Advanced Materials Research Vol. 320 (2011) pp 329-334
Online available since 2011/Aug/16 at www.scientific.net

doi:10.4028/www.scientific.net/AMR.320.329

A GPU Accelerated Continuous-based Discrete Element Method for Elastodynamics Analysis

Zhaosong Ma[1, a], Chun Feng[1,b], Tianping Liu[1,c], Shihai Li[1,d]

[1] Institute of Mechanics, Chinese Academy of Sciences, Beijing 100190, China

[a]marze@163.com, [b]fengchun@imech.ac.cn, [c]liutp@imech.ac.cn, [d]shli@imech.ac.cn

Keywords: DEM, Discrete Element Method, CUDA, GPU.

Abstract. This paper presents a GPU computing algorithm, used to accelerate the Continuous-based Discrete Element Method (CDEM). Using a NVIDIA GTX VGA card, the computing speed achieved an average 650 times speedup ratio vs. Intel Core-Dual 2.66 GHz CPU. To parallelize the CDEM algorithm, the clone node force refreshing process is separated from the elemental calculation, and is replaced by a "Node Group" force assignment process, which ensures the data independence in parallel execution.

Introduction

The GPU (Geographic Processing Units) computing is a new technique which makes use of VGA cards to provide high-performance computing capacity with very low costs. The Continuum-based Distinct Element Method (CDEM) is an approach to simulate the progressive failure of geological mass, which is mainly used in land-slide stability evaluation, coal and gas outburst analysis, as well as mining and tunnel designing.

CDEM is the combination of Finite Element Method (FEM) and Distinct Element Method (DEM), It contains two kinds of elements, blocks and interfaces (Fig 1). A discrete block is consisted of one or more FEM elements, all of which share the same nodes and faces. An interface contains several normal and tangent springs; each connects nodes in different blocks. Inside the block the FEM is used, while for the interface, the DEM is adopted.

Various constitutive models can be used in the block, including the linear elastic model, Drucker-Prager model, the block breakage model and the discrete spring model. For the interfaces, the linear elastic model and the linear crack model can be used. To simulate the randomness of dimension and the geometrical features of geo-material, the random joint model is used. A fluid-solid interaction model is introduced to CDEM to study the influence of the seepage flow in fractures.

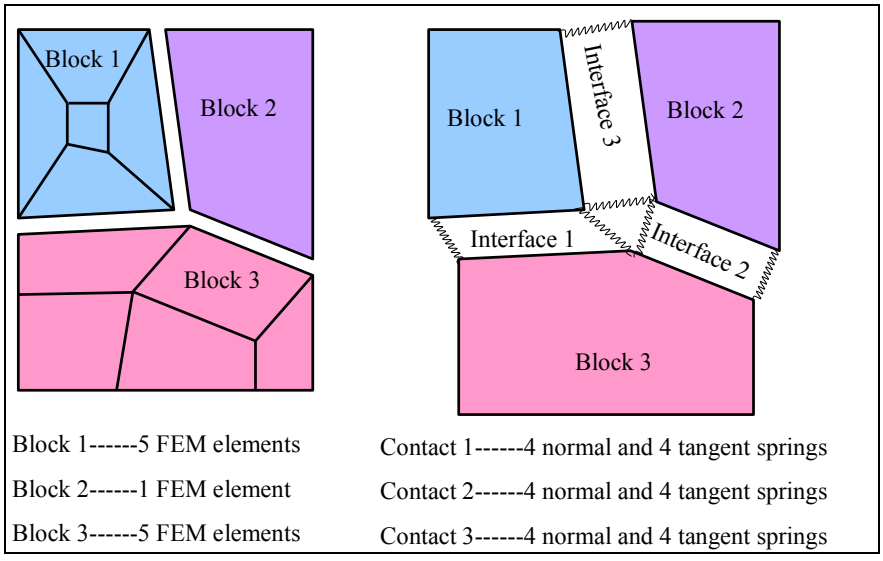


Fig. 1 Blocks and interfaces for 8 nodes hexahedron

CDEM is an explicit time history-analysis FEM/DEM approach on finite difference principles and forward-difference approximation is adopted to calculate the progressive process through a time marching scheme. During the calculation, the dynamic relaxation method is used to achieve convergence in a reasonable period time with small time steps, and the convergence is reached when the total magnitude of the kinetic energy is minimized.

Fig 2 shows the main process to solve a classical geological problem.

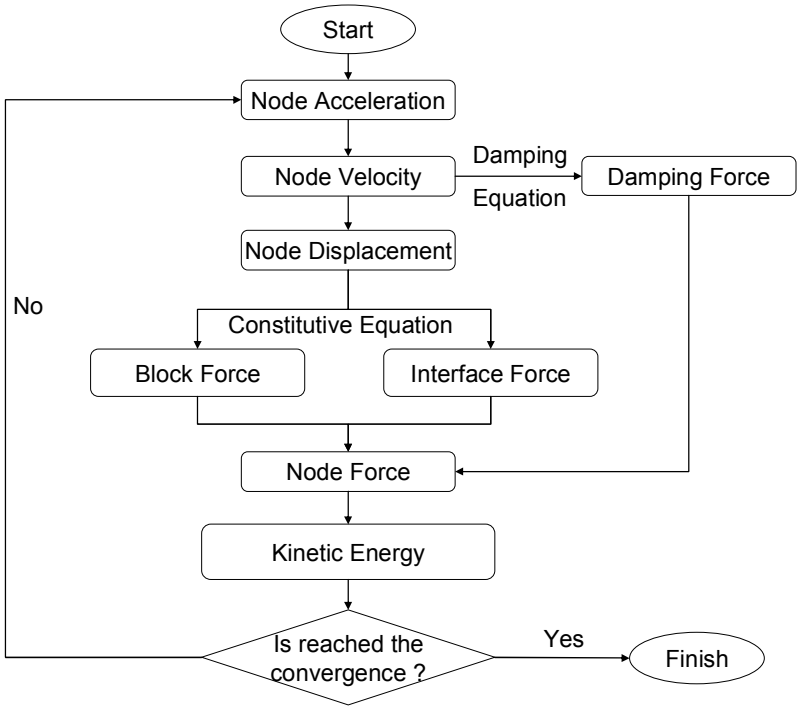


Fig. 2 The process to solve a geological problem

The constitutive equation of block is show in Equation 1 (just for liner elastic model), where $\{F\}_i^e$ denotes the vector of node force of element i, $\{u\}_i^e$ means the vector of node displacement of element i, and $[K]_i^e$ represents element stiffness matrix of element i.

$$\{F\}_i^e = [K]_i^e \{u\}_i^e \quad (1)$$

The constitutive equation of interface is show in Equation 2 (just for liner elastic model), Where F_n^j and F_s^j denotes the normal and tangent force of spring j, K_n^j and K_s^j means the normal and tangent stiffness of spring j, Δd_n^j and Δd_s^j represents the normal and tangent displacement increment of spring j.

$$\begin{cases} F_n^j = -K_n^j \times \Delta d_n^j \\ F_s^j = -K_s^j \times \Delta d_s^j \end{cases} \quad (2)$$

For simulating the failure of interface, liner crack model is adopted. During elastic stage, the same as liner elastic model, equation 2 is used to calculate the spring force in the interface. If the spring force exceeds the strength of the interface, the crack on interface begins and the Mohr-Coulomb criterion (Equation 3) is adopted. It is positive when F_n is compression, and negative when F_n is tensile. In Equation 3, T denotes the tension, ϕ means inner friction, and C represents cohesion.

$$\begin{cases} If \quad -F_n^{\,j} \geq T \quad then \quad F_n = F_s = 0, T = 0 \\ If \quad F_s^{\,j} \geq F_n^{\,j} \times \tan(\phi) + C \quad then \quad F_s = F_n \times \tan(\phi) + C, C = 0 \end{cases} \tag{3}$$

To dissipate the excessive energy in block system, the Rayleigh damping is introduced, which contains two kinds of damping, the mass proportional damping (α) and the stiffness proportional damping(β). The damping force is calculated from Equation 4, where $\{f\}_i^e$ means the vector of damping force of element *i*, $[M]_i^e$ denotes the element mass matrix of element i, and $\{v\}_i^e$ represents the vector of velocity of element *i*.

$$\{f\}_i^e = \alpha[M]_i^e\{v\}_i^e + \beta[K]_i^e\{v\}_i^e \tag{4}$$

Solution Algorithm in Parallel

Unlike general CDEM algorithm which uses clone node strategy to ensure node consistency among elements, the parallel algorithm does the same work by means of a "nodal force group" strategy. The difference is out of reason of access conflict. For general CDEM, a node's force is cloned to the associated nodes just when the node's force calculation is done. This works well in serial execution, but may cause data access conflict in parallel execution, for various threads may clone their forces to a same node at the same time. Differently, the parallel algorithm uses a data structure to temporarily store the forces calculated. The nodal forces will be redistributed in a separate procedure after all of the node's force calculation is done, as shown in Fig 3.

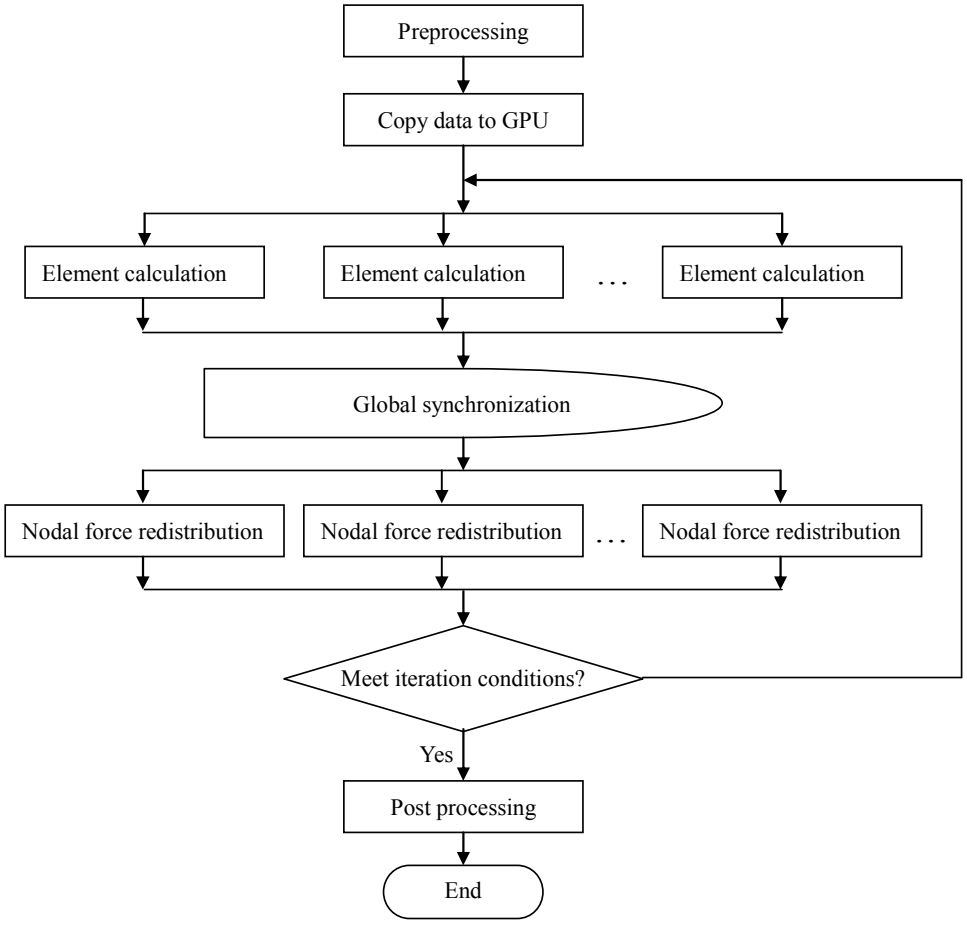


Fig. 3 Flowchart of CDEM Parallel Calculation

The element calculation procedure is designed according to the specialty of GPU. As is known, a GPU consists of a set of SIMT (single-instruction, multiple-thread) multi-processors, which are mapped to CUDA blocks[1]; each multi-processor has an instruction unit and several scalar processor cores, which are mapped to CUDA threads, and each scalar thread executes independently with its own instruction address and register state. Accordingly, the element calculation processes are divided into parallel CUDA blocks, each of which contains certain number of elements. In this work, a CUDA block contains 32 elements for best performance. As is shown in Fig 4, the CUDA blocks are further divided into threads, each of which processes one node.

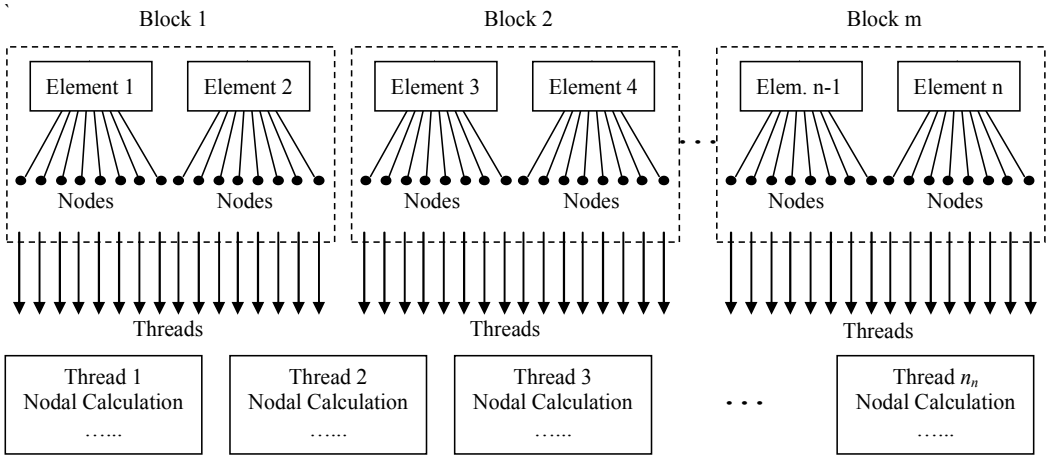


Fig 4. Thread Division Scheme for Element Calculation
* Practically in the program, each block contains 32 elements.

Similarly, the nodal force redistribution procedure is designed following the block-thread architecture, namely, each CUDA block contains certain number of nodal force groups as threads, as is shown in Fig 5.

The number of nodal force groups in a CUDA block should be properly determined to obtain best performance, which is discussed in the following section.

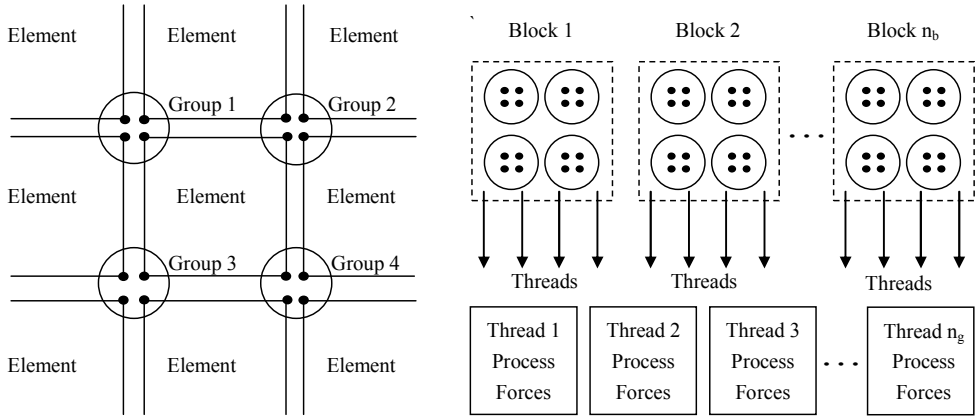


Fig 5. Nodal Force Group

Performance Optimization

The program is developed and tested using NVDIA CUDA Toolkit 2.1. The 8 nodes isoperimetric element is chosen as element model.

To maximize parallel execution, the number of threads per block is carefully chosen. As is recommended by NVIDIA, the best results will be achieved when the number of threads per block is a multiple of 64, and for ideal performance, exceed 192 threads per block is recommended. In practice, best performance is achieved when the number of elements per block is 16 or 32. Because of the 8 nodes element model, the number of threads per block is equal to Equation 5.

$$N_t = N_e \cdot 8 = \begin{cases} 256, N_e = 32 \\ 128, N_e = 16 \end{cases} \quad (5)$$

Where N_t is the number of threads per block; Ne is the number of elements per block. When problem scale is small (1000 elements), 16 elements per block is a little faster; for large-scale problems, 32 elements per block is a little better. The test results are shown in Table 1:

Table 1 Running Time: 16 vs. 32 Elements per Block

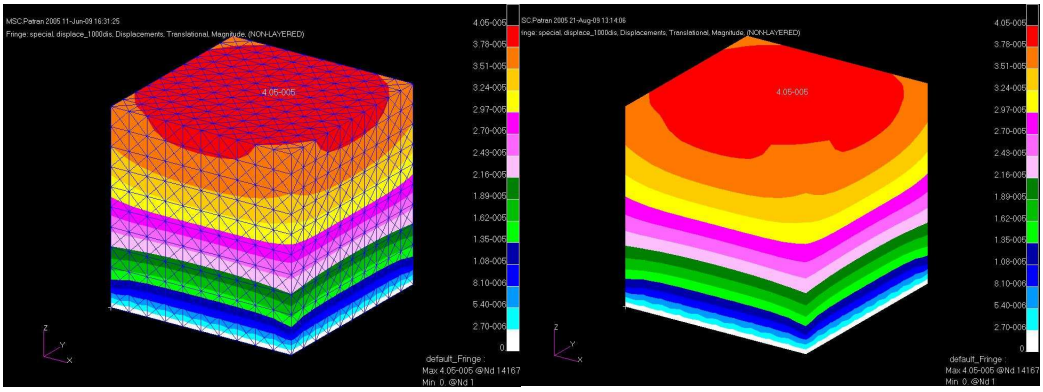
Problem scale	16 elements per block	32 elements per block
1000 elements, 8000 steps	0.460 sec	0.478 sec
41232 elements, 4000 steps	6.23 sec	6.10 sec
Problem scale	16 elements per block	32 elements per block
1000 elements, 8000 steps	0.460 sec	0.478 sec
41232 elements, 4000 steps	6.23 sec	6.10 sec

The effective bandwidth of each memory space depends significantly on the memory access pattern. As described in CUDA Programming Guide, global memory bandwidth is used most efficiently when the simultaneous memory accesses by threads in a half-warp (during the execution of a single read or write instruction) can be coalesced into a single memory transaction of 32, 64, or 128 bytes. To achieve maximum global memory bandwidth, the data structures are realigned to thread. Besides, shared memory is used to store nodal forces, for the shared memory space is much faster than the local and global memory spaces, and nodal forces are reused among threads.

Evaluation

A simple 1000 elements model is used to verify the algorithm. The model consists of a 10x10x10 hexahedron elements mesh, with the nodes on the bottom constrained, and gravity applied. Fig 6 shows the result comparison between the CPU version and the GPU version. Table 2 shows the time cost comparison.

Another instance is a fortress of the Great Wall, which contains 41232 hexahedron elements. The results are shown in Fig 7.



a. CPU version b. GPU version

Fig 6. Result Comparison between the CPU Version and the GPU Version

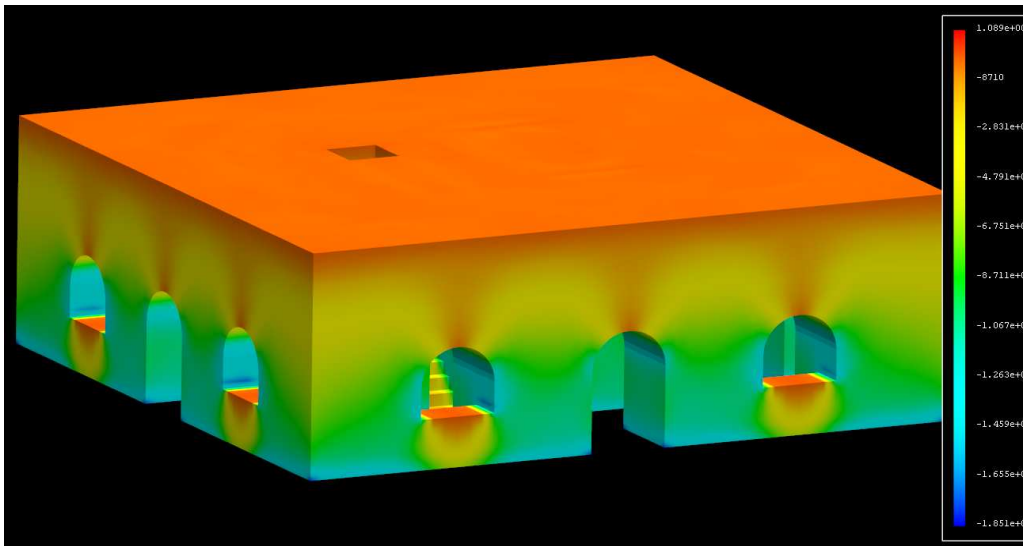


Fig 7. z Result of the Fortress

Table 2 Performance Comparison between CPU and GPU

Problem Scale	CPU	GPU
1000 elements brick, 8000 steps	252 sec	0.478 sec
41232 elements Fortress, 4000 steps	95 min	6.10 sec
Problem Scale	CPU	GPU
1000 elements brick, 8000 steps	252 sec	0.478 sec
41232 elements Fortress, 4000 steps	95 min	6.10 sec

The results of the CPU version and the GPU version are not the very same. Average error is less than 0.01%. This is because the parallel algorithm has a different accumulation order from the serial one. The accumulation occurs in the nodal force redistribution procedure. The CPU version accumulates the nodal forces one by one in the order of the elements, while the GPU version has a different order, which depends on the generation of nodal force group. Besides, the Arithmetic Logical Units on CPU and GPU have different respective error bounds.

References

[1] Information on http://developer.download.nvidia.com

[2] Li SH, Zhao MH, Wang YN, Rao Y: International Journal of Rock Mechanics and Mining Sciences Vol. 41 (2004), p. 436

Key Engineering Materials and Computer Science
10.4028/www.scientific.net/AMR.320

A GPU Accelerated Continuous-Based Discrete Element Method for Elastodynamics Analysis
10.4028/www.scientific.net/AMR.320.329

DOI References
[2] Li SH, Zhao MH, Wang YN, Rao Y: International Journal of Rock Mechanics and Mining Sciences Vol. 41 (2004), p.436.
doi:10.1016/j.ijrmms.2003.12.076