OTC 27386

Accelerating Numerical Ice Engineering Tools Using GPGPU

Shadi Alawneh, Oakland University;
Jan Willem Thijssen, C-CORE;
Martin Richard, C-CORE, Memorial University of Newfoundland;

This paper was prepared for presentation at the Arctic Technology Conference held in St. John's, Newfoundland and Labrador, 24-26 October 2016.

Abstract

C-CORE is engaged in understanding the iceberg and sea ice design loads needs of the energy sector. As the energy industry ventures into oceans with greater ice cover and more icebergs, there is a significant need for efficient engineering tools to plan and manage operations in exploration, production, and safety. Industry requires a range of scenarios for their risk assessments, where existing simulations can be computationally and time intensive.

C-CORE has recently started using the benefit of the General Purpose Computing on Graphical Processing Units (GPGPU) approach. This approach has shown significant speed up of several numerical ice engineering applications related to icebergs and sea ice. The investigated model types are Monte-Carlo type approaches for probabilistic design method, and quadratic discriminant. GPU computing with Compute Unified Device Architecture (CUDA) is a new approach to solve complex problems and transform the GPU into a massively parallel processor.

The present study applies the GPGPU technology to a Monte-Carlo simulation, used for a sea ice load application. The objective of this study is to measure the performance of the GPU using CUDA, and compare against the serial Central Processing Unit (CPU) using C++ and MATLAB implementations. Results show a speedup of up to 2,600 times of the GPGPU implementation compared to the MATLAB implementation, reducing the elapsed time from about 1.5 hour to about 2 seconds. This strongly indicates that the GPGPU approach can help the industry to significantly reduce the time required for the simulations.

Keywords: GPGPU; CUDA; Ice Loads; Ice-Structure Interactions, Monte-Carlo;

1. Introduction

"Commodity computer graphics chips, known generically as Graphics Processing Units or GPUs, are probably today's most powerful computational hardware for the dollar. Researchers and developers have become interested in harnessing this power for general-purpose computing, an effort known collectively as GPGPU (for General-Purpose computing on the GPU)." (Owens et.al 2007). GPUs are an attractive option for many numerical problems where the simulation time is a bottleneck. Significant computational power is provided at a relatively low cost, and this power/cost ratio is increasing faster than for traditional CPUs.

GPUs have a large number of high-performance processors that are able to perform high computation and data throughput. It is only recently that GPUs have support for accessible programming interfaces and industry-standard languages such as C. Hence, these GPUs have the ability to perform more than the specific graphics computations for which they were originally designed. Developers who use GPUs to implement their numerical applications often achieve speedups that are orders of magnitude when compared to optimized CPU implementations.

There are several advantages of GPGPU that makes it particularly attractive. GPUs are developed with dedicated chips to aid high performance computing in parallel. Recent graphics architectures provide significant memory bandwidth and computational horsepower. Also, the performance of the graphics hardware increases more rapidly than that of CPUs because

of semiconductor capability, driven by advances in fabrication technology, increases at the same rate for both platforms. Recently, improvements in the programmability and computing power of GPUs accelerated the development of code related to applications for which GPUs could be used. Significant work has been done to accelerate the speedup of GPUs over CPU implementations. The reason for this advantage is the bandwidth and computational horsepower of recent GPU architectures. This development in GPU computation has enabled the simulating of massive event set in timely and practical way.

C-CORE is engaged in understanding the iceberg and sea ice design loads needs of the energy sector and has several applications related to icebergs and sea ice including the sea ice loads software (SILS) (Thijssen et al., 2014), iceberg load software (ILS) (Fuglem et al., 2014) and models for sea ice forecasting (Howell et al., 2015). SILS is used to determine design loads for fixed offshore structures encountering sea ice. The ILS software is used to determine design iceberg impact loads and accounts for the effectiveness of iceberg detection, management and disconnection (when applicable) that reduce the likelihood of iceberg impact. Local ice pressures associated with the design loads are determined in ILS based on the encounter frequency and the duration of collisions. SILS and ILS are fully probabilistic, considering a large number of loading events based on the expected range of ice and environmental conditions as well as contact locations and ice pressures. To determine design ice loads, loads associated with each interaction event are calculated and ranked on an annual maximum basis. Both algorithms employ a Monte-Carlo simulation approach, as it provides an accurate representation of specific loading situations, in the spirit of codes such as ISO 19906 (ISO 19906:2010).

The above applications are computationally intensive and a GPGPU-based approach is recently applied to accelerate the performance. A paper has been published to show the performance benefits of GPGPU for sea ice forecasting by using fast quadratic discriminant analysis (Alawneh et.al 2015a). Previous research has also been done to demonstrate the performance benefit of GPGPUs for simulating the complex mechanics of a ship operating in pack ice, wherein it was shown that using GPUs can result in significantly reduced computational time (Alawneh et.al 2015b).

The work reported in this paper is an extention of the recent work of (Ayubian et.al 2016). The main goal of this project is to initiate the development of sea ice applications using the GPGPU technology. The value and novelty of the project is to use the benefit of the high performance of the GPU to re-develop the algorithms in these applications and increase the efficiency and speed at which these applications can run. The present study demonstrates how to achieve high performance computing solutions by utilizing GPU to implement a Monte-Carlo simulation for extreme level sea ice loads. The objective of this paper is to analyze the performance of a Monte-Carlo type experiment coded in the CUDA environment (i.e., using the GPU) and compare the performance results with regard to different CPU implementations. We evaluated the performance of the GPU and Multi-GPU against the serial Central Processing Unit (CPU), parallel CPU (OpenMP), MATLAB and MATLAB (Parallel for) implementations.

2. Methodology

2.1 Scenario Description and Monte-Carlo Framework

Ice interactions can result in significant forces on offshore structures potentially causing damage to the structure. Therefore, ice load analysis is an important aspect for designing offshore structures exposed to ice conditions. The current work uses a scenario that models level ice and ridges, and includes considerations of a kinetic energy limit. Once the floe hits the structure, the initial kinetic energy is now diminished by the energy dissipated in crushing. At the same time, energy is added through driving force (FY ice rubbling, wind and current). The purpose of this is that in some cases the floe will not crush far enough into the structure to reach the full contact width, or contact the ridge that is located somewhere inside the floe (assumed perpendicular to the flow direction). This has a load-reducing effect. To do this we determine after each meter of crushing what the remaining kinetic energy is of the floe, starting with the initial kinetic energy minus the dissipated crushing energy plus the floe driving forces. Once the kinetic energy is depleted, there will be no further crushing of this floe. The maximum load is determined for the part of the floe that is crushed, and likely to originate from the thickest or widest part of the floe/ridge. The global average crushing pressure p is calculated as follows (ISO 19906):

$$p = C_R \left(\frac{h}{h^*}\right)^n \left(\frac{W_s}{h}\right)^m$$

in which p is the global average ice pressure (MPa), h is the ice thickness, W_s is the contact width (m), m and n are empirical coefficients ($m = -0.16$ and $n = -0.5 + 0.2h$ or $n = -0.3$, depending whether h is less than 1 m or not, respectively), C_R is an ice strength coefficient (MPa), and h^* is a coefficient is equal to 1 (m) to keep units consistent.

Many numerical problems in science, engineering, finance and statistics are modeled through Monte-Carlo simulations. The study of Monte-Carlo techniques requires knowledge in a wide range of fields, for instance probabilities to describe random processes, statistics to analyze the data, computational science to efficiently implement the algorithms, and mathematical

programming to formulate and solve a problem of interest (Kroese et.al 2014). Monte-Carlo simulation is a very robust and relatively simple method to implement. However, using Monte-Carlo simulation can require long execution times (Veach 1997).

The experiment performed as part of this project is configured to facilitate the use of Monte-Carlo simulation to determine the design load using the following steps (as also shown in Figure 1):

- Specify an interaction model corresponding to level sea ice and ridges;
- Define probability distributions for the environmental inputs;
- Determine the maximum annual force on the structure;
- Perform this experiment for a sufficient number of years to achieve stable results; and
- Rank the annual maximum forces.

Based on Figure 1, the calculations of the ice loads on the structure are independent for each year. Therefore, one thread can be assigned for each year to calculcate the ice load on the structure.

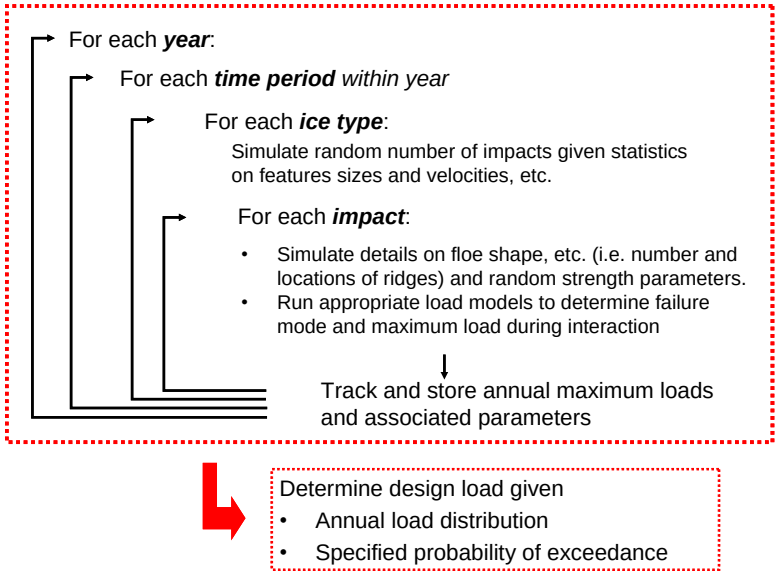

Figure 1. Monte-Carlo simulation framework

2.2 **Implementation in CUDA**

In November 2006, NVIDIA had an opportunity to bring GPUs into the mainstream by bringing a programming interface, which it dubbed CUDA. It was an attempt to make the programming environment on GPUs more accessible to programmers. CUDA interface uses standard C language to implement an algorithm on GPU without having any knowledge about graphics programming using OpenGL, DirectX, and shading language. CUDA has produced significant progress in the computer software industry by moving from serial to parallel programming. It can take a simple model of data parallelism into a programming model.

CUDA contains some libraries, which are not different than system libraries or user-built libraries. They can refer to a set of functions' definitions whose signatures are exposed through header files. The CUDA libraries are special in that all computation implemented in the library is accelerated using a GPU, as opposed to a CPU. Shared features exist and concepts in many CUDA libraries which can be called from a host application. This scenario demonstrated the application of the two most important libraries in CUDA (Nvidia 2008).

One of the critical parts of the Monte-Carlo application used in this project is random number generation. CUDA provides a library, *cuRAND*, which can focus on the efficient generation of high quality pseudo-random and quasi-random numbers. *cuRAND* is equipped by library on the host (CPU) side and a device (GPU) header file. These random numbers can be generated on the device or on the host CPU. In this work, the Mersenne Twister algorithm (Matsumoto and Nishimura, 1998) is used.

For device generation of random numbers, the actual work occurs on the device and the result is stored in global memory on the device. The user can copy random numbers back to the host (CPU) for further processing or call their own kernels (parallel functions) to use the random numbers. However, for host CPU generation, all of the work is done on the host, and the random numbers would be stored in host memory (Nvidia, C. Curand library 2010).

The next library, *Thrust*, is a powerful library of parallel algorithms and data structures. It has a leverage to implement high performance application with minimal programming effort. *Thrust* provides a flexible, high-level interface for GPU programming that enhances developer productivity and the robustness of CUDA applications. Using *Thrust*, the programmer can write just a few lines of code to perform operations faster than the latest multi-core CPUs (Hoberock et al., 2010).

CUDA libraries and intensive GPU-accelerated applications are available from NVIDIA and the use of those libraries is key to obtain significant improvement in speedupThe goal of this study is to measure the performance of the GPU using CUDA against the serial Central Processing Unit (CPU) using C++ and MATLAB implementations. Fixed and distributed parameters are defined in order to calculate the ice forces in each year, using the algorithm described above. The *cuRAND* library is used on the CUDA environment to generate random numbers based on those probabilistic distributions and using Monte-Carlo simulation for interaction between the sea ice and the structure. Design loads (annual maximums) were then generated for different numbers of years (between 10,000 to 50,000 years, by increment of 10,000) to evaluate the speedup of the CUDA implementation.

3. Performance Results

We have used an Intel(R) Xeon(R) CPU E5-2620 processor @2.10GHz and a GPU GeForce GTX TITAN Black card. This card has 2880 processor cores, 889 MHz processor clock and 336 GB/sec memory bandwidth. This is a consumer grade card that was listed around $600-700 USD as of August 2016.

Figure 3 shows the elapsed time of the C++ (CPU), MATLAB (CPU) and CUDA (GPU) implementations, normalized over the time required to perform a 10,000 years simulation. It is observed that the time required to perform a simulation increases linearly and on a proportion of a 1:1 (i.e., twice as many years will result with twice the run time, and so on) with the number of years simulated for both CPU implementations; on the other hand, there seem to be less overhead when using the GPU implementation as performing 5 times more years of simulation resulting in less than 5 times more time (closer to 4 times more time). Table 1 shows the absolute time required to perform each simulation as well as the speedup of the CUDA (GPU) implementation over the two CPU implementations. It can be observed that the elapsed time to run a full Monte-Carlo simulation for 50,000 years is reduced from about 1.5 hours (MATLAB serial CPU implementation) to approximately 2 seconds when the GPUs are used. The elapsed time is also reduced from about 10 minutes to approximately 2 seconds for the 50,000 years when we use the CUDA (GPU) implementation instead of the C++ (CPU) implementation. The CUDA (GPU) implementation is the fastest implementation for the ice interaction scenario considered in this paper.

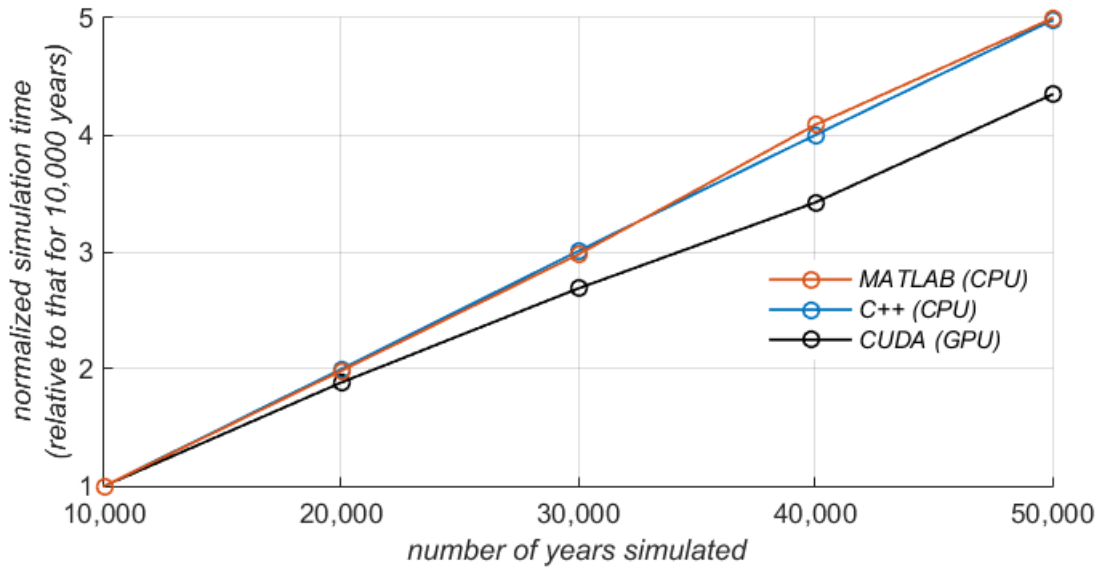

Figure 2. Normalized time required for simulations relative to that for 10,000 years simulations for different Monte-Carlo implementations

Table 1. Speedup and elapsed time of the implementions in MATLAB (CPU), C++ (CPU) and CUDA (GPU)

Number of Simulation Years	Time elapsed for simulation (s)			Speedup	
	MATLAB (CPU)	C++ (CPU)	CUDA (GPU)	CUDA (GPU) over C++ (CPU)	CUDA (GPU) over MATLAB (CPU)
10,000	1,100	120	0.46	250	2,300
20,000	2,100	230	0.87	270	2,400
30,000	3,200	350	1.30	280	2,500
40,000	4,300	470	1.60	300	2,700
50,000	5,300	580	2.00	290	2,600

4. Conclusions

In this study, GPU performance benefits are discussed for a Monte-Carlo simulation of a design ice load scenario. Specific ice parameters were selected for demonstrative purposes, which allow the engineering designers to have a better understanding of those factors. The speedups attainable with different types of GPUs were shown to be significant. It should be mentioned that while we have used CUDA to implement the parallel algorithm, the results are not specific to this method or to GPUs. There exist many core processor markets with different devices and architectures, which take advantage of this improvement.

5. Future Work

The present study focused on a scenario that models level ice and ridges, and includes considerations of a kinetic energy limit, but there exist many other more complex scenarios that worth investigating. Also, GPGPU implementations are well suited and could be used for other types of application relevant to the ice engineering world, such as modelling pack ice using the Discrete Element Method (DEM), and modelling hydrodynamics (for iceberg motions, for example) using the Smoothed Particle Hydrodynamics (SPH) method.

In this experiment, memory limited us to work on more data, hence finding a way to manage the memory would be helpful. Sometimes, the operating system on the co-processor allows us to allocate more memory space than is physical available. Typically, it is time consuming to start code from scratch to optimize the existing implementation. Potentially another processor can be used that enables significant performance gains for highly parallel code, for instance the Intel Xeon Phi coprocessor. This type of coprocessor is optimized to be the right choice for highly parallel workloads.

6. References

Owens, J. D., D. Luebke, N. Govindaraju, M. Harris, J. Krger, A. Lefohn and T. J. Purcell. 2007. “A survey of general-purpose computation on graphics hardware.” Computer Graphics Forum, vol. 26, no. 1, pp. 80–113.

Thijssen, J.W., M. Fuglem, M. Richard, T. King. 2014. “Implementation of ISO 19906 for Probabilistic Assessment of Global Sea Ice Loads on Offshore Structures Encountering First-Year Sea Ice.” In Proceedings of OCEANS ’14 Conference. St. John’s, NL, Canada.

Fuglem, M., M. Richard and T. King. 2014. “An Implementation of ISO 19906 Formulae for Ice Loads on Fixed Structures within a Probabilistic Framework using the Sea Ice Loads Software.” In proceedings of Arctic Technology Conference. Houston, Texas.

Howell, C., M. Richard, J. Barnes and T. King. 2015. “Short-term Operational Sea Ice Forecasting for Arctic Shipping.” In Proceedings of the 34th International Conference on Ocean, Offshore and Arctic Engineering (OMAE 2015). St. John’s, NL, Canada.

ISO. 2010. “ISO 19906, Petroleum and Natural Gas Industries - Arctic Offshore Structures.” International Standard.

Alawneh, S., C. Howell, and M. Richard. 2015a. “Fast quadratic discriminant analysis using gpgpu for sea ice forecasting.” In Proceedings of High Performance Computing and Communications (HPCC) Conference. New York, USA.

Alawneh, S., R. Dragt, D. Peters, C. Daley, and S. Bruneau. 2015b. “Hyper-real-time ice simulation and modeling using gpgpu.” IEEE Transactions on Computers. 64, 12 (2015), 3475–3487.

Ayubian, S., S. Alawneh, and J. Thijssen. 2016. “GPU-Based Monte-Carlo Simulation for a Sea Ice Load Application.” In Proceedings of the 2016 Summer Computer Simulation Conference (SCSC 2016). Montreal, Quebec, Canada.

Thijssen, J., M. Fuglem, K. Muggeridge, T. Morrison, P. Spencer, and others. 2015. “Update on probabilistic assessment of multi-year sea ice loads on vertical-faced structures.” In Proceedings of OTC Arctic Technology Conference, Offshore Technology Conference (2015).

Kroese, D. P., T. Brereton, T. Taimre, and Z.I. Botev. 2014. “Why the Monte Carlo method is so important today.” Wiley Interdisciplinary Reviews: Computational Statistics 6.

Veach, E. 1997. “Robust Monte Carlo methods for light transport simulation.” PhD thesis, Stanford University.

Nvidia. 2008. “C. Programming guide.” Santa Clara, CA

Nvidia. 2010. “C. Curand library. NVIDIA Corporation.” Santa Clara, CA.

Hoberock, J., and N. Bell. 2010. “Thrust: a parallel template library.” Version 1.3.

Matsumoto, M., and T. Nishimura. 1998. “Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator.” ACM Trans. on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3-30.