Towards Parallelization of Large-scale Ada Simulations Using Time Warp

Article · January 1998

CITATIONS
6

READS
183

3 authors:

Christopher D. Carothers
Rensselaer Polytechnic Institute
88 PUBLICATIONS **1,653** CITATIONS
SEE PROFILE

Maria Hybinette
University of Georgia
41 PUBLICATIONS **1,475** CITATIONS
SEE PROFILE

Richard M. Fujimoto
Georgia Institute of Technology
97 PUBLICATIONS **2,256** CITATIONS
SEE PROFILE

Towards Parallelization of Large-scale Ada Simulations Using Time Warp

Christopher D. Carothers
Maria I. Hybinette
Richard M. Fujimoto

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{*chrisc,ingrid,fujimoto*}@cc.gatech.edu

Abstract

Parallel simulation uses multiple processors to reduce the execution time of a single simulation model. This technology enables off-line decision aide simulations to be transformed into real-time decision aides for time critical situations without loss of model accuracy.

In this paper, we present a case study of our experiences in the parallelization of an existing large-scale Ada simulator called TISES. TISES stands for THAAD Integrated System Effectiveness Simulation and models all activities performed by a collection of THAAD missile batteries during an engagement scenario. TISES is approximately 300,000 lines of source code. Because of its enormous size and complexity, parallelization of the entire system was beyond the scope of our proof-of-concept effort. Instead, we selected a sub-component of the simulation that performs threat evaluation and weapon assignment.

Our approach for the parallelization of this sub-component makes use of an optimistic synchronization protocol, called Time Warp. Using our highly optimized Time Warp simulation executive, called Georgia Tech Time Warp (GTW) we obtain speedups on the order of 12 on 16 processors

Using the knowledge gained from this effort and our parallelization approach, we believe the entire TISES system could be parallelized using Time Warp, as well as other missile defense applications.

1. Introduction

Webster's Dictionary defines simulation as the act of pretending or creating a falsehood. In order to turn these "lies" into "truths", modelers must ensure that the simulation model correctly reproduces the behavior of the system being modeled. As systems become arbitrarily complex, so does the simulation model. The caveat here is that despite the continuing advances being made in processor speed, complex simulation models consistently require more CPU cycles than the state-of-the-art can provide. Today, many simulation applications require days or weeks of CPU time on the fastest uniprocessor machines.

The consequence of these long simulation execution times can have a significant impact on how these analysis tools are used. For example in the arena of ballistic missile defense, many of today's high-fidelity, system simulations can take hours to simulate a single threat scenario that on the real battlefield would have only taken ten minutes. Because these simulations run slower than real-time, they can only be used in the pre-planning phase of a potential conflict. When operating in this mode, the simulation performs "what-if" analysis for a battle that might be days or even weeks away. The output from this analysis is given to battle commanders in the form of "canned"scenarios. Then at the time of the battle, a commander will select which defense strategy best mitigates the current threat based on these potentially dated simulation "canned" scenarios. Ideally, a battle commander should have the ability formulate a defense strategy that is based on the most current threat information and have the simulations and "what-if" planning done in real-time in response the dynamic and chaotic changes of the battlefield.

In order to give battle commanders this capability, the critical issue becomes how do we enable these computationally intensive simulations to executed in real-time without loss of model fidelity or accuracy. One approach is to distribute the simulation computation across many computers or processors, thereby reducing the execution time.

As a technology proof-of-concept, we parallelized a portion of an existing large-scale battle management simulation system, called *TISES* (*THAAD Integrated System Effectiveness Simulation)*, a high-fidelity simulation of THAAD systems containing multiple missile batteries. Our approach to parallelization utilizes optimistic parallel simulation technology.

The primary technical objectives of this work were:

- to understand the technical issues in applying optimistic parallel simulation technology to a real-world battle management simulation system (BMC^{3}I in particular) and develop solutions,
- to develop a practical methodology to parallelizing an existing, large-scale Ada sequential simulation,
- to demonstrate the technical feasibility of this approach by implementing a scaled-down parallel version of the simulator, and
- to evaluate the increase in simulator performance that can be obtained by exploiting optimistic parallel simulation technology.

The remainder of this paper is organized as follows. In Section 2, we describe the TISES system followed by a detailed discussion of optimistic parallel simulation in Section 3. Next, our approach to parallelization is presented in Section 4, and in Section 5 our performance results are shown. Last, we summarize our findings and present our conclusions in Section 6.

2. Overview of TISES

TISES is the *THAAD Integrated System Effectiveness Simulation*. The objective for this system is to achieve and end-to-end system simulation of a THAAD multi-battery configuration. The output data produced by *TISES* for a particular scenario includes the following measures of effectiveness:

- system and timeliness,
- radar and loading performance,
- communication network performance,
- BMC^{3}I performance, and
- missile performance.

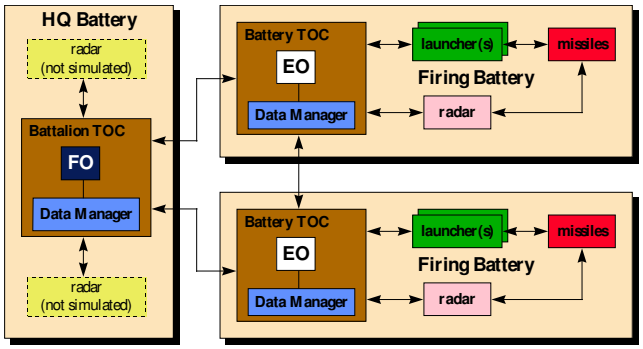

Figure 1: TISES Simulated THAAD Communications Architecture.

As depicted in Figure 1, a multi-battery THAAD system consists of a Battalion Level Tactical Operations Center (BTN-TOC) and a number of subordinate Battery Level Tactical Operations Centers (BTY-TOCs). Each BTN-TOC contains a number of co-located radars, which are not simulated by TISES, as well as communication links to each subordinate BTY-TOC. A BTY-TOC contains three to nine local launchers with up to fifteen missiles each, one local radar, and communication links back to the BTN-TOC plus communications links between BTY-TOCs.

2.1 Implementation

In terms of software implementation, TISES is a non real-time discrete event, engineering design-level simulation of the THAAD system. While not completely object-oriented because it is written in Ada, this system was designed to be highly modular. Figure 2 shows the functional decomposition of the TISES system. This system is decomposed into nine sub-components. Each sub-component is realized as an Ada-task. Controlling which task executes is the responsibility of the simulation Auto Driver. This task peers into the global event list and selects the event with the smallest time-stamp and removes it from the list. Next, the Auto Driver will examine which sub-component is responsible for processing this particular event type and proceeds to schedule the Ada-task modeling that sub-component, passing the event as an argument. When a sub-component task is given control, it invokes its generic event handler and based upon the event type determines the appropriate event processing procedure to invoke. This procedure will execute the necessary steps to complete the event. Should it be necessary, an event processing procedure may schedule new events into the future, which are posted to the global event list. Once the procedure is complete, the generic event handler will return control to the Auto Driver, which in-turn will select the next event to be processed.

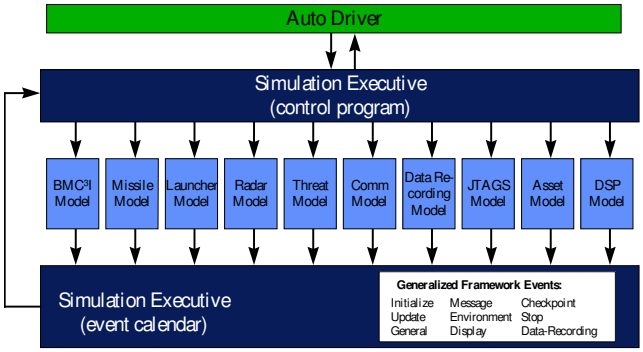

Figure 2: TISES Simulation Executive (sequential).

The TISES system, due to its high-fidelity, is a large system simulation consisting of about 300,000 line of code, mostly in Ada, however, some components are written C and Fortran. Because of the size of this simulation, parallelization of the entire TISES system was well beyond the scope of our proof-of-concept effort. Consequently, we narrowed our parallelization effort to the *Threat Evaluation & Weapon Assignment (TEWA)* module of BMC^{3}I sub-component. In the following sections, we will first provide a high-level

overview of the BMC^{3}I sub-component and then discuss the TEWA module.

2.2 BMC^{3}I Sub-component

The goal of the TISES BMC^{3}I sub-component is to accurately simulate the functionality associated with the THAAD tactical operations battle management, command, control, communications, and intelligence behavior. Simulation of TOC processes is the primary focus of this sub-component. These processes can be divided into the following five functional areas:

- *Force operations:* each battery TOC is sent a set of areas of responsibility and rules of engagement by the battalion TOC.
- *Sensor management:* generates and sends search/stop commands and search cue commands.
- *Track management:* receives object and discrimination reports as well as distributes track update reports.
- *Threat evaluation and weapons assignment:* allocates assets to targets, estimates asset damage, generates and selects engagements.
- *Engagement control:* monitors remote and local engagements.

2.3 TEWA Module

The Threat Evaluation and Weapon Assignment module utilizes sophisticated look-ahead planning techniques. This module consists of the following three major parts:

- *Threat Evaluation:* associates targets to assets and estimates asset damage using a greedy optimization strategy that is based upon a likelihood function.
- *Generate Engagements:* a set of candidate engagements is produced using an interpolation method that individually meet all timing and geometry constraints.
- *Select Engagements:* selects the set of coordinated engagements which satisfies the rules of engagement within the collective timing and resource constraints.

Once a threat has been evaluated and an acceptable engagement has been determined, the TOC will transmit a Launch Command message to the selected launcher and a Interceptor Support Plan message to the supporting radar. At the same time the Launch Command message is being sent and Engagement Status / Coordination Report (ESCR) messages are sent to each adjacent TOC to denote previously unreported changes in engagement status. These ESCR reports will play a pivotal role in determining the performance of the parallelized TEWA module. This aspect as will be discussed in future sections.

3. Optimistic Parallel Simulation

A critical issue that must be addressed when executing a discrete-event simulation program on parallel or distributed computers is ensuring that the program is properly synchronized. The distributed simulator consists of a collection of logical processes or LPs, each modeling a distinct component of the system, such as battalion or batter TOC, as shown in Figure 3. LPs communicate by exchanging time-stamped event messages, e.g., denoting a Launch Command or ESCR message. The synchronization mechanism must ensure that each LP processes events in time-stamp order and prevent events in the simulated future from affecting those in the past.

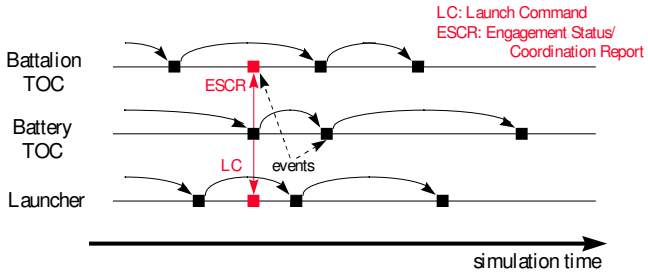

Figure 3: TISES Discrete Event Simulation Computation.

3.1 Time Warp

Time Warp is a well known synchronization protocol that detects out-of-order executions of events as they occur, and recovers using a rollback mechanism (Jefferson 1985). Time Warp is consider to be "optimistic" because it allows LPs to speculatively process events and only synchronizes LPs when events are processed out of time-stamp order. This protocol has demonstrated some success in speeding up a variety of simulation applications, including combat models (Wieland 1989), communication networks (Presley 1989), wireless networks (Carothers 1995), queuing networks (Fujimoto 1989), and digital logic circuits (Briner 1991), among others.

The Time Warp synchronization protocol can be divided into two parts: a local control mechanism and a global control mechanism, as show in Figure 4. The local control mechanism detects any out-of-order event computations. When such an error occurs at an LP, that LP must be "rolled back" to an event time that is just prior to the causality error. For example, as depicted in Figure 4, a Battalion TOC LP, Battery TOC LP and a launcher LP are each assigned to a separate processor. Consider the case where the Battalion TOC schedules a new Area of Responsibility (AOR) message in the "past" of the Battery TOC LP. How do we "rollback" the Battery TOC LP? To undo this incorrect event computation is a two step process. First, any changes made to the Battery TOC LP's state must be undone. This mechanism is supported by check-pointing the state of the LP prior to executing the event. Second, any future events that were scheduled as part of the incorrect event

computation are retracted. Now, if an event that is to be retracted has already been processed by its destination LP, that LP must be rolled back as well. Consequently, cascaded rollbacks are possible, thus making it imperative that incorrect event computations be erased as soon as possible before it spreads throughout the entire simulation.

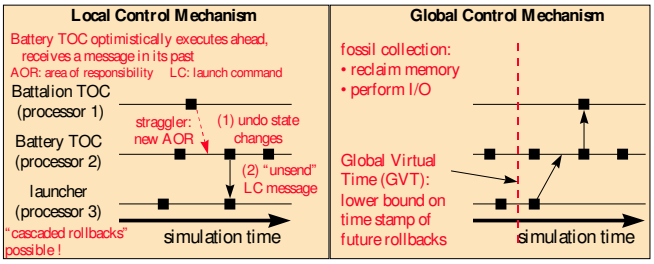

Figure 4: Time Warp Synchronization Mechanism.

Because changes made to an LP's state are recorded, a computer's memory would be exhausted shortly without some mechanism to reclaim "old" versions of state, and processed events. This action is performed by the global control mechanism. Here, a lower bound on the time-stamp of any future rollback is determined. This time is called *Global Virtual Time*. Any recorded state changes or processed events that are "older" than GVT may be reclaimed for future use. This process is called *fossil collection*. Additionally, irrevocable operations such as I/O to disks or a display can be performed when GVT sweeps past.

3.2 Georgia Tech Time Warp (GTW)

If one were to naively implement the Time Warp algorithm, very little if any increase in performance would be realized. For the past decade, researchers at Georgia Tech have been developing new algorithms and techniques that are necessary to make Time Warp a viable technology. This research has resulted in a highly optimized parallel simulation executive called *Georgia Tech Time Warp (GTW)*.

As new algorithms and techniques are developed, they are incorporated into GTW. Representative results from our research include:

- Analytic models that aide to improve our understanding of Time Warp performance (Gupta et al. 1991).
- Adaptive memory-based throttling (Das and Fujimoto 1994) and fast error correction that prevents rollback thrashing (Fujimoto 1989) .
- Fast GVT algorithm for shared memory multi-processors (Fujimoto and Hybinette, 1997).
- Fast, incremental *on-the-fly fossil collection* (Fujimoto and Hybinette, 1997).
- Dynamic load balancing on cluster computing environments (Carothers and Fujimoto 1996).

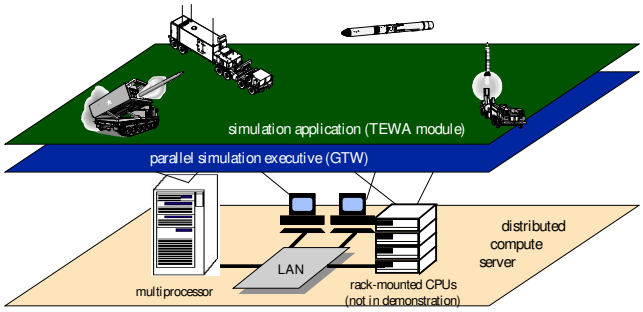

Figure 5: System Architecture.

The system architecture for GTW is shown in Figure 5. GTW provides an API that insulates the application from the underlying hardware platforms. Currently, GTW executes on a combination of shared-memory multiprocessors and heterogeneous workstations.

4. Parallelization Approach

When parallelizing an existing piece of software, there are two approaches one can take. The first approach is "do it all at once". Here, parallelization is a very high risk venture since parallel execution significantly increases software complexity. Moreover, when the software to be parallelized is already large and complex, as is the case for TISES, adding the additional complexity of parallel execution could make the task unmanageable or beyond the scope of available resources, as was the case for our proof-of-concept effort.

An alternative approach is to develop an incremental parallelization methodology. Here, there is a spiral software development process where piece by piece the system is parallelized. As a new piece is added to the parallelized version, it can be tested for correct functionality, thus a working parallel version of the system is always maintained.

The specific implementation of our incremental approach is shown in Figure 6. First, we extracted the TEWA module from the BMC^{3}I sub-component of TISES. However, in order to reproduce the behavior of the non-parallelized portions we built a "scaffolding" layer around the TEWA module.

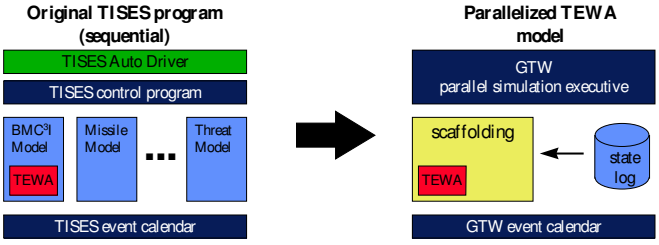

Figure 6: Incremental Parallelization Methodology.

To implement the scaffolding layer, we first instrument the BMC^{3}I sub-component to capture snapshots of the BMC^{3}I system state just prior to processing each TEWA event and write them to log files located on local disk. We then re-compile and execute the modified TISES system using the China-Taipei scenario, which was included with the software distribution. The TEWA/GTW system will then dynamically load the appropriate state snapshots from the log files prior to processing each TEWA event, giving the parallelized TEWA module the illusion that all of TISES is running.

In order to realize a parallelized TEWA module, we needed to determine how best to map TISES simulation objects to GTW's underlying simulation paradigm, which consists of LPs and events that travel between LPs. How this mapping is realized can be a critical factor in determining parallel system performance. The key lies in finding simulation objects that communicate or send messages infrequently, since more remote or "off-processor" messages increase the likelihood an LP will roll back. In the TEWA module, TOC objects were chosen to serve as LPs since they only communicate when a missile is launched. Recall, that after a missile launch, a TOC will inform all other TOCs by sending an ESCR message.

5. Performance Results

In order to assess the performance of any parallelized system, the sequential system execution time is divided by the parallelized version's execution time and a measure of speedup is obtained. However, in our case this approach would not provide an accurate picture of parallel performance. The problem is that the parallelized TEWA module contains the scaffolding which is not contained in the sequential version. This scaffolding perturbs the computational granularity of the TEWA module due to disk assesses and additional functionality that is required to rebuild the entire state of the BMC^{3}I sub-component prior to processing each TEWA event. Consequently, the scaffolding invalidates any performance measurements that can be obtained from the parallelized TEWA/GTW system.

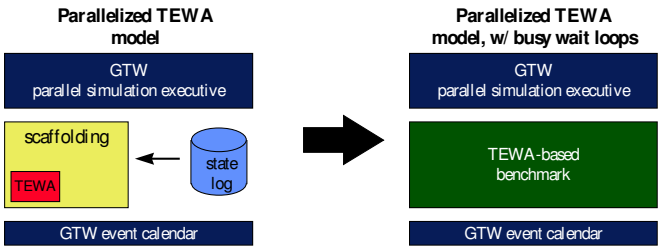

Figure 7: Performance TEWA benchmark program from parallelized TEWA model.

To solve the performance accuracy problem, we constructed a second program that precisely models the computational requirements the real TEWA module, as shown in Figure 7. In this TEWA-based benchmark program, busy wait loops are used to reproduce the event computation time of each TEWA event. A single TEWA event may take between 50 milli-seconds to 6 seconds to execute, depending on the number of threats a TOC needs to consider. For all experiments, the China-Taipei scenario was used. This scenario generates 57 threat objects, of which 36 are determined to be lethal. Additionally, this benchmark program includes all Time Warp specific state saving and parallel execution overheads (i.e., rollbacks, message cancelations, etc.).

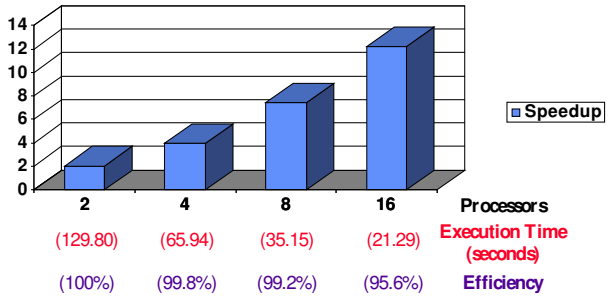

Figure 8: Performance measurements across varying number of processors for a fixed problem size.

Using the benchmark program, we conducted a series of experiments designed to determine the performance of the parallelized TEWA/GTW system. In the first experiment, a 64 TOC configuration is used and the speedup as a function of the number of processor used on a fixed problem size is calculated. The sequential TEWA module's execution time was empirically determined to be 259 seconds. As shown in Figure 8, we observe that the TEWA/GTW system obtains good speedups for all numbers of processors (up to 12 on 16 processors). Efficiency is defined to be the percentage of computation that *was not* rolled back. Here, we clearly see high efficiencies for all processor configurations, which is due to the high degree of parallelism that is available in the TEWA module.

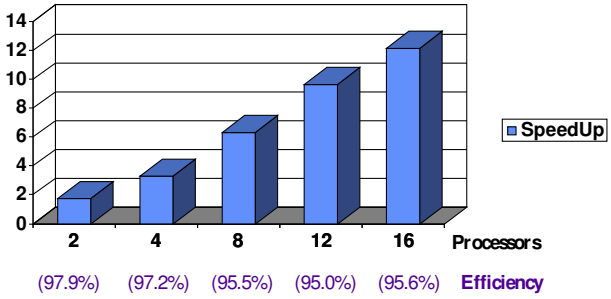

Figure 9: Performance measurements across varying number of processors for a scaled problem size.

In the second set of experiments, speedup is calculated for a scaled problem size as a function of the number of processors used. In scaling the problem size, we used 4 TOCs as our scaling factor. By that we mean that each processor has 4 TOC LPs mapped to it. For example, the 2 processor case models 8 TOC LPs, the 4 processor case models 16 TOC LPs and so on. The results from these experiments are shown in Figure 9. Like the previous set of experiments, we observed high efficiencies and good speedup across all processor configurations, which is again attributed to the high degree of parallelism that is available within the TEWA module.

6. Conclusions

As part of a proof-of-concept effort, we have successfully demonstrated that optimistic synchronization mechanisms provide a viable approach to reducing the execution time of BMC^3I applications. These mechanisms serve as an enabling technology for the development of faster-than-real-time BMC^3I applications which has the potential to provide battle commanders a "hot-situation" analysis tool and decision aide.

In addition to our performance results, we developed and demonstrated an incremental approach to re-engineering an existing large-scale sequential battle management simulation. Because of the similarities between Ballistic Missile Defense (BMD) and National Missile Defense (NMD), particularly in BMC^3I arena, we believe our approach could be used to reduce the execution time of NMD applications as well.

Acknowledgments

This work was supported by U.S. Army Contract DSAG60-95-C-0103 funded by the Ballistic Missile Defense Organization.

References

Briner, Jack, 1991, *Fast Parallel Simulation of Digital Systems,* Advances in Parallel and Distributed Simulation, volume 23, pp. 71—77, SCS Simulation Series, January.

Carothers, Christopher D., Richard M. Fujimoto and Yi-Bing Lin, 1995, *A Case Study in Simulating PCS Networks Using Time Warp,* In proceedings of the 9th Workshop on Parallel and Distributed Simulation, pp. 87—94, June.

Carothers, Christopher D. and Richard M. Fujimoto 1996, *Background Execution of Time Warp Programs,* In proceedings of the 10th Workshop on Parallel and Distributed Simulation, pp.12—19, May.

Das, Samir R. and Richard M. Fujimoto 1994, *An Adaptive Memory Management Protocol for Time Warp,* In proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 201—210, May.

Fujimoto, Richard M., 1989, *Time Warp on a Shared Memory Multiprocessor,* Proceedings of the 1989 International Conference on Parallel Processing, volume 3, pp. 242—249, August.

Fujimoto, Richard M., and Maria Hybinette, 1997, *Computing global virtual time in shared-memory multiprocessors*, ACM Transactions on Modeling and Computer Simulation, 7 (4): pp. 425--446.

Gupta, Anurag, Ian F. Akyildiz and Richard M. Fujimoto 1991, *Performance Analysis of Time Warp with Multiple Homogenous Processors,* IEEE Transactions on Software Engineering, volume 17, pp. 1013—1027, October.

Jefferson, David R. 1985, *Virtual Time,* ACM Transactions on Programming Languages and Systems, volume 7, number 3, pp. 404—425, July.

Presley M. et al. 1989, *Benchmarking the Time Warp Operating System with a Computer Network Simulation,* In Proceedings of the SCS Multiconference on Distributed Simulation, SCS Simulation Series, volume 21, pp. 8—13, March.

Wieland, Fredrick P. et al. 1989, *Distributed Combat Simulation and Time Warp: The Model and its Performance,* In proceedings of the SCS Multiconference on Distributed Simulation, SCS Simulation Series, volume 21, pp. 14—20, March.

Distribution Statement A.

Approved for public release; distribution is unlimited.