GPU-Based Monte Carlo Simulation for the Design Sea Ice Load

By

Sara Ayubian

A Thesis
Submitted to the School of Graduate Studies in
Partial Fulfilment of the Requirements for the Degree
of
Master of Science

Memorial University of Newfoundland

Master of Computer Science (2017)

Memorial University of Newfoundland
St. John's, Newfoundland

Title: GPU Based Monte Carlo Simulation for the Design of Sea Ice Load

Author: Sara Ayubian, B.Sc (Shahid Beheshti University, Tehran, Iran)

Supervisors: Dr. Shadi Alawneh, Dr. George Miminis, Dr. Martin Richard

Number of pages: 72

Abstract

Modern Graphics Processing Units (GPUs) with massive number of threads and many-core architectural components 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 then implemented in both GPU and CPU (central processing unit). Results show a speedup of up to 130 times for the 4 Tesla K80 GPUs over an optimized CPU OpenMP (Open Multi-Processing) implementation and a 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 reduced from about 2.5 hours to 0.7 seconds.

Acknowledgements

I would first like to thank my supervisors Dr. Shadi Alawneh, Dr. George Miminis, and Dr. Martin Richard at Memorial University of Newfoundland. The door to Dr. Alawneh's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He steered me in the right the direction whenever he thought I needed it. I would also like to acknowledge Jan Thijssen at C-CORE Company for his very valuable comments on this thesis.

Furthermore, I must express my very profound gratitude to my mother (Zohreh) for providing me with unfailing support and continuous encouragement throughout my life and through the process of researching and writing this thesis. I especially appreciate my amazing late father (Mahmoud) for showing me that the key to life is enjoyment. I hope you were here with me to celebrate this achievement. This accomplishment would not have been possible without you. Thank you.

Contents

ABSTRACT **III**

ACKNOWLEDGEMENTS **IV**

LIST OF TABLES **VII**

LIST OF FIGURES **VIII**

LIST OF ACRONYMS **XI**

1. INTRODUCTION **2**

1.1 PURPOSE……..……....…...……...3-4

1.2 RESEARCH QUESTIONS & THESIS OUTLINE……..…...……5

2. BACKGROUND **6**

2.1 GPUS VS CPUS……..……..……..……..6-8

2.2 MONTE CARLO SIMULATION …….……..……...…….8-10

2.3 CUDA PROGRAMMING…….……..……..…...…...10-11

2.4 SEA ICE LOAD……..……11-12

3. PROBLEM STATEMENT **13**

3.2 SEA ICE ……..…….……..…….……14-15

3.2 SEA ICE GROWTH……...16

3.3 SEA ICE GEOMETRY……16

3.3.1 ICE TICKNESS…….…….…...17

3.3 .2 FLOE DIAMETER…….……..…….17

3.3.3 ICE STRENGTH……..……...18-19

3.4 ICE MOVEMENT……..……..….20

3.5 SIMULATION OF SEA ICE LOAD…….…...21-25

3.5.1 LOAD ON STRUCTURE……..……..……. 26-28

4. METHODOLOGY 29

3.3 METHODOLOGY……..……29-30

3.4 MONTE CARLO FRAMEWORK…….30-33

3.3 PARALLEL FRAMEWORK……..….33

3.3 GPU IMPLEMENTATION OF PARALLEL ALGORITHM……..….34

3.3 TESLA GPU ACCELERATOR……..……35-36

3.3 CUDA…… 36- 37

3.3 PARALLEL COMPUTING ON CUDA ENVIRONMENT……..…….. 37-38

3.3 CUDA LIBRARIES……...38-46

3.3 CUDA ADVANTAGES…….. 47

4.1EXPERIMENT PROCEDURES……..…… 48

6. RESULTS & CONCLUSION 49

6.1 PERFORMANCE RESULTS……..... 49-52

3.3 ICE LOADS……..……... 53

3.3 CONCLUSION……..54-55

8. FUTURE WORK **56-58**

9. REFERENCES 59-66

List of Figures

3.1 Sea Ice [Credit: National Snow and Ice Data Center]…….15

3.2 Ice in the central Arctic Ocean has thinned by more than 60 percent since 1975…15

3.3 Level-Ridge Sea ice Interaction with Vertical Structure……..23

3.4 Driving Force on Thick Floe Due to Surrounding Ice…….28

4.1 Model Framework……31

4.2 Monte Carlo Simulation Framework……32

4.3 Tesla K80……..36

5.1 Speedup Diagram……..49

5.2 Speedup of GPUs over Optimized CPU Implementation……51

5.3 Elapsed Time for Different Implementations……52

5.4 Probability of Exceedance……53

List of Tables

3.1 Ice Strength……..……19

3.2 Model parameters……..……24

3.3 Ice Level Thickness……..…….25

3.4 Ridge Thickness……..…….25

3.5 Impact Velocity……25

List of Acronyms

Acronym	Description
HPC	High Performance Computing
GPGPU	General-Purpose Computing on Graphics Processing Units
GPU	Graphic Processing Unit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
RNG	Random Number Generator
CURAND	CUDA RANDOM
ECC	Error Correcting Code

CHAPTER 1

Introduction

Initially, high performance computing (HPC) was restricted to governments, militaries and university research. Now, a super computer can be created for anyone who needs performance analysis, computation and data processing capacity [1].

The development of general purpose graphics processing units (GPGPU) has been an innovative development for HPC. This method can be applied by using graphics processing units (GPUs) to analyze data, perform non-specialized calculations and accelerate algorithms that would traditionally be conducted by the central processing unit (CPU) [2]. A CPU acts as the brain of the computer system and contains few cores dedicated to the sequential serial part of the program, while a GPU is an electronic circuit unit made of thousands of cores, which are responsible for handling multiple tasks concurrently [3].

GPUs have become a commercially attractive technology because these chips have the power to run projects that are computationally intensive. The powerful computational capabilities of GPUs stem from their vast available parallelism, which results in a significant speedup compared to conventional CPUs. In other words, the architecture of GPUs is designed such that most of the transistors are devoted to data processing rather

than data caching and flow control. A computer can be utilized with GPUs to execute massive concurrent computations and to achieve efficient implementation [4].

1.1. Objectives

When sea ice moves, it may collide with structures and create huge forces. When wind, current and kinetic energy are considered, this interaction becomes more complex. Therefore, engineers estimate probabilistic distributions of sea ice parameters in order to simulate the interaction between sea ice and offshore structures.

The present scenario assumes that when sea ice interacts with vertical sided structures in its way and creates huge forces, either the ice or structures may fail. While this failure is important to consider, the present study focuses on the simulation of the interaction model in order to find the maximum force between sea ice and a vertical sided structure. This is because, sometimes, there may not always be a chance for interaction such as when there is no structure in the path of sea ice. Therefore, a sufficient number of years (e.g. 1,000,000 years) should be simulated such that convincing results are achieved at the design probability levels of interest.

The Monte Carlo simulation, which is very robust and relatively simple to implement, has been used in this study to evaluate the annual maximum force between sea ice and a vertical faced offshore structure when wind, current and kinetic energy are present. In other words, it uses probabilistic methods to simulate sea ice-structure interactions, rank the annual maximum loads, and plot them in lognormal scales

Monte Carlo simulations are a broad class of numerical algorithms for a large and complex problem where it is impractical or impossible to obtain analytical solutions, as it uses repeated sampling to determine the properties of certain phenomena. 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 [5].

The present study uses a HPC framework to compare Monte Carlo simulations run on CPUs and GPUs to determine which is better suited for evaluating the sea ice loads on a vertical structure. For achieving a high level of confidence with this simulation technique, this scenario was simulated over 1,000,000 years in the compute unified device architecture (CUDA) environment.

As this experiment involves different probability distributions for its parameters, a random number generator (RNG) algorithm is of great importance. Therefore, the Monte Carlo simulation, as a probabilistic framework for repeated random sampling, is a good fit in this study for generating random numbers. Furthermore, a CUDA programming environment provides specific libraries for RNG, which are useful for implementing this complex scenario using the Monte Carlo simulation on GPU to achieve HPC. The goal of this research was to analyze the performance results between GPUs and CPUs when both are processing the same algorithm, and to interpret the speedup of the optimized implementations.

1.2. Research Questions & Thesis Outline

The present study aims at comparing GPU- and CPU-based implementation of Monte Carlo framework to calculate design sea ice loads on offshore structure with a vertical face at the waterline.

Therefore, the following research questions will be addressed:

- What is the simulation process for the sea ice load application?
- What is a sufficient time period in which the simulation should be run?
- What type of algorithm is best fitted to this research?
- If the speedup is non-linear, what are the factors for this?

To answer these questions, the chapters of this thesis have been arranged as follow:

Chapter 2 describes the necessary background such as GPU Vs CPU, Monte Carlo simulation, CUDA programming, and sea ice load which help situate this research. Next, Chapter 3 explains the methodology used in this study to evaluate the sea ice load applications. Chapter 4 discusses the results and the conclusion of this study. Finally, Chapter 5 explores the potential of this research and areas for future study.

CHAPTER 2

Background

2.1. 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 cores designed for performing multiple tasks concurrently [3]. 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 [7].

When parallel computing involves large-scale data, having the highest speedup becomes increasingly important for scientific computations [11]. 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 $\boldsymbol{p}$ processors [71].

For instance, an article related to the GPU accelerated Monte Carlo simulation of Brownian motors dynamics with CUDA had a speedup of about 3,000 compared to that of a typical CPU. Furthermore, there is no optimization in this study for comparing a GPU efficient code against a CPU optimized implementation [13].

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 has 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 [15]. 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 [16]. 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. [17].

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 [13].

2.2. Monte Carlo Simulations

The Monte Carlo simulations, as a broad class of computational algorithms, are used in many different areas [19]. 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 [20]. 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 $\pi/4$ ($\frac{Area\ of\ Circle}{Area\ of\ Square} = \frac{\pi\, r^2}{4r^2} = \frac{\pi}{4}$) and multiply the result by 4 to estimate π [21].

Monte Carlo methods are also helpful for simulating real-life random system and deterministic numerical computations [22].

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 [23][24].

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 [25].

There are many problems in computer science which are considered very difficult or defined as NP (nondeterministic polynomial time) problems in the computational complexity class [26]. 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 [27].

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.

2.3. 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]. A common example of the Thrust library in CUDA is to estimate the value of the constant π using a Monte Carlo simulation as mentioned in previous section [30].

A simple example of using CUDA programming is the addition of two vectors and storing the result in a third vector. The following code explains the `main` part of the program for adding two vectors in CUDA. Allocating the memory on the GPU by `cudaMalloc ()` for three arrays, filling the array ‘x’ and ‘y’ on the CPU, copying the array ‘x’ and ‘y’ to the

GPU, copying the array ‘z’ back from the GPU to the CPU by `cudaMemcpy ()`, displaying the results, cleaning up the memory allocated on the GPU by `cudaFree()` are the parallel steps required for adding two vectors on CUDA [75].

```
#define N 10
 int main( void ) {
 int x[N], y[N], z[N];
 int *device_x, *device_y, *device_z;
 HANDLE_ERROR( cudaMalloc( (void**)&device_x, N * sizeof(int) ) );
 HANDLE_ERROR( cudaMalloc( (void**)&device_y, N * sizeof(int) ) );
 HANDLE_ERROR( cudaMalloc( (void**)&device_z, N * sizeof(int) ) );
for (int i=0; i< N;i++){
x[i]=-i; y[i]=y*i;}
 HANDLE_ERROR( cudaMemcpy( device_x, x, N * sizeof(int),
cudaMemcpyHostToDevice ) );
 HANDLE_ERROR( cudaMemcpy( device_y, y, N * sizeof(int),
cudaMemcpyHostToDevice ) );
 add<<N,1>>( device_x, device_y, device_z );
 HANDLE_ERROR( cudaMemcpy( z, device_z, N * sizeof(int),
cudaMemcpyDeviceToHost ) );
// display the results
for (int i=0; i<N;i++){
printf( “%d+%d=%d\n”,x[i],y[i],z[i]); }
 cudaFree( device_x );
 cudaFree( device_y );
 cudaFree( device_z );
return 0;}
```

In another studies, 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 [31].

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 [32].

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 [33].

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. This will be discussed in the next section.

2.4. 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 [34].

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.

CHAPTER 3

Problem Statement

In the present study, 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 RNG algorithm to calculate their appropriate distributions. Therefore, using an efficient algorithm, such as Monte Carlo, 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 find the maximum annual force between sea ice and a vertical structure corresponding to a probability of 1 in 10,000. Therefore, a sufficient number of years should be simulated such that stable results are achieved at the design probability levels of interest. As this experiment is difficult and time consuming, it would be effective to use a parallel environment such as, CUDA interface programming on GPU, and to calculate the speedup over CPU implementations.

Sea ice is not a uniform sheet of ice, but is a complex surface, and has many characteristics that should be considered in each experiment. When ice moves and interacts with a

structure, it can create huge forces [35]. Therefore, understanding the characteristics of ice is important before the loads can be simulated.

3.1. Sea Ice

Sea ice, frozen ocean water, forms, grows and melts in the Arctic Ocean. On average, sea ice covers about 25 million square kilometers of the earth, or about two-and-a-half times the area of Canada, which is shown in **Figure 3.1** [36]. In summer, the sea ice will break into distinct pieces, or floes that separate from or converge into each other in the ocean as shown in **Figure 3.2**. Therefore, floe diameters range from several meters to a hundred kilometers [37].

Ice progressively melts and continuously deforms as wind and current pushes it south, resulting in higher drift speeds. Therefore, ice is critically important in many contexts, such as understanding its characteristics at the molecular level, its influences on climate, lifespan, and the people who live in the Arctic, and its interactions with natural features or man-made structures [38] [39].

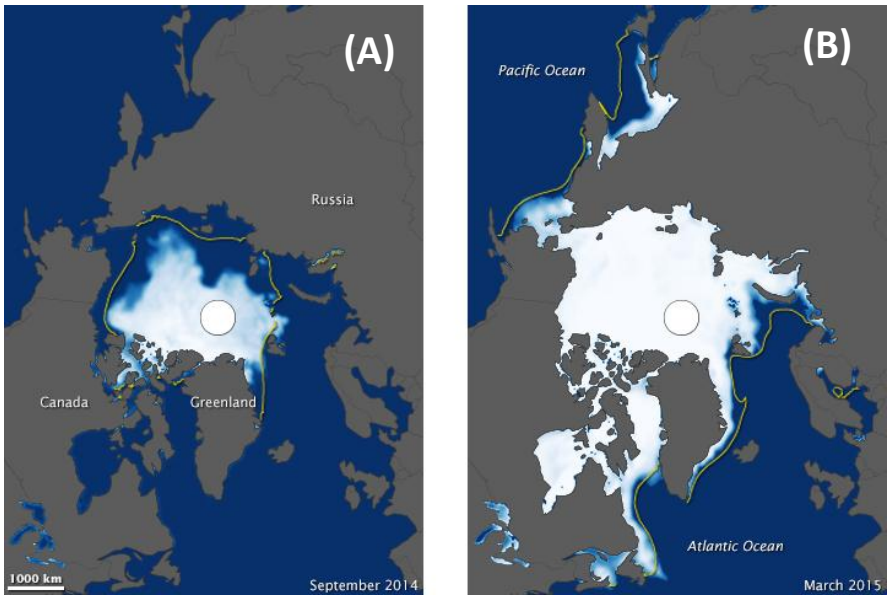

Figure 3.1 Sea Ice [Credit: National Snow and Ice Data Center]

Extent: **(A)** At the End of summer, and **(B)** at the End of Winter

Figure 3.2: Sea Ice. (Photo: Pablo Clemente-Colon/NOAA)

3.2. Sea Ice Growth

Sea ice has a complex structure resulting from its formation history and amount of salt. Ice loading begins with the formation of small separate crystals. As freezing continues in calm water, the fragments freeze together to form a continuous sheet through the winter. When weather becomes warmer, the first-year ice begins to melt. In the fall, only the thickest classes of first year ice will survive the summer melt period to become second year ice. In this condition, ice will remain until the following winter when it grows and is classified as multiyear ice [40].

There are several different classes of ice thickness which is related to the sea ice geometry and will be discussed in next section.

3.3. Properties of Sea Ice Floes

Researchers are interested in the geometry of sea ice due to a number of practical applications, such as transportation in arctic environments and design of offshore structures intended to survive in the presence of ice. Therefore, it is important to clarify ideas about floe diameters and ice thickness to propose techniques for measuring them [38] [40].

3.3.1. Ice Thickness

In the old days, the only method which was used to obtain ice thickness was to drill through the ice with human-powered auger and use a measuring stick through the hole to get the thickness. Recently determination of the ice thickness has assumed considerable importance in the view of the transportation into Arctic where the sea is covered with a layer of ice permanently. For instance, commercial vessels may be able to break through the ice which is accumulated over a period of years. Old sea ice has been found relatively hard and brittle and is much easier to break than is ice of recent region [41] [42] [43]. It became an interesting issue to be able to quickly and accurately measure the ice thickness and determine if the sea ice is relatively recent formation or whether it is a multi-year ice.

3.3.2. Floe Diameter

Sea ice floes break continuously into smaller pieces, and in the cold season they are at the same time frozen together into larger pieces. Floe size distribution shows statistical regularity based on random floe break-up mechanism. The characteristic of floe size can be defined from its surface area [38] [44].

The question is then how to formulate the breakage probability that would lead to the exact form of the statistical distribution. Logarithmic normal and exponential distributions should be treated as specific cases, depending on the formulation of breakage probability.

Until many more distributions are obtained at different regions and time, there is no point in speculating what the most common distribution is or what the range of possibilities might be.

3.3.3. Ice Strength

In most cases ice strength is problematic. Ice strength is related to a combination of factors such as ice thickness, temperature of the ice and the amount of salts in the ice. Ice strength on a large scale can be dominated by fracture processes and natural flaws and this creates the "size effect". Based on the target, evaluation of equation for the nominal crushing pressure of the sea ice, small scale strengths were the only available input [44].

The ice pressure equation does have the 'strength coefficient" C_R in it which is given in different values for different environments. For a deterministic analysis, C_R could be estimated by the approach suggested by ISO 19906 with respect to the region as **Table 3.1**

Table 3.1: Ice Strength

Region	**C_R Value**
Arctic	**2.8**
Sub-Arctic	**2.4**
Temperate	**1.8**
	1.9

Based on the probabilistic distribution of the sea ice floe and its strength, in this present document, C_R will follow uniform distribution with 1.8 and 2.8 MPa as lower and upper bounds.

Disintegration of the sea ice into floes will influence its large-scale geophysical properties. Geometry of sea ice can play an important role the way the ice deforms in response to forces applied by wind and currents. Understanding the geometry of floes and how these geometries change during the annual cycle has stimulated research into the physical process and implementation of ice interactions [38] [40].

3.4. Ice Movement

It is hard to understand that ice moves, given that it is solid, but it can and it does. Sea ice can move at speeds and in surge condition is known to move at up to three-hundred kilometers per year [76].

Satellites and marine radars are good to map general ice movement. For floe size and ridge length, aerial photography has been used extensively in the past and still gives excellent information to find the specific distribution of the floe interaction rate [45][46].

The user can select one of the two methods to determine floe encounter rate per year or floe interaction rates based on an average number of interaction per day as specified by user or calculated based on the distribution of concentration, floe diameter and floe velocity.

In the second case, it is possible to estimate mean from the number of floes of a given type that has interaction with the structure during a period of time. For each year, the number of interaction is sampled from a gamma distribution.

3.5. Simulating Sea Ice Loads

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 [47].

Previous simulations of sea ice loads only considered the interaction level between sea ice and vertical sided structure [18]. The present study improves upon this previous research by additionally considering the forces generated by wind, current, and kinetic energy. Therefore, the objective of this work is to develop a complex model for this sea ice load application to see how GPUs can be utilized for a large and complex system.

This section describes the random nature of the environmental forces and how the sea ice load is calculated. There exist different types of parameters based on the sea ice characteristics, offshore structures and surrounding environment. More than one approach is possible to estimate the characteristics of ice loads, and there are arguments in favour of all of them. In this experiment, one of the common approaches is used as suggested in ISO 19906.

Once the interaction between sea ice and a vertical sided structure 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. The initial kinetic energy may be calculated as follows:

$$k_0 = \frac{1}{2}\, mv^2 \qquad \textbf{(1)}$$

Where *m* is mass in *kg* and *v* in *m/s* is the impact velocity. The remaining kinetic energy after each meter crushed may be calculated by subtracting the dissipated crushing energy and adding the floe driving forces.

Once the kinetic energy is depleted, the floe is no longer crushed. 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**.** In this study, ridges are modeled as triangular shape with top width equal to 5 times the ridge thickness and they are perpendicular to the flow direction. Each second floe has still one ridge, but the ridge should be placed somewhere in the floe based on the uniform distribution between the floe start and end as shown in **Figure 3.3**. Ridge thicknesses are based on user-defined parameter, which means we were not able determine specific distributions for their thicknesses. Therefore, one needs to define the level of ridge thickness by picking up a random number in each specific interval with considering of ice cumulative distribution functions.

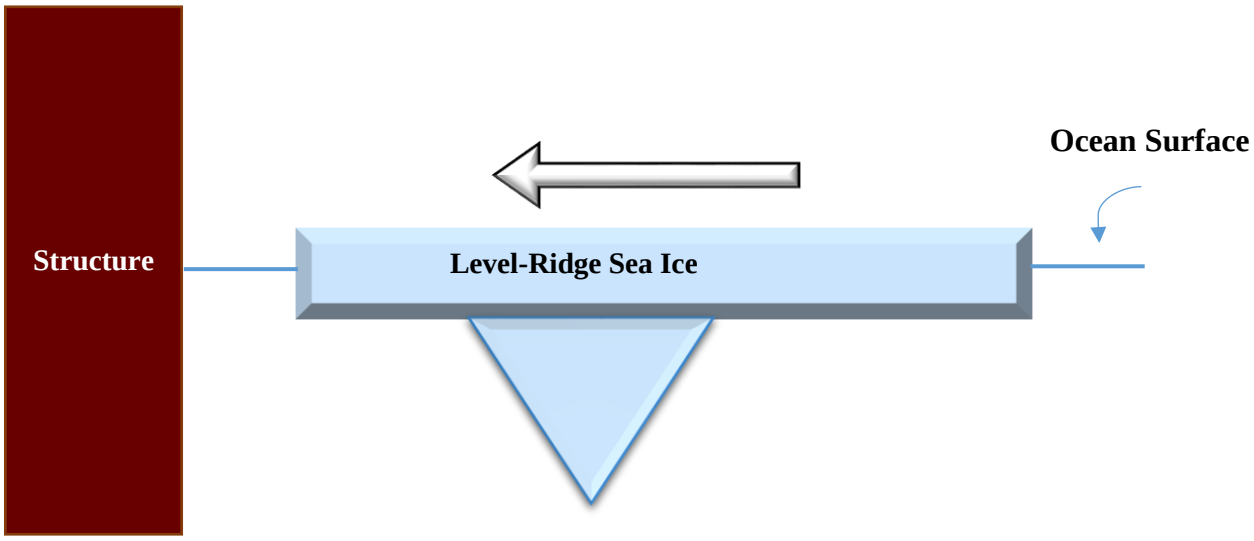

Figure 3.3: Level-Ridge Sea ice Interaction with Vertical Structure

Table 3.2 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 as shown in **Table 3.3, 3.4** and **3.5** respectively.

Table 3.2: Model parameteres

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

Table 3.3: Level Ice

Bin (m)	Cumulative
1.5	0.0%
1.5-2	10.2%
2-3	25.0%
3-4	56.1%
4-5	76.7%
5-7	89.1%
7-8	93.7%
8-9	97.3%
9-10	98.5%
10-11	99.7%
11-13	99.9%
13-14	100.0%

Table 3.4: Ridge Thickness

Bin (m)	Cumulative
< 3	0.0%
3 - 5	6.0%
5 - 6	27.0%
6 - 8	51.0%
8 - 9	74.0%
9 - 12	87.0%
12 - 15	96.0%
15 - 18	97.5%
18 - 21	99.5%
21 - 24	99.9%
24 - 27	100.0%

Table 3.5: Impact Velocity

Speed (m/s)	Cumulative
0	0
0- 0.1	0.2
0.1- 0.2	0.6
0.2- 0.3	0.8
03- 0.4	0.9
0.4- 0.5	0.95
0.5- 0.6	0.975
0.6- 0.7	0.99
0.7- 0.8	0.995
0.8- 0.9	0.999
0.9- 1.0	0.9995
1.0- 1.1	0.9999
1.1-1.2	1

3.5.1. Load on Structure

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 [48]. Given:

$$F_{wind} = A_{floe}\rho_a C_{10} u_{10}^2 \qquad (\mathbf{2})$$

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

the load on structure due to the surrounding ice based on ISO 19906 is modeled as:

$$F = F_{wind} + F_{current} + F_{ice} \qquad (\mathbf{4})$$

The distribution of wind speed follows a Weibull distribution and the current velocity is assumed 3.3 percent of the wind velocity [6].

In equation (**2**) $A_{floe}(m^2)$ is floe surface area, ρ_a (kg/m^3) is air density, C_{10}=0.01 is drag coefficient, and u_{10} (m/s) *i*s wind velocity. While in equation **(3)**, A_{floe} (m^2) is floe surface area, ρ_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:

$$F_{ice} = wD \qquad (\mathbf{5})$$

where D is the ice feature diameter or width and w is the ridging force per unit width. For a default, the model implemented in ISO 19906 (as suggested by K.Croasdale) is recommended:

$$w = Rh^{1.25}D^{-0.54} \qquad (\mathbf{6})$$

where $R = 4\ (MN/m)$, h is the ice thickness acting on the thicker ice feature which uniformly distributed between 0.5 and 1 *(m)* and D is the width of the thicker ice feature expressed in meters.

Following equations **2**, **3**, **5**, and **6** the load on the structure can be calculated (equation **4**) for a designed model which is shown in **Figure 3.4 [48]**.

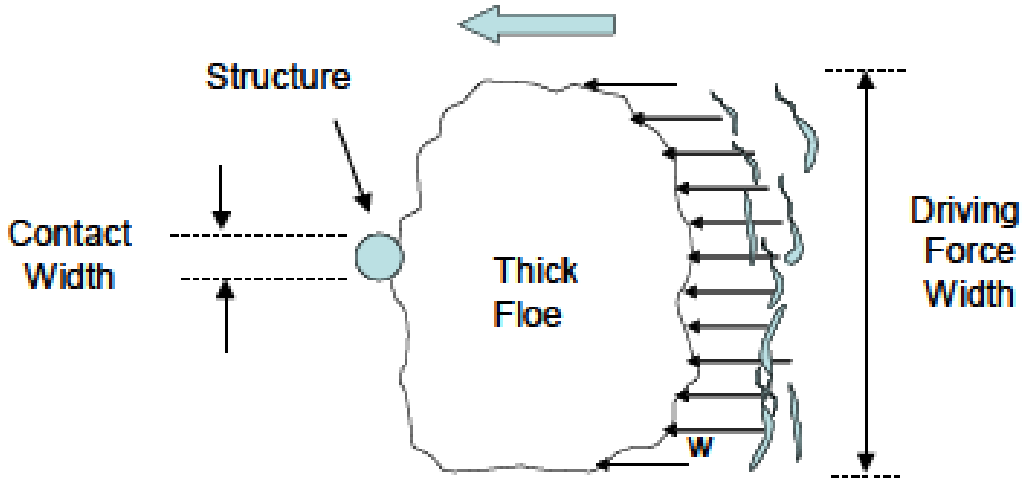

Figure 3.4: Driving Force on Thick Floe Due to Surrounding Ice

CHAPTER 4

Methodology

The interaction model of sea ice and a vertical sided structure was conducted in a parallel environment using a GPU-Based Monte Carlo simulation. This chapter introduces the techniques used in this scenario and describes how these techniques accelerate the process of evaluating sea ice load on structures.

When running a probabilistic scenario, one needs to: (1) consider each ice type and year; (2) determine the number of interactions; (3) update the input distributions appropriate for the features interacting with the structure; and (4) apply the appropriate model to determine the maximum force during each interaction. These steps may vary somewhat depending on the ice type and the model chosen [48].

4.1. Probabilistic Environment

One process for implementing and designing a model of a real system is numerical computer simulation [49]. In order to understand the statistical behavior of the system, it is possible to conduct numerical experiments on the model. Random values with a specific probability distribution can be used for a sampling experiment [50]. For example, Monte

Carlo simulations are a broad class of numerical algorithms that can be helpful for solving complicated systems and predicting future behavior of the model [22].

4.1.1 Monte Carlo Framework

A Monte Carlo simulation is a stochastic process for solving complex problems. It requires knowledge in a wide range of fields, such as probability to describe the random process, statistics to analyze the data, computational science to efficiently implement the algorithms, and programming to formulate and solve the problem of interest [22].

The present study involves different types of fixed and distributed parameters, as mentioned in **Table 3.2**. These distributed parameters follow specific probability distributions and are dependent upon the use of random numbers. The Monte Carlo simulation approaches the problem by generating random numbers, evaluating functions and aggregating the result based on the simulated data [29]. Generally, a Monte Carlo 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, a sufficient number of years should be simulated such that stable results are achieved at the design probability levels of interest (e.g. 1,000,000).

By using the Monte Carlo technique and breaking the problem into smaller components, the implementation of each interaction model is simplified. 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. The overall modeling framework is outlined in **Figure 4.1** and calls for the use of the Monte Carlo technique to determine the extreme loads between ice and structure as shown in **Figure 4.2**. Before specifying the distribution and values for the model input parameters, the time periods used to characterize variations in ice conditions through the year must be defined. Any number of time periods can be used,

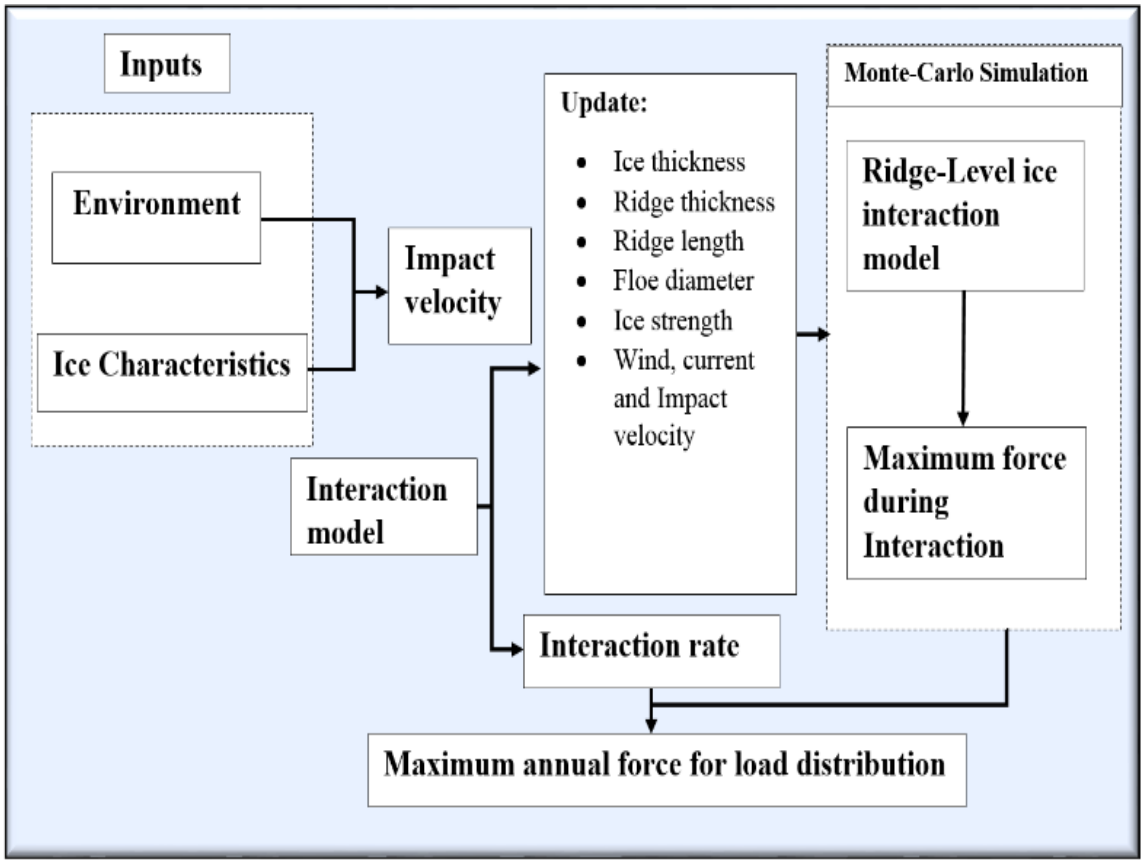

Figure 4.1: Model Framework

and the periods can be of fixed or variable durations. It may be appropriate in some analyses to ignore periods that do not contribute to the design load level. Once the annual maximum loads are determined, they are sorted and ranked so that the design value associated with a specified probability of exceedance can be assessed.

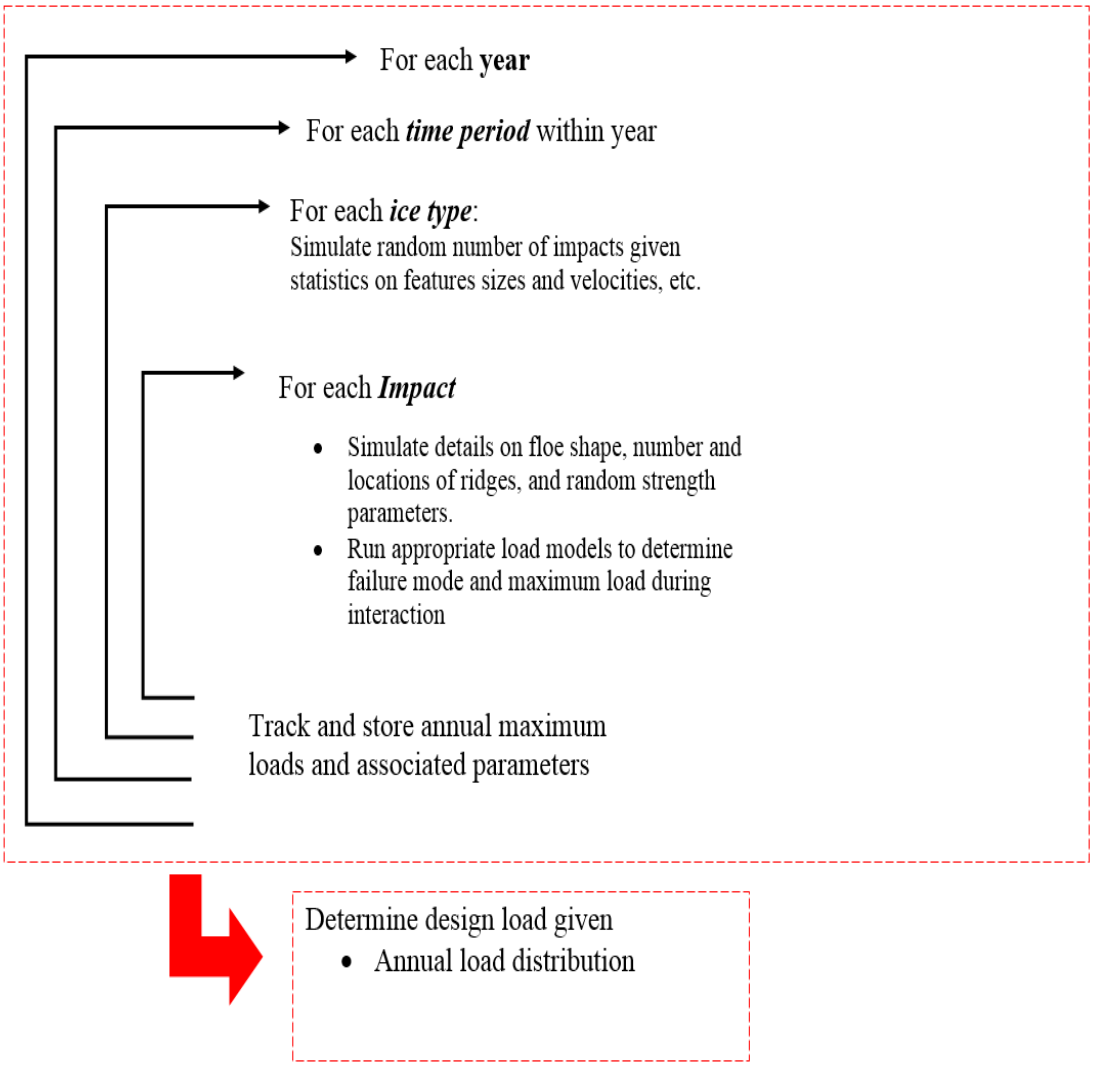

Figure 4.2: Monte Carlo Simulation Framework

Not only is it necessary to have a probabilistic environment to calculate maximum annual force between sea ice and a vertical structure over 1,000,000 years, but using a parallel environment also improves the performance result.

4.2. Parallel Environment

As parallel computing has gained more importance in programming, engineers recognize the future need of new processing architecture. This untapped market would be very interested in the powerful processing architecture offered by parallel algorithms.

Parallel algorithms are implemented in parallel computing environments, and are strongly dependent on the computer architecture for which they are designed [52]. In the parallel computing environment, specific problems may be solved using the associated software and libraries. Programming tools, performance tools and a debugger are included in this software to improve efficiency. The programming tools are used to develop algorithm that would exchange data during its execution, and match its underlying programming design. Performance tools are helpful for understanding and improving the parallel implementations, and also support the synchronization and communication of the various components. Finally, the debugger is an effective tool for finding bugs of a program [53].

The parallel computing environment benefits from GPU technology. The next two sections explain how the parallel processor GPU is used for implementing a parallel algorithm.

4.2.1. GPU Implementation of Parallel Algorithm

In 1999, NVIDIA introduced GeForce 256 as the world's first GPU with a sophisticated single-core design rather than a chip-scale parallel processor. The architecture of the GPU not only has a powerful graphics engine, but also has a high number of parallel processors. This is increasingly important because these processors can be applied to very large and complex problems [54]. Programming thousands of parallel threads in a GPU is difficult when the characteristics of the processors are not well understood. It is, however, even more challenging to evaluate the performance bottlenecks of a GPU for improving application performance. Fortunately, the architecture of the modern GPU has improved in a different way than that of the CPU and has become a general purpose architecture for scientific computing [55].

The Monte Carlo method has become more interesting as computers have become more powerful. As the main concern of this experiment involves random number generation (RNG) for a complex sea ice scenario, it is necessary to use an efficient parallel algorithm on a GPU to achieve optimal processing time. Importantly, GPUs can increase the speed of parallel processes and improve the performance results for different applications. Computers send each process to different processors, with each performing a calculation in parallel. Upgrading a system to a modern GPU provides additional power in the simulation process. A parallel algorithm divides that problem into discrete and smaller components that can be solved concurrently in multiple processors [56].

4.3. TESLA GPU Accelerator

The present study uses the CUDA environment on the NVIDIA Tesla K80 GPU in order to accelerate HPC when simulating ice load scenarios. This GPU chip allows large data sets to be processed, and accelerates algorithms up to 10 times faster than optimized CPU implementations. The NVIDIA Tesla K80 GPU has a dual slot computing module, meaning that it has two identical Tesla CK210 GPUs with 2496 processor cores each. This chip is designed for servers and has 24 GB (12 GB each) memory as shown in **Figure 4.3** [51]. However, if the error correcting code (ECC) is turned on, only 22.5 GB are available for the user, and the memory is reduced by 6.25 %. The core clock of a GPU is the actual speed on a video card. The boost clock is another characteristic specific to Tesla GPUs. When the number of GPU cores and memory clock rates increase, and thermal headroom and sufficient power are available, application performance improves and ensures maximum utilization [57]. In Tesla K80, the GPUs in each block have a base core clock of 560 MHz, and the boost clock varies from 562 MHz to 875 MHz. If the boost clock is enabled automatically, each GPU works independently, which can be useful for a scenario with many headrooms.

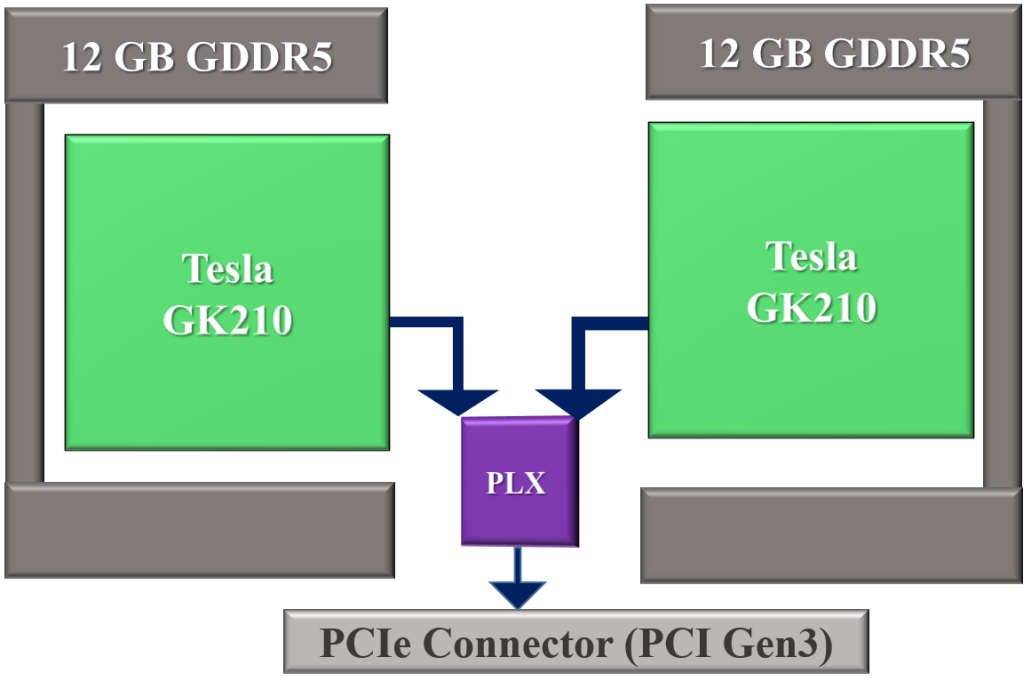

Figure 4.3: Tesla K80 Block Diagram

To improve performance, the present scenario uses 4 Tesla K80 GPUs which contain dual cores, meaning that there are 8 cores in total.

4.5. CUDA

In November 2006, NVIDIA had an opportunity to bring GPUs into the mainstream by introducing a programming interface, which it dubbed CUDA. This was an attempt to make the programming environment of GPUs more accessible to programmers. The 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 achieved great progress in the computer software industry by moving from serial to parallel programming [14]. This programming environment can take a simple model of data parallelism into a programming model without the need for graphic primitives. In other words, the CUDA environment makes the GPU look like another programmable device [59].

4.5.1. Parallel Computing On CUDA Environment

In order to fully leverage the computational resources of the GPU with minimal effort, CUDA must scale hundreds of cores and thousands of threads [60]. 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 [61]. 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 [62].

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 [60]. The present scenario assigns one thread to each year. Therefore, there are one million

threads responsible for calculating the maximum force between sea ice and a vertical structure in parallel.

The 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.

The next section discusses GPU optimized libraries, data structure and algorithms such as CURAND and CUDA Thrust, which accelerate the process of this implementation.

4.5.2. CUDA Libraries

The CUDA toolkit as a free application contains GPU accelerated libraries such as Fast Fourier Transform (CUFFT), Basic Linear Algebra Subroutines (CUBLAS), Sparse Matrix Routines (CUSPARSE), Dense and Sparse Direct Solver (CUSOLVER), Random Number Generation (CURAND), Image & Video Processing Primitives (NPP), NVIDIA Graph Analytic Library, Template Parallel Algorithm & Data Structure (Thrust) and CUDA Math Library [63].

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 [64]. The NVIDIA CUDA random number generator library (CURAND) focuses on the efficient generation of pseudo-random and quasi-random numbers [65]. It has a flexible interface which allows the user to apply random number generator (RNG) algorithms either in the CPU or the GPU. Furthermore, 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 [66] [67]. 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, which would be stored in host memory [65].

The CUDA toolkit includes another library named Thrust that is a C++ template. Thrust is a high-level interface that can imulate 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 [68].

There are general guideline principles for using a GPU-Based Monte Carlo simulation especially in a CUDA environment [29]. Not only is this method of simulation efficient for the present scenario, it is also very robust.

The following are some of the examples of kernel functions in the CURAND to calculate the floe encounter, floe diameter, and wind speed based on the Gamma, Exponential, and Weibull distribution respectively.

```
//Floe Encounter Rate/Year
__device__ user_data_t ran_gamma (curandStateMRG32k3a_t
*localState, const user_data_t a, const user_data_t b){

      int n = ceil(a);
      user_data_t sum = 0;
      for (int i = 0; i < n; i++)
      {
            user_data_t r = ran_exp(localState, b);
            sum += r;
      }
      return sum;
}
```

```
// Floe Diameter

__device__ user_data_t ran_exp(curandStateMRG32k3a_t
*localState, const user_data_t mean)
{

      user_data_t r = curand_uniform(localState);

      return (-log(r)*mean);
}
```

```
// Wind Speed

__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));

}
```

As mentioned earlier, level ice thickness, ridge thickness, and impact velocity are user-defined parameters that could be implemented by the CURAND library. The following is the implementation of level ice thickness on CUDA:

```
// Calculate Level Ice Thickness
__device__ user_data_t
UserDefineLevelIceThickness(curandStateMRG32k3a_t
*localState){

      user_data_t r = curand_uniform(localState);
      user_data_t res = 0.0;

      if (r >= 0.0&&r <= 0.102)
            res = RandomBetween(localState, 1.5, 2);
      else if (r > 0.102&&r <= 0.25)
            res = RandomBetween(localState, 2, 3);
      else if (r > 0.25&&r <= 0.561)
            res = RandomBetween(localState, 3, 4);
      else if (r > 0.561&&r <= 0.767)
            res = RandomBetween(localState, 4, 5);
      else if (r > 0.767&&r <= 0.891)
            res = RandomBetween(localState, 5, 7);
      else if (r > 0.891&&r <= 0.937)
            res = RandomBetween(localState, 7, 8);
      else if (r > 0.937&&r <= 0.973)
            res = RandomBetween(localState, 8, 9);
      else if (r > 0.973&&r <= 0.985)
            res = RandomBetween(localState, 9, 10);
      else if (r > 0.985&&r <= 0.997)
            res = RandomBetween(localState, 10, 11);
      else if (r > 0.997&&r <= 0.999)
            res = RandomBetween(localState, 11, 13);
      else if (r > 0.999&&r <= 1)
            res = RandomBetween(localState, 13, 14);

      return res;
}
```

The ***UserDefineLevelIceThickness*** function picks random numbers from specific intervals based on Table 3.2, which is defined for ice thickness. The following ***RandomBetween*** function is responsible for generating random numbers in specific intervals, using the CURAND library to accelerate the process of this simulation.

```
// Random Between

__device__ user_data_t RandomBetween(curandStateMRG32k3a_t
*localState, user_data_t smallNumber, user_data_t bigNumber)

{

      user_data_t diff = bigNumber - smallNumber;
      user_data_t u = curand_uniform(localState);

      return (u* diff) + smallNumber;
}
```

The following code shows how maximum annual force is calculated using the Monte Carlo simulation using the CUDA Thrust library on a single GPU. Random numbers are generated in parallel and data is stored on the GPU directly. Function evaluations and aggregation are done on the GPU using parallel constructs and highly GPU-optimized algorithms.

```
int main()
{
size_t N = 1,000,000;  //Number of years
thrust::device_vector<yearResult> maxAnnualForce(N);

thrust::counting_iterator<unsigned int> index(0);

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

4.5.3. CUDA advantages

One of the advantages of CUDA over a legacy GPGPU is that generic operations are mapped onto the graphics pipeline in CUDA memory access which makes CUDA more flexible. The low learning curve of CUDA makes life easier for those programmers who are familiar with standard programming languages such as C. Furthermore, it is not required to have knowledge about graphics API. CUDA does not hide the graphics API from the programmer's view, but it has been designed in both software and hardware to perform a different route to the computational resources. In CUDA, threads can access any memory location, while cooperating within each block. The threads also load data into shared memory. This operation results in significant bandwidth reduction and avoidance of performance penalty [69]. In summary, there are several key abstractions in CUDA which allow a programmer to easily and efficiently harness the computational power of the GPU for non-graphics applications. The first abstraction is that CUDA makes the programmer serve many lightweight threads, since over a trillion threads can be launched in the CUDA kernel simultaneously. For instance, the present scenario uses one thread for each year, which means one million threads are working in parallel to calculate the maximum annual force between sea ice and a vertical structure.

The negligible overhead associated with thread creation and switching permits the granularity parallelism to be very small without impacting performance. This results in a very simple data decomposition model and unique thread ID, which is used to access array elements and to perform computations on these elements. CUDA has a simple two-level

hierarchy of concurrent threads. Threads are batched together in blocks, such that the independence of thread, blocks gives the hardware great flexibility in scheduling threads, which results in excellent scaling using this simple execution model [70].

4.6. 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 [5]. 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. `RandomBetween` is one of the example of using CURAND device application programming interface (API) to generate pseudo random numbers using MRG32k3a, which is one of the high quality random number generator.

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 as discussed in section 4.5.2. *`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 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. The reason for choosing this GPU was that, the GPU Tesla K80 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, which will be explained in the next chapter.

CHAPTER 5

Results & Discussion

5.1. Performance Results

The present study used large scale data, a probabilistic framework and parallel computing environment to achieve a high speedup. There are three types of speedups: sub-linear, linear and super-linear as shown in **Figure 5.1** [74].

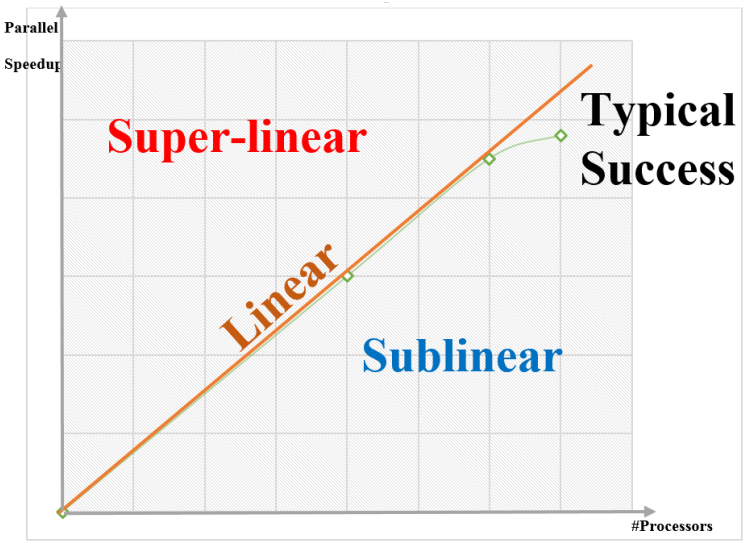

Figure 5.1: Speedup Diagram

There are some factors which might change the speedup of the implementation in this study. First, 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 [71]. However, using different ways of memory utilization causes a reduction in performance, because multiple GPUs distribute the application's data on among their processors. 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.

A comparison happens when there are optimized versions of both implementations on GPUs and CPU. Therefore, speedups are given for those optimized versions of implementations on both GPUs and CPU. **Figure 5.2** shows a speedup of up to 8 when comparing an optimized implementation of 4 GPU Tesla K80 against a single GPU Tesla K80. It also demonstrates a speedup of up to 130 for 4 Tesla K80 GPUs against an optimized CPU Open MP implementation.

Figure 5.3 shows that the elapsed time of different implementations is reduced from about 2.5 hours to 0.7 seconds. These implementations include 4 Tesla K80 GPUs, a single Tesla K80 GPU, serial CPUs, and a parallel CPU (Open MP). The computational time for 4 GPUs is approximately the same as 1 GPU for 10,000 years, 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.

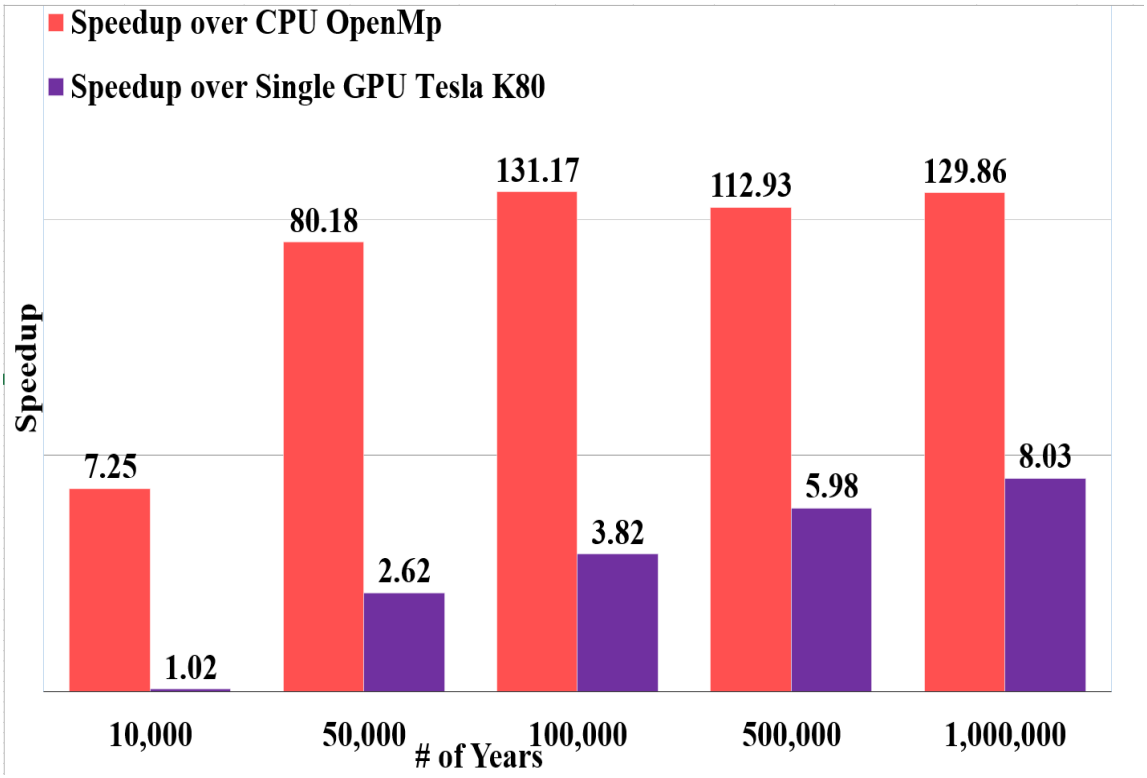

Figure 5.2: Speed Up Of Multi GPU over CPU Open MP and Single GPU

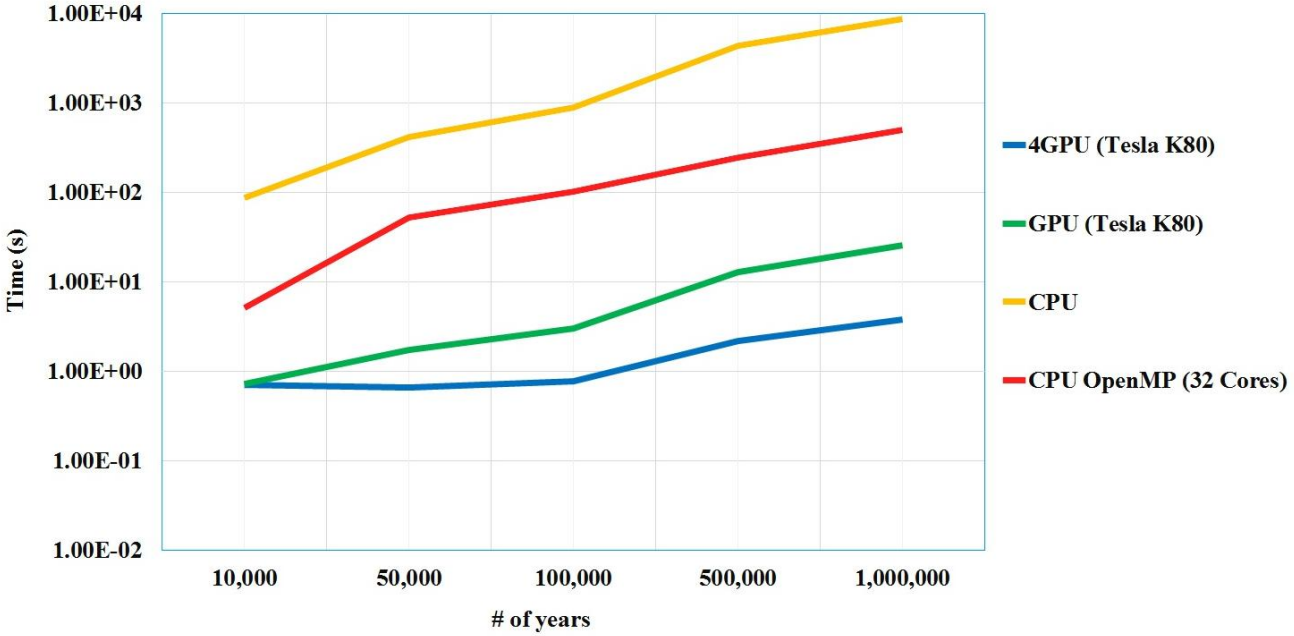

Figure 5.3: Elapsed Time for Different Implementations

5.2. Conclusion

In this report, the implementation of a model for determining the maximum annual force between sea ice and a vertical structure within a probabilistic framework is discussed. A Specific scenario with fixed and distributed parameters was selected for demonstrative purpose. Based on our assumption for ice parameters, significant differences in load calculation can result and these factors are not necessarily straightforward. In general, the

use of a probabilistic method such as the Monte-Carlo simulation, allows the engineering designer to understand these factors better [60] [70].

Load calculation between sea ice and an offshore structure is a major concern for marine engineers which needs further development and validation for other types of scenarios such as considering different types of sea ice types and structures. In some cases, there is considerable scatter, based on limited test data and the model's fit to environmental inputs.

Actual structures and sea ice in nature tend to have irregular and complex shapes and an assumption of their shape may not be sufficient and accurate. It would be very difficult but helpful to find full-scale information and investigate possible factors, such as impact of different types of sea ice on structures or the development of methods for ice load calculation.

However, a computer engineer focuses on available data and tries to find an appropriate simulation for each scenario. Therefore, finding efficient algorithms and numerical processes which are compatible with such large data like the Monte-Carlo simulation help to get more accurate result.

CHAPTER 6

Future Research

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 [71].

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 [72]. 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 [72].

If P = 1, it means we have 100 % efficiency. Therefore, improving the computational efficiency 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 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.

Bibliography

[1] http://www.cirrascale.com/industries_highperformancecomputing.aspx

[2] http://whatis.techtarget.com/definition/GPGPU-general-purpose-graphics-processing-unit

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

[4] Trujillo, Carlos, and Garcia-Sucerquia, Jorge. "Graphics processing units: more than the pathway to realistic video-games." Dyna 78.168 (2011): 164-172.

[5] Lee, Anthony, Yau, C., Giles, M. B., Doucet, A., & Holmes, C. C. "On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods." Journal of computational and graphical statistics 19.4 (2010): 769-789.

[6] Monahan, Adam Hugh. "The probability distribution of sea surface wind speeds. Part I: Theory and SeaWinds observations." Journal of Climate 19.4 (2006): 497-520.

[7] 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).

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

[9] "Code Optimization at HPC - High Performance Computing Cluster at CWRU." Crawford Hall. 2017.

[10] 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 News 38.3 (2010): 451-460.

[11] 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.

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

[13] 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.

[14] Alawneh, Shadi, Howell,Carl and Richard, Martin. "Fast quadratic discriminant analysis using GPGPU for sea ice forecasting." High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conference on Embedded Software and Systems (ICESS).

[15] Brandao, Diego, et al. "Performance evaluation of optimized implementations of finite difference method for wave propagation problems on GPU architecture." Computer

Architecture and High Performance Computing Workshops (SBAC-PADW), 2010 22nd International Symposium on. IEEE, 2010.

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

[17] 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 News 38.3 (2010): 451-460.

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

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

[20] “Monte Carlo simulation and its efficient implementation”. http://www.nag.com/Market.

[21] “Monte Carlo methods in CUDA”. http://www.thalesians.com

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

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

[24] Metropolis, Nicholas, et al. "Equation of state calculations by fast computing machines." The journal of chemical physics 21.6 (1953): 1087-1092. [25] R. H. Swendsen and J.-S. Wang. Nonuniversal critical dynamics in Monte

Carlo simulations. Physical Review Letters, 58(2):86–88, 1987

[26] Weisstein, Eric W. "NP-Problem." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/NP-Problem.html

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

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

[29] Sheppard, Andrew. "CUDA Accelerated Monte Carlo for HPC." Sc11, Seattle, WA (November 2011).

[30] CUDA accelerated Monte-Carlo for HPC. http://www.nvidia.com/docs/IO/116711/sc11-montecarlo.pdf.

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

[32] 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.

[33] 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.

[34] Thijssen, J., Fuglem, M., Muggeridge, K., Morrison, T., Spencer, P., et al. Update on probabilistic assessment of multi-year sea ice loads on vertical-faced structures. In OTC Arctic Technology Conference, Offshore Technology Conference (2015).

[35] Sea Ice.Retrieved from. http://www.eoearth.org/view/article/155931 (2012)

[36] "All About Sea Ice." National Snow and Ice Data Center. https://nsidc.org/cryosphere/seaice/characteristics/index.html

[37] Rothrock, D. A., and A. S. Thorndike. "Measuring the sea ice floe size distribution." Journal of Geophysical Research: Oceans 89.C4 (1984): 6477-6486.

[38] Palmer, Andrew, and Croasdale, Ken. “Arctic offshore engineering”. World Scientific, 2013.

[39] Polyak, Leonid, et al. "History of sea ice in the Arctic." Quaternary Science Reviews 29.15 (2010): 1757-1778.

[40] Gillis, Justin. "Ending Its Summer Melt, Arctic Sea Ice Sets a New Low That Leads to Warnings." The New York Times (2012).

[41] Zhang, Jinlun, and D. A. Rothrock. "Modeling global sea ice with a thickness and enthalpy distribution model in generalized curvilinear coordinates." Monthly Weather Review 131.5 (2003): 845-861.

[42] Rothrock, D. A., and A. S. Thorndike. "Measuring the sea ice floe size distribution." Journal of Geophysical Research: Oceans (1978–2012) 89.C4 (1984): 6477-6486.

[43] Kelln, D. E., P. Eng, and F. R. J. Martin. "Determination of Ice Thickness"." (1990).

[44] Weeks, Willy." On sea ice". University of Alaska Press, 2010.

[45]Data, Ice Extent. "National Snow and Ice Data Center." (2007).

[46]Wadhams, Peter. "How does Arctic sea ice form and decay?" May19.2008 (2003): 275-332.

[47] Faltinsen, Odd. "Sea loads on ships and offshore structures". Vol. 1. Cambridge university press, 1993.

[48] Fuglem, M., T. J. W. Richard, and J. Thijssen. "Challenges implementing iso 19906 for probabilistic assessment of multi-year sea ice loads on sloping structures." Icetech (2014): 14-154.

[49] Winsberg, Eric. "Computer Simulations in Science." Stanford Encyclopedia of Philosophy. Stanford University, 06 May 2013

[50] Kelton, W. David. "Simulation with ARENA". McGraw-Hill, 2002.

[51] https://images.nvidia.com/content/pdf/kepler/Tesla-K80-BoardSpec-07317-001-v05.pdf

[52] Foster, Ian. Designing and building parallel programs. Vol. 191. Reading: Addison Wesley Publishing Company, 1995.

[53] Simone de L. Martins, Celso C. Ribeiro,and Noemi Rodriguez," Parallel Computing Environments", http://www2.ic.uff.br/~celso/artigos/environments.pdf

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

[55] Hong, Sunpyo, and Hyesoon Kim. "Memory-level and thread-level parallelism aware GPU architecture performance analytical model." (2009).

[56] Kumar, Mahesh, Sapna Jain, and Snehalata. "Parallel Processing using multiple CPU".

[57] Hassan Mujtaba 2014. "NVIDIA introduces Tesla K80 with Two GK210 GPUs - world's fastest accelerator with 2.9 TFlops of compute".

[58] Corporation, N. 2015. "TESLA K80 GPU ACCELERATOR". Technical report.

[59] Sengupta, Shubhabrata, et al. "Scan primitives for GPU computing." Graphics hardware. Vol. 2007. 2007.

[60] NVIDIA Corporation 2016a. "CUDA FAQ".

[61] Greg Ruetsch, Brent Oste, Getting Started with CUDA, 2008 NVIDIA Corporation

[62] Ross, Gregory. "High performance histogramming on massively parallel processors." (2014).

[63] "CUDA Toolkit." NVIDIA Developer. N.p., 16 Feb. 2017. Web. 29 Mar. 2017.

[64] Cheng, John, Max Grossman, and Ty McKercher. Professional Cuda C Programming. John Wiley & Sons, 2014.

[65] NVIDIA Developer 2016. “Programming guide”.

[66] Gao, Shuang, and Gregory D. Peterson. "GPU accelerated scalable parallel random number generators." Proc. 2010 Symposium on Application Accelerators in High Performance Computing (SAAHPC’10). Vol. 76. 2010.

[67] Corporation, N. 2016b. “CURAND library”.

[68] Bell, Nathan, and Jared Hoberock. "Thrust: A productivity-oriented library for CUDA." GPU computing gems Jade edition 2 (2011): 359-371.

[69] NVIDIA Corporation 2016c. “NVIDIA on GPU computing and the difference between GPUs and CPUs”.

[70] Kumar, M., M. S. Jain, and M. Snehalata. 2014. “Parallel Processing using multiple CPU”.

[71] Shan, Jing. "Super linear Speedup in Parallel”. Computation." CCS, Northeastern Univ., Massachusetts, Course Report (2002).

[72] SciNet 2015. “Introduction to performance – SciNetWiki”

[73] Richard McKenna, Mark Fuglem, Greg Crocker,” Uncertainty in 100 and 10,000 Year Ice Loads on Offshore Structures, ICETECH14-167-RF.

[74] Sutter, Herb. "Super linearity and the bigger machine." (2008): 52-54.

[75] Sanders, Jason, and Edward Kandrot. CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents. Addison-Wesley Professional, 2010.

[76] Austin Post and Edward R. LaChapelle, "*Glacier Ice*"University of Washington Press, Seattle, 1971 and 2000.