Parallel Implementation of Schooling Behavior Modeling using a KNN Topological Distance Model

Victor A. Colton

Proposal

This project proposes to model basic schooling behavior with a k-nearest-neighbors topological distance model, using a parallel implementation in C and LAM/MPI. Members of the school or swarm (hereafter referred to as agents or Boids) will be modeled using the following simple rules:

1. Move in the same direction as neighboring agents
2. Remain close to neighboring agents
3. Avoid collisions with neighboring agents

Additionally, each agent will only pay attention to the k nearest other agents in the school; a topological distance model. This is in contrast to a metric distance model, where each agent pays attention to all other agents within a given range.

One primary Server process will be responsible for coordinating the activities of several Client processes, which will separately calculate the movements of a queue of agents. The number of agents and processes will be determined at run time and need not be the same.

The agents will initially be placed with random position and velocity within a rectangular area of fixed proportions (also determined at run time) and will not be allowed to travel outside of the area. During each time step, the Server will inform each Client of the current position and velocity of each agent via broadcast. It will then deal out agents from the queue to each Worker one at a time.

When a Client receives an agent to process, it will measure the velocity and positions of the k nearest neighboring agents to determine the new velocity and position of that agent. When a Client is finished with an agent, it will send a return message to the Server with the updated position and velocity of that agent, and then await orders to process the next agent.

The Server will deal out agents in this fashion until all agents in the queue have been processed, at which point it will send the next broadcast of positions and velocities out to the Client, and the cycle will begin again. At each time-step, the Server will flush the current position and velocity data to an output file for data visualization purposes, so that the swarm data can later be compiled into a gif animation. The process will run for a predetermined number of iterations.

The Server will also record the total time for each cycle in order to determine the average time required to process each agent and the average time per cycle. These average times will be measured under varying number for k, the number of agents, the number of processes total, and the number of processes per node on the cluster. Motion will initially be restricted to two dimensions, but may be expanded to include three, time permitting. The timing will be tested on the Beowulf cluster in Eastern Washington University's Computer Science department.

Design

MPI Setup and Communication

The Server is the Rank 0 process and is set up on Node 0. Processes Rank 1 – 9 are Client processes, with one residing on Node 0, and the other eight divided evenly between the other nodes. Each of the 10 processes initializes the queue for the Boids, allocating local memory for a linked list of sufficient size to hold each Boid. The message tags used are DATA and DONE, indicating either Boid data being passed, or the end of current cycle, respectively.

The Server process also determines random starting conditions for each Boid, including random position and velocity. It also allocates memory for a queue of Client structures, to track which Clients are busy and which are available, and which Boid is being processed by which busy Client. Initially, all Client structs are stored in the 'available' queue, and the 'busy' queue is empty. The Client processes allocate additional memory to store the current Boid being processed.

After initialization, barrier synchronization is used to gate the processes between two communication phases. Each time-step is composed of these two phases. The number of time-steps to run is specified as an initial condition by the user.

In the first phase, the Server broadcasts the current position and velocity data of each Boid to the Clients so that they will have a current set of Boid data in local memory during the second phase. In the second phase, Clients receive individual Boids from the server for processing, and send them back when complete. After all Boids have been sent and reported back by the Server, it sends each Client a DONE message to complete the phase, and the cycle begins again.

In phase one, the Server pops a Boid from the queue, packs the data into a message buffer, and broadcasts the DATA message to each Client, then pushes the Boid back onto the queue. The Clients also pop a Boid from the local queue, unpack the data into a Boid struct and then push the Boid onto the queue. This repeats until all Boids have been sent, at which point the phase ends.

In phase two, the Server checks if there are Clients in the available queue and also if there are Boids remaining to be dealt out. If the conditions are met, the Server pops a Client struct from the available queue, pops a Boid from the Boid queue, hands the Boid to the Client struct, pushes the Client onto the busy queue, and then sends a DATA message to the appropriate Client process with the Boid data packed in the buffer. The number of Boids dealt is then incremented.

While there are no free Clients or all Boids have been dealt, the Server checks if there are Boids that have not been reported. If so, the Server probes for a DATA message from a Client and, upon receiving it, finds the Client from the busy queue and returns the updated Boid data into that Client's Boid struct, before pushing the Client onto the available queue and the Boid onto the Boid queue. Once all Boids have been dealt and reported, the Server sends a DONE message to each Client.

During phase two the Clients will probe for both DONE and DATA messages. Upon receiving DATA, it will unpack the Boid into local memory and process it, before returning the DATA result to the Server.

Upon receiving a DONE message, it will leave phase two and return to the start of the cycle. This cycle will repeat until the predetermined number of iterations has been completed.

Boid Handling Rules

Three simple rules are used to modulate Boid behavior, in addition to a position bounding or wrapping rule. Each Boid uses a topological distance model in which it monitors its K nearest neighbors, where K is a constant specified by the user as an initial parameter.

Rule 1 instructs each Boid to navigate toward the perceived center of mass of its neighbors. During each time-step, each Boid will average the positions of the K nearest neighbors and then navigate toward that point.

Rule 2 instructs each Boid to avoid collisions with its neighbors. During each time-step, each Boid will determine if any neighbors are within a predefined buffer distance. For each neighbor within that distance, the Boid will move directly away from that neighbor's position.

Rule 3 instructs each Boid to align velocities with each of its neighbors. During each time-step, each Boid will average the velocities of the neighboring Boids and add a predefined fraction of the average velocity to its own velocity.

Rule 4 is a position bounding rule that instructs each Boid to limit its position to the view area. This can be accomplished in one of two ways. First, Boids may be instructed to accelerate back into the view area after having left the area by monitoring the position and then increasing the velocity of the Boid in the opposite direction after it has left the area. Second, a wrap-around rule can be used to place each Boid that leaves the view area at the opposite end of the area after having left it.

Together, these simple rules lead to emergent flocking or schooling behavior. Additionally, it is possible to specify a field of view for each Boid, such that it will only monitor neighbors that it can 'see'. By adjusting the parameters in the above rules, the behavior can range between flocking and swarming.

References

[1] Charles M. Macal and Michael J. North. 2005. Tutorial on agent-based modeling and simulation. In *Proceedings of the 37th conference on Winter simulation* (WSC '05). Winter Simulation Conference 2-15.

[2] Charles M. Macal and Michael J. North. 2009. Agent-based modeling and simulation. In *Winter Simulation Conference* (WSC '09). Winter Simulation Conference 86-98.

[3] Robert Siegfried, Axel Lehmann, Rachid El Abdouni Khayari, and Tobias Kiesling. 2009. A reference model for agent-based modeling and simulation. In *Proceedings of the 2009 Spring Simulation Multiconference* (SpringSim '09). Society for Computer Simulation International, San Diego, CA, USA, , Article 23 , 8 pages.