Implementing Multi-Agents Spatial Simulation (MASS) Library on Graphics Processing Units

Term report – summer 2012

1 Introduction

Multi-agent spatial simulation (MASS) is a parallel library (Emau, Chuang, & Fukuda, 2011) that improves the performance of compute-intensive functions when objects in the library are extended. A previous implementation of the MASS library using Java running on multiple CPU cores showed improvements on data parallel algorithms but degraded on larger arrays. A new approach is presented in this research work by optimizing the algorithm used to perform communication and synchronization among the data elements and implementing the library on the graphics processing units (GPUs). The Compute Unified Device Architecture (CUDA) (Sanders & Kandrot, 2011) programming language (a GPU API developed by NVIDIA) is used to implement the MASS library on a *manycore* single GPU. This research study implements the Schrodinger wave simulation as a multi-agents simulation and a performance of up to 80% is obtained over the *multi-core* CPU implementation of this wave simulation.

2 Experimental setup

Operating system: Red Hat Enterprise Client Linux release 5.8 (Tikanga)

CPU Core: Intel® Xeon® CPU E5520 @ 2.27GHz

Number of CPU Cores: 16

A wave simulation program based on the Schrödinger wave equation (Starr, 1940) was programmed in C++, MASS-C++, CUDA, and MASS-CUDA. Each version of the wave simulation is run 10 times, and an average of 8 runs is taken after discarding the execution times of the highest and lowest value.

2.1 Wave2d CPU

The simulation version of the program is parallelized using OpenMP to execute from 1 up to 16 parallel threads. This is repeated for both the C++ and MASS-C++ version of the wave simulation.

2.2 Wave2d GPU

The simulation version of the program is parallelized using CUDA to execute up to 256 blocks with up to 256 threads in each block. The overall experiment is run by executing increments of 16 blocks with

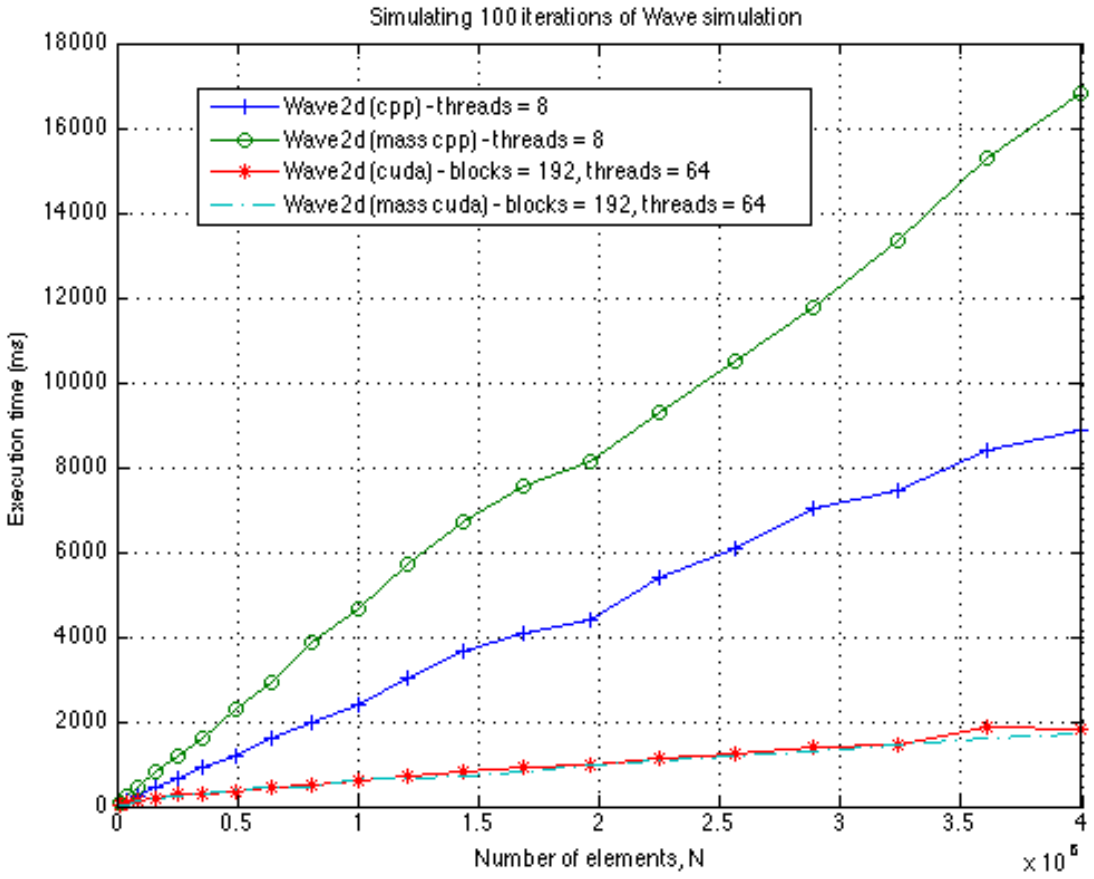
corresponding increments of 16 threads up to 256 threads and 256 blocks. However, the best performing thread/block combination of 192 blocks and 64 threads per block is used for the results.

3 Results

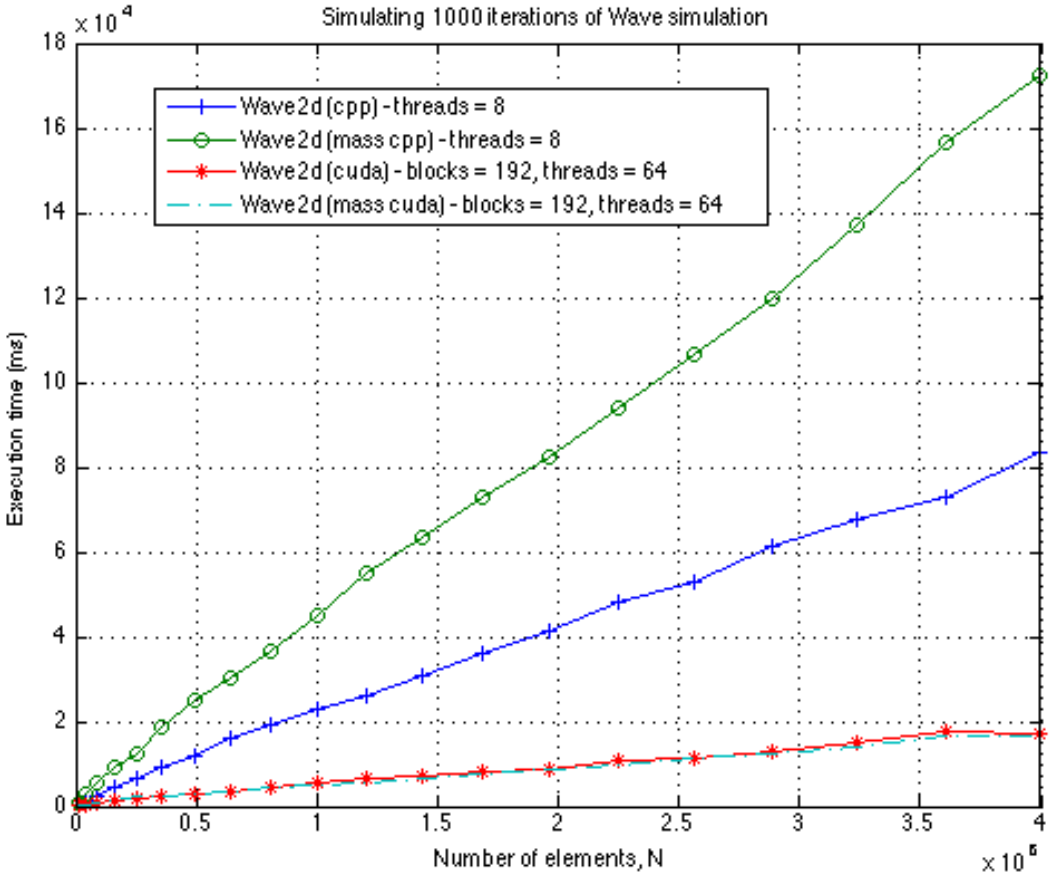
3.1 Table 1: Performance improvement of MASS-CUDA over MASS-C++

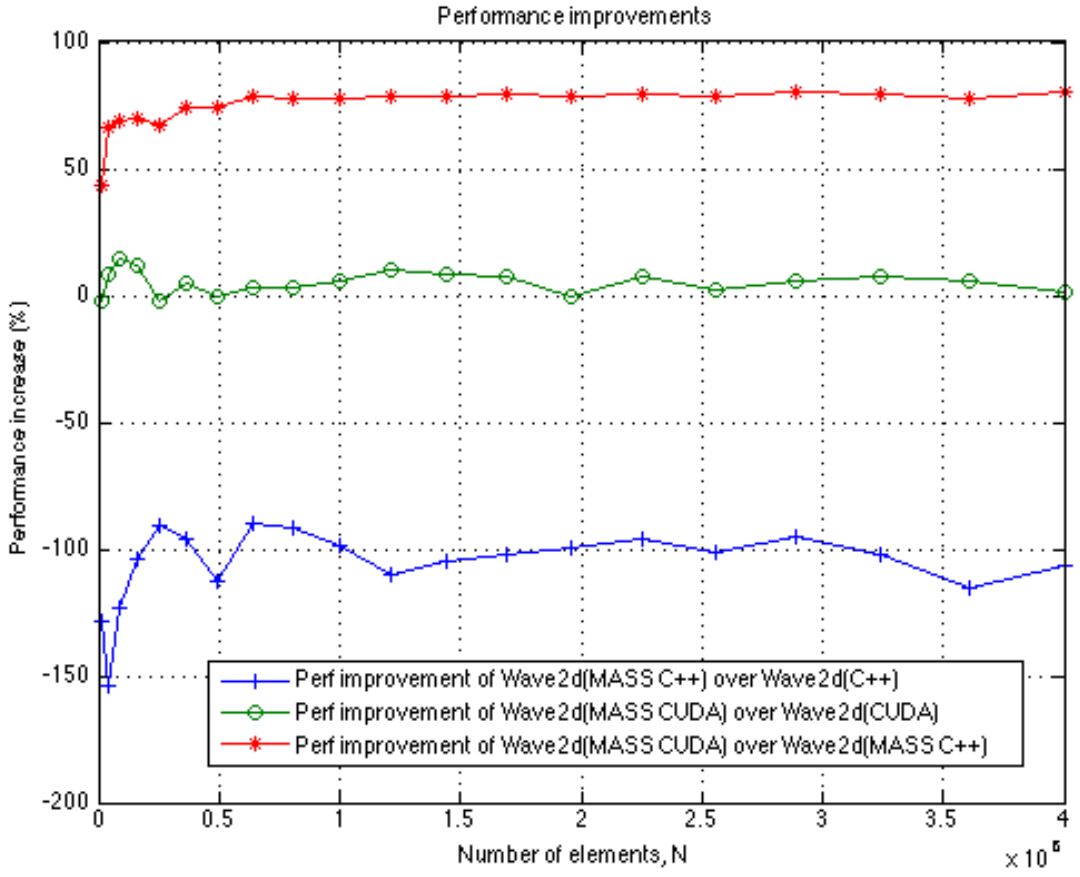
size	wave2d_cpp 8 thr (ms)	wave2d_mass_cpp 8 thr (ms)	wave2d_cuda 192 64 (ms)	wave2d_mass_cuda 192 64 (ms)	mass_cpp vs cpp (%)	mass_cuda vs cuda (%)	mass_cuda vs mass_cpp (%)
10000	295.716	675.54	164.612	168.02	-128.44	-2.07	43.18
40000	1218.17	3095.19	451.571	415.514	-154.09	7.98	65.89
90000	2541.58	5674.73	925.614	789.667	-123.28	14.69	68.93
160000	4422.06	9024.16	1526.39	1350.53	-104.07	11.52	69.46
250000	6553.46	12481.3	2097.79	2143.42	-90.45	-2.18	67.29
360000	9441.41	18540	2604.39	2479.75	-96.37	4.79	73.74
490000	11730.7	24941.4	3054.36	3068.4	-112.62	-0.46	73.84
640000	15854.9	30082.4	3581.71	3460.73	-89.74	3.38	78.17
810000	19014.2	36475.5	4438.73	4293.8	-91.83	3.27	77.42
1000000	22763.5	45204.4	5409.2	5078.75	-98.58	6.11	77.69
1210000	26146.1	54852.1	6341.4	5720.3	-109.79	9.79	78.12
1440000	30996.4	63529.3	7194.86	6608.75	-104.96	8.15	78.68
1690000	36231.7	73113.5	8199.62	7592.85	-101.79	7.40	79.04
1960000	41360.5	82375.1	8881.61	8916.5	-99.16	-0.39	78.44
2250000	48012.8	94038.5	10604.3	9802.38	-95.86	7.56	79.58
2560000	52820.2	106346	11413.7	11188.4	-101.34	1.97	78.82
2890000	61450.6	119909	13031.4	12339	-95.13	5.31	79.92
3240000	67637.9	136987	15020.3	13860.1	-102.53	7.72	79.51
3610000	72701.2	156648	17435.6	16404.1	-115.47	5.92	77.44
4000000	83412.1	172124	16916.5	16661.4	-106.35	1.51	80.03

3.2 Running wave2d simulation on the CPU and GPU using 100 iterations



3.3 Running wave2d simulation on the CPU and GPU using 1000 iterations





4 Performance evaluation

From the results in the previous section, it can be seen that there is up to an 80% performance increase when comparing the execution of Wave2d (MASS-C++ version) to Wave2d (MASS-CUDA version). This represents the raw performance of running 1000 iterations of the wave simulation program and running 10 instances of each, discarding the highest and lowest execution time and getting the average of 8 values.

Running the Wave2d simulation on the CPU by using the MASS-C++ version of the MASS library resulted performance hits because of the overhead of using this parallel library. The Wave2d simulation program written in C++ is also parallelized so the performance that is seen here is a comparison of parallelized Wave2d in C++ versus parallelized Wave2d in MASS-C++. Here the calls to the MASS library all take place on the CPU.

Comparing Wave2d (MASS-CUDA) with Wave2d (CUDA), there is no performance hit of using the MASS library functions as can be observed in the MASS-C++ versus C++ version of the wave simulation. This is good news because it shows in this instance that MASS causes no overhead when

the MASS library functions are strategically placed in a client code that runs a scientific application like this.

5 Conclusions

Implementing MASS on the GPU has proven to have some positive results just as was expected with up to 80% improvement of running the simulation program on MASS-CUDA (GPU) versus MASS-C++ (CPU only). Another outcome of the experiments performed is the overhead that MASS-C++ causes. It is tempting to add a comparison for a single threaded run of the wave simulation program implemented in C++, and compare directly with the inherently multithreaded version implemented in MASS-C++. However, the need for parallelism is not the subject of this research; instead this work focuses on how to improve the implementation of a parallel application in order to benefit from performance gains by running on the GPU.

6 Future work

This research effort focused on showing the performance benefit of implementing the MASS library by complementing the CPU with the GPU. In future, this work would be extended to implementation on multiple GPUs and other parallel algorithms that might improve the performance of the MASS library as a whole. Another future work item would be to implement a source to source translator (Cooper & Torczon, 2012) that would completely eliminate the need for the user of the MASS library to implement any GPU functions. For example, the user would implement the compute intensive functions (e.g. compute_zt()), while the library would take care of generating the appropriate GPU-specific source code that would perform its execution on the GPU.

References

Cooper, K. D., & Torczon, L. (2012). *Engineering a compiler* (2nd ed.). Amsterdam ; London: Elsevier/Morgan Kaufmann.

Emau, J., Chuang, T., & Fukuda, M. (2011). A multi-process library for multi-agent and spatial simulation. Paper presented at the *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on,* 369-375.

Sanders, J., & Kandrot, E. (2011). *CUDA by example: An introduction to general-purpose GPU programming*. Upper Saddle River ; London: Addison-Wesley.

Starr, D. W. (1940). *The schrödinger wave equation from the point of view of singular integral equations*.

Urbana, Ill.: