Tomasz TOPA[1], Artur NOGA[1]

Silesian University of Technology (1)

doi:10.15199/48.2015.03.08

Implementacja metody momentów z wykorzystaniem kart graficznych i architektury CUDA

Streszczenie. *W artykule omówiono możliwości zastosowania kart graficznych do przyspieszania obliczeń numerycznych bazujących na metodzie momentów. Opisano algorytmy metody momentów implementowane w heterogenicznym środowisku CPU/GPU oraz przeprowadzono szczegółową analizę możliwych do uzyskania przyspieszeń dla różnych generacji architektury CUDA.*

Abstract. *The using of GPU to accelerate of the numerical simulations based on the Method of Moments (MoM) is presented in this paper. Implementation of the MoM in heterogeneous CPU/GPU platform and the measured speedups for the three generation of CUDA architecture is also demonstrated.* ***(Implementation of the Method of Momentsof on GPU with CUDA architecture)***

Słowa kluczowe: metoda momentów, procesory graficzne GPU, CUDA, akceleracja obliczeń
Keywords: Method of Moments (MoM), graphics processing unit (GPU), CUDA, GPU acceleration

Wprowadzenie

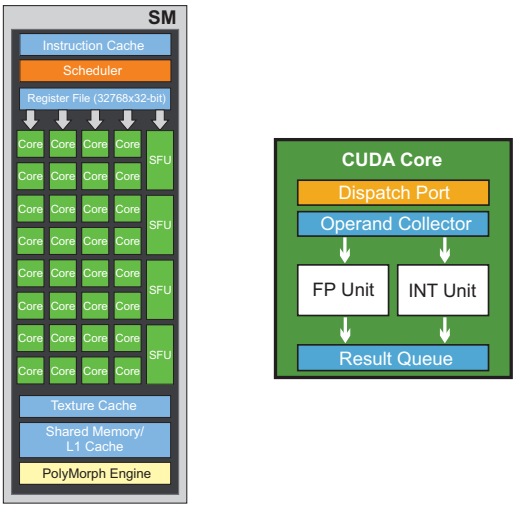
W chwili obecnej, w wielu dziedzinach działalności inżynierskiej na etapie projektowania coraz większe znaczenie odgrywają badania symulacyjne. Przydatność symulacji komputerowych jest szczególnie wyraźnie widoczna w rozwiązywaniu zadań inżynierii pola elektromagnetycznego (ocena poziomu zakłóceń, prognozowanie narażeń, badanie zjawiska promieniowania i rozpraszania fal elektromagnetycznych, itp.). Wynika to z faktu, że metody analityczne mają tutaj bardzo ograniczony zakres zastosowań chociażby z tego powodu, że umożliwiają badanie rozkładu pola tylko w układach wykazujących określony charakter symetrii i przy założeniu liniowości, jednorodności i anizotropii środowiska, w którym rozchodzi się pole elektromagnetyczne.

Wśród metod numerycznego modelowania pól elektromagnetycznych chyba najbardziej ugruntowaną pozycję mają metody pełnofalowe, tj. metoda momentów, metoda elementów skończonych, czy metoda FDTD [1]. Wspólną ułomnością tych metod są duże wymagania odnośnie do zasobów komputerowych (pamięć operacyjna, czas obliczeń), co przekłada się bezpośrednio na wielkość możliwych do rozwiązania zadań i dokładność otrzymywanych wyników końcowych. Bardzo często prowadzi to do sytuacji, w której rozwiązanie wielu praktycznych zadań inżynierii pola elektromagnetycznego jest albo niemożliwe, albo wymaga zastosowania kosztownych, wysokowydajnych i wyspecjalizowanych narzędzi i metod obliczeniowych.

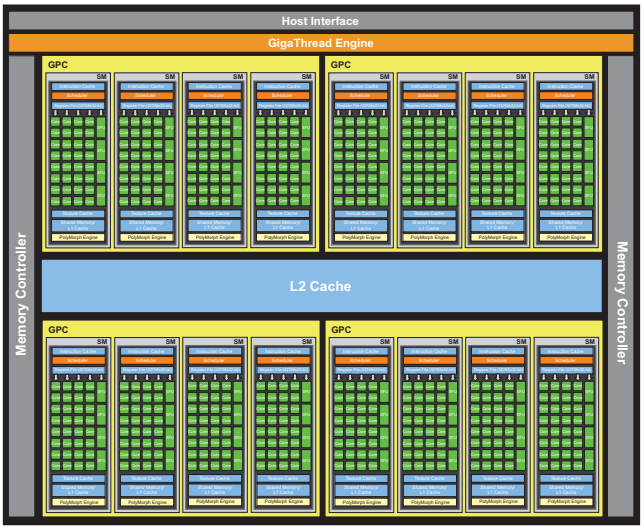
Barierę ograniczeń, o których mowa wyżej, można częściowo przełamać poprzez zastosowanie akceleracji sprzętowej z wykorzystaniem procesorów strumieniowych GPU (*ang.* Graphics Processing Units) i paradygmatu programowania masowo-równoległego [2]-[4]. Procesory GPU zaprojektowano do przetwarzania grafiki komputerowej, co na ogół sprowadza się do wykonywania pewnej liczby operacji arytmetycznych na macierzach i wektorach danych opisujących w sposób sformalizowany wyświetlany na ekranie monitora obraz. Co ważne, dane te muszą być przetwarzane w czasie rzeczywistym tak, aby zapewnić ciągłość wyświetlania obrazu. Wymusza to dużą wydajność obliczeniową współczesnych kart graficznych, a tym samym wchodzących w ich skład procesorów strumieniowych. Pomimo wciąż mocnego powiązania architektury układów GPU z ich pierwotnym przeznaczeniem, tj. grafiką komputerową, istnieje możliwość wykorzystania ich mocy obliczeniowej do innych zadań charakteryzujących się dużą intensywnością operacji arytmetycznych. Obliczenia inżynierskie wykonywane za pomocą kart graficznych określane są mianem obliczeń ogólnego przeznaczenia i oznaczane skrótem GPGPU (*ang.* General-Purpose computing on Graphics Processing Units) [2]. Obliczenia GPGPU wykonywane na współczesnych kartach graficznych umożliwiają znaczne skrócenie czasu analizy numerycznej w porównaniu do czasu obliczeń prowadzonych z wykorzystaniem jednostki centralnej CPU.

Schemat obliczeń numerycznych ogólnego przeznaczenia wymusza ścisłą współpracę jednostki centralnej CPU z układem GPU w ramach środowiska obliczeniowego o tzw. heterogenicznym modelu przetwarzania danych. Sekwencyjna część aplikacji wykonywana jest przez jednostkę CPU podczas, gdy złożone obliczeniowo fragmenty kodu przetwarzane są przez układ GPU pełniący rolę koprocesora arytmetycznego o dużej wydajności. Twórca aplikacji musi więc tak napisać lub zmodyfikować jej kod, aby wyodrębnić w nim fragmenty przeznaczone do realizacji na GPU. Proces ten nosi nazwę mapowania funkcji jądra i pociąga za sobą konieczność napisania kodu aplikacji w taki sposób, aby wyrazić równoległość realizowanych przez nią zadań. Wykorzystanie kart graficznych do obliczeń inżynierskich wiązało się początkowo z koniecznością bardzo skomplikowanego mapowania funkcji jądra, które należało przeprowadzić tak, aby funkcje te wyglądały i działały jak programy graficzne. Procedura programowania obliczeń inżynierskich na GPU uległa obecnie wyraźnemu uproszczeniu głównie, za sprawą opracowanej przez firmę NVIDIA masowo-równoległej architektury CUDA (*ang.* Compute Unified Device Architecture), co nie znaczy, że jest to zadanie łatwe [3]-[5]. Z myślą o bardziej wymagających użytkownikach (wielkie centra obliczeniowe, ośrodki naukowe) producenci kart graficznych wprowadzili również na rynek, obok standardowej, wysoko wyspecjalizowaną klasę procesorów graficznych GPU do zastosowań profesjonalnych. Procesory te wyposażono w dodatkowe jednostki arytmetyczne wspierające obliczenia zmiennoprzecinkowe podwójnej precyzji oraz dedykowany interfejs komunikacyjny. Mowa tutaj o akceleratorach graficznych Tesla firmy Nvidia, czy FireStream firmy ATI/AMD. O wzroście znaczenia obliczeń prowadzonych na kartach graficznych świadczy dodatkowo fakt, że większość z obecnie tworzonych kompilatorów języka C/C++, czy Fortran zostało wyposażonych w obsługę procesorów graficznych.

W literaturze można znaleźć wiele opracowań dokumentujących zastosowanie procesorów graficznych do wspomagania obliczeń naukowo-inżynierskich. Również w obrębie elektromagnetyzmu obliczeniowego od kilku lat prowadzone są w tym zakresie intensywne badania [6]-[13]. W przy-



Rys. 1. Budowa multiprocesora strumieniowego SM oraz procesora strumieniowego SP na przykładzie architektury Fermi (GF110)



Rys. 2. Schemat blokowy architektury Fermi (GF110) na przykładzie karty graficznej GeForce GTX 580

padku techniki antenowej istotną klasę analizowanych struktur stanowią obiekty złożone wyłącznie z bardzo dobrych przewodników. Wydaje się, że do badania i projektowania tego typu struktur najlepiej nadaje się metoda momentów [14].

W ramach tego artykułu opisano implementację metody momentów w heterogenicznym środowisku obliczeniowym CPU/GPU. Przeprowadzono dyskusję możliwych do uzyskania przyspieszeń dla różnych konfiguracji sprzętowych, przedstawiono ograniczenia każdej z wykorzystanych platform obliczeniowych oraz wskazano konfigurację optymalną. Rozważono trzy przypadki: pierwszy, w którym platformę obliczeniową wyposażono w jedną kartę graficzną GeForce GTX 275 (architektura GT200); drugi, w którym platformę obliczeniową wyposażono w jedną kartę graficzną GeForce GTX 580 (architektura Fermi) i trzeci, w którym platformę obliczeniową wyposażono w jedną kartę graficzną GeForce GTX 680 (architektura Kepler). Artykuł podzielono na cztery podstawowe rozdziały. W rozdziale pierwszym omówiono masowo-równoległą architekturę CUDA firmy NVIDIA II-giej, III-ciej i IV-tej generacji. Szczególną uwagę zwrócono na organizację oraz realizowany z poziomu software'u sposób zarządzania zasobami sprzętowymi. W kolejnych dwóch rozdziałach przedstawiono ideę metody momentów oraz opisano sposób jej implementacji w heterogenicznym środowisku obliczeniowym CPU/GPU. Ostatni rozdział zawiera szczegółowe rozważania na temat możliwych do uzyskania przyspieszeń w zależności od typu wykorzystywanej platformy obliczeniowej. Pokazano również, w jaki sposób dla zadanej platformy sprzętowej kształtuje się przebieg zysku na czasie analizy w funkcji liczby niewiadomych, tj. rozmiaru analizowanych problemów. We wszystkich opisywanych przypadkach testowane karty graficzne posłużyły jako koprocesory arytmetyczne wspomagające jednostkę centralną CPU (procesor Intel Core i7-3820). Algorytmy przewidziane do realizacji w heterogenicznym środowisku obliczeniowym CPU/GPU napisano w języku programowania Fortran i skompilowano za pomocą kompilatora firmy PGI ze wsparciem biblioteki CULA Tools [15], [16]. Otrzymane wyniki obliczeń porównano z wynikami uzyskanymi dla tzw. klasycznej implementacji metody momentów, tj. jej wersji uruchamianej wyłącznie na CPU (wersja jednordzeniowa). Kody źródłowe implementujące metodę momentów również i w tym przypadku napisano w języku programowania Fortran i skompilowano za pomocą kompilatora firmy Intel ze wspomaganiem matematycznych bibliotek MKL (*ang.* Math Kernel Library). Wszystkie obliczenia wykonano z wykorzystaniem zmiennych zespolonych podwójnej precyzji.

Architektura CUDA

Zgodnie z zaproponowaną przez firmę NVIDIA koncepcją, architekturę CUDA tworzy zespół wzajemnie ze sobą powiązanych jednostek wykonawczych o strukturze hierarchicznej. Za podstawową jednostkę wykonawczą zwykło się przyjmować procesor strumieniowy SP (*ang.* Streaming Processor) nazywany często również rdzeniem CUDA. Każdy procesor strumieniowy jest 32-bitową jednostką skalarną, która w jednym cyklu pracy może wykonać maksymalnie dwie operacje dodawania oraz jedną operację mnożenia. Aby móc wykonywać operacje na danych wektorowych (instrukcje typu SIMD), rdzenie CUDA łączy się w większe bloki funkcjonalne nazywane multiprocesorami strumieniowymi SM (*ang.* Streaming Multiprocessors) – patrz rys. 1. Multiprocesor strumieniowy wyposażony jest w co najmniej jedną, w pełni autonomiczną jednostkę DU (*ang.* Dispatch Unit) szeregującą zadania oraz posiada dostęp do pamięci globalnej *cache* drugiego poziomu (L2D *cache*). Dzięki temu operacje na danych wektorowych wykonywane są praktycznie bez opóźnień (multiprocesor strumieniowy zleca jedno zadanie do wykonania co najmniej kilku jednostkom SP, które wykonują je na różnych elementach wektora równolegle). Liczba jednostek SP tworzących jeden multiprocesor strumieniowy zależy od wersji architektury i zawiera się w przedziale od 8 do 192 (patrz Tabela 1). Oprócz rdzeni CUDA, w skład multiprocesora strumieniowego wchodzą także jednostki specjalne SFU (*ang.* Special Function Units). Zaprojektowano je z myślą o zwiększeniu wydajności obliczeniowej multiprocesorów strumieniowych i zoptymalizowano pod kątem wykonywania takich operacji matematycznych jak logarytmowanie, potęgowanie, pierwiastkowanie i obliczanie wartości funkcji trygonometrycznych. Aby podnieść efektywność obliczeniową na poziomie realizacji różnych zadań (instrukcje typu MIMD), multiprocesory strumieniowe łączy się bloki nazywane klastrami obliczeniowymi GPC (*ang.* Graphics Processing Clusters) – patrz rys. 2. W odróżnieniu od architektury II-giej generacji, architektury III-ciej i IV-tej generacji wyposażono dodatkowo w pamięć podręczną *cache* pierwszego poziomu o programowanym rozmiarze (L1D *cache*). Zadaniem

Tablica 1. Porównanie zasobów sprzętowych dostępnych w architekturze CUDA II-giej, III-ciej i IV-tej generacji

Zasoby	Architektura		
	GT200	GF110 (Fermi)	GK104 (Kepler)
Całkowita liczba procesorów strumieniowych SP	240	512	1536
Całkowita liczba multiprocesorów strumieniowych MP	30	16	8
Liczba klastrów obliczeniowych GPC	10	4	4
Całkowita liczba jednostek specjalnych SPU	60	64	256
Rozmiar pamięci L1D *cache*	–	16 kB lub 48 kB	16 kB lub 32 kB lub 48 kB
Rozmiar pamięci L2D *cache*	256 kB	768 kB	512 kB
Rozmiar pamięci globalnej	896 MB	1536 MB	4096 MB

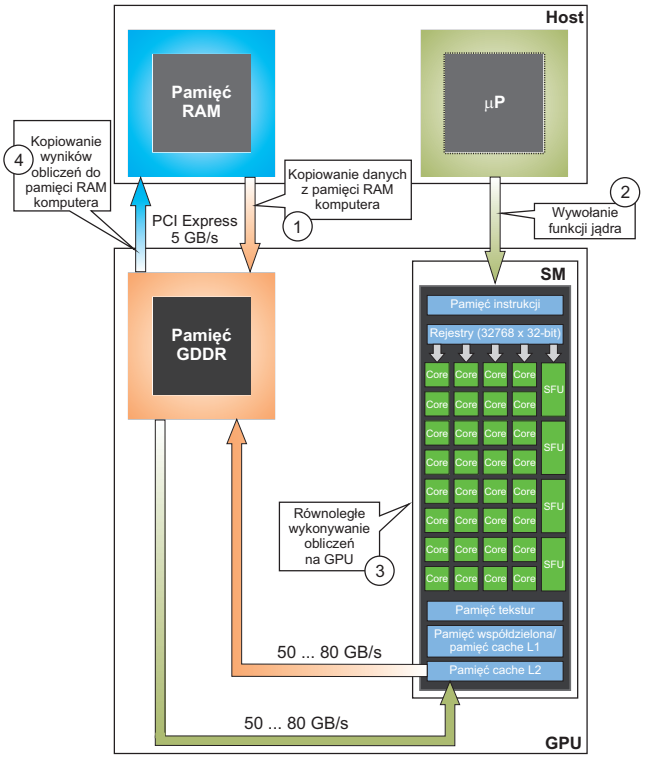
pamięci L1D *cache* jest, podobnie zresztą jak w przypadku zwykłej jednostki centralnej CPU, zmniejszenie obciążenia wewnętrznej magistrali danych oraz skrócenie czasu dostępu procesora/procesorów strumieniowych do danych zapisanych w pamięci globalnej karty. Warto w tym miejscu jednak podkreślić, że możliwość programowania rozmiaru pamięci *cache* pierwszego poziomu została przez producenta kart graficznych, firmę NVIDIA w znacznym stopniu ograniczona. Na dzień dzisiejszy jej rozmiar można ustawić tylko na 16, 32 lub 48 kB. Jak pokazuje doświadczenie programistyczne konfiguracja pierwsza, w której rozmiar pamięci L1D *cache* ustawi się na 16 kB, sprawdza się najlepiej w przypadku aplikacji stricte graficznych, natomiast konfiguracja ostatnia – w przypadku obliczeń GPGPU (pamięć L1D *cache* jest wydzielona z obszaru pamięci współdzielonej o łącznym rozmiarze 64 kB; zwiększenie jej rozmiaru odbywa się więc zawsze kosztem rozmiaru pamięci współdzielonej, co na ogół prowadzi do zmniejszenia wydajności przetwarzania grafiki komputerowej).

Na rys. 3 pokazano diagram przepływu danych w heterogenicznym środowisku obliczeniowym CPU/GPU. Zgodnie z nim w kroku pierwszym dane wejściowe kopiowane są z pamięci komputera do pamięci karty graficznej. W kroku drugim dochodzi, z poziomu jednostki centralnej CPU, do inicjalizacji obliczeń na karcie graficznej, czyli do tzw. wywołania funkcji jądra obliczeniowego. Wywołanie funkcji jądra rozpoczyna obliczenia równoległe na każdym procesorze strumieniowym SP zgodnie z programem przesłanym przez jednostkę centralną CPU. Po zakończeniu obliczeń na karcie graficznej otrzymane wyniki kopiowane są z pamięci globalnej karty graficznej do pamięci RAM komputera.

Metoda momentów

Podejście do zagadnienia tytułowego wygodnie objaśnić zaczynając od przedstawienia wspomnianej już we Wprowadzeniu idei tzw. metody momentów, czyli sposobu konstruowania przybliżonych, numerycznych rozwiązań zagadnień promieniowania lub rozpraszania fal elektromagnetycznych. Weźmy zatem pod uwagę bryłę materiału o nieskończenie dużej konduktywności, umieszczoną w jednorodnym, bezstratnym, izotropowym ośrodku o parametrach ε i μ. Przyjmijmy, że bryła jest umieszczona w polu elektromagnetycznym zmieniającym się w czasie harmonicznie z częstotliwością kątową ω.

Problem, który sformułujemy tutaj analitycznie, a następnie rozwiążemy numerycznie, sprowadza się do odpowiedzi



Rys. 3. Organizacja przepływu danych w heterogenicznym środowisku obliczeniowym CPU/GPU

na pytanie: jaki jest rozkład prądu indukowanego na powierzchni S bryły przez zewnętrzne pole elektromagnetyczne o znanym rozkładzie? Do sformułowania analitycznego nietrudno dojść rozumując jak następuje. Prąd indukowany na powierzchni bryły przez pole „oświetlające" jest źródłem wtórnego pola elektromagnetycznego. Pole wtórne sumuje się z polem pierwotnym, a wynikiem tego sumowania musi być pole ze znikającą składową lokalnie styczną do powierzchni bryły (założono bowiem nieskończoną konduktywność materiału, z którego wykonano bryłę). Inaczej mówiąc, pole pierwotne indukuje na powierzchni bryły prąd o takim rozkładzie, który gwarantuje spełnienie warunku znikania składowej stycznej pola wypadkowego tuż przy powierzchni bryły. Warunek ten można zapisać w postaci

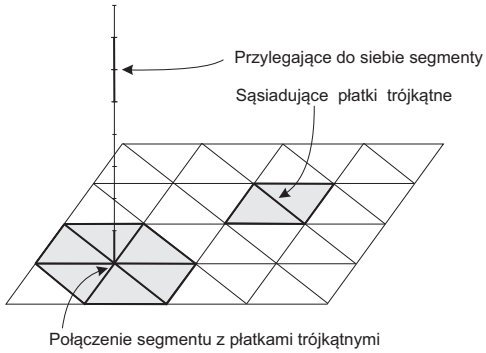
$$(j\omega\mathbf{A} + \nabla\Phi) \cdot \mathbf{1}_l = \mathbf{E}^i \cdot \mathbf{1}_l, \tag{1}$$

gdzie $\mathbf{E}^i$ jest wektorem natężenia zewnętrznego, przyłożonego pola elektrycznego, $\mathbf{A}$ i Φ, oznaczają, odpowiednio, magnetyczny potencjał wektorowy i elektryczny potencjał skalarny pola wtórnego, pochodzącego od prądu i ładunku wzbudzonego w przewodniku, a $\mathbf{1}_l$ jest wektorem jednostkowym lokalnie stycznym do powierzchni bryły. Warunek (1) jest w istocie zapisanym w niejawnej i zwartej postaci równaniem dla funkcji opisującej rozkład prądu indukowanego na powierzchni bryły. Nietrudno to stwierdzić, gdy weźmie się pod uwagę, że potencjały elektrodynamiczne $\mathbf{A}$ i Φ są związane z rozkładami, odpowiednio, prądu $\mathbf{J}$ i q ładunku na powierzchni bryły, tzn.

$$\mathbf{A} = \frac{\mu}{4\pi}\int_S \mathbf{J}(\mathbf{r}')\frac{e^{-jkR}}{R}dS', \tag{2}$$

$$\Phi = \frac{1}{j4\pi\omega\varepsilon}\int_S \nabla'\cdot\mathbf{J}(\mathbf{r}')\frac{e^{-jkR}}{R}dS', \tag{3}$$

w którym k oznacza liczbę falową ośrodka, natomiast $R =$



Rys. 4. Elementarne struktury źródłowe występujące w modelu

$|\mathbf{r} - \mathbf{r}'|$ jest odległością między punktem obserwacji i punktem źródłowym, przy czym oba te punkty są umiejscowione na powierzchni bryły, a ich położenie w globalnym układzie współrzędnych określają wektory miejsca $\mathbf{r}$ i $\mathbf{r}'$, odpowiednio. Całkowanie w równaniach (2) i (3) przebiega po współrzędnych punktu źródłowego na powierzchni bryły.

Mając na uwadze związki (2) i (3) dochodzimy do wniosku, że równanie zapisane w zwartej postaci jako (1) rzeczywiście jest złożonym równaniem różniczkowo-całkowym, w którym wielkością znaną jest funkcja po prawej stronie opisująca pobudzenie, natomiast niewiadomą jest funkcja $\mathbf{J}$ opisująca rozkład prądu wzbudzonego na powierzchni bryły. Równanie to jest nazywane równaniem dwupotencjałowym, ponieważ występują w nim explicite potencjały wektorowy i skalarny. Ścisłe, analityczne rozwiązanie równania dwupotencjałowego nie jest znane. Przybliżone rozwiązanie numeryczne tego równania jest najczęściej konstruowane wspomnianą wcześniej metodą momentów. U podstaw metody leży dyskretyzacja analizowanej struktury, tzn. podział struktury na małe elektrycznie elementy; w omawianym tutaj przypadku powierzchnia przewodzącej bryły jest przybliżana zbiorem małych, płaskich, trójkątnych płatków powierzchniowych, natomiast przewody są „prostowane", tzn. aproksymowane zbiorem krótkich cylindrycznych segmentów prostoliniowych. Warunek elektrycznie małych wymiarów płatka i segmentu oznacza, że długość segmentu i długość każdej krawędzi płatka musi być znacznie mniejsza od długości fali roboczej. W zbudowanym w ten sposób modelu dyskretnym wyróżnia się trzy rodzaje elementarnych struktur źródłowych (patrz rys. 4) [17], [18]:

- dwa płatki trójkątne przylegające do siebie wzdłuż wspólnej krawędzi,
- dwa segmenty cylindryczne mające wspólny punkt,
- otoczenie węzła, w którym przewód łączy się z bryłą, składające się z segmentu liniowego mającego koniec w węźle i wszystkich płatków powierzchniowych, które mają jeden z wierzchołków w węźle.

Z każdą z tych struktur skojarzona jest odpowiednia funkcja bazowa skonstruowana w taki sposób, że zapewnia ona ciągłość prądu w miejscu połączenia elementów struktury źródłowej.

Przybliżony rozkład gęstości prądu $\mathbf{J}$ na powierzchni brył wchodzących w skład analizowanej struktury postulujemy w postaci kombinacji liniowej

$$\mathbf{J}(\mathbf{r}) \approx \sum_{n=1}^{N_B} I_n^B \mathbf{\Lambda}_n^B(\mathbf{r}) + \sum_{n=1}^{N_J} I_n^J \mathbf{\Lambda}_n^J(\mathbf{r}) \quad \mathbf{r} \in S^B \tag{4}$$

funkcji bazowych z nieznanymi współczynnikami I_n^B i I_n^J rozpiętymi na płatkach. Indeksy górne B i J pochodzą od angielskich słów *body* i *junction*, natomiast granice sumowania N_B i N_J odnoszą się do liczby funkcji bazowych rozpiętych, odpowiednio, na płatkach i węzłach. Podobnie dla rozkładu prądu wzdłuż przewodów przyjmujemy rozwinięcie o postaci

$$\mathbf{J}(\mathbf{r}) \approx \sum_{n=1}^{N_W} I_n^W \mathbf{\Lambda}_n^W(\mathbf{r}) + \sum_{n=1}^{N_J} I_n^J \mathbf{\Lambda}_n^J(\mathbf{r}) \quad \mathbf{r} \in S^W, \tag{5}$$

gdzie indeks W pochodzi od angielskiego słowa *wire*, a N_W oznacza liczbę funkcji bazowych rozpiętych na przewodach. Równania dla poszukiwanych współczynników rozwinięcia otrzymuje się w wyniku tzw. testowania rozwiązywanego równania. Ogólnie, testowanie polega na przyrównaniu średnich ważonych obu stron rozwiązywanego równania, wziętych po kolei dla każdej funkcji z N-elementowego zbioru przyjętych funkcji wagowych (jako funkcje wagowe przyjęto funkcje identyczne z wagowymi – schemat Galerkina). W ten sposób dochodzi się do układu N liniowych równań algebraicznych dla poszukiwanych współczynników rozwinięcia (4)-(5). W notacji macierzowej układ ten można zapisać jako

$$\mathbf{ZI} = \mathbf{U}, \tag{6}$$

gdzie $\mathbf{Z}$ jest kwadratową macierzą impedancyjną stopnia N opisującą analizowany obiekt (ściślej – ujmującą wzajemne sprzężenia elektromagnetyczne między elementarnymi strukturami źródłowymi, na które zdekomponowano obiekt), $\mathbf{U}$ jest kolumnowym wektorem reprezentującym pobudzenie obiektu, natomiast $\mathbf{I}$ oznacza wektor poszukiwanych współczynników rozwinięcia. Macierz impedancyjna może być zapisana jak następuje

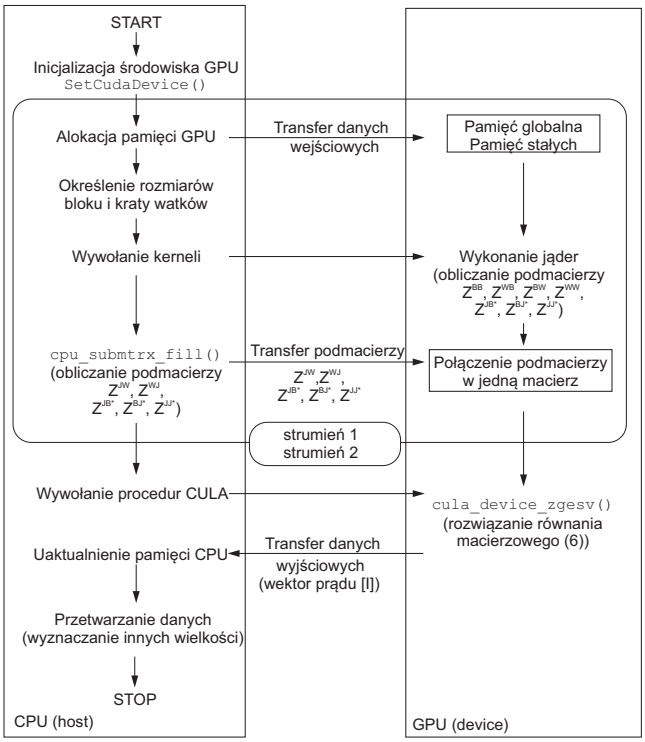
$$\mathbf{Z} = \begin{bmatrix} \mathbf{Z}^{\mathrm{BB}} & \mathbf{Z}^{\mathrm{BW}} & \mathbf{Z}^{\mathrm{BJ}} \\ \mathbf{Z}^{\mathrm{WB}} & \mathbf{Z}^{\mathrm{WW}} & \mathbf{Z}^{\mathrm{WJ}} \\ \mathbf{Z}^{\mathrm{JB}} & \mathbf{Z}^{\mathrm{JW}} & \mathbf{Z}^{\mathrm{JJ}} \end{bmatrix}. \tag{7}$$

Podmacierze $\mathbf{Z}^{\gamma\beta}$ (gdzie $\gamma, \beta = B, W, J$) ujmują wzajemne oddziaływania pomiędzy strukturami źródłowymi, o których mowa wyżej, a ich wyznaczenie wymaga wielokrotnego obliczania całek (2) i (3). Rozwiązanie równania (6) jest równoznaczne z wyznaczeniem wartości współczynników rozwinięcia I_n. Podstawiając otrzymane w ten sposób współczynniki do równań (4) i (5) otrzymujemy przybliżony rozkład prądu na powierzchni analizowanego obiektu, na podstawie którego można wyznaczyć pole wtórne i wszystkie inne interesujące wielkości.

Implementacja metody momentów na GPU

Charakter obliczeń numerycznych wykonywanych za pomocą metody momentów pozwala wyodrębnić z jej implementacji komputerowej dwa główne fragmenty kodu cechujące się wysokim poziomem intensywności obliczeniowej. Mowa tutaj o wypełnianiu macierzy impedancyjnej (7) oraz rozwiązaniu równania macierzowego (6). Rozsądnym wydaje się więc podejście, w którym w ramach procesu tworzenia aplikacji heterogenicznej CPU/GPU, kody implementujące wypełnianie macierzy impedancyjnej $\mathbf{Z}$ oraz umożliwiające rozwiązywanie równania (6) przystosowane zostaną do realizacji na kartach graficznych [12], [13].

Na rys. 5 pokazano schemat blokowy algorytmu komputerowego implementującego metodę momentów w heterogenicznym środowisku obliczeniowym CPU/GPU. Cechą charakterystyczną przedstawionego na rys. 5 sposobu implementacji jest wykorzystanie tzw. techniki nakładających się zdarzeń (*ang.* overlapping), która umożliwia jednoczesne

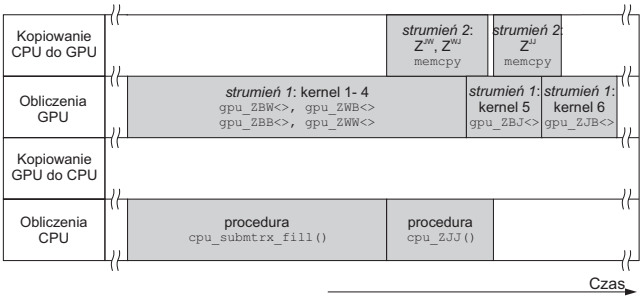


$Z^{\alpha\beta *}$- obliczenia wykonywane na CPU lub GPU w zależności od platformy sprzętowej

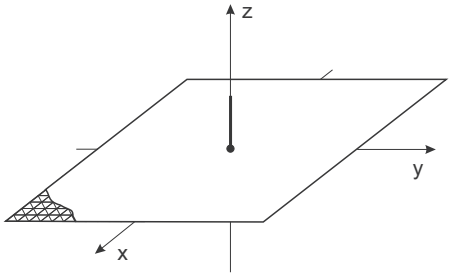
Rys. 5. Schemat blokowy algorytmu implementującego metodę momentów w heterogenicznym środowisku obliczeniowym CPU/GPU

przesyłanie danych między pamięcią karty graficznej i pamięcią RAM komputera oraz wykonywanie obliczeń na karcie graficznej. Zastosowanie tej techniki wymaga zdefiniowania co najmniej dwóch strumieni zdarzeń kontrolujących, w tym przypadku wspólnie, proces wykonywania funkcji jądra oraz operacji kopiowania danych [3]. Jak pokazano na rys. 5 proces obliczeń masowo-równoległych rozpoczyna wstępna inicjalizacja sprzętu, w ramach której program zarządzający alokuje pamięć na karcie graficznej dla macierzy impedancyjnej $\mathbf{Z}$ i danych wejściowych oraz określa sposób wykorzystania dostępnych zasobów (podział ma kraty i bloki wątków).

Opisany w rozdziale 3 sposób konstrukcji macierzy impedancyjnej sprawia, że proces ten można podzielić na kilka zadań cząstkowych realizowanych wspólnie przez kartę graficzną i jednostkę centralną CPU. Dla analizowanej klasy struktur zaproponowano rozwiązanie, w którym podmacierze $\mathbf{Z}^{WW}$, $\mathbf{Z}^{WB}$, $\mathbf{Z}^{BW}$ oraz $\mathbf{Z}^{BB}$ wyznaczane są zawsze przez procesory strumieniowe karty graficznej, niezależnie od konfiguracji sprzętowej, natomiast podmacierze $\mathbf{Z}^{JW}$ i $\mathbf{Z}^{WJ}$ przez jednostkę centralną CPU. Sposób wyznaczania pozostałych trzech podmacierzy uwarunkowany jest przez klasę wykorzystywanej platformy sprzętowej i może przebiegać według jednego z następujących trzech scenariuszy działania: 1) podmacierz $\mathbf{Z}^{JJ}$ wyznaczana jest przez jednostkę centralną CPU natomiast podmacierze $\mathbf{Z}^{BJ}$ i $\mathbf{Z}^{JB}$ przez procesory strumieniowe karty graficznej, 2) podmacierze $\mathbf{Z}^{JB}$ i $\mathbf{Z}^{BJ}$ wyznaczane są przez jednostkę centralną CPU podczas, gdy podmacierz $\mathbf{Z}^{JJ}$ przez procesory strumieniowe karty graficznej, 3) wszystkie trzy podmacierze $\mathbf{Z}^{JJ}$, $\mathbf{Z}^{BJ}$ i $\mathbf{Z}^{JB}$ są wyznaczane przez jednostkę centralną CPU. Decyzję o realizacji wybranego scenariusza działania podejmuje program zarządzający wywoływany z poziomu jednostki centralnej CPU. Porównuje on czas wypełniania podmacierzy $\mathbf{Z}^{JW}$ i $\mathbf{Z}^{WJ}$ (realizacja na CPU) z czasem wypełniania podmacierzy $\mathbf{Z}^{WW}$ i $\mathbf{Z}^{BB}$ (realizacja na GPU).



Rys. 6. Przykładowy podział na zadania cząstkowe realizowane w ramach procesu wypełniania macierzy impedancyjnej wraz z ich zależnością czasową



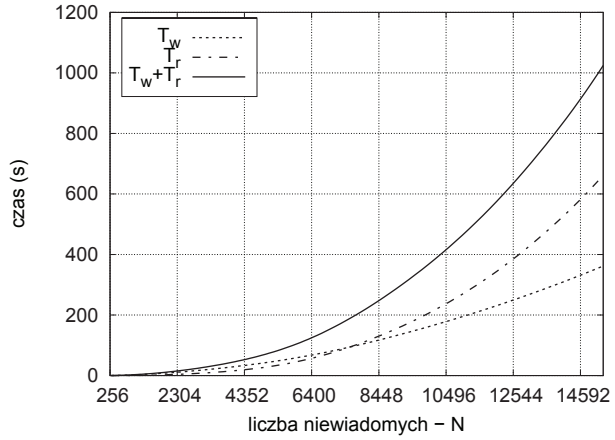
Rys. 7. Struktura testowa złożona z unipola umieszczonego na środku płaskiej, kwadratowej platformy

Jeżeli $T_{GPU} > 1.25 \cdot T_{CPU}$ to realizowany jest scenariusz trzeci, jeżeli $T_{GPU} > 1.2 \cdot T_{CPU}$ to realizowany jest scenariusz drugi natomiast, gdy $T_{GPU} > 1.1 \cdot T_{CPU}$ to realizowany jest scenariusz pierwszy. Rozwiązanie to zastosowano w celu jak najlepszego zbilansowania czasów pracy jednostek CPU i GPU, a tym samym zwiększenia wydajności obliczeniowej procesu wypełniania macierzy impedancyjnej. Działanie algorytmu odpowiedzialnego za wypełnienie macierzy impedancyjnej, przewidzianego do realizacji w heterogenicznym środowisku obliczeniowym CPU/GPU (scenariusz pierwszy), pokazano schematycznie na rys. 6. Do rozwiązania równania macierzowego (6) wykorzystano procedurę `cula_device_zgesv()` z biblioteki numerycznej CULA Tools [16]. Składa się ona z dwóch podprodecur wywoływanych sekwencyjnie, jednej odpowiedzialnej za faktoryzację macierzy impedancyjnej i drugiej realizującej podstawianie wsteczne. W wyniku działania procedury `cula_device_zgesv()` wyznaczony zostaje wektor współczynników I_n, a następnie na jego podstawie, rozkład prądu $\mathbf{J}$ na powierzchni analizowanej struktury.

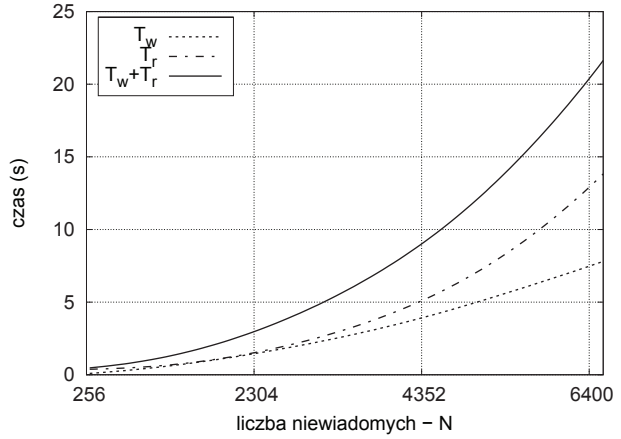
Obliczenia numeryczne

Do testowania opisanych algorytmów wykorzystano strukturę pokazaną na rys. 7. Składa się ona z unipola umieszczonego na środku płaskiej, doskonale przewodzącej platformy. Strukturę zasilono za pomocą idealnego, punktowego generatora napięciowego wpiętego w miejscu połączenia unipola z platformą. Do celów analizy numerycznej platformę zamodelowano za pomocą trójkątnych płatków powierzchniowych, natomiast unipol podzielono na segmenty liniowe. Zaletą wykorzystanej struktury jest to, że pozwala ona na stosunkowo łatwą zmianę stopnia dyskretyzacji, co w konsekwencji znacznie przyspiesza proces testowania zaimplementowanych algorytmów.

Jak już wcześniej wspomniano, działanie zaimplementowanego w ramach metody momentów algorytmu obliczeniowego wymaga wypełnienia macierzy impedancyjnej $\mathbf{Z}$, wyznaczenia wektora pobudzenia $\mathbf{U}$ oraz rozwiązania układu równań (6). W praktyce, o czasie analizy numerycznej prowadzonej za pomocą metody momen-

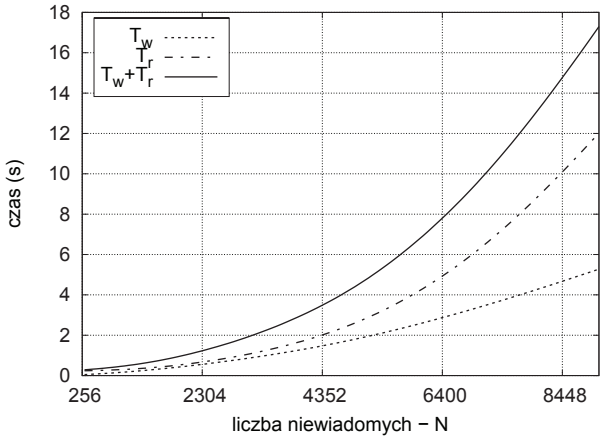


Rys. 8. Zależność czasu wypełniania macierzy impedancyjnej T_w, czasu rozwiązywania układu równań T_r oraz czasu całkowitego $T_w + T_r$ od liczby niewiadomych N dla implementacji na CPU

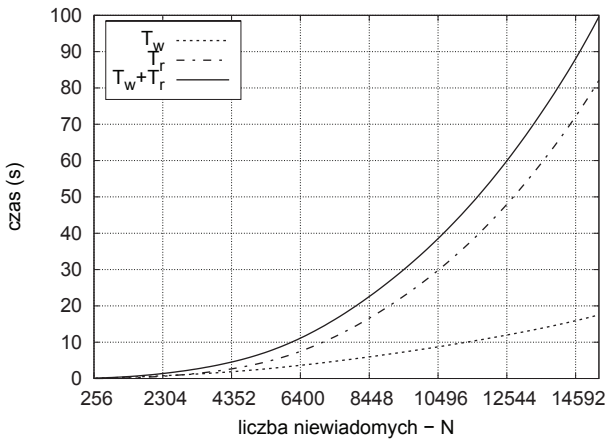


Rys. 9. Zależność czasu wypełniania macierzy impedancyjnej T_w, czasu rozwiązywania układu równań T_r oraz czasu całkowitego $T_w + T_r$ od liczby niewiadomych N dla heterogenicznej platformy obliczeniowej CPU/GPU z kartą graficzną GTX 275

tów decyduje czas T_w wypełniania macierzy impedancyjnej $\mathbf{Z}$ oraz czas T_r rozwiązywania równania macierzowego (w porównaniu do czasów T_w i T_r czas wypełniania wektora $\mathbf{U}$ jest niewielki i można go pominąć). Na rys. 8 pokazano zależność czasu wypełniania macierzy impedancyjnej T_w, czasu rozwiązywania układu równań T_r oraz czasu całkowitego $T_w + T_r$ od liczby niewiadomych N dla implementacji na CPU. Jak łatwo zauważyć, dla niewielkiej liczby niewiadomych ($N < 7000$) dominującą rolę w całkowitym czasie obliczeń odgrywa czas wypełniania T_w. Dla $N > 7000$ mamy do czynienia z sytuacją odwrotną tzn. taką, w której całkowity czas analizy zdominowany jest przez czas T_r rozwiązywania układu równań (6). W przypadku analizy numerycznej prowadzonej w heterogenicznym środowisku obliczeniowym CPU/GPU z kartami II-giej, III-ciej i IV-tej generacji trudno wskazać punkt, w którym $T_w > T_r$ (patrz rys. 9-11). Oznacza to, że niezależnie od liczby niewiadomych N, całkowity czas analizy numerycznej pozostanie zawsze w mniejszym lub większym stopniu zdominowany przez czas T_r rozwiązywania równania macierzowego (6). Warto w tym miejscu podkreślić, że procedurę testowania opisanych algorytmów ograniczono do przypadku, w którym wszystkie dane wejściowe i dane wynikowe mieszczą się w pamięci globalnej karty graficznej (algorytmy typu *in core*). W sytuacji, w której rozmiar dostępnej pamięci



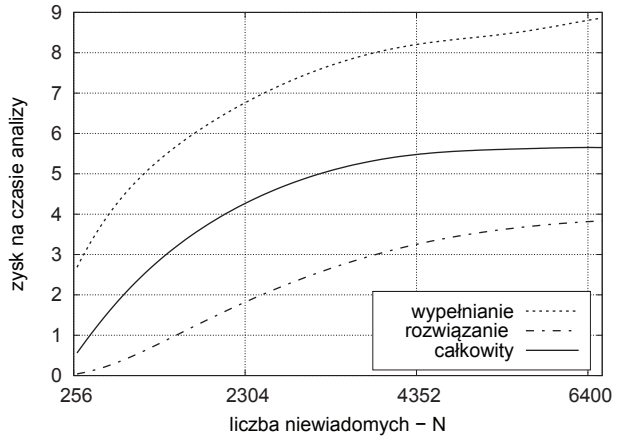
Rys. 10. Zależność czasu wypełniania macierzy impedancyjnej T_w, czasu rozwiązywania układu równań T_r oraz czasu całkowitego $T_w + T_r$ od liczby niewiadomych N dla heterogenicznej platformy obliczeniowej CPU/GPU z kartą graficzną GTX 580



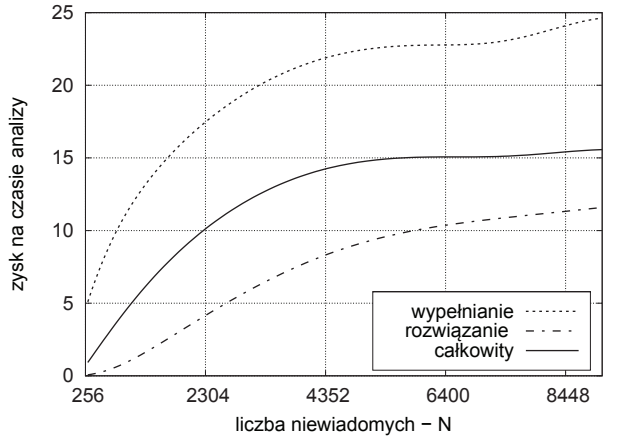
Rys. 11. Zależność czasu wypełniania macierzy impedancyjnej T_w, czasu rozwiązywania układu równań T_r oraz czasu całkowitego $T_w + T_r$ od liczby niewiadomych N dla heterogenicznej platformy obliczeniowej CPU/GPU z kartą graficzną GTX 680

globalnej systematycznie rośnie (patrz Tabela 1), daje to możliwość rozwiązywania na karcie graficznej coraz większych problemów numerycznych (patrz rys. 9-11)

Na rys. 12-14 pokazano przebieg zysku osiąganego na czasie analizy w funkcji liczby niewiadomych N dla różnych konfiguracji platform obliczeniowych. Już wstępna analiza tych wykresów pozwala stwierdzić, że największy zysk na czasie analizy ($15x$) można osiągnąć dzięki wykorzystaniu karty graficznej GTX 580 natomiast najmniejszy ($6x$), dzięki wykorzystaniu karty graficznej GTX 275. Wykorzystanie karty graficznej GTX 680, pomimo zdecydowanie największej liczby dostępnych zasobów sprzętowych, pozwala na 11-krotne skrócenie czasu analizy. W porównaniu do zysku na czasie analizy, który można osiągnąć dzięki wykorzystaniu karty graficznej GTX 580 jest to wynik gorszy o około 30 %. Wynik ten, choć początkowo trochę zaskakujący, jest bezpośrednią konsekwencją zastosowanego w architekturze IV-tej generacji sposobu organizacji zasobów sprzętowych. Sześciokrotne zwiększenie (w porównaniu do architektury III-ciej generacji) liczby jednostek SP wchodzących w skład jednego multiprocesora strumieniowego, przy jednoczesnym zmniejszeniu całkowitej liczby dostępnych multiprocesorów strumieniowych podnosi wprawdzie w ostatecznym rozrachunku wydajność przetwarzania grafiki komputerowej,



Rys. 12. Przebieg zysku na czasie analizy w funkcji liczby niewiadomych dla heterogenicznej platformy obliczeniowej CPU/GPU z kartą graficzną GTX 275



Rys. 13. Przebieg zysku na czasie analizy w funkcji liczby niewiadomych dla heterogenicznej platformy obliczeniowej CPU/GPU z kartą graficzną GTX 580

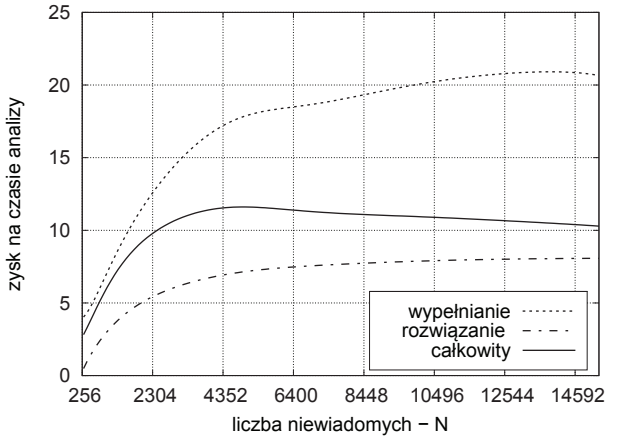
natomiast nie sprawdza się w przypadku realizacji algorytmów obliczeniowych metody momentów. Widoczny na rys. 14 spadek zysku na czasie analizy w funkcji liczby niewiadomych N nie jest niczym nadzwyczajnym i odzwierciedla ogólną tendencję obserwowaną w systemach równoległego przetwarzania danych, w których potencjalne zyski na czasie analizy mogą zostać całkowicie lub częściowo zniwelowane przez narzut czasowy związany z wymianą danych użytecznych.

Podsumowanie

W artykule omówiono możliwości zastosowania kart graficznych o architekturze CUDA do przyspieszania obliczeń numerycznych związanych z metodą momentów. Pokazano, że w zależności od wykorzystywanej platformy obliczeniowej całkowity zysk na czasie analizy może zmieniać się w zakresie od kilku do kilkunastu razy. Wskazano optymalną pod względem wydajności obliczeniowej konfigurację sprzętową oraz zaproponowano metodę efektywnego wypełniania macierzy impedancyjnej.

LITERATURA

[1] Sadiku M. N. O.: Numerical Techniques in Electromagnetics, CRC Press, 2001.
[2] Owens J. D., Houston M., Luebke D., Green S., Stone J. E., Phillips J. C.: GPU computing, Proceedings of the IEEE, 96(5), pp. 879-–899, 2008.



Rys. 14. Przebieg zysku na czasie analizy w funkcji liczby niewiadomych dla heterogenicznej platformy obliczeniowej CPU/GPU z kartą graficzną GTX 680

[3] Sanders J., Kandrot E.: CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley Press, 2010.
[4] Kirk D. B., Hwu W-M. W.: Programming Massively Parallel Processors: A Hands on Approach, Elsevier, 2010.
[5] NVIDIA Corporation, CUDA Programming Guide, Santa Clara, 2011.
[6] Krakiwsky S. E., Turner L. E., Okoniewski M.: Acceleration of finite difference time-domain (FDTD) using graphics processor units (GPU), IEEE MTT-S International Microwave Symposium Digest, 2004, pp. 1033-–1036.
[7] Lee K. H., Ahmed I., Goh R. S. M., Khoo E. H., Li E. P., Hung T. G. G.: Implementation of the FDTD method based on Lorentz-Drude dispersive model on GPU for plasmonics applications, Progress In Electromagnetics Research, 116, pp. 441-–456, 2011.
[8] Lezar E., Davidson D.: GPU-based LU decomposition for large method of moments problems, Electronic Letters, 46(17), pp. 1194-–1196, 2010.
[9] Peng S., Nie Z.: Acceleration of the method of moments calculations by using graphics processing units, IEEE Trans. Antennas Propag., 56(7), pp. 2130-–2133, 2008.
[10] Lezar E., Davidson D.: GPU-accelerated method of moments by example: monostatic scattering, IEEE Antennas and Propagation Magazine, 52(6), pp. 120-–135, 2010.
[11] Vilacha C., Otero A. F., Moreira J. C., Miguez E.: Accelerated EMF Evaluation Using a SIMD Algorithm, Przegląd Elektrotechniczny, nr 3a/2013.
[12] Topa T., Karwowski A., Noga A.: Using GPU with CUDA to accelerate MoM-based electromagnetic simulation of wire-grid models, IEEE Antennas Wireless Propag. Lett., 10, pp. 342—345, 2011.
[13] Topa T., Noga A., Karwowski A.: Adapting MoM with RWG Basis Functions to GPU Technology Using CUDA, IEEE Antennas Wireless Propag. Lett., 10, pp. 480-–483, 2011.
[14] Harrington R. F.: Field Computation by Moment Methods, New York, USA: Macmillan, 1968.
[15] PGI Corporation, CUDA Fortran Programming Guide and Reference, Portland, OR, 2010.
[16] EMPhotonics, CULA Tools – GPU Accelerated LAPACK, [strona www] http://www.culatools.com/.
[17] Hwu S. U. , Wilton D. R.: Electromagnetic scattering and radiation by arbitrary configurations of conducting bodies and wires, Dept. Elect. Eng., Applied Electromagn. Lab., Univ. Houston, Tech. Rep. 87-17, 1987.
[18] Rao S. M., Wilton D. R., Glisson A. W.: Electromagnetic scattering by surfaces of arbitrary shape, IEEE Trans. Antennas Propag., 30(3), pp. 409–418, May 1982.

***Autorzy**: dr inż. Tomasz Topa, dr inż. Artur Noga, Instytut Elektroniki, Wydział Automatyki Elektroniki i Informatyki, Politechnika Śląska, ul. Akademicka 16, 44-100 Gliwice, email: ttopa@polsl.pl, anoga@polsl.pl*