Project Proposal: Parallel Cellular Automata

Zacree Carroll, Troy Clendenen, Raul Patel

October 20, 2021

1 Introduction

1.1 What is Cellular Automata

A cellular automaton is a single system with a collection of cells. The state of a cell during any particular discrete time step is governed by the combination of its previous state and the previous states of its neighbors. Complex behavior can arise from these systems (often governed by a simple set of rules). Figure 1 is an example of what is known as a "glider" in John Conway's Game of Life. Gliders are one of the more famous patterns in Game of Life. They emerge after a certain time step and they will move diagonally across the screen oscillating through the different forms shown in the example. This is but one of many entities which arise from the simple set of rules of which Game of Life is composed.

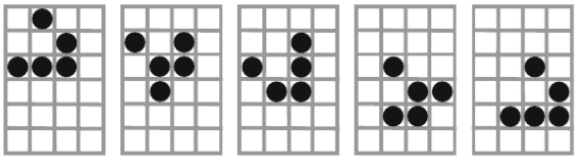

Figure 1: Five time steps of a glider.

1.2 Interesting historical facts

In the past computation of any particular cellular automata was considered impractical due to the modality of computing at that time and the nature of the differential equations which were used to describe them.[CDGR+95] Recent breakthroughs in the field of parallel computation have provided a more practical framework within which cellular automata can be modelled. Cellular automata were first designed by John Von Neumann in the 1950s as a way to study self-replicating systems.[Neu66] Since their conception cellular automata have primarily been used to study parallel computing methodology but have also been used to simulate a wide variety of phenomena.[CDGR+95]

2 Different Problems in Cellular Automata

2.1 Simulation of Wild Fires

Cellular automata are frequently used in computational simulation. One such area that is quite popular is the simulation of wild fires. Many of the approaches to this problem vary in their considerations for their particular model but the majority of them share the goal of predicting wild fire behavior given a set of input parameters. Some of those input parameters may include cell size, moisture levels, weather patterns, and the heterogeneity of the area. Different collections of these parameters are referred to as fire models and one such fire model is the EMBYR model.

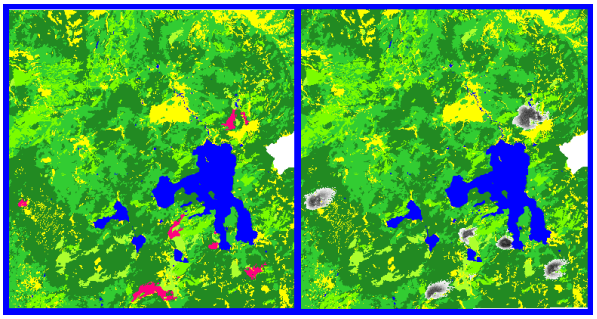

Figure 2: Example image of an EMBYR simulation.

The EMBYR model has a cell size of 50m. The considerations that this model makes are diffusive spread from cell to cell, the effects of windspeed and direction, distant cell ignition due to firebrands (an object which is airborne and carried some distance in an airstream), and differing combustibility throughout the landscape due to variance in moisture.[HGT+00]

2.2 Simulation of Bioremediation

Another use of the cellular automata approach can be found in the simulation of bioremedation. Bioremediation involves contaminated soils and the use of organisms such as bacteria as a means of site clean-up. Previously before massively parallel computing was available this problem was only capable of being modelled by differential calculus and the use of such a method had to negate some of the nuance of the interactions taking place. By using the cellular automata approach such negations become unnecessary.[GRS+96]

3 Possible Directions for the Project

3.1 Implementing a Parallel CA Simulation

Parallelizing cellular automata is not a new idea. On the other hand, it's almost inherent given the amount of local interactions cells may have with each other, and essential in order to implement large-scale simulations. There have been many projects that have designed and implemented parallel models of CA, such as CAMEL, StarLogo,

NEMO, and CAPE [Tal07]. While our group does not expect to create a system as sophisticated as any of these, our main direction is to simply design and implement one such parallel cellular automata application using OpenMP. Ideally, we'd like to parallelize a forest fire simulation using CA. However, depending on our ability to deal with a complex idea for a simulation, we may deliver a project with a simpler set of rules (such as Conway's Game of Life [Ada10]).

3.2 Optimizing Simulations for Shared-Memory Systems

Utilizing the benefits of specific architectures is important in parallel computing, along with HPC as a whole. Data locality is one of the most important concepts in optimizing high-performance codes in computer science. One such concept that exists to help take advantage of locality is Partitioned Global Address Space (PGAS). While we don't think anyone in our group can implement an incredibly sophisticated system, it might be nice to try and implement a way for our specific application to take advantage of manycore CPU clusters using MPI. MPI has actually been shown to be a decent alternative to a native implementation for PGAS functionality in situations when one may not be readily available [DVP+14]. While there may be better approaches to this, given the scope of the class and the concepts we will be learning, it might be a good idea to work on this if we have time in order to gain more experience with MPI.

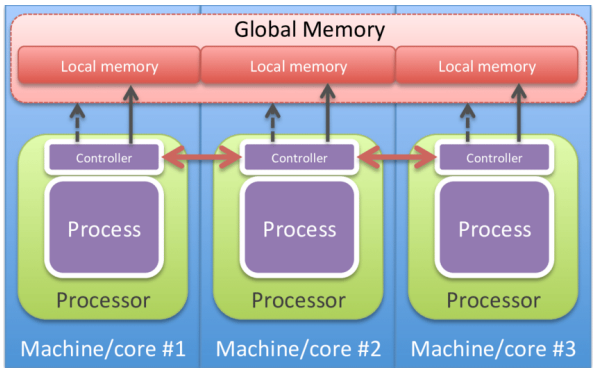

Figure 3: Basic outline of a PGAS runtime.

3.3 Analyzing Parallelizable Code Segments

While it may not contribute to explicit parallel programming experience, writing static analysis tools to assist our group to develop and improve our project is something that might be helpful to work on. It would also help us understand how compilers analyze and work with parallel programs. Most analysis would probably involve loops and memory access patterns, and we could use things like the polyhedral model to potentially transform and optimize our loop structure. While it's unlikely our group goes down this path, it's something that's interesting to think about going in the future.

4 Expected Results

4.1 Base Project Deliverables

As stated earlier, we are intending to start with our own implementation of a parallel CA simulation. By doing so, we hope we can better determine how to expand upon the basic structure of CA. Because of this, we know at the very least we want to build our own Conway's Game of Life simulation using C++ and OpenMP. Because many cell-to-cell interactions are inherently parallel within CA simulations (and there is no real reason to write serial CA applications), we want to evaluate our project with weak scaling in mind. There's merit in comparing serial vs parallel execution time, however our group's focus will be on the scalability of our model. We plan on increasing the number of cells for each simulation to scale the problem size and evaluate our program's performance for each size on a single node. Hopefully our project can scale relatively well given time constraints of the term, and if we have time, there are a few more things we'd like to be able to include in our final submission.

4.2 Expanding the Project if Time Allows

4.2.1 Streamlined API For Generalizing Parallel CA Simulations

One thing that would be really great to work on is a more generalized API that can be used in designing parallel CA simulations. While the Game of Life is cool, creating an API to help other scientists design their own simulations is something that really interests us. We'd keep our focus on streamlined parallelization of these applications, and attempt to keep the same level of efficiency that we hope to achieve with our base project. This could also support extensions of CA simulations to non-academic areas, such as assigning musical notes to different stages of the cells to use as a software instrument. There have been other projects made for the purpose of general scientific simulations using CA, however our idea is that this would give us experience with parallel computing in context of general software engineering practices.

4.2.2 Further Optimizations

We'd like to utilize the machines to the best of our abilities in this project, and if all goes well with our initial build, then we'd like to also design a PGAS runtime using MPI to improve our application's scalablity by utilizing multiple CPU nodes efficiently. If our group manages to design a working PGAS runtime for this project, then we would make sure to compare program performance with multiple CPU nodes to performance without utilizing multiple CPU nodes. We could even test strong vs weak scaling in the context of our project and see how well it line up with Ahmdal's law and Gustafson's law. Hopefully this could provide us with further insight in the nature of CA problems, however this may be unlikely within the scope of this class.

References

[Ada10] Andrew Adamatzky. Game of life cellular automata. 2010.

[CDGR+95] Mario Cannataro, Salvatore Di Gregorio, Rocco Rongo, William Spataro, Giandomenico Spezzano, and Domenico Talia. A parallel cellular automata environment on multicomputers for computational science. *Parallel Computing*, 21:803–823, 01 1995.

[DVP+14] Jeff A. Daily, Abhinav Vishnu, Bruce J. Palmer, Hubertus Van Dam, and Darren J. Kerbyson. On the suitability of mpi as a pgas runtime. *2014 21st International Conference on High Performance Computing (HiPC)*, pages 1–10, 2014.

[GRS+96] S. Di Gregorio, R. Rongo, W. Spataro, G. Spezzano, and D. Talia. A parallel cellular simulator for bioremediation of contaminated soils. *Transactions on Ecology and the Environment*, 10(2):685–694, 1996.

[HGT+00] W.W Hargrove, R.H Gardner, M.G Turner, W.H Romme, and D.G Despain. Simulating fire patterns in heterogeneous landscapes. *Ecological Modelling*, 135(2):243–263, 2000.

[Neu66] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.

[Tal07] Domenico Talia. Cellular automata + parallel computing = computational simulation. 2007.