Proceedings of the 1986 Winter Simulation Conference
J. Wilson, J. Henriksen, S. Roberts (eds.)

REPLICATED STATE SPACE APPROACH FOR PARALLEL SIMULATION

John B. Gilmer, Jr., Ph.D.
The BDM Corporation
7915 Jones Branch Drive
McLean, Virginia 22102

Jung Pyo Hong
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

ABSTRACT

Parallel processing offers the possibility of greatly increased performance for simulations which are computationally bound on existing machines. On shared memory machines, such as the BBN Butterfly, a natural approach is to allocate entities to be processed on different processors with locks used to prevent synchronization problems for a state space in global memory. Parallel processors having local memory only, such as the hypercube architectures, cannot use this approach. Such machines are potentially less expensive than shared memory architectures with similar local computational power, since the interconnection network is simpler. The most natural simulation paradigm for such machines, object oriented programming with interactions limited to messages, may become communications bound if the entities represented are tightly coupled. This paper presents an alternative approach based on use of replicated state spaces on each processor, and consolidation of these changes on a processor basis rather than an interaction basis to minimize message passing. The effect is to trade a parallel processing synchronization problem for a data consistency problem. The approach then relies only on a message broadcasting or passing architecture. For small degrees of parallelism, this requirement can be met by a variety of architectures. The method described is being applied to parallelize the CORBAN combat simulation for operation on a hypercube architecture as part of preliminary feasibility analysis concerning simulation support of AirLand Battle Management (ALBM).

1. BACKGROUND

Combat simulation is expected to be an important technique used as part of automated support systems for battle management to evaluate the likely results of various proposed courses of action or the impact of reported events. This application is currently computationally limited. For example, one such simulation, CORBAN, requires about ten minutes of run time on a Sun 2 microcomputer to execute a simulated ten-minute time step. The speedup needed for use in an operational environment will come from a variety of sources, but parallel processing is expected to provide most of the power. The BDM Corporation, the US Army Ballistic Research Laboratory, and Los Alamos National Laboratory have been investigating approaches for parallel simulation as part of the Defense Advanced Research Projects Agency's AirLand Battle Management program. In August 1985, a small combat simulation problem run on the BBN Butterfly was found to scale well with the number of processors (Gilmer June 1986). Subsequently, larger scale prototyping has begun using a subset of the CORBAN simulation for both the Butterfly and for a hypercube architecture (Gilmer, Hartwig, and Kokinakis 1986). The hypercube version was planned for an emulator at Los Alamos, and may also be put on an Intel iPSC and the Los Alamos/UNM "HC", in order to allow comparative benchmarks between it and the Butterfly (Hong, Patel, and Tomlinsen 1985) (iPSC System Product Summary) (Hong, Tomlinsen, Patel, and Pollard 1985).

This paper describes the technical approach to be used in a parallel version of CORBAN. The approach is applicable to message passing architectures in general and the hypercube in particular. The approach used on the Butterfly, a shared memory machine, is illustrated in Figure 1. There is one state space, located in globally accessible memory, in which all entities are located. References to such data are slower than local memory references, but still take only a few microseconds. The memory controller and communications network necessary to support such global references must be a major cost for such machines.

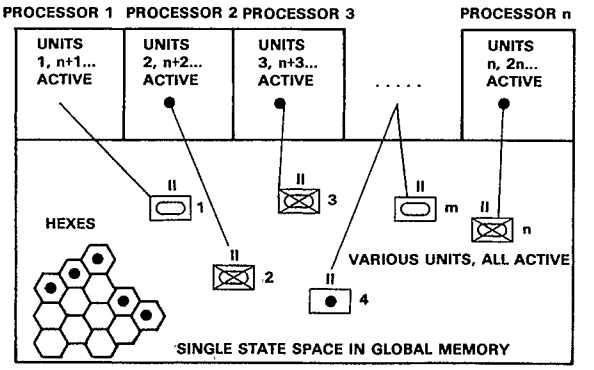


Figure 1. Butterfly Parallelism Approach

The CORBAN simulation is a Corps scope, battalion resolution, time stepped simulation having a hexagonal terrain representation with a maximum resolution of 3.5 km. The simulation includes representation of command/control so that it can operate in fully automated fashion given top-level operation orders for each side (Gilmer 1984). A typical scenario includes several hundred entities, and runs for two to five days of simulation time, using ten-minute

time steps. As currently structured, entities in the simulation make direct reference to state variables of other entities, and to terrain objects to which unit occupancy lists are attached. This use of memory reference is efficient but contrasts with a feature of many object oriented paradigms that restrict all interactions to message passing.

The consequences of such a restriction can be assessed given statistics collected from CORBAN simulation runs. One of the most important interactions represented from a performance point of view is target acquisition to support perception. For this purpose, each unit observes the hexagonal regions in its area, every time step. For each unit found, a test is made to determine whether detection occurred. Each time step, about 25,000 hexes are searched in this manner, with about 6,500 units checked. If a restriction to message passing is made and the terrain regions are each considered an object, then 50,000 messages are needed for inquiries to hexes and their replies, and 13,000 for inquiries to units and replies. On an iPSC, where each message requires greater than 75 microseconds, the time required would be on the order of at least five seconds per time step, allocated among the processors, or more than an hour for a five-day scenario (Peak Communication Rate Estimates 1985). This figure includes only actual message passing time, and does not include task switching, data transfer within memory, contention, or overhead. An estimate of at least 250 microseconds was made in a later study, and also is a lower bound assuming no contention (Gilmer 1985). With enough processors, this is within the range of practicality, but there is risk that performance could be much worse in practice. Therefore, an approach was sought that would minimize the number of messages.

The replicated state space approach described in the remainder of this paper is applicable to time stepped, entity centered simulations which must execute on parallel processors having no shared memory. An assumption is made that each entity perceives only values of state variables that were valid at the start of the time step. This is consistent with the time step being the highest resolution for time and is similar to treatment in a uniprocessor version of the simulation. It represents an alternative to the pure object oriented paradigm which reduces message passing by making state space reconciliation rather than entity interaction the basic framework for message passing. In doing so, this approach trades a multiprocessor synchronization problem for a data consistency problem. The next section examines the nature of the data and describes the state space replication and supporting data consistency approaches. In the conclusions, an estimate is made of the resulting message volume for comparison with that necessary with an object oriented message passing approach.

2. STATE SPACE CHARACTERISTICS

The simulation state space contains the variables which define the entities, their relationships, and their current state. In CORBAN, the same dynamically allocated data space is also used for data defining entity types, asset characteristics, and such. But as these data do not change during the course of a simulation replication, they are not part of the state space proper. In CORBAN, all state space variables are associated either with units or terrain regions, with the exception of a few global variables for such things as time and weather.

The state space variables can be classified by access characteristics, which are important in considering parallelism:

1. Constant data: These variables do not change once an instance of a unit or region is defined. An example is unit type. As constants, these data may be replicated without need for synchronization or update. These data are broadcast at the beginning of the simulation run then accessed by local reference.

2. Local use data: These variables are used only by the processor responsible for computations of a given entity and are not referenced by any other entities. In an object oriented approach limited to message passing, all variables are of this type (except for the message lists). Thus, these are local read and write only.

3. Globally read referenced data: These variables are read by any entities in the simulation but are only written by the owning entity. Within the simulation, this type of variable can be implemented as a composite of the previous two: a variable which is constant during a given time step and another which is updated locally. The globally read constant is updated during a synchronization interval from the locally accessed variable at the end of each time step. The synchronization interval must also provide for distribution of such changes if the state space is replicated. After broadcast, these data are local read only. On a hypercube, broadcast can be accomplished in $\log_2 n$ message passing steps.

4. Globally write referenced data: These variables are not only read but written to by various entities in the simulation. The writes are not arbitrary changes, which would be very difficult to handle. In CORBAN, each such variable represents the accumulation of some effect from multiple sources. For example, one variable may represent the number of tanks a unit has. The information that is passed in a global write is not so much a new value for the number of tanks but the change in number of tanks due to attrition or supply functions. Similarly, other variables may accumulate lists of messages or units. Some act as flags which indicate the existence of some condition, which may be set by any of a number of sources. Thus, globally written variables can be implemented as a globally read variable plus an accumulator which is modified with atomic operations, but not read. The read referenced variable is updated during the synchronization interval.

In summary, these data require collection, requiring a broadcast from all nodes, then a dissemination, requiring rebroadcast.

Note that data of types 3 and 4 are prohibited in message based, object oriented programming.

3. REPLICATED STATE SPACE APPROACH

The purpose of this approach is to replicate the simulation state space in the local memory of every processor of the parallel computer. Thus, during simulation processing, there is no need for messages among processors except as necessary at the end of each time step to enforce consistency among the different instances of the state space. Further, consistency need only be enforced for those variables which are globally read or written, as constant data do not change and local use data are not referenced except on the processor where the changes occur. Figure 2 illustrates the approach.

State space consistency can be enforced by processing each time step in three phases:

1. Execution of simulation algorithms for one time step,
2. Passing global write modifications from processors on which they occur to the processor on which the written variable's entity resides, and
3. Dissemination of changes to globally accessed variables from the processors on which these variables' entities reside to all other processors.

In phase one, all algorithms execute just as they would on single processor computers, except that the entity set is partitioned into subsets that reside and execute on different processors. It is expected that this phase would dominate run time.

In phase two, each processor would check all variables used to accumulate changes to globally written variables. For each such variable which is modified, action must be taken to ensure consistency. A list of such variables and their modifications must be broadcast to all other processors in the system. Null lists would be needed so that the destination processor will know when all such incoming messages have been received.

To illustrate, consider unit 2 in the situation shown in Figure 2. The unit receives fire from units 1, 2n, and n+1. The actual combat calculations take place in processors 1 and n. In processor 1, the initial value of 30 for the number of tanks unit 2 has is decremented by 2 for losses inflicted by unit 1. and by 1 for loss to fire from unit n+1. These decrements are stored separately as the value -3. Simultaneously, in processor n, a loss of 1 tank from the initial value of 30 is registered. At the end of the time step, messages are sent to processor 2 from all other processors. The message from processor 1 will list the number of tanks in unit 2 as being decremented by three, the message from n will show the loss of one, and other processors will not mention this variable. After receiving all such messages, processor 2 updates the number of tanks unit 2 has, changing it to 26. Messages are then sent out to update this variable and any others which have changed on all other processors, as phase three.

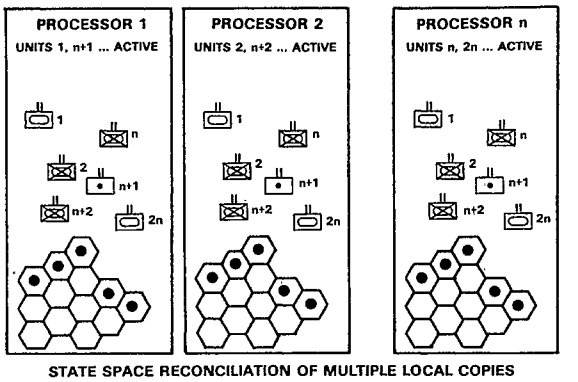
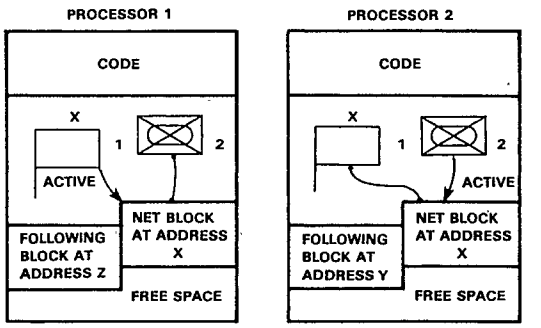


Figure 2. Hypercube Parallelism Approach

If the space allocations were static for the duration of a simulation replication, one could identify globally accessed variables by memory address, and the state spaces on different processors would be made to be identical at the end of each state space reconciliation. Such is not the case, however. In the course of a CORBAN replication, units, messages, and other block types are allocated and returned. This brings up the likelihood of the same memory being allocated differently on different processors. Consider the simple case illustrated in Figure 3. Some of the objects which may thus be allocated are entities themselves. Consequently, addresses as such cannot be used in messages except in reference to unchanging descriptor data which will remain identical on all processors. All objects must have some tag or name, which will be generated in such a way that no two processors can generate



ASSUME: STATE SPACES ON PROCESSORS 1 AND 2 ARE IDENTICAL AT START OF TIME STEP.

SEQUENCE:
(1) ON PROCESSOR 1, UNIT 1 GENERATES A MESSAGE TO UNIT 2. A BLOCK OF FREE SPACE AT X IS ALLOCATED FOR THIS. THE MESSAGE IS ATTACHED TO UNIT 2 AS A MODIFICATION TO UNIT 2'S INCOMING MESSAGE LIST.
(2) ON PROCESSOR 2, UNIT 2 SIMILARLY SENDS UNIT 1 A MESSAGE, ALSO ALLOCATING A BLOCK STARTING AT X.
(3) THE MESSAGES ARE EXCHANGED DURING STATE SPACE RECONCILIATION. ON PROCESSOR 1, THE MESSAGE FROM 2 TO 1 MUST BE ALLOCATED FROM FREE SPACE STARTING AT Z.
(4) THE STATE SPACES ARE NO LONGER IDENTICAL.

CONCLUSIONS VARIABLES MUST BE REFERENCED IN TERMS OF OBJECTS (BY "NAME") RATHER THAN BY ADDRESS.

Figure 3. Global Addressing Problem

the same object identifier. For example, in CORBAN, units are referenced by an integer unit number. In such a parallel simulation, the units may be allocated to the processors such that the unit number modulo n is the processor number. A new unit generated by processor m would thus be restricted to the set {m |m modulo n = p}, where n is the number of processors and p the particular processor. Messages, orders, and other blocks which might be allocated that are attached to entities, may be referenced by that association rather than by unique identification numbers. For example, a message may contain three words which are passed to the processor having the receiving entity. They need be identified only as "a message for unit m."

Since pointers will have no consistency for such data, it will be necessary to pass information by transmitting the entire contents rather than just a pointer, as is possible in shared memory architectures. When the message conveys a large, complex data structure such as an operation plan, this can make quite a difference in quantity of data passed. In CORBAN, such messages are not very frequent compared to other interactions. The difference in data volume is probably not significant compared to the time required for message passing overhead.

Development of code to implement the state space consistency is not expected to be an easy exercise, especially for operation in the general case directly from data structure definitions. For purposes of the current project, code to do this will be tailored to the requirements of the CORBAN subset that is the basis for the research, which will be a much easier job. If this research shows that the state space reconciliation is a promising approach, then how to build a general purpose tool to perform state space reconciliation will become an active issue, but it is beyond the scope of the current effort.

The CORBAN subset used for the research underway includes perception, combat, and movement processes, where most unit interactions take place. The decision making processes were left out since they account for very little of the processing time. They also affect only local data except for messages sent, and their impact on the physical processes only becomes significant over multiple time steps. Thus, in this subset of CORBAN, it would be possible to maintain identical state spaces since no block allocations take place. Nevertheless, the implementation planned will not assume this, so that the experiment will address problems and issues in the full simulation, and not just the subset.

4. CONCLUSIONS

With the state space reconciliation approach described, each processor broadcasts two times each time step, requiring n messages and up to log n message steps for each such message on a hypercube. For a 128-node machine, this is 33K message per time step. This contrasts with 63K messages just for perception interactions in an object oriented message approach. The average message length is not the same, and message lengths will depend greatly on the scenario. Unless the difference in length exceeds hundreds of bytes, however, message overhead will be dominant. The state space reconciliation process offers a further advantage in that the message passing is well structured, and can be organized to minimize contention. Message generation on an object basis cannot be so structured. On hypercubes, this may make a significant difference.

Data gathered from an implementation of this approach is expected to further quantify the merits of the proposed method, as well as allow comparison of shared memory versus local memory parallel processors in general.

ACKNOWLEDGEMENTS

The approach reported in this paper was developed as part of the AirLand Battle Management program of the Defense Advanced Research Projects Agency, under Contract Numbers F29601-86-C-0022 (BDM) and BRL.PO.76-85 (Los Alamos). Help has been received from the US Army Ballistics Research Laboratory. The content of this paper presents the views of the authors and in no way represents the position of The BDM Corporation, Los Alamos National Laboratory, or any agency of the United States Government.

REFERENCES

Gilmer, John B., Jr. (August 9, 1985). "Comparison of Intel iPSC and BBN Butterfly Computers," The BDM Corporation, BDM/R-85-0976-TR, Appendix B.

Gilmer, John B., Jr. (December 1984). "Dynamic Variable Resolution in the Quickscreen Combat Model," Proceedings of the Winter Simulation Conference, IEEE, pp 597-602.

Gilmer, John B., Jr. (June 1986). "Statistical Measurements of the CORBAN Simulation to Support Parallel Processing," The BDM Corporation, Tech nical Report BDM/ROS-86-0326-TR. This paper has been submitted to the 54th Military Operations Research Society Symposium.

Gilmer, John B., Jr., Hartwig, George, and Kokinakis, Louis (1986). "Parallel Entity Centered Simulation on the Butterfly Computer." This paper has been submitted to the 1986 International Conference on Parallel Processing.

Hong, Jung Pyo, Patel, Nisheetn R., and Tomlinsen, Robert D. (May 18, 1985). "An Example of How to Use the Hypercube Simulator with a FORTRAN Program," Report LA-UR-85-1405, Los Alamos National Laboratory.

Hong, Jung Pyo, Tomlinsen, Robert D., Patel, Nishetn R., and Pollard, L. Howard (August 1985). "A Hypercube Project and Simulator for a Hypercube of Computers," Report LA-UR-86-1151, Los Alamos National Laboratory; submitted to the Proceedings of the Conference on Hypercube Micro processors held in Knoxville, Tennessee, to be published by SIAM Publications.

iPSC System Product Summary, Intel Corporation publication 280101-002.

"Peak Communication Rate Estimates," handout provided by Intel Scientific Computers, undated, received May 1985.

AUTHORS' BIOGRAPHIES

JOHN B. GILMER, JR., was born in 1949 in Augusta, Georgia. He is a graduate of the US Naval Academy, and received a Ph.D. in Electrical Engineering from Virginia Polytechnic Institute and State University in 1983. He has worked for The BDM Corporation since 1977 in the fields of combat simulation and computer and systems architecture, with recent emphasis on representation of command and control and parallelism.

John B. Gilmer, Jr., Ph.D.
The BDM Corporation
7915 Jones Branch Drive
McLean, Virginia 22102

JUNG PYO HONG was born in 1948 in Seoul, Korea. He attended California Institute of Technology, the University of California at Berkeley, and received a Ph.D. in Electrical Engineering from the University of Southern California in 1971. He has since worked at the Jet Propulsion Laboratory, The Electromagnetic Systems Laboratory, and Los Alamos National Laboratory. He has been active in the field of parallel computer algorithms and architectures.

Jung Pyo Hong
Los Alamos National Laboratory
Los Alamos, New Mexico 87454