Advances in Engineering Software 79 (2015) 27–35

Contents lists available at ScienceDirect

Advances in Engineering Software

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

GPU-based acceleration of computations in elasticity problems solving by parametric integral equations system

Andrzej Kuzelewski *, Eugeniusz Zieniuk

Faculty of Mathematics and Informatics, University in Bialystok, Sosnowa 64, 15-887 Bialystok, Poland

ARTICLE INFO

Article history:
Received 9 July 2014
Received in revised form 2 September 2014
Accepted 16 September 2014
Available online 8 October 2014

Keywords:
Parametric integral equations system
Elasticity problems
Graphics Processing Unit
GPGPU
CUDA
Multithreaded computing

ABSTRACT

Application of techniques for modelling of boundary value problems implies two conflicting requirements: obtaining high accuracy of the results and speed of the solution. Accurate results can be obtained only by using appropriate models and algorithms. In the previous papers the authors applied the parametric integral equations system (PIES) in modelling and solving boundary value problems. The first requirement was satisfied – the results were obtained with very high accuracy. This paper fulfils the second requirement by novel approach to accelerate PIES. Graphics cards (GPUs) programming for numerical calculations in general purpose applications (GPGPU) using NVIDIA CUDA is used for this purpose. The speed of calculations increased up to 80 times whereas high accuracy of the solutions was maintained. Examples included in this paper concern solving elasticity problems which are modelled by three-dimensional Navier–Lamé equations.

1. Introduction

For many years the authors of this paper have being worked on development and application of parametric integral equations system (PIES) to solve boundary value problems. PIES has already been used to solve problems modelled by 2D and 3D partial differential equations, such as: Laplace's [1,2], Helmholtz [3] or Navier–Lamé [4–6]. The remarkable advantage of PIES, compared with traditional boundary integral equation (BIE), is direct inclusion in its mathematical formalism a shape of boundary of a considered problem [7]. The shape of boundary is generally defined using particular functions. For this purpose, the curves (eg. Bézier, B-spline, etc.) or surface patches (Coons, Bézier, and others) widely used in computer graphics, were applied in PIES. PIES is an analytical modification of traditional BIE. The above mentioned curves and surface patches are applied in modelling the shape of the boundary, instead of the contour integral as in the case of BIE. Therefore in practice, a small number of control points is required to define a shape of the boundary. This way of modelling is definitely much easier in comparison with the other methods: boundary element method (BEM) [8] or finite element method (FEM) [9]. Moreover, the accuracy of solutions can be efficiently improved without interference in modelling of a shape of boundary.

The former studies focused on high accuracy and efficiency of the results obtained using PIES, which were compared to the other algorithms (FEM or BEM) as well as analytical methods. These studies confirmed the high effectiveness and accuracy of PIES in solving 2D [1,4] and 3D [2,5,6] boundary value problems. However, the problem of increasing computing time of complex numerical algorithms forced the authors' attention, as well. In general, the greater number of input data, the longer an algorithm works. It was noticed in case of some more complex problems solved by PIES (eg. modelled by Navier–Lamé equations or increasing the dimensionality of the problem from 2D to 3D). PIES uses the strategy of global numerical integration on the entire surface patches. This integration does not require the splitting of patches on smaller elements, however it is necessary to use a large number of weight coefficients in cubatures in order to obtain high accuracy of solutions. The number of these coefficients has a significant influence on the duration of the implementation of PIES. Acceleration of the algorithm can be achieved in the following ways: by optimisation of the existing code as well as use of more advanced computers, multiprocessor machines, clusters or GPUs.

Recently, many researchers in various fields have increased their attention on the implementation of graphics cards (GPUs) for numerical calculations in general-purpose applications (GPGPU). Due to great possibility of improving computing performance, scientists have already investigated application of the mentioned technology in problems in the following areas: applied mathematics [10–12], physics [13], biology and medicine [14,15], seismic research [16,17], peridynamics [18], and others. Numerical modelling and solving boundary problems were accelerated using the modern techniques of graphic cards programming adapted to

* Corresponding author.
E-mail address: akuzel@ii.uwb.edu.pl (A. Kuzelewski).

http://dx.doi.org/10.1016/j.advengsoft.2014.09.003
0965-9978/

BEM [19,20], FEM [21,22], finite difference method (FDM) [23] or meshless methods [24,25], as well. The interests of scientists are related to the modern architecture of graphics cards (multiprocessor and multithreaded), very fast floating-point arithmetic units, use of high-speed memory and, first of all, ease of programming. In November 2006, NVIDIA has produced the first series of graphics cards that employ a new parallel computing platform and programming model called Compute Unified Device Architecture (CUDA) [26]. It was a break-through in programming and feasibility of general-purpose applications executed on GPU. CUDA C programming language is an extension to the C/C++ and definitely simplifies writing of non-graphical programs compared with graphics-oriented languages such as Cg, GLSL or HLSL.

The architecture of graphics processing units (GPUs) significantly differs from central processing units (CPUs). GPU is composed of multiple floating point units (FPU) and arithmetic and logic units (ALUs). It is connected with the nature of performed calculations – the same operations are executed in parallel on large amounts of data. Using a lot of pixels, texels or vertices is typical in graphics applications, therefore GPUs are rated as SIMT (single instruction, multiple thread).

CUDA-enabled NVIDIA GPUs are multithreaded, parallel, many-core processors. They are composed of a set of multithreaded streaming multiprocessors (SMs). Each multiprocessor contains a number of general-purpose streaming processors cores (SPs), multithreaded instruction unit and on-chip shared memory. Also, it comprises a number of 32-bit registers, a texture cache and a read-only constant cache. SPs can compute one arithmetic multiply or add operation per clock cycle. In the case of multiply-and-add operation it is equivalent to two operations per clock cycle. The execution of other instructions require greater number of clock cycles (eg. square root is computed per 10 clock cycles).

GPU is directly connected to a read-write off-chip device memory (DRAM). It can store a large amount of data according to the type of applied hardware (hundreds megabytes to few gigabytes). CPU exchanges data with GPU via host and device memory. The main disadvantage of device memory is its latency. SM requires about 400–600 clock cycles for access DRAM, whilst SP takes only 4 clock cycle for accessing to registers or shared memory. More detailed studies are presented in [27].

GPU works in close connection with CPU – it operates as an additional processor attached to CPU. GPU performs operations assigned to it by the application that must start running on CPU. Only a part of original program, which requires paralellization, should be recoded. Serial part of CUDA application code runs on host (CPU) and parallel part on CUDA device (GPU). The set of all functions performed on host is called host program, while functions performed on device are called kernels. Host program is responsible for initiation and transfer of data to/from device memory. During execution of a program the host calls kernels, as well. Data flow in CUDA is divided into four basic steps: 1. initiation of the program (host), 2. copying data from host to device, 3. calculations on GPU, 4. copying data from device to host.

In this paper, the authors decided to examine the application of modern parallel computing solutions to increase the efficiency of calculations in the numerical solution of PIES. Numerical calculations were performed for three dimensional Navier–Lamé equations using CUDA-enabled NVIDIA GPU.

2. PIES for three-dimensional Navier–Lamé equations

PIES for three-dimensional Navier–Lamé equations was obtained as the result of analytical modification of BIE. Detailed studies of the methodology of modification for two-dimensional problems modelled by various differential equations are presented in [1,4]. Generalization of the mentioned methodology to three-dimensional problems with smooth boundary results in the following formula of PIES [5,7]:

$$0.5\mathbf{u}_l(v_1,w_1)=\sum_{j=1}^{n}\int_{\bar{v}_{j-1}}^{\bar{v}_j}\int_{\bar{w}_{j-1}}^{\bar{w}_j}\left\{\overline{\mathbf{U}}_{lj}^{*}(v_1,w_1,v,w)\mathbf{p}_j(v,w)-\overline{\mathbf{P}}_{lj}^{*}(v_1,w_1,v,w)\mathbf{u}_j(v,w)\right\}J_j(v,w)\,dv\,dw, \tag{1}$$

where $\bar{v}_{l-1}\leqslant v_1\leqslant\bar{v}_l$, $\bar{w}_{l-1}\leqslant w_1\leqslant\bar{w}_l$, $\bar{v}_{j-1}\leqslant v\leqslant\bar{v}_j$, $\bar{w}_{j-1}\leqslant w\leqslant\bar{w}_j$, $\{l,j\}=1,2,\ldots,n$, n – is the number of parametric patches that create the domain boundary in 3D, whilst function $J_j(v,w)$ is the Jacobian.

Integrands $\overline{\mathbf{U}}_{lj}^{*}(v_1,w_1,v,w)$ and $\overline{\mathbf{P}}_{lj}^{*}(v_1,w_1,v,w)$ in (1) are presented in the following matrix form [5,7]:

$$\overline{\mathbf{U}}_{lj}^{*}(v_1,w_1,v,w)=\frac{1}{16\pi(1-\nu)\mu\eta}\begin{bmatrix}U_{11}&U_{12}&U_{13}\\U_{21}&U_{22}&U_{23}\\U_{31}&U_{32}&U_{33}\end{bmatrix},\quad \mu=\frac{E}{2(1+\nu)}, \tag{2}$$

$$\overline{\mathbf{P}}_{lj}^{*}(v_1,w_1,v,w)=\frac{1}{8\pi(1-\nu)\eta^2}\begin{bmatrix}P_{11}&P_{12}&P_{13}\\P_{21}&P_{22}&P_{23}\\P_{31}&P_{32}&P_{33}\end{bmatrix}. \tag{3}$$

The individual elements in the matrix (2) in an explicit form are presented as follows:

$$U_{11}=(3-4\nu)+\frac{\eta_1^2}{\eta^2},\quad U_{12}=\frac{\eta_1\eta_2}{\eta^2},\quad U_{13}=\frac{\eta_1\eta_3}{\eta^2},$$
$$U_{21}=\frac{\eta_2\eta_1}{\eta^2},\quad U_{22}=(3-4\nu)+\frac{\eta_2^2}{\eta^2},\quad U_{23}=\frac{\eta_2\eta_3}{\eta^2},$$
$$U_{31}=\frac{\eta_3\eta_1}{\eta^2},\quad U_{32}=\frac{\eta_3\eta_2}{\eta^2},\quad U_{33}=(3-4\nu)+\frac{\eta_3^2}{\eta^2},$$

while in the matrix (3):

$$P_{11}=\left((1-2\nu)+3\frac{\eta_1^2}{\eta^2}\right)\frac{\partial\eta}{\partial n},\quad P_{12}=3\frac{\eta_1\eta_2}{\eta^2}\frac{\partial\eta}{\partial n}-(1-2\nu)\frac{\eta_1n_2-\eta_2n_1}{\eta},$$
$$P_{13}=3\frac{\eta_1\eta_3}{\eta^2}\frac{\partial\eta}{\partial n}-(1-2\nu)\frac{\eta_1n_3-\eta_3n_1}{\eta},$$
$$P_{21}=3\frac{\eta_2\eta_1}{\eta^2}\frac{\partial\eta}{\partial n}-(1-2\nu)\frac{\eta_2n_1-\eta_1n_2}{\eta},$$
$$P_{22}=\left((1-2\nu)+3\frac{\eta_2^2}{\eta^2}\right)\frac{\partial\eta}{\partial n},\quad P_{23}=3\frac{\eta_2\eta_3}{\eta^2}\frac{\partial\eta}{\partial n}-(1-2\nu)\frac{\eta_2n_3-\eta_3n_2}{\eta},$$
$$P_{31}=3\frac{\eta_3\eta_1}{\eta^2}\frac{\partial\eta}{\partial n}-(1-2\nu)\frac{\eta_3n_1-\eta_1n_3}{\eta},$$
$$P_{32}=3\frac{\eta_3\eta_2}{\eta^2}\frac{\partial\eta}{\partial n}-(1-2\nu)\frac{\eta_3n_2-\eta_2n_3}{\eta},$$
$$P_{33}=\left((1-2\nu)+3\frac{\eta_3^2}{\eta^2}\right)\frac{\partial\eta}{\partial n},$$

wherein $n_1\equiv n_1(v,w)$, $n_2\equiv n_2(v,w)$, $n_3\equiv n_3(v,w)$ are the components of $\mathbf{n}_j$ – the normal vector to the surface j. Parameters ν and E are material constants: Poisson's ratio and Young modulus respectively.

Integrands (2) and (3) include in their mathematical formalism the shape of a closed boundary surface. It is created using appropriate relationships between patches ($\{l,j\}=1,2,\ldots,n$), which are defined in Cartesian coordinates system as follows:

$$\eta_1=P_j^{(1)}(v,w)-P_l^{(1)}(v_1,w_1),\quad \eta_2=P_j^{(2)}(v,w)-P_l^{(2)}(v_1,w_1),$$
$$\eta_3=P_j^{(3)}(v,w)-P_l^{(3)}(v_1,w_1),\quad \eta=\sqrt{\eta_1^2+\eta_2^2+\eta_3^2},$$
$$\frac{\partial\eta}{\partial n}=\frac{\eta_1}{\eta}n_1+\frac{\eta_2}{\eta}n_2+\frac{\eta_3}{\eta}n_3, \tag{4}$$

where $P_j^{(1)},P_j^{(2)},P_j^{(3)}$ are the scalar components of the vector surface

$$\mathbf{P}_j(v,w)=\left[P_j^{(1)}(v,w),P_j^{(2)}(v,w),P_j^{(3)}(v,w)\right]^T$$