Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

PROCESSOR ALLOCATION IN PARALLEL BATTLEFIELD SIMULATION

Narsingh Deo   Muralidhar Medidi

Department of Computer Science
University of Central Florida
Orlando, Florida 32816, U.S.A.

Sushil Prasad

Math & Computer Science Department
Georgia State University
Atlanta, Georgia 30303, U.S.A.

ABSTRACT

Load balancing is a critical issue for exploiting the parallelism in any application and, particularly, in battlefield simulation where the computational load dynamically changes with both time and space. Domain decomposition is an effective means to balance the load distribution in battlefield simulation. However, finer domain decompositions that lead to better load balance incur heavier communication overhead. Earlier attempts in parallelizing battlefield simulation have traded load balance in favor of low communication overhead. In this paper, we present three parallel battlefield simulators, implemented on Intel's iPSC/2 and BBN Butterfly GP-1000 multicomputers, with finer domain decomposition and address the communication overhead problem by processor allocation strategies that suit the underlying architecture of the machine. On the shared-memory BBN Butterfly, the strategy leads to a new parallel battlefield simulation with dynamic load balancing. Execution times of these simulators are provided, which show that the communication overhead is tolerable.

1 INTRODUCTION

Battlefield simulations are amongst the most irregular, computationally intensive, and complex simulations in existence. Parallel processing offers the possibility of greatly increased performance for simulations which are computationally bound on existing sequential machines. However, mapping a simulation onto a multicomputer is especially difficult when the computational load changes dynamically as a function of time and space as in battlefield simulations such as CORBAN (Gilmer 1986). CORBAN, a time-stepped simulation, focuses on military units such as regiments and battalions. The units move through a two-dimensional domain tessellated by hexagons, and combat with nearby units from the opposing side.

Zipscreen, a simplified version of the CORBAN, was presented in an attempt to parallelize the battlefield simulation (Gilmer, Hartwig, and Kokinakis 1986). It included only the perception, combat and movement processes and was implemented on the Butterfly using software locks to protect the shared data. On a large-scale simulation, Zipscreen provided 50% efficiency on a 40-processor Butterfly. Zipscreen was later adapted to the distributed-memory architectures by replicating the state-space (Gilmer and Hong 1986). Computational load is distributed by assigning equal number of combat units to each processor at the beginning of the simulation. Each processor maintains a replica of the states of all units. In every time-step, each processor (i) simulates the actions of the units assigned to it; (ii) sends messages to the processor assigned to the units modified during the simulation; and (iii) disseminates the states of its units to all other processors. Thus, the replicated state-space approach requires that every processor communicate with every other processor resulting in heavy communication overhead.

The replicated state-space approach also suffers from load imbalance because serious computation takes place only when units are geographically close, so that during any given time-step a combat unit may or may not demand substantial computation. The expected speedup (ratio of the execution time with 1 processor to that with p processors) in the replicated state-space approach is inversely proportional to the number of combat units (Nicol 1988). To improve speedup, Nicol proposed parallelizing the simulation by distributing the battlefield domain instead of the combat units. The battlefield is covered by w-by-h hexagon (hex, from now on) rectangles and the processors are viewed as forming an r-by-c rectangle. The battlefield rectangle (u, v) is assigned to the processor occupying the $(u \bmod r, v \bmod c)$ position in the processor rectangle. Each processor also holds only the copies of units on the adjacent hexes. Communication and redundant computation are required in every time-step to properly maintain the states of the units on the adjacent hexes.

Noting the dependence of speedups obtained on the size of the rectangles into which the battlefield is divided, a pseudo-dynamic remapping of the battlefield onto the processors is also proposed (Nicol 1989). The decision to remap must take into account the performance gains and costs involved and simulation's future behavior. However, many of these factors cannot be quantified in battlefield simulation.

Some other reported works on event-driven parallel battlefield simulation include the application of QSIM, a tool to model division level tactical communication systems (Malmberg et al. 1984); the usage of an object-oriented message-passing language ROSS to model a ground battle (Klahr et al. 1986); and the utilization of the Timewarp operating system (Weiland et al. 1989).

In this paper, we consider the parallelization of only time-driven battlefield simulation. We describe three parallel battlefield simulation programs implemented on the Intel's iPSC/2 and the BBN Butterfly GP1000. Earlier domain-decomposition strategies (Nicol 1988) for message-passing architectures like the iPSC/2 sacrificed load balancing to reduce the communication overhead. When the rectangles that divide the battlefield contain only 1 hex, the efficiency of the simulator drops to about 20%. Increasing the rectangle size reduces the communication overhead but results in load imbalance. The improvement in efficiency due to the expansion of the rectangle was attributed to a reduction in the communication overhead. However, as the rectangle size increases, the redundant computation decreases which also contributes to the improvement in efficiency. Moreover, the technological advances in the interprocessor communication hardware, as the improvements of interprocessor communications of the iPSC/2 over that of the iPSC/1 indicate (Chorafas and Steinmann 1990), decrease the emphasis on communication overhead. Our first battlefield simulator, implemented on a 16-processor iPSC/2, employs a static mapping of the battlefield to the processors with one hex per rectangle and without 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. Then, we ported the simulator from the iPSC/2 to the shared-memory Butterfly for comparison purposes.

The bottlenecks caused by the software locks and the load imbalance inherent in the Zipscreen simulator reduce its scalability. The static-assignment battlefield simulator on the Butterfly can be easily modified to use a dynamic mapping of the hexes to the processors. We implemented such a simulator, the third parallel battlefield simulator, on a 32-processor Butterfly using a processor allocation strategy effective for the underlying shared-memory architecture.

2 BATTLEFIELD SIMULATORS: LOAD ASSIGNMENT STRATEGIES

All our simulators are modeled after Zipscreen and are time-stepped and unit-centered. Each unit is of battalion 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. The actions of these units during the battle are simulated. 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 coordinate system. The opposing units move through the hexes, engaging in combat only when they are on the same hex or adjacent hexes.
PHASES: The simulation in each step consists of the following four phases:

1. Perception: Enemy target information is gathered by exchanging combat-unit-information between neighboring hexes.
2. Combat: Enemy target units are identified and are fired at. The casualty reports are exchanged between neighboring hexes to update the status of the combat units. If all assets of a combat unit are destroyed, it ceases to exist.
3. Migration: Depending on the firing done in different directions, each combat unit identifies a direction of movement, if any, and moves in that direction.
4. Update-Battlefield: The migrating units are placed in the proper hexes.

The simulation program outputs a description of the battlefield in terms of the combat units. All the three simulators allow the size of the battlefield itself to be varied and interactively prompt the user for the number of simulation steps. The simulators load the initial unit-assignment on the battlefield from an input file.

The computational load in the simulation is distributed by dividing the battlefield hexes among processors. In the static allocation described in Subsection 2.1, the assignment is done at the beginning of the simulation and does not change during the simulation. The dynamic allocation, on the other hand, views the simulation as a collection of subprocesses, one each for the processing required at every hex of the battlefield. The load is distributed by allowing the free processor to work on the next subprocess to be computed.

2.1 The Static Assignment

The static assignment suits the distributed-memory hypercube machines, like the Intel's iPSC/2 computer. The iPSC/2 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 1. To map the processors onto the battlefield domain, they are viewed as forming a two-dimensional grid with $c = 2^{\lceil d/2 \rceil}$ columns and $r = 2^{\lfloor d/2 \rfloor}$ rows. The location (i, j) of the grid is occupied by the processor k, $0 \leq k \leq n-1$, 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 $(i \bmod r, j \bmod c)$. 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. We implemented simulators using the static assignment on both the iPSC/2 and the Butterfly. The required inter-processor communication on the Butterfly is performed using the shared-memory.

During each phase in a simulation step, each processor needs to know about the combat units on all the neighboring hexes. In the battlefield simulator on the iPSC/2, enemy-unit-information is obtained by exchanging messages with the processors associated with the adjacent hexes. Since the distance between any two processors assigned to adjacent hexes is at most two, the communication cost is dependent only upon the number of hexes assigned to each processor (Prasad 1990). The message exchanges also serve to synchronize the processors within each phase. In the static-assignment battlefield simulator on the Butterfly, the combat-unit-information is exchanged using the shared-memory. The processor which needs the information copies it from the shared-memory. Processors are explicitly synchronized in the static-assignment simulator on the Butterfly, using a global counter.

2.2 The Dynamic Assignment

The dynamic assignment strategy is designed for shared-memory machines like the BBN Butterfly. The battlefield is not mapped onto the processors *a priori*; instead, the data structures representing the battlefield are stored in the shared-memory such that any processor can access any hex of the battlefield. To reduce memory contention, these data structures are distributed across the memory modules. A global counter is maintained which indicates, in any phase, the next hex to be worked upon. In each of the four phases, the next free processor starts executing the action required by the next hex. Thus the computational load within each phase is more or less evenly distributed across the processors. Since one phase cannot start before another completely ends, the processors need to be synchronized between the phases. A battlefield simulator using dynamic load allocation is implemented on the Butterfly.

2.3 Static vs. Dynamic Assignments

In the static-assignment simulators, the hexes of the battlefield are assigned *a priori* to the processors. The combat units move unpredictably and sometimes cease to exist as battle progresses. Since the hex-to-processor mapping is permanent, there is no guarantee that the combat units will be uniformly distributed among the processors as battle progresses, which could create load imbalance. The main advantage of static mapping is the simple and easy-to-compute communication patterns between the processors.

In the dynamic-assignment simulator, the computational load within each phase is more or less evenly distributed across the available processors. Thus the dynamic mapping can adapt to the unpredictable movements of the combat units and distribute the computational load evenly among processors in each time-step. However, it is advantageous only when the cost of dynamic assignment is small as in shared-memory machines like the Butterfly.

3 EXECUTION TIME

We conducted experiments on our three simulators by varying (i) the size of the battlefield as 8x8, 16x16, 24x24 and 32x32 hexes, (ii) the number of simulation steps as 5, 15 and 25 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

TABLE 1: Execution Times, in Seconds, of the Simulator on the iPSC/2 for a 32x32 hex Battlefield

	Combat Units on Each Side								
	50			100			250		
Processor	Simulation Steps			Simulation Steps			Simulation Steps		
	5	15	25	5	15	25	5	15	25
1	69.93	217.26	355.59	75.75	235.55	377.36	87.18	296.45	440.11
2	46.35	144.79	239.18	49.61	154.90	249.98	55.09	187.06	283.18
4	27.22	83.18	137.75	29.54	89.31	142.80	32.38	106.44	162.72
8	14.21	43.51	71.80	15.72	48.66	77.15	17.98	57.67	86.59
16	7.60	22.87	37.36	8.53	26.36	40.86	9.60	32.87	47.70

bigger battlefield of 32x32 hexes. The number of processors was varied as 1, 2, 4, 8, and 16 (powers of 2) in the experiments on static-assignment battlefield simulators. For the dynamic-assignment simulator on the Butterfly, the number of processors was varied from 1 to 27 in increments of 1. Initially, the combat units were randomly distributed in the two halves of the battlefield in all these experiments. In these timing studies, we suppressed the output generating routines.

3.1 Static-Assignment Simulator on the iPSC/2

In this paper, we provide the execution times of the battlefield simulator on the iPSC/2 obtained only for the 32x32 hex battlefield due to the space limitations. A complete listing of the execution times of the three simulators obtained for all the experiments is given in (Deo and Medidi 1992). Table 1 lists the execution times for a 32x32 hex battlefield as the number of combat units is varied with 50, 100 and 250 units on each side and for 5, 15, and 25 step simulations.

As expected, the execution time is linearly proportional to the number of simulation steps. On the other hand, the execution time grows slowly with the number of combat units because (i) the units engaged in combat are only those that are on same or adjacent hexes and (ii) the Perception phase, which forms the bulk of the execution time, is independent of the number of combat units. As can be seen, the execution time is almost inversely proportional to the number of processors used. Figure 1 shows the speedup obtained, as the size of the battlefield varies, against the number of processors. The number of combat units initially placed on the battlefield for each side is kept fixed at 50 and the number of simulation steps at 15 for the speedups shown. As expected, the speedup increases as the size of the battlefield increases.

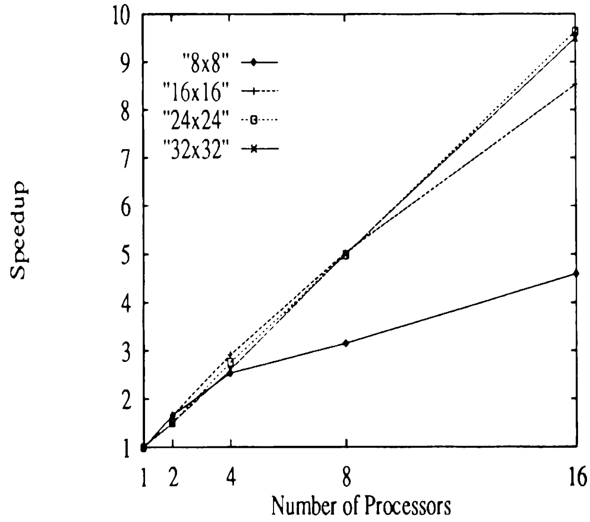

Figure 1. Speedup of the Battlefield Simulator on the iPSC/2 with different Battlefield Sizes
No. of Simulation Steps = 15
No. of Combat Units on Each Side = 50

3.2 Static-Assignment Simulator on the BBN Butterfly

Table 2 lists the execution times of the static-assignment simulator on the BBN Butterfly for a 32x32 hex battlefield. Again, the execution time is almost linearly proportional to the number of simulation steps and grows slowly with the number of combat units. The execution time of this simulator is greater than that of the simulator on the iPSC/2 for the same input configurations as the Butterfly employs slower and less powerful processors than the iPSC/2.

TABLE 2: Execution Times, in Seconds, of the Static-Assignment Simulator on the BBN Butterfly for a 32x32 hex Battlefield

	Combat Units on Each Side								
	50			100			250		
Processor	Simulation Steps			Simulation Steps			Simulation Steps		
	5	15	25	5	15	25	5	15	25
1	329.44	992.84	1650.15	336.83	1007.20	1664.78	339.45	1035.65	1696.25
2	170.15	516.60	876.86	171.41	518.84	887.58	181.05	550.10	903.03
4	93.59	274.40	469.39	94.13	280.78	467.49	97.98	292.45	476.57
8	54.55	158.92	263.41	55.10	163.70	268.17	56.25	169.73	274.64
16	44.87	127.08	208.33	43.35	145.23	214.00	44.51	134.88	220.91

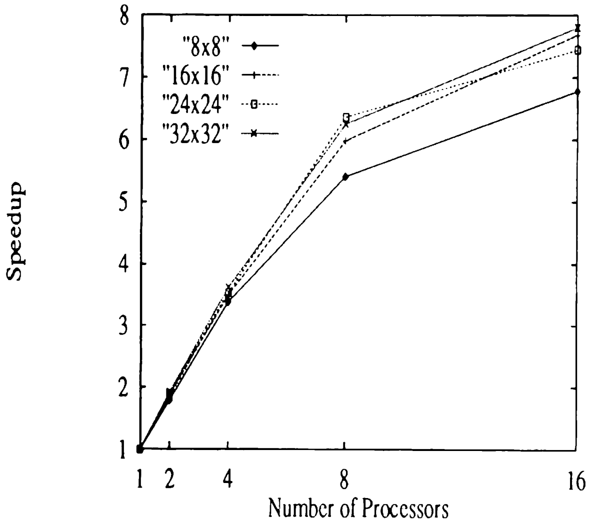

Figure 2. Speedup of the Static-Assignment Battlefield Simulator on the BBN Butterfly with different Battlefield Sizes
No. of Simulation Steps = 15
No. of Combat Units on Each Side = 50

The inverse relation between the processors and execution time, once again, can also be discerned. However, the speedup is decreasing, at a very slow rate, as the number of combat units are increasing. Figure 2 shows the speedup obtained, plotted against the number of processors, in experiments with 50 combat units on each side initially and simulated for 15 steps, as the size of the battlefield is varied. Even though the speedup increases as the battlefield size increases, the speedup values are smaller than those obtained by the simulator on the iPSC/2 as the shared-memory used in the simulator on the Butterfly could suffer from memory contention in addition to the load imbalance and communication overhead inherent in the static assignment utilized.

3.3 Dynamic-Assignment Simulator on the BBN Butterfly

Table 3 lists the execution times obtained for a 32x32 hex battlefield, as the number of processors is varied, the number of combat units is varied with 50, 100 and 250 units on each side and for 5, 15, and 25 step simulations. The execution times of the simulator running on 1, 2, 4, 8, 16, and 27 processors only are provided.

The execution time, as can be seen from Table 3, is almost linearly proportional to the number of simulation steps and grows slowly with the number of combat units. Also, the execution times of the two simulators on the Butterfly are comparable for the same input configuration when the number of processors is one. Hence, the cost of dynamically assigning hexes to processors is very small. The inverse relation between the execution time and the number of processors can also be discerned from Table 3. In addition, the speedup does not seem to vary with the number of combat units. Hence, the dynamic assignment seems to adapt to the unpredictable movement of the combat units very well. In Figure 3, we plotted the speedup, against the number of processors, of the dynamic-assignment simulator in experiments with 50 combat units on each side and 15 simulation steps as the battlefield size is varied. Compared to the static ones, the dynamic-assignment simulator provides more speedup that is increasing with the size of the battlefield.

3.4 Analysis

To analyze the execution times of the battlefield simulators, we used the mathematical package *Mathemat-*

TABLE 3: Execution Times, in Seconds, of the Dynamic-Assignment Simulator on the BBN Butterfly for a 32x32 hex Battlefield

	Combat Units on Each Side								
	50			100			250		
Processor	Simulation Steps			Simulation Steps			Simulation Steps		
	5	15	25	5	15	25	5	15	25
1	340.55	1026.18	1706.21	344.86	1042.13	1721.60	354.67	1085.15	1766.20
2	173.48	520.16	865.27	175.65	528.60	873.11	182.02	549.67	895.02
4	90.44	268.38	446.88	91.96	272.20	447.45	93.59	281.75	457.99
8	49.47	147.52	240.14	50.54	146.59	239.43	51.00	151.76	246.94
16	27.26	81.99	137.50	27.76	83.12	139.28	28.56	87.30	141.83
27	19.98	60.42	100.47	20.31	61.35	101.94	20.83	64.65	104.89

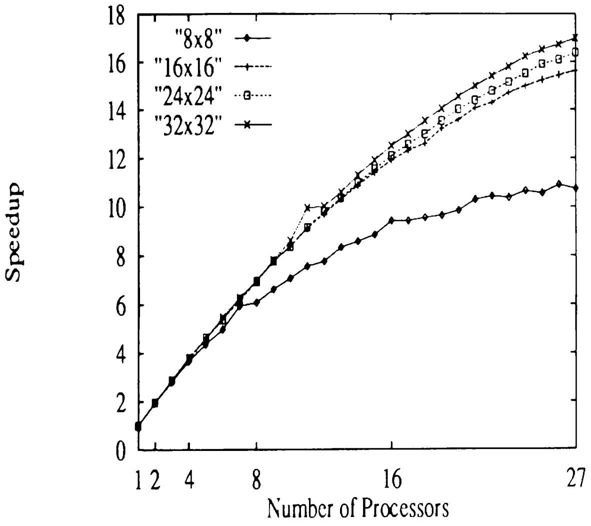

Figure 3. Speedup of the Dynamic-Assignment Battlefield Simulator on the BBN Butterfly with different Battlefield Sizes
No. of Simulation Steps = 15
No. of Combat Units on Each Side = 50

ica for curve-fitting in the data obtained. *Mathematica* allows fitting polynomials into given data using least-squares optimization technique (Wolfram 1988).

For the execution times of the simulators employing static mapping, the following equation is obtained from *Mathematica*:

$$T_p = \alpha_1 + (\beta_1 \times steps) + \left(\delta_1 \times \frac{steps \times area}{p}\right) + \left(\varepsilon_1 \times \frac{steps \times units}{p}\right) + (\gamma_1 \times units),$$

where *area* is the number of hexes in the battlefield, *units* is the total number of combat units, p is the number of processors, *steps* is the number of simulation steps and T_p is the execution time in seconds. The constants α_1, β_1, γ_1, δ_1 and ε_1 depend on the machine.

The constant α_1 represents the start-up time required by the program. Time for allocation and initialization of local variables contribute the constant term. Each processor simulates for the given number of steps using a control loop. The term $(\beta_1 \times steps)$ represents the overhead due to the control loop and the synchronizations within each simulation step. The computation due to the $(area/p)$ hexes, assigned to each processor, for the given number of steps will be proportional to $(steps \times (area/p))$ and contributes the third term to the execution time. The combat units also get distributed among the processors and hence the processing time to simulate the actions of the combat units for all steps which accounts for the fourth term. In the static mapping of the battlefield onto the processors, there is no guarantee that the load due to the combat units will be balanced evenly among the processors (Nicol 1989). Even though the initial unit-assignments in the experiments placed the combat units randomly in the battlefield, the combat units move unpredictably as battle progresses and can create load imbalance among the processors; hence the presence of the term $(\gamma_1 \times units)$. However, the speedup curves show that the effect of the load imbalance is insignificant as speedup did not deteriorate with increase in combat units. Strictly speaking, the initial distribution of the combat units affects the load imbalance and hence γ_1.

The first, second and fifth terms in the equation do not depend on the number of processors and form the sequential portion of the execution time. The other terms form the parallelizable portion. The sequential portion is independent of the size of the battlefield and hence, as battlefield size increases, parallelism increases. The number of combat units contributes to the sequential portion indicating that as the number of combat units increase, the load imbalance increases and the speedup decreases. The values, for the data obtained from the simulator on the iPSC/2, for α_1, β_1, δ_1, ε_1 and γ_1 are 5.05, 0.38, 0.011, 0.017 and 0.25, respectively.

For the data on execution times of the simulation program employing dynamic mapping, the following equation was obtained from *Mathematica*:

$$T_p = \alpha_2 + (\beta_2 \times steps) + \left(\delta_2 \times \frac{steps \times area}{p}\right) + \left(\varepsilon_2 \times \frac{steps \times units}{p}\right),$$

where α_2, β_2, δ_2 and ε_2 are constants.

The main difference between this equation and the one obtained for static mapping is the absence of a term dependent on combat units but independent of the number of processors. Since hexes are assigned to the next available free processor, the computational load due to the hexes and also the combat units will be evenly distributed among the processors. Hence, the sequential portion of the execution time is independent of the size of the battlefield and the number of combat units. In the equation obtained, the first two terms form the sequential portion of the execution time. Once again, the start-up time forms the first term and the synchronization needed in each step contributes the second term. The third and fourth terms represent the computational load due to hexes and combat units, respectively. The values we got, for the data obtained from the simulator on the Butterfly, for α_2, β_2, δ_2 and ε_2 are 7.89, 0.7, 0.063 and 0.021, respectively.

4 CONCLUSIONS

We presented static and dynamic processor assignments for parallel battlefield simulation. Our static assignment strategy showed better speedup than earlier domain partitioning approach because no replicated computation took place. Since the simulator on the Butterfly using static assignment did not give better speedups than the one on the iPSC/2, the message exchanges in the hypercube did not suffer from any contention. The speedups reported for the simulator on the iPSC/2, which are better than those reported earlier (Nicol 1989), show that the communication overhead is tolerable in our finer-grained domain decomposition.

The dynamic-assignment simulator on the Butterfly has better speedup, as expected. It showed speedups almost independent of the number of combat units indicating that the assignment strategy is adapting well to the load imbalances created by combat unit movement. The controlled replication of the state space obviates the need for any software locks to maintain the data consistency. Software locks (Gilmer, Hartwig, and Kokinakis 1986), cause memory contention and limit the scalability of the parallel battlefield simulation. Memory contention does not seem to be a problem in the dynamic-assignment battlefield simulator as the speedups did not suffer when the processors are increased.

The execution times obtained from varying the battlefield size, the number of combat units, the number of simulation steps and the number of processors were used to find analytical equations. As expected, the equations show that as the resolution of the battlefield is improved, that is, the battlefield is divided into more hexes, the speedup improves. The equation obtained for static assignment shows that the sequential portion of the execution time is dependent on the number of combat units indicating load imbalance. For the dynamic assignment, the sequential overhead is independent of the battlefield size and the number of combat units. Hence, massive parallelism can be exploited in battlefield simulation (Gustafson 1988). As the complexity of the simulation grows with added realism, the parallelizable portion of the execution time increases improving the granularity of the parallel tasks and hence speedup.

A formal analysis of domain decomposition was performed by Nicol and Saltz for the restricted case of a one-dimensional domain with the assumptions: (i) the workload is a convex function of distance, (ii) the workload is a second-order stationary process, and (iii) the communication and synchronization costs can be ignored (Nicol and Saltz 1990). A similar analysis for a two-dimensional case would be valuable in establishing the effectiveness of domain decomposition technique for two-dimensional workloads.

ACKNOWLEDGMENTS

This research was supported by US Army's PM-TRADE under contract N61339-88-G-0002 and National Science Foundation grant CDA-89-20890.

REFERENCES

Chorafas, D. N., and H. Steinmann. 1990. *Supercomputers*. New York: McGraw-Hill.

Deo, N., and M. Medidi. 1992. Parallel battlefield simulation on Intel's iPSC/2 and BBN Butterfly. Technical Report CS-TR-92-07, Department of Computer Science, University of Central Florida, Orlando, Florida.

Gilmer, J. B. 1986. Statistical measurements of the CORBAN simulation to support parallel processing. Technical Report BDM/ROS-86-0326, BDM Corporation, Arlington, Virginia.

Gilmer, J. B., G. Hartwig, and L. Kokinakis. 1986. Parallel entity centered simulation on the Butterfly computer. In *Proceedings of the 1986 International Conference on Parallel Processing*, ed. K. Hwang, S. M. Jacobs, and E. E. Swartzlander, 793–795. Institute of Electrical and Electronics Engineers, New York, New York.

Gilmer, J. B., and J. P. Hong. 1986. Replicated state space approach for parallel simulation. In *Proceedings of the 1986 Winter Simulation Conference*, ed. J. R. Wilson, J. O. Henriksen, and S. D. Roberts, 430–433. Institute of Electrical and Electronics Engineers, New York, New York.

Gustafson, J. L. 1988. Reevaluating Amdahl's law. *Communications of the ACM.* 31:532–533.

Klahr, P., J. W. Ellis, W. D. Giarla, S. Narain, E. M. Cesar, and S. R. Turner. 1986. TWIRL: tactical warfare in the ROSS language. In *Expert Systems: Techniques, Tools and Applications*, ed. P. Klahr and D. A. Waterman, 224–268. Reading: Addison-Wesley.

Malmberg, A. F., S. A. Hutchinson, R. D. Riggin, and W. N. Friend. 1984. Modeling tactical communication with QSIM. In *Proceedings of the 1984 Winter Simulation Conference*, ed. S. Sheppard, U. W. Pooch, and C. D. Pegden, 729–734. Institute of Electrical and Electronics Engineers, New York, New York.

Nicol, D. M. 1988. Mapping a battlefield simulation onto message-passing parallel architectures. In *Distributed Simulation*, ed. B. Unger, and D. Jefferson, 141–146. Society for Computer Simulation International, San Diego, California.

Nicol, D. M. 1989. Dynamic remapping of parallel time-stepped simulations. In *Distributed Simulation*, ed. B. Unger, and R. Fujimoto, 121–125. Society for Computer Simulation, San Diego, California.

Nicol, D. M., and J. H. Saltz. 1990. An analysis of scatter decomposition. *IEEE Transactions on Computers.* 39:1337–1345.

Prasad, S. 1990. Efficient parallel algorithms and data structures for discrete-event simulation. Ph.D. dissertation, Department of Computer Science, University of Central Florida, Orlando, Florida.

Weiland, F., L. Hawley, A. Feinberg, M. D. Loreto, L. Blume, P. Reiher, B. Bechman, P. Hontalas, S. Bellenot, and D. Jefferson. 1989. Distributed combat simulation and time warp: the model and its performance. In *Distributed Simulation*, ed. B. Unger, and R. Fujimoto, 14–20. Society for Computer Simulation International, San Diego, California.

Wolfram, S. 1988. *Mathematica, a system for doing mathematics by computer.* Reading: Addison-Wesley.

AUTHOR BIOGRAPHIES

NARSINGH DEO is the Charles N. Millican Chair Professor of Computer Science and Director of the Center for Parallel Computation at University of Central Florida, Orlando. His research interests include parallel processing, combinatorial algorithms, and graph theory. A Fellow of IEEE, Dr. Deo has authored four textbooks and over 80 research papers.

MURALIDHAR MEDIDI received an M.Tech. in Computer Engineering from Indian Institute of Technology, Kharagpur, in 1986. Currently, he is a Ph.D. student at University of Central Florida, Orlando. His research interests include data structures in parallel processing and algorithmic graph theory.

SUSHIL PRASAD has done his Ph.D. in Computer Science from University of Central Florida, Orlando, in 1990. Currently, he is an Assistant Professor at Georgia State University, Atlanta. His research interests are parallel algorithms and data structures, parallel simulation, graph algorithms, and complexity theory.