Implementing a Distributed Combat Simulation on the Time Warp Operating System

Frederick Wieland
Lawrence Hawley

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109

Abe Feinberg

Jet Propulsion Laboratory and
California State University, Northridge

Abstract. Utilizing the Time Warp Operating System, the CTLS project at JPL has produced a distributed combat simulation called STB-87 and measured its performance on the JPL Mark III Hypercube. By applying the spiral model of software development, the CTLS project will produce a series of software test beds, to culminate in the completion of a working prototype theater level simulation three to five years hence. STB-87, the first software test bed, is a ground-based combat simulation decomposed into objects which communicate via time-stamped messages. The use of incremental object-based design, coding, and testing has been helpful when developing a parallel simulation. The performance measurements show that, with the appropriate choice of object granularity, STB-87 is able to achieve a speedup factor of 12 running on a 32-node Mark III Hypercube.

The work described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the United States Army Model Improvement Program Management Office through an agreement with the National Aeronautics and Space Administration, under NASA contract NAS7-918.

1. Introduction

The Concurrent Theater Level Simulation (CTLS) Task at the Jet Propulsion Laboratory (JPL) has implemented the first of a series of software test beds on the JPL Mark III Hypercube. The first test bed, called STB-87, is an analytical ground-based discrete event combat simulation running on top of the Time Warp Operating System. The objectives of STB-87 are (1) to develop paradigms to distribute data structures of algorithms that traditionally use shared memory; (2) to develop a method for designing parallel discrete event simulations; and (3) to study the performance of large scale theater-level combat simulations. This paper reports on the results of these three objectives as they applied to STB-87.

2. The Time Warp Mechanism

The Time Warp mechanism is a concurrency control and synchronization strategy for any problem which can be mapped into a virtual-time coordinate system. Computer simulations and transaction processing systems are two problems which may exploit the mechanism. First developed by Jefferson and Sowizral [Jefferson 82], the mechanism relies on a decomposition of the application into objects which communicate via the exchange of time-stamped messages. The fundamental invariant maintained by the mechanism is that objects appear to process incoming messages in increasing time stamp order, and there is no test

the application programmer can perform to determine otherwise.

In actuality, the Time Warp mechanism executes all objects asynchronously and may execute messages out of time stamp order as long as there is no causal link between the two messages. Whenever two messages are executed out of order and there *is* a causal link between them, the mechanism causes a *rollback* to an earlier saved state of the objects and then proceeds forward, cancelling all effects of the earlier incorrect ordering. A more detailed explanation of the mechanism can be found in [Jefferson 85].

The JPL implementation of the Time Warp mechanism is called the Time Warp Operating System, or TWOS. Because the Time Warp mechanism affects every decision made by a normal operating system (memory management, garbage collection, process scheduling, queueing, etc.), the JPL implementation is a stand-alone operating system running on the 32-node Mark III Hypercube. Details of the Hypercube architecture can be found in [Peterson 1985]. TWOS handles all details of synchronization and concurrency control at a level invisible to the simulation, and provides the user with two methods of interprocess communication. The first method allows objects to read data from other objects via a *query message,* while the second method allows an object to send data (write to) another object via an *event message.* Further details of the programming model can be found in [Jefferson 87].

3. Project Background

The CTLS project is concerned primarily with faster execution and improved realism of combat simulations, and secondarily with the exploration of new simulation paradigms. The overall goal is to produce a prototype theater-level combat simulation in three to five years that can serve as a model for future simulation efforts. To achieve this goal, the project has adopted Barry Boehm's Spiral model of software development [Boehm, 1985], diagrammed in Figure 1.

The Spiral model is a risk-driven approach characterized by successive iterations through progressively larger (and more costly) cycles of determining objectives, alternatives, and constraints; evaluating alternatives and

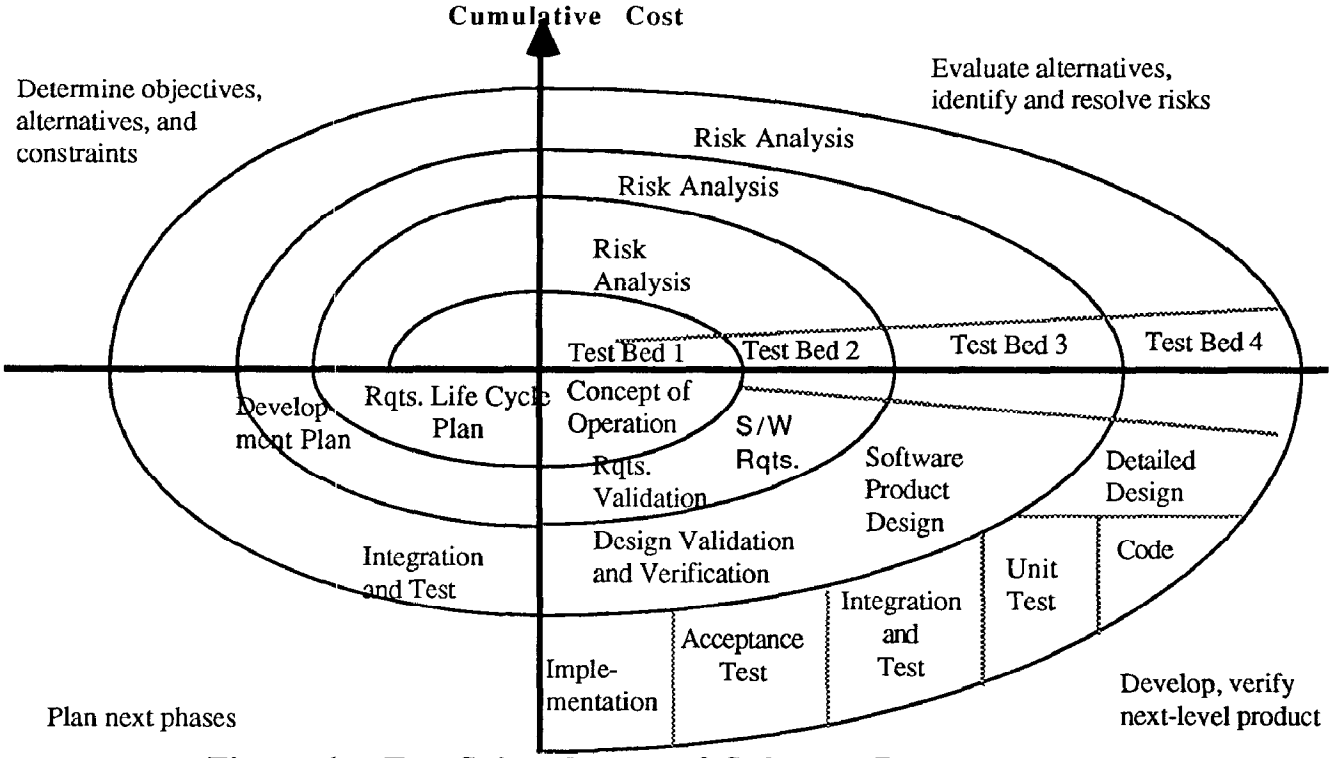


Figure 1. The Spiral Model of Software Development
Adapted from [Boehm 85]

identifying and resolving risks; developing and verifying the next level product; and planning the next phase. This software development method is particularly appropriate in a research environment where the product has not been produced before.

The CTLS approach to the Spiral model is to identify a series of software test beds of increasing complexity over the next three years. Each test bed is more elaborate than the previous one, and cycles through the stages of design, coding, and testing, followed by an elaborate performance evaluation procedure. During the performance evaluation, useful insight is obtained about TWOS as well as the particular test bed. Information gathered during the performance analysis is used both to identify further areas of study and to guide future test bed designs. To date, five test beds have been identified for production. The first such test bed, called Software Test Bed 87 (STB-87), has completed the first cycle of the spiral and is the subject of this paper.

4. The STB-87 Model

STB-87 is a division-on-division combat model incorporating limited intelligence as well as command and control. Lanchester attrition [Taylor 80] with one weapon type is used to calculate combat casualties. The model consists of two opposing sides, Red and Blue, initially owning 46 and 51 divisions respectively. Input data for each division includes its position on a 15 by 15 square array of grid tiles, its initial move orders, and its initial strength. By varying these parameters, the user can study the effect of combat geometry and force allocation on the outcome of a hypothetical battle.

The model is set up so that Red, which is positioned on the East side of the grid, marches West until it encounters Blue. Each division is informed about the position and strength of any other division within a 50 kilometer radius. Although each side has one corps unit, its purpose is to keep track of the forces; it does not make any substantive command or control decisions. Rather, decisions are made locally by each division based on the intelligence data it receives, its current posture (Attack, Defend, Delay, or Withdraw), and its mission (Attack for Red and Defend for Blue). A division is destroyed and removed from the game when its strength falls below 15% of its initial strength. Consumption and resupply is not modelled in this test bed.

Optional color graphics output from the model displays the locations of each division, and two histograms represent the strengths of the Red and Blue forces as a percent of their initial strengths. The graphics output is used for model verification, while textual output is provided for detailed analysis of the scenario.

5. The Design of STB-87

STB-87 is an object-based discrete event simulation. Each object communicates with other objects via a series TWOS event and query messages. Objects read messages in increasing time stamp order, and invoke local behaviors to process each message. Although this computational model is similar to object oriented programming (OOP) [Booch 86], it differs from standard OOP in at least two respects. First, behaviors cannot be inherited from more general objects. This limitation arises from the use of C as a programming language and will be rectified in the near future. Secondly, STB-87 objects are more coarse-grained than objects in a typical OOP system. However, the programming model does incorporate data abstraction and information hiding, two features commonly found in OOP systems.

The first step in model decomposition is to identify objects and the interactions between them. For STB-87, several different types of objects were identified, namely a division object, a grid object (to keep track of unit positions), and a corps object. Ancillary objects such as statistics objects and output objects are optionally part of the simulation, but do not affect the outcome if not included. The interactions between the objects are relatively straightforward. The grid objects are responsible for providing intelligence data to the division objects. The division objects are responsible for informing the grids of their planned series of moves, and for exchanging attrition information with an enemy division during a combat. The corps object, although normally responsible for

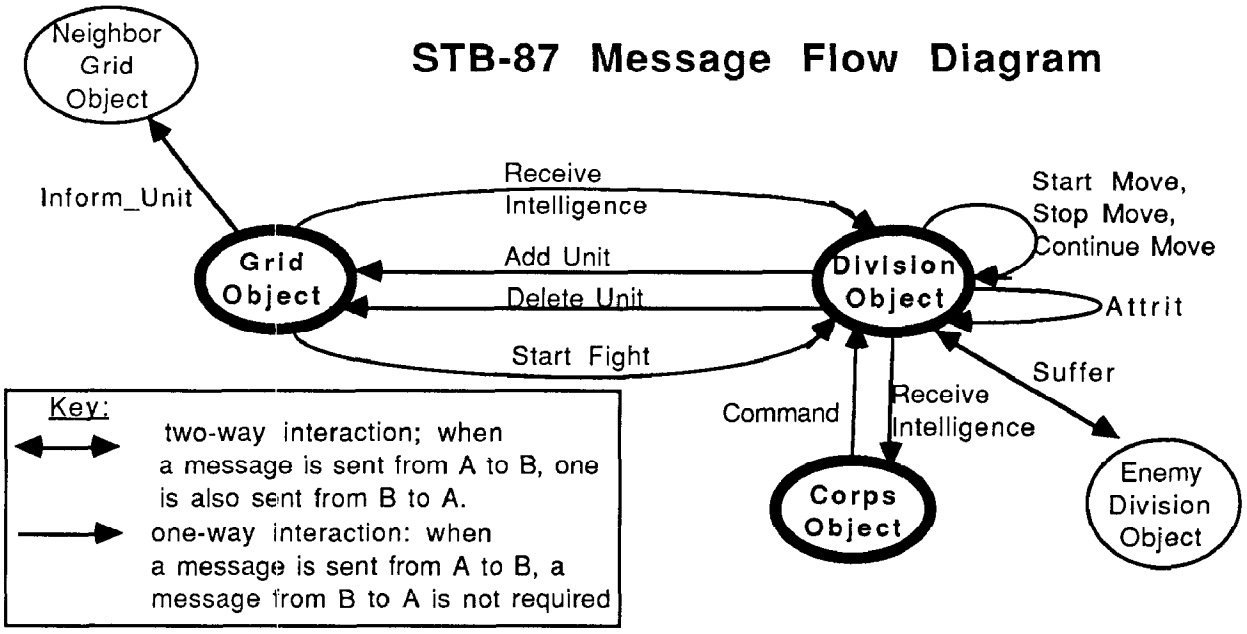


Figure 2. STB-87 Data Flow Diagram

command and control, does not provide substantive commands but rather gathers information about its subordinate divisions. A dataflow diagram for this scheme can be found in Figure 2.

A number of questions can be asked about the design in order to estimate its degree of parallelism [Beckman 88]. Among these are the presence of object bottlenecks, the complexity of message communication, the granularity of objects, and the dynamic recomposition of objects. The STB-87 design is analyzed with respect to these factors below.

Object bottlenecks. Although in this design the corps object does not materially participate in the model, in a general military simulation the objects in the upper echelon of the hierarchy will be natural bottlenecks. These object may have to be repartitioned into subobjects to avoid potential bottlenecks.

Message Passing Complexity. The number of messages in this simulation grows linearly with the number of objects and the number of battles. Since message passing is restricted to objects which interact locally in a geographic sense, the design can be scaled up to many hundreds or thousands of objects.

Object granularity. Objects were chosen which represent real world entities--divisions, corps, and grids (which correspond to map objects). In general, however, one can trade off object granularity with parallelism--small granularity objects provide much potential parallelism, but at the risk of losing that parallelism through message passing overhead. As will be shown later, the granularity of objects in STB-87 was found to be too small for the current TWOS implementation.

Dynamic recomposition of objects. The ability for objects to dynamically aggregate and decompose is not present in STB-87. In general, this feature is a very difficult one to incorporate in any model. The interactions between STB-87 objects are irregular both in time and in space (i.e. when and to whom each object communicates with is irregular and changes throughout the simulation). Such irregularity argues for some type of load balancing in the operating system, a feature absent from the current version of TWOS but which will be included in a future version.

6. Distributing Shared Memory Data Structures

Current combat simulations run on uniprocessor machines with a large virtual memory address space. Fortran and Simscript tend to be the dominant implementation languages. The

programming model assumed by the simulation designers consists of a large address space shared by the various objects in the system. One example of this programming model is the handling of unit locations, which is stored in a centralized list sorted by geographical region. As units move around the battlefield, the list is updated sequentially and always contains the most accurate information.

In a distributed memory environment, objects in STB-87 do not have access to a single global list. Although it is possible to imagine a design in which one object managed such a list, the message passing bottleneck imposed by such a design would minimize speedup gains from the parallel execution. Instead, the unit location data structure must be distributed among many different objects. In STB-87, the grid objects are responsible for managing the distributed unit location data structure. Each grid object manages a small contiguous piece of the gameboard, and as units move from grid to grid, messages are passed among the affected objects to keep the data structure current.

In distributing the unit location list, the STB-87 model becomes more complex due to the additional objects and messages needed to distribute and update the unit locations. One of the objectives of STB-87 was to determine if the added message passing overhead of the distributed implementation limited the speedup inherent in the model.

7. Implementation of STB-87

STB-87 is designed to run with TWOS on the Mark III Hypercube. It contains approximately 5,000 lines of C code, including comments and type declaration files. The model was first specified as a high level dataflow diagram, and then designed object-by-object in a top-down fashion. The design was hand-translated into C, then tested and integrated with TWOS. The whole process took about ten work weeks between April and September 1987. The performance measurement program, whose results are reported in section 5, was begun in October 1987 and continued through the beginning of 1988.

During the performance measurement program, STB-87 was changed once. Implementation 1 contained 277 objects, representing 225 grid objects (a 15 x 15 square array), 25 red and 25 blue divisions, and 2 corps objects. Implementation 2 contained 389 objects, representing 225 grid objects, 46 red divisions and 51 blue divisions, 32 initialization objects, 2 corps objects, 1 statistics object, and 32 output objects. The initialization, statistics, and output objects are ancillary objects not shown in Figure 2.

The rapid success of STB-87 development can be attributed to its small size, its modular decomposition, and the availability of software tools during the coding phase. Specifically, software tools available on a Unix-based workstation greatly eased the development of the final C code.

8. The Performance of STB-87

One of the initial objectives entailed measuring the speedup obtained from STB-87. Because the model contains highly irregular interactions in time and space, as well as data structures (such as unit positions) which are more efficiently implemented in shared rather than distributed memory, there were valid concerns as to the potential speedup inherent in the model. The speedup results, discussed below, are displayed in figures 3-5.

All speedup curves presented herein measure the speedup of the parallel execution of STB-87 running under TWOS against a non-TWOS serial execution. The non-TWOS system is a sequential discrete event simulator, in which all inter-object messages are stored in one global queue in timestamp order. The sequential simulator does not incur the synchronization and concurrency control overhead necessary in the parallel Time Warp system. On both the serial and parallel systems, STB-87 remained unchanged. Thus the speedup compares the best parallel execution time against an efficient sequential execution of the same program.

Figure 3 shows the speedup of the first implementation of STB-87 as a function of the number of processors (nodes). Although the model did speed up as the number of nodes

increased, the amount of speedup was a disappointing factor of 4.5 on 32 nodes.

Further study revealed that the fine granularity of CTLS-87 objects in both implementation 1 and 2 limited the speedup. On the average, objects perform 5 ms of computation for each message; this number varied from a low of 3 ms to a high of about 12 ms. Since the current TWOS/Hypercube system requires a few milliseconds to transmit a message (most of the time is spent in packetizing, depacketizing, and queueing), communication overhead dominated the system.

In order to determine a reasonable granularity for objects, each object in implementation 2 was artificially padded with a "do-nothing" delay loop, which effectively increased the computation time per message and caused the simulation to become more uniform. Figure 4 shows how the speedup increased as the computation time per message increased when this delay was added. The increased speedup is due both to the increased computation time per message and to the fact that the objects run more uniformly in time. Whereas in the unmodified STB-87, objects had roughly a 4:1 ratio between the longest running (12 ms) and the shortest running (3 ms), in the delayed STB-87 objects this ratio approaches 1:1. For example, with a 40 ms delay, the longest running object takes 40 + 12 = 52 ms, while the shortest takes 40 + 3 = 43 ms, no longer a 4:1 ratio.

In a reasonably large theater-level combat simulation, it is estimated that each object will perform a few tens of milliseconds of computation between messages, and that the computation differences will approach 1:1. Thus it was appealing to run a complete STB-87 speedup curve with the amount of computation fixed at around 20 ms per message. Figure 5 shows the speedup results of implementation 2 with 20 ms of computation per message. Note that, unlike Figure 3, the speedup for this system increases monotonically with the number of nodes. On more than 32 processors, it is quite likely that the system will exhibit a speedup greater than 12.

The purpose of this delay experiment was to discover a reasonable object granularity for the current TWOS system. As the TWOS systemis optimized and ported to different host multiprocessors, the optimal object granularity will decrease.

Figure 3. First STB-87 Speedup Results

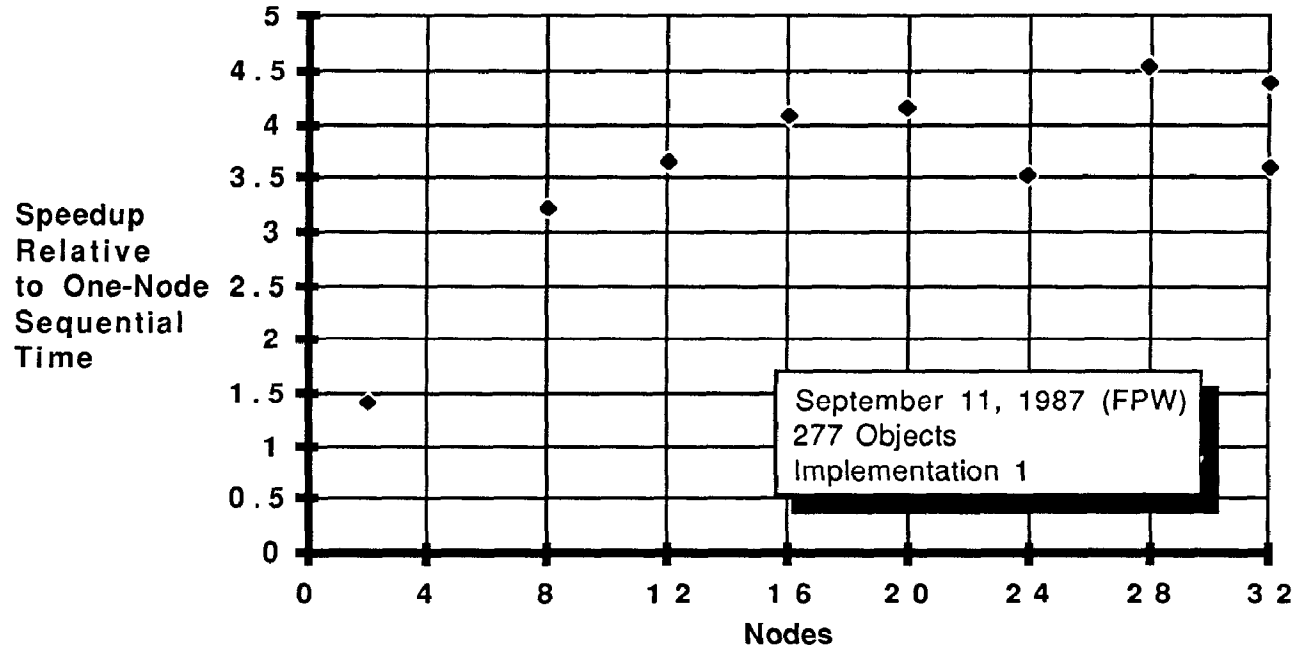


Figure 4 Effect of Calculation/Communication Ratio on Speedup

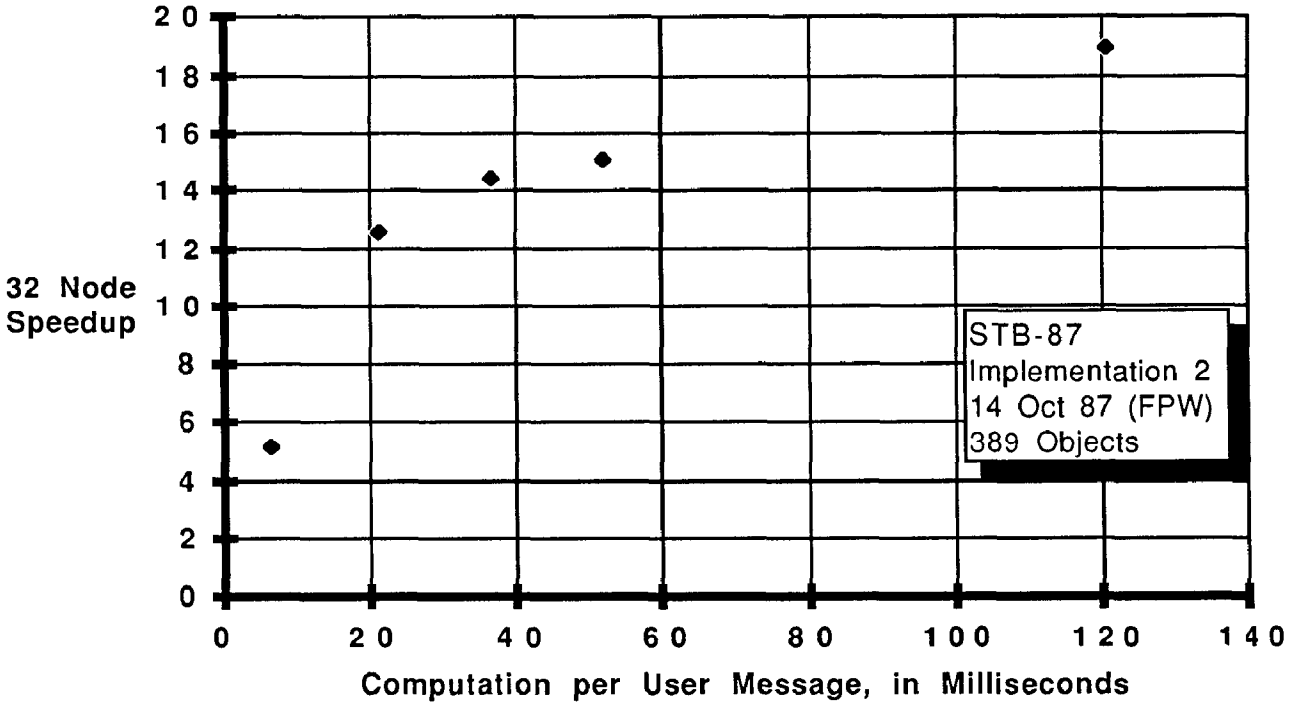
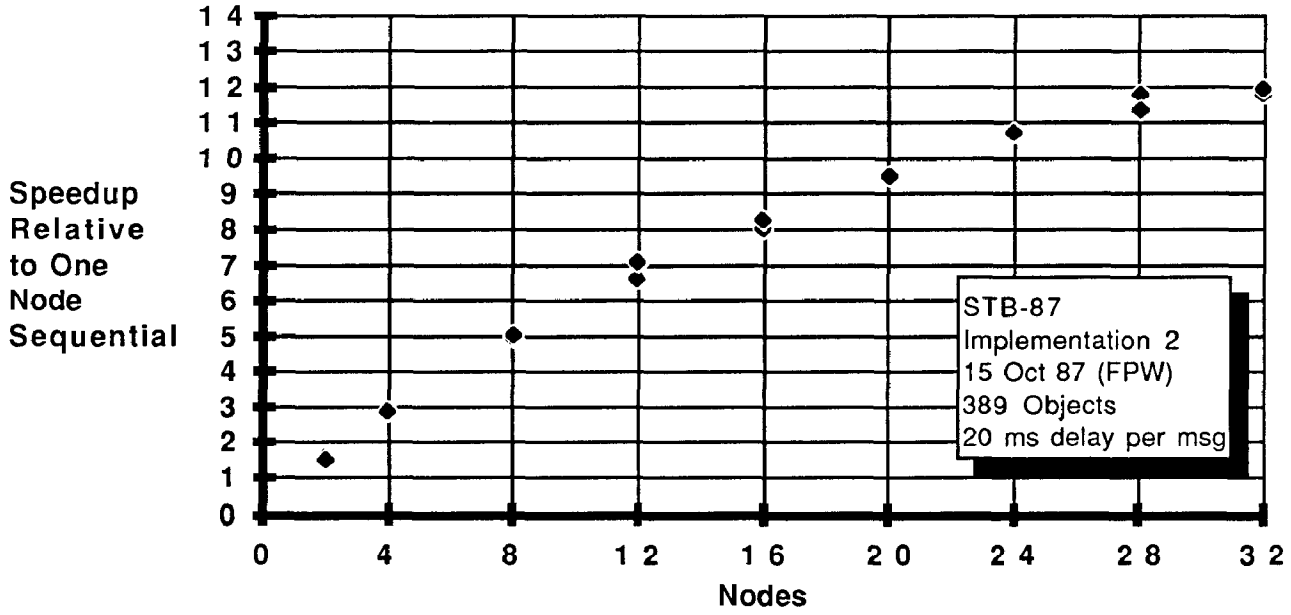


Figure 5. Final STB-87 Speedup Curve



9. Conclusions

Positive results have been achieved with respect to the original three objectives. Utilizing the first of a planned series of software test beds, it has been demonstrated that shared data structures, such as unit position information, can be changed to a distributed memory data structure with no loss in the fidelity of the model.

Secondly, the project has shown that the use of an object-based decomposition method aids in the development of a parallel program. Although other methods for parallel decomposition have not been explored, it is believed that this method is superior to no method and holds promise for future implementations. Finally, the performance of STB-87 was studied sufficiently enough to gather some insights useful for the design of future test beds. Specifically, it was discovered that objects need to be fairly coarse-grained and uniform in execution time to run effectively with the current version of TWOS.

As larger and more complex models are built, a number of unresolved issues remain. The first is how to map the military command and control hierarchy onto a set of objects that will minimize bottlenecks. The hierarchical structure of military chain of command imposes natural bottlenecks on the design, which must be minimized for good parallel performance. The second is how to incorporate large databases on a distributed memory machine. Military simulations are characterized by large input databases containing such information as the effectiveness of weapon i against target j (for large numbers of i and j), attrition coefficients, logistics information, scenario data, etc. This database must be efficiently accessed for acceptable parallel performance. The third concerns further issues involved in modelling distributed data structures. Combat simulations traditionally have run on very large serial machines, and the algorithms and data structures have assumed a shared memory, single-processor programming model. As more military objects are incorporated, such as air support and ground radar, the challenge of utilizing distributed memory modelling techniques that differ from a shared memory design must be met.

Acknowledgements and References

We would like to thank the current TWOS team at JPL responsible for maintaining and improving the system, which includes Brian Beckman, Mike DiLoreto, Phil Hontalas, Leo Blume, Peter Reiher, and John Wedel. We want to thank David Jefferson of UCLA for his invaluable input to the project. We also thank Bob Miller, Herb Younger, and Jack Tupman of JPL for providing a productive work environment.

[Beckman 88] Brian Beckman, *et. al.* "Distributed Simulation and Time Warp Part 1: Design of *Colliding Pucks*", *Proceedings of the SCS Conference on Distributed Simulation,* Volume 19 Number 3, February 1988.

[Boehm 85] Barry Boehm, "A Spiral Model of Software Development and Enhancement," TRW Defense Systems Group, Redondo Beach, CA.

[Booch 86] Grady Booch, "Object-Oriented Development," IEEE *Transactions on Software Engineering,* Vol SE-12, No 2, February 1986.

[Jefferson 87] David Jefferson, *et. al.* "Distributed Simulation and the Time Warp Operating System," ACM *Operating Systems Review,* November 1987.

[Jefferson 85] David Jefferson, "Virtual Time," ACM *Transactions on Programming Languages and Systems,* Vol 7, No 3, July 1985.

[Jefferson 82] David Jefferson and Henry Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism Part I: Local Control," Rand Note N-1906AF, The Rand Corporation, Santa Monica, CA, December 1982.

[Peterson 85] J. C. Peterson, *et. al.* "The Mark III Hypercube-Ensemble Concurrent Computer," *Proceedings of the 1985 International Conference on Parallel Processing,* IEEE Computer Society Press, 1985.

[Taylor 80] James G. Taylor, *Lanchester-Type Models of Warfare,* US Naval Postgraduate School, Monterey, CA 1980.