This paper provides implementations of opto-electrical interfaces, their characteristics and functionalities using a hybrid M&S approach based on CD++ and Modelica.
- Jafer, S., Wainer, G. "Event Behavior of Discrete Event Simulations in CD++ Vs. NS-2". Poster Proceedings of Spring Simulation Multiconference, SpringSim, Ottawa, April 2008 [92]. This paper analyzes Future Event Set data structures of two discrete event simulators: CD++ and NS-2. Varied simulations are conducted on each simulator to describe a real event behavior by observing event timestamps, life times into the FES and event execution time.

Chapter 4: Conservative DEVS Protocols

This chapter proposes three conservative DEVS protocols (**Lower-Bound-Time-Stamp (LBTS) DEVS**, **Global Lookahead Management (GLM)**, and **Chandy-Misra-Bryant (CMB) DEVS**) for efficient conservative simulation of P-DEVS and Cell-DEVS models on distributed-memory multiprocessor clusters. Section 4.1 outlines the research problem and the underlying design rationales. Section 4.2 introduces the LBTS protocol and the lookahead and LVT computation mechanism that is commonly used by the three protocols. . Section 4.3 describes the CMB DEVS protocol. Section 4.4 covers the GLM mechanism, while Section 4.5 compares the three protocols. The classical P-DEVS protocol is investigated in Section 4.6, while the zero-lookahead issue in P-DEVS is explained in Section 4.7.

4.1 Problem Statement and Design Methodologies

This chapter aims to tackle the various challenges of DEVS-based conservative simulation on distributed-memory multiprocessors by exploiting the core computational properties of DEVS. Specifically, the research attempts to address the issues of conservative DEVS-based simulation systematically, improving the simulation performance (in terms of both execution time and memory consumption) without complicating the synchronization mechanism. The first proposed mechanism, referred to as the **Lower-Bound-Time-Stamp (LBTS) Conservative DEVS** protocol, is developed based on the following rationale.

- **Pure conservative synchronization**

The proposed protocols take a purely conservative approach to simulation synchronization, using the Chandy-Misra-Bryant null message strategy with deadlock avoidance. The protocols make use of the underlying simulation parameters to exploit increased degree of parallelism. That is, for lookahead calculation and LVT computation, the protocols use existing parameters that are extracted from the model automatically, reducing the overhead of the underlying synchronization protocol significantly. Doing so makes the system less reliant on the knowledge of model behaviour, resulting in *general-purpose* DEVS-based simulations.

- **Coordinator-centered optimization**

As discussed in Section 2.6, CCD++ employs a flat LP structure that consists of only two coordinators (i.e., one NC and one FC) on each node, while many Simulators are created in a typical large-scale simulation. Hence, by implementing the conservative protocols at the NC level, a substantial reduction in the operational overhead occurs resulting in a significant improvement in the overall simulation performance. The adoption of the NC-centered strategy drops the cost of null message distribution and the lookahead computation significantly.

- **Simultaneous reduction of memory consumption and execution time**

In general, the optimistic synchronization mechanisms make a trade-off between the execution time and the memory usage. In contrast, the conservative protocols like the ones proposed in this research try to achieve both objectives simultaneously by maintaining shorter data lists (there is no need to keep historic data as in optimistic protocols), while, at the same time they attempt to reduce the number of null messages and the cost of the lookahead computation as much as possible. Moreover, the proposed conservative protocols also try to speed up memory reclamation by performing it more frequent, without incurring negative impact on the overall simulation performance.

- **Event-queue management**

The protocols use a simple Least-Time-Stamp-First (LTSF) queuing mechanism to facilitate event queue operations, instead of using advanced data structures and algorithms. Although the simulation performance can be improved further using such data structures, by keeping the LTSF event queues relatively short throughout the simulation considerable speedups can be achieved. This is done by deleting the input queues' events as soon as they are executed (unlike optimistic protocols where historic data are kept in case rollback occurs). In addition, the protocols are specifically tailored for efficient execution of a large number of simultaneous events at each virtual time, directly addressing the computational property of large-scale, densely-interconnected, and highly-active DEVS-based models.

4.2 The Lower-Bound-Time-Stamp Protocol

In this section, the LBTS conservative DEVS protocol is presented, which serves as the base for the other two protocols (i.e., CMB and GLM) that will be presented next. The LBTS protocol (named after **Lower-Bound-Time-Stamp** concept [150], due to the way used to compute the next safe time) is

mainly based on the original Chandy-Misra-Bryant approach with deadlock avoidance. The protocol is implemented at the NC and extends the DEVS abstract simulator to provide means for lookahead computation and null message distribution. Processes communicate only through messaging with their neighbors; there are no shared variables and no central process for message routing or scheduling. Although each LP has its own Local Virtual Time (LVT), no events are received at virtual past time. Moreover, synchronization is maintained through null messages carrying lookahead information. The NC on each LP is the central synchronizer for driving the simulation on that node. The NC is responsible for lookahead calculation, null message distribution, suspending the LP, receiving null messages from other LPs while the LP is blocked, and resuming the LP when all remote null messages are received. That is, the NC drives the simulation at the LP, while other DEVS processors (FC and Simulators) are unaware of the underlying synchronization mechanism.

The LBTS protocol is deadlock-free, since null message cycles cannot occur. At the start of every synchronization phase, each LP computes its lookahead value, which is extracted dynamically from the model specifications, and forwards it to all other LPs. Then, the LP suspends and waits for all remote null messages to arrive from destination LPs. Once the null messages are received from all LPs participating in the simulation, the destination LP resumes and first computes its new LVT based on the lookahead values it has received via the remote null messages. Under this scheme, at any time, the LVT of every LP is equal to the Lower-Bound-Time-Stamp of any unprocessed event among all LPs. The major issue of this protocol is the large number of null messages that must be distributed at the start of every synchronization phase. Each LP not only sends null messages to its direct neighbors, but also to every other LP participating in the simulation to ensure correct computation of the LBTS value.

Figure 21 illustrates the architecture of CCD++ which was built on top of the WARPED kernel. WARPED is used just to provide services for defining processes (simulation objects), scheduling, memory, file, event, communication, and time management (and PCD++ uses the optimistic synchronization services). Simulation objects on a physical processor are grouped into an LP, and communicate through the Message Passing Interface (MPI).

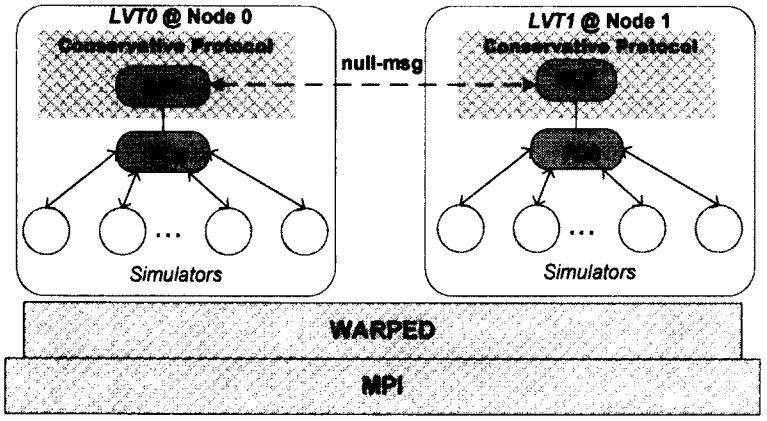


Figure 21. Conservative Architecture of CCD++

The conservative mechanism is invoked at the beginning of every *collect* phase at the NC. The LP suspension also takes place during the *collect* phase. Simulators can only communicate with their parent FC, which means there is no direct communication between Simulators (even the local ones), thus FCs are always aware of the timing of state changes of their child Simulators. When a Simulator sends a (y, t) message to its parent FC, the FC knows if the recipient is a local Simulator or a remote one (residing on another LP). In case that the destination Simulator is local, it simply translates it into an (x, t) message and sends it to the recipient Simulator. However, if the destination is remote, the FC forwards the received message (y, t) to the parent NC. The NC translates it into an (x, t) message and sends it through inter-LP communication to the parent NC of the recipient. Note that outgoing inter-LP communication happens only during the *collect* phases, whereas incoming inter-LP communication can occur at any phase. This implies that since *output* functions of imminent components are invoked only at *collect* phases, at any given simulation time, all *external* messages going to remote NCs are sent out by the end of the *collect* phase. On the other hand, an *external* message from a remote source can arrive at the destination NC at any phase.

The NC is invoked when it receives a *done* message from the FC. The *done* message could be in response to an (I, t), (@, t), or (*, t) message previously sent to the FC. On each node, the NC advances the simulation time. The NC updates the LVT of the LP at the beginning of every *collect* phase. The local FC and the Simulators do not send messages with a timestamp different from the current LVT.

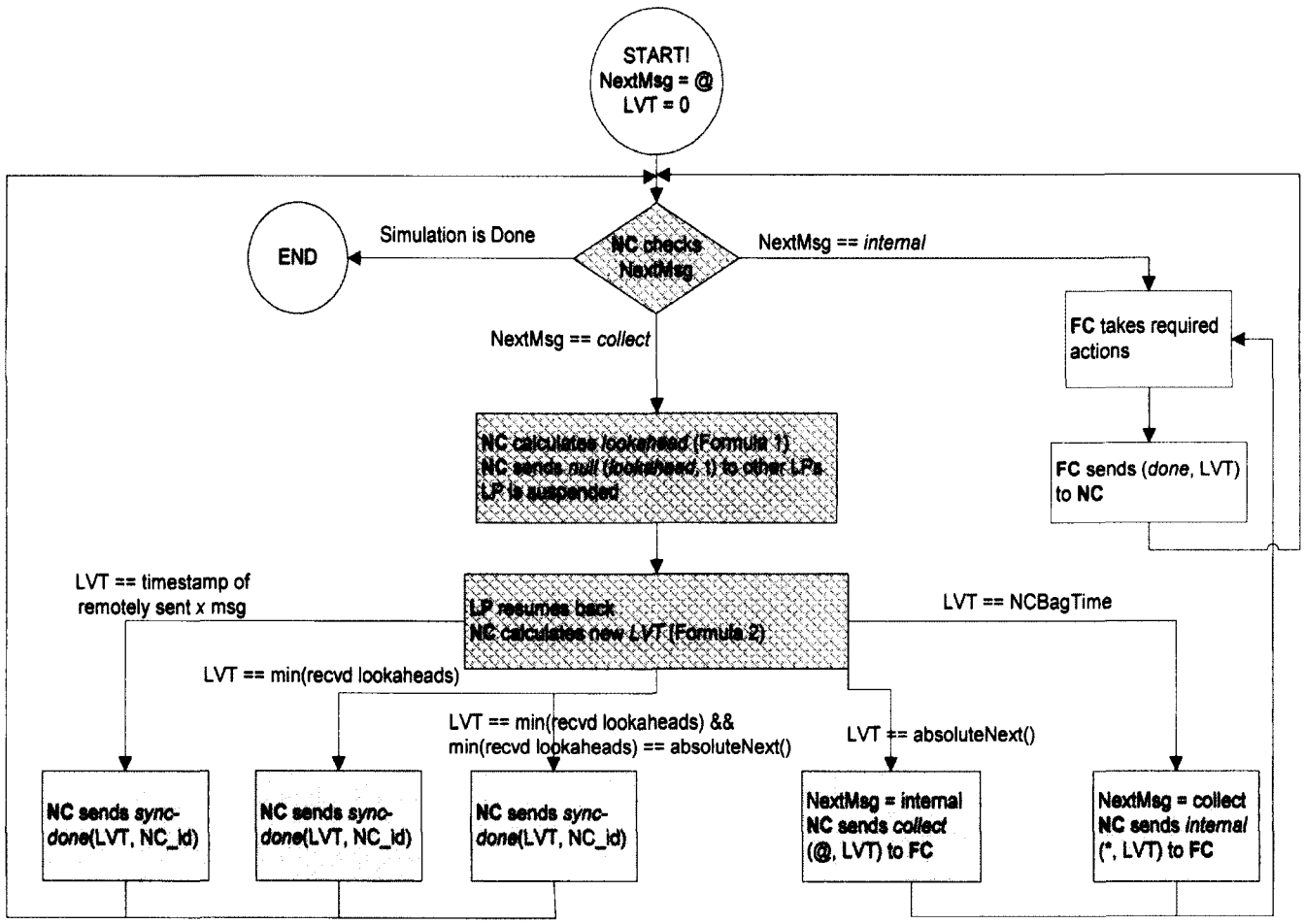


Figure 22. Flow Chart of the Conservative Algorithm on each LP

A) Lookahead and LVT Computations

Figure 22 illustrates a flow chart describing the conservative mechanism. The highlighted boxes are those implementing the algorithm. The first phase after initialization is a *collect* one, where the flow chart begins (*NextMsg* tells what the next phase is. Initially it is set to *collect* @). Note that the lookahead term in all the three conservative protocols (LBTS, CMB, GLM) refers to a quantity that is computed relative to the *current_time* (LVT) value of the LP (i.e., *lookahead* = *lookahead_value* + *current_time*).

When the NC receives a *Done* message form the FC, it checks if the next phase of the simulation is *collect* or *internal*. The conservative algorithm is only invoked if the next phase to take place is a *collect*. If the NC decides to issue an *internal* phase, it first sends an (*, *t*) message to the FC. The FC

will then forward this message to all imminent child Simulators. Internal transitions are triggered at these Simulators followed by *Done* messages emitted to the FC reporting their next state transition time (t_N). The FC sends the closest state transition time (minimum among all t_N values) to the NC through a *Done* message. In processing (*Done*, *t*), the NC issues a *collect* phase and invokes the conservative mechanism. First, it performs lookahead computation as following:

$$lookahead = \text{MIN}(timestamp\ of\ the\ x\ msg\ recently\ sent\ to\ a\ remote\ LP,\ time\ of\ the\ \text{NC}\ Message\ Bag,\ t_N) \quad (1)$$

where t_N is the closest state transition time given by the FC in the done message, and time of the NC *Message Bag* is the minimum timestamp of all those unprocessed *x* messages received from other NCs. Considering the NC *Message Bag* time ensures that the calculated lookahead takes into account possible *x* messages that arrived when the LP was suspended. This ensures that when other LPs receive such lookahead, their LVT would not advance beyond it (in case the sender LP sends *x* messages to them as a result of processing those messages in its NC *Message Bag*).

The NC then propagates the lookahead value to other NCs via null messages in the form of *null*(*lookahead*, *LVT*) and suspends. During the suspension, the LP is still able to receive messages; however, these are only inter-LP events which are either remote *x* messages or null events. When the NC receives all null messages it resumes and first calculates the new LVT as following:

$$LVT = \text{MIN}(timestamp\ of\ x\ msg\ recently\ sent\ to\ a\ remote\ LP,\ time\ of\ the\ \text{NC}MessageBag,\ minimum\ RemoteLookahead,\ t_N). \quad (2)$$

Thus, the NC computes the new LVT as the minimum value among: (i) the timestamp of the *x* messages recently sent to remote LPs; (ii) the time of the NC *Message Bag*; (iii) the minimum value among the received lookaheads from remote LPs; and (iv) *absoluteNext* which is the closest state transition time of child Simulators previously given by the FC. Once the LVT is computed, one of the following four scenarios happens:

1. ***LVT = timestamp of the x msg recently sent***

If there were x messages sent to remote LPs during the recent collect phase (right before the LP was suspended), the NC must send its current lookahead and block the LP again, because the remote lookahead that was just received from the receiver of the x message, was calculated before reception of

the x message. Thus, the LP must block and wait for new remote lookahead value in case the receiver of the x message generates a smaller lookahead than the one sent before, or sends x messages as a result of processing the received x messages. Not considering this scenario would cause causality violations at this LP because it has updated its LVT based on a larger lookahead and when it receives the new lookahead which happens to be smaller than before, then, the new LVT turns out to be smaller than the old LVT and this is strictly forbidden by the definition of conservative synchronization mechanism.

2. ***LVT = time of the NC Message Bag***

If there are external messages received from other LPs with receive time equal to the new LVT, the NC issues an internal phase and sends a (*, t) message to the FC and sets NextMsg to collect. In processing this internal message, the FC forwards the x messages to the local recipient Simulators.

3. ***LVT = minimum remote lookahead***

If the new LVT is equal to the minimum of all lookahead values received from other LPs, there is nothing to do and the LP must wait. Therefore, the NC recalculates the lookahead, sends null messages, and the LP is suspended.

4. ***LVT = minimum transition time of local Simulators***

If there are imminent children (Simulators) a *collect* phase is issued by sending a (@, *t*) message to the FC and the *NextMsg* is set to *internal.*

The LVT calculation scenarios explained above have processing priority and the NC only processes one of them every time. Special tie-breaking is performed when more than one case is true. An example of a tie situation is when case (iii) and (iv) are both true. This implies that there is a possibility of receiving an *x* messages from remote LPs exactly at the time of the closest transition time, thus it is safer to first wait for other LPs and then if there was no *x* message received from other LPs, continuing by issuing a *collect* phase and processing imminent child Simulators. This is all done by first sending current lookahead and suspending the LP. When the LP is suspended it will receive the remote *x* messages (if any) as well as new remote lookaheads, thus when it resumes, the tie is no longer true and if the newly calculated LVT is still same as case (iii), the NC would continue accordingly.

Figure 23 illustrates the pseudocode description of the NC functionalities when a (*done, t*) message is received from the FC. The first *Done* message received by the NC is the response to the initialization message previously forwarded to the FC to start the simulation on that machine. Since *next-message-*

type is initialized to @, the NC follows the second half of the algorithm (line 6 to 48). Based on our conservative algorithm, the NC starts the mechanism by first computing its lookahead according to Formula (1) (line 7). Each NC maintains a list of remote NCs residing on those LPs that will exchange messages with this LP. This list is initialized at the beginning of the simulation and is used by the NC when it distributes the lookahead value through null messages (line 7 to 9). On every LP, the NC acts as the local controller of the simulation and carries on the event execution loop if and only if the current time of the LP (i.e. LVT) is less than the simulation *Stop Time* which is set at the beginning of the simulation by the user's model. To ensure this, the NC performs a check (line 11 to 14) and only if the condition is satisfied then the LP is suspended, otherwise the simulation is complete at this LP and it will remain idle until the rest of LPs are finished. The simulation terminates when all the LPs are idle. When the LP is suspended, it waits for the lookahead of every LP in its *RemoteNCList*. When all the lookahead values are received, the NC resumes back to the point it got suspended at (line 14). For a start, the NC calculates the *min-time* using formula (2) of the conservative algorithm (line 15 to 16).

The resulting *min-time* is the next local simulation time (i.e. LVT of this LP) to which the NC should advance. After *min-time* calculation, the *LookaheadInfoArray* of the NC is erased so that it can be filled with new lookahead information round (line 17). There is a special situation that might occur on an LP (line 18 to 20) which is when the calculated *min-time* happens to be *Infinity*. This case arises when the LP is done with the simulation and there is no event to be received from other LPs because they have all sent an infinite lookahead value, therefore, the LP is done with the simulation. To finish the simulation at the LP, the NC sets the *min-time* back to the previous value which was the timestamp of the received *done* message and sends a *done* message to itself with *D.ta* equal to zero, where *D.ta* is the next transition time that is reported to the NC. When this *done* message is received at the NC (line 7) the LP will be marked as *idle* and no further event execution will take place at the LP (line 9). However, if the condition of line 18 is not met, the four cases that were mentioned in Section 4.2.1 are checked next (line 29 to 46). The tie-breaking mechanism is invoked as follows:

1. Case 1 (line 29 to 31) is given the highest priority so that if there was any tie between this case and other cases, it gets executed first.

2. If it happened that the minimum remote lookahead value is equal to the timestamp of the closest transition at this LP (i.e. t_N) then the priority is given to processing the *minimumRemoteLookahead* (line 32 to 34).

The *Done* messages sent from the NC to itself (line 30, 33, and 45) are for synchronization purposes only. They are easily differentiable from the true *done* messages (i.e. the ones sent form the child FC) by just looking at the sender and receiver *ID* of the message where in this case both are the NC itself. The $D.ta$ value carried by these messages is a special form which is calculated as the difference between the NC's current t_N and the LVT advancement:

$$D.ta = t_N - (\textit{min-time} - \textit{timestamp of the received D msg}),$$

where *minTime* is the new LVT, and *timestamp of the received D msg* is the previous LVT. When the NC receives this special *done* message it calculates its new lookahead and sends it across and suspends itself.

```
1. when a (D, t) is received from the child FC
2.        t_L = t; t_N = t_L + D.ta
3.        if next-message-type = * then
4.                send (*, t) to the child FC
5.                next-message-type = @
6.        else
7.                lookahead = MIN( timestamp of the x msg recently sent to a remote LP, time of the NC Message Bag, t_N )
8.                for each NC in the RemoteNCList do
9.                        sendNullMsg(lookahead);
10.               end for each
11.               if t_N != Inf and minimumRemoteLookahead != Inf and NCMessageBag != Empty then
12.                       suspend this LP
13.               else      this LP is DONE
14.               end if
15.               min-time = MIN( timestamp of the event pointed by event-pointer, timestamp of the x msg recently sent to a
16.                                       remote LP,time of the NC Message Bag, minimum RemoteLookahead, t_N )
17.               resetLookaheadInfoArray()
18.               if min-time = Inf then
19.                       min-time = the timestamp of the received D message
20.                       send (D, t) to this NC with D.ta = zero
21.               else
22.
23.                       if min-time = the timestamp of the event pointed by event-pointer then
24.                               for each x in the Event List with min-time do
25.                                       send (x, t) to the child FC
26.                                       move event-pointer to the next event
27.                               end for each
28.                       end if
29.                       else if  an x msg was recently sent to a remote LP then
30.                               send (D, t) to this NC with D.ta = t_N – (min-time – timestamp of the received D msg )
31.                       end if
32.                       else if min-time = minimumRemoteLookahead and minimumRemoteLookahead =  t_N then
33.                               send (D, t) to this NC with D.ta = t_N – (min-time – timestamp of the received D msg )
34.                       end if
35.                       else if min-time = t_N then
36.                               send (@, t) to the child FC
37.                               next-message-type = *
38.                       end if
39.                       else if min-time = the time of the NC Message Bag then
40.                               for each x in the NC Message Bag with min-time do
41.                                       send (x, t) to the child FC
42.                               end for each
43.                       end if
44.                       else if min-time = minimumRemoteLookahead then
45.                               send (D, t) to this NC with D.ta = t_N – (min-time – timestamp of the received D msg )
46.                       end if
47.               end if
48.       end if
49. end when
```

Figure 23. Conservative NC Algorithm for *Done* Message

B) Scheduling

The scheduling algorithm of LBTS protocol is given in Figure 24. Each node maintains an input queue, namely *inputQ* which is a simple linked list that is provided by the WARPED kernel. Queue manipulation functionalities are also provided by the kernel. Since in conservative simulation causality violations are not allowed, the *inputQ* follows the First-In-First-Out (FIFO) mechanism. On every node, all the DEVS messages as well as the null messages are treated as basic events and are inserted into this queue.

When the scheduler is invoked, it simply returns the head element of the *inputQ* and the event is deleted after execution to reclaim its memory location. Our modifications to the scheduling mechanism are for the purpose of LP suspension. When the NC decides that the LP should be suspended, it sends a special type of *Done message* to itself. When the event returned by the scheduler happens to be this *Done* message, the event is not executed until all remote null messages are received and inserted into the LP's *inputQ*. As illustrated on the algorithm, the *currentPos* variable represents the first unprocessed element of the *inputQ*. When a suspension event is detected (line 11) it is not returned until all required null messages are received. The number of null messages that must be received by an LP in order for it to resume back is equal to the total number of LPs minus one (line 17); since the LP does not need a null message from itself.

```
1. when the scheduler is invoked to return the first unprocessed element
2.         if currentPos != NULL then
3.                 if currentPos is a null-message then
4.                         currentPos = currentPos->next
5.                         return NULL
6.                 end if
7.                 else if currentPos is a remote x message then
8.                         currentPos = currentPos->next
9.                         return NULL
10.                end if
11.                else if currentPos is a suspension message then
12.                    for each unprocessed element of inputQ do
13.                                 if event is a null-msg then
14.                                     event->checked = true
15.                                     recvdNullMsg++
16.                                 end if
17.                        end for
18.                        if recvdNullMsg = totalLPs – 1 then
19.                                 return currentPos
20.                        else     return NULL
21.                        end if
22.                end if
23.                else     return currentPos
24.        end if
25. end when
```

Figure 24. Scheduler Mechanism of LBTS Protocol

C) Resuming a Blocked LP

The algorithm used by the LBTS protocol to resume a blocked LP is presented by Figure 25. When the event returned by the scheduler is the suspension event (the special *Done message* sent from a NC to itself), before it can be executed, two actions must be performed. Firs, all those null messages that were counted to increment *recvdNullMsg* must be first executed (line 5 to 7). Second, all unprocessed remote external messages that were sent from other NCs to this LP must be executed (line 8 to 10). Once these null messages and remote x messages are executed, then the special *Done* message is returned to the NC.

```
1. when executeProcess() is invoked
2.         event = Scheduler.getEvent()
3.         if event != NULL then
4.                 if event is suspension event then
5.                         for each null-msg with checked = true
6.                                 receive (null-msg)
7.                         end for
8.                         for each unprocessed remote x msg
9.                                 receive (x)
10.                        end for
11.                    receive (suspension)
12.                end if
13.        end if
14. end when
```

Figure 25. Suspension-Event Execution Algorithm

D) Null Message Handling

Figure 26 shows the null message handling mechanism which is invoked when a null message is received at the NC. Every NC maintains a queue, namely *lookaheadInfoArray* to store the received lookahead value carried by the null message. The size of this queue is equal to the total number of LPs minus one. This is from the assumption that in a simulation consisting of n LPs, every LP communicates directly with $n - 1$ LPs. When a null message is received, the NC saves the lookahead content into its *lookaheadInfoArray* (line 3 to 5) and calculates the *minimumRemoteLookahead* as the smallest element of this queue (line 8). The *minimumRemoteLookahead* is updated every time the NC receives a new null message. Hence, it is always the minimum lookahead value of all remote LPs.

E) Deadlock Avoidance

Since null message distribution occurs before LP suspension, deadlock is strictly avoided. The NC only suspends the LP after performing a lookahead computation and propagating it to all remote LPs via null messages. Thus, when an LP is suspended, it has already forwarded its null messages, and if every other LP is suspended as well, they would all resume because all the required null messages have been already distributed among them before the LP suspension has taken place. Aside, a simple strategy is used to resolve the zero-lookahead issue. When an LP receives a remote null message carrying a lookahead of zero, if the destination LP has an unprocessed event with current time stamp, then, the LP suspends for an additional phase, giving the sender LP the priority to execute its scheduled events first. This ensures

that the LP takes into consideration future events that might arrive from the sender LP as a result of the current execution.

```
1. when a null-message is received
2.         for i = 1 to arraySize do
3.                 if lookaheadInfoArray[i] = NULL then
4.                     lookaheadInfoArray[i] = recvdLookahead
5.                     break
6.                 end if
7.         end for
8.         latestRemoteLookahead = MIN (lookaheadInfoArray)
9. end when
```

Figure 26. NC Null Message Handling Algorithm

F) Simulation Termination

On each LP, the NC decides to terminate the simulation according to the algorithm presented in Figure 27. The criterion under which an LP terminates and becomes *idle* was discussed at the end of Section 4.2.A The WARPED kernel checks the status of the LPs on every period that can be determined by the user. When all participating LPs are reported as *idle*, then the simulation terminates and the rest of memory clean-ups are taken care of by the kernel.

```
1. when the kernel checkIdle() is invoked
2.  allIdle = true
3.  for each participating LP do
4.    if LP.inputQ has unprocessed event and is not a null-message then
5.                 allIdle = false;
6.    end if
7.  end for
8.         return allIdle
9. end when
```

Figure 27. Simulation Termination Algorithm

According to this algorithm, an LP is *idle* if it has no unprocessed event (excluding null messages) in its *inputQ*. The restriction that the event should not be a null message is for the case when the LP has finished its simulation and is idle waiting for other LPs to finish. While an LP is idle, it can still receive null messages. However, these events will not be executed. Such null message indicates the very last event that the sender LP distributes prior to entering the *idle* state. This type of null message carries a lookahead value of *Infinity*, stating that the originating LP has completed its simulation tasks.

G) Simulation Scenario in CCD++ with LBTS Protocol

In this section, a simulation scenario is presented which is based on the conservative LBTS mechanism of CCD++, its flat architecture and the messaging mechanism was introduced in Section 2.6. As illustrated in Figure 28, the simulation is carried on with two participating nodes. The simulation starts with message (I_1) at the NC at time 0. This initialization phase ends when the two local Simulators (S1 and S2) send back *Done* messages (D_5, D_6) to the FC, which causes forwarding message (D_7) to the NC. Every time the NC receives a *Done* message from its FC, it starts the next phase immediately. However, a lookahead computation and null message distribution is performed at every collect phase. After computing the lookahead, the NC sends a message ($null_1$) to the remote NC and blocks (shaded area). The NC remains blocked until all remote lookaheads (carried by remote null messages) are received at the LP. During suspension the LP can still receive messages; however these messages are only inter-LP events which are either remote x messages or null messages. When the NC receives all null messages ($null_2$) it resumes and calculates the new LVT, which is equal to the state transition time that was reported by the FC via *Done* message (D_7). At this time, all Simulators are imminent. Thus, the NC starts the first *collect* phase by sending a *collect* message ($@_8$) to the FC where it further distributes this message as two collect messages ($@_9$, $@_{10}$) to each of the Simulators.

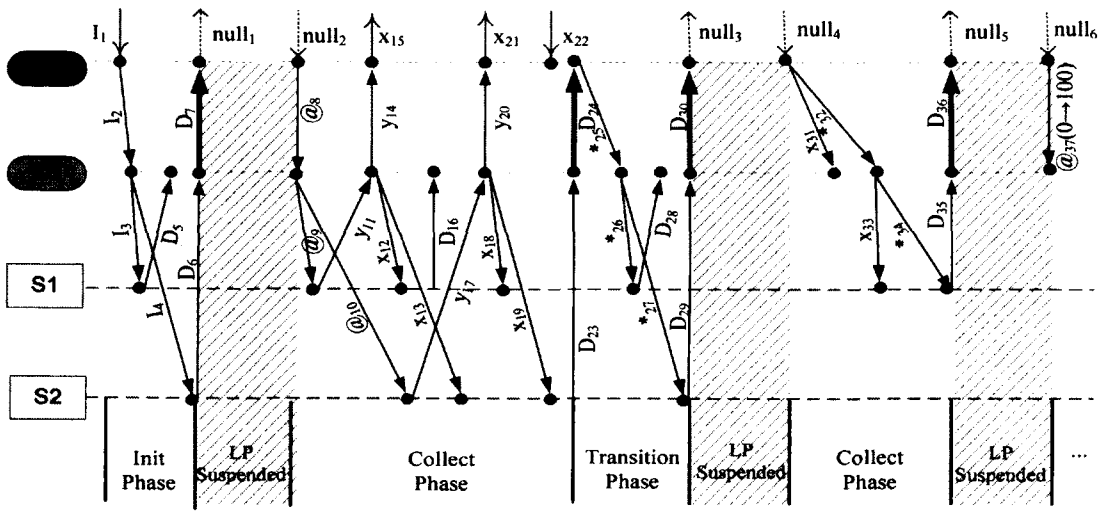


Figure 28. Sample Simulation Scenario in CCD++

Upon receiving the *collect* message, imminent Simulators execute their output functions and send *output* messages to their parent FC. S1 processes $@_9$ first and sends an output message (y_{11}) to the FC. If

it must be sent to S1, S2 (as well as to remote Simulators), FC translates it into an *external* message and sends one copy to each local Simulator (x_{12}, x_{13}). For the remote Simulators, the FC then forwards the output message (y_{14}) to the NC, which translates the message into an *external* one (x_{15}) and sends it remotely to all destination NCs. Similar actions are performed when the FC processes the output message (y_{17}) from S2 (x_{18}, x_{19}, y_{20}, x_{21}). During these steps, the remote external message x_{22} is received at the NC, which inserts it into the NC's Message Bag. When the FC receives the corresponding *Done* messages (D_{16}, D_{23}) from S1 and S2, it sends a *Done* message (D_{24}) to the NC, reporting the end of output operations at the local Simulators. This *Done* message triggers the next phase at the NC, thus the first *transition* phase starts immediately by sending an *internal* message ($*_{25}$) to the FC. This message is then forwarded to imminent Simulators S1 and S2 ($*_{26}$, $*_{27}$). Internal transitions are triggered at these Simulators followed by *Done* messages emitted to the FC (D_{28}, D_{29}). The FC then sends the closest state transition time to the NC through a *done* message (D_{30}). When processing D_{30}, the NC performs lookahead computation, sends the new value through $null_3$ to all remote NCs, and blocks. When the only remote null message ($null_4$) is received at the NC, the suspensions ends, and the NC calculates the new LVT. This LVT turns out to be equal to the timestamp of the external message x_{22} recently received from a remote NC and added to the NC Message Bag. Therefore, the NC sends it to the FC, followed by another internal message (x_{31}, $*_{32}$). When FC executes $*_{32}$, it flushes x_{31} to S1 followed by $*_{34}$. External message x_{33} is added into S1's message bag, accepting the value previously transmitted by x_{22} from a remote sender. After that, the internal message $*_{34}$ invokes S1's external transition, which consumes the value wrapped in x_{33}. The resulting *Done* message (D_{35}) is sent to the FC. When NC executes D_{36}, another lookahead computation takes place, *null5* is sent out, and the LP is blocked. After receiving $null_6$, NC calculates the new LVT. In this case there is no message in its *NC Message Bag*, and the remote lookahead reported by $null_6$ is larger than the closest state transition (time=100), therefore, the NC advances the local simulation time from 0 to 100 and sends to the FC a collect message ($@_{37}$) that has a send time of 0 and a receive time of 100, thereby starting a new cycle of simulation similar to that initiated by $@_8$.

4.3 The CMB DEVS Protocol

The second conservative DEVS protocol is the CMB protocol, which is a variation of the LBTS protocol aiming at reducing the number of null messages. The CMB protocol changes the way conservative synchronization is maintained by distributing the null messages only among the neighboring LPs, in contrast to broadcasting to all participant LPs as in LBTS protocol. An LP forwards null messages to only its direct neighbors as defined by the DEVS translation function. Under this scheme, at the start of every synchronization phase, the LP computes its lookahead as in LBTS (using *Formula (1)* of Section 4.2), but the null message is only sent to its neighbors. Then the LP blocks and it waits for its neighboring LPs to send their lookahead value via null messages. Once all neighbor null messages are received, the LP computes its new LVT based on the received lookahead values as in *Formula (2)* of Section 4.2, and it starts another lookahead computation and null message distribution round. This process continues until it is guaranteed that no smaller lookahead value will arrive from neighbor LPs later in time. Once the LP has received the smallest possible lookahead value, it computes the new LVT and resumes the simulation. Other aspects of the CMB protocol are very similar to those of the LBTS protocol. The only difference is the number of neighboring LPs, where with LBTS protocol each LP has $n - 1$ neighbors, while in CMB protocol each LP has either 1 or 2 neighbors (depending on the partitioning of the DEVS model).

With the CMB protocol, the overall number of null messages is reduced, but the multiple lookahead computation and null message redistribution rounds could have a negative effect on the overall simulation performance. These effects will be analyzed thoroughly in Chapter 6.

4.4 The Global Lookahead Management Protocol

In order to reduce the overhead of the LBTS protocol, the Global Lookahead Management (GLM) protocol is proposed. GLM is based on the Conservative Time Window algorithm [55] and the Distributed Snapshot mechanism [53]. The GLM mechanism dramatically reduces the number of null messages by organizing the conservative execution in such a way that every LP reports its lookahead only to the global manager rather than to every neighboring LP. The GLM protocol implements an asynchronous strategy [46] in the sense that there is no global clock (every process maintains its own local clock), and the GVT approximation is performed based on the LPs' lookahead information.

A) Phase-Based Simulation with GLM Protocol

The GLM protocol borrows the idea of safe processing intervals from the Conservative Time Window algorithm, and it maintains global synchronization in a similar fashion as the Distributed Snapshot technique. Under the GLM scheme, a central *lookahead manager* (LM) exists on LP0, which is in charge of three main tasks: 1) receiving every LP's lookahead, 2) identifying the *global minimum lookahead* of the system, and 3) broadcasting it via null messages to all LPs. This implies that the LPs are no longer required to send their lookahead information directly to each other as in the LBTS protocol; rather, they now send their lookahead via null messages to the LM only. In fact, the sole function of the LM is to detect suspension phase and initiate the resume phase by broadcasting the *global minimum lookahead.* The entire algorithm works using the following sequence of computations:

(i)**Parallel phase**: LPs run simulation until suspension.

(ii)**Broadcast phase**: LM broadcasts global minimum lookahead allowing LPs to advance their LVTs.

The key characteristic of the GLM protocol is that it is asynchronous and the central LM is not expected to be a bottleneck since the only message transmissions involving it take place at the end of *Parallel* and *Broadcast* phase. In fact, the LM does not carry out any computation.

In the LBTS algorithm, each LP had to send a null message to every LP and block until all LPs send back their null messages accordingly, resulting in total null messages of $n\ (n-1)$ per synchronization cycle (where n is the number of LPs). With the GLM protocol each LP sends one null message to the LM and blocks until the LM sends back a null message carrying the global minimum lookahead value. Thus, leading to a total number of $2n$ null messages for every synchronization cycle. The GLM not only attempts to reduce the total number of null messages, but theoretically it could also reduce the blocked time of LPs since they now wait for only a single null message (as opposed to $n-1$ messages) before they can resume. As the number of participating LPs increases, the performance achieved with GLM may increase, merely because the total number of LPs has a direct impact on the synchronization overhead. However, communication overhead could play a negative effect on the overall performance of the protocol. These criteria are investigated and explained in details in Chapter 6 by conducting various experiments comparing the GLM versus the LBTS and CMB protocols.

B) Lookahead and LVT Computation Strategy

Both the lookahead and the LVT of the GLM protocol are computed, in a similar fashion to the LBTS protocol, based on Formula (1) and (2) of Section 4.2. At the start of every synchronization phase, the NC performs the lookahead computation, and it sends the calculated value to the LM via a single null message, then, it suspends the LP. Upon receiving the response null message from the LM, the LP resumes by first calculating the new LVT as the minimum value among four quantities: (i) the timestamp of the *external* message recently sent to a remote LP; (ii) the time of the *NC Message Bag* which is the minimum timestamp among unprocessed input events; (iii) the closest state transition time of the local child processors previously given by the FC in the *Done* message; and (iv) the new *global lookahead* value received from LM via a null message.

Similar to the LBTS protocol, the NC is the local synchronizer at the LP and invokes the GLM protocol at the beginning of every *collect* phase. The NC issues the collect phase by first performing lookahead computation according to Formula (1), then, it sends the calculated value via a single null message to the LM and blocks the LP. Upon receiving the response null message form the LM, the LP resumes and the NC performs a LVT computation which will take into account the newly received *minimum global lookahead* from the LM.

C) Null Message Distribution

Based on the strategy used to partition a DEVS/Cell-DEVS model, each LP can only send/receive event messages (*external messages*) to/from those defined by neighboring DEVS models. That is, according to the DEVS-neighborhood specified at the model, if an atomic component (represented by a *Simulator* processor) on an LP is a neighbor of another atomic component residing on a different LP, then these two LPs are neighbors and they can communicate to each other through inter-LP communication. Therefore, it is possible that some LPs are not direct neighbors of each other, because the model's partitions they hold are not DEVS-neighbors of one another. The GLM neighboring strategy is based on this mechanism, where an LP is only connected to another LP if it happens to be its direct neighbor as specified by the DEVS model neighboring.

D) LP Block and Resume Mechanism

The GLM block and resume mechanism is slightly different from the LBTS algorithm in the sense that LPs are no longer distributing null messages to each other, neither they wait for reception of null messages from every participant LP. In return, each LP sends a single null message at the start of every collect phase to the LM only, and it stays blocked until the single null message reporting the new globally minimum lookahead is received from the LM.

E) Deadlock Avoidance

Since the null message distribution of LPs to the LM occurs before LPs are suspended, deadlock is strictly avoided. NC only suspends the LP after performing a lookahead computation, and reporting it to the LM via a null message. Thus, when an LP is suspended, it has already forwarded its null message, and if every other LP gets suspended as well, they would all resume because all the required null messages have been already sent to the LM before suspension has taken place. The strategy for handling zero-lookahead is the same as the one discussed for the LBTS protocol in Section 4.2.5.

F) I/O Operation

The only messages an LP sends out are the *external* and null messages. The *external* messages are sent out to neighbor LPs defined by DEVS neighboring, while null messages are sent out to the LM on *node0*. Similarly, an LP can only receive *external* messages from its DEVS neighbor LPs, and null messages from the LM.

G) Termination

The simulation terminates if *Stop Time* is reached or all LPs are idle and have no unprocessed event in their input queues. The NC sets the LP to *idle* when: 1) the LM sends a null message reporting *infinity* as the global lookahead value, 2) all local child Simulators have next transition time of *infinity*, and 3) there is no unprocessed event in the *NC Message Bag*.

4.5 Comparison of the Protocols

The three conservative protocols differ in the way null messages are distributed and thus the total number of null messages that must be propagated throughout the simulation. Figure 29 illustrates the null message distribution strategy of LBTS, CMB, and GLM protocols. Figure 29-A illustrates the partitioning of an 8 by 8 Cell-DEVS model on four LPs. The partitioning mechanism divides the cell space into four equal portions (8x2 cells per LP). Figure 29-B represents the LBTS neighboring of LPs where each LP sends and receives null messages to and from every other LP. With CMB mechanism (Figure 29-C), every LP only forwards null messages to its direct neighbors as defined by the DEVS translation function. On the other hand, the GLM protocol resolves this tight coupling of LPs by assigning a simple LP connectivity strategy where each LP is only coupled with the LP for which the LM resides on (i.e. LP0). Hence, the null message distribution is configured as in Figure 29-D.

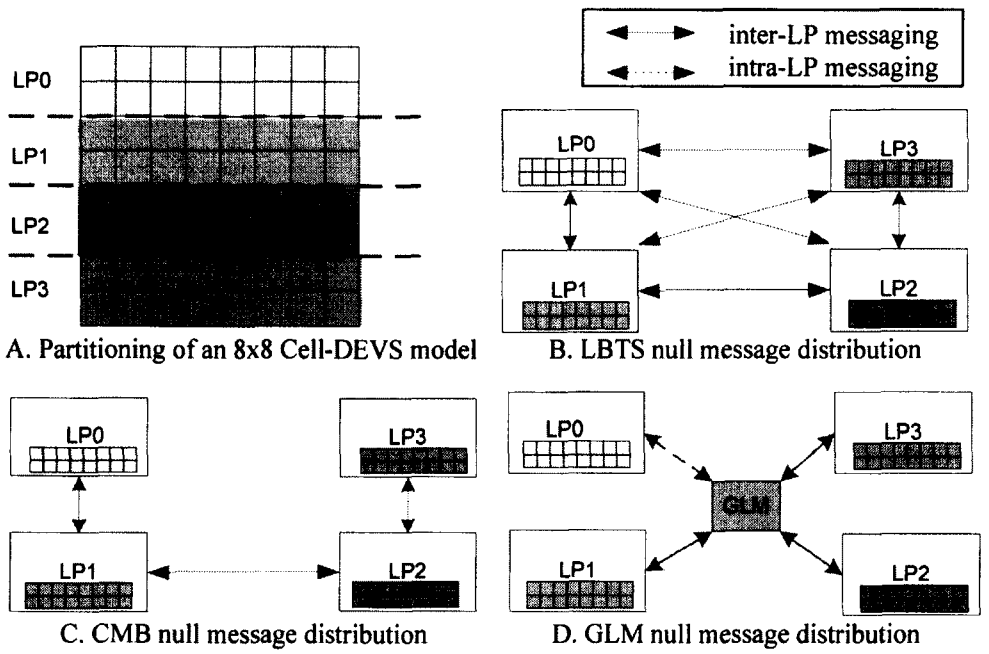


Figure 29. Null Message Distribution Strategy in LBTS, CMB, and GLM Protocol

Aside from the null message distribution mechanism (which is different in LBTS, CMB, and GLM protocol), the lookahead and LVT computations are performed dynamically based on the model's data, and the computation formulas are the same for the three protocols (Formula 1 and 2). Moreover, the three conservative protocols share the following common characteristics:

- ***NC-Driven Synchronization***: the protocols are implemented at the NC; the other DEVS processors are unaware of the underlying synchronization mechanism. The NC is the local controller and drives the simulation on that node. It is responsible for lookahead and LVT computation, LP suspension and resumption, and null message distribution and reception.

- ***Dynamic Lookahead***: Lookahead computation is performed after each LVT computation; hence, it is updated and distributed to the destination LP(s) prior to the LP suspension phase. This strategy ensures that the lookahead value of an LP represents the latest LVT update, as there is at least one lookahead computation per LVT update. The dynamic lookahead mechanism of our conservative algorithms states that lookahead value is not fixed, and every lookahead computation could result in a different value than of the previous stage. Unlike other existing conservative algorithms, the modeler is not required to specify the lookahead of the system; rather it is dynamically extracted from the model's specifications.

-***Low-Cost Lookahead Computation***: The lookahead computation is a fast, efficient, and low-cost method that involves a simple comparison between existing parameters. In fact, there is neither an actual computation nor a significant computation time required to calculate the lookahead. Rather, the lookahead is extracted from already computed data that existed in the simulator before the conservative protocol was integrated with it. Compared to other existing conservative mechanisms, this benefit reduces the overhead of the algorithm.

- ***Deadlock Avoidance***: Since the null message distribution occurs before the LP suspension, deadlock is strictly avoided. A NC only suspends the LP after performing the lookahead computation and propagating it to the destination LPs (or the LM in case of GLM protocol) via null messages. Thus, when an LP is suspended, it has already forwarded its null messages, and if every other LP gets suspended as well, they would all resume because all the required null messages have already been distributed before suspension takes place.

4.6 The Classical P-DEVS Protocol

The LBTS, CMB, and GLM protocols extend the CD++ parallel DEVS/Cell-DEVS framework to allow running the same model with different synchronization mechanisms. The synchronization protocols are separate layers and can be replaced with one another, allowing to run the same model in sequential,

optimistic (using the LTW protocol) or conservative (LBTS,GLM,CMB) versions. These three conservative protocols are comparative to the classical P-DEVS protocol of Chow [11] and Zeigler *et al.* [8]. The classical P-DEVS protocol can be viewed as an extreme form of *risk-free optimism* (not even local rollback occurs) without incurring the overheads of conservative and optimistic schemes. Instead of trying to overlap processing of input events with different time-stamps, it seeks to exploit parallelism in the simultaneous occurrence of *internal* events among many components. There is a global coordinator synchronizing the simulation entities driving the phases of the DEVS simulation cycle.

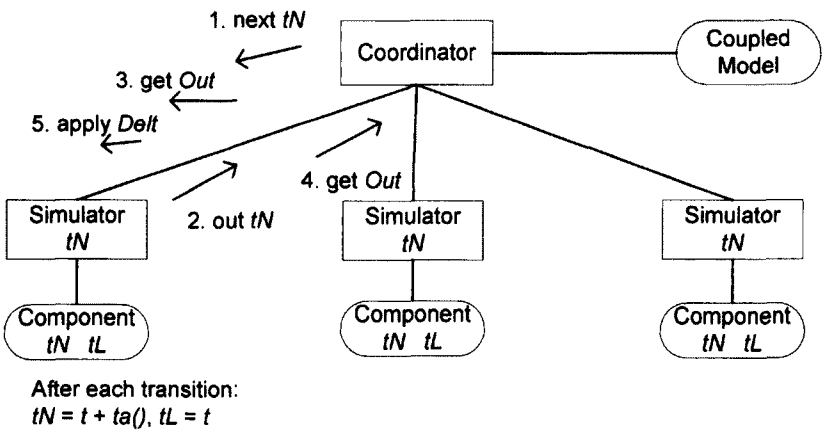


Figure 30. Parallel DEVS Simulation Protocol [82]

As shown in Figure 30, the Parallel DEVS scheme differs from the LP-based schemes in that there is a *coordinator* to synchronize the simulation cycle through its steps. The coordinator collects all times of next event from the component simulators. It sends the minimum of these times back to the components, thereby allowing them to determine whether they are imminent, and if so, to generate outputs. More than one component may be imminent and the outputs of all such imminents are sorted and distributed to others according to the coupling rules. The transition functions of the imminent components, as well as all other recipients of inputs, are applied. Depending on the state and input of a component a transition takes place – imminents with no inputs apply internal transition functions, imminents with inputs apply confluent transition functions, and non-imminent components with input apply external transition functions. The resulting changes in states may cause new values for time advances and these are sent to the coordinator to set the new global minimum time. The simulation cycle is outlined as following:

1. Set the current global time, t = the minimum of the components' t_N's (initially t_N and t_L of all components are set to zero)
2. Send t to each component
3. Each component, C, then compares t with its t_N, if $t = t_N$, this component is said to be imminent and
   - C generates its output (if any) stamped with time t
   - C executes its internal transition function
   - C sets $t_L = t$ and $t_N = t_N$ + time advance of the new state
4. The collected outputs move, as dictated by the coupling specification, to the input ports of other components
5. Each component examines its input ports and:
   - if it receives an input, it applies its external transition function with this input, using the elapsed time, $t - t_L$
   - sets $t_L = t$ and $t_N = t_N$ + time advance of the new state (if no input was received it does nothing)
6. If not at the end of the run, return to 1.

To compare the three conservative DEVS protocols (LBTS, CMB, and GLM) to the classical Parallel DEVS algorithm, recall their lookahead computation strategy. Formula (1) reveals that the lookahead is always a value in the interval [*current time*, t_N]. That is, t_N (minus the current time) would be the best lookahead that can be obtained. This might be reduced to the current time if there are unprocessed *external messages*. Under such circumstances, the protocols exploit same level of parallelism as in the classical Parallel DEVS algorithm. However, these protocols do not have a global time synchronizer, neither simultaneous output collection mechanism, nor a single centralized scheduler (*Root Coordinator*) which is proven to be a major bottleneck. Moreover, the underlying flattened architecture requires smaller number of messages when calculating the next time advance, i.e., the *flat coordinator* is the final destination of t_N values of all child *Simulators* while in Parallel DEVS protocol, there is an extra level of hierarchy (*Root Coordinator*) where the t_N values must be propagated to. Moreover, the P-DEVS simulation protocol is only able to exploit parallelism in the simultaneous occurrence of *internal* events among many components.

4.7 Zero-Lookahead in P-DEVS

Usually general DEVS/P-DEVS models have a lookahead of zero. External events can arrive at any time and the time advance function can be zero. This means that for computing a lookahead, all the time advance values in regards to the potential arrival of external messages need to be checked per atomic model. There have been a number of studies investigating the zero-lookahead problem in DEVS-HLA. Fujimoto [194] introduced zero-lookahead into HLA by extending the original time management services to allow reproducibility in the presence of simultaneous events and zero-delay events. In [82], an approach is given that uses the *Next Message Request Available - NMRA(t)* service provided by HLA instead of *Next Message Request - NMR(t)* to allow zero or negligible value of HLA lookahead for federates. The conservative protocols presented in this dissertation deal with zero-lookahead by ensuring that all possible future input events are considered when computing the lookahead value. This is done by performing a sanity check such that whenever there is an unprocessed input event in the LP's queue, the lookahead of the LP is reduced to the time stamp of such event. This avoids the case where execution of such event causes output generation with t smaller than previously computed time advance. In addition, external input events could also arrive from the environment. In a DEVS-based simulation, such messages are known to the system prior to the execution. The lookahead and the LVT computation mechanisms of the conservative DEVS protocols presented in this dissertation also take such messages into account. A special type of DEVS processor, called the *Root Coordinator* exists on LP0, which is in charge of flushing the external input events from an input file into the destination NC's input queue. Thus, when a NC invokes the LVT or the lookahead computation mechanism, by checking its *inputQueue* (as stated by Formula 1 and 2 of Section 4.2), it takes into consideration such external events.

Chapter 5: Comparative Study: Optimistic VS. Conservative Simulation

One of the objectives of this research is to perform a comparative study of optimistic versus conservative DEVS-based simulation by conducting extensive experiments. To achieve this goal, sensitivity analyses at both model- and underlying synchronization protocol-levels were studied. The analyses were carried on both the conservative simulator (CCD++, implementing the three conservative protocols: LBTS, CMB, and GLM) and the optimistic one (PCD++, implementing the LTW protocol). The results obtained from this part of the research aim to be a significant recommendation report to the parallel DEVS community to help to decide on choosing conservative or optimistic simulators for a particular model.

5.1 Model-based Sensitivity Analysis

The purpose of model-based sensitivity analysis is to analyze performance by investigating different model's characteristics, such as the following.

- **Size**

To analyze the scalability of both CCD++ and PCD++, each model is executed at different sizes. For Cell-DEVS models, the size of the model can be easily increased by altering the dimension of the cell space.

- **Type**

Two types of models are considered: communication-intensive, and computation-intensive. For the first type, models with high communication among atomic components were considered. For the latter type, models with high computations are investigated. The main reason behind testing models with different natures is to observe whether the model's type affects the performance of the underlying synchronization protocol (i.e. conservative versus optimistic).

- **Complexity**

For those models that are computation-intensive, an interesting behaviour to analyze is to increase the complexity of the model by introducing additional computations. For instance, the complexity of a

Cell-DEVS model can be increased by introducing time-consuming computations into the cells state transitions.

- **Activity**

The activity of a Cell-DEVS model is defined as the propagation pattern under which cells' values evolve. Once cells are initialized, the simulation starts when one or more cells' rules evaluate to true resulting in a propagation pattern that might eventually cover the entire cell space. Figure 31 illustrates sample activity patterns that could appear during a simulation. The activity pattern is affected by different parameters such as cells' initial value, neighborhood, and evaluation rules.

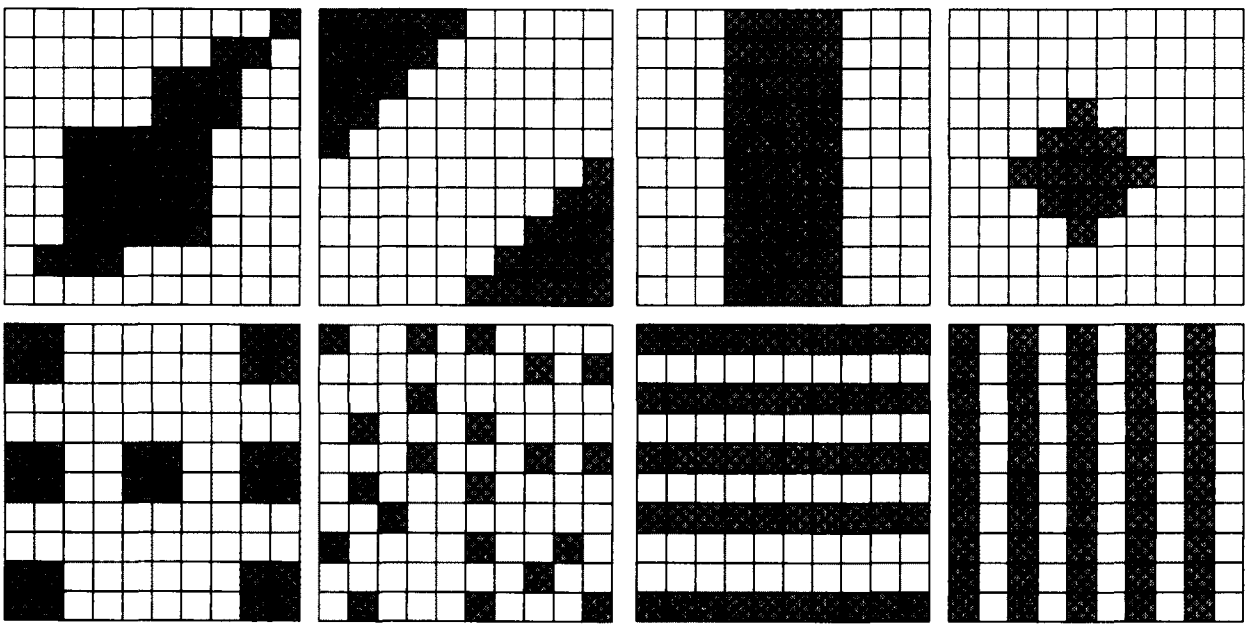


Figure 31. Sample Activity Patterns in Cell-DEVS Models

- **Partitioning**

Different partitioning strategies could result in different performance. Two partitioning mechanisms are used during the experiments: horizontal, where the cell space is divided into even columns, and vertical, which divides the cell space into even rows. These two strategies are shown in Figure 32.

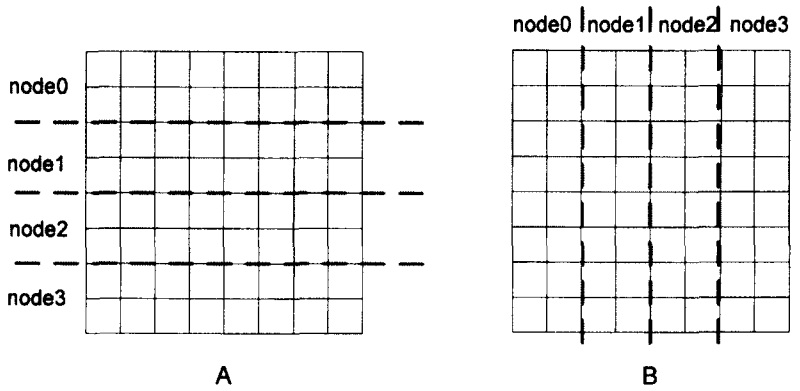


Figure 32. A: Horizontal, B: Vertical Partitioning of an 8x8 Cell-DEVS Model on Four Nodes

- **Connectivity**

In a Cell-DEVS model, cell neighboring defines the connectivity of the model by identifying how tight different partitions of the cell space are connected to each other. When in a model, a large neighborhood is defined for cells, the frequency at which cells communicate with each other increases, leading to high inter-LP communications if neighbor cells happen to be on different LPs. By altering the size of cell's neighborhood, the effect of cells' connectivity on the overall performance can be investigated. Examples of loose and tight connectivity are illustrated in Figure 33-A and Figure 33-B respectively, where the shaded cells represent the neighboring cells for the cell in the middle (note that the same neighborhood pattern applies for every cell of the lattice).

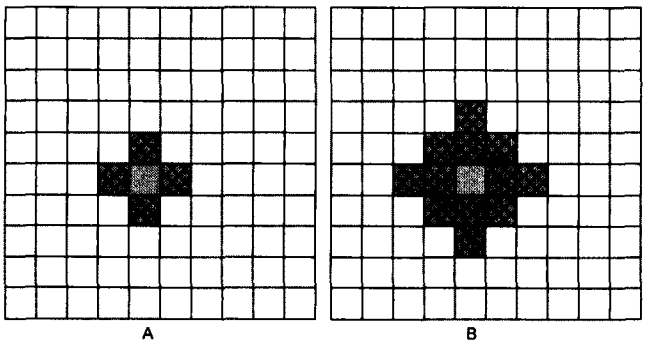


Figure 33. A: Small, and B: Large Cell Neighborhood

- **Initial-Load Distribution**

Various initial parallelism can be achieved by placing initial values at different points of the cell space as opposed to a single start-up point. This behavior significantly reduces LPs' initial idle time by allowing more parallelism among LPs especially at the beginning of the simulation. Conducting

experiments under these circumstances enables engaging more LPs throughout different stages of the simulation as opposed to keeping some LPs busy and some other waiting for cells propagation to reach their cell partitioning. Figure 34-A illustrates a scenario at which only a single initial value is placed on the gird which happens to be the partition assigned to *node1*. Thus, for some simulation stages, only *node1* will be busy while the rest of the nodes remain idle. Figure 34-B shows the scenario where there is a starting point on every node's partition, allowing full parallelism from the initial stage of the simulation.

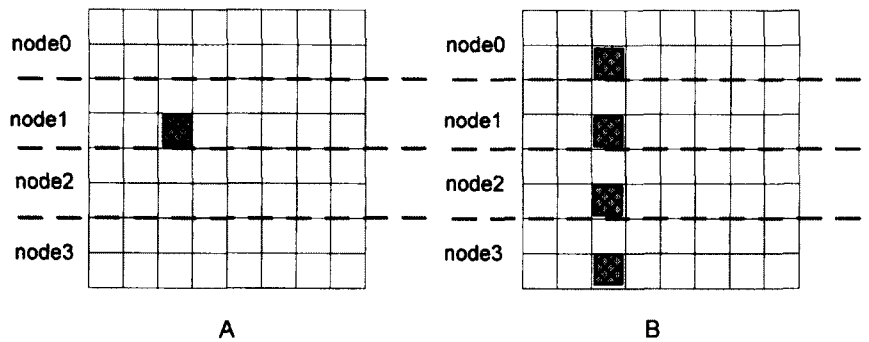


Figure 34. Initial-Load Distribution by Setting Multiple Initial Points

5.2 Protocol-based Sensitivity Analysis

The optimistic simulator, PCD++ [21], is based on Lightweight Time Warp (LTW) [34], a novel TW-based protocol. The protocol includes a rule-based event-scheduling mechanism using two types of event queues, an aggregated state-saving technique for optimal risk-free state management, and a new rollback algorithm that recovers lightweight LPs from causality errors without sending anti-messages.

Identifying the key metrics for a decent sensitivity analysis is a complex task, especially for a TW simulator that can be optimized in so many different ways. Aside from the model-based parameters, there are many options at the simulator level that can be combined to conduct various experiments. Some of these analyses, which are the characteristics of the TW algorithms, include GVT computation frequency, state-saving interval, event queue operation efficiency, etc.

The conservative simulator, CCD++, is tested under the three conservative synchronization protocols: LBTS, CMB, and GLM. All of the experiments will be conducted with these conservative protocols to provide a solid performance evaluation, analyzing the effect of the underlying conservative protocol on the simulation performance.

Sensitivity analysis is a complex task that requires many experiments to identify those factors that significantly affect the performance. The next chapter will present a thorough sensitivity analysis by conducting various tests addressing different scenarios that were mentioned in this chapter.

Chapter 6: Performance Analysis

This chapter analyzes the performance of the conservative protocols (LBTS, CMB, and GLM) and the optimistic LTW protocol. Section 6.1 introduces the benchmark models used in the experiments. Section 6.2 summarizes the experimental configurations and performance metrics. Section 6.3 presents a comparative performance evaluation of the LBTS, GLM, and the LTW protocols on distributed-memory multiprocessor clusters using CCD++ and PCD++ simulators, while Section 6.4 evaluates the three conservative protocols in terms of protocol efficiency and effect of null message distribution strategy on the overall performance. Finally, Section 6.5 provides detailed analyses of the three conservative protocols by evaluating some of the sensitivity metrics that were presented in Chapter 5.

6.1 Introduction to Benchmark Models

The clear separation of model and simulator concepts in the CD++ M&S framework offers a number of advantages. The tool provides a modelling and simulation environment that allows easily constructing and modifying any DEVS and Cell-DEVS model. Moreover, the same model can be used on different environments (PCD++ with LTW protocol and CCD++ with LBTS, CMB, or GLM protocols) without the need to make any changes.

Two environmental models with varied workload characteristics were tested in the experiments, namely a *wildfire spread model* and a *watershed model*. These models have been studied extensively in the DEVS research community (see, e.g., wildfire simulation [195][196][197][198] and watershed simulation [5][199][200][112]). In [116], the wildfire and watershed models have been redefined as executable Cell-DEVS models in the CD++ specification language, as briefly described in this section. In addition, a third model was also used in the experiments, the *Synth* model, which is a synthetic Cell-DEVS model consisting of a grid where cells are initially set to zero, then throughout the simulation, they toggle between the value of 0 and 1. The purpose of this model is to analyze the performance of parallel execution of communication-intensive models.

6.1.1 Definition of a Wildfire Model

Two versions of the wildfire model were evaluated in the experiments, including a *simplified* version, referred to as ***Fire1***, which uses predetermined fire spread rates at reduced runtime computational cost; and a *computational-based* version, referred to as ***Fire2***, which computes fire spread rates dynamically based on environmental parameters obtained at runtime, resulting in higher computational intensity. Both versions simulate fire propagation scenarios over 90 virtual hours in a 2D cell space.

- **The Fire1 model**

The *Fire1* model [116] uses the Rothermel method [114] to obtain the spread rate in every direction prior to the simulation based on a specific set of environmental parameters (a fuel model type value of 9, a southwest wind at a speed of 24.135 km/h, and a cell size of 15.24×15.24 m^2), as summarized in Figure 35.

```
Wind direction = 225.00 (bearing)  Wind speed = 24.135 km/h   Fuel model type = 9
Cell width = 15.24 m (E-W)         Cell height = 15.24 m (N-S)

Max spread rate = 17.967136
0° rate = 5.106976 distance = 15.24      45° rate = 17.967136 distance = 21.552615
90° rate = 5.106976 distance = 15.24     135° rate = 1.872060 distance = 21.552615
180° rate = 1.146091 distance = 15.24    225° rate = 0.987474 distance = 21.552615
270° rate = 1.146091 distance = 15.24    315° rate = 1.872060 distance = 21.552615
```

Figure 35. Predetermined Spread Rates for the *Fire1* Model [116]

Figure 36 gives a skeleton of the *Fire1* model definition. A cell's value stands for the virtual time when the cell is ignited (zero for a non-burning cell). The precondition of a transition rule is used to detect the presence of fire in a specific neighboring cell. For example, the first rule will be triggered if the current cell (0,0) is non-burning and the southwest neighbor (1,-1) has already been ignited. Hence, the fire will spread to the current cell at virtual time (1,-1) + (21.552615/17.967136), which becomes the new value of the current cell. This value will be sent to the neighboring cells after a delay of (21.552615/17.967136) * 60000 ms, the interval between the current virtual time and the expected ignition time at the cell.

```
type : cell          dim : (1024,1024)   delay: inertial     border : nowrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1) (0,0) (0,1) (1,-1) (1,0) (1,1)
localtransition : FireBehavior

[FireBehavior]
rule : {(1,-1)+(21.552615/17.967136)} {(21.552615/17.967136)*60000} {(0,0)=0 and 0<(1,-1)}
rule : {(1,0)+(15.24/5.106976)} {(15.24/5.106976)*60000} {(0,0)=0 and 0<(1,0)}
rule : {(0,-1)+(15.24/5.106976)} {(15.24/5.106976)*60000} {(0,0)=0 and 0<(0,-1)}
...
```

Figure 36. A Skeleton of the *Fire1* Model Definition in CD++ [116]

• The Fire2 model

The *Fire1* model has been generalized in [32] to allow for determination of fire spread rates by changing environmental parameters [23]. Specifically, the CD++ specification language has been extended to include a new syntax node, referred to as `fsr` [32], which calculates the spread rate in any given direction at runtime by invoking the `fireLib` library [115].

Figure 37 shows a skeleton of the generalized *Fire2* model when defined under the same environmental conditions as shown in Figure 35.

```
type : cell          dim : (1024,1024)   delay: inertial     border : nowrapped
neighbors : (-1,-1) (-1,0) (-1,1) (0,-1) (0,0) (0,1) (1,-1) (1,0) (1,1)
localtransition : FireBehavior

[FireBehavior]
rule : {(1,-1)+(21.552615/fsr(225,9,273.4,225))} {(21.552615/fsr(225,9,273.4,225))*60000}
                                                              {(0,0)=0 and 0<(1,-1)}
rule : {(1,0)+(15.24/fsr(180,9,273.4,225))} {(15.24/fsr(180,9,273.4,225))*60000}
                                                              {(0,0)=0 and 0<(1,0)}
rule : {(0,-1)+(15.24/fsr(270,9,273.4,225))} {(15.24/fsr(270,9,273.4,225))*60000}
                                                              {(0,0)=0 and 0<(0,-1)}
...
```

Figure 37. A Skeleton of the *Fire2* Model Definition in CD++

Comparing Figure 37 with Figure 36, the spread rate in a given direction is no longer a fixed constant in the *Fire2* model. Instead, it is the computation result of the `fsr` syntax node based on a set of four parameters (i.e., azimuth, fuel type, wind speed, and wind direction), which are provided by the runtime environment of an application. This allows obtaining highly dynamic and realistic simulation results by feeding real-time environmental data into the model. As expected, the time for processing a (*, t) event at the Simulators becomes 6.68 times longer than what is required in the *Fire1* model (calibrated on a 3.2GHz Intel Xeon processor), a significant increase in computational intensity.

6.1.2 Definition of a Watershed Model

The *Watershed* model [116] uses a 3D cell space to simulate water accumulation in a drainage basin over 30 virtual minutes under constant rain condition (7.62 mm/h) based on a set of hydrological equations [7]. In addition, different types of ground soil (grass and stones) are also considered in the *Watershed* model by defining zones with different local transition functions within the cell space. Figure 38 shows a skeleton of the *Watershed* model definition in the CD++ environment.

```
type : cell          dim : (320,320,2)   delay: inertial     border : nowrapped
neighbors : (-1,0,0) (0,-1,0) (0,0,0) (0,1,0) (1,0,0) (-1,0,1) (0,-1,1) (0,0,1) (0,1,1) (1,0,1)
zone : grass {(0,0,0)..(319,50,0)}              stones { (0,270,0)..(319,319,0) }
localtransition : hydrology

[grass]
rule : {0.07 + (0,0,0) - if(((((0,0,1) + (0,0,0))>((-1,0,1) + (-1,0,0)))),((((0,0,0) + (0,0,1)
- (-1,0,0) - (-1,0,1))/1000) * (0,0,0))/1000),0) - if(((((0,0,1) + (0,0,0))>((1,0,1)  +
(1,0,0)))),((((0,0,0) + (0,0,1) - (1,0,0) - (1,0,1))/1000) * (0,0,0))/1000),0) - if(((((0,0,1)
+ (0,0,0))>((0,-1,1)+(0,-1,0)))),((((0,0,0) + (0,0,1) - (0,-1,0) - (0,-1,1))/1000) * (0,0,0))/
1000),0) - if(((((0,0,1) + (0,0,0))>((0,1,1) + (0,1,0)))),((((0,0,0) + (0,0,1) - (0,1,0) -
(0,1,1))/1000) * (0,0,0))/1000),0) + if(((((-1,0,1) + (-1,0,0))>((0,0,1) + (0,0,0)))),((((-
1,0,0) + (-1,0,1) - (0,0,0) - (0,0,1)) * (-1,0,0))/1000),0) + if((((1,0,1) + (1,0,0))>((0,0,1)
+ (0,0,0)))),((((1,0,0) + (1,0,1) - (0,0,0) - (0,0,1)) * (1,0,0))/1000),0) + if(((((0,-1,1) +
(0,-1,0))>((0,0,1) + (0,0,0)))),((((0,-1,0) + (0,-1,1) - (0,0,0) - (0,0,1)) * (0,-1,0))/
1000),0) + if(((((0,1,1) + (0,1,0))>((0,0,1) + (0,0,0)))),((((0,1,0) + (0,1,1) - (0,0,0) -
(0,0,1)) * (0,1,0))/1000),0) } 1000 { cellpos(2)=0 }
rule : { (0,0,0) } 1000 { t }

... //local transition functions for [stones] and [hydrology] are omitted here
```

Figure 38. A Skeleton of the *Watershed* Model Definition in CD++ [Wai06]

In the model, the height of accumulated water at a cell depends on the rain intensity, the water exchanged with the neighboring cells (both inflows and outflows), and the amount of water absorbed by ground soil of different types. A local transition function thus computes future height values for the cells at each virtual time, taking into account the initial water level, the cumulative rain precipitation, the dynamic water flow between the cells, and the specific soil condition. The 3D cell space is composed of two planes: *plane 0*, for representing the ever-changing heights of retained water at different cells; and *plane 1*, for defining the topographical configuration of the terrain which remains unchanged throughout a simulation.

6.1.3 Definition of the Synthetic Model

The *Synth* model defines a 2D cell space that is initially filled with zeros. Each cell of the gird defines eight neighbors and evaluates two simple rules, changing its value from zero to one and vice versa throughout the simulation. Figure 39 shows a skeleton of the *Synth* model definition in the CD++

environment. The large cell's neighborhood and the simple low-computation rules create dense communication allowing analyzing parallelism with communication-intensive models. The simulation scenarios for this model are conducted over 100 milliseconds.

```
type : cell   dim : (100,100)  delay : inertial defaultDelayTime  : 0 border : nowrapped
neighbors : synth(-1,-1) synth(-1,0) synth(-1,1) synth(0,-1)  synth(0,0)  synth(0,1) synth(1,-1)
synth(1,0)  synth(1,1)
initialvalue : 0  localtransition : modelBehavior
[modelBehavior]
rule : 1       1  { (0,0) = 0}
rule : 0       1  { (0,0) = 1}
rule : {(0,0)} 1  { t }
```

Figure 39. A Skeleton of the *Synthetic* Model Definition in CD++

6.2 Experimental Configurations and Performance Metrics

The performance of the conservative protocols (LBTS, CMB, and GLM) and the optimistic LTW protocol were studied in the experiments using CCD++ and PCD++ simulators respectively. The performance results presented in the following sections not only depend on the degree of parallelism available in the tested models, but also depend on the specific experimental configurations summarized here. Consequently, they should be viewed as indicators of potential performance gain that is achievable by the proposed protocol. Experiments were conducted on a cluster of 28 HP Proliant DL140 servers running on Linux WS 2.4.21 and communicating over Gigabit Ethernet using MPICH 1.2.7 [Gro09], which is a portable implementation of the MPI standard [112]. Each cluster node features dual 3.2GHz Intel Xeon processors with 1GB 266MHz main memory and 2GB disk swap space. Note that severe memory-swapping activities will occur if the maximum space requirement of a simulation approaches (or goes beyond) the physical limit of 1GB on a node. Moreover, a simulation will fail to complete when the memory usage cannot be contained within the maximum allowable virtual memory space of 3GB (i.e., the accumulated size of physical memory and disk swap space).

To ensure a fair comparison, the two simulators (CCD++ and PCD++) were configured the same way. Table 2 shows a list of 6 performance metrics collected in the experiments through extensive measurement. Among them, the *total execution time* (T) and the *maximum memory consumption* (MEM) are the two primary metrics used to compare the overall simulation performance of optimistic simulation versus the conservative versions.

Table 2. Performance Metrics

Metrics	Description
T	Total execution time of the simulation (sec)
BT	Total blocked time during the simulation (sec)
MEM	Maximum memory consumption (MB)
PEV	Total number of positive events executed
NEV	Total number of null events executed
NMR	Null message ratio

To demonstrate the absolute performance of both protocols, the benchmark models were also executed using a sequential CD++ simulator on a single cluster node, with the corresponding metrics (denoted as T_{seq} and MEM_{seq}) collected in the experiments. Using the *sequential* simulation as the baseline case, the *overall speedup* of a parallel simulation on N cluster nodes is thus defined as follows.

$$\text{Overall Speedup} = \frac{Tseq}{T(N)}, \text{ where } N>1 \qquad (1)$$

The other metrics in Table 2 (i.e., BT, PEV, NEV, NMR) provide additional insight into the impact of the three conservative protocols, allowing for an objective assessment of the effectiveness of the proposed synchronization protocol. The experimental results for each test case were conducted based on 95% confidence interval for all the test cases. For the test cases on multiple nodes, the results were also averaged over the participating nodes to obtain a *per-node* evaluation (i.e. BT, MEM, PEV, and NEV represent the corresponding results per one node). The PEV values present the total number of DEVS messages executed during the simulation. The null message ratio (NMR) is a commonly used performance metric for null message-based conservative protocols. It is defined as the ratio of the number of null messages to the number of regular messages during the simulation as follows.

$$\text{NMR} = \frac{NEV}{PEV} \qquad (2)$$

6.3 Evaluation of the Conservative and Optimistic Protocols

This section analyzes the performance of the LBTS, GLM, and LTW protocols using the benchmark models introduced in Section 6.1. Using the logs that are generated during the simulation, these models

have been verified prior to the actual performance testing to ensure that the parallel simulations generate the same results as the corresponding sequential simulations. Later, the event-logging capability of the simulators was turned off in all test cases to minimize the impact of file I/O operations on simulation performance. A simple partition strategy was used which evenly divided the cell space into horizontal rectangles. As will be presented in Section 6.5, another partitioning mechanism was also used to observe the effect of partitioning scheme on the overall performance. The *Fire1* and *Fire2* models were tested using cell spaces of 100x100, 200x200, 300x300, and 500x500, and the *Watershed* model was tested with 25x25x2, 30x30x2, 50x50x2, and 100x100x2 cells. While the *Synthetic* model was tested with 100x100, 200x200, and 400x400 cells.

Figure 40 illustrates the total execution time on 1 to 26 nodes for *Fire1* model with 100x100, 200x200, 300x300, and 500x500 cells respectively. The graphs also show the min and max values for each scenario which happened to be in the 5% interval (to avoid cluttering the graphs, the results for the other models will only include the averaged values). Considering the two conservative protocols (GLM and LBTS) it can be seen that for each different size, the GLM protocol reduces the execution time compared to LBTS for every given number of nodes. Meaning that, for the four mentioned sizes, the smallest execution time is always achieved by the GLM protocol compared to LBTS. It is also shown that for any given number of nodes, the execution time always increases with the size of the model. While the LBTS protocol increases the execution time after it reaches the smallest execution time for that size, the GLM keeps reducing the execution time as the number of nodes increase in most cases. In fact, the GLM protocol shows much better performance compared to the LBTS algorithm for all scenarios. This is due to the significant null message reduction that the GLM provides as shown on the null message ratio graphs in Figure 42.

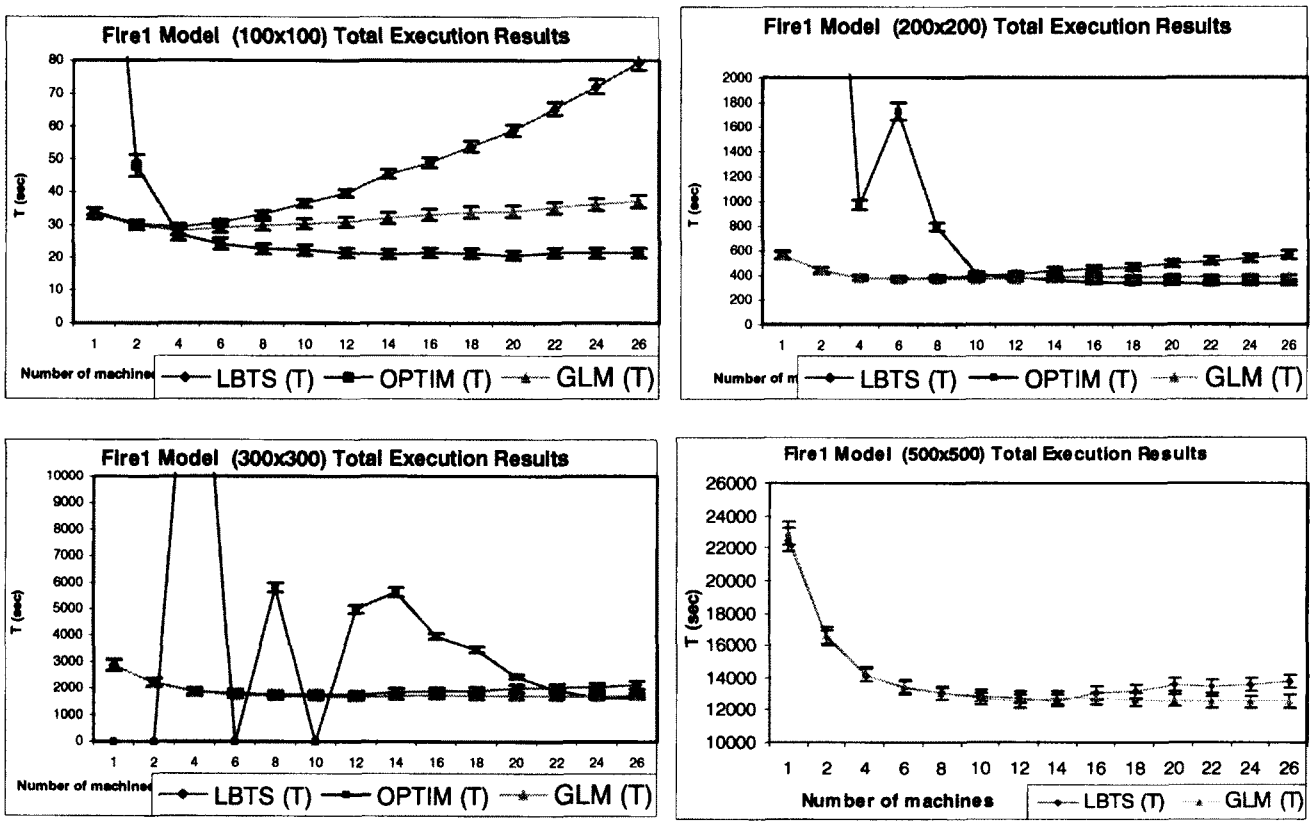


Figure 40. *Fire1* Results for Various Sizes

By comparing the GLM protocol with the optimistic one (OPTIM) different results are observed. For each size of the model, the optimistic protocol outperforms the conservative ones only after a certain number of nodes. That is, the OPTIM protocol shows smaller execution time compared to the conservative protocols only after 4, 14, and 24 nodes for 100x100, 200x200, and 300x300 sizes respectively. This is due to the high overhead of the optimistic protocol caused by numerous rollbacks, state savings, and memory consumption, which made the simulator unable to run any simulation for the 500x500 size of the model, and also failed to run the simulations on 1, 2, 6, and 10 nodes for the 300x300 size. Moreover, at some cases (e.g., simulation on 1 to 8 nodes for the 200x200 size) it was observed that the performance of the optimistic simulation is even worse than that of the sequential execution (with a speedup of less than 1), mainly because of the excessive communication and operational overhead incurred in the optimistic parallel simulation.

In terms of speedups, the GLM protocol always resulted in better speedup compared to the LBTS algorithm. It is also observed that the GLM protocol provides higher speedups compared to the

optimistic one as the size of the model increases. However, at smaller sizes of the model (i.e. 100x100 and 200x200) the overhead of both of the conservative protocols (GLM and LBTS), which is the null message distribution plus the total blocking time of the LPs, were much higher than the benefit that was gained by executing the model in parallel. That is, when the model is relatively small, the overhead of the optimistic simulator tends to be smaller than those of the conservative ones.

A general trend reflected in the experimental results is that the reduction in the total execution time and maximum memory consumption is greater for models with larger sizes, indicating an improved scalability of the synchronization algorithms.

Moreover, the blocked time (BT) and null message ratio (NMR) of the conservative protocols were also conducted to investigate their performance in details. As shown in Figure 41, the average blocked time values associated with the LBTS protocol are much larger compared to the GLM protocol. This is more clear as the number of participating nodes increases. However, there are scenarios at which the BT values of the GLM were slightly higher than those of the LBTS algorithm (e.g. 2 nodes scenario of 200x200, 300x300, and 500x500 sizes). This can be explained by the nature of the GLM protocol where the overhead it produced at 2 nodes is slightly higher than what was produced by LBTS (under the GLM scheme, the two nodes must send their null message to the central manager and then wait for the response null message from it, while with the LBTS, the nodes directly send to each other). However, as the number of nodes increases this behavior is no longer observed and the overhead of GLM stars to decrease compared to that of the LBTS because the null message distribution strategy of the GLM protocol causes less overhead. On the other hand, the NMR results shown in Figure 42 clearly explain how the GLM outperforms the LBTS protocol by significantly reducing the number of null messages, thus, markedly improving the NMR.

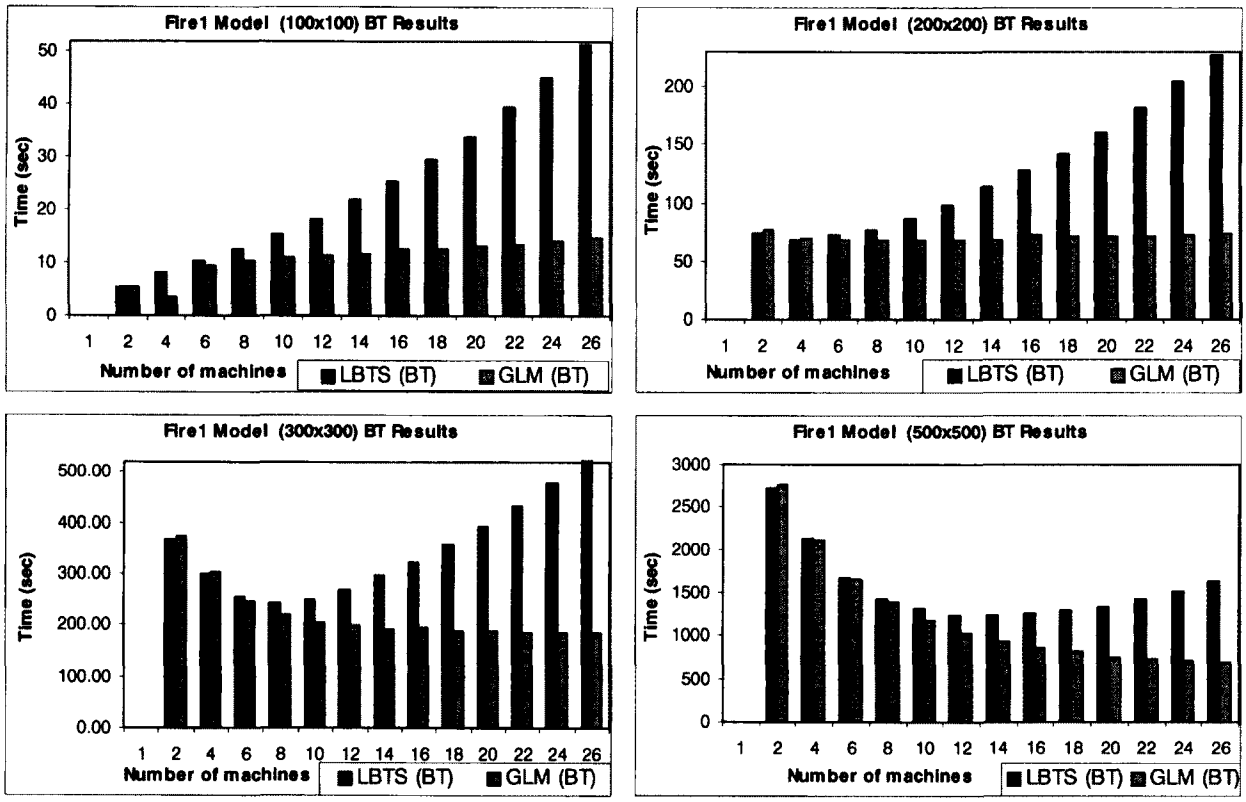


Figure 41. BT Results of *Fire1*

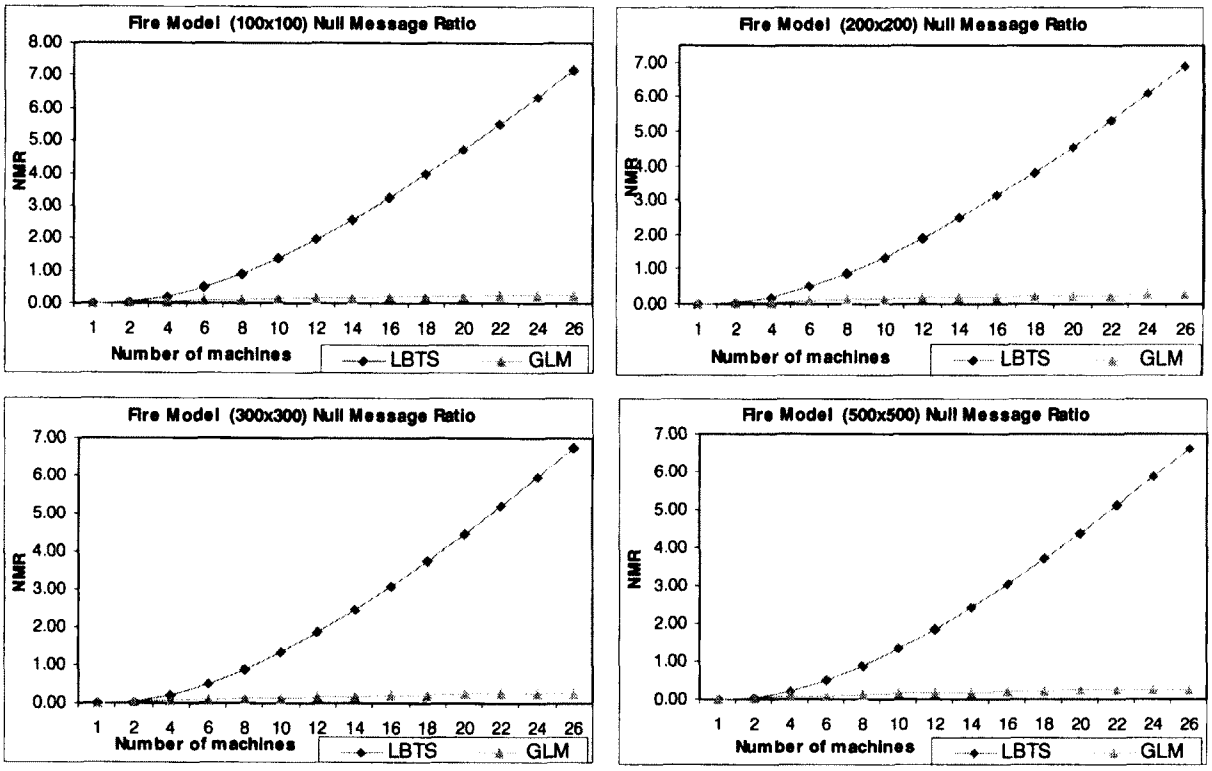


Figure 42. NMR Results of *Fire1*

Looking at the memory consumption of the three protocols as illustrated in Figure 43, it is shown that the average maximum memory consumption per node decreased in the same manner for both conservative protocols. However, the memory consumption associated with the optimistic protocol was much higher compared with the other two protocols which is as expected. At some cases, the maximum memory required per node under the optimistic scheme was so high that the simulation could not be completed due to memory exhaustion. In addition, it happened that memory consumption did not follow a steady reduction pattern, as shown for the case of 4 nodes running the 200x200 model, the average memory consumption is lower than the case when 6 nodes are participating. This is very dependent on the specific rollback and state savings that occurred in these two scenarios, causing the average memory consumption per node to be lower than when there are only 4 nodes compared to the 6 nodes scenario. Similar behavior was noticed for the 300x300 size with 8 nodes scenario.

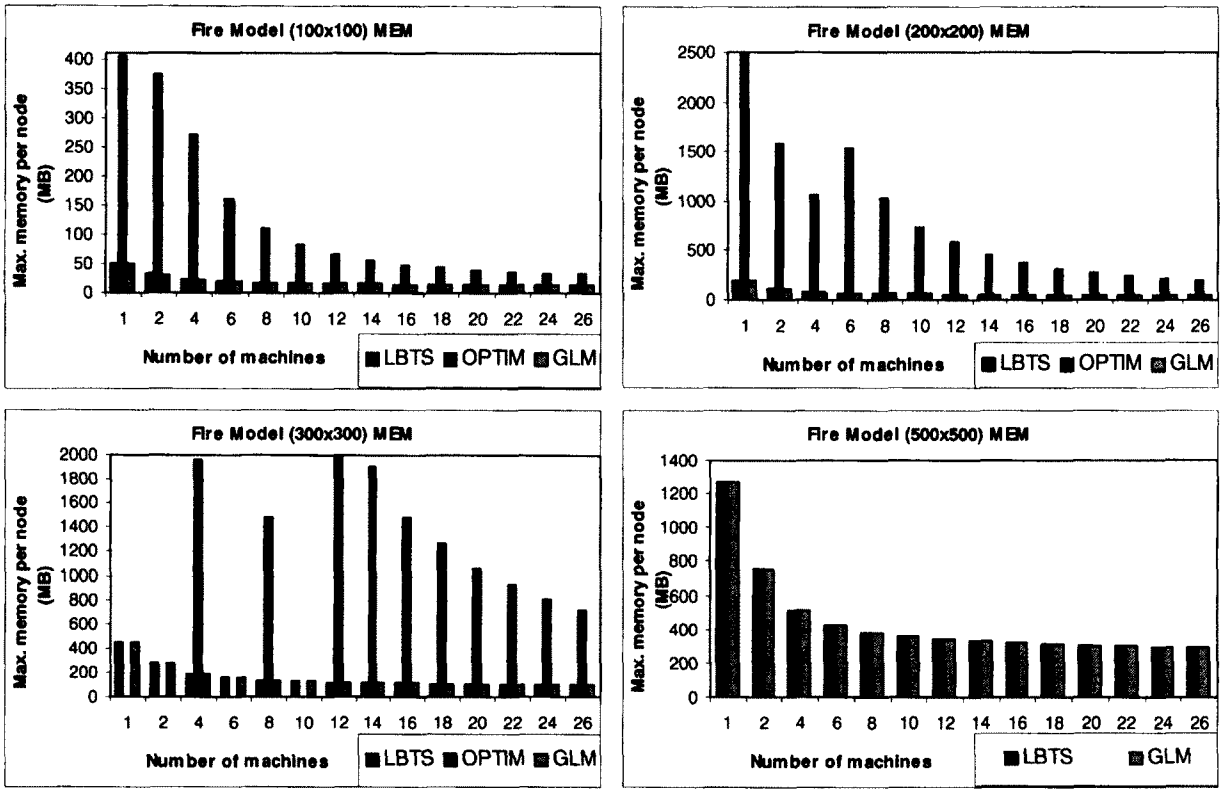


Figure 43. Memory Consumption for *Fire1* Model

Similarly, Figure 44 illustrates the total execution time for *Fire2* model at various sizes for the three protocols. As in *Fire1*, on each different size, the GLM protocol reduced the execution time and the total number of null messages significantly. The performance achieved by the GLM protocol stayed high as the number of nodes increased which was not the case for the LBTS protocol where the performance started to drop down as more nodes were engaged. The null message reductions results are presented in Figure 46. A different performance was observed at the largest size (the 500x500 model), where the LBTS outperformed the GLM with 4, 6, 8, 10, 12, and 14 nodes. However at 1, 2, and above 14 nodes, the GLM provided smaller execution times. This can be explained by considering the nature of the model which is computation-intensive, thus, as the model size is larger the computation phases take longer, and specifically within the GLM scheme larger overhead is introduced into the simulation because the LM had to wait for all LPs to send their lookahead before it could distribute the globally minimum lookahead value and thus resume the blocked LPs. Now, considering the optimistic protocol, it was observed that when the size of the model is small, the best performance is achieved by this protocol, but as the size of the model increased the GLM protocol outperformed the optimistic one,

specifically when fewer nodes were engaged. That is, for the 100x100 size, the GLM protocol resulted in smaller execution time with 1 and 2 nodes. In contrast, as the size grew (e.g., the 300x300 model), the GLM continued to perform better than the optimistic one until 20 nodes. This was while the optimistic simulator could not carry out the simulations at 1 to 10 nodes for the 300x300 size, and similarly not able to run any simulation even with the maximum number of available nodes for the largest size (i.e. 500x500) due to high memory consumption.

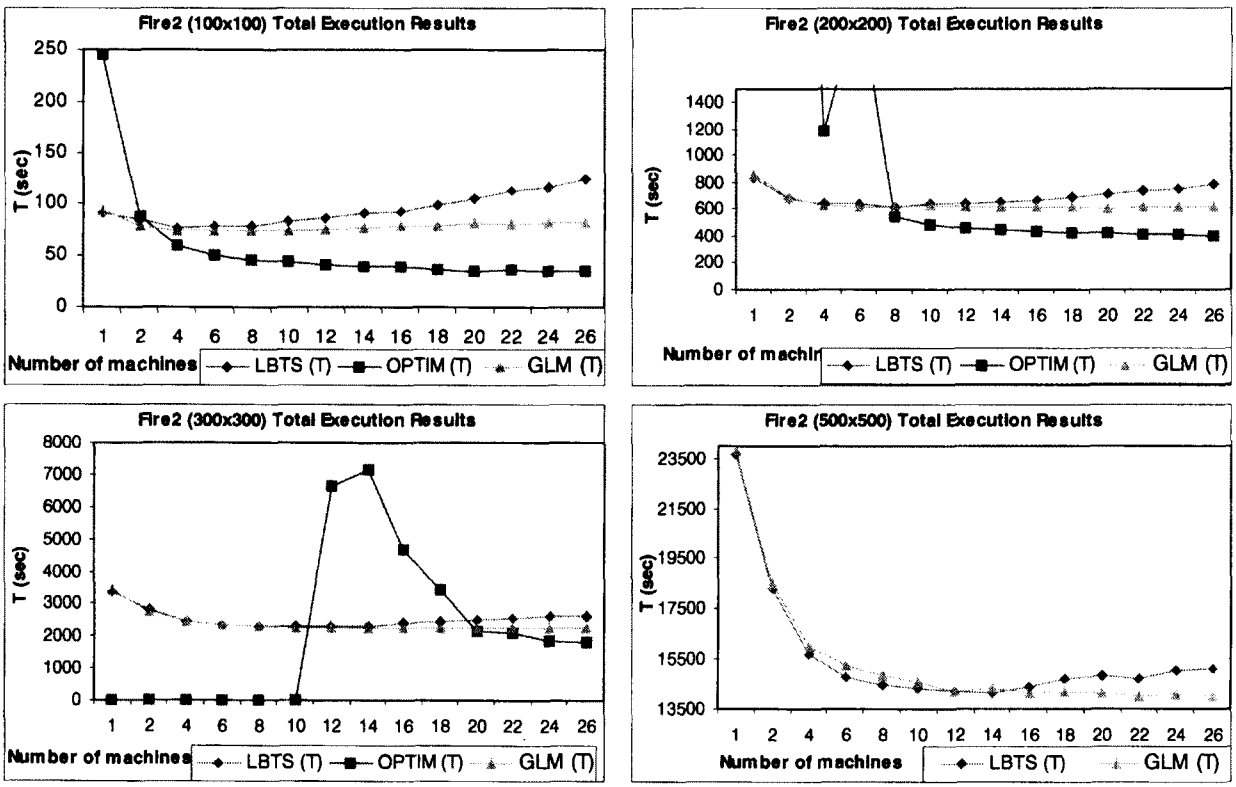


Figure 44. *Fire2* Results for Various Sizes

The average blocked time for the two conservative protocols for two sizes of the model (i.e. 100x100 and 500x500) are presented in Figure 45. As expected, with the smallest size model, the blocked time values for both of the conservative protocols increased as more nodes were engaged due to the overhead of the protocols which nullified the performance gain of parallel simulation. However, it can be seen that the BT associated with GLM is much smaller compared to that of LBTS. On the other hand, when the model was much larger, different behavior was observed. The GLM caused much smaller BT values, and at the same time, as the number of nodes increased, the BT values decreased linearly. This shows that when the model is large and complex, the benefits gained from the GLM

protocol overcome the overhead issues of the protocol, while this statement was not true for the LBTS protocol. That is, for the LBTS protocol when the simulation was conducted over 10 nodes, the BT values started to increase as more nodes were engaged. This shows that the overhead of the LBTS protocol starts to have a negative impact on the performance after the number of nodes reaches a specific range mainly due to the significant increase of the total number of null messages that got distributed among the LPs.

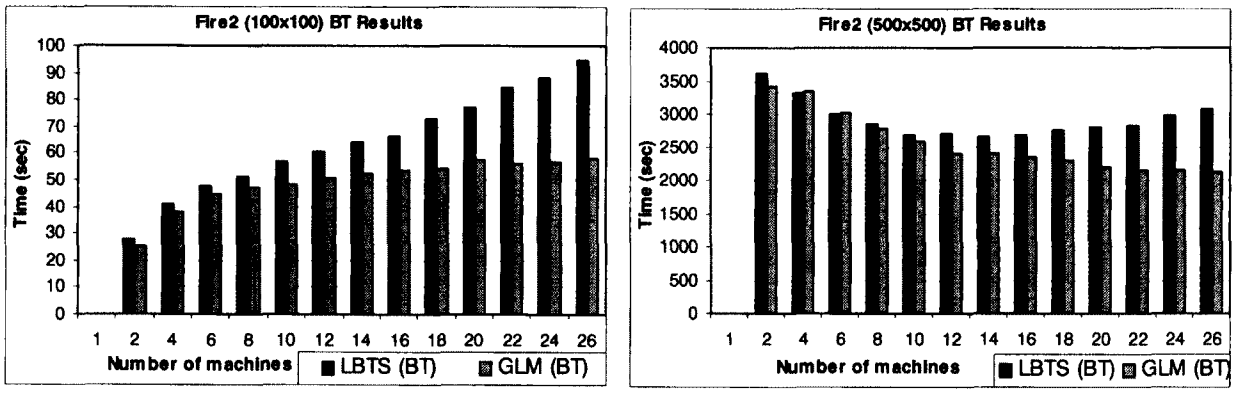


Figure 45. BT Results of *Fire2*

The NMR results of *Fire2* model are given in Figure 46. For the various sizes of the model, similar NMR results were obtained, thus only the results for two of the sizes (100x100 and 500x500) are illustrated. Similar to *Fire1* model, for various sizes of the model, the NMR values at different nodes configuration were significantly lowered by the GLM protocol. The memory consumption results illustrated in Figure 47 show the significant memory consumption reduction of the conservative protocols compared to the optimistic protocol.

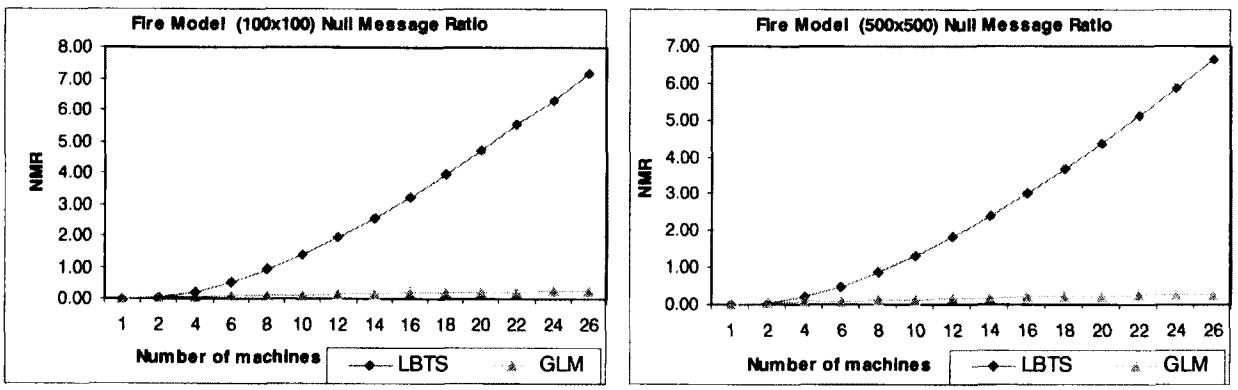


Figure 46. NMR Results of *Fire2*

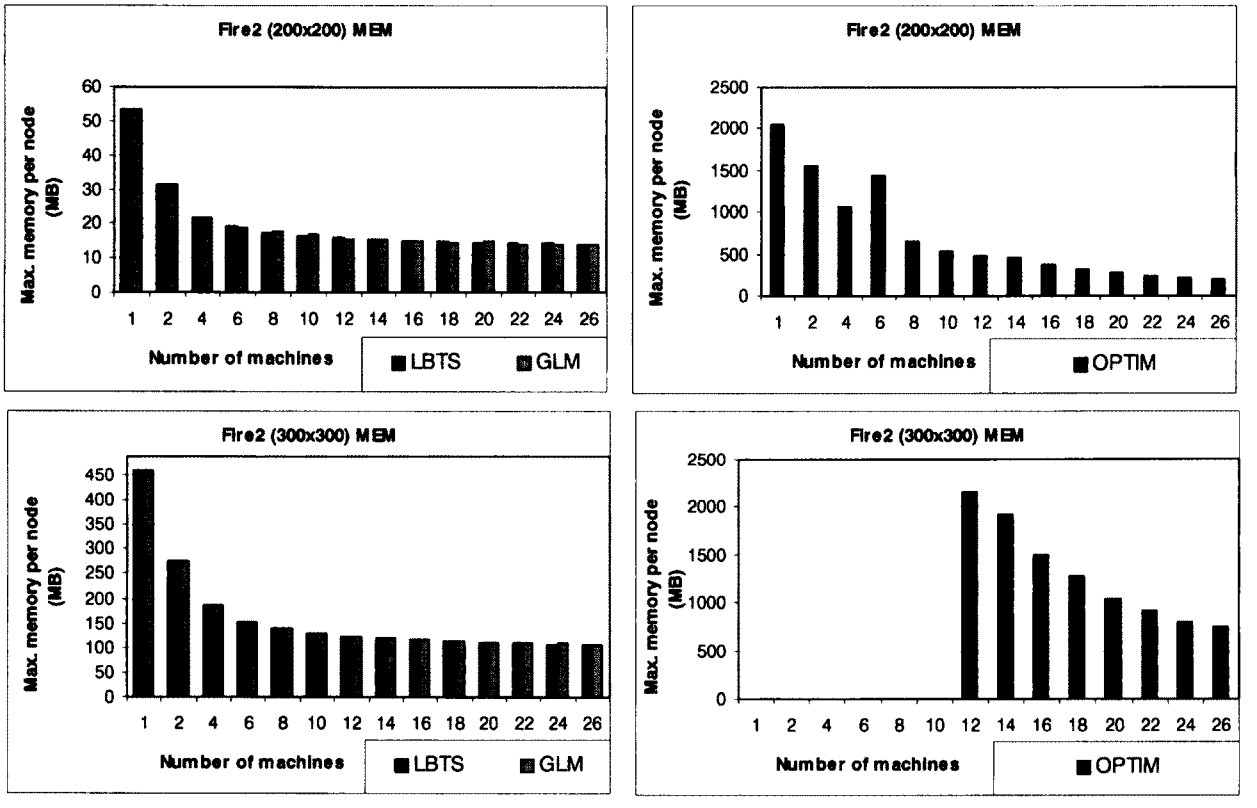


Figure 47. Memory Consumption for *Fire2* Model

Figure 48 presents the execution results of the third model, *watershed.* As was mentioned earlier, this model is three dimensional and communication-intensive, thus most of the execution time is spent on messaging rather than computation. From the total execution time values it is observed that overall, the GLM conservative protocol outperformed the other two mechanisms at small number of nodes, but as the number of participating nodes increased, the OPTIM protocol outperformed the conservative ones. However, with less number of nodes (e.g. 2, 4, and 6) the GLM protocol gave the smallest execution time, and after this, by increasing the number of nodes the optimistic protocol provided the best results. Although the GLM protocol resulted in very similar execution time values, however, it can be concluded that the impact of the optimistic protocol is most evident in the *Watershed* simulations, where the number of simultaneous events executed at each virtual time grew with the model sizes, making the optimistic protocol the synchronization of the choice when executing communication-intensive models.

With regards to the two conservative protocols, the GLM protocol outperformed the LBTS for nearly all cases indicating that the null message distribution mechanism of this protocol produces much smaller overhead compared to the LBTS mechanism.

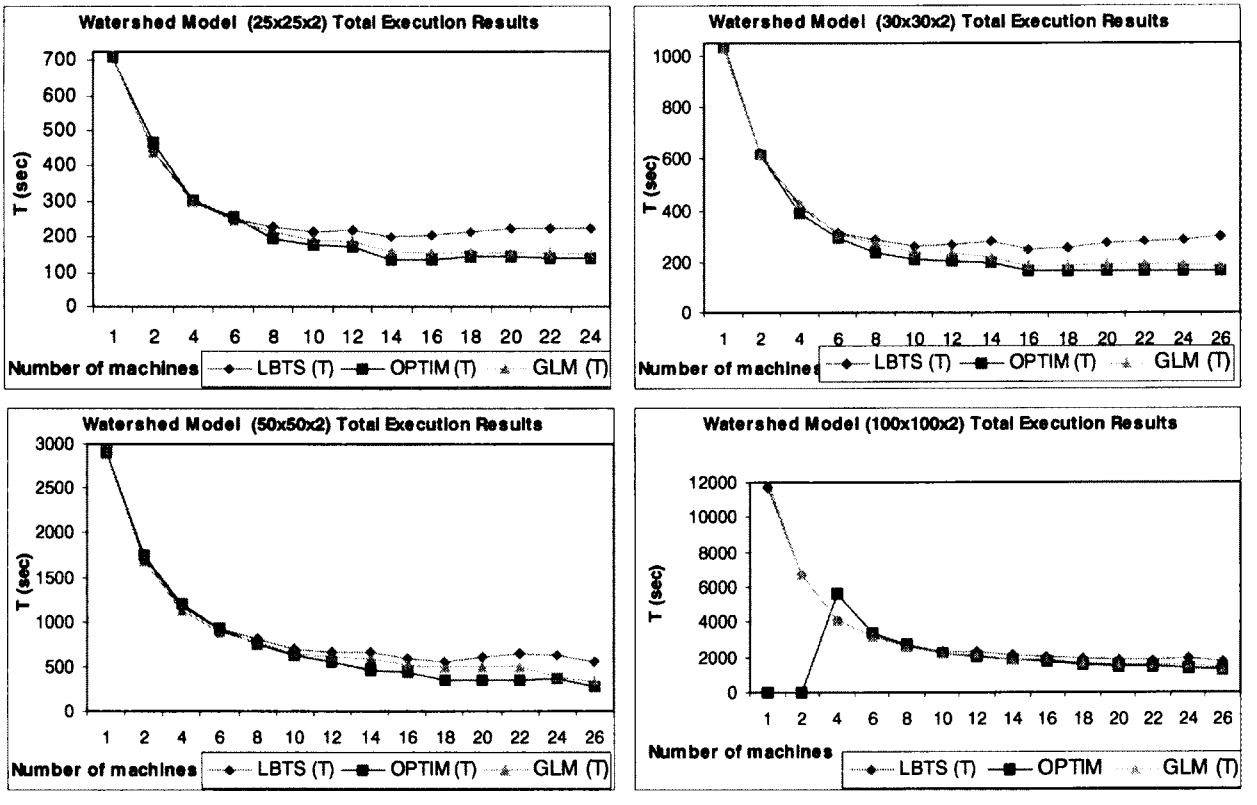


Figure 48. *Watershed* Results for Various Sizes

It is shown that much higher speedups are achieved compared to the two fire models because of the nature of the model, which involved numerous cell updates (large number of simultaneous events executed at each virtual time), thus, allowing to benefit from parallel simulation much better. For BT, as shown in Figure 49, with the first three sizes of the model, the BT values associated with GLM were smaller for nearly all cases. In addition, for each of these sizes, as there were more nodes, the difference between the BT values of the GLM and the LBTS protocols increased more. A different behavior was seen at the 100x100x2 size, where the GLM caused longer BT at most cases compared to LBTS. Although, the total execution time results showed that the GLM protocol outperforms the LBTS for this size of the model, however, due to the nature of the model which is 3D highly-communicative, and the tight neighboring that existed among the cells, the overall synchronization phases tended to take much

longer under the GLM scheme. That is, since each node had more atomic components (i.e. cells) assigned to it, and thus longer time to finish up every computation/communication phase, the central synchronizer was required to wait longer for all nodes to broadcast their null messages. As a result, longer waiting times (i.e. blocked times for each LP) were associated with each synchronization phase compared to the LBTS scheme.

However, this did not affect the overall performance (in terms of the execution time), because the GLM protocol keeps the event queue of each node much shorter (at any time, at most one null message is kept in the input queue) compared to LBTS (where at any time, there are at most *n-1* null messages in the queue, where *n* is the total number of participating nodes). Thus, by keeping the message queues shorter, the queue operations which are performed very often took shorter.

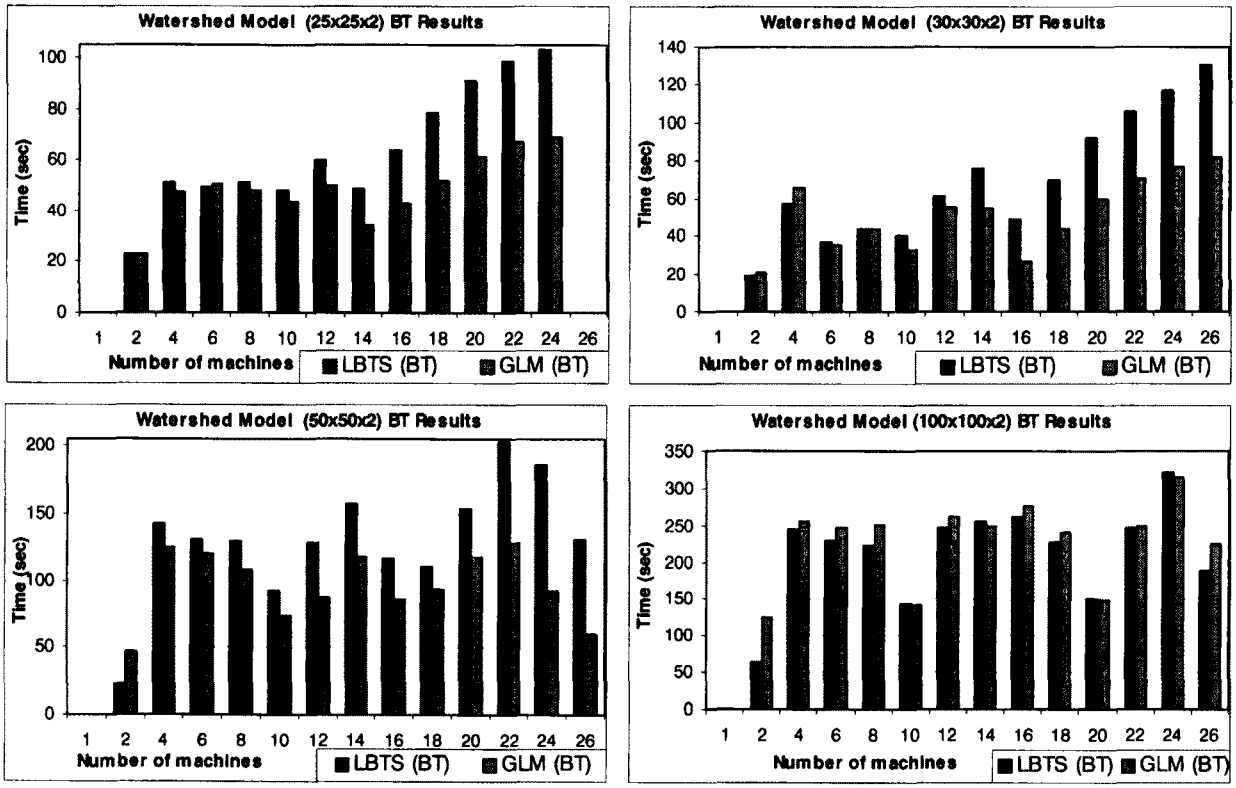


Figure 49. BT Results of *Watershed*

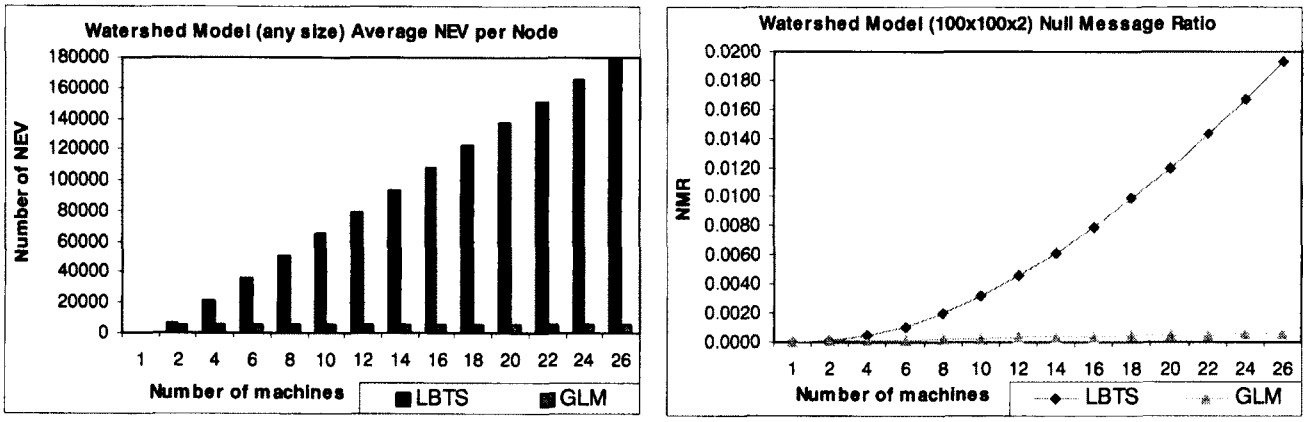


Figure 50. NMR Results of *Watershed*

The average number of null messages per node and the NMR results for the *Watershed* model are presented in Figure 50. The NEV results were the same for the different sizes of the model since the simulation was run for the same virtual-time duration in every case. The NEV values were significantly higher with the LBTS mechanism showing how successful was the GLM protocol in reducing the total number of null messages. The NMR results showed this huge performance gain by comparing the null events to the positive events and calculating the ratio.

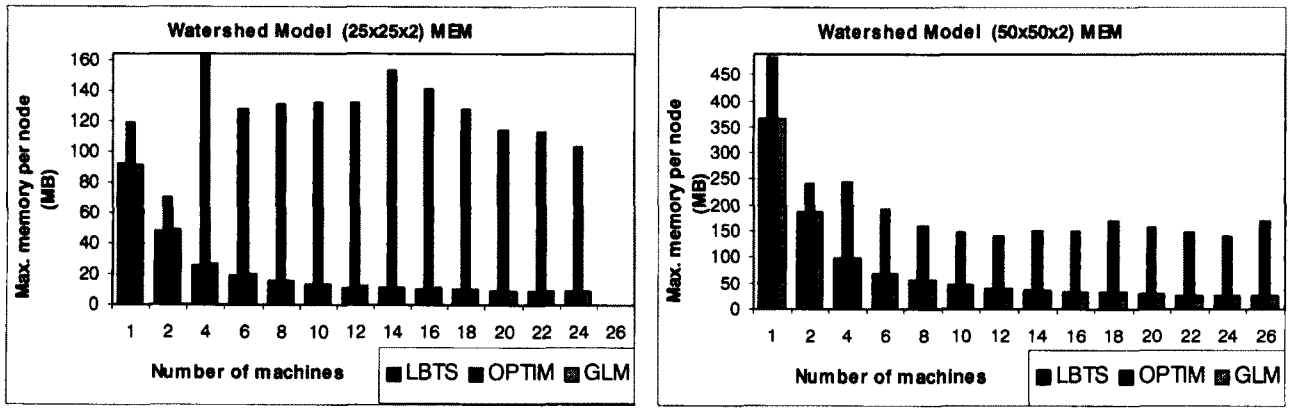


Figure 51. Memory consumption for *Watershed* model

Considering the maximum memory consumption, Figure 51 illustrates that average memory used per node decreased as the number of nodes increased. However this was only true for the conservative protocols. The optimist protocol showed fluctuating results at smaller sizes of the model which is very dependent to how the model is partitioned, which may lead to larger rollbacks and state saving overheads. When the largest size of the model was tested, the given partitioning mechanism was well

suited for the optimistic protocol resulting in memory reduction as the number of participating nodes increased.

6.4 Evaluation of the Conservative Protocols

This section evaluates the performance of the LBTS, CMB, and GLM protocols by presenting the results obtained from running the *Fire1*, *Watershed*, and *Synth* models under different scenarios. The protocols are analyzed in terms of the total execution time, maximum memory consumption, total number of null messages, and the null message ratio.

Figure 52 illustrates the T and BT results for *Fire1* model. The LBTS and GLM protocols reduced the execution time when more nodes were participating. However, this was only true until a certain point, where after that adding more nodes did not reduce the execution time. This is due to the overhead of the protocol, where the increased number of null messages and blocked times started to have a negative impact on the overall performance. In terms of the BT, GLM produced the smallest results in all cases, while CMB resulted in the largest blocked time values. Although CMB produces less null messages per synchronization phase, but it leads to larger total number of null messages and blocked periods compared to the LBTS protocol (because its strategy consists of multiple rounds of null message distribution).

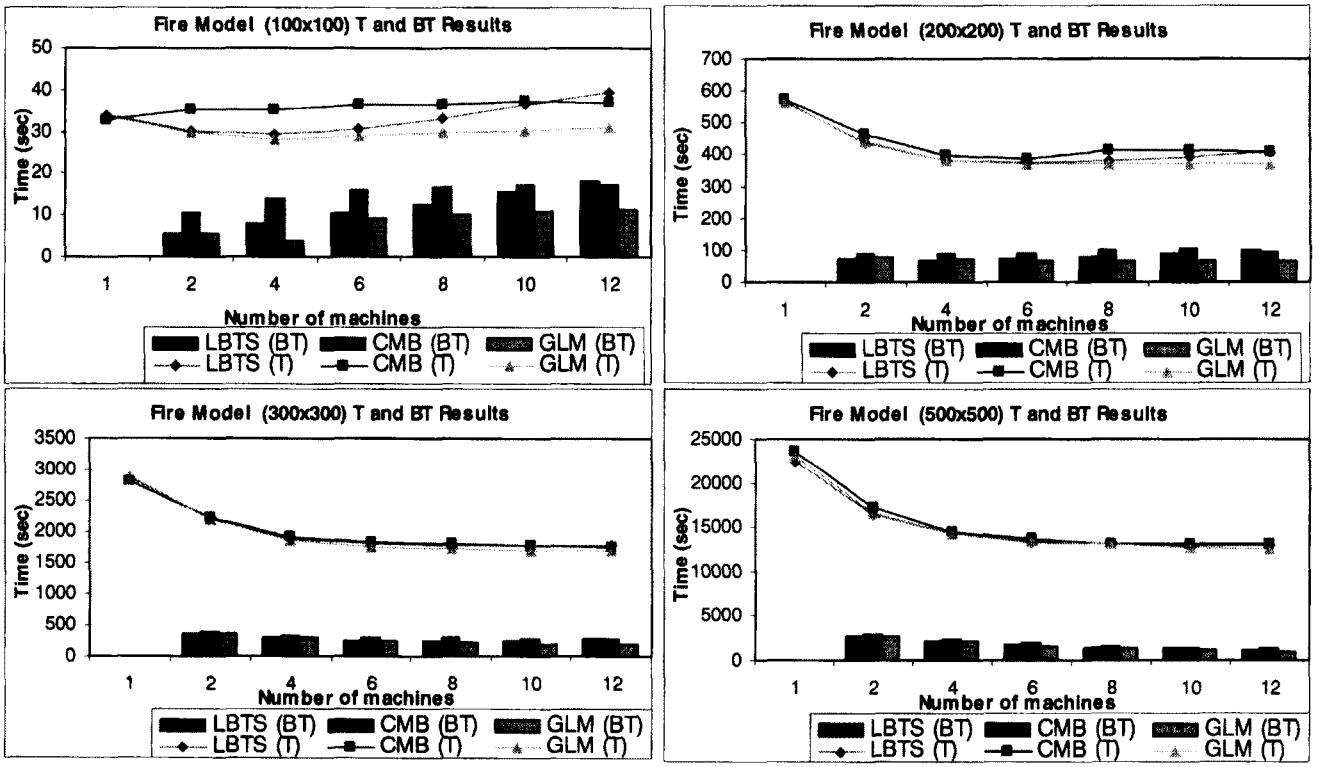


Figure 52. *Fire1* Model T and BT Results

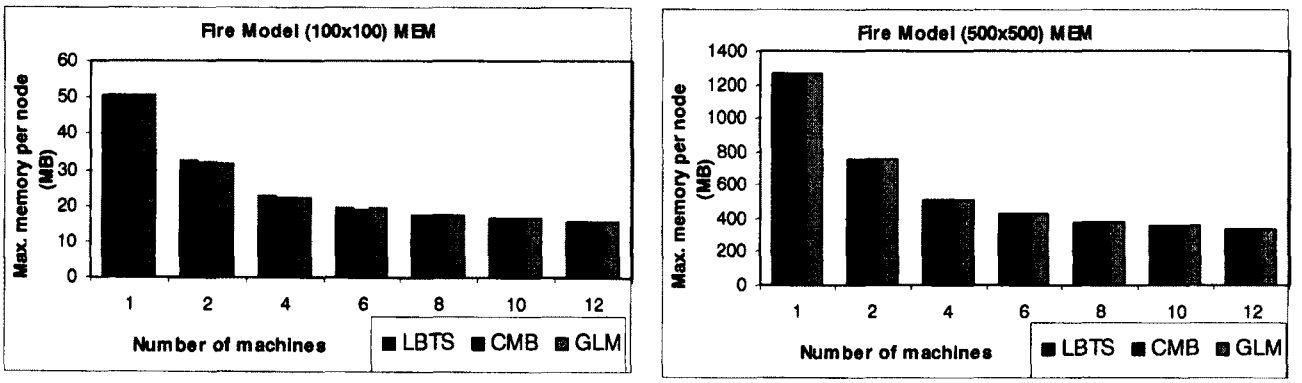


Figure 53. *Fire1* Model Memory Consumption Results

Memory consumption per node reduced at the same rate for different sizes as seen in Figure 53. The maximum memory consumption per node dropped considerably as more nodes were engaged, for all the three protocols.

The results of the *Watershed* model are given in Figure 54. Since the model is communication-intensive it was noticed that for all the protocols, the execution time reduced as more nodes were

engaged. The performance improved even with smaller size models (compared to *Fire1*). The GLM protocol provided the best performance in all cases, and the worst performance belonged to the CMB protocol. In most cases, only the BT of the CMB protocol was even larger than the T value of GLM and LBTS protocols. In all cases, with 2 nodes participating, the CMB protocol caused longer execution times compared to the results obtained from the sequential simulator. The large overhead of the CMB protocol overcomes the benefits of parallelization. However, as the number of processors increased, the execution time and the blocked period of CMB started to drop. For BT results, the tests showed that similar to the *Fire1* model, GLM produced the lowest blocked time; then the LBTS protocol, and finally the CMB mechanism. For the *Watershed* model, execution time and BT reduction rate for various sizes of the model were very close. The three protocols had the same performance gain regardless of the size of the model, which was due to the numerous events that were distributed throughout the simulation (this 3D model includes a large number of neighbors that must be updated more often).

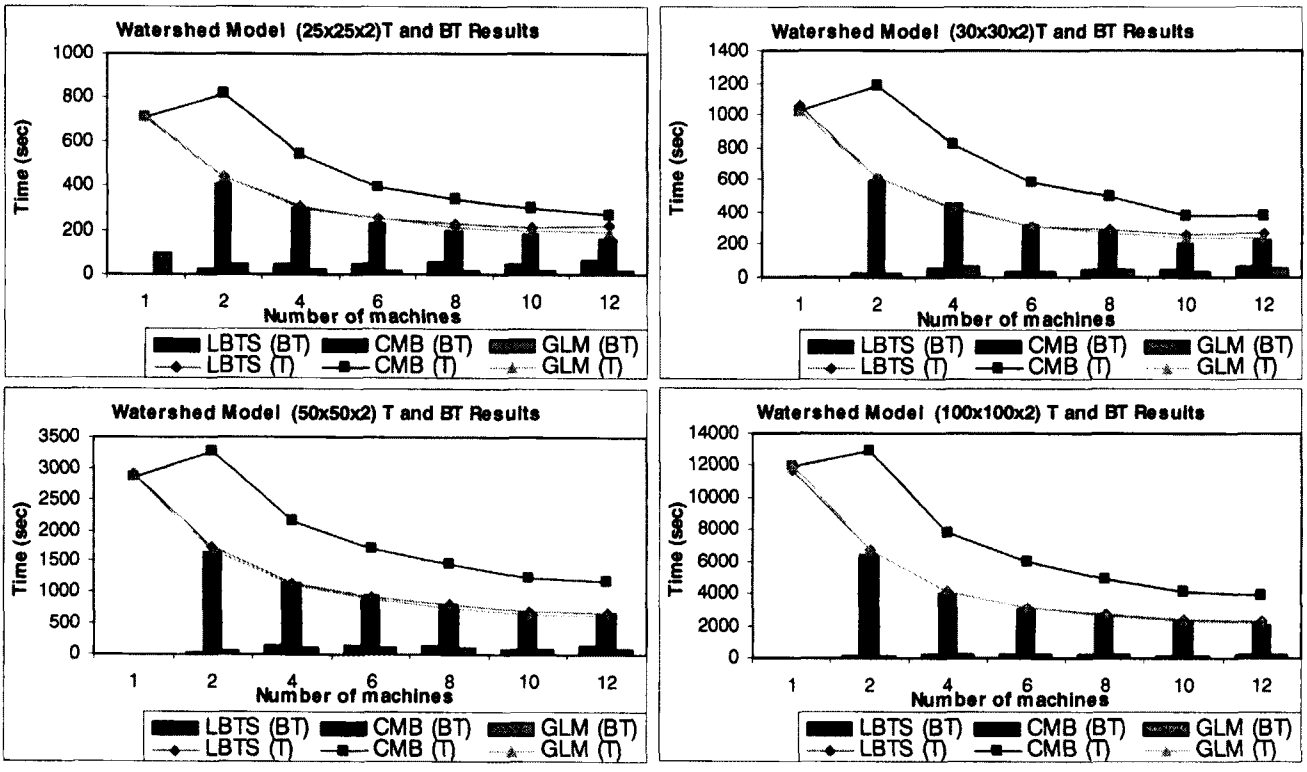


Figure 54. *Watershed* Model T and BT Results

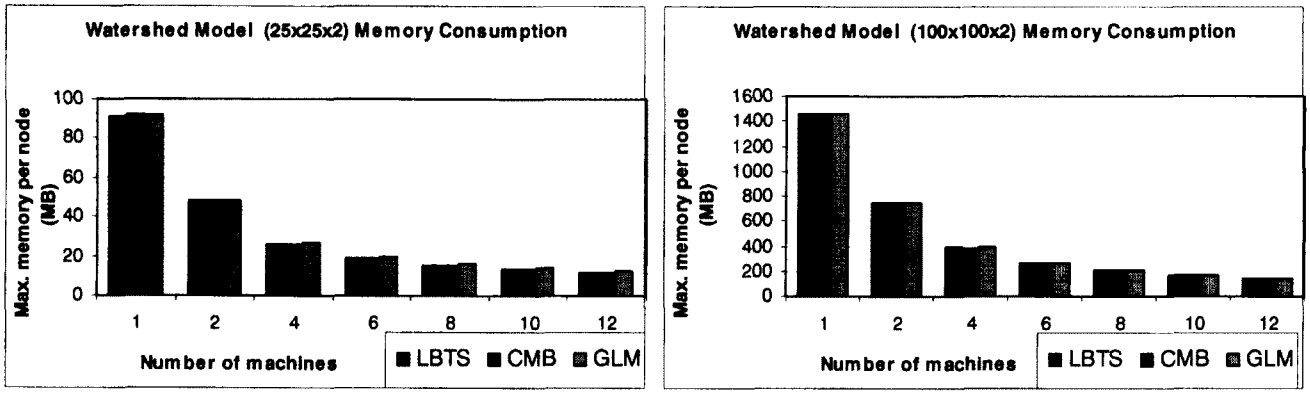


Figure 55. *Watershed* Model Memory Consumption Results

The MEM results are given in Figure 55. As in *Fire* model, memory consumption per node dropped as the number of machines increased. All the three protocols resulted in very similar MEM values, showing that the three protocols performed the same in terms of memory consumption.

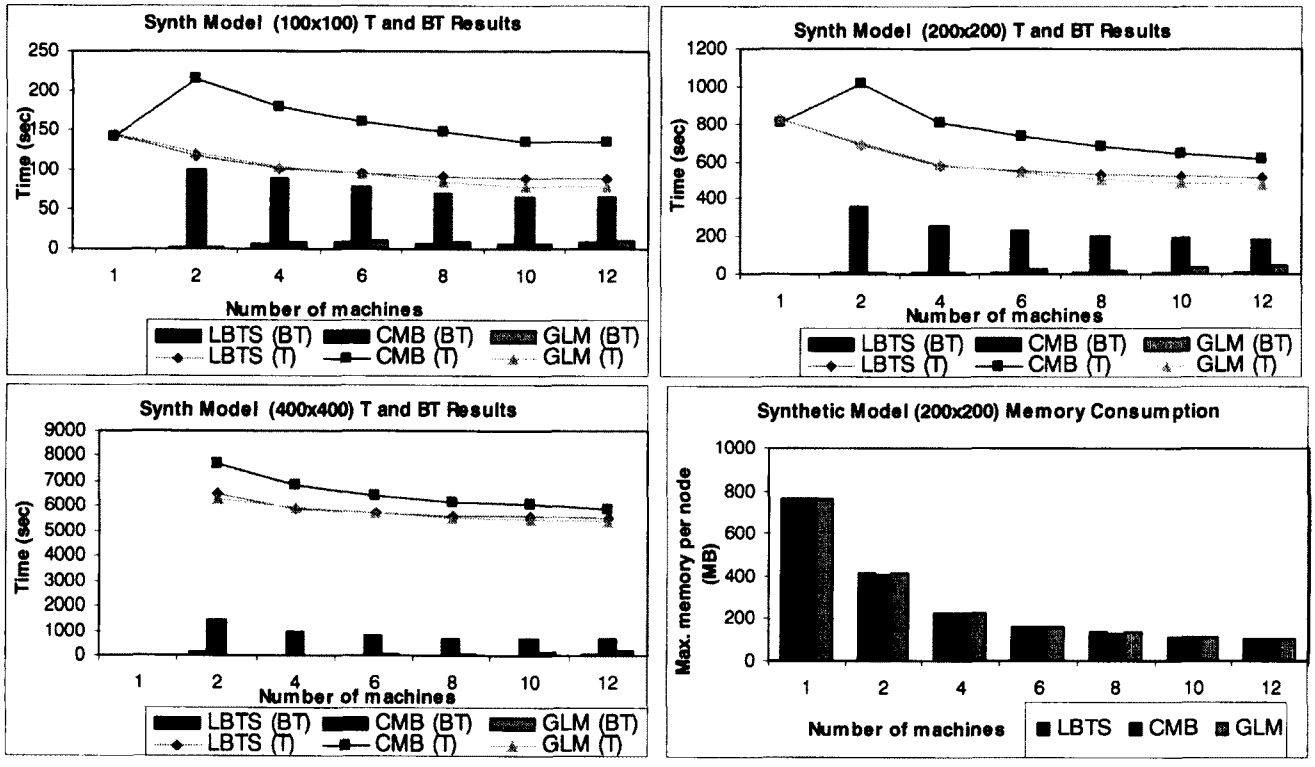


Figure 56. *Synth* Model Results

The T, BT, and MEM for the *Synth* model are shown in Figure 56. This model allows analyzing the performance of each of the protocols when full parallelism takes place. As can be observed from the

execution results, for all the protocols, the simulations benefited from the full parallelism such that the performance continued to improve as the number of nodes increased.

For GLM and LBTS, the BT value was considerably low compared to the T value in each case. The BT values were still too high with the CMB protocol compared to the other two protocols. As in previous models, the GLM resulted in best performance, while the CMB protocol had the worst results in every scenario. However, due to the nature of the model, overall the results were better than those obtained from the *Watershed* or *Fire* model executions. As shown by the memory consumption graph (for the 200x200 size) memory usage per node improved remarkably with the increase of the number of machines. All the three protocols reported very similar results for memory consumption.

The results of the three protocols in terms of the total number of null messages and the null message ratio were also collected. Figure 57 shows the NMR (i.e. NEV/PEV) results for various sizes of the *Fire1* model. Looking at the GLM graphs, it is shown that this protocol produced the smallest NMR at all cases.

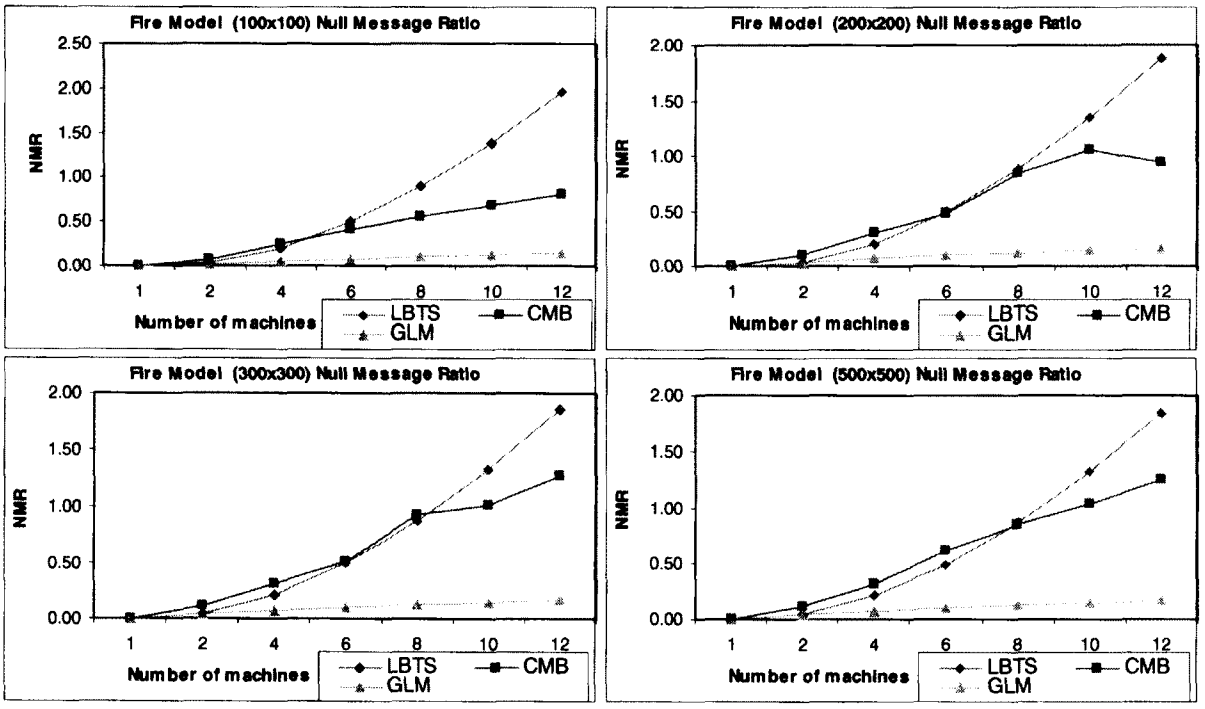


Figure 57. *Fire1* Model NMR Results

The CMB protocol, compared to the LBTS protocol, produced smaller NMR values after a certain number of participating nodes, which was 4, 6, 6, and 8 nodes for 100x100, 200x200, 300x300, and

500x500 sizes respectively. This behavior is explained by the fact that as the number of machines increased, the synchronization overhead associated with CMB got smaller than that of produced by the LBTS protocol. Meaning, with smaller number of machines, the total null messages produced by the LBTS protocol were less than the number of null message distribution rounds in CMB, thus resulting in lower NMR compared to the CMB protocol. On the other hand, when more nodes were participating, the total number of null messages that were distributed by the LBTS protocol were much higher than those produced by the CMB protocol, although the CMB protocol causes more synchronization rounds per each synchronization phase when more nodes are engaged.

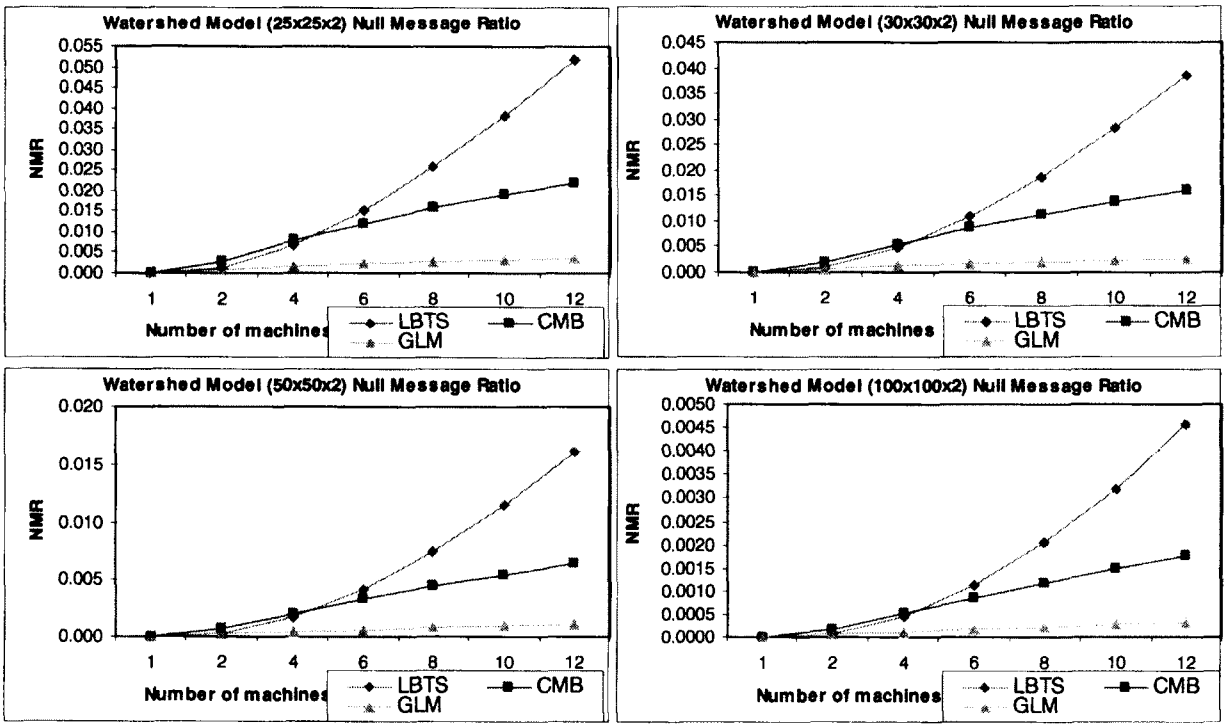


Figure 58. *Watershed* Model NMR Results

As expected, the GLM protocol resulted in the smallest number of null messages (average NEV per node) in all cases. Similar to the NMR results, the CMB outperformed the LBTS protocol after a certain point, while with smaller number of machines it led to the worse results compared to LBTS. The NMR results for the *Watershed* model are illustrated in Figure 58. Similar to *Fire1* model, the best results were obtained with GLM, and the CMB protocol outperformed the LBTS when more nodes were engaged.

Finally, the results for *Synth* model are presented in Figure 59. The performance of this model was similar to the *Watershed* model. The only performance difference between this model and the *Watershed* model is that much smaller NMR and NEV were produced, because the model was designed in such a way that at every step, all the cells change values and update their neighbors, which resulted in higher PEV values. On the other hand, the simulation was only conducted over 100 milliseconds (virtual time), which caused much fewer synchronization phases, thus, smaller NEV and NMR values were produced.

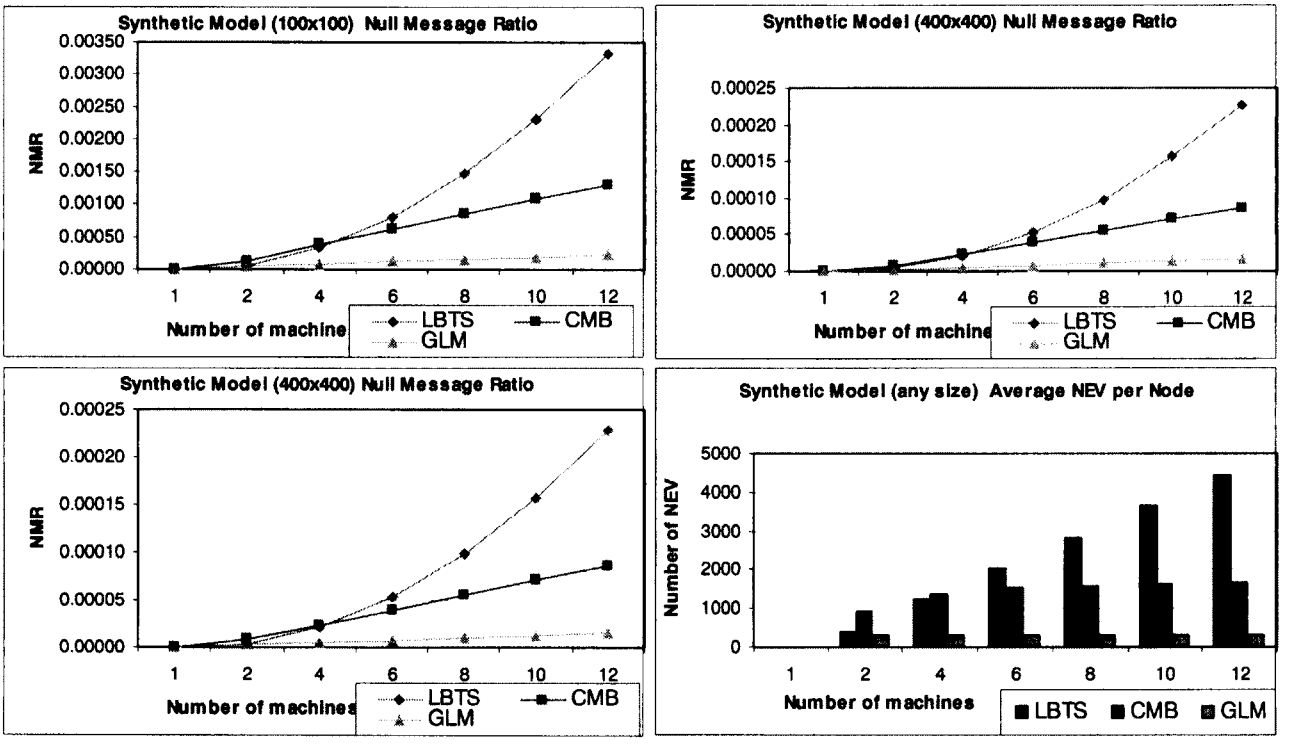


Figure 59. *Synth* Model NMR and NEV Results

6.5 Sensitivity Analysis of the Conservative Protocols

- **Initial-Load Distribution Analysis**

To study the effect of initial state values on the performance of the conservative protocols, the *Fire1* model was tested under three different scenarios for each given size as following:

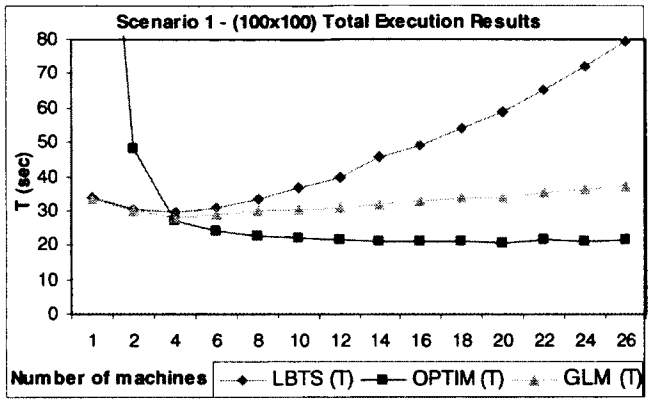
- **Scenario 1**: Placing an initial value at position (3, 78) of the cell space;
- **Scenario 2**: Placing an initial value at the middle of the cell space;

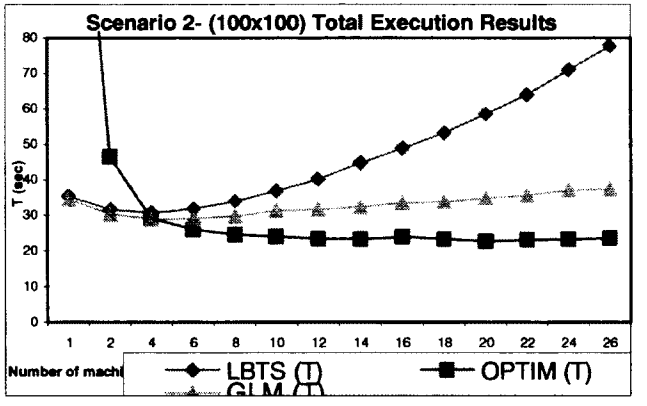
- **Scenario 3**: Placing an initial value at each partition of the cell space.

Figure 60 to Figure 62 illustrate the different simulation runs for *Fire1* model with three different sizes of cell space. Looking at the smallest model (i.e. 100x100 cells), it was observed that changing the initial states from scenario 1 to scenario 2 does not impact the performance of the protocols and very similar results were obtained. However, on the third scenario different behavior was noticed. The GLM protocol was affected significantly such that it outperformed the optimistic protocol for any number of nodes. While the results obtained from the LBTS protocol remained almost unchanged, the performance of the optimistic protocol degraded noticeably. This is because the simulation of scenario 3 resulted in much higher overall cells' activities, thus, the total number of state changes and rollbacks increased leading to higher execution times. In another words, since scenario 3 initialized an active cell on each partition at the initialization stage of the simulation, this caused more cells to be active at the beginning of the simulation, compared to the other two scenarios where only one cell was initialized at the beginning of the simulation on the entire cell space.

For the 300x300 cells case, only the optimistic protocol was affected. For instance, while under scenario 1, the optimistic protocol failed to run the simulation on 1, 2, 6, and 10 nodes, in scenario 2 the simulation could not be carried out even with 12 nodes. However, as more nodes were added the performance got better and the optimistic simulator reduced the execution time compared to scenario 1 experiments. The best performance for this size with the optimistic protocol was observed at scenario 3, reaching the execution times that were obtained by the GLM protocol as the number of participating nodes increased beyond 14. Compared to scenario 3 of the model at 100x100 cells, this scenario (i.e., scenario 3 at 300x300 cells) due to the large size of the model (i.e., 3 times larger) the simulation benefits from the parallel execution much larger than the drawbacks caused by numerous rollbacks and state savings. In another words, the advantages of parallel execution overcame the overhead of the synchronization protocol while this could not be achieved at smaller sizes of the model (e.g. 100x100).

Increasing the size of the model to 500x500 cells and running the simulations under the three different initial states scenarios was only possible for the conservative protocols. The optimistic simulator failed to run any simulation for the given size due to memory exhaustion. The three scenarios had very similar effect on the two conservative protocols while at most cases the best performance was achieved with the GLM protocol due to reasons explained in Section 6.3.





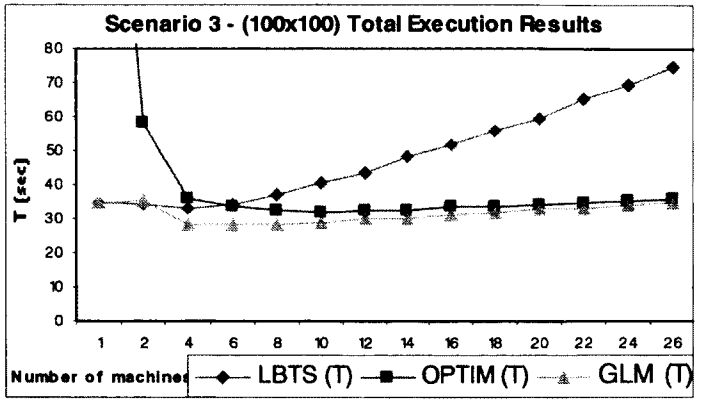
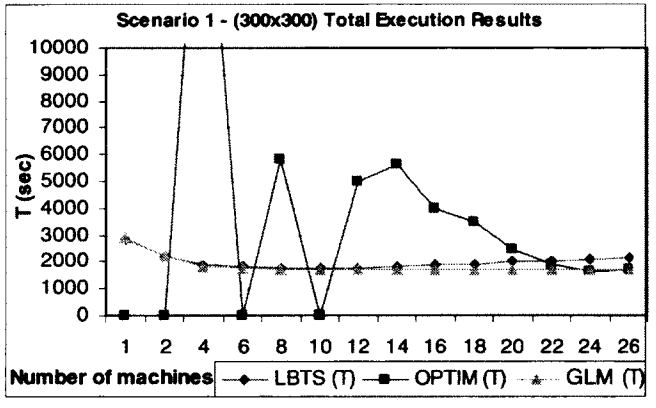
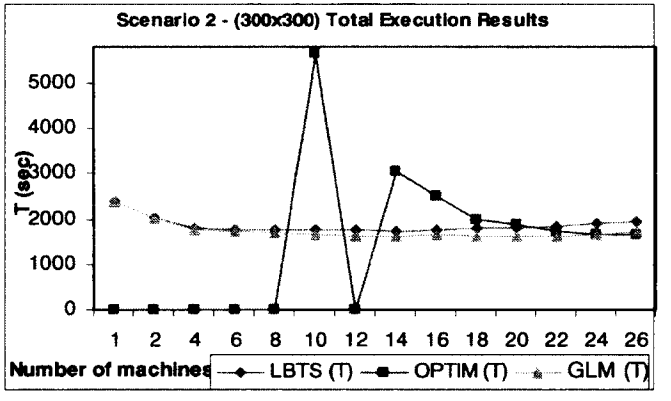


Figure 60. Initial States Analysis of *Fire1* Model (100x100)





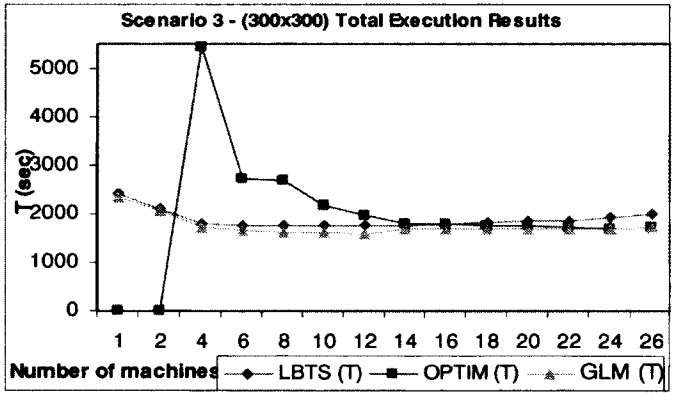
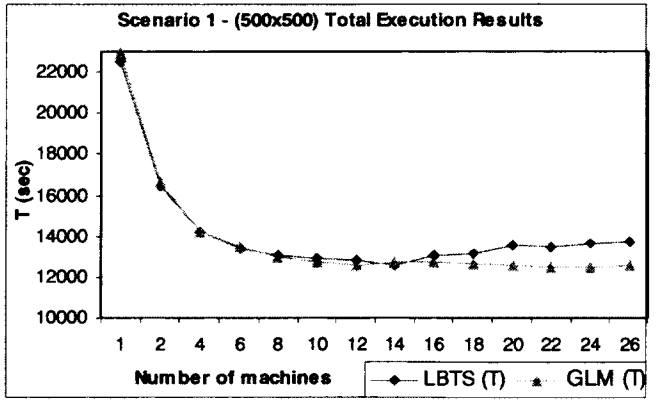
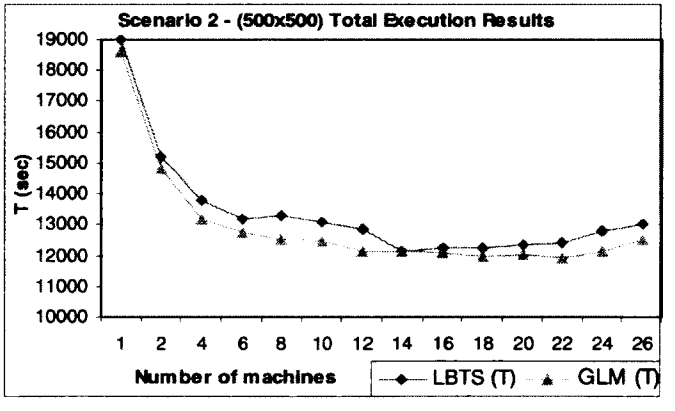


Figure 61. Initial States Analysis of *Fire1* Model (300x300)





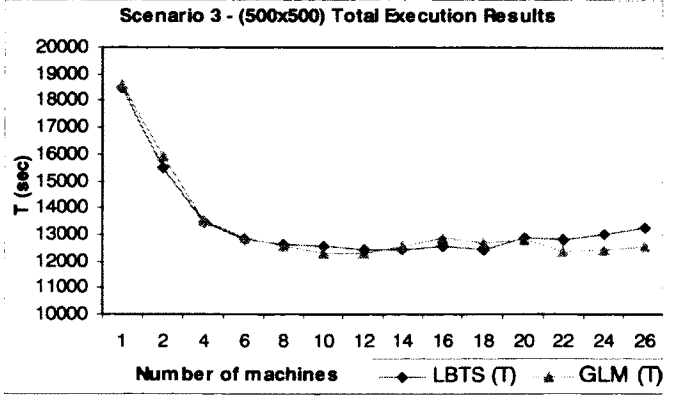


Figure 62. Initial States Analysis of *Fire1* Model (500x500)

- **Partitioning Analysis**

Different partitioning strategies may result in different simulation performance. Two partitioning mechanisms were used to analyze the effect on the overall execution performance of three of the models, *Fire1*, *Fire2*, and *Watershed*. The first partitioning strategy divided the cell space into equal horizontal rectangles, while the second strategy used vertical partitioning by dividing the cell space into even columns. These specific partitioning strategies were chosen because based on the nature of these three models and the behaviour of cells' values propagation, they result in different degree of parallelism especially at the initial stages of the simulation, allowing investigating how partitioning mechanisms affect the overall performance.

Figure 63 to Figure 65 illustrate the execution results obtained for different partitioning strategies for *Fire1* model with various sizes. The experiments with 100x100 cells were not affected by the partitioning strategies. Almost identical execution times were conducted for all the protocols. On the other hand, the experiments for the 300x300 size were slightly affected by modifying the partitioning mechanism. The impact was great on the optimistic protocol which resulted in much lower execution times with 14 nodes and beyond, when vertical partitioning was used. This showed that this specific partitioning mechanism improved the performance of the optimistic protocol by arranging the cells' neighboring in such a way that most of the neighbors of a cell were within the same partition, thus reducing the number of rollbacks and other associated optimistic synchronization overheads. For the largest size of the model (i.e., 500x500 cells), while the optimistic simulator was unable to run any experiment due to memory limitation, the two conservative protocols provided similar results in both cases of partitioning.

Similar behavior was observed for *Fire2* model. Figure 66 to Figure 68 present the results for this model with various sizes for each scenario. As in *Fire1*, the different partitioning mechanism yield similar results for the 100x100 and 500x500 sizes. Similarly, the results for 300x300 size showed that the experiments using the conservative protocols were not affected by altering the partitioning scheme mainly because independent of what partitioning is used, the conservative synchronization phases are invoked in the same manner and at the same frequency. However, the optimistic simulator showed better performance using the vertical partitioning strategy, thus, lower execution times were obtained.

Moreover, the simulation could be carried out with 8 and 10 nodes as well, while this was not possible with the horizontal partitioning scheme.

Finally, the experiments were also repeated for the *Watershed* model under various sizes (Figure 69 to Figure 71). Interesting results were observed showing that this model was not affected by altering the partitioning strategies for neither of the simulators (neither the conservative ones nor the optimistic version). This is due to the nature of the model which was a 3D cell space with initial values distributed all over the grid layers such that using different partitioning did not have any noticeable impact on the performance of the simulations. Meaning that, the overall distribution of the initial values and the neighboring of the cells followed very similar pattern on each partition when the model was divided vertically or horizontally.

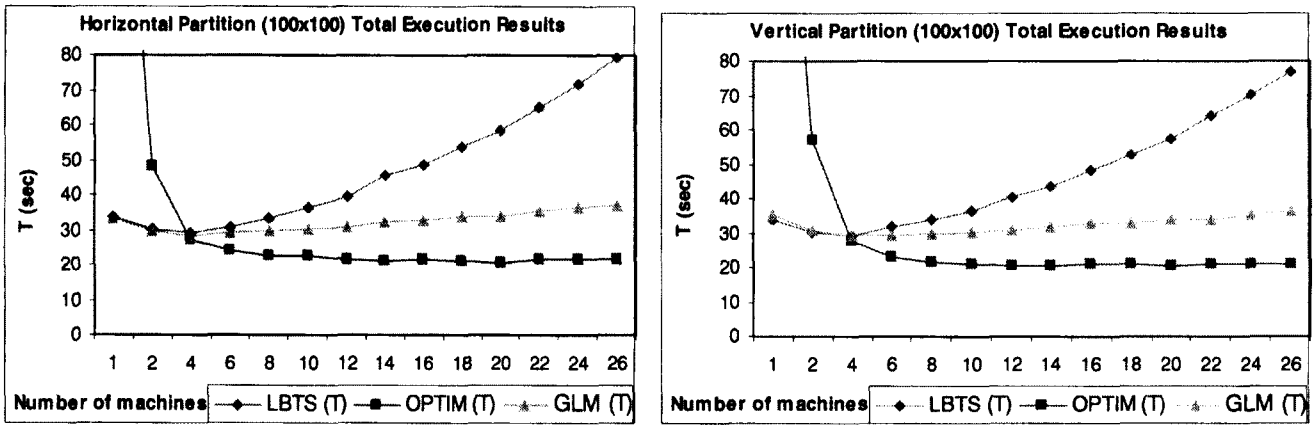


Figure 63. Partitioning Experiments of *Fire1* Model (100x100)

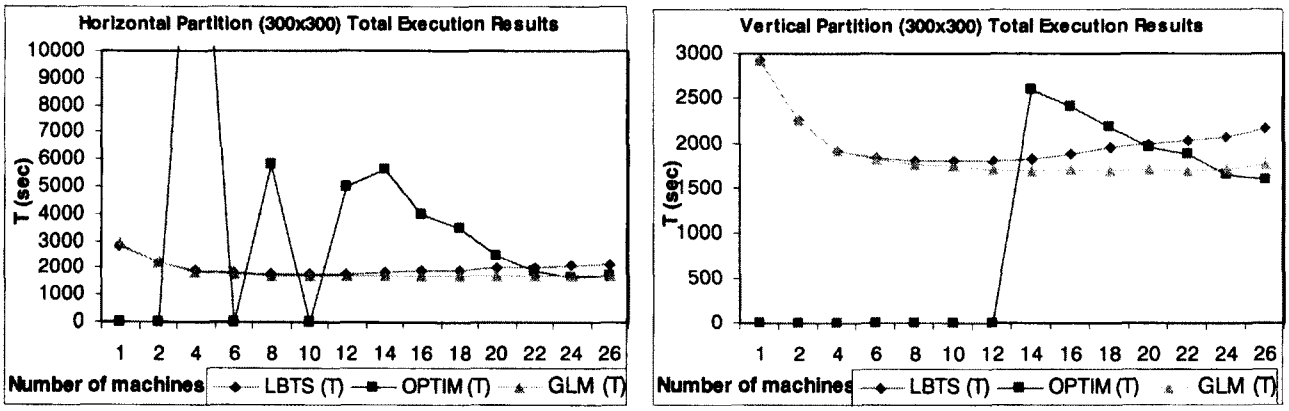


Figure 64. Partitioning Experiments of *Fire1* Model (300x300)

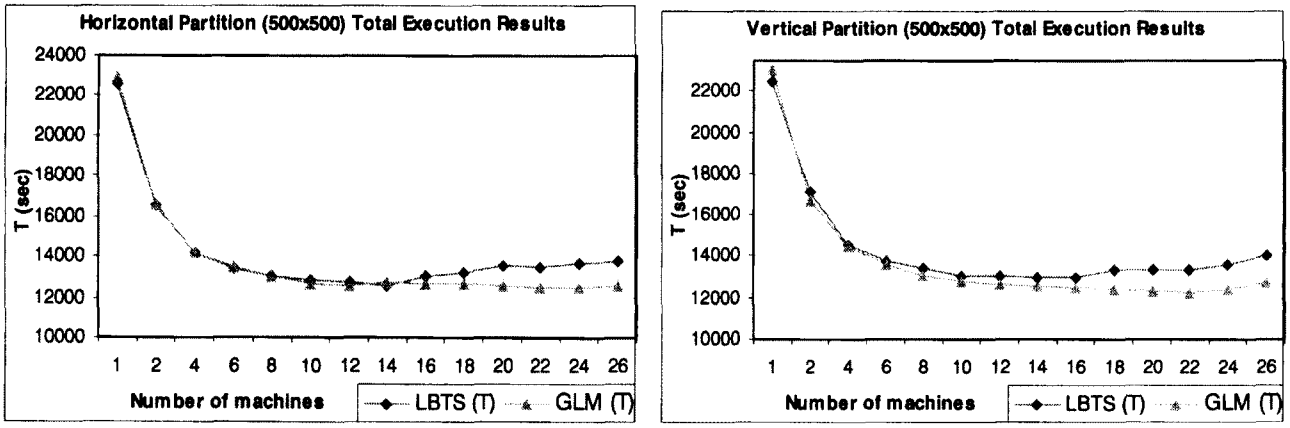


Figure 65. Partitioning Experiments of *Fire1* Model (500x500)

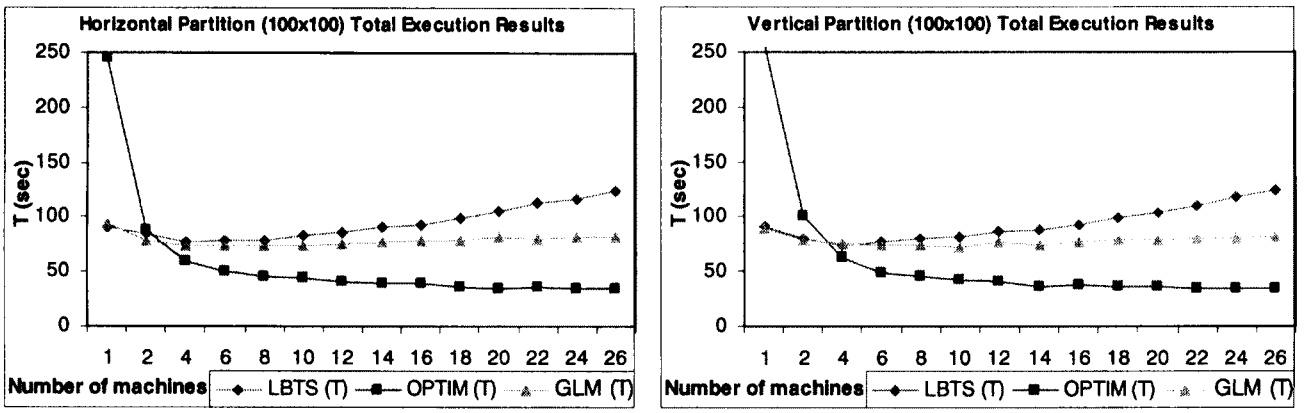


Figure 66. Partitioning Experiments of *Fire2* Model (100x100)

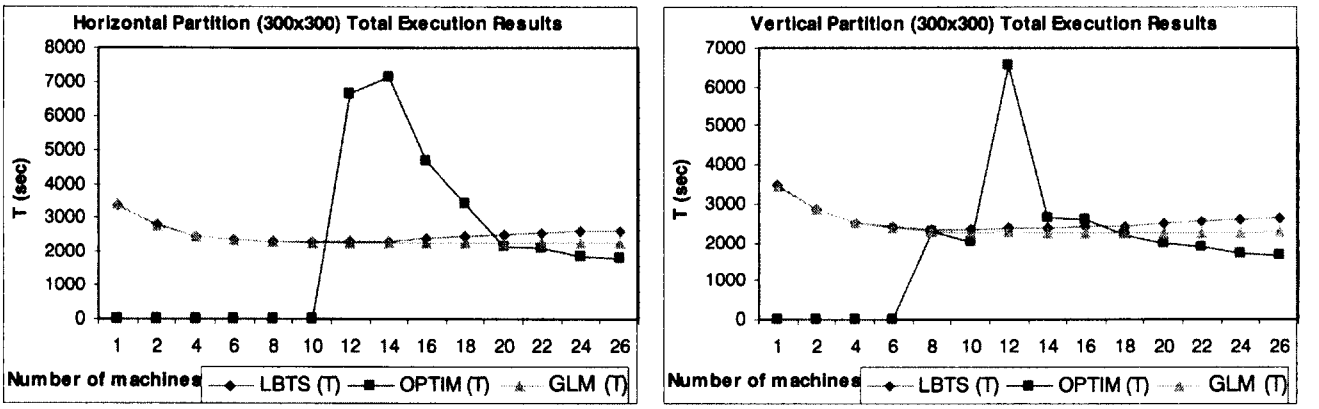


Figure 67. Partitioning Experiments of *Fire2* Model (300x300)

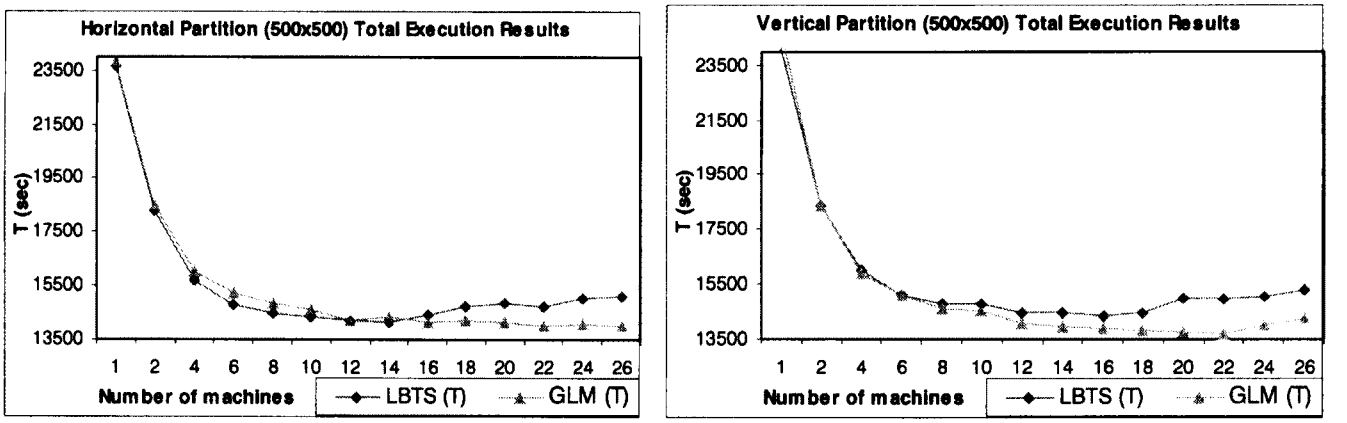


Figure 68. Partitioning Experiments of *Fire2* Model (500x500)

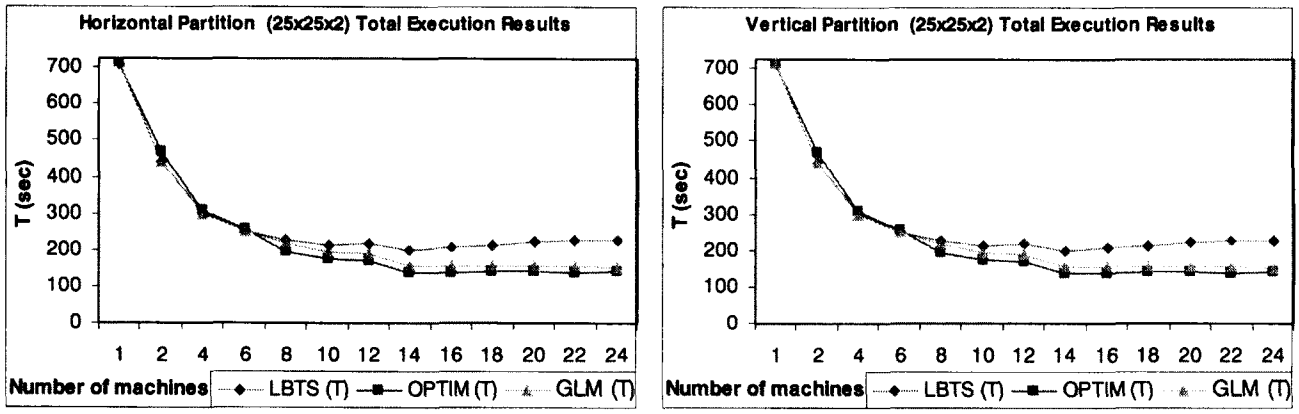


Figure 69. Partitioning Experiments of *Watershed* Model (25x25x2)

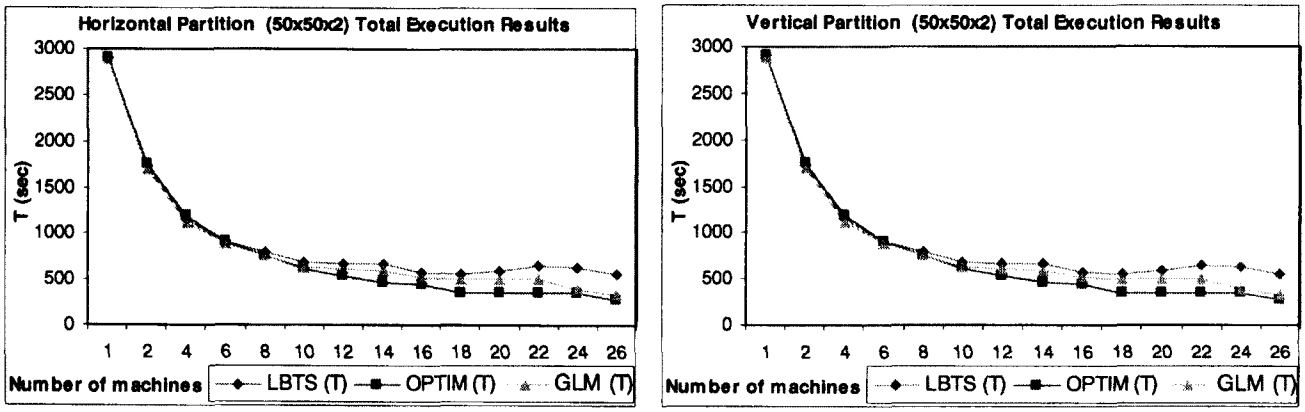


Figure 70. Partitioning Experiments of *Watershed* Model (50x50x2)

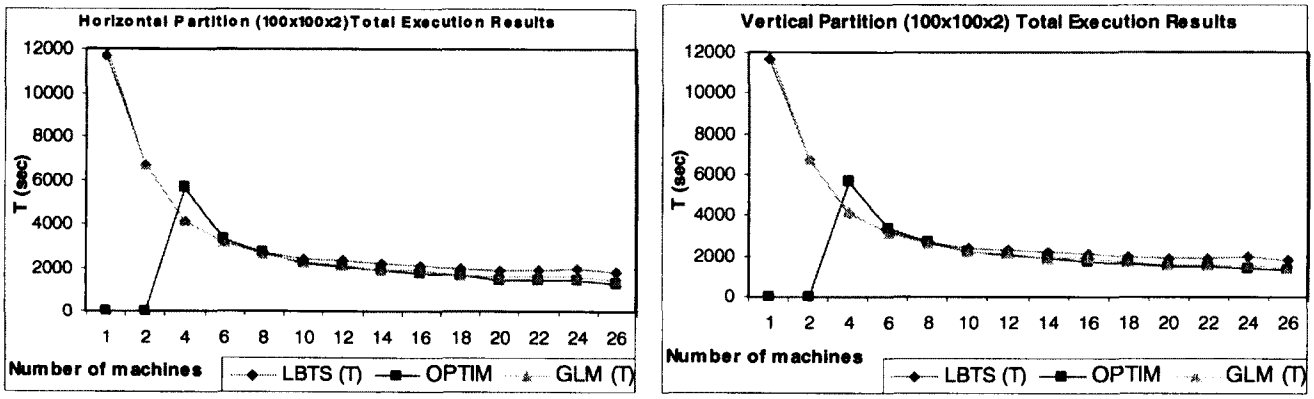


Figure 71. Partitioning Experiments of *Watershed* Model (100x100x2)

Chapter 7: Conclusion and Future Work

This dissertation addressed software development and performance issues that arise in large-scale parallel simulation of P-DEVS and Cell-DEVS models. In particular, this dissertation was primarily concerned with improving the performance of DEVS-based conservative simulation on distributed-memory multiprocessor clusters and performing comparative analyses versus optimistic simulations. To fulfill these objectives, three conservative DEVS protocols, **Lower-Bound-Time-Stamp (LBTS)**, **Chandy-Misra-Bryant (CMB)**, and **Global-Lookahead-Management (GLM)** have been proposed, and their effectiveness has been evaluated quantitatively in the CD++ environment using different benchmark models with varied characteristics.

The LBTS protocol [87], which serves as the base for the other two protocols, is the first purely conservative synchronization mechanism for running Cell-DEVS parallel simulations. The protocol attempts to reduce the communication overhead by implementing the mechanism at the NC (the highest level of the DEVS abstract simulator hierarchy). The two major challenges of conservative simulations i.e., the lookahead computation and null message distribution are handled by the NC at each LP. Thus, the NC is the central synchronizer on each node and forwards the simulation by issuing lookahead computation and LVT advancing phases. Communication among processes is performed merely through message distribution; there are no shared variables and no central process for message routing or scheduling. The LBTS protocol avoids cycles by implementing a deadlock-free mechanism which requires each LP to compute its lookahead value and to forward it to all participating LPs prior to blocking. When an LP receives all remote null messages, it resumes and calculates a new LVT based on the remote lookahead values it just received from other LPs. This scheme ensures that at any time, the LVT of every LP is equal to the Lower-Bound-Time-Stamp of any unprocessed event among all LPs. The major challenge of the LBTS protocol is the large number of null messages that must be distributed at the start of every synchronization phase.

The CMB protocol [83] overcomes the issues of large number of null message of the LBTS protocol by adopting the original Chandy-Misra-Bryant deadlock avoidance mechanism where null message are only sent among direct neighbors and in multiple rounds. Although each LP would be required to send

its lookahead value to less LPs, but the multiple rounds of null message distribution degrade the performance of the CMB protocol raising the need to for a protocol that manages such issues in a more efficient way.

The issue of large number of null messages of the two conservative DEVS protocols (i.e., LBTS and CMB) motivated proposing the GLM protocol [86]. The GLM mechanism proposed a phase-based simulation by introducing a central *Lookahead Manager* (LM) which is in charge of receiving every LP's lookahead, identifying the global minimum lookahead of the system, and broadcasting this value via null messages to all LPs. Under this scheme, LPs no longer distribute their null messages to each other, rather, they send their lookahead information directly to the LM. The LM responsibility is to detect the suspension phases, and to initiate the resume phases by broadcasting the globally computed minimum lookahead value. The asynchronous characteristic of the LM ensures that it is not a bottleneck, since the only message transmissions involving it take place at the end of the block and resume phases.

This research also provided a comparative study of conservative (LBTS, CMB, GLM) versus optimistic (LTW) DEVS-based parallel simulation by conducting thorough experiments and analyzing different sensitivity metrics [84][88]. Detailed discussions were presented investigating the performance efficiency of each of the protocols and the pros and cons of using each particular mechanism for different types of DEVS models.

7.1 Review of Key Contributions

The three conservative protocols presented in this research, namely LBTS, CMB, and GLM take proactive approaches to addressing the challenges of DEVS-based conservative simulations, improving performance without complicating the synchronization layer unnecessarily, sacrificing potential parallelism, or introducing a noticeable extra operational overhead. The key contributions of this dissertation are summarized as follows.

- Implemented *coordinator-centered* synchronization approach by implementing the conservative protocols at the NC level which led to a substantial reduction in the operational overhead resulting in a significant improvement in the overall simulation performance. By keeping other DEVS processors (i.e., FCs and child Simulators) unaware of the underlying synchronization

mechanism, the NC-centered strategy dropped the cost of null message distribution and lookahead computation significantly.

- Developed a simple *event scheduling* mechanism that maintains a FIFO input queue for each LP to simplify queue manipulation, thus reducing the associated queue management costs resulting in higher overall performance.
- Introduced *low-cost* and *efficient* lookahead and LVT computation strategies that make use of information that already exist in the simulation.
- Proposed *dynamic lookahead* computation mechanism that dynamically extracts the lookahead information from the model's specifications resulting in a *general-purpose, model-independent* parallel simulation.
- Implemented *deadlock-avoidance* strategy where LPs only block if they have already distributed their lookahead information, strictly avoiding deadlock cycles.
- Proposed various *null message distribution* mechanisms and studied their effect on the overall performance by implementing each of the conservative protocols with a different null message distribution strategy.
- Proposed *phase-based simulation* with GLM protocol by dividing the simulation into *parallel* and *broadcast* phase while maintaining a global synchronizer, namely Lookahead Manager to deal with organizing and issuing these phases.
- Performed a comparative study to evaluate optimistic TW-based simulation versus conservative approaches by conducting various tests on the optimistic simulator (PCD++ implementing the LTW protocol) and the conservative simulator (CCD++ implementing LBTS, CMB, and GLM protocol).
- Studied the conservative protocols under various metrics such as total number of null messages, total blocked time, and null message ratio.
- Investigated various sensitivity analyses both at model-level and protocol-level to address issues such as memory consumption and execution time for both conservative and optimistic simulators using various Cell-DEVS benchmark models.

7.2 Suggestions for Future Research

There are a number of interesting topics for future research on DEVS-based high performance parallel simulation with various extensions of the work presented in this dissertation. The following summarizes a list of issues that warrant further investigation in the context of the DEVS-based parallel simulation.

- Integration of the LBTS, CMB, and GLM protocols with other conservative optimization strategies to further improve the performance of P-DEVS and Cell-DEVS simulation on distributed memory multiprocessor cluster systems. This would include incorporating enhanced lookahead computations, reducing the number of null messages, and minimizing the overhead of null message distribution.
- Incorporation of hybrid synchronization mechanism by reducing the conservatism of the protocols and engaging some levels of optimism without introducing additional overhead into the protocol. For instance, investigating a hybrid protocol that allows only local rollbacks to limit the overhead of optimistic approaches.
- Incorporation of dynamic load balancing algorithms to support migration of LPS in DEVS-based simulations, taking advantage of the reduced overhead for transferring the LPs across cluster nodes. This would require investigating various mechanisms for dynamic creation and deletion of LPs to support runtime structural changes in conservative/optimistic DEVS systems.
- Performance evaluation using an extended set of models (both DEVS and Cell-DEVS) with different categories of sensitivity analysis such as: simulator configurations (e.g., event and state sizes, optimistic global time estimation and fossil collection frequency, checkpointing interval), and system parameters (e.g., size of available memory space, inter-node communication characteristics, and background load fluctuation). Besides, additional performance metrics could be collected in the experiments to evaluate other aspects of the simulation performance. For instance, analyzing the degree of parallelism by conducting the number of active nodes (or LPs) at each virtual time and the percentage of useful work performed by the LPs.

References

[1] Zeigler, B. P. "Theory of Modeling and Simulation". 1st Edition, New York: Wiley-Interscience, 1976.

[2] Zeigler, B. P. "Multifacetted Modelling and Discrete Event Simulation". Orlando: Academic Press, 1984.

[3] Zeigler, B. P. "Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems". Orlando: Academic Press, 1990.

[4] Zeigler, B. P., Zhang, G. "Mapping Hierarchical Discrete Event Models to Multiprocessor Systems: Concepts, Algorithm, and Simulation". Journal of Parallel and Distributed Computing, 9(3), pp. 271-281, 1990.

[5] Zeigler, B. P., Vahie, S. "DEVS Formalism and Methodology: Unity of Conception/Diversity of Application". Proceedings of the 1993 Winter Simulation Conference, Los Angeles, CA, pp. 573-579, 1993.

[6] Zeigler, B. P., Moon, Y., Kim, D., Kim, J. G. "DEVS-C++: A High Performance Modelling and Simulation Environment". Proceedings of the 29th Annual Hawaii International Conference on System Sciences, Maui, HI, pp. 350-359, 1996.

[7] Zeigler, B. P., Moon, Y. Kim, D., Ball, G. "The DEVS Environment for High-Performance Modeling and Simulation", IEEE Computational Science & Engineering, 4(3), pp. 61-71, 1997.

[8] Zeigler, B. P., Praehofer, H., Kim, T. G. "Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems". 2nd Edition, London: Academic Press, 2000.

[9] Zeigler, B. P., "DEVS Today: Recent Advances in Discrete Event-Based Information Technology". Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, Orlando, FL, pp. 148-161, 2003.

[10] Page, E., "Simulation Modeling Methodology: Principles and Etiology of Decision Support". PhD Dissertation, Virgina Polytechnic Institute and State University, 1994.

[11] Chow, A. C., Zeigler. B. "Parallel DEVS: A parallel, hierarchical, modular modeling formalism". Proceedings of the Winter Computer Simulation Conference, Orlando, FL. 1994.

[12] Wainer, G. "Discrete-Event Modeling and Simulation: a Practitioner's approach". CRC Press. Taylor and Francis. 2009.

[13] Cho, Y. K., Hu, X., Zeigler, B. "The RTDEVS/CORBA Environment for Simulation-Based Design of Distributed Real-Time Systems". SIMULATION, 79(4), pp. 197-210, 2003.

[14] Fujimoto, R. M. "Parallel and distributed simulation systems". New York: Wiley. 2000.

[15] Jefferson, D. R. "Virtual time". ACM Trans. Program. Lang. Syst. 7(3), pp. 404-425. 1985.

[16] Bryant, R. E. "Simulation of packet communication architecture computer systems". Massachusetts Institute of Technology. Cambridge, MA. USA. 1977.

[17] Chandy, K. M., Misra, J. "Distributed simulation: A case study in design and verification of distributed programs". IEEE Transactions on Software Engineering. pp.440-452. 1978.

[18] Wainer, G. "CD++: A toolkit to develop DEVS models". Software – Practice and Experience, 32:1261-1306. 2002.

[19] Troccoli, A., Wainer, G. "Implementing Parallel Cell-DEVS", Proceedings of the 36th Annual Simulation Symposium, Orlando, FL, pp. 273-280, 2003.
[20] Chidisiuc, C., Wainer, G. "CD++Builder: An Eclipse-based IDE for DEVS Modeling", Proceedings of the 2007 Spring Simulation Multiconference, Norfolk, VA, pp. 235-240, 2007.
[21] Liu, Q., Wainer, G. "Parallel Environment for DEVS and Cell-DEVS Models", SIMULATION, 83(6), pp. 449-471, 2007.
[22] Feng, B., Liu, Q., Wainer, G. "Parallel Simulation of DEVS and Cell-DEVS Models on Windows-based PC Cluster Systems", Proceedings of the 2008 Spring Simulation Multiconference: High Performance Computing Symposium, Ottawa, Canada, pp. 439-446, 2008.
[23] Harzallah, Y., Michel, V., Liu, Q., Wainer, G. "Distributed Simulation and Web Map Mash-Up for Forest Fire Spread", Proceedings of the 2008 IEEE Congress on Services – Part I, Honolulu, HI, pp. 176-183, 2008.
[24] Wainer, G., Liu, Q., Chazal, J., Quinet, L., Traore, M. K. "Performance Analysis of Web-based Distributed Simulation in DCD++: A Case Study across the Atlantic Ocean", Proceedings of the 2008 Spring Simulation Multiconference: High Performance Computing Symposium, Ottawa, Canada, pp. 413-420, 2008.
[25] Wainer, G., Madhoun, R., Al-Zoubi, K. "Distributed Simulation of DEVS and Cell-DEVS Models in CD++ using Web-Services", Simulation Modelling Practice and Theory, 16(9), pp. 1266-1292, 2008.
[26] Fujimoto, R. M. "Parallel Discrete Event Simulation", Communications of the ACM, 33(10), pp. 30-53, 1990.
[27] Christensen, E. R. "Hierarchical Optimistic Distributed Simulation: Combining DEVS and Time Warp". PhD Dissertation, University of Arizona, Tucson, AZ, 1990.
[28] Kim, K. H., Kim, T. G., Park, K. H. "Hierarchical Partitioning Algorithm for Optimistic Distributed Simulation of DEVS Models", Journal of Systems Architecture, 44(6-7), pp. 433-455, 1998.
[29] Nutaro, J., "Risk-Free Optimistic Simulation of DEVS Models", Proceedings of the 2004 Advanced Simulation Technologies Conference – Military, Government, and Aerospace Simulation, Arlington, VA, 2004.
[30] Nutaro, J., "On Constructing Optimistic Simulation Algorithms for the Discrete Event System Specification", ACM Transactions on Modeling and Computer Simulatiuon, 19(1), Article 1, 2008.
[31] Sun, Y., Nutaro, J. "Performance Improvement Using Parallel Simulation Protocol and Time Warp for DEVS Based Applications", Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, Vancouver, Canada, pp. 277-284, 2008.
[32] Liu, Q., Wainer, G. "A Performance Evaluation of the Light-weight Time Warp Protocol in Optimistic Parallel Simulation of DEVS-based Environmental Models", Proceedings of the 23rd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS), Lake Placid, New York, USA - June 2009.
[33] Wainer, G.; Giambiasi, N. "Specification, modeling and simulation of timed Cell-DEVS spaces". Technical Report n.: 98-007. Departamento de Computación. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. Argentina. 1998.
[34] Liu, Q., Wainer, G. "Lightweight Time Warp – A novel protocol for parallel optimistic simulation of large-scale DEVS and Cell-DEVS models". Proceedings of the 12th IEEE

International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2008), pp. 131-138, Vancouver, BC, Canada. 2008.

[35] Bain, W.L. Scott, D.S. "An Algorithm for Time Synchronization in Distributed Discrete Event Simulation." In Distributed Simulation: 30-33. Society for Computer Simulation. 1988.

[36] Misra, J. "Distributed Discrete-Event Simulation", Computing Surveys, Vol 18, No 1, pp.39-65. 1986.

[37] Peacock, J. K., Wong, J. W., Manning, E., "Synchronization of Distributed Simulation Using Broadcast Algorithms". Computer Networks, Vol.4, pp. 3-10, 1980.

[38] Nicol, D.M., Reynolds, P.F. "Problem oriented protocol design". Proceedings of the 1984 Winter Simulation Conference. Dec. 1984. 471-474.

[39] Su, W. K., Seitz, C. L."Variants of the Chandy-Misra-Bryant distributed discrete event simulation algorithm". Proceedings of the SCS Multiconference on Distributed Simulation. Vol. 21, 1989.

[40] Davis, N. J., Mannix, D. L., Shaw, W. H., Hartrum, T. C., "Distributed Discrete-Event Simulation Using Null Message Algorithms on Hypercube Architectures". Journal of Parallel and Distributed Computing, Vol. 8, No. 4, pp. 349-357, April 1990.

[41] Cai W., Turner, S.J. "An Algorithm for Distributed Discrete-Event Simulation - The `Carrier Null Message' Approach", Proceedings of the SCS Multiconference on Distributed Simulation, SCS, 1990, Vol. 22 (1), pp. 3-8.

[42] Wood, K. R., Turner, S. J., "A Generalized Carrier-Null Method for Conservative Parallel Simulation", Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS94), SCS, July 1994, pp. 50-57.

[43] Preiss, B. R., Loucks, W. M., MacIntyre, J. D., Field, J. A. "Null Message Cancellation in Conservative Distributed Simulation". Distributed Simulation 91 Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation, 1991.

[44] Dijkstra, E.W., Scholten, C.S. "Termination detection for diffusing computations". Inf. Proc. Lett. II 1 (August 1980), 1-4.

[45] Groselj, B., and Tropper, C. "A deadlock resolution scheme for distributed simulation". Proceedings of the SCS Multiconference on Distributed Simulation 21, 2, pp. 108-l 12. 1989.

[46] Chandy, K., and Misra, J. "Asynchronous Distributed Simulation Via a Sequence of Parallel Computations". Comm. of the ACM 24, 2, 198-205. 1981.

[47] Groselj, B., Tropper, C. "The time of next event algorithm". Proceedings of the 1988 SCS Multiconference on Distributed Simulation, 19(3):25–29, July 1988.

[48] Boukerche, A., Tropper, C. "SGTNE: Semi-Global Time of the Next Event Algorithm". Proceedings of 9th Workshop on Parallel and Distributed Simulation, Lake Placid, pp. 68-77. 1995.

[49] Venkatesh, K., Radhakrishnan, T., Li, H. F. "Discrete Event Simulation in a Distributed System". IEEE COMPSAC. IEEE Computer Society Press, 1986.

[50] Peacock, J.K., Wong, J.W., Manning, E. "Distributed simulation using a network of microcomputers". Computer Networks 3 , pp. 44–56. 1979.

[51] Concepcion, A. "Mapping Distributed Simulators onto the Hierarchical Multi-bus Multiprocessor Architecture". Distributed Simulation 1985, Jan. 24-26, 1985, pp. 8-13.

[52] Baik, D.K., Zeigler, B.P. "Performance evaluation of hierarchical distributed simulators". In: Proc. Winter Simulation Conference (1985).

[53] Chandy, K.M. Lamport, L. "Distributed snapshots: Determining global states of distributed systems". ACM Transactions on Computer Systems. 3(1):63–75. 1985.
[54] Ayani, R. "A parallel simulation scheme based on the distance between objects". Proceedings of the SCS Multiconference on Distributed Simulation 21, 2 (March 1989), pp. 113-118.
[55] Lubachevsky, B.D. "Efficient distributed event-driven simulations of multiple-loop networks". Commun. ACM 32, (January 1989), 111-123.
[56] Nicol, D. M., "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations", Journal of the ACM, 40(2), pp. 304-333, 1993.
[57] Nicol, D. M., "Noncommittal Barrier Synchronization", Parallel Computing, 21(4), pp. 529-549, 1995.
[58] Legedza, U., Weihl, W. E. "Reducing Synchronization Overhead in Parallel Simulation", Proceedings of the 10th International Workshop on Parallel and Distributed Simulation, Philadelphia, PA, pp. 86-95, 1996.
[59] Chandy, K. M., Misra, J. "Distributed Deadlock Detection", ACM Transactions on Computer Systems, 1(2), pp. 144-156, 1983.
[60] Lubachevsky, B. D. "Bounded lag distributed discrete event simulation".Proceedings of the 1988 SCS Multiconference on Distributed Simulation,19(3):183–191, July 1988.
[61] Chandy, K.M., Sherman, R. "The conditional event approach to distributed simulation". Proceedings of the SCS Multiconference on Distributed Simulation 21, 2 (March 1989), pp. 93-99.
[62] Nicol, D.M. "The cost of conservative synchronization in parallel discrete event simulations". Tech. Rep. 90-20, ICASE, June 1989.
[63] Sokol, L., Weissman, J. B., Mutchler, P. A. "MTW: an empirical performance study". Winter Simulation Conference 1991: 557-563.
[64] Ayani, R., Rajae, H. "Parallel Simulation Using Conservative Time Windows", Proceedings of the Winter Simulation Conference, pp. 709-717, 1992.
[65] Preiss, B. R. "The Yaddes Distributed Discrete Event Simulation Specification Language and Execution Environments". Proc. SCS Eastern Multiconf. — Distributed Simulation, Vol. 21, No. 2, Society for Computer Simulation. pp. 139-144. 1989.
[66] Steinman, J. S. "SPEEDES: A Unified Approach to Parallel Simulation. Proceedings of the 6th workshop on Parallel and Distributed Simulation". 1992. pp. 75-83.
[67] Radhakrishnan, R., Martin, D. E., Chetlur, M., Rao, D. M., Wilsey. P. A. "An object-oriented time warp simulation kernel". Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments, LNCS 1505, pp. 13-23. 1998.
[68] sIEEE std 1516.2-2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification. Institute of Electrical and Electronic Engineers, New York, NY, 2001.
[69] Steinman, J. "The WarpIV Simulation Kernel". Proceedings of the 2005 Workshop on Principles of Advanced and Distributed Simulation (PADS). 2005.
[70] Perumalla, K. S. "μsik - A Micro-Kernel for Parallel/Distributed Simulation Systems". Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation (PADS). 2005.
[71] Zeigler, B., Moon, Y., Kim, D., Kim, J. G. "DEVS-C++: A high performance modeling and simulation environment". The 29th Hawaii International Conference on System Sciences. 1996.

[72] Zeigler, B., Kim, D., Buckley, S. "Distributed supply chain simulation in a DEVS/CORBA execution environment". Proceedings of the 1999 Winter Simulation Conference. 1999.

[73] Kim, K., Kang, W. "CORBA-based, Multi-threaded Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-hierarchical One". International Conference on Computational Science and Its Applications (ICCSA).Assisi, Italy. 2004.

[74] Mittal, S., Risco-Martin, J. L., Zeigler, B. P. "DEVS/SOA: A Cross-Platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process", SIMULATION, 85(7), pp. 419-450, 2009.

[75] Cheon, S., Seo, C., Park, S., Zeigler, B. "Design and implementation of distributed DEVS simulation in a peer to peer network system". Advanced Simulation Technologies Conference – Design, Analysis, and Simulation of Distributed Systems Symposium. Arlington, USA. 2004.

[76] Zhang, M., Zeigler, B., Hammonds, P. "DEVS/RMI – An auto-adaptive and reconfigurable distributed simulation environment for engineering studies". DEVS Integrative M&S Symposium (DEVS'06). Huntsville, Alabama, USA. 2006.

[77] Kim, T.G., Park, S.B. "The DEVS formalism: Hierarchical modular systems specification in C++". Proceedings of European Simulation Multiconference. 1992.

[78] Seong, Y.R., Jung, S.H., Kim, T.G., Park, K.H. "Parallel simulation of hierarchical modular DEVS models: A modified Time Warp approach". Internat. J. Comput. Simulation 5 (3), 1995, pp.263-285.

[79] Praehofer, H., Reisinger, G. "Distributed Simulation of DEVS-based Multiformalism Models". AIS '94, Gainesville, FL, IEEE/CS Press, Dec. 1994, pp. 150-156.

[80] Nutaro, J. J. "On constructing optimistic simulation algorithms for the discrete event system specification". 2008. Transactions on Modeling and Computer Simulation.

[81] Sun, Y, Nutaro, J. "Performance Improvement Using Parallel Simulation Protocol and Time Warp for DEVS Based Applications". Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium. 2008. Page(s):277 – 284.

[82] Zeigler, B.P., Ball, G., Cho, H.J., Lee, J.S. "Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions", In Simulation Interoperation Workshop (SIW), number 99S-SIW-065, Orlando, FL, 1999.

[83] Jafer, S., Wainer, G. "Conservative Synchronization Methods for Parallel DEVS and Cell-DEVS". Proceedings of Summersim'11, Netherlands. 2011.

[84] Jafer, S., Wainer, G. "A Performance Evaluation of the Conservative DEVS Protocol in Parallel Simulation of DEVS-based Models ". Proceedings of Springsim'11, 2011.

[85] Moallemi, M., Jafer, S., Seyed, A., Wainer, G. "Interfacing DEVS and Visualization Models for Emergency Management". Proceedings of Springsim'11, 2011.

[86] Jafer, S., Wainer, G. "Global Lookahead Management (GLM) Protocol for Conservative DEVS Simulation". Proceedings of DS-RT 2010, Virginia, USA. 2010.

[87] Jafer, S., Wainer, G. "Conservative DEVS - A Novel Protocol for Parallel Conservative Simulation of DEVS and Cell-DEVS Models ", Proceedings of SpringSim'10, Orlando, USA. 2010.

[88] Jafer, S., Wainer, G. "Conservative vs. Optimistic Parallel Simulation of DEVS and Cell-DEVS: A Comparative Study ", SummerSim'10, Canada. 2010.

[89] Wainer, G., Liu, Q., Jafer, S. "Parallel Simulation of DEVS and Cell-DEVS models in CD++". In Discrete-Event Modeling and Simulation: Theory and Applications, Boca Raton, FL: CRC Press, pp. 226-272, 2010.

[90] Jafer, S., Wainer, G. "Flattened Conservative Parallel Simulator for DEVS and CELL-DEVS", Proceedings of International Conferences on Computational Science and Engineering, Vancouver, 2009.

[91] Sanz, V., Jafer, S., Wainer, G., Nicolescu, G., Urquia, A., Dormido, S. "Hybrid Modeling of OptoElectrical Interfaces Using DEVS and Modelica". Proceedings of the DEVS Integrative M&S Symposium, Springsim'09. San Diego, CA, USA. 2009.

[92] Jafer, S., Wainer, G. "Event Behavior of Discrete Event Simulations in CD++ Vs. NS-2". Poster Proceedings of Spring Simulation Multiconference, SpringSim, Ottawa, April 2008.

[93] Jafer, S., Wainer, G. "Advanced Parallel/Distributed Simulation Benchmark for Cellular Models". Poster Proceedings of AI/ GI/ CRV/ IS Annual Conference, Windsor, May 2008.

[94] Jafer, S., Wainer, G. "Synchronization Strategies for Parallel Simulation of Large-Scale DEVS-based Models ". Submitted to SIMULATION: Transactions of the Society for Modeling and Simulation International, 2011.

[95] Jafer, S., Liu, Q., Wainer, G. "Synchronization Methods in Parallel Discrete-Event Simulation ". Submitted to ACM Computing Surveys Journal, 2010.

[96] Praehofer, H., Zeigler, B. P. "On the Expressibility of Discrete Event Specified Systems", Proceedings of the 4th International Workshop on Computer Aided Systems Theory, LNCS 1105, Ottawa, Canada, pp. 65-79, 1994.

[97] Neumann, J. V., Burks, A. W. "Theory of Self-Reproducing Automata". Champaign: University of Illinois Press, 1966.

[98] Wainer, G., "Improved Cellular Models with Parallel Cell-DEVS", Transactions of the Society for Computer Simulation International, 17(2), pp. 73-88, 2000.

[99] Wainer, G., Giambiasi, N. "Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation", SIMULATION, 76(1), pp. 22-39, 2001.

[100] Wainer, G., Giambiasi, N. "N-dimensional Cell-DEVS Models", Discrete Event Dynamic Systems, 12(2), pp. 135-157, 2002.

[101] Wainer, G., "CD++: A Toolkit to Develop DEVS Models", Software – Practice and Experience, 32(13), pp. 1261-1306, 2002.

[102] Wainer, G., "Applying Cell-DEVS Methodology for Modeling the Environment", SIMULATION, 82(10), pp. 635-660. 2006.

[103] Lake, T., Zeigler, B.P., Sarjoughian, H.S., Nutaro, J. "DEVS Simulation and HLA Lookahead" In Simulation Interoperability Work-shop (SIW), number 00S-SIW-160,Orlando, FL, 2000.

[104] Zacharewicz, G., Giambiasi, N., Frydman, C. "Improving the DEVS/HLA Environment", In DEVS Integrative M&S Symposium, DEVS'05, Part of the 2005 SCS Spring Simulation Multiconference, SpringSim'05, San Diego, CA, USA, April 3-7 2005.

[105] Zacharewicz, G., Giambiasi, N., Frydman, C. "A New Algorithm for the HLA Lookahead Computing in the DEVS/HLA Environment", In European simulation Interoperability Workshop (EU-ROSIW),Toulouse, France, 2005.

[106] Zacharewicz, G., Giambiasi, N., Frydman, C. "Lookahead Computation in G-DEVS/HLA Environment". Simulation News Europe Journal (SNE) special issue 1 'Parallel and Distributed Simulation Methods and Environments' 16(2), 15–24. 2006.
[107] Zacharewicz, G., Giambiasi, N., Frydman, C. "G-DEVS/HLA Environment for Distributed Simulations of Workflows". SIMULATION, May 2008, Vol. 84, No. 5, 197-213.
[108] DeBenedictus, E., S Ghosh, M.-L- Yu. "A Novel Algorithm for Discrete Event Simulation". IEEE Computer, June 1991, pp. 21-33.
[109] Glinsky, E., Wainer. G. "New parallel simulation techniques of DEVS and Cell-DEVS in CD++". Proceedings of the 39th Annual Simulation Symposium, 2006, 244–251.
[110] Mattern, F. "Efficient algorithms for distributed snapshots and global virtual time approximation." Journal of Parallel and Distributed Comput. 18: 423–434. 1993.
[111] Martin, D. E., Wilsey, P. A., Hoekstra, R. J., Keiter, E. R., Hutchinson, S. A., Russo, T. V., Waters, L. J. "Redesigning the WARPED Simulation Kernel for Analysis and Application Development", Proceedings of the 36th Annual Simulation Symposium, Orlando, FL, pp. 216-223, 2003.
[112] Gropp, W., Lusk, E., Ashton, D., Balaji, P., Buntinas, D., Butler, R., Chan, A., Goodell, D., Krishna, J., Mercier, G., Ross, R., Thakur, R., Toonen, B. MPICH2 User's Guide, 2009, Available at:http://www.mcs.anl.gov/research/projects/mpich2/documentation/files/mpich2-1.2.userguide.pdf.
[113] Ameghino, J., Troccoli, A., Wainer, G. "Models of Complex Physical Systems Using Cell-DEVS". The 34th IEEE/SCS Annual Simulation Symposium. 2001.
[114] Rothermel, R. "A Mathematical Model for Predicting Fire Spread in Wild-land Fuels". Research Paper INT-115. Ogden, UT: U.S. Department of Agriculture, Forest Service, Intermountain Forest and Range Experiment Station. 1972.
[115] Bevins, C. D. "fireLib User Manual and Technical Reference". http://www.fire.org/, accessed in March 2010.
[116] Wainer, G. "Applying Cell-DEVS methodology for modeling the environment". SIMULATION 82(10), 2006, pp.635-660.
[117] Tropper, C. "Parallel Discrete-Event Simulation Applications". Journal of Parallel and Distributed Computing, 62(2), 327-335. 2002.
[118] Perumalla, K. S., Fujimoto, R. M. "Virtual Time Synchronization over Unreliable Network Transport". Proceedings of the 15th International Workshop on Parallel and Distributed Simulation, Lake Arrowhead, CA, 129-136. 2001.
[119] Perumalla, K. S. "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances". Proceedings of the 2006 Winter Simulation Conference, Monterey, CA, 84-95, 2006.
[120] Garg, R., Garg, V.K., Sabharwal, Y. "Scalable Algorithms for Global Snapshots in Distributed Systems". Proceedings of the 20th Ann. Int'l Conf. Supercomputing (ICS '06), 269-277. 2006.
[121] Garg, R., Garg, V.K., Sabharwal, Y. "Efficient Algorithms for Global Snapshots in Large Distributed Systems". IEEE Transactions on Parallel and Distributed Systems, vol. 21, no. 5, May 2010, 620-630. 2010.
[122] Lin, Y. B., Fishwick, P. A. "Asynchronous Parallel Discrete Event Simulation". IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, 26(4), 397-412. 1996.

[123] Nketsa, A., Khalifa, N. B. "Timed Petri Nets and Prediction to Improve the Chandy-Misra Conservative Distributed Simulation". Applied Mathematics and Computation, 120(1-3), 235-254. 2001.

[124] Porras, J., Hara, V., Jarju, J., and Ikonen, J. "Improving the Performance of the Chandy-Misra Parallel Simulation Algorithm in a Distributed Workstation Environment". Proceedings of the SCSC'97, 657-662. 1997.

[125] Xiao, Z., Unger, B., Simmonds, R., and Cleary, J. "Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation". Proceedings of the 13th International Workshop on Parallel and Distributed Simulation, Atlanta, GA, 20-28. 1999.

[126] Simmonds, R., Kiddle, C., and Unger, B. "Addressing blocking and scalability in critical channel traversing". Proceedings of the sixteenth workshop on Parallel and distributed simulation, May 12-15, 2002, Washington, D.C. 2002.

[127] Boukerche, A., and Das, S. "Reducing null messages overhead through load balancing in conservative distributed simulation systems". Journal of Parallel and Distributed Computing, 64(3):330-344. 2004.

[128] Rizvi, S., Elleithy, K.M., and Riasat, A. "Minimizing the Null Message Exchange in Conservative Distributed Simulation". International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering, CISSE, Bridgeport CT, 443-448. 2006.

[129] Rizvi, S., Elleithy, K.M., and Riasat, A. "A new mathematical model for optimizing the performance of parallel and discrete event simulation systems". Proceedings of the 2008 Spring Simulation Multiconference, Ottawa, Canada, 2008.

[130] Thomas, B., and Rizvi, S., and Elleithy, K.M. "Reducing Null Messages Using Grouping and Status Retrieval for a Conservative Discrete-Event Simulation System". Proceedings of the 2008 Spring Simulation Multiconference. 2008.

[131] Deelman, E., Bagrodia, R., Sakellariou, R., and Adve, V. "Improving Lookahead in Parallel Discrete Event Simulations of Large-scale Applications using Compiler Analysis". Proceedings of PADS'01, 5-13. 2001.

[132] Liu, J., Tan, K., and Nicol, D. "Lock-Free Scheduling of Logical Processes in Parallel Discrete-Event Simulation". Proceedings of PADS'01. 2001.

[133] V. Solcany, J. Safarik. "The Lookahead in a User-Transparent Conservative Parallel Simulator". Proceedings of PADS'02. 2002.

[134] Chung, M. K., and Kyung, C. M. "Improving Lookahead in Parallel Multiprocessor Simulation Using Dynamic Execution Path Prediction" Proceedings of PADS'06. 2006.

[135] Park, A., Fujimoto, R. M., and Perumalla, K. S. "Conservative Synchronization of Large-Scale Network Simulations". Proceedings of the 18th International Workshop on Parallel and Distributed Simulation, Kufstein, Austria, 153-161. 2004.

[136] Park, A., Fujimoto, R.M. "Aurora: An Approach to High Throughput Parallel Simulation". Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation. 2006.

[137] Park, A., Fujimoto, R.M. "A scalable framework for parallel discrete event simulations on desktop grids". The 8th IEEE/ACM International Conference on Grid Computing, 185–192. 2007.

[138] Park, A., Fujimoto, R.M. "Optimistic Parallel Simulation over Public Resource-Computing Infrastructures and Desktop Grids". Proceedings of 12th IEEE International Symposium on Distributed Simulation and Real Time Applications, Vancouver, BC, Canada. 2008.

[139] Park, A., Fujimoto, R.M. "Efficient Master/Worker Parallel Discrete Event Simulation". Proceedings of the 23rd Workshop on Principles of Advanced and Distributed Simulation. 2009.

[140] Bagrodia, R. L., and Takai, M. "Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages". IEEE Transactions on Parallel and Distributed Systems, 11(4), 395-411. 2000.

[141] Song, H.Y., Meyer, R.A., and Bagrodia, R. "An Empirical Study of conservative Scheduling". Proceedings of the 14th Workshop on Parallel and Distributed Simulation (PADS'00), Bologna, Italy. 2000.

[142] Kawabata. C., Santana, R., Santana, M., Bruschi, S., and Castelo Branco, K. "Performance evaluation of a CMB protocol". Proceedings of the Winter Simulation Conference (WSC'06), 1012-1019. 2006.

[143] Berry, O., and Jefferson, D. "Critical path analysis of distributed simulation". Proceedings of the SCS Conference on Distributed Simulation, 57-60. 1985.

[144] Jha, V., and Bagrodia, R. "A Performance Evaluation Methodology for Parallel Simulation Protocols". Proceedings of the 10th workshop on Parallel and distributed simulation (PADS'96), 180-185. 1996.

[145] Jha, V., and Bagrodia, R. "Simultaneous Events and Lookahead in Simulation Protocols". ACM Transactions on Modeling and Computer Simulation, 10(3), 241-267. 2000.

[146] Srinivasan, S., and Reynolds, P. "On Critical Path Analysis of Parallel Discrete Event Simulations". Technical Report No.CS-93-29. 1993.

[147] Srinivasan, S, and Reynolds P. "Elastic time". ACM Trans. Modeling Comput. Simulation 8(2), 103–139. 1998.

[148] Lin, S., Cheng, X., and Lv, J. "Micro-Synchronization in Conservative Parallel Network Simulation". Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation, 195-202. 2008.

[149] Lin, S., Cheng, X., and Lv, J. "State Causality Analysis of Conservative Parallel Network Simulation". Proceedings of 41st Annual Simulation Symposium. 2008.

[150] Fujimoto, R. M. "Parallel and Distributed Simulation Systems". Proceedings of the 2001 Winter Simulation Conference, Arlington, VA, 147-157. 2001.

[151] Fujimoto, R. M. "Distributed Simulation Systems". Proceedings of the 2003 Winter Simulation Conference, New Orleans, LA, 124-134. 2003.

[152] Mclean, T., And Fujimoto, R. M. "Predictable Time Management for Real-Time Distributed Simulation". Proceedings of the 17th International Workshop on Parallel and Distributed Simulation, San Diego, CA, 89-96. 2003.

[153] Curry, R., Kiddle, C., Simmonds, R., and Unger, B. "Sequential Performance of Asynchronous Conservative PDES Algorithms". Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation. 2005.

[154] Lemeire, J., and Dirkx, E. "Lookahead Accumulation in Conservative Parallel Discrete Event Simulation". Proceedings of the 18th European Simulation Multiconference, SCS Europe. 2004.

[155] Rajaei, H., Ayani, R., and Thorelli, L. E. "The Local Time Warp Approach to Parallel Simulation". ACM SIGSIM Simulation Digest, 23(1), 119-126. 1993.

[156] Rajaei, H. "Local Time Warp: An Implementation and Performance Analysis". Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, San Diego, CA, 163-170. 2007.

[157] Nicol, D., and Liu, J. "Composite Synchronization in Parallel Discrete-Event Simulation". IEEE Transactions on Parallel and Distributed Systems, vol. 13, No. 5, 433-446, May 2002.

[158] Meyer, R. A., and Bargrodia, R. L. "Path Lookahead: A Data Flow View of PDES Models". Proceedings of the 13th International Workshop on Parallel and Distributed Simulation, Atlanta, GA, 12-19. 1999.

[159] Boukerche, A. "Conservative Circuit Simulation on Multiprocessor Machines". Proceedings of the 7th International Conference on High Performance Computing, Bangalore, India, LNCS 1970, 415-424. 2000.

[160] Liu, J., and Nicol, D. M. "Lookahead Revisited in Wireless Network Simulations". Proceedings of the 16th International Workshop on Parallel and Distributed Simulation, Washington, D.C., 79-88. 2002.

[161] Jefferson, D. R., "Virtual Time II: Storage Management in Distributed Simulation", Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Canada, pp. 75-89, 1990.

[162] Groselj, B., and C. Tropper, "The Distributed Simulation of Clustered Processes", Distributed Computing, 4(3), pp. 111-121, 1991.

[163] Lin, Y. B., and Preiss, B. R. "Optimal Memory Management for Time Warp Parallel Simulation", ACM Transactions on Modeling and Computer Simulation, 1(4), pp. 283-307, 1991.

[164] Preiss, B. R., Loucks, W. M. "Memory Management Techniques for Time Warp on a Distributed Memory Machine", Proceedings of the 9th International Workshop on Parallel and Distributed Simulation, Lake Placid, NY, pp. 30-39, 1995.

[165] Young, C. H., Wilsey, P. A. "A Distributed Method to Bound Rollback Lengths for Fossil Collection in Time Warp Simulators", Information Processing Letters, 59(4), pp. 191-196, 1996.

[166] Young, C. H., Abu-Ghazaleh, N. B., Wilsey, P. A. "OFC: A Distributed Fossil Collection Algorithm for Time Warp", Proceedings of the 12th International Symposium on Distributed Computing, Andros, Greece, LNCS 1499, pp. 408-418, 1998.

[167] Chetlur, M., Wilsey, P. A. "Causality Information and Fossil Collection in Time Warp Simulations", Proceedings of the 2006 Winter Simulation Conference, Monterey, CA, pp. 987-994, 2006.

[168] Vee, V. Y., Hsu, W. J. "Pal: A New Fossil Collector for Time Warp", Proceedings of the 16th International Workshop on Parallel and Distributed Simulation, Washington, DC, pp. 35-42, 2002.

[169] Das, S. R., Fujimoto, R. M. "A Performance Study of the Cancelback Protocol for Time Warp", Proceedings of the 7th International Workshop on Parallel and Distributed Simulation, San Diego, CA, pp. 135-142, 1993.

[170] Akyildiz, I. F., Chen, L., Das, S. R., Fujimoto, R. M., Serfozo, R. F. "The Effect of Memory Capacity on Time Warp Performance", Journal of Parallel and Distributed Computing, 18(4), pp. 411-422, 1993.

[171] Lin, Y. B., "Memory Management Algorithms for Optimistic Parallel Simulation", Information Sciences, 77(1-2), pp. 119-140, 1994.

[172] Preiss, B. R., Loucks, W. M. "Memory Management Techniques for Time Warp on a Distributed Memory Machine", Proceedings of the 9th International Workshop on Parallel and Distributed Simulation, Lake Placid, NY, pp. 30-39, 1995.

[173] Lin, Y. B., Preiss, B. R., Loucks, W. M., and Lazowska, E. D. "Selecting the Checkpoint Interval in Time Warp Simulation", ACM SIGSIM Simulation Digest, 23(1), pp. 3-10, 1993.

[174] Preiss, B. R., Loucks, W. M., Macintyre, I. D. "Effects of the Checkpoint Interval on Time and Space in Time Warp", ACM Transactions on Modeling and Computer Simulation, 4(3), pp. 223-253, 1994.

[175] Ronngren, R., Ayani, R. "Adaptive Checkpointing in Time Warp", Proceedings of the 8th International Workshop on Parallel and Distributed Simulation, Edinburgh, UK, pp. 110-117, 1994.

[176] Fleischmann, J., and Wilsey, P. A. "Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulations", Proceedings of the 9th International Workshop on Parallel and Distributed Simulation, Lake Placid, NY, pp. 50-58, 1995.

[177] Skold, S., and Ronngren, R. "Event Sentitive State Saving in Time Warp Parallel Discrete Event Simulations", Proceedings of the 1996 Winter Simulation Conference, Coronado, CA, pp. 653-660, 1996.

[178] Quaglia, F., "Event History Based Sparse State Saving in Time Warp", ACM SIGSIM Simulation Digest, 28(1), pp. 72-79, 1998.

[179] Quaglia, F., "A Cost Model for Selecting Checkpoint Positions in Time Warp Parallel Simulation", IEEE Transactions on Parallel and Distributed Systems, 12(4), pp. 346-362, 2001.

[180] Bauer, H., and Sporrer, C. "Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving", Proceedings of the 26th Annual Simulation Symposium, Arlington, VA, pp. 12-20, 1993.

[181] West, D., and Panesar, K. "Automatic Incremental State Saving", ACM SIGSIM Simulation Digest, 26(1), pp. 78-85, 1996.

[182] Ronngren, R., Liljenstam, M., Ayani, R., and Montagnat, J. "Transparent Incremental State Saving in Time Warp Parallel Discrete Event Simulation", ACM SIGSIM Simulation Digest, 26(1), pp. 70-77, 1996.

[183] Feng, T. H., and Lee, E. A. "Incremental Checkpointing with Application to Distributed Discrete Event Simulation", Proceedings of the 2006 Winter Simulation Conference, Monterey, CA, pp. 1004-1011, 2006.

[184] Gomes, F., Unger, B., Cleary, J., and Franks, S. "Multiplexed State Saving for Bounded Rollback", Proceedings of the 1997 Winter Simulation Conference, Atlanta, GA, pp. 460-467, 1997.

[185] Tay, S. C., and Teo, Y. M. "Probabilistic Checkpointing in Time Warp Parallel Simulation", Proceedings of the 8th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, San Francisco, CA, pp. 366-373, 2000.

[186] Gafni, A., "Rollback Mechanisms for Optimistic Distributed Simulation Systems", Proceedings of the SCS Multiconference on Distributed Simulation, San Diego, CA, pp. 61-67, 1988.

[187] Lin, Y. B., and Lazowska, E. D. "A Study of Time Warp Rollback Mechanisms", ACM Transactions on Modeling and Computer Simulation, 1(1), pp. 51-72, 1991.

[188] Soliman, H. M., "Throttled Lazy Cancellation in Time Warp Parallel Simulation", SIMULATION, 84(2-3), pp. 149-160, 2008.

[189] Noronha, R., and Abu-Ghazaleh, N. B. "Early Cancellation: An Active NIC Optimization for Time-Warp", Proceedings of the 16th International Workshop on Parallel and Distributed Simulation, Washington, DC, pp. 43-50, 2002.

[190] Chetlur, M., and Wilsey, P. A. "Causality Representation and Cancellation Mechanisms in Time Warp Simulations", Proceedings of the 15th International Workshop on Parallel and Distributed Simulation, Lake Arrowhead, CA, pp. 165-172, 2001.

[191] Chetlur, M., and Wilsey, P. A. "Causality Information and Proactive Cancellation Mechanisms", Concurrency and Computation: Practice and Experience, 21(11), pp. 1483-2503, 2009.

[192] Zeng, Y., Cai, W., and Turner, S. J. "Batch Based Cancellation: A Rollback Optimal Cancellation Scheme in Time Warp Simulations", Proceedings of the 18th International Workshop on Principles of Advanced and Distributed Simulation, Kufstein, Austria, pp. 78-86, 2004.

[193] Himmelspach, J., Ewald, R., Leye, S., and Uhrmacher, A. M. "Parallel and Distributed Simulation of Parallel DEVS Models". Proceedings of the SpringSim '07, DEVS Integrative M&S Symposium, 249–256: SCS. 2007.

[194] Fujimoto, R.M. "Zero Lookahead and Repeatability in High Level Architecture". Proceedings of Spring Simulation Interoperability Workshop. 1997. Orlando, FL.

[195] Ntaimo, L., Zeigler, B. P., Vasconcelos, M. J., and Khargharia, B. "Forest Fire Spread and Suppression in DEVS", SIMULATION, 80(10), pp. 479-500, 2004.

[196] Hu, X., and Sun, Y. "Agent-Based Modeling and Simulation of Wildland Fire Suppression", Proceedings of the 2007 Winter Simulation Conference, San Diego, CA, pp. 1275-1283, 2007.

[197] Ntaimo, L., Hu, X., and Sun, Y. "DEVS-FIRE: Towards an Integrated Simulation Environment for Surface Wildfire Spread and Containment", SIMULATION, 84(4), pp. 137-155, 2008.

[198] Filippi, J. B., Morandini, F., Balbi, J. H., and Hill, D. "Discrete Event Front-Tracking Simulation of a Physical Fire-Spread Model", SIMULATION, 2009.

[199] Moon, Y., Zeigler, B. P., Ball, G., and Guertin, D. P., "DEVS Representation of Spatially Distributed Systems: Validity, Complexity Reduction". IEEE Transactions on Systems, Man and Cybernetics, pp. 288-296, 1996.

[200] Broutin, E., Paul, B., and Santucci, J. "Simulation of Heterogeneous DEVS Models: Application to the Study of Natural Systems", Proceedings of the 2009 Spring Simulation Multiconference, San Diego, CA, Article No. 148, 2009.

[201] Bagrodia, R., Meyer, R., Takai, M., Chen, Y., Zeng, X., Martin, J., and Song, H. Y. "Parsec: A parallel simulation environment for complex systems". Computer, 31(10):77–85, 1998.

[202] Weingartner, E., Lehn, H., and Wehrle, K. "A performance comparison of recent network simulators". In ICC 2009: IEEE International Conference on Communications, 2009

[203] Processes, N. O., Naroska, E., and Schwiegelshohn, U. "Conservative parallel simulation of a large number of processes". SIMULATION 72:3, 150-162. 1999.

[204] Y.-M. Teo, Y. M., and Tay, S. C. "Performance evaluation of a parallel simulation environment". In SS '99: Proceedings of the Thirty-Second Annual Simulation Symposium, page 86, Washington, DC, USA, 1999. IEEE Computer Society.