Paper

Preconditioned Conjugate Gradient Method for Solution of Large Finite Element Problems on CPU and GPU

Sergiy Yu. Fialko and Filip Zeglen

Institute of Computer Science, Faculty of Physics, Mathematics and Computer Science,
Tadeusz Kościuszko Cracow University of Technology, Cracow, Poland

Abstract—In this article the preconditioned conjugate gradient (PCG) method, realized on GPU and intended to solution of large finite element problems of structural mechanics, is considered. The mathematical formulation of problem results in solution of linear equation sets with sparse symmetrical positive definite matrices. The authors use incomplete Cholesky factorization by value approach, based on technique of sparse matrices, for creation of efficient preconditioning, which ensures a stable convergence for weakly conditioned problems mentioned above. The research focuses on realization of PCG solver on GPU with using of CUBLAS and CUSPARSE libraries. Taking into account a restricted amount of GPU core memory, the efficiency and reliability of GPU PCG solver are checked and these factors are compared with data obtained with using of CPU version of this solver, working on large amount of RAM. The real-life large problems, taken from SCAD Soft collection, are considered for such a comparison.

Keywords—conjugate gradient, incomplete Cholesky factorization, iterative solver, NVIDIA CUDA, preconditioned conjugate gradient.

1. Introduction

The computational power of modern PC's becomes enough to solve medium scale complex engineering problems. Intensive development of desktop computers and gaming rigs markets made that for some aspects High Performance Computing (HPC) solutions are no longer necessary for a lot of problems. In the future, this trend will be expanded onto range of issues where ability of PC computers is sufficient for their solution of given problem scale. Improvements in the hardware realizations enhance development capabilities and demands to develop computational methods directly into a specific computer architecture. Processor units for execution of the fast instructions need in efficient memory management. Achievement of peak performance on logical thread must be preceded by elimination of empty cycles on physical core. The ways of efficient memory management for distributed memory architecture Non-Uniform Memory Access (NUMA) of today's HPC systems substantially distinguish from techniques used in Uniform Memory Access (UMA) of PC solutions. On the other hand, computational units of clusters in many cases have the same processor architecture at the level of hardware node as used in desktop solutions. Consequence of these distinctions is the necessity for creation of different algorithms implementing computational methods for desktop systems.

Solving systems of linear algebraic equations, arising from analysis of problem of solid and structural mechanics, by the preconditioned conjugate gradient on the Graphic Processing Unit (GPU) appear in many papers. In example, an article [1] presents the acceleration of matrix-vector product procedure with usage of ELLPACK, BELLPACK, SBELL formats instead of CSR in Compute Unified Device Architecture (CUDA) for packing of sparse symmetrical matrix for 2D elastic problem of solid mechanics. The block compressed sparse row (BCSR) format is applied for acceleration of sparse matrix-vector multiplication (SpMV) procedure in [2] for conjugate gradient (CG) method using CUDA. The application of graphic accelerators in finite element structural analysis is discussed in [3]. Article [4] proposes a level scheduling based on approximate minimum degree reordering algorithm for acceleration the triangular solution procedure.

In presented article, authors limit themselves to solving systems of linear algebraic equations with symmetrical sparse matrices by preconditioned conjugate gradient method. Such matrices arise when finite element method is applied to the problems of structural or solid mechanics. Scientific publications about parallel implementation of conjugate gradient method in architecture Symmetric Multiprocessing (SMP) can be found in [5]. This paper is mainly focused on maximal effective use of RAM memory. In cases when at application of sparse direct method the size of factorized stiffness matrix exceeds the capacity of RAM, it is necessary to use a secondary storage on disk. It leads to drastic increase of the computation time because solver produces lot of IO operations. The proposed iterative method runs in core memory and in the case of fast convergence could be considerably faster. Small number of iterations is achieved primarily using appropriate sparse matrix techniques for constructing of preconditioning based on Cholesky factorization by value method with application of secondary rejection of small entries [6]. Given approach as well as [7], [8] is intended for solution of complex engineering problems and produces all computations on CPU. The leading procedures – matrix-vector multiplication and forward-back substitutions relatively preconditioning – are poorly accelerated due to parallelization on shared-memory

computers when number of threads increases. Usually several right hand parts – load cases – appear in problems of structural mechanics. Therefore, in [5] each right hand part iterates on separate thread, and number of threads is restricted by number of right hand parts. The development of modern graphics cards is driven by the development of PCs. Today, in era of rapid general-purpose GPU development, the calculations are a separated branch and professional computing accelerators are not used as graphics cards although their architecture is made for that and allow it. HPC solutions also are equipped by accelerators, based on GPU.

This paper is devoted to development of preconditioned conjugate gradient method with incomplete Cholesky factorization by value preconditioning [5] using GPU, based on CUDA technology and intended to solution of linear algebraic equation sets with sparse symmetric positive definite matrices. Described implementation involves the use of a single device with general-purpose GPU support. This is a typical situation for PC with one external graphics card or workstation with one computing accelerator.

2. Preconditioned Conjugate Gradient Method

Let us consider the linear equation set

$$\mathbf{Kx} = \mathbf{b}, \tag{1}$$

where $\mathbf{K}$ is a symmetric positive definite sparse matrix arising when the finite element method is applied to problems of structural and solid mechanics. The problems of structural mechanics often are poorly conditioned due to using of thin-walled plates, shells, bars and large scattering of stiffness in structural elements, and the slow convergence of iterative methods occurs in such a situation [9].

Algorithm 1: Incomplete Cholesky factorization

1: $\mathbf{v}_{ip,i} = 0$
2: $ip = 0, 1, \ldots, np-1$
3: $i = 1 \in [1 \ldots N]$
4: **for** $j \in [1 \ldots N]$ **do**
5: $\quad \mathbf{v}_{0,i} = \mathbf{K}_{ij}, i \in L_j$
6: $\quad$ *Parallel* **for** $k \in List_j$ **do**
7: $\quad\quad \mathbf{v}_{ip,i} = \mathbf{v}_{ip,i} - \mathbf{H}_{i,k}\mathbf{H}_{j,k}, i \in L_k$
8: $\quad$ **end for**
9: $\quad$ **for** $ip \in [1 \ldots np-1]$ **do**
10: $\quad\quad \mathbf{v}_{0,i} += \mathbf{v}_{ip,i}, i \in L_j$
11: $\quad$ **end for**
12: $\quad$ **if** $\mathbf{v}_{0i}{}^2 < \psi \mathbf{H}_{ii}\mathbf{H}_{jj}$, $i \in L_j$ **then**
13: $\quad\quad \mathbf{H}_{ii} += \left|\mathbf{H}_{ij}\right| \sqrt{\frac{\mathbf{H}_{ii}}{\mathbf{H}_{jj}}}$, $\mathbf{H}_{jj} += \left|\mathbf{H}_{ij}\right| \sqrt{\frac{\mathbf{H}_{jj}}{\mathbf{H}_{ii}}}, \mathbf{v}_{0,i} = 0$
14: $\quad$ **else**
15: $\quad\quad \mathbf{L}_j \leftarrow \frac{\mathbf{v}_{0,i}}{\sqrt{\mathbf{H}_{jj}}}, List_i = j, \mathbf{v}_{0,i} = 0$
16: $\quad$ **end if**
17: **end for**

The proposed approach allows keeping a small value of rejection parameter ψ and ensures a stable and fast convergence of PCG method even for weakly conditioned problems of structural mechanics. The term "weakly conditioned" means that matrix $\mathbf{K}$ is not singular, but the conditioning number $cond(\mathbf{K})$ is relatively large, and conventional iterative methods demonstrates a slow convergence. The article [5] contains the detail consideration of proposed approach.

The preconditioned problem $\mathbf{B}^{-1}\mathbf{Kx} = \mathbf{B}^{-1}\mathbf{b}$ is solved instead Eq. (1), where $\mathbf{B} = \mathbf{HH}^T$ and $\mathbf{H}$ is the lower triangular matrix. The looking-left column-by-column incomplete factorization procedure is applied [5] as shown in Algorithm 1.

The *ip* and *np* are the thread number and the number of threads, respectively. Next, nonzero entries of current column j of matrix $\mathbf{K}$ are put to the dense vector $\mathbf{v}_0$ ($\mathbf{v}_{0,i} = \mathbf{K}_{ij}$). Expression $i \in L_j$ means that row number i belongs to nonzero structure of current column j. In the loop "parallel for k" columns k located at left from $j (k < j)$ produce the update of column j. Expression $k \in List_j$ means that only such columns k which have nonzero elements $\mathbf{H}_{jk}$ in factorized matrix $\mathbf{H}$ are taken. Each thread *ip* writes results in own vector $\mathbf{v}_{ip}$. Then we sum the results of each thread and obtain updated column j in vector $\mathbf{v}_0$ (loop for $ip = 1$, $np-1$). In the next step, each nonzero entry of $\mathbf{v}_0 (i \in L_j)$ is analyzed and the small entries $\mathbf{v}_{0i}{}^2 < \psi \mathbf{H}_{ii}\mathbf{H}_{jj}$, where $0 < \psi < 1$ (if $\mathbf{v}_{0i}{}^2 < \psi \mathbf{H}_{ii}\mathbf{H}_{jj}$) are rejected. Each rejection results in correction of diagonal entries $\mathbf{H}_{ii}$, $\mathbf{H}_{jj}$ to keep the positive definiteness of $\mathbf{H}$ and preconditioning matrix $\mathbf{B}$ [11]. Only the "large" entries are retained and put it in nonzero structure L_j of matrix $\mathbf{H}$.

Algorithm 2: PCG method

1: $k = 0$, $\mathbf{x}_0 = 0$
2: $\mathbf{r}_0 = \mathbf{b} - \mathbf{Kx}_0$
3: **while** $||r_k||_2 > tol$ **do**
4: $\quad$ Solve $\mathbf{Bz}_k = \mathbf{r}_k$
5: $\quad k = k + 1$
6: $\quad$ **if** $k = 1$ **then**
7: $\quad\quad \mathbf{p}_1 = \mathbf{z}_0$
8: $\quad$ **else**
9: $\quad\quad \beta_k = \frac{\mathbf{r}_{k-1}^T \mathbf{z}_{k-1}^T}{\mathbf{r}_{k-2}^T \mathbf{z}_{k-2}^T}$
10: $\quad\quad \mathbf{p}_k = \mathbf{z}_{k-1} + \beta_k \mathbf{p}_{k-1}$
11: $\quad$ **end if**
12: $\quad \alpha_k = \frac{\mathbf{r}_{k-1}^T \mathbf{z}_{k-1}^T}{\mathbf{p}_k^T \mathbf{K} \mathbf{p}_k^T}$
13: $\quad \mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k$
14: $\quad \mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k \mathbf{K} \mathbf{p}_k$
15: **end while**
16: $\mathbf{x} = \mathbf{x}_k$

Also, the number of current column j is put in $List_i$ of column i, located at right. In addition, after incomplete factoring is finished, secondary rejection of small entries $\mathbf{H}_{ij}{}^2 < \psi_1 \mathbf{H}_{ii}\mathbf{H}_{jj}$, where $0 < \psi < \psi_1 < 1$ is produced. It allows on reduction of nonzero entries in incomplete fac-

tor $\mathbf{H}$ and accelerates triangular solution procedure without significant deterioration of preconditioning properties.
The minimum degree ordering algorithm is applied before incomplete factorization for reducing the number of nonzero entries in factorized matrix. It improves the ability of preconditioning to accelerate a convergence [5].
The Algorithm 2 presents the PCG method.
The residual vector for problem given by Eq. (1) on iteration step k is: $\mathbf{r}_k = \mathbf{b} - \mathbf{K}\mathbf{x}_k$, where $\mathbf{x}_k$ is approximation of exact solution $\mathbf{x}$. For preconditioned problem the residual vector $\mathbf{z}_k$ is evaluated from expression $\mathbf{B}^{-1}\mathbf{r}_k = \mathbf{B}^{-1}(\mathbf{b} - \mathbf{K}\mathbf{x}_k) = \mathbf{z}_k$. Then the set of linear equations relatively preconditioning $\mathbf{B}\mathbf{z}_k = \mathbf{r}_k$, or $\mathbf{H}\mathbf{H}^T\mathbf{z}_k = \mathbf{r}_k$ is solved. The forward substitution $\mathbf{H}\mathbf{y}_k = \mathbf{r}_k \rightarrow \mathbf{y}_k$ and back substitution $\mathbf{H}^T\mathbf{z}_k = \mathbf{y}_k \rightarrow \mathbf{z}_k$ are produced.
The incomplete Cholesky factorization procedure requires a large amount of core memory and authors use the parallel Algorithm 1 implemented on CPU. The both CPU and GPU versions of PCG method exactly correspond to presented Algorithm 2.

3. Conjugate Gradient Method on GPU and Implementation Using CUDA

Preconditioned conjugate gradient method performs the set of linear algebra operations on matrices and vectors. All operations on GPU are produced only with application of procedures from CUBLAS [11] and CUSPARSE [12] libraries: *cusparseDcsrsv-solve()* [13] for triangular solution $\mathbf{H}\mathbf{y}_k = \mathbf{r}_k \rightarrow \mathbf{y}_k$ (forward substitution) and $\mathbf{H}^T\mathbf{z}_k = \mathbf{y}_k \rightarrow \mathbf{z}_k$ (back substitution), *cusparseDcsrmv()* [14] for matrix-vector multiplication $\mathbf{w}_k = \mathbf{K}\mathbf{p}_k$, *cublasDdot()* for evaluation of dot products ${\mathbf{r}_{k-1}}^T\mathbf{z}_{k-1}$, ${\mathbf{p}_k}^T\mathbf{w}_k$, *cublasDaxpy()* for computing of saxpy procedures $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\mathbf{p}_k$, $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k\mathbf{w}_k$ and cublasDscal() for vector scalar multiplication $\mathbf{p}_k = \beta_k\mathbf{p}_{k-1}$.
The multiplication of sparse symmetric matrix by vector and triangular solutions during forward and back substitutions are the most complex procedures of PCG method. Their duration exceeds 90% of total solution time. The algorithm that performs any operations with sparse matrix must be consistent with the format in which this matrix is stored. CUSPARSE library supports following sparse matrix formats: COO, CSR, CSC, ELL, HYB, BSR, BSRX [15]. Procedures for Symmetric Positive Definite (SPD) matrices operate on matrices stored in Compressed Storage Row (CSR) format. This implementation of PCG method is practically the same as implementation of PCG method from CUDA library [16]. The main difference consists in the construction of preconditioning (see Algorithm 1). Utilization of compressed row format (CSR), which largely focused on low memory requirement, is perfect for the graphics card device, having the restricted memory amount. On the other hand, in the case of sparse matrix with specific structure CSR format greatly reduces possibility of blocking memory for CUDA thread blocks. The consequence of jumps in memory due to specific structure of sparse matrix leads to slowdown of instruction execution by pipelines in block of CUDA threads. Many publications concerning with implementation of sparse matrix-vector multiplication algorithm on the GPU [17] are devoted to achievement of high performance in operations on sparse positive definite matrices, stored in different formats and designed for the GPU. Thus, for GPU computing does no storage format for symmetric positive definite matrices exist, which would always give the most high-performance matrix-vector operations. Therefore, computing performance essentially depends on the structure of sparse matrix and its density.
The triangular solution procedure has a highly sequential nature – its parallelization does not result in considerable acceleration of computations on SMP computers as well as on GPU. Algorithm of triangular solution used in CUSPARSE is presented in [18]. Algorithm 3 contains the pseudocode of version on GPU. The CPU version is presented in [5].

Algorithm 3: Pseudocode of GPU version

1: Aggregate sparse stiffness matrix $\mathbf{K}$ and prepare lower triangular matrix $\mathbf{H}$ using Algorithm 1 (on CPU). Initiate CUDA device
2: Allocate memory on graphic accelerator (device memory) for matrices $\mathbf{K}$, $\mathbf{H}$, packed in CSR format and copy these matrices from host memory to device memory
3: Use *cusparseDcsrsm-analysis()* procedure twice for analysis of structure of the $\mathbf{H}$ and $\mathbf{H}^T$ matrices
4: Allocate device memory for vectors $\mathbf{x}$, $\mathbf{p}$, $\mathbf{r}$, $\mathbf{z}$ and working vector $\mathbf{w}$
5: Run Algorithm 2 until convergence will not be achieved (no transfers of data between host and device occurs)
6: Copy converged vector $\mathbf{x}$ from device to host
7: Deallocate device memory and deinitialize CUDA device

The version of solver on CPU uses in-home algorithms. Authors found in [19] that procedure *mkl-dcsrsymv* (sparse symmetric matrix-vector multiplication) taken from Intel MKL 11.2 accelerates the sparse symmetric matrix - vector multiplication about 2 times in compare with in-home procedure on single thread and about 3 times on multiple threads. In addition, the *mkl-dcsrtrsv* (triangular solution for sparse matrix) procedure from Intel MKL demonstrates the same time on the both: single thread and multiple threads, and is on 20% slower in compare with in-home triangular solution procedure. For proposed class of problems the density of lower triangular matrix $\mathbf{H}$ is in many times more than density of stiffness matrix $\mathbf{K}$, and the duration of triangular solution procedure is in several times longer than the procedure of matrix-vector multiplication (Tables 1, 4, and 7). Therefore, acceleration of matrix-vector multiplication procedure does not produce considerable impact on performance of PCG solver and allows us to use in-home procedures on CPU version.

The authors used Microsoft Visual Studio 2013 IDE and NVIDIA GPU Computing Toolkit v.6.5 with CUBLAS and CUSPARSE libraries. The C++ compiler v. 120 with flags /O2 in realize version and /arch:AVX (Advanced Vector Extension) was applied. Also, the unrolling of loop eight times in the both: in-home matrix-vector multiplication and in triangular solution procedures was used.

4. Test Problems and Hardware Configuration

The three design models with fully different properties of stiffness matrices are considered. Two models of multi-storey buildings with quite different construction schemes and the model of shopping center are analyzed. All these real-life design models are taken from SCAD Soft [20] collection. The research attention is focused on total computation time, on computation time of matrix-vector multiplication procedure and on computation time of triangular solution procedure at the stage of resolution respectively preconditioning (SpTr). These intervals of time encompass all iterations required for achievement of convergence.

Tests were made on following hardware configuration: IBM System x iDataPlex dx360 M4 Server running Windows 7 Ultimate 64 bit, CUDA Toolkit 6.5 with CPU – Intel Xeon E5-2620 2.0 GHz 6C 12T (2.5 GHz Turbo) 6x256 kB L2, 15 MB L3, 32 GB DDR3 (8x4 GB) PC3-10700U (1333 MHz) and GPU – Nvidia Tesla K20m 2496 CUDA cores 705 MHz, 5 GB GDDR5 5200 MHz (1300 MHz).

The following designations are used. Lx is calculation time of forward and backward substitutions for all iterations, Kv – calculation time of procedure SpMV for all iterations, Oth stands for duration of other computing included in PCG iteration, Total – total calculation time for all produced iterations, Itr is the number of iterations required to achieve of convergence, DenK and DenL – densities of matrices **K** and **H** correspondingly (number of nonzero entries/total number of elements in lower triangular part of matrix) expressed as a percentage, ψ – value of rejection parameter $\psi\left(\psi \in \left[10^{-9}, 10^{-20}\right]\right)$, ψ_1 – value of post-rejection parameter.

4.1. Problem 1

Aquamarine is a finite element model of multistory building (Fig. 1) and comprises 176,819 finite elements, equations 881,908 and 149,494 nodes. Authors present the computation times on the both: CPU and GPU with the fixed value of rejection parameter $\psi = 10^{-10}$, and different values of post-rejection parameter ψ_1 (Tables 1 and 2). Table 3 shows comparison of computation times on CPU and GPU.

Higher GPU performance arises only on relatively large values of dropping parameters (Fig. 2). In this case structure of the matrix **H** is more sparser than when using smaller value of dropping parameter ψ_1 (Table 3). In all other cases

Fig. 1. Computational model of Aquamarine.

Table 1
Problem 1 – computation times on CPU at different ψ, ψ_1 parameters

ψ	ψ_1	Kv [s]	Lx [s]	Oth [s]	Total [s]	Itr
10^{-10}	10^{-8}	3.5	44.4	1.1	49	142
10^{-10}	10^{-6}	4.3	31.9	1.8	38	179
10^{-10}	10^{-4}	23.9	102.1	10	136	968
10^{-10}	10^{-3}	169.8	543.7	92.5	806	6550

Table 2
Problem 1 – computation times on GPU at different ψ, ψ_1 parameters

ψ	ψ_1	Kv [s]	Lx [s]	Oth [s]	Total [s]	Itr
10^{-10}	10^{-8}	7.4	309	13.6	330	142
10^{-10}	10^{-6}	9.3	112	9.7	131	179
10^{-10}	10^{-4}	49.9	100	15.1	165	968
10^{-10}	10^{-3}	334.8	185.8	118.4	639	6550

Table 3
Problem 1 – comparison of computation time on CPU and GPU ($\psi = 10^{-10}$, DenK = 0.0033)

ψ_1	CPU [s]	GPU [s]	DenL
10^{-8}	49	330	0.0189
10^{-6}	38	131	0.0099
10^{-4}	136	165	0.0046
10^{-3}	806	639	0.0029

in which the parameter $\psi <= 10^{-4}$, density of the matrix **H** increases, and a considerable advantage of the CPU version (Table 1) over the GPU (Table 2) can be observed.
The matrix-vector multiplication procedure using CPU is less than GPU version for all considered values of dropping parameters (Tables 1 and 2). Duration of matrix-vector multiplication procedure depends on density of stiffness matrix **K**, which strictly depends on considered problem and does not depend on values of dropping parameters. In contrast to matrix-vector multiplication procedure, the duration of triangular solution procedure depends on values of ψ and ψ_1 parameters (Table 3).

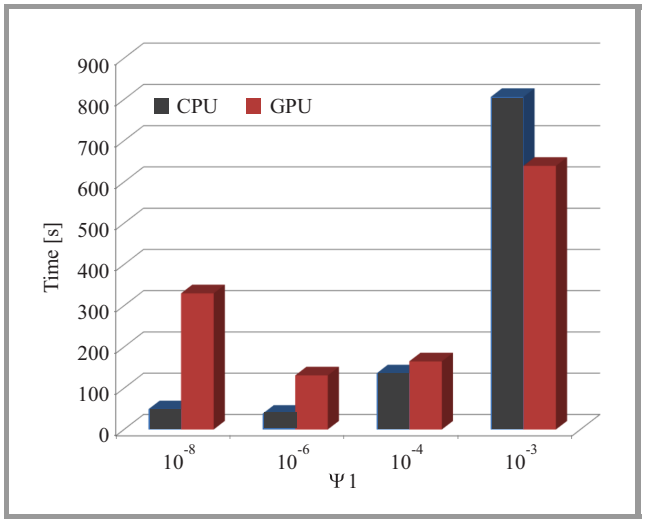

Fig. 2. Problem 1 – comparison of computation times on CPU and GPU for Aquamarine.

The performance of triangular solution procedure by CUDA [13], [17] considerably depends on matrix **H** sparsity. If **H** density is very small, the CUDA realization of triangular solution is faster than CPU version. With decreasing of drop parameter value the density of **H** increases and CPU realization of triangular solution becomes faster (Table 3). For drop parameters values, ensuring acceptably fast solution, CPU realization is more faster than GPU (Fig. 2).

4.2. Problem 2

Schemanew is a finite element model of multistory building (Fig. 3) and comprises 556,905 finite elements, equations 3,198,609 and 534,490 nodes. In this Subsection the CPU and GPU times for different values of rejection parameters ψ, ψ_1 (Tables 4 and 5) and their comparison (Table 6) are presented.
When applying the small post-dropping parameters $\psi_1 <= 10^{-6}$ which satisfy a fast convergence, the size of preconditioning matrix **H** exceeds capacity of GPU memory of Tesla equipped with 5.4 GB. In such cases in Fig. 4 and Tables 5 and 6 only the results obtained on CPU are depicted. In this Subsection authors show on plots two rejec-

Fig. 3. Computational model of Schemanew.

Table 4
Problem 2 – computation times on CPU at different ψ, ψ_1 parameters

ψ	ψ_1	Kv [s]	Lx [s]	Oth [s]	Total [s]	Itr
10^{-11}	10^{-8}	15.5	127.4	8.1	151	196
10^{-9}	10^{-6}	45.6	289.8	23.6	359	497
10^{-9}	10^{-5}	59.8	303	30.2	393	658
10^{-9}	10^{-4}	142	566	73	781	1584

Table 5
Problem 2 – computation times on GPU at different ψ, ψ_1 parameters

ψ	ψ_1	Kv [s]	Lx [s]	Oth [s]	Total [s]	Itr
10^{-11}	10^{-8}	–	–	–	–	–
10^{-9}	10^{-6}	–	–	–	–	–
10^{-9}	10^{-5}	104	139	19	262	659
10^{-9}	10^{-4}	248	148	33	429	1583

Table 6
Problem 2 – comparison of computation time on CPU and GPU (DenK = 0.000667)

ψ	ψ_1	CPU [s]	GPU [s]	DenL
10^{-11}	10^{-8}	151	–	0.00347
10^{-9}	10^{-6}	359	–	0.00196
10^{-9}	10^{-5}	393	262	0.00144
10^{-9}	10^{-4}	781	429	0.00099

tion parameters ψ and ψ_1, separated by semicolon. Higher performance of GPU realization (Table 5) of PCG solver comparing with CPU version (Table 4) is achieved only for respectively large values of drop parameters, which leads to increasing of number of iterations and slowdown of convergence. The acceptable solution time is achieved on CPU version. GPU version for proper values of drop parameter, which ensure the stable and fast convergence, requires essentially more amount of memory than given graphic accelerator possesses.

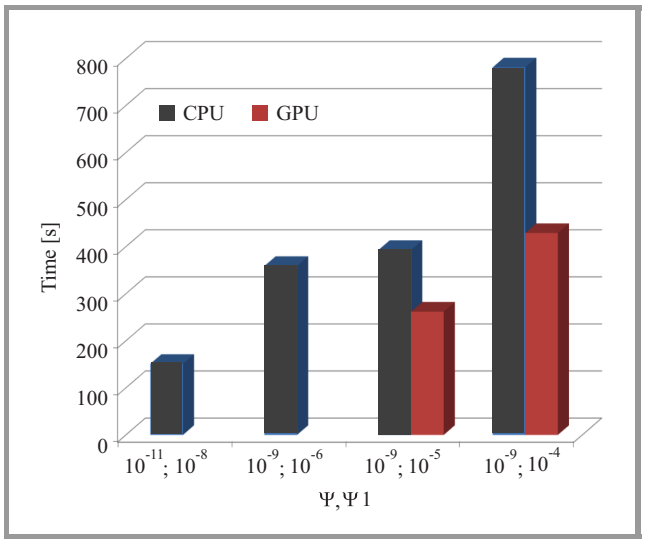

Fig. 4. Problem 2 – comparison of computation times on CPU and GPU for Schemanew.

For $\psi_1 > 10^{-5}$ is possible to put matrix **H** in memory of graphic accelerator, and GPU version is faster than CPU. However, with decreasing of ψ_1 density of matrix **H** increases, and advantage of GPU version becomes smaller (Fig. 4). This tendency suggests, that even if memory of the graphic accelerator would suffice for accommodation of matrix **H** at smaller values ψ_1, with decreasing of ψ_1 CPU becomes faster, than the version on GPU. Probably, it is caused by specifics of triangular solution algorithm [18] (Table 5), developed by NVIDIA, which is very fast for sparse matrices of low density, but with increase of density its performance considerably deteriorates.

4.3. Problem 3

TRK is a finite element model of market building (Fig. 5) and comprises 473,723 finite elements, equations 2,442,846 and 441,300 nodes.

For parameters $\psi = 10^{-8}$ and $\psi_1 = 10^{-7}$ matrix is to large and does not fit into GPU accelerator memory.

The matrix-vector multiplication procedure as well as in the previous problems, on CPU is about two times faster than on GPU (Tables 7 and 8).

The best performance of triangular solution procedure, which mostly affects total time, is achieved on GPU. In each case of preconditioning parameters, authors found considerable advantage of GPU version comparing with CPU (Table 9).

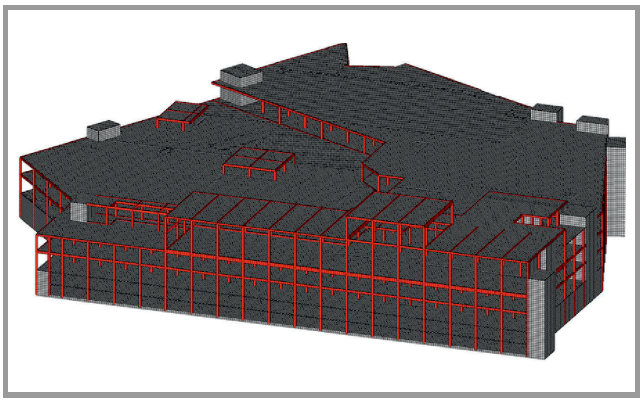

Fig. 5. Computational model of TRK.

Table 7
Problem 3 – computation times on CPU
at different ψ, ψ_1 parameters

ψ	ψ_1	Kv [s]	Lx [s]	Oth [s]	Total [s]	Itr
10^{-10}	10^{-5}	16.1	90.5	8.4	115	231
10^{-8}	10^{-7}	10.4	89.5	6.1	106	177
10^{-8}	10^{-6}	11.2	79.8	7	98	191
10^{-8}	10^{-5}	15	85.8	9.2	110	257
10^{-8}	10^{-4}	41.9	188.9	23.2	254	707

Table 8
Problem 3 – computation times on GPU
at different ψ, ψ_1 parameters

ψ	ψ_1	Kv [s]	Lx [s]	Oth [s]	Total [s]	Itr
10^{-10}	10^{-5}	26.7	36.1	7.2	70	231
10^{-8}	10^{-7}	–	–	–	–	–
10^{-8}	10^{-6}	22.2	41.5	6.3	70	191
10^{-8}	10^{-5}	29.5	28.4	8.1	66	256
10^{-8}	10^{-4}	81.4	30.6	25	137	705

Table 9
Problem 3 – comparison of computation times on CPU
and GPU (DenK = 0.00096)

ψ	ψ_1	CPU [s]	GPU [s]	DenL
10^{-10}	10^{-5}	115	70	0.00233
10^{-8}	10^{-7}	106	–	0.00384
10^{-8}	10^{-6}	98	70	0.00305
10^{-8}	10^{-5}	110	66	0.00224
10^{-8}	10^{-4}	254	137	0.00155

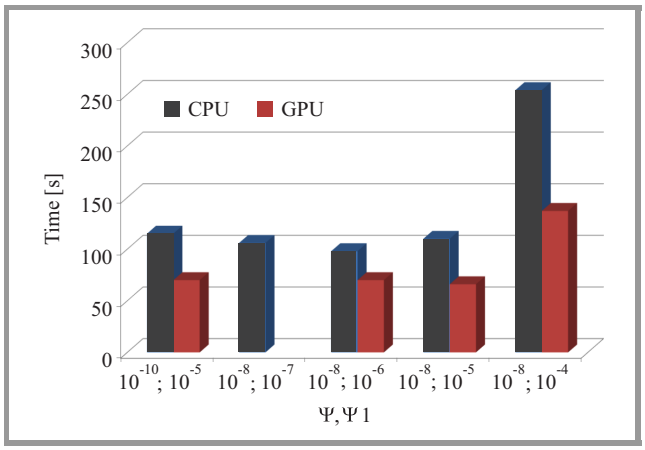

Fig. 6. Problem 3 – comparison of computation times on CPU and GPU for TRK.

5. Summary and Conclusions

For presented problems of structural mechanics, which we intend to solve by PCG method, the conducted research cannot clearly indicate which one of two comparable devices: CPU or GPU, will demonstrate a better performance. Which device is better, depends on the densities and structures of the matrices **K** and **H**. GPU demonstrates a better performance only for very sparse matrices, the location of nonzero elements of which allows on efficient split on parallel tasks during triangular solution, performed by algorithm [18]. For problem 1, where matrix **H** has highest density among all considered problems, one can observe significant CPU advantage at stage of triangular solution, which has a main impact on total solution time. For problems 2 and 3 matrix **H** is a more sparser, than in problem 1, and GPU version demonstrates a less time of triangular solution algorithm and correspondingly better solution time than CPU, until the amount of graphic accelerator memory is enough large to put matrices **H** and **K**. To improve ability of preconditioning and accelerate convergence, the values of drop parameters ψ, ψ_1 were decreased and density of matrix **H** was increased. When GPU version fails due to insufficient device memory, the CPU version of solver having sufficient amount of RAM, solves these problem faster than GPU version. Only for problem 3 authors found that GPU version occurs faster than CPU. Matrix-vector multiplication procedure is always faster in CPU version, regardless of density of matrix **K**. Decrease of drop parameter value leads to improving of preconditioning properties, reducing the number of iterations for achievement of convergence, but increasing the duration of each iteration due to considerable enlarging of triangular solution time.

Acknowledgements

The authors express their deep gratitude to SCAD Soft IT company for providing a collection of real-life problems from engineering practice.

References

[1] J. Zhang and L. Zhang, "Efficient CUDA polynomial preconditioned conjugate gradient solver for finite element computation of elasticity problems", *Mathem. Problems in Engin.*, article ID 398438, pp. 1–12, 2013.

[2] M. Verschoor and A. C. Jalba, "Analysis and performance estimation of the Conjugate Gradient method on multiple GPUs", *Parallel Comput.*, vol. 38, no. 10–11, pp. 552–575, 2012.

[3] S. Georgescu, P. Chow, and H. Okuda, "GPU Acceleration for FEM-Based Structural Analysis", *Arch. Comput. Methods Engin.*, vol. 20, no. 2, pp. 111–121, 2013.

[4] C. Yao, Z. Yonghua, Z. Wei, Z. Lian, "GPU-accelerated incomplete Cholesky factorization preconditioned conjugate gradient method", *J. of Comp. Res. & Develop.*, vol. 52, no. 4, pp. 843–850, 2015.

[5] S. Y. Fialko, "Iterative methods for solving large-scale problems of structural mechanics using multi-core computers", *Archiv. of Civil and Mechan. Engin.*, vol. 14, no. 1, pp. 190–203, 2014.

[6] M. Suarjana and K. H. Law, "A robust incomplete factorization based on value and space constraints", *Int. J. for Numerical Methods in Engin.*, vol. 38, pp. 1703–1719, 1995.

[7] K. Malkowski, I. Lee, P. Raghavan, and M. J. Irwin, "Conjugate gradient sparse solver: Performance-power characteristics", in *Proc. 20th IEEE Int. Parallel & Distrib. Process. Symp. IPDPS 2006*, Rhodes Island, Greece, 2006.

[8] M. Papadrakakis, "Solving Large-Scale Problems in Mechanics". *Wiley*, 1993.

[9] S. Y. Fialko, "Parallel direct solver for solving systems of linear equations resulting from finite element method on multi-core desktops and workstations", *Comp. & Mathem. with Appl.*, vol. 70, pp. 2968–2987, 2015.

[10] A. Jennings, "Development of an ICCG algorithm for large sparse systems", in *Preconditioned Methods. Theory and Applications*, D. J. Evans, Ed. Gordon and Breach Publishers, 1983, pp. 425–438.

[11] Nvidia Corporation, "cuBLAS API" [Online]. Available: http://docs.nvidia.com/cuda/cublas/

[12] Nvidia Corporation, "cuSPARSE API" [Online]. Available: http://docs.nvidia.com/cuda/cusparse/

[13] Nvidia Corporation, "csrsvsolve()" [Online]. Available: http://docs.nvidia.com/cuda/cusparse/#cusparse-lt-t-gt-csrsvsolve

[14] Nvidia Corporation, "csrmv()" [Online]. Available: http://docs.nvidia.com/cuda/cusparse/#cusparse-lt-t-gt-csrmv

[15] Nvidia Corporation, "cuSPARSE Indexing and Data Formats" [Online]. Available: http://docs.nvidia.com/cuda/cusparse/#cusparse-indexing-and-data-formats

[16] Nvidia Corporation, "CUDA Library – conjugateGradientPrecond – Preconditioned Conjugate Gradient" [Online]. Available: http://docs.nvidia.com/cuda/cuda-samples/#preconditioned-conjugate-gradient/

[17] B.-Y. Su and K. Keutzer, "clSpMV: A Cross-Platform OpenCL SpMV Framework on GPUs" [Online]. Available: http://parlab.eecs.berkeley.edu/sites/all/parlab/files/clspMV-%20Keutzer.pdf

[18] M. Naumov, "Parallel Solution of Sparse Triangular Linear Systems in the Preconditioned Iterative Methods on the GPU" [Online]. Available: https://research.nvidia.com/publication/parallel-solution-sparse-triangular-linear-systems-preconditioned-iterative-methods-gpu

[19] S. Fialko and F. Zeglen, " Block preconditioned conjugate gradient method for extraction of natural vibration frequencies in structural analysis", in *Proc. Feder. Conf. Comp. Sci. & Inform. Syst. FedCSIS 2015*, Łódź, Poland, 2015, vol. 3, pp. 655–66.

[20] V. S. Karpilovskii, E. Z. Kriksunov, A. A. Malyarenko, M. A. Mikitarenko, A. V. Perelmuter, and M. A. Perelmuter, "SCAD computational complex", *ASV*, 2004 (in Russian).

Sergiy Yu. Fialko studied structural mechanics and obtained his degree in 1983 in Institute of Mechanics in Kiev, Ukraine. He worked in Research Institute of Steel Constructions in Kiev, in software company RoboBAT in Cracow, Poland, in National University of Construction and Architecture in Kiev, where obtained his habilitation degree in 2004. Since 2007, he has been working in the Cracow University of Technology. For many years he is the scientific adviser of software company SCAD Soft.
E-mail: sergiy.fialko@gmail.com
Institute of Computer Science
Faculty of Physics, Mathematics and Computer Science
Tadeusz Kościuszko Cracow University of Technology
Warszawska st 24
31-155 Cracow, Poland

Filip Zeglen obtained his B.Sc. graduation in 2012 in Computer Science at Wyższa Szkoła Ekonomii i Informatyki in Cracow and M.Sc. graduation in 2014 at Cracow University of Technology. He worked as .NET, Java developer and freelancing indie games developer. Since 2014 he is Research and Teaching Assistant at Cracow University of Technology and Ph.D. student at Institute of Fundamental Technological Research Polish Academy of Sciences.
E-mail: filipzeglen@hotmail.com
Institute of Computer Science
Faculty of Physics, Mathematics and Computer Science
Tadeusz Kościuszko Cracow University of Technology
Warszawska st 24
31-155 Cracow, Poland