IMPLEMENTING SYNTHETIC FORCES SOFTWARE ON MASSIVELY PARALLEL PROCESSORS

Jeffrey M. Opper, William P. Niedringhaus, Brian R. Winner
The MITRE Corporation
McLean, Virginia

ABSTRACT

The DOD community has adopted Distributed Interactive Simulation (DIS) as a base for an evolving suite of standards to support virtual battlefield representation. At present, DIS-based applications have focused on relatively small-scale exercises involving less than 1000 manned simulators and computer-generated forces (CGF). Current estimates of 10,000 to 100,000 entities to support simulation of theater-wide operations may exceed the capabilities of existing computational hosts and interconnection networks. This paper presents strategies for using massively parallel processors to simulate large numbers of synthetic forces using a contemporary synthetic forces software system (ModSAF). Alternative functional decompositions of the software are presented which map to specific parallel programming paradigms. Factors are identified which constrain candidate implementation paths. Partitioning and filtering techniques are discussed which can be used to reduce or eliminate broadcast packet distribution in a message-passing system. Data distribution, partitioning, and locking techniques are presented to support use of private, near-shared, and globally-shared memory on a true shared-memory system. Test implementations of a parallel ModSAF designed to run on the Convex Exemplar and Cray T3D MPP systems are described and benchmark results for specific tests are presented.

ABOUT THE AUTHORS

Jeffrey M. Opper is a senior member of the technical staff in the Advanced Information Technologies Center of the MITRE Corporation and is the Principal Investigator for the Large Scale Simulation of Synthetic Forces project. He has been involved in the development of parallel and distributed simulations and interoperability protocols for five years most recently as a member of the Aggregate Level Simulation Protocol software development team. He began his career in the U. S. Naval Security Group and served from 1978 to 1988. Mr. Opper received a Bachelor of Science degree in Occupational Education from Southern Illinois University in 1985 and a Master of Science degree in Computer Science from Rensselaer Polytechnic Institute in 1988.

William P. Niedringhaus, born in Pittsburgh, PA in 1949, received the B.S. degree in Mathematics from Yale University (New Haven, CT) in 1971, and the Ph.D. degree in Electrical Engineering and Computer Science from Princeton University (Princeton, NJ) in 1980. He has worked at The MITRE Corporation in McLean, VA, since November 1979, on algorithms for automated airborne collision avoidance and automated ground based air traffic control, as a Group Leader (1983-85) and as a Lead Engineer (since 1986). In 1990 he joined MITRE's Modeling and Simulation Technical Center. He has had extensive experience in simulation design, object-oriented design, and parallelization of simulations. He is a senior member of AIAA.

Brian R. Winner is a senior member of the technical staff in the Advanced Information Technologies Center of the MITRE Corporation. He is currently involved in research assessing the applicability of massively-parallel processor systems to the simulation of computer-generated battlefield entities in theater-wide exercises. He has over ten years of software development experience in a wide variety of application areas. Mr. Winner holds a Bachelor of Science degree in Mathematics from Towson State University and a Master of Science degree in Computer Science from The Johns Hopkins University.

IMPLEMENTING SYNTHETIC FORCES SOFTWARE ON MASSIVELY PARALLEL PROCESSORS

Jeffrey M. Opper, William P. Niedringhaus, Brian R. Winner
The MITRE Corporation
McLean, Virginia

INTRODUCTION

Simulation is becoming an increasingly important component of the DOD technology base. In particular, Distributed Interactive Simulation (DIS) is being used more frequently to support simulated environments for training and concept evaluation. While DIS has demonstrated its utility in small scale applications, potential uses require scaling the technology to support substantially larger scenarios involving 10,000 to 100,000 entities.

Using DIS to support large scale exercises puts demands on both network and computational resources. The underlying architecture of DIS is based on broadcast of shared ground truth data to distributed simulations or simulators in real-time at rates sufficient to support battlefield visualization. While efforts are underway to broaden the acceptable high level architecture for DOD simulation applications, current DOD policy requires most simulation applications to operate within the current DIS constraints. In this context, most scaleability research has focused on network technology. Initial efforts addressed increasing network capacity through the use of mechanisms such as multicasting and packet compression. Until recently, little attention had been devoted to the computational resources required to sustain real-time simulation.

This research project seeks to assess whether high performance computing and communications systems can provide the resource base needed to support the current DIS system architecture in large-scale exercises. We are currently in the process of adapting a representative synthetic forces simulation to execute on several massively parallel processors (MPP). Our preliminary performance results indicate significant that gains can be achieved in entity loading by exploiting the unique capabilities that these architectures provide.

BACKGROUND

The majority of the battlefield entities represented in large-scale DIS-based exercises will be synthetic or semi-automated forces (SAF)—combat entities that are represented through simulation as opposed to man-in-the-loop networked simulators. Modular Semi-Automated Forces (ModSAF) is the software most commonly used today to represent synthetic forces in a DIS environment and was chosen for our research.

ModSAF is an entity-based, time-stepped simulation that can be replicated across one or more computational hosts depending upon the size of the exercise or location of the users. ModSAF consists of a user interface component, or SAFstation; the simulation component, or SAFsim; and a message logger, or SAFlogger. These components can also be physically distributed depending upon the exercise configuration. [1]

SCALEABILITY

Large scale exercises present daunting problems for DIS-based synthetic forces simulations. Three primary factors limit the potential scaleability of simulations such as ModSAF:

- The performance of the host processor(s)
- Network congestion when presented with high-volume broadcast traffic
- Limited ability to effectively control the evolution of a large scenario

Two programs have recently materialized that offer promising insights on extending the capability of a distributed ModSAF to support large-scale exercises.

Network-Based Scaleability Approaches

The Advanced Research Projects Agency (ARPA) Real-Time Information Transfer and Networking (RITN) program was formed to address network-based approaches to the management of DIS network traffic. In response to some of the issues addressed above, an Application Gateway (AG) prototype was fielded for the Synthetic Theater of War-Europe (STOW-E) demonstration. In the demonstration, the AG resided on a workstation host and acted as a gateway between the local simulations on each local-area network (LAN) and the wide-area network (WAN). See Figure 1. Multicasting, packet compression, quiescent entity state propagation, and other techniques were used to manage the flow of entity state data to and from the WAN as described in "An Approach to DIS Scaleability."[2]

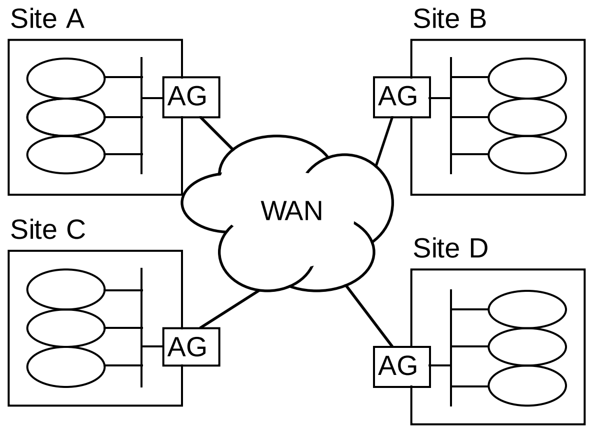

Figure 1. Application Gateway Utilization

Command Forces

Human factor issues related to interaction with large-scale exercises are being addressed in the Command Forces (CFOR) program [3,4,5]. Prior to the introduction of CFOR, battlefield entities were provided tasking directly by the human controller at each SAFstation. CFOR incorporates explicit modeling of battlefield command and control (C2) into virtual simulations. As illustrated in Figure 2, CFOR extends the DIS architecture through the introduction of Command Entities (CEs), which are software representations of the battlefield commander. The Command and Control Simulation Interface Language (CCSIL) is used for the exchange of information between CEs. Through the use of CFOR, standard operational orders can be provided at the battalion level by the human controller and reports can be received from the CEs.

GOAL AND OBJECTIVES

The goals of this project are to exploit the computational and communications capabilities of existing MPP architectures and leverage the results of the RITN and CFOR efforts to consolidate the execution of a large scale synthetic forces simulation to one or more parallel hosts. By achieving these goals, we hope to provide a run-time infrastructure (RTI) for ModSAF that will support exercises involving more than 10,000 entities.

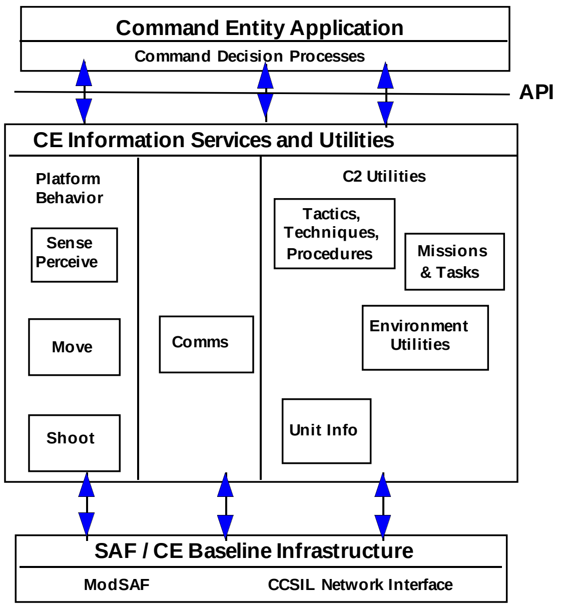
Command Entity Application
Command Decision Processes
API
CE Information Services and Utilities
Platform Behavior
Sense Perceive
Move
Shoot
Comms
C2 Utilities
Tactics, Techniques, Procedures
Missions & Tasks
Environment Utilities
Unit Info
SAF / CE Baseline Infrastructure
ModSAF
CCSIL Network Interface

Figure 2. Command Entity Technical Reference Model

In accomplishing these goals, we first must develop viable strategies for the modification of ModSAF to support selected parallel models of computation and conduct test bed experimentation on several MPP platforms to validate each approach. Once viability for one or more implementation paths is determined, we can then progress to adapting the parallel ModSAF RTI to support CFOR capabilities.

IMPLEMENTATION STRATEGIES

Three basic paradigms exist for the mapping of sequential codes to a parallel computer:

- Data Parallel
- Message-Passing
- Shared-memory

While a data parallel approach was employed successfully in the Massively Parallel Warfare Simulation (MasPaWS) [6], it was not considered here for several reasons. Data parallelism accommodates those codes having large, static data arrays that can be partitioned across processing elements within the MPP. Execution occurs in a lock-step fashion as each PE works within it's own data space. ModSAF relies on the use of dynamically allocated data structures to store simulation state data. The storage and access mechanisms used for these structures are optimized for the expected usage (e.g., linked list, binary or quad tree) and would require significant redesign to conform to a static data model.

The message-passing and shared-memory paradigms do not share these limitations and were considered for implementation and testing. The following sections describe the message-passing and shared-memory ModSAF implementations and present preliminary performance test results.

MESSAGE-PASSING MODSAF

Through the ARPA High Performance Computing (HPC) initiative, we were provided access to a 128-processor CRAY T3D located at Eglin AFB for the implementation of a message-passing parallel ModSAF.

The CRAY T3D System Architecture

The CRAY T3D is a multiple instruction multiple data (MIMD) parallel processor containing 32, 64, 128, 256, 512, 1024, or 2048 processing elements depending upon the system configuration. Memory in the T3D is physically distributed and globally addressable. The CRAY T3D connects to a host computer system that provides support for applications running on the CRAY T3D. All applications are compiled on the host system but execute on the CRAY T3D system. Suitable host systems include the CRAY Y-MP E series, the CRAY Y-MP M90 series, and the CRAY C90 series computer systems.

The CRAY T3D system is comprised of processing elements (PE), the interconnect network, and input/output (I/O) gateways. The PEs provide the system's computational resources. Each PE is comprised of a 150-MHz DECchip 21064 microprocessor (commonly known as the Alpha) accompanied by a local memory. The microprocessor is a 64-bit reduced instruction set computer (RISC). The interconnect network provides communication paths among the PEs and the I/O gateways. The topology of the interconnect network is a three-dimensional torus, which forms a three-dimensional matrix of paths connecting the PEs. The nodes of the interconnect network are physically interleaved. This serves to minimize the maximum wiring distance between nodes. I/O gateways transfer system data and control information between the host system and the CRAY T3D system. A master clock provides a 6.67 nanosecond clock signal, which is fanned out to all PEs and I/O gateways in the system.

Approach

We chose to adopt a host/node model for the message-passing implementation of ModSAF. In this model a user program is implemented on the host that spawns slave processes to the individual PEs. As illustrated in Figure 3, our approach uses the host process as both a SAF/CE gateway for interaction with the distributed exercise components and as an I/O server for the slave PEs. Although the programs downloaded to the PEs are identical, each may follow a distinctly different flow of control depending upon the data being processed.

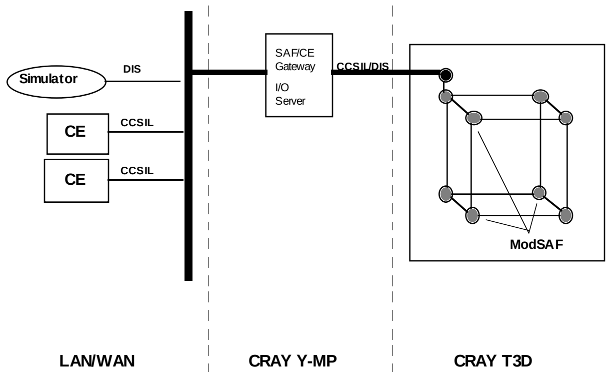

Figure 3. Message-Passing Implementation

In this approach, each PE acts as a single ModSAF host with the MPPs interconnection network mimicking the WAN and facilitating Protocol Data Unit (PDU) delivery. The effort required to parallelize ModSAF using this approach is minimal. To support inter-PE data distribution, a robust inter-PE packet distribution capability utilizing the MPPs interconnection network must be implemented. The remainder of the effort involves porting the applicable ModSAF code for execution on the microprocessor contained in the PEs of the MPP. It may also be necessary to implement synchronization support for the ModSAF scheduler if global synchronous operations are required.

Although such an approach may provide some gains in entity loading purely from the use of the low-latency, high-bandwidth interconnection network within the MPP alternative strategies for event distribution and packet filtering should be considered to optimize performance. We chose to augment the interprocessor communications capabilities of the message-passing implementation by incorporating PDU multicasting, relevance filtering, and packet bundling techniques developed for the RITN AG.

To ameliorate the effects of load imbalance, we embraced an approach similar to that considered for a message-passing parallelization of CORBAN [7] where each PE is assigned a terrain parcel and simulates entities on or directly over the parcel. A difusive, dynamic load balancing algorithm, described in "Diffusive Dynamic Load Balancing by Terrain Parcel Swaps for Communicating Vehicles" [8], will be considered to enforce the regularity of the terrain area and minimize extraneous communication.

Implementation

Prototyping of the message-passing software began in February 1995. We implemented inter-PE packet distribution services using a Cray Research supported version of Parallel Virtual Machine (PVM) [9] for the CRAY T3D. PVM provides a portable Application Programming Interface (API) for message-passing applications on a variety of uniprocessor and multiprocessor systems. This enabled a quick and efficient implementation that exploits the hardware capabilities of the CRAY T3D system to handle communication between PEs without resorting to a vendor-specific, proprietary interface.

The PVM-based inter-PE packet distribution capability was successfully integrated into the ModSAF libraries responsible for providing distributed communication services and required no changes to ModSAF behavioral software in order to be used.

Porting the remainder of the core simulation services libraries required minor modifications to the ModSAF software. These modifications were largely to software constructs that were incompatible with the Alpha 64-bit processor architecture. Such constructs included:

- Data structure alignment along 4-byte instead of 8-byte boundaries
- Use of intrinsic types not supported by the C compiler (e.g., 16-bit integer types)
- Casting of pointers to 32-bit integer types

As these features were discovered during testing and debugging, the software was modified and precompiler directives were added to isolate the modification to the T3D software build.

PDU multicasting and relevance filtering capabilities were introduced through the implementation of a multiple resolution grid that overlaid the ModSAF terrain as shown in Figure 4. PVM dynamic group facilities allowed the use of the *pvm_bcast()* to post PDUs to the group represented at a specific grid cell. Simulated entities used the services provided by the *pvm_joingroup()* and *pvm_lvgroup()* functions to express interest in receiving PDUs for specific cells within a parameterized perception range.

An algorithm similar to the one used in the sequential ModSAF to bundle PDUs was incorporated into the PVM-based packet distribution facilities. The implemented technique created a single bundle that accepted packets of the same multicast group and protocol family. The bundle flushed whenever the maximum size was reached or a packet of a different group or type was presented for transmission.

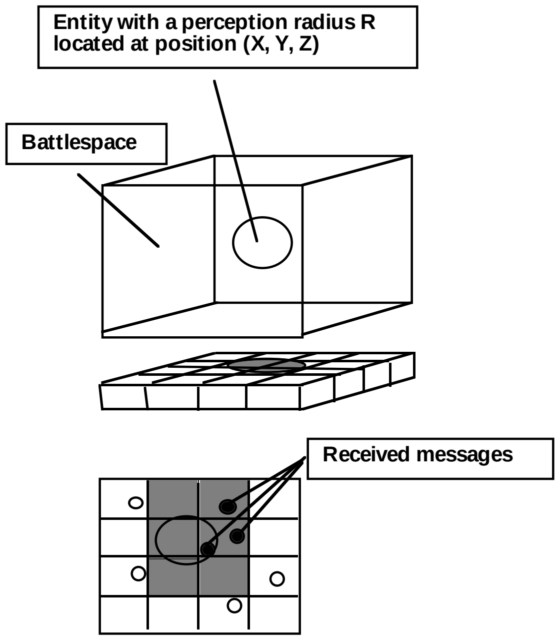
Entity with a perception radius R located at position (X, Y, Z)
Battlespace
Received messages

Figure 4. Relevance Filtering

At the time of this paper, dynamic load balancing algorithms were still under development.

Preliminary Test Results

To provide an initial indication of the performance of the message-passing implementation, we devised a simple test program that exercised the core simulation services. This program created notional (or behaviorless) entities that performed a random walk over the terrain and generated Entity State PDUs (ESPDUs) at specified intervals. Static terrain partitioning was applied so that each ModSAF PE simulated entities over a specified terrain area. The terrain was allocated to the PEs based on a simple row/column assignment mechanism.

The first series of tests performed broadcast distribution of all generated ESPDUs for varying numbers of processing elements. The data gathered during these tests included the packet send and receive rates, the number of locally simulated (owned) entities, the number of perceived but not simulated (reflected) entities, the total number of entities simulated, and the performance of the ModSAF event scheduler.

We then repeated the test series with relevance filtering activated. In this series, each entity expressed interest in receiving state data for all other entities residing in cells within a specified radius. In the second test series ESPDUs were multicast only to those PEs having expressed interest in the originating cell.

As shown in Figure 5-a, the number of packets received by each ModSAF process was reduced significantly when relevance filtering was activated. Figure 5-b illustrates that the number of reflected entities also dropped sharply, which would be expected because those entities would be responsible for the received ESPDUs. Figure 5-c tracks the number of multicast cell subscription and cancellation operations performed during a three minute run.

In all tests except the 32-processor broadcast, the ModSAF scheduler was able to effectively maintain real-time simulation. The 32-processor broadcast run failed when approaching sustained packet receive rates of 2,000/second. Preliminary indications are that the burden of processing thousands of packets per second overwhelmed the ModSAF scheduler and caused a PVM communications buffer overflow. This appears to validate our assumption that filtering and multicasting techniques were necessary even in high-performance parallel architectures.

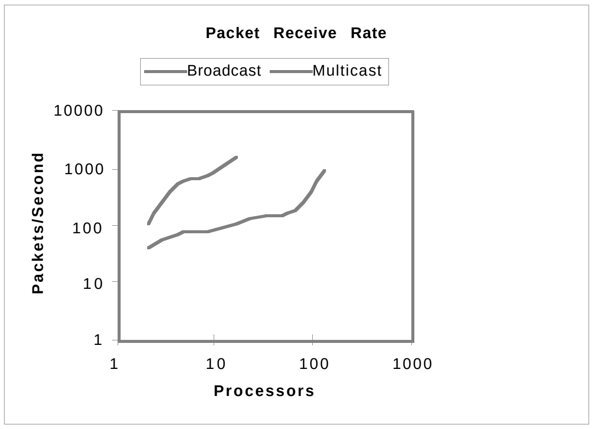

Figure 5-a. Packet Receive Rates

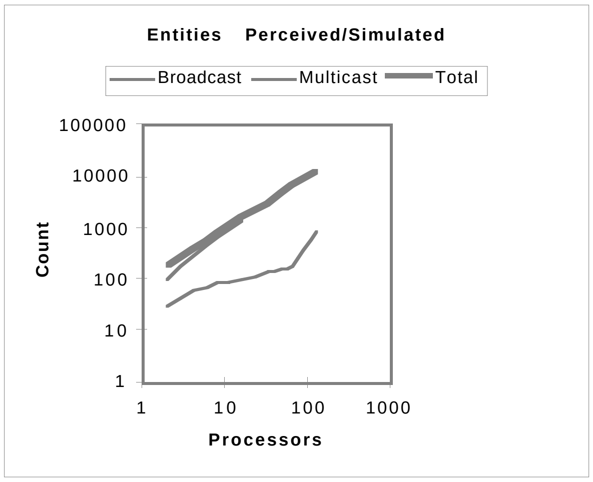

Figure 5-b. Entity Count

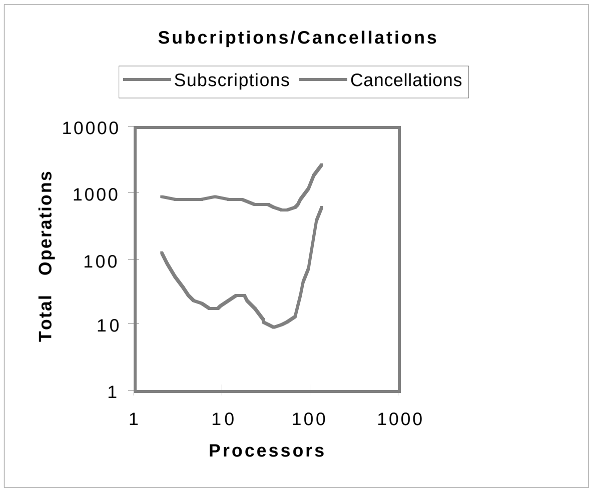

Figure 5-c. Cell Subscriptions

SHARED-MEMORY MODSAF

We chose the Convex Exemplar as the host platform for the shared-memory ModSAF implementation. This system had just been introduced when we were beginning this project and appeared to be an ideal candidate for our research. Several Convex Exemplar systems were used at the Convex Corporation facilities in Richardson, Texas.

The Convex Exemplar System Architecture

The Convex Exemplar consists of one to sixteen hypernodes, each hypernode containing an 8-node symmetric multiprocessor (SMP), for a total of 8 to 128 processors. Each processor is a Hewlett-Packard PA-RISC PA7100 with external data cache and instruction cache, and a clock frequency of 100 MHz. Both caches are virtually indexed with 64-bit data path and 800 Mbytes/second read rate.

The Exemplar uses a two-level coherent memory /interconnect hierarchy. Each hypernode contains one or more hypernode-private memories that provide local storage. Each hypernode also contains one or more global memory blocks that are accessible by all CPUs in the system.

The interconnect within each hypernode is a 5x5 non-blocking crossbar using semi-custom GaAs components. Each hypernode uses four uni-directional scaleable coherent interface (SCI) rings as the interconnection to other hypernodes.

Approach

As shown in Figure 6, rather than mapping N instances of distributed ModSAF onto the N PEs of an MPP, this approach implements a single ModSAF on an MPP having a globally-addressable shared-memory. This provides an end-run around the communications liabilities encountered in the message-passing approach. Here, the number of internal PDU messages is zero. Instead of broadcasting state data to every PE, remote data is accessed only when a PE specifically needs the data. Data is pulled, not pushed. If the load can be balanced so that most of the PEs can do useful work most of the time, the result is a scaleable parallel version of ModSAF.

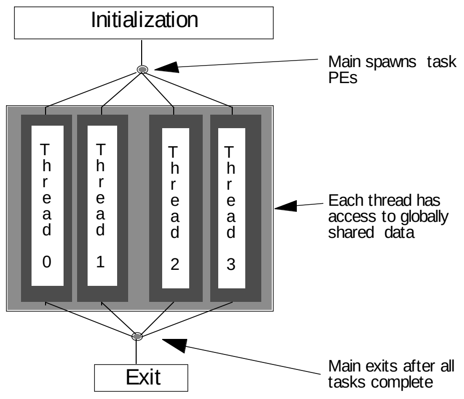
Initialization
Main spawns task PEs
Thread 0
Thread 1
Thread 2
Thread 3
Each thread has access to globally shared data
Main exits after all tasks complete
Exit

Figure 6. Shared-memory Approach

However, the implementation risks are greater in this approach. While the message-passing approach naturally embraces the existing ModSAF software architecture, significant modifications to the core libraries must be undertaken to take advantage of the capabilities that shared-memory MPPs provide.

These modifications focus on promoting key data structures to global memory to allow all PEs access to entity-related data and implementing semaphores and other locking mechanisms to enforce data integrity.

Implementation

Prototyping of the shared-memory ModSAF began in October 1994. We first created a stripped down ModSAF "surrogate" to explore alternative implementation paths. This program was used to verify that specific intended modifications to the actual ModSAF code would perform as expected.

Using the data gathered from the surrogate, we implemented a multi-threaded event scheduler that enabled each PE to control the simulation of a subset of the battlefield entities. We created a partitioned global data structure to allow each PE access to all entity data. This structure was instantiated as a binary tree to minimize traversal time during searches.

Entity-level locking was introduced to remove the possibility of simultaneous write/read operations by contending PEs. This resulted in a potential bottleneck at the root of the binary tree in an active simulation.

We combated this problem by creating a number of dummy nodes starting at the root and continuing to the $\log_2 N$ level where N is the number of PEs in the partition. To ensure freedom from deadlock, PEs only lock a single node in the binary tree at a time.

Standard memory allocation mechanisms on the Exemplar were found to be a performance bottleneck due to, in part, the granularity and frequency of the allocation. To improve performance, we instituted a memory manager that allocates memory from large, statically-allocated blocks.

To ameliorate the effects of load imbalance, we added entity hand-off capabilities to the software. This proved to be much simpler in a shared-memory implementation because the hand off of entity state only requires providing the receiving PE a pointer to the appropriate vehicle table entry. Following this approach, any events posted to the initiating PE scheduler at the time of hand off are canceled, repackaged, and provided to the receiving PE for posting.

Preliminary Test Results

The first significant result was to get ModSAF running correctly in parallel on the Exemplar. The locking techniques introduced for global memory access were thoroughly debugged, tested, and verified.

Second, we achieved scaleable performance over a two hypernode system (the largest available to date). Figure 7 shows wall clock time to run two ModSAF scenarios, as a function of the number of processors. Note the inverse scale, so that performance improvement has a positive slope. Performance of a perfectly scaleable application has a slope of 1.

Scenarios 1 and 2 are identical, except that 2 is run longer (20,000 vs. 12,000 seconds) and creates more entities (5,000 vs. 3,000). The entities are notional ModSAF vehicles and move randomly across the terrain. The vehicles are subject to damage by mines having an arbitrary detonation radius.

Each processor has periodic opportunities to perform dynamic load balancing by exporting a vehicle to a less busy processor. To parallelize the total work and assure a fair comparison in Figure 7, the global frequency of entity creation and mine detonation is independent of the number of processors (N), but the frequency *per processor* decreases linearly with N.

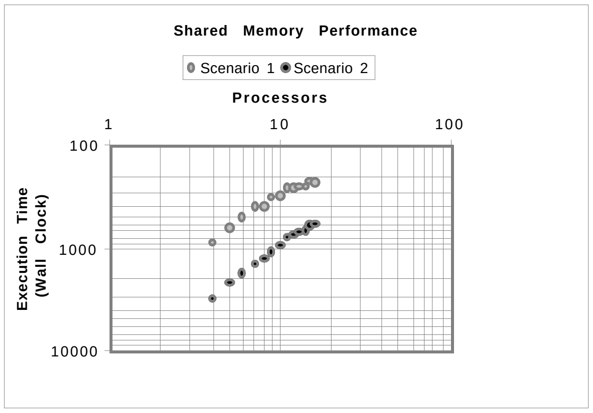

Figure 7. Scaleability Test Results

As can be seen in Figure 7, the performance of the first scenario has a slope near 1 for N < 10 with a more modest speedup for N through 16. The second scenario scales well for all values of N tried. Since the first 12,000 seconds of the second scenario are identical to the first, the lack of scaleability in the first sccenario can be attributed to initialization effects.

Dynamic load balancing, without which performance degrades for N > 8, proved crucial to achieving scaleable performance.

CONCLUSIONS

Our efforts have demonstrated the feasibility of adapting the ModSAF software architecture to execute on representative MPPs using both message-passing and shared-memory paradigms. Preliminary performance analysis shows that, by exploiting some of the unique capabilities of these MPPs, we can enhance and extend the performance of the simulation.

Our initial measures of performance on the 128-processor CRAY T3D seem to indicate that, while a message-passing approach will not scale, the introduction of bandwidth reduction techniques on the CRAY provide the leverage to support the communications load of over 10,000 entities. Actual entity loading on the T3D will depend upon many factors including entity type, distribution, and sensor capabilities. While we are not expecting super-linear performance, we do anticipate that per processor entity loading will be comparable to a uni-processor workstation. Further testing after behavioral libraries are ported will be necessary to benchmark actual performance.

Results from our shared-memory experimentation on the Convex Exemplar show that scaleability can be achieved (for up to 16-processor system) through incremental adaptations of the ModSAF RTI to accommodate multi-threaded execution. Our efforts on this track will next focus on introducing the behavioral libraries and extending scaleable performance. Since the behavioral libraries will access mostly thread-local instead of global data structures, we are optimistic that scaleability can be maintained for the full simulation.

ACKNOWLEDGMENTS

This work was supported in part by a grant of HPC time from the DOD HPC Shared Resource Center, Eglin AFB, Eglin Distributed Center Cray T3D.

REFERENCES

[1] Ceranowicz, A., 1994, “Modular Semi-Automated Forces,” Proceedings of the 1994 Winter Simulation Conference, J. D. Tew et al. (eds), 755-761, Institute of Electrical and Electronics Engineers, Piscataway, NJ.

[2] VanHook, D. J. and J. Calvin, September 1994, “An Approach to DIS Scaleabilty,” Proceedings of the Eleventh Workshop on Standards for the Interoperability of Distributed Simulation, Orlando, FL.

[3] Dahmann et al., September 1994, "Command Forces: An Extension of DIS Virtual Simulation," Proceedings of the Eleventh Workshop on Standards for the Interoperability of Defense Simulations, Orlando, FL.

[4] Salisbury, M. R., March 1995, "Command and Control Simulation Interface Language (CCSIL): Status Update," Proceedings of the Twelfth Workshop on Standards for the Interoperability of Defense Simulations, Orlando, FL.

[5] Seidel et al., April-June 1995, "CFOR Approach To Simulation Scaleability," Proceedings of the Electronic Conference on Scaleability in Training Simulation, The Society for Computer Simulation, Institute for Operations Research and Management Science.

[6] Chandrasekharan, N., Vemulapati, U. B., and A. T. Irwin, “MasPaWS - A Massively Parallel War Simulator,” Proceedings of the 1994 Winter Simulation Conference, J. D. Tew et al. (eds), 744-751. Institute of Electrical and Electronics Engineers, Piscataway, NJ.

[7] Nicol, D. M., 1988, “Mapping a Battlefield Simulation onto Message-Passing Parallel Architectures,” Proceedings of the SCS

Multiconference on Distributed Simulation, pp. 141-146.

[8] Niedringhaus, W. P., “Diffusive Dynamic Load Balancing by Terrain Parcel Swaps for Event-Driven Simulation of Communicating Vehicles,” Proceedings of the 28th Annual Simulation Symposium, Phoenix, AZ.

[9] PVM and HeNCE Programmer’s Manual, Cray Research, Inc., Eagan, MN.