Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

EMPLOYING PROXIES TO IMPROVE PARALLEL DISCRETE EVENT SIMULATION PERFORMANCE

David W. Mutschler

Naval Air Systems Command
22595 Saufley Rd, Bldg 3259
Patuxent River, MD 20670, USA

ABSTRACT

Proxies are caches of information maintained by one simulation object about other simulation objects. Though proxies can require significant overhead to maintain consistency, their judicious use can improve parallel performance by increasing speedup. This paper discusses three cases where careful use of proxies has improved speedup in a parallel discrete event simulator implemented using threaded worker pools.

1 INTRODUCTION – JOINT INTEGRATED MISSION MODEL (JIMM)

The Joint Integrated Mission Model (JIMM) is a general purpose, mission level, legacy discrete event simulator (JIMM Model Management Office 2008) currently used as a threat environment generator by the Air Combat Environment Test and Evaluation Facility (ACETEF) and previously used for requirements analysis and generation by the Joint Strike Fighter (JSF) program. In JIMM, aggregate or individual players such as planes, tanks, command centers, and missiles are comprised of programmable component systems. There are six basic types of systems: thinkers, sensors, disruptors (jammers), weapons, movers, and communicators. Thinkers are programmed by JIMM users to employ plans and react to knowledge (perceptions) of other players obtained through sensors and communications (messages) with other players, or known scenario start. JIMM also simulates communication nets, command chains, and other simulation entities.

JIMM was modified to exploit multiple processors to improve performance through the Common High Performance Computing (HPC) Software Support Initiative (CHSSI) as the seventh project under the Forces Modeling and Simulation (FMS #7) Computation Technology Area (CTA) (Mutschler 2005). JIMM was designed to operate within a symmetric multiprocessor (SMP) computer architecture. It employs a single event queue from which multiple threads (a worker pool) within a single process obtain events to execute in parallel optimistically. The event queue is kept in a critical region wherein serial access to the event queue is ensured.

Objects that may be accessed in parallel are known as "simulation" objects. These include players, command chains and communication nets as well as internal simulation maintenance objects such as the proximity (address) tree. There is no explicit assignment of simulation objects to specific threads. Also, threads do not exchange messages and hence, information transferred from one simulation object to another occurs solely via events in which both objects are accessed.

All simulation objects accessed by an event are identified when that event is created. Hence, a thread should access a simulation object's state only when such access is previously noted for the event. Within the critical region, this identification is used to determine potential collisions between events employing the same simulation object. Collisions are resolved where a later event must wait for an earlier event to complete or a later event (in simulation time) that is either executing or queued to be output when "safe" must be rolled back because an earlier event must be executed. Simulation time (also known as global

virtual time (GTV)) is also updated when safe output is determined. Lastly, event output is independent of the number of processors. This total ordering is ensured through the critical region.

To reduce the impact of the critical region as a potential bottleneck, JIMM contains optimizations to reduce processing within the critical region. For example, processing of safe events conducted in the worker threads includes lazy update of state changes (Mutschler 2008) and actual transmission of output. This is depicted in Figure 1.

Also, all events have a main simulation object (known as the "subject player"). Since these events would collide, the events are ordered by simulated time in a "player based event list" (PBEL) associated with the simulation object and then, these lists are ordered within the main JIMM event queue by the simulation time of their earliest events. Furthermore, to avoid redundant processing after a collision is detected, the later event (in simulation time) is placed in a "pending after queue" associated with the main simulation object of the earlier event. The events in the pending after queue are scheduled again only after the earlier event completes.

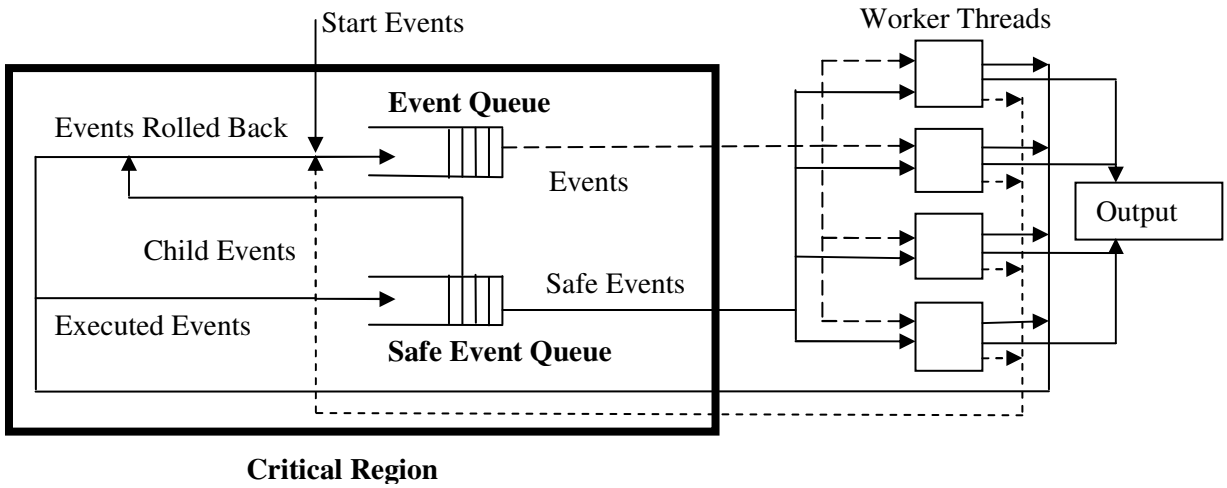

Figure 1: Event and Output Flow in JIMM

A major optimization is the use of proxies (Steinman 1998). A proxy is a cache of information maintained by one simulation object about another simulation object. Thereby, instead of associating the proxied object with an event and increasing the probability of collision, the proxy may be employed instead. Proxies incur significant overhead. In addition to space consumed and additional programming complexity, each time information is updated, events with the same simulation time (i.e. with no elapsed simulation time) must be generated for all the other simulation objects which maintain proxies. This overhead impacts serial processing time. However, this should be offset by better parallel performance.

2 TEST ENVIRONMENT

A symmetric multiprocessor (SMP) platform was not available for this study. A single personal computing (PC) platform was used instead. Fortunately, JIMM has a user input whereby a worker thread can attempt to obtain a set number of events to execute in serial. The earlier serial events can still collide with later events and collision prevention still occurs. The option allows a tradeoff of potential parallelism against the overhead of accessing the critical region. However, the option can also be leveraged to estimate speedup.

2.1 Effective Speedup as Measure of Effectiveness

In this study, it is assumed that increasing the number of events executed by one thread at a time shows the same trends in collisions and rollbacks to be expected when multiple threads execute one event at a time. The number of event rollbacks (also known as "discards") is measured. Effective speedup can be calculated as the quantity of the number of events executed subtracted by the number of event rollbacks all divided by number of times the worker thread obtains events from the event queue (events executed per thread execution).

Also, since all events have output, output is provided only if the event will not be rolled back, and since event output would be handled by the worker thread when it completes executing its set of events, the effect speedup is corroborated by the average number of events for which output is provided each time output occurs (also measured in events per thread execution).

$$EffectiveSpeedup \approx \frac{\#EventExecutions - \#EventRollbacks}{\#ThreadExecutions} = \frac{\#Events\,w.Output}{\#ThreadExecutions}$$

2.2 JIMM 3.2.0 and the Final Battle Test Scenario

This work was based on JIMM version 3.2.0. The JIMM distribution includes a number of scenarios used for both example and test. In this study, the "Final Battle" scenario was employed. This scenario contains a large and comprehensive set of simulation objects and exercises most of the simulator's features (JIMM Model Management Office 2008). It simulates a regional conflict where three opposing sides employ a total of nearly five hundred planes, tanks, ships, missiles, command centers, bridges, engineers, and other players of different types. Execution speed of the "Final Battle" scenario has been used since the beginning of JIMM as a measure of quality by comparing current scenario execution speed against the execution speeds of previous versions.

Many events in JIMM are based on operation of component systems. These include "thinking" events where players determine courses of action given perceived information, "sense" events for sensor operations, "talk" events where one player communicates information to another over a communication network, and other events for shooting, jamming, mode changes etc. The remaining large category is "address tree" events for maintenance of a single simulation object that optimizes proximity detection performance. A count of percentage of events for the final battle scenario is provided in Table 2.

Proxies were used in the initial transformation of JIMM to this architecture (Mutschler 2005). Counts for events specific to establishing and maintaining communication net proxies and other proxies (including players) are also provided in Table 2.

Table 2: Event Types in JIMM 3.2.0 (Final Battle Scenario)

Event Type	Simulation Objects	Count	**Percentage**
Talk	Sender & Receiver Players & Communication Net	33500	**1.8%**
Address Tree	Address Tree & Subject Player	18100	**1.0%**
Sense	Sensing & Target Players	260200	**13.8%**
Think	Subject Player Only	1384400	**73.2%**
Other (Shoot, Change Mode etc.)	Varied	11200	**0.6%**
Communication Net Proxies	Subject Player Only	143000	**7.5%**
Other Proxies	Subject Player Only	40200	**2.1%**
Total		**1890600**	**100%**

Lastly, within the JIMM conceptual model, a single player may contain one or more platforms where each platform may have a different location. Thinking occurs at the player level and multiple platforms may be considered when courses of action are determined. Normally however, a player only has one platform. This is assumed when a player is stated to have a location.

3 GENERAL JIMM CHANGES

Despite early work to ensure parallel execution of thinking events, effective speedup for the Final Battle scenario for JIMM 3.2.0 was about 2.1. The following changes to JIMM together with the speedup improvement efforts achieve an effective speedup of nearly 2.5.

3.1 Addition of Continuation Numbers to Event Time

Initially in JIMM 3.2.0, the ordering of events was determined by simulation time and a unique integer event identifier known as a "second key" assigned when the event could safely be placed on their main (subject) player event list (PBEL). Events with earlier simulation times are executed first. Should the simulation times of two events be equal, then the event with the lower "second key" is executed earlier.

As part of this effort, a third integer known as the "continuation number" was added to the simulation time. By default, second keys are automatically assigned. However, this assignment may be overridden if continuation numbers are used. Should the second keys of two events be the same, then the event with the lower continuation number is executed first. By default, the value of a continuation number is zero (0). Events with the same second key will have the same simulation time but must have unique continuation numbers. This ensures proper event execution order. Furthermore, to ensure proper event ordering, events with continuation numbers should not create events that in turn, also need continuation numbers. The determination of continuation numbers by these other events and hence, the ordering of their child events, would not be readily predictable.

Continuation numbers permit greater control over event ordering. For example, when an event to update a proxy is generated, it occurs at the same simulation time as the event generating it. However, it is possible that a different event at the same simulation time could still have a second key value between the originating (parent) event and the created (child) events. Should this event need the proxy information, it would access old and incorrect information. With the continuation number, the second key of the proxy updating event would be explicitly set to the second key of the originating event and the continuation number set (usually via an incrementing counter) for each event to ensure overall event time uniqueness.

Continuation numbers can also be used to improve parallel performance. For example, in cases where a simulation player may only be involved the latter portion of an event, the event could be split. The simulation player could then only be associated with the second event and the continuation number used to maintain event ordering. Furthermore, should the simulation player not be needed for the latter half of the event execution, the additional event need not be created.

3.2 Considering Associate Players in Events

In the JIMM conceptual model, outputs known as "incidents" involved at most three players known as the "subject player", "object player", and "indirect object player" (JIMM Model Management Office 2008). This construct is similar to English grammar. In the JIMM implementation, this construct was also carried forward into the base event class. In the initial parallel implementation of JIMM, this limit was retained and leveraged to improve parallel performance (Mutschler 2005). However, as JIMM evolved toward greater functionality, complexity, and verisimilitude, some JIMM events needed to access the simulation state contained within more than just three different player objects. When the additional players are readily identifiable and the interaction limited, proxies may be suitable. However, in cases where this identification is not so ready, the three player limit becomes an impediment.

The identification of more than three players with events was added to JIMM 3.2.0 in this effort. In these modifications, the additional players are known as "associate players". Events were modified to

employ them and collision detection algorithms were modified to consider them. Two additional functions were also added to the base event class: ConfirmAssocPlayers() and EstablishAssocPlayers().

The EstablishAssocPlayers() function identifies associated players needed for an event and places them on the event's associate player list. This ensures those players are considered in event collision detection. The function is called when an event is first created.

However, it is possible that additional players not initially considered may be needed for an event execution. This might occur if a platform moves within appropriate range or if a new player is created. The CheckAssocPlayers() routine ensures that all players needed for an event execution are associated with the event with regard to collision detection. If so, it returns one (1) and if not, it returns zero (0). It is executed at the beginning of the event before simulation state within other players is accessed. If other associate players are needed (i.e. zero is returned), then the same event is created again with the same input, simulation time, and second key but now employing a non-zero continuation number. The EstablishAssocPlayers() routine is executed again to reestablish the correct list of associate players for the new event.

4 SPEEDUP IMPROVEMENT EFFORTS

4.1 Proxies for Cooperative Perceptions

In thinking events, JIMM players may change their course of action based upon information about other players from the simulation. This information is collectively known as "perceptions". In JIMM, there are two types of perceptions: cooperative (also known as "perceived friends") and non-cooperative (JIMM Model Management Office 2008). A cooperative perception assumes perfect knowledge of another player. This occurs among immediate peers, subordinates, and commanders within a command chain. For non-cooperative perceptions (also known as "perceived platforms"), perfect knowledge does not exist and any information must be accumulated via sensing and communication. This information is maintained by the thinking (perceiving) player and no separate proxy is necessary.

However, a proxy is necessary for cooperative perceptions because before the initial parallel implementation, that player simulation state was accessed directly. Therefore, to allow parallel execution of thinking events, proxies for cooperative perceptions were created and maintained. This allowed significant speedup for scenarios with a large percentage of thinking events (Mutschler 2005).

However, these proxies consume a significant amount of memory. Additional events are needed to ensure consistency. The proxies are also complex and troublesome for programmers to maintain. To reduce the simulation impact, extensive filters were added to ensure that only that simulation state which need be accessed was actually maintained within the proxy. This did significantly reduce the simulation overhead but added additional complexity.

Use of proxies for cooperative perceptions (perceived friends) permitted the thinking events to execute within the three player limit of the base event class. Only simulation state within the "thinking" (or "subject") player (including the proxies) was accessed. Even so, it is possible that the overhead of proxy maintenance might be greater than the loss of parallel execution from the equivalent use of commanders, peers, and subordinates as associate players. These players are already identified within a pre-existing proxy of the player's command chain which can be readily used to create an associate players list.

In this investigation to determine the efficacy of the perceived friend proxies, thinking events in JIMM 3.2.0 were modified to employ perceived friends as associate players. Events maintaining the proxies were also removed. The results of the efforts are shown in figure 2, where the loss of estimated speedup for ten events per execution moved from 2.10 to 1.95.

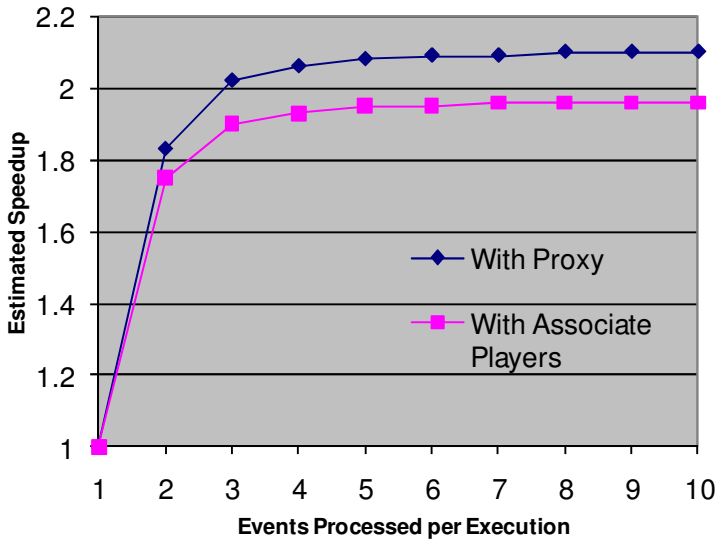

Figure 2: Loss of estimated speedup for ten events per execution moved from 2.10 with proxy to 1.95 with associated players

Overall, the loss of parallel performance is significant. Moreover, while memory utilization was reduced, the impact on serial performance was not statistically significant. Therefore, in this case, the use of perceived friend proxies should be maintained over employment of associate players.

4.2 Jamming

In JIMM, jamming can be modeled as an area of effect and the ability of a jammer to affect communication or sensing is determined partly by the distance of the jammer player platform as well as player-specific jammer system characteristics (JIMM Model Management Office 2008). Jamming was not considered in the initial implementation of parallel operation. However, when parallel operation was expanded, jamming did not fit within the three player limitation.

Instead of associate players however, the programmer placed proxies of the jammer and jammer player platform paths (used for location determination) within the address tree object itself. This required that the address tree now become a simulation object for all sensing and communication (talk) events. This is not an effective solution. To ensure appropriate (total) ordering, all events with the same simulation object must execute in the same serial order and effective parallel execution of those events is not possible.

As part of this effort, to eliminate use of proxies in the address tree and the association of the address tree simulation object with sensing and communication events, a list of simple proxies identifying jamming players were added to each of the affected player platforms. Jammer area of effect is still maintained within the address tree. During address tree maintenance events, events to initially form the simple proxy are created when a player potentially moves within that area of effect. Also, in cases where the jammer player platform moves or the jammer system is turned on or off, events to establish or update the proxy are created for each of the potentially affected systems.

The proxies are then used by the EstablishAssocPlayers() routine to associate the jamming players with any pertinent sensing or communication (talk) events. Since the proxy list could be updated after the event is created, the CheckAssocPlayers() function is employed to ensure collision free operation and EstablishAssocPlayers() is employed again if the event must be recreated with a new continuation number. The initial results are below. The results are showed in figure 3.

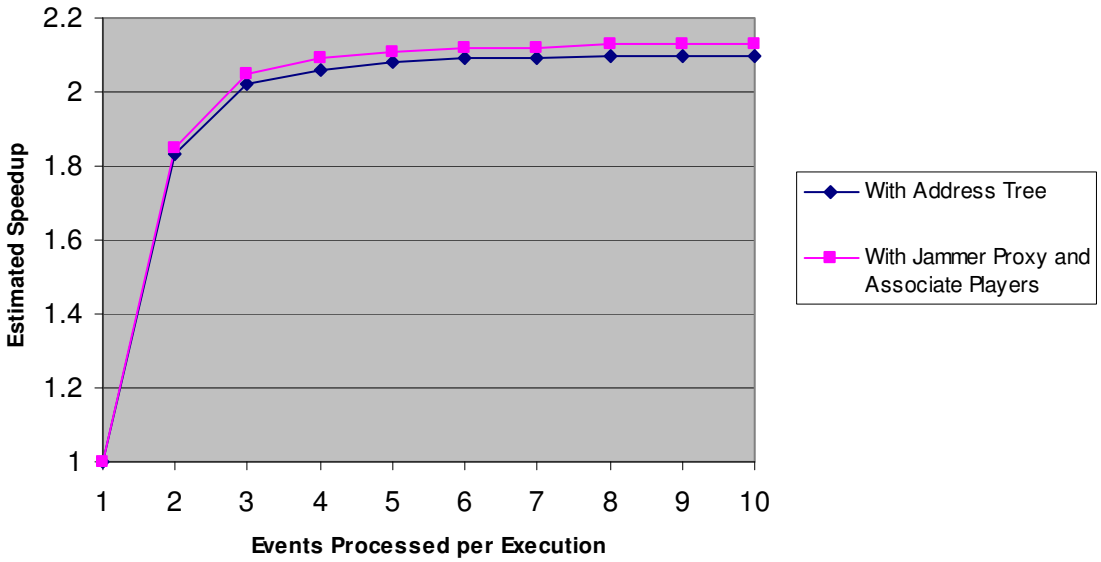

Figure 3: Estimated speedup for ten events per execution moved from 2.10 with address tree to 2.13 with jammer proxy and associated players

Though estimated speedup did improve, the increase in performance was not great. However, not all communication receiver systems or sensing systems are affected by the same jammers. Therefore a filter was added to ensure that jammer proxies were only created for those platforms with systems that could be affected by those jammers. In turn, this reduced the number of players that would have been associated with the sensing or communication events. The effect of the filter is shown in figure 4.

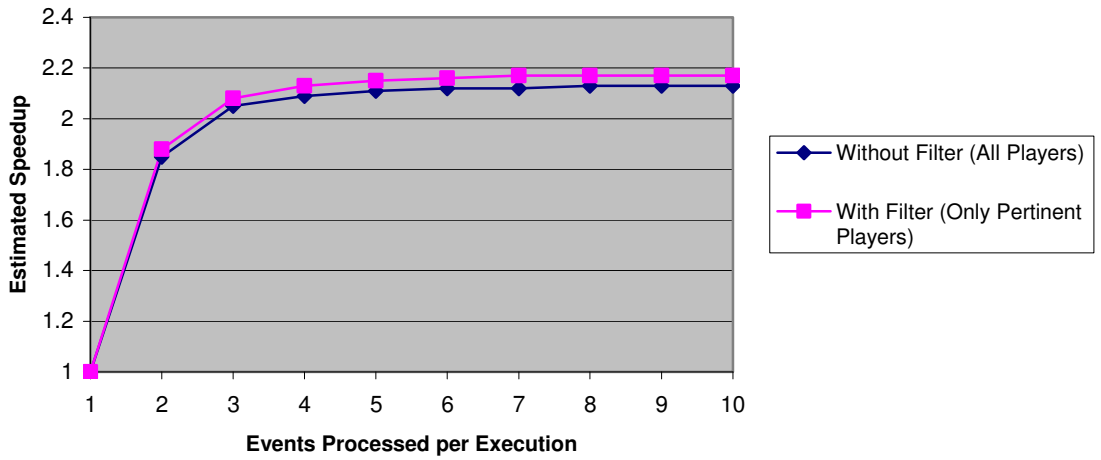

Figure 4: Loss of estimated speedup for ten events with and without filter

The use of a filter to restrict associated players to those with a more certain effect for jamming more than doubled the improvement in estimated speedup. For ten events per execution, the estimated speedup moved from 2.10 to 2.17 vice 2.13. Use of associated players to eliminate the need for the serializing association of the address tree for communication and sensing events did improve estimated speedup. This speedup was further improved when the associated players were further restricted to include only those players with potential effects to jamming.

4.3 Correct Erroneous Event Association of Communication Network Simulation Object

As a side effect of the later analysis from this work, a major error was identified. Its correction greatly increased estimated speedup. The error was the erroneous association of the communication net with events to update information within communication net proxies. Instead of executing in parallel, the error caused all the proxy maintenance events for the communication net to be executed serially because they the same simulation object was identified for each event. As shown in Table 1 above, the number of these events was large (7.5% of the total event count). The increase in estimated speedup is shown in figure 5.

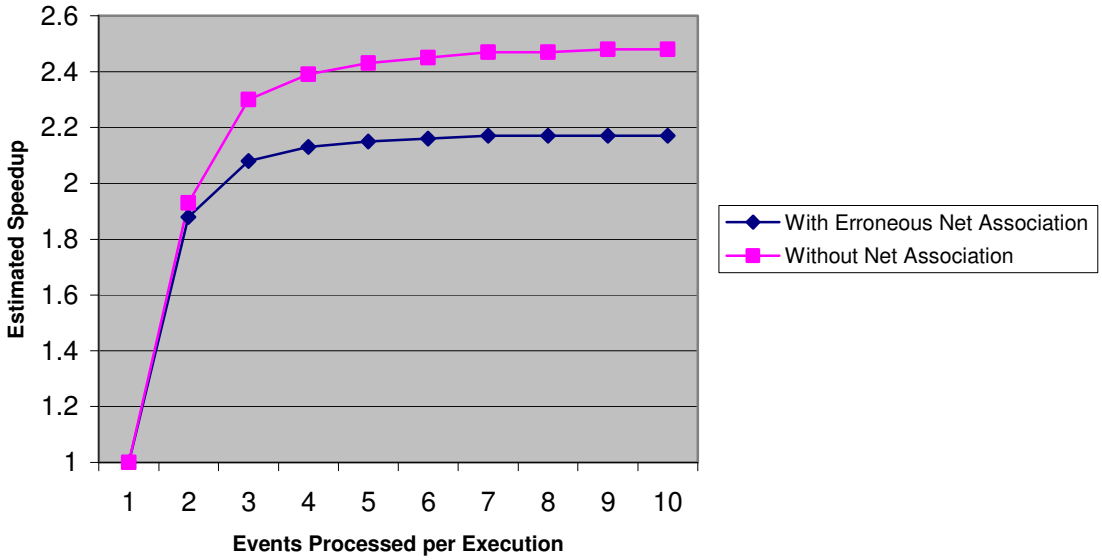

Figure 5: Loss of estimated speedup for ten events with and without net association

Estimated speedup for ten events processed per execution moved from 2.17 to 2.48. This is a very significant increase that further shows the importance of restricting the number of simulation objects associated with an event.

5 FUTURE WORK

Significant future work still exists to improve speedup for parallel execution in JIMM. First, in this modification, parallel execution does not consider opaque shapes that may move or be destroyed over the course of a simulation run. The technique employed for jamming could be extended to provide parallel execution in this case.

The simple jammer proxies used to identify associate players could be expanded to include all necessary information such as player platform movement paths and jammer system settings. This would increase the complexity of the simulation in that additional events would be necessary to maintain the proxies should jammer system settings change. This may also necessitate maintenance of a list of potentially affected player platforms within each jammer system object. This is currently not needed. Even so, like the case with cooperative perceptions, the increased complexity and overhead may be worth the additional improvement in parallel operation.

In theory, proxies could be further extended to eliminate the need for target players in sensing events. However, the effect of other players upon the target players can impact the outcome of a sensing event (JIMM Model Management Office 2008). For example, if a jammer is emitting toward a third player, that third player's position affects the orientation of the jammer's antenna. If a sensor is attempting to detect

this signal, this orientation would affect the outcome of that sensing event. This association of players with players with players may be overly difficult to process.

A further set of future work projects is to expand or add proxies for other cases. For example, a large portion of events associated with the address tree are to determine the simulation time for an initial sensing event when a platform or sensor movement results in that platform potentially being within sensor range. If associated sensing events already exist, then that check would be redundant. Use of a simple proxy lists already reduces the number of these events but could be expanded to reduce them further.

Lastly, a major restriction with the JIMM implementation of parallel execution is that only a single set of outputs or state changes may be queued for each simulation object. It is possible that multiple events employing the same simulation player may be executed safely where multiple outputs would be queued. This restriction prevents this possible speedup from being leveraged. Adapting methods for time management of proxies (Steinman 1998) is a possible approach.

6 CONCLUSION

This work shows that use of proxies can improve parallel performance. A major benefit of proxies is to eliminate the association of simulation objects with events. This allows those events to execute in parallel. When filters are applied to restrict the number of proxies and hence, the number of associated players, parallel performance can be further improved.

However, use of proxies must be judicious. In one case in JIMM, association of jammer proxies with the address time greatly impacted parallel performance by forcing serial execution of sense and communication events. Instead, associating the jammer proxies with affected players is a better solution because sense and communication events for players that are not interacting (e.g. when geographically separated) can execute in parallel.

ACKNOWLEDGMENTS

The Joint Integrated Mission Model (JIMM) is owned by the U.S. Navy and maintained by the JIMM Model Management Office (JMMO). JIMM is based on concepts developed by Peter Lattimore. The current JIMM Model Manager is Dr. Michael D. Chapman. Dr. Chapman reviewed this paper and provided useful comments. Though this work was done independently, per agreement, all source code associated with this change has been provided to the JMMO for potential incorporation into the baseline model. All inquires about JIMM should be submitted in writing to the JIMM Model Manager at 48150 Shaw Rd, Bldg 2109, Patuxent River, MD 20760 or via e-mail at <jmmo@navy.mil>.

REFERENCES

JIMM Model Management Office. 2008. JIMM 3.2 Users Guide Volume One. Available from the JIMM Model Management Office (JMMO) via <jmmo@navy.mil>.

Mutschler, D.W. 2008. Deferred vs. Immediate Modification of Simulation State in a Parallel Discrete Event Simulation Using Threaded Worker Pools. In *Proceedings of the 2008 Winter Simulation Conference*, eds. S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mutschler, D.W. 2006. Enhancement of Memory Pools Toward a Multi-Threaded Implementation of the Joint Integrated Mission Model (JIMM). In *Proceedings of the 2006 Winter Simulation Conference*, eds. L. R. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mutschler, D.W. 2005. Parallelization of the Joint Integrated Mission Model (JIMM) Using Cautious Optimistic Control. *Proceedings of the 2005 Summer Computer Simulation Conference*. Cherry Hill NJ, Society for Modeling and Simulation International, (pg. 145-152).

Steinman J. 1998. Time Managed Object Proxies in SPEEDES. *Proceedings of the Object Oriented Simulation Conference (OOS '98)*, (pg. 59-65).

AUTHOR BIOGRAPHY

DAVID W. MUTSCHLER has been employed by the Naval Air Systems Command (NAVAIR) since 1985. He obtained his doctorate in Computer and Information Science from Temple University in 1998. He worked on JIMM and its predecessor, the Simulated Warfare Environment Generator (SWEG), from 1996 to 2007. He served as principle investigator for the JIMM parallelization effort from 2000 to 2003 and later served as the JIMM Model Manager from July 2004 to Feb 2006. He is currently serving as the Government Software Integrated Product Team (IPT) Lead for the CH-53K Heavy Lift Replacement (HLR) rotorcraft. He is a member of the Association for Computing Machinery (ACM), its Special Interest Group in Simulation (ACM SIGSIM) as well as the Institute for Electrical and Electronics Engineers (IEEE) Computer Society (IEEE/CS). He is also an Associate Professor of Computer Science at the Florida Institute of Technology (FIT) School of Business – Patuxent River MD site. His e-mail address is <david.mutschler@navy.mil>.