Parallel Discrete-Event Simulation on COVID-19

Ethan Tuttle, Samuel Mei, Tommy Truong, Henry Qi
{tuttle2, meis, truonp2, qih3}@rpi.edu
Rensselaer Polytechnic Institute

Abstract

The COVID-19 Pandemic has highlighted the need for accurate and efficient models to simulate the spread of infectious diseases. Discrete-event Simulation (DES) is important in this context as it allows for modeling the spread of infectious disease like COVID-19. In this simulation, we will focus on the ratio of the population that is healthy, infected or dead. In addition to focusing on infection rates, we will also focus on system performance to ensure that these simulations can be run efficiently on a high-performance supercomputer such as RPI AiMOS. Moreover, we analyze the strong scalability of this model which makes this product suitable for generating simulations at scale in respect to simple serial simulation models.

Contents

1 Introduction

Parallel Discrete Event Simulation (PDES) is the execution of a single discrete event simulation program on a parallel computer. PDES has emerged in the past few years as as the strongest contender of simulating events at scale.[3] One of it's most important uses is the simulation of widespread diseases which have been significant given the pandemics that occurred in the recent years such as Ebola and Influenza. Due to the stochastic nature of pandemics, it is near impossible to predict the behavior and outcome of a pandemic but a simulation could provide the closest answer possible that scientists could need.

By modelling the spread and infection of a pandemic, scientists can take preemptive measures on containing or even curing the disease before the damage becomes uncontrollable. In particular, during

the rapidly spreading COVID-19 pandemic, scientists were able to control and study the effects of social distancing and masks which led to a more controlled infection levels of COVID-19 until a vaccine was developed for the disease.

Discrete Event Simulation (DES) naturally runs into computational issues when it comes to dealing with data on a scale of worldwide pandemics which is where PDES shines. The parallel versions of agent-based simulations for pandemic outbreaks will allow for significantly reduced replication time and allow for near instantaneous evaluation of a pandemic outcome during possibly prior to its spread.[10]

2 Approach

For the project we will create a parallel discrete event model that will simulate the world going through a pandemic, similar to covid-19. To create this model, simple structs to represent a person will be created. Each will have an integer representing whether they are alive (1), infected (2), or dead (3) and a list of randomized people representing their social circle that they have the potential to infect if they themselves are infected.

Each struct/person in the model will be assigned to an MPI thread based on the number of threads available at runtime. Each MPI thread will store a list of its active infections, and will run through each infected person it is assigned to.

The model will be based on a cyclical day model, where each day a person has a defined chance to infect each person in their social circle. If a person infected is managed by another thread, a message will be sent to that person's thread including who has been infected. Otherwise it is stored in a self buffer. Then a message is printed to I/O saying who has been infected and by whom. After infection, each person has a defined chance to die from the infection. The definitions of these chances should be able to be changed easily, using inputs or defined defaults if no inputs are provided. If a person dies, they are removed from the thread's list of infected people and a message is printed to I/O that they have died.

Once an MPI thread has run through all of its assigned infected people, the day is over. Threads should read self-buffers and MPI buffers and add the corresponding people to their list of infected persons to be used for the following day. This simulation will run until either everyone is infected/dead, or all of the infected people have died off and no more infected people remain.

2.1 Parallel Output

For parallel output, our project has each rank output to a shared file a summary of the status of the nodes in the rank. This includes how many nodes are fine, infected, and dead. They are printed in rank order using the *MPI_File_write_at_all* function

2.2 Definitions and Allowed Inputs

- Chance to start infected
- Chance to infect someone else (happens once for each person in social circle
- Chance to die from infection (happens once each day)
- Number of people in social circle (connectedness of graph).

Seeds will be used to create consistency in the randomness of the simulation for consistent measurements.

2.3 Scalability

This should be able to scale well to different tests. Tests changing each input can be run for different numbers of threads. This will create many different graphs showing the results of parallelization. Like the MPI homework, tests will be run with:

- 2 MPI ranks (1 compute node)
- 4 MPI ranks (1 compute node)
- 8 MPI ranks (1 compute node)
- 16 MPI ranks (1 compute node)
- 32 MPI ranks (1 compute node)
- 64 MPI ranks (2 compute nodes)
- 128 MPI ranks (4 compute nodes)
- 256 MPI ranks (8 compute nodes)

2.4 Code

Code for this project can be found at the github repository at this link: https://github.com/EthanTuttle/Parallel-Programming-Project-S2023

```
MPI_File file;
MPI_File_open(MPI_COMM_WORLD, "output.txt", \
 MPI_MODE_CREATE | MPI_MODE_WRONLY,
        MPI_INFO_NULL, &file);

void simulation()
{
    char buffer[150];
    memset(buffer, '\0', sizeof(buffer));
    int count = strlen(buffer);

    MPI_Offset file_size;
    MPI_File_get_size(file, &file_size);

    MPI_Offset offset = file_size + (count * \
     myrank);
    MPI_File_write_at_all(file, offset, \
     buffer, count, MPI_CHAR, \
     MPI_STATUS_IGNORE);
    MPI_File_sync(file);
  }
}
```

MPI Code: Parallel MPI I/O code to read and write from `MPI_File`

Collective I/O in MPI

Our program requires write access from three different locations. Since each process requires it's own access informations, the arguments list will be the same for all three. We will then be using the snippet of code on the left for all three cases. Let's take a look:

- First, we obtain the size of the buffer needed for I/O depending on each scenario.
- Then, we determine the size of the file.
- Finally, we set the offset to the end of the file and add our buffer there.

3 Computation and Performance

3.1 Strong Scaling Study

We took the data points between the given MPI ranks and ranked them accordingly below. By default, our simulation has an estimated population of 20 million and this chart estimates the time it takes for a given disease to fully spread and infect the entire population of 20 million. The simulation was ran and executed through AiMOS, RPI's Blue Gene/Q Supercomputer through subsequent batch jobs with 4x node priority. Execution time was measured through a clock-cycle counter developed by Christopher Carothers.

Table 1: Execution Time of Simulation through varying MPI Ranks

Ranks	Time (s)
2	58.4 s
4	42.2 s
8	24.1 s
16	13.2 s
32	7.9 s
64	6.5 s
128	4.3 s
256	3.2 s

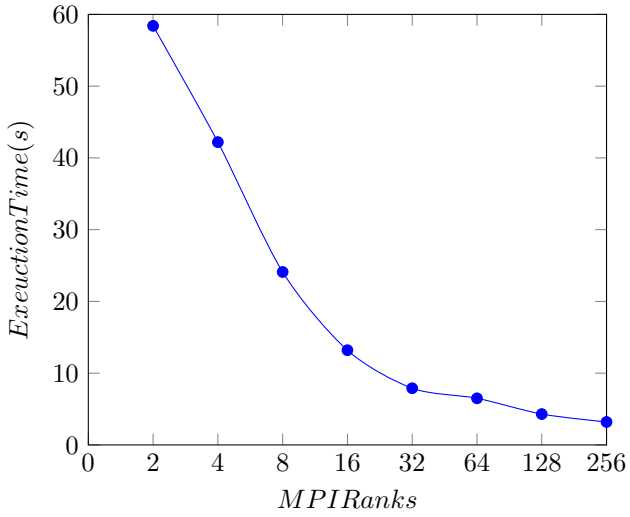

Figure 1: Alternative Execution Time of Simulation through varying MPI Ranks

Figure 1 shows the result of our execution of the simulation using 2, 4, 8, 16, 32, 64, 128 and 256 MPI Ranks with increasing amounts of AiMOS compute nodes. We can draw the idea that as the amount of MPI compute nodes that go into running this simulation, the amount of time that is spent executing and running this simulation drastically decreases.

Table 2: Performance Increase through increasing MPI Ranks

MPI Ranks	Performance Increase (%)
2 to 4	38.4%
4 to 8	57.1%
8 to 16	82.6%
16 to 32	67.1%
32 to 64	21.5%
64 to 128	51.2%
128 to 256	34.4%

From figure 2, we can see that there is a logarithmic regression in relation from execution time to the number of MPI Ranks spent. In particular, the strongest jump in performance boost was at MPI Rank 16, where we had approximately an 80% speedup compared to the previous MPI configuration of 8 MPI Ranks.

3.2 Weak Scaling Study

Table 3: Execution Time of Simulation through with increasing rank and population

Population	Rank	Execution Time (s)
2 Million	2	6.1 s
4 Million	4	8.9 s
8 Million	8	9.8 s
16 Million	16	10.7 s
32 Million	32	13.1 s

Table 4: Percent difference between increasing rank and population

Population	Rank	Performance Change (%)
2 Million	2	0
4 Million	4	-45%
8 Million	8	-10%
16 Million	16	-9%
32 Million	32	-22%

We performed a weak scaling study with increasing population and rank sizes to see how well the simulation scales when both factors are increased. We can conclude that the current simulation scales moderately well considering the execution time is within a close time window between each other. However, the numbers between the each population and rank are still somewhat far from each other giving us room to improve upon in this aspect.

4 Conclusion

Our goal with this project was to create a massively parallel simulation of the pandemic spread partially inspired by the passing event of COVID-19. Through our findings, we have found out that this simulation is incredibly suited for parallel systems as execution time increases drastically by a large margin when supplied with more MPI Ranks and Compute Nodes. Our strong scaling study showed us that there was a massive increase in performance when it comes to running a simulation in parallel. However, our weak scaling study also showed us that there is still improvement in the area when it comes to scaling the algorithm up to numbers as big as billions.

4.1 Future Work

An improvement that we could make to our algorithm is the implementation of CUDA instead of MPI. Currently, we only use MPI to work in parallel but by implementing CUDA and having CUDA calls that handles each rank loop, our algorithm could be paralyzed even more. Alternatively, we could also implement both MPI and CUDA, which when implemented together properly could be parallelized even more compared to using CUDA or MPI alone.

4.2 Future Direction

This project, when improved upon, could be used as one of the multiple simulations available for predictions of pandemic behavior. Being able to quickly and accurately predict possible spread and movement of a pandemic could provide valuable information that could help scientists and doctors prevent the spread of pandemics and possibly contain the endemic before it becomes a pandemic [7].

5 Contributions

5.1 Ethan Tuttle

- Set up serial scenario
- Set up message passing between MPI ranks

5.2 Samuel Mei

- Gathered, analyzed and summarized data
- Conducted the weak and strong scaling studies.

5.3 Tommy Truong

- Set up Parallel I/O
- Set up day summary to pass data along ranks.

5.4 Henry Qi

- Set up Parallel I/O

---

6 References

References

[1] Parantapa Bhattacharya, Jiangzhuo Chen, Stefan Hoops, Dustin Machi, Bryan Lewis, Srinivasan Venkatramanan, Mandy L. Wilson, Brian Klahn, Aniruddha Adiga, Benjamin Hurt, Joseph Outten, Abhijin Adiga, Andrew Warren, Young Yun Baek, Przemyslaw Porebski, Achla Marathe, Dawen Xie, Samarth Swarup, Anil Vullikanti, Henning Mortveit, Stephen Eubank, Christopher L. Barrett, and Madhav Marathe. Data-driven scalable pipeline using national agent-based models for real-time pandemic response and decision support. *The International Journal of High Performance Computing Applications*, 37(1):4–27, 2023.

[2] Jing Fu, Misun Min, Robert Latham, and Christopher D. Carothers. Parallel i/o performance for application-level checkpointing on the blue gene/p system. In *2011 IEEE International Conference on Cluster Computing*, pages 465–473, 2011.

[3] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[4] Bonan Hou, Yiping Yao, Bing Wang, and Dongsheng Liao. Modeling and simulation of large-scale social networks using parallel discrete event simulation. *SIMULATION*, 89(10):1173–1183, 2013.

[5] Michael Kaplan, Charles Kneifel, Victor Orlikowski, James Dorff, Mike Newton, Andy Howard, Don Shinn, Muath Bishawi, Simbarashe Chidyagwai, Peter Balogh, and Amanda Randles. Cloud computing for covid-19: Lessons learned from massively parallel models of ventilator splitting. *Computing in Science Engineering*, 22(6):37–47, 2020.

[6] Gonzalo Martín, David E. Singh, Maria-Cristina Marinescu, and Jesús Carretero. Towards efficient large scale epidemiological simulations in epigraph. *Parallel Computing*, 42:88–102, 2015. Parallelism in Bioinformatics.

[7] Miguel Guzman Merino, Maria Cristina Marinescu, and David E. Singh. Evaluating the spread of omicron covid-19 variant in spain. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 999–1006, 2022.

[8] Kalyan S. Perumalla and Sudip K. Seal. Reversible parallel discrete-event execution of large-scale epidemic outbreak models. In *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, pages 1–8, 2010.

[9] Che-Rung; Rego Vernon; Sang, Janche; Lee and Chung-Ta King. "experiences with implementing parallel discrete-event simulation on gpu". *Electrical Engineering Computer Science Faculty Publications.*, page 455, 2019.

[10] Milton Soto-Ferrari, Peter Holvenstot, Diana Prieto, Elise de Doncker, and John Kapenga. Parallel programming approaches for an agent-based simulation of concurrent pandemic and seasonal influenza outbreaks. *Procedia Computer Science*, 18:2187–2192, 2013. 2013 International Conference on Computational Science.