# Implementation and Performance of a GPU-Based Monte-Carlo Framework for Determining Design Ice Load

Sara Ayubian
Computer Science Department
Memorial University of Newfoundland, C-CORE
St. John's, NL, Canada
Email: sa7818@mun.ca

Shadi Alawneh
Electrical and Computer Department
Oakland University
Rochester, MI, USA
Email: shadialawneh@oakland.edu

Martin Richard
Civil Engineering Department
Memorial University of Newfoundland
St. John's, NL, Canada
Email: martin.richard@c-core.ca

Jan Thijssen
C-Core
St. John's, NL, Canada
Email: jan.thijssen@c-core.ca

***Abstract*—Modern Graphics Processing Units (GPUs) with massive number of threads and many-core architecture support both graphics and general purpose computing. NVIDIA's compute unified device architecture (CUDA) takes advantage of parallel computing and utilizes the tremendous power of GPUs. The present study demonstrates a high performance computing (HPC) framework for a Monte-Carlo simulation to determine design sea ice loads which is implemented in both GPU and CPU. Results show a speedup of up to 130 times for the 4 Tesla K80 GPUs over an optimized CPU OpenMP implementation and speedup of up to 8 times for the 4 Tesla K80 over a single Tesla K80 GPU implementation. The elapsed time of the different implementations has been reduced from about 2.5 hours to 0.7 seconds.**

***Keywords-GPGPU; CUDA; Monte-Carlo.***

## I. INTRODUCTION

General purpose graphics processing unit (GPGPU) is one of the best methodologies to introduce high performance computing (HPC). A computer can be utilized with GPUs to execute massive concurrent computations, and to achieve efficient implementation [1]. A GPU is made of thousands of cores which are responsible for handling many tasks concurrently, while a CPU has a few cores for the serial part of the program [2].

In the present study describes, the interaction between sea ice and vertically sided structures are simulated, with the goal of finding the maximum annual force over 10,000 years while considering wind, current and kinetic energy in the calculation. Each force involves different random parameters that require a random number generator (RNG) algorithm to calculate their appropriate distributions. Therefore, using an efficient algorithm, such as Monte Carlo, for generating random numbers helps to simulate the complex sea ice load scenario. This experiment was simulated over 1,000,000 years rather than 10,000 years, in order to achieve a stable result. As this experiment is difficult and time consuming, it would be effective to use a parallel environment such as, compute unified device architecture (CUDA) interface programming on Graphic processing unit (GPU), and to calculate the speedup over central processing unit (CPU) implementations.

The goal is to analyze the performance results between GPUs and CPUs when both are used for the processing of the same algorithm, and speedup interpretation of the optimized implementations.

## II. RELATED WORKS

### A. GPU Vs. CPU

One of the differences between a GPU and a CPU is the way they process tasks. For instance, a CPU with fewer cores is responsible for the sequential part of the program, while a GPU consists of thousands of smaller cores designed for performing multiple tasks concurrently [16]. In order to compare the performance of a GPU and a CPU, one needs to come up with a ratio illustrating which implementation is faster. Therefore, measuring the elapsed time of the implementation helps to evaluate the performance of an implemented algorithm. For example, speedup is a ratio that can be calculated by dividing the elapsed time of a parallel algorithm over the sequential algorithm [17]. When parallel computing involves large-scale data, having the highest speedup becomes increasingly important for scientific computations [18]. Speedup of a parallel computation is defined as $\frac{T_s}{Tp}$, where $T_s$ is the sequential time and $Tp$ is the parallel time to solve the problem using $p$ processors [19]. For instance, an article related to the GPU accelerated Monte

978-1-5386-3250-5/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCS.2017.27

Carlo simulation of Brownian motors dynamics with CUDA had a speedup of about 3,000 compared to that if a typical CPU. Furthermore, there is no optimization in this study for comparing a GPU efficient code against a CPU optimized implementation [20]. There are many studies that have been done to accelerate the speedup of GPUs over CPUs. Recently, an article reported a speedup of up to 426 times over a MATLAB (matrix laboratory) implementation of quadratic discriminant analysis by using CUDA application using a GPU. The authors also compared the performance of GPU against the optimized CPU and achieved a speedup of up to 23 times [14]. Performance analysis has been done on the Finite-Difference Time-Domain (FDTD) method on a GPU, and a speedup of up to 64 times have been achieved. In this study, theoretical prediction of high performance agreed with the experimental result, which suggested a suitable optimization method for the best performance. This indicates that GPGPU has potential for improving the processing time of highly parallel algorithms [21]. For example, research has been done to demonstrate the performance benefit of the GPGPUs for simulating a ship operating in pack ice, and found that GPUs have the potential to reduce computational time significantly [22]. Therefore, one way to understand the high performance result of a GPU over multicore CPUs is to execute a performance analysis and apply optimization techniques for both GPUs and CPUs. It is also possible to recommend a new set of architectural attributes which improve architectural efficiency [6].

Based on a previous study, the present study will investigate whether there is a significant speedup of a standard code for the Monte Carlo simulation of sea ice load on a GPU by using CUDA programming over CPU implementations. The significant speedup that came after using different types of GPUs expands on the range of problems solvable by using probabilistic simulations [20].

### *B. Monte Carlo Simulations*

The Monte Carlo simulations, as a broad class of computational algorithms, are used in many different areas [23]. Monte Carlo simulations are used when we have some applications with uncertain inputs, and for high dimensional problems with many degrees of freedom. The stages used to improve performance in Monte Carlo simulations are fairly simple, flexible, and highly scalable, and can reduce complex and large models to a set of basic interactions which could be implemented efficiently [24]. A typical Monte Carlo simulation consist of four steps: (1) defining a domain of possible inputs; (2) generating random number; (3) performing a deterministic computation on the environmental inputs; and (4) aggregating the results. For instance, these steps apply for calculating the value of π. First, consider a circle inscribed in a unit square. Second, generate uniformly scatter dots over the square. Next, count the scatter dots inside the square and the circle. Finally, the ratio of number of dots inside the circle and the square is approximately equal to π/4 ( $\frac{Area\ of\ Circle}{Area\ of\ Square} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$ ) and multiply the result by 4 to estimate π [25].

Monte Carlo methods are also helpful for simulating real-life random system and deterministic numerical computations [26]. Mathematical optimization is one of the indisputable aspects of Operation Research and Industrial Engineering. Monte Carlo techniques have been applied for providing the optimal design, scheduling and handling of industrial systems. This method is also helpful for the direct simulation of the process of neutron transport in physical processes, and for the study of chemical kinetics by means of stochastic simulation [27][28]. Monte Carlo methods are also effective in probability theory, statistical physics and computer science for studying the properties of random structures such as the Ising model, the Potts model and classical models of ferromagnetism [29].

Randomized algorithms, which means the use of a random number generator (RNG) in an algorithm, are very important in tackling those difficult computational problems through the use of the Monte Carlo method [30]. Parallel computing and especially those that are embarrassingly parallelizable would be suitable fit for certain inherently parallelizable Monte Carlo methods. Therefore, there is relatively little work needed in order to solve the problem efficiently in the parallel Monte Carlo framework [31]. As the Monte Carlo method is simple and applicable, it continues to be one of the most useful achievements in scientific computing. Maybe the next generation of this method will bring important tools for solving more complex optimization problems in statistics, mathematics, engineering, and the physical and computer sciences.

### *C. CUDA Programming*

Compute unified device architecture (CUDA) is a parallel computing platform and programming interface that is an appropriate environment for using Monte Carlo simulation. The CUDA program have a basic flow such as initializing an array of data in the host, copying the array from the host to the CUDA device, operating on the array of data by CUDA device, and copying the array back to the host.

CUDA random number generator (CURAND) and Thrust are C++ template libraries in the CUDA toolkit. These libraries include containers (e.g.thrust::host_vector and thrust::device_vector), iterators (pointer to array elements) and algorithms (e.g. reduction, transformation, sums and sorting) are suitable for conducting Monte Carlo simulation using GPU [29]. Researchers also took advantage of the computational performance of GPUs to simulate rarefied flows involving real gas effects by internal relaxation and chemical reactions using the direct simulation Monte Carlo (DSMC). This method achieved HPC which partially alleviated the main limitation of long computational run-times [32].

Another application of GPU-based Monte Carlo simulation in CUDA programming evaluated light-skin diffuse reflectance spectra for Multi-Layered Media. The speedup for this case was 71.19 times and varied across the wavelengths [33].

Another study implemented the Dose Planning Method (DPM) Monte Carlo calculation package in CUDA on a Tesla

C1060 GPU, and reached a speedup of up to 6 times against a 2.27GHz Xeon CPU processor [34]. Study of sea ice scenarios has gained more importance lately, and there are many researchers working on the simulation of sea ice load. However, while a new chapter of knowledge in this field has been opened, there is still much to learn about the implementation of sea ice load scenarios especially in an efficient and high speed environment, which is needed to achieve HPC.

### *D. Sea Ice Load*

Engineers work with different types of environmental inputs to estimate probabilistic distributions of sea ice parameters in order to simulate the interaction between sea ice and offshore structures. Monte Carlo simulations have been applied to model the impact forces between sea ice and offshore structures. However, these previous applications did not run the Monte Carlo simulation on a GPU [35].

Sea ice load scenarios are complex and difficult to simulate because of numerous unpredicted situations that happen in an environment. For instance, the type of ice, structures and environmental factors vary over the course of the simulation. Also, finding an efficient way to simulate sea ice load scenarios is time consuming with engineering tools (e.g. MATLAB). Therefore, it is significant to come up with an idea to implement this complex model in a parallel environment such as CUDA application programming interface and use a robust Monte Carlo algorithm to achieve HPC.

## III. METHODOLOGY

### *A. Interaction Model*

When ice moves and interacts with a structure, it can create huge forces. The present scenario is based on an offshore structure that encounters many ice floes during its lifetime, and therefore needs to be designed to withstand the possible ice forces. Once the interaction starts, the initial kinetic energy of the ice floe is decreased by ice failing at the ice-structure interface and driving force is added from surrounding ice and effects from wind and currents. Starting with the initial kinetic energy $k_0 = \frac{1}{2} mv^2$ (m (Kg) is the mass and v (m/s) is the impact velocity) minus the dissipated crushing energy plus the floe driving forces, one can calculate the remaining kinetic energy after each meter of crushing. Once the kinetic energy is depleted, there will be no further crushing of this floe. The impacting floe will eventually rotate and clear around the structure, either because of the initial eccentricity unequally loading across the back or a change in drift direction. 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 and ridge**.** Table. I shows the fixed and the distributed parameters needed to calculate the maximum force between ice and an offshore structure. In this scenario, user-defined cumulative distribution functions are used for ice thickness, ridge thickness, and impact velocity.

Consideration is given to the ice types likely to be present, as thicker ice features may arrive over time. The force due to wind and currents are generally much smaller than ridging forces associated with surrounding ice, but may play a role when surrounding ice is not present. When considering the wedges of rubble behind the impacting floe, the wind and current forces on the rubble field are included with the driving force. The load on structure due to the surrounding ice based on ISO 19906 is modeled as:

$$F = F_{wind} + F_{current} + F_{ice} \quad (1)$$

$$where \quad F_{wind} = A_{floe}\,\rho_a\,C_{10}\,u_{10}^2 \quad (2)$$

$$and \quad F_{current} = A_{floe}\,\rho_w\,C_w\,u_w^2 \quad (3)$$

The distribution of wind speed follows a Weibull distribution and the current velocity is assumed 3.3 percent of the wind velocity.

In equation (**2**) $A_{floe}\,(m^2)$ is floe surface area, $\rho_a\,(kg/m^3)$ is air density, $C_{10}$=0.01 is drag coefficient, and $u_{10}\,(m/s)$ *is* wind velocity. While in **(3)**, $A_{floe}\,(m^2)$ is floe surface area, $\rho_w\,(kg/m^3)$ is sea water density, $C_w$ =0.004, and $u_w(m/s)$ is current velocity. In ISO 19906, the ice ridging force on the back of a feature due to surrounding ice is approximated as:

$$F = F_L D \quad (4)$$

$$F_L = R t^{1.25} D^{-0.54} \quad (5)$$

In equation (**4**) $F_L$ is the unit driving force and $D$ is the diameter of the floe. In equation (**5**), $R = 4\ (MN/m)$ and $t$ is uniformly distributed between 0.5 and 1 *(m)*. The global average pressure $p_G$ , as used for this experiment, is as follows:

$$p_G = C_R \left(\frac{h}{h^*}\right)^n \left(\frac{W_s}{h}\right)^m \quad (6)$$

Where $h$ is the ice thickness (m), $h^* = 1(m)$, $C_R$ is an ice strength coefficient, $W_s(m)$ is structure width, $m = -0.16$ and $n = -0.5 + 0.2\,h \ \ if\ h < 1(m) \ \ and\ -0.3 \ \ if \ \ h \geq 1(m)$. At the start of each year, the number of impacts are determined, and for each impact the relevant parameters and resulting impact loads are determined. The ice force during the interaction between ridge and level sea ice and the vertical sided structure is determined as:

$$F_{ice} = P_G\,A_{floe} \quad (7)$$

Following equations **2**, **3**, **4**, **5**, **6** and **7** the maximum load on the structure can be calculated (equation **1**).

TABLE I. MODEL PARAMETERS

| Parameter | Symbol | Unit | Unit Value or Distribution Type |
|---|---|---|---|
| **Structure Width** | $W_s$ | m | 80 |
| **Level Ice Thickness** | h | m | User-Defined |
| **Ridge Thickness** | | m | User-Defined |
| **Ridge Length** | | m | Uniform (lower=50 (m), upper=300 (m)) |
| **Floe Encounter Rate** | | floes/year | Gamma (mean=50 (m) , Std=30, lower=10 (m), upper=150 (m)) |
| **Ridge Encounter Rate** | | ridge/ year | Every Second Floe |
| **Floe Diameter** | D | m | Exponential (mean=300, Std=100, lower=10, upper=inf) |
| **Ice Strength** | $C_R$ | MPa | Uniform (lower=1.8, upper=2.8) |
| **Mass** | $m$ | Kg | |
| **Impact Velocity** | v | m/s | User-Defined |
| **Wind Velocity** | $u_{10}$ | m/s | Weibull(mean=6, Std=3, a=6.774 (m) , b=2.1013 (m)) |
| **Current Velocity** | $u_w$ | $m/s$ | 3.3 % of Wind Velocity |

*B. Simulation Process*

One of the processes of implementing and designing a model of a real or an abstract system is numerical computer simulation. In order to understand the statistical behavior of the system, it is possible to conduct numerical experiments on the model. We can use random values with a specific probability distribution for a sampling experiment. This is a Monte-Carlo type simulation, a broad class of numerical algorithms that can be helpful to solve complicated systems and predict future behavior of the model. The present study has different types of fixed and distributed parameters. Those distributed ones follow specific probability distributions and they are dependent upon the use of random numbers. Generally a Monte-Carlo type simulation is well suited for modelling sea ice loads as it allows for capturing the random nature of the parameters and processes involved. The goal is to find the maximum annual force between sea ice and a vertical structure corresponding to a probability of 1 in 10,000. Therefore, this study simulates an order of magnitude years (e.g. 1,000,000) to find stable results. By using Monte-Carlo techniques and breaking the problem into smaller pieces, implementation of each interaction model is much easier. Based on the number of simulated years, one can use individual simulations or a series of iterations. In each iteration, there are several types of distributions that need to be calculated based on the generated random numbers as shown in Fig. 1. The present scenario tracks and stores the maximum ice forces for each interaction in each year, with associated parameters, as shown in Fig. 2 and calls for the use of the Monte-Carlo technique to determine the extreme loads between ice and structure as shown in Fig. 3.

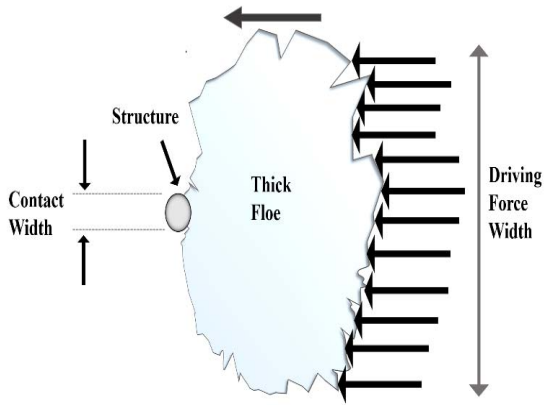


Figure 1. Driving Force on Thick Floe due to Surrounding Ice

*C. GPU Implementation of Parallel Algorithm*

While parallel computing gained more importance in programming, engineers recognize the future need of new processing architecture in which it is important to build a market before achieving that processing architecture.

In 1999, NVIDIA introduced GeForce 256 as the world's first GPU with sophisticated single-core design rather than a chip-scale parallel processor. In the multi-core era, GPU architecture is not only a powerful graphics engine, but also has high number of parallel processors, which becomes increasingly important [8]. There was always the challenge of programing thousands of parallel threads, but it is more difficult to have insight into the performance bottlenecks on a GPU to improve application performance. The architecture of the modern GPU improved in a different way than that of the CPU and also the GPU has become a general purpose architecture [9].

Also a Monte-Carlo method has become more interesting as computers became more powerful. As the main concern of this experiment involved with random number generation (RNG) for complex sea ice scenario, it is indisputable to use an efficient parallel algorithm on a GPU to save a lot of time. Therefore, they can increase the speed of parallel processes and improve the performance results for different applications. In that way computers send each process to different processors, with each performing a calculation in parallel. Sometimes upgrading a system gives additional power in the simulation process. A parallel algorithm divides that problem into discrete and smaller problems that can be solved concurrently and all those tasks can be done simultaneously in multiple processors [10].

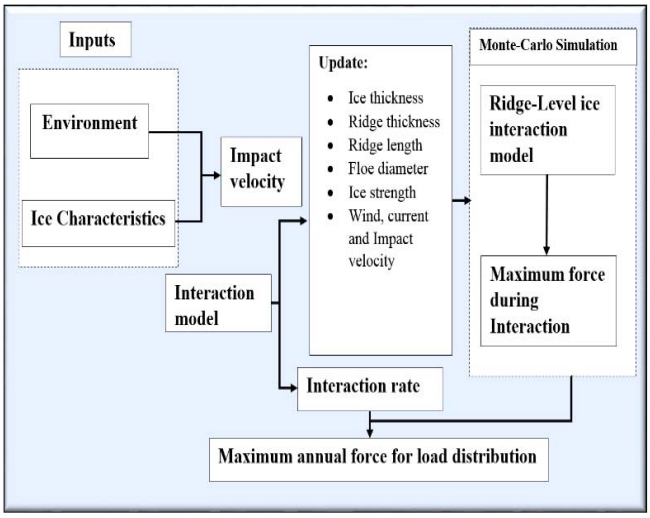


Figure 2. Monte Carlo Framework

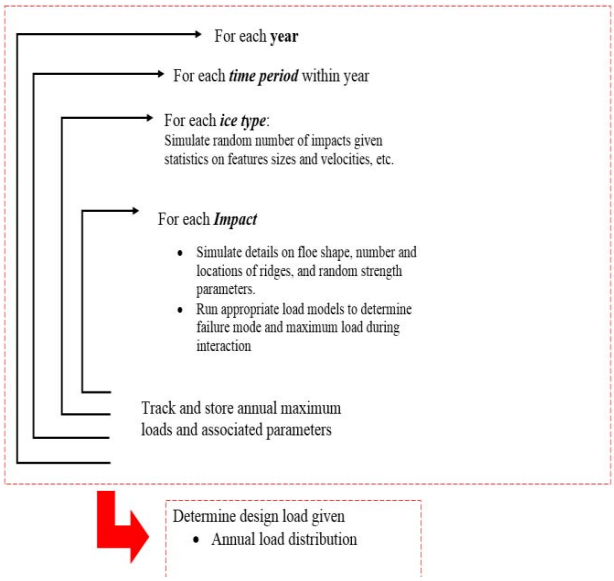


Figure 3. Probabilistic Framework of Load Characteristics

*D. CUDA Programming on Tesla K80*

The present study uses CUDA environment on the NVIDIA Tesla K80 GPU in order to accelerate our most demanding HPC in the simulation of the ice load scenario. This chip helps us to crunch large data and accelerate algorithms that can be 10 times faster than optimized CPU implementations. In order to fully leverage the computational resources of the GPU with minimal effort, CUDA must scale hundreds of cores and thousands of threads. CUDA can use both CPU and GPU as separate devices to do simultaneous computations without contention from memory resources. Serial portions of applications run in the CPU or the host while parallel portions of code are executed on the GPU or the device as computational kernels. CUDA threads are extremely lightweight in terms of the creation of overhead and switching. Thousands of CUDA threads can be created in just a few cycles. As a result, there is no creation overhead to be amortized over the execution of a kernel. So the kernel consists of just a few lines of code, resulting in performance gains. One of the basic tenants of achieving good performance in CUDA is to exploit the nature of the lightweight thread by launching kernels with thousands of concurrent threads.

Each thread has an ID and when threads run the code, they transfer different works and control decisions. While threads are executing independent works, they can pass their elements to a function, and store the result of an output array by reusing thread ID [11]. The present scenario assigns one thread to each year, therefore, there exist one million thread responsible for calculation of maximum force between sea ice and a vertical structure in parallel.

Monte-Carlo simulation applies in each thread to generate random numbers and pass them to the appropriate function which is defined in a specific kernel in order to calculate the maximum annual force. There exists GPU optimized libraries, data structure and algorithms such as CURAND and CUDA Thrust to fasten the process of this implementation.

*E. CUDA Libraries*

The present scenario shows the application of the two most important libraries in CUDA. The most important part of many scientific and functional applications is the generation of random numbers. The NVIDIA CUDA random number generator library (CURAND) focuses on the efficient generation of pseudo-random and quasi-random numbers [12]. It has a flexible interface which allows the user to use random number generator (RNG) algorithms either in the CPU or the GPU. It means CURAND includes two pieces: a device (GPU) header file and a library on the host (CPU). For a device generation of random numbers, the actual work occurs on the GPU [13]. The user can copy random numbers back to the host for further processing or call on their own kernels to use the random numbers. However, for the CPU generation of random numbers, all of the work is done on the host and they would be stored in host memory [12]. The CUDA toolkit includes another library named Thrust that is a C++ template. Thrust is a high-level interface that simulates the basic algorithms on the GPU. It defines two vector templates: host-vector and device vector. Thrust contains data parallel primitives such as sort, scan, and transform so the programmer can freely write just a few lines of code and reduce the operations significantly with regards to multi-core CPUs and create the most efficient implementation [13].

There are general guideline principles for using GPU-Based Monte-Carlo simulation especially in CUDA environment. Not only is this method of simulation efficient for the present scenario, it is also really robust. Here is one of the examples of kernel function used to calculate the wind speed based on the Weibull distribution by CURAND library.

```
__device__user_data_t
ran_weibul(curandStateMRG32k3a_t
*localState, const user_data_t a,const user_data_t b)
{
user_data_t r = curand_uniform(localState);
return a * pow(-log(r), (1 / b));
}
```

The following code shows how maximum annual force is calculated by using the Monte-Carlo simulation using CUDA Thrust library on a single GPU.

```
int main(){
size_t N = 1000000;  //Number of years
thrust::device_vector<yearResult> maxAnnualForce(N);
thrust::counting_iterator<unsigned int> index(0);

thrust::transform(index,index+N,maxAnnualForce.begin(
),MaxForces())
thrust::host_vector<yearResult> m = maxAnnualForce;
}
```

Random numbers are generated in parallel and data is stored on the GPU directly. Function evaluation and aggregation are done on the GPU using parallel constructs and highly GPU-optimized algorithms.

## IV. EXPERIMENT PROCEDURES

The present study focuses on the calculation of the maximum annual force between sea ice and a vertical sided structure using Monte Carlo simulation on the GPU. Monte Carlo simulations are ideally suited to GPU implementation and have been found to offer significant speedup over single CPU implementation in various lines of research [6].

For achieving a high level of confidence with this simulation technique, this scenario was simulated over 1,000,000 years in the CUDA environment. Large and parallelizable environment of this scenario call for the use of a GPU, with thousands of cores, in order to perform many calculations simultaneously. This method examined the sea ice load in 5 cases starting from 10,000, 50,000, 100,000, 500,000, and 1,000,000 years, and assigned one CUDA thread for each year. Therefore, 1,000,000 threads are working to calculate the maximum annual force between sea ice and a vertical sided structure over 1,000,000 years.

The present study has certain variables in the model with certain probability distributions. After performing sampling experiment upon the model, it is required to create a stochastic simulation of the system behavior which is called Monte Carlo simulation. Therefore, RNG algorithm could be used to generate random number for the specific probability distribution of the parameter. MRG32k3a is one of the high quality random number generator used in this study.

After defining the number of years, the thrust library in CUDA uses its device vector to call `maxAnnualForce`*(N),* and then floe encounter, floe diameter, wind speed, level ice thickness, ridge thickness, and impact velocity are calculated using the CURAND library. Thrust uses its *`counting_iterator`* to define the index of a thread and to transform the result of each thread's calculation from device memory (`device_vector`) to the host memory (`host_vector`). Therefore the threads will be emptied after transforming each result to the host memory, and the Monte Carlo framework will be updated in each year to generate random numbers based on the ice characteristics.

The reason for choosing this Tesla K80 GPU was that, this GPU allows large data sets to be processed, and accelerates algorithms up to 10 times faster than optimized CPU implementations. If the boost clock is enabled automatically, each GPU works independently, which can be useful for this scenario with many headrooms in the workload.

The goal of this study is not only to find the maximum annual force for the sea ice on an offshore structure, but to interpret the behavior of the GPU and the multi-GPU against the optimized CPU implementations.

## V. Performance Results

When parallel computing involves large-scale data, having the highest speedup becomes increasingly important in scientific computations.

There are three types of speedups based on its linearity: sub-linear, linear and super-linear. With efficient utilization of resources by multiprocessors, we may observe super-linear speedup which means the speedup with p processors is greater than *p* [14]. It is noteworthy that different GPUs need to communicate to transfer data between cores, and if the communication cost of the problem is large, achieving even linear speedup would be impossible. Next, using different memory utilization causes a reduction in performance, because multiple GPUs distribute the application's data on different GPUs. Finally, by having heterogeneous devices with different capabilities, one should not expect to have a linear speedup, meaning the speedup is equal to the number of processors [15].

This study uses Tesla K80 GPU which uses dual GPU design. It means there are 2 GPUs for a Tesla K80 and 8 GPUs for 4 Tesla K80 which work on this implementation simultaneously. A comparison happens when there are optimized version of both implementations on GPUs and CPU. Therefore, speedup are given for those optimized versions of implementations on both GPUs and CPU. Fig. 4 shows a speedup of up to 8 when comparing optimized implementation of 4 GPU Tesla K80 against the GPU Tesla K80. It also demonstrates a speedup of up to 130 for 4 Tesla K80 GPUs against optimized CPU OpenMP Implementation.

Fig.5 shows the elapsed time of different implementations reduced from about 2.5 hours to 0.7 seconds. These implementations include 4 Tesla K80 GPUs, single Tesla K80 GPU, serial CPU, parallel CPU (OpenMP). The simulation has been done by the GPU, multi-GPU, serial CPU, and parallel CPU (Open MP) implementations. The types of available GPU, multi-GPU, and CPU used in this study are Tesla K80, 4 Tesla K80s, and Intel Xeon R E5-2630 respectively.

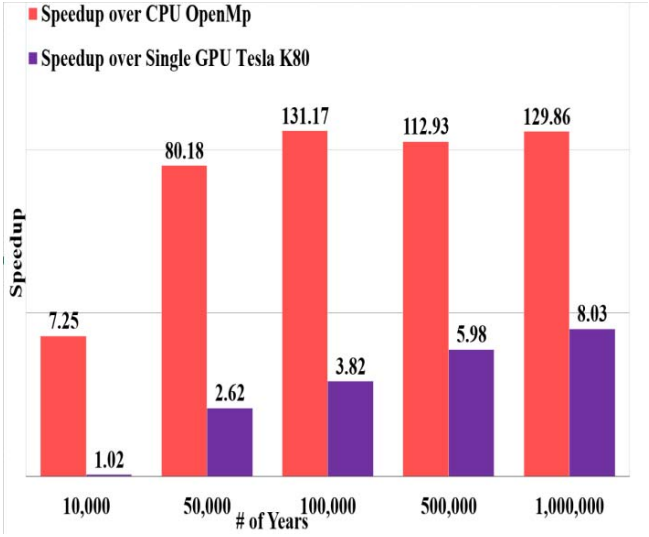


Figure 4. Speedup Comparison of Optimized Implementations

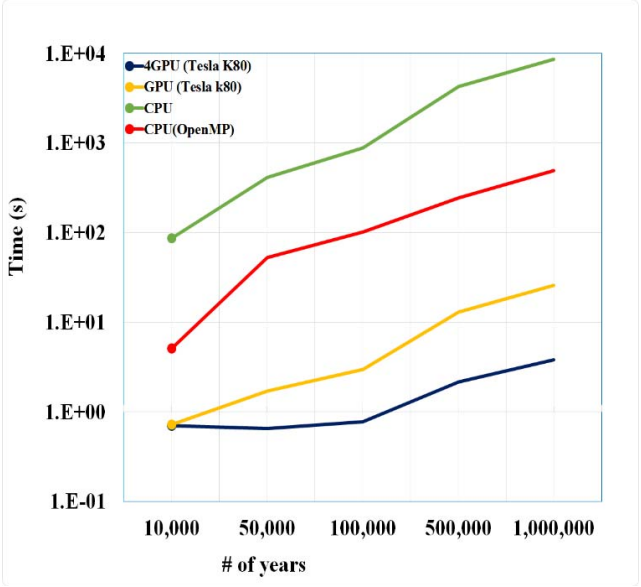


Figure 5. Elapsed Time

The computational time for 4 GPUs is approximately the same as single GPU for 10,000 year s, because the number of threads is less than the total number of CUDA cores. Therefore, as expected, the present study did not use the full efficiency of the cores on the GPUs. However, when considering 50, 000 years and more, the computation time for 4 GPUs was less than for a single GPU because it did not use the full efficiency of the cores on the GPUs.

## VI. CONCLUSION

The present study demonstrates a significant speedup for the complex simulation of the sea ice load. It is known that running a Monte-Carlo simulation requires a long execution time, but using powerful computers with recent GPUs decreases the computation time of the optimized implementation. It is also possible to increase throughput by running many independent computations simultaneously. There are always multicore processor markets to offer different types of devices and architectures, which make improvements in efficiency of the implementation. Therefore, our implementation is more optimistic than previously thought by using the art of parallel programming on a GPU, using recent devices that are more efficient and running highly optimized version of implementations.

## VII. FUTURE WORKS

The present study demonstrates a significant computational speedup for complex simulation of the sea ice load. The question now becomes how to achieve even better speedup and performance results. First, it is known that running a Monte Carlo simulation requires a long execution time, but using powerful computers with recent GPUs decreases the computation time of the optimized implementation. Next, understanding the basic idea of performance in a parallel environment is required. For instance, it is possible to improve throughput of the program by running the computation many times in many available processors. Although it takes a lot of work for the programmer to run an efficient code on multi-processors, it is worthwhile to get a substantial speedup for individual jobs. Therefore, it is possible to run the computation faster than before and measure the speedup and efficiency at the end. Finally, one can achieve a massive size up for a computation and run the computation on larger problems. One needs to measure the efficiency of computation and use a weak scaling test to see how large a problem one can efficiently run. For instance, the present scenario runs the program from 10,000 to 1,000,000 years. It should be clear that one can use any combination of methods to run the project faster in a parallel environment [14].

The present study only focused on one complex scenario which was the interaction between ridge-level sea ice and a vertical offshore structure when wind, currents, and kinetic energy are involved. There will be other factors that cause this interaction to be different than before. It is helpful to be familiar with the different characteristics of sea ice and explore efficient ways such as the Monte Carlo simulation to implement different scenarios. With the help of the performance analysis explained in the result section, one can improve an implementation by running the computation on a large scale, or with many processors. This can be helpful in making an efficient implementation and reaching a significant speedup. Also, finding a way to predict the efficiency of future scenarios by this implementation would help to save more time and energy. For example, when estimating the speedup of 4 GPUs based on the simulation's behavior, it is possible to predict what happens if there are more GPUs. Throughput is one of the key elements in performance analysis. If we consider ***n*** number of computations in a problem, how much faster can one run the project? This can be calculated by dividing the number of computation to unit times. If a programmer designs a problem by independent computations,

throughput can be increased by running them alongside each other simultaneously, which is limited only by the number of processors [15]. The present scenario contains a huge number of computations making it difficult to estimate the throughput of the code. Therefore, it is possible that this research can be continued in the future. The other method in performance analysis is to calculate the efficiency of the implementation with many processors that can be obtained by $E = \frac{S}{P}$ , where $S$ is speedup and $P$ is the number of available processors [15]. If $P$ = 1, it means we have 100 % efficiency.Therefore, improving the computational efficiency by means of parallel computation on the GPU, increases the throughput and speedup. Estimating the efficiency of the code will present opportunities for future research. With the help of this research and performance analysis, future studies related to the simulation of parallel algorithms on GPUs will become much easier to work with and can usher in a new chapter of high performance computing in this era.

## REFERENCES


[1] Boz, A. "Massively parallel Monte Carlo simulation using GPU". 2011.

[2] NVIDIA Corporation. "NVIDIA on GPU computing and the difference between GPUs and CPUs". 2016c.

[3] National Research Council. "The Future of Computing Performance: Game Over or Next Level". National Academies Press, 2011.

[4] Alawneh, Shadi, Carl Howell, and Martin Richard. "Fast quadratic discriminant analysis using gpgpu for sea ice forecasting." (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS).

[5] Shadi Alawneh, Roelof Draget, Dennis Peters, Claude Daley and Stephen Bruneau, IEEE Transactions On Computers, vol 64, No. 12, December 2015, pp. 3475-3487.

[6] Lee, Victor W., et al. "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU." ACM SIGARCH Computer Architecture (2010):

[7] Ayubian, Sara, Shadi Alawneh, and Jan Thijssen. "GPU-based monte-carlo simulation for a sea ice load application." Proceedings of the Summer Computer Simulation Conference. Society for Computer Simulation International, 2016.

[8] Glaskowsky, Peter N. "NVIDIA's Fermi: the first complete GPU computing architecture." White paper 18 (2009).

[9] Kim, Sunpyo Hong Hyesoon. Memory-level and Thread-level Parallelism Aware GPU Architecture Performance Analytical Model. Vol. 3. Technical Report TR-2009, 2011. [10] Kumar, Mahesh, Ms Sapna Jain, and Ms Snehalata. "Parallel Processing using multiple CPU", International Journal of Engineering and Technical Research (IJETR), Vol2, 2014

[11] NVIDIA Corporation 2016a. "CUDA FAQ", https://developer.nvidia.com/cuda-faq.

[12] NVIDIA Developer 2016. "Programming guide", http://docs.nvidia.com/cuda/cuda-c-programming-guide.

[13] Corporation, N. 2016b. "CURAND library", https://developer.nvidia.com/curand.

[14] Shan, Jing. "Superlinear Speedup in Parallel". Computation." CCS, Northeastern Univ., Massachusetts, Course Report (2002).

[15] SciNet 2015. "Introduction to performance – SciNetWiki, https://wiki.scinet.utoronto.ca/wiki/index.php/Introduction_To_Performance.

[16]http://www.nvidia.ca/object/what-is-gpu-computing.html

[17] Prinslow, Garrison. "Overview of performance measurement and analytical modeling techniques for multi-core processors." UR L: http://www. cs. wustl. edu/~ jain/cse567-11/ftp/multcore (2011).

[18] Suchard, Marc A., et al. "Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures." Journal of computational and graphical statistics 19.2 (2010): 419-438.

[19] Hannemann, Müller-, Matthias, and Schirra, Stefan, eds. "Algorithm engineering: bridging the gap between algorithm theory and practice". Vol. 5971. Springer, 2010.

[20] Spiechowicz, J., Kostur, Marcin, and Machura, Lukasz. "GPU accelerated Monte Carlo simulation of Brownian motors dynamics with CUDA." Computer Physics Communications 191 (2015): 140-149.

[21] Brandao, Diego, et al. "Performance evaluation of optimized implementations of finite difference method for wave propagation problems on GPU architecture (22nd International Symposium on. IEEE, 2010.

[22] Alawneh, Shadi, et al. "Hyper-real-time ice simulation and modeling using GPGPU." IEEE Transactions on Computers 64.12 (2015): 3475-3487.

[23] Cullinan, C., Wyant, C., Frattesi, T., and Huang, X. Computing performance benchmarks among CPU, GPU, and FGPA.

[24] "Monte Carlo simulation and its efficient implementation" http://www.nag.com/Market.

[25] "Monte Carlo methods in CUDA" http://www.thalesians. com

[26] Kroese, Dirk P., et al. "Why the Monte Carlo method is so important today." Wiley Interdisciplinary Reviews: Computational Statistics 6.6 (2014): 386-392.

[27] N. Metropolis. The beginning of the Monte Carlo method.Los Alamos Science, 15:125–130, 1987.

[28] Metropolis, Nicholas, et al. "Equation of state calculations by fast computing machines." The journal of chemical physics 21.6 (1953): 1087-1092.

[29] R. H. Swendsen and J.-S. Wang. Nonuniversal critical dynamics in Monte Carlo simulations. Physical Review Letters, 58(2):86–88, 1987

[30] Coddington, Paul D. "Analysis of random number generators using Monte Carlo simulation." International Journal of Modern Physics C 5.03 (1994): 547-560.

[31] Wang, Chun, et al. "Statistical methods and computing for big data." arXiv preprint arXiv: 1502.07989 (2015).

[32] Sheppard, Andrew. "CUDA Accelerated MonteCarlo for HPC." Sc11, Seattle, WA (November 2011).

[33] Goldsworthy, M. "A GPU–CUDA based direct simulation Monte Carlo algorithm for real gas flows.Computers & Fluids". 94 (2014), 58–68.

[34] Yusoff, M. S., and Jaafar, M. Performance of CUDA GPU in Monte Carlo simulation of light-skin diffuse reflectance spectra. In Biomedical Engineering and Sciences (IECBES), 2012 IEEE EMBS Conference on, IEEE (2012), 264–269.

[35] Jia, Xun, et al. "Development of a GPU-based Monte Carlo dose calculation code for coupled electron–photon transport." Physics in medicine and biology 55.11 (2010): 3077.