MASS CUDA Tuberculosis

Holt Ogden

Spring 2025 Term Report

University of Washington

Jun 13, 2025

Project Professor

Munehiro Fukuda

1. Introduction

1.1 Overview

The Tuberculosis simulation (TB) is a benchmark program for the Multi-Agent Spatial Simulation (MASS) library under development at the University of Washington, Bothell. This program simulates an infection of *mycobacterium tuberculosis* in cells of the human lungs, and the efforts of the body to fight the infection using T-Cells and Macrophages. An example of the output of this program is shown in Figure 1. It uses agent-based modeling (ABM) to model the behavior of these cells.

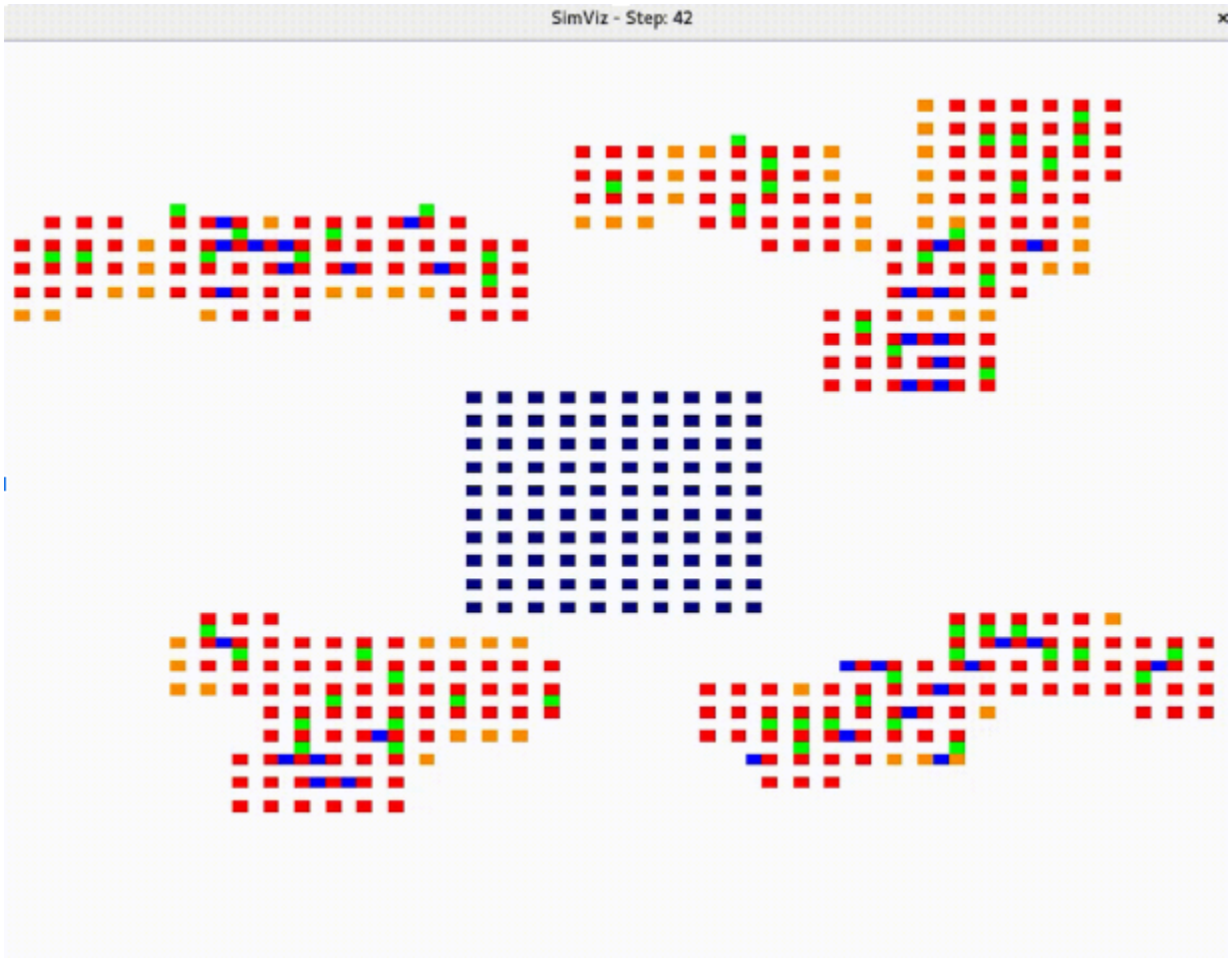

Figure 1. Simviz output showing spread of Tuberculosis bacteria and spawning of macrophages and T-Cells to fight the bacteria.

The MASS library is designed to allow for the parallelization of multi-entity interactions over multiple computing nodes and multiple threads within those nodes. It is based on two key components: places

and agents. A *Place* object is a location in memory that holds data or attributes and can also host agent instances. Places can exchange data between each other or allow residing agents to access its data. Agents are execution instances that are located within a place. They can contain their own data and also instructions that dictate their behavior. Agents can "jump" between places, effectively transferring an execution instance from one location to another. This allows data to move between places while also allowing agents to access data within the new Place.

The MASS CUDA library modifies the MASS library to take advantage of the performance capabilities of GPUs. In this version of MASS, places, agents, and their associated data are loaded into a GPU's device memory and are executed in parallel, taking advantage of the thousands of cores within modern GPUs. The library handles copying of data and execution of functions between the system memory controlled by the CPU (called the host) and the GPU memory used by the GPU (called the device). This version of TB is designed to run on the MASS CUDA library and therefore take advantage of its computing performance. Specifically, this project upgrades the TB program from running on MASS CUDA version 0.5.3 to version 0.7.1, the most current stable version.

1.2 Background

TB is used as a benchmark for comparing runtime performance between the various versions of MASS; MASS Java, MASS C++, and MASS CUDA. The MASS CUDA TB benchmark program was originally written by Christopher Sumali in Winter quarter of 2023 [3]. His program was written for MASS CUDA version 0.5.3, which was released in 2022 by Brian Luger, based on previous work by Lisa Kosiachenko and Nathaniel Hart [1].

However, the MASS CUDA library was significantly updated after this TB benchmark was written. Warren Liu updated the library to version 0.7.1 in Spring quarter of 2024, which rendered the library incompatible with previous benchmark programs. The main changes to the library made in this update are described below, and can be found in more detail in Warren's White Paper [2].

The primary change for version 0.7.1 was the complete rewrite of how place and agent data members are created, stored, and accessed by both the host and the device. Previously, data was stored for places and agents as two separate objects: *Place* & *PlaceState*, and *Agent* & *AgentState*. *Place* and *Agent* hold all the functionality of MASS objects; accessor functions for data, functions access or spawn agents, and the *callAll()* function that can call user-defined functions. A *Place* or *Agent* object then contained a

pointer to corresponding *PlaceState* or *AgentState* objects, respectively, which stored any data members used by those objects. These could include data such as their index, place neighbors, and agent travel paths. A user implementing a MASS program would extend the *Place*, *PlaceState*, *Agent*, and *AgentState* objects by adding their own functions to their *UserPlace* & *UserAgent* objects and their own data members to their *UserPlaceState* and *UserAgentState* objects. Once defined, arrays of *UserPlace*, *UserPlaceState*, *UserAgent*, and *UserAgentState* objects would then be created on the device and used to run MASS simulations.

However, this implementation led to performance problems due to uncoalesced memory access within the *PlaceState* and *AgentState*. Storing arrays of *PlaceState* and *AgentState* objects on the device meant that accessing the same data member for multiple places or agents, such as their index, required accessing widely separated memory locations. Because a place's *callAll()* function operates on the same data members for all places and agents simultaneously, each object is therefore making suboptimal memory accesses with this implementation.

To fix this problem, Warren's work rewrote the MASS CUDA library to remove the *PlaceState* and *AgentState* objects and changed all data members to be stored as attributes of *Place* and *Agent* objects. This means that each data member is stored in a contiguous array of attributes for all places and agents, with attribute index i corresponding to the data member for place or agent i. Effectively, data was reorganized from being stored from an array of structures to a structure of arrays. This allows data to be stored and accessed more efficiently within CUDA threads. These changes led to significant improvements in MASS CUDA performance, with benchmark programs such as Game of Life taking 85% less time to run than in previous versions of MASS CUDA [2]. However, one consequence of this change is that place and agent data is now created and accessed differently than previous MASS CUDA versions. This paper primarily describes the steps to update the TB simulation to use the attributes system in MASS CUDA version 0.7.1.

1.3 Purpose

The primary purpose of this project is to update the TB program to the newest version of MASS CUDA and compare the performance of the new and old TB versions. This will allow further assessment of the performance improvements gained with the addition of the attribute system of data storage for places and agents. In addition, this benchmark could be compared to TB implementations for FLAME GPU2, a competitor GPU-based ABM library, when that benchmark is completed in the future. Finally, this

benchmark program will be largely compatible with ongoing efforts to implement multi-GPU MASS CUDA and so can be used to test that library when it is completed.

2. Program Architecture

2.1 Overview

The places in this simulation represent cells within the human lung, while agents represent either macrophages or T-Cells. The grid of cells is divided into four quadrants, as shown in Figure 2. At the very center of the place array, the tuberculosis bacteria spawns and then spreads outward, with each location infecting all of its neighbors. At the center of each quadrant, one cell is set as a "blood vessel" cell, where macrophage and T-Cell agents can enter the simulation (spawn). Macrophage and T-Cell agents spawn at the blood vessels at each simulation step and then move to adjacent cells. Only one macrophage and one T-Cell can be present within one cell at a time.

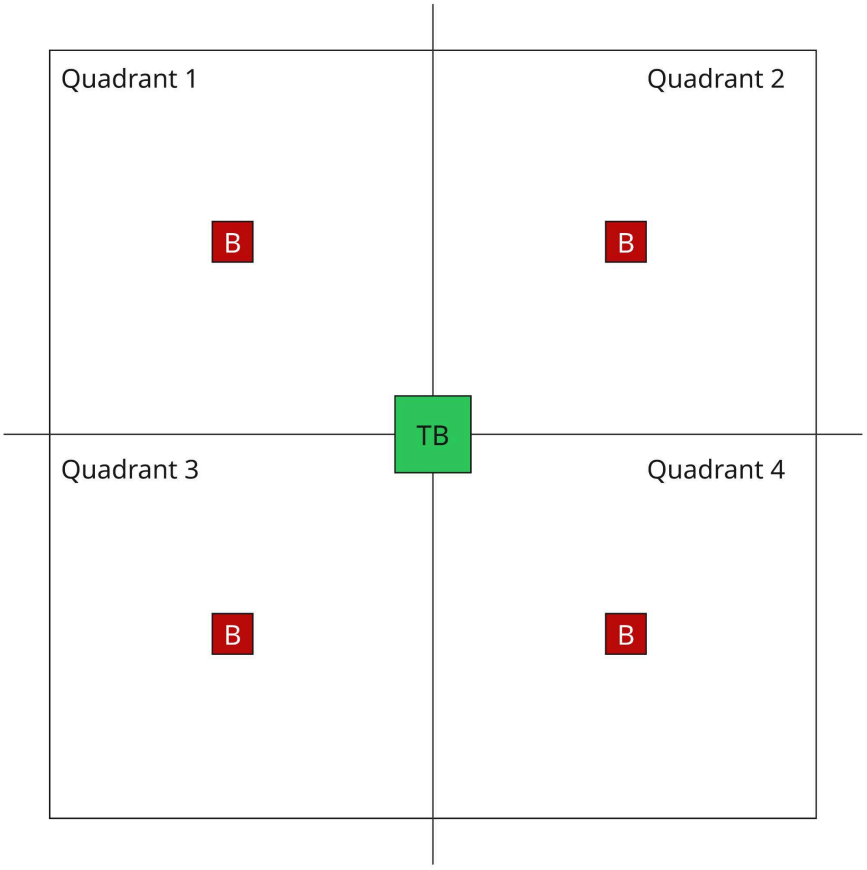

Figure 2: Spawn locations for bacteria and agents. TB bacteria spawns at the center of the cell grid (in green), while blood vessels (in red) serve as spawn points for macrophage and T-Cell agents.

As the bacteria spreads outwards and infects cells, it will encounter macrophage and T-Cell agents that have spawned at the blood vessels. When a macrophage agent enters an infected cell, it increases the chemokine level of the cell to the maximum. The chemokine acts as a beacon, and both macrophage and T-Cells will migrate at each simulation step to the neighboring cell with the highest chemokine level. If a macrophage enters the infected cell, it will contain the bacteria but the bacteria will continue to multiply within the macrophage. Therefore, the macrophage relies on the T-Cells to be drawn by chemokine levels to their cell to kill the bacteria. If bacteria multiplies and exceeds a macrophage's capacity before a T-Cell can reach it, the macrophage bursts and dies, spreading the bacteria.

During the simulation, bacteria continuously spreads between cells, while macrophages and T-Cells continuously spawn, contact, and destroy bacteria from those cells. The simulation ends when all the bacteria have been destroyed.

2.2 EnvironmentPlace

The cells within the lung are modeled as an *EnvironmentPlace* object, which extends the base *Place* class. These objects contain the following data members:

- Size: Integer tracking overall array of places, so the array is (size * size) = total places.
- Max Chemokine: Integer tracking the maximum chemokine levels within a cell, set to 2.
- T-Cell Entrance: Integer representing the first simulation step when T-Cells can start to enter via blood vessels (there is a delay to allow the bacteria to spread).
- Bacteria: Boolean tracking whether the bacteria is present in this cell.
- Blood Vessel: Boolean tracking whether this cell is a blood vessel.
- Chemokine: Integer tracking current chemokine levels.
- Macrophage: Boolean tracking whether a macrophage is present in this cell.
- Should Spawn Macro: Boolean tracking if a macrophage should be spawned at the end of this simulation step.
- Macrophage State: Integer state of the residing macrophage (see macrophage section).
- T-Cell: Boolean tracking whether a T-Cell is present in this cell.
- Should Spawn T-Cell: Boolean tracking if a T-Cell should be spawned at the end of this simulation step.

- Rand State: A random seed that is assigned to this cell.

The key functions of *EnvironmentPlace* are *chemokineDecay()*, *bacteriaGrowth()*, and *cellRecruitment()*.

The *chemokineDecay()* function updates the chemokine levels within the cell. The chemokine levels within every place automatically decay by one each simulation step, unless there is a macrophage on that cell. If there is, the macrophage maintains the chemokine levels at the maximum of two.

If a cell is currently infected with bacteria, the *bacteriaGrowth()* function spreads bacteria to all of the neighboring cells. This infection occurs once every 10 simulation steps. The cells are able to access their neighbors by calling the MASS *exchangeAll()* function.

At the beginning of each simulation step, the blood vessel cells run *cellRecruitment()* to spawn new macrophages and T-Cells. A blood vessel has four spawning options: Spawn nothing, spawn a macrophage only, spawn a T-Cell only, or spawn both. A blood vessel has a 50% chance of spawning a macrophage, and a separate 50% chance of spawning a T-Cell at each step.

2.3 Macrophage

Macrophages are modeled as *Macrophage* agents, which extends the base *Agent* class. These objects contain the following data members:

- Bacteria Capacity: An integer storing the maximum number of bacteria the macrophage can hold before bursting.
- Chronic Infection Limit: An integer storing the threshold of number of bacteria where the macrophage changes its state to “Chronically Infected”.
- Spawn Point Number: An integer storing the number of macrophages that reside on blood vessels and spawn new macrophages. Set to four to match the number of blood vessels.
- State: An integer storing the state of the macrophage. These are defined in an enum, with the options being:
    - Resting
    - Infected
    - Activated
    - Chronically Infected

- Is Spawner: A boolean storing whether this macrophage is one of the spawners that reside on the blood vessels.
- Internal Bacteria: An integer storing the number of bacteria residing within this macrophage.
- Infected Time: An integer storing the number of simulation steps that the bacteria has been infected for.

The key functions of *Macrophage* are *updateState()*, and *burst()*. The *updateState()* function performs a different action based on the four Macrophage states listed above. If a macrophage is resting, it changes to the infected state to search for bacteria. If it is infected, its bacteria grows within it, and it may change to chronically infected if there are enough bacteria. If there is a T-Cell in the same place, it changes to the activated state. In the activated state, the macrophage kills all the bacteria within it. In the chronically infected state, bacteria continue to increase, and if they reach the bacteria capacity the macrophage will burst. The *burst()* function terminates the macrophage agent in this place, and spreads the bacteria to all adjacent cells.

2.4 T-Cell

T-Cells are modeled as *TCell* agents, which extends the base *Agent* class. These objects contain the following data members:

- TCell is Spawner: A boolean storing whether this T-Cell is one of the spawners that reside on the blood vessels.

Because T-Cells only need to be present on a cell to trigger the killing of bacteria, they are simpler than macrophages in their operation. They simply spawn, migrate, and exchange their data with the current cell to inform it that a T-Cell is present.

3. Implementation

3.1 Code Cleanup and Comments

My first step in making changes to the Tuberculosis benchmark program was to do a full code walkthrough to try and understand the code. However, I found that some parts of the code did not have extensive commenting, especially what certain data members were and what they were used for. To

make this process easier for future users of the benchmark program, I worked to clean up the code and add comments to explain the function of every data member and method used in the program. This is also why I explained each data member in detail in this report, so that the next person to update the code hopefully can get a better understanding of the code through this report without having to do a full code review.

3.2 Conversion to Attributes

To convert to attributes, I first transferred every data member from the *EnvironmentState*, *MacrophageState*, and *TCellState* classes to a *setAttribute()* call within the main method, immediately after each *Place* or *Agent* subclass is created. For instance, rather than declaring *int bacteriaCapacity* within the *MacrophageState* private data members, I instead declared *macrophageAgents->setAttribute<int>(MacrophageAttributes::BACTERIA_CAPACITY, 1, 100)*. This initialized all data as attributes in MASS CUDA.

The next alteration I had to make was to change all data access within *EnvironmentPlace*, *Macrophage*, and *TCell* classes to access the memory using the *getAttribute()* function. For instance, this meant changing data access from *bacteriaCapacity = 100* to **getAttribute<int>(MacrophageAttributes::BACTERIA_CAPACITY, 1) = 100*. This had to be done for all data access within all place and agent functions.

Finally, I updated the SimViz output to correctly access the data from the simulation using *downloadAttribute()*. This allows SimViz to access the current results of the data for visualization. Note that this does not run if the interval is set to 0, to allow for faster runtimes for benchmarking. To output the data, I first allocated host memory for all the Simviz data that was needed to create the visualization and then called *downloadAttribute()* for each one. The *EnvironmentPlace* Bacteria, Macrophage, Macrophage State, T-Cell, and Chemokine attributes need to be downloaded for SimViz.

4. Evaluation

I ran the TB simulation for MASS CUDA versions 0.5.3 and 0.7.1 with varying place array sizes. The results are shown in Figure 3.

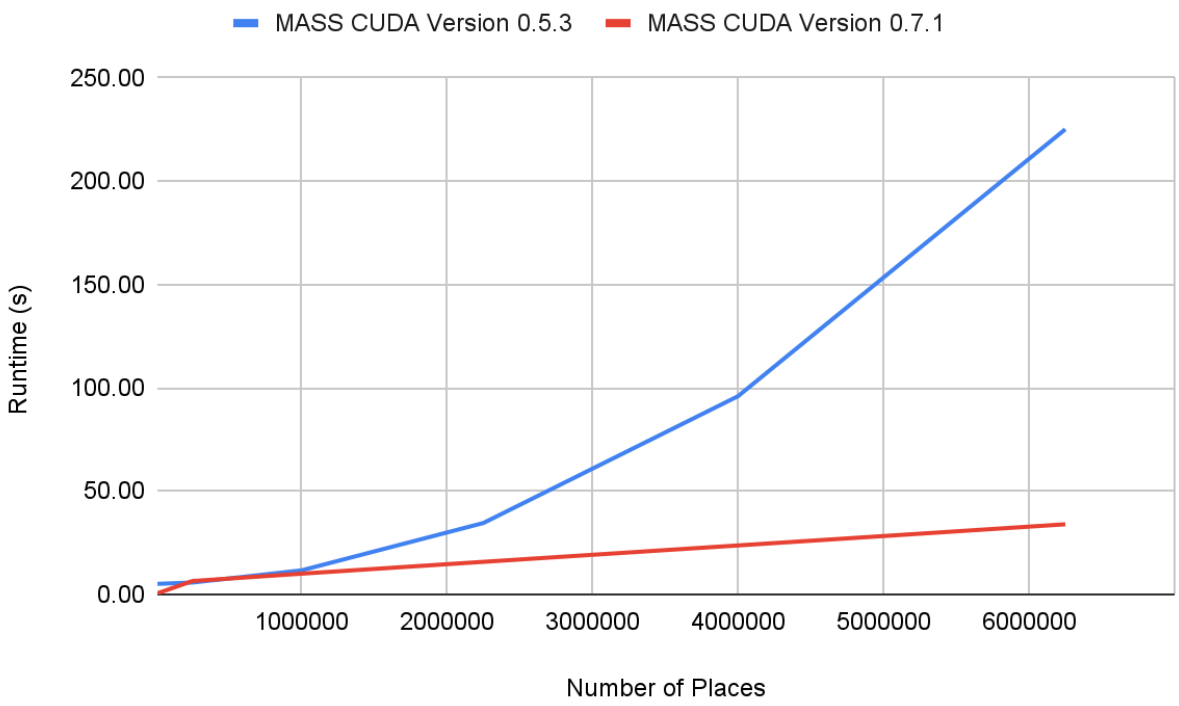

Figure 3: Runtimes of TB for MASS CUDA versions 0.5.3 and 0.7.1.

Figure 3 demonstrates that the updates made by Warren to implement attributes led to a significant improvement in the runtimes of the TB benchmark program. At the largest number of places I was able to run (about 6.25 million) the updated MASS CUDA version 0.7.1 ran 85% faster than the previous version. Based on the trends shown in the graph, this advantage is expected to increase as the simulation size increases. In terms of spatial scalability, as expected both versions capped out at the same maximum, as they store the same amount of data, just organized differently.

5. Future Work

5.1 Update to Multi-GPU MASS CUDA

This is a relatively simple program that can be used to test multiple agent types within a MASS CUDA simulation. Therefore, it would be a good candidate to update with the minor changes required to run the multi-GPU MASS CUDA. These required changes are included in my 2025 White Paper, titled Multi-GPU Parallelized Agent-Based Modeling.

6. How to Run Code

In the Tuberculosis directory of the `mass_cuda_appl` repository, run the following commands:

- `make develop` This sets up the development environment
- `make build` This compiles the code
- `make test` This runs tests on the code to ensure it is working properly
- `./bin/Tuberculosis` This runs the code with the default settings.
- To view the possible settings, run `./bin/Tuberculosis –help`

References

[1] L. Kosiachenko, N. Hart, and M. Fukuda, “MASS CUDA: A general GPU parallelization framework for agent-based models,” in Advances in Practical Applications of Survivable Agents and Multi-Agent Systems: The PAAMS Collection, Y. Demazeau, E. Matson, J. M. Corchado, and F. De la Prieta, Eds. Springer International Publishing. [Online]. Available: https://doi.org/10.1007/978-3-030-24209-1 12

[2] W. Liu, “Programmability and performance enhancement of MASS CUDA.”[Online]. Available: https://depts.washington.edu/dslab/MASS/reports/WarrenLiu whitepaper.pdf

[3] C. Sumali, “Porting Agent-Based Benchmarks to MASS CUDA,” University of Washington, Bothell, Term Report, Mar. 2023. [Online]. Available: https://depts.washington.edu/dslab/MASS/reports/ChristopherSumali_wi23.pdf