Parallelization of a Discrete Element Method (DEM) algorithm using the Graphical Processing Unit (GPU)

Mario Guillaume Cecile and Michel Roddy Lollchund*
Department of Physics, University of Mauritius
Réduit, Rep. of Mauritius
*r.lollchund@uom.ac.mu

Abstract—**This paper presents a GPU-based parallel implementation of a Discrete Element Method (DEM) algorithm. The technique is applied in the simulation of free-falling particles in a rectangular bed. The effects on computation time for different number of particles are compared for the cases of performing the calculations using the GPU and the CPU (Central Processing Unit). It is found that the GPU provides a subsequent speed-up as compared to the CPU as the number of particles is increased.**

Keywords- Graphical Processing Unit (GPU); Discrete Element Method (DEM); Rectangular bed; Free-Falling Particles

I. INTRODUCTION

The Discrete Element Method (DEM), developed by Cundall in 1971, is a computational tool which is used to compute the displacement of each individual element in a system of moving particles. The method takes into account the translational and rotational motions of the particles as well as the contact forces (collision, friction, etc.) between them by numerically solving the corresponding governing equations. Two distinct techniques have been proposed in literature to model the interaction between the particles in DEM: the soft-sphere and hard-sphere approaches. The soft-sphere approach, developed by Cundall & Strack (1979), models a particle that can deform during collision while the hard-sphere approach, established by Hoomans *et al*. (1996), assumes that a particle is perfectly rigid. The hard-sphere approach treats solid collisions as binary and does not delve into the detailed dynamics of the inter-particle collisions (Singh, 2007). This method is generally deterministic; with the same initial conditions the same final result will be obtained. It is generally used to model dilute and energetic granular flows. The soft-sphere approach is more flexible, but is however more time consuming due to small time integration time steps (Wassgren, 2013). It assumes that the particles can overlap during collisions.

Nowadays, DEM has become accepted as a powerful method to tackle problems involving the flow of granular and discontinuous materials in many fields, including process engineering, mining, chemical industry with particulate reaction engineering, storage of grain flows and geophysics. It is recognized as an effective method to study the fundamentals of granular materials. The particles may have different shape or size though the computation is not as straightforward as in the case of circular or spherical particles. There are many studies reported in literature which detailed examples of where DEM have been employed. They range from simple applications to more complex industrial processes such as hopper discharge (Langston *et al*., 1995, 1996 and 1997). For instance, Pinson (2005) and Zhou *et al*. (2005), demonstrated the application of DEM in industrial processes using only spherical particles. Other researchers have applied the method using elements of polygonal shape (Heuze *et al*., 1993; Kun & Herrmann, 1996; Bolander & Saito, 1997; Camborde *et al*., 2000; Prochazka, 2004). However, large-scale DEM simulations are relatively computational intensive, which limits either the length of a simulation or the number of particles. This paper details the advantages of using the Graphical Processing Unit (GPU) to enhance a soft-sphere DEM code in view of scaling up the number of particles or length of the simulation. The paper is organized as follows: Section II provides an outline of the model and section III discusses briefly about the GPU architecture and how to run a code on it. Results obtained and analysis are presented in section IV while section V gives the conclusion of this work.

II. DESCRIPTION OF THE MODEL

A. The system considered

Fig. 1 illustrates a schematic representation of the granular flow system studied in this work. It consists of a two-dimensional rectangular bed of width W=0.2 m and complete height H=1.1 m.

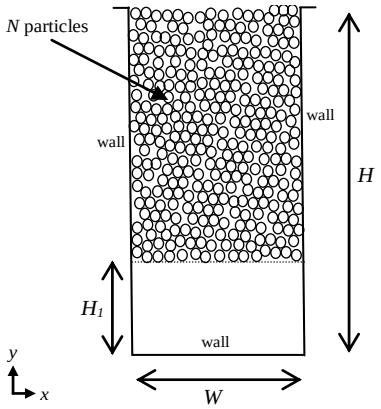


Figure 1. Schematic of the granular flow system.

N particles are initially randomly placed at H_1=0.2 m and above from the bottom wall. They are then allowed to fall freely under gravity. The particles are assumed to be spherical in shape (diameter d_p=5 mm) and made up of plastic (density=1080 kg/m^3). In short, a defined number of particles is randomly positioned in space and is allowed to fall freely in a rectangular bed. The particles are then expected to pile up in the bed after some time.

B. *Theoretical treatments of contacts*

Here, a dashpot-spring model is assumed for the contact between any two particles as well as between a particle and wall as illustrated in Fig. 2. A spring k_n and a dashpot c_n are included to control normal motion between the two particles while a spring k_s and a coefficient of friction c_s control the tangential motion.

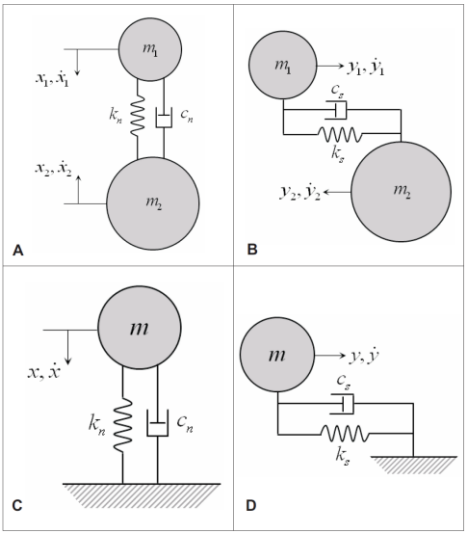


Figure 2. Illustration of the dashpot-spring soft-sphere contact model (Morisson and Wu, 2007). (A) Particle-to-particle contact in the normal direction, (B) particle-to-particle contact in the tangent direction, (C) particle-to-wall contact in the normal direction, (D) particle-to-wall contact in the tangential direction.

C. *The DEM algorithm*

In a DEM algorithm, particle trajectories are computed by considering the various forces acting on each particle. Let us consider a particle i with radius r_i, mass m_i and moment of inertia I_i. First, the number of nearest neighbor (M_i) of particle i is calculated to evaluate the number of contact pairs so as to reduce the computational requirement for the simulation. That is only the nearest neighbors will interact with the particle i during a given time step. A particle j is assumed to be a nearest neighbor of particle i if the distance between their centers is less than $2.5r_i$ (Singh, 2007). By then applying the laws of conservation of linear momentum and angular momentum to each particle with respect to its nearest neighbors and the wall boundaries the resultant force on each particle is obtained. These conservation laws are given as

$$m_i \frac{d\vec{v}_i}{dt} = \sum_{j=1}^{M_i} \left(\vec{F}_{c_{i,j}} + \vec{F}_{D_{i,j}}\right) + m_i\vec{g}, \qquad (1)$$

$$I_i \frac{d\vec{\omega}_i}{dt} = \sum_{j=1}^{M_i} \vec{\tau}_{i,j}, \qquad (2)$$

where $\vec{v}_i$ and $\vec{\omega}_i$ are respectively the translational and angular velocities of particle i. The contact and damping forces are respectively given as

$$\vec{F}_{c_{i,j}} = -k_n\delta_{i,j}\big|_n - k_s\delta_{i,j}\big|_s \qquad (3)$$

and

$$\vec{F}_{D_{i,j}} = -C_n\vec{v}_{i,j}\big|_n - C_s\vec{\omega}_{i,j}\big|_s, \qquad (4)$$

where $\delta_{i,j}$ is the separation distance, $\vec{v}_{i,j}$ and $\vec{\omega}_{i,j}$ are respectively the relative linear velocity and relative angular velocity between particles *i* and *j*. The contact forces are the forces that resist the inter-particle overlap due to collision. The term $m_i\vec{g}$ in equation (1) is the gravitational force and $\vec{\tau}_{i,j}$ in equation (2) is the torque on particle *i* due to particle *j*.

III. THE GRAPHICAL PROCESSING UNIT (GPU)

In 2003, Moreland & Angel recognized the ability of the GPU in performing large amount of computation in parallel. They were able to demonstrate the speedup obtained by implementing the Fast Fourier Transform (FFT) on the GPU as compared to the CPU. Since then, various research papers have been published on the application of GPU in scientific computing (Adinetz & Berezin, 2006; Anderson *et al*., 2007, Polyakov *et al*., 2013). In this section, the structure of the GPU is briefly discussed as well as the CUDA programming model which details how computations can be transferred from the CPU to the GPU.

A. *Architecture*

The GPU architecture is composed of more ALU's than a CPU on the same die space. As a result, they can do large amounts of mathematical calculations in a greater quantity compared to the CPU (Owens *et al.*, 2008). Fig. 3 depicts the architecture of a modern GPU. All the cores are connected to a very high bandwidth memory bus which allows the full use of the parallel processors. The CUDA architecture allows the GPU to execute the device code without intervention of the CPU. The latter is free to perform its task while waiting for the GPU to finish executing the Kernel. Therefore the GPU is not limited by the speed of the CPU (Nolan, 2009).

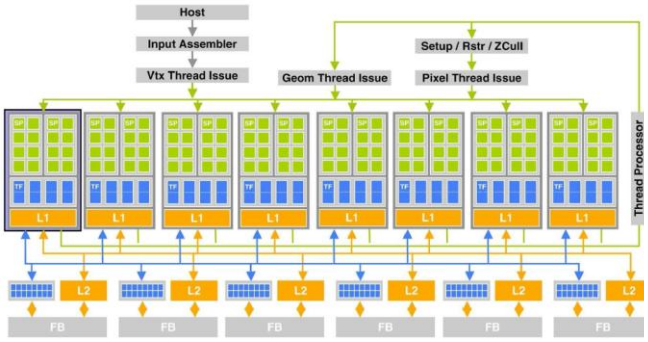


Figure 3. Architecture of a modern GPU (Kirk *et al.*, 2010).

B. *The CUDA programming model*

A CUDA program basically consists of parts that are executed on the host (CPU) and the device (GPU). Parts that show little or no parallelism are run on the CPU and parts that are rich in data parallelism are implemented on the GPU. The DEM algorithm employed in this work is coded in C++. Therefore, the programming for both the host code and the device code is in C++ with some keywords in the device code for indicating the special functions run by the GPU. The device code is called kernel. It generates a large number of threads to perform data parallelism. A typical CUDA program starts on the host which runs through the code. Before reaching a kernel, variables that will be used on the device are defined and memory is then allocated on the device. The host copies the variables to the device memory. The CUDA kernel is then launched.

A large number of threads are then generated on the device; the kernel is executed in parallel and all the threads are terminated at the end. Then, execution resumes on the host which copies the results of the kernel back to the host memory. The memory allocation on the GPU is cleared and host code can resume (Sanders & Kandrot, 2011). The scheme is illustrated in Fig. 4:

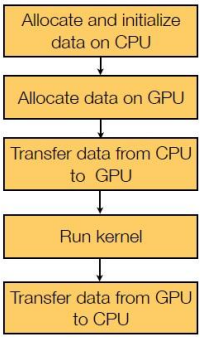


Figure 4. CUDA program execution (Cooper, 2011).

C. *Optimization strategy*

Before attempting to parallelize a particular code it is important to identify the part(s) that takes the most of the CPU execution time. A sequential version of the DEM code was written in ANSI C and has proven to give reliable results (Chamroo, 2011). The code contains three main functions which are described in table 1.

TABLE I. NAME AND TASK PERFORMED BY THE MAIN FUNCTIONS IN THE CODE

Function name	Task performed
force_calculation()	This function calculates the forces between the particles and updates the positions of the particle.
reset_calculation()	This function keeps a record of the numbers of nearest neighbors of each particle before calculating the nearest neighbor once again.
nearest_neighbour()	This function calculates the number of nearest neighbors of each particle.

By measuring the execution time of each function, the one which is the most computational intensive is identified. This function must be parallelized in order to obtain a good speedup. Fig. 5 shows the time taken for executing the three functions for 1 particle with different values of *N*. It can be deduced that nearest_neighbour() consumes most of the CPU execution time. Hence we will focus on the parallelization of this function using the GPU.

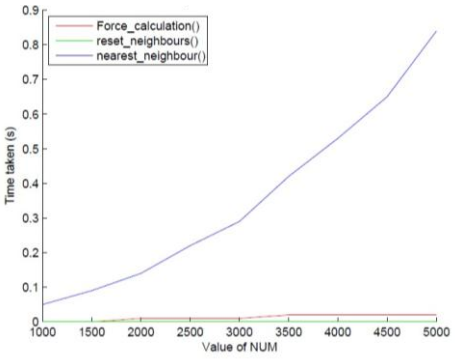


Figure 5. Execution time of each function for 1 particle on the CPU for different values of *N*.

Fig. 6 depicts a task dependency graph of the function nearest_neighbour(). It can be observed that all tasks are done sequentially and no tasks are independent of each other. However, each tasks from 1 to 6 can be done simultaneously, that is one processor can consider particle *i*, another processor can consider particle *i*+1 and each can perform the tasks 1 to 6 sequentially on the particle considered. Hence parallelization is achieved.

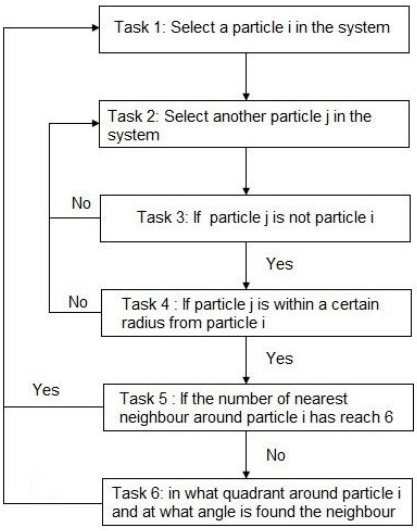


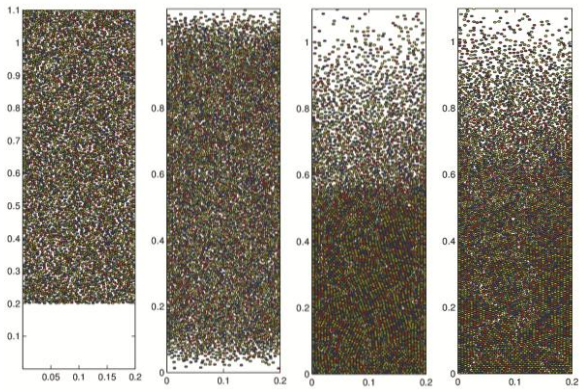
Figure 6. Task dependency graph of function nearest_neighbour().

CUDA allows the generation of large numbers of threads which are executed concurrently on the GPU. Since we are dealing with large number of particles, using the massively parallel architecture of the GPU using CUDA could help in enhancing the simulation. Hence, each thread can be assigned to work out the nearest neighbors for a particular particle *i*,

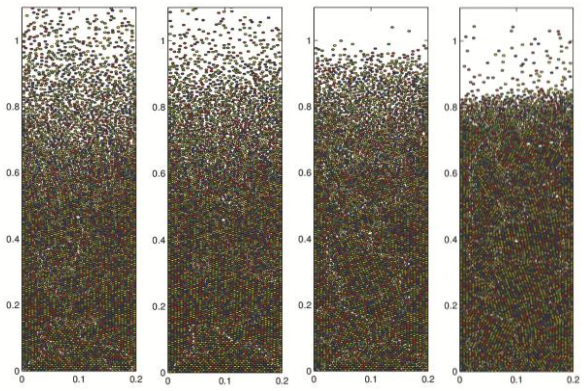
where *i* is equal to the index of the particular thread. Since each thread has a unique index, the 'for loop' is efficiently parallelized. Furthermore *i* for each thread is incremented by the total number of threads launched to avoid redundant computation. To obtain a good speedup, the idea is to launch as much threads as possible to keep the GPU busy.

IV. Results and Analysis

The code was run on a computer with the following specifications: Intel Core i5 running at 2.50GHz, 4GB RAM, Nvidia Geforce GT 525M with 1GB of memory. For the purpose of our simulations, 5000 particles are considered. Fig. 7(a) shows the initial positions of the particles randomly arranged 0.2 m above the bottom wall of the bed. They are then allowed to fall under the action of gravity. It is expected that these particles will all fall at the same speed, rebound due to collisions and eventually pile up inside the bed after a certain time. The position of each particle was noted at 0.2 second time interval as shown in Figs. 7(b) to (h). An ordered arrangement slowly begins to form inside the bed.



(a) 0 s (b) 0.2 s (c) 0.4 s (d) 0.6 s



(e) 0.8 s (f) 1.0 s (g) 1.2 s (h) 1.4 s

Figure 7. DEM snapshots of 5000 falling particles at different times.

As we have mentioned earlier, the nearest neighbor algorithm can be run in parallel by using threads. Each Nvidia graphic card has an upper limit concerning the number of threads that can be generated, depending on its compute capacity. The graphic card used in this work had a compute capacity of 2.1. As stated earlier, generating as much threads as possible is important in a CUDA program. However there exists a limit for the number of threads above which speedup is no longer obtained. Moreover choosing an appropriate number of threads allows maximum usage of the GPU. In term of portability, we should choose a number which is not too large so that it can run on older GPU that can support CUDA. The execution time of the algorithm for 1000 particles was measured with increasing number of threads. Fig. 8 shows the results obtained.

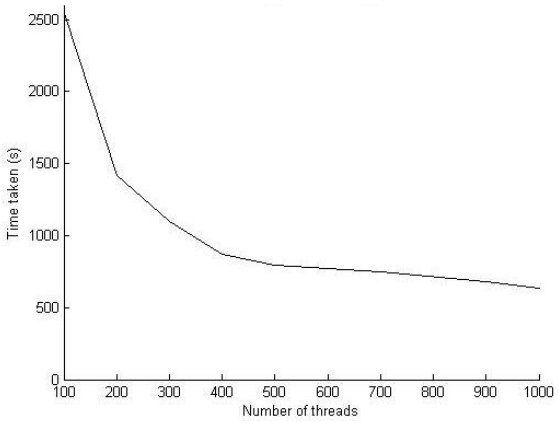


Figure 8. Execution time of function nearest_neighbour() on GPU applied to 1000 particle for different number of threads.

We can observe a noticeable decrease in the execution time as the number of threads is increased. Above 800 threads, no significant decrease in execution time is obtained for each case. To evaluate the performance of the GPU implementation of the algorithm, it was tested with increasing number of particles. The CPU implementation was also tested in the same way for comparison with the GPU version. Fig. 9 compares the total execution time of the serial version and the parallel version for different values of NUM from 100 to 600.

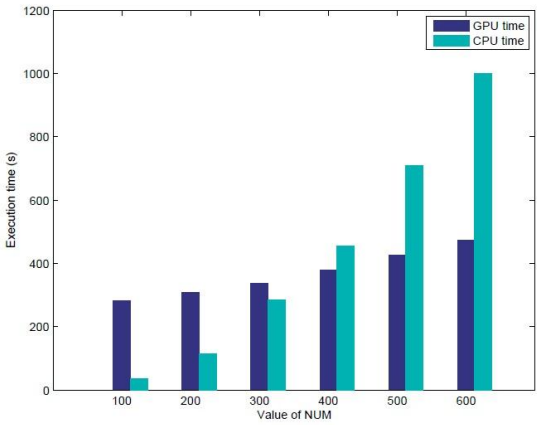


Figure 9. Comparison of total execution of the DEM code on both the CPU and GPU for some values of *N*.

For small number of particles (less than about 350), we see that the serial version is faster than the CUDA optimized program. This is due to the time for transferring data to and from the GPU being larger than actual execution of the

function on the GPU. However as the number of particles increases, the CUDA optimized version beats the serial version.

V. CONCLUSIONS

In this work, we explained how parallel computations using the Graphical Processing Unit (GPU) can enhance scientific computations. The simulation of falling particles in a rectangular bed using the soft-sphere DEM approach was considered. This algorithm is highly parallel. Codes for both a serial version (using CPU) and a CUDA version (using GPU) were written using C++. The execution time of the serial version was compared with the CUDA version and the latter proved to be significantly faster as the number of particles is increased. This demonstrates the high capability of enhancing a DEM code using GPU.

REFERENCES

[1] Cundall P.A., Strack O. D. L. (1979), "A discrete numerical model for granular assemblies", Geotechnique Volume 29, Issue 1, 01 March 1979, pp. 47-65.

[2] Hoomans, B.P.B., Kuipers, J.A.M., Briels, W.J., Van Swaaij, W.P.M., (1996), "Discrete particle simulation of bubble and slug formation in a two-dimensional gas-fluidised bed: a hard sphere approach", chem.. Engng Sci., 51, 99-118

[3] Singh V., Gupta, G.S. and Sarkar, S., (2007), "Study of gas cavity size hysteresis in a packed bed using DEM", Chemical Engineering Science, Vol. 62, No. 22, pp. 6102-6111.

[4] Wassgren, C. (2013), "DEM Modeling: Lecture 06 Introduction to Soft-Particle DEM Normal Contact Force Models. Part I [online]". Available from: http://pharmahub.org/resources/123/download/psl_purdue_dem_lecture06_no%rmalcontactforcemodels_parti.pdf [Accessed on 17-May-2014].

[5] Langston P. A., Tuzun U. and Heyes D. M., (1995), "Discrete element simulations of granular flow in 2D and 3D hoppers: dependence of discharge rate and wall stress on particle interactions", Chemical Engineering Science, 50, 6, 967 – 987.

[6] Langston P. A., Tuzun U. and Heyes D. M., (1996), "Distinct element simulations of interstitial air effects in axially symmetric granular flows in hoppers", Chemical Engineering Science, 51, 3, 876 – 891.

[7] Langston P. A. and Tuzun U., (1997), "Continuous potential discrete particle simulations of stress and velocity fields in hoppers: transition from fluid to granular flow", Chemical Engineering Science, 49, 8, 1259 – 1275.

[8] Pinson, D, (2005). "Application of discrete particle simulation to flow in a transfer chute", Australian Bulk Handling Review, February/March:77-80.

[9] Zhou Z., Zhu H., Yu A., Wright B., Pinson D. and Zulli P., (2005), "Discrete particle simulation of solid flow in a model blast furnace", ISIJ International, 45, 12, 1828 – 1837.

[10] Heuze, F.E., Walton, O.R., Maddix, D.M., Shaffer, R.J., and Butkovich, T.R., "Analysis of Explosions in Hard Rocks: The Power of Discrete Element Modeling," Comprehensive Rock Engineering - Analysis and Design Methods, Vol. 2, 1993, pp. 387-413.

[11] Kun F., Herrmann H.J., "A study of fragmentation processes using a discrete element method", Comp. Meth. in Appl. Mech. and Eng., 138, 3-18 (1996)

[12] Bolander, J.E. and Saito, S., "Discrete Modeling of Short-Fiber Reinforcement in Cementitious Composites," Advanced Cement Based Materials, Vol. 6, 1997, pp. 76-86.

[13] Camborde, F., Mariotti, C., and Donze, F.V., "Numerical Study of Rock and Concrete Behavior by Discrete Element Modeling," Computers and Geotechnics, Vol. 27, 2000, pp. 225-247.

[14] Prochazka, P.P., "Application of Discrete Element Methods to Fracture Mechanics of Rock Bursts," Engineering Fracture Mechanics, Vol. 71, 2004, pp. 601-618.

[15] Morrison, D.J., Wu, W. (2007). "Experimental Validation of the Discrete Element Method (DEM)", Iron ore- proceedings -cd-rom edition-; 341-352 iron ore conference, Iron ore, Austral-Asian Institute for Mining & Metallurgy.

[16] Moreland, K. and Angel, E. (2003), "The FFT on a GPU", *Graphics Hardware*.

[17] Adinetz A.V., Berezin S.B. (2006), "Implementation Classical Ray tracing on GPU-a case Study of GPU Programming", International Conference Graphicon, Novosibirk Akademgorodok, Russia.

[18] Anderson, A. G., III, W. A. G. and Schroder, P. (2007), "Quantum Monte Carlo on graphic processing units", Computer Physics Communications 177.

[19] Polyakov, T., Lyutyy, V., Denisov, S., Reva, V.V. and Hanggi, P. (2013), "Large-scale ferrofluid simulations on graphics processing units", *Computer Physics Communications* 184.

[20] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E. and Phillips, J. C. (2008), "Graphics Processing Units powerful, programmable, and highly parallel are increasingly targeting general-purpose computing applications.", *Proceedings of the IEEE* 96.

[21] Nolan, G. (2009), "Improving the k-Nearest Neighbour Algorithm with CUDA", Technical report, The University of Western Australia.

[22] Kirk, D. B., Mei, W. and Hwu, W. (2010), "Programming massively parallel processors hands- on approach", Morgan Kaufmann, 30 corporate drive, suite 100, MA 01803, Burlington, USA.

[23] Sanders, J. and Kandrot, E. (2011), "CUDA by Example: An introduction to General-Purpose GPU programming", Addison-Wesley.

[24] Cooper, C. (2011), "GPU Computing with CUDA Lecture 1: Introduction [online]". Available from:

http://www.bu.edu/pasi/files/2011/07/Lecture1.pdf [Accessed on 20-February-2014].

[25] Chamroo, S. (2011), "Experimental and Numerical investigation of granular flow in a packed bed", Technical report, University of Mauritius.