DOI: 10.5604/20830157.1121359

ZASTOSOWANIE BIBLIOTEK NUMERYCZNYCH W OBLICZENIACH MEB

Krzysztof Król[1], Maciej Pańczyk[2]
[1]Politechnika Lubelska, Instytut Elektroniki i Technik Informacyjnych, [2]Politechnika Lubelska, Instytut Informatyki

Streszczenie. *Zastosowanie bibliotek numerycznych pozwala na znaczne skrócenie czasu obliczeń i ułatwienie pisania kodu programu. Popularne biblioteki BLAS i LAPACK doczekały się dojrzałych implementacji pozwalających na wykorzystanie procesorów wielordzeniowych i środowisk obliczeń rozproszonych w postaci odpowiednio PBLAS i SCALAPACK. Aktualnie podobny proces rozwoju dotyczy środowisk związanych z obliczeniami wykonywanymi na procesorach GPU w dwóch głównych implementacjach GPGPU: NVIDIA CUDA i Kronos/ATI OpenCL. Równolegle z rozwojem tych ostatnich toczą się prace nad mieszanymi CPU-GPU wersjami tych bibliotek czego doskonałym przykładem jest MAGMA. W artykule przedstawione zostaną efekty implementacji kilku wybranych bibliotek z tego zakresu zastosowanych do rozwiązania dwuwymiarowego modelu kondensatora płaskiego metodą elementów brzegowych wykorzystującą stałe elementy brzegowe.*

Słowa kluczowe: MEB, biblioteki numeryczne, CUDA, OpenCL

NUMERICAL LIBRARY USAGE IN BEM

Abstract. *Numerical library usage effectively reduce computation time and facilitate code programming. There are modified versions of popular BLAS and LAPACK libraries, dedicated to multi-core and distributed programming respectively PBLAS and SCALAPACK. Currently, a similar development applies to the GPU programming in two major implementations of GPGPU: NVIDIA CUDA and Kronos / ATI OpenCL. In the same time hybrid CPU-GPU versions of these libraries are intensively developed, a good example of that is MAGMA. This paper will present the effects of some of those libraries implementation used to solve the two-dimensional planar capacitor model by the boundary element method with constant boundary elements.*

Keywords: MEB, numerical libraries, CUDA, OpenCL

Wstęp

W celu pełnego wykorzystania możliwości obliczeniowych współczesnego sprzętu niezbędne jest korzystanie z bibliotek numerycznych. Powszechnie znane i stosowane biblioteki BLAS (Basic Linear Algebra Subroutine) i LaPACK (Linear Algebra PACKage) pozwalają na efektywne wykorzystanie szybkiej pamięci typu cache procesorów. Już samo zastosowanie tych podstawowych bibliotek pozwala na przyspieszenie obliczeń nawet kilkadziesiąt razy. Drugim powodem używania bibliotek jest ułatwienie programiście korzystania ze zoptymalizowanych, zgromadzonych w postaci bibliotek procedur związanych z obliczeniami. Zamiast programować za każdym razem od nowa znane metody numeryczne wystarczy skorzystać z gotowych funkcji.

Nieustanny rozwój sprzętu pociąga za sobą powstawanie nowych wersji bibliotek. Kolejno powstały więc równoległa wersja PBLAS (Parallel BLAS), rozproszona wersja LAPACK określana jako ScaLAPACK (Scalable LAPACK). Również rozwój procesorów oferujących rozwiązania typu SIMD głównie jednostek GPU opartych o procesory wektorowe kart graficznych pociągnął za sobą nawet w samym ostatnim roku rozwój i pojawienie się wielu ciekawych rozwiązań. Standardy programowania takie jak Nvidia CUDA (Compute Unified Device Architecture) czy Kronos/ATI OpenCL (Open Computing Language) doczekały się swoich wersji bibliotek BLAS/LAPACK. Efekty implementacji wybranych bibliotek numerycznych warto prześledzić na przykładzie obliczeń Metodą Elementów Brzegowych. Operacje na dużych pełnych macierzach intensywnie wykorzystujących pamięć, w dużej mierze nie rekurencyjne, doskonale pasują do modelu obliczeń GPGPU. Prosty dwu-wymiarowy model kondensatora płaskiego, opisanego stałymi elementami brzegowymi pozwala skoncentrować się na zagadnieniach wykorzystania poszczególnych bibliotek w zależności od typów procesorów i środowisk obliczeniowych.

Rozwiązania wykorzystujące współczesne procesory wielordzeniowe i multiprocesory są bardzo użyteczne dla wielu problemów naukowych. W celu wykorzystania używania takich rozwiązań jak OpenMP i MPI potrzebne są dodatkowe umiejętności programowania. Obliczenia GPU (Graphics Processing Unit) mają ogromny potencjał, ale programowanie tych urządzeń jest dużym wyzwaniem. Jest to bariera dla wielu zastosowań technicznych i inżynierskich. Z tego powodu praktyczne zastosowanie kolejnych wersji współczesnych bibliotek numerycznych wydaje się warte prześledzenia.

1. Schemat metody elementów brzegowych

Metoda Elementów Brzegowych po przekształceniu równania różniczkowego do równania całkowo-brzegowego (Boundary Integral Equation – BIE) oraz dyskretyzacji elementów brzegowych może zostać zapisana jako [4]:

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

gdzie:

r – jest ustalonym punktem zewnętrznym tzw. punktem obserwacji, w którym wyznaczamy (obserwujemy) wartość Φ,

r' – odpowiada tzw. punktowi źródła, współczynnik C(r)=0,5,

M – jest całkowitą ilością elementów,

Φ – reprezentuje funkcję potencjalną (potencjał elektryczny w przypadku pola elektrostatycznego),

G – jest rozwiązaniem fundamentalnym równania Laplace'a,

J – jest Jakobianem transformacji ze współrzędnych globalnych (r) do współrzędnych lokalnych (ξ).

Wartości Φ i $\partial\Phi/\partial n$ (pochodna normalna Φ, n – wektor normalny skierowany na zewnątrz obszaru), są traktowane jako stałe dla każdego elementu, wartości te są wyznaczane w połowie długości każdego elementu brzegowego.

Zapisując funkcje podcałkowe z równania (1) jako elementy $A_{i,j}$ i $B_{i,j}$ otrzymujemy równanie:

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

Indeks *i* z równania (2) oznacza punkty źródłowe każdego elementu. Dla każdego węzła (punktu obserwacji) *r* wykonuje się całkowania wskazane w równaniu (1). Ostatecznie prowadzi to do zapisu macierzowego równania (1) postaci:

$$[A]\cdot[\Phi]=[B]\cdot\left[\frac{\partial\Phi}{\partial n}\right] \tag{3}$$

Macierze [A] oraz [B] zawierają odpowiednio funkcje $A_{i,j}$ oraz $B_{i,j}$.

Dla każdego elementu M znany jest warunek brzegowy Dirichleta lub warunek brzegowy Neumanna. Stąd równanie (3) musi być uporządkowane tak aby wprowadzić wymagane warunki brzegowe i pogrupować je w wektor **x** zawierający niewiadome oraz wektor **b'**, zawierający dane (znane warunki brzegowe).

Po stosownych przekształceniach równanie (3) przyjmie postać:

$$\mathbf{Ax} = \mathbf{Bb'} \tag{4}$$

W równaniu (4) macierze **A** i **B** są utworzone przez połączenie kolumn z obu macierzy w zależności od elementów brzegowych, tzn. w zależności od tego czy znane są wartości Φ lub $\partial\Phi/\partial$n w zadanym elemencie *j*. Poprzez przemnożenie macierzy **B** i wektora **b'**, otrzymuje się wektor **rhs** i w ostateczności równanie przyjmuje postać:

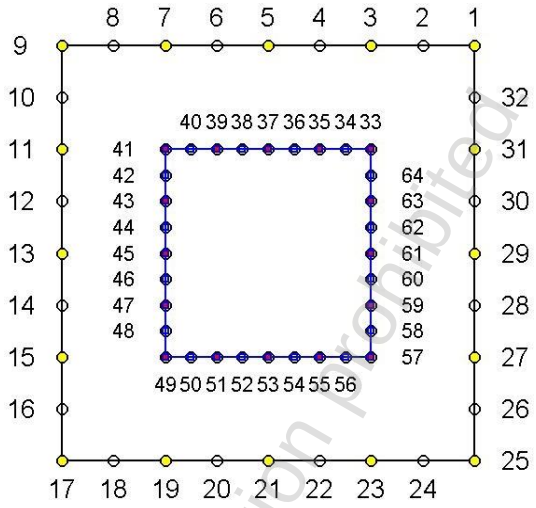
$$\mathbf{Ax} = \mathbf{rhs} \tag{5}$$

2. Model obiektu

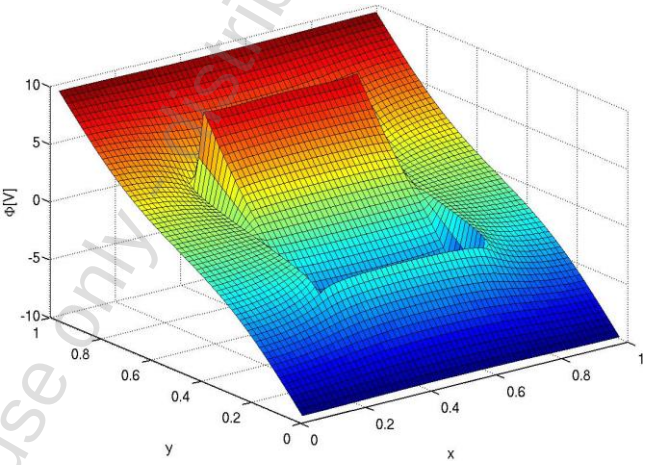
Prosty dwuwymiarowy model płaskiego kondensatora elektrycznego z wewnętrznym obszarem kwadratowym został dyskretyzowany w podobny sposób jak to jest przedstawione na rys. 1. W rzeczywistości dokonano podziału na 128 elementów, rozmieszczonych na każdym boku modelu zarówno zewnętrznym jak i wewnętrznym. W sumie otrzymano siatkę 1024 elementów brzegowych, dla których przeprowadzone zostały obliczenia. Powstałe kwadratowe macierze **A** i **B** oraz wektor **rhs** miały więc rozmiar 1024 elementów.

Warunki brzegowe Dirichlet'a zostały zdefiniowane w sposób następujący: górny element Φ=10V dolny Φ = -10V. Warunki brzegowe Neuman'a na lewym i prawym brzegu wynosiły $\partial\Phi/\partial$n=0. Na elementach górnym i dolnym poszukiwano wartości pochodnej normalnej $\partial\Phi/\partial$n. Na elementach bocznych obliczeniu podlegały wartości potencjałów Φ. Na brzegu obszaru wewnętrznego na każdym z elementów wyznaczano potencjały i pochodne normalne.

Obliczenia wykonano na komputerze wyposażonym w kartę graficzną GeForce GT 430 z pamięcią 1GB oraz sześciordzeniowy procesor AMD Phenom II X6 1090T.



Rys. 1. Model dwuwymiarowego kondensatora – dyskretyzacja 4 elementy na brzegu



Rys. 2. Wyznaczony rozkład potencjału dwuwymiarowego kondensatora, dyskretyzacja 64 elementy na brzegu

3. Przegląd bibliotek numerycznych

Oprócz tradycyjnych bibliotek numerycznych jak BLAS. LAPACK i ich równoległych odmian PBLAS i SCALAPACK, gwałtowny rozwój multiprocesorów realizujących idee SIMD pozwalających dla wybranych zagadnień osiągać bardzo wysokie wydajności, pociągnął za sobą intensyfikacje prac nad bibliotekami wspierającym tego typu sprzęt.

Prace dotyczą zarówno optymalizacji wykorzystania specyficznej struktury pamięci podręcznej multiprocesorów jak i zastąpienia klasycznych algorytmów numerycznych ich pozornie mniej efektywnymi nie-rekurencyjnymi wersjami możliwymi do uruchamiania na jednostkach GPU. Wspomniana pozorna nieefektywność wiąże się często z dodaniem pewnych kroków do algorytmów co zmniejsza przykładowo kilkukrotnie ich szybkość jednak przy możliwości uruchomienia zadania na kilkudziesięciu czy kilku tysiącach procesorów jednostek GPU oznacza to ogromny wzrost wydajności obliczeń.

Odpowiednio wymienić można biblioteki związane ze standardem Nvidia CUDA:

- CUBLAS – odpowiednik biblioteki BLAS dla procesorów GPU firmy NVIDIA.
- CULA tool – produkt firmy EM Photonics będący odpowiednikiem rodziny funkcji znanych z LAPACK.
- CUFFT – jak wskazuje nazwa równoległa wersja szybkiej transformaty Fouriera dla różnego typu danych.
- CURAND – równoległe generowanie liczb pseudo- i quasi-losowych.
- CUSPARSE – oferująca cztery poziomy operacji na macierzach rzadkich.

Dwa bardzo ciekawe rozwiązania stawiające na łatwość implementacji:

- Thrust – biblioteka szablonów bazujących na STL (Standard Template Library) oferująca gotowe algorytmy związane przykładowo z sortowaniem czy znanymi z głównymi standardami programowania operacjami typu redukcja.
- OpenACC – bardzo ciekawe rozwiązanie zapewniające przyjazny znany z OpenMP sposób programowania GPU oparty o "dyrektywy".

Więcej podobnych rozwiązań realizujących specyficzne zagadnienia jak przykładowo przetwarzanie obrazów, sygnałów, symulacja ruchu fal oceanicznych i inne można znaleźć na stronach NVIDIA Developer Zone [11] pod hasłem GPU-Accelerated Libraries.

Otwarty standard OpenCL również wspiera programistów bibliotekami takimi jak:

- APPML (Accelerated Parallel Processing Math Libraries) – biblioteka AMD łącząca funkcje BLAS i FFT dla procesorów GPU;
- ViennaCL – biblioteka typu open-source wspierająca standardy CUDA, OpenCL i OpenMP wszystkich poziomów BLAS dla procesorów GPU i wielordzeniowych CPU.

Należy jeszcze wspomnieć o bardzo istotnym trendzie jakim jest łączenie w postaci jednej biblioteki obliczeń CPU-GPU. Doskonałym przykładem takiego dojrzałego już rozwiązania jest biblioteka MAGMA. Jest to efekt projektu koncentrującego się na operacjach dotyczących macierzy gęstych podobnie jak LAPACK jednak w łączonych środowiskach wielordzeniowych CPU wspartych akceleratorami GPU.

Możliwe, że w przyszłości pojawią się również rozwiązania pozwalające na zaoferowanie gotowych narzędzi dla systemów z wieloma kartami multiprocesorowymi GPU, które jak dotąd wymagają wsparcia poprzez wymianę komunikatów w standardzie MPI.

4. Wybrane elementy bibliotek numerycznych

Pierwszym etapem prac przed zaimplementowaniem jakichkolwiek bibliotek numerycznych czy też zrównolegleniem kodu była analiza czasu obliczeń poszczególnych etapów algorytmu MEB. Wykorzystano do tego celu nieco prostszy model złożony z 64 stałych elementów brzegowych na każdym boku obiektu (analogicznie jak na rys. 1) zarówno zewnętrznego jak i wewnętrznego. Otrzymane czasy realizacji poszczególnych etapów obliczeń przedstawiono w tabeli 1.

Tabela 1. Analiza zagadnienia pod kątem czasów wykonania poszczególnych etapów algorytmu MEB zrealizowana dla modelu składającego się łącznie z 512 elementów

Lp.	Etap algorytmu MEB	Czas wykonania
1	Wprowadzenie danych o geometrii i warunkach brzegowych	0,140”
2	Pętle oddziaływań punktów źródła na kolejne punkty obserwacji - etap równania (4)	10”
3	Uporządkowanie danych stronami – etap równania (5)	1”
4	Rozkład LU macierzy równania (5)	2’26”
5	Rozwiązanie równania (5)	0,060”

Na podstawie otrzymanych wyników czasów obliczeń klasycznego kodu C++ rozwiązywane zadanie MEB podzielono na 3 części: przyspieszenie rozwiązania układu równań (4) – etapy 4 i 5 wg tabeli 1; przyspieszenie wyznaczania elementów $A_{i,j}$, $B_{i,j}$ i porządkowania danych – etapy 2 i 3 oraz „kosmetyka” pozostałych fragmentów kodu.

Najważniejszy pierwszy etap zrealizowano stosując kolejno klasyczne elementy bibliotek BLAS i LAPACK. W obliczeniach zastosowano funkcje DGEMM i DGESV, które pozwoliły wyznaczyć wartość wektora **x** z równania (5). Uzyskany czas obliczeń umieszczono w trzeciej kolumnie tabeli 2: BLAS/LAPACK – 54 s.

Do następnego porównania użyto biblioteki SCALAPACK – funkcji PDGESV zastosowanej do rozwiązania układu równań (5). Czas obliczeń 57 s, umieszczono w czwartej kolumnie tabeli 2.

Jako ostatnich w realizacji etapu pierwszego użyto obliczeń z wykorzystaniem karty graficznej. Czas obliczeń 22 s (kolumna 5 w tabeli 2) otrzymano wykorzystując środowisko Nvidia CUDA - funkcję DGESV biblioteki CULA.

Ostatni wynik, przedstawiony w kolumnie 6 tabeli 2, odpowiada zastosowaniu algorytmu rozkładu LU osiągalnego na stronach AMD Developer Zone (LUDOpenCLBLAS-Linux.zip) [11] w środowisku OpenCL. Kod ten nie jest jeszcze włączony do biblioteki APPML (Accelerated Parallel Processing Math Libraries).

Drugi etap – wyznaczenie oddziaływań punktów źródeł na kolejne punkty obserwacji - zrealizowano korzystając z przyjaznego programiście środowiska OpenMP. Standard OpenMP zastosowano jedynie do obliczeń wielowątkowych wykorzystujących wyłącznie jednostki CPU. Przedstawione w tabeli 2 implementacje Metody Elementów Brzegowych wykorzystujące zastosowania różnych bibliotek numerycznych zostały porównane z czystym algorytmem napisanym w języku C++. Wyniki obliczeń w każdym z przypadków były identyczne, ale czas obliczeń był różny. Rezultaty przedstawiono w tabeli 2.

Tabela 2. Czas obliczeń dla programu bez użycia bibliotek numerycznych, z bibliotekami BLAS i LAPACK, z wykorzystaniem biblioteki CULA (CUDA) oraz z wykorzystaniem APPML i LU w wersji OpenCL. Model zbudowany z 1024 elementów

	bez użycia bibliotek	BLAS / LAPACK	SCALAPACK + OpenMP	CUDA -CULA	OpenCL-APPML+LU
czas obliczeń	10’22”	54”	57”	22”	25”

5. Podsumowanie

Pierwsza grupa uwag dotyczących zastosowania bibliotek numerycznych w obliczeniach MEB związana jest z łatwością użycia funkcji bibliotecznych oraz podobnie rozumianą różną przyjaznością poszczególnych typów bibliotek.

Geneza bibliotek numerycznych BLAS i LAPACK związana jest z językiem FORTRAN. Większość bibliotek numerycznych do obliczeń równoległych dla kart graficznych została napisana w C/C++ oraz w Fortranie. W związku z tym w standardach OpenMP i MPI musimy pamiętać, że macierze **A** i **B** należy przechowywać w pamięci kolumnami jak w Fortranie.

Kolejnym ważnym rozróżnieniem bibliotek jest to, iż niektóre z bibliotek GPU wymienionych na NVIDIA Developer Zone [11] są dostępne w ramach otwartych licencji, podczas gdy inne są płatne. Wśród rozwiązań komercyjnych znajdują się przykładowo CULA [8] (chociaż jest ona darmowa dla środowisk akademickich). W pełni darmowa wersja umożliwia jedynie obliczenia pojedynczej precyzji, natomiast dla rozwiązań z podwójną precyzją wymagany jest w zastosowaniach komercyjnych zakup licencji.

Innym istotnym krokiem w kierunku popularyzacji użycia obliczeń GPGPU jest narzędzie Jacka Dongarra MAGMA. Oprogramowanie to może w pełni wykorzystać moc współczesnych systemów heterogenicznych procesorów wielordzeniowych i używać wielu kart graficznych. Bardzo ciekawym rozwiązaniem jest OpenACC, który zapewnia przyjazny sposób programowania oparty o "dyrektywy" podobne do stosowanych w OpenMP. Druga grupa omawianych problemów dotyczy przebiegu obliczeń i zastosowanego modelu.

Z analizy etapów i czasów obliczeń przedstawionych w tabeli 1 wynika, że zrównoleglenie procesu obliczania oddziaływania poszczególnych punktów źródeł na kolejne punkty obserwacji będzie miało znikomy wpływ na ogólny czas obliczeń. Decydujące znaczenie odgrywa czas rozkładu LU macierzy równania (5).

Rozmiar macierzy w sposób istotny wpływa na efektywność obliczeń równoległych. Przy stosunkowo niewielkiej ilości danych czasy związane z synchronizacją obliczeń bądź przesyłaniem danych z pamięci RAM komputera do pamięci karty graficznej mogą nawet spowolnić obliczenia. Zastosowanie modelu złożonego z 1024 stałych elementów brzegowych pozwoliło przekroczyć ową „granicę opłacalności" zastosowania obliczeń wielowątkowych. Zastosowanie wielowątkowej biblioteki SCALAPACK dedykowanej do obliczeń rozproszonych nie przyniosło w pełni zadowalających efektów – czas obliczeń jest o 3 s dłuższy niż w przypadku optymalizacji pamięci cache procesora uzyskanej poprzez zastosowanie bibliotek BLAS/LAPACK (54 s vs. 57 s - tabela 2). W zaproponowanym do testów modelu zwiększenie gęstości siatki nie wydaje się potrzebne. Bardziej uzasadnione mogłoby być zwiększenie ilości węzłów (i związanej z tym dokładności obliczeń) poprzez zastosowanie np. 3-wezłowych elementów brzegowych drugiego rzędu.

Rozważając dobór właściwego rozmiaru problemu należy uwzględnić fakt, że obliczenia realizowano na typowym komputerze PC a nie na rozbudowanym klastrze obliczeniowym.

Wyniki przedstawione w tabeli 2 potwierdzają konieczność wykorzystania bibliotek numerycznych. Generalnie w złożonych zagadnieniach biblioteki powinny być stosowane razem ze standardami OpenMP/MPI i CUDA/OpenCL dla GPU.

Prosta optymalizacja wykorzystania szybkiej pamięci podręcznej skutkuje ponad 10-cio krotnym przyspieszeniem obliczeń. Wykorzystanie procesora wektorowego karty graficznej przyniosło dalsze ok. dwukrotne przyspieszenie obliczeń. Biblioteka CULA stanowi dojrzały produkt i jej wydajność skutkowała najlepszymi wynikami skrócenia czasu obliczeń.

Ostatnim elementem, który nie znalazł się w powyższych rozważaniach, są biblioteki typy Intel MKL (Math Kernel Library) czy darmowa ATLAS (Automatically Tuned Linear Algebra Software). Podobnie nie poruszano kwestii instrukcji SIMD zaimplementowanych we współczesnych procesorach jak chociażby rozszerzenie SSE3 (Streaming SIMD Extensions 3). Techniki te są jednak częścią procesu działania i instalacji bibliotek typu LAPACK.

Generalnie użycie podstawowych bibliotek nie jest trudne a przynosi ogromny skok wydajności. Wersje GPU tych bibliotek nie są już tak proste w użyciu ale ciągle o wiele przyjaźniejsze niż całkowicie samodzielne programowanie GPGPU i optymalizacja związanych z nim parametrów podziału i przesyłania danych.

Literatura

[1] Labaki J., Ferreira L., Otávio S., Mesquita E.: Constant Boundary Elements on graphics hardware: a GPU-CPU complementary implementation, J. Braz. Soc. Mech. Sci. & Eng., vol. 33, 4/2011, pp. 475-482.
[2] Sanders J. Kandrot E.: CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley, 2011.
[3] Scarpino M.: OpenCl in Action: How to Accelerate Graphics and Computations;Manning Publications co., NY 2012.
[4] Sikora J.: Boundary Element Method for Impedance and Optical Tomography, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2007.
[5] Stpiczyński, P., Potiopa, J.: Solving a kind of boundary-value problem for ordinary differential equations using Fermi - the next generation CUDA computing architecture. J. Comp. Applied Mathematics 236, 3/2011, p. 384-393.
[6] AMD OpenCL Zone: developer.amd.com/resources/heterogeneous-computing/opencl-zone, 2013.
[7] BLAS Homepage: www.netlib.org/blas, 2013.
[8] CULA Programmer's guide: www.culatools.com/cula_dense_programmers_guide, 2013.
[9] LAPACK Homepage: www.netlib.org/lapack, 2013.
[10] KHRONOS OpenCL home page: www.khronos.org/opencl, 2013.
[11] NVIDIA CUDA Developer Zone: developer.nvidia.com, 2013.

Mgr inż. Krzysztof Król
e-mail: k.krol@pollub.pl

Ukończył studia na Wydziale Elektrotechniki i Informatyki Uniwersytetu Lubelskiego. W 2006 roku rozpoczął studia doktoranckie. Obecnie pracuje jako asystent na Politechnice Lubelskiej.

Dr inż. Maciej Pańczyk
e-mail: m.panczyk@pollub.pl

Ukończył studia na Wydziale Elektrycznym Politechniki Lubelskiej. Pracował w Katedrze Podstaw Elektrotechniki Politechniki Lubelskiej, następnie był pracownikiem Departamentu Informatyki banku BDK S.A. Od 2004 r. pracuje w Instytucie Informatyki Politechniki Lubelskiej. W pracy zawodowej koncentruje się na zagadnieniach pola elektromagnetycznego i programowania.

otrzymano/received: 27.06.2013 *przyjęto do druku/accepted: 04.03.2014*