GURU GHASIDAS VISHWAVIDYALAYA

BILASPUR, C.G.

(CENTRAL UNIVERSITY)

SESSION: 2016 – 2017

DEPARTMENT OF INFORMATION TECHNOLOGY, INSTITUTE OF TECHNOLOGY

A Project Report on:

Discrete Particles Simulation using OpenCL & OpenGL

by:

Arpitanshu [14107007]

Rishabh Khanna [14107035]

under supervision of:

Mr. Pankaj Chandra

.. 0 ..

Discrete Particles simulation using OpenCL & OpenGL

A PROJECT REPORT

Submitted in partial fulfillment of the requirement of

BACHELOR OF TECHNOLOGY
VIth SEMESTER
IN
INFORMATION TECHNOLOGY

Submitted By
Arpitanshu
&
Rishabh Khanna

Under the supervision of
Mr. Pankaj Chandra

TO

DEPARTMENT OF INFORMATION TECHNOLOGY
INSTITUTE OF TECHNOLOGY
GURU GHASIDAS VISHWAVIDYALAYA BILASPUR, INDIA
(CENTRAL UNIVERSITY)
SESSION: 2016 - 2017

DECLARATION

We hereby declare that the work presented in this dissertation entitled "*Discrete Particles Simulation using OpenCL & OpenGL*" has been done by us, and this dissertation embodies our own work.

_____

Arpitanshu

_____

Rishabh Khanna

Approved by:

_____ _____

Examiner **Head of Department**

Information Technology

DEPARTMENT OF INFORMATION TECHNOLOGY
INSTITUTE OF TECHNOLOGY,
GURU GHASIDAS VISHWAVIDYALAYA, BILASPUR
(CENTRAL UNIVERSITY)

CERTIFICATE

This is to certify that the project entitled

"**Discrete Particles Simulation using OpenCL & OpenGL**"

carried out by

Arpitanshu & **Rishabh Khanna**

under my supervision at

Dept. of Information Technology,

Institute of Technology,

Guru Ghasidas Vishwavidyalaya, Bilaspur.

To the best of my knowledge and belief the report

- Embodies the work of the candidates themselves.
- Has duly been completed.
- Fulfils the requirement of the B. Tech degree of the University

_____

(Signature of the Supervisor)

Mr. Pankaj Chandra

CERTIFICATE BY THE EXAMINERS

The Project Report entitled

'*Discrete Particles simulation using OpenCL &* OpenGL'

Submitted by

Arpitanshu (Roll No.: 14107007)

Rishabh Khanna (Roll No.: 14107035)

has been examined by the undersigned as a part of the examination and fulfills the requirement of

Bachelor of Technology

(VIth Semester)

Department of Information Technology,

Guru Ghasidas Vishwavidyalaya, BILASPUR.

_____ _____

(Signature)
Examiner 1

Name:

Designation:

Date:

(Signature)
Examiner 2

Name:

Designation:

Date:

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts with success. This acknowledgment transcends the reality of formality when we would like to express deep gratitude and respect to all those people behind the screen who guided, inspired and helped us for the completion of our project work.

We express our deepest gratitude to **Mr. Pankaj Chandra**, our *project guide* for his technical support and advice in completion of the project without which the project wouldn't be in the shape it is today, and for giving us an opportunity to do the project under his guidance.

We express our deepest gratitude to **Mr. Santosh Soni,** our *Head of Information Technology Department* for his moral support in completion of the project and for giving us an opportunity to create this project as part of the curriculum.

We also present sincere thanks to **Dr. Amit Khaskalam, Mr. Rajesh Mahule, Mr. Abhishek Jain, Mr. Agnivesh Pandey, Mr. Deepak Kant Neetam, Mr. Suhel Ahamed, Mr. Anand Prakash Rawal and Mrs. Akanksha Gupta**, who were instrumental in helping us in various ways and for providing inspiration through this year.

We are highly obliged to **Dr. Shailendra Kumar**, *Director of Institute of Technology* and **Mr. Mukesh Singh**, *our Dean* for their incredible support & providing all the facilities in the institute.

Last, but not the least, we present ineptness to our beloved Parents and Family member who were always there when we needed them to give their moral support and help by all the ways which were possible by them.

Preface

The main aim of the project is to demonstrate that computationally intensive problems that require huge computational power of a lots of CPUs can be mapped to readily available cheap commodity hardware such as a GPU or an Accelerator.

When we say heterogeneous computing, what we mean the ability to take advantage of all of the computer resources, with different ISAs, different memory models & different performance and power constraints between the CPU and GPU. Being able to program for both of those devices is intriguing.

GPGPU is becoming a popular choice in High Performance Computing. Hence the number of applications ported to GPGPU platforms like CUDA & OpenCL is high for more than a decade and it has proven its capabilities in delivering high performance in parallel applications.

Content

Chapter 1:
Overview of the Project

Discrete particle methods are relatively computationally intensive, which limits either the duration of a simulation or the number of particles. Several Discrete Particle methods, take advantage of parallel processing capabilities to scale up the number of particles and/or length of the simulation.

The main aim of the project is to demonstrate that computationally intensive problems that require huge computational power of a lots of CPUs can be mapped to readily available cheap commodity hardware such as a GPU or an Accelerator.

The performance scaling that can be achieved over by using just a CPU VS using a much cheaper vector processor like a GPU could be as much as 100X or even more, depending upon how efficiently we utilize the system resources.

In our simple demonstration, we achieved performance scaling of up to 10X on an embedded mobile GPU *Intel HD 5500*, and up to 25X on a discrete mobile GPU *NVIDIA GeForce GTX 860M*.

The problems we picked to demonstrate the Discrete Particles methods we

- N – Body Simulation
- Smooth Particles Hydrodynamics

In both these demonstrations, we started off with a number of discrete particles [8K, 16K, 24K, 36K, 48K and 64K] with some manually or randomly initialized positions and initial velocities to each of the particles in 3D space and let them move under the interaction forces of the described problems.

1.1:: N - Body Simulation Overview & Applications:

An N-body Simulation is a simulation of a dynamic system of particles usually under the influence of physical forces such as gravity or electrostatic forces.
N-body simulations are widely used in astrophysics:

- Investigating the dynamics of few body systems like the Earth-Moon-Sun system
- Understanding the evolution of large scale structures of the universe like formation of galaxies

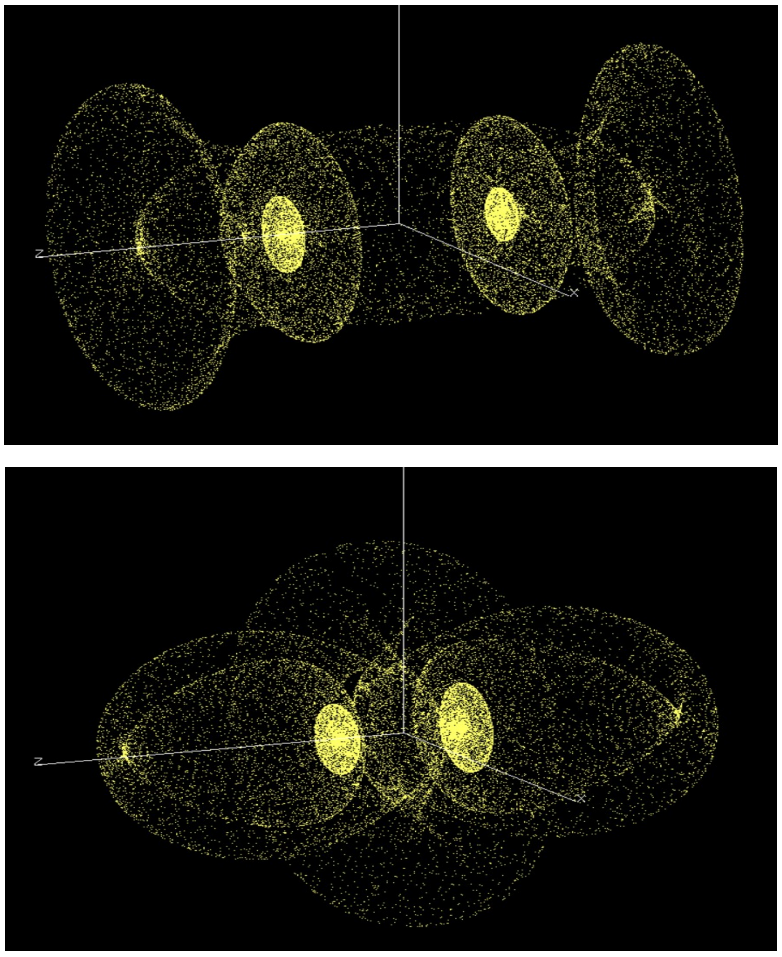


Figure 1: Screenshot of the N-Body Simulation with 32768 particles under Gravitational Force

1.2:: Smooth Particles Hydrodynamics Overview & Applications:

Smooth Particle Hydrodynamics (SPH) is a computational method which is used for simulating the dynamics of continuum medium, such as fluid flow.

The method was developed by Gingold & Monaghan [1977] and Lucy [1977] initially for astrophysical problems. However this method is versatile and can be applied to a variety of other fields such as:

- Ballistics
- Volcanology
- Oceanography

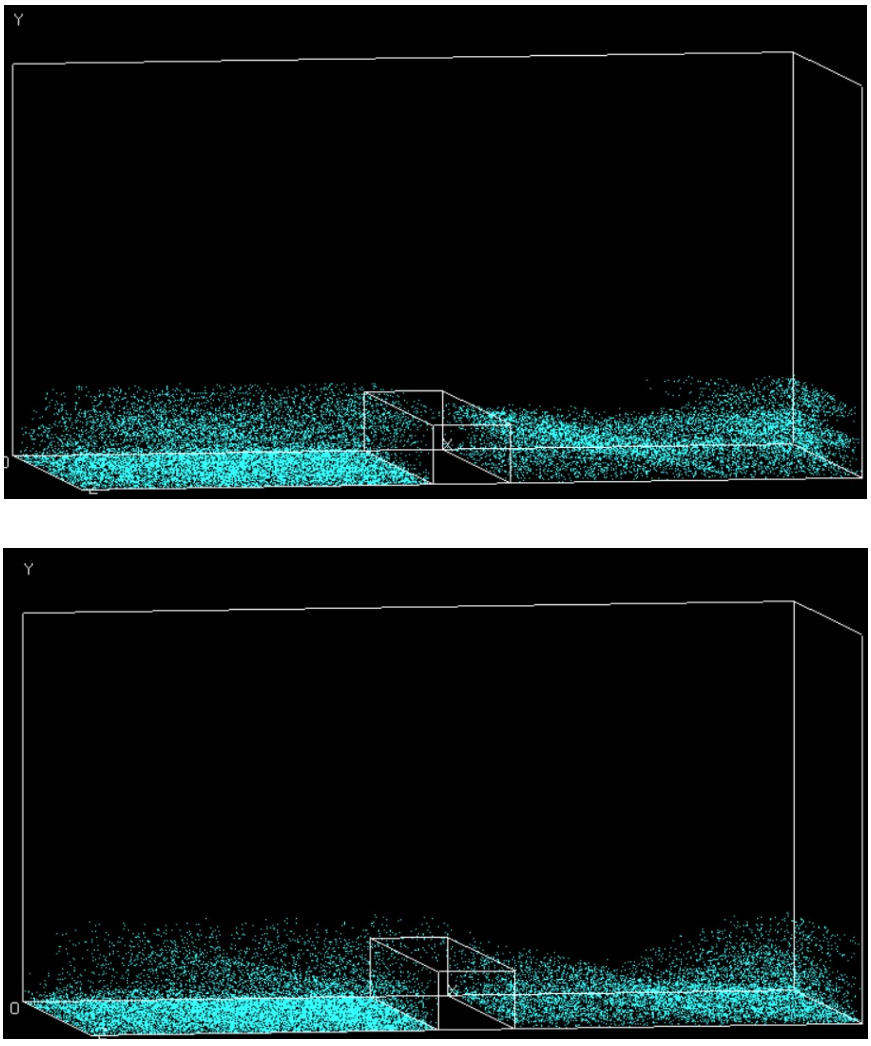


Figure 2 : Screenshot of the Fluid Motion Simulation with 16384 particles

Chapter 2:
Description of Tools and APIs used

Tools that were used in the project:

- OpenCL Framework
- FreeGLUT Toolkit implementing the OpenGL API

2.1:: OpenCL:

OpenCL is a framework for writing programs that execute across *'heterogeneous platforms'* consisting of CPUs, GPUs, DSPs or other hardware accelerators.

OpenCL provides:

- A Programming language *OpenCL C* (based on C99) for programming OpenCL compatible devices.
- An API to control the devices and execute programs on the compute devices.

It provides a standard interface for Parallel Computing using Task - Parallelism & Data – Parallelism.

2.1.1:: PLATFORM MODEL

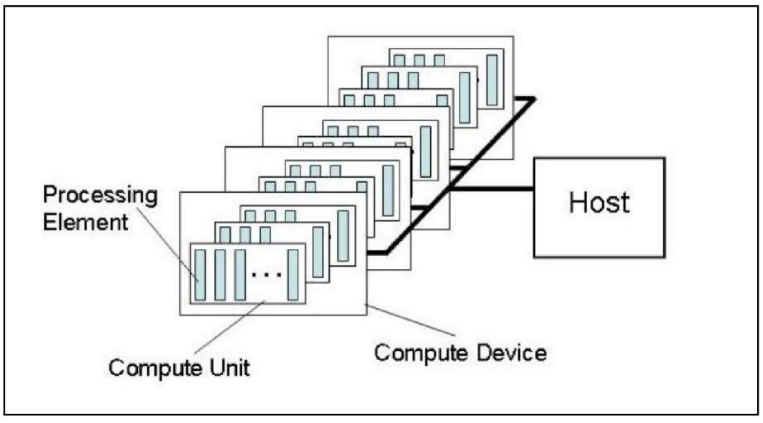


Figure 3 : OpenCL Platform Model: ref:: www.khronos.org

OpenCL views a computing system as a number of *'Compute Devices',* attached to a HOST processor (the CPU). A single Compute Device consist of a number of *'Compute Units'*, which in turn comprise of a number of *'Processing Elements'*.

How the Compute Device is divided into the Compute Units and the Processing Elements depends entirely on the vendor or manufacturer.

The functions executed on OpenCL devices are called '*kernels*'. A single kernel execution can run on all or many of the Processing Elements in parallel.

OpenCL defines an API that allows program running on the host to launch kernels on the OpenCL Compute Devices and manage the OpenCL device memory (hereafter referred to as '*Device Demory')*, which is separate from the '*Host Memory*' (i.e. RAM).

2.1.2:: MEMORY MODEL

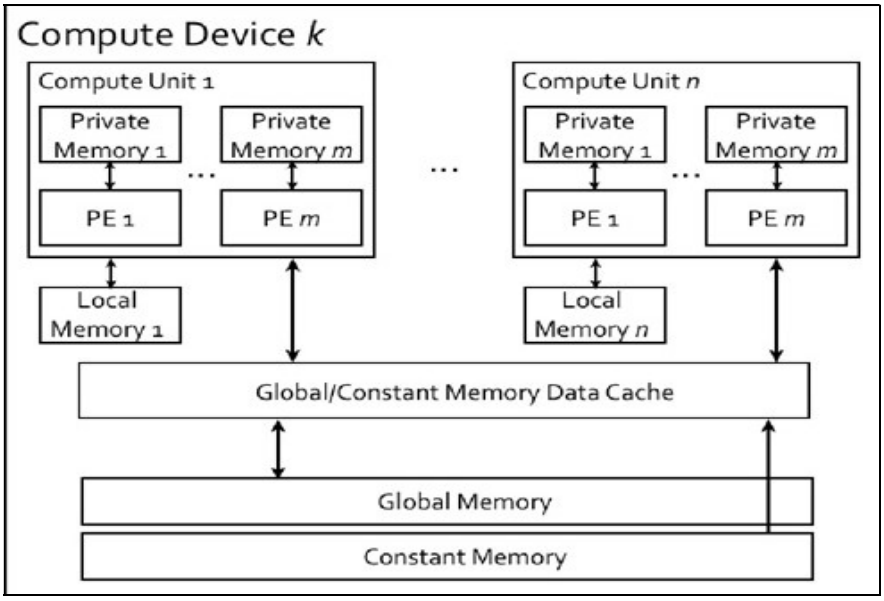
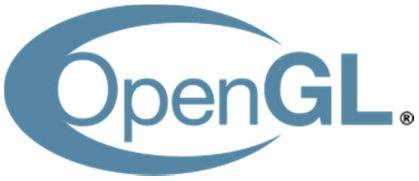


Figure 4 : OpenCL Memory Model: ref:: www.khronos.org

OpenCL defines a 4 – level memory hierarchy for the compute devices:

- Global Memory
  - Shared by all Processing Elements
  - High access Latency
- Constant Memory
  - Low Latency
  - Write access to only the host CPU, not the Compute Devices
- Local Memory
  - Shared by a group of Processing Elements
  - Very low latency
- Private Memory
  - Registers for the Processing Elements

2.2:: OpenGL:



OpenGL is a cross platform API for rendering 2D and 3D graphics. For the rendering parts in our project, we have used FreeGLUT, which is a toolkit implementing the OpenGL API.

FreeGLUT allowed us to create and manage windows containing OpenGL contexts on a wide range of platforms and also read the mouse and keyboard functions.

Chapter 3:
Description of Particle Algorithms

This section describes the *parameters fed to the system*, the *equations* and the *algorithms* that drive the particles motion in 3D space.

3.1:: N-Body Algorithm & Equations

The N-Body problem simulates the evolution of the system, where the force exerted on each body arises due to its interaction with all other bodies in the system.
The simulation proceeds over timesteps, each time computing the net force on every body, thereby its acceleration. The acceleration is integrated over each timestep to add to its velocity following its new position.

All pairwise forces are computed directly and this requires $O(N^2)$ operations at each timestep.

The n-body problem considers **n** point masses, each of mass $\mathbf{m}_i$ (i = 0, 1, 2… n), in a three dimensional space moving under the influence of mutual gravitational attraction. Each particle has a position vector **'$\mathbf{q}_i$'**.
Following Newton's Second Law, mass * acceleration $m_i\, d^2\mathbf{q}_i/dt^2$ is equal to the sum of the forces on the mass. The gravitational attraction felt on mass m_i by a single mass m_j is given by:

$$\mathbf{A_{ij} = Gm_j\,(q_j - q_i) / (q_j - q_i)^3} \quad \ldots \quad \text{equ}^n(1)$$

$$\mathbf{q_{i+1} = q_i + (v_i*dt) + (A_{ij}*dt^2/2)} \quad \ldots \quad \text{equ}^n(2)$$

$$\mathbf{v_{i+1} = v_i + A_{ij}*dt} \quad \ldots \quad \text{equ}^n(3)$$

After figuring out the net acceleration on each particle, we use the **integrator** to calculate $\mathbf{v_i}$ and $\mathbf{q_{i+1}}$ over timesteps **dt**, using equ^{n}s 2 & 3.

ALGORITHM:

For each particle i, do:

1. Cache in initial position and velocity from global buffers.
2. For each other particle, particle 'j':
   2.1. Calculate the acceleration due to this particle j on the particle i.
   2.2. Add to net acceleration on particle i, using equ^n(1).
   2.3. Wait for other threads to finish updating their accelerations.
3. Timestep integrate over **dt** to get q_{i+1} & v_{i+1}, using equ^{n}s 2 & 3.
4. Update new values in global buffers.

3.2:: SPH Algorithm & Equations

The SPH equations describes below is an implementation of SPH method describes by Muller et al for use in interactive graphics.

Each computational particle carries information about the fluid in a little region, such as the velocity and density; and during the simulation, these particles interact with each other in a way that models the dynamics of a fluid.
Each particle **i** is initialized with a position $\mathbf{r_i}$, a velocity $\mathbf{v_i}$. Particle i interacts with the set $\mathbf{N_i}$ of particles within interaction radius **h** of particle **i**, to calculate its local density.
The density is computed at each step by:

$$\rho_i = C_0 * {}^{j=N_i}\Sigma_{j=0} [h^2 - r_j^2]^3 \quad \ldots \quad \text{equ}^n(1)$$

$$A_{ij} = A_{ij+1} + (W_p + W_v) * q_{ij} \quad \ldots \quad \text{equ}^n(2)$$

Where $C_0 = 4m_i/\Pi h^8$
$q = r_j - r_i / h$
$u = 1 - (r_j - r_i / h)$
$W_o = C_0 * (u / \rho_i / \rho_j);$
$W_p = W_o * C_p * (\rho_i + \rho_j - (2 * \rho_0)) * (u / q);$
$W_v = W_o * C_v;$
ρ_0 = reference density of fluid
C_p = Pressure Term Constant
C_v = Viscosity Term Constant

The integration scheme followed is frequently used in particle simulation algorithms, as it is explicit, which makes it simple to code.
It is named *LeapFrog Time Integration*, as the velocities are updated on half steps and the positions on integer steps.
Hence, these two leap over each other. After computing accelerations, one step takes the form

$$v_{i+1/2} = v_{i-1/2} + a_i \Delta t \quad \ldots \quad \text{equ}^n(3)$$

$$r_{i+1} = r_i + v_{i+1/2} \Delta t \quad \ldots \quad \text{equ}^n(4)$$

After performing equ^{n}s 3 & 4, we will have the updated positions and velocities for each particle mimicking fluid behaviour as per the parameters of the fluid provided.

Chapter 4:
DFD for Simulation System

This section describes the flow of data through the system.
In general, the 2 algorithms follow a common scheme of how data flows between Host & Device Memory in the system.

4.1:: System DFD

The system has 2 main Modules namely *OCL* & *OGL*. How these modules bring data back & forth between the Host & Device momory is described in the following System DFD.

The OCL module first initializes the Device Memory by bringing in initialized buffers from the Host Memory. It then initiates the Kernel to run on the OpenCL device and compute upon it. The results are then brought back to the Host Memory by the module which is used for rendering.

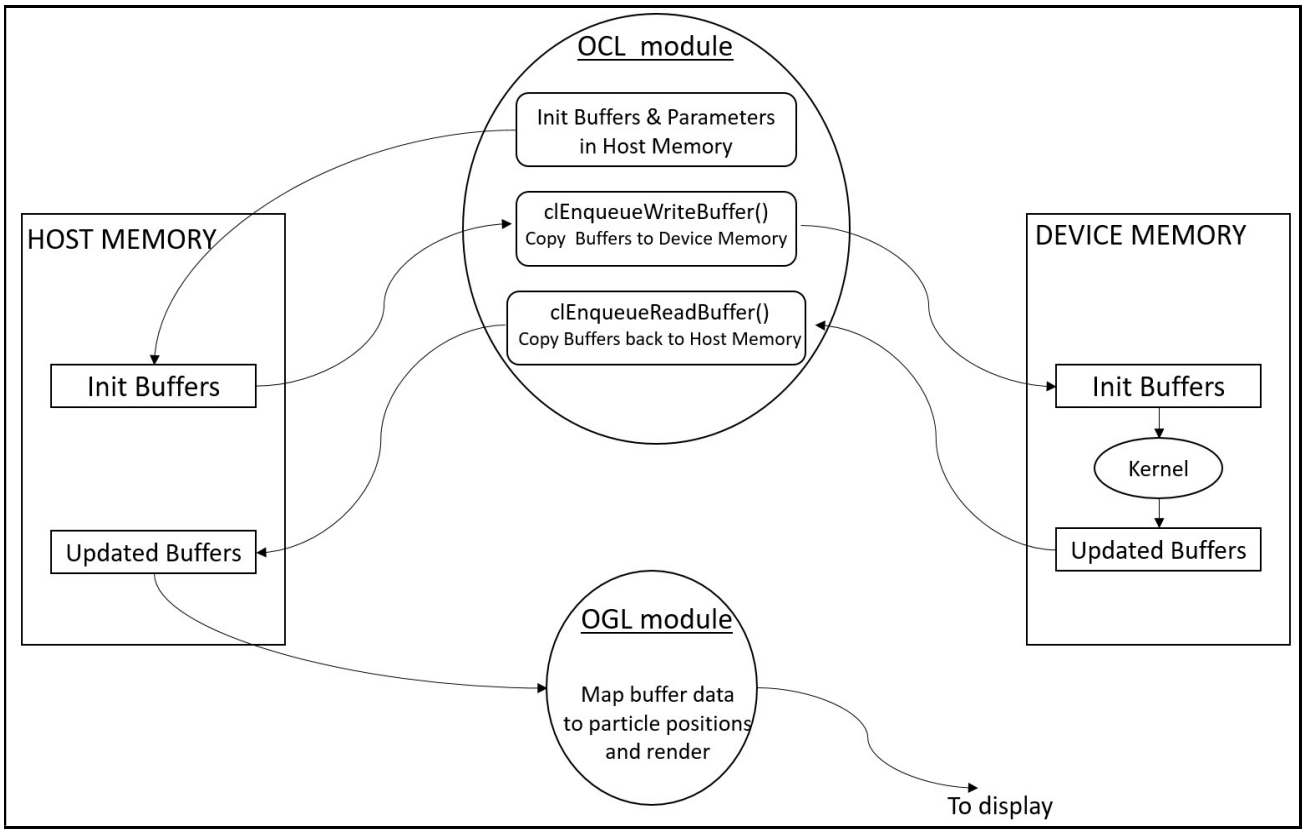


Figure 5 : Simulation System Data Flow Diagram

DFD for how data is manipulated in a Data Parallel manner in the Device Memory.

Data Parallelism is a form of parallelization across multiple processors in parallel computing environments. It focuses on distributing the data across different nodes, which operate on the data in parallel.

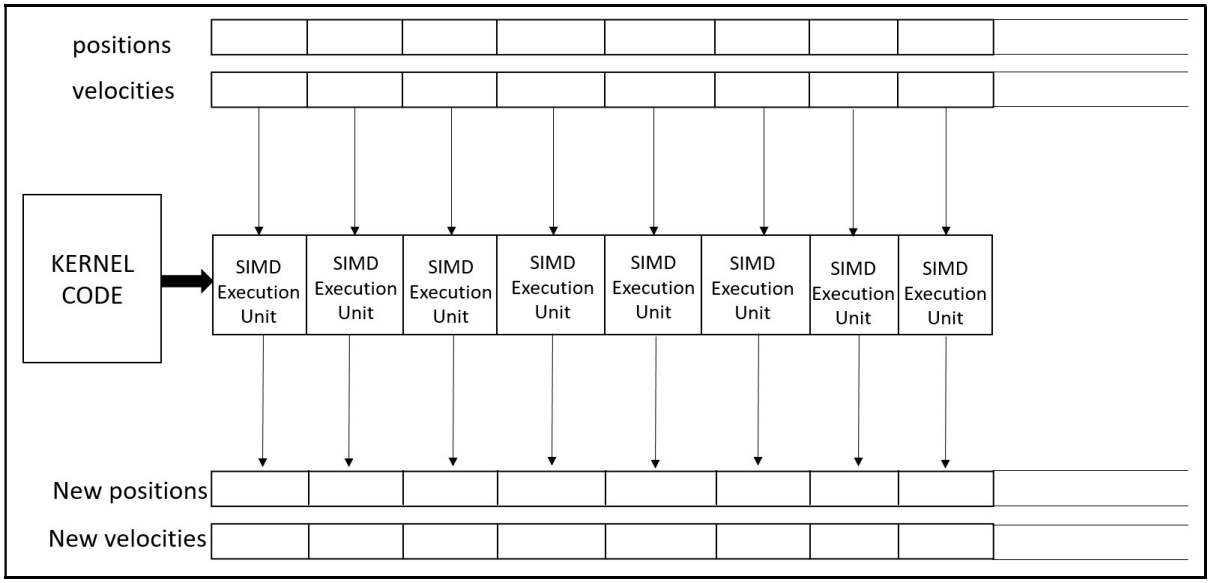


Figure 6 : Data Parallel model for Kernel Execution

4.2:: N-Body Kernel DFD

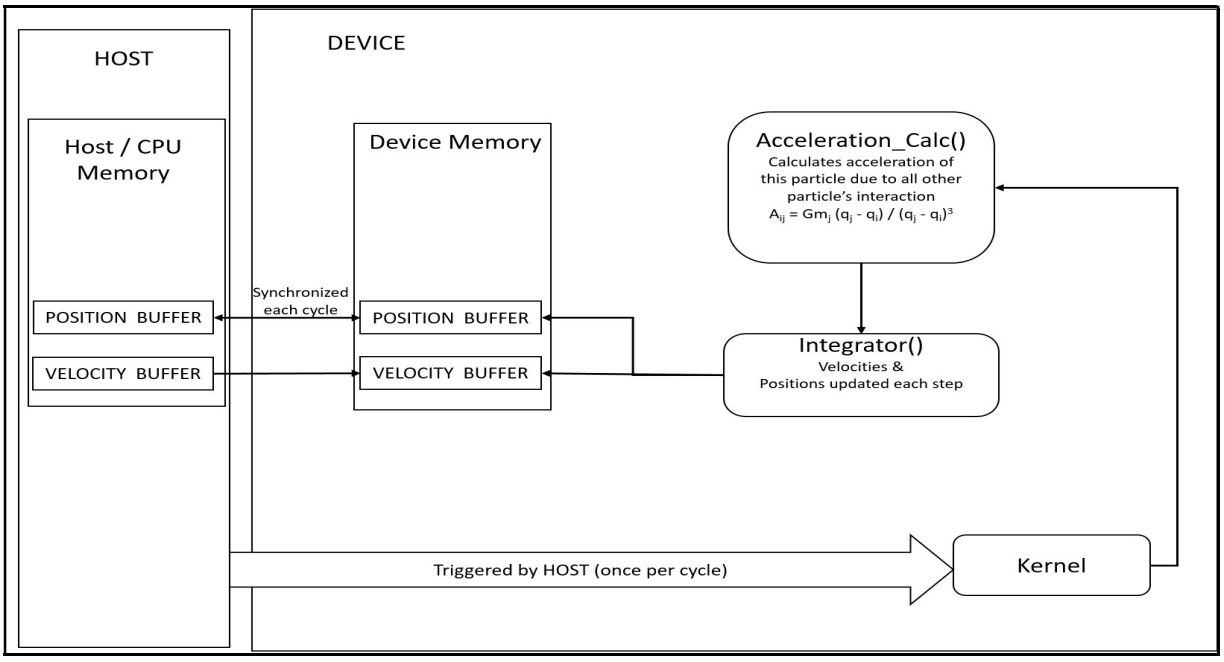


Figure 7 : N-Body Kernel Data Flow Diagram

4.3:: Smooth Particles Hydrodynamics Kernel DFD

The Kernel is divided into 3 modules. The first named Acceleration_Calc() is responsible for computing the density, followed by the acceleration. The second module, named LeapFrog_Integrator() integrates the result of the first module into velocities and positions vectors. The third module, named Boundary_Reflect(), performes a reflection operation, in which, if a particle, if tries to escape beyond the boundaries of the container or of an obstacle, is reflected back with a damped velocity.

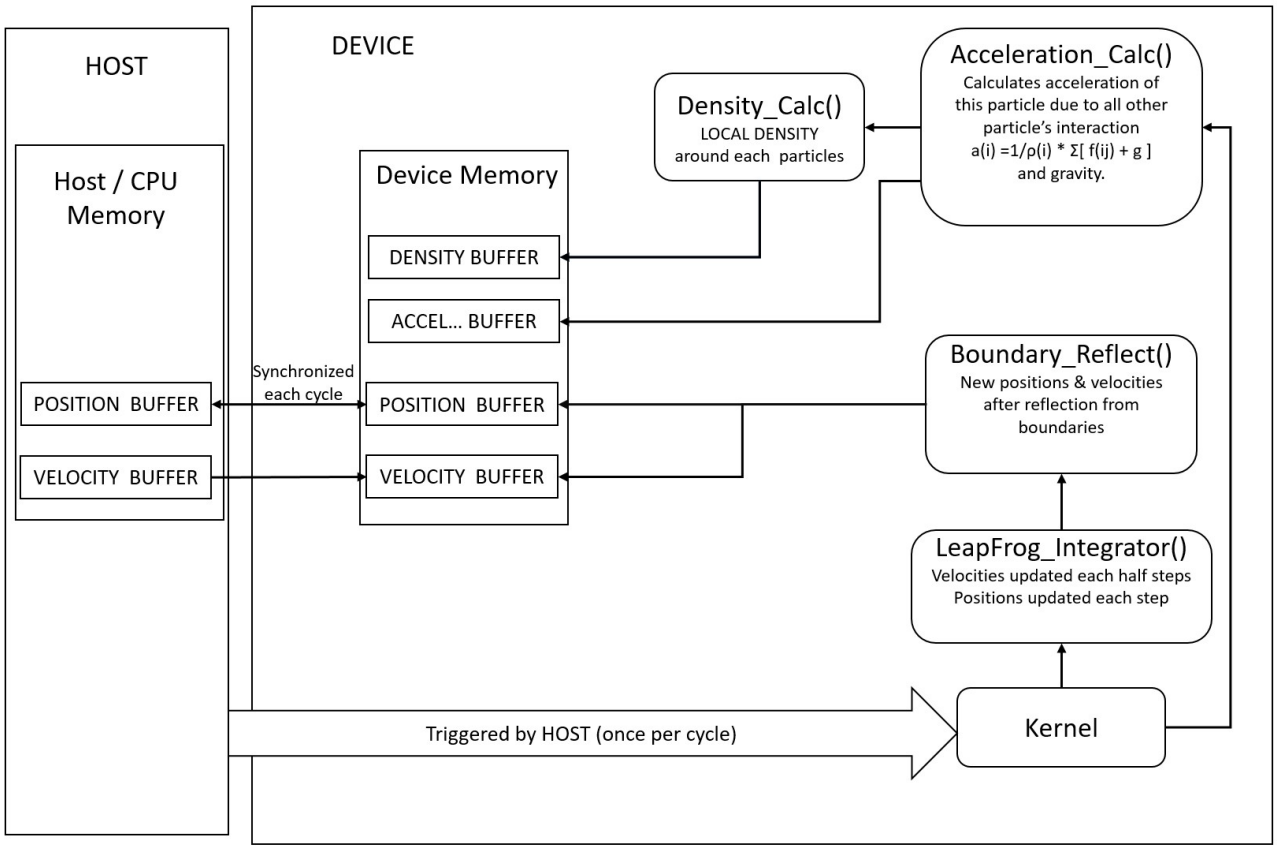


Figure 8 : SPH Kernel Data Flow Diagram

Chapter 5:
Technical Specifications of Test Run Devices

The specifications of the devices on which we performed our test runs are as follows:

Device Name	**Intel HD Graphics 5500**	**Nvidia GeForce GTX 860M**	**Intel Core i5 5200U**
Max Clock Frequency	900 MHz	1029 MHz	2200 MHz
No. of Compute Units	24	640	4
Global Memory	1488 MB	2047 MB	2047 MB
Global Memory Cache	576 KB	1024 KB	256 KB
Local Memory	64 KB	32 KB	64 KB
Max Workgroup Size	256	1024	8192

Table 1 : Specifications w.r.t OpenCL of our Test Run Devices

Test Run Conditions:

The result of the test runs is described in the next section. Now we describe the conditions under which the test were performed.

The test involved running the 2 simulations one at a time on one of the selected device. The Host code ran over the CPU at all times while the Device code was directed to one of the Test Run devices.

To check with the number of frames i.e. the number of cycles of calculations of kernel and rendering them on-screen completed per second, we used an open-source software "*Fraps*". *Fraps* enabled us to view the no of frames of the simulation generated per second during the real time run of our simulation.

Chapter 6:
Performance Comparison

Here we present the performance comparison of our simulation when running in real time over our test devices:

1. ***Intel HD 5500*** with **24** Execution Units
2. ***Nvidia GeForce GTX* 860M** with **640** Execution Units
3. **Intel Core i5 5200U** with **4** Execution Units

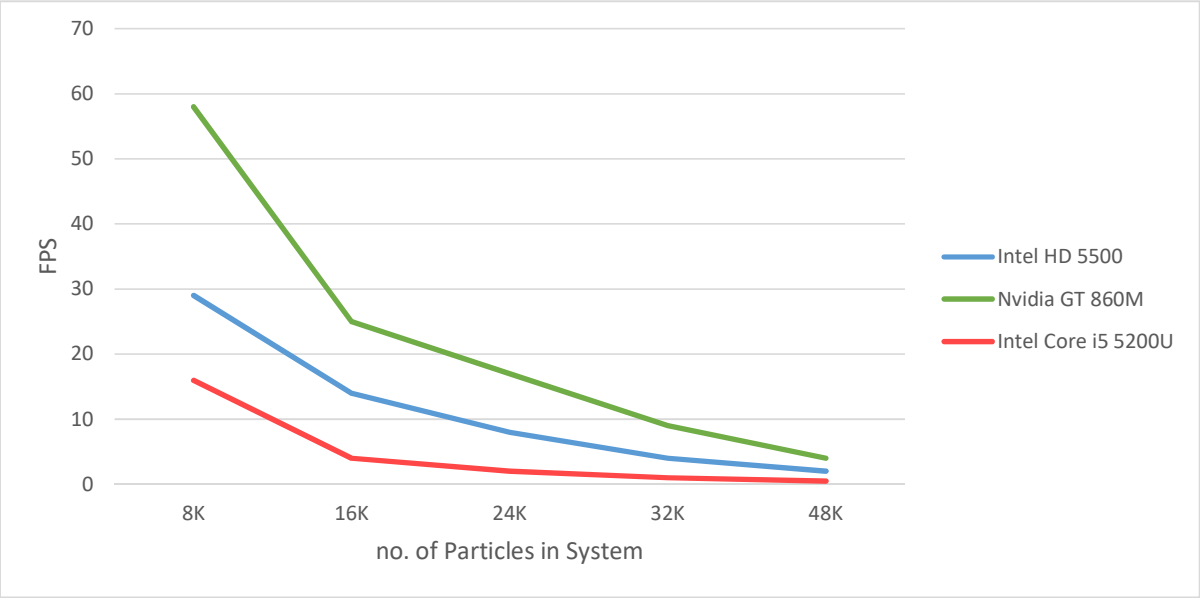


Figure 9 : Result of SPH Simulation on Various Test Devices: FPS vs number of particles

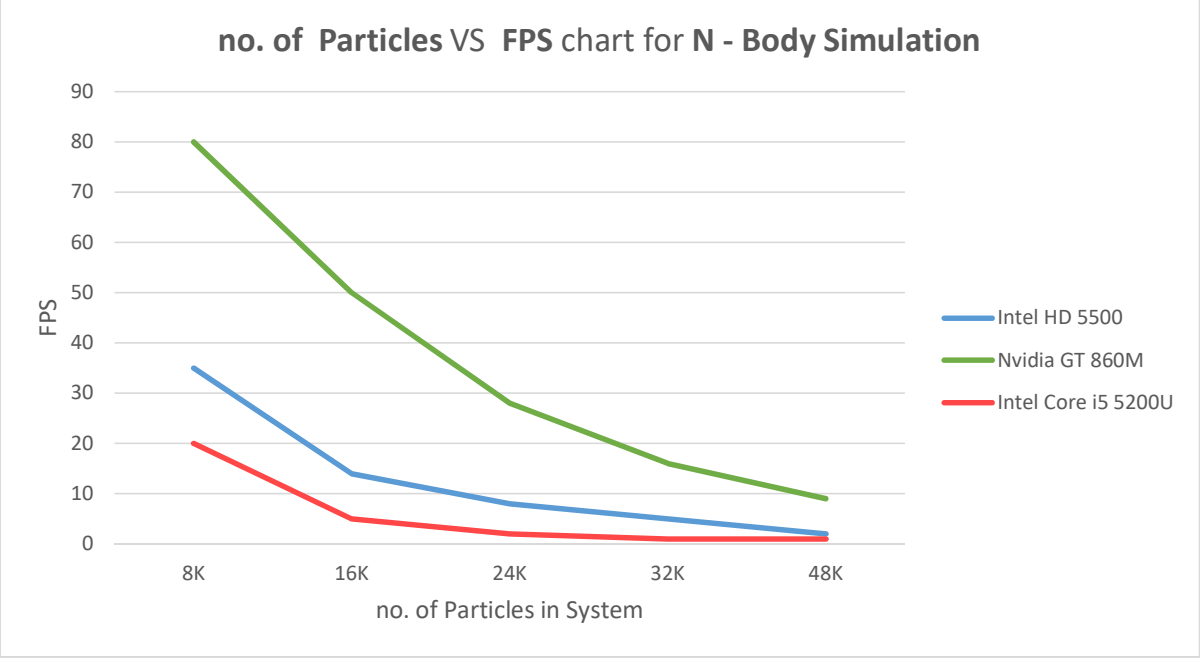


Figure 10 : Result of N-Body Simulation on Various Test Devices: FPS vs number of particles

Chapter 7:
References & Tools

References:

I. M. Muller, D. Charypar, and M. Gross.
Particle-based fluid simulation for interactive applications, in Proceedings of Eurographics/SIGGRAPH Symposium on Computer Animation.

II. OpenCL specifications - Khronos Group
https://www.khronos.org/opencl
https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf

III. OpenGL specifications
https://www.opengl.org/
https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/

IV. AMD Developer Central
- OpenCL Tutorials
https://www.youtube.com/channel/UCaP6OOB1bukhYFLyZyviU1A

V. David Parker YouTube Channel
- OpenGL Screencasts and Tutorials
https://www.youtube.com/playlist?list=PL2330214740B33712

Tools:

I. FreeGLUT – The Free OpenGL Utility Toolkit
http://freeglut.sourceforge.net/
FreeGlut API - http://freeglut.sourceforge.net/docs/api.php

II. Code::Blocks – Open Source, Cross Platform IDE
http://www.codeblocks.org/

III. FRAPS – Real Time Video Capture & Benchmarking
http://www.fraps.com/