Managing Flocking Objects with an Octree Spanning a Parallel Message-Passing Computer Cluster

Thomas E. Portegys, Kevin M. Greenan
Illinois State University
portegys@ilstu.edu, kmgreen2@ilstu.edu

Abstract

We investigate the management of flocking mobile objects using a parallel message-passing computer cluster. An octree, a data structure well-known for use in managing a 3D space, is adapted to "span" the cluster. Objects are distributed in the tree, and partitions of the tree are distributed among the processors in such a way that a minimum of global information is required to be shared by the processors. When objects move, the tree is modified accordingly; this in turn may cause partitions to migrate processors. Two constraints drive the distribution algorithm: (1) minimizing message traffic by clustering nearby objects on the same processor, and (2) processor load-balancing. Boids, flocking artificial life forms, embody the objects in this study. The performance of the system is measured in terms of the inter-processor message traffic as a function of the number, interactivity, and mobility of objects. An application of the scheme allows external clients to view objects in specified spatial loci.

Keywords: message-passing parallel computer, octree, flocking behavior, boids.

1. Introduction

Many systems involve large numbers of mobile and spatially related components. Examples include simulating molecular changes in chemical reactions, weather modeling, air and fluid dynamics, population modeling, forest fire simulation, and networked gaming. These systems require the sort of massive computational power that parallel processors can provide in an economical fashion.

We chose a message-passing parallel computer cluster as our platform. Message-passing machines are typically less efficient than shared-memory processors for tasks involving relatively small numbers of tightly connected objects, and in turn show superior performance and scaling for tasks involving large numbers of loosely coupled objects [8]. One of the aims of this project is to investigate the conditions under which the use of a message-passing system is effective. To this end, we measure the performance of the system in terms of the inter-processor message traffic as a function of the number of objects and their mobility.

A number of investigations of N-body tasks on message-passing parallel clusters appear in the literature [3,4,7]. In our study, boids [6], flocking artificial life forms, are used as test objects. A boid moves about in a swarming fashion that requires knowledge other boids in its vicinity. This produces group flocking movements that are characteristic of animal and human behavior in contrast to typical N-body movements caused by force-fields. The movement of flocks across processor boundaries provides an opportunity to study the processing and message-passing capabilities of the system, as well as to investigate the effectiveness of load-balancing.

1.1. Spanning octree

An octree, a data structure well-known for use in managing 3D space, is adapted to "span" the cluster. In our octree, space is recursively divided into smaller and cubic partitions such that terminal cubes usually contain a single object. The tree is partitioned by orthogonally bisecting space and assigning volumes to each processor in the form of bounds. Figure 1 depicts a 2D (quadtree) view of a tree spanning a space containing a number of objects.

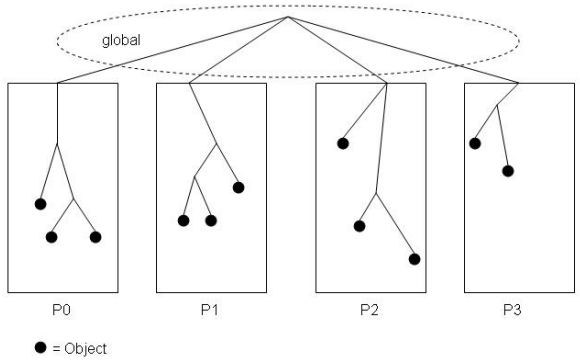

Figure 1. Initial Spanning Tree

In the figure, the partition bounds are globally known to processors P0 through P3. This is accomplished by replicating the bounds so that each processor has its own copy. The goal is to provide a fast determination of which partitions of the octree are managed by which processor so that objects in the corresponding volumes of space can be accessed. Each processor manages a single volume of space and the objects that it contains. The specific details of the objects in these local spaces, including the configurations of the sub-trees that track them, are known only to each processor.

1.2. Migrating objects

Figure 2 shows objects O1 and O2 migrating to the space owned by processor P3. This involves at a minimum messages from P2 to P3 to insert the migrating objects in the target space. P2 then deletes its copy of the objects.

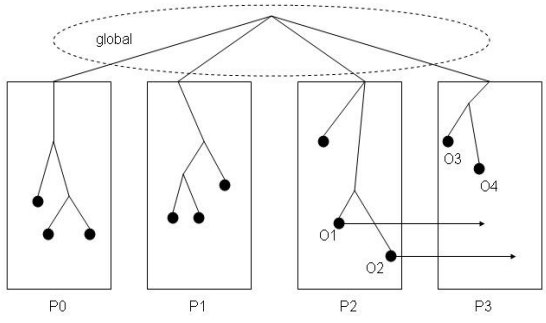

Figure 2. Migrating objects

1.3. Load-balancing

Two constraints drive the distribution algorithm: (1) minimizing message-passing by clustering nearby objects on the same processor, and (2) processor load-balancing. Figure 3 shows the result of load-balancing that has caused the global space to be repartitioned as a result of the migration of objects O1 and O2. This has caused the redistribution of objects O3 and O4 to P2, and forced an update to the global bounds. We use a variation of an orthogonal recursive bisection algorithm [5] to partition space.

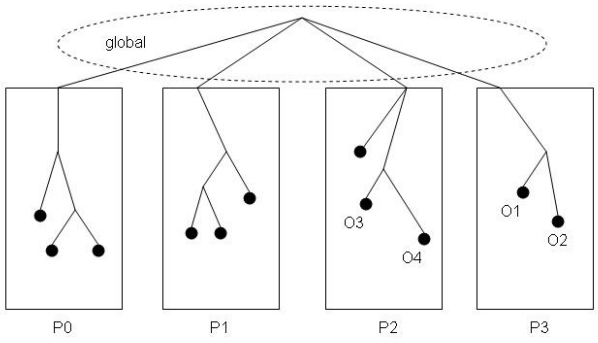

Figure 3. Repartitioned space

1.4. Object interaction

In contrast with typical N-body systems, a boid does not often interact with every other boid. A boid's movement depends only on the movements and positions of nearby boids. Because of this, a straightforward octree search is sufficient to implement efficient proximity checking, as opposed to using a cumulative scheme such as the multipole method [1,2].

Figure 4 illustrates a situation in which objects O1 and O3 reside in space managed by processor P0, and object O2 resides in space managed by P1. Consider proximity checking for O1. In the case of an O1-O3 interaction, the checking can be entirely local to P0. However, since O1 extends into space owned by P1, and the contents of this space are unknown to P0, a message to P1 must be sent containing O1's position so that P1 may conduct the proximity checking in its local space.

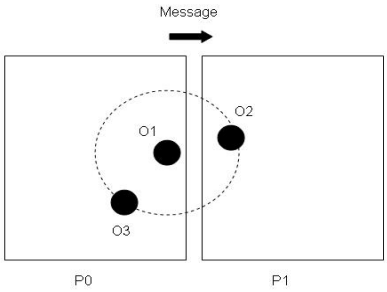

Figure 4. Proximity checking

1.5. Viewports

An application of the scheme allows external clients to view objects in specified spatial loci. This is accomplished by designating one of the processors as a viewport gateway. Alternatively, a dedicated machine may perform the gateway function. Clients connecting to the gateway specify a viewing frustum, which is essentially a bounding box. The gateway then makes appropriate searches on the various processors to obtain information about objects contained in the viewing frustum. A list of these is returned to the client, allowing a graphical view of a volume of space to be rendered.

2. Procedure

Our platform is the Applied Computer Science Department's Beowulf machine, a cluster of 16 SUN Ultra 10 workstations running SuSe Linux, connected by a 10mbps Ethernet. The software is the C++ programming language and PVM (Parallel Virtual Machine) to provide the message-passing infrastructure. The OpenGL graphics language is used to exercise the viewport feature.

The partition algorithm requires that the number of processors be a power of 8, yet for obvious reasons this is not a practically achievable machine configuration. The solution was to implement processors as virtual entities and distribute processors among physical machines in a clustered manner.

The boids code was initially obtained from an internet source. After a measure of re-writing and parameter tuning we obtained satisfactory flocking behavior: a variety of dynamically changing flock sizes.

A PVM program is a master-slave process configuration. Our master process spawned slave processes representing virtual processors on specified machines, initializing them with their bounds and a random distribution of boids.

Updating processor bounds for load-balancing is done by the master using information from the slave processors. Each processor reports its current load (number of boids) and the position of the median boid. The master then re-partitions based on these weighted boid positions.

After initialization, the master enters a loop, an iteration of which constitutes the following cycle:

1. Broadcast an *aim* message to slave processors causing them to determine the next position of each boid. This entails intra-processor and cross-processor searches not involving the master.
2. Broadcast a *move* message, causing processors to move boids to their new positions, possibly involving cross-processor insertions and deletions, also not involving the master. Insertions are unacknowledged for efficiency reasons.
3. If load-balancing, broadcast a *report* message, causing the processors to send back their load and median information. The master computes new bounds and broadcasts them in a *balance* message.
4. If gathering statistics, broadcast a *stats* messages and gather results.
5. If in viewing mode, broadcast the viewing planes in a *view* message and gather the results. Each processor determines which searches its octree for boids falling within the viewing box.

The cycle steps are synchronized; each step is completed before moving to the next. This means that the master waits for all processors to respond before issuing the next command. This can only happen after all searching and insertion activity for a particular step is completed by the slaves.

The viewing capability is implemented as a separate thread within the master process. This allows a user to navigate through space, selectively viewing boids, or to run in non-interfering "blind" mode.

3. Results

Figure 5 is a graphical depiction of a simulation of the system. Here, a number of mobile point objects are shown in their octree volumes.

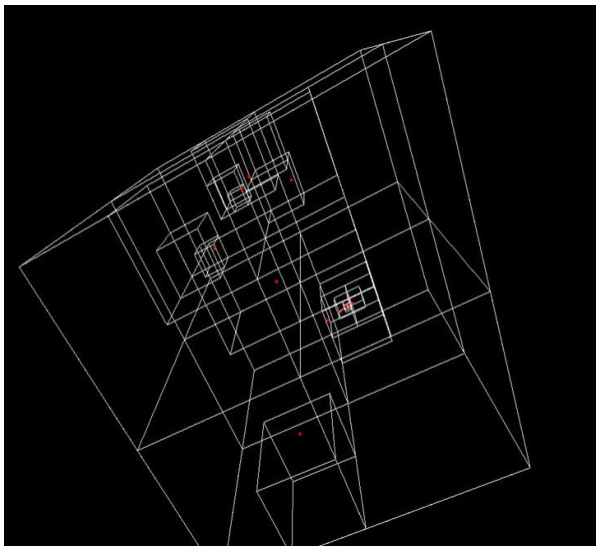

Figure 5. Octree simulation

The independent variables were: number of boids (25, 50, 100, 200), number of machines (4, 8), and load-balancing (on/off). Unfortunately, due to hardware problems we were not able to use the entire 16 processor cluster. The dependent variables for which data was gathered were: load (boids) per machine and message traffic. Each trial was run for 1000 cycles. The boids had an interaction range of 5 units. As the number of boids increased, the spatial dimension were increased: 25 boids in a 15x15x15 volume, 50 in 20x, 100 in 25x, and 200 in 30x.

Figures 6 and 7 show the average and standard deviation boid load for 8 and 4 machine configurations, respectively, under no load-balancing (NLB), and load-balancing (LB) conditions. The flocking aspect of the boids can be seen in the non-uniform distribution indicated by the standard deviation.

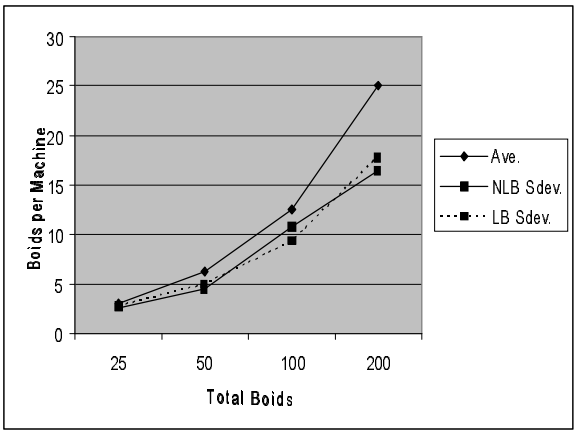

Figure 6. Load for 8 machines

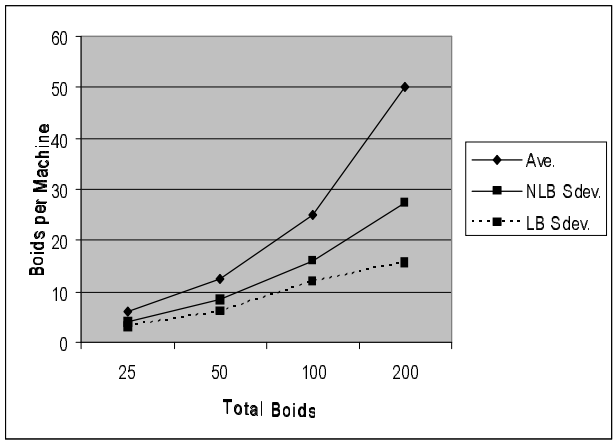

Figure 7. Load for 4 machines

Figures 8 and 9 show the message traffic for the 8 and 4 machine configurations. Notable here is the effect of load-balancing, which causes a significant decrease in message traffic, although not a correspondingly large decrease in burstiness as indicated by the standard deviation.

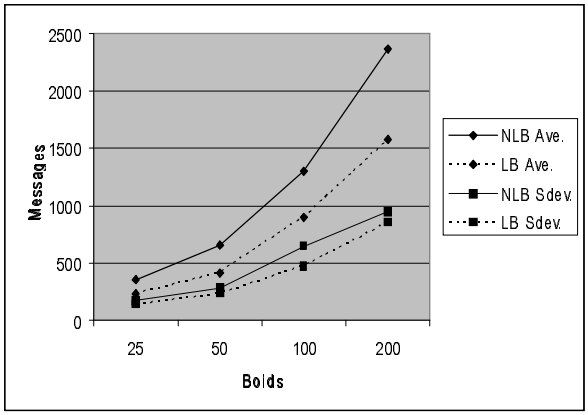

Figure 8. Message traffic for 8 machines

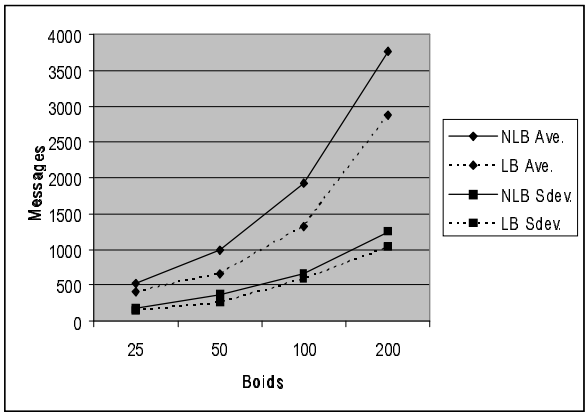

Figure 8. Message traffic for 4 machines

4. Conclusion

The overall observation is that load-balancing can be effective in reducing message traffic for flocking objects. The cost for this improvement in our scheme is an additional 2 steps in the processing cycle.

For future work, we propose to investigate decentralized load-balancing schemes to avoid the overhead cost. In addition, processor load could consist of factors other than simple numbers of objects. For example, the state of objects may be a viable factor. In a forest fire simulation, burning areas would take more computation resources, and thus these objects might "weigh" more heavily. In addition, in a cluster of heterogeneous processors, the resources of each processor could be taken into account for load-balancing.

The code is available at:
www.acs.ilstu.edu/faculty/portegys/research.html

5. Acknowledgements

The authors wish to especially thank Chris McBride for many valuable ideas and contributions. Thanks also to Andy Thayer and Tesh Shah for their insights.

6. References

[1] Anderson, C. R., "An implementation of the fast multipole method without multipoles", SIAM J. Sci. Stat. Comput. 13 (1992) 923.

[2] Greengard, L. Gropp, W.L., "A Parallel version of the Fast Multipole Method", Parallel Processing for Scientific Computing SIAM Conference proceedings, p213-222, 1988.

[3] Hariharan, B. and Aluru, S., "Efficient Parallel Algorithms and Software for Compressed Octrees with Applications to Hierarchical Methods", High Performance Computing - HiPC 2001 8th International Conference, Hyderabad, India, December, 17-20, 2001. Proceedings.

[4] Hu, Y., "Implementing O(N) N-body algorithms efficiently in data parallel languages (High Performance Fortran)", Journal of Scientific Programming, 1994.

[5] Salmon, J.K., "Parallel Hierarchical N-Body Methods", Ph.D. thesis, California Institute of Technology, 1990.

[6] Scientific American: Feature Article: "Boids of a Feather Flock Together", November 2000.

[7] Sun Y., Liang, Z., and Wang, C-L, "A Distributed Object-Oriented Method for Particle Simulations on Clusters", Proceedings of the 7th International Conference on High Performance Computing and Networking (HPCH Europe 1999), April 12-14, 1999, Amsterdam, The Netherlands, 1999.

[8] Wilkinson, B. and Allen M., "Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers", Prentice-Hall, Inc., 1999.