Vol.2011-HPC-130 No.12
2011/7/27

# GPU-based approach for elastic-plastic deformation simulations

Irina Demeshko,[†1] Matsuoka Satoshi[†1]
and Toshio Endo[†1]

In this article, we describe GPU-based parallel algorithm for the numerical simulation of elastic-plastic deformations by finite element analysis method (FEM). This kind of mechanical problems gives us non-symmetric bonded structure of the stiffness matrix, which requires some optimization on data assignments and assembling techniques. Described algorithm was implemented on TSUBAME2 supercomputer using CUDA programming environment.

## 1. Introduction

Dramatic increase in the amount of computations, necessary for the solution of scientific problems, leads to strong needs to look for a way to accelerate a computation time.

In recent years graphic processor unit (GPU) computing has been used to accelerate calculations performed on a classical central processor unit (CPU). GPU have been used for acceleration a wide range of scientific problems from earth and weather simulation to such physical problems as fluid dynamics or astrophysical simulations.

In this paper we describe the GPU-mapping algorithm for the numerical simulation of elastic-plastic problem with big plastic deformations by FEM.

The solution is based on the principle of virtual power in the speed form with the finite-element approximation, which results in the solution of linear algebraic equations system with banded non-symmetric matrix. Due to the fact that we have both elastic and plastic deformations, structure of the stiffness matrix is non-symmetric, that make the problem two times more computations and data intensive comparing to the symmetric case.

Banded structure of the matrix forced us to think about some changes in data storage model, which helps to reduce amount of memory and computation time.

†1 Tokyo Institute of Technology

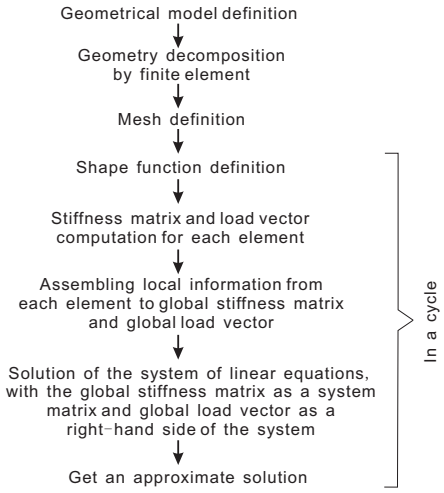


**Fig. 1** General FEM algorithm.

## 2. Common FEM algorithm

The standard procedure of FEM computation consists of obtaining week formulation of a problem, domain decomposition into finite elements, definition of a system of linear equations with the global stiffness matrix as the system matrix, by using basis functions, constructed from element shape functions and solving a system to obtain approximate solution.

More in detail this algorithm is presented in **Fig. 1**.

To find a solution $u$ for non-linear problems iterative strategies, such as a Newton-Raphson, are often used. Matrix $A$ depends on the value of each approximate solution $u$ and needs to be assembled on each iteration to solve the problem.

Two stages of this FEM most significantly impact performance of the algorithm:

- computation of global stiffness matrix $A$ and vector $F$
- solution of the system of linear algebraic equations (LAES) for $u$

The algorithm for an acceleration of these two stages by using CUDA programming environment is described below.

## 3. Related work

Recently the number of the numerical simulation problems, which have been mapped to GPU, starts significantly increasing.

Algorithms, such as finite-difference and finite-volume methods, are show large accelerations because they involve regular memory access pattern. On the other hand, finite-element method is more difficult to accelerate since they tend to have irregular memory access.

Dimitri Komatitsch and his co-workers [1, 2] have successfully implemented their finite-element algorithm on a CPU cluster enhanced by GPUs. The algorithm is based on the dividing elements to the block, by using coloring algorithm, and then computing blocks concurrently. They implemented it on the problem of numerical modeling of seismic wave propagation.

Comparison of CUDA and OpenCL techniques for FEM GPU implementation is presented in [4]

In [2] are presented several approaches for the implementation of the Global stiffness matrix assembling step on GPU. Authors compared next approaches: using coloring algorithm for element discretization, global memory algorithm shared memory algorithm and local memory algorithm. It was shown that the algorithm, based on using shared memory, has better performance.

In our paper we presented algorithm for parallelization FEM based on mapping to the GPUs local matrices computation and the solution of LAES steps. Our CUDA local matrices computation algorithm is similar to algorithms, described in [2], but modified for banded matrices. CUDA algorithm for the solution of LAES is based on iterative method parallelization.

There are also some other finite-element implementations for GPU not described here.

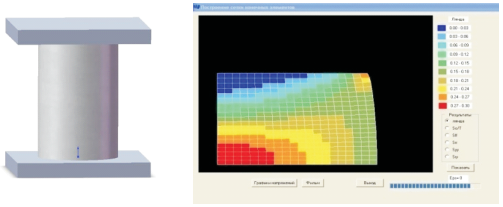

**Fig. 2** Simulation of elastic-plastic cylinder compression problem

## 4. Problem statement

As an example, we consider 2-dimensional problem of cylinder compression from an elastic-plastic isotropic and isotropic-strengthened material by flat plates (see **Fig. 2** ). The solution is based on a principle of virtual power in the speed form (1):

$$\int_V (\sigma + \Delta t\dot{\sigma}) \cdot \cdot \nabla h \, dV + \int_\Sigma (P + \Delta t\dot{P}) \cdot h d\Sigma = 0$$

With constitutive equations :

$$\dot{\sigma} = \lambda\dot{\Theta} + 2(\lambda\Theta + \mu)D - \nabla v \cdot \sigma - \sigma \cdot \nabla v^T - JbS,$$

$$S = \sigma - \sigma_0 I,$$

$$\sigma_0 = K\Theta$$

$$K = \lambda + 2/3\mu,$$

$$D = 0.5(\nabla v + \nabla v^T),$$

$$F(S, k) = 0.5S \cdot \cdot S - k^2 = 0,$$

$$b = \frac{\left[\left[\mu\left(1-\frac{2}{3}\Theta\right)S - S\cdot S\right]\cdot\cdot D - \frac{\mu}{3}t\dot{\Theta}S\cdot\cdot I\right]}{\left[k^2\left(1+\frac{1}{\mu}\frac{dk}{d\chi}\right)\right]}$$

Here $\sigma$ - is Cauchy's strain tensor; $P$ - surface force density; $\nabla t$ - period for an increment interval of load; $h$ and $\nabla h$ - kinematically admissible velocity fields variation and its inverted delta; $V$ , $\sum$- solid volume and surface; $dV$, $d\sum$ - volume and cylinder surface area elements.

The Coulomb's law was applied on contact with plates. A lateral surface of the cylinder is free from loadings.

We use finite-element discretization by rectangular elements.

The FEM algorithm for our problem is based on dividing the load by time iterations and computing of approximation solution for each load increment. About 1000-1500 time steps are necessary to get a satisfactory solution for described type of mechanical problems. Also, on each time step we perform about 10 computational iterations to get refined solution.

By means of finite element approximation the equation (1) is leads to the linear algebraic equation system (LAES):

$$Az = b,$$

where $A$, $b$, $z$ - parent matrix, right part vector and decision vector of the system, respectively. Here $A$ is the asymmetrical sparse matrix, which is expressed as a banded matrix in this paper.

To exclude unnecessary computation and reduce the amount of memory to be used, we change square matrix $A\ NxN$ to the band form matrix $A'\ NxW$, where $W$ - is the width of the band (see **Fig. 3**).

## 5. GPU algorithm

Global matrix computation and solution of the systems of linear equations resulting from FEM are the most computationally demanding steps of the method and, therefore we mapped these steps to GPU. As these steps are need to be computed sequentially, we realized them in 2 different CUDA kernels. Our GPU algorithm in general is presented in **Fig. 4**.

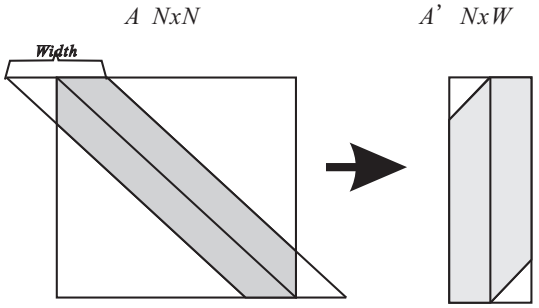


**Fig. 3** matrix $A$ transformation

A CUDA application consist of a sequential host program run on the CPU (white blocks in Fig. 4) that launches kernel 1 and kernel 2 (gray blocks in the Fig. 4) written in CUDA to be run on the parallel GPU device.

### 5.1 Sequential local matrices computation

Local matrices computation step in FEM consists of 3 sub-steps: shape function definition, stiffness matrix and load vector computation for each element, assembling local information from each element to global stiffness matrix and global load vector.

Typical local matrices and local load vectors computation algorithm based on given element subroutines to compute an element matrix $A^e$ and element forcing vector $F^e$.

Each element matrix $A^e$ has size $8x8$. After local matrices are computed for all elements, they are assembling to the Global stiffness matrix. This assembling is based on using connectivity matrix, which yields the local nodes numbers for the $e^{th}$ element to the global nodes numbers.

These element subroutines changes according to partial differential equation, the basis functions, the element type, boundary conditions and forces.

The input information is nodal data, contained in $N$ for each node in element.

### 5.2 local matrices computation on a GPU

Local stiffness matrices and load vectors are calculated independently for each element and therefore can be carried out concurrently.

Calculations on GPU are managed by CPU. General algorithm for the kernel

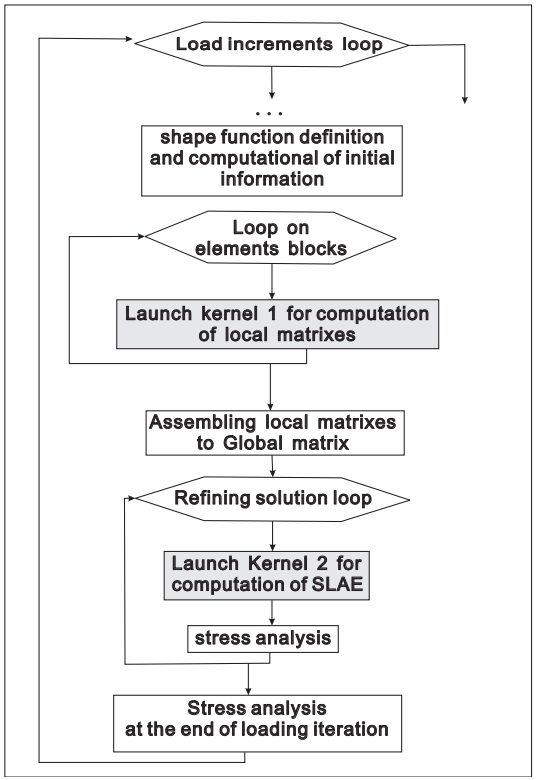


**Fig. 4** General CUDA algorithm for FEM. The computations, performs on CPU, are in the white blocks. GPU computations are colored in grey

1 can be described by 3 steps:

- uploading data into device
- local matrices and load vectors computations on GPU
- copy to CPU computed local matrices and load vectors

Data that are common for all elements is calculated once on CPU in precondition computations step.

Our approach is assigns one thread to compute the element data for one mesh element and, for the purpose to avoid race conditions, write the element data to global data for later assembling to the Global matrix and load vector.

In the case that the global memory cannot hold all the element data, the calculations can be divided into multiple passes. Each pass would compute the element data for some block of elements and copy result to CPU in precisely the same way.

More in detail the algorithm is presented in **Fig. 5**

Because of the small size of the matrices involved, this kernel performs a relatively large number of memory access compared to small amount computations.

As an optimization we group the input data into a single buffer to minimize uploading time by execution of one large CPU-GPU transfer instead of many small ones. Due to the fact that each thread responsible for one grid element we prevent any possible race conditions.

### 5.3 Iterative methods for the solution of LAES

Next iterative methods for the solution of LAES were compared in preliminary experiments with CPUs:

- Prime Iteration method
- Minimal residual method
- Steepest descent method
- Biconjugate Gradient method ( Conjugate Gradient Type Method with some precondition computations for non-symmetric matrices)

Prime Iteration method, Minimal residual method and Steepest descent method shows similar wall time results. Biconjugate Gradient method requires more iterations than the other examined methods and therefore needs more time for solving of linear algebraic equations.

Comparing scalability of methods, we found that Minimal residual method has

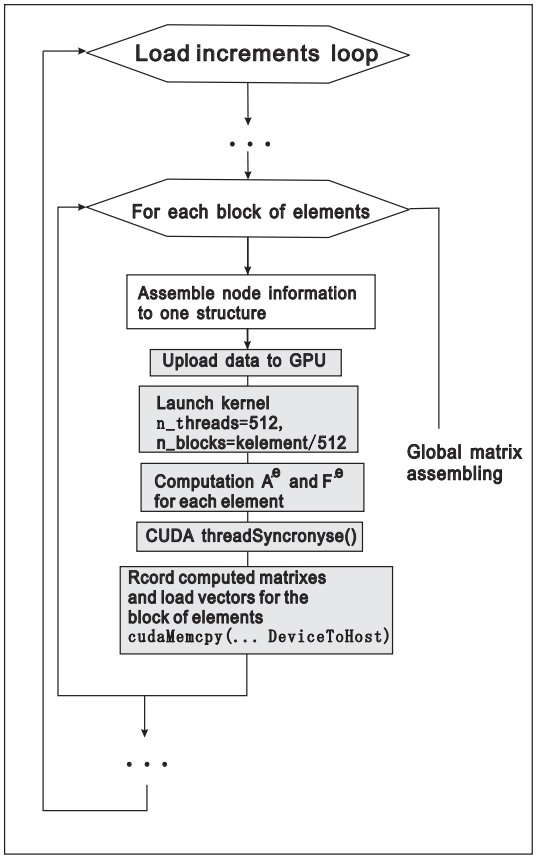


**Fig. 5** CUDA algorithm for the step of local matrices and local load vectors computation

potential better performance. Therefore, this method was chosen for the GPU implementation.

### 5.4 CUDA algorithm for the solution of LAES

Minimal residual method is based on iterative computation of next approximation solution vector $Z^{k+1}$, by using already known vector $Z^k$ (2):

$$z^{k+1} = z^k - \frac{(A(Az^k - b), Az^k - b)}{||(A(Az^k - b)||^2}(Az^k - b),$$

$$z^0 = 0$$

this iterations performs till $||AZ - b||/||B|| <= \epsilon$, where $\epsilon$ is accuracy of the solution.

It can be observed from the formula 2 that the base of the numerical computations in iterative method is matrix-vector multiplication. Therefore our algorithm is based on performing matrix-vector multiplication on GPU.

Due to specific of the matrix $A$ structure (non-symmetric bounded) general matrix multiplication CUDA algorithm was modified with the purpose to not perform unnecessary computations.

Some other optimization was applied for this kernel. Since the shared memory is banked, with the purpose to avoid bank conflict each thread accesses a different bank. This leads to the coalesced memory transactions. It is realized by dividing the non-zero bound of stiffness matrix to the set of square submatrices, which sizes are equal to the number of threads in one block.

## 6. Evaluation environment

Described GPU-mapping algorithm was implemented and evaluated on the TSUBAME 2.0 supercomputer, established at Tokyo Institute of Technology. TSUBAME 2.0 consist of 1408 compute nodes of two Intel Xeon Westreme-EP 2.9 GHz CPUs and three NVIDIA M2050 GPUs with 52GB and 3GB of system and GPU memory, running SUSE Linux Enterprise Server 11 SP1. Each node is interconnected by dual QDR Infinitband networks with a full bisection-bandwidth fat-tree topology. We use NVIDIA CUDA v 3.2 compiler for the GPU code and g++ v4.1.2 compiler for the CPU code.

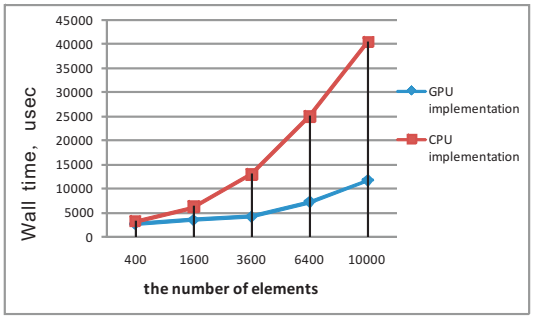


**Fig. 6** Average running time as a function of the number of elements for the CPU and GPU implementations of the step for local matrices an local load vectors computation

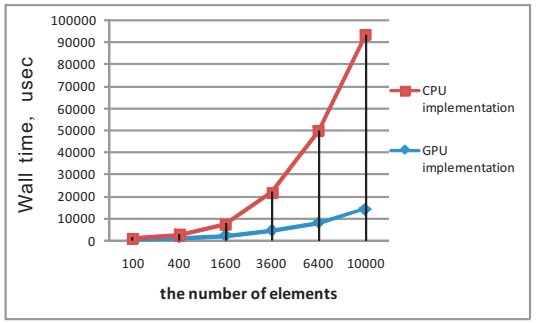


**Fig. 7** Average running time as a function of the number of elements for the CPU and GPU implementations of the LAES solution step

The algorithm was applied to the elastic-plastic cylinder compression problem. Results were compared with sequential CPU implementation for different grid sizes.

## 7. Performance evaluation

The running times of described GPU algorithms are shown in **Fig. 6** and **Fig. 7**. These times measured in wall time and include GPU call overheads, but do not include precomputational time of the program. To ensure accurate timing *cudaThreadSynchronize* is called after CUDA kernels.

Fig. 6 presents average running time as a function of the number of elements for the first CUDA kernel in our problem (CUDA local matrices and local load vectors computation algorithm).

It is shown that the CPU implementation runtime almost linearly increases as the number of elements continues increase due to the fact computation time for one element is almost the same for any grid dimension. The GPU implementation shows about 3.5 times faster speed with 3600 elements or more(see **Fig. 8**). However the speed up is smaller when we have less elements. We consider that this due to (1) CUDA kernel invocation overhead dominates, and (2) parallelism is insufficient to keep all CUDA cores busy.

Fig. 7 presents average running time as a function of the number of elements for CUDA kernel 2 (CUDA LAES computation algorithm). The time is measured

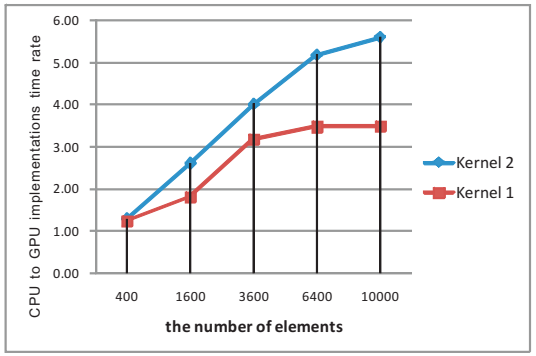


**Fig. 8** The rate of CPU to CUDA GPU implementations times for the described algorithms.

for one iteration, becouse for different matrices the number of iterations can be differen. Similar to the local matrices and local load vectors computation algorithm, CPU implementation runtime increases almost linearly to the increases number of elements. The CUDA implementation shows about 5.2 times faster speed with 6400 elements or more(see Fig. 8). Similar to Kernel 1 for smaller number of elements the speed up is lower because of the same reasons.

From the Fig. 8 we can see that GPU to CPU implementations acceleration higher for the Kernel2 comparing to the Kernel1. It can be explained by the fact that the rate of computations, performed on GPU to ones on CPU for the

Vol.2011-HPC-130 No.12
2011/7/27

CUDA LAES computation algorithm is higher then for the CUDA linear matrices computations algorithm.

## 8. Conclutions and future work

In this paper we investigate the use of single GPU to accelerate FEM for elastic-plastic mechanical problems.

We propose to use two independent CUDA kernels to accelerate the most time consuming steps in the FEM algorithm: computation of local matrices an local load vectors for all elements and solution of the system of linear equations with banded non-symmetric matrix, which is produced in the FEM discretization.

Described algorithms were implemented on the TSUBAME 2.0 supercomputer, established at Tokyo Institute of Technology.

Performance evaluation shows a significant acceleration comparing to single CPU implementation.

As future work we would like to modify our single-GPU approach to multi-GPU. Also we are going to investigate the possibility to map our sequential Global matrix assembling step to GPU approach.

## References

1) Dimitri Komatitsch, Gordon Erlebacher, Dominik Goddeke and David Michea:High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster, *Journal of Computational Physics*, Vol. 229, pp. 7692-7714 (2010).
2) Cristopher Cecka, Adrian J. Lew and Eric Darve: Asembly of finite element methods on graphics processors, *International Journal for Numerical Methods in Engineering*, Vol. 85, No 5, pp.640–669 (2011).
3) Dimitri Komatitsch, David Michea and Gordon Erlebacher: Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA, *Journal of Parallel and Distributed Computing*, Vol.69, No 5, pp.451-460, DOI:10.1016/j.jpdc.2009.01.006 (2009).
4) Dimitri Komatitsch, David Michea and Gordon Erlebacher: Higher order FEM numerical integration on GPUs with OpenCL, *IMCSIT*, pp.337-342 (2010).