Development of FLAMEGPU2 and MASS CUDA Benchmark Programs (Heat2D)

John Nguyen

CSS 497: Winter 2024 Term Report

Professor Munehiro Fukuda

March 13, 2024

Contents

[1. Introduction 3](#_Toc162202320)

[1.1 Motivation 3](#_Toc162202321)

[1.2 Project Goal 3](#_Toc162202322)

[2. Background 3](#_Toc162202323)

[2.1 Heat2D 3](#_Toc162202324)

[2.2 MASS Place V2 4](#_Toc162202325)

[3. Implementation 5](#_Toc162202326)

[3.1. FLAME GPU 2 Heat2D 5](#_Toc162202327)

[3.1.1. Place Agent 5](#_Toc162202328)

[3.1.2. Environment 6](#_Toc162202329)

[3.1.3. Agent Functions and Messages 7](#_Toc162202330)

[3.1.4. Layer 8](#_Toc162202331)

[3.1.5. Initialization 9](#_Toc162202332)

[3.2 MASS CUDA Heat2D 9](#_Toc162202333)

[3.2.1. Heat2d.cu 9](#_Toc162202334)

[3.2.2. Metal.h 10](#_Toc162202335)

[3.2.3. Metal.cu 11](#_Toc162202336)

[4. Verification 12](#_Toc162202337)

[4.1. FLAME GPU 2 Heat2D 12](#_Toc162202338)

[4.2. MASS CUDA Heat2D 14](#_Toc162202339)

[5. Conclusion 14](#_Toc162202340)

[5.1. Summary 14](#_Toc162202341)

[5.2. Future Development 14](#_Toc162202342)

[6. Appendix 15](#_Toc162202343)

[Code Location 15](#_Toc162202344)

[How to Run 15](#_Toc162202345)

# Introduction

## Motivation

MASS CUDA and FLAME GPU 2 are both GPU-based libraries used to simulate agent-based models (ABMs) in parallel. Both libraries have an older C++ version, which runs on a cluster system, that has a set of benchmarking applications to test performance. The motivation to switch to GPUs comes from the main challenge of ABMs, spatial scalability, and fast execution speeds. GPUs have been seen as a preferable option in comparison to cluster computing due to the number of computing cores (CPUs have around 8 to 16 cores, while GPUs have thousands), enabling ABMs to have high parallel processing power to meet their needs to scale.

FLAME GPU 2 is a well-known GPU library to parallelize ABM programs. Meanwhile, the MASS CUDA Library is still being developed by the Distributed Systems Laboratory (DSLab) at the University of Washington Bothell. By creating these benchmarking applications for both FLAME GPU 2 and MASS CUDA, the DSLab can understand the strengths and challenges of the MASS CUDA Library.

## Project Goal

The goal is to create benchmarking applications that can be used to measure the execution performance of each library. The applications created in this project will be used in a comparison between the programmability and execution performance of the FLAME GPU 2 and MASS CUDA libraries.

# Background

## Heat2D

The Heat2D simulation, which can be seen in Figure 1, simulates the dispersion of heat across a 2D square grid. The initial starting point for the heat begins on the top-middle-third of the edge of the square. This part of the square will be heated for an amount of time, which may or may not be the whole simulation time. The dispersion of the heat is simulated using the Forward Euler method, each space inside of the grid, except for those on the edge, will the Euler method to calculate its temperature. This calculation uses the space’s neighbors, these neighbors are the spaces above, below, to the left, and to the right of the space that is calculating its temperature. Spaces in the edges do not have all the required neighbors to do the calculation, so these spaces will copy the temperature of the neighboring spaces towards the center of the square.

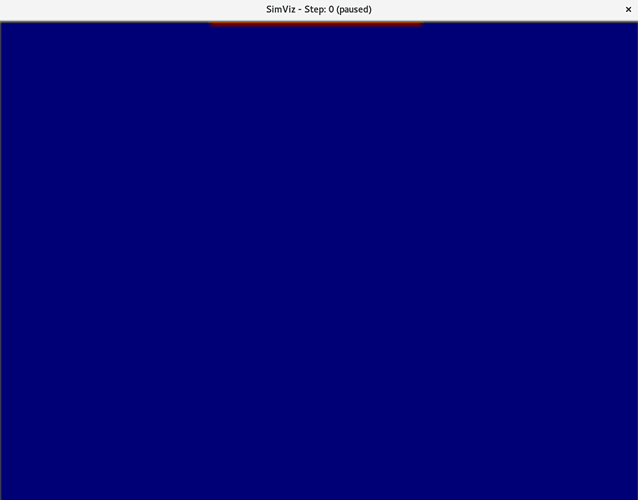


Figure 1. Heat2D Simulation

## MASS Place V2

During this quarter, MASS CUDA received a new update to its places. These changes greatly improved MASS CUDA’s initialization time and simulation step time. Places still function similarly to how they have worked before, but due to the changes, older benchmark applications need adjustments to their Place code to properly utilize the new Place V2 added to the MASS CUDA library.

Originally this quarter, I was going to port Social Net to FLAMEGPU2, but as these changes came out, Professor Fukuda decided that I should update MASS CUDA Heat2D to work with Place V2.

# Implementation

## FLAME GPU 2 Heat2D

To create a FLAME GPU 2 simulation a ModelDescription object must be made. This object contains information that is used to specify agents, messages, environment properties, and how they will interact. The hierarchy of a ModelDescription can be seen in Figure 2 below.

A diagram of a diagram

Description automatically generated

Figure 2. ModelDescription Hierarchy

### Place Agent

Unlike MASS CUDA, there is no concept of a place, instead, agents are used as the primary executors of the model’s behavior. All agents of the same type and state are executed in parallel.

For the implementation of the Heat2D simulation, I only used one agent called “place”. This agent represents a single space on the square. This agent can be seen in Figure 3.

A screen shot of a computer code

Description automatically generated

Figure 3. Place Agent

These “place” agents contain three variables:

* **“pos”** – an unsigned integer array of size 2. Represents the position of the place agent within the square.
* **“heat\_value”** – a double. Represents the temperature of a place.
* **“time\_unit”** – an unsigned integer. Represents how long the simulation has been running.

There are also four Agent Functions used by this agent, more information about the agent functions will be in the “3.1.2. Agent Functions and Messages” section.

### Environment

Environments in FLAME GPU2 are used to hold global state information by using properties and macro properties. For the Heat2D simulation, I only used properties.

Four main properties are used within the Agent Functions:

**“r”** – a double used as a coefficient for the Forward Euler method, which is used to calculate a place agent’s heat\_value.

**“middle\_heat**” – a double that is the maximum heat value of the simulation and the heat the top middle third of the square is set.

**“size”** – an integer of how big the length of the square is. Used in agent functions to get the position of certain areas of the square such as whether place agents are on or not on the edge.

**“heat\_time”** – an integer of how long (in steps) the top middle third of the square will be heated.

A computer code on a black background

Description automatically generated

Figure 4. Main Environment Properties

### Agent Functions and Messages

Heat2D four Agent Functions used by this agent:

**start** – Sends agent information into a MessageArray2D. This MessageArray2D can be used as an input for another function. Specifically, “start” has a message output called “set\_up” and this message is used as an input for the “copy\_and\_heat” agent function. The message contains each agent’s heat value, and each agent’s position is used to index the values.

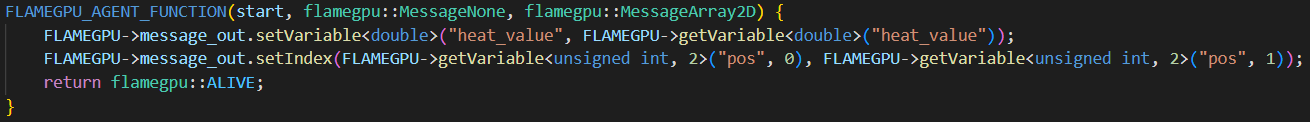


Figure 5. Code for "start" Agent Function

**copy\_and\_heat** – places on the edge of the square copy a neighbor closer to the center of the square’s heat. Since the heat\_values are coming for the MessageArray2D, place agents at the corner of the square cannot use the same logic. Their neighbors in the four cardinal directions do not have the correct heat\_value, as they are other edge places. Thus, place agents on the edge of the square must use its only diagonal neighbor.

**output** – Similar to “start”, “output” creates a MessageArray2D output with each agent’s heat\_value called “heat\_value\_message” and it is used as the input message for “update”. A new function with a different message is necessary because FLAME GPU 2 will raise an exception if there are multiple writes to the same index of a MessageArray2D.

**update** – for place agents that are not on the edge, they will calculate their new heat\_value using the Forward Euler method. The heat\_value information of their neighbors comes from the input MessageArray2D. As shown in Figure 6, the neighbors are the place agents above, below, to the left, and the right of the place agent. Place agents on the edge of the square cannot use the Forward Euler method because they do not have the required number of neighbors.

A screen shot of a computer code

Description automatically generated

Figure 6. Code Snippet of “update” Agent Function

### Layer

Layers are used to define the control flow of the model. Each layer is executed in sequence from top to bottom. Heat2D layers can be seen in Figure 7.

A screen shot of a computer program

Description automatically generated

Figure 7. Layers of Heat2D

Originally, “copy\_and\_heat” was done before “update” as this was how it was done in the older MASS CUDA Heat2D simulation. However, with this configuration the simulation logs were incorrect. It seems that since I was not using phases to keep track of the previous heat\_values like in the older MASS CUDA version, but instead MessageArray2D, I must use this execution order to get the correct output.

### Initialization

During initialization, each place agent is given a default heat\_value of 0, except for the places at the top middle third edge of the square (which start with a heat\_value of 19.0). I’ve also added extra variables to the initialization of the simulation which are:

**“heat\_time”** – an integer representing how many steps the top middle third of the square will be heated. The default value is 2700. Argument: --heat [int]

**“size”** – an integer representing how large the length of the square will be. For example, a size of 10 will produce a 10x10 square. The default value is 100. Argument: --size [int]

**“log\_int”** – an integer representing how often logs of the simulation will be produced. For example, if log\_int is set to 2, every 2 steps a log of the simulation will be created. The default value is 0 (no logs will be created). Argument: --log\_int [int]

These variables are modified using my own command line arguments, which are separate from the FLAME GPU 2’s configuration options. Use -u before using these arguments to silence FLAME GPU 2’s unknown arguments warnings.

## 3.2 MASS CUDA Heat2D

### 3.2.1. Heat2d.cu

MASS CUDA PlaceV2’s initialization of Places is different, as the attributes are defined using setAttribute() rather than in a separate state file such as MetalState.h. The old initialization of places can be seen in Figure 8, and the new way of initializing places can be seen in Figure 9.

A computer screen shot of a code

Description automatically generated

Figure 8. Old Initialization of Places

A computer screen shot of a program

Description automatically generated

Figure 9. New Initialization of Places

Otherwise, the main logic of runMassSim() is still the same as the older version of MASS CUDA Heat2D. For the simulation time, if the time is less than the heat\_time it will use callAll()to make each place use the APPLY\_HEAT function and then use exchangeAll() to make each place use the EULER\_METHOD function.

### 3.2.2. Metal.h

Based on GameOfLife\_PlaceV2 the functions used in callMethod() and the place’s attributes need to be enumerated, this can be seen in Figure 10.

A screen shot of a computer program

Description automatically generated

Figure 10. Snippet of Metal.h

It seems like to run the MASS simulation only APPLY\_HEAT and EULER\_METHOD are needed. In the older MASS Heat2D, there seem to be other ways of simulating Heat2D, and these other functions are used in those simulations. I will get rid of the unnecessary functions to run the MASS simulation at a later date.

### 3.2.3. Metal.cu

The main difference is the method of getting indexes and the attributes. Figure 11 and Figure 12, snippets of eulerMethod() can be used to notice the differences. Getting the size was harder to do inside the scope of Metal.cu, so I passed size as an argument during callAll() and exchangeAll(), because both APPLY\_HEAT and EULER\_METHOD use it.

A computer screen shot of a program

Description automatically generated

Figure 11. Old eulerMethod()

A computer screen shot of text

Description automatically generated

Figure 12. New eulerMethod()

# Verification

## FLAME GPU 2 Heat2D

For the visualization of Heat2D, I decided to use FLAME GPU 2’s built-in logging to export the step information to a JSON file. I built a separate program that reads these JSON files and creates a visualization of the simulation. This visualization is based on Professor Munehiro Fukuda’s Heat2D for CSS 434. Using the visualizer will a visual of the simulation based on the JSON files as seen in Figure 13 and Figure 14.

A screen shot of numbers and letters

Description automatically generated

Figure 13. Step 9 of 20x20 Square

A black background with white numbers

Description automatically generated

Figure 14. Step 99 of 20x20 Square

I have tested the results using various size squares ranging from 10x10 to 100x100 and compared the results to the CSS 434 Heat2D output and have consistently gotten the same results. At this point, I think that the FLAME GPU 2 Heat2D simulation is complete. If I have time next quarter, I will investigate ways to further optimize the application.

## MASS CUDA Heat2D

The new version of MASS CUDA Heat2D can run, however, while verifying the results I noticed that the simulation was not working as intended. As the square was heated from time 0 to time 1, there was an extra place that got heated to 19.0. I suspect that this issue occurred due to the neighbors, which is unexpected as I used the same integer arrays to initialize the neighbors which can be seen in Figure 8 and Figure 9 in section 3.2.1. I plan to test the application multiple times using different neighbors to see if it changes the results.

After getting the simulation to work correctly, I want to test if Simviz is still working as a way to visualize this simulation.

# Conclusion

## Summary

During the Winter 2024 quarter, I have successfully finished porting Heat2D to FLAME GPU 2 and I have a visualizer for the simulation. I have validated the results of this simulation by running a different Heat2D program from CSS 434 and comparing the output from each program. As for the updated MASS CUDA Heat2D, I have run into an unexpected blocker, although the program runs, it is not working as intended. Thus, some time to debug the application will be needed.

## Future Development

My plans for development in the future are to work on debugging the new MASS CUDA Heat2D, and after successfully debugging it, I would like to try to use Simviz to visualize it. Once MASS CUDA Heat2D is finished, I will move on to porting the Social Net simulation to FLAME GPU 2 and updating it in MASS CUDA. Finally, after finishing those applications, I can perform benchmark comparisons of Heat2D and Social Net to test each library’s programmability and performance.

# Appendix

**UPDATE 3/24/2024**: I fiddled with the neighbors in Heat2D\_PlaceV2 and got better results which were closer to the expected results. However, these results are still incorrect, the borders are not being set correctly. I do not think this is an issue involving the new MASS CUDA, rather an issue with the original implementation. I think the problem stems from how the calculation of each place’s temperature using the Euler method is being done in parallel with setting the borders.

## Code Location

For now, the code is on my own public repository. This repository can be found using the following URL:

<https://bitbucket.org/jworkspacetest999/john_develop/src/main/>

Each implementation of Heat2D can be found in the directories named after the library used to create the application.

## How to Run

**FLAME GPU 2 Heat2D:**

To recreate the output shown in Figure 13 and 14.

1. mkdir -p build && cd build && mkdir -p out
2. cmake .. -DCMAKE\_BUILD\_TYPE=Release -DCMAKE\_CUDA\_ARCHITECTURES=61
3. cmake --build . --target flamegpu heat2d -j 8
4. ./bin/Release/heat2d --steps 100 -u --size 20 --log\_int 1
5. Switch to the visual directory.
6. javac -cp gson-2.10.1.jar: Heat2DVisualizer.java
7. java -cp gson-2.10.1.jar: Heat2DVisualizer

Using *--help* will provide print a command line interface of FLAME GPU 2’s bulit-in configuration options.

Here are some flags that I created that can be used to change different aspects of the simulation:

*use -u to before typing these arguments to silence FLAMEGPU2's unknown argument warnings.*

*--heat [int] 🡪 Number of steps the middle of the square will be heated*

*default value: 2700*

*--size [int] 🡪 Side length of the square (ex. --size 10 ==> 10 x 10 square)*

*default value: 100*

*--log\_int [int] 🡪 Simulation logging interval*

*default value: 0 (no logging)*

**MASS CUDA Heat2D:**

Heat2D\_PlaceV2 is still debugging and has no visualizer yet, however it is still possible to run. Here are the steps to running Heat2D\_PlaceV2:

1. make develop
2. make build
3. ./bin/Heat2D\_PlaceV2

Using the --help flag will print different flags that can be used to define simulation parameters.

You must use the --verbose flag to see the square. For example, a line I am using to debug the application is:

./bin/Heat2D\_PlaceV2 --interval 1 --verbose

Currently the default values for Heat2D have been changed for my own testing purposes. I will change them back to the original after I finish debugging Heat2D\_PlaceV2.