Tomasz TOPA
Politechnika Śląska, Wydział Automatyki, Elektroniki i Informatyki
ORCID: 0000-0001-9870-5137

doi:10.15199/48.2023.08.03

Akceleracja sprzętowa metody momentów za pomocą układów FPGA

Streszczenie. *W artykule omówiono możliwości wykorzystania układów logiki programowalnej (FPGA) do przyspieszania obliczeń numerycznych związanych z metodą momentów (MoM), a konkretnie procesu wypełniania macierzy impedancyjnej. Wypełnianie macierzy impedancyjnej podzielono na osiem odrębnych zadań cząstkowych realizowanych w ramach jednego jądra obliczeniowego CPU/FPGA. Otrzymane wyniki symulacji komputerowej pokazują, że zastosowane podejście hybrydowe pozwala prawie trzykrotnie (2.96×) skrócić czas analizy numerycznej obiektów przewodzących w porównaniu do analizy prowadzonej przy użyciu konwencjonalnej (jednordzenowej) implementacji referencyjnej.*

Abstract. *In this paper, an FPGA-based acceleration of the matrix assembly phase of the method of moments (MoM) is presented. To take advantages of the given hardware resources, the assembly phase of the MoM is divided into eight different sub-tasks which are performed concurrently during the runtime. Numerical results show that the proposed FPGA-based approach is about triple as fast as the reference single-core CPU implementation.* ***(Hardware acceleration of the Method of Moments using FPGA)***

Słowa kluczowe: akceleracja sprzętowa obliczeń numerycznych, metoda momentów, układy logiki programowalnej
Keywords: hardware acceleration, Method of Moments (MoM), Field Programmable Gate Arrays (FPGA)

Wprowadzenie

Metody pełnofalowe takie jak metoda momentów (MoM), metoda elementów skończonych (FEM), czy metoda różnic skończonych (FDTD) na stałe weszły już do kanonu metod numerycznych powszechnie wykorzystywanych/wspierających projektowanie, analizę i modelowanie nowoczesnych systemów i urządzeń radiokomunikacyjnych. Dobrze znaną, fundamentalną ułomnością tych metod są ich duże wymagania odnośnie do zasobów komputerowych (pamięć operacyjna, czas pracy CPU). Ujawniają się one bardzo dotkliwie, gdy stajemy przed koniecznością analizy obiektów "dużych elektrycznie" (tzn. o dużych wymiarach odniesionych do długości fali roboczej), dla których poszukiwana liczba niewiadomych (składowe pola, współczynniki rozkładu źródeł pola, itp.) idzie w setki tysięcy [1]. Ograniczenia powyższe można jednak częściowo przełamać dzięki zastosowaniu paradygmatu programowania równoległego i/lub akceleracji sprzętowej. Szerokie perspektywy otwiera w tym zakresie wykorzystanie procesorów graficznych GPU, a ostatnio układów logiki programowalnej FPGA.

Zaimplementowane algorytmy metody momentów umożliwiają analizę struktur (obiektów) przewodzących o dowolnym kształcie. U podstaw tego podejścia leży równanie różniczkowo-całkowe typu elektrycznego (EFIE) wiążące pole elektryczne z rozkładem prądu na powierzchni analizowanej struktury (obiektu) [2]. Przybliżone rozwiązanie numeryczne tego równania konstruowane metodą zaproponowaną przez Harringtona [3] prowadzi do układu N równań liniowych, które w notacji macierzowej można zapisać jako

$$\mathbf{ZI} = \mathbf{V}, \tag{1}$$

gdzie $\mathbf{Z}$ jest kwadratową macierzą impedancyjną stopnia N opisującą analizowany obiekt dla jednej częstotliwości, $\mathbf{V}$ jest kolumnowym wektorem reprezentującym pobudzenie obiektu, natomiast $\mathbf{I}$ oznacza wektor poszukiwanych współczynników rozwinięcia prądu. Rozwiązanie tego równania, tzn.

$$\mathbf{I} = \mathbf{Z}^{-1}\mathbf{V} \tag{2}$$

daje współczynniki rozkładu prądu na zbiorze przyjętych funkcji bazowych. Rozwiązanie (2) jest konstruowane bezpośrednio poprzez faktoryzację LU macierzy $\mathbf{Z}$ [4]. Proces ten, podobnie jak wypełnianie macierzy impedancyjnej stanowi jeden z najbardziej intensywnych obliczeniowo fragmentów kodu implementującego MoM. Z tego powodu wydaje się, że zastosowanie układów FPGA w tych dwóch fazach obliczeń może doprowadzić do znaczącego skrócenia czasu analizy numerycznej prowadzonej za pomocą MoM.

W ramach tego artykułu opisano możliwości wykorzystania układów FPGA 10-tej generacji do akceleracji sprzętowej obliczeń związanych z MoM (wersja cienkoprzewodowa), a konkretnie procesu wypełniania macierzy impedancyjnej. Artykuł podzielono na trzy podstawowe rozdziały. W rozdziale pierwszym przedstawiono aktualny stan wiedzy dotyczący sprzętowej akceleracji obliczeń związanych z metodą momentów. Rozdział drugi zawiera opis implementacji MoM w heterogenicznym środowisku obliczeniowym CPU/FPGA. Szczególną uwagę zwrócono na zagadnienia związane z wielowątkowością oraz optymalizacją wydajności kodu/jądra obliczeniowego. Ostatni, trzeci rozdział zawiera wyniki analizy numerycznej anteny kierunkowej radiolatarni naziemnej systemu ILS [5]. Dodatkowo pokazano, w jaki sposób dla wykorzystywanej platformy sprzętowej (CPU/FPGA, platforma wielordzeniowa) zmienia się wartość przyspieszenia programów implementujących MoM w funkcji liczby niewiadomych, tj. rozmiaru analizowanych problemów. Algorytmy przewidziane do realizacji zarówno w heterogenicznym (CPU/FPGA) jak i homogenicznym środowisku obliczeniowym (wyniki odniesienia) napisano w języku programowania OpenCL [6] i skompilowano za pomocą kompilatora firmy Intel ze wspomaganiem matematycznych bibliotek MKL [7]. Wszystkie obliczenia wykonano z wykorzystaniem zmiennych zespolonych podwójnej precyzji.

Akceleracja sprzętowa MoM

W literaturze znaleźć można wiele publikacji poświęconych wykorzystaniu procesorów GPU i technologii CUDA do akceleracji obliczeń numerycznych związanych z MoM. Za ważne, w omawianym kontekście, należy uznać prace [8, 9, 10, 11], w których nakreślono pewien uniwersalny schemat postępowania pozwalający efektywnie zrównoleglić algorytmy MoM zarówno na poziomie procesu wypełniania macierzy impedancyjnej jak i rozwiązywania układu równań liniowych. Rozwinięcie opisanej w [8] idei na przypadek, w którym rozmiar analizowanej struktury/problemu nie mieści się w pamięci globalnej karty graficznej można

znaleźć w [12, 13, 14]. Warto podkreślić, że zaprezentowane w cytowanych pracach podejście nie nadaje się, nie otwiera pola do hybrydyzacji obliczeń związanych z MoM. W konsekwencji działanie jednostki centralnej CPU sprowadzone zostaje wyłącznie do roli jednostki nadzorującej pracę układu GPU. Punktem zwrotnym na drodze do lepszego wykorzystania dostępnych zasobów sprzętowych, a co za tym idzie na drodze do hybrydyzacji obliczeń MoM, okazał się pomysł szczegółowo opisany w [15]. Zgodnie z przyjętym założeniem elementy macierzy impedancyjnej leżące na głównej przekątnej (wypełnianie) oraz elementy podmacierzy $\mathbf{L}$ (rozwiązywanie) wyznaczone są za pomocą jednostki centralnej CPU w momencie, w którym elementy leżące poza główną przekątną (wypełnianie) oraz elementy podmacierzy $\mathbf{U}$ i $\mathbf{A}$ (rozwiązywanie) wyznaczane są na karcie graficznej [4]. Biorąc pod uwagę fakt, że rozwiązywanie układu równań liniowych rozpoczyna się dopiero w chwili zakończenia procesu wypełniania macierzy impedancyjnej, zaproponowane podejście praktycznie nie daje możliwości zrównoważenia obciążeń pomiędzy jednostkami wykonawczymi CPU i GPU. Kluczem do rozwiązania tego problemu, jak pokazano w [16], jest częściowe zaangażowanie jednostki centralnej CPU w proces wyznaczania elementów macierzy impedancyjnej leżących poza główną przekątną (wypełnianie) oraz proces wyznaczania elementów podmacierzy $\mathbf{A}$ (rozwiązywanie). Dodatkowo zgodnie z sugestią zawartą w [17] należy cyklicznie, co jakiś czas zmieniać kolejność wyznaczania elementów podmacierzy $\mathbf{A}$. Dobrym rozwiązaniem w tym przypadku wydaje się być zastosowanie opisanego w [18] schematu propagacji wstecznej zamiennie z schematem propagacji prostej.

Duża popularność technologii CUDA, a co za tym idzie stosunkowo duża liczba publikacji poświęconych wykorzystaniu kart graficznych do akceleracji obliczeń związanych z MoM nie oznacza, że na przestrzeni ostatnich kilkunastu lat nie podjęto żadnego wysiłku programistycznego zmierzającego do realizacji podejścia hybrydowego CPU/FPGA. Warto w tym miejscu odwołać się chociażby do prac [19, 20, 21], w których opisano sprzętową implementację algorytmów odpowiedzialnych za wypełnianie macierzy impedancyjnej (praca [19]) oraz rozwiązywanie układu równań liniowych (prace [20, 21]). Wypracowanie odpowiedniego zysku na czasie analizy względem jednordzeniowej implementacji referencyjnej może jednak wymagać, jak pokazano w pracy [21], zastosowania pewnych dedykowanych rozwiązań systemowych.

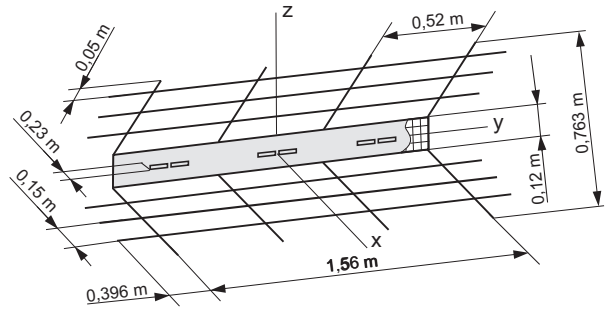
Implementacja hybrydowa CPU/FPGA

Do przyspieszenia obliczeń związanych z MoM wykorzystano akcelerator sprzętowy 385A firmy Nallatech zbudowany na bazie układu FPGA Arria 10 GX 1150 firmy Intel. Kartę akceleratora FPGA podłączono do gniazda magistrali PCI Express 2.0 niskobudżetowej stacji roboczej typu desktop wyposażonej w procesor Intel i7-8700K. Stanowiący integralną część akceleratora 385A układ Arria 10 GX 1150 zawiera w swojej strukturze 1518 procesorów DSP, 427,2 tys. modułów ALM, 1,7 mln rejestrów ALM, 1,15 mln elementów logicznych LE oraz 2731 bloków pamięci wbudowanej RAM o pojemności 20 kB (w zależności od potrzeb zasoby te mogą zostać przekształcone na drodze syntezy logicznej w wysoko wyspecjalizowane, wysoko wydajne jednostki obliczeniowe, które następnie mogą zostać wykorzystane do efektywnej realizacji wybranych fragmentów kodu komputerowego, w tym kodu implementującego MoM).

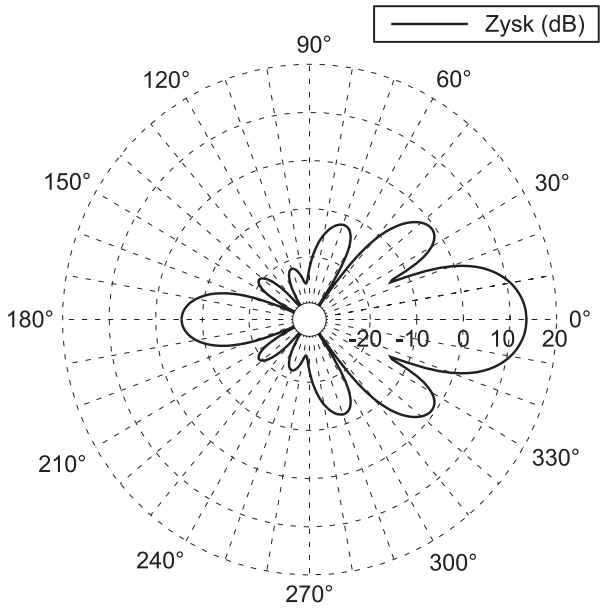
Na rys. 1 pokazano uproszczoną wersję kodu komputerowego implementującego MoM w heterogenicznym środowisku obliczeniowym CPU/FPGA. Jak łatwo zauważyć układ FPGA wykorzystywany jest do wyznaczenia elementów macierzy impedancyjnej leżących poza główną przekątną (procedura `Z_non_self_terms()`), natomiast jednostka centralna – do wyznaczenia elementów leżących na głównej przekątnej (procedura `Z_self_terms()`). Proces wyznaczania elementów macierzy impedancyjnej leżących poza główną przekątną (jądro MoM) podzielono na osiem niezależnych zadań cząstkowych realizowanych przez dwie zsyntezowane w układzie FPGA jednostki wykonawcze CU1 i CU2 (po cztery zadania na jedną jednostkę CU). Zastosowanie takiego rozwiązania/sposobu mapowania funkcji jądra daje możliwość kopiowania części wyników końcowych z pamięci globalnej karty akceleratora do pamięci RAM stacji roboczej w trakcie wykonywania funkcji jądra (*ang.* overlapping) [6], co bezpośrednio przekłada się na skrócenie czasu obliczeń podobnie jak wykorzystanie bloków pamięci wbudowanej RAM o pojemności 20 kB (20 kB BRAMs) w roli pamięci podręcznej cache pierwszego poziomu do przechowywania zmiennych lokalnych oraz danych wejściowych. W odróżnieniu od podejścia zaprezentowanego w [19] rozwiązywanie układu równań liniowych (1) na CPU rozpoczyna się jeszcze przed zakończeniem procesu

```
   #include "CL/opencl.h"
   #include "AOCLUtils/aocl_utils.h"

   // wczytanie modelu analizowanej struktury
1: call geoinput(x, y, z, rad, ...);
   ...
   // inicjalizacja obsługiwanej kolejki zdarzeń
2: cl_command_queue queue = clCreateCommandQueue(context,
   device, ..., CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE, ...);
   ...
   // utworzenie listy obsługiwanych zdarzeń
3: cl_event events[16];
   ...
   // przydzielenie zasobów pamięci globalnej do przechowywania wyników
      obliczeń
4: for (int i = 0; i < 8; i++)
5:   cl_mem cz_d[i] = clCreateBuffer(context,
     CL_MEM_WRITE_ONLY, size_cz, ...);
   ...
   // kopiowanie danych wejściowych z pamięci RAM komputera do pamięci
      globalnej karty akceleratora
6: call clEnqueueWriteBuffer(queue, x_d, CL_FALSE, 0, ...,
   x, ...);
   ...
   // wywołanie funkcji jądra (wyznaczenie elementów macierzy impedancyjnej
      leżących poza diagonalą)
7: call Z_non_self_terms(queue, device, x_d, ..., cz_d , ...,
        &events);
   // wyznaczenie elementów macierzy impedancyjnej leżących na diagonali
      na CPU
8: call Z_self_terms(cz, x, y, x, rad, ...);
   // rozwiązanie układu równań liniowych na CPU
9: call zgesv(n, 1, cz, n, ipv, ci, n, ...);
   ...
   // zwolnienie wykorzystywanych zasobów
10: call clReleaseMemObject(cz_d[i], x_d, y_d, z_d, rad_d, ...);

   //  ------------------ procedura Z_non_self_terms ------------------
11: void Z_non_self_terms(cl_command_queue q, ...,
    float* x_d, ..., clDoubleComplex* cz_d, ..., cl_event ev){
    ...
12: int num_iter = 4;
    ...
    // realizacja na tzw. zakładkę funkcji jądra i operacji kopiowania danych
       (wyników obliczeń)
13: for (int i = 0; i < num_iter; i++){
14:   call clEnqueueNDRangeKernel(q, „Z_ww_kernel_1”,
      cz_d[(n/8)*i], i, ..., ev[4*i]);
15:   call clEnqueueReadBuffer(q, cz_d[(n/8)*i],CL_FALSE, ...,
      cz[(n/8)*i],0, ev[1+(4*i)]);
16:   call clEnqueueNDRangeKernel(q, „Z_ww_kernel_2”,
      cz_d[(n/2)+(n/8)*i], i, ..., ev[2+(4*i)]);
17:   call clEnqueueReadBuffer(q, cz_d[(n/2)+(n/8)*i],CL_FALSE,
      ..., cz[(n/2)+(n/8)*i],0, ev[3+(4*i)]);
18: }
19: clFlush();
20: }
```

Rys. 1. Uproszczona wersja kodu implementującego MoM w heterogenicznym środowisku obliczeniowym CPU/FPGA



Rys. 2. Model analizowanej anteny systemu ILS



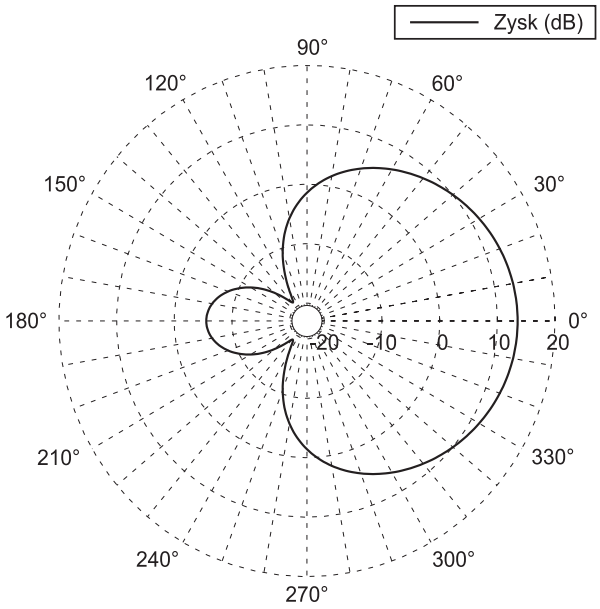
Rys. 3. Funkcja zysku energetycznego w płaszczyźnie $x-y$ anteny kierunkowej z rys. 2

wyznaczania elementów macierzy impedancyjnej na karcie akceleratora. Zamierzonym skutkiem takiego działania jest dalsze skrócenie czasu obliczeń związanych z MoM. Jednocześnie z powodów szeroko omówionych w [21, 22] oraz [23] zrezygnowano całkowicie z pomysłu wykorzystania układu FPGA do przyspieszenia procesu rozwiązywania układu równań liniowych (1).

Implementacja sprzętowa algorytmów MoM odpowiedzialnych za wyznaczenie elementów macierzy impedancyjnej leżących poza główną przekątną wymaga użycia 1220 procesorów DSP, 338 tys. modułów ALM, 1,236 mln rejestrów ALM oraz 1672 bloków pamięci wbudowanej RAM o pojemności 20 kB, co stanowi odpowiednio 80% łącznej liczby procesorów DSP w układzie FPGA, 79% łącznej liczby modułów ALM, 73% łącznej liczby rejestrów ALM oraz 61% łącznej liczby bloków pamięci RAM o pojemności 20 kB. Jak widać zaimplementowana funkcja jądra charakteryzuje się dużą liczbą wykorzystywanych procesorów DSP oraz modułów ALM, co w kontekście prowadzonych obliczeń zmiennoprzecinkowych podwójnej precyzji zgodnych ze standardem IEEE-754 nie powinno być zaskoczeniem. Częstotliwość taktowania obydwu zsyntezowanych w układzie FPGA jednostek wykonawczych CU1 i CU2 jest taka sama i wynosi 240 MHz.

Wyniki obliczeń numerycznych

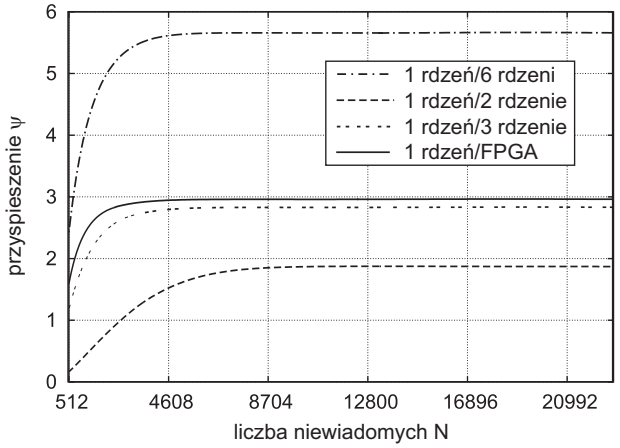
Efektywność działania opisanego sposobu hybrydyzacji obliczeń pokazano na przykładzie analizy anteny kierunkowej radiolatarni naziemnej systemu ILS. Antenę umieszczono wzdłuż osi y kartezjańskiego układu współrzędnych tak, aby kierunek maksymalnego promieniowania pokrywał się z osią x (rys. 2). Do celów analizy numerycznej antenę zamodelowano za pomocą 771 cienkich przewodów, na których rozpięto łącznie 2568 liniowych funkcji bazowych. Wszystkie przewody imitujące elementy promieniujące (dipole) o długości 320 mm mają średnicę równą 5 mm, natomiast przewody tworzące reflektor – średnicę 1 mm.



Rys. 4. Funkcja zysku energetycznego w płaszczyźnie $x-z$ anteny kierunkowej z rys. 2

Na rys. 3 i 4 pokazano rozkład funkcji zysku energetycznego dla analizowanej anteny w płaszczyźnie H (płaszczyzna $x-y$) i E (płaszczyzna $x-z$). Obliczenia wykonano dla częstotliwości $f = 332$ MHz i kąta ϕ (Θ) zmieniającego się z krokiem 1°. Wyznaczenie pokazanych na rys. 3 i 4 charakterystyk promieniowania za pomocą MoM zaimplementowanej w heterogenicznym środowisku obliczeniowym CPU/FPGA zajmuje 8,3 s. Czas ten ulega skróceniu do 4,4 s (1,88x) w przypadku, w którym zastosujemy implementację sześciordzeniową oraz wydłużeniu do 24,6 s (2,96x), 13,4 s (1,61x) oraz 8,8 s (1,06x) w przypadku, w którym zastosujemy, odpowiednio, implementację jedno-, dwu- i trzyrdzeniową. Należy podkreślić, że podany czas dla implementacji hybrydowej uwzględnia również wszystkie niezbędne transfery danych między jednostką centralną CPU i kartą akceleratora FPGA transfery, które dodajmy zostały częściowo ukryte. Nie zmienia to niestety faktu, że zaproponowane podejście hybrydowe ma nieco niższą wydajność obliczeniową niż implementacja czterordzeniowa, nie wspominając o implementacji pięcio-, czy sześciordzeniowej. Problem ten, jak zasygnalizowano w pracy [19], można rozwiązać na dwa wzajemnie uzupełniające się sposoby: (i) poprzez wykorzystanie układów FPGA o większej wydajności/mocy obliczeniowej (Stratix 10, UltraScale+) [24, 25] oraz (ii) poprzez zwiększenie stopnia hybrydyzacji i/lub granulacji obliczeń.

Na rys. 5 przedstawiono wykres ilustrujący zmianę przyspieszenia Ψ programów implementujących MoM (proponowana wersja hybrydowa, wersja wielordzeniowa) w funkcji liczby niewiadomych N. Obliczenia wykonano dla N zmieniającego się w zakresie od 512 do 22768. Największa wartość N z grubsza odpowiada największej macierzy



Rys. 5. Wykres zmian przyspieszenia Ψ programów implementujących MoM w funkcji liczby niewiadomych N dla heterogenicznej (CPU/FPGA) oraz homogenicznej (wielordzeniowej) platformy obliczeniowej

typu `double complex`, którą można pomieścić w pamięci globalnej karty akceleratora (8 GB). Jak łatwo zauważyć, pokazany na rys. 5 przebieg zmian przyspieszenia Ψ dla obliczeń wykonywanych ze wsparciem układu FPGA dość dobrze wpisuje się w ogólny charakter zmian obserwowanych w platformach równoległych, dla których po początkowej fazie wyraźnego wzrostu przyspieszenia Ψ następuje w miarę szybkie jego przejście do fazy względnej stabilizacji.

Podsumowanie

W artykule opisano możliwości wykorzystania układów logiki programowalnej do przyspieszenia obliczeń związanych z MoM. Jak pokazują wyniki przeprowadzonych eksperymentów zastosowanie opisanego schematu hybrydyzacji pozwala na 2,96-, 1,61- oraz 1,06-krotne skrócenie czasu analizy numerycznej obiektów przewodzących w porównaniu do implementacji, odpowiednio, jedno-, dwu- i trzyrdzeniowej. W porównaniu do implementacji cztero-, pięcio- i sześciordzeniowej czas ten ulega, odpowiednio, 1,25-, 1,56- oraz 1,88-krotnemu wydłużeniu. Rozwiązaniem pozwalającym na znaczne zwiększenie efektywności działania zaproponowanego schematu hybrydyzacji jest wykorzystanie układu FPGA o większej wydajności obliczeniowej np. Stratix 10 lub UltraScale+. Główną zaletą opisanego podejścia jest niskie zapotrzebowanie na energię. Całkowity pobór mocy z zasilacza w trakcie obliczeń nie przekracza 54 W (o 55 W mniej niż w przypadku wykorzystania implementacji sześciordzeniowej oraz o 36 W mniej niż w przypadku wykorzystania implementacji trzyrdzeniowej).

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

LITERATURA

[1] Sadiku M. N. O.: Numerical Techniques in Electromagnetics, CRC Press, 2001.
[2] Hwu S. U., Wilton D. R.: Electromagnetic Scattering and Radiation by Arbitrary Configurations of Conducting Bodies and Wires Tech. Doc. 1325, AEL, Univ. of Houston, 1988.
[3] Harrington R. F.: Field Computation by Moment Methods, Macmillan, 1968.
[4] Golub G. H., van Loan C. F.: Matrix Computations, The John Hopkins University Press, 1996.
[5] Kayton M., Fried W. R.: Avionics Navigation Systems, John Wiley & Sons, 1997.
[6] Khronos OpenCL Working Group, The OpenCL specification V3.0.12, Beaverton, 2022
[7] Intel Corporation, Intel OneAPI Math Kernel Library. Developer Reference, Santa Clara, 2021.
[8] 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.
[9] 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.
[10] De Donno D., Esposito A., Monti G., Tarricone L.: Efficient acceleration of sparse MPIE/MoM with graphics processing units, Proc. of the 41st European Microw. Conf., EuMC, Manchester, UK, 2011.
[11] De Donno D., Esposito A., Monti G., Tarricone L.: MPIE/MoM acceleration with a general-purpose graphics processing unit, IEEE Trans. Microw. Theory Tech., 60(9), pp. 2693–2701, 2012.
[12] Lezar E., Davidson D.: GPU-based LU decomposition for large method of moments problems, Electron. Lett., 46(17), pp. 1194-–1196, 2010.
[13] Lezar E., Davidson D.: GPU-accelerated method of moments by example: monostatic scattering, IEEE Antennas Propag. Mag., 52(6), pp. 120-–135, 2010.
[14] Mu X., Zhou H.-X., Chen K., Hong W.: Higher order method of moments with a parallel out-of-core LU solver on GPU/CPU platform, IEEE Trans. Antennas Propag., vol. 62(11), pp. 5634-5646, 2014.
[15] Kolundzija B., Zoric D.: Efficient evaluation of MoM matrix elements using CPU and/or GPU, European Conf. on Antennas and Propag., EuCAP, Prague, Czech Rep., pp. 702–706, 2012.
[16] Topa T.: Load balanced Fortran-based out-of-GPU memory implementation of the method of moments, IEEE Antennas Wireless Propag. Lett., 16, pp. 813–816, 2017.
[17] Karwowski A., Noga A., Topa T.: An Efficient Framework for Analysis of Wire-Grid Shielding Structures over a Broad Frequency Range, Radioengineering, 25(4), pp. 629–636, 2016
[18] Topa T.: Efficient out-of-GPU memory strategies for solving matrix equation generated by method of moments, Electron. Lett., 51(19), pp. 1542–1543, 2015.
[19] Topa T.: Porting wire-grid MoM framework to reconfigurable computing technology, IEEE Antennas Wirel. Propag. Lett., 19(9), pp. 1630–1633, 2020.
[20] Devi A., Gandhi M., Varghese K., Gope D.: Accelerating method of moments based package-board 3D parasitic extraction using FPGA, Microw. Opt. Technol. Lett., 58(4), pp. 776–782, 2016.
[21] Hauser T., Dasu A., Sudarsanam A., Young S.: Performance of a LU decomposition on a multi-FPGA system compared to a low power commodity microprocessor system, Journal of Scalable Computing: Practice and Experience, 8(4), pp. 373–385, 2007.
[22] Matties T., Fine Licht J., Hoefler T.: FBLAS: Streaming Linear Algebra Kernels on FPGA, Int. Conf. for HPC Netw. Stor. Anal., pp. 17–22, 2019.
[23] Kestur S., Devids J. D., Williams O.: BLAS Comparison on FPGA, CPU and GPU, Proc. IEEE Comput. Soc. Annu. Symp. VLSI, pp. 288–293, 2010.
[24] Intel, Intel Stratix 10 Overview, 2022.
[25] Xilinx, Alveo Data Center Accelerator Card Platforms. User Guide, 2022.