Maciej Panczyk[1]

EFFECTIVE USE OF MODERN HARDWARE ON THE EXAMPLE OF SELECTED NUMERICAL LIBRARIES IMPLEMENTATION IN BOUNDARY ELEMENT METHOD

Modern multicore CPU processors, frequently supported by many-core GPU processors, provide supercomputer performance for under $300. Unfortunately, it requires advanced programming methods related to OpenMP, MPI, CUDA or OpenCL standards. Instead, many engineering problems can be resolved much faster using such a hardware, due to simple numerical libraries implementations. The paper presents the effects of the use of libraries like BLAS or LAPACK and their recent developments for multithread programming.

Keywords: *boundary element method, numerical libraries, CUDA, OpenCL.*

Мачей Паньчик

ЕФЕКТИВНЕ ВИКОРИСТАННЯ СУЧАСНОЇ КОМП'ЮТЕРНОЇ ТЕХНІКИ НА ПРИКЛАДІ ЗАСТОСУВАННЯ БІБЛІОТЕК ПРОГРАМ ЧИСЕЛЬНОГО АНАЛІЗУ ЗА МЕТОДОМ ГРАНИЧНИХ ЕЛЕМЕНТІВ

У статті показано, що сучасні багатоядерні процесори, разом з багатоядерними графічними процесорами, можуть забезпечити суперкомп'ютерну потужність за вартості обладнання до 300 дол. США. Однак використання таких потужностей передбачає програмну підтримку стандартів OpenMP, MPI, CUDA або OpenCL. Однак багато обчислювальних завдань вирішуються набагато швидше із застосуванням бібліотек програм чисельного аналізу. Представлено приклади застосування таких бібліотек як BLAS або LAPACK та переваги їх застосування при багатопотоковому програмуванні.

Ключові слова: *метод граничних елементів, бібіліотеки програм чисельного аналізу, CUDA, OpenCL.*

Форм. 5. Рис. 2. Табл. 2. Літ. 11.

Мачей Паньчик

ЭФФЕКТИВНОЕ ИСПОЛЬЗОВАНИЕ СОВРЕМЕННОЙ КОМПЬЮТЕРНОЙ ТЕХНИКИ НА ПРИМЕРЕ ПРИМЕНЕНИЯ БИБЛИОТЕК ПРОГРАММ ЧИСЛЕННОГО АНАЛИЗА ПО МЕТОДУ ГРАНИЧНЫХ ЭЛЕМЕНТОВ

В статье показано, что современные многоядерные процессоры, вместе с многоядерными графическими процессорами, могут обеспечить суперкомпьютерную мощность при стоимости оборудования до 300 дол. США. Однако использование таких мощностей предполагает программную поддержку стандартов OpenMP, MPI, CUDA или OpenCL. Однако многие вычислительные задачи решаются намного быстрее с применением библиотек программ численного анализа. Представлены примеры применения таких библиотек как BLAS или LAPACK и преимущества их применения при многопотоковом программировании.

Ключевые слова: *метод граничных элементов, бибилиотеки программ численного анализа, CUDA, OpenCL.*

[1] PhD, Lublin University of Technology, Poland.

Effective use of the computational capabilities of modern many-processors hardware, supported by vector multiprocessors implemented on graphics cards, requires the use of complicated programming standards such as OpenMP, MPI, CUDA or OpenCL. For engineers or even programmers not familiar with these high performance computing standards it is an significant barrier. The question is how to use the enormous performance of modern cheap (available for under $300) equipment without the deep knowledge of programming or multiprocessor hardware. The solution is to allpy the existing and still developing numerical libraries which take full advantage of modern computing equipment.

Widely known and used BLAS (Basic Linear Algebra Subroutine) (BLAS, 2013) and LAPACK (Linear Algebra PACKage) libraries (LAPACK, 2013) allow for effective use of high-speed cache memory build in processors. The application of only those basic library allows to speed up the computation in dozens of times. The second reason for the use of libraries is to facilitate the programmer to deplay the optimized accumulated in the form of library procedures related to the calculations. Instead of coding each time the traditional numerical methods a programmer can simply use the library functions. Constant hardware development entails the creation of new libraries.

General-Purpose computing on Graphics Processing Units (GPGPU) also provides their implementations: cuBLAS – the library provided by NVIDIA Corporation, in their Compute Unified Device Architecture (CUDA) (CUDA, 2013) solution, CULA (CULA, 2013) – a set of GPU-accelerated linear algebra libraries utilizing the NVIDIA CUDA parallel computing architecture. Similar solutions for open standard – OpenCL (Open Computing Language) (KHRONOS, 2013): APPML (Accelerated Parallel Processing Math Libraries) (AMDCL, 2013) maintained by ATI and the non-profit technology consortium Khronos Group.

These technologies dramatically improve the computation speed of sophisticated mathematics. The effects of the implementation of selected numerical libraries were traced on the example of boundary element method calculation of two-dimensional planar capacitor model, described by constant boundary elements.

Diagram of the Boundary Element Method

To trace the possibilities of application of numerical libraries and multithread programming standards in a boundary element method a few steps of BEM (Sikora, 2007) scheme will be presented.

The first step to use the boundary element method is to transform partial differential equations into the system of boundary integral equations (BIE) which after discretization can be written as (Sikora, 2007):

$$C(r)\Phi(r)+\sum_{j=0}^{M-1}\Phi_j(r')\int_{-1}^{+1}\frac{\partial G(|r-r'|)}{\partial n}J(\xi)d\xi=\sum_{j=0}^{M-1}\frac{\partial \Phi_j(r')}{\partial n}\int_{-1}^{+1}G(|r-r'|)J(\xi)d\xi, \qquad (1)$$

where: r is the so-called load point on the boundary, r' is the "field" point, C is the free term $C(r)=0.5$, M is the total number of elements, Φ represents the potential function (electric potential in electrostatics), G is the fundamental solution for Laplace equation, J is the Jacobian of transformation from global (r) to local (ξ) coordinate system. The values of Φ and $\partial\Phi/\partial n$ (normal derivative of Φ, normal vector n points outward the domain) are assumed to be constant on each element and equal to the value at the mid-node of the constant boundary element.

The integral functions in the Eq. (1) can be lumped together in $A_{i,j}$ and $B_{i,j}$ as follows:

$$C(r)\Phi(r)+\sum_{j=0}^{M-1}\Phi_j(r')A_{i,j}(r,r')=\sum_{j=0}^{M-1}\frac{\partial\Phi_j(r')}{\partial n}B_{i,j}(r,r') \qquad (2)$$

The index i from Eq. (2) denotes source points on each element. So, the second step is to calculate each of $A_{i,j}$ and $B_{i,j}$ elements.

To form a set of linear algebraic equations, we take each node in turn as a load point r and perform the integrations indicated in Eq. (1). This will result in the following matrices:

$$[A][\Phi]=[B]\left[\frac{\partial\Phi}{\partial n}\right] \qquad (3)$$

Matrices $[A]$ and $[B]$ contain functions $A_{i,j}$ and $B_{i,j}$ respectively.

Each of the M elements has either a known Dirichlet boundary condition or an unknown Neumann boundary condition. Hence, Eq. (3) has to be rearranged in order to introduce the prescribed boundary conditions and group them in the vector x which contains the unknowns and the vector b' which contains known, prescribed boundary conditions, the Eq. (3) can be transformed to:

$$Ax=Bb' \qquad (4)$$

In Eq. (4), matrices A and B are formed by a combination of the columns of $[A]$ and $[B]$ depending on the boundary conditions, i.e., according to whether the values of $[\Phi]$ or $[\partial\Phi/\partial n]$ are known in a given element j or not. By multiplying matrix B and b' we will receive rhs vector and the final algebraic equation:

$$Ax=rhs \qquad (5)$$

Model description

A simple two-dimensional model of a planar electric capacitor with the inner square region was discretized in a similar way as presented in Figure 1. The real calculated object was divided into 128 elements on each side of the rectangle, both external and internal. A total of 1024 constant boundary elements were received for which calculations were performed. The resulting square matrices A and B and vector rhs have the size of 1024*1024 elements.

The Dirichlet boundary conditions were set as follows: on the upper edge $\Phi=10V$ and lower $\Phi=-10V$. Neuman boundary conditions $\partial\Phi/\partial n=0$ on the left and right edges.

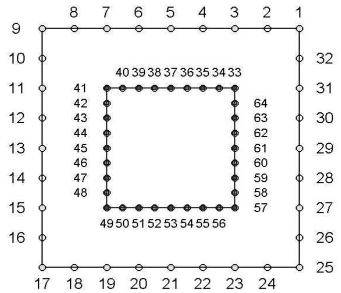

Figure 1. **Two-dimensional planar capacitor model with internal square dielectrical object discretizations (decreased for better readability)**

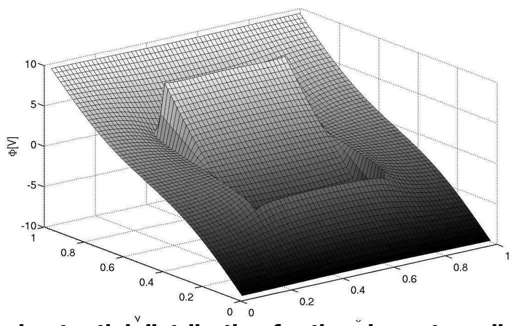

Figure 2. **Calculated potential distribution for the planar two-dimensional capacitor**

The calculations were performed on a computer with a six-core AMD Phenom II X6 1090T processor and a graphics cheap ($70) card GeForce GT 430.

Numerical libraries overview

In addition to traditional numerical libraries like BLAS, LAPACK and their parallel versions PBLAS and SCALAPACK rapid development of based on graphic cards GPU multiprocessors and related numerical libraries were performed. GPGPU programming (general-purpose computing on graphics processor units) uses SIMD (single instruction, multiple data) idea which allows for some issues to achieve a very high performance. It is necessary to adopt the traditional algorithms to specific GPU multiprocessor cache memory structure. Additionally SIMD processors can use only non-recursive algorithms. Usually it requires to replace the traditional numerical algorithms with their less effective non-recursive versions possible to run on the GPU units. Nevertheless executing such task on a few dozen or a few thousand GPU processors dramatically increase the calculation speed.

Adding a GPU-acceleration to our application could be performed simply by calling a library function. The list of some high performance GPU-accelerated libraries related to NVIDIA CUDA system is below (CUDA, 2013):

– CUBLAS: Dense Linear Algebra – the equivalent of BLAS libraries for NVIDIA GPUs implemented on top of the CUDA driver. CUBLAS allows to fill GPU memory with matrix or vector, to call a sequence of CUBLAS functions and finally to return results back to the host memory.

– CUSPARSE : Sparse Linear Algebra – used for sparse matrices.

– CUFFT: Fourier transforms – used to compute discrete Fourier transform of complex or real data sets

– LIBM: Standard C Math library,

– CURAND: Pseudo-random and Quasi-random numbers generation,

– NPP: The NVIDIA Performance Primitives library (NPP) – a collection of image, video, and signal processing functions,

– Thrust: STL-Like Primitives Library delivering GPU-accelerated sort, scan, transform, and reduction operations.

– OpenACC – user-friendly similar to OpenMP programming method based on GPU "directives",

– CULA tool – CULA dense – provides accelerated implementations of the LAPACK and BLAS libraries for dense linear algebra. Contains routines for systems solvers, singular value decompositions, and eigenproblems. CULA sparse which provides tools for solving large sparse systems.

More similar solutions can be found on the NVIDIA Developer Zone (CUDA, 2013).

An open standard OpenCL also supports such libraries development as (AMDCL, 2013):

– APPML (Accelerated Parallel Processing Math Libraries) – AMD library linking BLAS and FFT functions,

– ViennaCL – Library open-source library which support CUDA, OpenCL and OpenMP, all levels of BLAS for GPUs and multi-core CPU.

– MAGMA (Matrix Algebra on GPU and Multicore Architectures) – very interesting project delivering linear algebra algorithms for hybrid manycore CPU and multicore GPU systems.

It should be also mentioned that calculations on systems with multi-GPU cards require using the MPI standard to utilize more than one graphic card.

Implementation of selected numerical libraries

Before implementing any numerical libraries or code parallelization standard Boundary Element Method program without any accelerators was investigated to find its computation time and to analyze various stages of the BEM algorithm.

For the model composed of 256 constant boundary elements on each side of the boundary object computation time is presented in Table 1.

Table 1. **Analysis of execution times of selected stages of the BEM algorithm**

#	BEM algorithm stage	Computation time
1	Getting input data - geometry and boundary conditions	140 ms
2	Loops calculating source point impacts on observation points – equation (4) stage	10 s
3	Data sorting - the stage of equation (5)	1 s
4	Equation (5) matrix LU decomposition	2 min 26 s
5	The solution of equation (5)	60 ms

Based on these results of calculating the time presented in Table 1, MEB algorithm was divided into three parts (in order of their influence to global algorithm calculation speed improvement):

– acceleration of equations (5) solving – steps 4 and 5 according to Table 1,

– acceleration of determining elements $A_{i,j}$ and $B_{i,j}$ – steps 2 and 3 according to Table 1,

– minor acceleration related to the rest of the BEM algorithm.

The most important, first step, was carried out using a sequence of classical BLAS and LAPACK functions (DGESV and DGEMM) used to find the solution vector x of equation (5). Next the library SCALAPACK – PDGESV function was used. The last implementation used CUDA (SITPICZYNSKI, POTIOPA, 2011, SANDERS, KANDROT, 2011) and OpenCL (Scarpino, 2011) libraries (AMDCL, 2013) for calculations. For CUDA environment the CULA library commercial (but free for registered universities) double precision DGESV function was implemented.

Although OpenCL APPML (Accelerated Parallel Processing Math Libraries) does not include similar LU decomposition algorithm, available from AMD

Developer Zone web site example (LUDOpenCLBLAS-linux.zip) (AMDCL, 2013) was adopted to speed up the equation (5) solution. For the second and the third steps a user-friendly OpenMP environment was applied. All of these accelerated methods were compared with the pure algorithm, written in C++. The calculation results were identical, although the computation time was different. The results are presented in Table 2.

Table 2. **Computation time for the program without the use of numerical libraries, with BLAS and LAPACK libraries, using CULA library (CUDA) and using APPML and OpenCL LU method**

	Without acceleration	BLAS / LAPACK	BLAS / LAPACK + OpenMP	CUDA-CULA	OpenCL-APPML+LU
Calculation time	10 min 22 s	54 s	57 s	22 s	25 s

Conclusions

Effective usage of modern equipment allows us to decrease the computation time from 10 min 22 s to 22 seconds. Thet result was achieved on a simple, typical for home use hardware. More advanced applications can use solutions dedicated to massively parallel multiprocessing like NVIDIA K20 GPU multiprocessors for about $4000. It is interesting to compare it to famous chess matches between Gary Kasparov and IBM Deep Blue computer. That IBM RS/600 30-node system achieved the computation speed of about 11 Gflops. Nvidia K20 floating point performance is about 1,17 Tflops and used above PC with GT 430 card about 250 Gflops.

It is important to popularize the effective usage of modern hardware. Presented application is related with the computer tomography used for screening examinations of breast cancer.

The results presented in Table 2 confirm the need to use numerical libraries. Even simple optimization of CPU cache memory, allows us to speed up calculations for at least 10 times. Generally, complex solution should include CPU multithread programming standards like OpenMP and MPI, library usage with support of CUDA / OpenCL GPGPU programming elements (Labaki at al., 2011). This is the reason that points further considerations to MAGMA library. Low cost NVIDIA GeForceGT 430 unit allows us to shorten the calculation speed from 54 to 22 seconds – more than twice. Tested amount of data (maximum 4096 elements were calculated) was not enough to implement the typical OpenMP parallel loop and sections – the calculation time increased from 54s to 57s (see Table 2).

Most of the numerical libraries were originally written in FORTRAN and then transferred to the C/C++. Therefore, the matrices A and B should be passed to library functions in column order as in Fortran.

To use GPU computing it is still necessary to initialize the GPGPU environment, to allocate GPU memory and to transfer data from the host to the GPU unit and reverse.

Generally, the use of basic libraries is not difficult and brings a huge performance boost. GPU versions of these libraries are not so easy to apply, but still a lot easier than full GPGPU programming including a grid thread and data optimization (Labaki at al., 2011).

References:

AMD OpenCL Zone (2013). http://developer.amd.com/resources/heterogeneous-computing/opencl-zone.

BLAS Homepage: http://www.netlib.org/blas.

CULA Programmer's guide (2013). http://www.culatools.com/cula_dense_programmers_guide.

KHRONOS OpenCL Homepage: http://www.khronos.org/opencl.

Labaki, J., Ferreira, L., Otavio, S., Mesquita, E. (2011). Constant Boundary Elements on graphics hardware: a GPU-CPU complementary implementation, J. Braz. Soc. Mech. Sci. & Eng., 33(4): 475–482.

LAPACK Homepage: http://www.netlib.org/lapack.

NVIDIA CUDA Developer Zone (2013). https://developer.nvidia.com.

Sanders, J., Kandrot, E. (2011). CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley.

Scarpino, M. (2011). OpenCL in Action: How to Accelerate Graphics and Computations; Manning Publications co., NY.

Sikora, J. (2007). Boundary Element Method for Impedance and Optical Tomography, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa.

Stpiczynski, P., Potiopa, J. (2011). Solving a kind of boundary-value problem for ordinary differential equations using Fermi – the next generation CUDA computing architecture. J. Computational Applied Mathematics, 236(3): 384–393.

Стаття надійшла до редакції 22.07.2013.