Load Balancing a 2D Cellular Automata Over a Cluster

Jared DuPont
Winona State University
Computer Science
jdupont13@winona.edu

ABSTRACT

Cellular automata can be used to simulate various physical situations such as population or bacteria growth, but can get very computationally intensive as they grow larger. This paper demonstrates an algorithm to distribute the workload of a 2D cellular automaton across a computing cluster. This is done by dynamically partitioning the regions that each computer is responsible for simulating such that each region has roughly the same work load. This algorithm utilizes all of the computational power of the cluster avoiding some of the computers wasting time being idle. An example cellular automaton rule set is used that purposefully creates concentrated areas of high work load and areas of low work load to better demonstrate the algorithm. Small simulations are not effectively distributed across the cluster due to network overhead.

KEYWORDS

Cluster, Cellular Automata, Load Balancing

1 INTRODUCTION

A cellular automaton (CA) is a method to computationally model a variety of problems such as population or bacteria growth, cryptology, and more. They work by taking as input an environment, an initial state, and a set of rules that dictate how the simulation will progress or update. The environment is typically a grid of cells represented by an array, but they can also be any number of dimensions. The initial state is usually decided by the person running the simulation and is created to test something, for example a single bacteria can be placed somewhere inside the simulation to test how it grows or spreads. In every update cycle or step all the cells look at their neighbors or adjacent cells and then update based on the rule set, an example rule would be that if a cell is surrounded by too many neighbors then it should die from over crowding. With the combination of the rules and initial state of the grid, a CA can generate anywhere from simple patterns to incredibly complex ones. For the purpose of this research a set of rules has been created for a 2D grid with the goal of generating pockets of high intensity areas that require a lot of computation. Area in this case refers to a rectangular subsection of the entire simulation. This will help demonstrate the algorithm by exaggerating how much work needs to be done.

A computing cluster is a group of discrete computers, or nodes, that a master computer assigns tasks to. Clusters are particularly useful in doing tasks who's steps aren't required to be executed in sequential order. Given that CA are generally required to be executed in sequential steps, calculating them on a cluster presents a challenge. If one particular computer's simulation happens to be particularly light, it has to wait for all the other computers to finish before continuing. The goal of this research is to find a method of distributing the computation of the CA in a way that all of the power of a computer cluster can be used. This will be done by splitting up the entire simulation into smaller areas that each computer is responsible for.

Spatial partitioning or subdivision is the act of splitting up an area into smaller chunks based on some rule, a simple way to partition a 2D area is to split it into equally sized rectangles that span the area vertically. In our simulation, the entire 2D grid is split into rectangular chunks that span the area vertically and who's width depends on how much computation or work is in their area. The effect that this will have is that the computers will eventually tend towards taking a roughly equal amount of time to calculate the area that they are responsible for.

In theory a cluster should be able to simulate the CA faster than a computer on its own, and the research has been done based on that assumption. One particular paper[5] covers CA parallelization pretty thoroughly including load balancing, however it does not use dynamic load balancing and only determines what the size of the regions is once at the start of the simulation. In contrast this algorithm continuously updates the size of regions without incurring any additional network overhead, although it does use a much less efficient networking protocol.

2 METHODS

2.1 Equipment Specifications

A computer cluster was built with 15 Dell Optiplex 3020s and one Optiplex 3010. The 3010 was used as the master and the 3020's were used as the nodes. All the computers will be running Ubuntu Server 16.04 with a JVM installed. All the child nodes are mostly homogeneous with 4GB of memory, Core I5 processor, and greater than 100gb of memory. The computers are all wired together over Ethernet and connected to a one gigabit switch. After java code is compiled into a jar, it was uploaded to the master and placed on a networked drive that all nodes have access to. The master's job will be to delegate tasks to the nodes, and the nodes will execute those tasks in parallel.

2.2 Computer Roles

The role of the master node in the simulation will be to handle all high level decisions about the simulation. The master node sets up the initial state of the CA and distributes chunks of it evenly to all the nodes, after that it does not keep a copy of the current simulation or keep track of the current state of the CA. The master simply keeps track of how complete the simulation is and reports to the user important events such as initialization, percent done, and completion. In this way, the master node does not need to know the rules about the CA or really any information about it at all, just how each child node is operating. An advantage of this is that after the simulation starts, the master node contributes almost nothing to the network overhead.

The role of the child nodes is to simulate its area of the CA. The node computers will contain the rules of the CA as well as its current chunk and information about its neighbors. It will sequentially simulate steps of the CA for a certain number of steps before taking a snapshot of the current board as output. Because each cell of a CA needs the state of the cells around it, the cells at the borders of chunks will need information from neighboring chunks. The node computers will directly communicate with each other without using the master, drastically reducing the master's workload. In this setup, the nodes balance themselves without direction from the master.

2.3 Nature of the Cellular Automata

An CA was created as an example for this research. The rules of the CA produce a simulation that can operate on a large scale, the cells only require the state of cells imminently surrounding it (the 8 adjacent cells), the amount of work across the board is not homogeneous, and that each cell holds only a small amount of data (A byte). The CA roughly mimics bacteria growth and the cells will be referred to as bacteria.

A problem with the bacteria growth that needed addressing was that it spread too uniformly and quickly. To address this a persistent background variable known as 'food' was created that varied across the board in random patterns. To achieve this inconsistent food source, the amount of food at any given location is a function of its position in simplex noise, passed through a filter. Simplex noise is a fast and easy way to generate a gradient noise filled plane, this means that while the pattern simplex noise generates is random, the values are continuous across a line. An example of this can be seen in figure 1, the amount of food at a location is given by the color, black is no food and white is max food. In the uniform version, the amount of food across the board is the same and never changes. In the simplex version, the amount of food is concentrated in some locations and sparse in others. Raw simplex noise was not enough to slow down the growth of the bacteria so the noise was passed through a filtering function to sharpen the edges and create more defined borders as seen in the last image of figure 1. To put it mathematically, the amount of food at any given location is:

$$food = \frac{1 + (\frac{2}{3}\arctan(2\pi * \text{simplex}(x, y) + 1))}{2}$$

With the information about food, the basic function of the rule set is that a bacteria cell will try to eat as much food as it can to increase its health and then split once it is healthy enough. The amount of food a cell can eat is based on the simplex noise as well

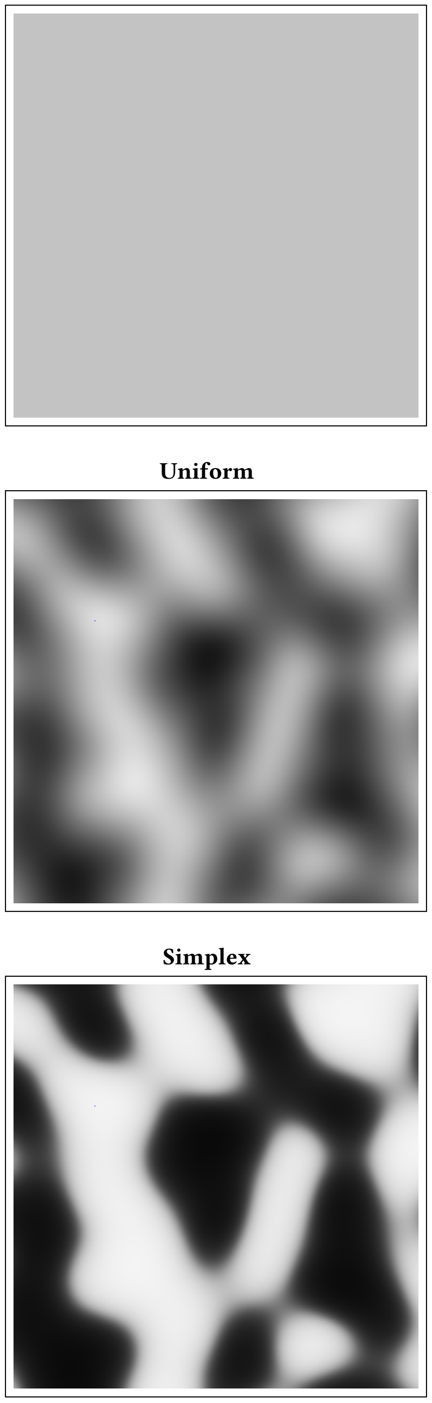


Figure 1. Various food densities.

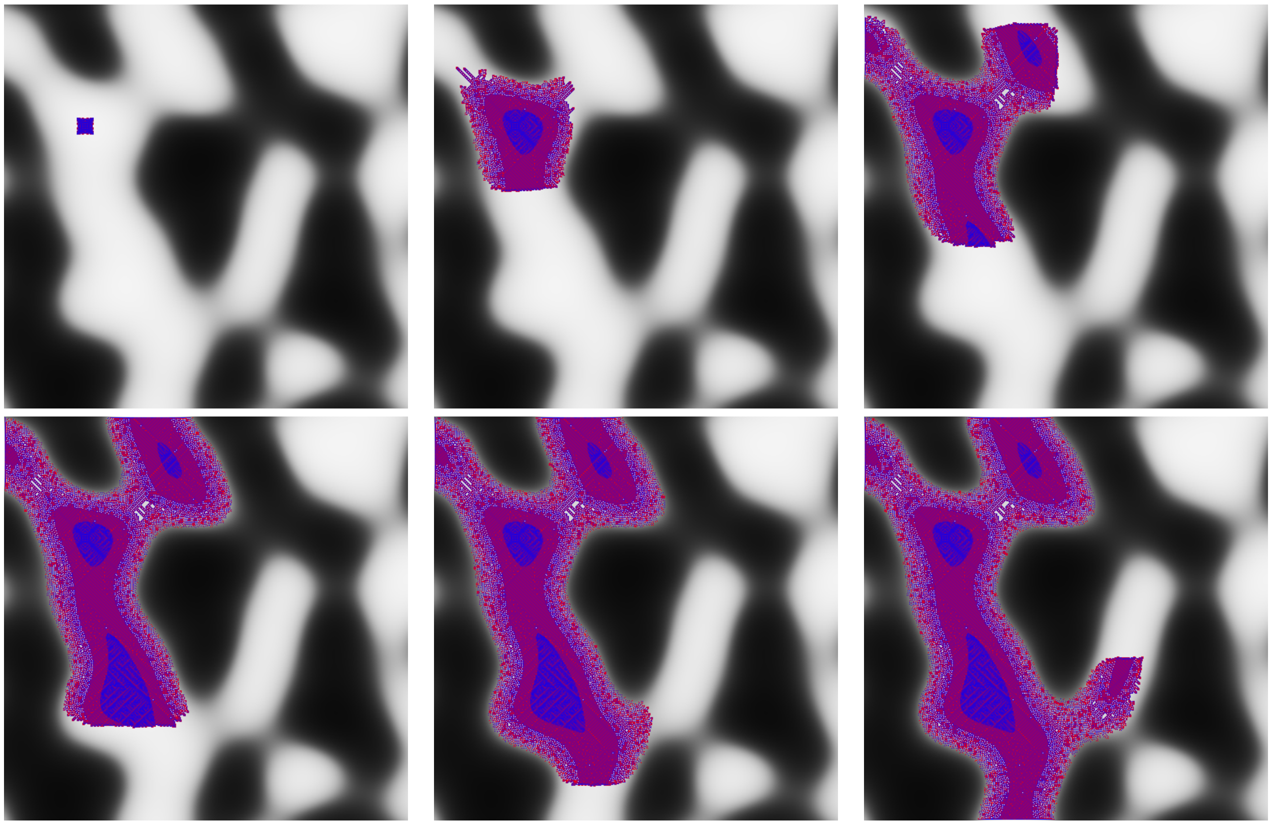


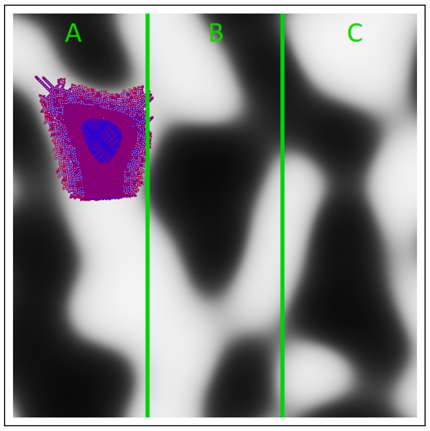
Figure 2. Step 250, 2250, 4250, 6500, 8750, and 11250 of a simulation.

as how many other cells are around it. The progression through time of an example simulation can be seen in figure 2 where the initial bacteria was placed in the top left corner and then left to spread. A colored pixel represents one bacteria cell, its color from red to blue represents how healthy it is from minimum to maximum. A more formal version of those rules is as follows:

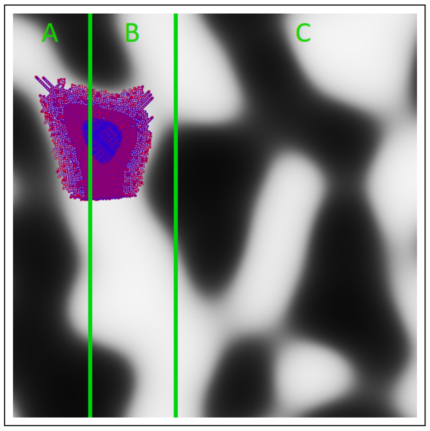
- If the current cell has bacteria:
  - Do some useless computational work to slow down simulation
  - Eat as much food as is available in the current location, given by the filtered food function in figure 1
  - If this cell is above 80% health, check each adjacent cell for a space to expand into and reduce 10% health for each spot it can expand to
  - Reduce health a fixed amount to simulate hunger
  - Reduce health a fixed amount for each neighbor to simulate crowding
  - If health is less than 1, die
- If the current cell has no bacteria:
  - Check each adjacent cell for a bacteria that can split and gain an amount of health for each bacteria that can split into this cell, thus making a new cell

2.4 Nature of the Subdivision

Two subdivision strategies were used, simple and adjusting. When the simulation is starting both the simple and adjusting algorithms divide the space into equal sized rectangles that span the height of the whole board. This means that when both simulations start, the two algorithms will initially look like simple partitioning in figure 3. The green lines represent the border between two nodes, so for example in figure 3 the simple version has three nodes, A, B, and C that are the same width. However, in the adjusting version the width of each node is changed dynamically over time, in figure 3 node A and B both grew smaller to concentrate computation on the bacteria and C grew larger to take over what the other nodes left behind. This approach greatly simplifies the algorithm by reducing the problem to one dimension and reduces the amount of data that needs to be swapped between nodes. One problem with this method is that if the work load is spread horizontally across the top of the simulation area uniformly, the master will not be able to allocate more nodes to the area of high work because it only has the ability to adjust width and not height. The algorithm to partition the space is simple in it's operation, as an example two nodes are next to each other, node 1 and node 2. Each pair of nodes keep an open TCP connection and upon completion of a step, send out a packet to their left neighbor containing a 64 bit long of how much time the node took to simulate in nanoseconds. So in the example node 2 sends the time packet to node 1 and node 1 then



Simple



Adjusting

Figure 3. Different partitioning types, images created as an example.

compares how long node 2 took to finish to itself. If node 1 finished first that means that node 1 was wasting time waiting for node 2 to finish. In this case node 1 sends node 2 the rightmost column of the simulation, shrinking node 1 and growing node 2 by one pixel in width. If node 1 finished after node 2 then the opposite is true and node 1 requests a column from node 2. This exchange happens every step of the simulation which means that the nodes are always being adjusted. If any node takes a long time to finish it gets smaller and if it finishes quickly then it gets bigger. In this way the entire simulation slowly adjusts to have a homogeneous work load.

Two CA simulations were be used to evaluate the algorithm, one that adjusts dynamically and one that does not adjust. These two methods can be compared to determine which makes best use of the hardware. To compare the two different types of partitioning, time spent doing the calculation and time spent waiting for network exchanges were recorded. With that information it is possible to evaluate if the complex partitioning offers any advantages over typical partitioning. The actual effect of the load balancing can be seen in figure 4, the green bars represent the borders between nodes and the area between the bars is what each node is simulating. As the simulation is progressing, nodes on the left start to shrink while nodes on the right start to grow due to the increased work load of the bacteria.

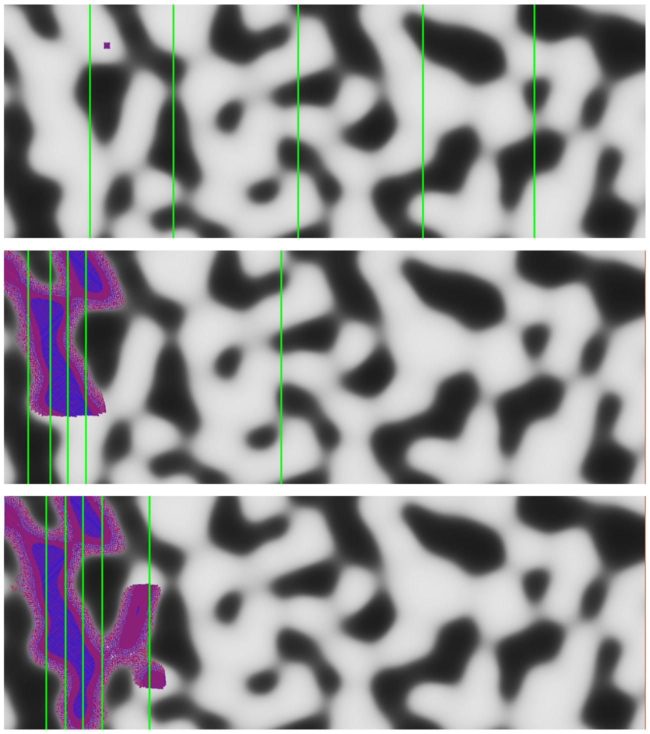


Figure 4. Boundaries of the nodes adjusting themselves as the simulation progresses. Images were generated from actual simulation.

2.5 Tests

The tests were conducted using 6 nodes. Four different sized CA were simulated with and without the balancing algorithm for a total of 8 simulations. The total height of the simulation remained constant at 1000 cells tall and the sizes describe how wide each node is at the start of the simulation, so a size 260 simulation is 1560 cells wide when put together. The four tested sizes were 260, 500, 1000, and 1500 because below a width of 250 it would be advantageous to run the simulation on a single computer and because above 1500 takes too long to reasonably measure.

The simulation time and the idle time were recorded for 50 step periods and printed for data collection. Additionally an image is rendered at these 50 step periods for data visualization purposes. At the end of a simulation, the images were downloaded from the cluster, stitched together, and then rendered into a video to help with debugging and ensure the system works.

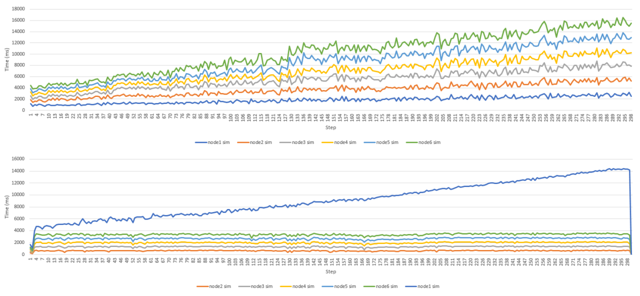


Figure 5. Stacked graph of time spent doing the CA simulation per 50 steps for each node. Top image is the load balanced version and the bottom is the non-load balanced version.

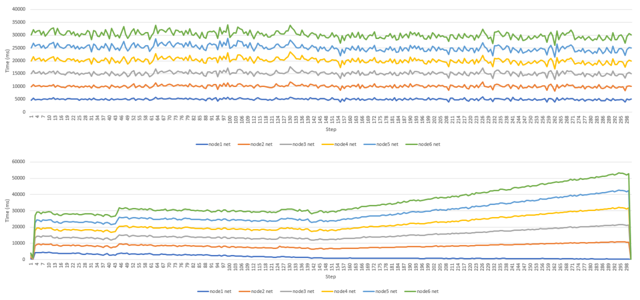


Figure 6. Stacked graph of time spent waiting for network transactions per 50 steps for each node. Top image is the load balanced version and the bottom is the non-load balanced version.

3 RESULTS AND ANALYSIS

Across all 4 sized simulations, the results were all the same, the amount of total time spent doing the simulation was the same between the load balanced and non load balanced, however the time spent waiting for network transactions was significantly less for the load balanced version.

The difference between the load balancing and non load balancing algorithms in reference to simulation time can be seen in figure 5. The graphs are stacked which means the topmost line also represents the total time spent in that frame between each node. Both graphs have the same profile which means that both simulations took nearly the exact same total time to simulate a frame. The difference though is that in the load balanced version all of the nodes simulated for a roughly equal amount of time where in the non load balancing version almost all of the simulation time is concentrated in node 1.

The advantage of load balancing can be seen in figure 6. In the load balancing simulation the time spent waiting for network remains constant throughout the entire simulation. In contrast the network idle time increases over the entire simulation.

Figure 5 and figure 6 are both size 500, but the other 3 sizes have similar results. This is shown in figure 7 which shows how much

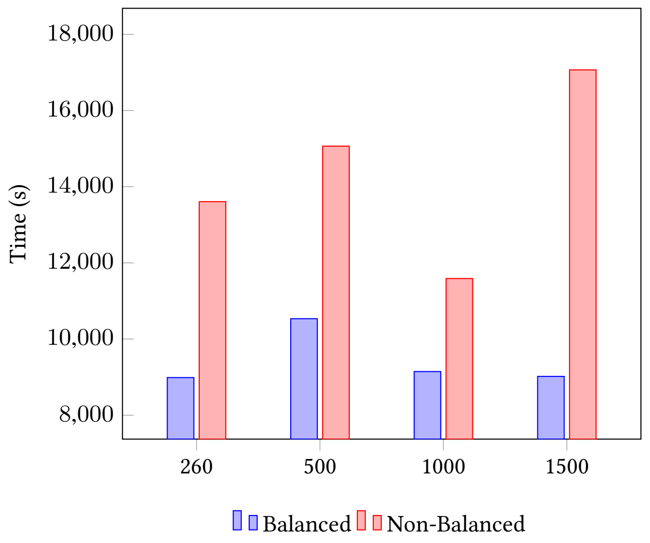


Figure 7. Total time spent idle for each of the simulations.

time was spent waiting for network transactions for each of the 8 simulations.

4 CONCLUSION

Cellular automata are tools that researchers use to simulate various physical phenomena such as bacteria or population growth. Due to their computational work load, it is advantageous to distribute the simulation across a cluster. It is important that the cluster does not waste time on tasks other than the simulation of the CA such as waiting for other nodes to finish, so a method to balance the work equally is important. A CA that is load balanced by nodes trading columns with their neighbors based on calculation time should reduce total time spent waiting for those neighbors to finish. When compared to a non load balancing version, the load balancing CA does in fact reduce idle time.

REFERENCES

[1] René Reitsma, Stanislav Trubin, Eric Mortensen *Weight-proportional Space Partitioning Using Adaptive Voronoi Diagrams.* Published online: 1 March 2007

[2] Robert Lubas, Jaroslaw Was, Jakub Porzycki. *Cellular Automata as the basis of effective and realistic agent-based models of crowd behavior.* Journal of Supercomputing. Jun2016, Vol. 72 Issue 6, p2170-2196. 27p.

[3] Tammy Dreznerl, Zvi Drezner. *Voronoi diagrams with overlapping regions.* Published online: 25 April 2012

[4] Xinli Ke, Lingyun Qi, and Chen Zeng. *A partitioned and asynchronous cellular automata model for urban growth simulation.* International Journal of Geographical Information Science. Apr2016, Vol. 30 Issue 4, p637-659. 23p.

[5] Xia Li, Xiaohu Zhang, Anthony Yeh, and Xiaoping Liu. *Parallel cellular automata for large-scale urban simulation using load-balancing techniques.* International Journal of Geographical Information Science Vol. 24, No. 6, June 2010, 803–820