Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

INFRASTRUCTURE FOR RAPID EXECUTION OF STRIKE-PLANNING SYSTEMS

Darrin West
Larry Mellon
Jim Ramsey

Science Applications Int. Corp.
4301 N. Fairfax Dr., Suite 370
Arlington, Virginia 22203, USA

John Cleary

The University of Waikato
Te Whare Wananga o Waikato
Private Bag 3105
Hamilton, NEW ZEALAND

Jim Hofmann

Naval Research Laboratories
4555 Overlook Ave
Washington, D.C. 20375, USA

ABSTRACT

A rapid-planning system for military aircraft strikes is under design. It is intended to be capable of creating aircraft routes through enemy and friendly space with minimum loss of aircraft and maximal damage to specified target areas. The system must support joint strike planning, where the effects of several simultaneous strikes by differing groups of aircraft are captured.

This paper describes a three-phased approach to the analysis of routes: static analysis to establish potential routes, detailed simulation to capture dynamic behaviors in the system, and human-in-the-loop evaluation of the most promising routes.

A parallel, discrete-event simulation technique is proposed to support the detailed simulation. Optimizations based on application characteristics are described. A technique to combine discrete-event and time-stepped models is proposed. Performance results of the current simulation engine are given.

1 INTRODUCTION

Strike planning involves the rapid evaluation of various options for achieving a given objective. Strike mission commanders are often given secondary targets in case access to primary targets proves too difficult. Flight paths need to be selected based on minimal risk to the crew and aircraft and maximal chance of success of the mission. Optimizing these variables is extremely challenging, given that there may be millions of potential route combinations to consider. Key aspects of the system include:

- execution speed: routes must be planned very quickly in an operational wartime system
- rapid data inputs: fresh input from the battlefield, such as effects of the last strike on the enemy detection ability, must be rapidly fed into the planner
- automatic creation and evaluation of potential strike plans

Currently, the set up and execution time to perform even a single evaluation is high. The execution time of a complex scenario modeling multiple simultaneous strikes with supporting command and control and opposing forces is expected to be even higher.

We will discuss an approach to solve these challenges, and discuss various optimizations and implementation techniques available.

2 OVERALL APPROACH

The rapid evaluation of millions of potential routes is only possible using methods having highly abstracted models. A number of methods based primarily on network flow analysis will be used to reduce the millions of potential routes to a much smaller subset of promising routes. The primary drawback to this technique is that it involves static analysis in which dynamic (or time-based) changes to the system are lost.

A second stage analysis of the promising routes will be done via a more detailed simulation, capable of capturing the dynamic aspects of the system.

Examples of dynamic effects required for correct modeling:

- reactions of the enemy defense network on sighting of incoming aircraft
- temporary suppression of enemy sensors and communications via electromagnetic jamming
- missile strikes on enemy sensors, command stations, and communications networks
- bi-polar sensors, which are capable of seeing in only one direction at a time.

A third stage analysis of the further reduced set of routes will be with a human-in-the-loop evaluation of each plan, via a detailed walk through (with editing power) of the plan using the dynamic simulator.

2.1 Problem Statement

The models used for strike planning can be described abstractly as a set of mobile entities whose kinematics

are continuous. Entities interact at unpredictable times to modify their kinematics in response to internal or external Command and Decision (C&D) entities. The C&D models are fed data by sensor models, which may execute in either continuous or discrete-event operation. Additional entities exist to model fixed-position systems, such as radar stations, command/control centers and communication systems. These also tend to interact in an unpredictable (i.e. discrete-event) fashion.

If the system is decomposed as in Figure 1, then the system consists of entities with continuous behavior and entities with primarily discrete-event behavior. Interactions between entities of different classes are expected to be occur infrequently, relative to the frequency of time-steps in the continuous entities.

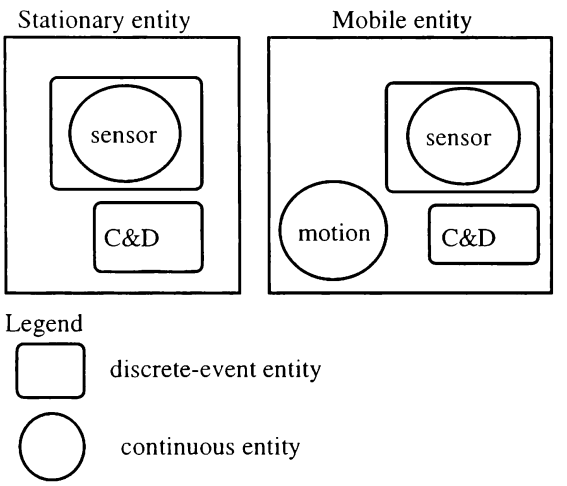

Figure 1: Classes of Entities

3 TECHNICAL APPROACH

The static analysis portion of the strike planner is described in Moret et al. (1995), including performance results from current experiments. Parallel simulation techniques are proposed for the dynamic simulator. The use of parallel hardware to improve the performance of dynamic simulations has been researched extensively in the Parallel and Distributed Simulation community. Several approaches exist, including functional parallelism, parallel replication, and inter-event parallelism (Fujimoto 1990).

1) **Functional Parallelism.** Functional parallelism executes different functional portions of a simulation on different processors. One could have random numbers generated on one processor in preparation for their use by event handlers on another, for example. This technique tends to reveal only limited amounts of parallelism.

2) **Parallel Replication.** Replication is done to reduce variance in stochastic simulations by varying the random number generator seeds, and performing sufficient experiments that statistical confidence is gained. Multiple executions will also be needed to evaluate each promising route. Parallel replication takes advantage of available processors by simultaneously spawning one sequential simulation execution per processor with different initial parameters. There is very little overhead in managing the simulation executions, resulting in very efficient use of computer resources, and significantly reduced elapsed time for the set of experiments. This holds true in both distributed memory and shared memory parallel computing environments, since this method is not very sensitive to communication efficiencies. This technique may require more memory resources than other parallel techniques due to the large number of entities executing simultaneously. It is a simple and well known technique that will be used in the planning system if appropriate, but will not be discussed further in this paper.

3) **Inter-event parallelism.** Interactive simulation, where an operator is involved in decision making, often has more demanding performance requirements. Real time interaction may require the parallelization of a complex or large scenario in order to meet ongoing deadlines. Iterative refinement of parameters (by hand or using automatic parameter searching), debugging, or execution in a time critical operational environment require the operator to wait for the simulation to complete. Reducing the end to end time of these executions through the use of inter-event parallelism will save time and frustration. This is done by selecting an event to execute for each available processor. It requires synchronization between the processors so that the events are executed in an acceptable order. We proceed to describe a kernel that uses this technique, and various optimizations possible within the specific domain of strike planning.

3.1 Parallel Discrete Event Simulation

We propose to solve the time critical aspect of the strike planning problem by applying well known parallel discrete event simulation (PDES) techniques (Fujimoto 1990). In PDES, entities interact solely via timestamped messages. Messages are received in timestamp order. Since entities will not proceed through simulation time at precisely the same speed on each processor, a synchronization mechanism is required to ensure that messages are received in order. There is often a significant amount of parallelism available between events in a large simulation, but allowing entities to execute correctly without significant synchronization overheads is a hard problem.

3.2 Time Warp

Normally one would select the lowest time stamped event (or set of equivalently time stamped events) to execute. This leads to only a small amount of

parallelism being extracted. Events at different times can be executed simultaneously when they do not influence each other. Ensuring that events remain synchronized while allowing inter-time, inter-event parallelism is a hard problem.

As opposed to blocking until the chance is zero of an earlier message arriving, Time Warp allows entities to optimistically receive future messages. When there are no earlier messages, the gamble pays off by making use of the time otherwise spent blocking. However, when a late message arrives in the entities past, the entity is rolled back to the time of the late message by restoring a copy of the entity's state at the time of that event. The cost of this state saving and the cost of transmitting messages between processors are the primary overheads in a Time Warp kernel.

3.3 Model Level Optimizations

By making use of characteristics specific to strike planning, we can optimize the time warp kernel's performance. The computation performed by sensor models can be reduced by limiting the input data to a set potentially visible to the sensor using a technique called sectorization, described in Beckman et al. (1988) and Mellon (1994). Entity data can be separated into sets based on position with respect to a fixed physical grid. Instead of inspecting data on each entity in the simulation, the sensor can inspect the subset of entities in sectors that overlap with the sensor footprint. The communication between a motion entity and the sensor model would be handled by having the motion models send updates to a sector manager, and having the sensor models read the updates from the sector manager. This reduces the number of interactions in a fully connected set of sensors and platforms from $m*n$ to $m+n$.

The simulation can make use of dead reckoning, a central principle of Distributed Interactive Simulation (DIS 1994), to further reduce the number of interactions between entities and sensors or sector managers. If the trajectory of the entity remains predictable for a long period of time, perhaps due to an infrequent adjustment by the tactics model, remote sensors can use earlier data and estimate the current position of an entity. Entities are thus not required to update their position and vector until it changes from what sensors can estimate. This will reduce communication traffic at the expense of extra computation for this estimation. Note that dead reckoning has been successful in systems with much higher communication overheads than the shared memory system proposed here. Further investigation will determine the usefulness of dead reckoning in this application.

3.4 Optimistic and Time-Stepped Execution

As described in the problem statement, C&D entities tend to be modeled in a discrete-event fashion, whereas motion models tend to be executed in a time-stepped fashion. This section introduces a synchronization approach to allow efficient execution of both classes of entities by the same simulation kernel.

Typically motion models are executed at periodic time steps. The positions of such motion entities are examined frequently by many other entities and potentially can be examined by nearly all entities. However, they rarely receive inputs from the rest of the simulation - such inputs typically are commands to maneuver or alter direction which occur infrequently on the time scales of the motion models. In contrast the command and decision entities are event driven with irregular communication patterns. They execute infrequently and as the result of a specific event. They often examine the positions of large numbers of motion entities and generate a small number of events for other C&D entities or for motion entities.

Taken on their own these two groups of entities would best be scheduled by quite different types of simulators. The motion entities could be executed by a simple time-stepped simulator where the state of each entity was globally readable. The overheads of such a simulator are very low, requiring only the dispatch of each entity once on each time-step. Well balanced parallelism is easily achieved by distributing the activation of the entities across a number of (shared memory) processors. Since each entity takes some action each time-step, simulation execution efficiency is high.

In contrast the command and decision entities would probably be best served by a discrete event simulator, as this class of entities act only on key events in the system, and thus do not perform actions each time-step. Such entities are best parallelized by systems generally referred to as optimistic. Optimistic parallelism is the only way currently known to effectively parallelize simulations where the communications between entities are irregular and infrequent (Fujimoto 1990). Unfortunately, optimistic carries with it significant overheads in both execution time and complexity. In particular, the state of optimistic entities cannot be directly examined by other entities and all communication must be via time stamped messages. This can have very high overheads when motion objects have their state examined by many other objects. Optimistic execution also requires the capability to roll back execution which requires in turn the state of the entities to be saved before executing each event. Again this implies a very high overhead for motion entities especially as they seldom receive messages from other entities and so are seldom rolled back.

The approach taken here is to use a hybrid simulator where the motion entities are scheduled in time-steps without state saving (i.e. conservatively) and the C&D entities are scheduled using an optimistic simulator. The motion entities have their state (positions) displayed in globally accessible memory. The command and

decision entities can read the positions directly from this memory without using messages. Communication from the C&D entities is via messages. To do this correctly requires some synchronization between the two different schedulers.

Note that some form of control must occur of optimistic entities reading the state of conservative entities, if the optimistic entity is more than one time step ahead of the conservative entity. Two possible approaches exist. First, the kernel may track the reads of optimistic entities and roll them back if the read turns out to be incorrect. Second, the kernel may block the optimistic entity until it is less than one time step ahead, and the data is thus correct. A tradeoff of increased parallelism versus increased overhead per read clearly exists. Further analysis will be done to determine the best approach.

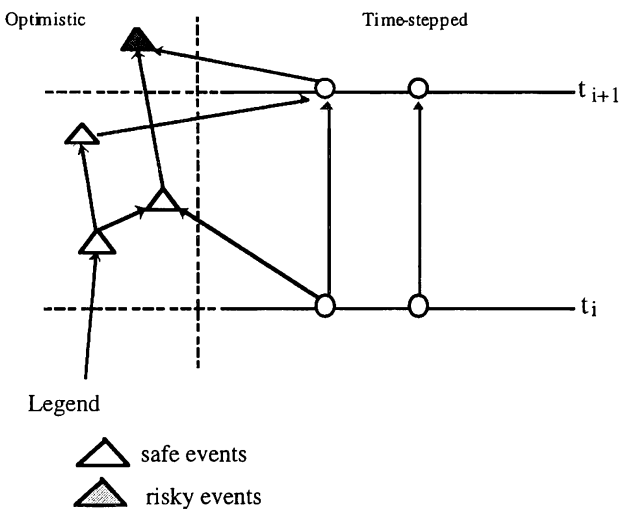

Figure 2: Interaction of Optimistic and Time-stepped Entities

The simplest form of synchronization consists of a simple alternation of time-stepped and optimistic execution. Assume that the time stepped simulator executes at times t_i, t_i+1,t_i+2, ..., then execution proceeds as follows:

- all motion updates at time t_i are executed (including receipt of any messages from C&D entities that have arrived since t_i-1)
- all C&D entities scheduled between t_i and t_i+1 are executed optimistically (and in parallel)
- all motion updates at time t_i+1 are executed
- and so on ...

This requires that the distributed scheduler resynchronize between each step. This is straightforward for the time-stepped scheduler as each processor need only report when it is finished its part of the computation. For the optimistic simulator it is necessary to compute the Global Virtual Time (GVT) of the C&D entities and stop when it passes the next time-step point. This can be done efficiently using published shared memory GVT algorithms, described in Zhonge et al. (1995) and Das et al. (1994). The first of these allows calculation of GVT to be started asynchronously by any processor. The overheads of GVT calculation can be optimized by the observation that it is only necessary to know when GVT passes a time-step. So the GVT calculation need only be initiated as each processor's LVT passes the next time-point and can be aborted whenever another processor is detected with an earlier LVT.

One potential problem with this hybrid scheduler will arise if the number of C&D entities to be executed within one time step is less than the available processors. Then processors will be idle during the optimistic phase reducing the overall efficiency of the simulator. This will be particularly acute if the individual C&D calculations are long compared with one time-step iteration. This situation can be ameliorated if static lookahead information is available about the communication delays between C&D entities and motion entities. This will allow motion entities to continue to execute instead of waiting for the C&D entity to finish its event.

Consider a C&D entity that never affects the motion entities. Such an object need not delay the start of the next time-step calculation and so can be excluded from the GVT calculation that synchronizes the start of the next time-step. The advantage of this is that the available optimistic parallelism is increased (the entity can be scheduled earlier or in parallel with the time-step calculation) and wasted CPU time in the optimistic phase can be decreased. This technique can be further generalized if it is known that there is a minimum delay from the time when a C&D entity executes and when it affects the motion of motion entities. For example, there may be a minimum time from an aircraft changing motion, to it being painted on a radar screen, to it being noticed by a human observer and then a voice command being given for other aircraft to intercept. This minimum delay (or lookahead) can be added to the LVT of each C&D entity when GVT is being computed. This augmented GVT may be somewhat ahead of the true GVT and thus allow a significant improvement in available parallelism and overall efficiency. This allows conservative time stepped models to execute sooner, and in turn, the optimistic models will have access to the global state data sooner, thus increasing the number of events that are allowed to run without risk (parallelism).

Time-stepped calculations have traditionally been done two ways: either one copy of the dynamic position information is kept and re-computed in place at each time-step; or two copies of the dynamic state are kept where the new state is recomputed completely from the old state and the old and new states are alternated. If the second, alternating state, is used this can be integrated with the optimistic calculation as a C&D entity can

look at either of the states depending upon its current time (note that optimistic entities may be behind some or all time-stepped entities in simulation time). Thus potentially twice as many optimistic entities would be available for execution at one time. Indeed this idea can be further generalized by allowing more than two copies of the motion state. The execution of a time-step will still be constrained by the advance of the augmented GVT and optimistic execution cannot start until the previous time-step calculation is complete. However, given sufficiently large lookahead values C&D entities could be executing at many different time-steps.

3.5 Technical Summary

We expect that reduction of data exchanges achieved via sectorization will significantly increase the potential parallelism in the system. Dead reckoning mechanisms may also increase performance by creating a significant amount of lookahead for use by the kernel. This translates into available parallelism without needing to perform state saving. Conservative execution of the time-stepped kinematics models should not result in a loss of parallelism, since we expect a large number of vehicles, and these models do not frequently receive new inputs. Access to committed data in shared memory will significantly improve sensor model efficiency as compared to relying on message based communication. We expect to have significant available/intrinsic parallelism, and low overheads, and thus high efficiency/utilization of the processors.

3.6 Related Work

Jah and Bagrodia (1994) present a similar theory to combined conservative / optimistic execution in the protocol ADAPT. An excellent comparison is given between protocols that support simultaneous use of both optimistic and conservative execution, and protocols which provide a single synchronization algorithm which encompasses aspects of both optimistic and conservative techniques. Aspects of ADAPT clearly apply to this problem domain, in particular, ASPECT's support for entities switching dynamically between optimistic and conservative execution.

Gilmer et al. (1990) present a time-stepped / discrete-event kernel for the Dynamic Ground Target Simulator (DGTS). In this system, a functional analysis of the event classes in the simulation was performed. Each event class was then categorized requiring time-stepped or discrete event execution. The kernel then optimized parallel execution based on the class events in the current event list. Very good performance was achieved for that model (approximately 6.5 times speedup on 10 processors), but is not considered a general purpose solution. Aspects of this work may be used for this strike-planning system, once event characteristics are defined.

Lee and Fishwick (1995) present an approach for determining the minimum set of seeds, samples and fidelity levels to be used in route-planning simulations. The high-level controller for this strike planning system will take use of such work in the setup of replicated computing runs, and in the tradeoff between many small runs of low-fidelity models (i.e. 20 low-fidelity replicated computing executions on 20 processors), versus fewer, high-fidelity runs (e.g. 4 high-fidelity runs, each on 5 processors).

4 TEMPO DESIGN

The simulation kernel we will use for the strike planning system is called Tempo. Tempo is a time warp based simulation engine. It is implemented in shared memory to take advantage of the low overhead needed to pass data from one processor to another. It uses an event-oriented paradigm to avoid the context switching associated with a process oriented paradigm (Pooch and Wall 1993). It has been designed to provide effective parallelism for simulations with very small computation per event (i.e. *granularity*). Event granularity of under 100 microseconds is expected to occur.

Tempo uses the shared-memory approach discussed in Das et al. (1994) where logical processes (LPs) communicate by exchanging time-stamped event messages. Memory for event messages is allocated from a pool of shared memory. These event messages are passed between processors by handing over a pointer to the message. This reduces the number of copies of data during event handling to the same minimum number as must be used in a sequential simulation, due to intrinsic delays in simulation time. The message is linked into a priority queue local to the destination processor using a mutual exclusion lock, since many senders may attempt this operation simultaneously. Upon receiving an event, an LP may change its own state and schedule events for other LPs in the future.

Unprocessed event messages are kept on a priority queue, ordered by receive time. The event list uses a calendar queue described in Brown (1988) and Jones (1986). This data structure is O(1) with a very low overhead for many stable distributions of events.

Global Virtual Time (GVT) is calculated using a shared data structure and a series of locks. We are in the process of replacing the GVT calculation with one that is more efficient and asynchronous. GVT is used to fossil collect memory from old state saves and processed events. It is also used to schedule the conservative processes, therefore the more current the value of GVT, the more processes may become available to execute in parallel.

On a multiprocessor computer, LPs are assigned to run on different CPUs. Each logical process stays on its assigned processor to maximize the locality of memory references (informal testing has shown a 10 to 1 cost

ratio in cache misses). Using time warp principles, in particular state saving and rollback, optimistic LPs can execute events in parallel with low overhead and perfect causality.

Specialized LPs which are scheduled conservatively and therefore do not have to be state saved may read and write data files. These conservative processes may also directly read and write shared memory. They are scheduled when the value of GVT equals or exceeds their next event time. Modifications based on lookahead information described in Section 4 allow these processes to execute in parallel with each other and with optimistic processes.

5. PERFORMANCE

5.1 Shared Memory Hardware

Small shared memory computers have been successful in the marketplace because of their ease of use and low cost. Most have one or a small set of common memory buses, making them significantly less scalable than machines such as hypercubes which do not have such a single shared resource. Non-uniform shared memory architectures promise scalability, but may be significantly more difficult to tune for performance. The main attraction of shared memory machines is that they are inexpensive and becoming more commonly available to users. Given the number of processors available in low-cost shared memory systems (32-64), and that the effective processor utilization of parallel discrete-event simulations tend to taper off at high numbers of processors, shared memory systems seem to provide a reasonable price/performance ratio. Also note that high-end shared memory systems have been developed, if additional processors are warranted. (e.g. KSR supports hundreds of processors).

Shared memory machines provide the only reasonable promise of achievable performance for very small granularity applications. Communication is possible by passing a pointer, and message copying is reduced to the same as what must be performed for a sequential simulation. Message passing architectures will necessarily have a second copy, and often have operating system overheads associated with the message I/O.

5.2 Tempo Scalability

Tempo minimizes the use of central shared data structures with their requirements for mutual exclusion locks. The few data structures that do require mutual exclusion have small critical sections (less than 20 machine instructions), and are being replaced when they are identified as causing a performance bottleneck.

The main synchronization overhead comes from the computation of Global Virtual Time. There are several GVT algorithms which do not have this synchronization overhead. Some are continuously calculated (Ghosh et al. 1994), some are asynchronously calculated (Fujimoto 1994, Xiao 1995).

We have developed a second version of Tempo; an efficient sequential simulator. It is a single processor discrete-event simulation framework, and is link-compatible with the parallel version of Tempo without the code or many of the overheads required for parallel operation. In particular, an LP does not save state, and events are handled by the local priority queue only. This version of Tempo is expected to be used during parallel replication runs, model development, and single processor execution of small scenarios.

5.3 Current Tempo Status

On a four processor SPARCStation-20, Model 514 (50MHz SuperSPARC, 1Mb secondary cache with MBus interconnection) the Tempo kernel displayed speed up as shown in Table 1. Speed up is the ratio of the execution time of a benchmark program linked with a sequential kernel divided by the execution time of the same benchmark linked with the parallel kernel. The independent variables are the synthetic workload and the number of CPUs used. The synthetic workload is an additional processing delay that represents the time an application LP would take to change its own state (per event). 999 LPs were used, with each LP randomly exchanging 10000 events with a uniformly distributed delay in the interval of (0,1]. The LP state size was 80 bytes, indicating very small state saving costs.

Table 1: Speedup of Parallel Execution Over Sequential Execution on the Sun SPARC Station-20/514

Synthetic Workload	Number of CPUs		
	1	2	4
0 uS	0.86	1.53	2.50
50 uS	0.92	1.77	3.27
100 uS	0.94	1.84	3.51
200 uS	0.97	1.90	3.69

Typical per event execution overheads for the sequential and parallel kernels are 12.0 and 27.8 microseconds respectively. Both versions of Tempo compare favorably to other existing simulation kernels. Sim++ (Baezner 1990) was measured as consuming per event overheads of 90 microseconds (sequential), and 1200 microseconds (parallel). In terms of other sequential simulation systems, Sim++ was found to perform similarly to five other sequential kernels, averaged across six test suites (Bensley 1991), thus by inference Tempo demonstrates above average performance. Further performance experimentation and analysis is ongoing.

The one CPU case never exhibits speed up over the sequential code because of the extra parallel overheads.

Even with a very small grained application (with sufficiently small state size), a multi-CPU run is always faster than the sequential run. In absolute terms, the sequential kernel executes about 26.3K events/second (with no synthetic workload). The parallel kernel executes 65.7K events/second on four CPUs.

On a sixteen processor Cray Research CS-6400 (60MHz SuperSPARC, 1Mb secondary cache with XDBus interconnection) the kernel displayed speed up as shown in Table 2.

Table 2: Speedup of Parallel Execution Over Sequential Execution on the Cray CS-6400

Synthetic Workload	Number of CPUs				
	1	2	4	8	15
0 uS	0.840	1.193	1.929	3.368	4.666
50 uS	0.925	1.627	2.785	5.481	8.661
100 uS	0.951	1.772	3.358	6.269	10.479
200 uS	0.984	1.871	3.572	6.902	11.955

Note that initial testing showed that 16 CPU runs returned significantly slower values than 15 CPU runs. This was identified as a OS problem, in that the benchmarks were run in multi-user mode with a variety of OS daemons running. At least one of the CPUs had to time share between the simulation kernel and other UNIX processes, reducing the total throughput of the kernel and disturbing simulation balance. The 16 CPU tests were thus eliminated from our test suite.

In absolute terms, the sequential kernel executes about 37.7K events/second (with no synthetic workload). The parallel kernel executes 175.8K events/second on 15 CPUs (141.6k on 16 CPUs).

6 CONCLUSIONS

We have proposed to use a parallel discrete event simulation kernel to address the end to end performance needs of strike planning systems, in particular for parameter searching and interactive planning. We have proposed using the Tempo parallel simulation kernel to meet this requirement and have shown its efficiency and applicability. We have addressed some performance issues by optimization at the application level using sectorization and dead reckoning. We have addressed kernel level performance using combined conservative time stepped and optimistic time warp synchronization. As shown in Tables 1 and 2, highly effective processor utilization is achieved in the traditionally difficult low-granularity cases.

7 FUTURE WORK

We will continue to investigate ways to improve the performance of parallel simulations such as strike planning. One way to further reduce the time warp overheads is to replace the standard state saving technique with an incremental approach. Only the variables that change will be saved in a back trace trail.

If one processor has more work assigned to it than any other, the simulation will run more slowly than it needs to, and the other processors will effectively have to wait for the overloaded processor to catch up. Dynamic load balancing promises to address this issue. When such a situation is detected, a logical process can be migrated to another less loaded processor. Issues of synchronization, caching and message forwarding must be addressed.

An additional tool to test for achieved parallelism will be created, where the simulation kernel is measured against the ideal case of a kernel with zero synchronization overheads. By measuring an application against the zero overhead kernel (with one processor per LP), we may establish the *intrinsic parallelism* of the application. By comparing the performance of the real simulation kernel against the zero overhead kernel, we may establish the *achieved parallelism*, i.e. the measure of how effective the simulation kernel was in extracting intrinsic parallelism.

ACKNOWLEDGMENTS

Tempo was developed under internal R&D funding by SAIC. Funding for the additional research presented here was provided by the Naval Research Laboratory. Access to the Cray-CS6400 was graciously provided by the manufacturer. We would like to acknowledge the contribution of the research teams of Dr. Richard Fujimoto of the Georgia Institute of Technology, and Dr. Brian Unger of the University of Calgary. Consultation with these groups has accelerated the development of Tempo.

REFERENCES

Baezner, D., G. Lomow, and B. Unger. 1990. Sim++: the Transition to Distributed Simulation. In the *Proceedings of the 1990 SCS Multiconference on Distributed Simulation.*

Beckman, B., et al. 1988. Distributed Simulation and Time Warp Part 1: Design of Colliding Pucks. In the *Proceedings of the SCS Multiconference on Distributed Simulation*, V 19, #3.

Bensley, E., V. Giddings, J. Leivent, and R. Watro. 1991. A Performance-based Comparison of Object-Oriented Simulation Tools. Technical Report, MITRE Corp.

Brown, R. 1988. Calendar Queues: A Fast *O(1)* Priority Queue Implementation for the Simulation Event Set Problem. *Communications of the ACM*, V 31, #10.

Das, S., R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. 1994. A Time Warp System for Shared

Memory Multiprocessors. In *Proceedings of the 1994 Winter Simulation Conference.*

DIS Steering Committee. 1994. The DIS Vision. Reference # IST-SP-94-01.

Fujimoto, R. M. 1990. Parallel Discrete Event Simulation. *Communications of the ACM* 33(10), pp. 30-53.

Fujimoto, R. M., and M. Hybinette. 1994. Computing Global Virtual Time in Shared-Memory Multiprocessors. Technical Report, GATech College of Computing.

Ghosh, K., K. Panesar, R., Fujimoto, and K. Schwan. 1994. PORTS: A Parallel, Optimistic, Real-Time Simulator. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation* .

Gilmer, J., D. O'Brien, and J. Payne. 1990. Toward Realtime Simulation: Prototyping of a Large Scale Parallel Ground Target Simulation. In *Proceedings of the 1990 Winter Simulation Conference.*

Jha, V., and R. Bagrodia. 1994. A Unified Framework for Conservative and Optimistic Distributed Simulation. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation.*

Jones, D. W. 1986. An Empirical Comparison of Priority-Queue and Event-Set Implementations. *Communications of the ACM*, V29, #4.

Lee, J., and P. Fishwick. 1995. Simulation-based Realtime Decision Making for Route Planning. To appear in the *Proceedings of the 1995 Winter Simulation Conference.*

Mellon, L. F. 1994. Sectorization -- Increasing the Parallel Performance of Simulations Containing Mobile, Sensing Entities. Technical Report, SAIC.

Moret, B., Z. Chen, A.T. Holle, J. Saia, and A. Bouroujerdi. 1995. Network Routing Models Applied to Aircraft Routing Problems. To appear in the *Proceedings of the 1995 Winter Simulation Conference.*

Pooch, U. N., and J. A. Wall. 1993. *Discrete Event Simulation (A Practical Approach).* CRC Press.

Van Hook D., J. Calvin, M. Newton, and D. Fusco. 1994. An Approach to DIS Scalability. *Proceedings of the 11th DIS Conference.*

Xiao, Z., J. Cleary, F. Gomes, and B. Unger. 1995. A Fast Asynchronous GVT Algorithm for Shared Memory Multiprocessor Architectures. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation.*

AUTHOR BIOGRAPHIES

DARRIN WEST is a senior computer scientist with Science Applications International Corporation. He received his M.Sc. degree from University of Calgary. His research interests include parallel simulation and distributed systems. Mr. West is currently principle investigator for the TEMPO internal R&D program, and is a lead architect for the Synthetic Theater of War program under ARPA funding. He is a member of the IEEE.

LARRY MELLON is a senior computer scientist and branch manager with Science Applications International Corporation. He received his B.Sc. degree from University of Calgary. His research interests include parallel simulation and distributed systems. Mr. Mellon is a lead architect for the ARPA funded Synthetic Theater of War project. He is the principle investigator for a parallel ATM simulation program, and a parallel simulation strike planning system.

JIM RAMSEY is a senior software engineer with Science Applications International Corporation. He received his M.Sc. degree from University of Maryland. His research interests include parallel simulation and distributed systems. Mr. Ramsey is the lead designer and implementor of the Tempo system.

JOHN CLEARY is a Reader in Computer Science, University of Waikato. He received his Ph.D. in EE from University of Canterbury, New Zealand. His research interests cover: distributed systems including parallel and distributed simulation, TimeWarp and applications of TimeWarp to other distributed problems; applications of complexity theory to machine learning and data compression; and logic programming. Dr. Cleary is a member of ACM, IEEE, NZCS (New Zealand Computer Society).

JIM HOFMANN is section head of the Decision Support and Force Level Planning project at the Naval Research Laboratory. He received his M.Sc. in Operations Research from George Washington University. His research interests include automated mission planning, data fusion and object oriented simulation.