Creating a Turn-Based Conflict Resolution Simulator FIX TEAMS INVADING A NODE

Noah Zimmt and Dakota Szabo

May 8, 2012

Abstract

In this paper, we describe a territory-conquest simulation modeled on the classic board game *RISK* and present a parallelized method for efficiently simulating multiple rounds of the model. Players compete on a gameboard represented by an undirected graph, with vertices and edges representing territories and their borders, respectively. Using MPI, we distributed the computation and resolution of player actions between multiple compute nodes in order to acheive the speed neccessary to simulate massive games in a reasonable amount of time. This was acheived via a randomized algorithm that partitioned the computational work between nodes, combined with by a pair of message-passing cycles that communicated the results of the computations to each node. In performance testing, we found this method to scale reasonably with the size of the input graph, with expected variation depending on the graph shape (complete graphs performing better than stars, etc).

Introduction

Our simulation is designed as a territory-conquest simulator, with multiple teams competing against one another to dominate the entirety of the gameboard. The gameboard itself is laid out as an undirected graph in which territories correspond to vertices and borders between territories correspond to edges. Each territory has a troop count, representing the number of soliders currently stationed in that territory, as well as a team affiliation, representing the team currently controlling it. Gameplay consists of a series of turns, each with two stages. During the troop placement stage, each territory determines how many troops from its army to place on each of its borders. Troops placed on a border with another territory may either be attacking that territory or defending against an attack. After all troops have been placed, the game proceeds to the conflict resolution stage. Along every edge of the graph (every border between territories), a battle takes place if at least one of the territories chose to attack during the placement stage (if both teams defend, no battle takes place and all troops survive). Battles are resolved in the following fashion: Territories flip a number of coins depending on the action they chose (attack or defend) and the amount of troops they have stationed on the edge where the battle is taking place. Attacking territories flip one coin per solider; defending territories flip two coins per solider. After all coins are flipped, the number of heads for each side is tallied and subtracted from the number of opposing soliders on that edge (each coin flip is a gunshot; each heads is a successful hit and kill). This coin flipping process repeats until one or both territories have no further soliders remaining on that edge. After all battles are resolved, surviving attacking troops proceed into the territory they were attacking, while defending troops retreat back to the territory they were defending. The team with the largest number of troops present in a given territory after the battle phase becomes the new owner of that territory, and troops from other teams currently occupying the territory are considered killed. Territories are then given a 'recruitment bonus' (a number of troops added to their troop count) depending on their current troop count - this is currently 20 percent but is easily configurable within the rules of the game. The turn is then considered completed, and the game proceeds back to the troop placement stage. The game ends if, at the end of a turn, one of three conditions is met. If every territory is controlled by a single team, that team is considered the winner. Likewise, if only one team still has troops remaining on the board (territories may lose all troops but remain unconquered), it is considered the winner. Finally, if no troops remain on the board, the game is considered a draw and the gameboard begins recovering from the armageddon that has transpired (for the sake of this simulation, we do not model the reconstruction of humanity).

Implementation

To process events in parallel, we partition the data such that each compute node has a subset of the data used to represent the entire simulation. In order to ensure the simulation information is easily partitioned and circulated between nodes, we store it in several matrices. The gameboard graph itself is represented as a 2-D adjacency matrix where $adjMatrix[x][y] = 1$ if there is an edge between territory x and territory y. We also store a 1-D matrix of troop counts as well as a 1-D matrix of team affiliations ($matrix[x]$ equals the troop counts/team affiliation for territory x). Each node contains a slice of the adjacency matrix, giving it the full adjacency data for a subset of the total vertices in the graph. We referto these vertices/territories as being 'owned' by the compute node. Each node also contains the entirety of the troop count and team affilication matrices as we determined that incurring extra memory overhead to store the entirety of these matrices on each compute node was preferrable to the performance overhead of passing around slices of two extra matrices. During the beginning of each simulation turn, each node uses its slice of adjacency data and the troop counts/team affiliation data to determine the edge-troop distributions for each vertex that it owns. This data is stored in a second matrix slice, which we referred to as the edge activity matrix. The edge activity matrix is similar to the adjacency matrix, where $edgeActivityMatrix[x][y] = T$ represents $|T|$ troops from territory x allocated to the edge (x, y) in the graph. If $T > 0$, x is attacking; if $T <= 0$, x is defending. From here, the simulation step can be divided into three main phases - the conflict resolution phase, data circulation phase, and the final inner-vertex conflict resolution phase.

Conflict Resolution

During the conflict resolution phase, we implement a round-robin inter-node communication schema that circulates each slice of edge activity to every compute node in turn. Each node scans each slice of *edgeActivity* that it receives to see which territories attacked/defended against it this turn. Since for any given conflict on an edge (x, y) either the compute node that owns x or the compute node that owns y could theoretically compute the battle resolution, we use a randomized algorithm to balance the computational work between nodes. Each territory flips a coin at the start of the turn. Upon entering a battle, the coin flips are compared. If the coin flips are the same, the compute node who owns the territory with a lower ID number is responsible for the calculation; if the coin flips are different, the compute node who owns the territory with a higher ID number is responsible for the calculation. Since this problem of dividing edge conflicts between owner processes reduces to the NP-Complete problem of edge coloring, we devised the aforementioned approximate solution to reasonably distribute the work in a quick, low-cost fashion. This coin flipping divides work evenly and also ensures that no redundant calculations are performed.

Data Circulation

After all the battle data is calculated, each territory needs to be made aware of the results of battles that it was a part of. Since the battle computation may not have occurred on the compute node that owns the territory, it is neccessary to again circulate data amongst the nodes. To do this, we again employ a round-robin passing schema that we referred to as the *report card scheme*. Each compute node creates blank a 'report card' about all the territories that it owns and fills in the data that it has about them. The report card itself is another matrix slice similar to *edgeActivity*, where $reportCard[x][y] = T$ represents $|T|$ troops from x's army surviving in the battle on edge (x, y) and advancing to y. If $x = y$, the implication is that x defended and surviving troops retreated back to x. After filling in its own data, the compute node passes the card to its right-hand neighbor (for simplicity, we visualize the compute nodes in a circular arrangement). The neighbor then fills in the information that *it* has about the territories on the report card and again hands the card off to the next node. This process continues until the report card arrives back at its owner, ensuring that at the end of the round-robin, each compute node's report card will have been filled in completely by each of the other compute nodes.

Inner-Vertex Conflict Resolution

Now that each compute node has all the relevant battle data pertaining to the territories that it owns, it can determine who (if anyone) has conquered these territories. For each territory it owns, it computes the total remaining number of troops from each team currently occupying that territory. The team with the maximum number of troops occupying the territory becomes the new owner of the territory. At this point, each compute node now has a new set of data about each of its territories: team membership and troop counts. Each compute node communicates this

data to all other compute nodes via MPIAllreduce. Simple checking is performed to see if the game is in an end state, and then the next simulation round begins.

Related Work

Proin scelerisque urna et velit rutrum non feugiat nibh luctus. Aliquam posuere viverra lectus ut varius. Nam at erat tellus. Donec sed consectetur felis. Praesent at justo sit amet est vehicula pharetra viverra et magna. Nulla dignissim consectetur facilisis. Vestibulum a dui ligula. Nullam ornare sollicitudin molestie. Integer fringilla lacus ut metus lobortis mollis. Fusce magna velit, pulvinar a molestie vel, cursus et lorem. Phasellus semper pellentesque turpis eget faucibus. Proin at neque eu neque elementum sollicitudin vitae sit amet nunc. Etiam nec erat ut enim sagittis facilisis. Curabitur sollicitudin convallis arcu sit amet dignissim. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Performance Analysis

Proin scelerisque urna et velit rutrum non feugiat nibh luctus. Aliquam posuere viverra lectus ut varius. Nam at erat tellus. Donec sed consectetur felis. Praesent at justo sit amet est vehicula pharetra viverra et magna. Nulla dignissim consectetur facilisis. Vestibulum a dui ligula. Nullam ornare sollicitudin molestie. Integer fringilla lacus ut metus lobortis mollis. Fusce magna velit, pulvinar a molestie vel, cursus et lorem. Phasellus semper pellentesque turpis eget faucibus. Proin at neque eu neque elementum sollicitudin vitae sit amet nunc. Etiam nec erat ut enim sagittis facilisis. Curabitur sollicitudin convallis arcu sit amet dignissim. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Conclusion

Nullam tortor mi, volutpat quis auctor non, aliquam id tellus. Morbi a massa libero, id tempus mi. Aliquam et sapien non est mollis interdum. Cras felis diam, luctus eget pellentesque ut, lacinia nec dui. Aenean eu orci neque, eu volutpat enim. Sed eleifend fringilla turpis, sed pharetra libero mollis non. Quisque ultrices, purus ut sagittis euismod, orci risus venenatis lacus, a auctor diam augue vitae dolor. Duis eleifend pulvinar enim ac mollis. Nulla auctor metus vel turpis consequat aliquam.

Future Work

Quisque velit neque, ullamcorper non iaculis sed, ornare pellentesque nulla. In pharetra cursus imperdiet. Duis nulla justo, bibendum et consequat congue, condimentum et turpis. Sed tempus, odio ut posuere volutpat, quam lacus interdum odio, ac porttitor elit turpis vitae turpis. Mauris eu ultrices augue. Sed fermentum elementum tristique. Nam ipsum ipsum, rhoncus vitae cursus suscipit, volutpat eu felis.