22nd International Conference
ENGINEERING MECHANICS 2016
Svratka, Czech Republic, 9 – 12 May 2016

USING GPU COMPUTING FOR THE SOLUTION OF A CDEM PROBLEM

V. Vančík[*], B. Patzák[**]

Abstract: *In this paper we introduce an approach for the solution of a coupled discrete and finite element method (CDEM) designed for effective implementation on a graphics processing unit (GPU). A static problem is solved using dynamic relaxation while an optimum load-time history keeps the inertial forces small, thus eliminating the need for damping. Explicit time integration using central difference method is used. Each element is integrated independently and potential interelement contacts are enforced using penalty forces. This allows to avoid the need for forming global characteristic matrices and leads to a formulation well suited to GPU processing. To test the stability and accuracy of the proposed method, we start with a serial CPU implementation and examine a three-point bending test with the beam composed of dicrete deformable blocks (represented by traditional finite elements) connected by linear springs. The CPU code will serve as the starting point of the GPU implementation.*

Keywords: GPU computing, CDEM, dynamic relaxation, explicit integration.

1. Introduction

A GPU is a piece of hardware with a many-core architecture where all the cores have to perform the same instruction with their unique data within one instruction cycle (the SIMD approach to parallelism). The significantly higher computing performance as opposed to its CPU counterpart (see ("The CUDA C Programming Guide", 2015) for a detailed comparison of performances) has always made it attractive for scientific computations. However, since it was specifically designed for efficient processing of the graphics pipeline ("Graphics pipeline", 2016), the first attempts at using it for general-purpose computing had to deal with a lack of appropriate software development tools.

A big step towards easier development of general purpose GPU (GPGPU) code was made in 2006 with the introduction of the CUDA platform ("CUDA Toolkit Documentation", 2015) by the company nVidia. The platform introduces minor extensions for existing high-level programming languages (C/C++ and Fortran among others), which provide direct control over the GPU.

Nowadays, GPGPU computing is used to accelerate calculations in many scientific fields. The reference (Fu et al., 2014) reports an 87x speedup on the stiffness matrix assembly and an 51x speedup on the linear system solver in their GPU implementation of the FEM pipeline. The reference (Wang et al., 2013), which provided inspiration for our method, reports a speedup of up to 400 for single precision calculations. An important aspect of GPU programming is memory management. For the optimum performance, the data have to be kept in local memory, the non-local data access can be prohibitively expensive leading to high communication/computation ratio. Furthermore, an efficient algorithm obviously has to be well suited for the SIMD architecture.

2. Method of solution

The problem we focus on is a domain consisting of individual discrete finite elements with mutual contacts (See Figure 1). This approach allows for the modelling of discrete cracking, for example, but as we are still in the early testing phase, the contacts are realized as linear elastic springs. We believe that if

[*] Ing. Vladimír Vančík: Department of mechanics, Czech technical university in Prague, Faculty of civil engineering, Thákurova 7/2077; 166 29, Prague; CZ, vladimir.vancik@fsv.cvut.cz

[**] Prof. Dr. Ing. Bořek Patzák: Department of mechanics, Czech technical university in Prague, Faculty of civil engineering, Thákurova 7/2077; 166 29, Prague; CZ, borek.patzak@fsv.cvut.cz

we avoid the global stiffness matrix assembly, we can come up with an efficient GPU algorithm for the solution of the described problem. This is why we choose to use dynamic relaxation with explicit time integration to solve the linear static problem. Using the explicit integration allows us to calculate the internal forces for each element individually. The only information needed from the rest of the domain are the displacements

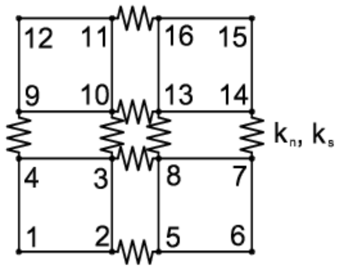

Figure 1: A sketch of the CDEM domain with numbered nodes. The values k_n and k_s denote the normal and shear (tangential) contact stiffness, respectively.

of the neighbor nodes (the nodes connected to the element's nodes by springs). In this study, we consider $2D$ plane stress elements. The element stiffness forces are obtained as

$$\boldsymbol{f_{es}} = -\boldsymbol{K_e u_e}$$

and the damping forces as

$$\boldsymbol{f_{da}} = -\boldsymbol{C\dot{u}_e}$$

where $\boldsymbol{u_e}$ is the vector of nodal displacements of the element, $\boldsymbol{\dot{u}_e}$ the vector of nodal velocities, $\boldsymbol{K_e}$ the element stiffness matrix and C the damping matrix. In this study, we model dumping using Rayleigh model, where the damping matrix is expressed as a mass matrix multiplied by a so called Rayleigh coefficient α_c. The contact stiffness forces, f_{cs}, are obtained node by node. In the simple case of full face contact, each node can have a maximum of two neighbors correspondingly to its two adjoining element edges. The contact stiffness is characterized by the local contact stiffness matrix

$$\boldsymbol{K_{c,l}} = \begin{pmatrix} k_n & 0 \\ 0 & k_s \end{pmatrix}$$

where the values $\mathrm{k_n}$ and $\mathrm{k_s}$ denote the normal and shear (tangential) contact stiffness, respectively. The transformation matrix is defined based on the normal vector $\boldsymbol{n}$ of the appropriate element edge as

$$\boldsymbol{T} = \begin{pmatrix} n_1 & n_2 \\ -n_2 & n_1 \end{pmatrix}$$

with n_1, n_2 being the components of the normal vector in the global coordinate system. The contact forces from each neighbor can now be calculated as

$$\boldsymbol{f_{cs,n}} = \boldsymbol{T^T K_{e,l} T \Delta u}$$

where $\boldsymbol{\Delta u} = \boldsymbol{u_{neighbor}} - \boldsymbol{u_{node}}$ is the relative displacement of the node and its neighbor in the global coordinate system. The contributions from the individual neighbors can be assembled into the element-level vector of contact forces $\boldsymbol{f_{cs}}$. The next step is to calculate the inertial forces from the force balance equation as

$$\boldsymbol{f_{in}} = \boldsymbol{f_{es}} + \boldsymbol{f_{cs}} + \boldsymbol{f_{da}} + \boldsymbol{f_{ex}} + \boldsymbol{f_{re}}$$

where the vectors $\boldsymbol{f_{ex}}$ and $\boldsymbol{f_{re}}$ represent the external forces (load) and reactions, respectively. Finally, we get the nodal accelerations for the next time step as

$$\boldsymbol{\ddot{u}_{e,i+1}} = \boldsymbol{M^{-1} f_{in}}$$

where M is the lumped mass matrix.

We examine two possible approaches to obtaining the static solution with this method. First option is to apply the load all at once and examine a damped system until the transient dynamic part of the solution disappears. Alternatively, we can use the optimum load-time function proposed in the paper (Řeřicha, 1986), which should reach the static solution exactly at the end of loading, the error depending on the ratio of the natural frequency of the system and the total time of loading.

3. Test problem – three-point bending

As a testing example, we use a model of a beam with the length of $2\ m$, square cross section with the side of $0.2\ m$, Young's modulus $E = 25\ GPa$, Poisson's ratio $\nu = 0.2$ and density $\rho = 2500\ kg/m^3$. The beam is loaded in the center by a force of $400\ kN$.

The difference between a CDEM solution with springs and a standard FEM solution with the elements sharing nodes is influenced by the stiffness of the springs – higher stiffness means lower difference but also limits the length of stable time step. The natural period of the beam can be deduced from the time-deflection curve of the undamped solution seen in Figure 2. The figure further compares the curves of two solutions with a load function and a solution with impulse load and a damping factor $\alpha_c = 1000$. We see that the accuracy of the solution with load function increases for a longer loading time of double the natural period.

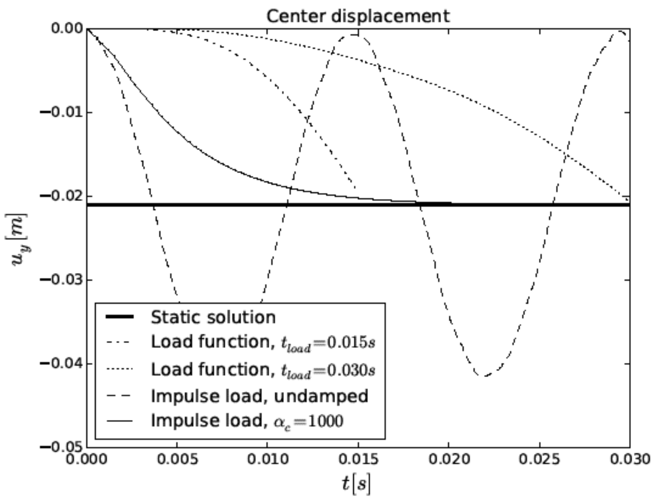

Figure 2: Deflection-time curves for various loading strategies for a beam with the spring stiffness factor of 10. Length of the time step is $1.5 * 10^{-8}$.

Following is the C++ code sample of the explicit integration as a loop across single degrees of freedom, ready for GPU parallelization.

```
for (int k = 1; k <= maxiter; k++) { // Loop of time steps
loadfunc = load_function(k*dt / t_load);
for (i = 0; i < nnodedofs*nnodes; i++) {
// Loop of dofs - increment displacement and velocity
  u[i] += dt*v[i] + 0.5*dt*dt*a[i];
  v[i] += dt*a[i];
}
for (i = 0; i < nnodedofs*nnodes; i++) {
// Loop of dofs - Calculate balance of forces and resulting acceleration
  int eid = i / stiffdim; // global number of element
  int nid = (i / nnodedofs) * nnodedofs; // number of dof 1 of this node
  int ned = i % stiffdim; // number of the dof within element
  int mdim = stiffdim*stiffdim; // no of elements of the stiffness matrix
  double kc11 = Kc[0], kc21 = Kc[1], kc12 = Kc[2], kc22 = Kc[3];

  // Element stiffness force:
  double F_k_e = 0;
  for (j = 0; j < stiffdim; j++) {
    F_k_e += -K[eid*mdim + j*stiffdim + ned] * u[eid*stiffdim + j];
  }
  // Contact stiffness force:
  double F_k_c = 0;
  for (j = 0; j < 2; j++) {
    int nbr = neighbors[nid + j];
    if (nbr != 0) {
      double t11 = n_vects[4 * (i / nnodedofs) + 2 * j];
      double t12 = n_vects[4 * (i / nnodedofs) + 2 * j + 1];
      double t21 = -t12, t22 = t11;
      double du_x = u[(nbr - 1)*nnodedofs] - u[nid];
      double du_y = u[(nbr - 1)*nnodedofs + 1] - u[nid + 1];

      if (i == nid) { // X-component
F_k_c += du_x * (t11*(t11*kc11 + t21*kc21) + t21*(t11*kc12 + t21*kc22)) +
        du_y * (t12*(t11*kc11 + t21*kc21) + t22*(t11*kc12 + t21*kc22));
// T_T * Kc * T * du_g

      } else { // Y-component
F_k_c += du_x * (t11*(t12*kc11 + t22*kc21) + t21*(t12*kc12 + t22*kc22)) +
        du_y * (t12*(t12*kc11 + t22*kc21) + t22*(t12*kc12 + t22*kc22));
// T_T * Kc * T * du_g
      }
    }
  }
  double F_c = -C[i] * v[i]; // Damping force
  // Reaction force - in supports, 1 means fixed dof and 0 means free dof
  double F_r = supports[i] * (-F_k_e - F_k_c - F_c - loadfunc*load[i]);
  z[i] = F_k_e + F_k_c + F_r + F_c;
  a[i] = Mi[i] * (z[i] + loadfunc*load[i]);
}
```

4. Conclusions

Two approaches to a dynamic solution of a linear static problem were compared. Although we could achieve an accurate solution, this was done with exclusive tuning of the calculation parameters (namely the time step length). Before we move on to the GPU implementation, we have yet to come up with a reliable method of estimating a stable time step length in order for the resulting code to be able to process arbitrary input.

Acknowledgements

Financial support received from the Grant Agency of the Czech Technical University in Prague under SGS project submitted with the number OHK1-093/16 is gratefully acknowledged.

References

CUDA Toolkit Documentation (2015). [Online]. Retrieved 2016-02-11 from http://docs.nvidia.com/cuda/

Fu, Z., James Lewis, T., Kirby, R. & Whitaker, R.. (2014). Architecting the finite element method pipeline for the GPU. *Journal of Computational and Applied Mathematics*, *vol. 257*, 195-211. http://doi.org/10.1016/j.cam.2013.09.001

Graphics pipeline (2016). [Online]. Retrieved 2016-02-11 from https://en.wikipedia.org/wiki/Graphics_pipeline

Řeřicha, P. (1986). Optimum load time history for non-linear analysis using dynamic relaxation. *International Journal for Numerical Methods in Engineering*, *23*(12), 2313-2324.

The CUDA C Programming Guide (2015). [Online]. Retrieved 2016-02-11 from http://docs.nvidia.com/cuda/cuda-c-programming-guide/

Wang, L., Li, S., Zhang, G., Ma, Z. & Zhang, L.. (2013). A GPU-Based Parallel Procedure for Nonlinear Analysis of Complex Structures Using a Coupled FEM/DEM Approach. *Mathematical Problems in Engineering*, *vol. 2013*, 1-15. http://doi.org/10.1155/2013/618980