AN OPTIMIZED ALGORITHM FOR DISCRETE ELEMENT SYSTEM ANALYSIS USING CUDA

Zhaosong Ma, Chun Feng, Tianping Liu, Shihai Li

Institute of Mechanics, Chinese Academy of Sciences

In this paper a parallel computing algorithm for discrete element systems is presented. The discrete model is consisted of finite elements and contacts among the elements. The algorithm is realized using C++ and CUDA and was optimized for NVIDIA GPUs. As a result, the performance of the GPU code is 43 times faster than the sequential code on CPU. The parallel algorithm, the optimism strategy, and the test results are discussed.

INTRODUCTION

The CDEM method is a kind of combined discrete-finite element system, which is consisted of finite elements and contact interfaces among the elements. A contact interface contains several springs that connect the two nodes of the elements of the both side, as was introduced in the theses [1] [2]. In the thesis [2], we introduced a parallel algorithm for continuous problems, but it was not available for discrete problems.

In this paper, a parallel algorithm for discrete problems is introduced, which is also realized on GPU, programmed with C++ and CUDA.

PARALLEL ALGORITHM

The method consists of three procedures: element elastic calculation, contact force calculation and dynamic response calculation. The element elastic calculation uses stiffness matrix to obtain nodal forces, given a set of nodal displacements. The contact calculation is to determine the springs' condition, and to calculate the spring forces.

The parallel algorithm meets a problem of access conflict, as shown in Fig 1. When the spring forces are calculated in parallel, the forces may be accumulated in the same place, for example, node 3 of element 1. This may lead to write conflict.

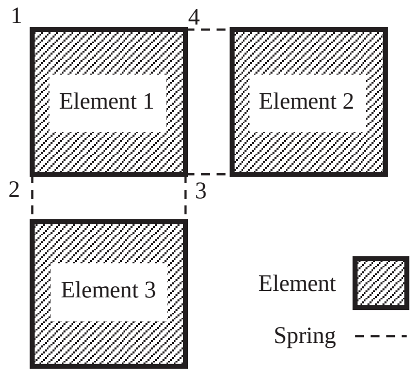

Fig 1. Elements and springs

To resolve the problem of write conflict, a spring force group strategy is used. The spring force group works as a buffer or cache. First, the spring forces are calculated in parallel and stored in the spring data structure, separately. Then, the spring groups are calculated in parallel, each thread accumulates spring forces and save the results in the associated node. A spring group is presented by a data structure which stores a pointer to the associated node and 2 pointers to springs (in 2D cases) or 3 pointers to springs (in 3D cases), as shown in Fig 2.

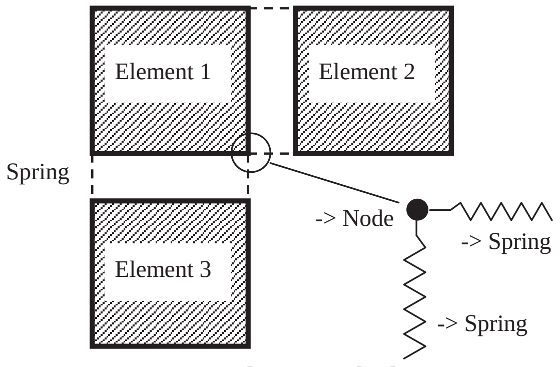

Fig 2. Node group linkage

Now both the element calculation and the contact calculation are parallelized. The problem of write conflict in element calculation is resolved in the very same way of thesis [2]. So the flowchart of the whole procedure is given in Fig 3.

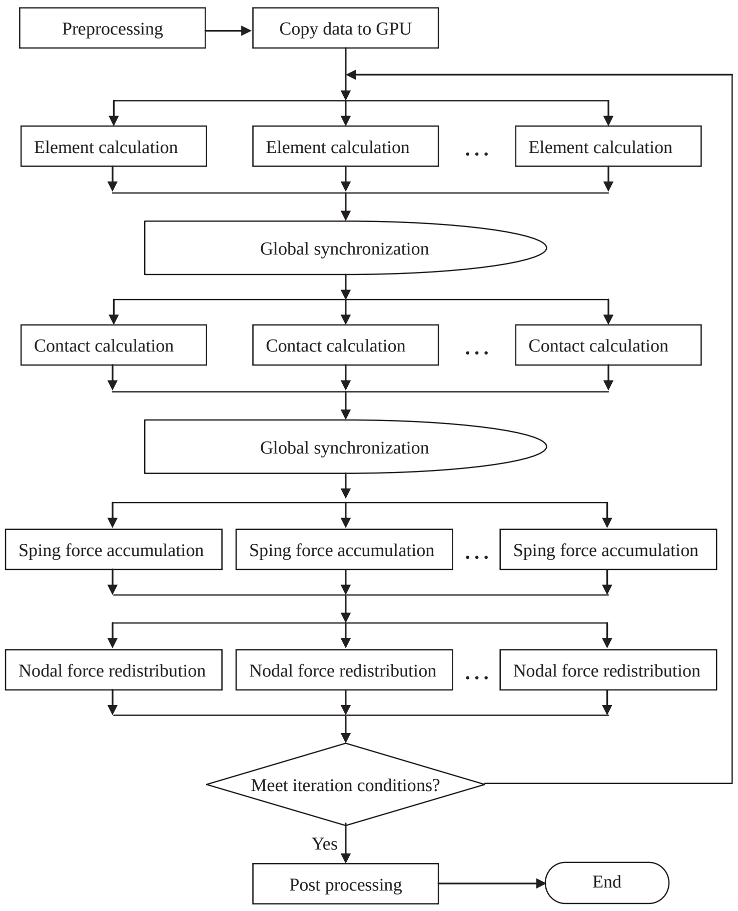

Fig 3. Flowchart of the parallel algorithm

OPTIMIZATION STRATEGIES

The program is written in C++ and developed based on the CUDA architecture, version 3.0. The shared memory strategy is used to maximize the memory bandwidth. Because the springs of a contact (four normal, eight tangential for quadrilateral contact as an example) share the common contact, and thus have a common transformation matrix, we load the transformation matrix into shared memory, so that one copy of the matrix can be used by multiple threads that calculate spring forces. The process of loading matrix is also in parallel, each loading of a column is assigned to a thread. Besides, the bandwidth of shared memory is much wider than that of conventional memory.

The best performance is achieved when the thread number (each block) is 256 for element calculation (whatever single or double precision). For double precision contact calculation, the thread number is 128, for single precision contact calculation, the thread number is 256. The program has been tested on NVIDIA GTX 285, GTX 480, GTX 460, GTX 580, and Tesla C2070. Some performance results are given in Table 1.

RESULTS

In the first case, the GPU boosts performance by up to 43 times, the damage state is shown in Fig 4. In the second case, the model is too large for the CPU program to achieve convergence in time.

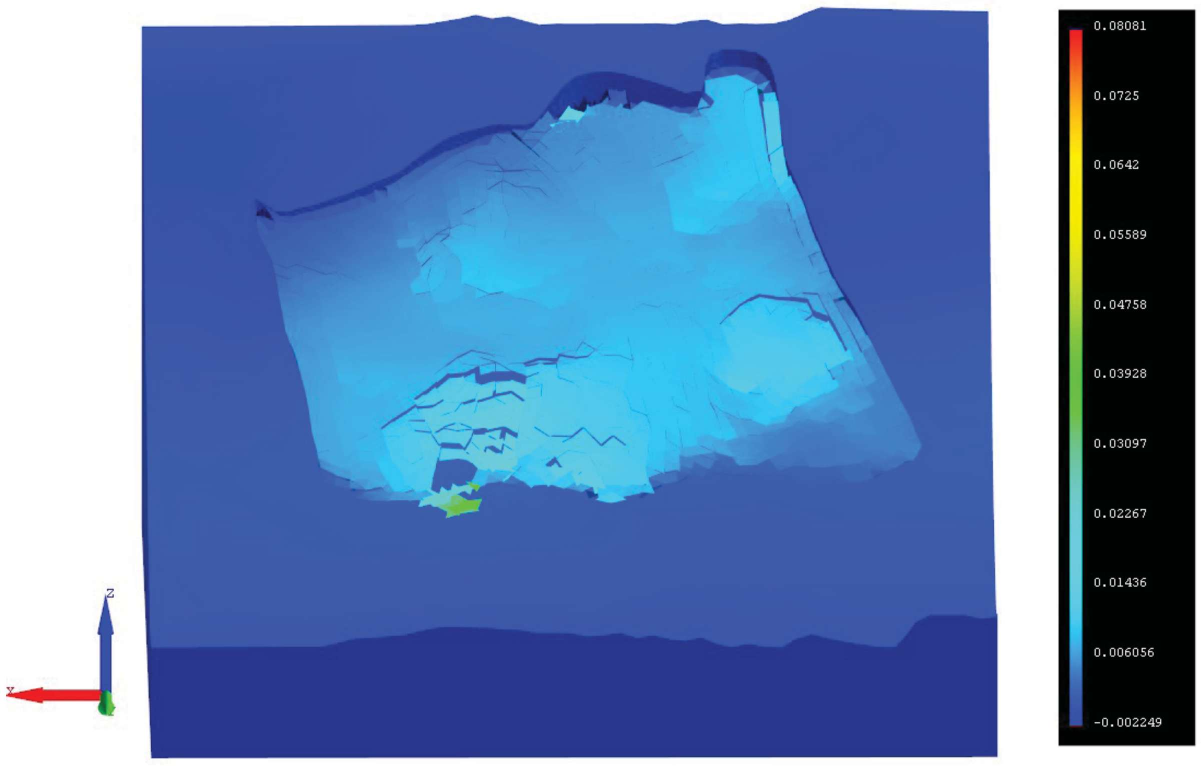

Fig 4. Displacement result of the 111,322 elements landslide simulation

Test case	Platform	Time cost (CPU)	Time cost (GPU)
111,322 elements with 216,923 contacts	Intel I5 3.30 GHz / NVIDIA GTX680	532.82 s/kilo step	12.41 s/kilo step
995,181 elements with 1,954,671 contacts	Intel Xeon E5506 2.13GHz / NVIDIA GTX580	-	76.20 s/ kilo step

Table 1. Performance results

ACKNOWLEDGMENTS

This work was sponsored by The National Basic Research Program (2010CB731500).

REFERENCES

1. Li SH, Zhao MH, Wang YN, Rao Y, 'A New Numerical Method for DEM-Block and Particle Model', *International Journal of Rock Mechanics and Mining Sciences*, **Volume**, No.3/41, 436, 2004.
2. Zhao Song Ma, Chun Feng, Tian Ping Liu, Shi Hai Li, 'A GPU Accelerated Continuous-Based Discrete Element Method for Elastodynamics Analysis', *Advanced Materials Research*, **Volume**, No.320, 329-334, 2011.