Computers and Geotechnics 101 (2018) 149–158

Contents lists available at ScienceDirect

Computers and Geotechnics

journal homepage: www.elsevier.com/locate/compgeo

Research Paper

Large scale parallelisation of the material point method with multiple GPUs

Youkou Dong[a,*], Jürgen Grabe[b]

[a] Institute for Geotechnical and Construction Engineering, Hamburg University of Technology, Harburger Schloßstraße 20, 21079 Hamburg, Germany
[b] Head of Institute for Geotechnical and Construction Engineering, Hamburg University of Technology, Harburger Schloßstraße 20, 21079 Hamburg, Germany

ARTICLE INFO

Keywords:
Material point method
Parallel computing
Large deformation
Surface pipe penetration
Slurry runout

ABSTRACT

The material point method (MPM), which is a combination of the finite element and meshfree methods, suffers from significant computational workload due to the fine mesh required in spite of its advantages in simulating large deformations. This paper presents a parallel computing strategy for the MPM with multiple Graphics Processing Units (GPUs) to boost the method's computational efficiency in large scale problems. Domain decomposition method is used to split the workload over subdomains onto a number of GPUs. Communication between the subdomains is implemented by the data transfer between GPUs and random access memory. On each GPU the MPM algorithm is parallelised over nodes or particles. Benchmark problems of slurry runout and surface pipe penetration are analysed to quantify the speedup of the multiple-GPU parallel simulations over the sequential counterparts on the central processing unit. The maximum speedup with 1 GPU is 84 and increases to ~1280 using 16 GPUs.

1. Introduction

The material point method (MPM), originating from the particle-in-cell method in computational fluid dynamics [12] and extended to solid mechanics by Sulsky et al. [23], can be regarded as a combination of the Finite Element (FE) and meshfree methods. In MPM, the continuum is discretised with a set of particles, and the history-dependent variables such as stresses, material properties and velocities are inherited by the particles. A background mesh is employed to update the state of the particles without carrying any permanent information. Since the background mesh is fixed in space, severe mesh distortion of the traditional FE methods in large deformation problems is avoided and hence the frequent re-meshing is not necessary. Therefore the MPM has been widely used to solve large deformation problems in geotechnical engineering as an alternative to the conventional mesh-based methods, such as the triggering, runout and impact on structures of submarine landslides [10,11,21,24,26], the penetration of penetrometers [4,5] and the pull-out of anchors [6,8].

Computational efficiency is a critical issue for the MPM to simulate problems of large scales, such as transport of vast volumes of submarine sediments and cone penetration test with penetration depth of more than 20 times the diameter of the cone. The mesh used in the MPM needs to be finer than in conventional FE analysis to achieve similar accuracies, since the material particles are not always located at the optimum positions for integration within the elements [3,25]. Currently, the MPM simulations are usually limited to small scale events or within two dimensional framework. To promote the efficiency of the MPM and enlarge the modelling scales, parallel computing on CPU or single-GPU has been exploited by a limited number of researchers. Huang et al. [14] and Zhang et al. [28] developed a single-CPU parallel framework using a loop-based parallel library OpenMP. Parker [20] and Ruggirello and Schumacher [22] presented a multiple-CPU parallel framework with Message Passing Interface (MPI). And Dong et al. [9] presented a parallel strategy on a single GPU based on the Compute Unified Device Architecture (CUDA). The main difficulty in parallelising the MPM is the potential data race between the computing cores writing to the common memory address concurrently for the interpolation from particles to nodes. Huang et al. [14] and Zhang et al. [28] categorised the particles in groups by domains and parallelised the interpolation of the particles over the groups. In Parker [20] and Ruggirello and Schumacher [22], the computational domain was decomposed into subdomains; the MPM algorithm was parallelised over the subdomains, while the interpolation of the particles in each subdomain was calculated sequentially. Dong et al. [9] generated a list of the associated particles for each node, and parallelised the interpolations over the nodes.

In this paper, a multiple-GPU parallel strategy for the MPM is developed using a hybrid MPI-CUDA framework. Domain decomposition is performed with the MPI to distribute the workload over GPUs in terms of subdomains. Intercommunication between the GPUs is implemented for the neighbouring nodes and the particles moving out of the original subdomains. For each subdomain on the corresponding

* Corresponding author.
E-mail addresses: youkou.dong@tuhh.de (Y. Dong), grabe@tu-harburg.de (J. Grabe).

https://doi.org/10.1016/j.compgeo.2018.04.001
Received 5 February 2018; Received in revised form 8 March 2018; Accepted 1 April 2018

GPU, the MPM algorithm is parallelised with the CUDA based on the parallel strategy presented in Dong et al. [9] with slight improvements. Two benchmark cases, slurry runout after a dam break and surface pipe penetration, are used to evaluate the performance of the multiple-GPU parallel strategy. Acceleration of the multiple-GPU parallel simulations over the CPU sequential counterparts is quantified in terms of speedup. The overhead on the time-consuming operations is also investigated.

2. Parallelisation of MPM

To further boost the computational efficiency of the MPM on the basis of the single-GPU parallelisation in Dong et al. [9], a multiple-GPU parallel strategy of the MPM was developed using a hybrid MPI-CUDA framework. The single-GPU parallelised program in Dong et al. [9], based on an explicit integration scheme and Generalised Interpolation Material Point (GIMP) method [2], was transplanted from Windows operating system to Linux as required by the high performance computing cluster used. To validate the applicability of the multiple-GPU computing strategy, interaction between a rigid structure and soil mass was considered along with a fixed-boundary slurry runout. Different to that in Dong et al. [9], the structure was described as an analytical shape without necessity to be discretised into particles. Therefore only one set of background mesh was generated [1] and the data transfer between the CPU and GPU for the structural particles was avoided. The contact was implemented using a technique termed 'Geo-contact' in which a penalty function is incorporated into the GIMP to minimise the computational noise on contact force [18]. The definitions of the stresses and strains followed finite strain theory taking account of incremental rotation of the configurations between time steps for objectivity: the stresses were measured with the Cauchy stress and updated with the Jaumann rate, and the strains were calculated with the logarithmic strain and updated with the deformation rate.

2.1. Functions of MPM

Prior to the introduction of the parallel strategy of the MPM, the main functions within each incremental step and the manipulations involved in the parallel computing are addressed briefly.

(i) The time step starts with the function 'Initialisation of nodal variables', which initialises the nodal variables of the soil, e.g. masses, velocities, momenta and internal forces.

(ii) The function 'Interpolation from particles to nodes' is to interpolate the masses, momenta and stresses of the associated particles to the corresponding nodes, which may be summarised as

$$m_i = \sum_p S_{ip} m_p \tag{1}$$

$$M_i = \sum_p S_{ip} m_p V_p \tag{2}$$

$$f_i^{\text{int}} = -\sum_p \nabla S_{ip} \sigma_p v_p \tag{3}$$

where m_i, M_i and f_i^{int} represent the mass, momentum and internal force at node i; m_p, V_p, σ_p and v_p are the mass, velocity, stress and volume of particle p; S_{ip} and ∇S_{ip} are the shape function and its gradient at node i evaluated at particle p; $\sum_p$ represents summation over all related particles.

(iii) The function 'Calculate nodal velocities and accelerations' is to conduct the explicit calculation to obtain the velocities and accelerations on the background mesh. At the commencement of the current incremental step, the velocity at the node is

$$V_i = \frac{M_i}{m_i} \tag{4}$$

The acceleration for the current time step at node i is

$$a_i = \frac{f_i^{\text{int}}}{m_i} \tag{5}$$

Then at the end of the current time step the velocity for the node is

$$V_i^{\text{new}} = V_i + a_i \Delta t \tag{6}$$

where Δt is the time increment. For the nodes in contact with a moving rigid structure, V_i^{new} is further adjusted depending on the contact technique employed, and the contact force, f^{cont}, is updated by

$$f^{\text{cont}} = -\frac{\sum_i m_i \Delta V_i^{\text{cont}}}{\Delta t} \tag{7}$$

where ΔV_i^{cont} is the adjusted velocity at node i and $\sum_i$ represents summation over all related nodes.

(iv) In the function 'Update state of particles', the stresses and material properties of particles are calculated with a constitutive model, and the velocities and positions are updated by mapping the nodal accelerations and velocities

$$V_p^{\text{new}} = V_p + \sum_i S_{ip} a_i \Delta t \tag{8}$$

$$X_p^{\text{new}} = X_p + \sum_i S_{ip} V_i^{\text{new}} \Delta t \tag{9}$$

where X_P represents the particle coordinates at the commencement of the current step.

2.2. Parallelisation of MPM using MPI-CUDA

2.2.1. Domain decomposition and communication

Domain decomposition method is used with the MPI to distribute the entire workload onto GPUs by dividing the computational domain geometrically into a number of subdomains (Fig. 1). Each subdomain includes the particles inside and the corresponding background mesh. The parallelised MPM algorithm for each subdomain is essentially calculated on a GPU with the CUDA, while some trivial operations (such as updating the particle list of the nodes) on CPU is also required. The detailed scheme of the workload distribution between CPU and GPU can be referred to Dong et al. [9], which will also be described later for the sake of completeness with slight improvements in terms of contact strategy.

To maintain the continuity of the particle variables at the borders of the subdomains, the interpolated information on the neighbouring nodes (i.e. mass, momentum and internal force) by the function 'Interpolation from particles to nodes' at each incremental step is exchanged between the subdomains. The exchange is implemented with a MPI send/recv communication in four steps (Fig. 2): (i) read the information, i.e. mass, momentum, and internal force, of the neighbouring nodes on the global memory of the local GPU; (ii) copy to the local Random Access Memory (RAM) with CUDA via PCI-Express; (iii) copy to the target RAM with MPI send/recv communication via network card; (iv) write to the global memory of the target GPU with CUDA via PCI-Express. In step 2 and 3, the manipulation of the RAM is performed by the CPU sending commands via the front side bus. At the end of each incremental step, the positions of the particles are updated with the function 'Update state of particles'. As free to flow through the background mesh, the particles tend naturally to cross the subdomains and accordingly transit from one GPU to another. In that case, information of the crossing particles needs to be transferred from the original subdomain to the new one in similar steps of the information exchange of the neighbouring nodes.

Although the data exchanged represents a small amount of the total calculated on each GPU, the overhead on the communication is non-