Scalability of Parallel Battlefield Management Simulators on Local-Memory Computers

Sushil K. Prasad, Ajay K. Gupta, Srinivas Danda, Narsingh Deo, and Tulasi Tripuraneni

Abstract

We present three parallel battlefield management simulators, initially implemented on Intel's iPSC/2 and then ported to nCUBE-2 and to a network of workstations employing Parallel Virtual Machine package (PVM). Speedups obtained are 61.6 on 128-processor nCUBE-2, 9.5 on 16-processor iPSC/2, and 5.5 on a network of 16 Sparc-I workstations. Scalability is expected to further improve for larger battlefields and longer simulations.

1 Introduction

Battlefield management simulations are amongst the most irregular, computationally intensive, and complex simulations in existence. For a detailed survey of the relevant literature, readers are referred to [3].

In this paper, we consider the scalability of our parallel time-driven battlefield management simulators. We ported our parallel battlefield simulator originally implemented on a 16-processor Intel's iPSC/2 [3] to a network of 16 Sparc-I workstations running PVM 3.1 [4] and to an nCUBE-2, a 128-processor local-memory computer with hypercube interconnect. Compared to iPSC/2 which uses cut-through routing for inter-processor communication, nCUBE-2 has more sophisticated communication system based on worm-hole routing [6]. Workstations are interconnected by standard 10 MB/s ethernet. We are able to obtain speedups of 38 and 61.5 on 64- and 128-processor nCUBE-2, respectively. On the workstations, the speedup was 5.48 on 16 processes. The current study was prompted by our previous results on iPSC/2 on which we had obtained a speedup of 9.5 on 16 processors, and had conjectured that almost linear speedup could be obtained for larger number of processors [3]. The results obtained so far are expected to improve with larger battlefields and longer simulation clocks as indicated by the preliminary experiments.

Earlier domain-decomposition strategies for message-passing architectures like iPSC/2 sacrificed load balancing to reduce the communication overhead [7]. When the rectangles that divided the battlefield contained only one hex, the efficiency of the simulator dropped to about 20%. Increasing the rectangle size reduced the communication overhead but resulted in load imbalance.

The technological advances in the interprocessor communication hardware, as the improvements of interprocessor communications of iPSC/2 over that of iPSC/1 [1] and that of nCUBE-2 over nCUBE-1 [6] indicate, decrease the emphasis on communication overhead. Our first battlefield simulator implemented on iPSC/2 employs a static mapping of the battlefield to the processors with one hex per rectangle and is free of any redundant computation. The mapping function exploits the characteristics of the hypercube interconnection and guarantees that adjacent hexes are mapped to processors that are at a distance of at most two communication links. We ported the simulator from iPSC/2 to the nCUBE-2 and to workstation cluster running PVM to see the effect of varying communication overheads and the scalability of our design.

"C" language conventions are used throughout. Section 2 describes the basic simulation program, the data structures and the partitioning scheme. Our experience with porting to different architectures are discussed in Section 3. The experimental data obtained is included in Section 4. Section 5 contains some concluding remarks.

2 Simulation Programs

All our simulators are modeled after Zipscreen [5] and are time-stepped and unit-centered. Each unit is of battalio size and consists of tanks and machine gun vehicles. On the battlefield, each combat unit seeks enemy units engages in combat, and moves following the combat.

Each combat unit is represented by an identification number, type and its assets. Each asset is attributed
with a weapon type, number of such assets, and the amount of ammunition. The two-dimensional battlefield is divided into regular hexes and each hex is referred to by a hex co-ordinate system. The opposing units move through the hexes, engaging in combat only when they are on the same hex or adjacent hexes.

Program and Data Structures

We describe the simulator on iPSC/2 first. The modifications to this program when ported to nCUBE and PVM would be discussed in the next section.

The host processor sends the initial assignments of units and some initialization data to the node processors and receives a snap-shot of the simulation state after each time step from them. The program running at a node (which is the same for each node) repeatedly simulates each unit residing at the hex positions assigned to it.

The structure to hold all the units assigned to a processor is an array my_units[ ] of structure hex_struct, one for each hex assigned. The structure hex_struct contains two arrays for holding the red and the blue units at a hex. In each simulation step, the copies of the units at a hex are sent to the six processors assigned to the six neighboring hexes and copies of the units from these six processors are received, once to form target lists, then to report casualties, and finally to report migration of units. For further details, refer to [3, 8].

Partitioning: An hypercube machine of dimension $d$ consists of $n = 2^d$ node processors labeled 0 through $n - 1$. Two processors $i$ and $j$ are connected via a direct physical link if the Hamming distance between the binary codes of integers $i$ and $j$ is one. To map the processors onto the battlefield domain, they are viewed as forming a two-dimensional grid with $c = 2^{[d/2]}$ columns and $r = 2^{[d/2]}$ rows. The location $(i, j)$ of the grid is occupied by the processor $i, 0 \leq i \leq n - 1$, such that the binary code of $k$ is the reflected binary Gray code of $i$ concatenated to that of $j$. The battlefield hex $(i, j)$ is assigned to the processor occupying the grid position $(i \mod r, j \mod c)$. Each processor simulates the actions of all the units on the hexes allocated to it. The mapping strategy ensures that any two adjacent hexes are assigned to processors which are either directly connected by a link or have a common processor to which both are linked. Since the distance between any two processors assigned to adjacent hexes is at most two in the hypercube, the communication cost is dependent only upon the number of hexes assigned to each processor [8]. The message exchanges also serve to synchronize the processors within each phase.

3 Porting to nCUBE-2 and PVM

The main problem we faced in porting the battlefield simulation code to nCUBE-2 was the memory requirements in the code versus the memory required for the communication buffers that are used during communication among the processors. In nCUBE-2, the system communication buffers use the memory out of the processor's memory and this can be specified as a parameter when executing a program. Since the complete information of all the hexes assigned to a processor must be communicated to the six neighbors of this processor, large communication buffers are needed. We also used dynamic memory allocation in the code as much as possible instead of a static memory allocation (such as the ones using multi-dimensional arrays of fixed length) so that the simulation code could be run on processors with small amount of memory as well.

Since PVM allows the use of a heterogeneous cluster of workstations, the main changes in our port of the battlefield simulation code were 'packing' and 'unpacking' calls before and after every process-to-process communication so that byte order conversions can be performed when necessary. A feature in PVM heuristically allocates multiple processes to processors (workstations) when the number of processes are more than the number of workstations in the cluster, in general. Allocating one process per workstation uses more TCP/IP or UDP communications whereas interprocess communication on the same workstation could be done using pipes and FIFOs (in Berkeley UNIX BSD 4.3 etc) or message queues (in System V). Interprocess communication on the same machine could be thus faster than a LAN communication. Thus, in our port and during our runs, we tried to employ more than one processes per workstation (one process per workstation typically resulted in more execution time, mainly due to communication).

4 Execution Times

We conducted experiments on our three simulators by varying (i) the size of the battlefield as 8x8, 16x16, 24x24, 32x32, and 40x40 hexes, (ii) the number of simulation steps as 5, 15, 25, and 35 steps, (iii) the number of combat units on each side, and (iv) the number of processors. For the battlefield with 8x8, 16x16 and 24x24 hexes, the number of combat units was varied as 10, 20 and 50 units on each side. The number of combat units was varied as 50, 100 and 250 on each side for the larger battlefields. The number of processors was varied as 1, 2, 4, 8, 16, 32, 64, and 128 on the nCUBE. Initially,
the combat units were randomly distributed in the two halves of the battlefield in all these experiments.

For brevity, we provide the data obtained only for the 32x32- and 40x40-cell battlefields. A complete listing of the execution times of the three simulators obtained for all the experiments is given in [2] and [9]. All the data reported is averaged over 10 runs. We measured the communication times also on each architecture (not shown here) and found that it was consistently about 50% of the total execution times on nCUBE. For other two architectures, it was higher – the communication was dominant on PVM which is the primary cause of its lack of scalability.

Figures 1, 2, and 3 show speedup curves with varying number of steps for 32x32 and 40x40 battlefields containing 50 units on each side. On nCUBE, increasing number of simulation steps and increasing battlefield sizes increase the speedup. Thus, there is potential for even larger speedups. We note that communication buffer size is currently the main cause limiting us from simulating larger battlefields on the nCUBE. On PVM, larger number of steps actually decreased the speedup. The reason was that the simulator became overwhelmingly communication-bound thus saturating the ethernet and causing increasing collisions with the length of the simulation. However, speedup increased with larger battlefield size.

5 Conclusions

We presented the performance of a time-stepped battlefield simulator on three different architectures with varying degree of inter-processor communication bandwidths. The primary conclusion is that the simulator scales to a good degree over a large number of processors on machines such as nCUBE-2 even with its static partitioning scheme, and that its scalability is expected to increase with faster communication networks. A network of workstations is effective only for a small number of processors with the current simulator design.

Among the enhancements we are currently working on include dynamic load sharing among neighboring processors and inclusion of fast moving objects such as airplanes and missiles into the simulator.

References


Computer Science, University of Central Florida, Orlando, Florida.


Figure 1: Speedup on NCUBE-2 for battlefields with 50 units.

Figure 2: Speedup on iPSC/2 for 32 x 32 hexes battlefield with 50 units.

Figure 3: Speedup on PVM for battlefields with 50 units.