Available online at www.sciencedirect.com

ScienceDirect

SoftwareX 5 (2016) 62–66

SoftwareX

www.elsevier.com/locate/softx

Blaze-DEMGPU: Modular high performance DEM framework for the GPU architecture

Nicolin Govender[a,*], Daniel N. Wilke[b], Schalk Kok[b]

[a] CSIR, Center for High Performance Computing, Rosebank, Cape Town, 7700, South Africa

[b] University of Pretoria, Department of Mechanical and Aeronautical Engineering, Pretoria, 0086, South Africa

Received 6 December 2015; received in revised form 13 April 2016; accepted 15 April 2016

Abstract

Blaze-DEMGPU is a modular GPU based discrete element method (DEM) framework that supports polyhedral shaped particles. The high level performance is attributed to the light weight and Single Instruction Multiple Data (SIMD) that the GPU architecture offers. Blaze-DEMGPU offers suitable algorithms to conduct DEM simulations on the GPU and these algorithms can be extended and modified. Since a large number of scientific simulations are particle based, many of the algorithms and strategies for GPU implementation present in Blaze-DEMGPU can be applied to other fields. Blaze-DEMGPU will make it easier for new researchers to use high performance GPU computing as well as stimulate wider GPU research efforts by the DEM community.

Keywords: DEM; GPU; Polyhedra; Particle transport

Code metadata

Code metadata description	
Current code version	v 02-2015
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-15-00085
Legal Code License	BSD license
Code versioning system used	git
Software code languages, tools, and services used	C++
Compilation requirements, operating environments & dependencies	NVIDIA CUDA version 6.0 or higher [11] and Python 2.7
If available Link to developer documentation/manual	https://github.com/ElsevierSoftwareX/SOFTX-D-15-00085
Support email for questions	nico.wilke@up.ac.za, wilkedn@gmail.com

1. Introduction

The discrete element method (DEM) [1] is becoming widely accepted as an effective method to address engineering problems involving granular and discontinuous materials. Applications where DEM excels include granular flows, powder mechanics and rock mechanics. DEM simulates particle motion by including rotational degrees-of-freedom, contact and complicated geometries. In our case, complicated geometries are described by polyhedra.

DEM is computationally intensive, which limits either the number of particles or the duration of the physical event to be

* Corresponding author.
E-mail address: govender.nicolin@gmail.com (N. Govender).

http://dx.doi.org/10.1016/j.softx.2016.04.004

simulated. Recent advances in numerical algorithms for nearest neighbor searching have made it possible to simulate larger number of particles on mainly multi-core central processing unit (CPU) architectures. The CPU architecture limits the potential to further increase the number of particles to be simulated, due to the limited number (typically 8–16) of high performance computing cores [2–4]. Several DEM codes take advantage of modern parallel processing capabilities through shared or distributed systems.

A modern addition to the spectrum of parallel processing capabilities is the graphical processing unit (GPU). The GPU is unlike shared and distributed parallel systems that often only require minor changes or additions to single core codes. While little additional effort is required to port parallel codes to the GPU, such an implementation is in most cases slower than the CPU due to the memory transaction costs on the GPU and significantly larger parallel computation that is required to take advantage of the larger number of cores on the GPU. This limitation requires a fundamental rethink to implement DEM efficiently, such that it scales well with the number of computing cores on single and multiple GPU setups. A significant benefit of the NVIDA-GPU architecture is that the same code will scale on low end GPUs with a few hundred cores to high end GPUs with thousands of cores, this also makes the code scale with multiple GPUs on clusters. Algorithm development also has to take into account that the GPU is a memory restricted device with various levels of memory executing at different speeds. This paper outlines `BLAZE-DEMGPU`, our modular DEM framework for the GPU computing architecture [5]. This framework is designed to be easily extended in terms of base capability and functionality. This makes the GPU architecture easily accessible and freely available to the DEM community.

2. Discrete element theory

The linear momentum of particle i in three dimensional space is

$$\mathbf{L} = m_i \mathbf{v}_i, \tag{1}$$

with m_i and $\mathbf{v}_i$ the mass and the velocity of the center of mass of particle i respectively. The angular momentum of particle i in three dimensional space is given by

$$\mathbf{H} = \boldsymbol{I}_i \boldsymbol{\omega}_i, \tag{2}$$

where $\boldsymbol{I}_i$ is the inertia tensor and $\boldsymbol{\omega}_i$ is the angular velocity vector. Given all forces $\boldsymbol{f}_i$ that act on particle i, the problem is reduced to integrating the change in linear momentum

$$\dot{\mathbf{L}}_i = m_i \frac{d^2 \boldsymbol{p}_i}{dt^2} = \boldsymbol{f}_i, \tag{3}$$

and integrating the change in angular momentum about the center of mass for an axis fixed to the body (Euler's moment equation),

$$\dot{\mathbf{H}}_G = \boldsymbol{I}_i \frac{d\boldsymbol{\omega}_i}{dt} + \boldsymbol{\omega} \times \boldsymbol{H} = \boldsymbol{t}_i. \tag{4}$$

After integration, we have the particle's position, velocity, acceleration, angular position, angular velocity and angular acceleration. Here, $\boldsymbol{p}_i$ is the position of the center of mass of particle i, $\boldsymbol{f}_i = \sum \boldsymbol{f}_i^s + \sum \boldsymbol{f}_i^b$ consists of the surface tractions $\boldsymbol{f}_i^s$, and body forces $\boldsymbol{f}_i^b$. The moments $\boldsymbol{t}_i = \boldsymbol{r}_i \times \boldsymbol{f}_i^s$, with $\boldsymbol{r}_i$ the position vector from the center of mass to the surface force vector. Since we only consider short-ranged interactions, this allows us to resolve forces solely through contact. Hence, the surface tractions on particle i are the result of contact with other particles, contact with the boundary of the domain or applied external loads. The benefit of this restriction is that we can resolve interaction pairs solely through nearest neighbor searches.

2.1. Broad and narrow phase contact resolution

Broad phase contact is resolved through nearest neighbor searches on a collision detection grid. The grid size is dictated by the size of the largest particle in the computing domain. The grid position is stored as a single integer using a hashing function. Hash values that are numerically close to each other indicate grid positions that are close to each other. This significantly improves the efficiency of searching for potential contact with nearest neighbors [6].

When two particles i and j have been identified as potentially being in contact during the broad phase, we enter the narrow contact resolution phase. Here, it is established whether actual contact has been made by considering whether vertex–face, face–face or edge–edge contact has been made. This determination is based on whether the intersection volume of particle i and particle j is greater than zero [6].

2.2. Normal contact force laws

Once contact between two particles i and j has been established we need to resolve the interaction forces through constitutive relations. A contact distance δ_{ij} and contact normal $\boldsymbol{n}_{ij}$ can then be established. In addition the velocity $\dot{\delta}_{ij}$ with which the contact distance changes can also be estimated. We implemented the simplest constitutive relationship which is a linear repulsive force and linear dissipative force model, given by

$$f_{ij}^n = k\delta_{ij} + \gamma_0 \dot{\delta}_{ij}. \tag{5}$$

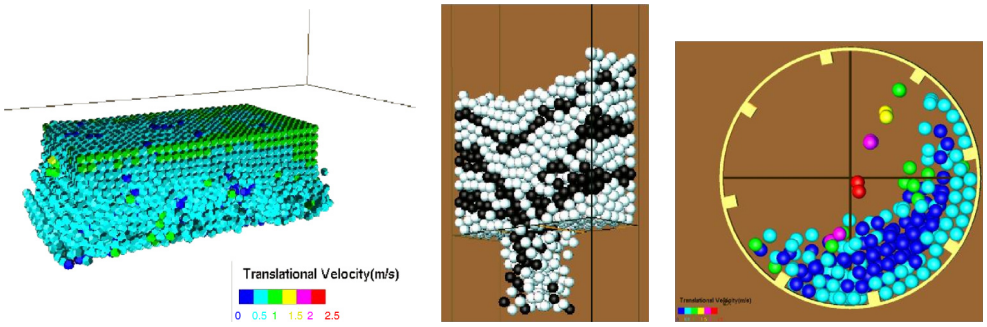
Here, k (N/m) is the normal contact stiffness and γ (Ns/m) is the normal contact damping between two particles.

2.3. Tangential contact force laws

In this framework we only consider friction resistance due to sliding. Friction resistance as a result of sliding is due to the relative tangential surface velocity $\boldsymbol{v}_{ij}^t$ between two particles i and j at contact

$$\boldsymbol{v}_{ij}^t = \boldsymbol{v}_{ij} - \boldsymbol{n}_{ij}(\boldsymbol{n}_{ij} \cdot \boldsymbol{v}_{ij}), \tag{6}$$

which depends on the relative surface velocity $\boldsymbol{v}_{ij}$ between particles and the contact normal $\boldsymbol{n}_{ij}$. The relative surface velocity at the point of contact is due to the relative translation



(a) Stack falling under gravity. (b) Silo discharge. (c) Ball mill.

Fig. 1. Practical example simulations included in the BLAZE-DEMGPU distribution using (a) polyhedral or (b)–(c) spherical particles.

and rotation of the two particles

$$\boldsymbol{v}_{ij} = \boldsymbol{v}_i - \boldsymbol{v}_j + \boldsymbol{r}_i \times \boldsymbol{\omega}_i + \boldsymbol{r}_j \times \boldsymbol{\omega}_j, \tag{7}$$

with $\boldsymbol{r}_k$ the vector from the center of mass to the contact point and $\boldsymbol{\omega}_k$ the angular velocity of particle k, $k = i, j$. The relative tangential surface velocity $\boldsymbol{v}_{ij}^t$ in addition to Coulomb's law dictates the tangential force f^t with direction such that it opposes the motion of the two particles. Numerical time integration is based on the classical forward Euler time integration scheme. The integration scheme can easily be extended to other integration schemes such as the class of Verlet algorithms [7].

3. Simulation capabilities

The BLAZE-DEMGPU framework is able to simulate polyhedral and spherical shaped particles as depicted in Figures (a)–(c). Polyhedra with up to 32 vertices can be included. A particle size aspect ratio of 4:1 can be handled, without significant performance degradation. Although a larger particle size distribution is possible, the computational benefit of the broad phase collision detection diminishes. The simulations are conducted using single precision arithmetic on the GPU, which limits the range of values in a single calculation to $<1 \times 10^{-6}$.

A simulation is constructed from three types of physical objects namely volume objects, surface objects and particle objects. Volume and surface objects can be stationary or may translate and rotate relative to the broad phase contact grid but are not considered in estimating the grid size. Particle objects move relative to the broad phase grid and are considered in estimating the size of the broad phase contact grid as we only grid the regions containing particles to improve efficiency and allow for large geometries to be simulated.

All physical objects are stored in constant memory (48kB) on the GPU, which is the fastest memory available with the lowest latency when reads are collapsed. The computational requirements for a *float* are 4 bytes. The vertices, edge pairs, face normals, face areas and centroid of the object are computed and stored for each object. Note: vertices are specified the Cartesian origin in the positive quadrant. For each type of particle object the moment of inertia tensor in the unrotated configuration is also stored. The moment of inertia tensor in the current configuration is then computed by pre and post multiplication of the proper orthogonal rotation matrix.

The memory requirements for the objects are 12 bytes per vertex, 12 bytes per centroid, 12 bytes per face normal, 4 bytes per edge pair, 4 bytes per face area and 24 bytes per inertia tensor as only the symmetric part of the inertia tensor is stored. It is possible to extend the storage of objects to the larger global memory (2 GB–24 GB) by sacrificing computational efficiency, since the global memory is about 100x that of the constant memory which we currently store the geometry.

Fig. 1 illustrates the basic simulation types possible with the code. Fig. 1(a) is a gravity deposition of particles in a box, which is commonly used during filling or simulation of dam breaks etc. It is also typically used for benchmarking as demonstrated in Table 1. Fig. 1(b) is a silo which is a common device used in industry for storage and dispensation of particulate materials. Fig. 1(c) is a ball mill that is used for the crushing and grinding of product in the mineral/mining processing industries.

The parallel computational efficiency of the BLAZE-DEM framework on the GPU architecture allows us to create simulations that are closer to reality by increasing the number of particles and shape complexity we can simulate in a shorter time frame as depicted in Table 1.

Fig. 2(a) shows the benefits of Blaze-DEM in terms of increasing the number of particles in a simulation. We see that current DEM simulations did not correctly capture the behavior of what was being simulated, while Blaze-DEM gives a very good match to what is seen in reality [12,14]. Fig. 2(b) shows the effect of particle shape, a major problem in silo flow is that of arching where flow stops or is non-smooth, we see that using spheres a smooth flow is predicted while the polyhedra exhibit the arching behavior observed in reality.

3.1. Impact

BLAZE-DEMGPU has been validated extensively against experimental results and traditional CPU DEM codes [5,6,14,15]. In [12,14] we applied the code to industrial ball mill and silo simulations respectively. The results obtained were in good agreement to experiment. More importantly simulations with increased number of particles revealed that in some cases like Fig. 2(a) the lack of particles in existing DEM simulations yielded results that are not correct. Hence using the code DEM can be used as an effective tool for the prediction of many

Table 1
Comparison to other codes for gravity stacking simulation.

Author	Shape	Physics fidelity	N particles	C number
Harida et al. [8]	Clumped	Low	1.64×10^4	0.66×10^6
Longmore et al. [9]	Clumped	High	2.56×10^5	1.49×10^6
Radake et al. [10]	***Sphere**	**High**	20×10^6	20×10^6
Nvidia SDK (2014) [11]	*Sphere	Low	2.5×10^5	125×10^6
BLAZE-DEM [12]	***Sphere**	**High**	60×10^6	100×10^6
Note: No published GPU/CPU parallel polyhedra codes				Compute time ($N = 5 \times 10^5$)
BLOCKS [13]	Poly^{cpu}	Highest	5×10^3	186 days
iDEM [13]	$\textbf{Poly}^{cpu}$	**Low**	5×10^5	**2.8 days**
BLAZE-DEM [6]	$\textbf{Poly}^{gpu}$	**High**	32×10^6	**32 min**

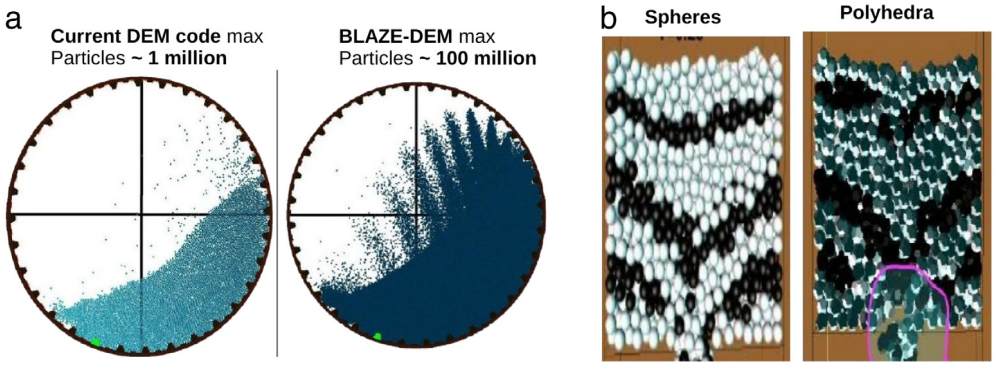


Fig. 2. `Benefits of BLAZE-DEMGPU` (a) particle number (b) particle size.

industrial particle flow problems. The generality of the collision detection methods can be applied to numerous fields as it is a commonly encountered computational problem. Furthermore our strategies for the GPU have been applied in areas outside DEM [16,17].

4. Software dependencies

BLAZE-DEM is free software distributed under the BSD license, which allows for forking into commercial applications. It requires NVIDIA CUDA version 6.0 or higher [11] and Python 2.7. Portability is assured by using only these two computing platforms, which are freely available on Windows, MacOS and Linux, in addition to the source code of the entire framework. Hardware is limited to NVIDIA GPU cards that support at least CUDA compute capability 3.0. In addition, we supply a Python module that generates the geometric information for a polyhedral particle object. Specifically, given the particle vertices the module computes the inertia tensor and center of mass of the particle. BLAZE-DEM commits are subject to mandatory code review and automatic unit testing before it is added to the public git repository (available via git clone https://github.com/ElsevierSoftwareX/SOFTX-D-15-00085).

5. Software framework

The interaction between the hardware and user specified simulation data with the code architecture is outlined in Fig. 3. Additional details are available in the developer's guide.

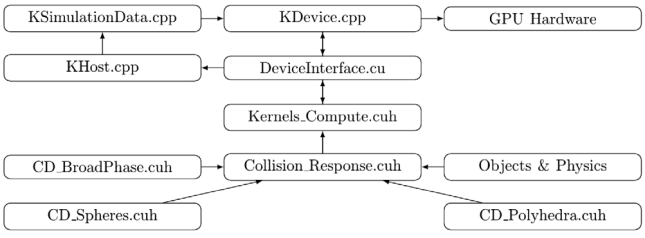


Fig. 3. Outline of the BLAZE-DEM developer overview. Note that "Collision_Detection" is abbreviated to CD.

6. Conclusions

In this paper we presented the modular GPU-based DEM framework BLAZE-DEMGPU that supports polyhedral shaped particles as well as large scale spherical particles. It is envisioned that this framework will make high performance GPU architecture based DEM simulations easily accessible and that it will stimulate wider GPU research efforts in the DEM community.

References

[1] Cundall P. Strack. A discrete numerical model for granular assemblies. Geotechnique 1979;29:47–65.
[2] Boon C, Houlsby G, Utili S. A new algorithm for contact detection between convex polygonal and polyhedral particles in the discrete element method. Comput Geotech 2012;44:73–82.
[3] Zhao D, Nezami E, Hashash Y, Ghaboussi J. Three-dimensional discrete element simulation for granular materials. Computer-Aided Eng. Comput.: Internat. J. Eng. Softw. 2006;23:749–70.

[4] Walther JH, Sbalzarini F. Large-scale parallel discrete element simulations of granular flow. Eng. Comput. 2009;26:688–97.
[5] Govender N, Wilke D, Kok S, Els R. Development of a convex polyhedral discrete element simulation framework for NVIDIA Kepler based GPUs. JCAM 2014;270:63–77.
[6] Govender N, Wilke D, Kok S. Collision detection of convex polyhedra on the NVIDIA GPU architecture for the discrete element method, J. Appl. Math. Comput. http://dx.doi.org/10.1016/j.amc.2014.10.013.
[7] Verlet L. Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. Phys Rev 1967;159:98–103.
[8] Harada T. GPU Gems 3: Real-time rigid body simulation on GPUs, Vol. 3. 2008.
[9] Longmore J, Marais P, Kuttel M. Towards realistic and interactive sand simulation: A GPU-based framework. Powder Technol. 2013;235: 983–1000.
[10] Neubauer G, Radek CA. GPU based particle simulation framework with fluid coupling ability, NVIDIA GTC 2014, San Jose, USA, 2014.
[11] NVIDIA, Cuda 6 (May 2014). http://www.nvidia.com/cuda.
[12] Govender N, Rajamani R, Wilke D, Kok S. Discrete element simulation of mill charge in 3d using the blaze-dem gpu framework., J. Miner. Eng. http://dx.doi.org/10.1016/j.mineng.2015.05.010.
[13] Jaelee S. Developments in large scale discrete element with polyhedral particles simulations. (Ph.D. thesis) University of Illinois at Urbana-Champaign; 2014. www.uiuc.edu.
[14] Govender N, Pizette P, Wilke D, Abriak N. Validation of the GPU based Blaze-DEM framework for hopper discharge. In: Proceedings of the international conference on particle-based methods 2015 Spain, 2015.
[15] Govender N, Wilke D, Kok S. A GPU based polyhedral particle DEM transport code, NVIDIA GTC 2014, San Jose, USA, 2014. http://on-demand.gputechconf.com/gtc/2014/poster/pdf/P4126.
[16] Mei G, et al. A generic paradigm for accelerating laplacian-based mesh smoothing on the gpu. Arab J Sci Eng 2014;39:7907–21.
[17] Mei G, Tian H. Performance impact of data layout on the GPU-accelerated IDW interpolation, Vol. 5. Springerplus. 2014. p. 104–9.