Winter 2026 Term Report

Zad Casteneda and Rhys Buckeye 3/19/2026

Introduction

Both MASS CUDA and FLAMEGPU2 are libraries designed to run agent-based simulations using NVIDIA GPU's. While they both achieve this goal with parallel processing, they differ in architecture and approach, resulting in different efficiencies.

Goal

Since MASS CUDA and FLAME GPU2 both have the same goal but get there differently we seek to see the runtime differences between the libraries. We conducted this by running the same benchmarks on both systems, with the same simulation parameters, and comparing runtimes.

Key findings

MASS CUDA and FLAME GPU2 share a total of four benchmarks including: Game of Life, Tuberculosis, Sugarscape, and Neural Net. Below are the compared runtimes and specific benchmark parameters.

Game of Life:

Parameters	**1,000,000 steps**	**100 x 100 size**
	Total Processing Time (sec)	Avg Step Time (ms/ step)
MASS CUDA	264.73	0.25
FLAME GPU2	250.31	0.40
Parameters	**1,000,000 steps**	**200 x 200 size**
MASS CUDA	78.3	.08
FLAME GPU2	134.26	0.12

Tuberculosis

Parameters	**300 steps 32 x 32 size**
	Processing Time (s)
MASS CUDA	56.23
FLAME-GPU2	163.43
Parameters	**300 steps 100 x 100 size**
MASS CUDA	591
FLAME-GPU2	84.25

Sugarscape:

Parameters	**1,000,000 steps**	**256 x 256 size**
	Total Processing Time (s)	Average Step Time (ms/ step)
MASS CUDA	282s	03.12
FLAMEGPU2	264.73	1.40
Parameters	**1,000,000 steps**	**512 x 512 size**
MASS CUDA	890	.89
FLAMEGPU2	1663s	1.632

Neural Net/ Brain Grid:

Parameters	**1,000,000 steps**	**100 x 100 size**
	Total Processing Time (sec)	Average Step Time (ms/ step)
MASS CUDA	62.21	0.062
FLAMEGPU2	453.89	0.392
Parameters	**1,000,000 steps**	**200 x 200 size**
MASS CUDA	190	0.19
FLAMEGPU2	1653.91	1.6

Issues Encountered

MASS: Tuberculosis and Neural net were written for older versions of MASS. This caused compilation issues, a workaround was found and the readme of both benchmarks has been updated.

FLAME: When trying to get the FLAME GPU2 program to run one issue encountered was each benchmark consistently throwing "*The -std=c++20 flag is not supported with the configured host compiler.*" A workaround used to run the benchmarks was by using this command "*scl enable gcc-toolset-12 bash*."

Conclusions

MASS CUDA executed simulations up to ~50% faster than FLAME-GPU2 when the simulation size was increased, with exception of the Tuberculosis benchmark. At smaller simulation size, the two were much more comparable. Again however, Tuberculosis was the exception and in fact we found the opposite trend to be true. MASS-CUDA was faster at a smaller simulation size, FLAME-GPU2 was faster at larger ones.

MASS CUDA Benchmark Phase Based Update System

One of the most important changes made during this quarter was the adoption of a phase based update mechanism, commonly referred to as *double buffering*, to eliminate nondeterminism.

In the original implementation, many benchmarks performed reads and writes to shared state within a single kernel invocation. While logically correct in a sequential model, this approach becomes problematic on the GPU, where threads execute concurrently without a guaranteed order. As a result, some threads may read neighbor values before they are updated, while others read values after they have been updated within the same timestep. This leads to nondeterministic simulation results.

To address this, I refactored benchmark applications to use a **two-phase update model**:

- Each state variable is stored in a buffer of size two (e.g., `temperature[2]`)
- A `PHASE` attribute tracks which buffer is currently active
- During computation:
  - All reads are performed from the **current phase buffer**
  - All writes are directed to the **next phase buffer**
- After computation completes, a second kernel invocation updates the phase:
  - `phase = 1 - phase`

This phase flip ensures that:

- All threads read from a **consistent snapshot of the system state**
- No thread observes partially updated neighbor data
- The simulation remains deterministic regardless of GPU scheduling

This pattern was implemented in the Heat2D benchmark and extended to other applications, forming the foundation for deterministic execution across MASS CUDA benchmarks.

Guidelines for Avoiding Concurrency Issues in Benchmark Development

Through this work, several recurring patterns of nondeterminism were identified. Based on these findings, I propose the following guidelines for future benchmark developers working within MASS CUDA or similar GPU-based frameworks:

1. Separate Read and Write Phases

Avoid performing reads and writes to the same logical state within a single kernel invocation. Instead:

- Use a **compute phase** that reads from a stable state
- Follow with a **commit phase** that applies updates

This prevents execution-order–dependent behavior.

---

2. Use Double Buffering for Shared State

Any attribute that depends on neighbor values (e.g., temperature, population, health) should be:

- Stored in multiple buffers

- Accessed through a phase-controlled indexing scheme

This ensures consistent data visibility across threads.

---

3. Do Not Assume Execution Order

GPU threads execute in parallel and may be scheduled unpredictably. Code should not rely on:

- Neighbor updates happening before or after the current thread
- Implicit synchronization within a kernel

All ordering guarantees must be enforced through kernel boundaries.

---

4. Treat `callAll()` as a Synchronization Boundary

Each invocation of `callAll()` represents a full kernel execution:

- All threads complete before the next call begins
- Use multiple `callAll()` calls to structure computation safely

For example:

```
callAll(COMPUTE_NEXT_STATE)
callAll(COMMIT_STATE)
```

---

5. Validate Determinism Explicitly

Determinism should be tested, not assumed. Recommended workflow:

- Run simulations multiple times with identical inputs
- Output raw state to binary files
- Compare outputs using tools such as `cmp`

Even small inconsistencies indicate underlying concurrency issues.

---

6. Be Cautious with Neighbor-Dependent Logic

Any computation involving neighbors (e.g., averaging, diffusion, migration decisions) is particularly sensitive to nondeterminism. Ensure that:

- Neighbor values are read from a stable, unmodified state
- Updates are deferred until all reads are complete

By following these guidelines, future benchmark implementations can avoid many of the concurrency-related issues encountered during this project and produce reliable, reproducible results.

Appendix:

How to run MASS CUDA Benchmarks:

1. Setup and execution of all benchmarks except Tuberculosis and Braingrid

Follow instructions in readme's for all benchmarks except Tuberculosis and Braingrid. For those benchmarks you must do the following.

The following instructions were provided by *Robert Foskin* and adapted for use in this report.

Tuberculosis:

1. Clone https://bitbucket.org/mass_application_developers/mass_cuda_appl/src/main/
2. Checkout holto/Tuberculosis -> "git checkout holto/Tuberculosis
3. Uncomment out line 16 in the Make file. We need to checkout the tag "0.7.1"
4. Make "install-mass:" look like this:

```
109  # previous checkout: cd $(MASS_REPO) && git checkout $(MASS_VERSION)
110  # cd $(MASS_REPO) && git checkout develop && git pull origin develop
111  install-mass:
112      echo "Installing MASS"
113      mkdir -p $(LIB_DIR)
114      rm -rf $(MASS_DIR)
115      mkdir -p $(MASS_DIR)
116      cd $(MASS_DIR) && git clone $(MASS_GIT_URL) $(MASS_REPO)
117      cd $(MASS_REPO) && git checkout $(MASS_VERSION)
118      cd $(MASS_REPO) && NVCC_OPTS="$(MASS_OPTS)" make install
119
```

5. Run Make develop in terminal -> it will fail due to outdated boost reference in MASSCUDA 0.7.1
6. In the file directory go to lib/mass/mass_repo and open make file
7. Change line 54 by removing test from the end of the line. It should look like this:

```
54    install:: clean install-google-test install-boost build
```

   This is due to the tests not passing

8. Change the url on line 69 for the boost library to https://archives.boost.io/release/$(BOOST_VERSION)/source/boost_$(BOOST_TAG).tar.gz

   It should look like this:

```
69      cd $(BUILD_DIR)/boost && wget -O boost-$(BOOST_VERSION).tgz https://archives.boost.io/release/$(BOOST_VERSION)/source/boost_$(BOOST_TAG).tar.gz
```

9. Go to Dispatcher.cu and comment out the contents of spawnAgents, don't comment out the entire function

10. Go to the tuberculosis make file and make a function "install-mass-2" that copies the last line of install-mass:

```
# previous checkout: cd $(MASS_REPO) && git checkout $(MASS_VERS
# cd $(MASS_REPO) && git checkout develop && git pull origin dev
install-mass:
    echo "Installing MASS"
    mkdir -p $(LIB_DIR)
    rm -rf $(MASS_DIR)
    mkdir -p $(MASS_DIR)
    cd $(MASS_DIR) && git clone $(MASS_GIT_URL) $(MASS_REPO)
    cd $(MASS_REPO) && git checkout $(MASS_VERSION)
    cd $(MASS_REPO) && NVCC_OPTS="$(MASS_OPTS)" make install

install-mass-2:
    cd $(MASS_REPO) && NVCC_OPTS="$(MASS_OPTS)" make install
```

11. Add this to "develop" In the make file, this will ensure that the masscuda src files that you changed will be compiled:

```
79  develop:: install-mass-2 install-google-test build-simviz-lib
80      mkdir -p build
81      mkdir -p bin
82      mkdir -p lib
83      echo "$(PROJECT_NAME) Dependencies installed."
```

12. Go to terminal and run "make develop" -> installs all libraries including masscuda

13. Go to tuberculosis.main and change output interval from 0 to 1

14. Go to terminal and run "make build" -> compiles TB application

Braingrid (Neural Net):

1. Make "install-mass:" look like this:

```
109  # previous checkout: cd $(MASS_REPO) && git checkout $(MASS_VERSION)
110  # cd $(MASS_REPO) && git checkout develop && git pull origin develop
111  install-mass:
112      echo "Installing MASS"
113      mkdir -p $(LIB_DIR)
114      rm -rf $(MASS_DIR)
115      mkdir -p $(MASS_DIR)
116      cd $(MASS_DIR) && git clone $(MASS_GIT_URL) $(MASS_REPO)
117      cd $(MASS_REPO) && git checkout $(MASS_VERSION)
118      cd $(MASS_REPO) && NVCC_OPTS="$(MASS_OPTS)" make install
119
```

2. Run Make develop in terminal -> it will fail due to outdated boost reference in MASSCUDA 0.7.1

3. In the file directory go to lib/mass/mass_repo and open make file

4. Change line 54 by removing test from the end of the line. It should look like this:

```
54   install:: clean install-google-test install-boost build
```

   This is due to the tests not passing

5. Change the url on line 69 for the boost library to https://archives.boost.io/release/$(BOOST_VERSION)/source/boost_$(BOOST_TAG).tar.gz

   It should look like this:

```
69     cd $(BUILD_DIR)/boost && wget -O boost-$(BOOST_VERSION).tgz https://archives.boost.io/release/$(BOOST_VERSION)/source/boost_$(BOOST_TAG).tar.gz
```

6. Go to the braingrid make file and make a function "install-mass-2" that copies the last line of install-mass:

```
# previous checkout: cd $(MASS_REPO) && git checkout $(MASS_VERS
# cd $(MASS_REPO) && git checkout develop && git pull origin dev
install-mass:
    echo "Installing MASS"
    mkdir -p $(LIB_DIR)
    rm -rf $(MASS_DIR)
    mkdir -p $(MASS_DIR)
    cd $(MASS_DIR) && git clone $(MASS_GIT_URL) $(MASS_REPO)
    cd $(MASS_REPO) && git checkout $(MASS_VERSION)
    cd $(MASS_REPO) && NVCC_OPTS="$(MASS_OPTS)" make install

install-mass-2:
    cd $(MASS_REPO) && NVCC_OPTS="$(MASS_OPTS)" make install
```

7. Add this to "develop" In the make file, this will ensure that the masscuda src files that you changed will be compiled:

```
79  develop:: install-mass-2 install-google-test build-simviz-lib
80      mkdir -p build
81      mkdir -p bin
82      mkdir -p lib
83      echo "$(PROJECT_NAME) Dependencies installed."
```

8. Go to terminal and run "make develop" -> installs all libraries including masscuda

How to run FLAME GPU2 benchmarks:

1. Prerequisites & Environment Setup

If any of the following give an error related to not using C++20, use the command "*scl enable gcc-toolset-12 bash*."

2. Core Repository Setup

The following steps are required for **all** benchmarks. You only need to do this once.

1. **Download/Clone** the FLAME GPU 2 library: GitHub - FLAMEGPU2
2. **Unzip** the repository: unzip FLAMEGPU2-master.zip
3. **Enter the directory**: cd FLAMEGPU2-master

3. Running Standard Examples

These models are included in the base FLAME GPU 2 repository.

Benchmark	GitHub Location	Build Target
Game of Life	examples/cpp/game_of_life	game_of_life
Sugarscape	examples/cpp/sugarscape	sugarscape

Build & Run Instructions:

```
Bash
# 1. Prepare build directory
mkdir -p build && cd build

# 2. Configure
cmake .. -DCMAKE_CUDA_ARCHITECTURES=61 -DCMAKE_BUILD_TYPE=Release

# 3. Build the specific target
cmake --build . --target <target_name> -j 8

# 4. Execute
./bin/Release/<target_name> --verbose --steps 10
```

Put the name of the benchmark you're trying to run in <target_name>.

4. Running External Benchmarks (Tuberculosis & Neural Net)

These examples require manual integration into the FLAME GPU 2 build system.

Benchmark	Source Repository	Build Target
Tuberculosis	BitBucket - Tuberculosis	tuberculosis
Neural Net	BitBucket - Neural Net	neural_net

Integration Steps:

1. **Download** the external source from the BitBucket links provided.
2. **Move** the directory into FLAMEGPU2-master/examples/cpp/.
3. **Modify CMakeLists.txt**: Open the root CMakeLists.txt and add the logic below to register the new examples.

CMake Modification Guide

Search for the section # Options to enable building individual examples and add:

CMake

```
# For Tuberculosis
cmake_dependent_option(FLAMEGPU_BUILD_EXAMPLE_TUBERCULOSIS "Enable building
examples/cpp/tuberculosis" OFF "FLAMEGPU_PROJECT_IS_TOP_LEVEL; NOT
FLAMEGPU_BUILD_ALL_EXAMPLES" OFF)

# For Neural Net
cmake_dependent_option(FLAMEGPU_BUILD_EXAMPLE_NEURAL_NET "Enable building
examples/cpp/neural_net" OFF "FLAMEGPU_PROJECT_IS_TOP_LEVEL; NOT
FLAMEGPU_BUILD_ALL_EXAMPLES" OFF)
```

Then, under `# Add each example`, add:

CMake

```
if(FLAMEGPU_BUILD_ALL_EXAMPLES OR FLAMEGPU_BUILD_EXAMPLE_TUBERCULOSIS)
    add_subdirectory(examples/cpp/tuberculosis)
endif()

if(FLAMEGPU_BUILD_ALL_EXAMPLES OR FLAMEGPU_BUILD_EXAMPLE_NEURAL_NET)
    add_subdirectory(examples/cpp/neural_net)
endif()
```

Build & Run: Follow the same build instructions as Section 3, using `tuberculosis` or `neural_net` as the target.

Helpful Execution Flags

- `--verbose`: Prints per-step processing time (Init, Step, and Exit).
- `--steps X`: Sets the simulation duration to X steps.

- `--seed X`: Ensures deterministic results (ideal for benchmarking).
- `--help`: Displays a full list of available command-line arguments.