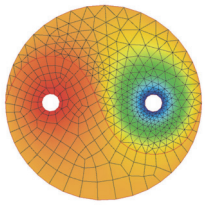
Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE

Informatyka w Technologii Materiałów

Vol. 16, 2016, No. 4



FINITE ELEMENT CORE CALCULATIONS AND STREAM PROCESSING

KRZYSZTOF BANAŚ*, JAN BIELAŃSKI, KAZIMIERZ CHŁOŃ

AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Kraków, Poland
**Corresponding author: kbanas@agh.edu.pl*

Abstract

We present the execution model and performance analysis for the important phase of finite element calcu- lations, the creation of systems of linear equations. We assume that the process is realized using a set of CPU cores and GPU multiprocessors, with CPU and GPU memories connected using PCIe links for data transfer. We analyse the use of linear data structures that are designed specially for GPU processing. We present the examples of calculations for the standard first order FEM approximation and typical contemporary hardware. We draw the conclusions on the feasibility of the proposed approach.

Key words: finite element method, solvers of linear equations, GPU, OpenCL, high performance computing, technical simulations

1. INTRODUCTION

The main hardware architectures on which finite element calculations are run nowadays are either single servers/workstations with multicore microprocessors or clusters of such servers/workstations. In both cases the hardware models usually contain a number of processing units and a hierarchy of memory levels, with communication channels between units (within computer network or through shared memory). In the simplest models there is only one level of memory, the main DRAM memory, but such models do not allow for detailed performance analysis and modelling and, in consequence, have limited applicability to performance optimization. On the other hand, when more memory levels are taken into account several difficulties appear. The number of cache levels vary from architecture to architecture and, moreover, the hardware designers utilize different, and usually not revealed to programmers, strategies for cache memory usage. In such situations theoretical performance modelling and prediction becomes difficult and often the software designers employ experimental techniques for code optimizations, such as e.g. explicit search of the whole space of parameters having impact on code performance (Du et al., 2012 ; Choi et al., 2010).

Graphics microprocessors (GPUs) form an attractive alternative to classical multicore architectures, due to its high theoretical performance, both in terms of computing power and DRAM memory bandwidth. The drawback of their use is the current hardware configuration in which they are employed. GPUs are equipped with fast DRAM memory but input data for GPU calculations have to be transferred to this memory from main CPU memory through slow PCIe links (figure 1). In order to analyse the performance of GPU computations, it is necessary to take into account, even if cache memory is neglected, at least two separate pools of memory, CPU DRAM memory and GPU DRAM memory, connected by communication channels. This channels include not only PCIe links

ISSN 1641-8581

between CPUs and GPUs but also network connections between CPUs and, possibly, separate links between GPUs.

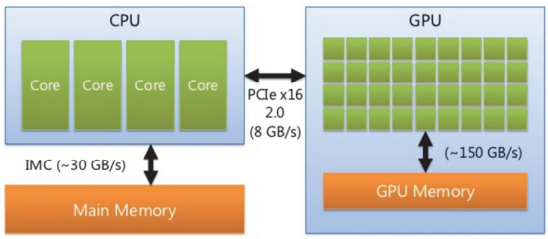


Fig. 1. *A typical architecture of contemporary computational nodes: CPU, GPU, two pools of memory and PCIe link*

In the current article we analyse the performance of finite element core computations, that include finite element numerical integration and assembly, executed on a hardware consisting of a set of classical microprocessor cores and a set of GPUs. We limit our attention to low order approximations for which the execution times of assembly process and communication between different memory pools are the most important. For higher order approximations these times can often be neglected. Moreover, the whole numerical integration procedure may become more complex, with special algorithms, data structures and problem dependent optimizations employed (Klöckner et al., 2009; Komatitsch et al., 2010; Karatarakis et al., 2014; Dziekonski et al., 2012). The phases that we consider are essential to finite element approximation and, in practical terms, lead to the creation of a system of linear equations that reflects the nature of the problem solved and approximation method used. We omit the phase of solving the created sparse system of linear equations, the phase that is extensively studied by many authors (Anzt et al., 2015; Geveler et al., 2013; Reguly & Giles, 2012; Lipski et al., 2015; Dziekonski et al., 2011) and that can be investigated without diving into the numerical details of the finite element method.

We assume that when large scale problems are solved, domain decomposition paradigm is employed for FEM (Smith et al., 1996). The whole computational domain is divided into subdomains and for each subdomain a part of the system of linear equations is created and than the whole system is solved. A set of CPU cores is assigned to a single subdomain together with a set of streaming multiprocessors (SMs) that belong to a single or several GPUs. Whether this SMs belong to one or several GPUs depend on the hardware platform, as well as the software environment capabilities for that platform. We make use of an important feature of domain decomposition technique: when overlapping domain decomposition is employed, then the creation of a part of the global system of linear equations that is necessary for performing calculations for the corresponding subdomain can be done independently for each subdomain, thus without requiring any communication between processors assigned to different subdomains (the communication is required in the later stage of the system solution).

Hence, the scope of the current article is performance analysis of the creation of systems of linear equations for a single subdomain of a finite element approximation problem, for an architecture consisting of several CPU cores and several GPU streaming multiprocessors, both equipped with their DRAM memories and linked through a communication channel with a known bandwidth. For such setting the perfect scalability (weak and strong) is assumed in case of multi-subdomain calculations.

1.1. Previous research

The work presented in the paper is a continuation of our investigations on execution of finite element core calculations on modern hardware platforms. Numerical integration for higher order approximations was discussed in (Banaś et al., 2014a) for GPUs and in (Krużel & Banaś, 2013) for processors with wide (vector) execution units. Numerical integration for low order approximation on multi- and many-core processors was presented in Banaś et al., 2016). These papers describe also a broader context of porting finite element calculations to modern processors and the related articles (such as e.g. Cecka et al., 2011; Dziekonski et al., 2013; Markall et al., 2013). The problem that we discuss in more detail in the current article was also introduced in (Krużel & Banaś, 2013).

2. MODEL PROBLEMS

We consider any finite element solution procedure that can be cast into one problem or a sequence of problems of the form:

Find an unknown function $\boldsymbol{u}$ belonging to a specified function space $\boldsymbol{V}^h$, such that

$$\int_\Omega \left(\boldsymbol{c}^{i;j}\boldsymbol{w}_{,i}\boldsymbol{u}_{,j}+\boldsymbol{c}^{i;0}\boldsymbol{w}_{,i}\boldsymbol{u}+\boldsymbol{c}^{0;i}\boldsymbol{w}\boldsymbol{u}_{,i}+\boldsymbol{c}^{0;0}\boldsymbol{w}\boldsymbol{u}\right)d\Omega$$

$$+\mathrm{BTL}=\int_\Omega \left(\boldsymbol{d}^0\boldsymbol{w}+\boldsymbol{d}^{\mathrm{i}}\boldsymbol{w}_{,\mathrm{i}}\right)\mathrm{d}\Omega+\mathrm{BTR}$$

for every test function $\boldsymbol{w}$ defined in the space $\boldsymbol{V}_0^h$.

Above, Ω is a computational domain in which a sin-

gle PDE or a set of PDEs is defined with suitable boundary conditions imposed on $\partial\Omega$, $\boldsymbol{c}^{i,j}$ and $\boldsymbol{d}^{i}$, $i,j = 0,..,N_D$ denote the coefficients specific to the weak formulation of the problem (with N_D being the number of physical space dimensions), "," denotes differentiation with respect to space coordinates and summation convention for repeated indices is used. BTL and BTR denote boundary terms, usually associated with integrals over the boundary $\partial\Omega$. The task of calculating boundary terms is usually much less computationally demanding, and we neglect it in the current paper, when discussing implementations for multi- and many-core architectures. In the actual computations described later in the paper this task is performed by the part of the code executed by CPU cores, using standard finite element techniques.

In particular, as one of problems solved in numerical examples section, we consider a typical problem in material science, the nonlinear convective heat transfer problem for the unknown temperature field $T(\boldsymbol{x},t)$:

$$\rho c\left(\frac{\partial T}{\partial t}+\boldsymbol{v}\cdot\boldsymbol{\nabla}T\right)-\boldsymbol{\nabla}\cdot(\lambda\boldsymbol{\nabla}T) \;=\; s \qquad (1)$$

with classical Dirichlet, Neumann and Robin boundary conditions. Parameters of density ρ, specific heat c, heat conductivity λ, source s (that takes into account e.g. the latent heat of fusion) are given as functions of materials and temperature (and possibly some other parameters, such as chemical composition). The vector of conductive velocity, $\boldsymbol{v}$, may be given as a function of time and space coordinates or supplied by some coupled solver for the fluid flow.

In the solution procedure we use SUPG stabilization, non-linear implicit Euler scheme in every time step and Picard's iterations for the solution of the resulting one-step non-linear problem (Banaś et al., 2014b). The weak formulation for this particular example is formulated for finding the next iteration T^{n+1}_{k+1} at time step $n+1$, given the solution T^{n+1}_{k} for the previous iteration and T^{n} at the previous time step:

$$\begin{aligned}
&\rho c\int_\Omega \frac{T^{n+1}_{k+1}}{\Delta t}w\,d\Omega \\
&+\int_\Omega \left\{\rho c v_i (T^{n+1}_{k+1})_{,i} w+\lambda (T^{n+1}_{k+1})_{,i} w_{,i}\right\} d\Omega \\
&+\sum_e \int_{\Omega_e} R_k(T^{n+1}_{k+1})\tau R_k(w)d\Omega+\mathrm{BTL} \\
&=\rho c\int_\Omega \frac{T^{n}}{\Delta t}w\,d\Omega \\
&+\int_\Omega s w d\Omega+\mathrm{BTR} \qquad (2)
\end{aligned}$$

Above, $R_k(T)$ and $R_k(w)$ denote residuals of the heat equation (1) computed for respective arguments, while τ is the coefficient of SUPG stabilization.

3. THE INPUT AND OUTPUT DATA OF FINITE ELEMENT CORE PROCESSING

We consider as the input to our computational problem a set of finite element data structures, that store data corresponding to a finite element mesh, one or several approximation fields and the problem solved.

We assume that the first phase of numerical integration is performed for all elements, leading to the creation of element contributions to the global system of equations, the contributions in the form of an element stiffness matrix and a load (right hand side) vector.

As the output of finite element core calculations we consider two variants: in the first the created element contributions are left for the use by the solver, in the second, the assembly process is performed, that lead to the construction of the global system matrix (and the right hand side vector). The first approach suits all strategies that we later term "element-by-element", independent whether they correspond to explicit solution techniques, matrix-free solvers or specific direct solver implementations.

3.1. The idea of stream processing

Finite element meshes are understood in the current article as sets of purely geometrical objects (that will be termed "mesh entities"). We concentrate on volumetric finite elements and 3D problems (the application of the investigations in the paper to 2D or 1D problems is straightforward) and, hence, we consider four types of mesh entities: vertices, edges, faces (quadrilateral and triangular) and elements (tetrahedral, hexahedral, prismatic, pyramidal). These entities are interlinked by several relations of different kinds: some entities *belong* to other entities, some entities are *composed* of other entities, some entities are *neighbours* of other entities, some entities are *parents* of other entities (i.e. have been divided into these entities, with the reverse relation of being a *child* entity). To accommodate such diverse relations, complex data structures for mesh entities are designed (Remacle et al., 2000; Banaś & Michalik, 2010).

Parallel to mesh data structures, structures for storing degrees of freedom data are designed (finite element solutions are represented as linear combinations of FEM basis functions, degrees of freedom are the coefficients of these combinations). The degrees

of freedom are associated with selected mesh entities (depending on the type of approximation (Demkowicz et al., 2007)). The data structures associated with degrees of freedom are usually simpler than those for mesh entities, with less relations, but they must follow changes to mesh data structures induced by FEM adaptivity processes.

We assume that mesh data structures are stored in standard CPU memories that are optimized for low latency accesses. This fact is crucial for efficient realization of operations related to mesh adaptivity, domain decomposition, colouring etc. Due to the close relationship between mesh entities and degrees of freedom, we also assume that data associated with degrees of freedom are stored in CPU main memories (at least during mesh adaptation processes).

Such a form of storage works properly for CPU processing. In order to create the contributions to the linear system matrix and the right hand side vector, the input data are retrieved directly from their original storage locations. The penalty associated with irregular access patterns is mitigated by the low latency of memory. The situation is different for GPU processing. The difference in access times between regular and irregular access patterns can reach two orders of magnitude, annihilating the advantage of fast GPU DRAM memory and making the use of GPUs questionable.

Hence we consider an additional step in processing, we rewrite the input data to the form suitable for stream processing, typical for GPUs, where the data during calculations are accessed from linear arrays in a consecutive way. In order to assess the plausability of the approach we perform the performance analysis of this additional step and the gains it can bring to the finite element processing.

3.2. The input data for finite element processing

Numerical integration is the first stage in the solution of a single finite element problem (a sequence of such problems may be used for solving time dependent and/or non-linear problems). For standard solution procedures numerical integration is the only stage where input data from finite element data structures is used (other, non-standard, procedures include e.g. geometric mutligrid, where mesh data are used for projecting solution between meshes). Therefore, the rewriting of data for stream processing concerns mainly the input data for numerical integration.

In the current article we assume the most common procedure of performing integration separately for each element and producing element stiffness matrices and load vectors, as contributions to the global system of equations. The problem that we want to solve by the numerical integration algorithm for a single finite element is the following: given a set of data defining the geometry of the element and a set of data used to compute the particular problem dependent coefficients for the element, calculate the element system matrix $\boldsymbol{A}^e$ and the load vector $\boldsymbol{b}^e$ corresponding to the weak formulation of the problem solved by the FEM.

The integration is assumed to be done using a numerical quadrature (in the example cases Gaussian quadrature is used). The quadrature is defined for a given problem, approximation and a specific reference element. The change of variables is done for each real finite element, in order to apply the quadrature defined for the reference element. Shape functions, that determine the approximation space for finite elements are also defined for the reference element, and are suitably manipulated by the numerical integration algorithm to create proper integrals over real finite elements (this includes e.g. transformation of local derivatives with respect to reference element coordinates to global derivatives with respect to physical coordinates).

To perform integration for a single element we need the following input data:

- the geometrical description of the element; in the paper we consider only geometrically linear and multi-linear elements for which such descriptions consist of coordinates of element's vertices only
- coefficients related to the weak statement of the problem; these coefficients may have different forms, ranging from several constant numbers for the whole problem, up to the coefficients, usually in non-linear problems, that are different for every element and every integration point within an element, possibly depending on the solution from the previous non-linear iteration or the previous time step.

As an additional input, Gauss quadrature data (coordinates of points and the associated weights) and the values of shape functions and their derivatives with respect to the reference element coordinates at every integration point can be considered. However, these values are the same for all the elements of the same type of approximation and have therefore relatively small size.

Table 1 presents the values of parameters that determine the computational characteristics of 3D finite element numerical integration for an example of reference element – the prismatic element with test func-

Table 1. Parameters determining the computational characteristics of 3D finite element numerical integration – the number of shape functions and the number of Gaussian integration points for the standard prismatic element and different degrees of approximating polynomials

	Degree of approximation p						
	1	2	3	4	5	6	7
N_{sh}	6	18	40	75	126	196	288
N_Q	6	18	48	80	150	231	336

tions being tensor products of 2D shape functions for triangular bases and 1D shape functions in vertical direction. The numbers of gaussian points have been selected to allow for the convergence of finite element solutions for general elliptic problems (Ciarlet, 1978).

3.3. The input data for assembly

The process of assembling the element contributions to the global system of equations gets as its input the output of the numerical integration procedure. Therefore, the assembly and integration are often merged together into a single routine, in order to prevent the hardware from writing element contributions to DRAM and then retrieving them.

Apart from element contributions, the assembly routine needs also the information on the locations where the entries from element contributions should be placed in the global structures. These information is usually stored in the form of local-to-global mapping, for each element the local numbers (identifiers) for degrees of freedom (usually one degree of freedom for one element shape function) are associated with some global numbers of DOFs. The translation of global DOFs' identifiers to the positions in the global system matrix depend on the storage format. Since the system matrices in the finite element computations are sparse (with the percentage of zeros easily exceeding 99.99% for large scale problems) some form of sparse matrix storage format has to be used.

There are many storage formats for sparse matrices (Barrett et al., 1994), row or column oriented, for single entries or blocks of entries, or even special hybrid formats developed recently for GPU solvers (Kreutzer et al., 2014; Koza et al., 2014). All these formats are used in practice and their optimality depends on many factors. There is no single format considered optimal for all problems solved, approximations used and hardware employed.

In the current article we assume that for the purpose of stream processing the input to the assembly procedure is also specially prepared (Banaś & Chłoń, 2016). Instead of having two separate assignments – local-to-global for DOFs and then for each pair of DOFs for which there is a non-zero entry in the system matrix an assignment depending on storage format – there is a single table (we call it "assembly table") that stores a position in the final data structure for the system matrix for each entry in the local element stiffness matrix. Separately, there is another table for the local-to-global mapping that is used for assembling the entries from the element load vector. The assembly tables form an additional input to the assembly procedure, that depends not only on the finite element mesh and approximation, but also on the storage format for system matrices. In the examples that we consider in the current paper, the assembly tables correspond to the CRS storage format, the most popular for CPUs, used also (possibly with some modifications (Koza et al., 2014)) for GPUs.

4. THE SIZE OF FINITE ELEMENT DATA STRUCTURES FOR STREAM PROCESSING

We consider several parameters to estimate the size of finite element data structures. The first one is the number of finite elements, N_E. Element stiffness matrices and load vectors are created for each element, hence the size of linear numerical integration input and output arrays is proportional to the number of elements. The size of input data for geometry depends on the geometrical properties of elements (it can grow substantially for isoparametric elements (Zienkiewicz & Taylor, 2000), for linear elements it is limited to few tens of numbers). The input data related to the problem solved include some number of coefficients, the key aspect is whether these coefficients are constant for all the elements (like some coefficients of linear PDEs), the same for a group of elements (like e.g. constant material properties) or varying across each element. The last case is typical for non-linear and/or time dependent problems where the coefficients depend on the actual solution. Since finite element solutions are stored in the form of degrees of

freedom, the size of input data depends on the number of degrees of freedom associated with different mesh entities. For higher order approximations, the number of degrees of freedom associated with a single edge, face and element interior can reach several tens. Moreover, the problem can be scalar or vector - with the latter case requiring the input being several times larger than for scalar problems.

These facts, together with the variants depending on the storage format for system matrices, make the performance analysis of finite element processing highly problem and approximation dependent. Because of that, in order to make our investigations more concrete, we restrict our attention from that moment to two selected model problems.

The first one is the example in which input and output matrices are small – there are no non-linear coefficients and the first order approximation is used. This case is the Poisson problem with the Laplace operator on the left hand side and an arbitrary right hand side. We study two types of elements: tetrahedral (geometrically linear) and prismatic (geometrically multi-linear). The varying right hand side term appears in the numerical integration as a single separate and possibly different coefficient for each element and each integration point.

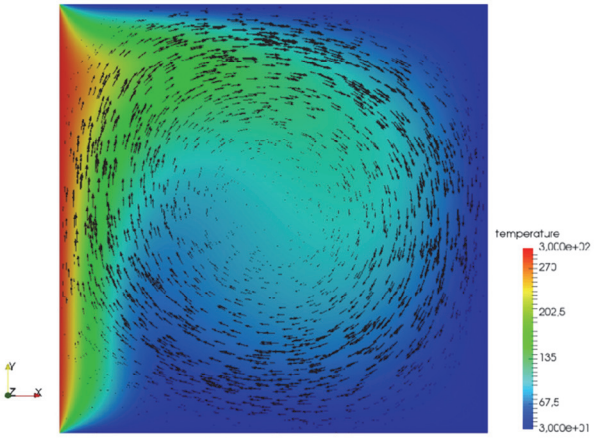


Fig. 2. *The illustration of convective heat transfer calculations (the arrows show the convective velocity field)*

The second problem that we consider is convective heat transfer, defined in Section 1. The same element type and approximation are used as for the first problem. Figure 2 shows the results of example calculations, with temperature field depicted, as well as arrows indicating the velocity field (the problem is stationary, but the solution was obtained using time dependent formulation and pseudo-transient continuation approach).

The difference between the two examples lies in the fact, that for the non-linear time dependent heat transfer problem the coefficients are calculated in a complex way (we use a form of SUPG stabilization), based on the values of solution from the previous nonlinear iteration and the previous time step. Hence, these data has to be retrieved by the processors performing calculations. In the case of GPU calculations, this may require additional transfers between CPU and GPU memories.

In both examples we use classical, continuous, first order approximation, due to at least two reasons. First, this is the most popular type of approximation. Second, for numerical integration and assembly it shows explicitly the influence of memory transfers on the performance of finite element core calculations. As is indicated by the numerical experiments for different orders of approximation (Banaś et al., 2014a), the percentage of time devoted to memory operations is the largest for the lowest orders. This concerns not only PCIe data transfers but also transfers to and from fast GPU memory during calculations. For the Poisson example problem and typical GPU architectures, the time of calculations is solely determined by the time of memory operations (actual arithmetic operations are performed in background) (Banaś & Krużel, 2014). Hence, in order to estimate time of calculations, we need only to estimate the time of all performed data transfers. In the rest of the paper we present such analysis for the selected approximation of the Poisson problem. Then we compare the theoretical estimates with the results of computational experiments. We show also the experimental results for the second problem of convective heat transfer.

4.1. The size of geometrical input data

For the selected for analysis types of geometrically linear elements, the size of geometrical data is determined by the number of vertices for a single element, N_V^e – 4 for tetrahedral elements and 6 for prismatic elements. Hence, this size can be estimated as:

$$N_V^e * N_E * 3 * size\_of\_data$$

4.2. The size of coefficient arrays

For the Poisson problem the size of coefficient arrays corresponds to the number of integration points for selected element and approximation types. It is, hence equal to:

$$N_Q * N_E * size\_of\_data$$

For the convective heat transfer case, where the coefficients are calculated based on the solutions from the previous iteration and the previous time step, i.e.

based on the values of degrees of freedom, there are two options. Either the input data is prepared for each element or the degrees of freedom are stored as a single global vector. In the first case, for the selected first order approximation, where the degrees of freedom are associated with vertices of element only, the size of input data is (for the solution from the previous iteration as well as the solution from the previous time step):

$$N_V^e * N_E * size\_of\_data$$

For the second case, the size is equal to the size of the vector of unknowns. We use the symbol N to denote the global number of degrees of freedom, the main parameter that determines the performance of the solver of linear equations, in our example cases equal to the total number of vertices in the mesh. We do not discuss further theoretically the presented options associated with the heat transfer problem (e.g. we do not discuss the ratio N_E/N for different types of meshes), in the numerical examples we use the first variant.

4.3. The size of output arrays from numerical integration

The output from numerical integration for a single element has the form of an element stiffness matrix having the size $N_{sh} \times N_{sh}$ and a load vector with N_{sh} unknowns (both sizes assume that the problem is scalar). Hence the total size of output data from the numerical integration is:

$$N_{sh} * (N_{sh} + 1) * N_E * size\_of\_data$$

4.4. The size of input arrays for assembling

Apart from element stiffness matrices and load vectors produced by numerical integration, the assembly procedure needs some form of local-to-global mapping for the proper placement of entries from the local arrays into the global data structures. In our case we use explicitly prepared assembly tables, that store directly for each entry in the local arrays its index in the global arrays. Hence the size of assembly tables is directly related to the size of output arrays from numerical integration and equal to:

$$N_{sh} * (N_{sh} + 1) * N_E * size\_of\_data$$

The only difference is that now the size of data corresponds to integer type, instead of some form of floating point.

4.5. The size of output arrays from assembling

The assembly procedure creates the global arrays with the data for the solvers of linear equations. In our case we use the CRS storage format that requires two tables with the size equal to the number of non-zero entries in the stored matrix and one table with N entries (equal to the number of unknowns and also the number of rows in the system matrix). The number of non-zero entries in the global system matrix depends on the characteristics of the finite element mesh and approximation employed. For typical 3D meshes and low order approximations of scalar problems, the number usually does not exceed several tens, while for higher order approximations it can reach several hundreds. In our estimates we assume that it is equal to 20, the value close to the real values that we obtain in our numerical experiments.

5. FINITE ELEMENT STREAM PROCESSING - PERFORMANCE ANALYSIS

As we already pointed out, we assume that the calculations are performed by a set of CPU cores and GPU cores. Our estimates try to express the total solution time taking into account the calculations and the necessary data transfers. We do not explain the details of our GPU programming model (see e.g. (Banaś & Krużel, 2014; Banaś et al., 2016). For the purpose of the current article we assume that, with the proper GPU algorithm design and implementation, the execution time on GPUs depends solely on the time of memory transfers, the fact confirmed by our experiments with low order approximation and several contemporary GPUs.

The general data flow for our finite element core calculations is the following:

- the whole set of finite elements and their faces, for which numerical integration is performed, is dived into colours – the elements or faces of a given colour can be integrated concurrently, since they do not contribute to the same entries in the global system matrix
- the set of all colours is divided into colours processed by CPU cores and colours send for processing to GPU multiprocessors
- the subsets of elements and faces are processed, applying numerical integration only or both, numerical integration and assembly procedures, that lead to the creation of the separate CPU and GPU contributions to the global system matrix and the load vector

- the created contributions are supplied to the solver, either in the assembled form or as a sequence of element contributions (the solver can operate on CPU cores only, GPU cores only or on both sets of cores – the placement of the solver determines the required transfers of its input data)

For the GPU cores performing calculations the data flow is the following:

- the arrays forming input data are copied to the GPU DRAM memory
- the integrals are calculated by subsequent threads and then the local arrays are stored linearly in the memory or assembled into the global system arrays
- the obtained final arrays are left in memory or copied back to the CPU memory, depending on which processors the linear solver is executed

5.1. Performance analysis

Given the sizes of input and output data the execution times can be estimated for different hardware configurations. In order to perform such analysis some assumptions have to be made concerning the execution of different phases of calculations.

The CPU cores performing calculations can retrieve their input data directly from finite element data structures. When GPUs are used for calculations there appears several additional steps forming the overhead of accelerator calculations. The input data have to be written to linear arrays, the arrays have to be transferred to the GPU memory using PCIe bus. As we already mentioned, in the case of the Poisson problem, the execution time of the integration procedure can be estimated with sufficient accuracy using only the times of reading input and writing output data, both for CPUs and GPUs (Banaś et al., 2016). GPU memories are several times faster than CPU memories, the question arises whether the gains coming from faster accesses to GPU memory can counterbalance the overhead associated with GPU calculations.

We present, in table 2, the data concerning the different stages of creating the system of linear equations for the Poisson problem and the two selected element types. We consider reading and writing of input data, for CPU memory, GPU memory and PCIe buses. We estimate the time in nanoseconds for a single element. Additionally, we estimate the time for writing output data as a set of element stiffness matrices and load vectors, taking into account two situations: writing data to consecutive memory locations (in the case without assembling) or writing to locations spread over the CRS arrays. In the last case the memory bandwidth is lower, we assume, to certain degree arbitrary, but based on some previous experience, that access times grow five times (Banaś et al., 2015).

In order to allow for comparisons and creating guidelines for mapping the calculations to architectures, we use exact speeds for different types of memory. These data are based on the performance of average components used in servers for high performance computing. We consider three generations of the popular PCIe bus technology: 2.0, 3.0 and 4.0. This results from the important observation that the times for transferring data between CPU and GPU memory form the most important overhead for GPU finite element core calculations.

Estimates presented in table 2, despite the arbitrariness of assumed speeds, can lead to several conclusions. The first conclusion is that with the current bus technologies it is difficult to make GPU calculation competitive with CPU computing. Especially in the case where numerical integration is performed on GPUs without the assembly. For GPUs performing integration and assembly one can expect the total times comparable to CPU times, still with only slight acceleration of execution speed.

6. NUMERICAL EXAMPLES

To validate the estimates we perform the example calculations on a workstation equipped with the AMD FirePro W8100 graphics card. The only GPU performance characteristic relevant to our estimates is the theoretical DRAM memory bandwidth of 320 GB/s. Simple computational experiments show that the GPU is able to transfer data to and from DRAM with maximal speeds in the range 150-250 GB/s, depending on data type. The system in which the card is deployed has single Intel Core i7 4790 processor, 4 DIMMs of 1600MHz, 12800 MB/s, DDR3 RAM (in dual channel configuration) and Linux operating system (CentOS 7 with 3.10.0 kernel). The experimental DRAM speed reaches 23 GB/s, while the PCIe 3.0 bus, used for CPU-GPU transfers, achieves 12 GB/s.

Tables 3 and 4 present the times for different stages of calculations for two example problems, Poisson and convective heat transfer, respectively. The CPU and GPU processing are considered separately in order to allow for comparisons. In practice CPU and GPU calculations were done concurrently and the created CRS arrays were summed up by the GPU.

Table 2. Estimates for the input and output data transfer times for different possible memory and bus types and the subsequent stages of creation of the system of linear equations for the Poisson problem using first order approximation and two popular types of finite elements: tetrahedral (tetra) and prismatic (prism) (times are reported in nanoseconds for a single element)

		Element type	
		tetra	prism
retrieving input from FEM data structures	CPU DRAM (10 GB/s)	13	19
writing output with assembly	CPU DRAM (10 GB/s)	16	33
writing to linear input arrays	CPU DRAM (50 GB/s)	2.6	4
transfer from CPU memory to GPU memory	PCIe 2.0 (8 GB/s)	16	24
transfer from CPU memory to GPU memory	PCIe 3.0 (15,75 GB/s)	8	12
transfer from CPU memory to GPU memory	PCIe 4.0 (31,5 GB/s)	4	6
reading input data from GPU memory	GPU DRAM (150 GB/s)	0.9	1.3
writing output directly	GPU DRAM (150 GB/s)	1	2
transfer from CPU memory to GPU memory	PCIe 2.0 (8 GB/s)	20	42
transfer from CPU memory to GPU memory	PCIe 3.0 (15.75 GB/s)	10	21
transfer from CPU memory to GPU memory	PCIe 4.0 (31,5 GB/s)	5	11
writing output with assembly	GPU DRAM (30 GB/s)	5	11

Table 3. Execution times for the subsequent stages of the Poisson problem simulation using first order approximation and prismatic elements (times are reported in seconds for a mesh with approx. 2 millions of elements and 1 million of vertices)

Stage of calculations	Execution time
Integration and assembly on CPU	1.545
Transferring input data to GPU	0.132
Integration and assembly on GPU	0.225
Transferring output CRS arrays to CPU	0.025
Solving linear system on CPU	49.69

Table 4. Execution times for the subsequent stages of convective heat transfer simulation using first order approximation and prismatic elements (times are reported in seconds for a mesh with approx. 2 millions of elements and 1 million of vertices)

Stage of calculations	Execution time
Integration and assembly on CPU	2.95
Transferring input data to GPU	0.47
Integration and assembly on GPU	0.66
Transferring output CRS arrays to CPU	0.025
Solving linear system on CPU	13.25

For both problems we show additionally the execution times for the GMRES solver of linear equations, with Gauss-Seidel preconditioning (for the heat problem the number is the average time for all subsequent time steps and non-linear iterations). For the Poisson problem the convergence limit was set as the reduction of the initial residual by 10^{-12}, to obtain almost exact solution. For this case the final percentage of time devoted to numerical integration and assembly performed exclusively on CPU reached around 3% of the total execution time.

The situation was different for the heat problem, where the same solver was used, but with the convergence limit reduced to 10^{-6}. The reduced limit suits well the overall solution procedure, typical for solving stationary problems, where the system of linear equations is solved for subsequent time steps and non-linear iterations, with only the final solution for the last time step required to be accurate. In this case the integration and assembly stages, when performed exclusively on CPU, occupied larger proportion of time, close to 20%. For that kind of simulations even greater proportion of time can be required by integration and assembly (for different problems or different hardware configurations), and hence the proper acceleration of integration and assembly using GPUs may significantly influence the total execution time.

7. CONCLUSIONS

We have shown that GPUs used for creating the systems of linear equations for the finite element simulations can process data faster than CPUs, however the overhead associated with transferring input and output data between CPU and GPU memories (at least for the current PCIe 3.0 technology) makes the feasibility of using GPU processing questionable. In subsequent papers we will investigate in more detail the scenarios of using GPU computing for non-linear and time dependent problems, were the advantages of GPU acceleration become more pronounced.

ACKNOWLEDGEMENT

Financial assistance of the NCN project DEC-2014/13/B/ST8/03812 is acknowledged.

REFERENCES

Anzt, H., Tomov, S., Luszczek, P., Sawyer, W., Dongarra, J., 2015, Acceleration of gpu-based krylov solvers via data transfer reduction, *International Journal of High Performance Computing Applications*, 29(3), 366-383.

Banaś, K., Kruzel, F., Bielanski, J., 2015, Finite element numerical integration for first order approximations on multi-core architectures, *CoRR*, abs/1504.01023.

Banaś, K., Michalik, K., 2010, Design and development of an adaptive mesh manipulation module for detailed FEM simulation of flows, *Proceedings of the International Conference on Computational Science, ICCS 2010*, eds, Peter M. A. Sloot, G. Dick van Albada, and Jack Dongarra, University of Amsterdam, The Netherlands, May 31-June 2, 2010, 1 of Procedia Computer Science, 2043-2051.

Banaś, K., Płaszewski, P., Macioł, P., 2014a, Numerical integration on GPUs for higher order finite elements, *Computers and Mathematics with Applications*, 67(6), 1319-1344.

Banaś, K., Chłoń, K., 2016, Design of interface modules for flexible coupling of finite element codes with solvers of linear equations, *Computer Assisted Methods in Engineering and Science*, 23(1), 3-17.

Banaś, K., Chłoń, K., Cybułka, P., Michalik, K., Płaszewski, P., Siwek, A., 2014b, Adaptive finite element modelling of welding processes, *eScience on Distributed Computing Infrastructure - Achievements of PLGrid Plus Domain-Specific Services and Tools*, eds, Bubak M., Kitowski, J., Wiatr, K., 8500 of Lecture Notes in Computer Science, Springer International Publishing, 391-406.

Banaś, K., Krużel, F., 2014, Opencl performance portability for xeon phi coprocessor and NVIDIA gpus: A case study of finite element numerical integration, *Euro-Par 2014: Parallel Processing Workshops - Euro-Par 2014 International Workshops*, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II, 8806 of Lecture Notes in Computer Science, Springer, 158-169.

Banaś, K., Krużel, F., Bielański, J., 2016, Finite element numerical integration for first order approximations on multi- and many-core architectures, *Computer Methods in Applied Mechanics and Engineering*, 305, 827-848.

Barrett, B., Berry, M., Chan, T.F., Demmel, J., Donato, J.M., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., van der Vorst, H., 1994, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM*, Philadelphia, PA.

Cecka, C., Lew, A. J., Darve, E., 2011, Assembly of 10 finite element methods on graphics processors, *International Journal for Numerical Methods in Engineering*, 85(5), 640-669.

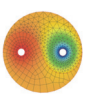
Choi, J. W., Singh, A., Vuduc, R. W., 2010, Modeldriven autotuning of sparse matrix-vector multiply on gpus, *SIGPLAN Not.*, 45(5), 115-126.

Ciarlet, P.G., 1978, T*he Finite Element Method for Elliptic Problems,* North-Holland, Amsterdam.

Demkowicz, L., Kurtz, J., Pardo, D., Paszynski, M., Rachowicz, W., Zdunek, A., 2007, Computing with Hp-Adaptive Finite Elements, *Frontiers Three Dimensional Elliptic and Maxwell Problems with Applications,* Chapman & Hall/CRC.

Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., Dongarra, J., 2012, From CUDA to opencl:
Towards a performance-portable solution for multi-platform GPU programming, P*arallel Computing*, 38(8), 391-407. (Application accelerators in HPC).

Dziekonski, A., Lamecki, A., Mrozowski, M., 2011, Gpu acceleration of multilevel solvers for analysis of microwave components with finite element method, *Microwave and Wireless Components Letters, IEEE,* 21(1), 1-3.



Dziekonski, A., Sypek, P., Lamecki, A., Mrozowski, M., 2012, Finite element matrix generation on a gpu, *Progress in Electromagnetics Research*, 128, 249-265.

Dziekonski, A., Sypek, P., Lamecki, A., Mrozowski, M., 2013, Generation of large finite-element matrices on multiple graphics processors, *International Journal for Numerical Methods in Engineering*, 94(2), 204-220.

Geveler, M., Ribbrock, D., Göddeke, D., Zajac, P., Turek, S., 2013, Towards a complete fembased simulation toolkit on gpus: Unstructured grid finite element geometric multigrid solvers with strong smoothers based on sparse approximate inverses, *Computers & Fluids*, 80(0), 327-332.

Karatarakis, A., Karakitsios, P., Papadrakakis, M., 2014, GPU accelerated computation of the isogeometric analysis stiffness matrix, *Computer Methods in Applied Mechanics and Engineering*, 269, 334-355.

Klöckner, A., Warburton, T., Bridge, J., Hesthaven, J. S., 2009, Nodal discontinuous galerkin methods on graphics processors, *J. Comput. Phys.*, 228, 7863-7882.

Komatitsch, D., Erlebacher, G., Göddeke, D., Michéa, D., 2010, High-order finite-element seismic wave propagation modeling with mpi on a large gpu cluster, *Journal of Computational Physics*, 229(20), 7692-7714.

Koza, Z., Matyka, M., Szkoda, S., Mirosław, Ł., 2014, Compressed multirow storage format for sparse matrices on graphics processing units, *SIAM Journal on Scientific Computing*, 36(2), C219-C239.

Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Bishop, A. R., 2014, A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units, *SIAM J. Scientific Computing*, 36(5).

Krużel, F., Banaś, K., 2013, Vectorized OpenCL implementation of numerical integration for higher order finite elements, *Computers and Mathematics with Applications*, 66(10), 2030-2044.

Krużel, F., Banaś, K., 2015, AMD APU systems as a platform for scientific computing, *Computer Methods in Materials Science*, 15(2), 362-369.

Lipski, P., Wózniak, M., Paszynski, M., 2015, Comparison of the structure of equation systems and the GPU multifrontal solver for finite difference, collocation and finite element method, *Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature*, eds, Koziel, S., Leifsson, L., Lees, M., Krzhizhanovskaya, V., Dongarra, J., Sloot, P. M. A., Reykjavík, Iceland, 1-3 June, 2015, 2014, 51, 1072-1081.

Markall, G. R., Slemmer, A., Ham, D. A., Kelly, P. H. J., Cantwell, C. D., Sherwin, S. J., 2013, Finite element assembly strategies on multi-core and many-core architectures, *International Journal for Numerical Methods in Fluids*, 71(1), 80-97.

Reguly, I., Giles, M., 2012, Efficient sparse matrixvector multiplication on cache-based gpus, *Innovative Parallel Computing (InPar)*, 1-12.

Remacle, J.-F., Karamete, B. K., Shephard, M. S., 2000, *Algorithm Oriented Mesh Database*, Report 5, SCOREC.

Smith, B., Bjorstad, P., Gropp,W., 1996, *Domain Decomposition. Parallel Multilevel Methods for Elliptic Partial Differential Equation,* Cambridge University Press, Cambridge.

Zienkiewicz, O.C., Taylor, R.L., 2000, *Finite element method*, 1-3, Butterworth Heinemann, London.

PODSTAWOWE OBLICZENIA W METODZIE ELEMENTÓW SKON´ CZONYCH I PRZETWARZANIE STRUMIENIOWE

Streszczenie

Artykuł prezentuje wydajnosciowy model wykonania oraz analyze wydajnośsci dla procedury tworzenia układu równań liniowych, która jest jedną z głównych faz obliczeń metodą elementów. Przyjęto założenia, że proces ten będzie wykonywany przez zbiór rdzeni CPU oraz zbiór multiprocesorów GPU. Pamięć wykorzystywana przez CPU i GPU są połąaczone interfejsem PCIe poprzez który przeprowadzany jest tranfer danych. Opracowany algorytm wyko-rzystuje liniową strukturę danych zaprojektowaną specjalnie pod kątem przetwarzania na procesorach GPU, dla którego została przeprowadzona analiza wykonania. W artykule przedstawione zostały wyniki uzyskane dla przykładów obliczeniowych, wykorzystujących liniową aproksymację MES na typowej współczesnej konfiguracj sprzętowej. Zakończenie zawiera wnioski dotyczące praktycznego znaczenia zastosowanego podejścia.

Received: November 11, 2016
Received in a revised form: February 02, 2017
Accepted: February 06, 2017