Overview

This quarter, my research focused on debugging and improving the correctness of benchmark applications built on the MASS CUDA framework for large-scale agent-based modeling on GPUs. The primary goal of my work was to identify sources of nondeterminism in existing applications, understand how GPU execution order and memory access patterns contributed to these issues, and refactor application-level code to enforce deterministic behavior without sacrificing parallel performance.

Rather than emphasizing breadth across many benchmarks, my work this quarter concentrated on deeply understanding the MASS CUDA execution model and developing a repeatable methodology for detecting and resolving race conditions in GPU-based simulations.

Technical Focus and Methodology

My work centered on three core activities:

1. **Understanding MASS CUDA Execution Semantics**
   I studied how `callAll`, `exchangeAll`, and kernel execution are scheduled in MASS CUDA, with particular attention to how place-level and agent-level computations interact with shared device memory. This included analyzing how interleaving computation and update phases can lead to nondeterministic results even when no explicit data races are reported by tools such as CUDA Racecheck.

2. **Determinism Testing and Debugging Workflow**
   I established a determinism-testing workflow based on:

   - Running simulations multiple times with identical inputs
   - Writing deterministic binary output snapshots of simulation state
   - Using byte-level comparison (`cmp`) to detect nondeterministic behavior

3. This approach proved essential in identifying subtle logical race conditions that are not caught by traditional race detection tools but still produce divergent simulation results.

4. **Application-Level Refactoring Strategy**
   I then focused on refactoring benchmark applications to enforce a clear separation between:

    - **Read-only compute phases**, where threads only read from shared state
    - **Write/commit phases**, where updates are applied deterministically
5. This mirrors double-buffering techniques commonly used in parallel numerical solvers and GPU simulation frameworks.

Key Findings

Through this process, I identified a recurring source of nondeterminism across multiple applications:
State updates were being performed in the same kernel invocation that depended on neighbor reads, causing execution-order–dependent behavior across GPU threads.

A concrete example of this occurred in the Heat2D benchmark, where temperature updates and phase transitions were interleaved within a single computation step. Although the code was logically correct in a serial context, parallel execution caused different threads to observe partially updated state, resulting in nondeterministic outputs across runs.

To resolve this, I refactored the application to:

- Introduce explicit phase buffers
- Perform all reads from a stable “current” buffer
- Defer writes to a “next” buffer
- Commit phase transitions in a separate kernel invocation

After this change, repeated runs of the simulation with identical parameters produced identical binary output, confirming deterministic behavior.

Progress Across Benchmarks

This quarter, I successfully:

- Identified and diagnosed sources of nondeterminism in GPU-based MASS CUDA benchmark applications by analyzing kernel execution order and shared-memory access

patterns.

- Designed and implemented a deterministic refactoring strategy that separates computation and state update phases using explicit buffering and multi-kernel execution.

- Fully refactored and validated deterministic behavior in benchmarks **Game of Life, Heat2D, SugarScape, and SocialNetwork**

While not all benchmark refactors were completed within the quarter, this was a conscious tradeoff in favor of building a robust debugging framework and gaining a deep understanding of the execution model. I plan to complete the remaining benchmark refactors during the break between Autumn and Winter quarters, using the methodology developed this quarter.

Challenges Encountered

Several challenges shaped the direction of my work:

- **Tooling limitations**: CUDA race detection tools correctly reported no data races in several cases where nondeterminism still occurred, requiring deeper reasoning about execution ordering rather than relying solely on tooling.

- **Incomplete abstractions**: Certain MASS CUDA APIs (e.g., `exchangeAll`) are not fully implemented for true data exchange, which required application-level workarounds and careful kernel sequencing.

- **Build and environment complexity**: Debugging large CUDA-based codebases on shared HPC systems required significant effort in build system configuration and dependency management.

These challenges reinforced the importance of systematic debugging practices and precise mental models of GPU execution.

Skills and Learning Outcomes

Through this research, I gained substantial experience in:

- CUDA C++ programming and kernel-level debugging

- Diagnosing nondeterminism in parallel systems
- Designing deterministic update schemes for GPU simulations
- Reading and modifying large, research-oriented codebases
- Communicating technical findings clearly in a research setting

Most importantly, this work strengthened my ability to reason about correctness in parallel computing environments — a skill directly applicable to future work in GPU engineering, high-performance computing, and large-scale simulation.

Future Work

During the break between Autumn and Winter quarters, I plan to:

- Complete deterministic refactors of the remaining benchmark applications
- Consolidate findings into clear documentation for future MASS CUDA users
- Develop a small determinism testing guide or checklist for new researchers

These efforts will build directly on the foundation established this quarter and help ensure that MASS CUDA benchmarks serve as reliable, reproducible research tools.

Conclusion

Although this quarter emphasized depth over breadth, the work completed represents meaningful progress toward improving the correctness and reliability of MASS CUDA applications. The debugging strategies, testing workflows, and refactoring techniques developed during this period have already proven effective and will continue to guide my work moving forward. This experience has prepared me well for future research and industry work in parallel computing and GPU systems.
