# CSS 600: Independent Study
# Lisa Kosiachenko
# Spring 2017

# Term Report

**University of Washington**
**2017**

# Table of contents

## Goals

The goal of the proposed thesis is to implement the techniques to optimize the performance of the agent-based models on the graphics processing units (GPU) as part of the MASS CUDA library based on the previous experiments and following the existing specification and APIs for MASS CUDA.

The scientific contribution of this work is generalizing the existing techniques for efficient implementation of agent-based models on the GPU, implementing them as part of the general-use library for the agent-based models and understanding under what applications, what conditions, and what problem sizes which approaches and techniques are the best choices. Very little work has been done in the area of general-use libraries for the ABM simulations on GPU (see Related work section) and the proposed work will be beneficial to the research community working on parallelization of large-scale agent-based simulations on GPUs.

## Criteria

The scope of the proposed thesis work is to test different approaches and techniques for implementing agent-based models in the GPU, test the performance of their implementations and identify their best use cases.

The aspirational target is to achieve an improved performance of the MASS CUDA library as a result of implementing new approaches and techniques.

## Related works

### Previous work on MASS CUDA

There have been several students who have made progress on CUDA versions of the MASS Library[2-4].

Most of these students (Tosa Ojiru, Piotr Warczak and Robert Jordan) worked on the GPU-parallelized Wave 2D simulation using CUDA APIs directly.

In 2014-2015 Nathaniel Hart developed a coherent MASS CUDA library that allows users to create and execute agent-based models on a GPU using MASS library APIs (without necessarily knowing anything about CUDA). The primary goal of Nathaniel's work was to implement the encapsulation of the details of GPU parallel programming, and it has been accomplished. However, the resulting library did not achieve the performance goals as it showed a performance slowdown when compared to sequential computation of an identical simulation - the resulting performance of the system was 19% to 54% slower compared to the sequential execution of the program depending on the problem size.

The current work aims to build up on top of Nathaniel's work, in particular to reuse the APIs and utilize the results of experiments and profiling of Nathaniel's test application.

## Previous work on the Agent-Based Modeling using GPU

There are a number of domain-specific implementations of Agent-Based Models on GPU that prove to provide good performance results and significant speed-up compared to the sequential execution. There are also some research papers dealing with the problems of abstracting out the specifics of the implementation and

Following is the brief overview of research papers identified grouped by the application domain. More details on some of these papers are provided in the section Preliminary Research.

1) Domain-specific agent-based models utilizing GPU:
   a) *Biology & Medicine*:
      i) Tuberculosis epidemic simulation[13];
      ii) Modeling of blood coagulation system[12];
      iii) Systemic inflammatory response simulation[6];
      iv) Protein structure prediction[10];
      v) Fish schooling simulation[15];
   b) *Physics*:
      i) Molecular dynamics simulation[25];
   c) *Mathematics*:
      i) Graph theory[11];
   d) *Social Studies*:
      i) Traffic simulation and traffic signal timing optimization [37,38,40,41,44];
      ii) Crowd simulation and path planning[7,8,9,29,30,42,43];
      iii) Bird flocking[19, 21];
      iv) Particle swarm optimization[24];
2) Agent-based modeling frameworks utilizing GPU:
   a) FLAME GPU: A High Performance Agent Based Modelling Framework on Graphics Card Hardware with CUDA[33-36];
   b) Turtlekit: Logo-based library for Multi-Agent-Based simulations utilizing GPU[28];
   c) MCMAS: An OpenGL-based toolkit to benefit from Many-Core Architecture in Agent-Based Simulations[45];
3) General techniques for implementing Agent-based models on GPU:
   a) Agent-based modeling techniques on multi-GPU clusters by Aaby, B. G., Perumalla, K. S. et al. from the Oak Ridge National Laboratory [5,32];
   b) GPU environmental delegation[16,17];
   c) Heterogeneous computing on CPU+GPU[18];
   d) Separating agent management module from agent interaction module[26];
4) Other:
   a) MPI-CUDA implementation for multi-GPU clusters[22];
   b) Benchmarking platform for ABMs[23];
   c) Task scheduling in ABMs on GPU[31].

## Preliminary Research

To conduct a preliminary research for the Thesis project I have completed the CSS600 Independent Study/Research course under supervision of prof. Munehiro Fukuda. As part of the research I studied the existing MASS CUDA code and documentation to understand the current library architecture and implementation techniques used. I also studied available research papers on efficient implementation of the agent-based models on GPU and

to developed a list of possible improvements to the MASS CUDA library based on the best practices in the field. Over 40 relevant papers were reviewed in order to identify potential improvement techniques that can be applied to the existing MASS CUDA library to achieve better performance.

## CUDA programmability model and its fitness to Agent-Based models

Graphics Processing Unit (GPU) provide an opportunity to accelerate the Agent-Based models execution. However, GPU is not spontaneously efficient to support ABS due to its specific programmability model and memory hierarchy, which are briefly outlined below[26].

GPU is composed of global memory (DRAM) and several stream multi-processors (SM). An SM has tens of stream processors (SP), which is equivalent to arithmetic-logic unit (ALU) of CPU. Each SM has tens of SPs but can support hundreds of concurrent threads by multiplexing controlled by the warp scheduler. SM creates, manages, schedules, and executes in groups of 32 parallel threads.

Every 32 threads form a warp executing in a lockstep manner. This is the single-instruction-multiple threads (SIMT) parallel programming model utilized by GPU. Full efficiency is realized when all threads of a warp agree on their execution path. At every instruction issue time, a warp scheduler selects a warp that has threads ready to execute its next instruction and issues the instruction to those threads. Execution context (program counters, registers, etc) of each warp processed by a multiprocessor is maintained on-chip during the entire lifetime of a warp. Therefore, switching execution between warps has no cost.

In CUDA, threads are organized into blocks. Threads of the same block run on the same SM and are not separable. One block runs on one SM, but one SM can support multiple blocks. Each SM has limited resources such as shared memories and registers. Blocks assigned to the same SM have to compete for these resources. If a single block requests too many resources, the number of blocks that can be concurrently supported by an SM decreases, and the performance will be affected. Fortunately, the numbers of blocks and the number of threads per block are configurable by developers. If one blocks requests too many resources, one can always reduce the number of threads per block to reduce the resources requested by one block.

The memory of GPU has a hierarchical design as shown in Figure 1[46]. The global memory can be accessed by all threads in the same or different blocks. Global memory is big but slow. On the other hand, the shared memory is limited in size but fast. It can only be accessed by threads within the same block.

The actual bandwidth of accessing data in global memory is critical to the performance of a CUDA application. The device coalesces global memory loads and stores into fewest transactions to maximize bandwidth utilization, when the threads of a warp access consecutive memory locations. In this favorable case, the advertised peak global memory bandwidth is achievable.

Shared memory is on-chip programmable cache with much smaller size but much higher peak bandwidth in comparison with off-chip global memory.

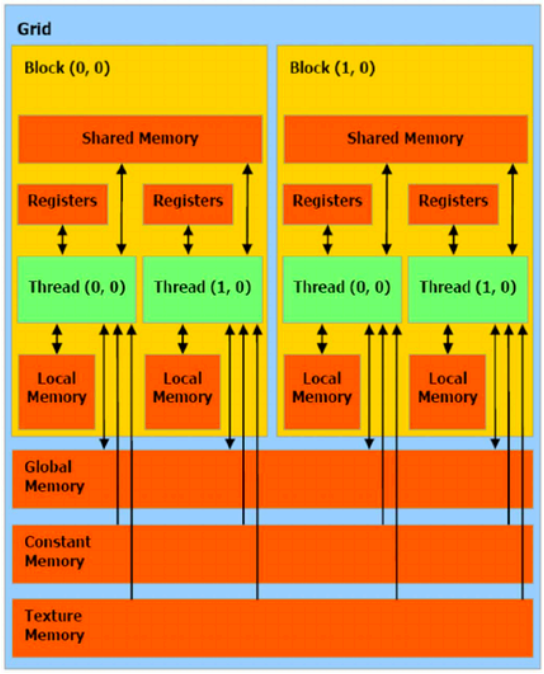

Figure 1. GPU memory hierarchy

As part of the literature survey there were identified a number of ***specific challenges*** related to implementing agent-based models on the GPUs:

- Incompatible memory access patterns:
  - Agent creation and termination requires *dynamic memory allocation*, which can be a severe bottleneck on the GPU, as it requires global synchronization of the device and thus stall of all the executing warps;
  - Neighbor searching can be inefficient on the GPU, as neighbors can be situated anywhere in the global memory and thus memory access is not *coalesced;*
- Branch divergence:
  - Agent-based models often include agents of different types, which have different behaviors and execution paths within a kernel. This results is thread divergence and thus reduces parallelism.

## Current MASS CUDA architecture and implementation details

The main goal of the previous work on the MASS CUDA focused on creating the API that works and feels in the manner identical to MASS Java or MASS C++ libraries.

The resulting library completely hides CUDA implementation details from the user with several exceptions:

- The project must be compiled using nvcc (Nvidia CUDA compiler) with the required flags/options;
- All files that are normally .cpp are .cu;
- Functions in user-defined Place or Agent classes should be prepended with the macro MASS_FUNCTION (stands for __host__ __device__), which enables compiling of both host and device code.

On the architectural level the project follows the Model-View-Presenter model, where View represents API of the library, Model represents the data model on the GPU and

Presenter represents the dispatcher coordinating the interaction of Model and View. The architecture is represented on the Figure 2 [2].

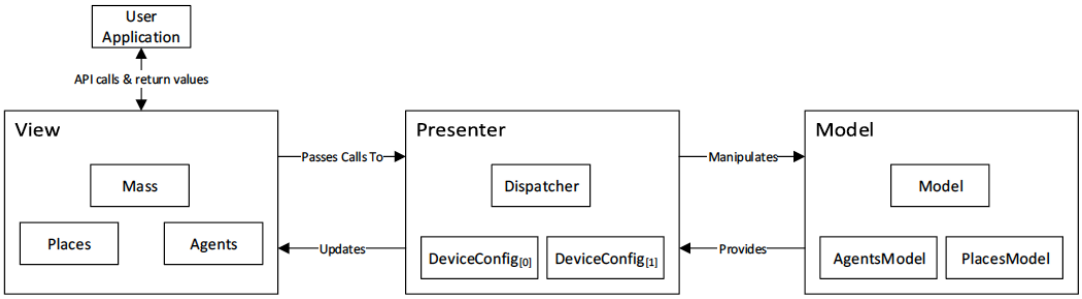


Figure 2. High-level architecture of MASS CUDA library

The files in the project folder are structured into "src" and "test" folders, where "test" is the Heat2D application.

The basic program flow and classes used by the Heat 2D application are represented on the Figure 3. The actual GPU kernel calls (such as *callAllPlacesKernel* and *setNeighborPlacesKernel*) start from the Dispatcher class. "MASS", "Places" and "Agents" classes have virtually no CUDA implementation details exposed. Another point to mention is that the manipulation of the DataModel is performed through the partitions("Partition" class), so that the code can be easily extended to run on different GPU devices. However, current implementation of the library can only run on a single GPU device.

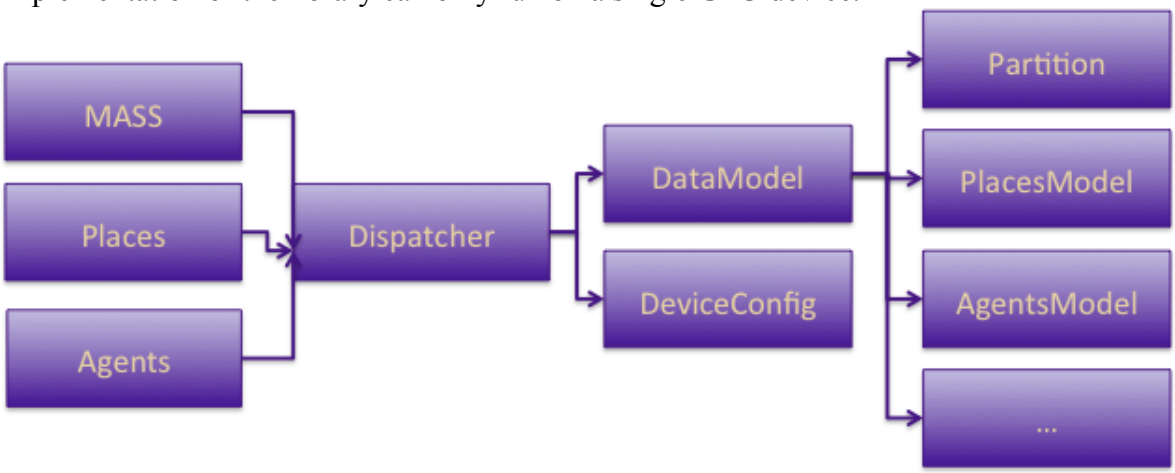


Figure 3. Program flow for Heat 2D application usage of MASS CUDA library

## MASS CUDA performance and profiling

While the goal of encapsulation of the details of GPU parallel programming has been successfully accomplished by the existing MASS CUDA code, the resulting library did not achieve the performance goals as it showed a performance slowdown when compared to sequential computation of an identical simulation.

As part of the preliminary research I performed a test run and profiling of the existing version of the MASS CUDA library using the Heat 2D application. As you can see on Figure 4, MASS CUDA performs approximately 20% more efficiently on the available hardware ("juno.uwb.edu" with Intel Xeon CPU E5-2630 v3 @ 2.40GHz and GeForce GTX Titan GPU with 2688 CUDA cores and compute capability of 3.5) than sequential CPU

implementation for significantly large simulations (simulation space of order 1000). However, when compared to the direct GPU implementation using CUDA API calls, the MASS CUDA version is 55 times slower. So, the potential for performance improvement is huge there.

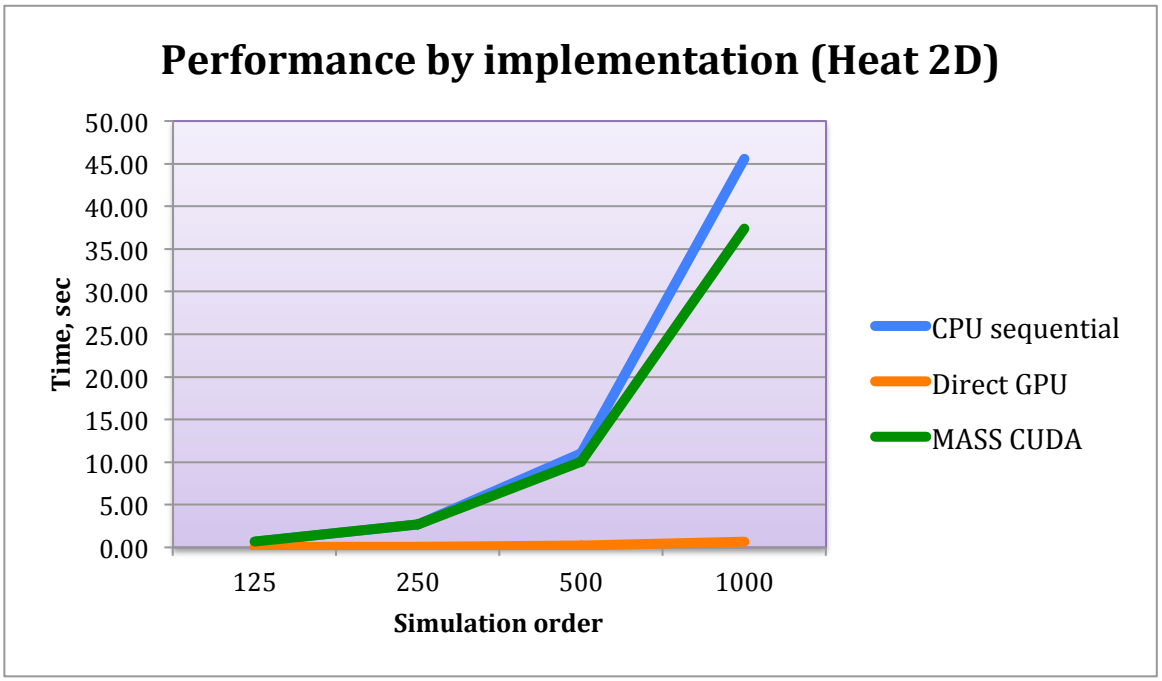


Figure 4. Execution time of different implementations of MASS CUDA library

Aside from measuring execution time for different version of the Heat 2D implementation, I also performed profiling of the MASS CUDA implementation using the "nvprof" profiling tool. As can be observed on the Figure 5, the majority of the execution time is taken by 3 kernels: *mass::setNeighborPlacesKernel*, *mass::callAllPlacesKernel* and *mass::instantiatePlacesKernel*.

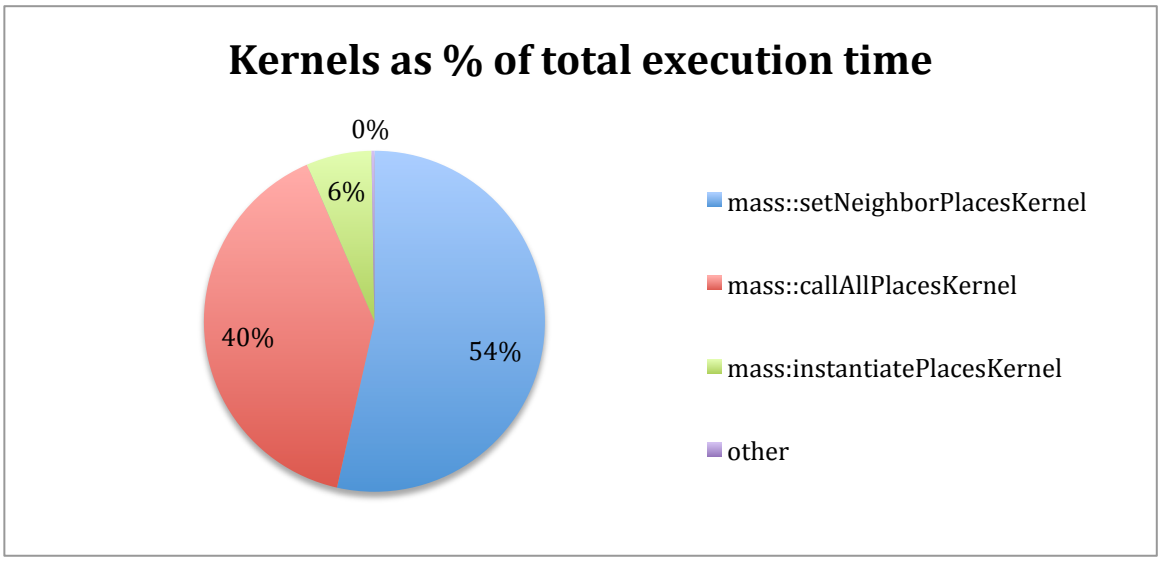


Figure 5. Heat 2D kernels execution times as a share of total

Further profiling of these kernels showed some apparent possibilities for performance optimization, such as increasing the utilization of the shared memory versus global memory.

Shared memory transaction on the Nvidia GPUs are performed up to 10 times faster than global memory transactions. Table 1 below demonstrates the current memory transactions by all three of the kernels. Also profiling revealed that for all three kernels the main reason for warp stall was memory throttle (over 65% of the cases), so the optimization of the memory usage should be a priority during the future work.

| | **mass::setNeighborPlacesKernel** | **mass::callAllPlacesKernel** | **mass:instantiatePlacesKernel** |
|---|---|---|---|
| Shared memory transactions | 638 267 | - | - |
| Global memory transactions | 21 757 737 | 11 114 410 | 3 539 155 045 |
| **% of shared memory transactions vs global memory transaction** | **3%** | **0%** | **0%** |

Table 1. Number of transactions by memory type for the Heat 2D test application

## Possible performance improvement techniques

As part of the literature survey stage the following techniques and approaches were identified as the most viable performance improvement techniques:

- Maximize the use of shared memory for the places objects and data;
- Data structure to store agents neighbor data in the shared memory (preliminary group agents into blocks by location in the grid)[26];
- Use Agent Pooling technique to addresses dynamic memory allocation issue[26];
- Heterogeneous approach: agent behaviors managed by the CPU, environmental dynamics (anything that doesn't modify agent state) handled by the GPU[16,17];
- Identify and implement a set of defined data structures and kernels utilized in popular ABMs (diffusion, path-finding, population dynamics)[45].

## Plan

**Autumn 2017 quarter**

| | |
|---|---|
| Week 1 | Detailed plan update |
| Week 2-4 | Technical survey |
| Week 5-10 | Technical implementation |
| Week 11 | Term Report |

**Winter 2018 quarter**

| | |
|---|---|
| Week 1-3 | Technical implementation |
| Week 4-5 | Testing of the technical implementation |
| Week 6 | Technical performance evaluation |
| Week 7-8 | Thesis defense preparation |
| Week 9-10 | Master thesis write-up |

## Required Resources

The resources required for the successful implementation, testing and performance

measurement of the current thesis requires specific hardware and software: modern CPU running Linux OS with CUDA-compatible GPU and the NVIDIA® CUDA® Toolkit installed. The required hardware is available on the "*juno.uwb.edu*" computer, which has Intel Xeon CPU E5-2630 v3 with 2.40GHz processor frequency and GeForce GTX Titan GPU with 2688 CUDA cores and compute capability of 3.5.

The required software was installed and tested on "juno" during the preliminary research stage.

## Constraints and Risks

Key risk for the proposed thesis is the possible inefficiency of the implemented techniques. Because very little work has been done in the area of general-use libraries for the ABM simulations on GPU, there is no guarantee that the techniques implemented will result in the improved performance of the MASS CUDA library. However, even if the thesis will fail to achieve significant performance gains for the MASS CUDA library, it will still have an important scientific value as it will generalize and evaluate the performance of the existing techniques for efficient implementation of agent-based models on the GPU as part of the library.

## References


1. T. Chuang, M. Fukuda, “A Parallel Multi-Agent Spatial Simulation Environment for Cluster Systems”, In Proc. of the 16th IEEE International Conference on Computational Science and Engineering - CSE 2013, pp. 143-150, Sydney, Australia, December, 2013. [Online]. Available: http://faculty.washington.edu/mfukuda/papers/cse2013_mass.pdf
2. Hart, Nathaniel B. “MASS CUDA: Abstracting Many Core Parallel Programming From Agent Based Modeling Frameworks”. Diss. University of Washington, 2015. Available: https://onedrive.live.com/view.aspx?resid=AFB10F7D10CB103C!36244&ithint=file%2cpdf&app=WordPdf&authkey=!APN9OEQ_n3ufVPk
3. P. Warczak, "Coordinating Multiple GPU Devices to Run MASS Applications," 2012. Available: https://depts.washington.edu/dslab/MASS/reports/PiotrWarczak_sp12.docx.
4. T. Ojiru, "Implementing the Multi-agent spatial simulation (MASS) library on the Graphics Processor Unit," 2012. Available: https://depts.washington.edu/dslab/MASS/reports/TosaOjiru_thesis.pdf.
5. Aaby, Brandon G., Kalyan S. Perumalla, and Sudip K. Seal. “Efficient Simulation of Agent-Based Models on Multi-Gpu and Multi-Core Clusters.” *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010. 29.
6. Alberts, Samuel et al. “Data-Parallel Techniques for Simulating a Mega-Scale Agent-Based Model of Systemic Inflammatory Response Syndrome on Graphics Processing Units.” *Simulation* 88.8 (2012): 895–907.
7. Bleiweiss, Avi. “Multi Agent Navigation on the GPU.” *Games Developpement Conference*. N.p., 2009. 39–42.
8. Caggianese, Giuseppe, and Ugo Erra. “Exploiting Gpus for Multi-Agent Path Planning on Grid Maps.” *High Performance Computing and Simulation (HPCS), 2012 International Conference on*. IEEE, 2012. 482–488.
9. Caggianese, Giuseppe, and Ugo Erra. “Gpu Accelerated Multi-Agent Path Planning Based on Grid Space Decomposition.” *Procedia Computer Science* 9 (2012): 1847–1856.
10. Campeotto, Federico, Agostino Dovier, and Enrico Pontelli. “Protein Structure Prediction on GPU: A Declarative Approach in a Multi-Agent Framework.” *Parallel Processing (Icpp), 2013 42nd International Conference on*. IEEE, 2013. 474–479.
11. Cecilia, José M. et al. “Enhancing Data Parallelism for Ant Colony Optimization on GPUs.” *Journal of Parallel and Distributed Computing* 73.1 (2013): 42–51.

12. Chen, W. et al. “Agent Based Modeling of Blood Coagulation System: Implementation Using a GPU Based High Speed Framework.” *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. N.p., 2011. 145–148.
13. D’Souza, Roshan M. et al. “Data-Parallel Algorithms for Agent-Based Model Simulation of Tuberculosis on Graphics Processing Units.” *Proceedings of the 2009 Spring Simulation Multiconference*. Society for Computer Simulation International, 2009. 21.
14. D’Souza, Roshan M., Mikola Lysenko, and Keyvan Rahmani. “SugarScape on Steroids: Simulating over a Million Agents at Interactive Rates.” *Proceedings of Agent2007 Conference. Chicago, IL*. N.p., 2007.
15. Erra, Ugo et al. “An Efficient GPU Implementation for Large Scale Individual-Based Simulation of Collective Behavior.” *High Performance Computational Systems Biology, 2009. HIBI’09. International Workshop on*. IEEE, 2009. 51–58.
16. Hermellin, Emmanuel, and Fabien Michel. “Gpu Delegation: Toward a Generic Approach for Developping Mabs Using Gpu Programming.” *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2016. 1249–1258.
17. Hermellin, Emmanuel, and Fabien Michel. “Overview of Case Studies on Adapting MABS Models to GPU Programming.” *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Springer, 2016. 125–136.
18. Hicham, F. et al. “A New Model for Programming Distributed Computer Based on GPU Chip and Mobile Agent.” *2016 11th International Conference on Intelligent Systems: Theories and Applications (SITA)*. N.p., 2016. 1–6.
19. Hidayat, R. et al. “Multi-Agent System with Multiple Group Modelling for Bird Flocking on GPU.” *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. N.p., 2016. 680–685.
20. Husselmann, A. V., and K. A. Hawick. “Spatial Data Structures, Sorting and Gpu Parallelism for Situated-Agent Simulation and Visualisation.” *Proceedings of the International Conference on Modeling, Simulation and Visualization Methods (MSV)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012. 1.
21. Husselmann, Alwyn V., and Ken A. Hawick. “Simulating Species Interactions and Complex Emergence in Multiple Flocks of Boids with Gpus.” *Proc. IASTED*

*International Conference on Parallel and Distributed Computing and Systems (PDCS 2011)*. N.p., 2011. 100–107.

22. Jacobsen, Dana, Julien Thibault, and Inanc Senocak. "An MPI-CUDA Implementation for Massively Parallel Incompressible Flow Computations on Multi-GPU Clusters." *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*. N.p., 2010. 522.
23. Kleiner, Alexander et al. "Rmasbench: Benchmarking Dynamic Multi-Agent Coordination in Urban Search and Rescue." *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013. 1195–1196.
24. Kumar, Jitendra, Lotika Singh, and Sandeep Paul. "GPU Based Parallel Cooperative Particle Swarm Optimization Using C-CUDA: A Case Study." *Fuzzy Systems (FUZZ), 2013 IEEE International Conference on*. IEEE, 2013. 1–8.
25. Li, D. et al. "A Efficient Algorithm for Molecular Dynamics Simulation on Hybrid CPU-GPU Computing Platforms." *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. N.p., 2016. 1357–1363.
26. Li, Xiaosong, Wentong Cai, and Stephen John Turner. "Supporting Efficient Execution of Continuous Space Agent-Based Simulation on GPU." *Concurrency and Computation: Practice and Experience* 28.12 (2016): 3313–3332.
27. Lysenko, Mikola, Roshan M. D'Souza, and others. "A Framework for Megascale Agent Based Model Simulations on Graphics Processing Units." *Journal of Artificial Societies and Social Simulation* 11.4 (2008): 10.
28. Michel, Fabien. "Translating Agent Perception Computations into Environmental Processes in Multi-Agent-Based Simulations: A Means for Integrating Graphics Processing Unit Programming within Usual Agent-Based Simulation Platforms." *Systems Research and Behavioral Science* 30.6 (2013): 703–715.
29. Passos, E. et al. "Supermassive Crowd Simulation on Gpu Based on Emergent Behavior." *Proceedings of the Seventh Brazilian Symposium on Computer Games and Digital Entertainment*. Citeseer, 2008. 70–75.
30. Passos, Erick Baptista et al. "A Bidimensional Data Structure and Spatial Optimization for Supermassive Crowd Simulation on GPU." *Computers in Entertainment (CIE)* 7.4 (2009): 60.

31. Pavlov, Roman, and Jörg P. Müller. “Multi-Agent Systems Meet GPU: Deploying Agent-Based Architectures on Graphics Processors.” *Doctoral Conference on Computing, Electrical and Industrial Systems*. Springer, 2013. 115–122.
32. Perumalla, Kalyan S., and Brandon G. Aaby. “Data Parallel Execution Challenges and Runtime Performance of Agent Simulations on GPUs.” *Proceedings of the 2008 Spring Simulation Multiconference*. Society for Computer Simulation International, 2008. 116–123.
33. Richmond, Paul et al. “High Performance Cellular Level Agent-Based Simulation with FLAME for the GPU.” *Briefings in bioinformatics* 11.3 (2010): 334–347.
34. Richmond, Paul, Simon Coakley, and Daniela Romano. “Cellular Level Agent Based Modelling on the Graphics Processing Unit.” *High Performance Computational Systems Biology, 2009. HIBI’09. International Workshop on*. IEEE, 2009. 43–50.
35. Richmond, Paul, Simon Coakley, and Daniela M. Romano. “A High Performance Agent Based Modelling Framework on Graphics Card Hardware with CUDA.” *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2009. 1125–1126.
36. Richmond, Paul, and Daniela Romano. “Agent Based Gpu, a Real-Time 3d Simulation and Interactive Visualisation Framework for Massive Agent Based Modelling on the Gpu.” *Proceedings International Workshop on Supervisualisation*. N.p., 2008.
37. Sano, Yoshihito, Yoshiaki Kadono, and Naoki Fukuta. “A Performance Optimization Support Framework for Gpu-Based Traffic Simulations with Negotiating Agents.” *Recent Advances in Agent-Based Complex Automated Negotiation*. Springer, 2016. 141–156.
38. Shen, Zhen, Kai Wang, and Fenghua Zhu. “Agent-Based Traffic Simulation and Traffic Signal Timing Optimization with GPU.” *Intelligent Transportation Systems (Itsc), 2011 14th International Ieee Conference on*. IEEE, 2011. 145–150.
39. Spicher, Antoine, Nazim Fates, and Olivier Simonin. “From Reactive Multi-Agent Models to Cellular Automata-Illustration on a Diffusion-Limited Aggregation Model.” *1st International Conference on Agents and Artificial Intelligence*. N.p., 2009.
40. Strippgen, David, and Kai Nagel. “Multi-Agent Traffic Simulation with CUDA.” *High Performance Computing & Simulation, 2009. HPCS’09. International Conference on*. IEEE, 2009. 106–114.

41. Strippgen, David, and Kai Nagel. “Using Common Graphics Hardware for Multi-Agent Traffic Simulation with CUDA.” *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009. 62.
42. Torchelsen, Rafael P. et al. “Real-Time Multi-Agent Path Planning on Arbitrary Surfaces.” *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2010. 47–54.
43. Vigueras, Guillermo, Juan M. Orduna, and Miguel Lozano. “A GPU-Based Multi-Agent System for Real-Time Simulations.” *Advances in Practical Applications of Agents and Multiagent Systems*. Springer, 2010. 15–24.
44. Wang, Kai, and Zhen Shen. “A GPU Based Trafficparallel Simulation Module of Artificial Transportation Systems.” *Service Operations and Logistics, and Informatics (SOLI), 2012 IEEE International Conference on*. IEEE, 2012. 160–165.
45. Laville, Guillaume et al. “MCMAS: A Toolkit to Benefit from Many-Core Architecure in Agent-Based Simulation.” *Euro-Par 2013: Parallel Processing Workshops*. Springer, Berlin, Heidelberg, 2013. 544–554.
46. NVIDIA. NVIDIA CUDA programming guide. Technical Report, 2013. (Available from: http://docs.nvidia.com/cuda/cuda-c-programming-guide)