Benchmarking MASS C++: An Overview of Program Progress over the Spring Quarter and Current Plan for the Summer Quarter

Table of Contents

Purpose

The purpose of this research project is to compare three multi-agent simulation platforms - MASS C++, RepastHPC, and FLAME by completing the following tasks:

- implementing the following seven benchmarks:
  - *Game of Life*
  - *Brain Grid*
  - *Tuberculosis*
  - *Social Networks*
  - *Bail In/Bail Out*
  - *MatSim*
  - *VDT*

- conducting quantitative and qualitative analyses of each of the three platforms using the process of creating and running the seven ported benchmarks

- co-authoring a journal paper with my advisor

My goal for the spring quarter was to continue creating benchmarks for MASS C++.

Summary of Progress for the Spring Quarter

The code for the benchmark *Brain Grid* was completed this quarter for MASS C++. The functionality of initialization and neuron growth was tested with the older version of MASS C++ (prior to fellow undergraduate student Josh Landron adding neighbor functionality during the Winter 2020 quarter). The functionality of neuronal connections and signal sending will be debugged and tested with the updated MASS C++ with neighbor functionality during a future quarter.

Additionally, the concept for the initialization program for the benchmark MATSIM for all three platforms was reworked; the details of this are outlined in section *Test Programs for the Updated MASS C++ Core*.

Finally, the updates for the new version of MASS C++ were investigated and tested.

MASS C++ Brain Grid

In this benchmark, the behavior of cell growth and signal sending of a self-organizing autonomous neural network is simulated using the derived Place class BrainPlace and the derived Agent class GrowingEnd. This code may be found in the following repository on the SARAH-MASS-BRAIN-GRID branch:

https://bitbucket.org/mass_application_developers/mass_cpp_appl/src/SARAH-MASS-BRAIN-GRID/Benchmarks/MASS/MASS_BrainGrid/

The code for this program has been written, but not tested or debugged with the most updated version of MASS C++, which is necessitated by this program's use of the function addNeighbor().

Initialization

For this simulation's initialization, a space composed of N by N BrainPlaces is created, where N is input by the user. Each BrainPlace has the probability to spawn an excitatory, inhibitory, or neutral neuron with E%, I%, and N%, respectively. For the BrainPlaces that don't spawn a neuron, these spaces remain unoccupied until a neuron requests to grow there later in the simulation.

The BrainPlaces that are initialized as having an occupying neuron are defined as the space that holds the soma, or cell body of the neuron. This particular BrainPlace holds most of the neuron

entity's information: when to spawn which cell part, what other neurons is this cell connected to, what is this neuron's cell type, etc. In order to grow different cell parts during future iterations, a Growing Agent migrates to each soma to act as a spawner.

Neuronal Growth

Next, the following phases occur for the user-input number of iterations : neuronal growth, additions of neuronal connections to each neuron, and signal sending.

During the neuronal growth phase, the soma checks to see if the current iteration is the spawn time for any of its cell parts. When this occurs, the soma spawns a new Growing End agent of the needed cell part type.

The newly spawned Growing End migrates to a BrainPlace adjacent to the soma and marks that Place as occupied. It then continues to grow and branch as specified by its type and associated probabilities. The dendrite and the axon are the only neuron part types that are spawned from the soma; the axon Growing End changes its part type to synaptic terminal with 100 - G% in order to grow that part for its neuron.

If a Growing End stops growing - which occurs at a 10% probability for each iteration - it exits the simulation by calling kill(). A Growing End branches by spawning another agent of the same cell part type and having the newly spawned Growing End migrate to a new BrainPlace that is +/- 45 degrees from the current BrainPlace.

Because a BrainPlace may have at most one neuron part occupying its space during the simulation, Growing Ends must request movement access from the desired BrainPlace by submitting a move request and checking that BrainPlace's outMessage for approval.

Neuronal Connections

Following the growth phase, the Growing Ends check their Moore's area for any neighboring neurons. If a neighboring neuron is discovered, the Growing End hops back to the BrainPlace holding its home neuron's soma and records that connection at that place. If the connection is an out-connection (the current neuron's synaptic terminals neighboring another neuron's dendrites), it is added to that BrainPlace's neighbors.

If any in-connections are found, a newConnectionMsg is sent to the neuron that forms the in-connection with the current cell, and the receiver adds the sender to its Place neighbors vector. An important note is that each neuron adds only its out-connections as a neighbor to forward a signal to, creating a unidirectional edge between the current Neuron and its out-connection.

Signal Sending

After the new neighbor connections are added in the current iteration, each BrainPlace with a neuron's soma takes the signal received from the past iteration (inputSignal) and modifies it based on its neuron type. Additionally, if the neuron at the current BrainPlace is excitatory, it activates an outgoing signal of 1.0 in addition to the signal received in the past iteration with a probability of 10%.

Once modifications to the received signal have been made and passes the threshold amount of .4, the neurons forward the signal to their out-connections using brainPlaces.exchangeAll().

Lastly, each neuron reads through its inMessages and adds any received signals (from its in-connections) to its total inputSignal. This signal will be sent out at the end of the next iteration, and the simulation continues with neuron growth at the start of the next iteration.

Changes to the MATSIM Initialization Program Specification

During this quarter, the initialization program for the benchmark *MATSIM* was also re-evaluated. The existing initialization program needed quite a few complex algorithms that could be considered outside the scope of this research project, so the MATSIM platform itself was revisited to create a network generation program that was closer to the source simulation.

After some research, I found that the *MATSIM* platform uses OpenStreetMap, Osmosis, and the function "RunPNetworkGenerator" to build the road system from existing, real-world OSM data and create a *MATSIM* network file. [1] The *MATSIM* network file is written in XML and contains a node adjacency list of the network as well as distances and road capacities between the nodes. [2] This XML file would be ideal for importing a realistic road network for our versions of *MATSIM* across the three ABM platforms.

Then, the demand (start and end points of each agent's trips) can be either generated synthetically by running the function "RunPPopulationGenerator". Alternatively, the demand can be created using a census file and the function "RunZPopulationGenerator" [1]; however, more research must be done to verify that the output of these population functions would be in a useful format for initializing the MASS C++, FLAME, and RepastHPC versions of this simulation.

Test Programs for the Updated MASS C++ Core

In addition to the work done for the main research project, I aided in debugging the most updated version of MASS C++ core by writing two test programs, "simple_mass_test" and

"simple_neighbor_test", to verify that the old functionality as well as the new neighbor functions were working.

These test programs can be found on the hermes cluster in the following directories:

- simple_mass_test - /CSSDIV/research/dslab/mass_cpp/test
- simple_neighbor_test - /CSSDIV/research/dslab/mass_cpp/neighbors_test

When using these tests, an illegal free issue was found during runtime as well as some strange relative indexes for newly-added neighbors. These issues will be investigated during future quarters.

Looking Forward: Plan for Future Quarters

My goal for the week between spring and summer quarter is to read and learn the code of the MASS C++ platform so I can debug some issues the updated version was having.

My goals for the summer quarter are as follows:

- finish editing the *MATSIM* specification to reflect the changes outlined in this document
- complete the *MATSIM* initialization program(s) according to the revised specification
- write the remaining specifications for *VDT* and *Bail-In/Bail-Out*
- implement the remaining two benchmarks for FLAME: *VDT* and *Bail-In/Bail-Out*
- learn how to use the RepastHPC platform
- implement at least three agent-based model benchmarks for RepastHPC

Sources

[1] Rieser, M, Nagel, K and Horni, A. 2016. Generation of the Initial MATSim Input. In: Horni, A, Nagel, K and Axhausen, K W. (eds.) The Multi-Agent Transport Simulation MATSim, Pp. 61–64. London: Ubiquity Press. DOI: http://dx.doi.org/10.5334/baw.7. License: CC-BY

[2] Network.xml. Matsim-org/matsim-libs. (2018, June 21). Retrieved June 13, 2020, from https://github.com/matsim-org/matsim-libs/blob/master/examples/scenarios/equil/network.xml