POLITECHNIKA GDAŃSKA
Wydział Elektroniki, Telekomunikacji
i Informatyki

Piotr Sypek

Metody sprzętowego oraz algorytmicznego skrócenia czasu numerycznej analizy zagadnień elektromagnetycznych opartej na metodzie różnic skończonych w dziedzinie czasu

Rozprawa doktorska

Promotor:

prof. dr hab. inż. Michał Mrozowski, prof. zw. PG

Wydział Elektroniki, Telekomunikacji
i Informatyki
Politechnika Gdańska

Gdańsk, 2011

Spis treści

Spis rysunków

Spis tablic

Wykaz zastosowanych skrótów i symboli

Wykaz skrótów

Podane w nawiasach kwadratowych liczby wskazują na numery stron, w których występuje istotne odwołanie się do danego skrótu.

ALU	-	jednostka arytmetyczno-logiczna (ang. *Arithmetic Logic Unit*) [lista stron: 78]
CPML	-	PML z zastosowaniem splotu funkcji; usprawniona wersja PML (ang. *Convolution PML*) [lista stron: 143]
CPU	-	procesor komputerowy (ang. *Central Processing Unit*) [lista stron: 41]
CRS	-	format kompresji wierszy w macierzy rzadkiej (ang. *Compressed Row Storage*) [lista stron: 96, 105]
CUDA	-	równoległa architektura obliczeniowa akceleratorów graficznych firmy Nvidia (ang. Compute Unified Device Architecture) [lista stron: 77–79]
ENOR	-	technika redukcji rzędu modelu (ang. *Efficient Nodal Order Reduction*) [lista stron: 126]
FDFD	-	metoda różnic skończonych w dziedzinie częstotliwości (ang. *Finite Difference Frequency Domain*) [lista stron: 36]
FDTD	-	metoda różnic skończonych w dziedzinie czasu (ang. *Finite Difference Time Domain*) [lista stron: 36]
FIT	-	metoda całek skończonych (ang. *Finite Integration Technique*) [lista stron: 37]

FPU - jednostka realizująca obliczenia zmiennoprzecinkowe (ang. *Floating-Point Unit*)
[lista stron: 48]

GPOF - metoda uogólnionego pęku funkcyjnego (ang. *Generalized Pencil-of-Function Method*)
[lista stron: 172]

GPU - jednostka przetwarzania graficznego (ang. *Graphics Processing Unit*)
[lista stron: 41, 77–80]

MPI - interfejs transmisji danych (ang. *Message Passing Interface*)
[lista stron: 65, 105]

OpenGL - interfejs programistyczny do przetwarzania grafiki komputerowej (ang. *Open Graphics Library*)
[lista stron: 77]

PML - dopasowana warstwa absorpcyjna (ang. *Perfectly Matched Layer*)
[lista stron: 171]

RAM - pamięć o dostępie swobodnym (ang. *Random Access Memory*)
[lista stron: 49, 58, 65, 79]

SIMD - jedna instrukcja, wiele danych; technika przetwarzania równoległego, w której jedna instrukcja wykonywana jest wielokrotnie na zbiorze danych w tym samym czasie (ang. *Single Instruction Multiple Data*)
[lista stron: 79]

SIMT - jedna instrukcja, wiele wątków; technika przetwarzania równoległego opracowana w ramach technologi CUDA, która umożliwia grupie wątków realizację w tym samym czasie tej samej instrukcji, przy czym dla każdej instrukcji możliwe jest warunkowe ograniczenie zakresu działania wątków do wybranego podzbioru (ang. *Single-Instruction, Multiple-Thread*)
[lista stron: 79]

SSE - zestaw instrukcji typu SIMD dostępnych w procesorach x86 (ang. *Streaming SIMD Extensions*)
[lista stron: 51, 175]

XMM - rejestr XMM stosowany jest w CPU do wykonywania instrukcji SSE
[lista stron: 53, 54, 175]

Wykaz symboli

Podana w nawiasach kwadratowych liczba wskazuje na numer strony, w której dany symbol zdefiniowano lub po raz pierwszy zastosowano.

$\mathbf{a}$ - wektor

$\mathbf{A}$ - macierz

E_x, E_y, E_z - składowe pola elektrycznego określone w układzie kartezjańskim, odpowiednio w kierunku x, y, z
[strona: 26]

$\vec{E}$ - wektor natężenia pola elektrycznego
[strona: 26]

H_x, H_y, H_z - składowe pola magnetycznego określone w układzie kartezjańskim, odpowiednio w kierunku x, y, z
[strona: 26]

$\vec{H}$ - wektor natężenia pola magnetycznego
[strona: 26]

$\nabla \times \vec{A}$ - rotacja pola wektorowego $\vec{A}$
[strona: 25]

t - czas
[strona: 25]

ϵ_x, ϵ_y, ϵ_z - składowa diagonalnego tensora przenikalności elektrycznej w kierunku odpowiednio x, y, z
[strona: 26]

μ_x, μ_y, μ_z - składowa diagonalnego tensora przenikalności magnetycznej w kierunku odpowiednio x, y, z
[strona: 26]

$\overleftrightarrow{\epsilon}$ - tensor przenikalności elektrycznej
[strona: 25]

$\overleftrightarrow{\mu}$ - tensor przenikalności magnetycznej
[strona: 25]

$\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$ - wektor próbek odpowiednio składowej e_x, e_y, e_z
[strona: 33]

$\mathbf{h}_x$, $\mathbf{h}_y$, $\mathbf{h}_z$ - wektor próbek odpowiednio składowej h_x, h_y, h_z
[strona: 33]

Δ - krok dyskretyzacji przestrzeni ciągłej
[strona: 27]

Δ_t - krok dyskretyzacji dziedziny czasu
[strona: 36]

Δ_x, Δ_y, Δ_z - krok dyskretyzacji przestrzeni odpowiednio w kierunku x, y, z
[strona: 28]

$\mathbf{e}$ - wektor próbek pola elektrycznego
[strona: 33]

$\mathbf{h}$ - wektor próbek pola magnetycznego
[strona: 33]

$\tilde{\mathbf{e}}$ - wektor próbek pola elektrycznego w symetrycznym schemacie różnicowym
[strona: 38]

$\tilde{\mathbf{h}}$ - wektor próbek pola magnetycznego w symetrycznym schemacie różnicowym
[strona: 38]

$\mathbf{e}^n$ - wektor próbek pola elektrycznego w czasie $t = n\Delta_t$
[strona: 36]

$\mathbf{h}^{n+0,5}$ - wektor próbek pola magnetycznego w czasie $t = (n + 0,5)\Delta_t$ [strona: 36]

$E_x(j)$ - j-ta próbka składowej E_x pola elektrycznego w problemie jednowymiarowym [strona: 28]

$H_z(j)$ - j-ta próbka składowej H_z pola magnetycznego w problemie jednowymiarowym [strona: 28]

$\epsilon_x(j)$ - j-ta próbka ϵ_x w problemie jednowymiarowym [strona: 28]

$\mu_z(j)$ - j-ta próbka μ_z w problemie jednowymiarowym [strona: 28]

$E_x(i,j,k)$, $E_y(i,j,k)$, $E_z(i,j,k)$ - próbka odpowiednio składowej E_x, E_y, E_z w problemie trójwymiarowym [strona: 29]

$H_x(i,j,k)$, $H_y(i,j,k)$, $H_z(i,j,k)$ - próbka odpowiednio składowej H_x, H_y, H_z w problemie trójwymiarowym [strona: 29]

$\epsilon_x(i,j,k)$, $\epsilon_y(i,j,k)$, $\epsilon_z(i,j,k)$ - próbka odpowiednio składowej ϵ_x, ϵ_y, ϵ_z w problemie trójwymiarowym [strona: 29]

$\mu_x(i,j,k)$, $\mu_y(i,j,k)$, $\mu_z(i,j,k)$ - próbka odpowiednio składowej μ_x, μ_y, μ_z w problemie trójwymiarowym [strona: 29]

I, J, K - liczba komórek Yee dziedziny obliczeniowej odpowiednio w kierunku osi x, y, z [strona: 33]

N - liczba komórek Yee dziedziny obliczeniowej [strona: 33]

$\mathbf{D}_\epsilon$ - macierz reprezentująca $\overleftrightarrow{\epsilon}$ w dziedzinie dyskretnej [strona: 35]

$\mathbf{D}_\mu$ - macierz reprezentująca $\overleftrightarrow{\mu}$ w dziedzinie dyskretnej [strona: 35]

$\mathbf{R}_E$ - macierz rotacji pola elektrycznego w dziedzinie dyskretnej [strona: 35]

$\mathbf{R}_H$ - macierz rotacji pola magnetycznego w dziedzinie dyskretnej [strona: 35]

$\tilde{\mathbf{R}}_E$ - macierz rotacji pola elektrycznego w dziedzinie dyskretnej w symetrycznym schemacie różnicowym [strona: 38]

$\tilde{\mathbf{R}}_H$ - macierz rotacji pola magnetycznego w dziedzinie dyskretnej w symetrycznym schemacie różnicowym [strona: 38]

α_F - liczba bajtów wymagana do reprezentacji liczby zmiennoprzecinkowej w programie komputerowym [strona: 47]

α_I - liczba bajtów wymagana do reprezentacji liczby całkowitej w programie komputerowym
[strona: 179]

S_p - przyspieszenie obliczeń równoległych
[strona: 67]

E_p - efektywność zrównoleglenia implementacji algorytmu
[strona: 67]

L_B - liczba bajtów wymagana przez algorytm FDTD w postaci jawnej do reprezentacji próbek pola elektromagnetycznego oraz parametrów materiałowych
[strona: 47]

L_F - liczba operacji zmiennoprzecinkowych wykonywanych w jednej iteracji algorytmu FDTD w postaci jawnej
[strona: 47]

S_B - liczba danych przesłanych pomiędzy jednostką obliczeniową FPU, a pamięcią
[strona: 48]

S_C - liczba komórek Yee, dla których wykonano aktualizację wartości pól w czasie jednej sekundy
[strona: 48]

S_F - liczba operacji zmiennoprzecinkowych wykonanych w ciągu jednej sekundy
[strona: 48]

ROZDZIAŁ 1

Wstęp

Fundament współczesnej analizy obwodów elektrycznych i fotonicznych stanowią równania Maxwella, [61]. Zbudowana w oparciu o nie analiza obwodowa umożliwia przeprowadzenie symulacji działania układów elektrycznych w zakresie niskich częstotliwości. Jednak gdy długość fali elektromagnetycznej jest porównywalna z gabarytami elementów fizycznych tworzących układ elektryczny, analiza obwodowa nie jest wystarczającym narzędziem do reprezentacji zjawisk elektromagnetycznych w nich występujących. W takiej sytuacji opis danego układu należy przeprowadzić za pomocą analizy pełnofalowej. Analiza taka może zostać wykonana analitycznie, poprzez wyznaczenie rozwiązania równań Maxwella dla badanej struktury lub numerycznie, np. poprzez dyskretyzację czasoprzestrzeni. Rozwiązanie analityczne stanowi dokładny opis struktury elektromagnetycznej, jednak wymaga ono indywidualnego podejścia do każdej rozważanej struktury. Natomiast zastosowanie metod numerycznych z zakresu elektrodynamiki obliczeniowej, umożliwia otrzymanie przybliżonego modelu działania struktur elektromagnetycznych o dużej złożoności.

Ponieważ współczesny rozwój technologiczny sprawia, że urządzenia elektroniczne działają z coraz większymi częstotliwościami sygnałów, to nieodłącznym składnikiem nowoczesnych programów komputerowych przeznaczonych do projektowania m.in. układów mikrofalowych są pełnofalowe symulatory propagacji fali elektromagnetycznej, [9, 22, 85]. Jedną z podstawowych metod pełnofalowej analizy struktur mikrofalowych i fotonicznych jest metoda różnic skończonych w dziedzinie czasu, FDTD (ang. *Finite Difference Time Domain*). Stanowi ona uniwersalne narzędzie analizy zagadnień elektromagnetycznych, które umożliwia przeprowadzenie symulacji działania układów mikrofalowych i fotonicznych o złożonej strukturze. Jej wadą jest konieczność przetwarzania dużej liczby zmiennych i związany z tym długi czas obliczeń, [105].

Skrócenie czasu symulacji przeprowadzonych przy pomocy algorytmu FDTD może zostać przeprowadzone dwoma podstawowymi metodami: sprzętową i algorytmiczną.

Najbardziej rozpowszechnioną sprzętową metodą przyspieszania tego algorytmu jest jego zrównoleglenie dostosowane do klastra lub grupy komputerów z sieciowym połączeniem. Dla tego typu środowiska udowodniono dobre dostosowanie algorytmu FDTD do zrównoleglenie już w latach dziewięćdziesiątych XX wieku, [29, 58, 30], chociaż napotykano trudności w osiągnięciu skalowalności [87, 110]. Z obliczeniami, które są przeprowadzone na wielu komputerach, ściśle związane są standardy, które ułatwiają zdefiniowanie komunikacji pomiędzy nimi. Najbardziej popularne wśród nich są: MPI, [64], (wersja 1.0 tego interfejsu została opublikowana w czerwcu 1994 r.) oraz PVM, [83], (wersja napisana w języku C została opublikowana w 1991 r.). Z PVM korzystano przy budowaniu implementacji FDTD opisanej w [110], jednak w XXI wieku do zrównoleglenia obliczeń w klastrze najczęściej stosuje się MPI, [31, 119], który umożliwia osiągnięcie skalowalności. W kolejnych latach rozmiar problemów analizowanych przy pomocy metody FDTD znacznie wzrósł [15], przy czym nadal wykazywano skalowalność tego algorytmu [121, 112]. Powszechnie stosowaną metodą zrównoleglenia stanowiła dekompozycja dziedziny obliczeniowej (ang. *domain decomposition*) na szereg poddziedzin, które były następnie przyporządkowane do dedykowanym im procesorów komputerowych. Tę metodę stosowano zarówno dla algorytmu FDTD, [98], jak i np. FEM, [111]. Wykazano przy tym, że szybkość obliczeń tak zrównoleglonego algorytmu zależy od sposobu podziału poddziedziny obliczeniowej, [122] (obliczenia dla klastra o znacznej liczbie węzłów), tzn. od liczby przekrojów zdefiniowanych dla zadanego kierunku przestrzeni.

W obliczeniach z procesorami jednordzeniowymi wykazywano skalowalność algorytmu FDTD, jednak wraz z pojawieniem się procesorów wielordzeniowych sytuacja zmieniła się. W niektórych artykułach dla takiej architektury przedstawiano brak skalowalności [95, 116, 70], która była szczególnie widoczna w porównaniu do procesorów Cell firmy IBM, [117]. Z drugiej strony, w artykule [18] wykazano, że przy obliczeniach zrównoleglonych możliwe jest osiągnięcie wzrostu szybkości obliczeń o wartości większej niż dodatkowa liczba rdzeni biorąca udział w obliczeniach. Jednym z sugerowanych rozwiązań mających na celu poprawę szybkości obliczeń dla procesorów wielordzeniowych stanowić miała metoda łączenia w jednej implementacji interfejsu MPI oraz OpenMP (ang. *Open Multi-Processing*, interfejs ułatwiający zrównoleglenie programów na komputerach z pamięcią wspólną dla wielu rdzeni procesorów komputerowych), [99, 89, 117].

Zrównoleglone implementacje algorytmu FDTD przedstawione w powyżej wskazanych artykułach dotyczą algorytmu FDTD w postaci jawnej. Skalowalność tego algorytmu w postaci macierzowej została przedstawiona w [88].

Równolegle do pojawienia się procesorów wielordzeniowych badacze próbowali skrócić czas symulacji FDTD poprzez zbudowanie urządzeń realizujących obliczenia nie na procesorach komputerowych, lecz na urządzenia specjalnie w tym celu zaprojektowanych: stosujących układy FPGA [24, 93] lub VLSI [74]. Urządzenia te umożliwiały przeprowadzenie obliczeń w czasie kilkunastokrotnie krótszym niż współczesne im procesory komputerowe. Jednak ich stopień komplikacji oraz trudna dostępność spowodowały, że w następnych latach prace w tym kierunku zostały zaniechane.

Dodatkową przyczyną zaprzestania prac nad produkcją urządzeń dedykowanych do obliczeń FDTD stanowił szybki rozwój zarówno procesorów komputerowych jak i akceleratorów graficznych. Szybko rosnąca moc obliczeniowa kart graficznych oraz znaczne ich rozpowszechnienie na rynku konsumenckim, sprawiły, że wśród badaczy powstała koncepcja zastosowania ich w obliczeniach numerycznych. Obliczenia FDTD na kartach graficznych wymagały operowania na OpenGl lub DirectX (jeszcze na początku XXI wieku biblioteki

te stanowiły jedyny sposób realizacji obliczeń na kartach graficznych), [41, 37], lub wymagały zastosowania specjalistycznego oprogramowania, [100]. Pomimo trudności programistycznych, w powyżej wskazanych artykułach udowodniono, że obliczenia przeprowadzone na kartach graficznych mogą charakteryzować się szybkością obliczeń wielokrotnie większą w porównaniu do procesorów komputerowych.

Projektowanie programów dla akceleratorów graficznych zostało znacznie ułatwione po udostępnieniu przez firmę Nvidia technologi CUDA w 2006 r. Po tym czasie pojawiło się wiele publikacji, w których udokumentowano znaczną szybkość obliczeń przeprowadzanych na karcie graficznej dla różnych rodzajów algorytmu: FDTD dwuwymiarowego [27, 38, 23, 109], FDTD trójwymiarowego [67, 17, 60], FDTD dla ośrodków dyspersyjnych [123], LOD-FDTD [106], ADI-FDTD [97], a także dla algorytmu TLM [91, 90]. Kolejnym etapem jest zastosowanie wielu kart graficznych w obliczeniach FDTD - w [72] wykazano znaczne skrócenie obliczeń w takim środowisku obliczeniowym. Pomimo znakomitych rezultatów w przyspieszaniu obliczeń osiągniętych w technologii CUDA trwają prace nad dalszym usprawnieniem procesu projektowania programów dedykowanym kartom graficznym. Jednym z przykładów najnowszych rozwiązań w tej dziedzinie jest OpenCL, który umożliwia napisanie jednego programu, który może być uruchomiony zarówno na wielu procesorach jak i na karcie graficznej. Jednak w chwili obecnej nie jest ona na tyle wydajna, żeby dorównać programom napisanym C/C++ zarówno dla procesorów jak i dla kart graficznych, [96].

Drugą z metod skrócenia czasu obliczeń symulacji FDTD są metody algorytmiczne, które polegają na wprowadzeniu nowych zależności do schematu różnicowego mających na celu przyspieszenie obliczeń. Metody algorytmiczne przyspieszania działania algorytmu różnic skończonych stanowią przedmiot wielu publikacji, w których wykazano ich wysoką skuteczność w obniżeniu czasu numerycznej analizy. Przez lata opracowano wiele rodzajów zmian w algorytmie FDTD. Najważniejsze z nich polegają na wprowadzenie do schematu różnicowego: siatek niejednorodnych [105], schematów lokalnych [16, 35, 40, 63, 82], lokalnego zagęszczenia siatki dyskretyzacji [103, 71, 107] lub makromodeli [49, 50, 51, 45, 48, 79, 102]. Obszerne omówienie tych zagadnień znajduje się w rozprawie [43]. Najważniejsze z punktu widzenia tej rozprawy są metody wprowadzenia lokalnego zagęszczenia siatki dyskretyzacji w schemacie różnicowym oraz metody redukcji liczby zmiennych w podobszarach dziedziny obliczeniowej (tzw. makromodele). W tej rozprawie badaniom poddano efektywność działania metody lokalnego zagęszczenia siatki dyskretyzacji oraz wprowadzenia makromodeli do schematu różnicowego również w kontekście własności ich zrównoleglenia.

1.1 Cel pracy

Cel pracy doktorskiej stanowiło wskazanie numerycznie efektywnego i ekonomicznie atrakcyjnego rozwiązania sprzętowego, które umożliwia przeprowadzenie analizy problemów o złożonej geometrii i znacznych wymiarach elektrycznych z wysoką dokładnością odwzorowywanego pola. Badaniom zostały poddane procesory wielordzeniowe, klastry oraz akceleratory graficzne (GPU). Każde z tych urządzeń wymaga zrównoleglenia algorytmu różnic skończonych, co stanowi zasadniczą część pracy doktorskiej. Przetwarzanie równoległe obejmuje również nowe rozwiązania algorytmiczne, które zwiększają efektywność metody różnic skończonych: zagęszczenie siatki dyskretyzacji z zachowaniem stabilności algorytmu

oraz wprowadzenie wielu makromodeli pozwalających lokalnie wielokrotnie zagęszczać siatkę dyskretyzacji bez konieczności skrócenia kroku czasowego. Główny nacisk pracy został położony na skrócenie czasu numerycznej analizy struktur quasi-periodycznych. Tezy rozprawy brzmią następująco:

1. Akceleratory graficzne zastosowane do obliczeń opartych na metodzie różnic skończonych w dziedzinie czasu i jej modyfikacjach stanowią tańsze i bardziej efektywne źródło mocy obliczeniowej niż kilkuprocesorowy klaster.

2. Wykorzystanie mocy obliczeniowej akceleratorów graficznych wymaga odmiennego podejścia przy konstruowaniu algorytmu niż stosowane w środowisku klastrowym.

1.2 Zakres pracy

Rozprawa składa się z ośmiu rozdziałów oraz trzech dodatków. Rozdział pierwszy stanowi wprowadzenie do zagadnień przyspieszania obliczeń metodami sprzętowymi oraz algorytmicznymi, a także formułuje cele i tezy rozprawy. W drugim rozdziale przedstawiono podstawowe własności metody różnic skończonych w dziedzinie czasu oraz wprowadzono podstawowe pojęcia stosowane w kolejnych rozdziałach.

Rozdział trzeci stanowi opis implementacji jawnej algorytmu FDTD przeznaczonej dla procesora komputerowego, klastra oraz akceleratora graficznego. Ponadto zawiera rezultaty badań wyjaśniających wpływ czynników charakteryzujących te środowiska sprzętowe na efektywność w/w implementacji. W następnych rozdziałach testy kolejnych implementacji również przeprowadzono dla w/w środowisk sprzętowych.

Rozdział czwarty zawiera analizę własności algorytmu FDTD w postaci macierzowej oraz porównanie ich z parametrami tego algorytmu w postaci jawnej.

W rozdziale piątym opisano metodę lokalnego zagęszczenia siatki dyskretyzacji w algorytmie FDTD oraz określono koszt numeryczny związany z jego wprowadzeniem do schematu różnicowego.

W rozdziale szóstym przedstawiono ideę wprowadzenia makromodeli do schematu różnicowego oraz przeprowadzono optymalizację tego algorytmu dla struktur zawierających powtarzające się elementy. Istotną część rozdziału stanowi ocena kosztów numerycznych wprowadzenia makromodeli do schematu różnicowego w części iteracyjnego algorytmu.

Rozdział siódmy zawiera zestawienie najbardziej efektywnych metod skrócenia czasu numerycznej analizy zagadnień elektromagnetycznych i porównanie ich efektywności obliczeń w symulacji działania rezonatora fotonicznego oraz anteny mikrofalowej. Przedstawiona w nim złożoność obliczeniowa algorytmu FDTD została zwiększona poprzez wprowadzenie dopasowanej warstwy absorpcyjnej na krańcach dziedziny obliczeniowej.

ROZDZIAŁ 2

Wprowadzenie do Metody Różnic Skończonych

W rozdziale tym scharakteryzowano zasadnicze cechy algorytmu różnic skończonych wprowadzając przy tej okazji podstawowe oznaczenia stosowane w dalszej części rozprawy. Zasygnalizowano tu również główne kierunki rozwoju nauki w zakresie elektrodynamiki obliczeniowej, które stanowiły wytyczne dla badań przedstawionych w dalszej części rozprawy.

2.1 Dyskretyzacja równań Maxwella

Równania Maxwella stanowią matematyczny zapis praw fizyki dotyczących zjawisk elektromagnetycznych. W tej rozprawie pełnią one centralną rolę i każdy aspekt dalszych rozważań opiera się na nich. Równania Maxwella zapisane w postaci różniczkowej dla ośrodka bezstratnego, anizotropowego, bez uwzględnienia źródeł, mają postać r. (2.1), [61].

$$\nabla \times \vec{H} = \frac{\partial}{\partial t} \overleftrightarrow{\epsilon} \vec{E} \tag{2.1a}$$

$$\nabla \times \vec{E} = -\frac{\partial}{\partial t} \overleftrightarrow{\mu} \vec{H} \tag{2.1b}$$

przy czym:
$\nabla \times \vec{A}$ - rotacja pola wektorowego $\vec{A}$, [115],
$\overleftrightarrow{\epsilon}$ - tensor przenikalności elektrycznej, [105],
$\overleftrightarrow{\mu}$ - tensor przenikalności magnetycznej, [105].

Do ich zapisu przyjęto kartezjański układ współrzędnych. W układzie tym pola magnetyczne oraz elektryczne są reprezentowane w formie zależnej od wektorów jednostkowych

$\vec{i}_x$, $\vec{i}_y$, $\vec{i}_z$, zgodnie z r. (2.2).

$$\vec{H} = \vec{i}_x H_x + \vec{i}_y H_y + \vec{i}_z H_z \tag{2.2a}$$

$$\vec{E} = \vec{i}_x E_x + \vec{i}_y E_y + \vec{i}_z E_z \tag{2.2b}$$

W dalszej części tego rozdziału omówiony zostanie sposób dyskretyzacji równań Maxwella dla współrzędnych kartezjańskich, który prowadzi do zdefiniowania algorytmu różnic skończonych. W celu uwydatnienia najistotniejszych cech wymaganych przekształceń zostaną one przeprowadzone dla ośrodka bezstratnego, z anizotropią reprezentowaną przez diagonalne tensory $\overleftrightarrow{\epsilon}$ i $\overleftrightarrow{\mu}$. Ponieważ ocena kosztów numerycznych analizy dla ośrodków bezstratnych jest dużo czytelniejsza niż dla ośrodków stratnych, dlatego w testach numerycznych zawartych w tej rozprawie nie wprowadzono strat ośrodków do analizy. Reguła ta nie dotyczy rozdziału 7, w którym ośrodek stratny występuje w warstwie PML stanowiącej warunek graniczny dziedziny obliczeniowej.

Zapis równań Maxwella w postaci r. (2.3), uzyskano poprzez wprowadzenie r. (2.2) do r. (2.1). W dalszej części rozdziału zostanie przedstawiony odpowiednik r. (2.3) w dziedzinie dyskretnej - r. (2.20). Parametry materiałowe analizowanego ośrodka są reprezentowane w r. (2.3) przez tensory diagonalne. Zapis taki wybrano w celu podkreślenia, że w algorytmie FDTD koszt numeryczny obliczeń dla ośrodka izotropowego oraz anizotropowego opisanego przez tensor diagonalny jest taki sam, co wynika z r. (2.11).

$$\begin{bmatrix} 0 & -\frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & -\frac{\partial}{\partial x} \\ -\frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} = \frac{\partial}{\partial t} \begin{bmatrix} \epsilon_x & 0 & 0 \\ 0 & \epsilon_y & 0 \\ 0 & 0 & \epsilon_z \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} \tag{2.3a}$$

$$\begin{bmatrix} 0 & -\frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & -\frac{\partial}{\partial x} \\ -\frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} = -\frac{\partial}{\partial t} \begin{bmatrix} \mu_x & 0 & 0 \\ 0 & \mu_y & 0 \\ 0 & 0 & \mu_z \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} \tag{2.3b}$$

Pierwszym etapem dyskretyzacji równań Maxwella jest dyskretyzacja dziedziny przestrzeni. Dyskretyzacja ta została przeprowadzana na podstawie równań Maxwella zapisanych w formie układu sześciu równań (2.4).

$$-\frac{\partial}{\partial z}H_y + \frac{\partial}{\partial y}H_z = \frac{\partial}{\partial t}\epsilon_x E_x \tag{2.4a}$$

$$\frac{\partial}{\partial z}H_x - \frac{\partial}{\partial x}H_z = \frac{\partial}{\partial t}\epsilon_y E_y \tag{2.4b}$$

$$-\frac{\partial}{\partial y}H_x + \frac{\partial}{\partial x}H_y = \frac{\partial}{\partial t}\epsilon_z E_z \tag{2.4c}$$

$$\frac{\partial}{\partial z}E_y - \frac{\partial}{\partial y}E_z = \frac{\partial}{\partial t}\mu_x H_x \tag{2.4d}$$

$$-\frac{\partial}{\partial z}E_x + \frac{\partial}{\partial x}E_z = \frac{\partial}{\partial t}\mu_y H_y \tag{2.4e}$$

$$\frac{\partial}{\partial y}E_x - \frac{\partial}{\partial x}E_y = \frac{\partial}{\partial t}\mu_z H_z \tag{2.4f}$$

Przejście z dziedziny ciągłej w równaniach Maxwella do dziedziny dyskretnej jest realizowane poprzez aproksymację pochodnych za pomocą metody różnic centralnych:

$$\left.\frac{\partial}{\partial x}f(x)\right|_{x=x_0} \approx \frac{f(x=x_0+\frac{1}{2}\Delta)-f(x=x_0-\frac{1}{2}\Delta)}{\Delta} \tag{2.5}$$

gdzie symbol Δ nazywany jest krokiem dyskretyzacji przestrzeni ciągłej.

W celu zwiększenia czytelności przeprowadzonych przekształceń proces dyskretyzacji równań Maxwella zostanie najpierw przedstawiony dla problemu jednowymiarowego.

2.1.1 Problem jednowymiarowy

W sytuacji jednowymiarowej, gdy $\frac{\partial}{\partial x} = 0$ i $\frac{\partial}{\partial z} = 0$, r. (2.4) przyjmują formę r. (2.6) (propagacja w kierunku osi y).

$$\frac{\partial}{\partial y}H_z = \frac{\partial}{\partial t}\epsilon_x E_x \tag{2.6a}$$

$$\frac{\partial}{\partial y}E_x = \frac{\partial}{\partial t}\mu_z H_z \tag{2.6b}$$

Zastosowanie przybliżenia z r. (2.5) w r. (2.6) prowadzi do:

$$\left.\frac{\partial}{\partial t}\mu_z(y)H_z(y)\right|_{y=y_0} = \left.\frac{\partial}{\partial y}E_x\right|_{y=y_0} \approx \frac{E_x(y=y_0+\frac{1}{2}\Delta)-E_x(y=y_0-\frac{1}{2}\Delta)}{\Delta} \tag{2.7}$$

Z relacji (2.7) wynika wzajemne położenie próbek pola elektrycznego i magnetycznego otrzymanych po dyskretyzacji: próbki pola magnetycznego zostały określone w punktach $y = y_0 + i\Delta$ dla $i \in \mathbb{Z}$ ($\mathbb{Z}$ - zbiór liczb całkowitych), natomiast próbki pola elektrycznego zostały określone w punktach $y = y_0 + (i+\frac{1}{2})\Delta$. Zatem wzajemne przesunięcie pomiędzy próbkami pola elektrycznego i magnetycznego jest równe $\frac{\Delta}{2}$, przy kroku dyskretyzacji równym Δ. Rozkład próbek pola w przestrzeni obrazuje rys. 2.1. W tym oraz w każdym kolejnym rysunku, który przedstawia próbki pól, kropką oznaczono punkt dyskretyzacji pola (interpretację fizyczną próbki pola podano w punkcie 2.1.5).

W rezultacie zastosowania dyskretyzacji składowe pola E_x oraz H_z są reprezentowane w dziedzinie dyskretnej przez wektory próbek tych pól, oznaczone odpowiednio jako $\mathbf{e}_x$ oraz $\mathbf{h}_z$. Elementy tych wektorów stanowią wartości próbek pól i są one oznaczane jako $E_x(j)$ oraz $H_z(j)$, przy czym dla kroku dyskretyzacji równego Δ są one zdefiniowane przez r. (2.8). Próbkowaniu zostają również poddane parametry materiałowe.

$$E_x(j) = E_x(y = y_j) \tag{2.8a}$$

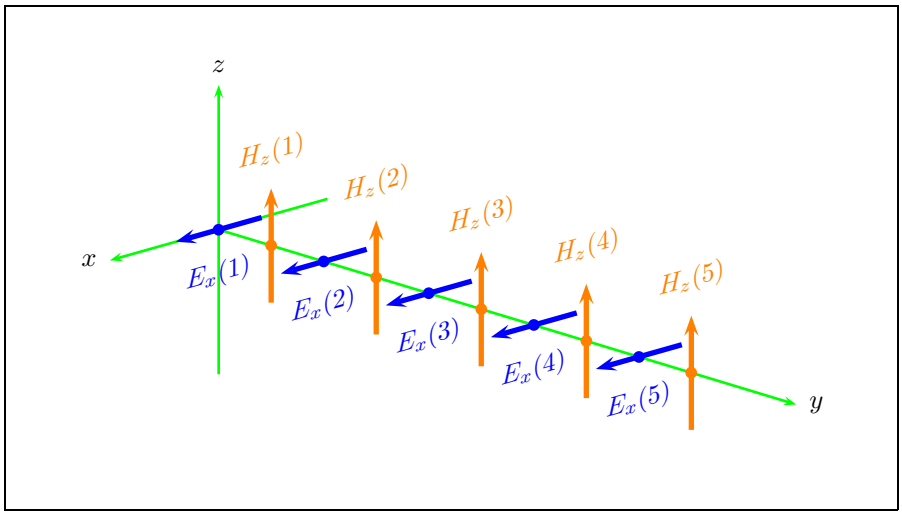
$$H_z(j) = H_z(y = y'_j) \tag{2.8b}$$

$$\epsilon_x(j) = \epsilon_x(y = y_j) \tag{2.8c}$$

$$\mu_z(j) = \mu_z(y = y'_j) \tag{2.8d}$$

$$y_j = j\Delta \tag{2.8e}$$

$$y'_j = j\Delta + \frac{\Delta}{2} \tag{2.8f}$$



Rysunek 2.1: Dyskretyzacja równań Maxwella. Problem jednowymiarowy.

Dla tak zdefiniowanych wektorów dyskretna postać r. (2.6) jest przedstawiona w r. (2.9).

$$H_z(j) - H_z(j-1) = \Delta \frac{\partial}{\partial t} \epsilon_x(j) E_x(j) \tag{2.9a}$$

$$E_x(j+1) - E_x(j) = \Delta \frac{\partial}{\partial t} \mu_z(j) H_z(j) \tag{2.9b}$$

Przedstawiona w tym punkcie koncepcja wzajemnych przesunięć pomiędzy próbkami pól stanowi podstawową zasadę konstrukcji siatki dyskretyzacji w metodzie różnic skończonych, zwaną siatką Yee, od nazwiska twórcy tej koncepcji [118]. Ponieważ problem jednowymiarowy nie jest w tej rozprawie stosowany w obliczeniach, dalsze wprowadzanie istotnych z punktu widzenia pracy pojęć, zostanie przeprowadzone dla przestrzeni trójwymiarowej.

2.1.2 Problem trójwymiarowy

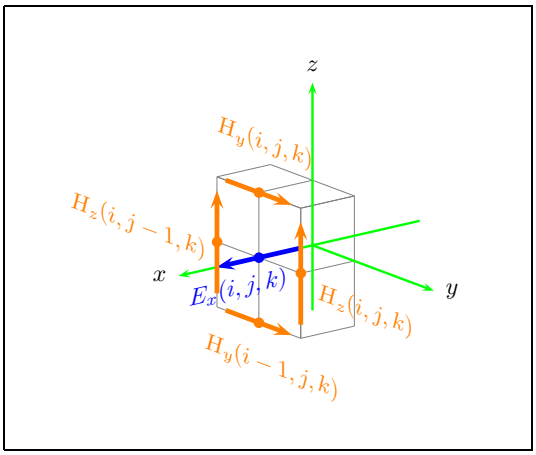
Analogicznie jak dla problemu jednowymiarowego, w procesie dyskretyzacji równań Maxwella definiowany jest wektor próbek pola elektrycznego **e** oraz magnetycznego **h**, który reprezentuje ciągłą postać wektorów $\vec{E}$ oraz $\vec{H}$. Elementy tych wektorów stanowią próbki pól oznaczane jako $A(i,j,k)$ $(A \in \{E_x, E_y, E_z, H_x, H_y, H_z\})$, przy czym ich położenie w przestrzeni trójwymiarowej jest określone przez r. (2.10). W zależnościach (2.10) i w dalszej części rozprawy przyjęto różny krok dyskretyzacji dla kierunków x, y, z przestrzeni, odpowiednio Δ_x, Δ_y, Δ_z.

Umiejscowienie próbek pola elektromagnetycznego w przestrzeni trójwymiarowej, które wynika z definicji różnic centralnych wyrażonej w r. (2.5), obrazowo przedstawiono na rys. 2.2. Na każdym z tych rysunków przedstawiono proces wyznaczania wartości próbki pola elektrycznego (magnetycznego) na podstawie sąsiednich próbek pola magnetycznego (elektrycznego). Algebraiczny opis równań Maxwella (2.4) z zastosowaną dyskretyzacją przestrzeni, który został przedstawiony na rys. 2.2, przyjmuje postać r. (2.11). Równania (2.11) stanowią podstawowe zależności stosowane w implementacji algorytmu różnic

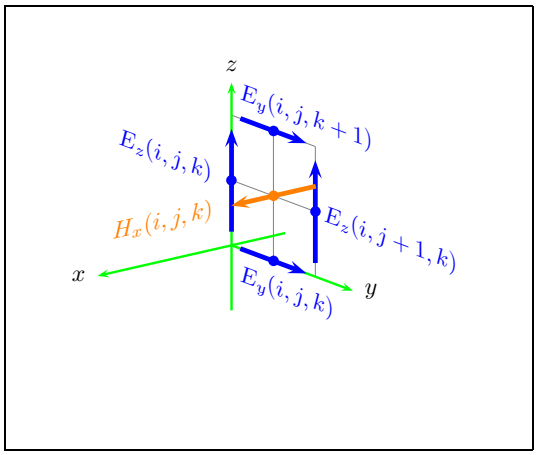
skończonych zdefiniowanego do analizy trójwymiarowych problemów elektromagnetycznych i określają punkt wyjściowy w dalszych rozważaniach zawartych w tej rozprawie.

$$E_x(i,j,k) = E_x(x = x'_i, y = y_j, z = z_k) \tag{2.10a}$$
$$E_y(i,j,k) = E_y(x = x_i, y = y'_j, z = z_k) \tag{2.10b}$$
$$E_z(i,j,k) = E_z(x = x_i, y = y_j, z = z'_k) \tag{2.10c}$$
$$H_x(i,j,k) = H_x(x = x_i, y = y'_j, z = z'_k) \tag{2.10d}$$
$$H_y(i,j,k) = H_y(x = x'_i, y = y_j, z = z'_k) \tag{2.10e}$$
$$H_z(i,j,k) = H_z(x = x'_i, y = y'_j, z = z_k) \tag{2.10f}$$
$$\epsilon_x(i,j,k) = \epsilon_x(x = x'_i, y = y_j, z = z_k) \tag{2.10g}$$
$$\epsilon_y(i,j,k) = \epsilon_y(x = x_i, y = y'_j, z = z_k) \tag{2.10h}$$
$$\epsilon_z(i,j,k) = \epsilon_z(x = x_i, y = y_j, z = z'_k) \tag{2.10i}$$
$$\mu_x(i,j,k) = \mu_x(x = x_i, y = y'_j, z = z'_k) \tag{2.10j}$$
$$\mu_y(i,j,k) = \mu_y(x = x'_i, y = y_j, z = z'_k) \tag{2.10k}$$
$$\mu_z(i,j,k) = \mu_z(x = x'_i, y = y'_j, z = z_k) \tag{2.10l}$$
$$x_i = i\Delta_x \tag{2.10m}$$
$$y_j = j\Delta_y \tag{2.10n}$$
$$z_k = k\Delta_z \tag{2.10o}$$
$$x'_i = i\Delta_x + \Delta_x/2 \tag{2.10p}$$
$$y'_j = j\Delta_y + \Delta_y/2 \tag{2.10q}$$
$$z'_k = k\Delta_z + \Delta_z/2 \tag{2.10r}$$

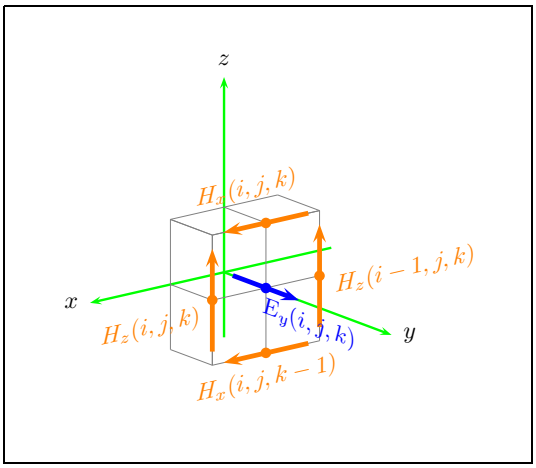
$$-\Delta_y\Big[H_y(i,j,k) - H_y(i,j,k-1)\Big] + \Delta_z\Big[H_z(i,j,k) - H_z(i,j-1,k)\Big] = \\ = \Delta_y\Delta_z\frac{\partial}{\partial t}\epsilon_x(i,j,k)E_x(i,j,k) \tag{2.11a}$$
$$\Delta_x\Big[H_x(i,j,k) - H_x(i,j,k-1)\Big] - \Delta_z\Big[H_z(i,j,k) - H_z(i-1,j,k)\Big] = \\ = \Delta_x\Delta_z\frac{\partial}{\partial t}\epsilon_y(i,j,k)E_y(i,j,k) \tag{2.11b}$$
$$-\Delta_x\Big[H_x(i,j,k) - H_x(i,j-1,k)\Big] + \Delta_y\Big[H_y(i,j,k) - H_y(i-1,j,k)\Big] = \\ = \Delta_x\Delta_y\frac{\partial}{\partial t}\epsilon_z(i,j,k)E_z(i,j,k) \tag{2.11c}$$
$$\Delta_y\Big[E_y(i,j,k+1) - E_y(i,j,k)\Big] - \Delta_z\Big[E_z(i,j+1,k) - E_z(i,j,k)\Big] = \\ = \Delta_y\Delta_z\frac{\partial}{\partial t}\mu_x(i,j,k)H_x(i,j,k) \tag{2.11d}$$
$$-\Delta_x\Big[E_x(i,j,k+1) - E_x(i,j,k)\Big] + \Delta_z\Big[E_z(i+1,j,k) - E_z(i,j,k)\Big] = \\ = \Delta_x\Delta_z\frac{\partial}{\partial t}\mu_y(i,j,k)H_y(i,j,k) \tag{2.11e}$$
$$\Delta_x\Big[E_x(i,j+1,k) - E_x(i,j,k)\Big] - \Delta_y\Big[E_y(i+1,j,k) - E_y(i,j,k)\Big] = \\ = \Delta_x\Delta_y\frac{\partial}{\partial t}\mu_z(i,j,k)H_z(i,j,k) \tag{2.11f}$$



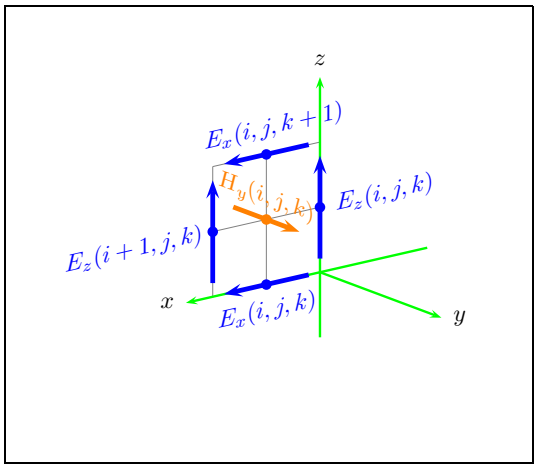
(a) Rotacja E_x. Ilustracja r. (2.11a).



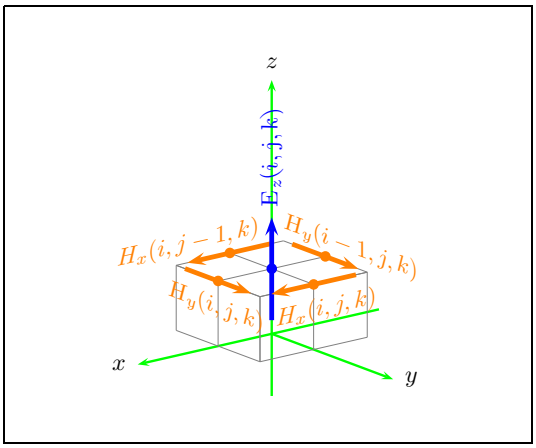
(b) Rotacja H_x. Ilustracja r. (2.11d).



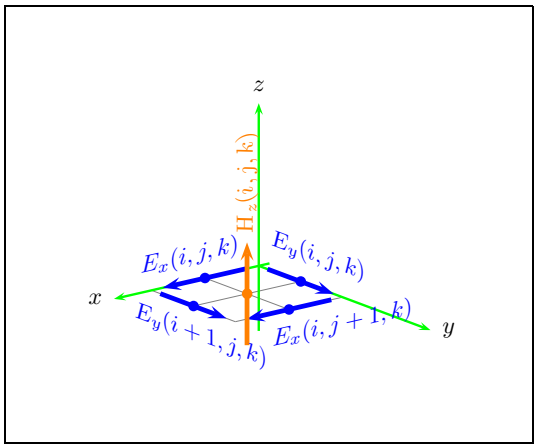
(c) Rotacja E_y. Ilustracja r. (2.11b).



(d) Rotacja H_y. Ilustracja r. (2.11e).



(e) Rotacja E_z. Ilustracja r. (2.11c).



(f) Rotacja H_z. Ilustracja r. (2.11f).

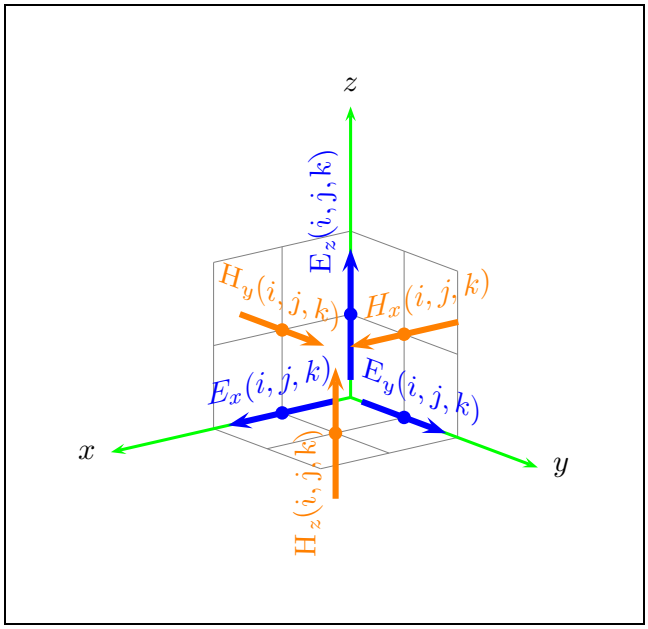
Rysunek 2.2: Ilustracja procesu wyznaczania rotacji pola elektromagnetycznego w siatce Yee dla pojedynczych próbek pól

Oznaczenie $\epsilon_x(x = x'_i, y = y_j, z = z_k)$ wskazuje na punkt próbkowania parametru materiałowego w przestrzeni. Reprezentowana przez to oznaczenie wartość może być równa dokładnie wartości parametru materiałowego w tym punkcie dziedziny ciągłej lub może zostać wyznaczona w sposób bardziej wyrafinowany, w celu dokładniejszego odzwierciedlenia parametrów materiałowych w okolicy punktu próbkowania, [28, 40, 120]. W obliczeniach opisanych w rozdziale 7 przyjęto, że wartość ta równa jest średniej wartości parametru materiałowego w obszarze prostopadłościanu o rozmiarach $[\Delta_x, \Delta_y, \Delta_z]$, którego centrum zgodne jest z oznaczeniami zastosowanymi w r. 2.11.

Podstawowy budulec siatki dyskretyzacji stanowi zbiór złożony z sześciu próbek pól o jednakowej numeracji:

$$\{E_x(i,j,k), E_y(i,j,k), E_z(i,j,k), H_x(i,j,k), H_y(i,j,k), H_z(i,j,k)\}.$$

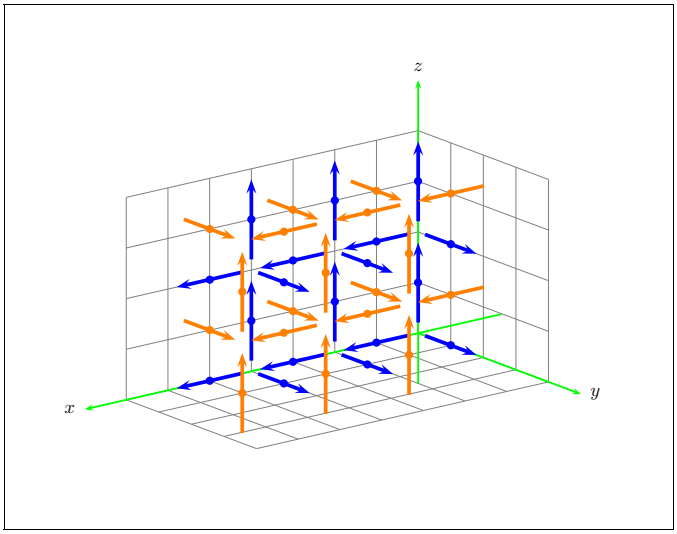
Graficzna reprezentacja tego zbioru nazywana jest komórką siatki dyskretyzacji lub komórką Yee. Została ona przedstawiona na rys. 2.3.



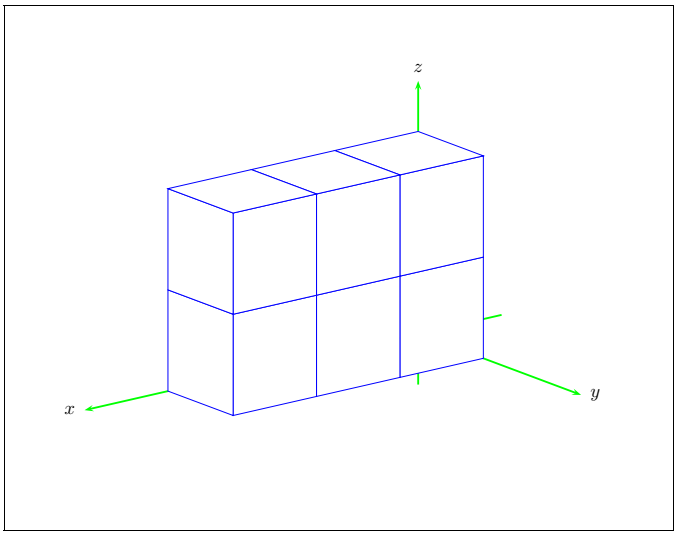
Rysunek 2.3: Pojedyncza komórka siatki Yee

Rozmiar siatki dyskretyzacji określany jest w oparciu o liczbę komórek Yee zdefiniowaną w kierunkach x, y, z przestrzeni. Przykład zobrazowania siatki dyskretyzacji o rozmiarze $3 \times 1 \times 2$ za pomocą graficznej reprezentacji komórek Yee przedstawia rys. 2.4. Taka ilustracja jest mało czytelna już dla małych rozmiarów siatki. Ponieważ w dalszej części rozprawy istotną rolę odgrywa wyodrębnienie z siatki dyskretyzacji większych podzbiorów komórek Yee pomocne jest jej zilustrowanie w prostszej formie. Taką uproszczoną wizualizację siatki dyskretyzacji reprezentuje rys. 2.5. Przedstawia on siatkę podstawową siatki dyskretyzacji. Przez siatkę podstawową rozumiany jest zbiór prostopadłościanów, których krawędzie zawierają punkty dyskretyzacji pola elektrycznego pochodzące z sąsiednich komórek Yee. Definiowana jest również siatka dualna, która stanowi zbiór prostopadłościanów, których krawędzie zawierają punkty dyskretyzacji pola magnetycznego

pochodzące z sąsiednich komórek Yee. Jednak w wielu sytuacjach opisanych w tej rozprawie wizualizacja siatki dualnej nie wniosłaby żadnych dodatkowych informacji i dlatego jest pomijana.



Rysunek 2.4: Siatka Yee dla obszaru dyskretyzacji równego $3 \times 1 \times 2$ komórek Yee



Rysunek 2.5: Siatka podstawowa dyskretyzacji dla obszaru dyskretyzacji równego $3 \times 1 \times 2$ komórek Yee

2.1.3 Notacje macierzowe i numeracja próbek pól

Jak stwierdzono w p. 2.1.2, w rezultacie zastosowania dyskretyzacji równań Maxwella, postać ciągła pola $\vec{E}$ i $\vec{H}$ jest zastąpiona przez zbiór próbek tych pól. Następnie zbiory próbek pól zostają uporządkowane w postaci wektorów $\mathbf{e}$ oraz $\mathbf{h}$. Warto przy tym podkreślić, że numeracja próbek pól może zostać przeprowadzona na wieloraki sposób z zachowaniem poprawności reprezentacji równań Maxwella w dziedzinie dyskretnej. Bardzo rzadko jest ona przedmiotem analizy osób stosujących metodę różnic skończonych w symulacjach,

jednak może ona mieć duży wpływ na efektywność obliczeń. Poprawa efektywności obliczeń otrzymana przez odpowiednie ułożenie próbek pól w wektorach **e**, **h**, która prowadzi do zdefiniowania kolejności działań podstawowych, stanowi istotną część tej rozprawy.

Najbardziej znanym sposobem numeracji próbek pól jest numeracja wprowadzona przez T. Weilanda, która została opisana w [113]. W numeracji tej wektory **e** i **h** zbudowane są z trzech podzbiorów próbek, które odpowiadają próbkom pól poszczególnych składowych, zgodnie z r. (2.12).

$$\mathbf{e} = \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \\ \mathbf{e}_z \end{bmatrix} \tag{2.12a}$$

$$\mathbf{h} = \begin{bmatrix} \mathbf{h}_x \\ \mathbf{h}_y \\ \mathbf{h}_z \end{bmatrix} \tag{2.12b}$$

Uporządkowanie wektora próbek składowej E_x zostało zdefiniowane w r. (2.13). Pozostałe wektory próbek składowych pól oraz parametrów materiałowych zostały zdefiniowane analogicznie do r. (2.13).

$$\mathbf{e}_x(n) = E_x(i,j,k) \tag{2.13a}$$

przy czym

$$n = i + j \cdot I + k \cdot I \cdot J \tag{2.13b}$$

$$0 \leq i < I \tag{2.13c}$$

$$0 \leq j < J \tag{2.13d}$$

$$0 \leq k < K \tag{2.13e}$$

$$0 \leq n < N \tag{2.13f}$$

$$N = IJK \tag{2.13g}$$

I – liczba komórek Yee dziedziny obliczeniowej w kierunku osi x

J – liczba komórek Yee dziedziny obliczeniowej w kierunku osi y

K – liczba komórek Yee dziedziny obliczeniowej w kierunku osi z

Zastosowanie takiej numeracji próbek pozwala na określenie przejrzystej formy macierzy rotacji pola elektrycznego i magnetycznego w postaci dyskretnej. Klarowne przedstawienie w wektorach **e** i **h** granic pomiędzy próbkami różnych składowych pozwala na zapis macierzy rotacji w formie z r. (2.14), co stanowi odpowiednik macierzy rotacji w dziedzinie ciągłej przestrzeni z r. (2.3b).

$$\mathbf{R}_E^{\textcircled{1}} = \begin{bmatrix} \mathbf{0} & -\mathbf{R}_z & \mathbf{R}_y \\ \mathbf{R}_z & \mathbf{0} & -\mathbf{R}_x \\ -\mathbf{R}_y & \mathbf{R}_x & \mathbf{0} \end{bmatrix} \tag{2.14}$$

Macierz $\mathbf{0}$ z r. (2.14) stanowi macierz o elementach zerowych i rozmiarze $N \times N$ (N zostało określone w r. (2.13g)). Macierze $\mathbf{R}_x$, $\mathbf{R}_y$ oraz $\mathbf{R}_z$, również o rozmiarze

$N \times N$, stanowią odpowiednik pochodnych odpowiednio $\frac{\partial}{\partial x}$, $\frac{\partial}{\partial y}$ i $\frac{\partial}{\partial z}$ w dziedzinie dyskretnej. Są one zdefiniowane w r. (2.15), a ich definicje wynikają z r. (2.11).

$$\mathbf{R}_x(p,q) = \begin{cases} 1 & \text{jeśli } p = q \\ -1 & \text{jeśli } (p = q+1) \wedge (i \neq I-1) \\ 0 & \text{w pozostałych sytuacjach} \end{cases} \tag{2.15a}$$

$$\mathbf{R}_y(p,q) = \begin{cases} 1 & \text{jeśli } p = q \\ -1 & \text{jeśli } (p = q+I) \wedge (j \neq J-1) \\ 0 & \text{w pozostałych sytuacjach} \end{cases} \tag{2.15b}$$

$$\mathbf{R}_z(p,q) = \begin{cases} 1 & \text{jeśli } p = q \\ -1 & \text{jeśli } (p = q+IJ) \wedge (k \neq K-1) \\ 0 & \text{w pozostałych sytuacjach} \end{cases} \tag{2.15c}$$

$$\text{dla} \quad p = 1 + i + jI + kIJ, \tag{2.15d}$$

$$0 \leq i < I, \tag{2.15e}$$

$$0 \leq j < J, \tag{2.15f}$$

$$0 \leq k < K \tag{2.15g}$$

Podstawowa wersja r. (2.15) została przedstawiona w [113]. Macierze z r. (2.15) uwzględniają warunki brzegowe na krańcach dziedziny obliczeniowej. Warunki dla krańców dziedziny obliczeniowej związane są z koniecznością uniknięcia sytuacji, w której wystąpiłoby numeryczne powiązanie próbek pól, które nie sąsiadują ze sobą w przestrzeni fizycznej. Taka sytuacja miałaby miejsce np. dla $i = I-1$, $j = 0$, $k = 0$. Wówczas $\mathbf{h}_x(I-1) = H_y(I-1,0,0)$ zależałoby od $\mathbf{e}_z(I-1) = E_z(I-1,0,0)$ oraz $\mathbf{e}_z(I) = E_z(0,1,0)$. Zależność taka byłaby reprezentowana w macierzy $\mathbf{R}_x$ poprzez wartości $\mathbf{R}_x(I-1,I-1) = 1$ oraz $\mathbf{R}_x(I-1,I) = -1$, co stanowiłoby zależność niefizyczną, a przez to, z punktu widzenia procesu dyskretyzacji, błędną. Warunek $i \neq (I-1)$, który na krańcu dziedziny obliczeniowej wprowadza ściankę elektryczną, stanowi zatem istotną część definicji dyskretnej postaci pochodnej $\frac{\partial}{\partial x}$.

Warto zauważyć, że zarówno wektor $\mathbf{e}$ jak i $\mathbf{h}$ można dowolnie spermutować, a całkowity opis tej operacji można zawrzeć w macierzach permutacji odpowiednio $\mathbf{P}_{pE}$ oraz $\mathbf{P}_{pH}$. Wówczas algorytm nadal będzie działał poprawnie (zależność pomiędzy próbkami pól zostanie zachowana), jednak efektywność obliczeń może ulec zmianie. W następnych rozdziałach zostanie przeprowadzona analiza wpływu niektórych sposobów permutacji na efektywność obliczeń. Istotne jest stwierdzenie, że dla standardowej analizy metodą różnic skończonych macierz rotacji pola elektrycznego w dziedzinie dyskretnej zawsze przyjmie formę $\mathbf{P}_{pE}^T \mathbf{R}_E \mathbf{P}_{pH}$.

Macierz rotacji pola elektrycznego w dziedzinie dyskretnej z uwzględnieniem kroku dyskretyzacji oznaczona jest w tej rozprawie przez $\mathbf{R}_E$ i zdefiniowana przez r. (2.16). Macierze $\mathbf{D}_S$ i $\mathbf{D}_A$ wynikają z przyjętego kroku dyskretyzacji, który może mieć różną wartość dla każdego z kierunków przestrzeni x, y, z. Macierze jednostkowe, $\mathbf{I}$, w r. (2.17)

oraz r. (2.18) mają rozmiar $N \times N$.

$$\mathbf{R}_E = \mathbf{D}_A^{-1}\mathbf{R}_E^{①}\mathbf{D}_S \tag{2.16}$$

$$\mathbf{D}_S = \begin{bmatrix} \Delta_x\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Delta_y\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \Delta_z\mathbf{I} \end{bmatrix} \tag{2.17}$$

$$\mathbf{D}_A = \begin{bmatrix} \Delta_y\Delta_z\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \Delta_x\Delta_z\mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \Delta_x\Delta_y\mathbf{I} \end{bmatrix} \tag{2.18}$$

Jak wykazano w [113] macierz rotacji pola magnetycznego w postaci dyskretnej $\mathbf{R}_H$ spełnia zależność (2.19).

$$\mathbf{R}_H = \mathbf{D}_A^{-1}\left(\mathbf{R}_E^{①}\right)^T\mathbf{D}_S \tag{2.19}$$

Zdefiniowanie macierzy rotacji $\mathbf{R}_E$ oraz $\mathbf{R}_H$ pozwala na zapis równań Maxwella, po przeprowadzeniu dyskretyzacji dziedziny przestrzeni, w formie macierzowej r. (2.20). Równania (2.20) nazywane są siatkowymi (dyskretnymi) równaniami Maxwella [105].

$$\frac{\partial}{\partial t}\mathbf{e} = \mathbf{D}_\epsilon^{-1}\mathbf{R}_H\mathbf{h} \tag{2.20a}$$

$$\frac{\partial}{\partial t}\mathbf{h} = -\mathbf{D}_\mu^{-1}\mathbf{R}_E\mathbf{e} \tag{2.20b}$$

Macierze $\mathbf{D}_\epsilon$ oraz $\mathbf{D}_\mu$ reprezentują parametry materiałowe obszaru poddanego dyskretyzacji. Są to macierze diagonalne, które dla sposobu numeracji przyjętego przez T. Weilanda są zdefiniowane zgodnie z r. (2.21).

$$\mathbf{D}_\epsilon = \begin{bmatrix} \mathbf{D}_{x\epsilon} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_{y\epsilon} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_{z\epsilon} \end{bmatrix} \tag{2.21a}$$

$$\mathbf{D}_\mu = \begin{bmatrix} \mathbf{D}_{x\mu} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_{y\mu} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{D}_{z\mu} \end{bmatrix} \tag{2.21b}$$

$$\mathbf{D}_{x\epsilon}(p,p) = \epsilon_x(i,j,k) \tag{2.21c}$$

$$\mathbf{D}_{y\epsilon}(p,p) = \epsilon_y(i,j,k) \tag{2.21d}$$

$$\mathbf{D}_{z\epsilon}(p,p) = \epsilon_z(i,j,k) \tag{2.21e}$$

$$\mathbf{D}_{x\mu}(p,p) = \mu_x(i,j,k) \tag{2.21f}$$

$$\mathbf{D}_{y\mu}(p,p) = \mu_y(i,j,k) \tag{2.21g}$$

$$\mathbf{D}_{z\mu}(p,p) = \mu_z(i,j,k) \tag{2.21h}$$

$$p = i + jI + kIJ \tag{2.21i}$$

2.1.4 Dyskretyzacja dziedziny czasu

Wyróżnia się dwa zasadnicze rodzaje metody różnic skończonych, które są stosowane do rozwiązywania problemów elektromagnetycznych. Podział ten wynika ze sposobu traktowania dziedziny czasu w równaniach Maxwella:

- metoda różnic skończonych w dziedzinie czasu, znana jako FDTD (ang. *Finite Difference Time Domain*),
- metoda różnic skończonych w dziedzinie częstotliwości. Obejmuje ona zarówno FDFD (ang. *Finite Difference Frequency Domain*) jak i inne metody (m.in. rozwiązywania zagadnień własnych), przy czym za każdym razem opiera się na transformacji dziedziny czasu do dziedziny Laplace'a.

W niniejszej rozprawie rozpatrywane są zagadnienia związane tylko z analizą FDTD i przede wszystkim w kontekście badania sposobów zwiększenia jej efektywności obliczeń.

Dyskretyzacja dziedziny czasu, podobnie jak dyskretyzacja przestrzeni, opiera się na różnicach centralnych z r. (2.5). Analogicznie jak w dziedzinie przestrzeni, również w dziedzinie czasu występuje przesunięcie pomiędzy próbkami pola elektrycznego i magnetycznego, przy czym jest ono równe $\frac{\Delta_t}{2}$, gdzie Δ_t oznacza krok dyskretyzacji dziedziny czasu. Przesunięcie to wynika z bezpośredniego zastosowania r. (2.5) w r. (2.20), co zostało przedstawione w r. (2.22).

$$\frac{\mathbf{e}^{n+1} - \mathbf{e}^n}{\Delta_t} = \mathbf{D}_\epsilon^{-1}\mathbf{R}_H\mathbf{h}^{n+0,5} \tag{2.22a}$$

$$\frac{\mathbf{h}^{n+1,5} - \mathbf{h}^{n+0,5}}{\Delta_t} = \mathbf{D}_\mu^{-1}\mathbf{R}_E\mathbf{e}^{n+1} \tag{2.22b}$$

Oznaczenie $\mathbf{e}^n$ reprezentuje wektor próbek pola $\vec{E}$ w czasie t równym $t = n\ \Delta_t$. Analogicznie wektor próbek pola $\vec{H}$ w czasie $t = (n + 0,5)\ \Delta_t$ oznaczany jest w tej rozprawie przez $\mathbf{h}^{n+0,5}$.

Uporządkowanie r. (2.22) prowadzi do określenia algorytmu FDTD w postaci macierzowej:

$$\mathbf{e}^{n+1} = \mathbf{e}^n + \Delta_t\mathbf{D}_\epsilon^{-1}\mathbf{R}_H\mathbf{h}^{n+0,5} \tag{2.23a}$$

$$\mathbf{h}^{n+1,5} = \mathbf{h}^{n+0,5} + \Delta_t\mathbf{D}_\mu^{-1}\mathbf{R}_E\mathbf{e}^{n+1} \tag{2.23b}$$

Postać macierzowa jest bardzo użyteczna z punktu widzenia analizy własności metody, jednak w literaturze najczęściej stosuje się postać jawną algorytmu FDTD, którą otrzymuje się poprzez dyskretyzację dziedziny czasu w r. (2.11). Pośrednie przekształcenia można znaleźć w [105].

$$E_x(i,j,k)^{n+1} = E_x(i,j,k)^n + \frac{\Delta_t}{\epsilon_x(i,j,k)\Delta_y\Delta_z}\Big\{ \\ - \Delta_y\Big[H_y(i,j,k)^{n+0,5} - H_y(i,j,k-1)^{n+0,5}\Big] + \Delta_z\Big[H_z(i,j,k)^{n+0,5} - H_z(i,j-1,k)^{n+0,5}\Big]\Big\} \tag{2.24a}$$

$$E_y(i,j,k)^{n+1} = E_y(i,j,k)^n + \frac{\Delta_t}{\epsilon_y(i,j,k)\Delta_x\Delta_z}\Big\{ \Delta_x\Big[H_x(i,j,k)^{n+0,5} - H_x(i,j,k-1)^{n+0,5}\Big] - \Delta_z\Big[H_z(i,j,k)^{n+0,5} - H_z(i-1,j,k)^{n+0,5}\Big]\Big\} \tag{2.24b}$$

$$E_z(i,j,k)^{n+1} = E_z(i,j,k)^n + \frac{\Delta_t}{\epsilon_z(i,j,k)\Delta_x\Delta_y}\Big\{ -\Delta_x\Big[H_x(i,j,k)^{n+0,5} - H_x(i,j-1,k)^{n+0,5}\Big] + \Delta_y\Big[H_y(i,j,k)^{n+0,5} - H_y(i-1,j,k)^{n+0,5}\Big]\Big\} \tag{2.24c}$$

$$H_x(i,j,k)^{n+1,5} = H_x(i,j,k)^{n+0,5} + \frac{\Delta_t}{\mu_x(i,j,k)\Delta_y\Delta_z}\Big\{ \Delta_y\Big[E_y(i,j,k+1)^{n+1} - E_y(i,j,k)^{n+1}\Big] - \Delta_z\Big[E_z(i,j+1,k)^{n+1} - E_z(i,j,k)^{n+1}\Big]\Big\} \tag{2.24d}$$

$$H_y(i,j,k)^{n+1,5} = H_y(i,j,k)^{n+0,5} + \frac{\Delta_t}{\mu_y(i,j,k)\Delta_x\Delta_z}\Big\{ -\Delta_x\Big[E_x(i,j,k+1)^{n+1} - E_x(i,j,k)^{n+1}\Big] + \Delta_z\Big[E_z(i+1,j,k)^{n+1} - E_z(i,j,k)^{n+1}\Big]\Big\} \tag{2.24e}$$

$$H_z(i,j,k)^{n+1,5} = H_z(i,j,k)^{n+0,5} + \frac{\Delta_t}{\mu_z(i,j,k)\Delta_x\Delta_y}\Big\{ \Delta_x\Big[E_x(i,j+1,k)^{n+1} - E_x(i,j,k)^{n+1}\Big] - \Delta_y\Big[E_y(i+1,j,k)^{n+1} - E_y(i,j,k)^{n+1}\Big]\Big\} \tag{2.24f}$$

2.1.5 Interpretacja fizyczna procesu dyskretyzacji pól

W literaturze spotykane są wielorakie interpretacje fizyczne wielkości wchodzących w skład schematu różnicowego przedstawionego w r. (2.11). Jeśli schemat różnicowy wyrażony przez r. (2.11) został otrzymany z zastosowaniem różnic centralnych z r. (2.5), to próbkę pola w dziedzinie dyskretnej można określić jako wartość pola elektrycznego bądź magnetycznego dokładnie w punkcie dyskretyzacji.

Jednak schemat różnicowy można otrzymać na wiele sposobów, nie tylko za pomocą różnic centralnych. Można go także otrzymać stosując np. metodę momentów wobec równań Maxwella przy zastosowaniu falek Haara jako funkcji bazowych [73]. Wówczas próbki pola tworzące wektory **e** oraz **h** mogą zostać zinterpretowane jako amplitudy falki Haara. Zgodnie z tym rozumowaniem, przeskalowana falka Haara reprezentuje kształt pola w schemacie różnicowym wokół punktu w przestrzeni, dla którego została określona dyskretna wartość pola.

Trzecim znaczącym sposobem otrzymania schematu różnic skończonych jest metoda całek skończonych FIT (ang. *Finite Integration Technique*) zdefiniowana przez Weilanda. Polega ona na dyskretyzacji równań Maxwella w postaci całkowej, wówczas próbka pola zdefiniowana jest jako całka po drodze równej krokowi dyskretyzacji,
np. $\hat{e}_x(i,j,k) = \int_{(x_i,y_j,z_k)}^{(x_{i+1},y_j,z_k)} \vec{E}d\vec{S}$, [19]. Jednak metoda ta jest tożsama z metodą różnic skończonych, co wykazano w [43].

Połączenie interpretacji całkowej oraz różnicowej przedstawił dr Przybyszewski w rozprawie [81]. W r. (2.25) przedstawiono interpretację całkową próbki pola w problemie jednowymiarowym, zgodnie z relacją (2.2.11) zaprezentowaną w [81]. Przedstawione równanie pozwala na zinterpretowanie schematu różnicowego jako działania na uśrednionych wartościach pochodnej pola elektromagnetycznego, przy czym w problemie trójwymiarowym uśrednianie to zdefiniowane jest w obszarze prostopadłościanów o rozmiarach $\Delta_x \times \Delta_y \times \Delta_z$.

$$\frac{1}{\Delta_y}\int_{y=y'-\frac{1}{2}\Delta_y}^{y=y'+\frac{1}{2}\Delta_y}\frac{\partial}{\partial y}E_x dy = \frac{E_x(y=y'-\frac{1}{2}\Delta_y') - E_x(y=y'+\frac{1}{2}\Delta_y)}{\Delta_y} \tag{2.25}$$

W tej rozprawie interpretacja fizycznego znaczenia próbki pola jest istotna przy określaniu relacji pomiędzy obszarami schematu różnicowego, w których zdefiniowane różne wartości kroku dyskretyzacji. Taka sytuacja ma miejsce przy definiowaniu metody lokalnego zagęszczenia siatki opisanej w rozdziale 5. Wówczas próbka pola jest rozważana zgodnie z metodą zaproponowaną przez dr. Przybyszewskiego.

2.1.6 Symetryzacja schematu różnicowego

Symetryzacja schematu różnicowego stosowana jest wobec jego macierzowego sformułowania. Zaletą przeprowadzenia symetryzacji jest uzyskanie dodatkowych własności zapisu macierzowego, z których najważniejsze wyglądają następująco:

- Po przeprowadzeniu symetryzacji występuje istotna relacja pomiędzy symetryczną postacią macierzy rotacji pola elektrycznego $\tilde{\mathbf{R}}_E$ oraz magnetycznego $\tilde{\mathbf{R}}_H$: $\tilde{\mathbf{R}}_E = \tilde{\mathbf{R}}_H^T$. Relacja ta stanowi gwarancję stabilności schematu różnicowego, co jest szczególnie istotne jeśli schemat ten poddawany jest modyfikacjom.
- Sformułowanie problemu częstotliwościowego zawiera macierz $\tilde{\mathbf{R}}_E\tilde{\mathbf{R}}_H$, która jest symetryczna. Taka własność pozwala na szybsze przeprowadzenie niektórych działań, np. wyznaczenia wartości własnych macierzy.

Przeprowadzenie symetryzacji schematu różnicowego zapisanego w formie macierzowej sprowadza się do zastosowania działań opisanych r. (2.26a) ÷ (2.26d).

$$\tilde{\mathbf{e}} = \mathbf{D}_\epsilon^{\frac{1}{2}}\mathbf{e} \tag{2.26a}$$

$$\tilde{\mathbf{h}} = \mathbf{D}_\mu^{\frac{1}{2}}\mathbf{h} \tag{2.26b}$$

$$\tilde{\mathbf{R}}_E = \mathbf{D}_\mu^{\frac{1}{2}}\mathbf{R}_E\mathbf{D}_\epsilon^{\frac{1}{2}} \tag{2.26c}$$

$$\tilde{\mathbf{R}}_H = \mathbf{D}_\epsilon^{\frac{1}{2}}\mathbf{R}_H\mathbf{D}_\mu^{\frac{1}{2}} \tag{2.26d}$$

Wówczas zdyskretyzowane równania Maxwella zapisane w formie macierzowej, po zastosowaniu symetryzacji macierzy rotacji, opisane są przez r. (2.27).

$$\frac{\partial}{\partial t}\tilde{\mathbf{e}} = \tilde{\mathbf{R}}_H\tilde{\mathbf{h}} \tag{2.27a}$$

$$\frac{\partial}{\partial t}\tilde{\mathbf{h}} = -\tilde{\mathbf{R}}_E\tilde{\mathbf{e}} \tag{2.27b}$$

Ponieważ $\tilde{\mathbf{R}}_E = \tilde{\mathbf{R}}_H^T$, to do reprezentacji równań Maxwella w dziedzinie dyskretnej wymagana jest tylko jedna macierz rotacji w dziedzinie dyskretnej, co korzystnie odróżnia ten zapis od macierzowego sformułowania równań Maxwella bez symetryzacji. Ponadto znormalizowane wektory próbek pól $\mathbf{e}$ i $\mathbf{h}$ poprzez zastosowanie symetryzacji posiadają wartości tego samego rzędu. Stosowanie symetrycznego zapisu jest wysoce wskazane przy obliczeniach z makromodelami włączonymi do schematu różnicowego (rozdział 6), gdyż bez tej operacji schemat ten jest niestabilny, co szczególnie silnie jest widoczne przy obliczeniach wykonywanych na zmiennych o pojedynczej precyzji.

ROZDZIAŁ 3

Algorytm FDTD w postaci jawnej

W tym rozdziale omówiono cechy implementacji[1] algorytmu FDTD w postaci jawnej, które decydują o czasie numerycznej symulacji propagacji fali elektromagnetycznej przeprowadzonej za pomocą tego algorytmu. Czas symulacji jest oczywiście ściśle związany z urządzeniami, które wykonują obliczenia i dlatego przedstawione zostały tu również podstawowe cechy architektury następujących urządzeń zastosowanych do obliczeń:

- procesor komputerowy, CPU (ang. *Central Processing Unit*), (w sytuacji procesora wielordzeniowego, do obliczeń wykorzystany zostanie jeden rdzeń),
- procesor wielordzeniowy w sytuacji, gdy obliczenia przeprowadzane są na wielu rdzeniach,
- środowisko wieloprocesorowe,
- karta graficzna, która zawiera akcelerator graficzny, GPU (ang. *Graphics Processing Unit*), zgodny z technologią CUDA (technologia CUDA została opisana w p. 3.5).

Jednym z celów tego rozdziału jest przeanalizowanie wpływu własności wymienionych urządzeń na czas obliczeń. Omówione własności są następnie uwzględnione w procesie optymalizacji implementacji algorytmu FDTD dla każdej z wymienionych architektur z osobna, tak aby doprowadzić do minimalizacji czasu symulacji. Ponieważ w tym i kolejnych rozdziałach analizie poddano czynniki, które determinują czas obliczeń, to jako strukturę referencyjną, dla której zostały przeprowadzone symulacje propagacji fali, wybrano rezonator prostopadłościenny. Wybór prostej struktury ma na celu skupienie uwagi na najważniejszych własnościach algorytmu FDTD oraz cechach urządzeń, które mają decydujący wpływ na czas symulacji. Również z tego powodu wybrano wersję algorytmu FDTD, która nie uwzględnia strat materiałowych i zawiera informację o parametrach

[1] Implementacja algorytmu oznacza kod programu, który realizuje działania wskazane przez algorytm

materiałowych $\overleftrightarrow{\epsilon}$ oraz $\overleftrightarrow{\mu}$. Analiza złożonej struktury, w której wprowadzono straty materiałowe w warstwie absorbującej falę elektromagnetyczną i która ilustruje przedstawione w tym punkcie wnioski w szerszym kontekście, została opisana w rozdziale 7.

3.1 Jawne sformułowanie problemu

Jawna postać algorytmu FDTD stanowi implementację r. (2.24). Przykład jednowątkowej implementacji algorytmu FDTD przeznaczonej dla CPU, zrealizowanej w języku programowania C, przedstawia wydruk 3.1.

Kod przedstawiony na wydruku 3.1 posłuży do wskazania różnic przy programowaniu z przeznaczeniem dla GPU w odniesieniu do CPU i z tego względu zostanie on tu dokładnie omówiony. Przedstawiony kod programu stanowi pełną implementację algorytmu FDTD, jednak zostały pominięte tu kwestie formowania impulsu pobudzającego, które określają zasady ustalania wartości elementów wektora `src` oraz analizy rezultatów symulacji, czyli wartości elementów wektora `sep`.

Podstawowe cechy implementacji algorytmu FDTD przedstawionej na wydruku 3.1:

1. Algorytm stanowi proces iteracyjny, którego liczbę iteracji ustala zmienna `iterNo` (linia 1).

2. Zmienne stanu są umieszczone w sześciu tablicach: dla pola magnetycznego w `hx`, `hy`, `hz` oraz dla pola elektrycznego w `ex`, `ey`, `ez`. Wymiar każdej z tych tablic wynosi N.

3. Pobudzenie miękkie [59, 105] jest zrealizowane w liniach 3 oraz 4. Polega ono na dodaniu w każdej iteracji wartości przechowywanej w `i`-tym elemencie wektora `src` do `psrcLen` elementów wektora `ex`, których indeksy są określone przez wektor `psrc`. Wybrany rodzaj pobudzenia reprezentuje jedną z wielu możliwości, bowiem pobudzenie może przyjąć bardziej rozbudowaną formę [105] i obejmować również pole magnetyczne.

4. Wyznaczenie nowych wartości próbek pola elektrycznego odbywa się w liniach $7 \div 23$. Wartości zastosowanych zmiennych ustalone są poprzez zależności:

```
dzy = dz / dy;
dzx = dz / dx;
dyx = dy / dx;
cex[ i ] = dt / epsx[ i ] / dz;
cey[ i ] = dt / epsy[ i ] / dz;
cez[ i ] = dt / epsz[ i ] / dy;
chx[ i ] = dt / mux[ i ] / dz;
chy[ i ] = dt / muy[ i ] / dz;
chz[ i ] = dt / muz[ i ] / dy;
```

gdzie:
`dt` - krok dyskretyzacji dziedziny czasu,
`dx`, `dy`, `dz` - krok dyskretyzacji dziedziny przestrzeni określony odpowiednio dla kierunków x, y, z,
`epsx[ i ]`, `epsy[ i ]`, `epsz[ i ]` - przenikalność elektryczna bezwzględna określona

```
1  for( it = 0; it < iterNo; it++ ){
2    // pobudzenie miekkie
3    for( xx=0; xx<psrcLen; xx++ )
4      ex[ psrc[ xx ] ] += src[ it ];
5
6    // wyznaczenie wartosci skladowych pola e
7    for( pp=IJ+II, zz=1; zz<KK-1; zz++, pp+=2*II )
8      for( yy=1; yy<JJ-1; yy++, pp++ )
9        for( xx = 0; xx<II-1; xx++, pp++ )
10         ex[ pp ] += cex[ pp ] * (         - hy[ pp ] + hy[ pp - IJ ]   +
11                                  dzy * (   hz[ pp ] - hz[ pp - II ] ) );
12
13   for( pp=IJ, zz=1; zz<KK-1; zz++, pp+=II )
14     for( yy=0; yy<JJ-1; yy++, pp++ )
15       for( xx=1, pp++; xx<II-1; xx++, pp++ )
16         ey[ pp ] += cey[ pp ] * (           hx[ pp ] - hx[ pp - IJ ]   +
17                                  dzx * ( - hz[ pp ] + hz[ pp -  1 ] ) );
18
19   for( pp=0, zz=0; zz<KK-1; zz++, pp+=II )
20     for( yy=1, pp+=II; yy<JJ-1; yy++, pp++ )
21       for( xx=1, pp++; xx<II-1; xx++, pp++ )
22         ez[ pp ] += cez[ pp ] * (         - hx[ pp ] + hx[ pp - II ]   +
23                                  dyx * (   hy[ pp ] - hy[ pp -  1 ] ) );
24
25   // wyznaczenie wartosci skladowych pola h
26   for( pp=0, zz=0; zz<KK-1; zz++, pp+=II )
27     for( yy=0; yy<JJ-1; yy++, pp++ )
28       for( xx=1, pp++; xx<II-1; xx++, pp++ )
29         hx[ pp ] -= chx[ pp ] * (       (   ey[ pp ] - ey[ pp + IJ ] ) +
30                                  dzy * ( - ez[ pp ] + ez[ pp + II ] ) );
31
32   for( pp=0, zz=0; zz<KK-1; zz++, pp+=II )
33     for( yy=1, pp+=II; yy<JJ-1; yy++, pp++ )
34       for( xx=0; xx<II-1; xx++, pp++ )
35         hy[ pp ] -= chy[ pp ] * (       ( - ex[ pp ] + ex[ pp + IJ ] ) +
36                                  dzx * (   ez[ pp ] - ez[ pp +  1 ] ) );
37
38   for( pp=IJ, zz=1; zz<KK-1; zz++, pp+=II )
39     for( yy=0; yy<JJ-1; yy++, pp++ )
40       for( xx=0; xx<II-1; xx++, pp++ )
41         hz[ pp ] -= chz[ pp ] * (       (   ex[ pp ] - ex[ pp + II ] ) +
42                                  dyx * ( - ey[ pp ] + ey[ pp +  1 ] ) );
43
44   // pobranie wartosci pola elektrycznego
45   for( xx=0; xx<psepLen; xx++ )
46     sep[ it ] += ex[ psep[ xx ] ];
47 }
```

Wydruk 3.1: Jednowątkowa implementacja jawna algorytmu FDTD przeznaczona dla CPU

dla składowej odpowiednio E_x, E_y oraz E_z w i-tej komórce Yee (numeracja komórek została omówiona w p. 2.1.3),
`mux[ i ]`, `muy[ i ]`, `muz[ i ]` - przenikalność magnetyczna bezwzględna określona dla składowej odpowiednio H_x, H_y oraz H_z w i-tej komórce Yee.

5. Wyznaczenie nowych wartości próbek pola magnetycznego odbywa się w liniach $26 \div 42$.

6. Pobranie rezultatów symulacji i umieszczenie ich w wektorze wynikowym `sep` ma miejsce w liniach $45 \div 46$. Zrealizowany wariant pobierania odpowiedzi sprowadza się do zsumowanie `psepLen` wartości próbek pola elektrycznego, których indeksy wskazuje wektor `psep`.

3.1.1 Ścianka elektryczna jako warunek brzegowy schematu różnicowego

Zależności pomiędzy próbkami pola zawarte w algorytmie FDTD wymagają zdefiniowania sposobu postępowania z próbkami znajdującymi się na krańcach dziedziny obliczeniowej. Przykładowo przy wyznaczeniu wartości próbki pola $Ex(x = 0, y = 0, z = 0)$ algorytm wymaga znajomości wartości próbki pola $Hz(x = 0, y = -1, z = 0)$ reprezentowanej przez `hz[ pp - II ]` na wydruku 3.1. Oczywiście próbka taka nie istnieje, więc wymagane jest szczególne zachowanie algorytmu w takiej sytuacji. Jedną z możliwości jest wymuszenie ścianki elektrycznej na krańcach dziedziny obliczeniowej. Do realizacji ścianki elektrycznej można posłużyć się jedną z dwóch podstawowych metod:

- wariant A - wyzerowanie dla $x = I$, $y = J$, $z = K$ wszystkich wartości próbek pól (wariant ten został zrealizowany w kodzie przedstawionym na wydruku 3.1),
- wariant B - zastosowanie odmiennego schematu obliczeń na krańcach dziedziny obliczeniowej (fragment tego rozwiązania został przedstawiony na wydruku 3.2).

Oba warianty działają analogicznie dla próbek pól znajdujących się na płaszczyznach $x = 0$, $y = 0$, $z = 0$: wartości próbek składowych pola elektrycznego stycznych do danej płaszczyzny oraz składowych pola magnetycznego prostopadłych do danej płaszczyzny nie są aktualizowane, czyli zachowują zawsze wartość równą zero, która została nadana wszystkim próbkom pól przed rozpoczęciem procesu iteracyjnego. Ten prosty warunek prowadzi do implementacji ścianek elektrycznych we wskazanych płaszczyznach.

Odmienna sytuacja występuje na płaszczyznach $x = I$, $y = J$, $z = K$. Algorytm w wersji A nie aktualizuje żadnych wartości próbek pola elektrycznego oraz magnetycznego w tych płaszczyznach, a przez to wszystkie wskazane wartości są równe zeru. Zawarte w algorytmie FDTD zależności prowadzą do tego, iż w takiej sytuacji każde odwołanie się do próbek pola elektrycznego o zerowych wartościach, które należą do tych płaszczyzn, np. przez zapis `ez[ pp + 1 ]`, ma miejsce, gdy składowe te są styczne do granicy dziedziny. W rezultacie algorytm implementuje w ten sposób ściankę elektryczną. Wadą tego rozwiązania jest przechowywanie w pamięci znacznej liczby próbek pól, których wartości w każdej iteracji równe są zero.

Natomiast w wariancie B algorytm wyróżnia z obliczeń wartości próbek pola magnetycznego z płaszczyzn $x = I$, $y = J$, $z = K$ i stosuje dla nich zmodyfikowaną postać zależności, która zakłada zerową wartość próbek pola elektrycznego znajdujących

```
1  for( zz = 0, pp = 0; zz < KK - 1; zz++ ){
2    for( yy = 0; yy < JJ - 1; yy++ )
3      for( xx = 1, pp++; xx < II; xx++, pp++ )
4        hx[ pp ] -= chx[ pp ] * (          ey[ pp ] - ey[ pp + IJ ] +
5                                 dzy * (-ez[ pp ] + ez[ pp + II ] ) );
6        // dla yy = JJ: ez[ pp + II ] = 0
7      for( xx = 1, pp++; xx < II; xx++, pp++ )
8        hx[ pp ] -= chx[ pp ] * (          ey[ pp ] - ey[ pp + IJ ] +
9                                 dzy * (-ez[ pp ]                 ) );
10 }
11       // dla zz = KK: ey[ pp + IJ ] = 0
12   for( yy = 0; yy < JJ - 1; yy++ )
13     for( xx = 1, pp++; xx < II; xx++, pp++ )
14       hx[ pp ] -= chx[ pp ] * (          ey[ pp ]                 +
15                                 dzy * (-ez[ pp ] + ez[ pp + II ] ) );
16       // dla yy = JJ oraz zz = KK
17     for( xx = 1, pp++; xx < II; xx++, pp++ )
18       hx[ pp ] -= chx[ pp ] * (          ey[ pp ]                 +
19                                 dzy * (-ez[ pp ]                 ) );
```

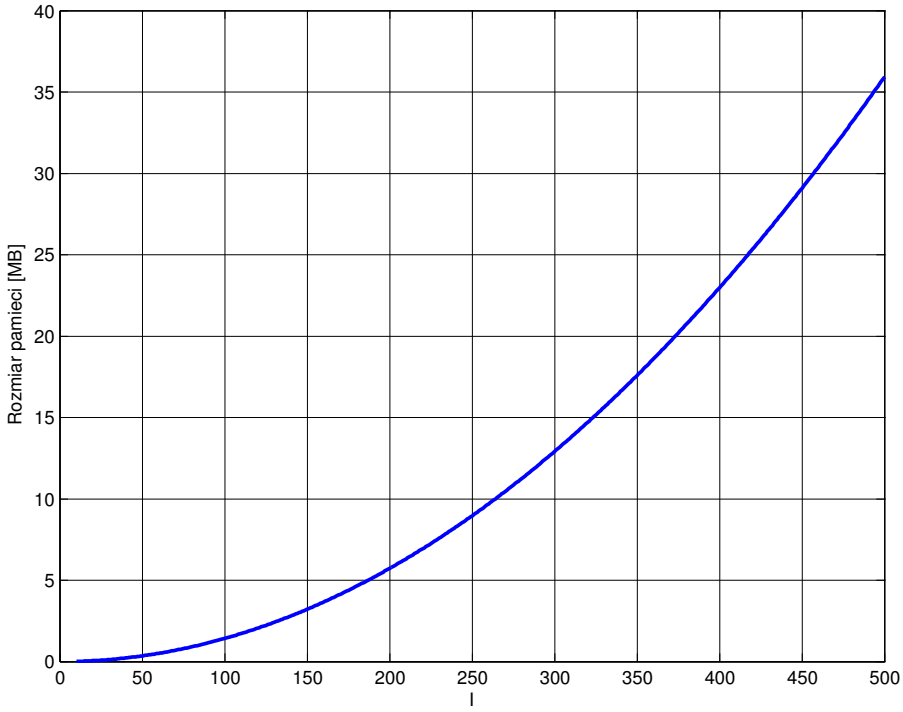
Wydruk 3.2: Przykład implementacji ścianki elektrycznej w wersji B. Fragment kodu, który wyznacza wartości próbek składkowej H_x.

się poza dziedziną. Z tego powodu długość wektorów ex, ey itd. jest równa wartości N_c wyrażonej w r. (3.1).

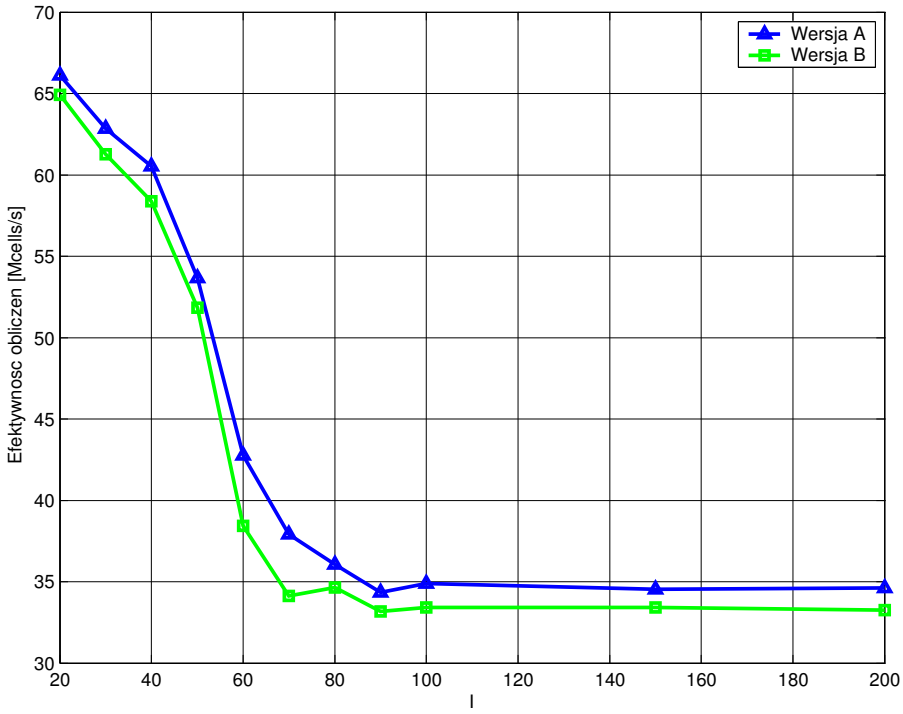
$$N_c = (I-1)(J-1)(K-1) \tag{3.1}$$

Podstawową zaletą tej wersji implementacji warunków brzegowych jest oszczędność pamięci. Stopień oszczędności zasobów pamięci przedstawiono na rys. 3.1, w sytuacji, gdy $I = J = K$, czyli $N = I^3$. Z uwagi na uwzględnienie zerowej wartości próbek w wersji B omawianej implementacji charakteryzuje się ona również mniejszą liczbą operacji zmiennoprzecinkowych oraz mniejszym transferem danych z pamięci w porównaniu do wersji A. Paradoksalnie testy numeryczne przeprowadzone na komputerze PC1 (patrz tab. 3.1), których wynik przedstawiono na rys. 3.2, wykazują, że efektywność obliczeń zmierzona dla implementacji w wersji A jest większa niż dla implementacji w wersji B. Zdaniem autora wynika to z większej liczby instrukcji realizującej obliczenia w wersji B (większa liczba pętli, jednak liczba iteracji po próbkach pól jest taka sama). Wynika stąd, że obniżenie poziomu transferu danych, kosztem rozbudowania kodu programu, nie musi prowadzić do wzrostu efektywności obliczeń. Jest to informacja istotna biorąc pod uwagę wnioski zawarte w kolejnych punktach tego rozdziału.

Z uwagi na chęć zachowania kompatybilności z zoptymalizowanym algorytmem przeznaczonym dla GPU oraz na oszczędność pamięci, która jest zwielokrotniona dla zrównoleglonej wersji algorytmu, wersję B implementacji warunków brzegowych umieszczono w każdej implementacji jawnej algorytmu FDTD zastosowanej do przeprowadzenia testów numerycznych, których wyniki zaprezentowano w dalszej części rozprawy. W celu zachowania zgodności oznaczeń z powszechnie przyjętym zwyczajem [105], rozmiar dziedziny określony jest zawsze przez wartość N, gdzie $N = IJK$.



Rysunek 3.1: Oszczędność pamięci występująca przy implementacji warunków brzegowych w wersji A względem implementacji w wersji B



Rysunek 3.2: Porównanie efektywności obliczeń[a] algorytmu FDTD przy implementacji warunków brzegowych w wersji A i B

[a]Definicja oznaczenia Mcells/s znajduje się w p. 3.2.1

3.2 Koszty numeryczne jawnej implementacji FDTD

Dokładność odwzorowania propagacji fali elektromagnetycznej przez algorytm FDTD jest tym większa, im mniejszy jest zastosowany do obliczeń krok dyskretyzacji. Jednak zmniejszenie kroku dyskretyzacji powoduje znaczne zwiększenie liczby oczek siatki Yee, co prowadzi do wzrostu czasu analizy. W tym punkcie zostaną przedstawione metody estymacji czasu analizy przy pomocy algorytmu FDTD oraz zostaną opisane wymagane przez ten algorytm koszty numeryczne.

Liczba oczek siatki Yee bezpośrednio określa dwa podstawowe z punktu widzenia kosztów numerycznych parametry analizy FDTD:

- koszt pamięciowy, tj. liczba bajtów potrzebna do reprezentacji schematu różnicowego: wymagane jest w przybliżeniu $12N$ zmiennych do reprezentacji algorytmu FDTD (po $3N$ dla reprezentacji wektorów $\mathbf{e}$, $\mathbf{h}$ oraz parametrów materiałowych ϵ i μ),
- koszt obliczeniowy, tj. liczba operacji zmiennoprzecinkowych wykonanych w jednej iteracji FDTD: przeprowadzenie aktualizacji wartości próbek pola elektrycznego i magnetycznego z pojedynczej komórki Yee wymaga wykonania 42 operacji zmiennoprzecinkowych w każdej iteracji (każde z r. (2.24) wymaga przeprowadzenia czterech operacji dodawania oraz trzech operacji mnożenia).

Te dwa podstawowe czynniki można określić poprzez r. (3.2).

$$L_B = 12N\alpha_F \tag{3.2a}$$

$$L_F = 42N \tag{3.2b}$$

gdzie:
L_B - koszt pamięciowy, tj. liczba bajtów wymagana przez algorytm FDTD w postaci jawnej do reprezentacji próbek pola elektromagnetycznego oraz parametrów materiałowych; jednostka parametru: [B], ang. *byte*,
L_F - koszt obliczeniowy, tj. liczba operacji zmiennoprzecinkowych wykonywanych w jednej iteracji algorytmu FDTD w postaci jawnej; jednostka parametru: flop, ang. *FLoating-point OPeration*,
α_F - liczba bajtów wymagana do reprezentacji liczby zmiennoprzecinkowej w programie komputerowym; dla CPU oraz GPU $\alpha_F = 4B$ dla zmiennej o pojedynczej precyzji oraz $\alpha_F = 8B$ dla zmiennej o podwójnej precyzji.

Obliczenia wykonywane na zmiennych o podwójnej precyzji wymagają większego transferu danych pomiędzy pamięcią RAM a jednostkami obliczeniowymi oraz zarówno dla kart graficznych jak i procesorów komputerowych czas realizacji obliczeń na zmiennych o podwójnej precyzji jest większy w porównaniu do czasu obliczeń realizowanych na zmiennych o pojedynczej precyzji. Z tego względu w zdecydowanej większości testów przedstawionych w tej rozprawie do obliczeń zastosowano zmienne w pojedynczej precyzji, co pozwala osiągnąć większą efektywność obliczeń. W rozdziale 7 wykazano stopień zmniejszenia efektywności obliczeń przy zastosowaniu zmiennych o podwójnej precyzji oraz udokumentowano, że rezultaty analiz wykonanych na zmiennych o pojedynczej i podwójnej precyzji są do siebie bardzo zbliżone.

Znaczny wpływ rozmiaru analizowanego problemu na czas symulacji przedstawiono poniżej dla sytuacji dwukrotnego zmniejszenia kroku dyskretyzacji przestrzeni ciągłej w każdym kierunku przestrzeni. Wówczas nowo określona analiza wymaga:

- dwukrotnego zwiększenia liczby iteracji w celu przeprowadzenia symulacji propagacji fali elektromagnetycznej w przedziale czasowym o takiej samej długości jak przed zmianą kroku dyskretyzacji, zgodnie z r. (A.5),
- ośmiokrotny wzrost liczby komórek Yee w analizie, zgodnie z r. (2.13g), a przez to również ośmiokrotny wzrost zapotrzebowania na pamięć współpracującą z jednostką obliczeniową,
- szesnastokrotny wzrost liczby operacji zmiennoprzecinkowych (dwukrotne zwiększenie liczby iteracji oraz ośmiokrotny wzrost liczby komórek Yee).

3.2.1 Metody oceny efektywności obliczeń

W tej rozprawie pod pojęciem efektywność obliczeń rozumiana jest szybkość działania implementacji algorytmu FDTD z uwzględnieniem stopnia wykorzystania dostępnych zasobów sprzętowych. Im efektywność obliczeń jest większa, tym czas obliczeń jest krótszy i stopień wykorzystania zasobów sprzętowych jest większy.

Efektywność obliczeń scharakteryzowano za pomocą następujących parametrów:

- S_F - liczba operacji zmiennoprzecinkowych wykonanych w ciągu jednej sekundy; przyjęto oznaczenie jednostki tego parametru jako flops/s, przy czym Gflops/s oznacza 10^9 flops/s.
- S_B - liczba danych przesłanych pomiędzy jednostką obliczeniową FPU (ang. *Floating-Point Unit*), a pamięcią; przyjęto oznaczenie jednostki tego parametru jako B/s, przy czym GB/s oznacza 10^9 B/s. W rozprawie nie jest stosowana liczność danych opisana przez potęgę liczby 2^{10}, oznaczonej w [7] jako KiB (ang. *kibibyte*).
- S_C - liczba komórek Yee, dla których wykonano aktualizację wartości pól w czasie jednej sekundy; przyjęto oznaczenie jednostki tego parametru jako cells/s, przy czym Mcells/s oznacza 10^6 cells/s.

Wartości powyższych parametrów dla implementacji algorytmu FDTD w postaci jawnej można określić po przeprowadzeniu symulacji FDTD w określonym środowisku sprzętowym na podstawie znajomości czasu symulacji t_{sim}, liczby przeprowadzonych iteracji n_{it} oraz liczby komórek Yee N zastosowanej w tej symulacji, zgodnie z r. (3.3).

$$S_F = \frac{L_F n_{it}}{t_{sim}} = \frac{42 N n_{it}}{t_{sim}} \tag{3.3a}$$

$$S_B = \frac{L_B n_{it}}{t_{sim}} = \frac{12 N \alpha_F n_{it}}{t_{sim}} \tag{3.3b}$$

$$S_C = \frac{N n_{it}}{t_{sim}} \tag{3.3c}$$

Jak można zauważyć parametry te są wzajemnie ze sobą powiązane i na podstawie znajomości wartości jednego parametru można wyznaczyć wartości pozostałych parametrów. Należy przy tym podkreślić, że S_B uwzględnia tylko jednokrotny transfer danych do pamięci, lecz w algorytmie FDTD jedna próbka pola jest brana pod uwagę przy wyznaczaniu wartości czterech próbek pól i z tego względu rzeczywisty poziom transferu jest większy. Z uwagi na niewielki wpływ programisty na sposób zarządzania pamięcią

podręczną przez CPU trudno jest oszacować rzeczywisty poziom transferu danych i m.in. dlatego pozostawiono definicję parametru S_B z założeniem jednokrotnego transferu każdego elementu wektorów ze schematu różnicowego. Tak określona definicja parametru S_B oznacza zatem poziom minimalnego transferu danych jakiego wymaga algorytm FDTD.

Dodatkowego wyjaśnienia wymaga również opis rodzaju transferu danych. Jak można bowiem zauważyć na wydruku 3.1 do obliczeń wymagany jest odczyt $12N$ zmiennych (próbki pól oraz parametrów materiałowych) oraz zapis $6N$ zmiennych, czyli sumaryczny transfer danych uwzględniony w r. (3.3b) teoretycznie powinien wynosić $18N$. Jednak specyfiką procesorów komputerowych jest intensywne korzystanie z pamięci podręcznej nawet w sytuacji zapisu danych, tj. pomimo, że przykładowy program wymagać może tylko i wyłącznie nadania nowych wartości zmiennym z danego obszaru pamięci RAM, to procesor wykona najpierw odczyt tego obszaru pamięci oraz umieści jego zawartość w pamięci podręcznej, po czym wykona wymagany zapis wartości, [1]. Z tego względu można przyjąć, że minimalny poziom transferu danych równy jest $12N$ zmiennych, przy czym relacja pomiędzy odczytem a zapisem zmiennych wynosi 6:6.

Procesory komputerowe oraz akceleratory graficzne często są charakteryzowane przez producentów poprzez podanie maksymalnej mocy obliczeniowej $S_{F\,max}$ oraz maksymalnego poziomu transferu danych $S_{B\,max}$ pomiędzy FPU (ang. *Floating-Point Unit*) a pamięcią zewnętrzną typu RAM (ang. *Random Access Memory*). Przy pomocy tych parametrów można określić teoretyczną efektywność obliczeń algorytmu FDTD, zgodnie z r. (3.4).

$$S_{Cf} = \frac{S_{F\,max}}{42} \tag{3.4a}$$

$$S_{Cb} = \frac{S_{B\,max}}{12\alpha_F} \tag{3.4b}$$

Znajomość wartości parametrów S_{Cb} oraz S_{Cf} pozwala wskazać czynnik, który ogranicza efektywność obliczeń dla danych zasobów sprzętowych:

- Jeśli $S_{Cb} > S_{Cf}$, to ograniczenie efektywności obliczeń wynika z czasu obliczeń zmiennoprzecinkowych.
- Jeśli $S_{Cb} < S_{Cf}$, to ograniczenie efektywności obliczeń wynika z niskiego poziomu transferu danych pomiędzy FPU a pamięcią.
- Jeśli $S_{Cb} \approx S_{Cf}$, to algorytm potencjalnie jest w pełni dostosowany do zasobów sprzętowych, gdyż jednocześnie wykorzystuje jego maksymalną moc obliczeniową oraz maksymalny transfer danych. Rzeczywisty poziom dostosowania algorytmu do zasobów sprzętowych zwykle różni się od potencjalnego, co zostanie wykazane w kolejnych punktach tego rozdziału.

3.3 Efektywność obliczeń wykonywanych na CPU

W obliczeniach przeprowadzonych z udziałem dowolnych zasobów sprzętowych występuje zwykle znaczna rozbieżność pomiędzy parametrami opisującymi jej teoretyczną wydajność, a praktycznym jej wykorzystaniem przez program komputerowy. Celem kolejnych punktów jest wykazanie dlaczego implementacja algorytmu FDTD nie jest w stanie nawet zbliżyć się do stu procentowej skuteczności wykorzystania dostępnych zasobów sprzętowych oraz jaki jest wpływ architektury procesorów na uzyskaną efektywność obliczeń.

3.3.1 Istotne cechy architektury procesora komputerowego

Testy efektywności obliczeń wykonano na dwóch komputerach. Ich parametry przedstawiono w tab. 3.1. Testy, których wyniki zostały umieszczone w tym punkcie rozprawy, zostały przeprowadzone dla jednego rdzenia procesora, przy braku obciążenia obliczeniami pozostałych rdzeni danego zestawu.

Tablica 3.1: Parametry komputerów zastosowanych do obliczeń

Nazwa komputera	PC1	PC2
Nazwa procesora	Intel(R) Xeon(R) CPU E5345	Dual Core AMD Opteron(tm) Processor 275
Liczba rdzeni w procesorze	4	2
Liczba procesorów w komputerze	2	2
Częstotliwość taktowania procesora	2,33 GHz	2,2 GHz
Rozmiar pamięci podręcznej procesora pierwszego poziomu (L1) przeznaczonej dla danych	4 x 32K	2 x 64K
Rozmiar pamięci podręcznej procesora drugiego poziomu (L2)	2 x 2048K	2 x 1024K
Rozmiar pamięci RAM	8 GB	4 GB
Częstotliwość taktowania pamięci RAM	166 MHz	200 MHz
Rodzaj pamięci RAM	DDR2	DDR
Liczba kanałów pamięci dla jednego CPU	2	2

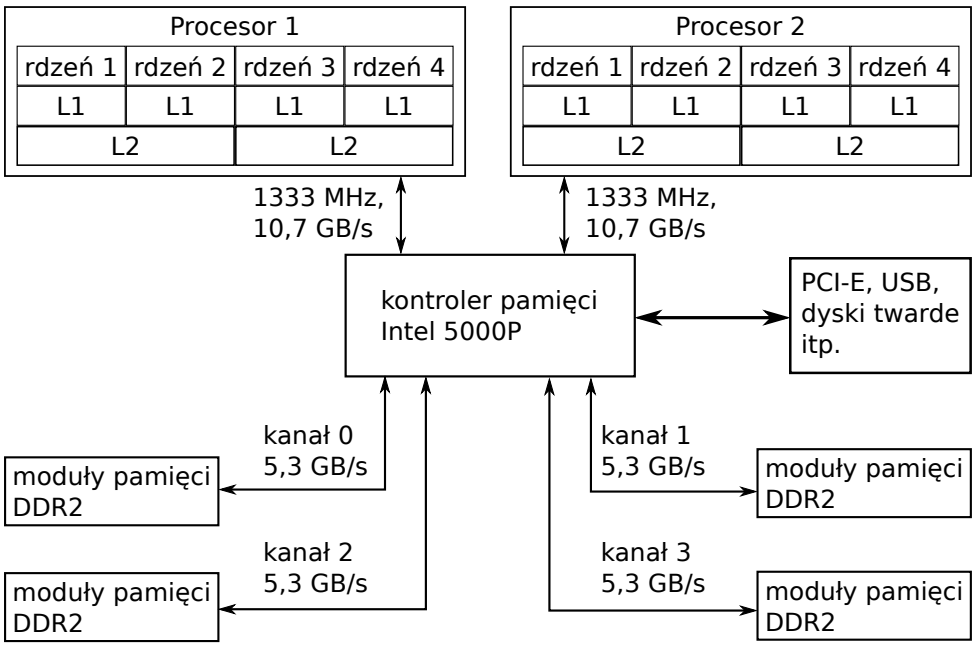
Znajomość konfiguracji, rodzaju i częstotliwości taktowania pamięci RAM umożliwia określenie teoretycznego maksymalnego poziomu transmisji danych pomiędzy procesorem a pamięcią zewnętrzną $S_{B\,max}$, zgodnie z r. (3.5) otrzymanym na podstawie [108].

$$S_{B\,max} = \frac{f_{mem} k_{bus} k_d w_{bus} k_{ch}}{8\frac{bit}{B}} \tag{3.5}$$

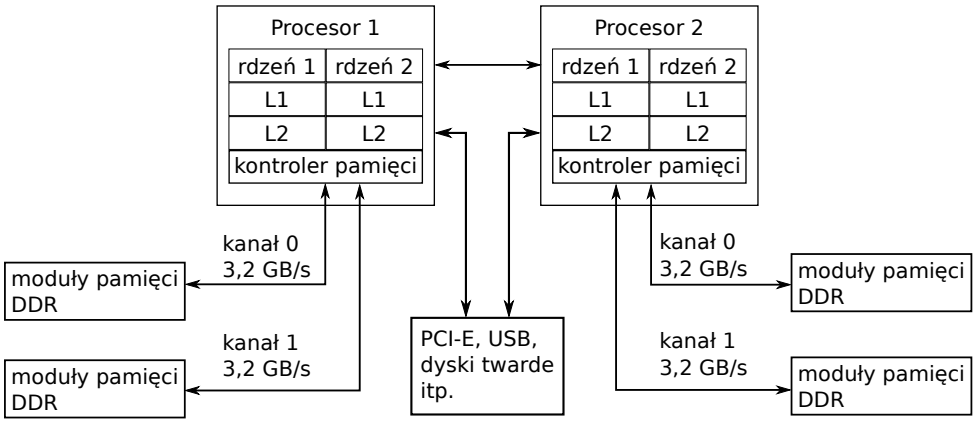
gdzie:
$S_{B\,max}$ - maksymalny teoretyczny transfer danych dla określonej pamięci [B/s],
f_{mem} - częstotliwość taktowania pamięci [Hz],
k_{bus} - zwielokrotnienie częstotliwości szyny systemowej w odniesieniu do f_{mem} (dla pamięci DDR $k_{bus} = 1$, DDR2 $k_{bus} = 2$, DDR3 $k_{bus} = 4$),
k_d - liczba bitów przesyłana przez jedną linię szyny danych w czasie jednego okresu zegara $(1/f_{mem})$ (dla pamięci DDR, DDR2, DDR3 $k_d = 2$),
w_{bus} - szerokość systemowej szyny danych, powszechnie wynosi ona 64 bity dla jednego kontrolera pamięci,
k_{ch} - liczba kanałów pamięci (ang. *memory channel*).

Zarys architektury procesorów oraz sposób połączenia pamięci przedstawiono na rys. 3.3 oraz rys. 3.4. Wynika z niej, że całkowity strumień danych, który może być transmitowany z pamięci, jest dzielony na wiele szyn danych, a przez to transfer danych dostępny

dla pojedynczego rdzenia procesora jest mniejszy od maksymalnego transferu danych wypływającego/wpływającego do pamięci. Całkowity poziom transferu danych będzie zatem większy dla aplikacji wielowątkowych.



Rysunek 3.3: Uproszczona architektura komputera PC1, który zawiera dwa procesory Intel Xeon E5345



Rysunek 3.4: Uproszczona architektura komputera PC2, który zawiera dwa procesory AMD Opteron 275

Znajomość parametrów procesorów umożliwia wyznaczenie teoretycznej maksymalnej liczby operacji zmiennoprzecinkowych $S_{F\,max}$, które może wykonać rdzeń, procesor oraz komputer. Największą efektywność obliczeń dla procesorów komputerowych można uzyskać poprzez zastosowanie instrukcji SSE (ang. *Streaming SIMD Extensions*), które umożliwiają wykonanie w jednej instrukcji czterech mnożeń (dodawań) liczb zmiennoprzecinkowych o pojedynczej precyzji lub dwóch mnożeń (dodawań) liczb zmiennoprzecinkowych o podwójnej precyzji [1]. Jeden rdzeń procesora Intel Xeon E5345 pozwala

wykonać taką instrukcję w czasie jednego cyklu zegara [1], natomiast jeden rdzeń procesora AMD Opteron 275 wymaga dwóch cykli zegara [6] (kolejne generacje procesorów AMD wymagają jednego cyklu zegara na wskazane instrukcje [5]). Teoretyczną wartość $S_{F\,max}$ można wyznaczyć na podstawie r. (3.6), które otrzymano na podstawie [1].

$$S_{F\,max} = f_{cpu} k_{cpu} k_{core} k_{sse} / k_{cycle} \tag{3.6}$$

gdzie:
$S_{F\,max}$ - maksymalna teoretyczna liczba operacji zmiennoprzecinkowych wykonana przez procesor $[flops/s]$,
f_{cpu} - częstotliwość taktowania procesora komputerowego [Hz],
k_{cpu} - liczba procesorów komputerowych,
k_{core} - liczba rdzeni umieszczona w procesorze,
k_{sse} - maksymalna liczba operacji zmiennoprzecinkowych wykonywana przez instrukcje SSE; dla liczb o pojedynczej precyzji, wynosi ona cztery [1] np. mulps (żadna z instrukcji SSE nie wykonuje równocześnie operacji mnożenia i dodawania liczb zmiennoprzecinkowych),
k_{cycle} - liczba cykli zegara taktującego procesor wymagana do realizacji działań zdefiniowanych wybraną przez instrukcję SSE.

Na podstawie r. (3.5) oraz r. (3.6) można określić potencjalne możliwości komputerów PC1 oraz PC2. Zostały one przedstawione w tab. 3.2. Na jej podstawie można stwierdzić, że maksymalne wykorzystanie mocy obliczeniowej komputerów wymaga zastosowania implementacji zrównoleglonej. Natomiast implementacja jednowątkowa jest w stanie wykorzystać jedynie część potencjalnych możliwości opisanych komputerów, a mianowicie połowę poziomu transferu danych dla PC1 i PC2 oraz odpowiednio 1/8 i 1/4 liczbę operacji zmiennoprzecinkowych dla PC1 i PC2.

Tablica 3.2: Wartości parametrów $S_{B\,max}$ oraz $S_{F\,max}$ dla komputerów PC1 i PC2

Nazwa komputera	PC1	PC2
$S_{B\,max}$ [GB/s]	21,3	12,8
$S_{F\,max}$ [Gflops/s]	74,67	17,6
$S_{B\,max}$ dostępny dla jednego rdzenia [GB/s]	10,7	6,4
$S_{F\,max}$ dostępny dla jednego rdzenia [Gflops/s]	9,33	4,4

Przedstawiona analiza potencjalnych możliwości procesorów komputerowych pozwala określić teoretyczną maksymalną efektywność obliczeń z udziałem algorytmu FDTD. Wyznaczone wartości zawiera tab. 3.3. Wynika z niej, że teoretycznie poziom transferu danych jest wystarczający do przeprowadzenia maksymalnej dopuszczalnej liczby operacji zmiennoprzecinkowych. Wartości te pozwolą na oszacowanie stopnia wykorzystania potencjalnych możliwości procesorów przez implementację algorytmu FDTD, co zostało omówione w kolejnych punktach tego rozdziału.

3.3.2 Ocena wydajności komputerów zastosowanych do obliczeń

W p. 3.3.1 przedstawiono ocenę teoretycznej efektywności obliczeń dla procesorów komputerowych. Jednak rzeczywista efektywność obliczeń jest znacząco niższa. Czynniki, które

Tablica 3.3: Teoretyczna efektywność obliczeń jednowątkowej implementacji algorytmu FDTD

Opis mierzonego parametru	Symbol parametru	PC1	PC2
maksymalna efektywność obliczeń wynikająca z ograniczenia związanego z szybkością wykonywania operacji zmiennoprzecinkowych	S_{Cf} [Mcells/s]	222	105
maksymalna efektywność obliczeń wynikająca z ograniczenia związanego ze skończoną szybkością transmisji danych do FPU	S_{Cb} [Mcells/s]	223	133

mają zasadniczy wpływ na różnicę pomiędzy teoretyczną a rzeczywistą efektywnością obliczeń zostaną opisane w tym punkcie.

Do oceny rzeczywistych możliwości komputerów można posłużyć się jednym z programów, które w tym celu zostały zbudowane. Przykładem takiego programu jest LMbench, [4], który został opisany w [62]. Program ten umożliwia wykonanie pomiaru m.in. szybkości transmisji danych pomiędzy pamięcią a procesorem oraz szybkości wykonania operacji zmiennoprzecinkowych.

3.3.2.1 Rzeczywista maksymalna moc obliczeniowa procesorów komputerowych

Ocena szybkości realizacji operacji zmiennoprzecinkowych wykonana za pomocą programu LMbench została umieszczona w tab. 3.4. Z pomiarów tych wynika, iż jedna operacja mnożenia liczb o pojedynczej precyzji zajmuje obu procesorom cztery cykle zegara wobec zakładanego wykonania czterech operacji mnożenia w jednym cyklu zegara dla PC1 oraz dwóch cykli zegara dla PC2. Zatem pomiary te wskazują, że w rzeczywistości moc obliczeniowa dla PC1 jest szesnastokrotnie mniejsza od teoretycznie dostępnej, a dla PC2 ośmiokrotnie.

Tablica 3.4: Średni czas wykonywania operacji podstawowych dla liczb o pojedynczej precyzji

Nazwa komputera	PC1	PC2
Czas dodawania [ns]	1,29	1,82
Czas mnożenia [ns]	1,75	1,85
Czas dodawania [cykle zegara]	3,01	3,99
Czas mnożenia [cykle zegara]	4,08	4,06

W dodatku C umieszczono wydruki programów, które umożliwiają wskazanie przyczyn tak dużej rozbieżności pomiędzy parametrami teoretycznymi a rzeczywistymi. Programy te przeprowadzają pomiar czasu wykonania instrukcji ADDPS, która realizuje cztery operacje dodawania liczb zmiennoprzecinkowych [2], w dwóch sytuacjach:

- wariant A: mierzony jest czas wykonania tylko instrukcji ADDPS (wydruk C.1),
- wariant B: mierzony jest czas wykonania instrukcji ADDPS oraz instrukcji MOVAPS, która umożliwia umieszczenie i pobranie zmiennych w rejestrach XMM (wydruk C.2).

Oba warianty zostały dokładniej opisane w p. C.1. Rezultaty przeprowadzonych pomiarów umieszczono w tab. 3.5. Wynika z nich, że teoretyczna moc obliczeniowa obu komputerów jest możliwa do osiągnięcia pod warunkiem braku wymiany informacji z rejestrami XMM, co jest cechą niespotykaną w algorytmach numerycznych. Drugi istotny wniosek, który wynika z tab. 3.5 stanowi wskazanie znacznego spadku mocy obliczeniowej przy obecności wymiany informacji pomiędzy rejestrami XMM a pamięcią podręczną L1 procesorów: optymalna implementacja algorytmu, który wykonuje tylko jedną operację zmiennoprzecinkową dla jednej liczby, będzie wykorzystywać moc obliczeniową w 40% dla PC1 oraz w 56% dla PC2.

Przedstawione dane dowodzą, że teoretyczne możliwości procesorów komputerowych są niezmiernie trudne do osiągnięcia przez algorytmy numeryczne. Wykazano przy tym, że zmniejszenie wartości mocy obliczeniowej rozpoczyna się już na etapie uwzględnienia czasu umieszczenia danych w rejestrach.

Tablica 3.5: Średni czas wykonania czterech mnożeń liczb o pojedynczej precyzji

Nazwa komputera	PC1	PC2
Wariant A [cykle zegara]	1,00	2,00
Wariant B [cykle zegara]	2,44	3,58

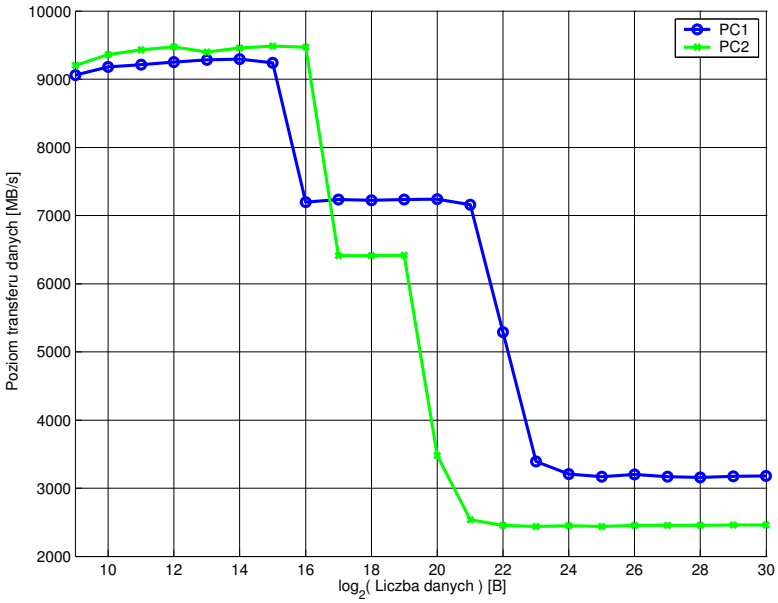
Dużo mniejszy poziom mocy obliczeniowej wskazany przez program LMbench wynika z nieumiejętnie dobranego algorytmu testującego. Wykonuje on obliczenia tylko na dwóch zmiennych typu `float`, a przez to nie korzysta z instrukcji SSE, które wykonują cztery operacje zmiennoprzecinkowe równocześnie (wnioski te pochodzą z analizy kodu asemblera funkcji realizującej obliczenia). Jednak rezultaty testów wykonanych przez program LMbench są pożyteczne, gdyż obrazują wpływ postaci kodu na szybkość jego wykonania przez procesor.

3.3.2.2 Rzeczywisty maksymalny poziom transmisji danych do FPU

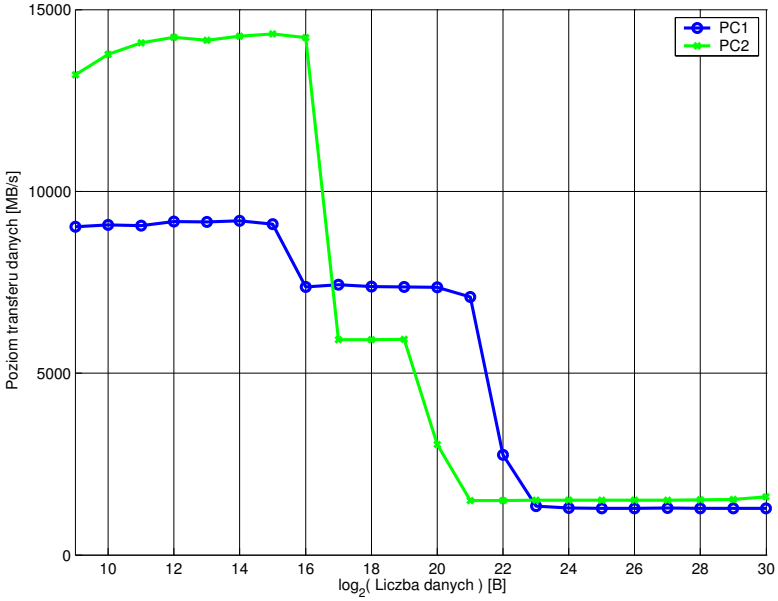
Czas transmisji danych pomiędzy procesorem a pamięcią zewnętrzną jest znaczący w odniesieniu do czasu wykonania instrukcji realizujących działania numeryczne. W celu zmniejszenia kosztu transmisji danych pomiędzy pamięcią zewnętrzną a procesorem powszechne jest zastosowanie w procesorach pamięci podręcznej, do której czas dostępu jest znacząco krótszy w porównaniu do czasu dostępu do pamięci zewnętrznej. Popularnym rozwiązaniem stosowanym przez firmę Intel oraz AMD jest podział pamięci na kilka poziomów (zwykle dwa lub trzy), które charakteryzują się różną wielkością oraz czasem dostępu. Występowanie różnych rodzajów pamięci stanowi efekt kompromisu pomiędzy znaczną ceną pamięci najszybszej oraz rozmiarem pamięci, który może być większy dla pamięci tańszych lecz wolniejszych. Informację na temat rozmieszczenia pamięci podręcznej w procesorach zastosowanych w obliczeniach przedstawiono na rys. 3.3 oraz rys. 3.4, natomiast rozmiary pamięci podręcznej podano w tab. 3.1.

W celu oszacowaniu poziomu transmisji danych pomiędzy procesorem a pamięcią najpierw wykonano testy przy pomocy programu LMbench [62]. Ich rezultaty przedstawiono na rys. 3.5 oraz rys. 3.6.

Widoczne różnice poziomu transmisji odpowiadają różnym rodzajom pamięci, przez co można informacje z rys. 3.5 oraz rys. 3.6 ująć w formie tab. 3.6.



Rysunek 3.5: Szybkość odczytu danych z pamięci dla programu jednowątkowego



Rysunek 3.6: Szybkość zapisu danych do pamięci dla programu jednowątkowego

Podobnie jak przy analizie mocy obliczeniowej procesorów komputerowych, tak i przy pomiarze rzeczywistego maksymalnego poziomu transmisji danych z procesora do pamięci zewnętrznej, otrzymana z pomiarów wartość jest dużo mniejsza niż wartość teoretyczna przedstawiona w tab. 3.2. Znacznie mniejszy poziom transferu danych wynika z konieczności zastosowania sygnałów sterujących pamięć [108], co oznacza, że nie w każdym takcie zegara sterującego pamięć następuje transfer danych, co stanowi założenie obliczeń teoretycznych.

Znajomość zmierzonych wartości maksymalnych mocy obliczeniowej oraz poziomu transmisji danych umożliwia na nowo estymować efektywność obliczeniową procesorów komputerowych. Przy założeniu, że poziom transmisji danych zostanie określony na pod-

Tablica 3.6: Poziom transmisji danych przy komunikacji z FPU dla programu jednowątkowego

Nazwa komputera		PC1	PC2
Zapis danych do pamięci podręcznej L1	[GB/s]	9,1	14,0
Zapis danych do pamięci podręcznej L2	[GB/s]	7,3	5,9
Zapis danych do pamięci zewnętrznej	[GB/s]	1,28	1,5
Odczyt danych z pamięci podręcznej L1	[GB/s]	9,3	9,4
Odczyt danych z pamięci podręcznej L2	[GB/s]	7,2	6,4
Odczyt danych z pamięci zewnętrznej	[GB/s]	3,2	2,45

stawie tab. 3.6 w proporcjach 6:6 względem danych z zapisu i odczytu (patrz p. 3.2.1) oraz przy wykorzystywaniu mocy obliczeniowej w 40% dla PC1 oraz w 56% dla PC2 (patrz p. 3.3.2.1), można wyznaczyć przewidywaną efektywność jednowątkowej implementacji algorytmu FDTD z uwzględnieniem pomiarów. Jej wartość przedstawiono w tab. 3.7.

Uwzględnienie pomiarów wykonanych przy pomocy programu LMbench znacząco obniżyło szacowaną efektywność obliczeń. Widoczne jest również przewidywana przyczyna niskiej efektywności obliczeń: transmisja danych pomiędzy pamięcią a procesorem. Z przeprowadzonych rozważań można wyciągnąć następujący wniosek: dla rzeczywistych problemów o co najmniej średnich rozmiarach (koszt pamięciowy jest większy od rozmiaru pamięci podręcznej procesora dostępnej dla FPU), szacowana maksymalna efektywność obliczeń algorytmu FDTD wynosi 47 Mcells/s dla PC1 oraz 41 Mcells/s dla PC2.

Tablica 3.7: Teoretyczna efektywność obliczeń jednowątkowej implementacji algorytmu FDTD z uwzględnieniem pomiarów

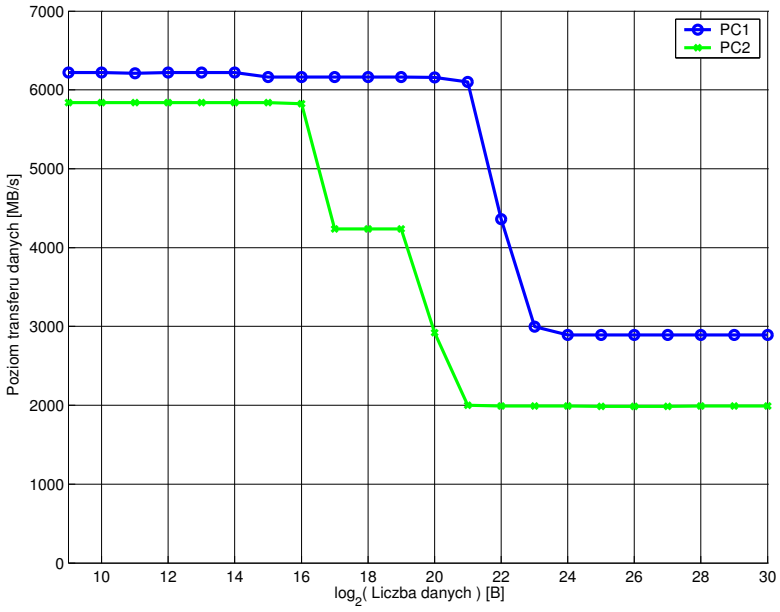
Nazwa komputera	PC1	PC2
S_{Cf} [Mcells/s]	88,8	58,8
S_{Cb} [Mcells/s] RAM	47	41
S_{Cb} [Mcells/s] L2	151	128
S_{Cb} [Mcells/s] L1	191	244

3.3.3 Wpływ struktury danych na poziom transmisji

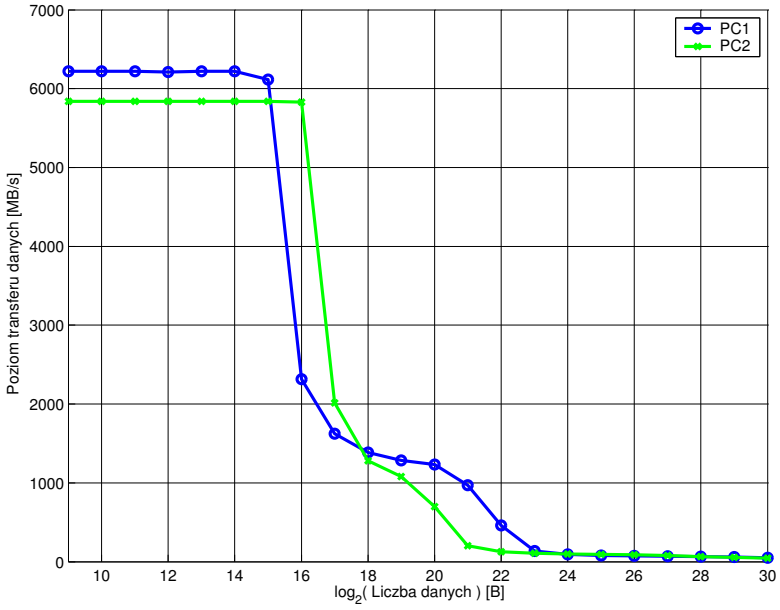
Poziom transferu danych pomiędzy pamięcią zewnętrzną a procesorem w dużej mierze zależy od struktury danych, która zostanie przez programistę zastosowana. Wynika to ze sposobu zarządzania pamięcią podręczną przez procesor. Program jest w stanie uzyskać największą prędkość transmisji danych pomiędzy procesorem a pamięcią, jeśli dane ułożone są sekwencyjnie w pamięci.

W [108] zaproponowano prosty test, który doskonale obrazuje silny wpływ ułożenia danych w pamięci na uzyskany transfer danych. Test ten polega na pomiarze czasu przejścia z jednego elementu listy do kolejnego w dwóch sytuacjach: gdy następny element listy znajduje się w sąsiednim obszarze pamięci oraz gdy znajduje się w losowo wybranym obszarze pamięci. Analogiczny pomiar uzyskano dla kodu przedstawionego w p. C.2. Wyniki tego pomiaru umieszczono na rys. 3.7 i 3.8. W sytuacji, gdy elementy są ułożone sekwencyjnie, transfer danych jest zbliżony do poziomu wskazanego na rys. 3.5, co ukazuje

rys. 3.7. Jednak, gdy powiązania pomiędzy elementami listy są ułożone losowo, poziom transferu danych ulega drastycznemu obniżeniu, aż do poziomu 50,93 MB/s dla PC1 oraz 42,98 MB/s dla PC2, co można zaobserwować na rys. 3.8.



Rysunek 3.7: Poziom transferu danych przy poruszaniu się po liście jednokierunkowej ułożonej sekwencyjnie



Rysunek 3.8: Poziom transferu danych przy poruszaniu się po liście jednokierunkowej z losowym powiązaniem pomiędzy jej elementami

Wykres z rys. 3.8 dowodzi, że ułożenie danych w pamięci komputera ma duże znaczenie przy próbie osiągnięcia dużej efektywności obliczeń. Własność ta jest bezpośrednio związana z metodą przechowywania w pamięci podręcznej procesora informacji z pamięci zewnętrznej: pamięć podręczna umożliwia przechowywanie spójnych fragmentów

pamięci zewnętrznej o określonej liczbie bajtów. Fragmenty te nazywane są wierszami pamięci (ang. *cache lines*) i dla procesorów firmy AMD i Intel typu x86 lub x86-64 zawierają 64 bajty danych. Konsekwencją takiej decyzji jest konieczność wykonywania operacji zapisu i odczytu danych do/z pamięci zewnętrznej poprzez wymianę 64 bajtowych bloków danych. Takie działanie zwiększa szybkość transmisji danych pomiędzy procesorem a pamięcią tylko w sytuacji, gdy działania są wykonywane na zmiennych umieszczonych sekwencyjnie w pamięci (wówczas wymagana jest mniejsza liczba sygnałów sterujących pamięć).

Z uwagi na tą cechę pamięci podręcznej optymalne ułożenie danych dla algorytmu FDTD stanowi przydział każdemu rodzajowi próbek pól lub parametrów materiałowych oddzielnych i spójnych obszarów pamięci. Można to zrealizować poprzez zdefiniowanie 12 oddzielnych tablic dla każdego zbioru próbek: pola elektrycznego (E_x, E_y, E_z), pola magnetycznego (H_x, H_y, H_z), parametrów materiałowych.

3.3.4 Pomiar efektywności obliczeń dla pojedynczego procesora komputerowego

Rezultaty pomiarów efektywności obliczeń jednowątkowej implementacji algorytmu FDTD przeznaczonej dla CPU przedstawiono na rys. 3.9. Parametry poszczególnych symulacji umieszczono w tab. 3.8. Jak można zauważyć z rys. 3.9 oraz rys. 3.5 istnieje silna korelacja pomiędzy efektywnością obliczeń a poziomem transferu danych. Fakt ten potwierdza tezę, iż ograniczenie efektywności obliczeń dla CPU wynika z niskiego poziomu transferu danych pomiędzy CPU a pamięcią RAM.

Maksymalną efektywność obliczeń uzyskano dla problemu o najmniejszym rozmiarze, gdyż rozmiar tablic reprezentujących próbki pól i parametrów materiałowych jest na tyle mały, iż mieszczą się one w pamięci podręcznej L1, dla której poziom transferu danych jest największy. Wartość efektywności, odpowiednio 93 [Mcells/s] dla PC1 oraz 49 [Mcells/s] dla PC2 odpowiada szacowanej efektywności S_{Cf} zawartej w tab. 3.7, która określa ograniczenia wynikające z mocy obliczeniowej procesorów.

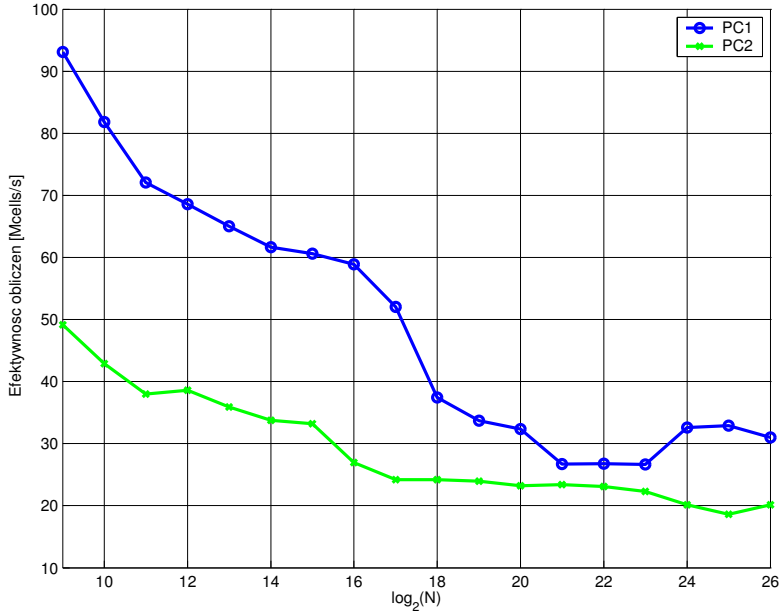
Natomiast dla problemów o dużych rozmiarach efektywność obliczeń wynosi około 30 [Mcells/s] dla PC1 oraz 23 [Mcells/s] dla PC2. Są to wartości mniejsze od efektywności S_{Cb} zawartej w tab. 3.7, która określa ograniczenia wynikające z poziomu transferu danych pomiędzy procesorem a pamięcią RAM. Ponieważ dla małych rozmiarów problemu uzyskano efektywność obliczeń bliską maksymalnej możliwej, zatem obniżenie efektywności wynika ze sposobu transferu danych: niektóre zmienne są transmitowane wielokrotnie, co wynika z r. (2.24) oraz dodatkowo poziom transferu danych określony jest przez mniejszą wartość od maksymalnej zmierzonej na skutek konieczności odwoływania się do danych umieszczonych w różnych fragmentach pamięci, co wywołuje ograniczenia opisane w p. 3.3.3.

W p. 3.3 wskazano zatem główne cechy architektury procesorów, które mają decydujący wpływ na efektywność obliczeń jednowątkowej implementacji FDTD. Zmierzona wartość efektywności przedstawiona na rys. 3.9 znacząco odbiega od wartości estymowanej na podstawie parametrów procesorów przedstawionej w tab. 3.3: zmierzony poziom efektywności dla problemów o dużych rozmiarach jest mniejszy 7,4 krotnie dla PC1 oraz 4,5 krotnie dla PC2 w odniesieniu do efektywności obliczeń estymowanej na podstawie podstawowych parametrów komputera (tab. 3.3). Otrzymane wartości dowodzą, iż szacowanie efektywności implementacji algorytmu FDTD przeznaczonego dla CPU na podstawie parametrów procesorów powinno uwzględniać niski stopień wykorzystania zasobów

sprzętowych z uwagi na strukturę danych i rodzaj przeprowadzonych obliczeń wymaganych przez algorytm FDTD.

Tablica 3.8: Wybrane parametry symulacji zastosowanych do pomiaru efektywności jednowątkowej implementacji algorytmu FDTD

Numer symulacji	I	J	K	N	$log_2(N)$	L_B [MB]
1	8	8	8	512	9	0,0246
2	8	8	16	1024	10	0,0492
3	8	16	16	2048	11	0,0983
4	16	16	16	4096	12	0,197
5	16	16	32	8192	13	0,393
6	16	32	32	16384	14	0,796
7	32	32	32	32768	15	1,57
8	32	32	64	65536	16	3,15
9	32	64	64	131072	17	6,29
10	64	64	64	262144	18	12,6
11	64	64	128	524288	19	25,2
12	64	128	128	1048576	20	50,3
13	128	128	128	2097152	21	101
14	128	128	256	4194304	22	201
15	128	256	256	8388608	23	403
16	256	256	256	16777216	24	805
17	256	256	512	33554432	25	1611
18	256	512	512	67108864	26	3221



Rysunek 3.9: Efektywność jednowątkowej implementacji algorytmu FDTD przeznaczonej dla CPU

3.4 Efektywność obliczeń wykonywanych na wielu CPU

Przetwarzanie równoległe, [36], w porównaniu z jednowątkowym, oferuje znaczny wzrost efektywności obliczeń i z tego powodu budzi wielkie zainteresowanie osób związanych z elektrodynamiką obliczeniową. W latach 90 XX wieku dostosowanie algorytmu FDTD do specyfiki przetwarzania wielowątkowego było dedykowane przede wszystkim klastrom, czyli urządzeniom mało dostępnym dla szerszego kręgu odbiorców. Z tego względu podejście to nie było szerzej stosowane w aplikacjach komercyjnych. Jednak trend ten ulega aktualnie zmianie ze względu na coraz bardziej powszechne wielordzeniowe procesory komputerowe. W związku z tym, zrównoleglenie algorytmu FDTD stanowi nieuchronną drogę rozwoju elektrodynamiki obliczeniowej.

W punkcie tym zostanie opisana metoda zrównoleglenia algorytmu FDTD dostosowana do wielu procesorów komputerowych, która jest wzorowana na pracach [31, 110]. W porównaniu do [31] działanie algorytmu zostało zmodyfikowane oraz dostosowane do współpracy z procesorami wielordzeniowymi. Następnie przeanalizowano czynniki ograniczające efektywność obliczeń implementacji zrównoleglonego algorytmu FDTD.

Podstawową zasadą, która pozwala na zastosowanie wielu procesorów komputerowych, przy jednej symulacji FDTD, jest podział dziedziny obliczeniowej na zadaną liczbę poddziedzin oraz zdefiniowanie pomiędzy nimi metody komunikacji. W celu zobrazowania wymaganych działań zostanie przedstawiony sposób zrównoleglenia obliczeń najpierw dla problemu najprostszego, czyli dla dwóch procesorów komputerowych. Następnie opis zostanie rozszerzony dla zagadnienia podziału dziedziny obliczeniowej na N_p poddziedzin.

3.4.1 Obliczenia dla dwóch procesorów komputerowych

Sposób zdefiniowania obliczeń dla dwóch procesorów komputerowych zostanie omówiony na przykładzie obliczeń dla dziedziny o rozmiarze $3 \times 7 \times 3$ oczek siatki Yee. Dziedzinę tę podzielono na dwie poddziedziny: pierwszą o rozmiarze $3 \times 3 \times 3$ oczek siatki Yee, oraz drugą o rozmiarze $3 \times 4 \times 3$ oczek siatki Yee. Podział ten zobrazowano na rys. 3.10.

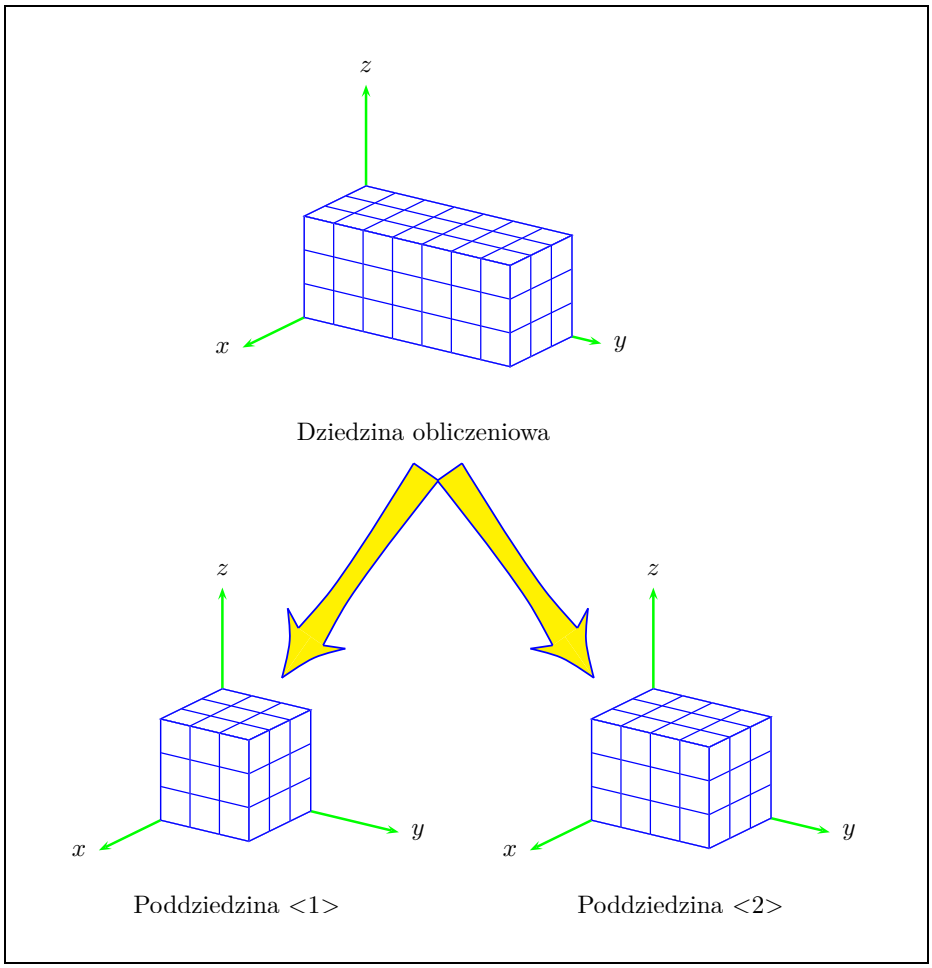
Każdy z procesorów wykonywać będzie obliczenia dla jednej poddziedziny, w sposób analogiczny jak w algorytmie jednowątkowym. Jednak, aby obliczenia były poprawne, należy zdefiniować sposób wymiany danych pomiędzy procesorami.

W celu wyjaśnienia metody komunikacji pomiędzy dziedzinami na rys. 3.12 przedstawiono granicę pomiędzy nimi z uwzględnieniem komórek Yee[2]. Należy przy tym zaznaczyć, że podział dziedzin obliczeniowych zgodny z podziałem składowych ustalonym przez zasięg działania komórek Yee nie jest jedyną możliwością. Przykładowo, można podzielić dziedzinę obliczeniową tak, aby składowe pola E_x należały do jednej poddziedziny, a składowe E_z, z tego samego zakresu komórek Yee, należały do poddziedziny sąsiedniej. Jednak zastosowany podział wzdłuż komórek Yee, pozwala na zachowanie zgodności z algorytmem jednowątkowym oraz zapewnia minimalny transfer danych pomiędzy procesami/wątkami, które realizują obliczenia dla różnych poddziedzin.

Zobrazowanie granicy poddziedzin na rys. 3.12 pozwala wyraźnie zauważyć zależności pomiędzy poddziedzinami, które wynikają z r. (2.24). W tab. 3.9 przedstawiono umowny zapis zależności pomiędzy poddziedzinami, przy czym symbol $X^{<Y>}(\mathbb{Z}_X, \mathbb{Z}_Y, \mathbb{Z}_Z)$

[2]Na rys. 3.12, 3.14 i 3.13 kolorem zielonym zaznaczono próbki pól, które są transmitowane do sąsedniej poddziedziny.

oznacza próbki składowej pola X z poddziedziny Y z zakresu komórek Yee o numerach (i, j, k), gdzie $i \in \mathbb{Z}_X$, $j \in \mathbb{Z}_Y$, $k \in \mathbb{Z}_Z$ (zbiór liczb naturalnych $\mathbb{Z}_i$ może stanowić zbiór jednoelementowy, wówczas np. $\mathbb{Z}_Y = 1$, lub może stanowić zbiór liczb naturalnych m z zakresu $a \leq m \leq b$ oznaczonych jako $\mathbb{Z}_i = a : b$).

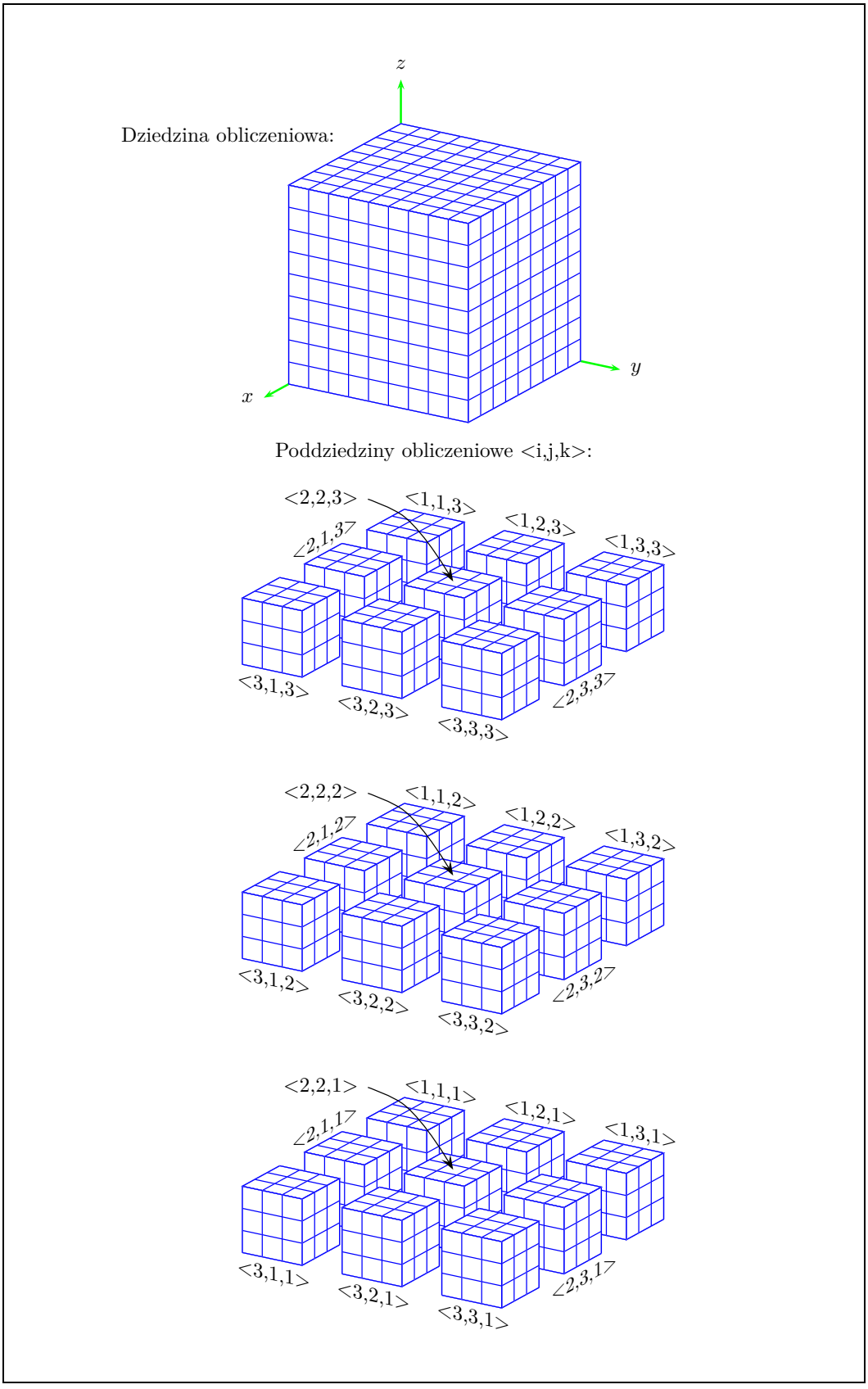


Rysunek 3.10: Podział dziedziny obliczeniowej na dwie poddziedziny

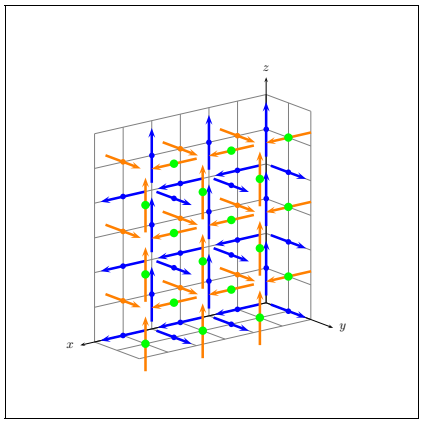
3.4.2 Obliczenia dla N_p procesorów komputerowych

Najważniejsze aspekty zrównoleglenia algorytmu FDTD dla N_p procesorów zostaną przedstawione na przykładzie podziału dziedziny obliczeniowej na 27 poddziedzin, który poglądowo przedstawiono na rys. 3.11. Najistotniejsza z punktu widzenia opisu procesu zrównoleglenia, poddziedzina <2,2,2>, pozwala na przedstawienie wszystkich możliwych relacji, które mogą zaistnieć przy komunikacji pomiędzy sąsiadującymi ze sobą poddziedzinami. Komunikację pomiędzy poddziedzinami, których granica przebiega wzdłuż $y = const$, omówiono w punkcie 3.4.1. Komunikacja taka ma miejsce np. dla poddziedzin <2,1,2> i <2,2,2> oraz <2,2,2> i <2,3,2> z rys. 3.11.

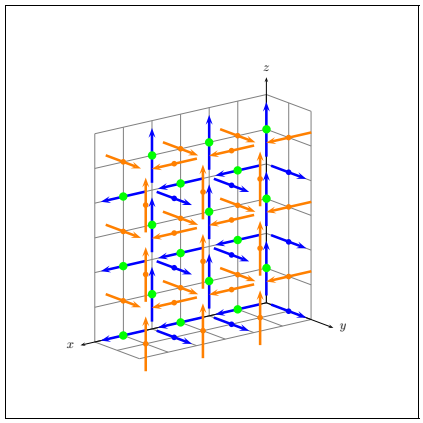
Na rys. 3.13 zobrazowano komórki Yee, które biorą udział w procesie komunikacji pomiędzy poddziedzinami, których granica przebiega wzdłuż $z = const$. Taka sytuacja ma miejsce np. pomiędzy poddziedzinami <2,2,1> i <2,2,2> oraz <2,2,2> i <2,2,3> z rys. 3.11. Ostatni rodzaj komunikacji dotyczy poddziedzin, których granica przebiega



Rysunek 3.11: Podział dziedziny obliczeniowej na 27 poddziedzin

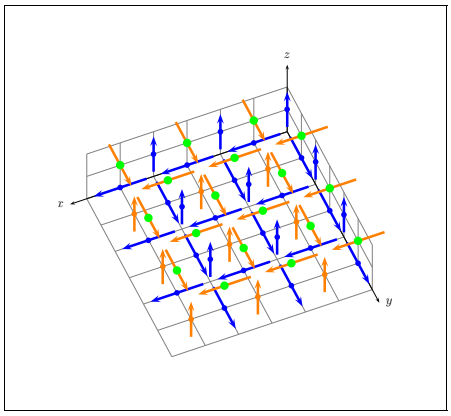


(a) Poddziedzina w kierunku -y

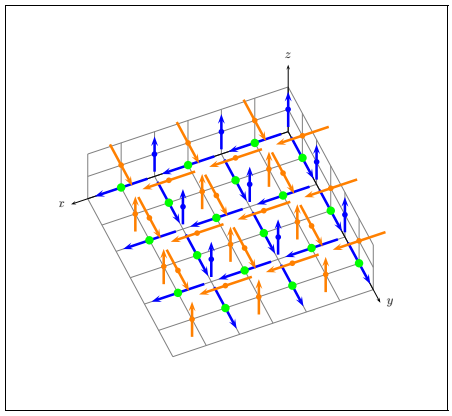


(b) Poddziedzina w kierunku +y

Rysunek 3.12: Komunikacja pomiędzy poddziedzinami w kierunku y

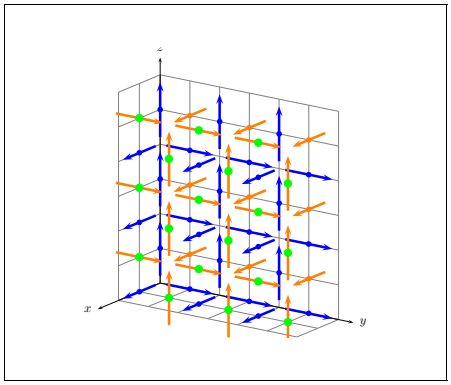


(a) Poddziedzina w kierunku -z

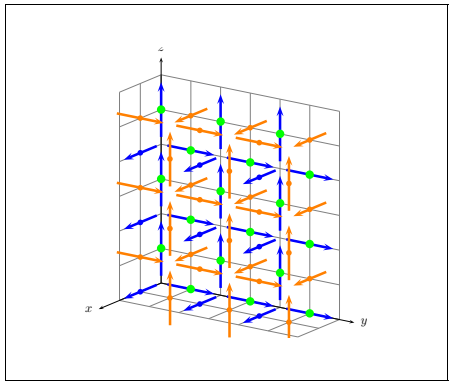


(b) Poddziedzina w kierunku +z

Rysunek 3.13: Komunikacja pomiędzy poddziedzinami w kierunku z



(a) Poddziedzina w kierunku -x



(b) Poddziedzina w kierunku +x

Rysunek 3.14: Komunikacja pomiędzy poddziedzinami w kierunku x

wzdłuż $x = const$. Rys. 3.14 przedstawia komórki Yee, które biorą udział w trakcie takiej komunikacji. Komunikacja ta ma miejsce np. pomiędzy poddziedzinami <1,2,2> i <2,2,2> oraz <2,2,2> i <3,2,2> z rys. 3.11. Wymagany transfer danych, który powinien zostać zdefiniowany pomiędzy tak określonymi poddziedzinami, ujęto w tab. 3.11 i 3.10.

Z przedstawionego opisu wynika, że zrównoleglenie algorytmu FDTD w postaci jawnej dla wielu procesorów lub dla procesorów wielordzeniowych polega na przydzieleniu każdej jednostce obliczeniowej znacznej liczby obliczeń, a następnie wymaga zdefiniowania komunikacji pomiędzy nimi. Taki sposób zrównoleglenia nazywany jest zrównoleglenieniem gruboziarnistym (ang. *coarse-grained parallelism*), [36].

Tablica 3.9: Zależność pomiędzy próbkami pól wzdłuż granicy $y = const$ pomiędzy poddziedzinami

Wartości wyznaczane	Próbki wymagane z dziedziny sąsiadującej
$E_z^{<2>}(1:I,1,1:K)$	$H_x^{<1>}(1:I,J,1:K)$
$E_x^{<2>}(1:I,1,1:K)$	$H_z^{<1>}(1:I,J,1:K)$
$H_x^{<1>}(1:I,J,1:K)$	$E_z^{<2>}(1:I,1,1:K)$
$H_z^{<1>}(1:I,J,1:K)$	$E_x^{<2>}(1:I,1,1:K)$

Tablica 3.10: Zależność pomiędzy próbkami pól wzdłuż granicy $z = const$ pomiędzy poddziedzinami

Wartości wyznaczane	Próbki wymagane z dziedziny sąsiadującej
$E_x^{<2,2,3>}(1:I,1:J,1)$	$H_y^{<2,2,2>}(1:I,1:J,K)$
$E_y^{<2,2,3>}(1:I,1:J,1)$	$H_x^{<2,2,2>}(1:I,1:J,K)$
$H_y^{<2,2,2>}(1:I,1:J,K)$	$E_x^{<2,2,3>}(1:I,1:J,1)$
$H_x^{<2,2,2>}(1:I,1:J,K)$	$E_y^{<2,2,3>}(1:I,1:J,1)$

Tablica 3.11: Zależność pomiędzy próbkami pól wzdłuż granicy $x = const$ pomiędzy poddziedzinami

Wartości wyznaczane	Próbki wymagane z dziedziny sąsiadującej
$E_z^{<3,2,2>}(1,1:J,1:K)$	$H_y^{<2,2,2>}(I,1:J,1:K)$
$E_y^{<3,2,2>}(1,1:J,1:K)$	$H_z^{<2,2,2>}(I,1:J,1:K)$
$H_y^{<2,2,2>}(I,1:J,1:K)$	$E_z^{<3,2,2>}(1,1:J,1:K)$
$H_z^{<2,2,2>}(I,1:J,1:K)$	$E_y^{<3,2,2>}(1,1:J,1:K)$

W zrównoleglonej implementacji algorytmu FDTD w postaci jawnej przeznaczonej dla procesorów do realizacji transmisji danych pomiędzy jednostkami obliczeniowymi zastoso-

wano interfejs MPI (ang. *Message Passing Interface*) [64] zaimplementowany w bibliotece MPICH2 [65] dla PC2 i MVAPICH2 [66] dla PC1.

3.4.3 Transfer danych w komputerach wieloprocesorowych

W p. 3.3.4 wskazano na dużą korelację pomiędzy efektywnością obliczeń implementacji algorytmu FDTD a maksymalnym poziomem transferu danych pomiędzy procesorem komputerowym i pamięcią RAM. Przy obliczeniach zrównoleglonych wzajemne powiązanie tych parametrów jest jeszcze bardziej zauważalne. Przyczyna takiej własności ma swoje źródło w tym, iż procesor komputerowy jest urządzeniem znacznie szybszym niż współpracująca z nim pamięć RAM, a przez to nawet program jednowątkowy jest w stanie wymagać transferu danych o poziomie bliskim maksymalnemu transferowi danych, który może zapewnić kontroler pamięci.

W celu oszacowania stopnia tego zjawiska przeprowadzono testy, w których ustalona liczba procesów równocześnie przeprowadzała kopiowanie elementów z jednej tablicy do drugiej (każdy proces działał na osobnych tablicach), dzięki czemu można oszacować maksymalny poziom transferu danych w programie zrównoleglonym. Dla komputera PC1, w którym jeden kontroler pamięci zarządza transferem danych do dwóch procesorów (rys. 3.3), spadek poziomu transferu danych do pamięci RAM, widoczny z poziomu procesu, pomiędzy programem jedno- a dwuprocesowym wynosi aż 59% (tab. 3.12). Zatem sumaryczny poziom transferu danych dla dwóch procesów wynosi blisko 82% wartości otrzymanej dla jednego procesu (tab. 3.13). Ewidentnie równoczesna współpraca z wieloma procesami nie jest łatwa do przeprowadzenia dla pojedynczego kontrolera pamięci. Maksymalny sumaryczny poziom transferu danych do pamięci RAM dla PC1 uzyskano przy uruchomionych ośmiu procesach. Wynosi on 5296 MB/s (tab. 3.13), czyli 39% więcej niż dla uruchomionego jednego procesu, co stanowi wartość znacząco mniejszą niż teoretycznie wymagany przez zrównoleglony algorytm ośmiokrotnie większy poziom transferu danych. Z punktu widzenia jednego procesu transfer danych, przy uruchomionych ośmiu procesach kopiujących dane, jest 5,72 krotnie mniejszy (tab. 3.12) w porównaniu do transferu danych uzyskanego przez tylko jeden samodzielnie działający proces.

Dla komputera PC2, w którym każdy procesor posiada osobny kontroler pamięci (rys. 3.4), sumaryczny poziom transferu danych do pamięci RAM dla dwóch procesów jest dokładnie dwukrotnie większy niż poziom transferu danych dla jednego procesu (tab. 3.15). Zatem poziom transferu danych widoczny przez jeden proces jest na takim samym poziomie zarówno dla uruchomionego jednego jak i dwóch procesów (tab. 3.14). Jednak przy uruchomieniu czterech procesów poziom transferu danych widoczny przez jeden wątek wynosi jedynie 56% wartości otrzymanej dla jednego procesu. Pomimo to, można stwierdzić, że włączenie do zasobów komputera dodatkowego kontrolera prowadzi do znaczącego wzrostu maksymalnego poziomu transferu danych pomiędzy procesorem a pamięcią RAM.

Wpływ tak silnych spadków wartości transferu danych do RAM dla programów wieloprocesowych na efektywność zrównoleglenia zostanie wykazany w kolejnym punkcie.

Na podstawie przeprowadzonych pomiarów można również stwierdzić, iż poziom transferu danych do pamięci podręcznej procesorów w przybliżeniu jest równy liczbie procesów (nie większej niż liczba rdzeni) pomnożonej przez poziom transferu danych zmierzony dla programu jednoprocesowego, zarówno dla PC1 (tab. 3.12, tab. 3.13) jak i PC2 (tab. 3.14, tab. 3.15). Zatem pamięć podręczna obu procesorów dobrze współpracuje z programem

wieloprocesowym, w którym nie występuje komunikacja pomiędzy procesami.

Istotne przy tym jest, iż powyższe testy opierają się na założeniu, że poszczególne procesy nie wykonują operacji na pamięci przydzielonej przez inne procesy. Operowanie na wspólnym obszarze pamięci jest często stosowane w programach wielowątkowych, jednak wówczas, przy dostępie do jednego fragmentu obszaru pamięci podręcznej realizowanym przez wiele wątków, występuje silne ograniczenie poziomu transferu danych związane z koniecznością zastosowania protokołu MESI (ang. *Modified, Exclusive, Shared, Invalid*) [108]. W implementacji równoległej algorytmu FDTD prezentowanej w tej rozprawie procesy nie realizują działań na wspólnym obszarze pamięci, zatem założono, że wpływ wspomnianego protokołu na poziom transferu danych w badanej implementacji nie istnieje.

Tablica 3.12: Transfer danych otrzymany dla PC1, który przypada na jeden proces

Liczba procesów	L1 [GB/s]	L2 [GB/s]	RAM [GB/s]
1	14,73	11,58	3,81
2	13,05	10,14	1,55
3	12,92	10,09	1,18
4	12,97	10,14	0,81
5	10,14	9,13	0,78
6	12,34	9,74	0,73
7	11,64	9,45	0,71
8	12,18	9,68	0,66

Tablica 3.13: Sumaryczny transfer danych otrzymany dla PC1

Liczba procesów	L1 [GB/s]	L2 [GB/s]	RAM [GB/s]
1	14,73	11,58	3,81
2	26,10	20,28	3,12
3	38,75	30,27	3,53
4	51,90	40,56	3,24
5	50,71	45,65	3,90
6	74,06	58,45	4,36
7	81,49	66,14	4,95
8	97,46	77,41	5,30

Tablica 3.14: Transfer danych otrzymany dla PC2, który przypada na jeden proces

Liczba procesów	L1 [GB/s]	L2 [GB/s]	RAM [GB/s]
1	6,10	3,30	2,22
2	6,10	3,30	2,22
3	6,10	3,00	1,32
4	6,10	3,00	1,24

Tablica 3.15: Sumaryczny transfer danych otrzymany dla PC2

Liczba procesów	L1 [GB/s]	L2 [GB/s]	RAM [GB/s]
1	6,10	3,30	2,22
2	12,20	6,60	4,44
3	18,30	9,00	3,96
4	24,40	12,00	4,96

3.4.4 Testy numeryczne

Do oceny jakości zrównoleglenia algorytmów zastosowano dwa podstawowe parametry:

1. Przyspieszenie obliczeń S_p - informuje o relacji pomiędzy czasem obliczeń T_p wykonanych z zastosowaniem p procesorów jednordzeniowych lub rdzeni procesorów wieloprocesorowych oraz czasem obliczeń jednowątkowych T_1. Przyspieszenie oznaczane jest przez S_p i jego wartość wyraża r. 3.7. Przyspieszenie nazywamy liniowym jeśli $S_p = p$. W dalszej części rozprawy wartość p nazywana będzie stopniem zrównoleglenia. Jeśli badana implementacja algorytmu wykazuje przyspieszenie liniowe, to w tej rozprawie określono ją jako implementację skalowalną.

$$S_p = \frac{T_1}{T_p} \tag{3.7}$$

2. Efektywność zrównoleglenia E_p - stanowi miarę jakości zrównoleglenia algorytmu i wyraża stopień wykorzystania potencjalnych możliwości zasobów sprzętowych. W tej rozprawie efektywność zrównoleglenia wyrażona jest w procentach, zgodnie z r. (3.8). Dla algorytmów z liniowym przyspieszeniem efektywność zrównoleglenia wynosi 100%.

$$E_p = \frac{S_p}{p}100\% = \frac{T_1}{pT_p}100\% \tag{3.8}$$

3.4.4.1 Testy przeprowadzone na pojedynczym komputerze z procesorami wielordzeniowymi

Ponieważ efektywność obliczeń jednowątkowej implementacji FDTD silnie zależy od rozmiaru problemu (p. 3.3.4), to taka zależność również będzie występować przy implementacji zrównoleglonej. W punkcie tym zostanie wykazane, iż efektywność zrównoleglenia algorytmu FDTD zależy zarówno od rozmiaru badanego problemu jak i stopnia zrównoleglenia.

Przykład różnego poziomu przyspieszenia obliczeń wykonanych na PC1 dla różnych rozmiarów problemu przedstawia rys. 3.15. Jak można zauważyć na rys. 3.15 poziom przyspieszenia jest liniowy tylko dla problemów o rozmiarach $N = 2^{17}$ oraz $N = 2^{18}$, natomiast dla problemów o mniejszych rozmiarach, np. $N = 2^{12}$, lub większych, np. $N = 2^{24}$ przyspieszenie obliczeń jest dalekie od liniowego. Jest to związane z tym, że dla problemów o niewielkich rozmiarach, dla których efektywność programu jednowątkowego jest największa, czas komunikacji pomiędzy poddziedzinami jest na tyle duży, że wprowadza on znaczne zwiększenie czasu obliczeń. Natomiast dla problemów o dużych rozmiarach, przy stopniu zrównoleglenia wynoszącym p, wymagany jest p krotne zwiększenie transferu danych pomiędzy procesorem a pamięcią RAM. Jak wykazano w p. 3.4.3 nie jest to

przy zastosowaniu współczesnej technologii osiągalne. Z tego względu przyspieszenie obliczeń w przybliżeniu odpowiada rzeczywistym możliwościom zwiększenia poziomu transferu danych dla programów wieloprocesowych, które dla PC1 wynosi maksymalnie 39%, natomiast dla PC2 123%.

Z rys. 3.15 wynika, że np. dla $N = 2^{18}$, implementacja algorytmu FDTD wykazuje skalowalność. Decydujący wpływ na tą sytuację ma wzrost efektywności obliczeń jednowątkowych w miarę zmniejszania rozmiaru problemu dla $N = 2^{16}$, $N = 2^{17}$ oraz $N = 2^{18}$, zgodnie z pomiarami przedstawionymi na rys. 3.9. Wzrost efektywności wynika z procentowo większego wykorzystania danych z pamięci podręcznej L2 procesorów. Jak wynika z tab. 3.12 transfer danych pomiędzy pamięcią podręczną L2 a rdzeniami procesorów nie jest silnie ograniczany przy wzroście liczby strumieni danych wykonujących transfer do poszczególnych rdzeni procesorów. Z uwagi na to, przy podziale dziedziny obliczeniowej na wiele poddziedzin, otrzymujemy dodatkowy wzrost efektywności, który może prowadzić nawet do efektywności większej od 100%, co można zaobserwować na rys. 3.16.

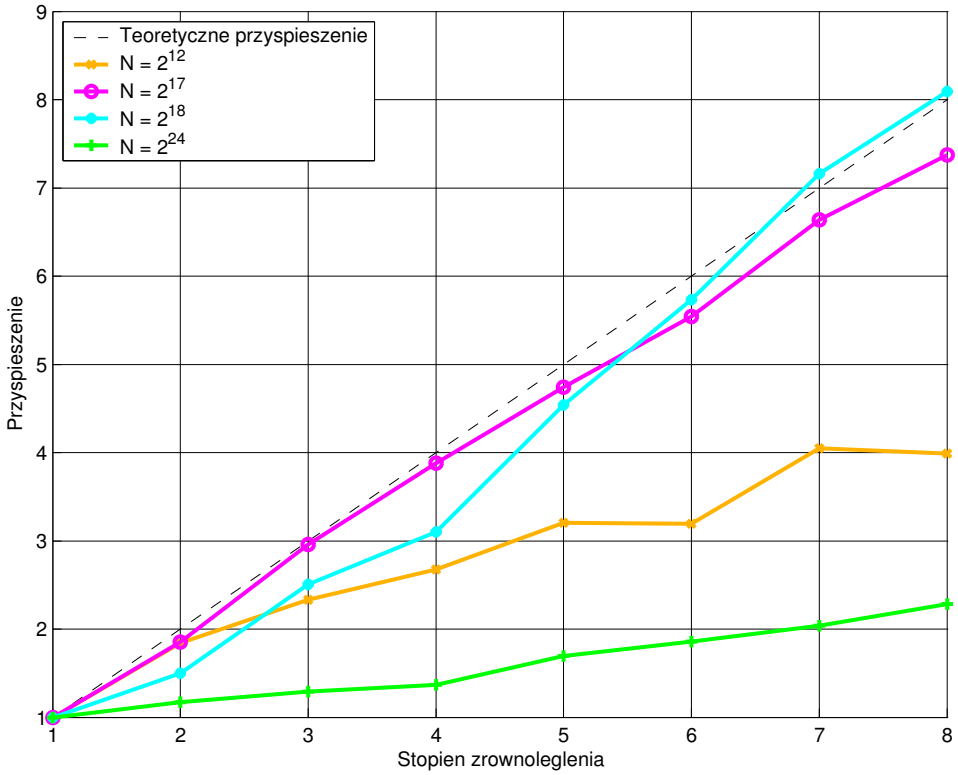
Przedstawiona na rys. 3.16 efektywność zrównoleglenia silnie zależy od rozmiaru problemu. Zakres, w którym jest ona większa od 90% jest niewielki, szczególnie dla zrównoleglenia o stopniu 4 i 8. Duża wartość efektywności zrównoleglenia w szerokim zakresie rozmiarów problemu występuje dla stopnia zrównoleglenia równego dwa.

O lokalnym charakterze występowania silnego wzrostu efektywności obliczeń świadczy rys. 3.19, który przedstawia zmianę jej wartości w funkcji rozmiaru problemu oraz stopnia zrównoleglenia dla PC1. Jak można zauważyć największa efektywność obliczeń występuje dla problemów o rozmiarze w przybliżeniu równym $N = 2^{18}$ dla stopnia zrównoleglenia wynoszącego osiem. Również w tym zakresie występuje największa efektywność zrównoleglenia, co przedstawiono na rys. 3.20.

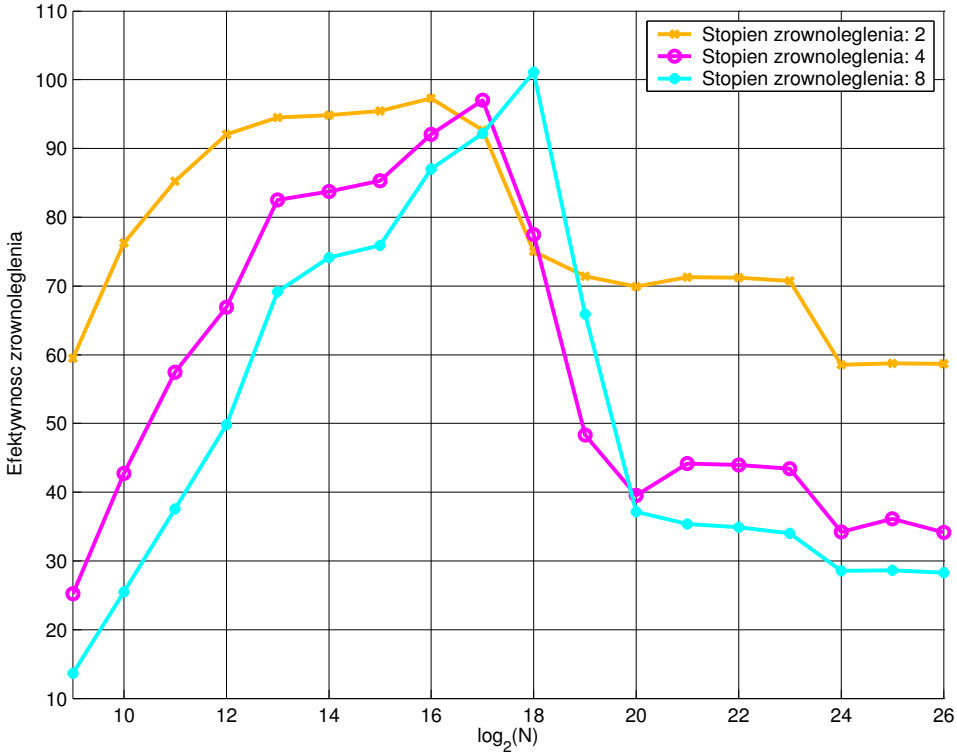
Pomiary dla PC2 wskazują na gorszą jakość zrównoleglenia algorytmu FDTD dla procesorów AMD. Wysoką efektywność zrównoleglenia (bliską 100%) uzyskano jedynie dla stopnia zrównoleglenia równego dwa (rys. 3.18), co odpowiada liczbie procesorów oraz niezależnych kontrolerów pamięci. Na rys. 3.17 jest widoczne, iż przydzielenie większej liczby procesów wykonujących obliczenia do jednego procesora może skutkować zarówno zwiększeniem przyspieszenia obliczeń ($N = 2^{15}$) jak i jego znacznym spadkiem ($N = 2^{18}$) i to zależnie również od stopnia zrównoleglenia. Zagadnienie w szerszej perspektywie przedstawia rys. 3.22, który potwierdza, że największą efektywność zrównoleglenia można uzyskać tylko dla stopnia zrównoleglenia równego dwa (stopień zrównoleglenia równy jeden oznacza program jednowątkowy). Wykres 3.21 świadczy o tym, że trudno jest przewidzieć w jakich warunkach zrównoleglenie procesorów AMD doprowadzi do największego poziomu efektywności obliczeń. Zdaniem autora tej rozprawy jest to związane z występowaniem konfliktów w pamięci podręcznej L2, które wynikają ze sposobu odwzorowania do niej pamięci wirtualnej. Dla procesorów firmy Intel nie mają one takiego znaczenia, gdyż pamięć podręczna procesorów tej firmy ma odmienne cechy: AMD stosuje najczęściej pamięć podręczną z własnością ang. *exclusive cache*, natomiast Intel z własnością ang. *inclusive cache*, [108].

Podsumowując, dla implementacji zrównoleglonej, dla różnych rozmiarów problemu skalowalność będzie występować warunkowo:

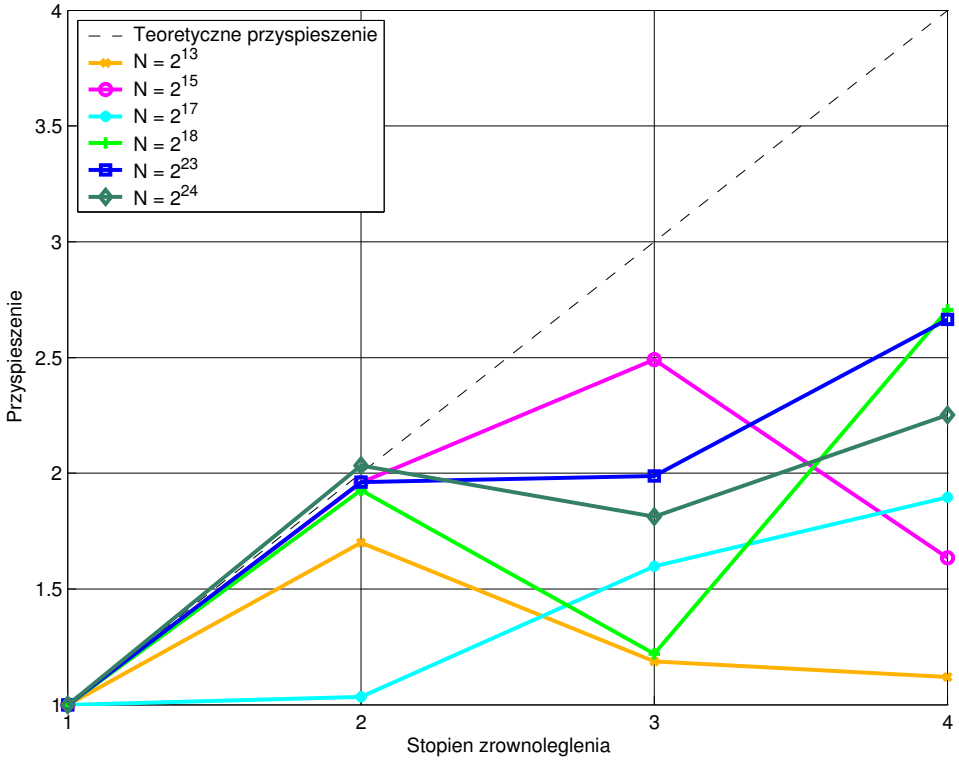
- Jeśli problem początkowy (dla implementacji jednowątkowej), będzie na tyle duży, że obliczenia będą wymagały silnego wykorzystania pamięci RAM oraz na tyle mały, aby po podziale na poddziedziny otrzymać problem, dla którego implementacja jednowątkowa posiada znacząco większą efektywność obliczeń niż dla problemu po-



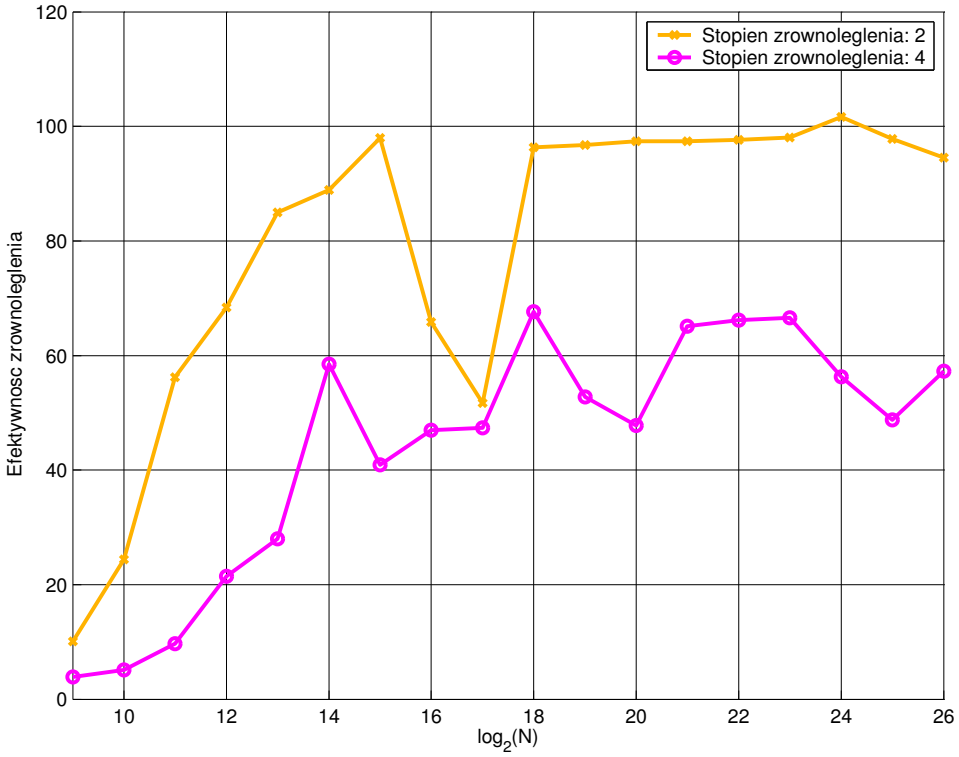
Rysunek 3.15: Przyspieszenie obliczeń w zależności od stopnia zrównoleglenia dla wybranych rozmiarów problemu zmierzone dla PC1



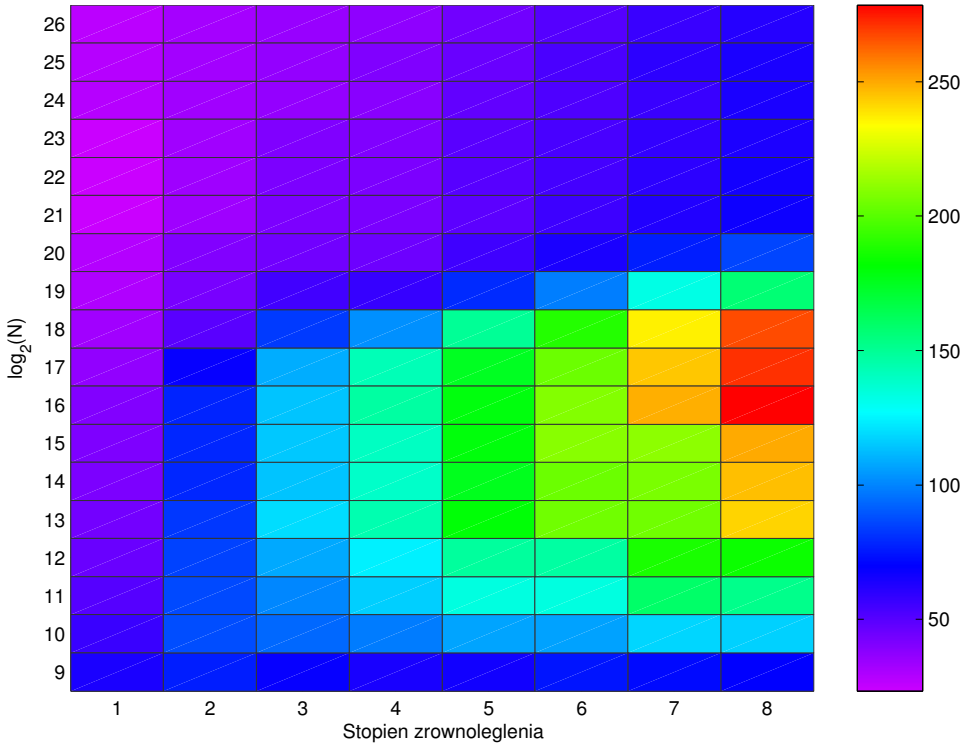
Rysunek 3.16: Efektywność zrównoleglenia w zależności od rozmiaru problemu dla wybranych stopni zrównoleglenia zmierzona dla PC1



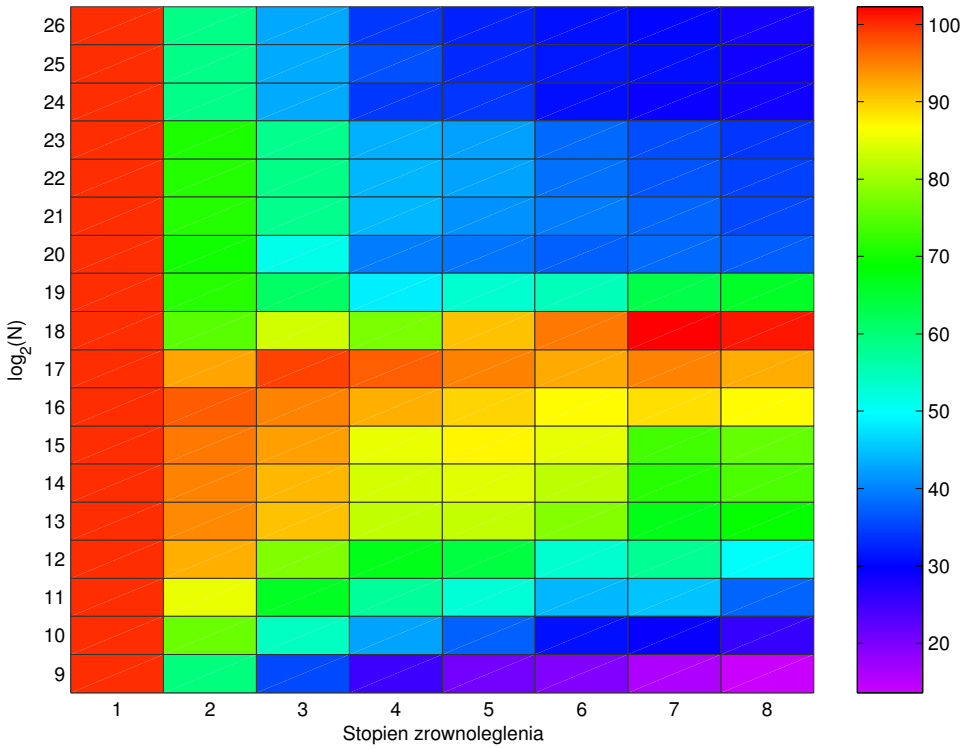
Rysunek 3.17: Przyspieszenie obliczeń w zależności od stopnia zrównoleglenia dla wybranych rozmiarów problemu zmierzone dla PC2



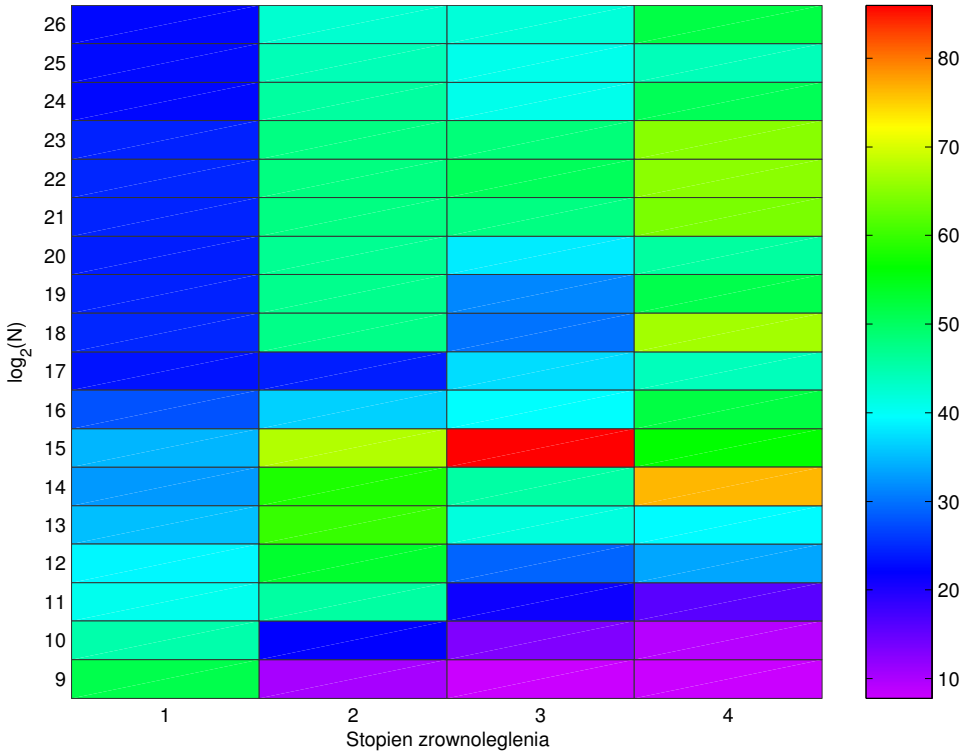
Rysunek 3.18: Efektywność zrównoleglenia w zależności od rozmiaru problemu dla wybranych stopni zrównoleglenia zmierzona dla PC2



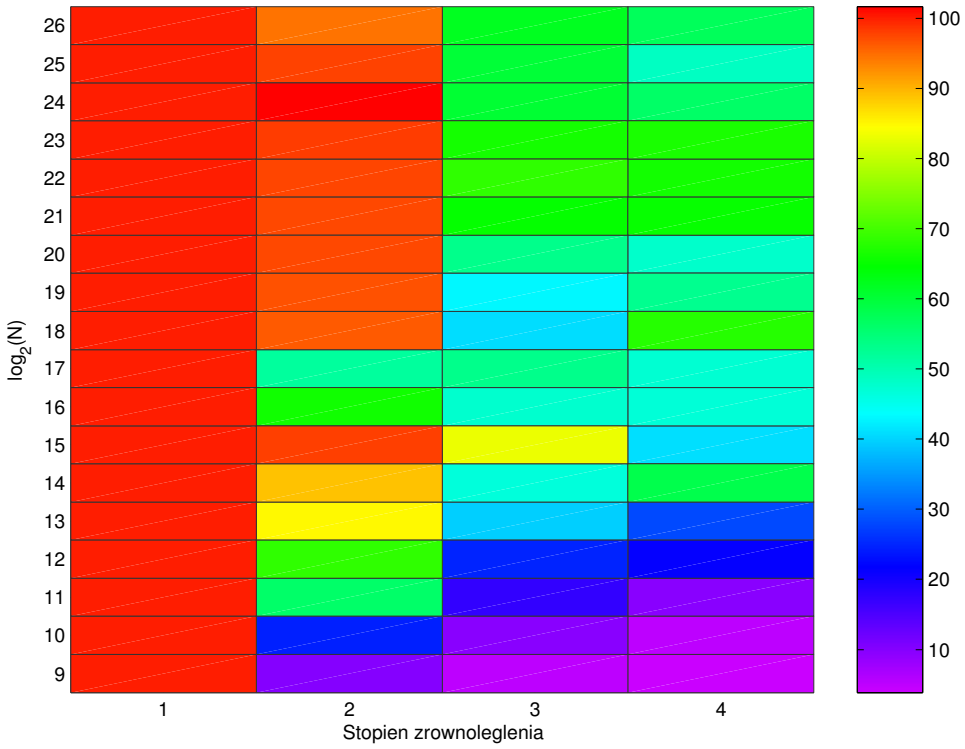
Rysunek 3.19: Efektywność obliczeniowa S_C [Mcells/s] w zależności od rozmiaru problemu oraz stopnia zrównoleglenia zmierzone dla PC1



Rysunek 3.20: Efektywność zrównoleglenia w zależności od rozmiaru problemu oraz stopnia zrównoleglenia zmierzona dla PC1



Rysunek 3.21: Efektywność obliczeniowa S_C [Mcells/s] w zależności od rozmiaru problemu oraz stopnia zrównoleglenia zmierzone dla PC2



Rysunek 3.22: Efektywność zrównoleglenia w zależności od rozmiaru problemu oraz stopnia zrównoleglenia zmierzona dla PC2

czątkowego, to wówczas wzrost efektywności będzie spowodowany zarówno poprzez wykonywanie algorytmu równolegle jak i przez wzrost efektywności obliczeń dla problemów o mniejszych rozmiarach. Wówczas przyspieszenie obliczeń będzie liniowe.
- Jeśli zarówno problem początkowy jak i rozmiar problemu po podziale na poddziedziny będzie wymagać w obliczeniach silnego wykorzystania pamięci RAM, to skalowalność nie będzie występować z uwagi na ograniczoną przepustowość szyny systemowej łączącej procesor(y) z pamięcią RAM.
- Jeśli problem początkowy jest małego rozmiaru, to czas komunikacji pomiędzy procesami będzie na tyle znaczący, że skalowalność nie będzie występować.

3.4.4.2 Testy przeprowadzone na superkomputerze Galera

Zaimplementowany przez autora rozprawy równoległy algorytm FDTD umożliwia również przeprowadzenie obliczeń na wielu węzłach obliczeniowych. W celu oszacowania efektywności zrównoleglenia dla klastra o znacznej mocy obliczeniowej symulacje FDTD przeprowadzono na superkomputerze Galera, który należy do Centrum Informatycznego Trójmiejskiej Akademickiej Sieci Komputerowej (CI TASK). W dotychczasowych testach w obliczeniach stosowano jeden węzeł tego klastra i oznaczono go w tej rozprawie jako PC1. Galera składa się z 672 takich węzłów, czyli umożliwia przeprowadzenie obliczeń na 5376 rdzeniach procesorów.

Do przeprowadzenia symulacji wybrano problem o rozmiarze $336 \times 336 \times 336$, $N = 37\,933\,056$. W każdej symulacji dziedzina obliczeniowa była dzielona na N_d poddziedzin, przy czym podział wzdłuż osi x, y, z wynosił odpowiednio $I_d \times J_d \times K_d$, ($N_d = I_d J_d K_d$). Podział dziedziny obliczeniowej w poszczególnych symulacjach określa tab. 3.16.

Program poddany testom umożliwia zarezerwowanie do obliczeń większej liczby rdzeni niż wynosi liczba poddziedzin obliczeniowych. W momencie zaistnienia takiej sytuacji program dobiera tak rozkład procesów, aby w poszczególnych węzłach w obliczeniach brała udział jak najmniejsza liczba rdzeni. Dzięki temu można zmniejszyć wpływ obniżenia efektywności zrównoleglenia, który występuje przy wykonywaniu obliczeń na procesorach wielordzeniowych, co wykazano w p. 3.4.4.1. Średnia liczba rdzeni, które wykonują obliczenia w węźle zawierającym osiem rdzeni, została podana dla każdej symulacji w tab. 3.16.

Zmierzone przyspieszenie prezentuje wykres 3.23. Na podstawie tego wykresu można stwierdzić, iż przyspieszenie dla stopnia zrównoleglenia (tożsamego z liczbą poddziedzin) mniejszego od 512 jest bliskie liniowemu. Pomiar ten potwierdza zatem, że niska efektywność zrównoleglenia algorytmu FDTD zmierzona dla procesorów wielordzeniowych wynika z ograniczeń związanych z architekturą tych procesorów, a nie z własności implementacji algorytmu. Zjawisko ograniczenia efektywności zrównoleglenia związane z przydzieleniem obliczeń do wielu rdzeni procesora wielordzeniowego znajduje i tym razem potwierdzenie na wykresie 3.23 dla stopni zrównoleglenia większych od 512. W tym zakresie stopnia zrównoleglenia wartość przyspieszenia daleka jest od poziomu przyspieszenia liniowego, co wynika ze wzrostu średniej liczby aktywnych rdzeni w węźle (tab. 3.16) oraz wzrostu czasu komunikacji pomiędzy poszczególnymi poddziedzinami.

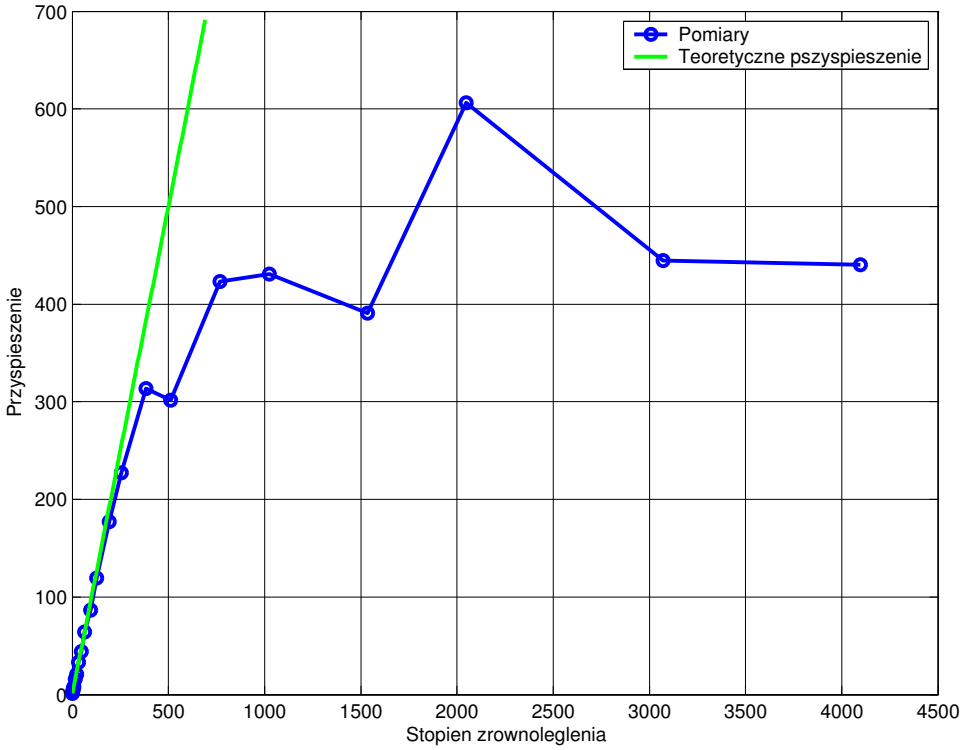
Na rys. 3.24 przedstawiono poziom efektywności zrównoleglenia otrzymany dla stopni zrównoleglenie mniejszych od 512. Wartość efektywności jest większa od 90% dla stopni zrównoleglenia mniejszych od 200, co można uznać za zadowalający rezultat [31, 39, 110, 122] (wykazana w podanych artykułach efektywność zrównoleglenia zależna jest od roz-

miaru analizowanych problemów i wynosi od 60% do 95%), który świadczy o dobrej jakości algorytmu zastosowanego do przeprowadzenia testów.

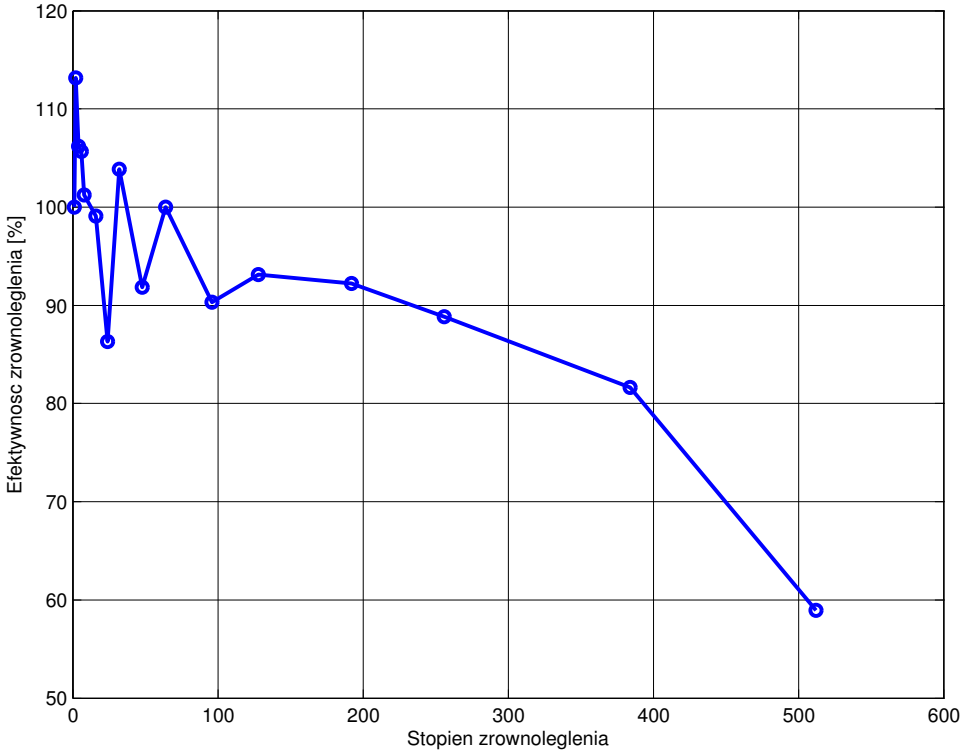
Skalę zmierzonego przyspieszenia uzyskanego dzięki zrównoległeniu algorytmu FDTD można ocenić poprzez porównanie wartości parametrów otrzymanych przy symulacji o największej efektywności zrównoleglenia (symulacja numer 20) oraz symulacji jednowątkowej: osiągnięto 610-krotny wzrost efektywności obliczeń z poziomu 21 Mcells/s dla symulacji jednowątkowej do poziomu 12806 Mcells/s dla symulacji równoległej, co oznacza spadek czasu symulacji z 29 minut 55 sekund do 2,96 sekundy. Korzyść wynikająca ze zrównoleglenia algorytmu FDTD jest zatem oczywista.

Tablica 3.16: Wybrane parametry symulacji przeprowadzonych na superkomputerze Galera, które zastosowano do pomiaru szybkości działania implementacji zrównoleglonego algorytmu FDTD

Numer symulacji	I_d	J_d	K_d	N_d	średnia liczba aktywnych rdzeni w węźle
1	1	1	1	1	1
2	1	2	1	2	1
3	1	2	2	4	1
4	1	2	3	6	1
5	1	2	4	8	1
6	1	4	4	16	1
7	2	3	4	24	1
8	2	4	4	32	1
9	2	4	6	48	1
10	4	4	4	64	1
11	2	6	8	96	1
12	2	8	8	128	1
13	4	6	8	192	1
14	4	8	8	256	1
15	4	8	12	384	1
16	4	8	16	512	1
17	4	12	16	768	1,5
18	8	8	16	1024	2
19	8	12	16	1536	3
20	8	16	16	2048	4
21	12	16	16	3072	6
22	8	16	32	4096	8



Rysunek 3.23: Przyspieszenie obliczeń w zależności od stopnia zrównoleglenia zmierzone dla klastra Galera

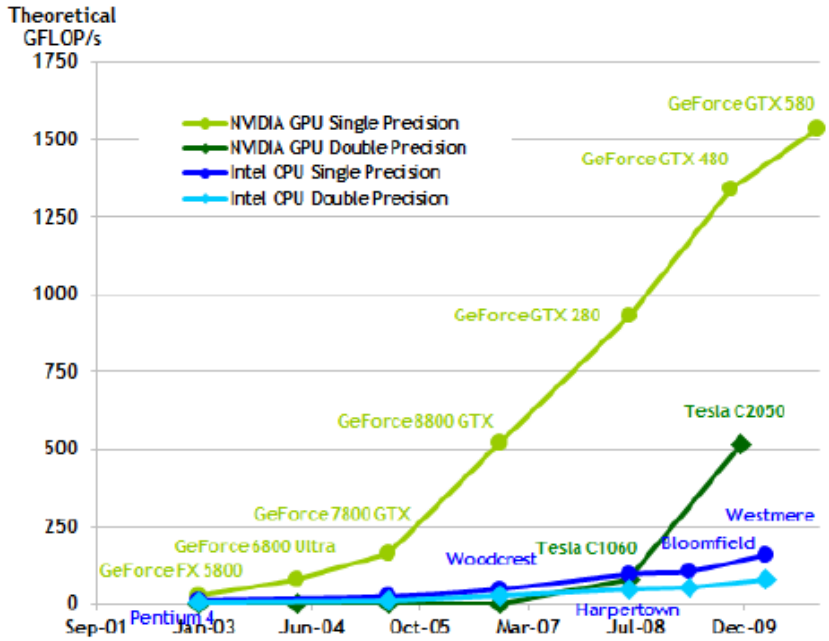


Rysunek 3.24: Efektywność zrównoleglenia w zależności od stopnia zrównoleglenia zmierzona dla klastra Galera

3.5 Efektywność obliczeń wykonywanych na GPU

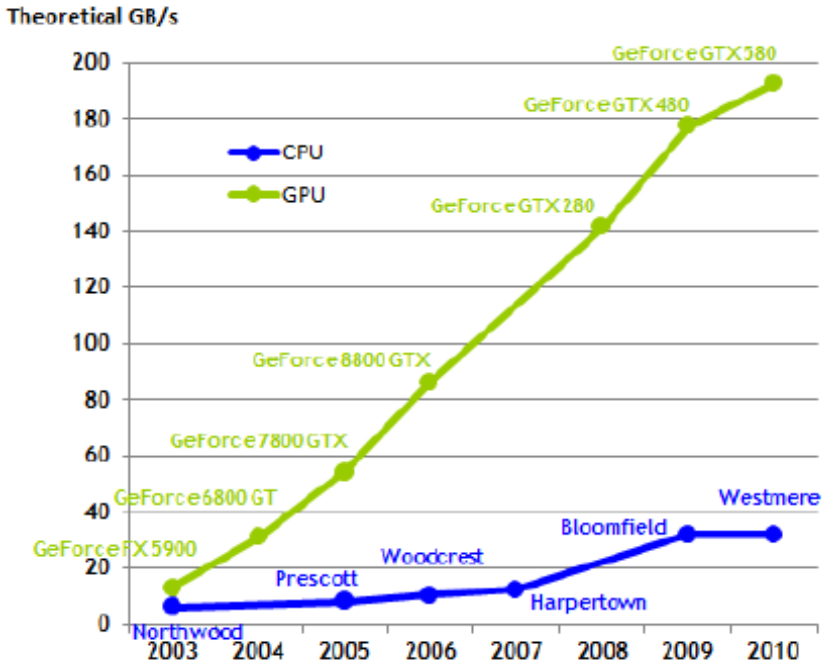
Kolejną architekturą sprzętową rozważaną w niniejszej rozprawie są akceleratory graficzne.

3.5.1 Uzasadnienie wyboru akceleratora graficznego jako jednostki obliczeniowej



Rysunek 3.25: Porównanie szybkości wykonywania operacji zmiennoprzecinkowych CPU oraz GPU, [69]

W poprzedniej części rozdziału przedstawiono testy numeryczne, które realizowały obliczenia na procesorach komputerowych. Jest to zgodne z popularną opinią, że podstawowym źródłem mocy obliczeniowej komputerów są procesory komputerowe. Opinia ta ma swoje uzasadnienie historycznie, gdyż od początku lat osiemdziesiątych ich dynamiczny rozwój prowadził do podwojenia mocy obliczeniowej komputerów co osiemnaście miesięcy, zgodnie z prawem Moore'a. Mimo tak znaczącego postępu, w ostatnim czasie status najpotężniejszego źródła mocy obliczeniowej straciły one na rzecz akceleratorów graficznych. W chwili obecnej najwydajniejszy procesor komputerowy, dziesięciordzeniowy Intel Xeon E7-8870 2.4GHz, teoretycznie jest w stanie osiągnąć 192 Gflops/s dla zmiennych w pojedynczej precyzji. Natomiast najwydajniejszy akcelerator graficzny, produkowany przez firmę Nvidia GeForce GTX 580, charakteryzuje się wydajnością zmiennoprzecinkową równą 1581 Gflops/s dla zmiennych w pojedynczej precyzji. Tak znacząca różnica efektywności obu jednostek obliczeniowych skłania do zastosowanie akceleratorów graficznych w intensywnych obliczeniach numerycznych. Dodatkowo za takim wyborem przemawia szybki wzrost ich wydajności - moc obliczeniowa akceleratorów graficznych podwaja się co dwanaście miesięcy, więc są one bardziej perspektywicznym źródłem mocy obliczeniowej (rys. 3.25, [69]).
Ponadto, istotnym parametrem jednostek obliczeniowych jest ich poziom transferu danych do pamięci zewnętrznej RAM. Również w tym zakresie przewaga akceleratorów graficznych nad procesorami komputerowymi jest znacząca, co ilustruje rys. 3.26.



Rysunek 3.26: Porównanie maksymalnego poziomu transmisji danych osiągalnego z CPU oraz z GPU, [69]

Pomimo, że od wielu lat akceleratory graficzne są konkurencyjnym wobec procesorów komputerowych źródłem mocy obliczeniowej, to jednak do 2006 roku ich popularność w zastosowaniach niezwiązanych z grafiką była znikoma. Taki stan rzeczy wynikał przede wszystkim z ograniczonej funkcjonalności akceleratorów. Jeszcze w roku 2005 istniały zasadnicze ograniczenia w zastosowaniu kart graficznych do analizy numerycznej:

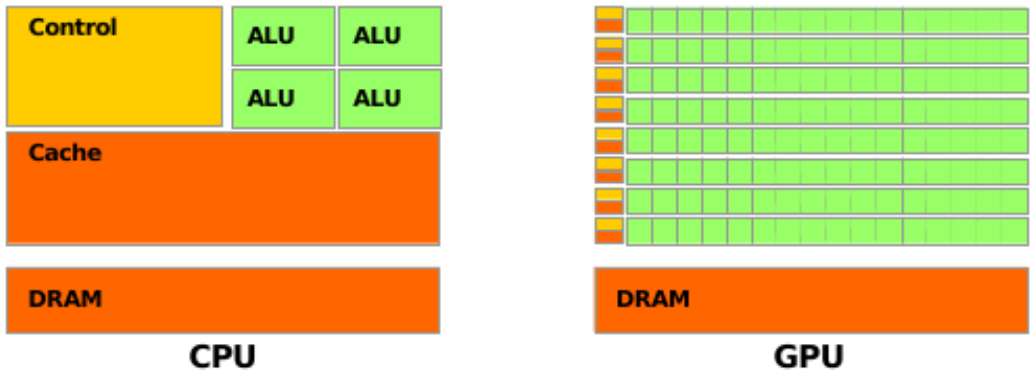
- górna granica pojemności pamięci RAM kart graficznych wynosiła 256 MiB,
- metody implementacji algorytmów przeznaczonych dla kart graficznych były skomplikowane: implementacja wymagała zastosowania bibliotek przeznaczonych do przetwarzania grafiki komputerowej, np. OpenGL (ang. *Open Graphics Library*), DirectX,
- architektura akceleratorów charakteryzowała się małą elastycznością, m.in. występował brak możliwości dowolnego dostępu do pamięci RAM, brak wpływu na niskopoziomowe parametry realizowanych obliczeń, np. liczbę wątków, podział pamięci wewnętrznej akceleratora pomiędzy wątki.

W roku 2006 nastąpił przełom w architekturze akceleratorów graficznych, gdyż firma Nvidia wprowadziła na rynek technologię CUDA (ang. *Compute Unified Device Architecture*) [68], która zniwelowała większość dotychczasowych ograniczeń funkcjonalnych akceleratorów graficznych. Technologia ta pozwala na zaimplementowanie algorytmów przeznaczonych dla akceleratorów w krótszym czasie, przy mniejszej komplikacji kodu i często z większą efektywnością niż to miało miejsce przy implementacji algorytmów dla akceleratorów bez zastosowania tej technologii. Z uwagi na to akceleratory stały się dużo bardziej dostępne dla obliczeń numerycznych. Również prezentowana w tej rozprawie implementacja algorytmu FDTD dla GPU została opracowana w technologii CUDA.

Równocześnie znacząco zwiększono wielkość pamięci współpracującej z akceleratorami. W roku 2005, w celu uzyskania całkowitej pamięci RAM o pojemności 1 GiB, należało połączyć ze sobą co najmniej cztery karty graficzne. Testy zaprezentowane w tej rozprawie przeprowadzono z zastosowaniem karty graficznej GTX 580 firmy Nvidia, która zawiera pamięć RAM o rozmiarze 1,5 GiB. Dokonany postęp technologiczny umożliwia zatem osiągnięcie znacznej mocy obliczeniowej przy zastosowaniu akceleratorów graficznych.

3.5.2 Cechy architektury akceleratorów graficznych oraz model programowania

Akceleratory w komputerze spełniają rolę koprocesora. Ich znaczna moc obliczeniowa ma swoje źródło w zastosowaniu wielu rdzeni w jednym akceleratorze, które zoptymalizowane są do przetwarzania ograniczonej w porównaniu do CPU liczby instrukcji. Ograniczenie funkcjonalności pojedynczego rdzenia pozwala na efektywne zarządzanie przepływem informacji przy zastosowaniu przetwarzania równoległego, a przez to na umieszczenie w jednym akceleratorze znacznej liczby rdzeni. Podstawowe różnice w architekturze GPU oraz CPU przedstawia rys. 3.27. Rysunek ten pochodzi z [68] i jak można zauważyć firma Nvidia podkreśliła w nim, że w akceleratorach graficznych przeznaczono znacznie mniej miejsca na pamięć podręczną oraz zarządzanie przetwarzaniem instrukcji i danych, a przez to więcej miejsca przydzielono jednostkom arytmetyczno-logicznym, ALU (ang. *Arithmetic Logic Unit*), co stanowi źródło mocy obliczeniowej GPU.



Rysunek 3.27: Podstawowe różnice architektury CPU oraz GPU, [68]

Dużą zaletę technologii CUDA stanowi metoda implementacji programów przeznaczona na akceleratory kompatybilne z tą technologią (w dalszej części pracy przez pojęcia GPU oraz akcelerator graficzny rozumiane będą urządzenia kompatybilne z technologią CUDA). Wspomniana implementacja budowana jest w języku C nieznacznie wzbogaconym przez komendy charakterystyczne dla CUDA, co oznacza, że programista łatwo może opanować sposób przetwarzania danych na GPU. Uruchomienie obliczeń na akceleratorze sprowadza się do wywołania jednej funkcji, zwanej w CUDA kernelem, która stanowi część większego programu napisanego w C/C++. Istotnym wymogiem budowania programu współpracującego z CUDA jest konieczność uwzględnienia koncepcji architektury sprzętowej akceleratorów, w celu uzyskania wysokiej wydajności obliczeń.

Ponieważ liczba rdzeni, które realizują obliczenia równolegle, liczona jest w setkach, np. GeForce GTX 580 posiada ich 512, to efektywna implementacja algorytmów, dla nich przeznaczonych, opiera się na przetwarzaniu równoległym. Sposób zarządzania wątkami jest ściśle związany z architekturą akceleratorów graficznych, a przez to zostanie

ona tutaj krótko opisana. Podstawowym składnikiem akceleratora graficznego jest multiprocesor (ang. *Streaming Multiprocessor*, SM), co przedstawiono na rys. 3.28. Obliczenia w multiprocesorze realizowane są przez rdzenie (ang. *Scalar Processor*, SP), których liczba zależna jest od wersji architektury (ang. *compute capability*) akceleratora i wynosi 8, 32 lub 48 odpowiednio dla wersji architektury 1.x, 2.0 i 2.1, [69]. Każdy rdzeń akceleratora może współpracować z sześcioma rodzajami pamięci, [68], przy czym w opisanych w tej rozprawie programach intensywnie optymalizowano działanie na trzech z nich:

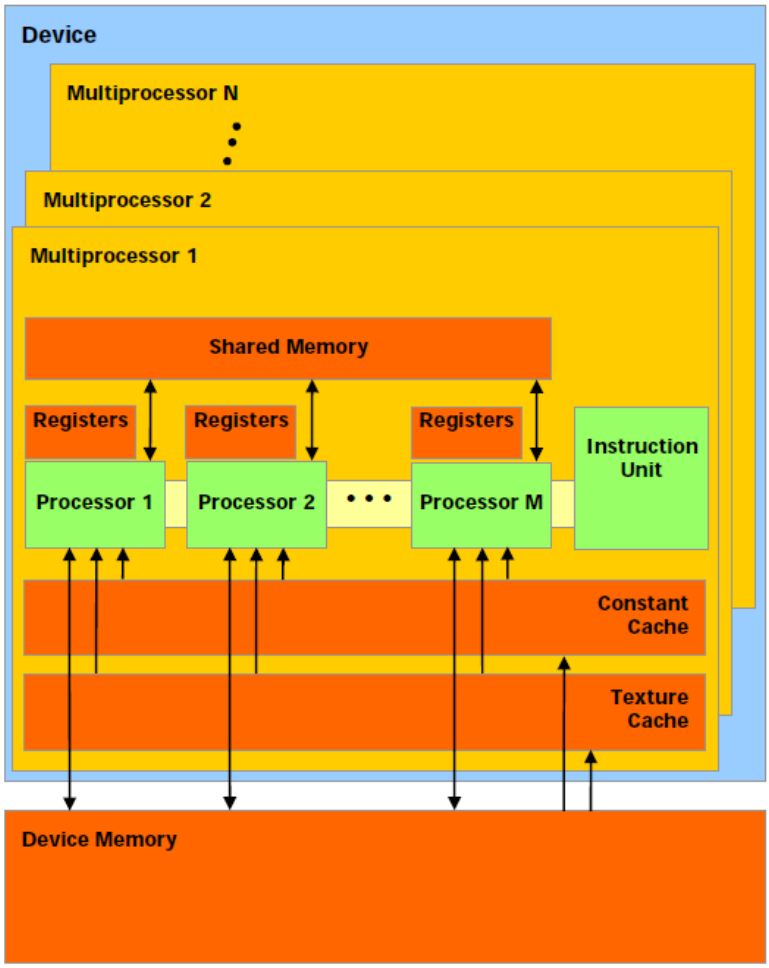
1. pamięć globalna - pamięć RAM, która znajduje się na karcie graficznej, na zewnątrz akceleratora, do której dostęp mają wszystkie wątki programu uruchomionego na GPU. Maksymalny poziom transferu danych dla tego typu pamięci jest najmniejszy spośród wszystkich rodzajów pamięci karty graficznej współpracujących z GPU.
2. pamięć dzielona (ang. *shared*) - pamięć wewnętrzna akceleratora, która stanowi część multiprocesora. Dostęp do niej posiadają wszystkie wątki uruchomione na danym multiprocesorze, przez co możliwa jest szybka komunikacja pomiędzy nimi. Nie jest możliwy natomiast dostęp do pamięci dzielonej wątku uruchomionego na innym multiprocesorze.
3. rejestry - najszybszy rodzaj pamięci wewnętrznej akceleratora. Służą do realizacji zadań pojedynczego wątku i dostęp do jednego rejestru nie może być współdzielony przez grupę wątków.

Każdy multiprocesor pracuje w trybie SIMT (ang. *Single-Instruction, Multiple-Thread*). Termin SIMT opisuje własność technologii CUDA, która umożliwia realizowanie przez pojedynczy wątek zarówno instrukcji przydzielonych grupie wątków jak i instrukcji określonej do indywidualnej realizacji przez pojedynczy wątek. Odróżniono w ten sposób SIMT od SIMD (ang. *Single Instruction Multiple Data*), które zawsze wymusza uwzględnienie w kodzie programu wektorowej organizacji przetwarzania instrukcji. Maksymalna liczba równolegle działających wątków przypadająca na jeden multiprocesor wynosi 768 dla wersji architektury 1.x oraz 1024 dla wersji 2.x. W każdym takcie zegara jeden rdzeń multiprocesora może wykonać jedną operację zmiennoprzecinkową (dodawania, mnożenia lub równoczesnego dodawania połączonego z mnożeniem) operując na zmiennych o pojedynczej precyzji. Dla zmiennych o podwójnej precyzji rdzeń multiprocesora w architekturze 2.0, potrzebuje dwóch taktów zegara do przeprowadzenia operacji zmiennoprzecinkowej na zmiennych o podwójnej precyzji (rezultaty symulacji przeprowadzonych w tej rozprawie wykonywane są na karcie GTX 580, która posiada akcelerator w wersji architektury 2.0).

Fizyczne własności architektury kart graficznych determinują sposób zarządzania wątkami. Wątki zgrupowane są na dwóch poziomach (rys. 3.29):

1. każdy wątek należy do bloku wątków (ang. *thread block*),
2. bloki wątków umieszczono w siatce bloków (ang. *grid of thread blocks*), która realizuje zadania kernela.

Jeden multiprocesor może zarządzać maksymalnie ośmioma blokami wątków równocześnie. Współdzielenie pamięci dzielonej możliwe jest jedynie pomiędzy wątkami, które należą do jednego bloku wątków. Synchronizacja wątków odbywa się na poziomie bloku wątków, jednak synchronizacja pomiędzy blokami wątków wewnątrz kernala nie jest możliwa. Jedynym sposobem na synchronizację działań wszystkich wątków stanowi synchronizacja po zakończeniu działania kernela. Efektywne implementacje kernela wymagają, aby rozmiar bloku wątków należał do zbioru {96, 128, 192, 256, 384}.

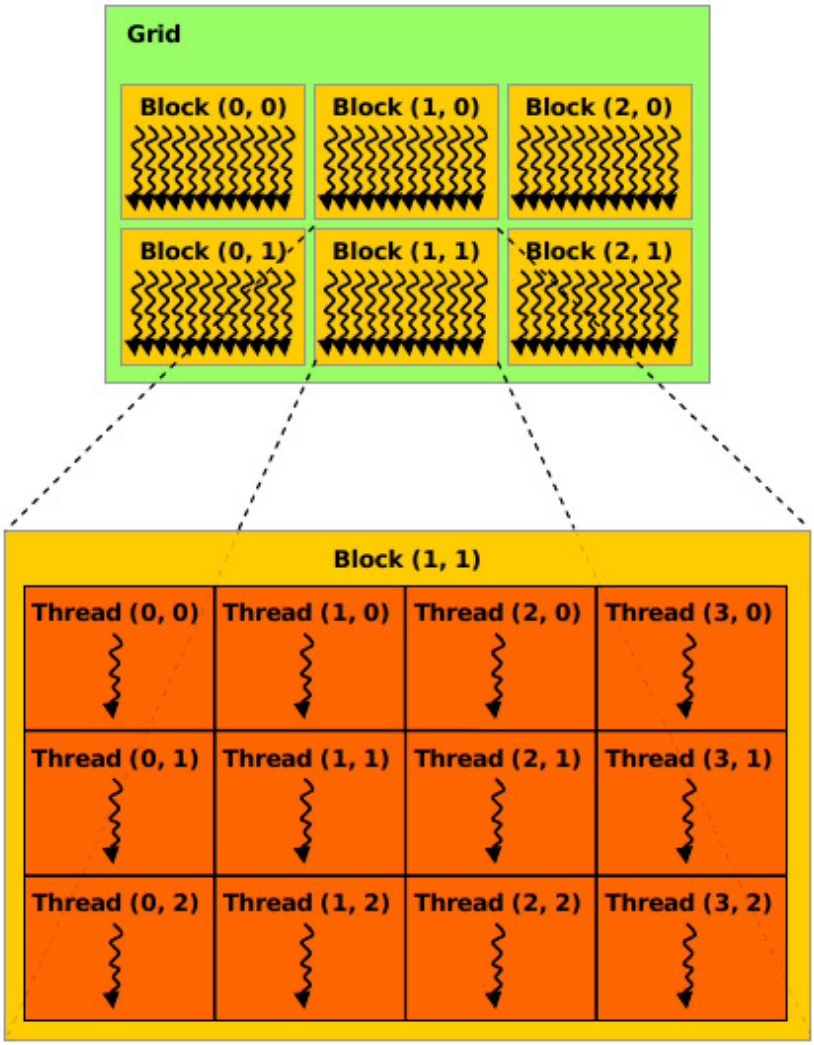


Rysunek 3.28: Główne cechy architektury akceleratorów graficznych zgodnych z technologią CUDA, [68]

W obliczeniach, których rezultaty przedstawiono w tej rozprawie, zastosowano kartę GTX 580 firmy Nvidia. Zawiera ona 16 multiprocesorów i współpracuje z 1,5 GB pamięci RAM. Na każdy multiprocesor przypada 48 KiB pamięci dzielonej oraz 32 KiB rejestrów. Teoretyczna szybkość transmisji danych pomiędzy GPU a zainstalowaną na karcie graficznej pamięcią RAM wynosi 192,4 MB/s (szyna danych ma szerokość 384 bitów i jest taktowana sygnałem o częstotliwości 1002 MHz, pamięć jest typu DDR5). Przy częstotliwości taktowania akceleratora graficznego równej 772 MHz teoretyczna maksymalna moc obliczeniowa tej karty wynosi 1581 Gflops/s.

3.5.3 Zrównoleglenie algorytmu jawnego FDTD dla akceleratorów graficznych

Programowanie akceleratorów graficznych różni się znacząco od programowania procesorów komputerowych. W celu podkreślenia różnic w implementacji algorytmu FDTD w formie jawnej dla GPU oraz dla CPU opisano sposób transformacji kodu przeznaczonego dla CPU do kodu przeznaczonego dla GPU. Kolejne etapy transformacji charakteryzują się zwiększeniem efektywności obliczeń na GPU oraz coraz lepszym dostosowaniem implementacji do specyfiki architektury akceleratorów graficznych. Tekst zawarty w tym



Rysunek 3.29: Struktura wątków w akceleratorach graficznych zgodnych z technologią CUDA [68]

punkcie stanowi rozszerzenie opisu zagadnień z artykułów [26, 101]. Istotne jest przy tym, że opisane optymalizacje kodu programu przeprowadzono dla akceleratorów graficznych w wersji architektury 1.x. Akceleratory w wersji architektury 2.x posiadają pamięć podręczną pierwszego (L1) i drugiego (L2) poziomu, co prowadzi do znacznego wzrostu szybkości transferu danych pomiędzy akceleratorem a pamięcią RAM. Opisane w tym punkcie testy przeprowadzono na karcie graficznej Quadro FX 5600 (wersja architektury 1.0), z wyjątkiem p. 3.5.3.7, w którym udokumentowano rezultaty obliczeń dla karty GTX 580. Przedstawione optymalizacje kodu mają na celu uwydatnienie specyfiki programowania dla akceleratorów graficznych, wskazanie na znaczny postęp technologiczny, który wprowadza na rynek firma Nvidia, oraz przedstawienie podstaw implementacji algorytmu FDTD w wersji mieszanej, którą opisano w p. 4.5.

3.5.3.1 Tradycyjna metoda zrównoleglenia algorytmu FDTD zastosowana dla GPU

Stan początkowy kodu stanowi implementacja algorytmu FDTD w postaci jawnej przedstawiona na wydruku 3.1. Proces zmian wykonanych na tym kodzie zaprezentowano na fragmencie kodu zasadniczego, który wyznacza wartości próbek składowej E_x. Fragment ten umieszczono na wydruku 3.4. Zmienne I, J, K oznaczają liczbę komórek siatki Yee w kierunku odpowiednio x, y, z, natomiast $IJ = I \cdot J$. Dyskretyzacja przestrzeni jest taka sama dla każdego kierunku przestrzeni. Informacja o przenikalność elektrycznej dla każdej komórki Yee została zawarta w tablicy `cex`.

Ocenę efektywności implementacji algorytmu dla GPU otrzymano poprzez porównanie z efektywnością obliczeń jednowątkowych przeprowadzonych dla procesorów (patrz p. 3.3.4), której wartość, na podstawie pomiarów przedstawionych na rys. 3.9, przyjęto jako 30 Mcells/s. Wartość ta różni się od poziomu odniesienia równego 9,9 Mcells/s, który ustalono w [26, 101], z uwagi na słabsze parametry zestawu komputerowego zastosowanego w testach zaprezentowanych w w/w artykułach.

Pierwszy etap transformacji implementacji algorytmu FDTD przeznaczonej dla CPU polega na przekształceniu implementacji jednowątkowej w wielowątkową. Pośredni wynik tej transformacji stanowi uproszczenie algorytmu zastosowanego do zrównoleglenia obliczeń algorytmu FDTD przeznaczonego dla procesorów komputerowych, który został opisany w punkcie 3.4. Uproszczenie to sprowadza się do podziału dziedziny obliczeniowej tylko wzdłuż płaszczyzn $z = const$. Zastosowane uproszczenie pozwala na wskazanie istotnych ograniczeń zastosowanej metody transformacji kodu, które słuszne są również w sytuacji zrównoleglenia uniwersalnego, w którym podział dziedziny obliczeniowej występuje w każdym kierunku przestrzeni.

Wstępne przejście od implementacji jednowątkowej przeznaczonej dla CPU do implementacji wielowątkowej przeznaczonej dla GPU stanowi kod przedstawiony na wydruku 3.3. Algorytm ten wzorowany jest na sposobie zrównoleglenia algorytmu FDTD w środowisku stosującym do obliczeń procesory komputerowe [31, 110]. Zasada jego realizacji opiera się na podziale dziedziny obliczeniowej na zbiór poddziedzin, które stanowią wyodrębnione obszary przestrzeni dyskretnej. Obliczenia wartości próbek, które należą do danej poddziedziny, są wykonywane przez przydzielony do niej rdzeń procesora. Ten sposób zrównoleglenia algorytmu nazywany jest zrównolegleniem gruboziarnistym, gdyż jeden wątek wykonuje dużo obliczeń oraz liczba wątków jest stosunkowo niewielka.

Rozmiar dziedziny obliczeniowej wybrany do testów został tak zdefiniowany, aby umożliwić przeprowadzenie testów dla znacznej (jak dla takiej wersji algorytmu) liczby wątków. Rozmiar ten został ustalony na [16,16,4000] komórek siatki Yee w kierunku odpowiednio x, y, z.

Specyfika GPU wymaga, aby do obliczeń zastosować znaczną liczbę wątków. Jednak zaprezentowany kod pozwala na przeprowadzenie oceny wpływu liczby wątków na efektywność obliczeń również dla małej liczby wątków, od pojedynczego wątku zaczynając. Taka możliwość istnieje dzięki selekcji ze znacznej liczby wątków uruchomionych na GPU tylko pewnej ich części w celu wykonania przez nie obliczeń. Wątki wykonujące obliczenia zostały tu określone jako wątki aktywne. Ich liczbę można ustalić poprzez nadanie wartości zmiennej `TTN` widocznej na wydruku 3.3.

Wpływ zmiany liczby wątków na efektywność obliczeń przedstawia rys. 3.30. Warto przy tym podkreślić, iż obliczenia na GPU wykonane tylko przez jeden wątek pozwalają osiągnąć efektywność obliczeń na poziomie jedynie 0,118 Mcells/s, co stanowi wynik około

255 razy mniejszy w porównaniu do efektywności zmierzonej dla CPU. Jednak wraz ze wzrostem liczby aktywnych wątków efektywność obliczeń szybko wzrasta. Dla małej liczby wątków (nie większej niż 64) obserwowana jest skalowalność zastosowanej implementacji algorytmu FDTD. Jednak ten trend utrzymany jest jedynie do pewnego poziomu, po którym wzrost liczby aktywnych wątków nie wpływa na efektywność obliczeń. Wynika to ze słabego dostosowania zastosowanej implementacji do specyfiki akceleratorów graficznych.

Opisany sposób zrównoleglenia algorytmu FDTD umożliwia osiągnięcie wysokiej efektywności obliczeń przy zastosowaniu w obliczeniach procesorów komputerowych i jak wykazano jest on również możliwy do dostosowania do środowiska GPU. Jednak rozwiązanie, w którym jeden wątek wykonuje znaczną część obliczeń nie jest to optymalne dla akceleratorów graficznych. W kolejnych punktach zostaną wskazane rozwiązania, które prowadzą do zwiększenia efektywności obliczeń na GPU.

```
__global__ void calcE( float * ex, float * ey, ..., int   * zrange ){
 // NBL - number of thread blocks
 // TTN - number of active threads
 int id = threadIdx.x * NBL + blockIdx.x;
 if( id >= TTN )    return;
 int zb = zrange[ id ],
 int ze = zrange[ id + 1 ],
 // ...
 for( s=zb*IJ, z=zb; z<ze; z++, s+=I )
  for( yy = 1; yy < J; yy++ )
   for( xx = 0; xx < I; xx++, s++ )
    ex[s] += cex[s] *( -hy[s] + hy[s-IJ]
                       +hz[s] - hz[s-I] );
 // ...  }
```

Wydruk 3.3: Fragment implementacji FDTD przeznaczonej dla GPU (zrównoleglenie gruboziarniste)

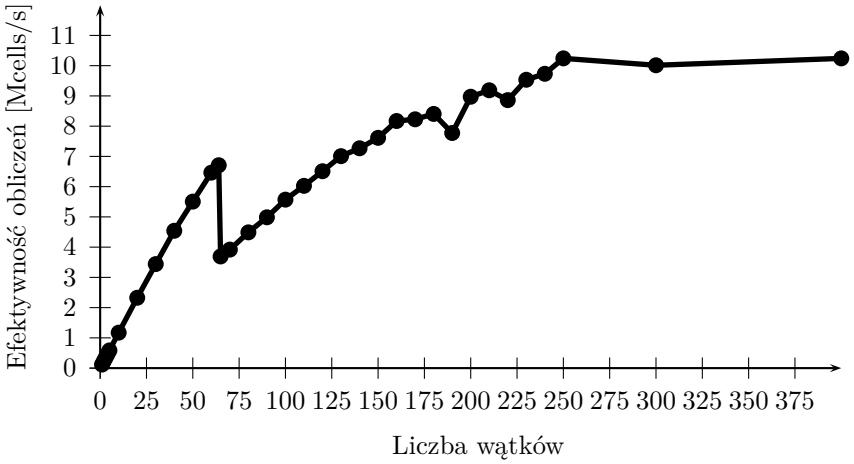
```
for( s=IJ+I, zz=1; zz<K; zz++, s+=I )
 for( yy = 1; yy < J; yy++ )
  for( xx = 0; xx < I; xx++, s++ )
   ex[s] += cex[s] *( -hy[s] + hy[s-IJ]
                      + hz[s] - hz[s-I] );
```

Wydruk 3.4: Fragment implementacji FDTD przeznaczonej dla CPU

3.5.3.2 Zastąpienie instrukcji iteracyjnych przez wątki

Cechy akceleratorów graficznych sprawiają, że bardziej efektywne oprogramowanie można opracować poprzez zastosowanie znacznej liczby wątków, które wykonują przede wszystkim operacje arytmetyczne. Ważne jest przy tym, aby jak największa liczba wątków wykonywała te same operacje dla różnych porcji danych. Z tego punktu widzenia realizacja instrukcji iteracyjnych charakteryzuje się niską efektywnością.

Skutecznym sposobem na eliminację pętli iteracyjnej (np. pętla `for` w języku C) w implementacji przeznaczonej dla GPU jest uruchomienie tylu wątków, ile wykonywanych



Rysunek 3.30: Efektywność implementacji dla GPU wzorowanej na metodzie zrównoleglenia algorytmu FDTD dla procesorów komputerowych. Pomiary wykonano na karcie Quadro FX 5600.

jest iteracji w tej pętli. Metoda ta została przez autora tej rozprawy dostosowana do realizacji implementacji algorytmu FDTD na akceleratorze graficznym, [26, 101]: z implementacji FDTD przeznaczonej dla GPU zostały wyeliminowane pętle iteracyjne, które w każdej iteracji określały indeks próbki pola, której wartość była aktualizowana. Przykład implementacji tych pętli zawiera wydruk 3.1 (pętle `for` iterujące po `xx`, `yy` i `zz`). Jedyną pętlą z algorytmu FDTD, która pozostała w zmodyfikowanej wersji implementacji FDTD jest pętla odpowiedzialna za określenie chwili czasu, dla której przeprowadzana jest kolejna aktualizacja wartości próbek pola elektromagnetycznego. Opisana metoda usprawnienia kodu przeznaczonego dla GPU prowadzi do przydzielenia do jednego wątku obliczeń związanych z aktualizacją wartości próbek pola elektrycznego lub magnetycznego z jednej komórce Yee. W rezultacie do przeprowadzenia aktualizacji wartości pola elektrycznego lub magnetycznego, dla problemu o rozmiarze $N = I \cdot J \cdot K$, wymagane jest uruchomienie N wątków. Instrukcje realizowane przez jeden wątek przy aktualizacji wartości próbek pola elektrycznego przedstawiono na wydruku 3.5. Uruchomienie N wątków, które realizują obliczenia zawarte na wydruku 3.5, przeprowadzone jest przez wywołanie `calcE<<<bl,th>>>(ex,ey,...)`, gdzie:
`bl` - informacja na temat rozmiaru siatki bloków wątków,
`th` - informacje na temat rozmiaru bloku wątków.

Identyfikacja numeru wątku przeprowadzana jest na podstawie znajomości numeru bloku (zmienna `blockIdx.x`) oraz numeru wątku (zmienna `threadIdx.x`). Na tej podstawie określany jest numer komórki Yee (zmienna `fid` na wydruku 3.5), której wartości próbek będą aktualizowane przez dany wątek.

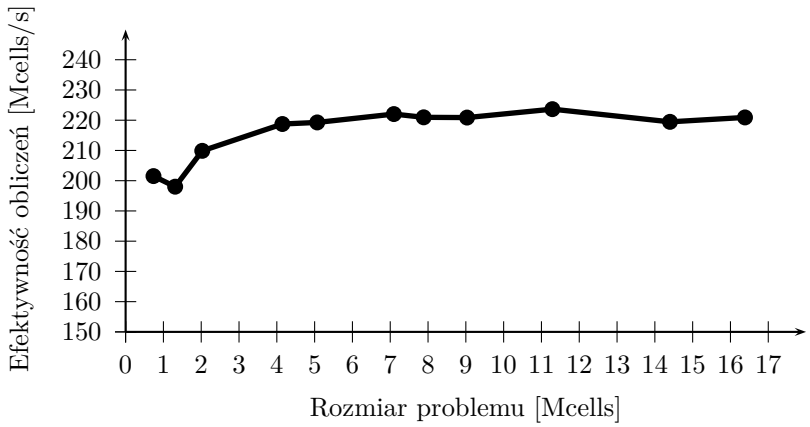
Efektywność kodu widocznego na wydruku 3.5 zaprezentowano na rys. 3.31. Wykazana efektywność jest znacznie większa niż efektywność implementacji z zastosowaniem dekompozycji dziedziny obliczeniowej. Główną przyczyną takiej sytuacji jest lepszy podział obliczeń na wątki w odniesieniu do implementacji masywnego zrównoleglenia.

Jednak przedstawiona implementacja może zostać poddana dalszej optymalizacji. Zasadniczym elementem, którego zmiany są wskazane z punktu widzenia większej efektywności, jest ograniczenie transferu danych pomiędzy rdzeniami akceleratora graficznego

a pamięcią globalną.

```
__global__ void calcE( float * ex,
                       float * ey, ... ){
 const int fid = threadIdx.x
               + blockIdx.x * NUM_TH;
               // NUM_TH - rozmiar bloku wątków
 // ...
 ex[ fid ] += cex[ fid ] * (
           - hy[ fid ] + hy[ fid - IJ] +
             hz[ fid ] - hz[ fid - I ] );
 ey[ fid ] += cey[ fid ] * (
             hx[ fid ] - hx[ fid - IJ] -
             hz[ fid ] + hz[ fid - 1 ] );
 ez[ fid ] += cez[ fid ] * (
           - hx[ fid ] + hx[ fid - I ] +
             hy[ fid ] - hy[ fid - 1 ] );
 // ...  }
```

Wydruk 3.5: Fragment kodu przeznaczonego dla GPU, który aktualizuje wartości próbek pola elektrycznego



Rysunek 3.31: Efektywność implementacji algorytmu FDTD w wersji podstawowej masowego zrównoleglenia. Pomiary wykonano na karcie Quadro FX 5600.

W ogólności do aktualizacji N wartości próbek pola elektrycznego wymagane są obliczenia z udziałem N próbek pola magnetycznego oraz N próbek parametrów materiałowych. Jednak do wyznaczenia wartości jednej próbki pola trzeba dostarczyć wartość pięciu zmiennych, np. do aktualizacji wartości próbki pola `ex[fid]` każdy wątek wymaga znajomości `cex[fid]`, `hy[fid]`, `hy[fid-IJ]`, `hz[fid]`, `hz[fid-I]`. W implementacji z wydruku 3.5 wymagane dane są przekazywane z pamięci globalnej do rejestrów multiprocesorów. Zatem całkowity transfer danych w tej implementacji wynosi $5N$. Wartość ta stanowi zasadniczy czynnik, który wpływa na obniżenie maksymalnej efektywności obliczeń.

Zmniejszenie poziomu transferu danych można osiągnąć poprzez zastosowanie w obliczeniach pamięci dzielonej. Ten sposób optymalizacji kodu został przedstawiony w następnym punkcie tej rozprawy.

3.5.3.3 Zastosowanie pamięci dzielonej

Rdzenie akceleratorów graficznych posiadają szybki dostęp do pamięci dzielonej umieszczonej w multiprocesorach. W porównaniu do pamięci globalnej osiągalna szybkość transferu jest ponad stukrotnie większa. Z tego powodu celem dalszej optymalizacji kodu jest przesłanie danych do pamięci dzielonej i wykonanie na nich jak największej liczby operacji. Wówczas przy wielu operacjach arytmetycznych zmienne pobierane będą z pamięci dzielonej, zamiast z pamięci globalnej, co zwiększy efektywność obliczeń.

W pierwszym etapie optymalizacji kodu odpowiedni sposób umieszczenia danych w pamięci dzielonej pozwala na dwukrotne zastosowanie w obliczeniach znacznej liczby próbek pól (liczba ta jest wyznaczona poniżej).

Fragment kodu, który realizuje powyższe założenie przedstawiono na wydruku 3.6.

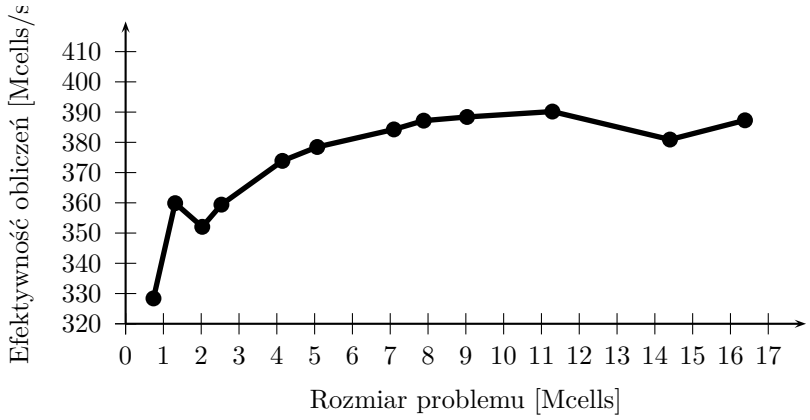
```
const int tid = threadIdx.x;
extern __shared__ float sh[];
float * hxs = (float*) & sh;
float * hys = (float*) & hxs[ NUM_TH ];
// ...
hxs[ tid ] = hx[ fid ];
a1s[ tid ] = hy[ fid - IJ ];
a2s[ tid ] = hz[ fid - I  ];
__syncthreads(); // ...
 ex[ fid ] += cexs[ tid ] * (
        - hys[ tid + 1 ] + a1s[ tid ] +
          hzs[ tid + 1 ] - a2s[ tid ] );
```

Wydruk 3.6: Fragment algorytmu zrównoleglonego masowo z podstawowym zastosowaniem pamięci dzielonej

Program ten w pierwszym etapie przesyła wszystkie niezbędne dane do obliczeń wykonywanych przez dany blok wątków z pamięci globalnej do dzielonej. Wówczas, w obliczeniach pochodnej w kierunku x, możliwe jest skorzystanie z jednej tablicy, np. `hxs`. Obliczenie pochodnej ma wówczas postać `hxs[tid] - hxs[tid+1]`. W rezultacie większość danych potrzebna do wyznaczenia pochodnej w kierunku x jest przesyłana z pamięci globalnej do dzielonej tylko jednokrotnie. Jedyne dane, które są przesyłane dwukrotnie odpowiadają za wyznaczenie pochodnej w kierunku x dla ostatniego elementu bloku wątków. Dodatkowo, ponieważ wyznaczenie pochodnej w kierunku x dotyczy tylko składowych E_y, E_z, H_y, H_z, to transfer danych zostanie zmniejszony o $2/3N - blcks$ próbek pól, przy aktualizacji wartości próbek pola elektrycznego lub magnetycznego ($blcks$ oznacza tu liczbę bloków wątków uruchomionych w trakcie aktualizacji wartości próbek pola elektrycznego/magnetycznego w trakcie jednej iteracji). Zatem całkowity transfer danych tak zoptymalizowanego kodu wynosi $4N + 1/3N + blcks$.

Efektywność zoptymalizowanego kodu przedstawiono na rys. 3.32. W porównaniu do poprzedniej wersji kodu efektywność została zwiększona o 74%, do wartości 390 Mcells/s dla problemu o rozmiarze 11,3 Mcells.

Możliwe jest dalsze zwiększenie efektywności poprzez zwiększenie liczby operacji arytmetycznych na zmiennych przesłanych do pamięci dzielonej, co zostało opisane w następnym punkcie tej rozprawy.



Rysunek 3.32: Efektywność algorytmu FDTD zrównoleglonego masowo dla GPU. Wersja z podstawowym zastosowaniem pamięci dzielonej. Pomiary wykonano na karcie Quadro FX 5600.

3.5.3.4 Grupowanie danych z jednego przekroju

W poprzednim etapie optymalizacji kodu wykazano, że przesłanie danych do pamięci dzielonej umożliwia zmniejszenie transferu danych przy wyznaczeniu pochodnej w kierunku x. Dalsze zmniejszenie transferu danych wykonano poprzez dwukrotne zastosowanie w obliczeniach jednej próbki pola przy wyznaczeniu pochodnej w kierunku y. Wyznaczenie pochodnej w kierunku y wymaga obecności w pamięci dzielonej w jednej tablicy próbek odpowiadającym zarówno e(i) jak i e(i-J). Realizację tak określonego celu autor tej rozprawy osiągnął poprzez podział wektorów **e** oraz **h** na segmenty, które zawierają wartości próbek pól należących do fragmentu dziedziny obliczeniowej stanowiącego prostokątny obszar z przekroju tej dziedziny. W takiej sytuacji każdy blok wątków pobiera wartości próbek z jednego prostokątnego obszaru dziedziny obliczeniowej i umieszcza je w pamięci dzielonej z przeznaczeniem do zastosowania w obliczeniach.

Wielkość obszaru prostokątnego zdeterminowana jest przez dostępne rozmiary bloku wątków: 64, 96, 128, 196, 256, 384. Wybór jednej z powyższych wartości został przeprowadzony na podstawie analizy efektywność obliczeń z zastosowaniem pamięci dzielonej. Efektywność ta jest tym większa, im mniej jest konfliktów w dostępie do pamięci dzielonej, [68]. Konflikty te zaś powiązane są z nieodpowiednim uporządkowaniem odwołań wątków do pamięci. Wysoką efektywność obliczeń z zastosowaniem pamięci dzielonej można osiągnąć poprzez takie uporządkowanie odwołań do pamięci dzielonej, które gwarantuje, że każdy wątek odwołuje się do różnego bloku tej pamięci (w akceleratorach graficznych z architekturą w wersji 1.x pamięć dzielona składa się z szesnastu bloków pamięci, przy czym adres zmiennej modulo 16 decyduje o przydziale zmiennej do poszczególnego bloku pamięci, w akceleratorach z architekturą w wersji 2.x pamięć dzielona składa się z 32 bloków pamięci, [69]). Największą efektywność obliczeń zmierzono, gdy blok 256 wątków aktualizował wartości próbek pól poddziedziny obliczeniowej o rozmiarze $16 \times 16 \times 1$ oczek siatki Yee.

Z wyżej wskazanych powodów optymalizacja implementacji algorytmu FDTD przeznaczonej dla GPU przedstawiona w tym punkcie sprowadza się do zgrupowania próbek pól z obszarów 16×16 oczek siatki Yee, które należą do jednej płaszczyzny $z = const$. Fragment implementacji, która realizuje powyższe założenia, został przedstawiony na wy-

druku 3.7.

```
const int tiy = threadIdx.y;
const int tid = threadIdx.x + (tiy<<4);
const int yid = tid + tiy + 1;
const int fid = (blockIdx.x<<8) + tid;
float* sx0 = sh;
float* sy0 = (float*) &sx0[ NUM_TH ];
float* shx = (float*) &sy0[ NUM_TH ];
float* shy = (float*) &shx[ NUM_TH+16 ];
// ...
shy[ yid ] = hy[ fid ];
if(calcEx) ex[ fid ] += cex[fid ] * (
     - shy[ yid ]    + sy0[ tid ]
     + shz[ yid+TI ] - shz[ yid-1 ] );
//...
```

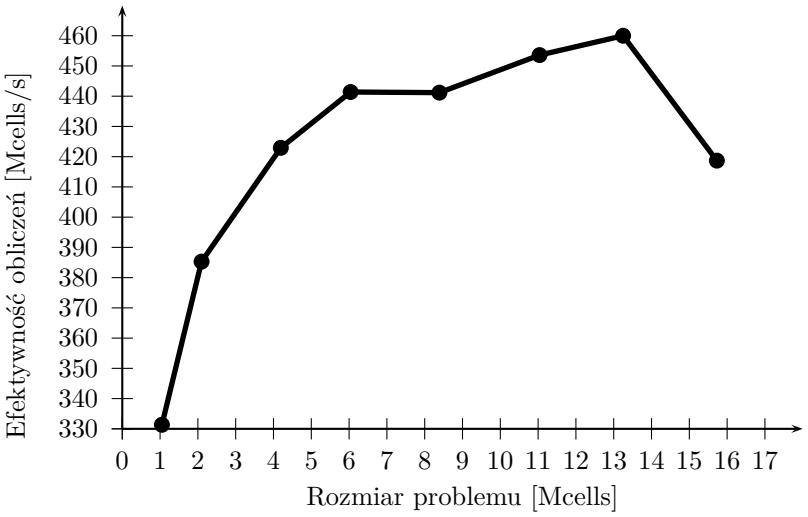
Wydruk 3.7: Fragment algorytmu zrównoleglonego masowo z optymalizacją powierzchniową dostępu do pamięci dzielonej

W opracowanej implementacji wątek identyfikowany jest przez dwie współrzędne $[tix, tiy]$, przy czym $tix, tiy \in \{0, 1, 2, \ldots, 15\}$. Prowadzi to do prostej metody przydzielenia jednej komórki Yee do jednego wątku. Ponadto, zastosowane rozwiązanie umożliwia efektywne rozpoznanie wątków, które wykonują obliczenia dla komórek siatki Yee, które znajdują się na brzegach obszaru prostokątnego. Własność ta pozawala na zastosowanie efektywnej komunikacji pomiędzy obszarami prostokątnymi oraz na przeprowadzenie obliczeń związanych z komórkami brzegowymi obszarów prostokątnych z takim samym kosztem numerycznym jak dla pozostałych komórek siatki Yee.

Nieco inaczej przedstawia się identyfikacja na poziomie bloków wątków. Najbardziej efektywnym rozwiązaniem byłoby zastosowanie numeracji obszaru prostokątnego w siatce Yee przez trzy współrzędne: $[bx, by, bz]$, gdzie $[bx, by]$ określa obszar prostokątny dla płaszczyzny $k = bz$ dyskretnej dziedziny obliczeniowej. Niestety w akceleratorach w wersji architektury 1.x możliwa jest identyfikacja bloku wątków jedynie w przestrzeni dwuwymiarowej: $[bx, by]$. Transformacja z numeracji dwuwymiarowej do trójwymiarowej jest możliwa, jednak jest kosztowna numerycznie: każdy wątek musiałby m.in. wykonywać operacje modulo i dodatkowe operacje, które stanowiłyby znaczny koszt numeryczny biorąc pod uwagę znikomą liczbę działań przeprowadzoną przez jeden wątek. Z tego powodu identyfikacja danych z sąsiedniego obszaru prostokątnego została przeprowadzona z zastosowaniem wektora indeksów, który zawiera informację o tym, w którym miejscu spermutowanych wektorów **e** oraz **h** znajdują się dane odpowiadające wybranemu obszaru prostokątnemu. Z przeprowadzonych testów wynika, że takie rozwiązanie, pomimo że wymaga dodatkowego transferu danych z pamięci globalnej, jest rozwiązaniem efektywniejszym w porównaniu do metody identyfikacji obszaru prostokątnego przy pomocy wyznaczenia trójwymiarowego położenia obszaru prostokątnego w przestrzeni dyskretnej.

Opracowane rozwiązanie zmniejsza wymagany transfer danych, co prowadzi do wzrostu efektywności obliczeń. Wartość osiągniętej maksymalnej efektywności obliczeń wynosi 459 Mcells/s. Stanowi to wzrost wydajności obliczeń o 18% w porównaniu do drugiej wersji implementacji oraz 105% do wstępnej wersji implementacji algorytmu FDTD. Koszt

wykazanego wzrostu efektywności stanowi ograniczenie rozmiaru przestrzeni obliczeniowej do rozmiarów równych $(FI \cdot 16) \times (FJ \cdot 16) \times K$ oczek siatki Yee oraz wzrost nakładów numerycznych realizowanych przez jeden wątek.



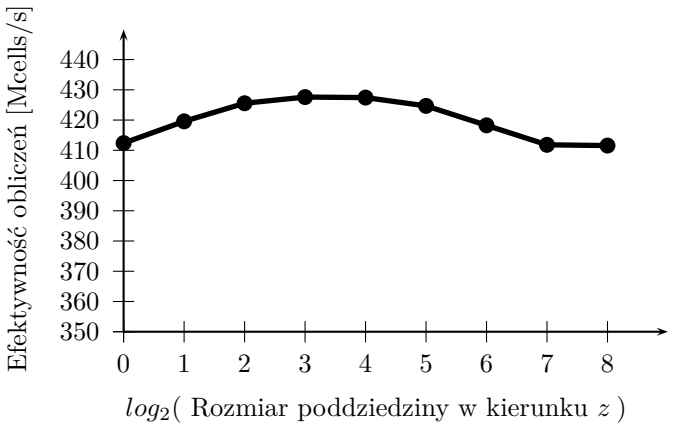
Rysunek 3.33: Efektywność algorytmu FDTD zrównoleglonego masowo dla GPU. Wersja z optymalizacją powierzchniową dostępu do pamięci dzielonej. Pomiary wykonano na karcie Quadro FX 5600.

3.5.3.5 Grupowanie danych w prostopadłościany

W kolejnym etapie optymalizacji kodu ograniczono nadmiarowy transfer danych, który występuje w algorytmie opisanym w poprzednim punkcie przy wyznaczeniu pochodnej w kierunku z. Ograniczenie to zrealizowano poprzez przydzielenie do jednego bloku wątków kliku obszarów prostokątnych, które należą do jednego prostopadłościanu przestrzeni obliczeniowej. Wówczas wartości próbek pól są aktualizowane w kolejnych iteracjach, przy czym w jednej iteracji wyznaczane są wartości pól z jednego przekroju dziedziny obliczeniowej (16×16 komórek Yee). Kod, który realizuje opisane działania otrzymano poprzez wprowadzenie jednej pętli do kodu z wydruku 3.7. Przy tak zdefiniowanej metodzie aktualizacji wartości próbek pól jedynie elementy wektorów $\mathbf{e}$ i $\mathbf{h}$, które należą do w/w boków prostopadłościanów przestrzeni obliczeniowej są przesyłane dwukrotnie.

Przykładowo, przy obliczeniach wartości próbek składowych E_x oraz E_y wymagana jest znajomość składowej H_x oraz H_y zarówno z tej samej płaszczyzny siatki Yee (są one umieszczone w tablicy `shx` oraz `shy`, wydruk 3.7) jak i z płaszczyzny sąsiedniej (tablice `sx0`, `sy0`). Zatem przy wyznaczeniu wartości próbek składowych E_x oraz E_y z sąsiedniego obszaru prostokątnego wartość połowy wymaganych próbek składowych H_x oraz H_y można otrzymać poprzez transfer w obrębie pamięci dzielonej, np. `shx[fid] = sx0[tid]`. Prowadzi to do zmniejszenia transferu danych z pamięci globalnej. Jednak zmniejszenie transferu danych nie prowadzi do wzrostu efektywności obliczeń w porównaniu do poprzedniej wersji programu, co można zaobserwować na wykresie 3.34.

Zdaniem autora, jest to spowodowane przez znaczny wzrost operacji numerycznych przeprowadzonych przez jeden wątek. Wynika z tego konieczności osiągnięcia kompromisu pomiędzy złożonością obliczeń wykonywanych przez jeden wątek oraz wymagany



Rysunek 3.34: Efektywność implementacji algorytmu FDTD przeznaczonej dla GPU w zależności od rozmiaru podprzestrzeni przydzielonej do jednego bloku wątków w kierunku z. Wersja algorytmu z optymalizacją objętościową dostępu do pamięci dzielonej. Rozmiar problemu ustalono na $256 \times 256 \times 256$ komórek Yee. Pomiary wykonano na karcie Quadro FX 5600.

przez daną implementację poziom transferu z pamięci globalnej. Z doświadczeń autora wynika, że wskazanie najkorzystniejszego rozwiązania jest możliwe tylko poprzez weryfikację praktyczną różnych wersji implementacji.

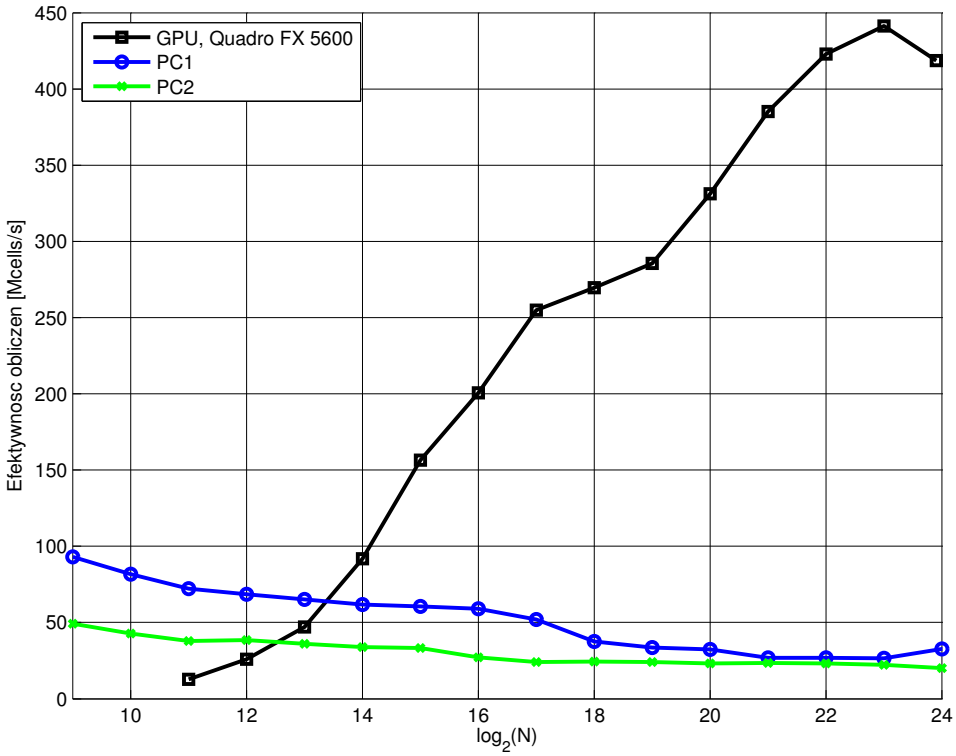
3.5.3.6 Porównanie opracowanych rozwiązań

Tablica 3.17: Porównanie efektywności obliczeń implementacji algorytmu FDTD przeznaczonej dla GPU oraz dla pojedynczego CPU. Pomiary wykonano na karcie Quadro FX 5600 oraz procesorze komputerowym z PC1.

Sposób zrównoleglenia	Mcells/s	Względne przyspieszenie obliczeń
obliczenia na CPU, rys. 3.9	30	1
zrównoleglenie gruboziarniste (300 wątków)	10	0,33
podstawowe zrównoleglenie drobnoziarniste	224	7,47
zrównoleglenie drobnoziarniste z prostym zastosowaniem pamięci dzielonej	390	13,0
zrównoleglenie drobnoziarniste z grupowaniem danych z jednego przekroju	440	14,7
zrównoleglenie drobnoziarniste z grupowaniem danych z prostopadłościanów	427	14,2

Porównanie efektywności obliczeń opracowanych rozwiązań przedstawia tabela 3.17. Osiągnięte wartości zostały porównane z efektywnością implementacji algorytmu FDTD przeznaczonej dla CPU i uruchomionej na PC1. Najlepsze z opracowanych rozwiązań pozwala na osiągnięcie wzrostu efektywności obliczeń na poziomie ponad 14-krotnie większym dla problemów o znacznych rozmiarach w porównaniu do obliczeń realizowanych na CPU. Porównanie efektywności obliczeń implementacji algorytmu w postaci jawnej

dla różnych rozmiarów problemów przedstawiono na rys. 3.35. Można na nim zauważyć bardzo korzystny trend: efektywność obliczeń silnie rośnie wraz ze wzrostem rozmiaru problemu. Wynika stąd, iż architektura akceleratorów umożliwia efektywne zrównoleglenie algorytmu FDTD, tak iż wzrost liczby uruchomionych wątków prowadzi do wzrostu wydajności obliczeniowej.



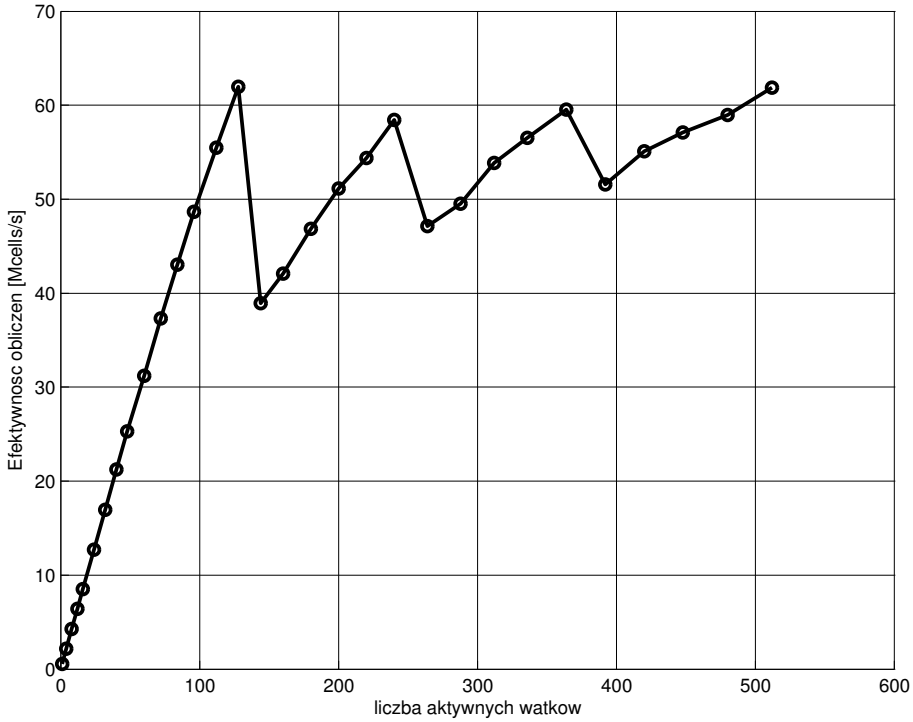
Rysunek 3.35: Porównanie efektywności obliczeń jednowątkowej implementacji algorytmu FDTD przeznaczonej dla CPU z efektywnością otrzymaną dla algorytmu FDTD zrównoleglonego masowo dla GPU w wersji z optymalizacją powierzchniową dostępu do pamięci dzielonej.

3.5.3.7 Obliczenia przeprowadzone na akceleratorze z architekturą Fermi

Wszystkie dotychczas przedstawione testy z użyciem kart graficznych przeprowadzono na karcie Quadro FX 5600, której akcelerator graficzny posiada architekturę w wersji 1.0. W dalszej części rozprawy pomiary czasu przeprowadzenia symulacji zostały przeprowadzone z użyciem karty GeForce GTX 580 firmy Nvidia, która zawiera akcelerator graficzny w architekturze 2.0. Wprowadzenie na rynek konsumencki akceleratorów graficznych w architekturze 2.x, nazywanej potocznie architekturą Fermi, przyniosło kolejny wzrost szybkości działania dostępnych kart graficznych. Silnie do tego przyczyniło się wprowadzenie do tych akceleratorów pamięci podręcznej L1 i L2. Pamięć L1 stanowi usprawnienie działania pamięci dzielonej - oba rodzaje pamięci stanowią część wspólnego obszaru pamięci, tak iż pamięć L1 dla wybranego kernela może zostać zwiększona kosztem pamięci dzielonej i odwrotnie. Zasadnicza różnica pomiędzy tymi rodzajami pamięci sprowadza się do sposobu zarządzania danymi, które w nich są umieszczane.

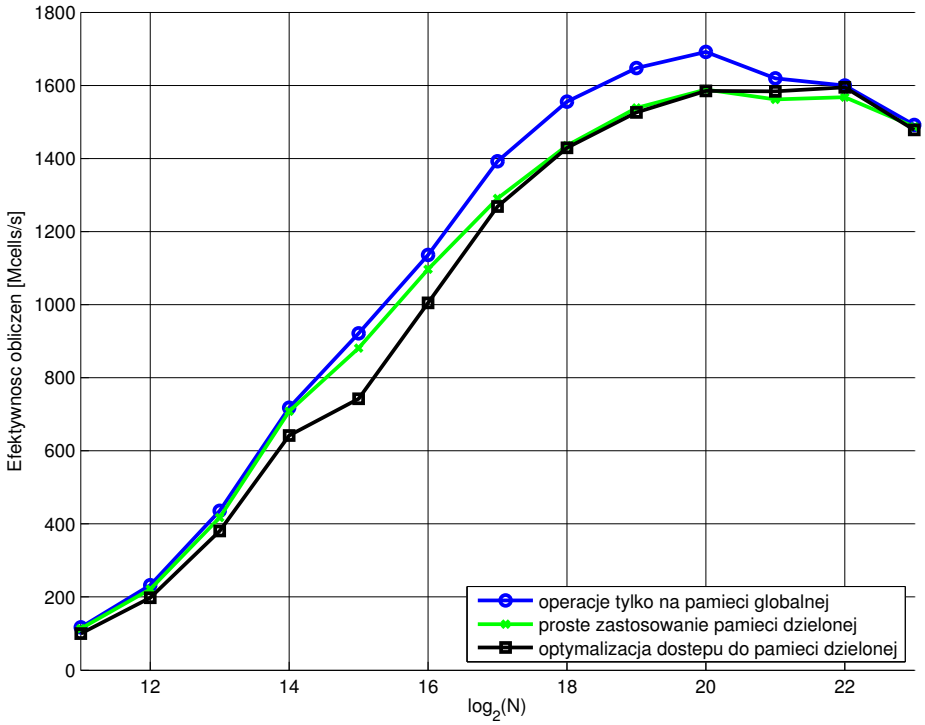
Nad zawartością L1 pełną kontrolę przejmuje akcelerator, podczas gdy pamięć dzielona w pełni kontrolowana jest przez programistę.

Wpływ wprowadzenia pamięci podręcznej do akceleratorów na efektywność obliczeń algorytmu FDTD można ocenić na podstawie rys. 3.36 oraz rys. 3.37. W pierwszym etapie zmierzono szybkość działania programu przy zastosowaniu tradycyjnej metody zrównoleglenia algorytmu opisanej w p. 3.5.3.1. Rezultaty pomiarów przedstawia rys. 3.36.



Rysunek 3.36: Efektywność implementacji algorytmu FDTD dla GPU wzorowanej na metodzie zrównoleglenia tego algorytmu dla procesorów komputerowych. Pomiary wykonano na karcie GTX 580.

Znacznie większą efektywność obliczeń zmierzono przy pomiarze szybkości działania implementacji, w której zastosowano zrównoleglenie drobnoziarniste, rys. 3.37. Z porównania rys. 3.36 oraz rys. 3.37 wynika, że optymalizacja kodu przeznaczonego dla GPU prowadzi do ponad dwudziestokrotnego wzrostu efektywności obliczeń, więc wprowadzenie pamięci podręcznej nie niweluje konieczności dostosowania kodu programu do architektury GPU. Charakterystyczne dla nowej architektury GPU jest przy tym, że wprowadzenie pamięci podręcznej umożliwia przeprowadzenie optymalizacji dostępu do pamięci dzielonej i globalnej w czasie rzeczywistym, co wynika z pomiarów przedstawionych na rys. 3.37. Jednak za podstawową implementację algorytmu FDTD uznano wersję, w której zastosowano grupowanie danych z jednego przekroju (opis w p. 3.5.3.4, optymalizacja dostępu do pamięci dzielonej z rys. 3.37). Za takim wyborem przemawia odpowiednia organizacja danych, która umożliwia zdefiniowanie efektywnej implementacji algorytmu mieszanego FDTD (p. 4.5) oraz algorytmu zawierającego warstwę absorpcyjną, CPML, opisaną w p. 7.1.



Rysunek 3.37: Efektywność implementacji algorytmu FDTD dla GPU dla różnych wersji optymalizacji kodu. Pomiary wykonano na karcie GTX 580.

3.5.3.8 Podsumowanie opisu implementacji algorytmu FDTD dla GPU

W p. 3.5.3 wykazano, że efektywność obliczeniowa symulacji FDTD przeprowadzonej dla dużych rozmiarów problemu na akceleratorze graficznym przy użyciu zmiennych o pojedynczej precyzji jest **50-krotnie** większa w porównaniu do efektywności zmierzonej dla pojedynczego procesora komputerowego (rys. 3.37). Jednak opracowanie wydajnej implementacji algorytmu FDTD przeznaczonej dla GPU wymaga odmiennej koncepcji budowy tej implementacji w porównaniu do kodów pisanych dla CPU. Wykazany wysoki poziom efektywności jest możliwy do osiągnięcia poprzez zastosowanie w obliczeniach znacznej liczby wątków. Model takiego programowania nazywany jest drobnoziarnistym równoległym przetwarzaniem danych (ang. *fine-grained parallelism*), [36].

3.6 Podsumowanie

Efektywność obliczeniowa implementacji algorytmu FDTD w postaci jawnej przeznaczona dla CPU silnie zależy od dostępnego maksymalnego poziomu transferu danych pomiędzy procesorem komputerowym a pamięcią RAM. Z tego względu jest ona znacznie większa dla problemów o mniejszych rozmiarach, gdzie w większym stopniu występuje szybki transfer danych do pamięci podręcznej procesora. Silne ograniczenie poziomu transferu danych w procesorach wielordzeniowych stanowi zasadniczą przeszkodę w osiągnięciu skalowalności algorytmu FDTD w tym środowisku. Skalowalność tego algorytmu w szerokim zakresie rozmiarów analizowanych problemów jest możliwa do osiągnięcia dla komputerów wieloprocesorowych jeśli dla każdego procesu przydzielony zostanie osobny kontroler pamięci.

Udowodniono, że akceleratory graficzne umożliwiają przeprowadzenie symulacji fali elektromagnetycznej za pomocą algorytmu FDTD z dużo większą efektywnością niż ma to miejsce dla procesorów komputerowych z pojedynczego zestawu komputerowego. Architektura GPU pozwala osiągnąć wysoki poziom efektywności obliczeń dla problemów o znacznych rozmiarach, co pozytywnie odróżnia ją od architektury CPU.

Największy poziom efektywności obliczeń otrzymano przy uruchomieniu obliczeń na klastrze Galera. Przy zachowaniu reguły przydzielenia jednego kontrolera pamięci dla jednej poddziedziny obliczeniowej osiągnięto skalowalność zrównoleglonej implementacji algorytmu FDTD, co pozwala na przeprowadzenie symulacji problemów elektromagnetycznych opisanych przez dziedzinę obliczeniową o największym rozmiarze (z uwagi na dostępną pamięć RAM o największym rozmiarze spośród badanych środowisk sprzętowych) oraz z największą efektywnością obliczeń, przy założeniu, że obliczenia zostaną przeprowadzone na znacznej liczbie węzłów. Należy przy tym podkreślić, że koszt takiego rozwiązania jest również największy. Zmierzona dla karty graficznej efektywność obliczeniowa dla problemów o dużych rozmiarach jest 50-krotnie większa w porównania do jednowątkowych obliczeń wykonanych na pojedynczym procesorze komputerowym.

Przy założeniu, że pojedynczy węzeł obliczeniowy z klastra zawiera jeden procesor wielordzeniowy, który umożliwia dwukrotne przyspieszenie obliczeń przy zastosowaniu zrównoleglonej implementacji algorytmu FDTD, klaster oferujący zbliżoną efektywność obliczeń powinien zawierać co najmniej 25 węzłów. Warto przy tym zauważyć, że przewagą klastra w porównaniu do karty graficznej jest znacznie większy zasób pamięci RAM, co ma istotne znaczenie przy przeprowadzaniu symulacji FDTD dla problemów o bardzo dużych rozmiarach. Jednak koszt zakupu takiego rozwiązania byłby wielokrotnie większy w porównaniu do ceny zestawu komputerowego wyposażonego w pojedynczą kartę graficzną. Również koszt użytkowania klastra jest znacznie większy: uwzględniając jedynie pobieraną energię elektryczną, przy założeniu, że obciążony obliczeniami komputer z kartą graficzną zużywa w ciągu godziny 1000Wh energii elektrycznej, a jeden węzeł klastra zużywa jej 750Wh, to koszt użytkowania klastra jest ponad osiemnastokrotnie większy w porównaniu z kosztem użytkowania komputera z kartą graficzną.

Biorąc pod uwagę powyższe wnioski, można stwierdzić, że symulacje elektromagnetyczne można przeprowadzić najefektywniej i najekonomiczniej na kartach graficznych.

ROZDZIAŁ 4

Algorytm FDTD w postaci macierzowej

Algorytm FDTD oparty jest na dyskretyzacji w równaniach Maxwella dziedziny czasu oraz przestrzeni. Jak wykazano w rozdziale 2 wartość pochodnej w przestrzeni w postaci dyskretnej może zostać wyznaczona w algorytmie w dwóch wariantach:

1. implementacja macierzowa - pochodna pola w przestrzeni jest wyznaczona przez mnożenie wektora próbek pola elektromagnetycznego przez macierz rzadką, zgodnie z r. (2.23).
2. implementacja jawna - pochodna pola w przestrzeni jest wyznaczona za pomocą algorytmu, który odwołuje się bezpośrednio do poszczególnych elementów wektorów **e** oraz **h** za pomocą indeksów, zgodnie z r. (2.24)

Przewagą implementacji jawnej nad macierzową jest większa efektywność obliczeń dla niezmodyfikowanej wersji algorytmu, natomiast macierzowej nad jawną duża elastyczność przy modyfikacji algorytmu FDTD. Modyfikacja algorytmu FDTD w postaci macierzowej może prowadzić do wzrostu szybkości obliczeń, co zostanie opisane w rozdziałach 5 i 6. Jednak przejście na zapis macierzowy algorytmu FDTD, przy zachowaniu wszystkich zależności pomiędzy próbkami pól w niezmienionej formie, prowadzi do spadku efektywności obliczeniowej.

W tym rozdziale zostaną przeanalizowane parametry, które wpływają na efektywność obliczeń implementacji algorytmu FDTD w formie macierzowej wraz z porównaniem otrzymanych rezultatów z wynikami otrzymanymi przy analizie algorytmu w formie jawnej. Wspomniana analiza obejmuje również ocenę efektywności zrównoleglenia algorytmu w formie macierzowej.

Do przeprowadzenia symulacji opisanych w tym rozdziale zastosowano kartę graficzną GeForce GTX 580.

4.1 Jednowątkowa implementacja przeznaczona dla CPU

Zasadniczą część macierzowej postaci algorytmu FDTD stanowi algorytm mnożenia macierzy rzadkiej przez wektor, zgodnie z r. (2.23). Dlatego sposób reprezentacji macierzy rzadkiej w programie ma znaczący wpływ na efektywność obliczeń. Informacje o podstawowych formatach reprezentacji macierzy rzadkiej w programach komputerowych zawiera p. C.3. W programach przedstawionych w tej rozprawie, które są przeznaczonych dla procesorów komputerowych i stosują w obliczeniach operacje na macierzach rzadkich, do reprezentacji macierzy rzadkiej wybrano format CRS (ang. *Compressed Row Storage*).

```
1  for( it = 0; it < iterNo; it++ ){
2   // pobudzenie miekkie
3   for( xx=0; xx<psrcLen; xx++ )
4    ex[ psrc[ xx ] ] += src[ it ];
5
6   // wyznaczenie wartosci skladowych pola e
7   matvecP( &roth, h, e );
8   // wyznaczenie wartosci skladowych pola h
9   matvecN( &rote, e, h );
10
11  // pobranie wartosci pola elektrycznego
12  for( xx=0; xx<psepLen; xx++ )
13   sep[ it ] += ex[ psep[ xx ] ];
14 }
```

Wydruk 4.1: Jednowątkowa implementacja macierzowa algorytmu FDTD przeznaczona dla CPU [1]

```
1  inline void matvecP( int* rj, int* ri, double* rvr, int m1, int n1, int nzmax,
2                       double* xx, double* yy ){
3   int j  = 0;
4   int p  = 0;
5   register double  tt = 0;
6   for( ; j<n1; j++ ){
7    tt = 0.0;
8    for( ; p< ri[j+1]; p++ )
9     tt += rvr[p] * xx[ rj[p] ];
10   yy[j] += tt;
11  }
12 }
```

Wydruk 4.2: Implementacja mnożenia macierzy rzadkiej w formacie CRS przez wektor przeznaczona dla CPU

[1] Funkcja `matvecN` różni się od funkcji `matvecP` tylko w 10 linii, gdzie zamiast `yy[j] += tt` występuje odejmowanie `yy[j] -= tt`.

Wydruk 4.2 przedstawia implementację algorytmu mnożenia macierzy rzadkiej w formacie CRS przez wektor. Na podstawie wydruku 4.1 można stwierdzić, że implementacja algorytmu FDTD w formie macierzowej sprowadza się w głównej mierze do dwukrotnego wywołania algorytmu mnożenie macierzy rzadkiej przez wektor.

Macierze rzadkie, które można zastosować we wskazanej implementacji, mogą mieć postać zdefiniowaną przez zależności (2.16) oraz (2.19). Należy jednak podkreślić, że implementacja z wydruku 4.1 pozwala również na przeprowadzenie obliczeń z algorytmem FDTD poddanym modyfikacji, np. poprzez włączenie obszaru z lokalnie zmniejszonym krokiem dyskretyzacji lub poprzez włączenie makromodeli.

4.2 Koszty numeryczne implementacji algorytmu FDTD w postaci macierzowej

Koszty numeryczne dla algorytmu w postaci jawnej zostały wykazane w p. 3.2, jednak z uwagi na zastosowanie macierzy do reprezentacji zależności pomiędzy wektorami $\mathbf{e}$ i $\mathbf{h}$, koszty implementacji algorytmu w postaci macierzowej będą większe. W poniższych zależnościach założono, że macierz rzadka reprezentowana jest w formacie CRS.

Z r. (2.14), (2.16) i (2.19) wynika, że macierze $\mathbf{R}_E$ oraz $\mathbf{R}_H$ posiadają w zdecydowanej większości wierszy cztery elementy niezerowe (warunki brzegowe zawarte w tych macierzach wymuszają istnienie wierszy z dwoma lub trzema elementami niezerowymi, jednak liczba takich wierszy jest na poziomie $2\div5\%$ w zależności od wielkości problemu i w celu zwiększenia przejrzystości dalszych wyprowadzeń nie została ona w nich uwzględniona). Zgodnie z r.(2.12) i (2.13) rozmiar wektorów $\mathbf{e}$ i $\mathbf{h}$, wynosi $3N$. Zatem liczba elementów niezerowych macierzy $\mathbf{R}_E$ i $\mathbf{R}_H$ wynosi $nnz = 12N$. Biorąc pod uwagę r. (C.1), można oszacować liczbę bajtów wymaganych przez algorytm FDTD w postaci macierzowej L_{BM}, wedle r. (4.1), które uwzględnia macierz $\mathbf{R}_E$ i $\mathbf{R}_H$ oraz wektory $\mathbf{e}$ i $\mathbf{h}$.

$$L_{BM} \approx 30N\alpha_F + 30N\alpha_I \tag{4.1}$$

Porównując r. (3.2a) i (4.1) można stwierdzić, że przy obliczeniach na zmiennych o pojedynczej precyzji ($\alpha_F = \alpha_I$) zapis macierzowy wymaga w przybliżeniu pięć razy więcej pamięci w porównaniu do algorytmu w postaci jawnej.

Wykonanie jednej aktualizacji wartości pól wedle r. (2.23) wymaga wykonania dwóch operacji mnożenia przez macierz rzadką oraz czterech dodawań wektorów o rozmiarze $3N$. Uwzględniając r. (C.2), liczbę operacji zmiennoprzecinkowych wykonywanych w jednej iteracji algorytmu FDTD w postaci macierzowej L_{FT} określa r. (4.2).

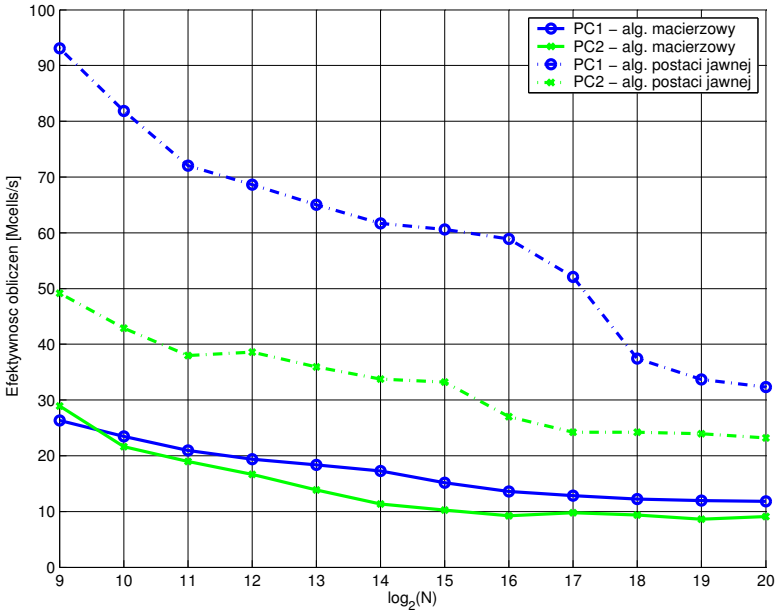
$$L_{FT} \approx 54N \tag{4.2}$$

Biorąc pod uwagę r. (3.2b) oraz r. (4.2) widoczne jest, iż algorytm w formie macierzowej wymaga przeprowadzenia około 24% większej liczby operacji zmiennoprzecinkowych.

Istotny wzrost kosztów numerycznych przy przejściu z zapisu w postaci jawnej do zapisu macierzowego prowadzi do wniosku, iż zastosowaniu zapisu macierzowego może być efektywne tylko wtedy, gdy zostanie on zmodyfikowany, np. poprzez wprowadzenie lokalnego zagęszczenia siatki (rozdział 5) lub makromodeli (rozdział 6).

4.2.1 Pomiar efektywności obliczeń

Rezultaty pomiarów efektywności obliczeń implementacji algorytmu FDTD w formie macierzowej przeznaczonej dla CPU przedstawia rys. 4.1. Porównując wykresy z rys. 4.1 oraz 3.9 można stwierdzić, iż postać macierzowa algorytmu charakteryzuje się ponad dwukrotnym spadkiem efektywności obliczeń w porównaniu do implementacji algorytmu w postaci jawnej. Zatem wprowadzenie modyfikacji do algorytmu FDTD w postaci macierzowej powinno prowadzić do co najmniej dwukrotnego przyspieszenia szybkości działania algorytmu, aby otrzymać efektywność obliczeń większą niż dla implementacji w postaci jawnej. Przykłady takich sytuacji zostały opisane w rozdziale 7.



Rysunek 4.1: Efektywność jednowątkowej implementacji algorytmu FDTD w formie macierzowej przeznaczonej dla CPU

4.3 Implementacja przeznaczona dla akceleratorów graficznych

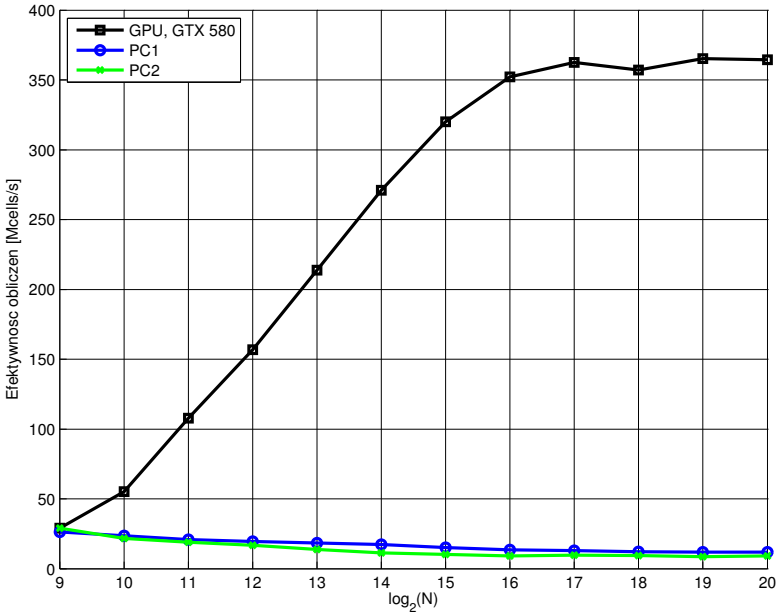
Implementacja algorytmu mnożenia macierzy rzadkiej przez wektor przeznaczona dla akceleratorów graficznych przedstawiona została na wydruku 4.3. Każdy wątek w tej implementacji odpowiada za mnożenie jednego wiersza macierzy przez wektor. Rozwiązanie takie nie jest optymalne, gdyż akcelerator graficzny przetwarza dane na zbiorach 32 wątków w czasie odpowiadającym wykonywaniu obliczeń przez najbardziej czasochłonny wątek, np. jeśli jeden wątek wykonuje sto mnożeń, a reszta wątków tylko jedno mnożenie, to czas wykonania tego zbioru wątków odpowiada czasowi wykonania 3200 mnożeń. Jednak dla macierzy rotacji w podstawowej wersji, w których około 90% wierszy stanowią wiersze o czterech elementach niezerowych, ta implementacja okazała się najbardziej efektywna spośród implementacji rozpatrywanych przez autora tej rozprawy. Możliwe do przeprowadzenia optymalizacje dla macierzy o większym zróżnicowaniu liczby elementów niezerowych w wierszu przedstawiono np. w [25].

```
1  __global__ void matvecD_T( float * x, float * y,
2                             int * ri, int * rj, float *rvr, int len ){
3      // indeks watku w siatce blokow
4      int index = threadIdx.x + blockDim.x * blockIdx.x;
5
6      if( index>=len ) // czy indeks nie wykracza poza dlugosc wektora y
7          return;
8
9      float Csub = y[index];
10
11     for( int p = rj[index]; p < rj[index + 1]; p++ )
12         Csub += rvr[p] * x[ ri[p] ];
13
14     y[ index ] = Csub;
15 }
```

Wydruk 4.3: Implementacja mnożenia macierzy rzadkiej w formacie CRS przez wektor preznaczona dla GPU

4.3.1 Pomiar efektywności obliczeń

Rezultaty pomiarów efektywności implementacji algorytmu FDTD w formie macierzowej przeznaczonej dla GPU przedstawiono na rys. 4.2.



Rysunek 4.2: Efektywność implementacji algorytmu FDTD w formie macierzowej przeznaczonej dla GPU oraz CPU. Pomiary wykonano na karcie GTX 580 oraz komputerach PC1 i PC2.

Zmierzona efektywność obliczeń dla problemów o znacznych rozmiarach ($N > 2^{16}$) jest **30-krotnie** większa w porównaniu do rezultatu otrzymanego dla procesorów komputerowych dla analogicznego algorytmu (rys. 4.1). Podobnie jak dla algorytmu w postaci

jawnej, również i dla algorytmu macierzowego można zaobserwować, iż efektywność dla znacznych rozmiarów problemu zachowuje podobny poziom, co stanowi przewagę w porównaniu do procesorów komputerowych. Jednak w porównaniu do najbardziej efektywnej implementacji algorytmu FDTD w postaci jawnej przeznaczonej dla GPU spadek efektywności implementacji macierzowej jest ponad **czterokrotny**. Tak znaczna różnica wynika przede wszystkim z mniejszej możliwości ukrywania opóźnienia związanego z transferem danych przez instrukcje realizujące działania matematyczne. W implementacji jawnej występuje prosty schemat działania algorytmu: pobranie danych, wykonanie obliczeń i zapisanie rezultatu obliczeń do pamięci. Natomiast w implementacji macierzowej występuje dwuetapowe pobranie danych z pamięci: najpierw pobierany jest indeks `ri`, a w drugiej kolejności następuje pobranie wartości z wektora `x`, co znacznie zwiększa czas wymagany do transferu danych, tym bardziej, że dane nie są pobierane sekwencyjnie z tego samego obszaru pamięci. Dodatkowo wątki wykonują różną liczbę działań, jeśli liczba elementów niezerowych w poszczególnych wierszach różni się. Na podstawie tego przykładu, można stwierdzić, iż programowanie dla akceleratorów graficznych wymaga silnego dostosowania algorytmu do ich własności, w przeciwnym razie efektywność wykorzystania zasobów sprzętowych będzie niska.

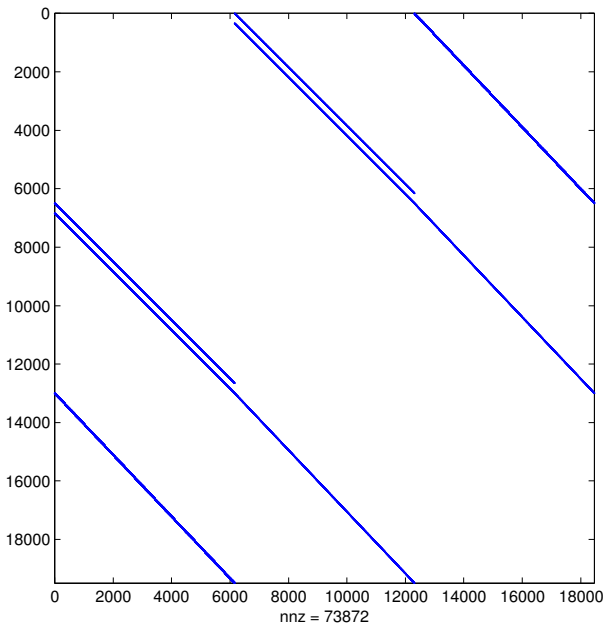
4.4 Zrównoleglenie algorytmu macierzowego na potrzeby klastra

Istotną cechą macierzy $\mathbf{R}_E$ oraz $\mathbf{R}_H$, która powinna być brana pod uwagę przy ocenie szybkości działania implementacji algorytmu mnożenia macierzy rzadkiej przez wektor, stanowi występowanie co najwyżej czterech elementów niezerowych w każdym wierszu oraz każdej kolumnie. Cecha ta wynika z postaci dyskretnej równań Maxwella (r. (2.11)), gdzie wartość jednej próbki pola wyznaczana jest w zależności od czterech innych próbek pola. Przykładowe położenie miejsc niezerowych macierzy zdefiniowanej przez r. (2.14), dla problemu o rozmiarze $20 \times 20 \times 20$, zostało przedstawione na rys. 4.3. Regularność ułożenia miejsc niezerowych jest wielką zaletą macierzy $\mathbf{R}_E$ oraz $\mathbf{R}_H$ z punktu widzenia efektywności obliczeń jednowątkowych. Jednak, co zostanie wykazane w następnym punkcie rozprawy, takie ułożenie miejsc niezerowych w macierzach nie jest optymalne z punktu widzenia efektywności obliczeń równoległych dedykowanych procesorom komputerowym.

4.4.1 Zrównoleglenie obliczeń dla macierzy rotacji o standardowej postaci

W punkcie tym zostanie wykazane, iż macierze zdefiniowane w r. (2.14) charakteryzują się słabym dostosowaniem do równoległej operacji mnożenia macierzy rzadkiej przez wektor, co wykazano również w [104]. Opis zagadnienia zostanie przeprowadzony dla środowiska obliczeniowego, które składa się z węzłów o takiej samej mocy obliczeniowej, czyli zarówno dla klastra jak i procesora wielordzeniowego. Takie założenie zwiększa przejrzystość opisu przy zachowaniu poprawności wniosków również dla środowiska o nierównym rozkładzie mocy obliczeniowej pomiędzy węzłami obliczeniowymi.

W zaimplementowanym przez autora tej rozprawy zrównoleglonym algorytmie mnożenia macierzy rzadkiej przez wektor, w sytuacji, gdy program zostaje uruchomiony w środowisku składającym się z węzłów o takiej samej mocy obliczeniowej, do każdego wątku programu zostaje przydzielony fragment wektorów $\mathbf{e}$ oraz $\mathbf{h}$ o zbliżonych rozmiarach. Z uwagi na warunki brzegowe rozmiar wektorów $\mathbf{e}$ i $\mathbf{h}$ nieznacznie się różni, jednak w celu



Rysunek 4.3: Elementy niezerowe macierzy rotacji $\mathbf{R}_E$ dla problemu o rozmiarze $20 \times 20 \times 20$

zwiększenia przejrzystości dalszego opisu przyjęto, że liczba próbek pola z wektorów **e** oraz **h** przydzielona do każdego wątku wynosi N_l. Założono przy tym, że każdy wątek zarządza obliczeniami na przydzielonym do niego rdzeniu procesora komputerowego (liczba wątków odpowiada liczbie rdzeni procesorów komputerowych). Biorąc pod uwagę r. (2.12), fragment próbek o długości N_l może obejmować:

1. próbki jednej składowej pola,
2. próbki dwóch składowych pola - fragment będzie obejmować ostatnie elementy wektora próbek jednej składowej oraz pierwsze elementy wektora próbek drugiej składowej, np. $[\,\ldots E_x(N-1)\; E_x(N)\; E_y(1)\; E_y(2)\; \ldots\,]$. Zakładając, że analizowany problem nie jest bardzo mały lub liczba węzłów nie jest bardzo duża, czyli że $N_l < N$, można stwierdzić, że próbki te określają wartość pola odległych fragmentów przestrzeni dyskretnej, a więc żadne dwa elementy z tego zakresu wektora nie będą brać udziału w wyznaczeniu wartości tej samej próbki pola.

Ponieważ podział wektorów **e** oraz **h** jest analogiczny, to każdy wątek programu będzie przechowywać wartości próbek pól zupełnie ze sobą niezwiązanych, tzn. żaden fragment wektora **e** przydzielony do danego wątku nie zostanie zastosowany przy aktualizacji wartości fragmentu wektora **h** przydzielonego do tego samego wątku i na odwrót (w równaniach Maxwella składowa E_i pola nie zależy bezpośrednio od składowej H_i i na odwrót). W rezultacie wszystkie zmienne potrzebne do wykonania aktualizacji wartości próbek pól zawartych w **e** oraz **h** wymagają transferu danych pomiędzy wątkami, co w klastrze wymaga komunikacji pomiędzy węzłami obliczeniowymi. Poziom tego transferu zostanie oszacowany na przykładzie wyznaczenia wartości próbek składowej E_x z zastosowaniem zależności wywodzącej się od r. (2.11a). Transfer danych przy wyznaczeniu wartości tej składowej może mieć różny poziom w zależności od wielkości problemu:

1. Największy transfer danych będzie miał miejsce dla problemu o bardzo dużych rozmiarach lub gdy liczba węzłów obliczeniowych jest duża, tzn. gdy $J > N_l$ oraz

$K > N_l$. Wówczas nie ma części wspólnej zbiorów próbek $H_y(i,j,k)$, $H_y(i,j,k-1)$, $H_z(i,j,k)$ oraz $H_z(i,j-1,k)$. W związku z tym, wymagany transfer danych jest czterokrotnie większy od liczby próbek fragmentu wektora $\mathbf{e}$ przydzielonego do danego wątku programu, czyli wynosi $4N_l$.

2. W drugiej skrajnej sytuacji, gdy część wspólna zbiorów próbek $H_y(i,j,k)$ i $H_y(i,j,k-1)$ wynosi $N_l - K$ próbek oraz część wspólna zbiorów próbek $H_z(i,j,k)$ i $H_z(i,j-1,k)$ wynosi $N_l - J$ próbek, transfer jest najmniejszy i wynosi $2N_l + K + J$ próbek pola.

Zatem przy równoległym mnożeniu macierzy rzadkiej przez wektor, dla macierzy zdefiniowanej przez r. (2.14), transfer danych będzie większy od $2N_l$. Z tego powodu sumaryczny transfer danych przy pojedynczym mnożeniu macierzy rotacji $\mathbf{R}_E$ przez wektor $\mathbf{e}$ ($\mathbf{R}_H$ przez $\mathbf{h}$) jest większy od dwukrotności rozmiaru wektora $\mathbf{e}$ ($\mathbf{h}$). Prowadzi to do znacznego spadku efektywności w porównaniu do sytuacji, w której transfer danych jest zoptymalizowany.

4.4.2 Dowolna permutacja elementów

Istnieją dwa kierunki optymalizacji algorytmu FDTD w postaci macierzowej:

1. można zmienić zbiór elementów wektorów $\mathbf{e}$ oraz $\mathbf{h}$ przydzielanych do węzłów obliczeniowych,
2. można zmienić położenie elementów niezerowych w macierzy poprzez zastosowanie permutacji macierzy.

Można udowodnić, że każda optymalizacja transferu danych wykonana przy pomocy pierwszej metody ma swój odpowiednik w postaci optymalizacji wykonanej przy pomocy drugiej metody i na odwrót. W związku z tym rozwiązania te są równoważne pod względem poziomu transferu danych. Dowód tego stwierdzenia można przeprowadzić stosując opis matematyczny.

Przydział elementów wektorów do węzłów obliczeniowych można opisać jako odwzorowanie od numerów elementów w wektorze globalnym do elementów wektora w wektorze lokalnym (przydzielonym do danego wątku). Wówczas wektory lokalne pola elektrycznego $\mathbf{e}_{v1}$ oraz magnetycznego $\mathbf{h}_{v1}$ są zdefiniowane jako

$$\mathbf{e}_{v1}(p) = \mathbf{e}(q) \tag{4.3}$$
$$\mathbf{h}_{v1}(r) = \mathbf{h}(s) \tag{4.4}$$
$$f(p) = q \tag{4.5}$$
$$g(r) = s \tag{4.6}$$

gdzie:
$f(p)$ - funkcja, która odwzorowuje numer próbki w wektorze lokalnym $\mathbf{e}_{v1}$ na numer próbki w wektorze globalnym $\mathbf{e}$,
$g(p)$ - funkcja, która odwzorowuje numer próbki w wektorze lokalnym $\mathbf{h}_{v1}$ na numer próbki w wektorze globalnym $\mathbf{h}$.

Alternatywnym rozwiązaniem jest zdefiniowanie macierzy permutacji próbek pola elektrycznego P_e i magnetycznego P_h. Są one bezpośrednio związane z funkcjami $f(p)$ oraz

$g(p)$ i są określone przez r. (4.7) oraz r. (4.8).

$$P_e(a,b) = \begin{cases} 1 & \text{dla } a = f(b) \\ 0 & \text{dla } a \neq f(b) \end{cases} \tag{4.7}$$

$$P_h(a,b) = \begin{cases} 1 & \text{dla } a = g(b) \\ 0 & \text{dla } a \neq g(b) \end{cases} \tag{4.8}$$

Przy pomocy P_e i P_h zostają zdefiniowane zmienne spermutowane:

$$\mathbf{e}_{v2} = P_e\mathbf{e} \tag{4.9}$$

$$\mathbf{h}_{v2} = P_h\mathbf{h} \tag{4.10}$$

$$\mathbf{R}_{E\,v2} = P_h\mathbf{R}_E P_e^T \tag{4.11}$$

$$\mathbf{R}_{H\,v2} = P_e\mathbf{R}_E P_h^T \tag{4.12}$$

Wówczas równomierny przydział zakresów wektorów $\mathbf{e}_{v2}$ oraz $\mathbf{h}_{v2}$ prowadzi do uzyskania identycznego transferu danych jak przy zastosowaniu selektywnego transferu wybranych elementów wektorów $\mathbf{e}_{v1}$ i $\mathbf{e}_{v1}$ do poszczególnych węzłów obliczeniowych. Różnica pomiędzy obydwoma rozwiązaniami sprowadza się do następujących zagadnień:

1. rozwiązanie z selektywnym wyborem elementów wektora wymaga zdefiniowania dodatkowych tablic, które przechowują wartości funkcji $f(a)$ oraz $f(b)$,

2. zastosowanie odpowiedniej permutacji w metodzie drugiej pozwala uporządkować wektory wymagane do transferu danych pomiędzy węzłami w jednolite bloki, a przez to nie jest wymagany wektor indeksów elementów przeznaczonych do transferu danych, a jedynie zbiory par liczb: indeks pierwszego elementu przeznaczonego do transferu oraz liczba elementów przeznaczona do przesłania,

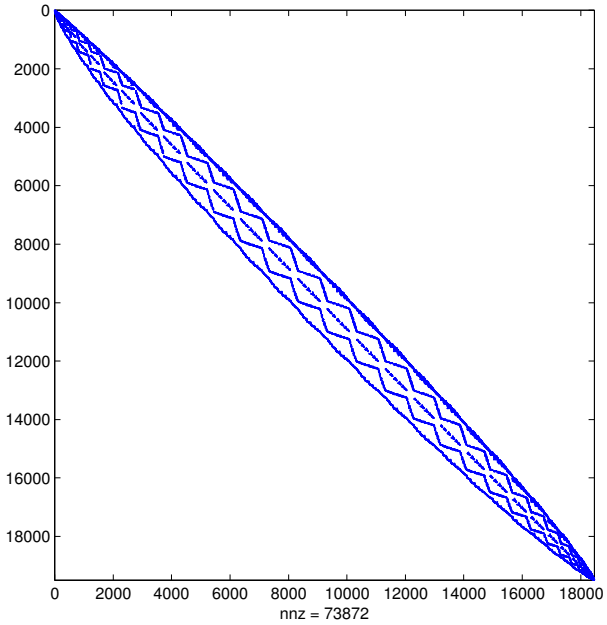
3. zastosowanie permutacji prowadzi do uzyskania mniejszej długości wektora lokalnego w porównaniu do rozwiązania opierającego się o selektywny transfer danych. Jest to związane z tym, że selektywny transfer danych wymaga wykonywanie obliczeń na całym wektorze $\mathbf{e}$ oraz $\mathbf{h}$, podczas, gdy odpowiednia permutacja prowadzi do zmniejszenia zakresu elementów wektorów $\mathbf{e}$ oraz $\mathbf{h}$, do których będzie się odwoływał algorytm w każdym z wątków.

Do analizy numerycznej efektywności wybrano wersję z permutacją elementów macierzy z uwagi na wyżej opisaną przewagę tego rozwiązania.

Optymalizację transferu danych poprzez zastosowanie permutacji można przeprowadzić stosując uniwersalne algorytmy np. odwrotnego porządkowania Cuthill-McKee. Dla problemu o rozmiarze $20 \times 20 \times 20$ rozmieszczenie elementów niezerowych macierzy $\mathbf{R}_E$ po zastosowaniu algorytmu odwrotnego porządkowania Cuthill-McKee przedstawia rys. 4.4. Ten rodzaj permutacji pozwala na znaczące zmniejszenie liczby zmiennych transmitowanych pomiędzy węzłami obliczeniowymi podczas działania algorytmu mnożenia macierzy rzadkiej przez wektor, co wykazano w p. 4.4.4.

4.4.3 Permutacja oparta o własności fizyczne macierzy rotacji

W p. 3.4 wykazano, że algorytm FDTD może być skalowalny w środowisku klastra. Skalowalność algorytmu FDTD jest możliwa do osiągnięcia przy zapewnieniu odpowiednio



Rysunek 4.4: Elementy niezerowe macierzy rotacji $\mathbf{R}_E$ spermutowanej za pomocą algorytmu odwrotnego porządkowania Cuthill-McKee

niskiego poziomu transferu danych pomiędzy węzłami klastra. W zrównoleglonej implementacji algorytmu FDTD w postaci jawnej niski poziom transferu danych uzyskano poprzez uwzględnienie powiązania pomiędzy elementami wektorów $\mathbf{e}$ i $\mathbf{h}$ wynikającego z fizycznej budowy siatki dyskretyzacji. Wówczas wymianie danych podlegają jedynie wybrane wartości próbek pól, które należą do oczek siatki Yee tworzących granicę poddziedzin.

Z powyższych rozważań wynika, że w celu uzyskania dobrej skalowalności algorytmu FDTD w formie macierzowej należy tak zdefiniować ułożenie elementów niezerowych w macierzy, aby występował transfer danych analogiczny do sytuacji występującej w zrównolegleniu algorytmu FDTD w postaci jawnej. Permutacja, która prowadzi do wskazanego ułożenia danych została nazwana w tej rozprawie permutacją opartą o własności fizyczne macierzy rotacji, [104]. Analogiczne rozwiązanie w obliczeniach problemów własnych opisano w [124], [125].

Permutacja ta prowadzi do określenia alternatywnego w odniesieniu do zdefiniowanego w r. (2.12) ułożenia próbek pól w wektorach $\mathbf{e}$ oraz $\mathbf{h}$. Ułożenie to sprowadza się do zgrupowania próbek pól z jednego przekroju dziedziny dyskretnej, np. dla $z = const$, w jednolitym zakresie próbek pola, co w formie matematycznej ujmują r. (4.13).

Tak zdefiniowana permutacja wektora $\mathbf{e}$ prowadzi do ułożenia w bliskim sąsiedztwie próbek pól, które są w bezpośredniej relacji wynikającej z równań Maxwella. W efekcie tych przekształceń macierz $\mathbf{R}_E$ stanowi macierz wstęgową. Przykład macierzy rotacji pola elektrycznego po zastosowaniu permutacji wektorów $\mathbf{e}$ oraz $\mathbf{h}$ przestawiono na rys. 4.5. W zmodyfikowanej w ten sposób macierzy rotacji szerokość wstęgi wynosi $max\{I, J, K\}$, co stanowi znacznie mniejszą wartość niż szerokość wstęgi macierzy podstawowej wynosząca $I \cdot J$.

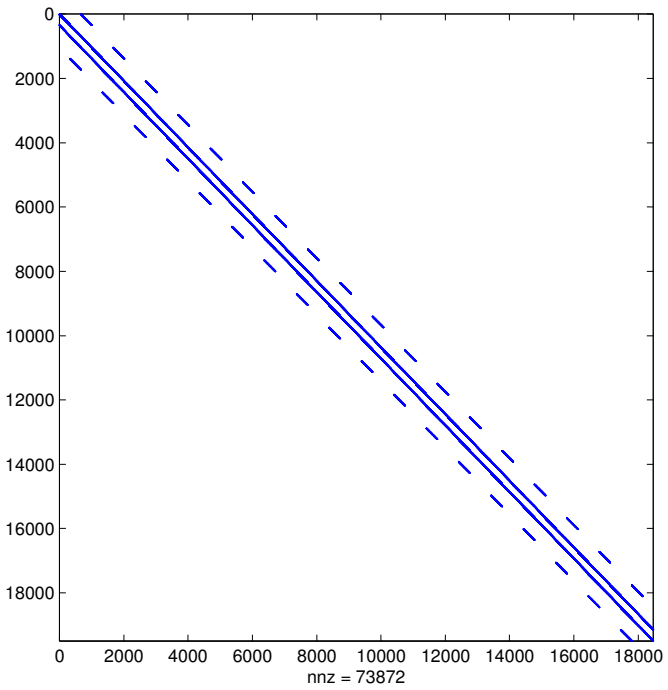
Dodatkową korzyść z tak określonego uporządkowania próbek pól w wektorach $\mathbf{e}$ oraz $\mathbf{h}$ stanowi uzyskanie możliwości przeprowadzenia komunikacji jednokierunkowej pomię-

dzy węzłami przy wyznaczaniu wartości dyskretnej rotacji pola elektrycznego lub magnetycznego. Prowadzi to do znacznego zmniejszenia liczby operacji wymaganych w procesie wymiany danych. Komunikacja jednokierunkowa jest możliwa do osiągnięcia przy zastosowaniu podziału wektorów $\mathbf{e}$ oraz $\mathbf{h}$ pomiędzy poddziedziny zgodnie z granicami wyznaczonymi przez przekrój dziedziny obliczeniowej. Jeśli podział wektorów nie będzie umożliwiał przeprowadzenia jednokierunkowej transmisji danych, to wówczas transmisja taka będzie dwukierunkowa.

$$\mathbf{e} = \left[\mathbf{e}(z=1)^T, \mathbf{e}(z=2)^T, \cdots, \mathbf{e}(z=K)^T\right]^T \tag{4.13a}$$

$$\mathbf{e}(z=k) = \left[\mathbf{e}_x(z=k)^T, \mathbf{e}_y(z=k)^T, \mathbf{e}_z(z=k)^T\right]^T \tag{4.13b}$$

$$\mathbf{e}_\alpha(z=k) = \begin{bmatrix} E_\alpha(1,1,k) \\ E_\alpha(2,1,k) \\ \vdots \\ E_\alpha(I,1,k) \\ E_\alpha(1,2,k) \\ E_\alpha(2,2,k) \\ E_\alpha(3,2,k) \\ \vdots \\ E_\alpha(I,2,k) \\ \vdots \\ E_\alpha(I,J,k) \end{bmatrix} \tag{4.13c}$$



Rysunek 4.5: Elementy niezerowe macierzy dla macierzy powstałej po zastosowaniu permutacji opartej o własności fizyczne macierzy rotacji

4.4.4 Testy numeryczne

W celu zilustrowania różnicy efektywności zrównoleglonego algorytmu FDTD w formie macierzowej przy zastosowaniu różnych wariantów ułożenia próbek pól w wektorach $\mathbf{e}$ oraz $\mathbf{h}$ zostaną zaprezentowane wyniki testów wykonanych dla prostopadłościennego rezonatora.

Implementacja zrównoleglonego algorytmu FDTD w formie macierzowej została wykonana z zastosowaniem biblioteki MPI, [64]. Macierze rzadkie są reprezentowane w formacie CRS, który opisano w p. C.3.

4.4.4.1 Symulacje problemów o niewielkich rozmiarach

Rezultaty testów, które zostały opisany w tym podpunkcie, zostały przeprowadzone na komputerze PC2. Do testów wybrano problem o rozmiarze $21 \times 21 \times 23$ komórek siatki Yee ($N \approx 10000$). Wówczas $log_2(N) \approx 13,3$, czyli biorąc pod uwagę r. (4.1), rozmiar problemu odpowiada zakresowi, dla którego w implementacji jawnej uzyskano skalowalność algorytmu najbliższą liniowej (rys. 3.17).

Symulacje zostały przeprowadzone dla trzech rodzajów ułożenia elementów niezerowych w macierzy:

1. macierzy podstawowej zdefiniowanej przez r. (2.14),
2. macierzy z zastosowaną permutacją wynikającą z odwrotnego porządkowania Cuthill-McKee,
3. macierzy z zastosowaną permutacją zgodną z własnościami fizycznymi dziedziny dyskretnej.

Testy zrealizowane dla problemu o małych rozmiarach stanowią istotną część oceny skalowalności algorytmu FDTD w formie macierzowej, gdyż ilustrują one dużą wrażliwość efektywności zrównoleglenia na czas transferu danych.

Znacząca różnica pomiędzy liczbą danych przesyłanych pomiędzy wątkami przy zastosowaniu różnego uporządkowania wektorów próbek pól została przedstawiona w tab. 4.1. Podane w tabeli wartości potwierdzają znaczną liczbę danych niezbędną do przesłania w sytuacji, gdy w obliczeniach zastosowana jest podstawowa postać macierzy rotacji. Natomiast permutacja oparta o własności fizyczne dziedziny dyskretnej wyróżnia się zdecydowanie najmniejszym poziomem transmisji danych w obliczeniach zrównoleglonego macierzowego algorytmu FDTD.

W tabelach 4.2 ÷ 4.5 przedstawiono wielkość transmisji danych pomiędzy wątkami, która jest wymagana przy aktualizacji wartości próbek pól magnetycznych. W tabelach tych przez komunikację w zakresie tego samego wątku oznaczono stopień wykorzystania w obliczeniach danych przydzielonych do tego wątku. W tab. 4.5 wykazano, że przy uporządkowaniu danych z uwzględnieniem własności fizycznych dziedziny dyskretnej, występuje najmniejszy poziom transmisji danych pomiędzy wątkami oraz największe wykorzystanie do obliczeń danych przydzielonych do poszczególnych wątków. Natomiast najmniej korzystna sytuacja, z punktu widzenia efektywności obliczeń, została zaprezentowana w tab. 4.2, gdzie liczba danych wymagana do transmisji jest zdecydowanie największa spośród wszystkich zaprezentowanych rozwiązań, co pośrednio jest związane z tym, że jedna próbka pola jest stosowana w obliczeniach co najwyżej dwukrotnie.

Różnica pomiędzy transmisją dwukierunkową a jednokierunkową jest widoczna przy porównaniu tab. 4.4 oraz tab. 4.5. Kosztem osiągnięcia transmisji jednokierunkowej jest

Tablica 4.1: Całkowity transfer danych dla różnego uporządkowania wektorów próbek pól

	macierz podstawowa	Cuthill-McKee	permutacja oparta o własności fizyczne dziedziny dyskretnej
całkowita liczba próbek pola elektrycznego	29106	29106	29106
całkowita liczba próbek pola magnetycznego	30492	30492	30492
liczba próbek pola elektrycznego przesłana w każdej iteracji	61953	3824	2772
liczba próbek pola magnetycznego przesłana w każdej iteracji	59180	3605	2772

Tablica 4.2: Liczba próbek pól transmitowana pomiędzy wątkami w każdej iteracji przy zastosowaniu w obliczeniach macierzy podstawowej

wątki nadawcze:	wątek 1	wątek 2	wątek 3	wątek 4
wątek 1 (odbiorczy)	0	5312	5324	5093
wątek 2 (odbiorczy)	5082	0	5082	5083
wątek 3 (odbiorczy)	5197	5429	0	5082
wątek 4 (odbiorczy)	5087	5098	5084	0

Tablica 4.3: Liczba próbek pól transmitowana pomiędzy wątkami w każdej iteracji przy zastosowaniu permutacji macierzy podstawowej wynikającej z odwrotnego porządkowania Cuthill-McKee

wątki nadawcze:	wątek 1	wątek 2	wątek 3	wątek 4
wątek 1 (odbiorczy)	7617	367	0	0
wątek 2 (odbiorczy)	836	7621	551	0
wątek 3 (odbiorczy)	0	821	7579	573
wątek 4 (odbiorczy)	0	0	676	7623

nierównomierny przydział wektorów próbek do wątków, który jest równoznaczny z nierównomierną liczbą działań wykonywanych przez wątki. Jednak testy numeryczne, których rezultaty przedstawia rys. 4.6, wskazują, że zmniejszenie liczby operacji realizujących transmisję danych pomiędzy wątkami powoduje wyraźny wzrost efektywności zrównoleglenia algorytmu FDTD.

Zobrazowanie zmian efektywności zrównoleglenia przy zastosowaniu różnej liczby procesorów oraz dla różnego uporządkowania elementów wektorów $\mathbf{e}$ i $\mathbf{h}$ przedstawiono na rys. 4.6. Rysunek ten wyraźnie wykazuje, że skalowalność algorytmu silnie zależy od wielkości transmisji danych. Najlepszy poziom skalowalności jest porównywalny lub nawet

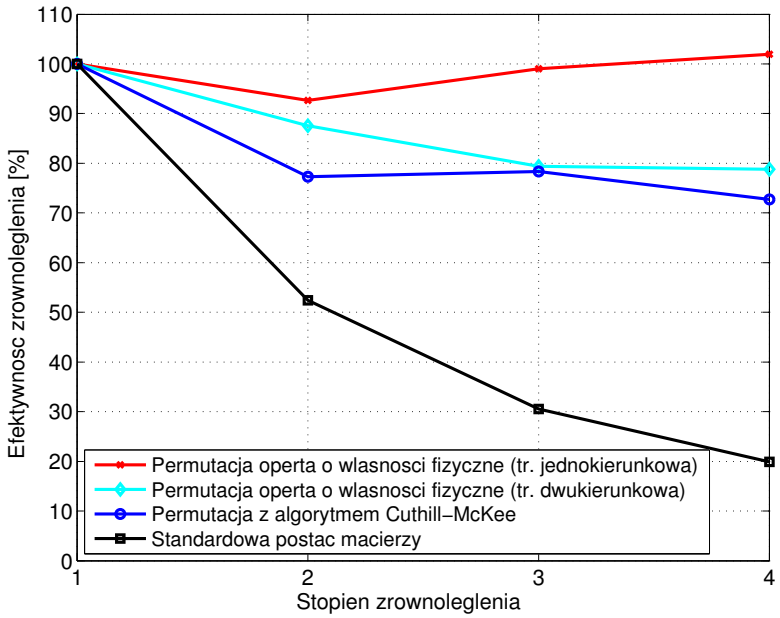
Tablica 4.4: Liczba próbek pól transmitowana pomiędzy wątkami w każdej iteracji przy zastosowaniu permutacji macierzy podstawowej opartej o własności fizyczne dziedziny dyskretnej i zastosowaniu transmisji dwukierunkowej

wątki nadawcze:	wątek 1	wątek 2	wątek 3	wątek 4
wątek 1 (odbiorczy)	7623	573	0	0
wątek 2 (odbiorczy)	371	7623	705	0
wątek 3 (odbiorczy)	0	702	7623	364
wątek 4 (odbiorczy)	0	0	572	7623

Tablica 4.5: Liczba próbek pól transmitowana pomiędzy wątkami w każdej iteracji przy zastosowaniu permutacji macierzy podstawowej opartej o własności fizyczne dziedziny dyskretnej i zastosowaniu transmisji jednokierunkowej

wątki nadawcze:	wątek 1	wątek 2	wątek 3	wątek 4
wątek 1 (odbiorczy)	7964	924	0	0
wątek 2 (odbiorczy)	0	8448	924	0
wątek 3 (odbiorczy)	0	0	7040	924
wątek 4 (odbiorczy)	0	0	0	7040

lepszy od skalowalności uzyskanej przy zrównolegleniu algorytmu FDTD w postaci jawnej udokumentowanej w pracach [33, 122]. Dodatkowo można zauważyć, że przy permutacji opartej o własności fizyczne dziedziny dyskretnej i przy zastosowaniu transmisji jednokierunkowej efektywność zrównoleglenia osiąga wartość ponad 100%. Jest to związane z większą efektywnością algorytmu mnożenia macierzy rzadkiej przez wektor przy obliczeniach wykonywanych na mniejszych macierzach, zgodnie z rys. 4.1.



Rysunek 4.6: Skalowalność zrównoleglonego macierzowego algorytmu FDTD w zależności od zastosowanej permutacji macierzy rotacji pól

Podsumowując, można stwierdzić, że chociaż sformułowanie macierzowe problemu jest dużo bardziej elastyczne w przeprowadzaniu modyfikacji algorytmu, to jednak korzystnie jest uwzględnić własności dziedziny dyskretnej w procesie zrównoleglenia algorytmu FDTD w postaci macierzowej.

4.4.4.2 Symulacje problemów o znacznych rozmiarach

W tym punkcie porównano skalowalność napisanego przez autora tej rozprawy algorytmu do skalowalności zrównoleglonego algorytmu FDTD w postaci jawnej zaprezentowanej w pracach [33, 39, 122]. Autorzy tych prac uruchomili swoje programy w sieci komputerowej przy stosującej do komunikacji *Fast Ethernet*, [39], oraz *Gigabit Ethernet*, [33, 122].

Podobnie jak w poprzednim punkcie analizowana struktura stanowi rezonator prostopadłościenny. Rozmiary dziedziny dyskretnej zostały ustalone na $55 \times 55 \times 120$ oczek siatki Yee w celu otrzymania rozmiaru problemu zbliżonego do tych zastosowanych we wspomnianych pracach.

W celu ukazania efektywności obliczeń w środowisku heterogenicznym do obliczeń zostały zastosowane cztery komputery osobiste o różnej mocy obliczeniowej, nazywane w dalszej części tego punktu węzłami, połączonych ze sobą w sieci *Fast Ethernet*.

Tab. 4.6 przedstawia czas symulacji całego problemu zmierzony na wskazanym komputerze oraz teoretyczne przyspieszenie obliczeń S_n jeśli obliczenia zostałyby przydzielone do k węzłów obliczeniowych ($k \in \{1, 2, 3, 4\}$).

Wartość teoretycznego przyspieszenia obliczeń została ustalona przy użyciu r. (4.14).

$$S_n = \frac{T_n}{\min\{t_{wi}\}} \tag{4.14}$$

$$T_n = \frac{1}{\sum_{i=1}^{n} t_{wi}^{-1}} \tag{4.15}$$

$$E_n = \frac{t_n}{T_n} \tag{4.16}$$

gdzie
t_{wi} - [s] czas symulacji całego problemu wymagany przez i-ty węzeł,
T_n - minimalny czas wymagany przez symulację z zastosowaniem pierwszych n węzłów,
S_n - przyspieszenie obliczeń z zastosowaniem pierwszych n węzłów w odniesieniu do czasu obliczeń zmierzonego dla najszybszego węzła ($i < n$),
t_n - [s] zmierzony czas symulacji z zastosowaniem pierwszych n węzłów,
E_n - efektywność zrównoleglenia algorytmu FDTD.

Tablica 4.6: Czas symulacji problemu o rozmiarze jednego miliona zmiennych, teoretyczne przyspieszenie obliczeń przy obliczeniach przeprowadzonych na pierwszych k węzłach, efektywność zrównoleglenia dla permutacji opartej na fizycznych własnościach dziedziny obliczeniowej z zastosowaniem transmisji jednokierunkowej

	węzeł 1	węzeł 2	węzeł 3	węzeł 4
czas symulacji [s]	148	185	209	570
teoretyczne przyspieszenie obliczeń	-	1,80	2,51	2,76
efektywność zrównoleglenia [%]	-	96,7	100,2	93,1

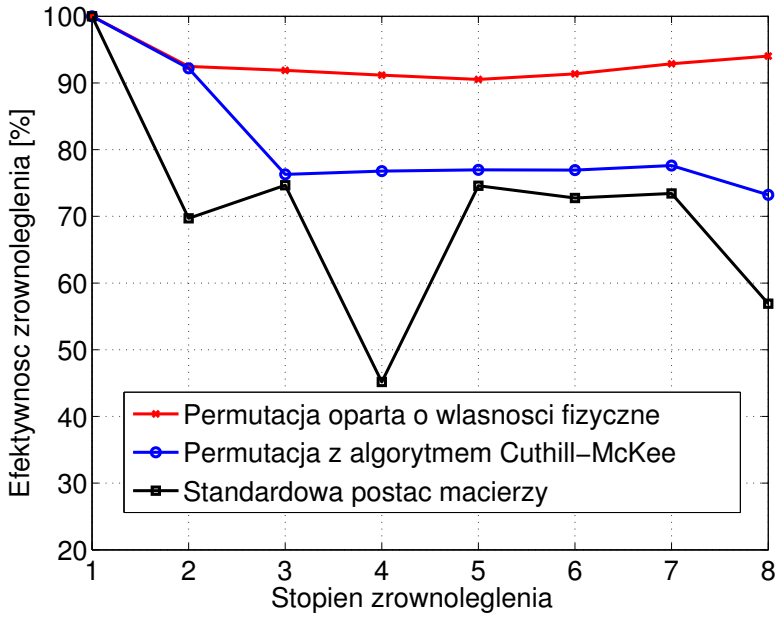
Obserwowany w tab. 4.6 wzrost efektywności powyżej 100% związany jest z większą efektywnością algorytmu mnożenia macierzy rzadkiej przez wektor w sytuacji zastosowania go wobec macierzy mniejszych rozmiarów, rys. 4.1.

Wysoka efektywność opracowanego algorytmu wynika z równoległego przeprowadzenia obliczeń oraz transmisji danych podczas wykonywania jednego mnożenia macierzy rzadkiej przez wektor. Algorytm mnożenia macierzy rzadkiej A przez wektor x został zrealizowany w następujących etapach:

1. zacznij transmisję danych wymaganych przez inne węzły,
2. oblicz $A_{Inner}x$,
3. zaczekaj na dokończenie transmisji danych,
4. oblicz $A_{External}x$.

gdzie:
A_{Inner} oznaczą tą część macierzy A, która wymaga do obliczeń danych przechowywanych lokalnie,
$A_{External} = A - A_{Inner}$ reprezentuje tą część macierzy A, która wymaga do obliczeń danych z innych węzłów.

Przedstawiony algorytm został również zastosowany do obliczeń na klastrze z procesorami jednordzeniowymi Itanium2 1,3 GHz. Rys. 4.7 przedstawia stopień osiągniętego przyspieszenia w tym środowisku.



Rysunek 4.7: Efektywność algorytmu FDTD w postaci macierzowej z zastosowaną permutacją opartą o fizyczne własności dziedziny dyskretnej przeprowadzone na klastrze

Jak zostało przedstawione w tym punkcie, skalowalność algorytmu FDTD w postaci macierzowej jest możliwa do osiągnięcia. Zmierzona efektywność zrównoleglenia jest porównywalna lub większa od wartości otrzymanych dla implementacji jawnej, które zostały przedstawione w publikacjach naukowych (patrz tab. 4.7). Wysoką efektywność obliczeń udokumentowano zarówno dla klastra jak i dla grupy komputerów osobistych połączonych siecią lokalną.

Tablica 4.7: Porównanie skalowalności algorytmu FDTD udokumentowanego w publikacjach naukowych oraz w tej rozprawie

źródło	efektywność	rozmiar problemu	zasoby sprzętowe
[39]	95 %	4,8M	PC grid Fast Eth.
[33]	75 %	1M	PC grid G. Eth.
[122]	91 %	1,15M	PC grid G. Eth.
opracowanie własne	91 %	1M	klaster z Itanium2
opracowanie własne	93 %	1M	PC grid Fast Eth.

4.5 Implementacja mieszana algorytmu FDTD przeznaczona dla GPU

Niższy poziom efektywności obliczeń implementacji macierzowej algorytmu FDTD w porównaniu do implementacji jawnej skłania do poszukiwania nowych rozwiązań algorytmicznych, które pozwoliłyby połączyć zalety obu rozwiązań: szybkość obliczeń schematu różnicowego nie poddanego modyfikacjom oraz możliwość wprowadzenia usprawnień do algorytmu FDTD w wybranym obszarze dziedziny obliczeniowej[2]. W rezultacie opracowano implementację mieszaną algorytmu FDTD, która zawiera elementy algorytmu FDTD zarówno w formie jawnej jak i macierzowej. W procesie łącznia obu form algorytmu najważniejszy etap prac stanowi odpowiednie uporządkowanie danych reprezentujących próbki pola elektrycznego i magnetycznego.

Uporządkowanie danych w implementacji jawnej przeznaczonej dla GPU zostało określone poprzez podział dziedziny dyskretnej na bloki danych grupujące $16 \times 16 \times 1$ oczek siatki Yee. W rezultacie wektory próbek pola elektrycznego i magnetycznego składają się z trzech wektorów odpowiadających składowym pola w danym kierunku przestrzeni, z których każdy składa się z grupy zbiorów 256 próbek pola zawartych w poszczególnych blokach danych. Ponieważ struktura danych zdefiniowana w języku C, która reprezentuje blok danych, zawiera informację czy w danym kierunku przestrzeni sąsiaduje z nim kolejny blok danych, to w trakcie obliczeń dla poszczególnych bloków, dane z sąsiedniego bloku są uwzględnianie w obliczeniach warunkowo i jedynie za pośrednictwem wskaźnika do odpowiedniego elementu wektora danych. Jeden blok, który w każdym kierunku dziedziny obliczeniowej posiada sąsiadujący blok, wymaga dostępu do następującego zbioru danych:

- 256 próbek pola E_x oraz 256 pola E_y z bloku danych sąsiadującego w kierunku +z,
- 16 próbek pola E_y oraz 16 pola E_z z bloku danych sąsiadującego w kierunku +x,
- 16 próbek pola E_x oraz 16 pola E_z z bloku danych sąsiadującego w kierunku +y,
- 256 próbek pola H_x oraz 256 pola H_y z bloku danych sąsiadującego w kierunku -z,
- 16 próbek pola H_y oraz 16 pola H_z z bloku danych sąsiadującego w kierunku -x,
- 16 próbek pola H_x oraz 16 pola H_z z bloku danych sąsiadującego w kierunku -y.

Odpowiednia organizacja danych w algorytmie jawnym umożliwia przeprowadzenie łączenia z algorytmem mieszanym w elastyczny sposób, tzn. może ono zostać przeprowadzone dla różnych rozmiarów zarówno dziedziny obliczeniowej jak i obszaru macierzowego.

[2] w szczególności zastosowanie makromodeli (rozdział 7) wymaga zastosowania zapisu macierzowego

Polega ono na wyłączeniu z obliczeń w algorytmie jawnym określonego zbioru bloków danych grupujących $16 \times 16 \times 1$ oczek siatki Yee, które prowadzi do zmniejszenia długości wektorów próbek pól zarządzanych przez algorytm jawny oraz wymaga zakodowania informacji w numerze sąsiedniego bloku o tym, że dane powinny zostać pobrane z części macierzowej algorytmu.

Proces wprowadzenia zmian do implementacji algorytmu FDTD w formie jawnej został tak przeprowadzony, że po jego zakończeniu efektywność obliczeń pozostała na wysokim poziomie. Jednak taka własność implementacji mieszanej została osiągnięta po przeprowadzeniu wnikliwej optymalizacji kodu programu, tak iż w obliczeniach, które uwzględniają algorytm CPML (p. 7.1), bierze udział 100% dostępnych rejestrów oraz 99% dostępnej pamięci dzielonej (dla urządzeń kompatybilnych z CUDA generacji (ang. *compute capability*) 1.0). Optymalizacja taka była możliwa do przeprowadzenia dzięki pozostawieniu programiście znacznie większej kontroli nad zasobami sprzętowymi przy programowaniu akceleratorów graficznych w porównaniu do programowania procesorów komputerowych.

Prawidłowa komunikacja pomiędzy algorytmem jawnym i macierzowym wymaga przeprowadzenia następujących zmian w algorytmie macierzowym:

1. uporządkowania wektora danych algorytmu macierzowego w taki sposób, aby w części początkowej zawierał on tylko próbki graniczne tzn. próbki, które są styczne do granicy obszaru macierzowego,
2. uporządkowania próbek granicznych w taki sposób, aby algorytm jawny mógł pobierać z niego dane w sposób analogiczny jak pobiera z bloków danych z algorytmu jawnego.

Ponieważ część macierzowa implementacji jawnej posiada narzuconą postać tylko początkowej części wektorów danych $\mathbf{e}$ i $\mathbf{h}$, to możliwe jest wprowadzenie modyfikacji schematu różnicowego w obszarze nie należącym do granicy obszaru macierzowego. W ten sposób zrealizowano współpracę implementacji jawnej ze schematem różnicowym zawierającym makromodel. Modyfikacja algorytmu macierzowego wymaga ponadto jeszcze dwóch działań:

1. przeprowadzenia odsymetryzowania schematu różnicowego dla próbek granicznych,
2. zdefiniowania macierzy sprzęgającej część macierzową z jawną: aktualizuje ona wartości próbek z części macierzowej na podstawie wartości próbek z części jawnej. Ponieważ wektor próbek części jawnej jest różny dla różnych rozmiarów dziedziny obliczeniowej, wymagana jest odpowiednia modyfikacja indeksów z macierzy rzadkiej wykonana przez część jawną algorytmu.

Odsymetryzowanie schematu różnicowego dla próbek granicznych wymagane jest ponieważ algorytm jawny realizuje obliczenia na nieprzeskalowanych wektorach próbek pól $\mathbf{e}$ i $\mathbf{h}$, natomiast z uwagi na chęć zachowania stabilności algorytmu FDTD również po wprowadzeniu zmian do schematu różnicowego, część macierzowa algorytmu realizuje działania na przeskalowanych wektorach próbek pól $\tilde{\mathbf{e}}$ i $\tilde{\mathbf{h}}$ (patrz r. (2.26a) i (2.26b)). Poprawność obliczeń wykonywanych w algorytmie mieszanym osiągnięto poprzez odsymetryzowanie wektorów próbek pól $\tilde{\mathbf{e}}$ i $\tilde{\mathbf{h}}$ dla macierzy sprzęgających $\mathbf{R}_{ES}$ i $\mathbf{R}_{HS}$ (opis poniżej) oraz dla próbek pól należących do komórek Yee w algorytmie macierzowym stycznych do komórek Yee zarządzanych przez algorytm jawny. Odsymetryzowanie jest realizowane poprzez nadanie wartości równej jeden w odpowiednich wierszach macierzy $\mathbf{D}_{\epsilon}^{\frac{1}{2}}$ i $\mathbf{D}_{\mu}^{\frac{1}{2}}$ (patrz r. (2.26)).

Algorytm mieszany, który zawiera algorytm macierzowy z makromodelem, przedstawiają w sposób poglądowy r. (4.17).

$$\mathbf{e}_j^n = \mathbf{e}_j^{n-1} + rot_H\left(\mathbf{h}_j^{n-0,5}, \mathbf{h}_m^{n-0,5}\right) \tag{4.17a}$$

$$\mathbf{e}_m^n = \mathbf{e}_m^{n-1} + \mathbf{R}_{Hp}\mathbf{h}_m^{n-0,5} + \mathbf{R}_{HS}\mathbf{h}_j^{n-0,5} + \mathbf{h}_b^{n-0,5} \tag{4.17b}$$

$$\mathbf{h}_j^{n+0,5} = \mathbf{h}_j^{n-0,5} - rot_E\left(\mathbf{e}_j^n, \mathbf{e}_m^n\right) \tag{4.17c}$$

$$\mathbf{h}_j^{n+0,5} = \mathbf{h}_m^{n-0,5} - \mathbf{R}_{Ep}\mathbf{e}_m^n - \mathbf{R}_{ES}\mathbf{e}_j^n \tag{4.17d}$$

$$\mathbf{e}_M^n = \mathbf{L}_E\mathbf{e}_m^n \tag{4.17e}$$

$$\mathbf{f}_M^n = \mathbf{B}_E\mathbf{e}_M^n \tag{4.17f}$$

$$\mathbf{g}_M^{n+0,5} = \mathbf{\Gamma}_p\mathbf{g}_M^{n-0,5} - \mathbf{g}_M^{n-1,5} + \mathbf{F}_M^n - \mathbf{F}_M^{n-1} \tag{4.17g}$$

$$\mathbf{h}_M^{n+0,5} = \mathbf{B}_H\mathbf{g}_M^{n+0,5} \tag{4.17h}$$

$$\mathbf{h}_b^{n+0,5} = \mathbf{L}_H\mathbf{h}_M^{n+0,5} \tag{4.17i}$$

gdzie:
$\mathbf{e}_j^n$, $\mathbf{h}_j^{n-0,5}$ - wektory próbek pól elektrycznego i magnetycznego, które zostały zdefiniowane w implementacji jawnej i reprezentują pole w czasie odpowiednio $n\Delta_t$ i $(n-0,5)\Delta_t$ (Δ_t - krok dyskretyzacji dziedziny czasu),
$\mathbf{e}_j^n$, $\mathbf{h}_j^{n-0,5}$ - wektory próbek pól elektrycznego i magnetycznego, które zostały zdefiniowane w implementacji macierzowej i reprezentują pole w czasie odpowiednio $n\Delta_t$ i $(n-0,5)\Delta_t$,
$\mathbf{e}_M^n$, $\mathbf{f}_M^n$, $\mathbf{g}_M^{n+0,5}$, $\mathbf{h}_M^{n+0,5}$, $\mathbf{h}_b^{n+0,5}$ - wektory stanu makromodelu (patrz rozdział 6),
rot_E, rot_H - obliczenie rotacji pola elektrycznego oraz magnetycznego na podstawie wektorów stanu w algorytmie jawnym,
$\mathbf{R}_{Ep}$, $\mathbf{R}_{Rp}$ - przeskalowane macierze rotacji pola elektrycznego oraz magnetycznego,
$\mathbf{R}_{ES}$, $\mathbf{R}_{RS}$ - macierze sprzęgające algorytm jawny oraz macierzowy,
$\mathbf{\Gamma}_p$, $\mathbf{B}_E$, $\mathbf{B}_H$ - macierze, które opisują funkcję przejścia makromodelu,
$\mathbf{L}_E$, $\mathbf{L}_H$ - macierze sprzęgające makromodel z obszarem podstawowym algorytmu macierzowego.

Równania (4.17) przedstawiają algorytm FDTD, w którym współistnieją ze sobą:

1. implementacja jawna algorytmu FDTD (r. (4.17a) i (4.17c)),
2. implementacja macierzowa algorytmu FDTD (r. (4.17b) i (4.17d) ÷ (4.17i)),
3. implementacja makromodeli w algorytmie macierzowym (r. (4.17e) ÷ (4.17i)).

Przedstawione równania stanowią jedynie poglądowe zobrazowanie przeprowadzonych działań. W rzeczywistej implementacji każde równanie jest realizowane przez jedno lub więcej funkcji przeznaczonych dla akceleratora graficznego. Przedstawiony zestaw zadań wyraźnie ilustruje wyodrębnienie działań podstawowych, które mogę zostać zoptymalizowane do obliczeń jak najbardziej efektywnych, w szczególności:

1. r. (4.17a) oraz (4.17c) reprezentują obliczenia wykonane za pomocą algorytmu jawnego, przy czym musi on zostać dostosowany do efektywnego pobierania danych z wektora próbek zdefiniowanych przez algorytm macierzowy,
2. r. (4.17b) oraz (4.17d) stanowią operacje mnożenia macierzy rzadkich przez wektor (realizacja rotacji pola elektrycznego oraz magnetycznego oraz uwzględnienie obszarów, które należą do algorytmu jawnego oraz do makromodelu),

3. r. (4.17e) oraz (4.17i) realizują operacje wybierania pól stanowiących pobudzenie makromodeli oraz umieszczania odpowiedzi makromodeli w wektorze odpowiedzi; zostały one poddane silnej optymalizacji, tak iż realizują minimalną liczbę niezbędnych działań i mimo, że opierają się na mnożeniu macierzy rzadkiej przez wektor, czas ich wykonania jest znikomy w odniesieniu do czasu wykonania całego algorytmu,
4. r. (4.17g) uwzględnia zastosowanie makromodeli w algorytmie, przy czym wszelkie operacje wykonane w tej części stanowią operacje na macierzach gęstych,
5. r. (4.17f) i r. (4.17h) dokonują przejścia z przestrzeni dyskretnej obszaru macierzowego do przestrzeni zmiennych stanu makromodelu poprzez wykonanie mnożenia macierzy gęstej przez wektor stanu.

Ocena efektywności implementacji mieszanej została przeprowadzona w rozdziale 7.

ROZDZIAŁ 5

Algorytm FDTD z lokalnym zagęszczeniem siatki dyskretyzacji

Przeprowadzenie dokładnej, trójwymiarowej, pełnofalowej analizy struktury za pomocą algorytmu FDTD wymaga znacznej mocy obliczeniowej procesorów oraz znacznych zasobów pamięci z nimi współpracujących. Z uwagi na to, w celu uzyskania większej efektywności obliczeń dla tych samych zasobów sprzętowych, dąży się m.in. do wprowadzenia takich modyfikacji podstawowego algorytmu różnicowego, aby zadowalająca dokładność obliczeń została osiągnięta przy zastosowaniu jak najmniejszej liczby zmiennych.

Przedstawione w tej rozprawie usprawnienia algorytmiczne, które zwiększają efektywność obliczeń metody FDTD, sprowadzają się do zastosowania jak najmniejszej liczby zmiennych przy jak najmniejszym lokalnym kroku dyskretyzacji.

Pierwsze z proponowanych rozwiązań polega na wprowadzeniu lokalnego zagęszczenia siatki dyskretyzacji do podstawowego schematu różnicowego. Rezultatem takiego postępowania jest brak konieczności zagęszczenia siatki w całej dziedzinie obliczeniowej w celu zwiększenia dokładności obliczeń, [56, 55, 53, 46]. Drugim rozpatrywanym w rozprawie rozwiązaniem poprawiającym efektywność algorytmu FDTD są makromodele. Makromodel oznacza fragment przestrzeni obliczeniowej, którego liczba zmiennych stanu została zmniejszona za pomocą technik redukcji rzędu modelu, [47, 49, 50, 52, 53, 57, 76, 103]. Metoda ta jest silnie związana z metodą lokalnego zagęszczenia siatki, co zostanie wykazane w rozdziale 6.

Dokładny opis procesu wyodrębnienia podprzestrzeni obliczeniowej $\widehat{\Omega}$, definiowania lokalnego zagęszczenia siatki oraz budowania makromodelu zawarto w rozprawie [43]. W rozdziałach 5 i 6 przedstawiono analizę kosztów numerycznych jakie są związane z wprowadzeniem modyfikacji do części iteracyjnej algorytmu FDTD, zaprezentowano rezultaty badań efektywności implementacji tych algorytmów dla różnych zasobów sprzętowych,

a także zaproponowano metodę optymalizacji opisanych rozwiązań algorytmicznych.

5.1 Wprowadzenie lokalnego zagęszczenia siatki dyskretyzacji do schematu różnicowego

W obu wymienionych we wstępie tego rozdziału metodach kluczową rolę odgrywa wyodrębnienie podprzestrzeni obliczeniowej $\widehat{\Omega}$ z całej przestrzeni obliczeniowej Ω, co poglądowo przedstawiono na rys. 5.1. Dziedzina obliczeniowa Ω z wyłączeniem $\widehat{\Omega}$ została w dalszej części rozprawy oznaczona jako $\Omega\backslash\widehat{\Omega}$.

Pierwszym etapem wyodrębnienia podprzestrzeni obliczeniowej $\widehat{\Omega}$ jest usunięcie zmiennych stanu tej podprzestrzeni z macierzy związanych z opisem całej przestrzeni Ω. Proces usuwania zmiennych stanu przeprowadzony jest za pomocą operatorów projekcji. Opis sposobu budowania operatorów projekcji dla problemów różnicowych jest zawarty w rozprawie [114].

Eliminacja próbek pól wybranych komórek Yee z wektorów stanu oraz macierzy, które opisują badany problem elektromagnetyczny, wymaga zdefiniowania dwóch operatorów projekcji: operatora projekcji dla pola elektrycznego $\mathbf{P}_E$ oraz magnetycznego $\mathbf{P}_H$. Operator projekcji $\mathbf{P}_E$ budowany jest poprzez modyfikację macierzy jednostkowej o wymiarze równym liczbie próbek pola elektrycznego z dziedziny obliczeniowej. W celu eliminacji próbek pola elektrycznego z wektora $\mathbf{e}$ o numerach ze zbioru $C_E = \{c_1, c_2, \cdots, c_p\}$ należy we wspomnianej macierzy jednostkowej usunąć wszystkie kolumny o numerach zawartych w C_E, co prowadzi do zdefiniowania $\mathbf{P}_E$. Operator projekcji $\mathbf{P}_H$ budowany jest w sposób analogiczny.

Zastosowanie operatorów projekcji pozwala na zdefiniowanie macierzy oraz wektorów stanu, które reprezentują $\Omega\backslash\widehat{\Omega}$. Zostały one określone w r. (5.1)

$$\mathbf{e}' = \mathbf{P}_E^T\mathbf{e} \tag{5.1a}$$

$$\mathbf{h}' = \mathbf{P}_H^T\mathbf{h} \tag{5.1b}$$

$$\mathbf{R}'_E = \mathbf{P}_H^T\mathbf{R}_E\mathbf{P}_E \tag{5.1c}$$

$$\mathbf{R}'_H = \mathbf{P}_E^T\mathbf{R}_H\mathbf{P}_H \tag{5.1d}$$

$$\mathbf{D}'_\epsilon = \mathbf{P}_E^T\mathbf{D}_\epsilon\mathbf{P}_E \tag{5.1e}$$

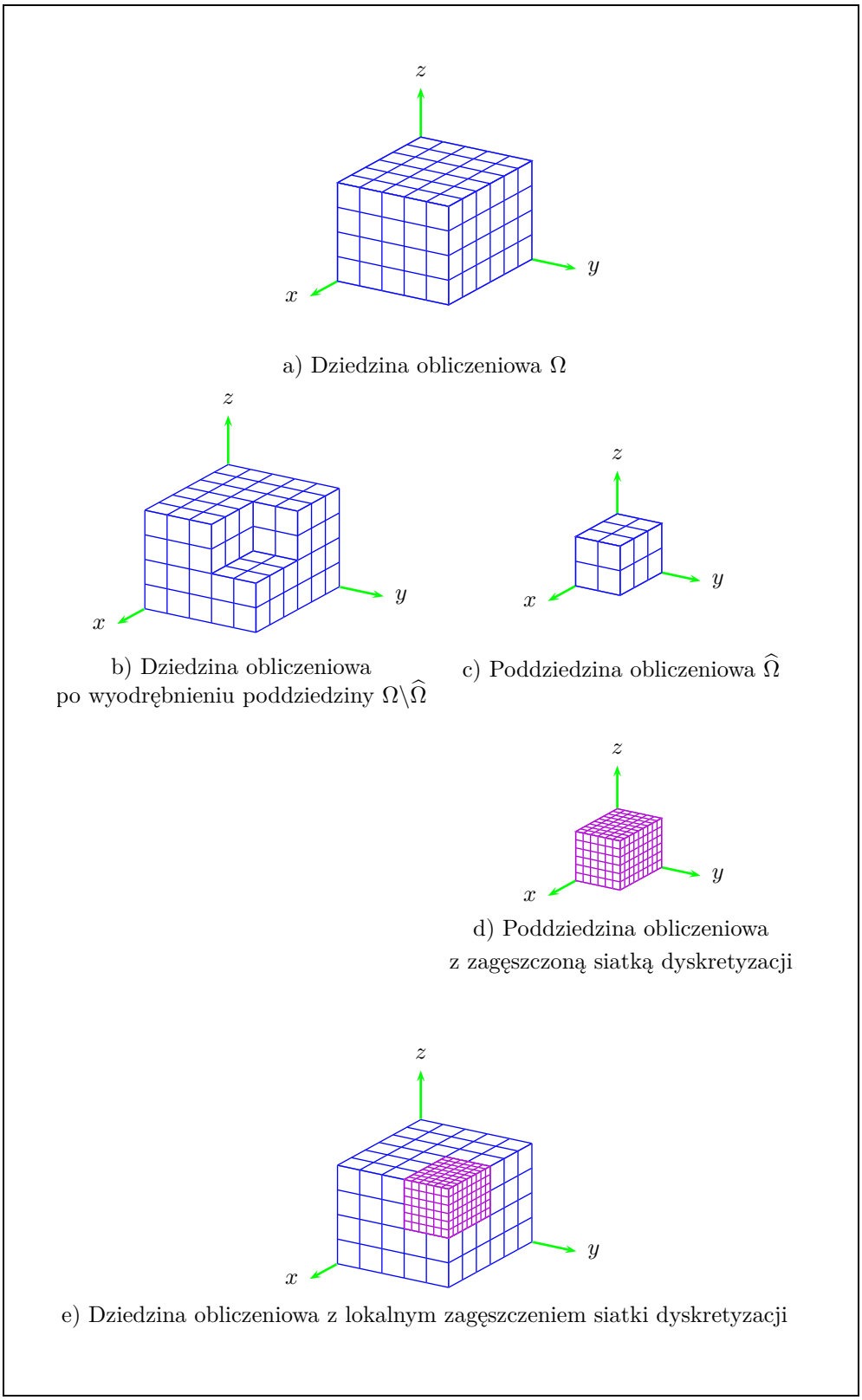
$$\mathbf{D}'_\mu = \mathbf{P}_H^T\mathbf{D}_\mu\mathbf{P}_H \tag{5.1f}$$

Następnie budowane są macierze przedstawione w r. (5.2), które opisują relacje pomiędzy próbkami pól wewnątrz poddziedziny obliczeniowej $\widehat{\Omega}$. Macierze te mogą zostać zbudowane dla siatki dyskretyzacji, której krok dyskretyzacji jest mniejszy w porównaniu do kroku dyskretyzacji siatki podstawowej zastosowanej w Ω. Wówczas mamy do czynienia z lokalnym zagęszczeniem siatki dyskretyzacji (ang. *subgridding*).

$$\widehat{\mathbf{R}}_E\widehat{\mathbf{e}} = -s\widehat{\mathbf{D}}_\mu\widehat{\mathbf{h}} \tag{5.2a}$$

$$\widehat{\mathbf{R}}_H\widehat{\mathbf{h}} = s\widehat{\mathbf{D}}_\epsilon\widehat{\mathbf{e}} \tag{5.2b}$$

Uwzględnienie relacji pomiędzy próbkami z $\Omega\backslash\widehat{\Omega}$ oraz z $\widehat{\Omega}$ zrealizowane jest poprzez zastosowanie macierzy sprzęgających: $\widehat{\mathbf{S}}_E$ oraz $\mathbf{S}_H$. Macierz $\widehat{\mathbf{S}}_E$ wprowadza informację



Rysunek 5.1: Etapy wprowadzania lokalnego zagęszczenia siatki do schematu różnicowego

o polu elektrycznym z wektora stanu $\Omega\backslash\widehat{\Omega}$ do wektora stanu $\widehat{\Omega}$, natomiast macierz $\mathbf{S}_H$ wprowadza informację z wektora stanu $\widehat{\Omega}$ do wektora stanu $\Omega\backslash\widehat{\Omega}$. Pełny układ równań algorytmu FDTD z wyodrębnioną poddziedziną $\widehat{\Omega}$ stanowią r. (5.3).

$$\mathbf{R}'_E\mathbf{e}' = -s\mathbf{D}'_\mu\mathbf{h}' \tag{5.3a}$$
$$\mathbf{h}_b + \mathbf{R}'_H\mathbf{h}' = s\mathbf{D}'_\epsilon\mathbf{e}' \tag{5.3b}$$
$$\widehat{\mathbf{e}}_b + \widehat{\mathbf{R}}_E\widehat{\mathbf{e}} = -s\widehat{\mathbf{D}}_\mu\widehat{\mathbf{h}} \tag{5.3c}$$
$$\widehat{\mathbf{R}}_H\widehat{\mathbf{h}} = s\widehat{\mathbf{D}}_\epsilon\widehat{\mathbf{e}} \tag{5.3d}$$
$$\widehat{\mathbf{e}}_b = \widehat{\mathbf{S}}_E\mathbf{e}' \tag{5.3e}$$
$$\mathbf{h}_b = \mathbf{S}_H\mathbf{h}' \tag{5.3f}$$

Definicję macierzy sprzęgających $\widehat{\mathbf{S}}_E$ oraz $\mathbf{S}_H$ prezentują równania (5.4) oraz (5.5).

$$\widehat{\mathbf{e}}_b = \widehat{\mathbf{B}}_E\mathbf{I}_E\mathbf{L}_E\mathbf{e}' = \widehat{\mathbf{S}}_E\mathbf{e}' \tag{5.4}$$
$$\mathbf{h}_b = \mathbf{B}_H\mathbf{I}_H\widehat{\mathbf{L}}_H\widehat{\mathbf{h}} = \mathbf{S}_H\widehat{\mathbf{h}} \tag{5.5}$$

Zgodnie z r. (5.4) macierz $\widehat{\mathbf{S}}_E$ jest złożeniem trzech działań:

1. Pobranie wartości próbek pola elektrycznego z dziedziny $\Omega\backslash\widehat{\Omega}$, które stanowią pobudzenie poddziedziny $\widehat{\Omega}$. Działanie to jest reprezentowane przez macierz wybierającą $\mathbf{L}_E$ w r. (5.4).
2. Jeśli siatki dyskretyzacji $\Omega\backslash\widehat{\Omega}$ i $\widehat{\Omega}$ posiadają różny krok dyskretyzacji, to realizowana jest interpolacja pól pomiędzy dziedziną $\Omega\backslash\widehat{\Omega}$ i poddziedziną $\widehat{\Omega}$. Działanie to jest reprezentowane przez macierz interpolującą $\mathbf{I}_E$ w r. (5.4). Jeśli krok dyskretyzacji jest taki sam dla $\Omega\backslash\widehat{\Omega}$ i $\widehat{\Omega}$, to macierz interpolacji jest macierzą jednostkową.
3. Wprowadzenie pobudzenia do odpowiednich elementów wektora stanu poddziedziny $\widehat{\Omega}$. Działanie to jest reprezentowane przez macierz brzegową $\widehat{\mathbf{B}}_E$ w r. (5.4).

Macierz $\mathbf{S}_H$ również jest złożeniem trzech działań, zgodnie z r. (5.5):

1. Pobranie wartości próbek pola magnetycznego, które stanowią odpowiedź poddziedziny $\widehat{\Omega}$. Działanie to jest reprezentowane przez macierz wybierającą $\widehat{\mathbf{L}}_H$ w r. (5.5).
2. Interpolacja pól (jeśli jest wymagana) realizowana jest przez macierz $\mathbf{I}_H$ w r. (5.5).
3. Wprowadzenie odpowiedzi do odpowiednich elementów wektora stanu dziedziny $\Omega\backslash\widehat{\Omega}$. Działanie to jest reprezentowane przez macierz brzegową $\mathbf{B}_H$ w r. (5.5).

Dokładny opis konstrukcji macierzy sprzęgających oraz ich własności można znaleźć w pracach [43, 44, 49, 54].

Układ równań (5.3) można zapisać w formie macierzowej przedstawionej w r. (5.6). Zapis macierzowy pozwala na zastosowanie lokalnego zagęszczenia siatki zarówno w analizie czasowej FDTD jak i w analizie częstotliwościowej FDFD.

W dalszej części pracy stopień zmniejszenia kroku dyskretyzacji oznaczono przez M, np. jeśli $M = 3$, to znaczy, że krok dyskretyzacji w poddziedzinie, w każdym kierunku przestrzeni, został zmniejszony trzykrotnie.

$$\begin{bmatrix} \mathbf{R}'_E & \mathbf{0} \\ \widehat{\mathbf{S}}_E & \widehat{\mathbf{R}}_E \end{bmatrix} \begin{bmatrix} \mathbf{e}' \\ \widehat{\mathbf{e}} \end{bmatrix} = -s \begin{bmatrix} \mathbf{D}'_\mu & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{D}}_\mu \end{bmatrix} \begin{bmatrix} \mathbf{h}' \\ \widehat{\mathbf{h}} \end{bmatrix} \tag{5.6a}$$

$$\begin{bmatrix} \mathbf{R}'_H & \mathbf{S}_H \\ \mathbf{0} & \widehat{\mathbf{R}}_H \end{bmatrix} \begin{bmatrix} \mathbf{h}' \\ \widehat{\mathbf{h}} \end{bmatrix} = s \begin{bmatrix} \mathbf{D}'_\epsilon & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{D}}_\epsilon \end{bmatrix} \begin{bmatrix} \mathbf{e}' \\ \widehat{\mathbf{e}} \end{bmatrix} \tag{5.6b}$$

5.2 Pomiary efektywności obliczeń

Ocena efektywności algorytmu FDTD zawierającego lokalne zagęszczenie siatki wymaga wyznaczenia liczby komórek Yee, których wartości próbek pól są aktualizowane w każdej iteracji algorytmu FDTD. Liczba ta, oznaczona symbolem N_{SG}, równa jest sumie komórek Yee zastosowanych w siatce podstawowej N' oraz komórek Yee z R poddziedzin z siatką zagęszczoną, zgodnie z r. (5.8).

$$N' = N - \sum_{i=1}^{R} N_{P\,i} \tag{5.7}$$

$$N_{SG} = N' + \sum_{i=1}^{R} N_{D\,i} \tag{5.8}$$

gdzie:
N' - liczba komórek Yee z siatki podstawowej, które nie zostały zmodyfikowane przez wprowadzenie zagęszczenia siatki dyskretyzacji,
$N_{P\,i}$ - liczba komórek Yee z siatki podstawowej, która została zastąpiona przez i-tą poddziedzinę,
$N_{D\,i}$ - liczba komórek Yee występująca w i-tej poddziedzinie,
N_{SG} - liczba komórek Yee przetwarzana w algorytmie FDTD z lokalnym zagęszczeniem siatki.

Tak określona liczba komórek Yee umożliwia zdefiniowanie dla schematu różnicowego zawierającego lokalne zagęszczenie siatki parametru S_C, który pozwala ocenić efektywność obliczeń badanej implementacji tego schematu (porównaj p. 3.2.1), zgodnie z r. (5.9).

$$S_C = \frac{N_{SG} n_{it}}{t_{sim}} \tag{5.9}$$

gdzie:
t_{sim} - czas symulacji,
n_{it} - liczba przeprowadzonych iteracji schematu różnicowego.

Jednak w ocenie efektywności obliczeń można również uwzględnić wpływ obliczeń związanych z wymianą informacji pomiędzy dziedziną a poddziedzinami, którą realizują macierze $\mathbf{S}_H$ oraz $\widehat{\mathbf{S}}_E$, r. (5.6). Liczba operacji, które realizują te macierze może być różna w zależności od zastosowanego algorytmu, [53, 55, 56]. W [56] przejście z siatki rzadkiej

do siatki gęstej jest realizowane poprzez ekstrapolację wartości jednej próbki pola z siatki rzadkiej do $M(M-1)$ próbek pól z siatki gęstej. Przejście z siatki gęstej do rzadkiej polega na analogicznej interpolacji próbek pól. W takiej sytuacji liczba operacji realizowanych przy mnożeniu wektora $\mathbf{e}'$ przez macierz $\widehat{\mathbf{S}}_E$ przy wprowadzeniu do schematu różnicowego R poddziedzin określa r. (5.11). Liczba operacji stanowi podwojoną liczbę elementów niezerowych macierzy $\widehat{\mathbf{S}}_E$, gdyż dla każdego z nich wymagana jest jedna operacji dodawania oraz mnożenia. W zależności (5.11) założono, że poddziedzina nie jest styczna z krańcem dziedziny obliczeniowej.

$$L_{Se} = \sum_{i=0}^{R} L_{Se\,p} \tag{5.10}$$

$$\begin{aligned} L_{Se\,p} = {} & 2(2M^2 - 2M)\,(r_{p\,i}r_{p\,j} + r_{p\,j}r_{p\,k}) + \\ & + 2(2M^2 - M)\,(3r_{p\,i}r_{p\,j} + 4r_{p\,i}r_{p\,k} + 3r_{p\,j}r_{p\,k}) \end{aligned} \tag{5.11}$$

gdzie:
$r_{p\,i}$, $r_{p\,j}$, $r_{p\,k}$ - rozmiar p-tej poddziedziny, odpowiednio w kierunkach x, y, z, podany w liczbie komórek Yee siatki rzadkiej.

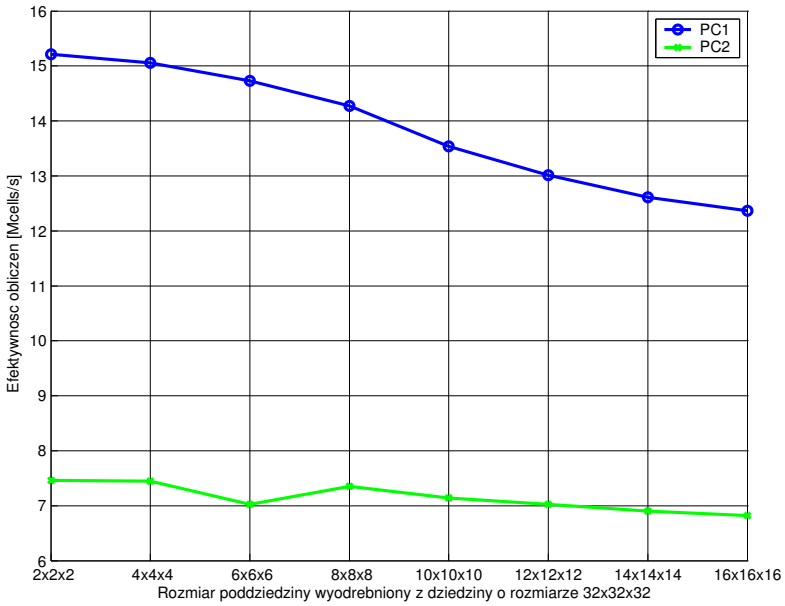
W związku z tym całkowita liczba operacji zmiennoprzecinkowych wymagana przy aktualizacji pól w schemacie różnicowym z lokalnym zagęszczeniem siatki jest wyrażona w r. (5.12).

$$L_{F\,SG} = 54N_{SG} + L_{Se} + L_{Sh} \tag{5.12}$$

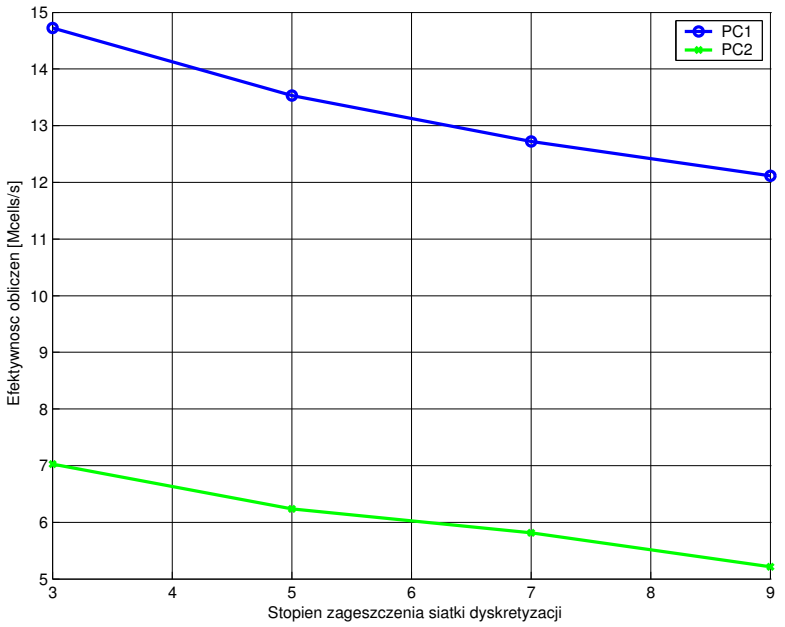
Przy tak zdefiniowanych parametrach służących do oceny efektywności obliczeń, przystąpiono do pomiarów. Rys. 5.2 przedstawia wpływ rozmiaru poddziedziny z zagęszczoną siatką na efektywność obliczeń. Rozmiar dziedziny obliczeniowej ustalono na $32 \times 32 \times 32$. Można zauważyć, że wraz ze wzrostem rozmiaru poddziedziny efektywność nieznacznie maleje. Jednak przy wzroście rozmiaru problemu w algorytmie FDTD bez modyfikacji efektywność obliczeń również maleje (rys. 4.1). Porównując rys. 4.1 ($log_2(32^3) = 15$) oraz rys. 5.2 można stwierdzić, że efektywność obliczeń zależy przede wszystkim od liczby zastosowanych w analizie komórek Yee i wprowadzenie przejścia pomiędzy siatką gęstą a rzadką związane jest z niskim kosztem numerycznym. Teza ta znajduje również swoje potwierdzenie przy zastosowaniu różnego stopnia zagęszczenia siatki dyskretyzacji, co wynika z wykresu 5.3, który prezentuje rezultaty pomiaru przeprowadzone dla dziedziny obliczeniowej o rozmiarze $32 \times 32 \times 32$ oraz poddziedziny o rozmiarze $6 \times 6 \times 6$.

Analogiczne pomiary wykonano dla akceleratora graficznego GTX 580. Na podstawie rys. 5.4 można stwierdzić, iż rozmiar poddziedziny obliczeniowej przy zagęszczeniu siatki $M = 3$ ma istotny wpływ na efektywność obliczeń (wartość minimalna efektywności jest o 13% mniejsza od wartości maksymalnej). Ponadto z porównania rys. 5.4 i rys. 4.2 wynika, że w odniesieniu do obliczeń przeprowadzonych przy pomocy algorytmu FDTD bez modyfikacji w formie macierzowej, koszt wprowadzenia lokalnego zagęszczenia siatki dla $M = 3$ powoduje spadek efektywności obliczeń w zakresie od 14% do 26%, przy czym wzrasta on wraz z rozmiarem poddziedziny. Wynika stąd, że wzrost liczby wierszy w macierzy rotacji, w których liczba elementów niezerowych jest większa od czterech ma negatywny wpływ na efektywność obliczeń.

Podobnie wygląda wpływ zagęszczenia siatki dyskretyzacji na efektywność obliczeń dla akceleratorów graficznych, co przedstawia rys. 5.5 (rozmiar dziedziny obliczeniowej wynosił $32 \times 32 \times 32$, a poddziedziny $6 \times 6 \times 6$). Przy większym stopniu zagęszczenia siatki

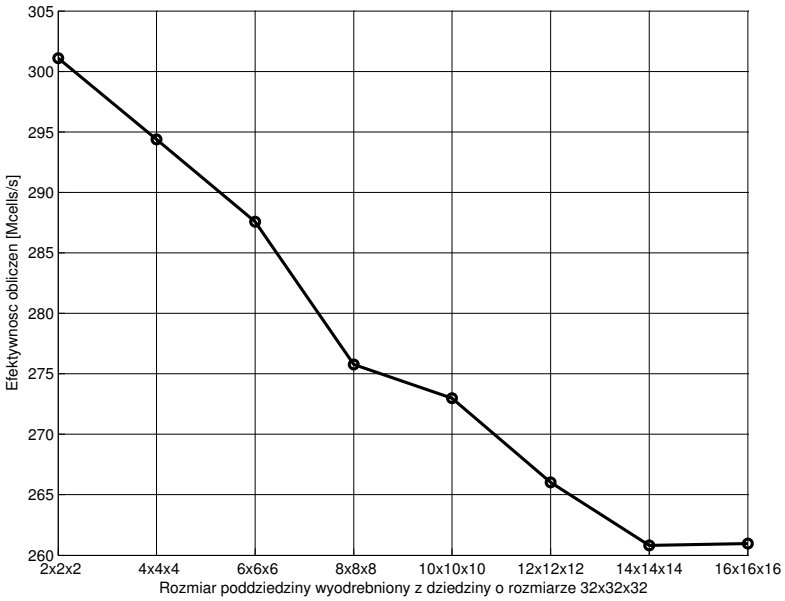


Rysunek 5.2: Wpływ rozmiaru obszaru z zagęszczoną siatką dyskretyzacji na efektywność implementacji algorytmu FDTD w formie macierzowej przeznaczonej dla CPU

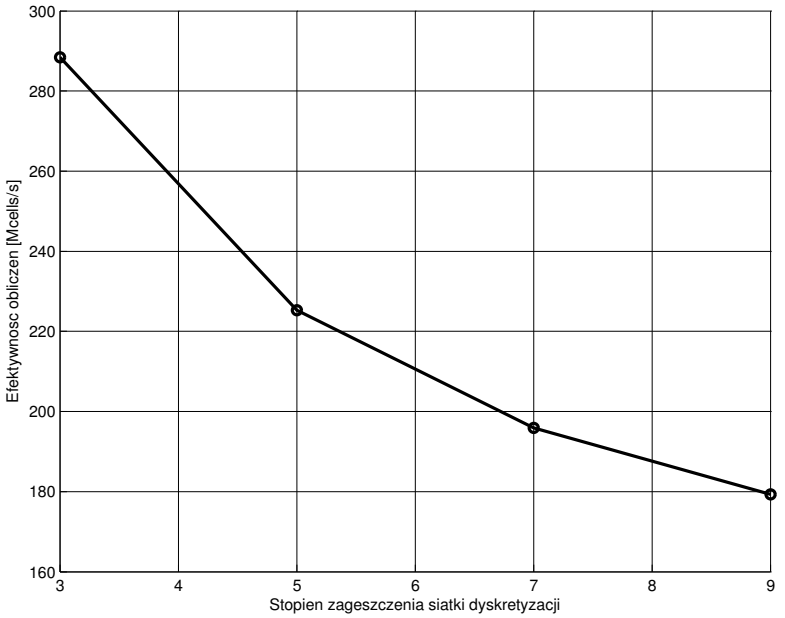


Rysunek 5.3: Wpływ wzrostu stopnia zagęszczenia siatki dyskretyzacji na efektywność implementacji algorytmu FDTD w formie macierzowej przeznaczonej dla CPU

macierze $\widehat{\mathbf{S}}_E$ oraz $\mathbf{S}_H$ zawierają znaczną liczbę elementów niezerowych w wierszu, przez co algorytm mnożenia macierzy rzadkiej przez wektor z wydruku 4.3 jest nieefektywny. Wynika to z nierównomiernego rozłożenia obliczeń na poszczególne wątki. Oczywistym zatem rozwiązaniem wydaje się zastosowanie innego algorytmu mnożenia wektora przez macierz rzadką. Jednak w pracach [11, 12, 13] przedstawiono rozwiązania, które w głównej mierze opierają się na wyodrębnieniu w macierzy rzadkiej podmacierzy gęstych, które zawierają również elementy równe zero. O ile w wielu wykazanych sytuacjach jest to



Rysunek 5.4: Wpływ rozmiaru obszaru z zagęszczoną siatką dyskretyzacji na efektywność implementacji algorytmu FDTD w formie macierzowej przeznaczonej dla GPU. Pomiary wykonano na karcie GTX 580.



Rysunek 5.5: Wpływ wzrostu stopnia zagęszczenia siatki dyskretyzacji na efektywność implementacji algorytmu FDTD w formie macierzowej przeznaczonej dla GPU. Pomiary wykonano na karcie GTX 580.

rozwiązanie efektywne, to jednak nie jest ono dostosowane do analizowanego algorytmu różnicowego z lokalnym zagęszczeniem siatki. Niska efektywność ma swoje źródło w postaci macierzy z r. (5.6), w których zdecydowana większość wierszy posiada posiada liczbę elementów niezerowych nie większą niż cztery. Tab. 5.1 i 5.2 przedstawiają liczby elementów niezerowych zawartych w macierzach rotacji zdefiniowanych dla problemu o rozmiarze

$32 \times 32 \times 32$ zawierającego poddziedzinę o rozmiarze $6 \times 6 \times 6$ i zagęszczeniu $M = 9$. Taki rozkład elementów niezerowych znacznie utrudnia realizację implementacji mnożenia macierzy rzadkiej przez wektor, która charakteryzowałaby się wysoką efektywnością. Najskuteczniejszym rozwiązaniem w zaistniałej sytuacji okazało się rozbicie zmodyfikowanych macierzy rotacji na grupę podmacierzy, w których można wyróżnić macierze gęste oraz rzadkie o liczbie elementów niezerowych nie większej niż cztery, co zaprezentowano w rozdziale 6.

Tablica 5.1: Parametry macierzy rotacji pola magnetycznego dla dziedziny zawierającej lokalne zagęszczenie siatki

Liczba elementów niezerowych w wierszu	Liczba wierszy z podaną liczbą elementów niezerowych	Procentowy udział wierszy z podaną liczbą elementów niezerowych
4	537876	99,92
148	72	0,01
156	360	0,07

Tablica 5.2: Parametry macierzy rotacji pola elektrycznego dla dziedziny zawierającej lokalne zagęszczenie siatki

Liczba elementów niezerowych w wierszu	Liczba wierszy z podaną liczbą elementów niezerowych	Procentowy udział wierszy z podaną liczbą elementów niezerowych
2	360	0,07
3	10440	1,90
4	508266	92,48
5	29952	5,45
6	576	0,10

5.3 Podsumowanie

Zysk z zastosowania lokalnego zagęszczenia siatki dyskretyzacji jest znaczny w porównaniu do analizy przeprowadzonej z dziedziną obliczeniową o kroku dyskretyzacji równym najmniejszemu krokowi dyskretyzacji występującym w poddziedzinie.

Przedstawione rezultaty pomiarów dowodzą, że efektywność obliczeń algorytmu FDTD z lokalnym zagęszczeniem siatki jest proporcjonalna do liczby komórek Yee N_{SG}. Zatem zysk z zastosowania lokalnego zagęszczenia siatki wynika ze zmniejszenia liczby komórek Yee, co wyraża r. (5.13).

$$Z_{SG} = \frac{N_{SG}}{N_d} \tag{5.13}$$

gdzie:
N_d - liczba komórek Yee schematu różnicowego z siatką gęstą, przy czym krok dyskretyzacji w tym schemacie jest zmniejszony k-krotnie w porównaniu do siatki podstawowej

zastosowanej w algorytmie FDTD z lokalnym zagęszczeniem siatki dyskretyzacji, gdzie k oznacza największy stopień zagęszczenia siatki zastosowany w dowolnej poddziedzinie obliczeniowej,
Z_{SG} - stopień zwiększenia efektywności obliczeń dla schematu różnicowego z lokalnym zagęszczeniem siatki dyskretyzacji w porównaniu do schematu różnicowego z siatką gęstą.

Przykładowo, dla lokalnego zagęszczenia siatki określonego w 5% objętości dziedziny obliczeniowej, przy zagęszczeniu siatki równym 7, zwiększenie efektywności obliczeń, równoznaczne ze skróceniem czasu analizy, wynosi w przybliżeniu 5,4.

Porównanie efektywności algorytmu z lokalnym zagęszczeniem siatki dyskretyzacji w odniesieniu do efektywności obliczeń algorytmu w postaci jawnej zostało przedstawione również w bardziej rozbudowanym przykładzie opisanym w rozdziale 7.

ROZDZIAŁ 6

Algorytm FDTD z makromodelami

Lokalne zagęszczenie statki dyskretyzacji, przedstawione w rozdziale 5, prowadzi do zwiększenia efektywności obliczeń poprzez dokładniejsze odwzorowanie pola tylko w wybranych podobszarach dziedziny obliczeniowej. Dalsze zwiększenie efektywności obliczeń można osiągnąć poprzez zastosowanie makromodeli [43], które umożliwiają redukcję liczby zmiennych wyodrębnionej poddziedziny obliczeniowej. W tym rozdziale zostanie przedstawiona ocena kosztów numerycznych działania algorytmu FDTD zawierającego makromodele oraz wskazana zostanie metoda optymalizacji tego algorytmu dla struktur zawierających powtarzające się elementy.

6.1 Konstrukcja makromodeli

Redukcja liczby zmiennych stanu jest możliwa do przeprowadzenia dla odpowiednio zdefiniowanej funkcji przejścia, która reprezentuje relację pomiędzy polami $\mathbf{e}_M$ (r. (6.1)) pobudzającymi $\widehat{\Omega}$ oraz polami $\mathbf{h}_M$ (r. (6.2)), które stanowią odpowiedź $\widehat{\Omega}$ na pobudzenie.

$$\mathbf{e}_M = \mathbf{L}_E \mathbf{e}' \tag{6.1}$$

$$\mathbf{h}_M = \mathbf{I}_H \widehat{\mathbf{L}}_H \widehat{\mathbf{h}} \tag{6.2}$$

Wstępną relację pomiędzy próbkami pól $\mathbf{e}_M$ oraz $\mathbf{h}_M$ reprezentuje r. (6.3), którego postać wynika z uwzględnienia r. (5.3d) oraz r. (5.4) w r. (5.3c).

$$\frac{1}{s}\widehat{\mathbf{R}}_E \widehat{\mathbf{D}}_\epsilon^{-1} \widehat{\mathbf{R}}_H \widehat{\mathbf{h}} + s\widehat{\mathbf{D}}_\mu \widehat{\mathbf{h}} = -\widehat{\mathbf{B}}_E \mathbf{I}_E \mathbf{e}_M \tag{6.3}$$

Na podstawie znajomości r. (6.3) oraz r. (6.2) zdefiniowano funkcję przejścia $\mathbf{H}(s)$ w r. (6.4).

$$\mathbf{h}_M = \mathbf{L}^T \left(\frac{1}{s}\hat{\mathbf{\Gamma}} + s\hat{\mathbf{C}} \right)^{-1} \mathbf{B}\mathbf{e}_M = \mathbf{H}(s)\mathbf{e}_M \tag{6.4}$$

$$\mathbf{L}^T = \mathbf{I}_H \hat{\mathbf{L}}_H \tag{6.5}$$

$$\hat{\mathbf{\Gamma}} = \widehat{\mathbf{R}}_E \widehat{\mathbf{D}}_\epsilon^{-1} \widehat{\mathbf{R}}_H \tag{6.6}$$

$$\hat{\mathbf{C}} = \widehat{\mathbf{D}}_\mu \tag{6.7}$$

$$\mathbf{B} = -\hat{\mathbf{B}}_E \mathbf{I}_E \tag{6.8}$$

Rezultatem działania algorytmu redukcji rzędu modelu jest macierz projekcji $\widehat{\mathbf{V}}$, która po odpowiednim wymnożeniu przez macierz $\mathbf{H}(s)$ zmniejsza jej rozmiar przy zachowaniu własności częstotliwościowych reprezentowanej przez nią relacji w zadanym zakresie częstotliwości. Wpływ redukcji na macierze przedstawiają r. (6.9a) ÷ (6.9e). Rezultatem zastosowania redukcji rzędu modelu jest zmniejszenie wymiaru wektora $\hat{\mathbf{h}}$ z $3N_D$ do m, zgodnie z wymiarem macierzy projekcji $\widehat{\mathbf{V}}$, $3N_D \times m$ (N_D oznacza liczbę komórek Yee zawartych w poddziedzinie obliczeniowej).

W algorytmach przedstawionych w tej rozprawie do budowy macierzy projekcji $\widehat{\mathbf{V}}$ zastosowano algorytm ENOR (ang. *Efficient Nodal Order Reduction*), [94].

$$\mathbf{H}_m(s) = \mathbf{L}_m^T \left(\frac{1}{s}\hat{\mathbf{\Gamma}}_m + s\hat{\mathbf{C}}_m \right)^{-1} \mathbf{B}_m \tag{6.9a}$$

$$\mathbf{L}_m = \widehat{\mathbf{V}}^T \mathbf{L} \tag{6.9b}$$

$$\hat{\mathbf{\Gamma}}_m = \widehat{\mathbf{V}}^T \hat{\mathbf{\Gamma}} \widehat{\mathbf{V}} \tag{6.9c}$$

$$\hat{\mathbf{C}}_m = \widehat{\mathbf{V}}^T \hat{\mathbf{C}} \widehat{\mathbf{V}} \tag{6.9d}$$

$$\mathbf{B}_m = \widehat{\mathbf{V}}^T \mathbf{B} \tag{6.9e}$$

$$\hat{\mathbf{h}}_m = \widehat{\mathbf{V}}^T \hat{\mathbf{h}} \tag{6.9f}$$

Zbudowana macierz projekcji $\widehat{\mathbf{V}}$ może zostać zastosowana w macierzowym sformułowaniu FDTD z wyodrębnioną dziedziną obliczeniową, czyli w r. (5.6). Efekt takiej modyfikacji przedstawiają r. (6.10). Taka forma zapisu jest przydatna do analizy częstotliwościowej i również jest poprawna dla analizy czasowej FDTD. Jednak dużo bardziej efektywną wersję algorytmu FDTD z makromodelami przedstawiono w p. 6.3.

$$\begin{bmatrix} \mathbf{R}'_E & \mathbf{0} \\ \widehat{\mathbf{V}}^T \hat{\mathbf{S}}_E & \widehat{\mathbf{V}}^T \widehat{\mathbf{R}}_E \end{bmatrix} \begin{bmatrix} \mathbf{e}' \\ \hat{\mathbf{e}} \end{bmatrix} = -s \begin{bmatrix} \mathbf{D}'_\mu & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{V}}^T \widehat{\mathbf{D}}_\mu \widehat{\mathbf{V}} \end{bmatrix} \begin{bmatrix} \mathbf{h}' \\ \hat{\mathbf{h}}_m \end{bmatrix} \tag{6.10a}$$

$$\begin{bmatrix} \mathbf{R}'_H & \mathbf{S}_H \widehat{\mathbf{V}} \\ \mathbf{0} & \widehat{\mathbf{R}}_H \widehat{\mathbf{V}} \end{bmatrix} \begin{bmatrix} \mathbf{h}' \\ \hat{\mathbf{h}}_m \end{bmatrix} = s \begin{bmatrix} \mathbf{D}'_\epsilon & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{D}}_\epsilon \end{bmatrix} \begin{bmatrix} \mathbf{e}' \\ \hat{\mathbf{e}} \end{bmatrix} \tag{6.10b}$$

6.2 Makromodele symetryzowane

Rezultatem projekcji macierzy diagonalnej $\widehat{\mathbf{D}}_\mu$, wykonanej przy pomocy macierzy $\widehat{\mathbf{V}}$, jest macierz gęsta. Jest to oczywista wada tej wersji włączenia makromodelu do schematu

różnicowego, gdyż obliczenia związane z macierzami gęstymi są znacznie kosztowniejsze od obliczeń wykonanych na macierzach diagonalnych. Optymalizacja podstawowej wersji algorytmu FDTD z makromodelami wyrażonej przez r. (6.10), której celem jest uniknięcie powstania w tym miejscu algorytmu macierzy gęstej, sprowadza się do przeprowadzenia symetryzacji makromodelu.

Symetryzacja algorytmu FDTD zawierającego lokalne zagęszczenie siatki dyskretyzacji, która równocześnie stanowi jeden z warunków stabilności tego algorytmu, wymaga spełnienia zależności (6.11), [51, 53].

$$\tilde{\hat{\mathbf{S}}}_E = \tilde{\mathbf{S}}_H^T \tag{6.11}$$

Symetryzacja makromodelu jest zaprezentowana w r. (6.12).

$$\tilde{\hat{\mathbf{h}}} = \widehat{\mathbf{D}}_\mu^{\frac{1}{2}} \hat{\mathbf{h}} \tag{6.12a}$$

$$\tilde{\hat{\mathbf{\Gamma}}} = \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \widehat{\mathbf{R}}_E \widehat{\mathbf{D}}_\epsilon^{-1} \widehat{\mathbf{R}}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} = \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \hat{\mathbf{\Gamma}} \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \tag{6.12b}$$

$$\tilde{\hat{\mathbf{C}}} = \hat{\mathbf{I}} = \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \hat{\mathbf{C}} \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \tag{6.12c}$$

$$\tilde{\mathbf{L}}^T = \mathbf{I}_H \hat{\mathbf{L}}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} = \mathbf{L}^T \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \tag{6.12d}$$

$$\tilde{\mathbf{B}} = -\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \hat{\mathbf{B}}_E \mathbf{I}_E = -\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \mathbf{B} \tag{6.12e}$$

Symetryzowana funkcja przejścia ma zatem postać r. (6.13). Dowód symetryczności $\tilde{\mathbf{H}}(s)$ znajduje się w punkcie B.1.

$$\tilde{\mathbf{H}}(s) = \tilde{\mathbf{L}}^T \left(\frac{1}{s} \tilde{\hat{\mathbf{\Gamma}}} + s \hat{\mathbf{I}} \right)^{-1} \tilde{\mathbf{B}} \tag{6.13}$$

Macierz projekcji $\tilde{\hat{\mathbf{V}}}$ otrzymana dla r. (6.13) prowadzi do zdefiniowania algorytmu FDTD z makromodelami w postaci r. (6.14).

$$\begin{bmatrix} \mathbf{R}'_E & \mathbf{0} \\ \tilde{\hat{\mathbf{V}}}^T \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \hat{\mathbf{S}}_E & \tilde{\hat{\mathbf{V}}}^T \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \widehat{\mathbf{R}}_E \end{bmatrix} \begin{bmatrix} \mathbf{e}' \\ \hat{\mathbf{e}} \end{bmatrix} = -s \begin{bmatrix} \mathbf{D}'_\mu & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{I}}_m \end{bmatrix} \begin{bmatrix} \mathbf{h}' \\ \tilde{\hat{\mathbf{h}}}_m \end{bmatrix} \tag{6.14a}$$

$$\begin{bmatrix} \mathbf{R}'_H & \mathbf{S}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \tilde{\hat{\mathbf{V}}} \\ \mathbf{0} & \widehat{\mathbf{R}}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \tilde{\hat{\mathbf{V}}} \end{bmatrix} \begin{bmatrix} \mathbf{h}' \\ \tilde{\hat{\mathbf{h}}}_m \end{bmatrix} = s \begin{bmatrix} \mathbf{D}'_\epsilon & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{D}}_\epsilon \end{bmatrix} \begin{bmatrix} \mathbf{e}' \\ \hat{\mathbf{e}} \end{bmatrix} \tag{6.14b}$$

przy czym:

$$\tilde{\hat{\mathbf{h}}}_m = \tilde{\hat{\mathbf{V}}}^T \tilde{\hat{\mathbf{h}}} \tag{6.15}$$

Drugim istotnym powodem zastosowania procesu symetryzacji makromodelu jest zapewnienie stabilności algorytmu FDTD z makromodelami, które wymaga również symetryzacji macierzy reprezentujących dziedzinę $\Omega \backslash \widehat{\Omega}$. Dodatkowe przekształcenia wynikają z warunku stabilności opisanego w p. A.2, który wymaga zachowania relacji $\mathbf{R}_E = \mathbf{R}_H^T$

pomiędzy macierzami rotacji pola elektrycznego i magnetycznego. W rezultacie symetryczny algorytm FDTD wraz z włączonym symetryzowanym makromodelem ma postać r. (6.16).

$$\begin{bmatrix} \tilde{\mathbf{R}}'_E & \mathbf{0} \\ \tilde{\widehat{\mathbf{V}}}^T \tilde{\widehat{\mathbf{S}}}_E & \tilde{\widehat{\mathbf{V}}}^T \tilde{\widehat{\mathbf{R}}}_E \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{e}}' \\ \tilde{\widehat{\mathbf{e}}} \end{bmatrix} = -s \begin{bmatrix} \tilde{\mathbf{h}}' \\ \tilde{\widehat{\mathbf{h}}}_m \end{bmatrix} \tag{6.16a}$$

$$\begin{bmatrix} \tilde{\mathbf{R}}'_H & \tilde{\mathbf{S}}_H \tilde{\widehat{\mathbf{V}}} \\ \mathbf{0} & \tilde{\widehat{\mathbf{R}}}_H \tilde{\widehat{\mathbf{V}}} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{h}}' \\ \tilde{\widehat{\mathbf{h}}}_m \end{bmatrix} = s \begin{bmatrix} \tilde{\mathbf{e}}' \\ \tilde{\widehat{\mathbf{e}}} \end{bmatrix} \tag{6.16b}$$

przy czym zastosowanie projekcji wobec zsymetryzowanych macierzy opisujących dziedzinę $\Omega \backslash \widehat{\Omega}$ wyrażają r. (6.17),

$$\tilde{\mathbf{R}}'_E = \mathbf{P}_H^T \tilde{\mathbf{R}}_E \mathbf{P}_E \tag{6.17a}$$

$$\tilde{\mathbf{R}}'_H = \mathbf{P}_E^T \tilde{\mathbf{R}}_H \mathbf{P}_H \tag{6.17b}$$

$$\tilde{\mathbf{e}}' = \mathbf{P}_E^T \tilde{\mathbf{e}} = \mathbf{P}_E^T \mathbf{D}_\epsilon^{\frac{1}{2}} \mathbf{e} \tag{6.17c}$$

$$\tilde{\mathbf{h}}' = \mathbf{P}_H^T \tilde{\mathbf{h}} = \mathbf{P}_H^T \mathbf{D}_\mu^{\frac{1}{2}} \mathbf{h} \tag{6.17d}$$

natomiast symetryzację macierzy i wektorów związanych z poddziedziną $\widehat{\Omega}$ wyrażają r. (6.18).

$$\tilde{\widehat{\mathbf{e}}} = \widehat{\mathbf{D}}_\epsilon^{\frac{1}{2}} \widehat{\mathbf{e}} \tag{6.18a}$$

$$\tilde{\widehat{\mathbf{R}}}_E = \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \tilde{\mathbf{R}}_E \widehat{\mathbf{D}}_\epsilon^{-\frac{1}{2}} \tag{6.18b}$$

$$\tilde{\widehat{\mathbf{R}}}_H = \widehat{\mathbf{D}}_\epsilon^{-\frac{1}{2}} \tilde{\mathbf{R}}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \tag{6.18c}$$

$$\tilde{\widehat{\mathbf{S}}}_E = \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \widehat{\mathbf{S}}_E \mathbf{D}_\epsilon^{-\frac{1}{2}} \tag{6.18d}$$

$$\tilde{\mathbf{S}}_H = \mathbf{D}_\epsilon^{-\frac{1}{2}} \mathbf{S}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \tag{6.18e}$$

Zachowanie relacji $\mathbf{R}_E = \mathbf{R}_H^T$ pomiędzy macierzami rotacji pola elektrycznego i magnetycznego zostało wykazane w r. (6.19), przy wykorzystaniu r. (6.11) oraz r. (2.19).

$$\begin{bmatrix} \tilde{\mathbf{R}}'_E & \mathbf{0} \\ \tilde{\widehat{\mathbf{V}}}^T \tilde{\widehat{\mathbf{S}}}_E & \tilde{\widehat{\mathbf{V}}}^T \tilde{\widehat{\mathbf{R}}}_E \end{bmatrix}^T = \begin{bmatrix} \tilde{\mathbf{R}}'^T_E & \tilde{\widehat{\mathbf{S}}}_E^T \tilde{\widehat{\mathbf{V}}} \\ \mathbf{0} & \tilde{\widehat{\mathbf{R}}}_E^T \tilde{\widehat{\mathbf{V}}} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{R}}'_H & \tilde{\mathbf{S}}_H \tilde{\widehat{\mathbf{V}}} \\ \mathbf{0} & \tilde{\widehat{\mathbf{R}}}_H \tilde{\widehat{\mathbf{V}}} \end{bmatrix} \tag{6.19}$$

W praktyce brak stabilności algorytmu FDTD z makromodelami bez zastosowanej symetryzacji był przez autora tej rozprawy wielokrotnie obserwowany. Z uwagi na to, w dalszej części rozprawy algorytmy macierzowe zostały poddane symetryzacji w celu zapewnienia stabilności algorytmu.

6.3 Algorytm FDTD z lokalnym zagęszczeniem siatki lub makromodelami

Algorytm FDTD z wyodrębnioną poddziedziną zapisany w postaci r. (5.6) stanowi zmodyfikowaną postać standardowego algorytmu FDTD w postaci macierzowej wyrażonej przez

r. (2.20). W rezultacie proces dyskretyzacji dziedziny czasu w r. (5.6) oraz w r. (2.20) przebiega analogicznie.

Jednak zsymetryzowany algorytm FDTD z makromodelami z r. (6.16) posiada istotną wadę: redukcja liczby zmiennych dotyczy jedynie wektora próbek pola magnetycznego, natomiast liczba zmiennych reprezentująca pole elektryczne jest taka sama jak przed zastosowaniem redukcji rzędu modelu. W związku z tym korzystne jest przeprowadzenie modyfikacji algorytmu, tak aby w obliczeniach wymagany był jedynie wektor próbek pola magnetycznego z zadanej poddziedziny. Zmodyfikowana wersja algorytmu została opisana w [49] oraz w rozprawie [43]. Postać tego algorytmu opisują r. (6.20).

$$\tilde{\mathbf{e}}'^{\tau} = \tilde{\mathbf{e}}'^{\tau-1} + \Delta_t \tilde{\mathbf{R}}'_H \mathbf{h}'^{\tau-0,5} + \Delta_t \tilde{\mathbf{B}}_H \mathbf{h}_M'^{\tau-0,5} \tag{6.20a}$$

$$\tilde{\mathbf{h}}'^{\tau+0,5} = \tilde{\mathbf{h}}'^{\tau-0,5} - \Delta_t \tilde{\mathbf{R}}'_E \tilde{\mathbf{e}}'^{\tau} \tag{6.20b}$$

$$\mathbf{e}_M^{\tau} = \tilde{\mathbf{L}}_E \tilde{\mathbf{e}}'^{\tau} \tag{6.20c}$$

$$\tilde{\tilde{\mathbf{h}}}_m^{\tau+0,5} = \left(2\mathbf{I} - \Delta_t^2 \tilde{\tilde{\mathbf{\Gamma}}}_m\right) \tilde{\tilde{\mathbf{h}}}_m^{\tau-0,5} - \tilde{\tilde{\mathbf{h}}}_m^{\tau-1,5} + \Delta_t \tilde{\mathbf{B}}_m \left(\mathbf{e}_M^{\tau} - \mathbf{e}_M^{\tau-1}\right) \tag{6.20d}$$

$$\mathbf{h}_M^{\tau+0,5} = \tilde{\mathbf{L}}_m^T \tilde{\tilde{\mathbf{h}}}_m^{\tau+0,5} \tag{6.20e}$$

Wprowadzone do r. (6.20) oznaczenia ($\tilde{\mathbf{B}}$, $\tilde{\tilde{\mathbf{\Gamma}}}$ oraz $\tilde{\mathbf{L}}$ zdefiniowano w r. (6.12)) reprezentują następujące relacje:

$$\tilde{\tilde{\mathbf{S}}}_E = \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \widehat{\mathbf{S}}_E \mathbf{D}_\epsilon^{-\frac{1}{2}} = \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \widehat{\mathbf{B}}_E \mathbf{I}_E \mathbf{L}_E \mathbf{D}_\epsilon^{-\frac{1}{2}} = \tilde{\mathbf{B}} \tilde{\mathbf{L}}_E \tag{6.21a}$$

$$\tilde{\mathbf{L}}_E = \mathbf{L}_E \mathbf{D}_\epsilon^{-\frac{1}{2}} \tag{6.21b}$$

$$\tilde{\mathbf{S}}_H = \mathbf{D}_\epsilon^{-\frac{1}{2}} \mathbf{S}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} = \mathbf{D}_\epsilon^{-\frac{1}{2}} \widehat{\mathbf{B}}_E \mathbf{I}_E \mathbf{L}_E \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} = \tilde{\mathbf{B}}_H \tilde{\mathbf{L}} \tag{6.21c}$$

$$\tilde{\mathbf{B}}_H = \mathbf{D}_\epsilon^{-\frac{1}{2}} \mathbf{B}_H \tag{6.21d}$$

$$\tilde{\mathbf{L}}_m = \widehat{\mathbf{V}}^T \tilde{\mathbf{L}} \tag{6.21e}$$

$$\tilde{\tilde{\mathbf{\Gamma}}}_m = \widehat{\mathbf{V}}^T \tilde{\tilde{\mathbf{\Gamma}}} \widehat{\mathbf{V}} \tag{6.21f}$$

$$\tilde{\mathbf{B}}_m = \widehat{\mathbf{V}}^T \tilde{\mathbf{B}} \tag{6.21g}$$

Dla makromodeli symetryzowanych długość wektorów $\mathbf{e}_M$ i $\mathbf{h}_M$ równa jest p, przy czym p jest określone dla pojedynczego makromodelu przez r. (6.23).

$$p = 4\left(r_i r_j + r_i r_k + r_j r_k\right) \tag{6.22}$$

$$m = pq \tag{6.23}$$

Wprowadzenie do obliczeń macierzy $\widehat{\mathbf{V}}$ o rozmiarze $3N_D \times m$ ustala rozmiar wektora $\tilde{\tilde{\mathbf{h}}}_m$ na wartość równą m. Równocześnie rozmiar macierzy $\tilde{\tilde{\mathbf{\Gamma}}}_m$ wynosi $m \times m$, a macierzy $\tilde{\mathbf{B}}_m$ i $\tilde{\mathbf{L}}_m$ $m \times p$.

Warto podkreślić, iż rozmiar macierzy tworzących makromodel nie zależy od stopnia zagęszczenia siatki lokalnej w poddziedzinie, którą makromodel opisuje, lecz jedynie od rzędu makromodelu oraz od rozmiaru poddziedziny wyrażonej w liczbie komórek siatki podstawowej.

Poniżej rozpisano poszczególne czynniki składające się na całkowity koszt numeryczny schematu różnicowego zawierającego pojedynczy makromodel (założono, że przed rozpoczęciem procesu iteracyjnego wykonano działania na macierzach, np. przemnożono macierz $\tilde{\mathbf{R}}'_H$ przez Δ_t):

1. koszt numeryczny części schematu różnicowego odpowiadającej za aktualizację wartości próbek pól w dziedzinie $\Omega\backslash\widehat{\Omega}$, która zawiera N' komórek siatki Yee, można wyznaczyć z r. (4.2),
2. macierze $\tilde{\mathbf{B}}_H$ i $\tilde{\mathbf{L}}_E$ posiadają p elementów niezerowych, zatem uwzględniając r. (6.20), ich obecność wymusza przeprowadzenie $3p$ działań,
3. r. (6.20d) i r. (6.20e) realizują $2m^2 + 2mp + 2m + p$ działań.

Z powyższych rozważań wynika, że całkowity koszt numeryczny algorytmu FDTD, który zawiera jeden makromodel wynosi L_{FM1}, zgodnie z r. (6.24).

$$L_{FM1} = 54N' + 2m^2 + 2mp + 2m + 4p \tag{6.24}$$

Przykład liczbowy obrazujący skalę redukcji liczby zmiennych podany jest w p. 6.8.

6.4 Diagonalizacja macierzy Gamma

Jedną z metod usprawniających działanie algorytmu FDTD z makromodelami jest diagonalizacja macierzy $\hat{\tilde{\mathbf{\Gamma}}}_m$. Została ona również zastosowana w algorytmach opisanych w tej rozprawie. Sprowadza się ona do wyznaczenia rozkładu własnego macierzy $\hat{\tilde{\mathbf{\Gamma}}}_m$, co opisuje r. (6.25), a następnie zdefiniowania nowej macierzy projekcji $\hat{\tilde{\mathbf{V}}}'$, zgodnie z r. (6.26).

$$\hat{\tilde{\mathbf{\Gamma}}}_m = \mathbf{W}\mathbf{\Lambda}\mathbf{W} \tag{6.25}$$

$$\hat{\tilde{\mathbf{V}}}' = \hat{\tilde{\mathbf{V}}}\mathbf{W} \tag{6.26}$$

$\mathbf{\Lambda}$ - macierz diagonalna zawierająca wartości własne macierzy $\hat{\tilde{\mathbf{\Gamma}}}_m$,
$\mathbf{W}$ - macierz wektorów własnych.

Zatem zastosowanie zmodyfikowanej macierzy projekcji $\hat{\tilde{\mathbf{V}}}'$ wobec macierzy $\hat{\mathbf{\Gamma}}$ prowadzi do otrzymania macierzy diagonalnej.

Konsekwencją wprowadzenia do obliczeń diagonalnej macierzy $\hat{\tilde{\mathbf{\Gamma}}}_m$ jest zmniejszenie liczby operacji zmiennoprzecinkowych oraz zapotrzebowania na zasoby pamięci komputera. Koszty numeryczne tak zmodyfikowanego algorytmu określa r. (6.27).

$$L_{FM2} = 54N' + 2mp + 4m + 4p \tag{6.27}$$

6.5 Wyodrębnienie wielu poddziedzin z dziedziny obliczeniowej

Technika wyodrębniania poddziedziny obliczeniowej $\widehat{\Omega}$ z dziedziny obliczeniowej Ω pozwala na zastosowanie w jednej analizie FDTD wielu obszarów z lokalnie zagęszczoną siatką dyskretyzacji oraz wielu makromodeli. Proces budowania macierzy wymaganych w podstawowej wersji takiej analizy, która zawiera υ makromodeli, przebiega w sposób analogiczny jak dla jednej poddziedziny:

1. Budowane są macierze projekcji $\mathbf{P}_E$ oraz $\mathbf{P}_H$, które wyłączają z dziedziny Ω wszystkie próbki pól należące do poddziedzin obliczeniowych $\widehat{\Omega}_i$. Następnie wyznaczane są macierze $\tilde{\mathbf{R}}'_E$, $\tilde{\mathbf{R}}'_H$ i wektory $\tilde{\mathbf{e}}'$, $\tilde{\mathbf{h}}'$, zgodnie z r. (6.17).
2. Zdefiniowane zostają macierze reprezentujące $\widehat{\Omega}_i$: $\tilde{\widehat{\mathbf{R}}}_{E\,i}$, $\tilde{\widehat{\mathbf{R}}}_{H\,i}$.
3. Budowane są macierze sprzęgające $\tilde{\widehat{\mathbf{S}}}_{E\,i}$, $\tilde{\mathbf{S}}_{H\,i}$ dla każdej poddziedziny $\widehat{\Omega}_i$, które wiążą ją z dziedziną obliczeniową $\Omega\backslash\widehat{\Omega}$.
4. Tworzony jest układ równań postaci (6.28).

$$\begin{bmatrix} \tilde{\mathbf{R}}'_E & \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \tilde{\widehat{\mathbf{V}}}_1^T \tilde{\widehat{\mathbf{S}}}_{E\,1} & \tilde{\widehat{\mathbf{V}}}_1^T \tilde{\widehat{\mathbf{R}}}_{E\,1} & \mathbf{0} & \dots & \mathbf{0} \\ \tilde{\widehat{\mathbf{V}}}_2^T \tilde{\widehat{\mathbf{S}}}_{E\,2} & \mathbf{0} & \tilde{\widehat{\mathbf{V}}}_2^T \tilde{\widehat{\mathbf{R}}}_{E\,2} & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \mathbf{0} \\ \tilde{\widehat{\mathbf{V}}}_\upsilon^T \tilde{\widehat{\mathbf{S}}}_{E\,\upsilon} & \mathbf{0} & \mathbf{0} & \dots & \tilde{\widehat{\mathbf{V}}}_\upsilon^T \tilde{\widehat{\mathbf{R}}}_{E\,\upsilon} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{e}}' \\ \tilde{\widehat{\mathbf{e}}}_1 \\ \tilde{\widehat{\mathbf{e}}}_2 \\ \vdots \\ \tilde{\widehat{\mathbf{e}}}_\upsilon \end{bmatrix} = -s \begin{bmatrix} \tilde{\mathbf{h}}' \\ \tilde{\widehat{\mathbf{h}}}_{m\,1} \\ \tilde{\widehat{\mathbf{h}}}_{m\,2} \\ \vdots \\ \tilde{\widehat{\mathbf{h}}}_{m\,\upsilon} \end{bmatrix} \tag{6.28a}$$

$$\begin{bmatrix} \tilde{\mathbf{R}}'_E & \tilde{\mathbf{S}}_{H\,1} \tilde{\widehat{\mathbf{V}}}_1 & \tilde{\mathbf{S}}_{H\,2} \tilde{\widehat{\mathbf{V}}}_2 & \dots & \tilde{\mathbf{S}}_{H\,\upsilon} \tilde{\widehat{\mathbf{V}}}_\upsilon \\ \mathbf{0} & \tilde{\widehat{\mathbf{R}}}_{H\,1} \tilde{\widehat{\mathbf{V}}}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \tilde{\widehat{\mathbf{R}}}_{H\,2} \tilde{\widehat{\mathbf{V}}}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots & \tilde{\widehat{\mathbf{R}}}_{H\,\upsilon} \tilde{\widehat{\mathbf{V}}}_\upsilon \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{h}}' \\ \tilde{\widehat{\mathbf{h}}}_{m\,1} \\ \tilde{\widehat{\mathbf{h}}}_{m\,2} \\ \vdots \\ \tilde{\widehat{\mathbf{h}}}_{m\,\upsilon} \end{bmatrix} = s \begin{bmatrix} \tilde{\mathbf{e}}' \\ \tilde{\widehat{\mathbf{e}}}_1 \\ \tilde{\widehat{\mathbf{e}}}_2 \\ \vdots \\ \tilde{\widehat{\mathbf{e}}}_\upsilon \end{bmatrix} \tag{6.28b}$$

Postać równań (6.28) można czytelniej wyrazić przy pomocy macierzy pomocniczych zdefiniowanych w r. (6.29).

$$\tilde{\widehat{\mathbf{e}}}_\Upsilon = \begin{bmatrix} \tilde{\widehat{\mathbf{e}}}_1 \\ \tilde{\widehat{\mathbf{e}}}_2 \\ \vdots \\ \tilde{\widehat{\mathbf{e}}}_\upsilon \end{bmatrix} \tag{6.29a}$$

$$\tilde{\widehat{\mathbf{h}}}_{m\,\Upsilon} = \begin{bmatrix} \tilde{\widehat{\mathbf{h}}}_{m\,1} \\ \tilde{\widehat{\mathbf{h}}}_{m\,2} \\ \vdots \\ \tilde{\widehat{\mathbf{h}}}_{m\,\upsilon} \end{bmatrix} \tag{6.29b}$$

$$\tilde{\widehat{\mathbf{V}}}_\Upsilon = \begin{bmatrix} \tilde{\widehat{\mathbf{V}}}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \tilde{\widehat{\mathbf{V}}}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \tilde{\widehat{\mathbf{V}}}_\upsilon \end{bmatrix} \tag{6.29c}$$

$$\tilde{\widehat{\mathbf{R}}}_{E\,\Upsilon} = \begin{bmatrix} \tilde{\widehat{\mathbf{R}}}_{E\,1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \tilde{\widehat{\mathbf{R}}}_{E\,2} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \tilde{\widehat{\mathbf{R}}}_{E\,\upsilon} \end{bmatrix} \tag{6.29d}$$

$$\widehat{\tilde{\mathbf{R}}}_{H\,\Upsilon} = \begin{bmatrix} \widehat{\tilde{\mathbf{R}}}_{H\,1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \widehat{\tilde{\mathbf{R}}}_{H\,2} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \widehat{\tilde{\mathbf{R}}}_{H\,\upsilon} \end{bmatrix} \tag{6.29e}$$

$$\tilde{\hat{\mathbf{S}}}_{E\,\Upsilon} = \begin{bmatrix} \tilde{\hat{\mathbf{S}}}_{E\,1} \\ \tilde{\hat{\mathbf{S}}}_{E\,2} \\ \vdots \\ \tilde{\hat{\mathbf{S}}}_{E\,\upsilon} \end{bmatrix} \tag{6.29f}$$

$$\tilde{\mathbf{S}}_{H\,\Upsilon} = \begin{bmatrix} \tilde{\mathbf{S}}_{H\,1} & \tilde{\mathbf{S}}_{H\,2} & \dots & \tilde{\mathbf{S}}_{H\,\upsilon} \end{bmatrix} \tag{6.29g}$$

Po uwzględnieniu macierzy pomocniczych z r. (6.29) w r. (6.28) algorytm FDTD z wieloma makromodelami przedstawiają r. (6.30).

$$\begin{bmatrix} \tilde{\mathbf{R}}'_E & \mathbf{0} \\ \tilde{\widehat{\mathbf{V}}}_{\Upsilon}^{T} \tilde{\hat{\mathbf{S}}}_{E\,\Upsilon} & \tilde{\widehat{\mathbf{V}}}_{\Upsilon}^{T} \widehat{\tilde{\mathbf{R}}}_{E\,\Upsilon} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{e}}' \\ \tilde{\hat{\mathbf{e}}}_{\Upsilon} \end{bmatrix} = -s \begin{bmatrix} \tilde{\mathbf{h}}' \\ \tilde{\hat{\mathbf{h}}}_{m\,\Upsilon} \end{bmatrix} \tag{6.30a}$$

$$\begin{bmatrix} \tilde{\mathbf{R}}'_H & \tilde{\mathbf{S}}_{H\,\Upsilon} \tilde{\hat{\mathbf{V}}}_{\Upsilon} \\ \mathbf{0} & \widehat{\tilde{\mathbf{R}}}_{H\,\Upsilon} \tilde{\hat{\mathbf{V}}}_{\Upsilon} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{h}}' \\ \tilde{\hat{\mathbf{h}}}_{m\,\Upsilon} \end{bmatrix} = s \begin{bmatrix} \tilde{\mathbf{e}}' \\ \tilde{\hat{\mathbf{e}}}_{\Upsilon} \end{bmatrix} \tag{6.30b}$$

Możliwe jest przeprowadzenie symulacji FDTD wyrażonej przez r. (6.30), jednak, jak zaznaczono w p. 6.3, większą efektywność obliczeń można osiągnąć poprzez modyfikację algorytmu do postaci analogicznej do r. (6.20). Wymagane przekształcenia dla algorytmu w wieloma makromodelami są analogiczne jak dla algorytmu z jednym makromodelem i prowadzą do zdefiniowania algorytmu FDTD w postaci r. (6.31),

$$\tilde{\mathbf{e}}'^{\tau} = \tilde{\mathbf{e}}'^{\tau-1} + \Delta_t \tilde{\mathbf{R}}'_H \mathbf{h}'^{\tau-0,5} + \Delta_t \tilde{\mathbf{B}}_{H\,\Upsilon} \mathbf{h}_M'^{\tau-0,5} \tag{6.31a}$$

$$\tilde{\mathbf{h}}'^{\tau+0,5} = \tilde{\mathbf{h}}'^{\tau-0,5} - \Delta_t \tilde{\mathbf{R}}'_E \tilde{\mathbf{e}}'^{\tau} \tag{6.31b}$$

$$\mathbf{e}_{M\,\Upsilon}^{\tau} = \Delta_t \tilde{\mathbf{B}}_{m\,\Upsilon} \tilde{\mathbf{L}}_{E\,\Upsilon} \tilde{\mathbf{e}}'^{\tau} \tag{6.31c}$$

$$\tilde{\hat{\mathbf{h}}}_{m\,\Upsilon}^{\tau+0,5} = \left(2\mathbf{I} - \Delta_t^2 \tilde{\hat{\mathbf{\Gamma}}}_{m\,\Upsilon}\right) \tilde{\hat{\mathbf{h}}}_{m\,\Upsilon}^{\tau-0,5} - \tilde{\hat{\mathbf{h}}}_{m\,\Upsilon}^{\tau-1,5} + \mathbf{e}_{M\,\Upsilon}^{\tau} - \mathbf{e}_{M\,\Upsilon}^{\tau-1} \tag{6.31d}$$

$$\mathbf{h}_{M\,\Upsilon}^{\tau+0,5} = \tilde{\mathbf{L}}_{m\,\Upsilon}^{T} \tilde{\hat{\mathbf{h}}}_{m\,\Upsilon}^{\tau+0,5} \tag{6.31e}$$

przy czym

$$\tilde{\hat{\mathbf{\Gamma}}}_{\Upsilon} = \begin{bmatrix} \tilde{\hat{\mathbf{\Gamma}}}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \tilde{\hat{\mathbf{\Gamma}}}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \tilde{\hat{\mathbf{\Gamma}}}_{\upsilon} \end{bmatrix} \tag{6.32a}$$

$$\tilde{\mathbf{B}}_{\Upsilon} = \begin{bmatrix} \tilde{\mathbf{B}}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{B}}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \tilde{\mathbf{B}}_{\upsilon} \end{bmatrix} \tag{6.32b}$$

$$\tilde{\mathbf{L}}_{\Upsilon} = \begin{bmatrix} \tilde{\mathbf{L}}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{L}}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \tilde{\mathbf{L}}_{\upsilon} \end{bmatrix} \tag{6.32c}$$

$$\tilde{\mathbf{L}}_{E\,\Upsilon} = \begin{bmatrix} \tilde{\mathbf{L}}_{E\,1}^T & \tilde{\mathbf{L}}_{E\,2}^T & \cdots & \tilde{\mathbf{L}}_{E\,\upsilon}^T \end{bmatrix}^T \tag{6.32d}$$

$$\tilde{\mathbf{B}}_{H\,\Upsilon} = \begin{bmatrix} \tilde{\mathbf{B}}_{H\,1} & \tilde{\mathbf{B}}_{H\,2} & \cdots & \tilde{\mathbf{B}}_{H\,\upsilon} \end{bmatrix} \tag{6.32e}$$

$$\tilde{\hat{\mathbf{\Gamma}}}_{m\,\Upsilon} = \tilde{\hat{\mathbf{V}}}_{\Upsilon}^T \tilde{\hat{\mathbf{\Gamma}}}_{\Upsilon} \tilde{\hat{\mathbf{V}}}_{\Upsilon} \tag{6.32f}$$

$$\tilde{\mathbf{B}}_{m\,\Upsilon} = \tilde{\hat{\mathbf{V}}}_{\Upsilon}^T \tilde{\mathbf{B}}_{\Upsilon} \tag{6.32g}$$

$$\tilde{\mathbf{L}}_{m\,\Upsilon} = \tilde{\hat{\mathbf{V}}}_{\Upsilon}^T \tilde{\mathbf{L}}_{\Upsilon} \tag{6.32h}$$

$$\tilde{\hat{\mathbf{S}}}_{E\,\Upsilon} = \tilde{\mathbf{B}}_{\Upsilon} \tilde{\mathbf{L}}_{E\,\Upsilon} \tag{6.32i}$$

$$\tilde{\mathbf{S}}_{H\,\Upsilon} = \tilde{\mathbf{B}}_{H\,\Upsilon} \tilde{\mathbf{L}}_{\Upsilon} \tag{6.32j}$$

Koszty numeryczne algorytmu FDTD z wieloma makromodelami mogą zostać określone podobnie jak dla algorytmu z pojedynczym makromodelem przez r. (6.24) lub r. (6.27), przy założeniu, że zmianie ulegnie definicja wartości p oraz m, zgodnie z r. (6.33) oraz że macierze $\tilde{\mathbf{L}}_{m\,\Upsilon}$ i $\tilde{\mathbf{B}}_{m\,\Upsilon}$ będą macierzami pełnymi.

$$p = \sum_{u=1}^{\upsilon} m_u \tag{6.33a}$$

$$m = \sum_{u=1}^{\upsilon} p_u q_u \tag{6.33b}$$

$$p_u = 4\left(r_{u\,i} r_{u\,j} + r_{u\,i} r_{u\,k} + r_{u\,j} r_{u\,k}\right) \tag{6.33c}$$

przy czym:
q_u - rząd u-tego makromodelu.

Jednak zwykle wyższą efektywność otrzymuje się, gdy $\tilde{\mathbf{L}}_{m\,\Upsilon}$ i $\tilde{\mathbf{B}}_{m\,\Upsilon}$ stanowią macierze rzadkie. Wówczas koszt numeryczny algorytmu FDTD zawierającego wiele makromodeli, z zastosowaną diagonalizacją macierzy $\tilde{\hat{\mathbf{\Gamma}}}_{m\,\Upsilon}$, wynosi $L_{F\,M3}$.

$$L_{F\,M3} = 54N' + 2\sum_{u=1}^{\upsilon} p_u^2 q_u + 4m + 4p \tag{6.34}$$

Zastąpienie przez makromodele obszarów z lokalnie zagęszczoną siatką dyskretyzacji znacznie zmniejsza liczbę zmiennych zawartych w schemacie różnicowym, jednak nie zmienia liczby komórek Yee, która w danym problemie jest reprezentowana. Z tego powodu przyjęto, że efektywność obliczeń, również dla schematu różnicowego zawierającego makromodele, jest wyznaczana wedle r. (5.9).

6.6 Klonowanie makromodeli

Włączenie makromodeli do schematu różnicowego pozwala osiągnąć wzrost efektywności obliczeń [47, 49, 50, 52, 53, 57, 76, 103]. Jednak przy zastosowaniu znacznej liczby makromodeli duży wpływ na efektywność obliczeń może mieć czas budowania makromodeli

oraz macierzy sprzęgających i w takiej sytuacji wskazana jest dalsza optymalizacja algorytmu FDTD z makromodelami. Szczególnie dobre cechy pozwalające na optymalizację tego algorytmu posiadają struktury zawierające powtarzające się elementy, [78].

Podstawową zaletą struktur z powtarzającymi się elementami, z punktu widzenia optymalizacji algorytmu, jest możliwość wyodrębnienia podobszarów przestrzeni $\widehat{\Omega}_i$ o takich samych własnościach tzn. obszarów, które charakteryzują się identyczną liczbą komórek siatki Yee w każdym kierunku przestrzeni oraz identycznymi parametrami materiałowymi każdej komórki siatki Yee. Ponieważ podobszary $\widehat{\Omega}_i$ o takich samych własnościach są reprezentowane przez identyczne macierze $\widehat{\widetilde{\mathbf{R}}}_{E\,i}$, $\widehat{\widetilde{\mathbf{R}}}_{H\,i}$, $\tilde{\mathbf{L}}_i$, $\tilde{\mathbf{B}}_i$ to również wektor projekcji $\widehat{\widetilde{\mathbf{V}}}_i$ dla każdego z tych obszarów będzie taki sam, więc dla zbioru poddziedzin $\widehat{\Omega}_i$ o takich samych własnościach wymagane będzie zbudowanie tylko jednego makromodelu, który następnie będzie kopiowany dla każdej poddziedziny z tego zbioru. Proces kopiowania makromodeli został określony jako klonowanie makromodeli, [78]. Jedynymi macierzami, które należy zbudować indywidualnie dla każdej poddziedziny $\widehat{\Omega}_i$ są: macierz wybierająca $\tilde{\mathbf{L}}_E$ oraz macierz brzegowa $\tilde{\mathbf{B}}_H$.

Podstawowa technika klonowania makromodeli została przedstawiona w [78]. Zasada jej działania zostanie tu przedstawiona na przykładzie wyodrębnienia w dziedzinie obliczeniowej Ω trzech poddziedzin $\widehat{\Omega}_i$ o takich samych własnościach. Poszczególne etapy budowania komputerowej reprezentacji tego problemu przedstawiają się następująco:

1. Zdefiniowanie macierzy $\tilde{\mathbf{R}}'_H$ oraz $\tilde{\mathbf{R}}'_E$ reprezentujących dziedzinę obliczeniową $\Omega\backslash\widehat{\Omega}$.
2. Zdefiniowanie macierzy $\widehat{\widetilde{\mathbf{R}}}_{E\,1}$, $\widehat{\widetilde{\mathbf{R}}}_{H\,1}$, $\tilde{\mathbf{L}}$ oraz $\tilde{\mathbf{B}}$ reprezentujących pierwszą poddziedzinę obliczeniową $\widehat{\Omega}_1$.
3. Wyznaczenie w procesie redukcji rzędu modelu macierzy projekcji $\widehat{\widetilde{\mathbf{V}}}_1$.
4. Zbudowanie macierzy $\widehat{\widetilde{\mathbf{V}}}_\Upsilon$, $\widehat{\tilde{\mathbf{\Gamma}}}_\Upsilon$, $\tilde{\mathbf{B}}_\Upsilon$, $\tilde{\mathbf{L}}_\Upsilon$ przy zastosowaniu kopiowania macierzy reprezentujących poddziedzinę $\widehat{\Omega}_1$, zgodnie z r. (6.35).
5. Zastosowanie macierzy projekcji $\widehat{\widetilde{\mathbf{V}}}_\Upsilon$ wobec zbudowanych macierzy, w celu wyznaczenia macierzy zredukowanych: $\widehat{\tilde{\mathbf{\Gamma}}}_{m\,\Upsilon}$, $\tilde{\mathbf{B}}_{m\,\Upsilon}$, $\tilde{\mathbf{L}}_{m\,\Upsilon}$, zgodnie z r. (6.32f), (6.32g), (6.32h).
6. Zdefiniowanie wybierających $\tilde{\mathbf{L}}_{E\,i}$ i macierzy brzegowych $\tilde{\mathbf{B}}_{H\,i}$ oraz zbudowanie macierzy globalnych $\tilde{\mathbf{L}}_{E\,\Upsilon}$, $\tilde{\mathbf{B}}_{H\,\Upsilon}$, zgodnie z r. (6.36).

$$\widehat{\widetilde{\mathbf{V}}}_\Upsilon = \begin{bmatrix} \widehat{\widetilde{\mathbf{V}}}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \widehat{\widetilde{\mathbf{V}}}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \widehat{\widetilde{\mathbf{V}}}_1 \end{bmatrix} \tag{6.35a}$$

$$\widehat{\tilde{\mathbf{\Gamma}}}_\Upsilon = \begin{bmatrix} \widehat{\tilde{\mathbf{\Gamma}}}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \widehat{\tilde{\mathbf{\Gamma}}}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \widehat{\tilde{\mathbf{\Gamma}}}_1 \end{bmatrix} \tag{6.35b}$$

$$\widehat{\tilde{\mathbf{\Gamma}}}_1 = \widehat{\widetilde{\mathbf{R}}}_{E\,1}\widehat{\widetilde{\mathbf{R}}}_{H\,1} \tag{6.35c}$$

$$\tilde{\mathbf{B}}_\Upsilon = \begin{bmatrix} \tilde{\mathbf{B}}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{B}}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \tilde{\mathbf{B}}_1 \end{bmatrix} \tag{6.35d}$$

$$\tilde{\mathbf{L}}_{\Upsilon} = \begin{bmatrix} \tilde{\mathbf{L}}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{L}}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \tilde{\mathbf{L}}_1 \end{bmatrix} \tag{6.35e}$$

$$\tilde{\mathbf{L}}_{E\,\Upsilon} = \begin{bmatrix} \tilde{\mathbf{L}}_{E\,1}^T & \tilde{\mathbf{L}}_{E\,2}^T & \tilde{\mathbf{L}}_{E\,3}^T \end{bmatrix}^T \tag{6.36a}$$

$$\tilde{\mathbf{B}}_{H\,\Upsilon} = \begin{bmatrix} \tilde{\mathbf{B}}_{H\,1} & \tilde{\mathbf{B}}_{H\,2} & \tilde{\mathbf{B}}_{H\,3} \end{bmatrix} \tag{6.36b}$$

W przedstawionym przykładzie wyraźnie widać dwa zasadnicze miejsca, w których klonowanie makromodeli prowadzi do zmniejszenia czasu budowania macierzy opisujących badany problem:

1. Redukcja przeprowadzona jest jednokrotnie dla zbioru poddziedzin $\widehat{\Omega}_i$ o takich samych własnościach.
2. Budowa macierzy globalnych $\tilde{\widehat{\mathbf{V}}}_{\Upsilon}$, $\tilde{\hat{\mathbf{\Gamma}}}_{\Upsilon}$, $\tilde{\mathbf{B}}_{\Upsilon}$ i $\tilde{\mathbf{L}}_{\Upsilon}$ zostaje przyspieszona poprzez wielokrotne zastosowaniu raz już zdefiniowanych macierzy.

Przedstawiona technika umożliwia efektywne przeprowadzenie analizy FDTD z włączeniem kilkuset makromodeli. Należy przy tym podkreślić, że w tej wersji technika ta nie redukuje czasu symulacji iteracyjnej części algorytmu FDTD (r. (6.31)), lecz czas budowania schematu różnicowego zawierającego makromodele.

Skutecznym sposobem optymalizacji efektywności algorytmu zapisanego równaniami (6.31) jest diagonalizacja macierzy $\tilde{\hat{\mathbf{\Gamma}}}_m$, [78]. Proces ten został również opisany w p. 6.4. Efektem diagonalizacji jest efektywny zapis macierzy $\tilde{\hat{\mathbf{\Gamma}}}_m$ w postaci macierzy rzadkiej diagonalnej, z którym wiąże się zastosowanie do redukcji liczby zmiennych zmodyfikowanej macierzy projekcji $\tilde{\widehat{\mathbf{V}}}'$ zdefiniowanej w r. (6.26).

6.7 Optymalizacja klonowania makromodeli

Klonowanie makromodeli znacząco usprawnia proces budowania macierzy związanych z makromodelami. Jednak przy zastosowaniu dziesiątek, a nawet setek identycznych makromodeli pojawia się możliwość dalszego usprawnienia algorytmu. Usprawnienie to zostało przeprowadzone przez autora tej rozprawy.

Kierunki zmian algorytmu wynikają z postaci macierzy wymaganych do reprezentacji makromodeli w r. (6.31). Macierze te posiadają różne własności, a w związku z tym każde z r. (6.31) charakteryzuje się różną złożonością obliczeń:

- macierze rotacji pola elektrycznego $\tilde{\mathbf{R}}'_E$ oraz magnetycznego $\tilde{\mathbf{R}}'_H$ są macierzami, których wiersze posiadają małą liczbę elementów niezerowych, przez co ogólny algorytm mnożenia macierzy rzadkiej przez wektor jest efektywnym rozwiązaniem,
- macierze wybierająca pobudzenie makromodelu $\tilde{\mathbf{L}}_{E\,\Upsilon}$ oraz brzegowa $\tilde{\mathbf{B}}_{H\,\Upsilon}$ są macierzami o bardzo niskiej zawartości elementów niezerowych (co najwyżej jeden element niezerowy w wierszu i kolumnie macierzy), przez co macierz rzadka stanowi dla nich efektywną formę reprezentacji danych,

- macierz $\tilde{\tilde{\mathbf{\Gamma}}}_m$ w postaci podstawowej, r. (6.32f) i (6.32a), posiada strukturę macierzy rzadkiej z obszarami podmacierzy gęstych, co stanowi mało efektywny zapis w symulacjach obejmujących znaczną liczbę makromodeli,
- macierze $\tilde{\mathbf{B}}_m$ oraz $\tilde{\mathbf{L}}_m$ reprezentują zarówno interpolację jak i rzutowanie wynikające z redukcji, więc stanowią macierze rzadkie z zawartością podmacierzy gęstych - ponownie mało efektywne rozwiązanie (r. (6.32b) i (6.32c)).

Biorąc pod uwagę powyżej przedstawione obserwacje, zwiększenie efektywności obliczeń podstawowego algorytmu FDTD zawierającego sklonowane makromodele (r. (6.31)) zostało zrealizowane poprzez:

- ograniczenie zastosowania macierzy rzadkich do obliczeń,
- usunięcia z algorytmu danych powtarzających się.

Realizacja przedstawionych celów opiera się na rezygnacji z budowy macierzy $\tilde{\tilde{\mathbf{\Gamma}}}_\Upsilon$, $\tilde{\mathbf{B}}_\Upsilon$, $\tilde{\mathbf{L}}_\Upsilon$, zdefiniowanych w r. (6.32a), (6.32b), (6.32c) i prowadzi do dalszego usprawnienia techniki klonowania. Pierwszy etap modyfikacji algorytmu FDTD z klonowanymi makromodelami, który pozwala zrezygnować z budowania dużych macierzy $\tilde{\tilde{\mathbf{\Gamma}}}_m$, $\tilde{\mathbf{B}}_m$, $\tilde{\mathbf{L}}_m$, polega na uporządkowaniu wektorów granicznych makromodeli: pola elektrycznego $\mathbf{e}_M$ oraz magnetycznego $\mathbf{h}_M$. Do ilustracji opracowanych reguł postępowania posłuży poniższy przykład.

W dziedzinie obliczeniowej znajduje się υ makromodeli, które można podzielić na trzy zbiory ze względu na identyczne macierze je opisujące. Macierz wybierająca pobudzenie dla wszystkich makromodeli, $\tilde{\mathbf{L}}_{E\,\Upsilon}$, układa próbki pól w sposób przedstawiony w r. (6.37).

$$\mathbf{e}_{M\,\Upsilon} = [\ \underbrace{\mathbf{f}_{1,1}}_{K_1}\ \underbrace{\mathbf{f}_{1,2}\ \ \mathbf{f}_{2,2}\ \ \ldots\ \ \mathbf{f}_{K_2,2}}_{K_2}\ \underbrace{\mathbf{f}_{1,3}\ \ \mathbf{f}_{2,3}\ \ \ldots\ \ \mathbf{f}_{K_3,3}}_{K_3}\ ] \tag{6.37}$$

przy czym:
$\mathbf{f}_{i,j}$ - wektor próbek pola elektrycznego, który stanowi pobudzenie i-tego makromodelu z j-tego zbioru klonów,
K_j - liczba makromodeli w j-tym zbiorze klonów.

Każdy wektor $\mathbf{f}_{i,j}$ reprezentuje inny zakres pól. Jednak, zgodnie z r. (6.31d), każdy z tych wektorów, który należy do tego samego zbioru klonów, jest mnożony przez takie same macierze $\tilde{\tilde{\mathbf{V}}}_j\tilde{\mathbf{B}}_j$, (realizacja m.in. interpolacji pól oraz pobudzenia makromodeli) oraz przez takie same macierze $\tilde{\tilde{\mathbf{V}}}_j^T\tilde{\tilde{\mathbf{\Gamma}}}_j\tilde{\tilde{\mathbf{V}}}_j$. Następnie, zgodnie z r. (6.31e), odpowiednie zakresy elementów wektora $\tilde{\tilde{\mathbf{h}}}_{m\,\Upsilon}$ mnożone są przez takie same macierze $\tilde{\mathbf{L}}_j$. W konsekwencji można zapisać odpowiedzi makromodeli w postaci analogicznej do pobudzenia:

$$\mathbf{h}_{M\,\Upsilon} = [\ \underbrace{\mathbf{g}_{1,1}}_{K_1}\ \underbrace{\mathbf{g}_{1,2}\ \ \mathbf{g}_{2,2}\ \ \cdots\ \ \mathbf{g}_{K_2,2}}_{K_2}\ \underbrace{\mathbf{g}_{1,3}\ \ \mathbf{g}_{2,3}\ \ \cdots\ \ \mathbf{g}_{K_3,3}}_{K_3}\ ] \tag{6.38}$$

Ponieważ dla j-tego zbioru klonów wykonywane są takie same działania, algorytm został zmodyfikowany poprzez zdefiniowanie macierzy pobudzenia F_j oraz odpowiedzi G_j dla każdego zbioru klonów:

$$\mathbf{F}_j = [\ \mathbf{f}_{1,j}\ \ \mathbf{f}_{2,j}\ \ \ldots\ \ \mathbf{f}_{K_j,j}\ ] \tag{6.39}$$

$$\mathbf{G}_j = [\ \mathbf{g}_{1,j}\ \ \mathbf{g}_{2,j}\ \ \ldots\ \ \mathbf{g}_{K_j,j}\ ] \tag{6.40}$$

W konsekwencji układ równań (6.31) przyjmuje postać r. (6.41).

$$\tilde{\mathbf{e}}'^{\tau} = \tilde{\mathbf{e}}'^{\tau-1} + \Delta_t \tilde{\mathbf{R}}'_H \mathbf{h}'^{\tau-0,5} + \Delta_t \tilde{\mathbf{B}}_{H\,\Upsilon} \mathbf{h}'^{\tau-0,5}_M \tag{6.41a}$$

$$\tilde{\mathbf{h}}'^{\tau+0,5} = \tilde{\mathbf{h}}'^{\tau-0,5} - \Delta_t \tilde{\mathbf{R}}'_E \tilde{\mathbf{e}}'^{\tau} \tag{6.41b}$$

$$\mathbf{e}^{\tau}_{M\,\Upsilon} = \tilde{\mathbf{L}}_{E\,\Upsilon} \tilde{\mathbf{e}}'^{\tau} \tag{6.41c}$$

Dla $j \in \{1, 2, .., N_K\}$ wykonaj

$$\begin{cases} F^{\tau}_j \leftarrow \mathbf{e}^{\tau}_{M\,\Upsilon} & (6.41d) \\ G^{\tau+0,5}_j = 2G^{\tau-0,5}_j - \Delta_t^2 \tilde{\tilde{\mathbf{\Gamma}}}_{m\,j} G^{\tau-0,5}_j - G^{\tau-1,5}_j + \Delta_t \tilde{\mathbf{B}}_{m\,j} \left(F^{\tau}_j - F^{\tau-1}_j \right) & (6.41e) \\ \mathbf{h}^{\tau+0,5}_{M\,\Upsilon} \leftarrow \tilde{\mathbf{L}}^T_{m\,j} G^{\tau+0,5}_j & (6.41f) \end{cases}$$

przy czym:
N_K - liczba zbiorów klonów,
$\leftarrow$ - oznacza zmianę interpretacji zbioru danych z postaci wektorowej na macierzową (r. (6.41d)) lub z macierzowej na wektorową (r. (6.41f)).

$$\tilde{\tilde{\mathbf{\Gamma}}}_{m\,j} = \tilde{\tilde{\mathbf{V}}}^T_j \tilde{\tilde{\mathbf{\Gamma}}}_j \tilde{\tilde{\mathbf{V}}}_j \tag{6.42a}$$

$$\tilde{\mathbf{B}}_{m\,j} = \tilde{\tilde{\mathbf{V}}}^T_j \tilde{\mathbf{B}}_j \tag{6.42b}$$

$$\tilde{\mathbf{L}}_{m\,j} = \tilde{\tilde{\mathbf{V}}}^T_j \tilde{\mathbf{L}}_j \tag{6.42c}$$

Przeprowadzona optymalizacja algorytmu FDTD z makromodelami prowadzi zarówno do znacznej oszczędności pamięci jak i do wzrostu efektywności obliczeń:

- Oszczędność pamięci związana jest z przechowywaniem tylko jednego zestawu macierzy $\tilde{\tilde{\mathbf{\Gamma}}}_{m\,i}$, $\tilde{\mathbf{B}}_{m\,i}$, $\tilde{\mathbf{L}}_{m\,i}$ dla jednego zbioru klonów makromodeli, przez co unika się kopiowania danych, które występowało przy budowaniu macierzy $\tilde{\tilde{\mathbf{\Gamma}}}_{\Upsilon}$, $\tilde{\mathbf{B}}_{\Upsilon}$, $\tilde{\mathbf{L}}_{\Upsilon}$.

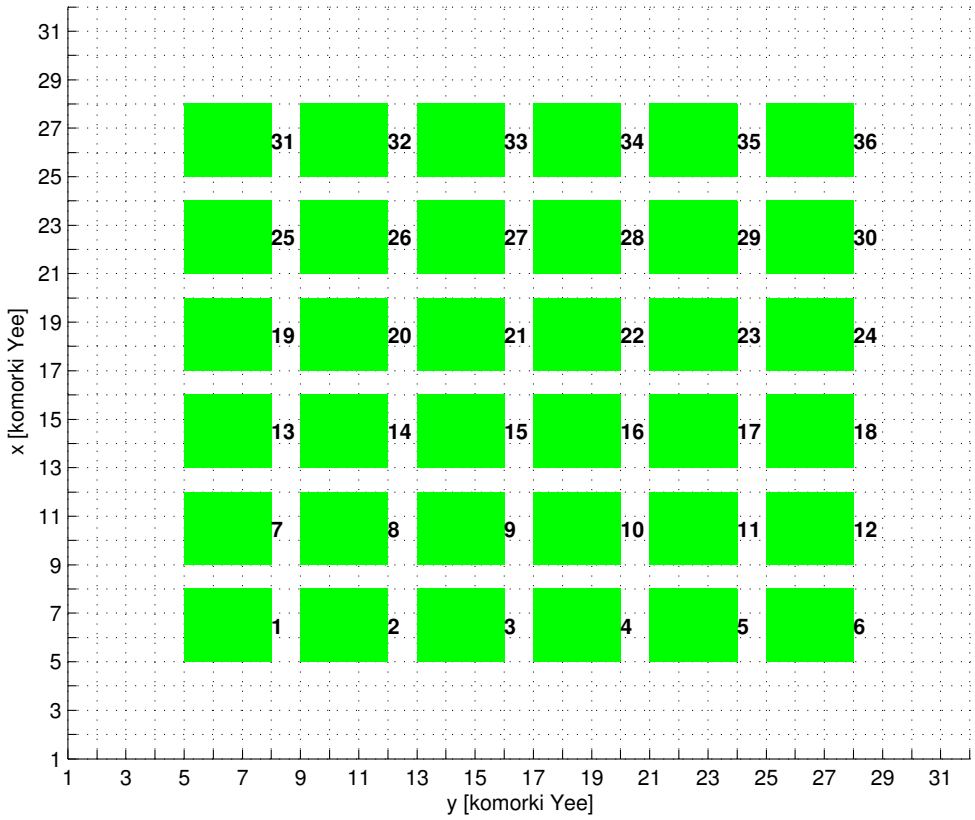
- Wzrost efektywności wynika z redukcji obliczeń wykonywanych przy pomocy macierzy rzadkich (r. (6.31d) i r. (6.31e)) na rzecz obliczeń wykonywanych przy pomocy macierzy gęstych (r. (6.41d) i r. (6.41f)).

Przeprowadzona optymalizacja klonowania makromodeli nie zmienia liczby operacji zmiennoprzecinkowych określonej przez r. (6.34), lecz poprzez znaczne poprawienie organizacji danych i ograniczenie stosowania macierzy rzadkich uzyskano znaczny wzrost efektywności obliczeń, co wykazano w p. 6.8 oraz w rozdziale 7.

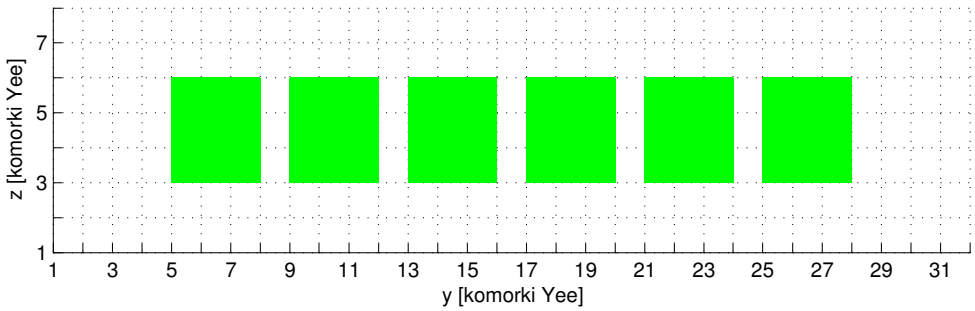
Dodatkowo, warto zaznaczyć, że zastosowanie redukcji liczby zmiennych dla obszaru z lokalnym zagęszczeniem siatki prowadzi do zwiększenia maksymalnej wartości kroku czasowego, jaki może zostać użyty w schemacie różnicowym opisującym analizowany problem. Korzystnie odróżnia to makromodele od obszaru z M-krotnym lokalnym zagęszczeniem siatki, dla którego krok czasowy musi zostać zmniejszony M-krotnie w porównaniu do kroku czasowego wymaganego przez siatkę podstawową dziedziny obliczeniowej. Wartość kroku czasowego w schemacie różnicowym z makromodelami określana jest indywidualnie dla każdego problemu. Przykłady liczbowe zostały opisane w p. 6.8 oraz w p. 7.2

6.8 Testy numeryczne

W celu demonstracji zysku z zastosowania klonowania makromodeli do analizy wybrano problem o rozmiarze $32 \times 32 \times 8$ komórek siatki Yee, w którym zdefiniowano 36 makromodeli o rozmiarze $3 \times 3 \times 3$ komórek siatki Yee. Położenie makromodeli w siatce dyskretyzacji prezentują rys. 6.1 oraz rys. 6.2. Każdy makromodel reprezentuje poddziedzinę o identycznych własnościach fizycznych.



Rysunek 6.1: Układ makromodeli w analizowanym problemie - rzut na płaszczyznę XY



Rysunek 6.2: Układ makromodeli w analizowanym problemie - rzut na płaszczyznę YZ

Symulacje przeprowadzono dla różnych stopni zagęszczenia siatki dyskretyzacji w wyodrębnionych podobszarach, przy czym za każdym razem określono rząd makromodeli

równy trzy. Zmierzoną efektywność obliczeń dla procesorów komputerowych prezentuje tab. 6.1, natomiast dla akceleratora graficznego tab. 6.2.

Zrównoleglenie obliczeń nie zostało przeprowadzone dla procesorów, gdyż równomierny podział obliczeń na wiele jednostek obliczeniowych przy gruboziarnistym zrównolegleniu algorytmu o znacznym poziomie skomplikowania jest zadaniem dużo trudniejszym i rozbudowanym niż zrównoleglenie drobnoziarniste. Zrównoleglenie drobnoziarniste polega na efektywnym zrównolegleniu na poziomie operacji podstawowych, natomiast gruboziarniste wymaga posegregowania danych z uwzględnieniem szacowanego czasu obliczeń realizowanego przez poszczególne bloki zadań i przydzielenie ich do rdzeni procesorów. Dodatkowa komplikacja wynika również z faktu, iż zrównoleglenie gruboziarniste nie zawsze musi być opłacalne, gdyż obliczenia np. mnożenia macierzy gęstej przez wektor o niewielkich rozmiarach (dane mieszczą się w pamięci podręcznej L1) może zostać wykonane szybciej dla jednego procesora niż dla wielu z uwagi na pominięcie czasu komunikacji pomiędzy procesami oraz brak opóźnień związanych z dostępem do tego samego zakresu pamięci przez wiele wątków. Uznano, że wystarczającą informację o efektywności badanej implementacji algorytmu przeznaczonej dla CPU dostarczą rezultaty pomiarów implementacji jednowątkowej.

Na podstawie danych zawartych w tab. 6.1 i tab. 6.2 można stwierdzić, że zysk z zastosowania makromodeli jest znaczący i tym większy, im większe jest lokalne zagęszczenie siatki dyskretyzacji. Z porównania obu tabel wynika, że obliczenia na akceleratorze graficznym są przeprowadzane ponad **40-krotnie** szybciej w porównaniu do PC1 oraz ponad **57-krotnie** szybciej w porównaniu do PC2.

Tablica 6.1: Efektywność obliczeń implementacji algorytmu FDTD z makromodelami przeznaczonej dla CPU

CPU		Efektywność obliczeń [Mcells/s]
PC1	FDTD bez makromodeli	24,0
PC1	$M = 3$	9,0
PC1	$M = 5$	32,8
PC1	$M = 7$	83,3
PC2	FDTD bez makromodeli	15,1
PC2	$M = 3$	6,2
PC2	$M = 5$	22,8
PC2	$M = 7$	58,7

Tablica 6.2: Efektywność obliczeń implementacji algorytmu FDTD z makromodelami przeznaczonej dla GPU. Pomiary wykonano na karcie GTX 580.

	Efektywność obliczeń [Mcells/s]
FDTD bez makromodeli	20
$M = 3$	360
$M = 5$	1320
$M = 7$	3353

Dla określonego w tym podpunkcie problemu wyznaczono również skalę zmniejszenia liczby operacji zmiennoprzecinkowych, wykonywanych w trakcie jednej iteracji schematu

różnicowego, przy zastosowaniu różnych modyfikacji algorytmu FDTD. Dla problemu podstawowego o rozmiarze $32 \times 32 \times 8$ liczba operacji zmiennoprzecinkowych dla implementacji jawnej wynosi $L_F = 1,38$ Gflops. Poziom zwiększenia kosztów numerycznych przy zmniejszeniu kroku dyskretyzacji w całej dziedzinie obliczeniowej o M zawiera tab. 6.3 we wierszu oznaczonym przez L_{FT}. W tab. 6.3 przedstawiono również zysk z zastosowania makromodeli oraz lokalnego zagęszczenia siatki dyskretyzacji, zdefiniowanego w 36 poddziedzinach, odpowiadającego schematowi różnicowemu z rys. 6.1 oraz rys. 6.2.

Przedstawione w tab. 6.3 dane ukazują, iż koszt numeryczny towarzyszący zmniejszeniu kroku dyskretyzacji w całej poddziedzinie jest znaczący. Zastosowanie lokalnego zagęszczenia siatki dyskretyzacji w dużym stopniu koszt ten redukuje, jednak wraz ze zwiększaniem zagęszczenia siatki również obserwowany jest jego duży wzrost. Wprowadzenie do schematu różnicowego makromodeli prowadzi do znacznego obniżenia kosztów numerycznych przy dużym zagęszczeniu siatki dyskretyzacji. Widoczne jest przy tym silne obniżenie kosztów numerycznych po zastosowaniu diagonalizacji macierzy $\widehat{\widetilde{\mathbf{\Gamma}}}_m$.

Tablica 6.3: Przykład redukcji liczby operacji zmiennoprzecinkowych przy zastosowaniu lokalnego zagęszczenia siatki dyskretyzacji oraz makromodeli

	M		3	5	7
algorytm macierzowy bez modyfikacji	L_{FT}	[Gflops]	37,16	172,03	472,06
lokalne zagęszczenie siatki	L_{FSG}	[Gflops]	3,36	8,96	21,12
makromodele bez diagonalizacji	L_{FM1}	[Gflops]	11,83	11,83	11,83
makromodele z diagonalizacją	L_{FM2}	[Gflops]	4,30	4,30	4,30

Dodatkowym czynnikiem, który wpływa na czas przeprowadzenia symulacji jest zastosowany krok czasowy. Zmianę jego wartości dla różnych wariantów analizy przedstawiono w tab. 6.4. Ponieważ do przeprowadzenia symulacji działania analizowanej struktury w odcinku czasu T_a wymagane jest przeprowadzenie α iteracji, tak iż $T_a = \alpha\Delta_t$, to poprzez k-krotne zmniejszenie wartości kroku czasowego wymagane jest przeprowadzenie k-krotnie więcej iteracji. Ograniczenie skali wzrostu kroku czasowego po wprowadzeniu makromodeli zamiast np. lokalnego zagęszczenia siatki ma zatem istotne znaczenie przy ocenie metod skrócenia czasu numerycznej analizy zagadnień elektromagnetycznych. Na podstawie danych z tab. 6.4 można zauważyć, że wprowadzenie makromodeli do schematu różnicowego wymaga znacznie mniejszego skrócenia kroku czasowego niż wynikałoby to ze skali lokalnego zagęszczenia siatki dyskretyzacji.

Tablica 6.4: Zmiana kroku czasowego Δ_t dla różnych wariantów analizy w odniesieniu do kroku czasowego Δ_{tB} ze schematu różnicowego bez modyfikacji (z siatką dla $M = 1$). Tabela zawiera wartości Δ_{tB}/Δ_t.

M	3	5	7
algorytm macierzowy bez modyfikacji	3	5	7
FDTD z lokalnym zagęszczeniem siatki	3	5	7
FDTD z makromodelami	2,09	2,53	3,07

6.9 Podsumowanie

Implementacja algorytmu FDTD zawierającego makromodele pozwala na znaczne zwiększenie efektywności obliczeń przy zastosowaniu znaczniej mniejszej liczby zmiennych. Algorytm ten można efektywnie dostosować do akceleratora graficznego, co umożliwia dalsze skrócenie czasu analizy. Efektywność obliczeń jest tym większa im większy jest stopień zagęszczenia siatki dyskretyzacji w wyodrębnionych poddziedzinach. W rozdziale 7 wykazano, iż zysk z modyfikacji algorytmu FDTD jest znaczący również w sytuacji, gdy analizowane są złożone struktury.

ROZDZIAŁ 7

Testy numeryczne

Przeprowadzona w poprzednich rozdziałach analiza efektywności obliczeń dla różnych rozwiązań sprzętowych oraz algorytmicznych wykonana została przy badaniu drgań fali elektromagnetycznej w rezonatorze prostopadłościennym. Wybór prostej struktury umożliwił uwydatnienie w rozprawie przede wszystkim czynników wpływających na efektywność obliczeń. Dodatkowo, z uwagi na rezonansowy charakter zjawiska, struktura ta umożliwia łatwe przeprowadzenie weryfikacji stabilność schematu różnicowego. Po rozpoznaniu podstawowych własności omówionych rozwiązań w dalszej części rozprawy zademonstrowano ich porównanie przy symulacji działania struktury fotonicznej [84] oraz anteny mikrofalowej [42].

Przeprowadzone symulacje zakładają, że materiały tworzące analizowane struktury są bezstratne. Wymagany w analizach warunek graniczny dyskretnej przestrzeni obliczeniowej (patrz p. A.3), który symuluje otwartą przestrzeń, zrealizowano za pomocą algorytmu CPML (ang. *Convolution PML*), [86]. Wpływ na efektywność obliczeń wprowadzenia do schematu różnicowego algorytmu CPML jest znaczny zarówno dla procesorów komputerowych jak i akceleratorów graficznych, co zostało udokumentowane w p. 7.1.

7.1 Wpływ zastosowania algorytmu CPML na efektywność obliczeń

Algorytm CPML wprowadza znaczące zmiany do algorytmu FDTD, które mają wpływ na jego efektywność obliczeń. Zmiany te są zróżnicowane dla różnych komórek Yee w zależności od ich położenia, co przedstawia rys. 7.1. Wpływ położenia komórki Yee na sposób aktualizacji wartości próbek pól wynika ze sposobu symulacji otwartej przestrzeni, która sprowadza się do tłumienia propagowanej w określonym kierunku fali elektromagnetycznej. Istotną własnością algorytmu CPML jest wprowadzenie dodatkowych zmiennych oraz stałych (w obrębie komórki Yee) współczynników, które odpowiadają za tłumienie fali

tylko w jednym kierunku.

Z tego względu w dziedzinie obliczeniowej można wyróżnić cztery rodzaje podobszarów, które charakteryzują się różną złożonością numeryczną:

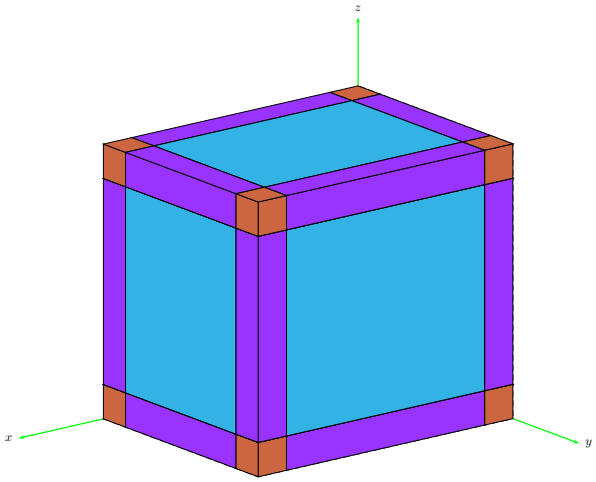
- obszar A1, w którym fala tłumiona jest w jednym kierunku,
- obszar A2, w którym fala tłumiona jest w dwóch kierunkach,
- obszar A3, w którym fala tłumiona jest w trzech kierunkach,
- obszar A0 wolny od modyfikacji spowodowanej przez wprowadzenie do schematu różnicowego algorytmu CPML.

Koszt numeryczny aktualizacji wartości pól dla każdego z wymienionych powyżej obszarów znacząco się różni. W algorytmie FDTD do reprezentacji jednej komórki Yee stosuje się sześć próbek pól oraz sześć współczynników, które biorą udział w aktualizacji wartości tych próbek w każdej iteracji. Po wprowadzeniu algorytmu CPML do tłumienia fali przy propagacji w jednym kierunku każda komórka Yee zawiera dodatkowo cztery zmienne. Liczbę stałych współczynników wymaganych przez CPML można uznać za pomijalnie małą, gdyż jest ona w przybliżeniu równa czterokrotności sumy szerokości warstwy CPML w każdym kierunku przestrzeni. Również liczba operacji wykonywana przy aktualizacji wartości pól z pojedynczej komórki Yee silnie zależy od tego, do którego z wymienionych wyżej obszarów dziedziny obliczeniowej należy: w podstawowym algorytmie FDTD wynosi ona 42 (patrz p. 3.2), dodanie tłumienia fali przez CPML w jednym kierunku zwiększa tę liczbę o 24 operacje zmiennoprzecinkowe. Zatem dla obszaru, w który fala tłumiona jest w trzech kierunkach liczba zmiennych opisująca pojedynczą komórkę Yee wynosi 24, a do jej aktualizacji wymagane jest wykonanie 114 operacji zmiennoprzecinkowych. Liczby te obrazują znaczny wzrost kosztów numerycznych związanych z wprowadzeniem algorytmu CPML do schematu różnicowego, co dokumentuje tab. 7.1.

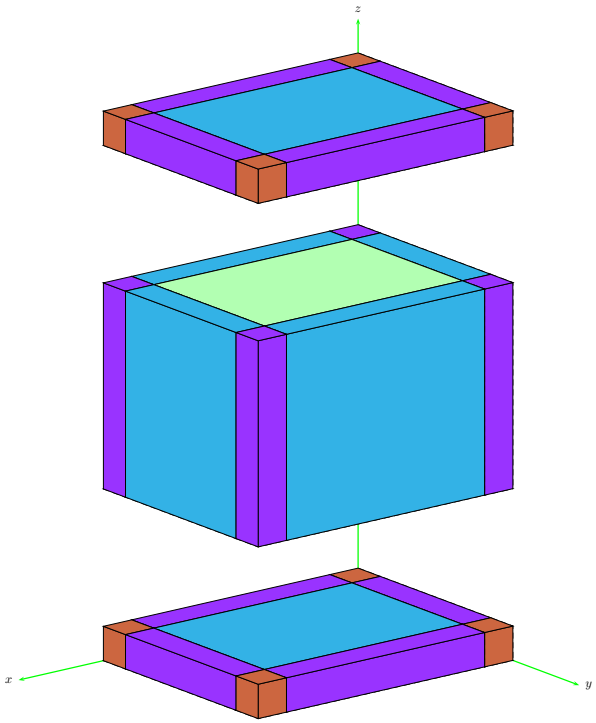
Tablica 7.1: Koszt numeryczny występujący w algorytmie FDTD, w którym zaimplementowano algorytm CPML

podobszar	liczba zmiennych wymagana do opisania wartości próbek pola w jednej komórce Yee	liczba operacji zmiennoprzecinkowych wykonywana w jednej iteracji przy aktualizacji wartości próbek pól jednej komórki Yee
A0	6	42
A1	10	66
A2	14	90
A3	18	114

Różny stopień złożoności obliczeń w zależności od położenia komórek Yee jest cechą niekorzystną z punktu widzenia zrównoleglenia gruboziarnistego. W takiej sytuacji, przy równomiernym podziale dziedziny obliczeniowej na poddziedziny, procesory, które wykonują obliczenia związane z obszarem zmodyfikowanym przez algorytm CPML, wymagają dłuższego czasu na wykonanie obliczeń w porównaniu do procesorów, które realizują obliczenia dla podstawowej wersji algorytmu FDTD. Zastosowanie w takiej sytuacji prostego algorytmu, który różnicowałby rozmiar poddziedzin w zależności od liczby operacji wymaganych przez algorytm CPML w danej poddziedzinie również byłoby ułomne. Wynika to z silnego wpływu rozmiaru poddziedziny na efektywność obliczeń. W rezultacie docelowa



a) Dziedzina obliczeniowa z wyróżnionymi obszarami o różnym stopniu złożoności obliczeń ze względu na algorytm CPML



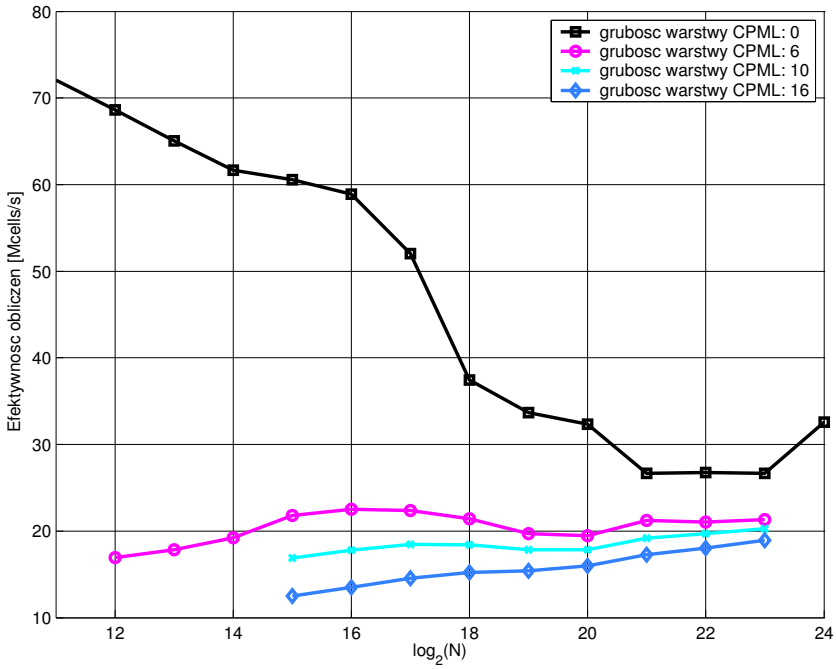
b) Przekrój przez dziedzinę obliczeniową

- obszar, w którym algorytm CPML tłumi falę w jednym kierunku
- obszar, w którym algorytm CPML tłumi falę w dwóch kierunkach
- obszar, w którym algorytm CPML tłumi falę w trzech kierunkach
- obszar poza zasięgiem działania algorytmu CPML

Rysunek 7.1: Wyróżnienie w dziedzinie obliczeniowej obszaru modyfikowanego przez algorytm CPML

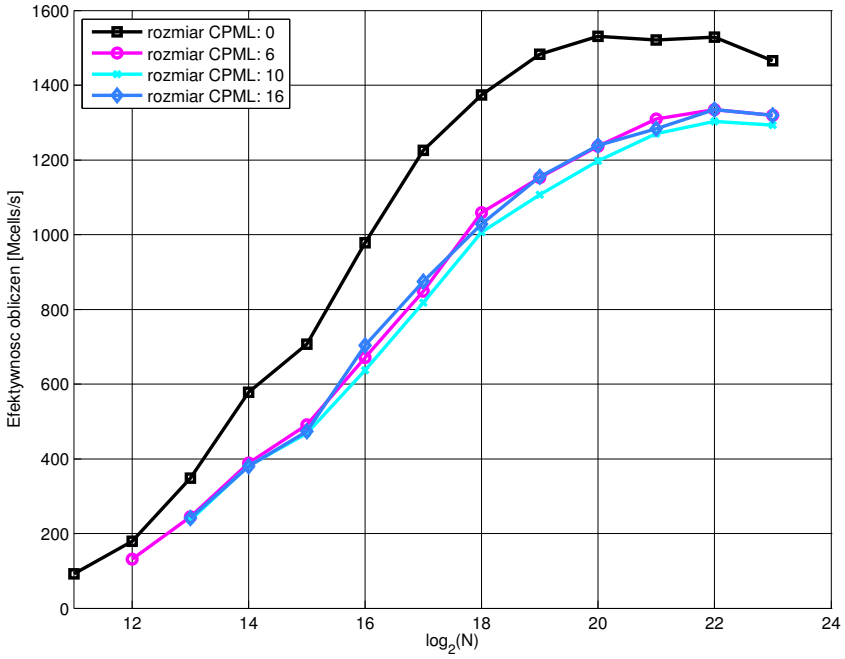
implementacja algorytmu FDTD zawierającego CPML mogłaby dokonać równomiernego podziału obliczeń tylko w sposób dynamiczny - poprzez pomiar czasu obliczeń poszczególnej poddziedziny i adekwatnej zmiany rozmiaru dziedziny obliczeniowej. Z uwagi na stopień komplikacji przewidywanego rozwiązania ograniczono się do podziału dziedziny obliczeniowej równomiernego ze względu na rozmiar poddziedziny.

Należy przy tym podkreślić, że w zrównolegleniu drobnoziarnistym sytuacja jest znacznie bardziej korzystna. W zrównolegleniu algorytmu FDTD dla akceleratorów graficznych obliczenia wykonywane są dla obszarów o rozmiarze $16 \times 16 \times 1$ oczek siatki Yee. Implementacja algorytmu FDTD zawierająca CPML wykonuje test dla całego tego obszaru, czy realizowany jest w nim warunek brzegowy. Jeśli nie, to działania numeryczne są przeprowadzane bez modyfikacji, co prowadzi do zachowania znacznego poziomu efektywności obliczeń. Należy to uznać za kolejną przewagę środowiska akceleratora graficznego nad środowiskiem wykonującym obliczenia na procesorach komputerowych.

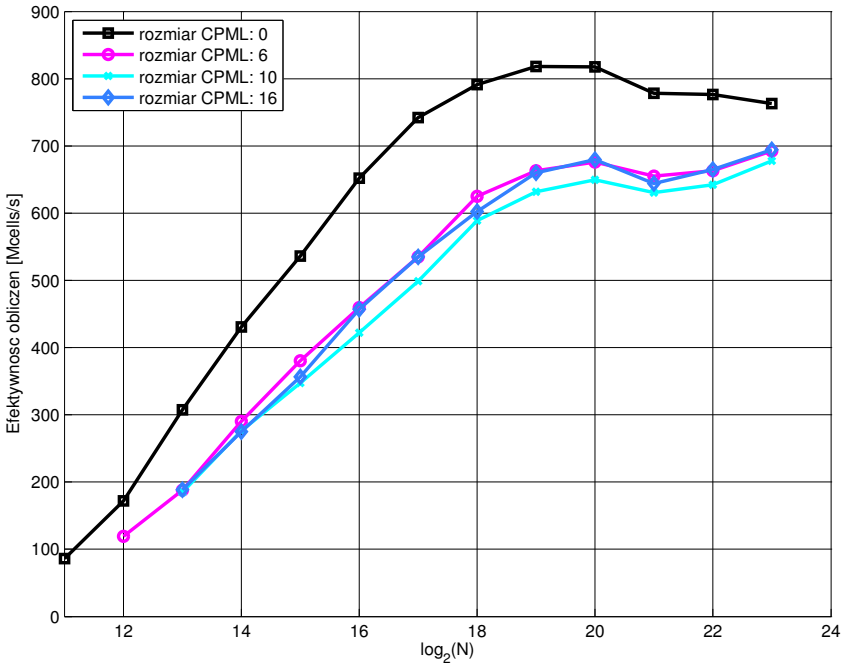


Rysunek 7.2: Efektywność obliczeń implementacji algorytmu FDTD przeznaczonej dla CPU (PC1) w zależności od rozmiaru obszaru tworzącego CPML

Długość obszaru, w którym obowiązuje algorytm CPML wynosi zwykle od ośmiu do piętnastu komórek Yee, przy czym jakość warunku brzegowego (poziom fali odbitej od obszaru zmodyfikowanego przez CPML) jest tym lepsza im obszar stanowiący CPML jest większy. Jednak zwiększenie obszaru zawierającego CPML, powoduje wzrost kosztów numerycznych, a więc również zmniejsza efektywność obliczeń. Wpływ rozmiaru warstwy absorpcyjnej na efektywność obliczeń dla CPML dla CPU przedstawia rys. 7.2. Rezultaty analogicznych testów przeprowadzonych dla akceleratora graficznego przedstawia rys. 7.3. W obu środowiskach dodatkowe obliczenia silnie zmniejszają efektywność obliczeń, jednak spadek ten jest znacznie większy dla procesorów komputerowych. Można założyć, że jest to spowodowane przez znacznie większe zapotrzebowanie na rejestry procesora do wykonania dodatkowych operacji oraz na konieczność operowania na znacznej liczbie obszarów



Rysunek 7.3: Efektywność obliczeń implementacji algorytmu FDTD przeznaczonej dla GPU w zależności od rozmiaru obszaru tworzącego CPML. Obliczenia przeprowadzono ze zmiennymi w pojedynczej precyzji na karcie Nvidia GTX 580.



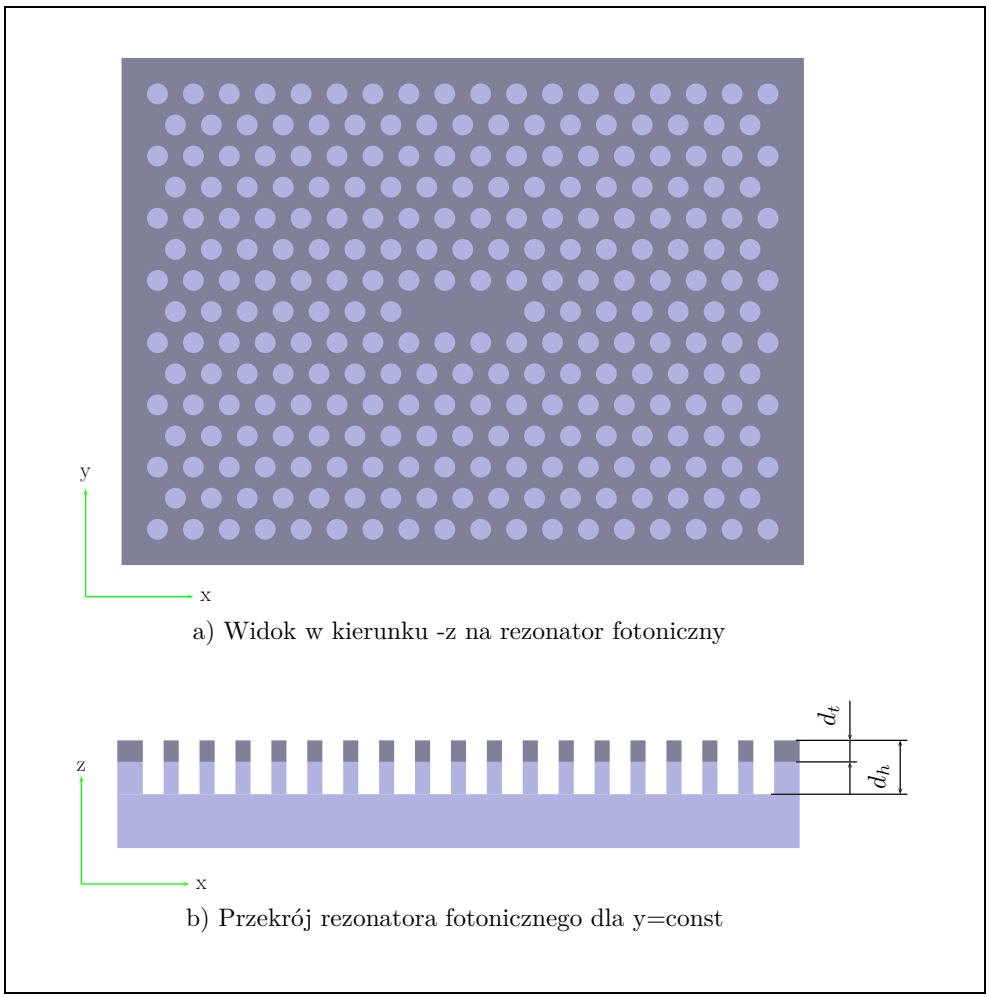
Rysunek 7.4: Efektywność obliczeń implementacji algorytmu FDTD przeznaczonej dla GPU w zależności od rozmiaru obszaru tworzącego CPML. Obliczenia przeprowadzono ze zmiennymi w podwójnej precyzji na karcie Nvidia GTX 580.

pamięci (procesor jest w stanie śledzić ograniczoną liczbę strumieni danych).

Z porównania rys. 7.3 i rys. 7.2 wynika, że akcelerator graficzny posiada również przewagę nad procesorami komputerowymi, w sytuacji, gdy obliczenia są złożone. W sytuacji, gdy obliczenia są przeprowadzane na akceleratorze graficznym na zmiennych w podwójnej precyzji (rys. 7.4), efektywność obliczeń jest dwukrotnie mniejsza w odniesieniu do obliczeń przeprowadzonych na zmiennych w pojedynczej precyzji (rys. 7.2).

7.2 Struktura fotoniczna

Do analizy wybrano strukturę fotoniczną, którą przedstawia rys. 7.5. Górny obszar dziedziny obliczeniowej stanowi powietrze, natomiast dolny złożony jest z dwóch warstw dielektryków, w których znajduje się 260 otworów zawierających powietrze.



Rysunek 7.5: Rezonator fotoniczny

Podobnie jak w [84] gabaryty struktury określone zostały w zależności od współczynnika skalującego a. Jego wartość w symulacjach ustalono, zgodnie z [10], na $a = 420$ nm.

Dielektryczne podłoże rezonatora stanowi krzemionka ($\epsilon_r = 2,25$). Ponad nią znajduje się warstwa krzemu o grubości $d_t = 0,6a$ i $\epsilon_r = 11,56$. Wysokość otworów powietrznych

wynosi $d_h = 1,5a$, a ich promień równy jest $0,29a$. Odległości pomiędzy środkami sąsiadujących ze sobą otworów zostały tak dobrane, aby tworzyły one siatkę trójkątów równobocznych o długości boków równej a. Siatka dyskretyzacji została ustalona w trzech wariantach, które zostały scharakteryzowane w tab. 7.2. Tak ustalone wartości umożliwiają przeprowadzenie porównania efektywności obliczeń dla różnych stopni zagęszczenia siatki dyskretyzacji: poszczególne wartości kroków dyskretyzacji przestrzeni zachowują relację $5{:}\frac{5}{3}{:}1$. W ten sposób można zarówno dokonać porównania różnego poziomu efektywności dla różnego rozmiaru dyskretnej dziedziny obliczeniowej jak i ocenić wpływ wprowadzonych w dalszym etapie modyfikacji algorytmicznych. Kroki dyskretyzacji z wariantu C analizy wybrano zgodnie z danymi podanymi w artykule [84].

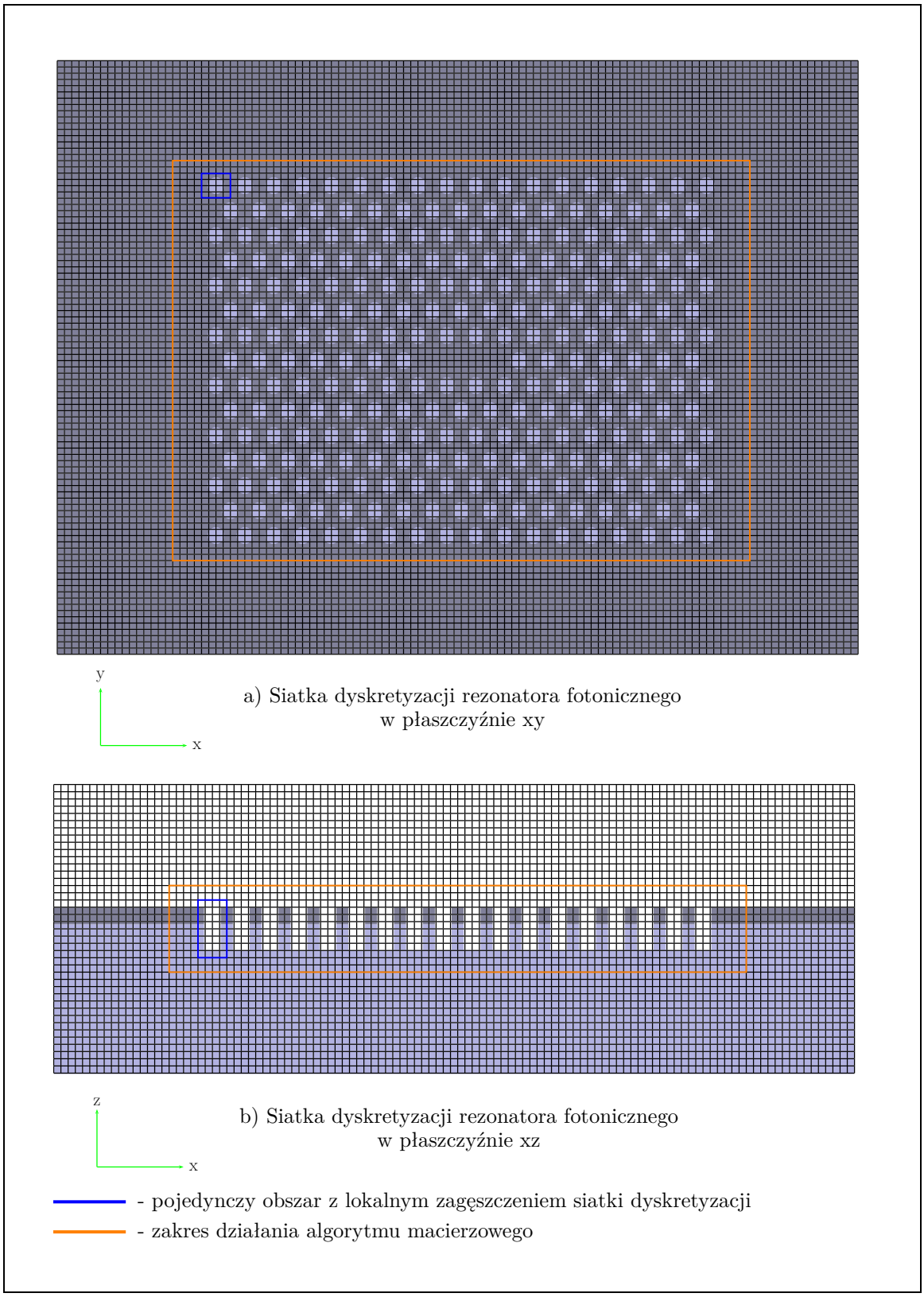
Tablica 7.2: Podstawowe parametry analizy dla różnych jej wariantów

	wariant A	wariant B	wariant C
dx	$5\frac{a}{20}$	$\frac{5}{3}\frac{a}{20}$	$\frac{a}{20}$
dy	$5\frac{a\sqrt{3}}{40}$	$\frac{5}{3}\frac{a\sqrt{3}}{40}$	$\frac{a\sqrt{3}}{40}$
dz	$5\frac{a}{20}$	$\frac{5}{3}\frac{a}{20}$	$\frac{a}{20}$
dt [ps]	4,104	1,368	0,821
I_r	80	240	400
J_r	64	192	320
K_r	17	51	85
N_r	0,087 Mcells	2,35 Mcells	10,88 Mcells
I	112	272	432
J	96	224	352
K	41	75	109
N	0,441 Mcells	4,570 Mcells	16,57 Mcells
liczba iteracji	14 000	42 000	70 000

Rozmiar dziedziny obliczeniowej poszczególnych symulacji wynosi $I \times J \times K$ i uwzględnia on obszar dopasowanej warstwy absorpcyjnej. Liczbę komórek dziedziny obliczeniowej oznaczono przez N. Grubość warstwy z CPML ustalono na 14 komórek Yee w kierunkach x i y oraz 9 komórek Yee w kierunku osi z (rys. 7.6). Za zasadniczy obszar struktury uznano obszar o rozmiarze $I_r \times J_r \times K_r$ oczek siatki Yee, przy czym $I = 32 + I_r$, $J = 32 + J_r$, $K = 25 + K_r$. Sposób dyskretyzacji badanej struktury dla symulacji w wariancie A przedstawia rys. 7.6. W kolejnych wariantach analizy siatka dyskretyzacji zagęszczana jest trzy i pięciokrotnie względem siatki z wariantu A analizy. Liczba komórek obszaru zasadniczego kolejnych wariantów symulacji zachowuje proporcje $1{:}3^3{:}5^3$, jednak z uwagi na CPML rozmiar dziedziny obliczeniowej N nie zachowuje takich proporcji.

7.2.1 Sprzętowe metody zmniejszenia czasu numerycznej analizy

Czas symulacji wykonanych dla trzech różnych stopni zagęszczenia siatki dyskretyzacji dla akceleratora graficznego oraz procesora komputerowego PC1 przedstawia tab. 7.3. Wraz z k-krotnym zagęszczeniem siatki dyskretyzacji koszt numeryczny wzrasta k^4-krotnie (patrz p. 3.2). Jednak wzrost czasu obliczeń dla procesora (współczynnik κ_1 z tab. 7.3) oraz dla akceleratora graficznego (współczynnik κ_2 z tab. 7.3), który wynika z zagęszczenia



Rysunek 7.6: Siatka dyskretyzacji rezonatora fotonicznego w wariancie A

siatki dyskretyzacji, jest znacznie mniejszy niż k^4. Jest to spowodowane przez wprowadzenie do schematu różnicowego algorytmu CPML, przez co wzrost liczby komórek jest mniejszy od k^3 (szerokość warstwy CPML jest identyczna dla każdego wariantu analizy - nie jest ona zależna od stopnia zagęszczenia siatki dyskretyzacji), a także przez wzrost efektywności obliczeń wraz ze wzrostem rozmiaru problemu, który obserwowano zarówno dla procesorów (rys. 7.2) jak i akceleratorów graficznych (rys. 7.3 i rys.7.4). Dla badanych CPU oraz GPU wzrost czasu obliczeń przy zwiększeniu rozmiaru dziedziny obliczeniowej jest bardzo zbliżony: dla GPU wynosi on 158×, natomiast dla CPU 155×. Z przytoczonych danych jednoznacznie wynika, że czas symulacji wykonanej na GPU jest znacząco mniejszy w porównaniu do obliczeń wykonanych na CPU, w każdym z rozważanych wariantów symulacji: akcelerator jest szybszy od procesora komputerowego w przybliżeniu sześćdziesięciokrotne zarówno dla problemu o małym jak i znacznym rozmiarze.

Tablica 7.3: Czas analizy rezonatora fotonicznego otrzymany dla PC1 (implementacja jednowątkowa algorytmu FDTD) oraz karty Nvidia GTX 580

	środowisko	wariant A	wariant B	wariant C
S_C	CPU	18,13	20,58	22,08
S_C	GPU	1107	1269	1321
t_1	CPU	340 s [5 min. 40 s]	9324 s [2 h 35 min. 24 s]	52 556 s [14 h 35 min. 56 s]
t_2	GPU	5,58 s	151 s [2 min. 31 s]	879 s [14 min. 39 s]
$\kappa_1 = t_1/340$	CPU	1	27,4	154,6
$\kappa_2 = t_2/5,58$	GPU	1	27,1	157,5
$\kappa_3 = t_1/t_2$	-	**60,9**	**61,7**	**59,8**

Porównanie wydajności implementacji dla wielu procesorów komputerowych w zależności od stopnia zrównoleglenia przedstawia tab. 7.4 oraz tab. 7.5. Pierwsza seria testów, których rezultaty zaprezentowano w tab. 7.4, miała za zadanie wykazanie skalowalności zrównoleglonej implementacji FDTD w postaci jawnej. Skalowalność bliską liniowej otrzymano poprzez przydzielenie do jednego węzła klastra tylko jednego procesu, co dokumentuje rys. 7.7. Porównując tab. 7.3 oraz tab. 7.4 można stwierdzić, że dopiero zastosowanie około 64 węzłów klastra, przy założeniu, że w jednym węźle uruchomiony jest tylko jeden proces (obliczenia w jednym węźle przeprowadzane są z zastosowaniem tylko jednego rdzenia procesora komputerowego), pozwala osiągnąć efektywność obliczeń zmierzoną dla akceleratora graficznego dla wariantów symulacji B oraz C. Wartość efektywności obliczeń, którą zmierzono przy symulacji przeprowadzonej w wariacie A na karcie graficznej, nie jest możliwa do osiągnięcia dla tego wariantu symulacji w stosowanym do obliczeń klastrze. Jest to spowodowane znacznym spadkiem czasu obliczeń przeprowadzonych przez jeden węzeł, który osiąga wartość porównywalną do czasu komunikacji pomiędzy węzłami, co prowadzi do pojawienia się sytuacji, w których procesy czekają na dostarczenie danych w celu przeprowadzenia obliczeń.

Drugim wariantem obliczeń przeprowadzonych na klastrze jest przydzielenie do każdego rdzenia procesorów komputerowych biorących udział w obliczeniach jednego procesu realizującego obliczenia. Pomimo, że wówczas efektywność zrównoleglenia jest niska, co przedstawia rys. 7.8, to efektywność obliczeniowa posiada większą wartość (tab. 7.5) w porównaniu do sytuacji, gdy jeden proces był przydzielony do jednego węzła (tab. 7.4).

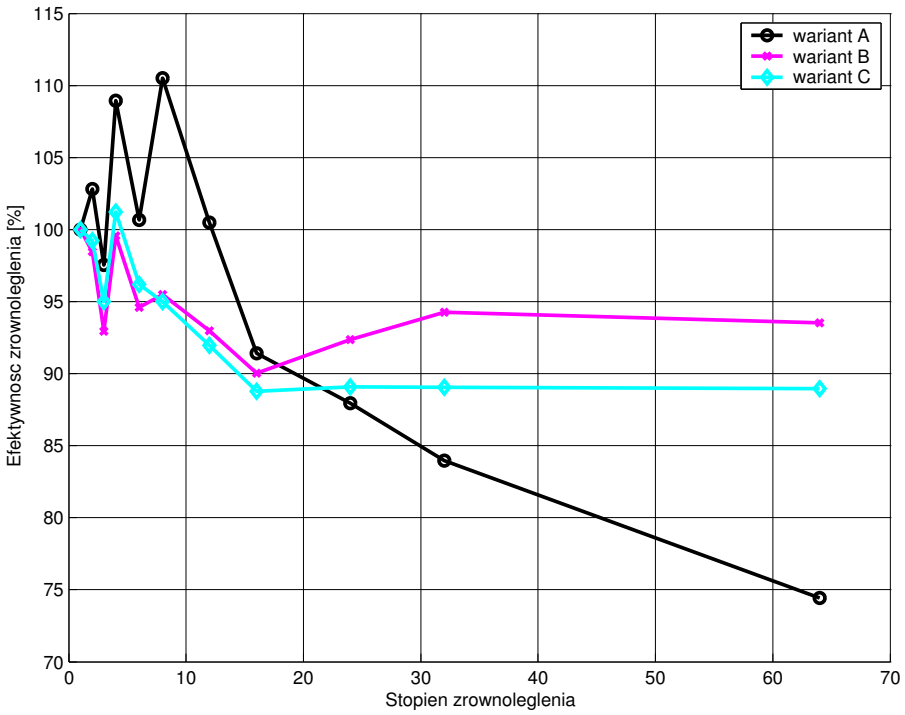
Tablica 7.4: Efektywność obliczeń zmierzona przy analizie rezonatora fotonicznego z uruchomionym jednym procesem na jednym węźle klastra

L.p.	stopień zrównoleglenia	liczba węzłów	S_C [Mcells/s]		
			wariant A	wariant B	wariant C
1	1	1	18	21	22
2	2	2	37	41	44
3	3	3	53	57	63
4	4	4	79	82	89
5	6	6	110	117	127
6	8	8	160	157	168
7	12	12	219	230	244
8	16	16	265	297	314
9	24	24	383	456	472
10	32	32	487	621	629
11	64	64	864	1232	1257
12	80	80	892	1510	1547
13	96	96	786	1695	1751

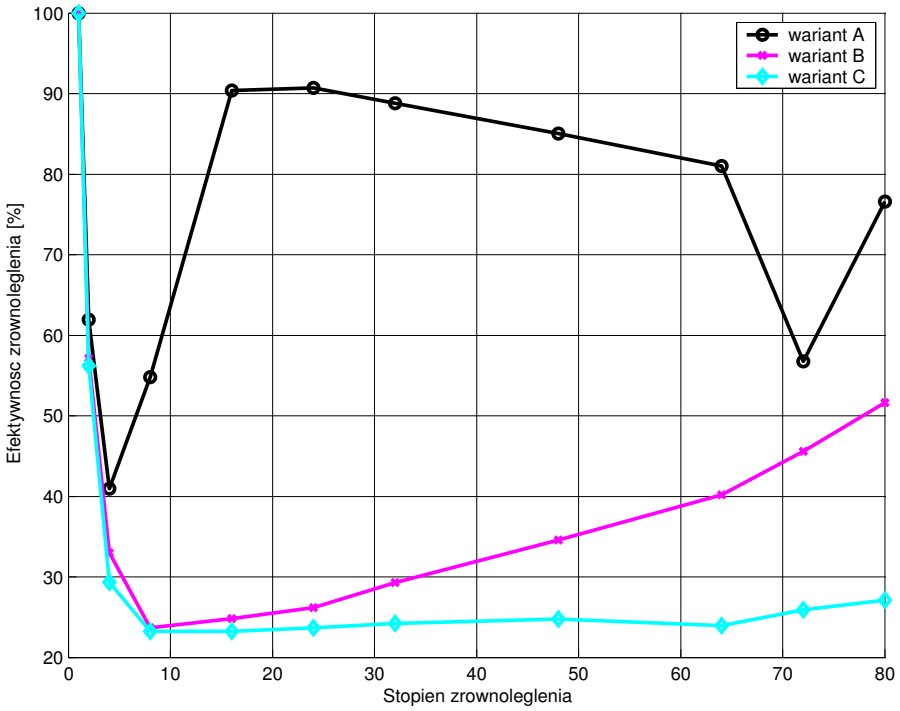
Tablica 7.5: Efektywność obliczeniowa zmierzona przy analizie rezonatora fotonicznego przy liczbie procesów równej liczbie rdzeni procesorów komputerowych

L.p.	stopień zrównoleglenia	liczba węzłów	S_C [Mcells/s]		
			wariant A	wariant B	wariant C
1	1	1	18	21	22
2	2	1	22	24	25
3	4	1	30	27	26
4	8	1	79	39	41
5	16	2	262	82	82
6	24	3	395	129	125
7	32	4	515	193	171
8	48	6	740	342	262
9	64	8	940	530	338
10	72	9	741	675	412
11	80	10	1111	850	479
12	96	12	1334	1163	556
13	128	16	410	1736	791
14	144	18	436	1950	978
15	160	20	557	2160	1106
16	192	24	537	1660	1441

Wynika stąd, iż określenie stopienia zrównoleglenia równego liczbie rdzeni CPU skutecznie prowadzi do zwiększenia wydajności obliczeń, pomimo niskiej skalowalności. Porównując tab. 7.3 oraz tab. 7.5 można stwierdzić, że zastosowanie odpowiednio 10, 13 oraz 22 węzłów obliczeniowych, przy założeniu, że jeden proces został przydzielony do każdego rdzenia CPU, pozwala osiągnąć efektywność obliczeń zmierzoną dla akceleratora graficznego odpowiednio dla wariantu symulacji A, B oraz C.



Rysunek 7.7: Efektywność zrównoleglenia zmierzona przy analizie rezonatora fotonicznego z uruchomionym jednym procesem na jednym węźle klastra



Rysunek 7.8: Efektywność zrównoleglenia zmierzona przy analizie rezonatora fotonicznego przy liczbie procesów równej liczbie rdzeni procesorów komputerowych

Dodatkowo, można zaobserwować, że przy zwiększaniu stopnia zrównoleglenia implementacja algorytmu FDTD z CPML posiada dwie cechy algorytmu FDTD bez CPML, które zostały omówione w p. 3.4:

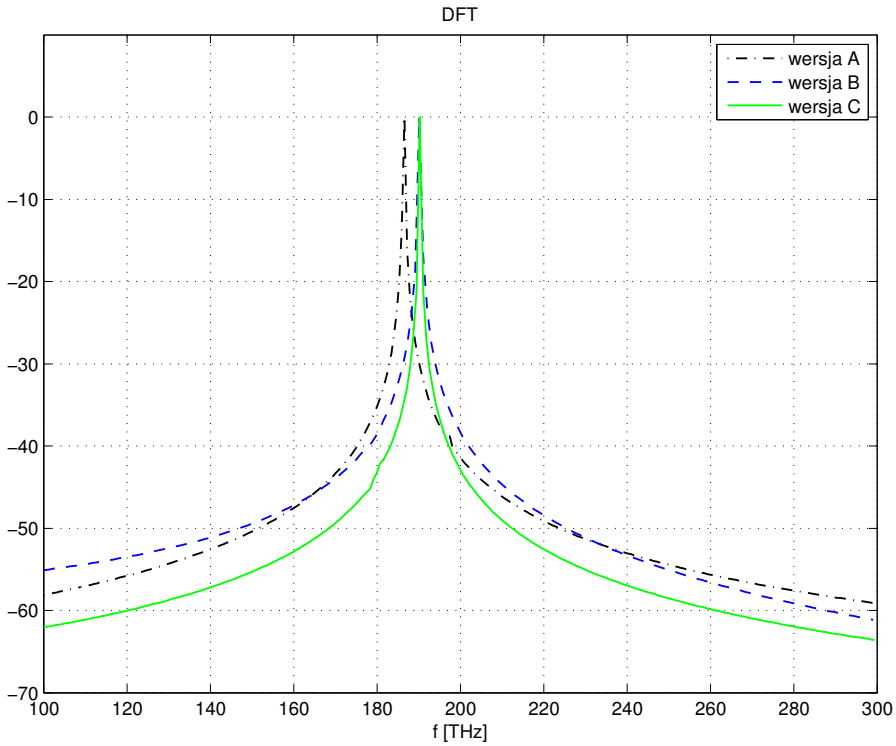
- skalowalność - rys. 7.7 stanowi potwierdzenie, że dla odpowiednio dużych problemów (wariant B i C analizy) struktura algorytmu FDTD z CPML pozwala na osiągnięcie skalowalności bliskiej liniowej. Widoczne przy tym jest, że dla problemów o małych rozmiarach (wariant A analizy) efektywność zrównoleglenia silnie maleje ze wzrostem stopnia zrównoleglenia, co wynika ze wzrostu relacji pomiędzy czasem komunikacji pomiędzy węzłami a czasem obliczeń.
- zależność czasu obliczeń od poziomu transferu danych - wpływ stopnia zrównoleglenia na czas analizy w wariancie A, przedstawiony na rys. 7.8, potwierdza, że skalowalność bliską liniowej można osiągnąć również dla obliczeń przeprowadzonych przy zastosowaniu wszystkich rdzeni w procesorach biorących udział w obliczeniach, jeśli dane wymagane do obliczeń przechowywane są przede wszystkim w pamięci podręcznej procesorów. Wpływ ten jest również widoczny w rezultatach pomiarów dla wariantu B analizy (rys. 7.8), gdzie występuje wzrost efektywności obliczeń przy wzroście stopnia zrównoleglenia, co wynika ze wzrostu procentowego udziału danych przechowywanych w pamięci podręcznej procesorów. Dla problemów o znacznych rozmiarach (wariant C), pomimo przyporządkowania zbioru danych do wielu wątków, nadal wymagają one znacznego obszaru pamięci, a przez to są one przechowywane w pamięci RAM węzłów obliczeniowych. W takiej sytuacji efektywność zrównoleglenia jest na niskim poziomie również dla znacznych wartości stopni zrównoleglenia (rys. 7.8).

Z opisanych pomiarów wynika więc, że przeprowadzenie obliczeń na karcie graficznej, pozwala znacznie skrócić czas symulacji w porównaniu do obliczeń przeprowadzonych na procesorach komputerowych. Przeprowadzenie symulacji problemu o znacznym rozmiarze w tym samym czasie wymaga zastosowania pojedynczego akceleratora graficznego lub **dwudziestu dwóch** węzłów obliczeniowych, które zawierają czterordzeniowe procesory. Akcelerator graficzny jest zatem bardzo ekonomicznym źródłem znacznej mocy obliczeniowej.

Tablica 7.6: Wartości parametrów charakteryzujących strukturę fotoniczną oszacowane dla różnych wariantów analizy

	wariant A	wariant B	wariant C
f [THz]	185,0	190,23	190,29
dobroć Q	1874	1271	1362

Zmniejszenie kroku dyskretyzacji ma bezpośredni wpływ na dokładność szacowania wartości parametrów wyznaczonych w trakcie symulacji. Podstawowymi parametrami, które charakteryzują badaną strukturę są: najniższa częstotliwość rezonansowa oraz dobroć zmierzona dla tej częstotliwości. Są one bezpośrednio związane z widmem odpowiedzi badanego rezonatora (rys. 7.9). Ich wartości dla poszczególnych wariantów symulacji przedstawia tab. 7.6. Zmierzone wartości potwierdzają, iż symulacja wykonana dla siatki najrzadszej charakteryzuje się niską dokładnością odwzorowanego pola, a wartości parametrów wyznaczone na jej podstawie powinny być traktowane jako wartości wstępne



Rysunek 7.9: Widmo odpowiedzi rezonatora fotonicznego dla wariantu A, B i C analizy

do dalszych etapów symulacji. Wartości częstotliwości rezonansowej dla symulacji w wariancie B oraz C różnią się między sobą tylko o 0.03%, jednak rozbieżność w szacowaniu wartości dobroci wynosi 6,5% co uzasadnia zastosowanie gęstej siatki dyskretyzacji w obliczeniach mających na celu dokładne wyznaczenie wartości poszukiwanych parametrów.

7.2.2 Algorytmiczne metody zmniejszenia czasu numerycznej analizy

W p. 7.2.1 wykazano, że czas symulacji przeprowadzonej przy pomocy algorytmu FDTD wzrasta znacząco na skutek zwiększenia gęstości siatki dyskretyzacji oraz że można go zmniejszyć metodami sprzętowymi. W rozdziałach 5 i 6 omówiono alternatywne do sprzętowych, algorytmiczne sposoby redukcji czasu obliczeń. Wymagają one przeprowadzenia modyfikacji algorytmu FDTD w postaci macierzowej. Ponieważ dla implementacji algorytmu FDTD w postaci macierzowej, która nie zawiera żadnych modyfikacji schematu różnicowego, efektywność obliczeń jest niska, co wykazano w rozdziale 4, to korzystne jest takie ujęcie problemu, aby postać macierzowa obejmowała tylko tę część dziedziny obliczeniowej, dla której schemat różnicowy zostanie zmodyfikowany, a pozostała część dziedziny dyskretnej była zarządzana przez implementację algorytmu FDTD w postaci jawnej. Sposób implementacji takiego rozwiązania (tzw. implementacja mieszana) opisano w p. 4.5.

W dalszej części tego punktu badaniom poddano implementację mieszaną algorytmu FDTD przeznaczoną dla akceleratora graficznego. Wybór akceleratora graficznego ma na celu zbadanie dalszych sposobów skrócenia czasu numerycznej analizy struktur mikrofalowych dla środowiska sprzętowego, które jest najefektywniejsze przy najniższym koszcie

jego budowy spośród rozważanych w tej rozprawie rozwiązań. Obszar macierzowy dla analizowanej struktury ustalono na $80 \times 64 \times 17$ oczek siatki Yee. Jego położenie w siatce dyskretyzacji przedstawiono na rys. 7.6. Celem kolejnych modyfikacji schematu różnicowego jest zmniejszenie kroku dyskretyzacji w obszarach, które zawierają otwory w dielektryku, gdyż wówczas zwiększona zostanie dokładność odwzorowania zmienności parametrów materiałowych w przestrzeni dyskretnej. Dla każdego z otworów zdefiniowano makromodel, który lokalnie zwiększa zagęszczenie siatki dyskretyzacji. Rozmiar makromodeli określono na $4 \times 4 \times 8$ oczek siatki Yee, a ich położenie zdefiniowano tak, aby w centralnej części każdego z nich znajdował się obszar z otworem wypełnionym powietrzem. Rozmiar i położenie jednego z makromodeli przedstawiono na rys. 7.6.

Pierwszy test przeprowadzono w celu określenia skali zmniejszenia efektywności obliczeń przy zastosowaniu algorytmu w formie mieszanej. W sytuacji, gdy schemat różnicowy w części macierzowej nie został zmodyfikowany przez wprowadzenie makromodeli do schematu różnicowego, zmierzona wartość efektywności obliczeń wyniosła 754 Mcells/s, co stanowi 32% spadek efektywności w porównaniu do implementacji w postaci jawnej. Wynika stąd, że zastosowanie implementacji FDTD w formie macierzowej może prowadzić do wzrostu efektywności obliczeń w porównaniu do implementacji FDTD w formie jawnej tylko jeśli część macierzowa zawiera modyfikacje algorytmu FDTD, np. makromodele.

W kolejnym etapie prac przeprowadzono pomiary efektywności obliczeń po wprowadzeniu do schematu różnicowego makromodeli z optymalizacją klonowania. Określono przy tym kolejne warianty analizy:

- wariant D - FDTD z makromodelami, rząd modelu równy dwa, stopień zagęszczenia siatki dyskretyzacji równy trzy,
- wariant E - FDTD z makromodelami, rząd modelu równy dwa, stopień zagęszczenia siatki dyskretyzacji równy pięć,
- wariant F - FDTD z makromodelami, rząd modelu równy dwa, stopień zagęszczenia siatki dyskretyzacji równy siedem.

Rezultaty pomiarów efektywności dla w/w wariantów analizy przedstawia tab. 7.7. Zawarte w niej wartości potwierdzają, że wprowadzone zmiany algorytmiczne pozwalają znacząco skrócić czas numerycznej analizy struktury fotonicznej. Można porównać ze sobą efektywność obliczeń zmierzoną przy obliczeniach wykonanych bez modyfikacji algorytmu FDTD (tab. 7.3) oraz z modyfikacją algorytmu FDTD (tab. 7.7), przy zachowaniu takiej samej wartości najmniejszego kroku dyskretyzacji:

- Zastosowanie algorytmu FDTD z makromodelami o trzykrotnym zagęszczeniu siatki dyskretyzacji w porównaniu do wariantu B analizy (implementacja algorytmu FDTD w formie jawnej przeznaczona dla GPU) prowadzi do osiągnięcia wzrostu efektywności obliczeń na poziomie **0,4%**. Wynika stąd, że przy analizie z tak dużą liczbą makromodeli koszt numeryczny wprowadzonych modyfikacji algorytmu FDTD jest na tyle znaczący, że czas analizy badanej struktury w przybliżeniu pozostaje bez zmian.
- Zastosowanie algorytmu FDTD z makromodelami o pięciokrotnym zagęszczeniu siatki dyskretyzacji w porównaniu do wariantu C analizy prowadzi do wzrostu efektywności obliczeń na poziomie **317%**. Osiągnięte przyspieszenie obliczeń jest znaczące i potwierdza celowość wprowadzonych modyfikacji algorytmu FDTD.

- Przeprowadzenie analizy rezonatora fotonicznego przy zastosowaniu algorytmu FDTD z siatką siedmiokrotnie gęstszą od podstawowej nie jest możliwe przy obliczeniach na karcie graficznej Nvidia GTX580 z uwagi na niewystarczające zasoby pamięci. Jednak zastosowanie w obliczeniach makromodeli pozwala na przeprowadzenie obliczeń z siatką dyskretyzacji zagęszczoną siedmiokrotnie w wybranych obszarach dziedziny obliczeniowej. Oszczędność pamięci w obliczeniach stanowi kolejną korzyść zastosowanie makromodeli w schemacie różnicowym. Dodatkowo, osiągnięta efektywność obliczeń S_C wynosi 10444 Mcells/s, co stanowi ponad **dziewięciokrotne** zwiększenie jej wartości w porównaniu do rezultatu pomiaru efektywności obliczeń zmierzonej przy analizie w wariancie A (siatka rzadka, implementacja algorytmu FDTD w formie jawnej przeznaczona dla GPU).

Tablica 7.7: Efektywność obliczeniowa implementacji mieszanej algorytmu FDTD

Rodzaj analizy	S_C	czas symulacji [s]
FDTD z obszarem macierzowym bez modyfikacji	754	8,19
wariant D	1274	42,48
wariant E	4181	70,75
wariant F	10444	99,05

Wprowadzenie modyfikacji do algorytmu FDTD ma na celu przeprowadzenie analizy w krótszym czasie przy odpowiednio wysokim poziomie dokładności obliczeń. Wpływ lokalnego zagęszczenia siatki dyskretyzacji na dokładność obliczeń prezentowano w wielu publikacjach, ([43, 44, 49, 54]), które potwierdzają, że wprowadzenie lokalnego zagęszczenia siatki dyskretyzacji zwiększa dokładność analizy. Na podstawie tab. 7.8 można zauważyć, że wprowadzenie lokalnego zagęszczenia siatki dyskretyzacji prowadzi do wzrostu dokładności szacowania wartości częstotliwości rezonansowej badanej struktury. Wartości te są zbliżone do rezultatów pomiarów otrzymanych dla algorytmu FDTD bez modyfikacji przy adekwatnych wartościach kroków dyskretyzacji (tab. 7.6). Odchyłka częstotliwości przy porównaniu wariantów B i D analizy wynosi 0,44%, natomiast przy porównaniu wariantów C i E wynosi ona 0,04%. Podane wartości potwierdzają, że zastosowanie makromodeli w analizie pozwala poprawnie oszacować wartość częstotliwości rezonansowej badanej struktury. Natomiast z porównania tab. 7.6 i tab. 7.8 można stwierdzić, że oszacowanie wartości dobroci rezonatora po wprowadzeniu makromodeli do schematu różnicowego obarczone jest znacznym błędem (na poziomie ok. 50%). Wynika stąd, że opracowany dotychczas algorytm lokalnego zagęszczenia siatki dyskretyzacji wymaga dalszych usprawnień. Dalsza poprawa dokładności odwzorowania propagacji pola elektromagnetycznego po wprowadzeniu do schematu różnicowego lokalnego zagęszczenia siatki dyskretyzacji oraz makromodeli stanowi jeden z celów pracy Jakuba Podwalskiego, [75, 80, 77] i z tego powodu nie jest przedmiotem dalszych analiz w tej rozprawie.

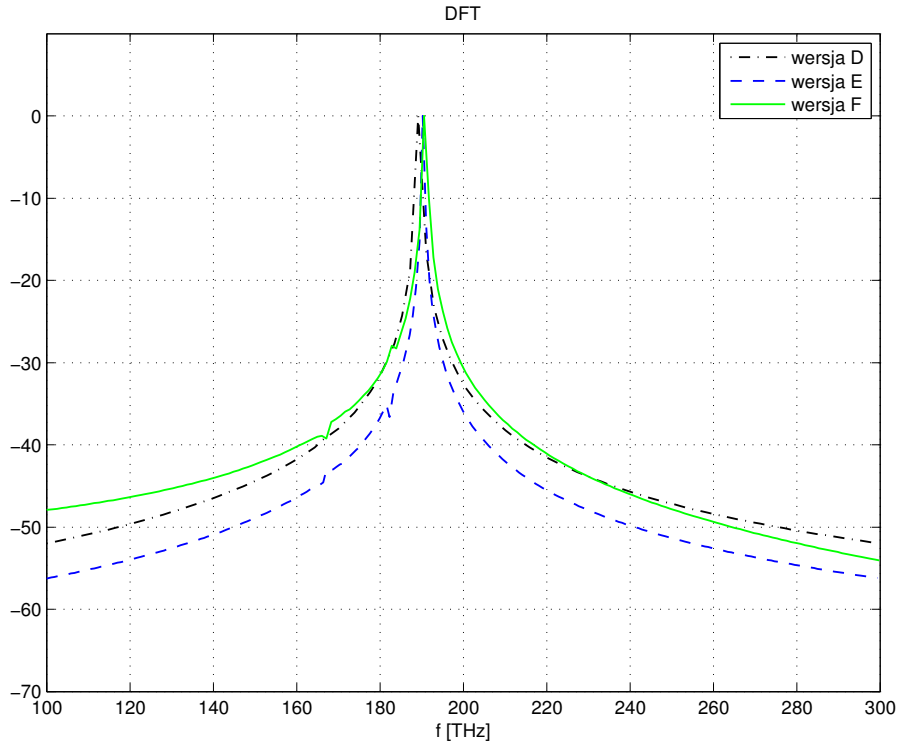
Na podstawie rys. 7.10 można stwierdzić, że po wprowadzeniu do obliczeń makromodeli widmo odpowiedzi rezonatora fotonicznego pozostaje w dużej zgodności z rezultatami otrzymanymi w symulacjach FDTD bez modyfikacji (rys. 7.9).

7.2.3 Porównanie metod skrócenia czasu numerycznej analizy rezonatora fotonicznego

Praktyka projektowania układów mikrofalowych składnia do porównania czasu symulacji działania rezonatora fotonicznego w zadanym przedziale czasowym. Długość odcinka

Tablica 7.8: Wartości parametrów charakteryzujących strukturę fotoniczną oszacowane dla różnych wariantów analizy (FDTD z makromodelami)

	wariant D	wariant E	wariant F
f [THz]	189,4	190,37	190,85
dobroć Q	856	740	693



Rysunek 7.10: Widmo odpowiedzi rezonatora fotonicznego dla wariantów D, E i F analizy

czasu ustalono na $t_s = 14000\Delta_{tA}$, przy czym Δ_{tA} oznacza krok czasowy zastosowany w symulacji w wariancie A. Wartości kroków czasowych i ich relację w odniesieniu do Δ_{tA} zawiera tab. 7.9.
Skrócenie czasu obliczeń można scharakteryzować poprzez współczynnik C_s, r. 7.1.

$$t_2 = t_1 \left(1 - C_s(t_1, t_2)\right) \tag{7.1a}$$

$$C_s(t_1, t_2) = \frac{t_2 - t_1}{t_1} \tag{7.1b}$$

gdzie:
t_1 - czas przeprowadzenia symulacji odniesienia,
t_2 - czas przeprowadzenia symulacji po zastosowaniu metod skrócenia czasu symulacji.

Na podstawie tab. 7.9 można określić przyspieszenie obliczeń oraz skrócenie czasu obliczeń. Wyznaczone wartości umieszczono w tab. 7.10. Przyspieszenie obliczeń rezonatora fotonicznego po zastosowaniu akceleratora graficznego zamiast procesora komputerowego wynosi 60 (l.p. $1 \div 3$ z tab. 7.10). Wprowadzenie do schematu różnicowego makromodeli, przy obliczeniach przeprowadzanych na karcie graficznej, prowadzi do dalszego skrócenia

Tablica 7.9: Zestawienie czasów symulacji rezonatora fotonicznego dla różnych wariantów symulacji

wariant symulacji	środowisko	Δ_t [as]	Δ_{tA}/Δ_t	liczba iteracji	czas symulacji [s]
A	CPU	191,6	1,00	14000	$t_{Ac} = 340$
A	GPU	191,6	1,00	14000	$t_{Ag} = 5,58$
B	CPU	63,88	3,00	42000	$t_{Bc} = 9324$
B	GPU	63,88	3,00	42000	$t_{Bg} = 151$
C	CPU	38,33	5,00	70000	$t_{Cc} = 52556$
C	GPU	38,33	5,00	70000	$t_{Cg} = 879$
D	GPU	135,0	1,42	19890	$t_D = 20,10$
E	GPU	110,6	1,73	24251	$t_E = 24,51$
F	GPU	98,39	1,95	27268	$t_F = 27,56$

czasu analizy w odniesieniu do symulacji FDTD bez modyfikacji, z krokiem dyskretyzacji o wartości równej tej zastosowanej w makromodelach. W zależności od wariantów porównywanych symulacji przyspieszenie obliczeń jest siedmiokrotne (warianty B i D, l.p. 4 z tab. 7.10) lub 36-krotne (warianty C i E, l.p. 5 z tab. 7.10). Sumaryczne przyspieszenie obliczeń, które uwzględnia zastosowanie sprzętowej oraz algorytmicznej metody skrócenia czasu obliczeń, w odniesieniu do symulacji przeprowadzonej na procesorze komputerowym, wynosi 464 przy porównaniu wariantów B i D symulacji oraz 2144 przy porównaniu wariantów C i E. Skala redukcji czasu symulacji odzwierciedla znaczną korzyść z wprowadzania do schematu różnicowego makromodeli oraz stosowania w obliczeniach akceleratorów graficznych.

Tablica 7.10: Skrócenie czasu obliczeń oraz przyspieszenie obliczeń przy symulacji FDTD w różnych wariantach

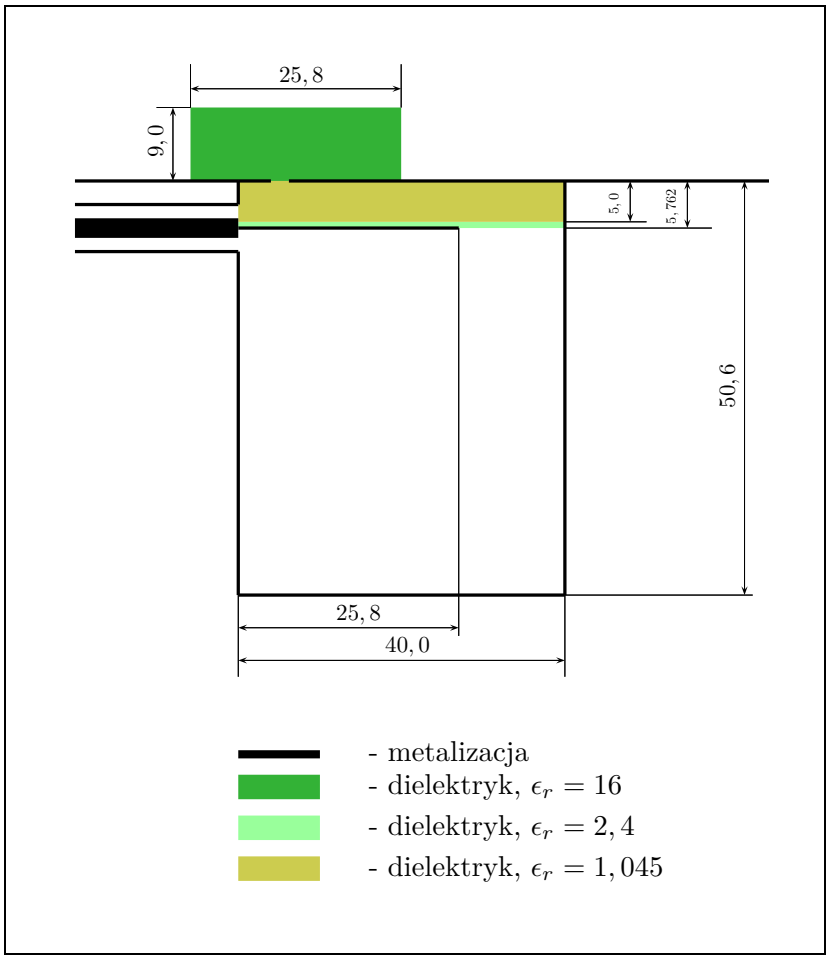
L.p.	identyfikacja symulacji	skrócenie czasu obliczeń [%]	przyspieszenie obliczeń
1	$C_s(t_{Ac}, t_{Ag})$	98,36	60,9
2	$C_s(t_{Bc}, t_{Bg})$	98,38	61,8
3	$C_s(t_{Cc}, t_{Cg})$	98,33	59,8
4	$C_s(t_{Bg}, t_{Dg})$	86,69	7,51
5	$C_s(t_{Cg}, t_{Eg})$	97,21	35,9
6	$C_s(t_{Bc}, t_{Dg})$	99,78	464
7	$C_s(t_{Cc}, t_{Eg})$	99,95	2144

7.3 Antena mikrofalowa

W p. 7.2 udokumentowano, że wprowadzenie do schematu różnicowego makromodeli prowadzi do znacznego wzrostu efektywności obliczeń pod warunkiem, że siatka dyskretyzacji w obszarze poddziedziny jest co najmniej pięciokrotnie gęstsza w porównaniu do siatki podstawowej. W tym punkcie, na przykładzie analizy anteny mikrofalowej [42], zademonstrowano, że wprowadzenie makromodeli może prowadzić do zwiększenia efektywności

obliczeń nawet bez zastosowania zagęszczenia siatki dyskretyzacji. Warunkiem jest przy tym odpowiednio mała liczba zmiennych, które stanowią pobudzenie wyodrębnionej poddziedziny.

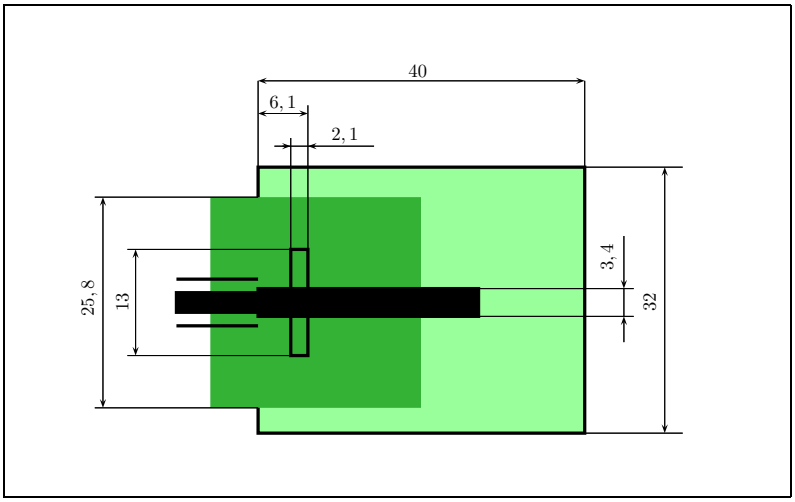
Wymiary analizowanej anteny przedstawiono na rys. 7.11 oraz rys. 7.12. Pobudzenie anteny doprowadzone jest przez linię TEM o kołowym przekroju przewodu centralnego (promień równy jest $1,2mm$) oraz kwadratowym przekroju metalizacji zewnętrznej (długość boku wynosi $6mm$). Kształt pobudzenia w przekroju linii TEM stanowi rozkład modu podstawowego linii.



Rysunek 7.11: Wymiary badanej anteny mikrofalowej. Przekrój pierwszy.

Wyznaczenie parametrów rozproszenia anteny przeprowadzono z zastosowaniem algorytmu przedstawionego w [32]. Zwiększenie dokładności odwzorowania propagacji pola elektromagnetycznego na granicy dielektryków oraz metalizacji uzyskano poprzez wprowadzenie wartości efektywnych współczynników przenikalności elektrycznej oraz magnetycznej, zgodnie z algorytmami opisanymi w [81].

Sposób dyskretyzacji przestrzeni ciągłej obrazują rys. 7.13 i rys. 7.14. Wartości kroków dyskretyzacji dla kierunków x, y, z wynoszą odpowiednio: $dx = 0,57143mm$, $dy = 2,00000mm$, $dz = 0,57620mm$. Rozmiar przestrzeni dyskretnej równy jest $80 \times 64 \times 114$ oczek siatki Yee, natomiast długość wektora stanu dla dziedziny ciągłej wynosi



Rysunek 7.12: Wymiary badanej anteny mikrofalowej. Przekrój drugi.

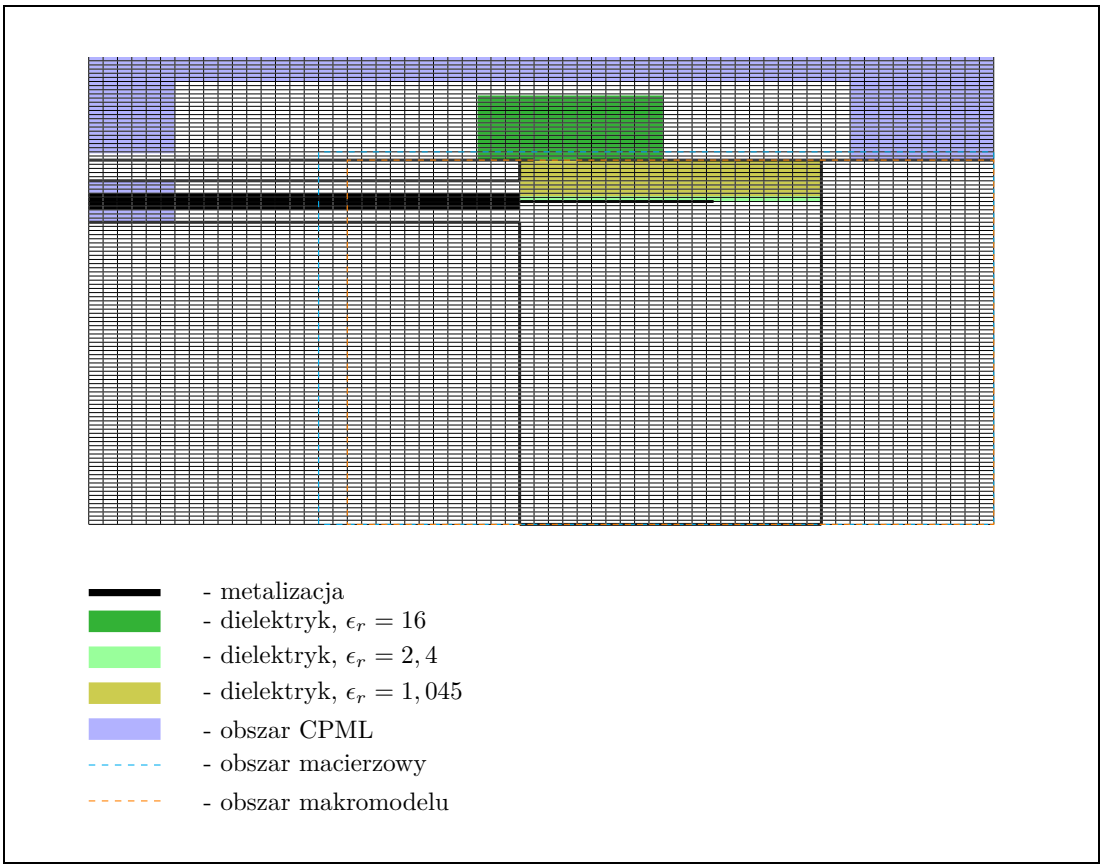
$3IJK = 1,75 \cdot 10^6$. Ponieważ znaczna część dziedziny dyskretnej stanowi obszar, w którym nie występuje propagacja fali elektromagnetycznej, to wskazana jest modyfikacja algorytmu, tak aby wartości próbek pól były wyznaczane tylko w określonych obszarach przestrzeni dyskretnej. Implementacja taka jest łatwa do realizacji w algorytmie macierzowym, w którym odpowiednie wiersze oraz kolumny macierzy rotacji zostają usunięte. Zastosowanie tej procedury doprowadziło do zmniejszenia długości wektora próbek pola elektrycznego i magnetycznego do odpowiednio 626077 oraz 642969 elementów.

Symulację propagacji fali elektromagnetycznej w badanej antenie przeprowadzono dla dwóch środowisk sprzętowych: procesora komputerowego AMD Opteron 6174 oraz karty graficznej Nvidia GTX 580. Rezultaty pomiaru czasu analizy badanej anteny mikrofalowej dla różnych wariantów implementacji algorytmu FDTD umieszczono w tab. 7.11. Przedstawione w niej dane dokumentują, iż czas analizy wykonanej na w środowisku Matlab jest 121 krotnie dłuższy w porównaniu do czasu analizy wykonanej na akceleratorze graficznym, przy zachowaniu tej samej precyzji obliczeń. W sytuacji, gdy obliczenia na akceleratorze graficznym wykonywane są w pojedynczej precyzji, czas obliczeń w Matlabie jest ponad 213 krotnie dłuższy. Skrócenie czasu analizy jest zatem znaczące.

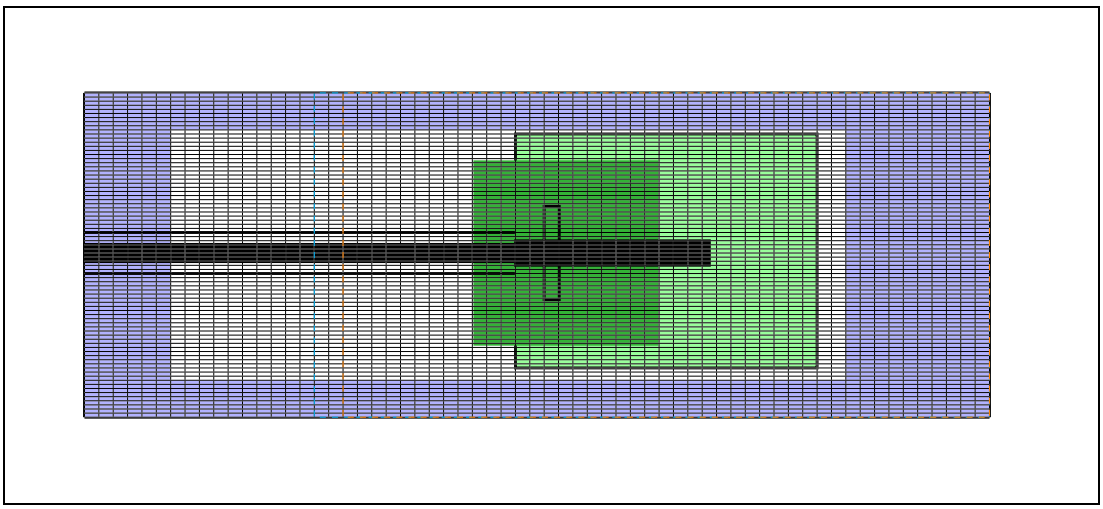
Tablica 7.11: Czasy symulacji anteny mikrofalowej przy pomocy metody FDTD

	algorytm FDTD bez modyfikacji	algorytm FDTD z makromodelem
Matlab	1574 s	1160 s
GPU double	13,01 s	7,47 s
GPU float	7,38 s	4,87 s

Dalsze skrócenie czasu obliczeń osiągnięto poprzez wprowadzenie do obliczeń makromodelu. Poddziedzina poddana redukcji liczby zmiennych obejmuje wnękę rezonansową, co ilustrują rys. 7.13 i 7.14. Rozmiar dziedziny dyskretnej poddawanej redukcji wyniósł 290 842. Ponieważ granicę makromodelu wyznaczono w przekroju linii TEM oraz w szczelinie wnęki rezonansowej, to liczba portów, które stanowią pobudzenie makromodelu została znacznie zredukowana w porównaniu do teoretycznych wartości wymaganych przez



Rysunek 7.13: Ilustracja dziedziny dyskretnej zastosowanej w analizie anteny mikrofalowej. Przekrój pierwszy.



Rysunek 7.14: Ilustracja dziedziny dyskretnej zastosowanej w analizie anteny mikrofalowej. Przekrój drugi.

poddziedzinę o takich rozmiarach. Liczba portów wynosiła 174, co w połączeniu z zastosowaniem rzędu modelu równego pięć, prowadzi do rozmiaru wektora pola magnetycznego po redukcji: 870. Redukcja liczby zmiennych stanu jest zatem na poziomie 99,70%. Należy przy tym podkreślić, że liczba operacji wykonywanych dla pojedynczego elementu wektora zmiennych stanu po redukcji jest znacznie większa w porównaniu do liczby operacji wykonywanych na pojedynczym elemencie wektora zmiennych stanu przed redukcją. Wynika to przede wszystkim z konieczności przeprowadzenia w każdej iteracji algorytmu FDTD dwukrotnego mnożenia macierzy gęstej o rozmiarze 870×174 przez wektor zmiennych stanu po redukcji. Z przeprowadzonych badań wynika, że koszt numerycznych tych operacji stanowi zdecydowaną większość wśród działań stanowiących implementację makromodelu w schemacie różnicowym.

Po wprowadzeniu makromodelu do schematu różnicowego, dla każdego z badanych środowisk obliczeniowych, zaobserwowano skrócenie czasu analizy. Wynosiło ono od 26,3% dla środowiska Matlab do 42,6% dla obliczeń wykonanych na GPU na zmiennych o podwójnej precyzji (tab. 7.12). Liczby te stanowią potwierdzenie, że wprowadzenie do schematu różnicowego makromodeli, które nie zawierają lokalnego zagęszczenia siatki dyskretyzacji, może prowadzić do skrócenia czasu analizy badanej struktury.

Tablica 7.12: Relacje pomiędzy czasami symulacji anteny mikrofalowej w zależności od rodzaju jednostki obliczeniowej

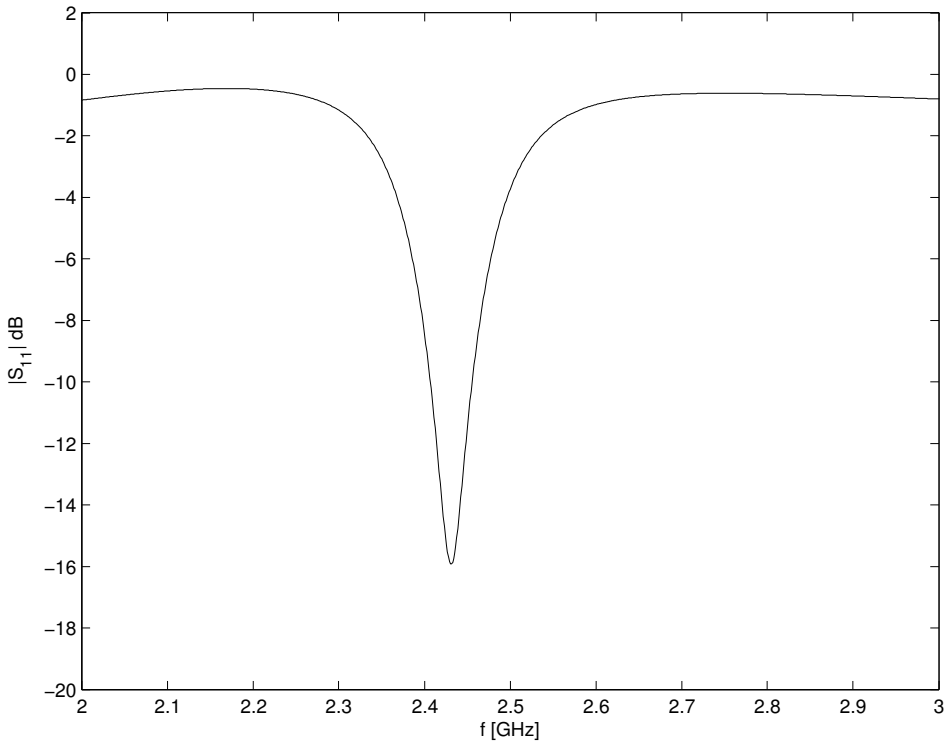
	stopień skrócenia czasu symulacji po wprowadzeniu do FDTD makromodelu	relacja do czasu analizy GPU float	
		algorytm FDTD bez makromodelu	algorytm FDTD z makromodelem
Matlab	26,30 %	213,3	238,2
GPU double	42,58 %	1,76	1,53
GPU float	34,01 %	1	1

Iloraz czasu symulacji przeprowadzonej z zastosowaniem algorytmu FDTD w środowisku Matlab (1574 s) do czasu symulacji przeprowadzonej z zastosowaniem algorytmu FDTD z makromodelem na karcie graficznej (4,87 s) określa sumaryczne przyspieszenie obliczeń po zastosowaniu algorytmicznych i sprzętowych metod skrócenia czasu symulacji, które wynosi 323.

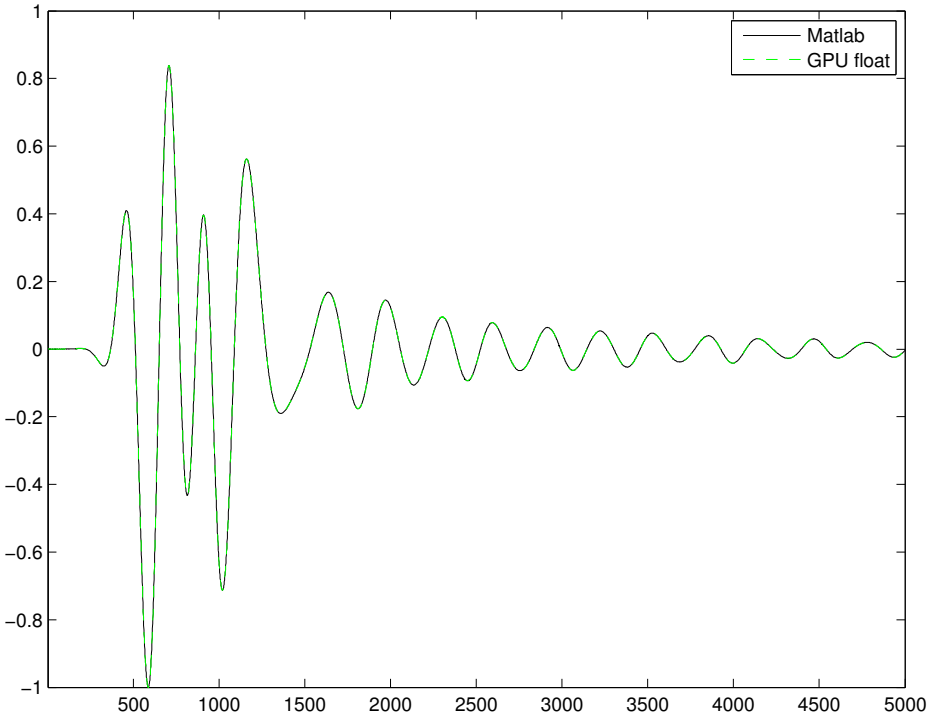
7.3.1 Dokładność analizy

Końcowy rezultat analizy anteny stanowi wykres współczynnika odbicia obserwowanego z linii TEM w kierunku anteny, który przedstawiono na rys. 7.15. Minimalna wartość tego współczynnika wynosi -16dB i występuje ona dla częstotliwości 2,43 GHz. W porównaniu do rezultatów symulacji przedstawionych w [42], gdzie minimum wyznaczono dla częstotliwości 2,39 GHz, różnica położenia minimum jest na poziomie 1,76%. Rozbieżność tą można tłumaczyć brakiem uwzględnienia strat występujących w dielektrykach. Uwzględnienie strat wymuszałoby dalsze rozbudowanie opracowanego programu.

Ważnym aspektem, który należy rozpatrzyć przy ocenie dokładności symulacji, stanowi ocena wpływu wprowadzenia makromodelu do analizy oraz przeprowadzenie symulacji za pomocą zmiennych o pojedynczej precyzji. Rys. 7.16 przedstawia porównanie sygnału pobieranego w trakcie symulacji w Matlabie (obliczenia w podwójnej precyzji, FDTD bez modyfikacji) wraz z obliczeniami wykonanymi w pojedynczej precyzji na



Rysunek 7.15: Współczynnik odbicia zmierzony na wejściu badanej anteny



Rysunek 7.16: Porównanie sygnału pobieranego w trakcie symulacji w Matlabie oraz na GPU z obliczeniami wykonanymi w pojedynczej precyzji

GPU przy włączonym do algorytmu FDTD makromodelem. Jak można zauważyć wpływ na dokładność obliczeń jest na bardzo niskim poziomie, co potwierdza, że obliczenia na zmiennych o pojedynczej precyzji oraz włączenie makromodeli do schematu różnicowego ma pomijalnie mały wpływ na dokładność obliczeń wykonanych przez implementację algorytmu FDTD.

7.4 Podsumowanie

Na podstawie przedstawionych rezultatów pomiarów można stwierdzić, że akceleratory graficzne stanowią znacznie wydajniejsze źródło mocy obliczeniowej w porównaniu do procesorów komputerowych. W analizie dla dziedziny obliczeniowej o znacznym rozmiarze dopiero przy zastosowaniu dwudziestu dwóch komputerów z czterordzeniowymi procesorami została zmierzona efektywność obliczeń równa tej otrzymanej dla akceleratora graficznego.

Dalszy wzrost efektywności można osiągnąć poprzez wprowadzenie modyfikacji algorytmu FDTD. Jak wykazano efektywność algorytmu z klonowaniem makromodeli pozwala zwiększyć dziewięciokrotnie efektywność obliczeń w porównaniu do implementacji algorytmu FDTD w podstawowej wersji przeznaczonej dla GPU. W rezultacie osiągnięto sumaryczne przyspieszenie obliczeń, po zastosowaniu algorytmicznych i sprzętowych metod skrócenia czasu symulacji FDTD, równe 323 przy analizie anteny mikrofalowej oraz 2144 przy analizie rezonatora fotonicznego.

Wykazano zatem, że akcelerator graficzny stanowi bardzo konkurencyjne źródło mocy obliczeniowej, które można zastosować również w algorytmach o znacznym poziomie złożoności.

ROZDZIAŁ 8

Podsumowanie

W rozprawie zademonstrowano silną zależność efektywności obliczeń zmierzonej dla implementacji algorytmu FDTD od rozmiaru problemu oraz od zastosowanych do obliczeń zasobów sprzętowych. Przedstawiono podstawowe czynniki, które wpływają na taki stan rzeczy, czyli przede wszystkim sposób transferu danych z pamięci do jednostki obliczeniowej, liczbę jednostek obliczeniowych oraz występujący między nimi sposób komunikacji. Udowodniono, że efektywność obliczeń zmierzona dla akceleratorów graficznych posiada znacznie większą wartość w porównaniu do efektywności obliczeń zmierzonej dla procesora komputerowego, a także jest porównywalna do efektywności otrzymanej dla klastra złożonego z dwudziestu dwóch komputerów, z których każdy posiada dwa czterordzeniowe procesory (przy symulacji przeprowadzonej dla dziedziny obliczeniowej znacznych rozmiarów i z warstwą absorpcyjną). Biorąc pod uwagę koszty takiego klastra, akceleratory graficzne stanowią znacznie tańsze źródło mocy obliczeniowej.

Ponadto, wykazano w jaki sposób należy zmodyfikować implementację algorytmu przeznaczonego dla procesorów komputerowych w celu otrzymania efektywnej implementacji przeznaczonej dla akceleratorów graficznych. Przedstawiono podstawowe różnice w koncepcji zrównoleglania algorytmu dla klastra oraz akceleratora graficznego, a ich wpływ na efektywność obliczeń został zobrazowany na przykładach wielu testów numerycznych. Udowodniono również, że architektura akceleratorów graficznych pozwala na szybkie przetwarzanie również złożonych obliczeń, których przykład stanowi implementacja algorytmu FDTD z włączonym klonowaniem makromodeli.

W przedstawionej rozprawie omówiono następujące oryginalne rozwiązania własne:

- implementację algorytmu FDTD w postaci jawnej przeznaczoną do uruchomienia na kartach graficznych napisaną przy zastosowaniu technologii CUDA (p. 3.5.3), której budowa posiada cechy znacząco je różniące od implementacji algorytmu FDTD przeznaczonej dla CPU,

- implementację algorytmu FDTD w postaci macierzowej zarówno z jak i bez modyfikacji schematu różnicowego z przeznaczeniem do uruchomienia na procesorach komputerowych oraz (osobna implementacja) z akceleratorami graficznymi zgodnymi z technologią CUDA (rozdziały 4 i 5),
- opracowanie i implementacja dla akceleratorów graficznych algorytmu FDTD w postaci mieszanej, która łączy w sobie implementację jawną i macierzową (opis w p. 4.5, rezultaty testów w rozdziale 7),
- wykazanie znacznej zależności efektywności obliczeń algorytmu FDTD uruchomionego w środowisku wielordzeniowych procesorów komputerowych od dostępnej liczby kontrolerów pamięci w danym zestawie komputerowym oraz od dostępnego maksymalnego transferu danych pomiędzy procesorem a pamięcią RAM (p. 3.4),
- opracowanie modyfikacji algorytmu FDTD z klonowaniem makromodeli, która prowadzi do dalszego zwiększenia efektywności obliczeń (rozdział 6).

W rozdziale 7 wykazano, że sumaryczne zastosowanie sprzętowej oraz algorytmicznej metody skrócenia czasu numerycznej analizy zagadnień elektromagnetycznych może prowadzić do 2144-krotnego przyspieszenia obliczeń.

Przewidywane kierunki rozwoju

Firma Nvidia dąży do tego, aby akceleratory graficzne stanowiły coraz bardziej powszechne źródło znacznej mocy obliczeniowej również w superkomputerach. Plany rozwoju firmy Nvidia przewidują, że następna generacja kart graficznych posiadać będzie akceleratory graficzne z architekturą Kepler, która będzie się charakteryzować, w porównaniu do architektury Fermi, czterokrotnie mniejszym zapotrzebowaniem na energię elektryczną w celu otrzymania takiej samej mocy obliczeniowej. Nvidia zapowiedziała również, że następczyni architektury Kepler, architektura Maxwell, również czterokrotnie zmniejszy swoje zapotrzebowanie na energię w porównaniu do poprzedniczki. Rozpatrując obrany kierunek rozwoju kart graficznych można założyć, że w niedalekiej przyszłości znaczna moc obliczeniowa większości superkomputerów pochodzić będzie z kart graficznych. Z tego względu naturalnym rozwojem implementacji algorytmu FDTD przeznaczonej dla GPU powinno być dostosowanie jej do środowiska z wieloma akceleratorami graficznymi. Już opisywane są sukcesy w tej dziedzinie, [72], zatem jest to perspektywiczny kierunek rozwoju badań.

Ponieważ akceleratory graficzne są silnie rozwijane przez firmy Nvidia i ATI, to ich zakres zastosowań w elektrodynamice obliczeniowej będzie wzrastał. Już w chwili obecnej dostępne są karty graficzne, które pozwalają wykonywać obliczenia z podwójną precyzją obliczeń, a przez to stanowią one platformę sprzętową odpowiednią do przeprowadzenia obliczeń, w których wymagana jest duża dokładność numeryczna, czyli np. w rozwiązywaniu układu równań liniowych o znacznych rozmiarach, czy wyznaczaniu wartości własnych macierzy o znacznych rozmiarach. W związku z tym możliwe jest przeprowadzenie badań efektywności obliczeń algorytmu różnicowego w dziedzinie częstotliwości.

Udowodniono możliwość osiągnięcia znacznej mocy obliczeniowej w algorytmie FDTD oraz wskazano metody budowania efektywnej implementacji tego algorytmu. Kolejny etap rozwoju może stanowić sprawdzenie dostosowania innych algorytmów z dziedziny elektrodynamiki obliczeniowej do architektury akceleratorów graficznych.

DODATEK A

Istotne cechy algorytmu FDTD

A.1 Warunki początkowe analizy FDTD

Ponieważ FDTD jest metodą iteracyjną, która symuluje propagację fali elektromagnetycznej w czasie, istotną rolę spełnia sposób pobudzenia fali elektromagnetycznej w dziedzinie dyskretnej. W celu opisania pobudzenia należy uwzględnić dwa podstawowe kryteria:

- miejsce pobudzenia,
- przebieg czasowy pobudzenia.

W metodzie FDTD widmo częstotliwościowe sygnału pobudzenia zwykle jest dostosowane do przewidywanego częstotliwościowego pasma pracy analizowanej struktury. W analizach numerycznych zaprezentowanych w tej rozprawie odpowiednie własności widma sygnału pobudzenia zostały osiągnięte poprzez określenie jego przebiegu w kształcie zmodulowanego impulsu Gaussa. Sposób budowania wektora próbek tak określonego pobudzenia przedstawia r. (A.1). Elementy tego wektora są pobierane w kolejnych iteracjach algorytmu FDTD i służą do modyfikacji wartości wybranych próbek wektora **e** i/lub **h**.

$$src(i) = sin(2\pi f_{mod}\Delta_t i)e^{-\frac{(t_d-i)^2}{2w^2}} \tag{A.1}$$

przy czym
f_{mod} - częstotliwość sygnału, który moduluje impuls Gaussa
t_d - przesunięcie impulsu Gaussa w czasie
w - decyduje o szerokości impulsu Gaussa

W analizach rezonatorów przeprowadzonych w tej rozprawie pobudzenie zawsze wprowadzane jest do wektora próbek pola magnetycznego **h**, jednak w ogólności możliwe jest

również jego wprowadzenie do wektora **h**. Pełną informację o miejscu pobudzenia jest reprezentowana przez zbiór indeksów $A_e = \alpha_1, \alpha_2, \cdots, \alpha_n$ oraz $A_h = \beta_1, \beta_2, \cdots, \beta_m$, które wskazują na elementy wektorów **e** oraz **h**, które będą modyfikowane w procesie wprowadzania pobudzenia.

Wyróżnia się dwa sposoby wprowadzania pobudzenia do struktury [105]:

- pobudzenie miękkie

$$\mathbf{e}^n(A_e) = \mathbf{e}^n(A_e) + src(n) \tag{A.2}$$

- pobudzenie twarde

$$\mathbf{e}^n(A_e) = src(n) \tag{A.3}$$

Pobudzenie stosowane w testach numerycznych opisanych w tej rozprawie jest zawsze pobudzeniem miękkim.

A.2 Stabilność algorytmu FDTD

Algorytm FDTD jest algorytmem warunkowo stabilnym, co oznacza, że w celu zagwarantowania skończonej wartości każdej próbki pola przy zastosowaniu wektora pobudzenia o skończonej wartości każdego elementu, należy wyznaczyć odpowiednią wartość kroku dyskretyzacji Δ_t w dziedzinie czasu. Minimalna wartość Δ_t, która zapewnia stabilność algorytmu FDTD, jest określana na podstawie warunku stabilności.

Dla dyskretnej postaci równań Maxwella i przy rozpatrywaniu przestrzeni trójwymiarowej z krokiem dyskretyzacji $\Delta = \Delta_x = \Delta_y = \Delta_z$ warunek stabilności nazywany jest, od nazwisk jego twórców, warunkiem Courant-Friedrichs-Lewy - w skrócie warunkiem CFL, [21], [20]. Ma on postać r. (A.4),

$$\Delta_t \leq \frac{\Delta}{\sqrt{3} v_{max}} \tag{A.4}$$

przy czym v_{max} oznacza maksymalną prędkość fali elektromagnetycznej w obszarze dziedziny obliczeniowej.

Jeśli krok dyskretyzacji przestrzeni jest różny dla każdego kierunku, to warunek stabilności przyjmuje postać r. (A.5), [105].

$$\Delta_t \leq \frac{1}{v_{max}\sqrt{\frac{1}{\Delta_x^2} + \frac{1}{\Delta_y^2} + \frac{1}{\Delta_z^2}}} \tag{A.5}$$

Warunek stabilności zdefiniowany dla zagadnienia macierzowego jest bardziej rozbudowany, ponieważ r. (2.23) reprezentuje szerszą grupę problemów numerycznych. Dokładniejszy opis tego warunku można znaleźć w pracach [8], [81]. Warunki stabilności dla algorytmu FDTD zapisanego w formie macierzowej zawarte są w r. (A.6). Należy przy tym podkreślić, że wiele modyfikacji algorytmu FDTD można zapisać w formie r. (2.23) (m.in. lokalne zagęszczenie siatki oraz makromodele, które zostały opisane w rozdziałach 5 i 6 tej rozprawy) i również dla nich podany warunek stabilności jest słuszny.

$$\forall_{\lambda_L \in L} \ \lambda_L \in \mathbb{R}, \qquad \forall_{\lambda_L \in L} \ \lambda_L \geq 0, \qquad \Delta_t \leq \frac{2}{\sqrt{\lambda_{L\,max}}} \tag{A.6}$$

gdzie
L - zbiór wartości własnych macierzy $\mathbf{L_h} = -\mathbf{R}_E\mathbf{R}_H$ lub macierzy $\mathbf{L_e} = -\mathbf{R}_H\mathbf{R}_E$ (zbiory wartości własnych macierzy $\mathbf{L_h}$ oraz $\mathbf{L_e}$ są identyczne),
$\mathbb{R}$ - zbiór liczb rzeczywistych,
$\lambda_{L\,max}$ - maksymalna wartość własna ze zbioru L.

A.3 Warunki brzegowe

Istotnym zagadnieniem przy konstrukcji algorytmu FDTD są warunki brzegowych na krańcach dziedziny obliczeniowej, czyli dla $i = 0$, $i = I - 1$, $j = 0$, $j = J - 1$, $k = 0$, $k = K - 1$. Konieczność ich zastosowania wynika z r. (2.11) (np. poprawne wyznaczenie wartości pola $E_x(0, 0, 0)$ wymaga znajomości pól $H_y(0, 0, -1)$ oraz $H_z(0, -1, 0)$, które w algorytmie FDTD nie są zdefiniowane).

W analizach numerycznych prezentowanych w tej rozprawie występuje jeden z dwóch rodzai warunków brzegowych:

1. Najprostszym w implementacji warunkiem brzegowym jest zastosowanie na krańcach dziedziny warunku idealnego przewodnika, gdyż wystarczy wyzerować próbki pól elektrycznych, które są styczne do płaszczyzn $i = 0$, $j = 0$, $k = 0$, $i = I$, $j = J$, $k = K$.
2. Popularnym warunkiem brzegowym, stosowanym do symulacji otwartej przestrzeni, jest PML (ang. *Perfectly Matched Layer*) wprowadzony przez Berengera, [14]. W testach numerycznych opisanych w tej pracy zastosowano zmodyfikowaną wersję tego algorytmu w postaci CPML zaprezentowanej w [86].

A.4 Dokładność algorytmu FDTD

Dokładność analizy problemu uzyskanego za pomocą metody FDTD jest tym większa, im mniejszy jest krok dyskretyzacji. Przyjmuje się przy tym, że wartość kroku dyskretyzacji przestrzeni powinna być nie mniejsza niż $\lambda/10$, gdzie λ oznacza najmniejszą długość fali, która występuje w analizowanej strukturze.

Algorytm FDTD umożliwia zwiększenie dokładności obliczeń poprzez zmniejszenie kroku dyskretyzacji przestrzeni. Jednak takie działanie jest bardzo kosztowne numerycznie, gdyż, dla analizy tego samego obszaru przestrzeni, zmniejszenie kroku dyskretyzacji prowadzi do zwiększenia rozmiaru wektorów $\mathbf{e}$ oraz $\mathbf{h}$ oraz wymaga przeprowadzenia obliczeń dla większej liczby iteracji. Przykładowo, do przeprowadzenia analizy z dwukrotnie zmniejszonym krokiem dyskretyzacji wymagany jest ośmiokrotny wzrost liczby komórek Yee oraz podwojenie liczby iteracji.

Ograniczenia dokładności analizy wynikają przede wszystkim z dyspersji numerycznej. Zjawisko dyspersji numerycznej opisuje niefizyczne własności propagacji fali w dziedzinie dyskretnej. Zostało ono dokładnie omówione w książce autorstwa A. Taflove'a oraz S. C. Hagness [105]. W testach numerycznych opisanych w tej rozprawie nie wykonano żadnych

działań, które miałyby na celu redukcję błędów wynikających z dyspersji numerycznej, z uwagi na chęć uwydatnienia najistotniejszych zagadnień będących przedmiotem tej rozprawy.

Kolejnym źródłem błędów numerycznych jest przybliżony charakter warunków brzegowych na krańcach dyskretnej dziedziny obliczeniowej. Zwłaszcza, gdy warunek brzegowy symuluje otwartą przestrzeń. W chwili obecnej żaden znany autorowi warunek brzegowy nie jest idealny i stanowi źródło fali odbitej od krańców przestrzeni obliczeniowej, co prowadzi do nieprecyzyjnego oszacowania wartości parametrów analizowanych struktur. Zastosowane przez autora tej rozprawy warunki brzegowe zostały opisane w punkcie A.3.

A.5 Ekstrakcja parametrów analizy

Analiza rezonatorów dostarcza informacje m.in. o częstotliwościach rezonansowych badanej struktury. Ekstrakcja wartości częstotliwości rezonansowych wykonywana jest za pomocą metody uogólnionego pęku funkcyjnego GPOF (ang. *Generalized Pencil-of-Function Method*) opisanej w [34].

Analiza struktur prowadzących falę elektromagnetyczną dostarcza informację o parametrach rozproszenia analizowanej struktury. Ekstrakcja parametrów rozproszenia została przeprowadzona za pomocą metody zaproponowanej przez W. Gwarka i M. Celuch-Marcysiak w [32].

DODATEK B

Dowody wybranych zależności

W dodatku tym zostały zaprezentowane dowody wybranych zależności.

B.1 Dowód symetryczności funkcji przejścia $\tilde{\mathbf{H}}(s)$

Zostanie tu wykazane, iż $\tilde{\mathbf{H}}(s) = \tilde{\mathbf{H}}(s)^T$.
Zgodnie z r. (6.13):

$$\tilde{\mathbf{H}}(s)^T = \left(\tilde{\mathbf{L}}^T \left(\frac{1}{s}\tilde{\hat{\mathbf{\Gamma}}} + s\hat{\mathbf{I}}\right)^{-1} \tilde{\mathbf{B}}\right)^T = \tilde{\mathbf{B}}^T \left(\left(\frac{1}{s}\tilde{\hat{\mathbf{\Gamma}}} + s\hat{\mathbf{I}}\right)^{-1}\right)^T \tilde{\mathbf{L}} \tag{B.1}$$

Powiązanie pomiędzy $\tilde{\mathbf{B}}$ oraz $\tilde{\mathbf{L}}$ wynika z warunku stabilności schematu lokalnego zagęszczenia siatki lokalnej zapisanego w r. (6.11), który można rozpisać następująco:

powtórzone r. (6.11)
$$\tilde{\hat{\mathbf{S}}}_E = \tilde{\mathbf{S}}_H^T \tag{B.2}$$

uwzgl. (6.18d) i (6.18e)
$$\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \hat{\mathbf{S}}_E \mathbf{D}_\epsilon^{-\frac{1}{2}} = \left(\mathbf{D}_\epsilon^{-\frac{1}{2}} \mathbf{S}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}}\right)^T \tag{B.3}$$

uwzgl. (5.4) i (5.5)
$$\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \hat{\mathbf{B}}_E \mathbf{I}_E \mathbf{L}_E \mathbf{D}_\epsilon^{-\frac{1}{2}} = \left(\mathbf{D}_\epsilon^{-\frac{1}{2}} \mathbf{B}_H \mathbf{I}_H \hat{\mathbf{L}}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}}\right)^T \tag{B.4}$$

uwzgl. (6.5) i (6.8)
$$-\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \mathbf{B} \mathbf{L}_E \mathbf{D}_\epsilon^{-\frac{1}{2}} = \left(\mathbf{D}_\epsilon^{-\frac{1}{2}} \mathbf{B}_H \mathbf{L}^T \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}}\right)^T \tag{B.5}$$

$$-\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \mathbf{B} \mathbf{L}_E \mathbf{D}_\epsilon^{-\frac{1}{2}} = \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \mathbf{L} \mathbf{B}_H^T \mathbf{D}_\epsilon^{-\frac{1}{2}} \tag{B.6}$$

co prowadzi do warunków:
$$\mathbf{B} = -\mathbf{L} \tag{B.7}$$

$$\mathbf{L}_E = \mathbf{B}_H^T \tag{B.8}$$

Uwzględniając r. (B.7) w r. (B.1):

$$\begin{aligned}\tilde{\mathbf{H}}(s)^T &= \tilde{\mathbf{L}}^T \left(\left(\frac{1}{s}\hat{\tilde{\mathbf{\Gamma}}} + s\hat{\mathbf{I}} \right)^{-1} \right)^T \tilde{\mathbf{B}} = \tilde{\mathbf{L}}^T \left(\left(\frac{1}{s}\hat{\tilde{\mathbf{\Gamma}}} + s\hat{\mathbf{I}} \right)^T \right)^{-1} \tilde{\mathbf{B}} = \\ &= \tilde{\mathbf{L}}^T \left(\frac{1}{s}\hat{\tilde{\mathbf{\Gamma}}}^T + s\hat{\mathbf{I}}^T \right)^{-1} \tilde{\mathbf{B}} = \\ &= \tilde{\mathbf{L}}^T \left(\frac{1}{s} \left(\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \widehat{\mathbf{R}}_E \widehat{\mathbf{D}}_\epsilon^{-1} \widehat{\mathbf{R}}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \right)^T + s\hat{\mathbf{I}} \right)^{-1} \tilde{\mathbf{B}} = \\ &= \tilde{\mathbf{L}}^T \left(\frac{1}{s} \left(\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \right)^T \widehat{\mathbf{R}}_H^T \left(\widehat{\mathbf{D}}_\epsilon^{-1} \right)^T \widehat{\mathbf{R}}_E^T \left(\widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \right)^T + s\hat{\mathbf{I}} \right)^{-1} \tilde{\mathbf{B}} \end{aligned} \tag{B.9}$$

Ponieważ macierze $\widehat{\mathbf{D}}_\mu$ oraz $\widehat{\mathbf{D}}_\epsilon$ są macierzami diagonalnymi oraz $\widehat{\mathbf{R}}_E = \widehat{\mathbf{R}}_H^T$ (zgodnie z r. (2.19)), dalsze przekształcenia mają postać:

$$\begin{aligned}\tilde{\mathbf{H}}(s)^T &= \tilde{\mathbf{L}}^T \left(\frac{1}{s} \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} \widehat{\mathbf{R}}_E \widehat{\mathbf{D}}_\epsilon^{-1} \widehat{\mathbf{R}}_H \widehat{\mathbf{D}}_\mu^{-\frac{1}{2}} + s\hat{\mathbf{I}} \right)^{-1} \tilde{\mathbf{B}} = \\ &= \tilde{\mathbf{L}}^T \left(\frac{1}{s}\hat{\tilde{\mathbf{\Gamma}}} + s\hat{\mathbf{I}} \right)^{-1} \tilde{\mathbf{B}} = \\ &= \tilde{\mathbf{H}}(s) \end{aligned}$$

Co należało wykazać.

DODATEK C

Szczegółowy opis wybranych implementacji

W dodatku tym opisano wybrane implementacje, które są pomocne w zrozumieniu zagadnień opisanych w zasadniczych rozdziałach tej pracy. Materiał tu zawarty nie tworzy spójnej całości, lecz obejmuje zbiór informacji, które doprecyzowują opisy zawarte w dotychczasowej części pracy, w których występują odwołania do poszczególnych punktów tego dodatku.

C.1 Pomiar czasu wykonywania instrukcji SSE

Instrukcje SSE wykonują operacje na liczbach umieszczonych w rejestrach XMM. Każdy rejestr posiada 128 bity długości, co pozwala na przechowywanie w nim czterech liczb zmiennoprzecinkowych o pojedynczej precyzji (typu `float`) lub dwóch liczb zmiennoprzecinkowych o podwójnej precyzji (typu `double`). Dane w tych rejestrach można umieścić m.in. poprzez odpowiednie wywołanie instrukcji `_mm_load_ps`. Instrukcja ta należy do grupy funkcji określanych przez firmę Intel jako funkcje wbudowane (ang. *intrinsic functions*) [2, 3], które posiadają ściśle określoną reprezentację w asemblerze i zwykle stanowią odpowiednik jednej instrukcji procesora. Przykładowo `_mm_load_ps` odpowiada instrukcji MOVAPS, a `_mm_add_ps` instrukcji ADDPS. Stanowią one zatem przejrzystą i przyjazną dla programisty formę wprowadzania instrukcji procesora do kodu programu, w odróżnieniu od wprowadzenia kodu asemblera do programu napisanego w języku C.

Obie wymienione funkcje operują na czteroelementowym zbiorze liczb typu `float`, który odpowiada zawartości jednego rejestru XMM i jest reprezentowany w języku C przez typ `__m128`. Należy przy tym podkreślić, że adres tablic współpracujących z większością funkcji SSE, np. `float * x` z wydruku C.1, musi być wyrównany do 16 bajtów, tzn. ostatnie 4 bity adresu ($2^4 = 16$) muszą być równe zero (jeżeli tak nie jest, program wygeneruje komunikat o błędzie działania). W systemie Linux odpowiedni przydział pa-

mięci można zrealizować poprzez wywołanie np. funkcji `posix_memalign` (funkcja `malloc` nie posiada wskazanej funkcjonalności).

Pomiar czasu wykonania funkcji przedstawionej na wydruku C.1 umożliwia wyznaczenie czasu wykonania instrukcji ADDPS, a przez to pozwala zmierzyć maksymalną moc obliczeniową wybranego procesora jednordzeniowego lub moc obliczeniową jednego rdzenia procesora wielordzeniowego (instrukcja ADDPS wykonuje cztery operacje dodawania liczb typu `float` na czterech parach tych liczb umieszczonych w dwóch rejestrach XMM). Natomiast pomiar czasu wykonania funkcji przedstawionej na wydruku C.2 umożliwia określenie w jakim stopniu program komputerowy jest w stanie z maksymalnej mocy obliczeniowej skorzystać. Opis pomiarów czasu działania poszczególnych funkcji zawarto w p. 3.3.2.1.

```
1  /**
2   * Funkcja wykonuje steps krotnie cztery instrukcje SSE _mm_add_ps.
3   */
4  void ssetestA( float * x, float * y, int steps ){
5      __m128 xmm0,
6             xmm1,
7             xmm2,
8             xmm3,
9             xmm4,
10            xmm5,
11            xmm6,
12            xmm7;
13
14     xmm0 = _mm_load_ps( x      );
15     xmm1 = _mm_load_ps( x +  4 );
16     xmm2 = _mm_load_ps( x +  8 );
17     xmm3 = _mm_load_ps( x + 12 );
18
19     xmm4 = _mm_load_ps( y      );
20     xmm5 = _mm_load_ps( y +  4 );
21     xmm6 = _mm_load_ps( y +  8 );
22     xmm7 = _mm_load_ps( y + 12 );
23
24     while( steps-- ){
25         xmm0 = _mm_add_ps( xmm0, xmm4 );
26         xmm1 = _mm_add_ps( xmm1, xmm5 );
27         xmm2 = _mm_add_ps( xmm2, xmm6 );
28         xmm3 = _mm_add_ps( xmm3, xmm7 );
29     }
30     _mm_store_ps( x     , xmm0  );
31     _mm_store_ps( x +  4, xmm1  );
32     _mm_store_ps( x +  8, xmm2  );
33     _mm_store_ps( x + 12, xmm3  );
34 }
```

Wydruk C.1: Funkcja przeznaczona do pomiaru czasu wykonywania instrukcji ADDPS

```
1  /**
2   * Funkcja wykonuje steps krotnie cztery instrukcje SSE _mm_add_ps.
3   * Dodatkowo w kazdej iteracji 32 liczby typu float sa wczytywane
4   * do rejestrow XMM oraz 16 liczb jest zapisywanych z rejestrow do pamieci.
5   */
6  void ssetestB( float * x, float * y, int steps ){
7      __m128 xmm0,
8             xmm1,
9             xmm2,
10            xmm3,
11            xmm4,
12            xmm5,
13            xmm6,
14            xmm7;
15
16     while( steps-- ){
17         xmm0 = _mm_load_ps( x      );
18         xmm1 = _mm_load_ps( x +  4 );
19         xmm2 = _mm_load_ps( x +  8 );
20         xmm3 = _mm_load_ps( x + 12 );
21
22         xmm4 = _mm_load_ps( y      );
23         xmm5 = _mm_load_ps( y +  4 );
24         xmm6 = _mm_load_ps( y +  8 );
25         xmm7 = _mm_load_ps( y + 12 );
26
27         xmm0 = _mm_add_ps( xmm0, xmm4 );
28         xmm1 = _mm_add_ps( xmm1, xmm5 );
29         xmm2 = _mm_add_ps( xmm2, xmm6 );
30         xmm3 = _mm_add_ps( xmm3, xmm7 );
31
32         _mm_store_ps( x     , xmm0  );
33         _mm_store_ps( x +  4, xmm1  );
34         _mm_store_ps( x +  8, xmm2  );
35         _mm_store_ps( x + 12, xmm3  );
36     }
37 }
```

Wydruk C.2: Funkcja przeznaczona do pomiaru czasu wykonywania instrukcji ADDPS wraz z czasem wymiany informacji z rejestrami XMM

C.2 Pomiar czasu przejścia z jednego elementu listy do kolejnego

Pomiar czasu przejścia z jednego elementu listy typu `struct l` (zdefiniowanego w wydruku C.3) pozwala określić wpływ rozmieszczenia danych w pamięci na poziom transmisji danych. Transfer danych jest wymagany każdorazowo przy pobraniu adresu pamięci kolejnego elementu listy. Rezultat testów przeprowadzonych na podstawie programu, którego fragmenty prezentuje wydruk C.3, zaprezentowano w p. 3.3.3.

```
/* Struktura danych przeznaczona do testow. */
struct l {
    struct l *n;
};

/* Fragment kodu, ktory sluzy do wyznaczenia czasu
   poruszania sie po liscie. */
    struct timeval   tstart,
                     tend;
    double comptime; /* [ms] czas steps krotnego przejscia do
                        nastepnego elementu listy */

    gettimeofday( &tstart, NULL );

    while( steps-- > 0 ){
        sl = sl->n;
        asm volatile ("" :: "r" (sl));
    }

    gettimeofday( &tend, NULL );
    comptime  = ((double)(tend.tv_sec  - tstart.tv_sec ))*1e3;
    comptime += ((double)(tend.tv_usec - tstart.tv_usec))*1e-3;
```

Wydruk C.3: Zasadnicze fragmenty kodu, ktore pozwalaja zmierzyc czas przejscia z jednego elementu listy do kolejnego

C.3 Reprezentacja macierzy rzadkiej w programie komputerowym

Pod pojęciem macierzy rzadkiej rozumiana jest macierz, która posiada znaczącą liczbę elementów o wartości równej zero. Dla takiej macierzy można zdefiniować format reprezentacji danych, który w porównaniu do macierzy pełnej[1] będzie wymagać mniej pamięci komputera oraz umożliwi przeprowadzenie w krótszym czasie podstawowych operacji matematycznych, np. mnożenia macierzy przez wektor.

Istnieje wiele formatów reprezentacji macierzy rzadkiej w programie komputerowym [92]. Najpopularniejsze z nich to:

- format kompresji kolumn, CCS (ang. *Compressed Column Storage*),
- format kompresji wierszy, CRS (ang. *Compressed Row Storage*),
- format kompresji diagonalnych, CDS (ang. *Compressed Diagonal Storage*),
- format blokowej kompresji wierszy, BCRS (ang. *Block Compresed Row Storage*).

Do reprezentacji macierzy rzadkiej w programach stosujących macierz rzadką zaprezentowanych w tej rozprawie zastosowano format CRS.

Format CRS reprezentuje macierz rzadką zawierającą liczby rzeczywiste przy pomocy trzech tablic[2]:

[1] Macierz pełna - reprezentacja macierzy w komputerze, która przechowuje wartości wszystkich elementów tej macierzy w pamięci.

[2] Przedstawione tu oznaczenia tablic oraz liczby elementów niezerowych w macierzy rzadkiej nie mają charakteru powszechnie stosowanego zwyczaju, lecz odnoszą się do odwołań zawartych w tej rozprawie.

1. Pierwsza tablica, `rvr`, przechowuje wartości elementów niezerowych macierzy. Jej rozmiar równy jest liczbie elementów niezerowych macierzy rzadkiej, `nnz`. Tablica ta może zawierać liczby zmiennoprzecinkowe pojedynczej precyzji (typ `float` w języku C) lub podwójnej precyzji (typ `double` w języku C).

2. Druga tablica, `rj`, przechowuje numery kolumn elementów niezerowych, zgodnie z ich ułożeniem w tablicy `rvr`. Jej rozmiar równy jest liczbie elementów niezerowych macierzy rzadkiej, `nnz`. Założono, że zawiera ona liczby typu `int`.

3. Trzecie tablica, `ri`, zawiera informacje na temat numerów wierszy poszczególnych elementów niezerowych w postaci skompresowanej. Jej rozmiar równy jest liczbie wierszy powiększonej o jeden. Założono, że zawiera ona liczby typu `int`. Wartość `i`-tego elementu tablicy `ri` określa, który element tablic `rvr` oraz `rj` zawiera informację o pierwszym niezerowym elemencie `i`-tego wiersza macierzy rzadkiej. Wyjątek stanowi ostatni element tej tablicy, który równy jest `nnz+1`, gdy zliczanie indeksów rozpoczyna się od jeden lub `nnz`, gdy zliczanie indeksów rozpoczyna się od zera. Przy takim zdefiniowaniu tej tablicy wartość `ri[i+1]-ri[i]` równa jest liczbie elementów w `i`-tym wierszu macierzy rzadkiej.

Sposób reprezentacji macierzy rzadkiej w formacie CRS zostanie przedstawiony na przykładzie macierzy **A**:

$$\mathbf{A} = \begin{bmatrix} 0 & 8 & 3 & 0 & 5 \\ 9 & 0 & 0 & 0 & 4 \\ 0 & 6 & 7 & 0 & 0 \\ 0 & 1 & 0 & 2 & 0 \end{bmatrix}$$

W tym przykładzie przyjęto zliczanie indeksów od zera, zgodnie z regułą przyjętą w numeracji elementów tablicy w języku C. Dla macierzy **A** poszczególne wektory zawierają następujące wartości:

rvr:	8	3	5	9	4	6	7	1	2
rj:	1	2	4	0	4	1	2	1	3
ri:	0	3	5	7	9				

Do reprezentacji macierzy rzadkiej o n wierszach i nnz elementach niezerowych format CRS wymaga pamięci o rozmiarze $L_{B\,CRS}$ zdefiniowanym w r. (C.1).

$$L_{B\,CRS} = \alpha_F \cdot nnz + \alpha_I(nnz + n + 1) \tag{C.1}$$

gdzie:
α_F - liczba bajtów wymagana do reprezentacji liczby zmiennoprzecinkowej,
α_I - liczba bajtów wymagana do reprezentacji liczby całkowitej.

Przy założeniu, że w każdym wierszu macierzy rzadkiej występuje co najmniej jeden element niezerowy, liczba działań zmiennoprzecinkowych wykonanych przy mnożeniu macierzy rzadkiej przez wektor wynosi $L_{F\,CRS}$, zgodnie z r. (C.2).

$$L_{F\,CRS} = 2nnz - n \tag{C.2}$$

Bibliografia

[1] Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture. http://www.intel.com/products/processor/manuals/.
Cytowania: 49, 51, 52

[2] Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2A: Instruction Set Reference, A-M. http://www.intel.com/products/processor/manuals/.
Cytowania: 53, 175

[3] Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 2B: Instruction Set Reference, N-Z. http://www.intel.com/products/processor/manuals/.
Cytowania: 175

[4] LMbench project. http://sourceforge.net/projects/lmbench.
Cytowania: 53

[5] Software Optimization Guide for AMD Family 10h Processors.
http://developer.amd.com/documentation/guides/pages/default.aspx.
Cytowania: 52

[6] Software Optimization Guide for AMD64 Processors.
https://wwwsecure.amd.com/us-en/Processors/ProductInformation/0,,30_118_15328,00.html.
Cytowania: 52

[7] Letter symbols to be used in electrical technology - Part 2: Telecommunications and electronics, Second edition. Raport instytutowy, IEC 60027-2, 2000.
Cytowania: 48

[8] A.A. Samarskiy. *Theory of Finite Difference Schemes*. Nauka, Moscow, 1977.
Cytowania: 170

[9] Advanced Design System (ADS). http://agilent.com.
Cytowania: 21

[10] Yoshihiro Akahane, Takashi Asano, Bong-Shik Song, Susumu Noda. High-Q photonic nanocavity in a two-dimensional photonic crystal. *Nature*, 425(5):944–947, Oct 2003.
Cytowania: 148

[11] Muthu Manikandan Baskaran, Rajesh Bordawekar. Optimizing Sparse Matrix-Vector Multiplication on GPUs. IBM Technical Report RC24704. Raport instytutowy, IBM, 2008.
Cytowania: 121

[12] Nathan Bell, Michael Garland. Efficient Sparse Matrix-Vector Multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, Dec 2008.
Cytowania: 121

[13] Nathan Bell, Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, strony 1–11, 2009.
Cytowania: 121

[14] J.-P. Berenger. A Perfectly Matched Layer for the Absorption of Electromagnetic Waves. *Journal of Computational Physics*, Oct 1994.
Cytowania: 171

[15] L. Catarinucci, P. Palazzari, L. Tarricone. Human exposure to the near field of radiobase antennas - a full-wave solution using parallel FDTD. *Microwave Theory and Techniques, IEEE Transactions on*, 51(3):935 – 940, mar 2003.
Cytowania: 22

[16] M. Celuch-Marcysiak, W.Gwarek. Higher order modelling of media interfaces for enhanced FDTD analysis of microwave circuits. *Proc. 24th European Microwave Conference*, strony 1530–1535, sep 1994.
Cytowania: 23

[17] J. Chi, F. Liu, J. Jin, D.G. Mason, S. Crozier. GPU accelerated FDTD solver and its application in MRI. *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, strony 3305 –3308, 31 2010-sept. 4 2010.
Cytowania: 23

[18] T. Ciamulski, M. Sypniewski. Linear and superlinear speedup in parallel FDTD processing. *Antennas and Propagation Society International Symposium, 2007 IEEE*, strony 4897 –4900, june 2007.
Cytowania: 22

[19] M. Clemens, T. Weiland. Discrete Electromagnetism with the Finite Integration Technique. *Progress In Electromagnetics Research*, 32:65–87, 2001.
Cytowania: 37

[20] R. Courant, K. Friedrichs, H. Lewy. Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74, 1928.
Cytowania: 170, 183

[21] R. Courant, K. Friedrichs, H. Lewy. On the Partial Difference Equations of Mathematical Physics. *IBM J. Res. Develop.*, 11:215–234, March 1967. (angielskie tłumaczenie [20]).
Cytowania: 170

[22] CST Microwave Studio. http://www.cst.com.
Cytowania: 21

[23] D. De Donno, A. Esposito, L. Tarricone, L. Catarinucci. Introduction to GPU Computing and CUDA Programming: A Case Study on FDTD [EM Programmer's Notebook]. *Antennas and Propagation Magazine, IEEE*, 52(3):116 –122, june 2010.
Cytowania: 23

[24] J.P. Durbano, F.E. Ortiz, J.R. Humphrey, M.S. Mirotznik, D.W. Prather. Hardware implementation of a three-dimensional finite-difference time-domain algorithm. *Antennas and Wireless Propagation Letters, IEEE*, 2:54 – 57, 2003.
Cytowania: 22

[25] A. Dziekonski, P. Sypek, L. Kulas, M. Mrozowski. Implementation of matrix-type FDTD algorithm on a graphics accelerator. *Microwaves, Radar and Wireless Communications, 2008. MIKON 2008. 17th International Conference on*, strony 1 –4, may 2008.
Cytowania: 98

[26] A. Dziekonski, P. Sypek, M. Mrozowski. How to Reforge Rendering Engines into an Efficient Platform for FDTD Computations. *13th Biennial IEEE Conference on Electromagnetic Field Computation, CEFC 2008*, maj 2008.
Cytowania: 81, 82, 84

[27] Paul Eller, Jing-Ru C. Cheng, Donald Albert. Acceleration of 2-D Finite Difference Time Domain Acoustic Wave Simulation Using GPUs. *High Performance Computing Modernization Program Users Group Conference (HPCMP-UGC), 2010 DoD*, strony 350 –356, june 2010.
Cytowania: 23

[28] A. Farjadpour, D. Roundy, A. Rodriguez, M. Ibanescu, P. Bermel, J. D. Joannopoulos, S. G. Johnson, Burr G. W. Improving accuracy by subpixel smoothing in the finite-difference time domain. *Optics Letters*, 31(20):2972–2974, Oct 2006.
Cytowania: 31

[29] A. Fijany, M.A. Jensen, Y. Rahmat-Samii, J. Barhen. A massively parallel computation strategy for FDTD: time and space parallelism applied to electromagnetics problems. *Antennas and Propagation, IEEE Transactions on*, 43(12):1441 –1449, dec 1995.
Cytowania: 22

[30] S.D. Gedney. Finite-difference time-domain analysis of microwave circuit devices on high performance vector/parallel computers. *Microwave Theory and Techniques, IEEE Transactions on*, 43(10):2510 –2514, oct 1995.
Cytowania: 22

[31] C. Guiffaut, K. Mahdjoubi. A parallel FDTD algorithm using the MPI library. *Antennas and Propagation Magazine*, 43:94–103, kwiecień 2001.
Cytowania: 22, 60, 73, 82

[32] W.K. Gwarek, M. Celuch-Marcysiak. Wide-Band S-Parameter Extraction From FD-TD Simulations for Propagating and Evanescent Modes in Inhomogeneous Guides. *Microwave Theory and Techniques, IEEE Transactions on*, 51(8):1920–1928, Aug. 2003.
Cytowania: 160, 172

[33] T. Hanawa, M. Kurosawa, S. Ikuno. Investigation on 3-D implicit FDTD method for parallel processing. *Magnetics, IEEE Transactions on*, 41(5):1696–1699, May 2005.
Cytowania: 108, 109, 111

[34] Y. Hua, T.K. Sarkar. Generalized Pencil-of-Function Method for Extracting Poles of an EM System from Its Transient Response. *Antennas and Propagation, IEEE Transactions on*, 37(2):229–234, Feb 1989.
Cytowania: 172

[35] N. H. Huynh, W. Heinrich. FDTD accuracy improvement by incorporation of 3D edge singularities. *Proc. Of Int. Microwave Symposium, IMS-1999*, strony 1573–1576, jun 1999.
Cytowania: 23

[36] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.
Cytowania: 60, 64, 93

[37] M.J. Inman, A.Z. Elsherbeni. Programming video cards for computational electromagnetics applications. *Antennas and Propagation Magazine, IEEE*, 47(6):71 –78, dec. 2005.
Cytowania: 23

[38] Feng Jin, Xiaoli Xi, Hu Liu, Zhensheng Shi, Daocheng Wu. Study on acceleration technique for two-dimensional FDTD algorithm based on GPU. *Antennas Propagation and EM Theory (ISAPE), 2010 9th International Symposium on*, strony 798 –801, 29 2010-dec. 2 2010.
Cytowania: 23

[39] Feng Ju, Cheng Xing. A study of parallel FDTD for simulating complex antennas on a cluster system. *Microwave Conference Proceedings, 2005. APMC 2005. Asia-Pacific Conference Proceedings*, 5:1–4, Dec. 2005.
Cytowania: 73, 109, 111

[40] N. Kaneda, B. Houshmand, T. Itoh. FDTD analysis of dielectric resonators with curved surfaces. *Microwave Theory and Techniques, IEEE Transactions on*, 45:1645–1649, sep 1997.
Cytowania: 23, 31

[41] S.E. Krakiwsky, L.E. Turner, M.M. Okoniewski. Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU). *Microwave Symposium Digest, 2004 IEEE MTT-S International*, wolumen 2, strony 1033 – 1036 Vol.2, june 2004.
Cytowania: 23

[42] A.A. Kucharski, P.M. Slobodzian. The Application of Macromodels to the Analysis of a Dielectric Resonator Antenna Excited by a Cavity Backed Slot. *Microwave Conference, 2008. EuMC 2008. 38th European*, strony 519 –522, oct. 2008.
Cytowania: 143, 159, 163

[43] L. Kulas. *Metoda redukcji rzędu modelu w schematach różnicowych elektrodynamiki obliczeniowej*. Praca doktorska, Politechnika Gdańska, 2006.
Cytowania: 23, 37, 115, 118, 125, 129, 157

[44] L. Kulas, M. Mrozowski. Reduced-order models in FDTD. *Microwave and Wireless Components Letters, IEEE*, 11:422–424, oct 2001.
Cytowania: 118, 157

[45] L. Kulas, M. Mrozowski. Implementing the concept of a macromodel in the *FDTD* method. *14th International Conference on Microwaves, Radar and Wireless Communications, MIKON-2002*, wolumen 2, strony 537 –540, may 2002.
Cytowania: 23

[46] L. Kulas, M. Mrozowski. A Simple High-Accuracy Subgridding Scheme. *Proc. 33rd European Microwave Conference*, strony 347–350, oct 2003.
Cytowania: 115

[47] L. Kulas, M. Mrozowski. Reduced order models of refined yee's cells. *Microwave and Wireless Components Letters, IEEE*, 13:164–166, apr 2003.
Cytowania: 115, 133

[48] L. Kulas, M. Mrozowski. 3D Macromodels in the *FDTD* Filter Analysis. *15th International Conference on Microwaves, Radar and Wireless Communications, MIKON-2004*, wolumen 2, strony 710–713, may 2004.
Cytowania: 23

[49] L. Kulas, M. Mrozowski. A Fast High-Resolution 3-D Finite-Difference Time-Domain Scheme With Macromodels. *Microwave Theory and Techniques, IEEE Transactions on*, 52:2330–2335, sep 2004.
Cytowania: 23, 115, 118, 129, 133, 157

[50] L. Kulas, M. Mrozowski. Macromodels in the Frequency Domain Analysis of Microwave Resonators. *Microwave and Wireless Components Letters, IEEE*, 14:94–96, mar 2004.
Cytowania: 23, 115, 133

[51] L. Kulas, M. Mrozowski. Stability of the FDTD Scheme Containing Macromodels. *Microwave and Wireless Components Letters, IEEE*, 14:484–486, oct 2004.
Cytowania: 23, 127

[52] L. Kulas, M. Mrozowski. Yee's Macrocells In Three Dimensions. *Proc. Of Int. Microwave Symposium, IMS-2004*, strony 1717–1720, jun 2004.
Cytowania: 115, 133

[53] L. Kulas, M. Mrozowski. Low-reflection subgridding. *Microwave Theory and Techniques, IEEE Transactions on*, 53:1587–1592, may 2005.
Cytowania: 115, 119, 127, 133

[54] L. Kulas, M. Mrozowski. Simple and Accurate Field Interpolation in Finite Difference Methods. *16th International Conference on Microwaves, Radar and Wireless Communications, MIKON-2006*, wolumen 2, strony 711–714, may 2006.
Cytowania: 118, 157

[55] L. Kulas, M. Mrozowski. Reciprocity Principle for Stable Subgridding in the Finite Difference Time Domain Method. *International Conference on Ćomputer as a Tool", EUROCON-2007*, strony 106–111, sep 2007.
Cytowania: 115, 119

[56] L. Kulas, M. Mrozowski. A Stable High-Accuracy Subgridding Scheme. *Microwave Conference, 2008. EuMC 2008. 38th European*, strony 658–661, oct 2008.
Cytowania: 115, 119

[57] Kulas, L. and Mrozowski, M. 3D macromodels in the FDTD filter analysis. *Microwaves, Radar and Wireless Communications, 2004. MIKON-2004. 15th International Conference on*, 2:710–713 vol. 2, May 2004.
Cytowania: 115, 133

[58] Z.M. Liu, A.S. Mohan, T.A. Aubrey, W.R. Belcher. Techniques for implementation of the FDTD method on a CM-5 parallel computer. *Antennas and Propagation Magazine, IEEE*, 37(5):64 –71, oct 1995.
Cytowania: 22

[59] M. Mansourabadi, A. Pourkazemi. FDTD Hard Source and Soft Source Reviews and Modifications. *Progress In Electromagnetics Research C*, 3:143–160, 2008.
Cytowania: 42

[60] L. Mattes, S. Kofuji. The use of overlapping subgrids to accelerate the FDTD on GPU devices. *Radar Conference, 2010 IEEE*, strony 807 –810, may 2010.
Cytowania: 23

[61] James Clerk Maxwell. *A Treatise on Electricity and Magnetism*. University of Oxford, 1873.
Cytowania: 21, 25

[62] Larry McVoy, Carl Staelin. LMbench: Portable Tools for Performance Analysis. 1996.
Cytowania: 53, 54

[63] J. Meixner. The behavior of electromagnetic fields at edges. 20:442–446, jul 1972.
Cytowania: 23

[64] MPI. `http://www.mpi-forum.org/docs`.
Cytowania: 22, 65, 106

[65] MPICH. `http://www.mcs.anl.gov/research/projects/mpich2`.
Cytowania: 65

[66] MVAPICH2. `http://mvapich.cse.ohio-state.edu`.
Cytowania: 65

[67] T. Nagaoka, S. Watanabe. A GPU-based calculation using the three-dimensional FDTD method for electromagnetic field analysis. *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, strony 327 –330, 31 2010-sept. 4 2010.
Cytowania: 23

[68] Nvidia. *CUDA Programming Guide Version 2.3*. Nvidia, sierpień 2009.
`http://www.nvidia.com/object/cuda_develop.html`.
Cytowania: 8, 77, 78, 79, 80, 81, 87

[69] Nvidia. *CUDA Programming Guide Version 4.0*. Nvidia, czewiec 2011.
`http://www.nvidia.com/object/cuda_develop.html`.
Cytowania: 8, 76, 77, 79, 87

[70] N. Oguni, H. Aasai. Estimation of parallel FDTD-based electromagnetic field solver on PC cluster with multi-core CPUs. *Advanced Packaging and Systems Symposium, 2008. EDAPS 2008. Electrical Design of*, strony 159 –162, dec. 2008.
Cytowania: 22

[71] M. Okoniewski, E. Okoniewska, M. A. Stuchly. Three-dimensional subgridding algorithm for FDTD. 45:422–429, mar 1997.
Cytowania: 23

[72] C. Ong, M. Weldon, D. Cyca, M. Okoniewski. Acceleration of large-scale FDTD simulations on high performance GPU clusters. *Antennas and Propagation Society International Symposium, 2009. APSURSI '09. IEEE*, strony 1–4, 1-5 2009.
Cytowania: 23, 168

[73] George W. Pan. *Wavelets in Electromagnetics and Device Modeling*. John Wiley & Sons, New Jersey, 2003.
Cytowania: 37

[74] P. Placidi, L. Verducci, G. Matrella, L. Roselli, P. Ciampolini. A custom VLSI architecture for the solution of FDTD equations. *IEICE Transactions on Electronics*, wolumen E85-C, strony 572–577, 2002.
Cytowania: 22

[75] J. Podwalski, L. Kulas, M. Mrozowski. Grouping macromodels by using multilevel model order reduction. *Microwaves, Radar and Wireless Communications, 2008. MIKON 2008. 17th International Conference on*, strony 1–4, may 2008.
Cytowania: 157

[76] J. Podwalski, L. Kulas, P. Sypek, M. Mrozowski. Analysis of a High-Quality Photonic Crystal Resonator. *16th International Conference on Microwaves, Radar and Wireless Communications, MIKON-2006*, wolumen 3, strony 793–796, maj 2006.
Cytowania: 115, 133

[77] J. Podwalski, M. Mrozowski. Advances in macromodeling technique. *Electromagnetic Theory (EMTS), 2010 URSI International Symposium on*, strony 41–43, aug. 2010.
Cytowania: 157

[78] J. Podwalski, P. Sypek, L. Kulas, M. Mrozowski. FDTD Analysis of EBG Structures with Macromodel Cloning. *Microwave Symposium Digest, 2006. IEEE MTT-S International*, strony 296–299, czerwiec 2006.
Cytowania: 134, 135

[79] J. Podwalski, P. Sypek, L. Kulas, M. Mrozowski. Macromodel Cloning Approach in Efficient *FDTD* Analysis of *EBG* structures. *Proc. Of Int. Microwave Symposium, IMS-2006*, strony 296–299, jun 2006.
Cytowania: 23

[80] Jakub Podwalski, Lukasz Kulas, Michal Mrozowski. Advanced macromodel matrix structure cloning for FDTD. *Microwave Radar and Wireless Communications (MIKON), 2010 18th International Conference on*, strony 1 –2, june 2010.
Cytowania: 157

[81] P. Przybyszewski. *Fast finite difference numerical techniques for the time and frequency domain solution of electromagnetic problems*. Praca doktorska, Politechnika Gdańska, 2001.
Cytowania: 38, 160, 170

[82] P. Przybyszewski, M. Mrozowski. A Conductive Wedge in Yee's Mesh. 8:66–68, feb 1998.
Cytowania: 23

[83] PVM. `http://www.csm.ornl.gov/pvm/pvm_home.html`.
Cytowania: 22

[84] Min Qiu. Micro-cavities in silicon-on-insulator photonic crystal slabs: Determining resonant frequencies and quality factors accurately. *Microwave and Optical Technology Letters*, 45(5):381–385, Nov 2005.
Cytowania: 143, 148, 149

[85] Quickwave-3D. http://www.qwed.com.pl.
Cytowania: 21

[86] J. Alan Roden, Stephen D. Gedney. Convolution PML (CPML): An efficient FDTD implementation of the CFS-PML for arbitrary media. *Microwave and Optical Technology Letters*, 27(5):334–339, Dec 2000.
Cytowania: 143, 171

[87] D.P. Rodohan, S.R. Saunders. Parallel implementations of the finite difference time domain (FDTD) method. *Computation in Electromagnetics, 1994. Second International Conference on*, strony 367 –370, apr 1994.
Cytowania: 22

[88] G. Rodriguez, Y. Miyazaki, N. Goto. Matrix-based FDTD parallel algorithm for big areas and its applications to high-speed wireless communications. *Antennas and Propagation, IEEE Transactions on*, 54(3):785 –796, march 2006.
Cytowania: 22

[89] Robert Rosenberg, Guy Norton, Jorge C. Novarini, Wendell Anderson, Marco Lanzagorta. Modeling Pulse Propagation and Scattering in a Dispersive Medium: Performance of MPI/OpenMP Hybrid Code. *SC Conference*, 0:47, 2006.
Cytowania: 22

[90] F. Rossi, P.P.M. So. Hardware accelerated symmetric condensed node TLM procedure for NVIDIA graphics processing units. *Antennas and Propagation Society International Symposium, 2009. APSURSI '09. IEEE*, strony 1 –4, june 2009.
Cytowania: 23

[91] F.V. Rossi, P.P.M. So, N. Fichtner, P. Russer. Massively parallel two-dimensional TLM algorithm on graphics processing units. *Microwave Symposium Digest, 2008 IEEE MTT-S International*, strony 153 –156, june 2008.
Cytowania: 23

[92] Youcef Saad. SPARSKIT: a basic tool kit for sparse matrix computations - Version 2, 1994.
Cytowania: 178

[93] Ryan N. Schneider, Laurence E. Turner, Michal M. Okoniewski. Application of FPGA technology to accelerate the finite-difference time-domain (FDTD) method. *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays*, FPGA '02, strony 97–105, New York, NY, USA, 2002. ACM.
Cytowania: 22

[94] B. N. Sheehan. *ENOR*: Model Order Reduction of RLC Circuits Using Nodal Equations for Efficient Factorization. *Proc. IEEE 36th Design Automat. Conf.*, strony 17–21, jun 1999.
Cytowania: 126

[95] T. Stefanski, T.D. Drysdale. Parallel Implementation of ADI-FDTD on Shared and Distributed Memory Computers. *Antennas and Propagation, 2007. EuCAP 2007. The Second European Conference on*, strony 1 –6, nov. 2007.
Cytowania: 22

[96] T.P. Stefanski, S. Benkler, N. Chavannes, N. Kuster. Parallel implementation of the Finite-Difference Time-Domain method in Open Computing Language. *Electromagnetics in Advanced Applications (ICEAA), 2010 International Conference on*, strony 557 –560, sept. 2010.
Cytowania: 23

[97] T.P. Stefanski, T.D. Drysdale. Acceleration of the 3D ADI-FDTD method using graphics processor units. *Microwave Symposium Digest, 2009. MTT '09. IEEE MTT-S International*, strony 241 –244, june 2009.
Cytowania: 23

[98] B. Stupfel. A fast-domain decomposition method for the solution of electromagnetic scattering by large objects. *Antennas and Propagation, IEEE Transactions on*, 44(10):1375 –1385, oct 1996.
Cytowania: 22

[99] M.F. Su, I. El-Kady, D.A. Bader, S.-Y. Lin. A novel FDTD application featuring OpenMP-MPI hybrid parallelization. *Parallel Processing, 2004. ICPP 2004. International Conference on*, strony 373 – 379 vol.1, aug. 2004.
Cytowania: 22

[100] D.M. Sullivan, Yang Xia, A. Mansoori. Large scale underwater FDTD ELF simulations using Acceleware and MPI parallel processing. *Electromagnetic Theory (EMTS), 2010 URSI International Symposium on*, strony 104 –106, aug. 2010.
Cytowania: 23

[101] P. Sypek, A. Dziekonski, M. Mrozowski. How to Render FDTD Computations More Effective Using a Graphics Accelerator. *Magnetics, IEEE Transactions on*, 45(3):1324–1327, March 2009.
Cytowania: 81, 82, 84

[102] P. Sypek, L. Kulas, M. Mrozowski. Low Reflection Macromodels for a Stable *FDTD* Scheme Operating with Highly Refined Local Meshes. *Proc. Of Int. Microwave Symposium, IMS-2005*, strony 195–198, jun 2005.
Cytowania: 23

[103] P. Sypek, L. Kulas, M. Mrozowski. Low reflection macromodels for a stable FDTD scheme operating with highly refined local meshes. *Microwave Symposium Digest, 2005 IEEE MTT-S International*, strony 4 pp.–, June 2005.
Cytowania: 23, 115, 133

[104] P. Sypek, M. Wiktor, M. Mrozowski. Parallel Implementation of the Matrix Formulation of the FDTD Scheme. *EUROCON, 2007. The International Conference on "Computer as a Tool"*, strony 5–9, Sept. 2007.
Cytowania: 100, 104

[105] A. Taflove. *Computational Electrodynamics: The Finite-Difference Time-Domain Method. Third edition.* Artech House, Boston, London, 2005.
Cytowania: 21, 23, 25, 35, 36, 42, 45, 170, 171

[106] Eng Leong Tan. Acceleration of LOD-FDTD Method Using Fundamental Scheme on Graphics Processor Units. *Microwave and Wireless Components Letters, IEEE*, 20(12):648 –650, dec. 2010.
Cytowania: 23

[107] P. Thoma, T. Weiland. A Consistent Subgridding Scheme for the Finite Difference Time Domain Method. *Int. Journ. of Numerical Modelling*, 9:359–374, sep 1996.
Cytowania: 23

[108] Ulrich Drepper. What Every Programmer Should Know About Memory. `http://people.redhat.com/drepper/cpumemory.pdf`.
Cytowania: 50, 55, 56, 66, 68

[109] A. Valcarce, G. De La Roche, Jie Zhang. A GPU approach to FDTD for radio coverage prediction. *Communication Systems, 2008. ICCS 2008. 11th IEEE Singapore International Conference on*, strony 1585 –1590, nov. 2008.
Cytowania: 23

[110] V. Varadarajan, R. Mittra. Finite-difference time-domain (FDTD) analysis using distributed computing. *Microwave and Guided Wave Letters*, 4:144–145, maj 1994.
Cytowania: 22, 60, 73, 82

[111] M.N. Vouvakis, Z. Cendes, Jin-Fa Lee. A FEM domain decomposition method for photonic and electromagnetic band gap structures. *Antennas and Propagation, IEEE Transactions on*, 54(2):721 – 733, feb. 2006.
Cytowania: 22

[112] J. Wang, O. Fujiwara, S. Watanabe, Y. Yamanaka. Computation with a parallel FDTD system of human-body effect on electromagnetic absorption for portable telephones. *Microwave Theory and Techniques, IEEE Transactions on*, 52(1):53 – 58, jan. 2004.
Cytowania: 22

[113] T. Weiland. Maxwell's Grid Equations. *Frequenz*, 44:9–16, styczeń 1990.
Cytowania: 33, 34, 35

[114] Michał Wiktor. *Zastosowanie metod projekcji w algorytmach różnicowych elektrodynamiki obliczeniowej.* Praca doktorska, Politechnika Gdańska, 2006.
Cytowania: 116

[115] Wojciech Rubinowicz. *Wektory i tensory.* 1950.
Cytowania: 25

[116] A. Woods, L. Ludeking. Performance Enhancement of FDTD-PIC Beam-Wave Simulations Using Multi-core Platforms. *presentation prepared for PIERS 2010 Xi'An, China*, marzec 2010.
Cytowania: 22

[117] Meilian Xu, P. Thulasiraman. Parallel Algorithm Design and Performance Evaluation of FDTD on 3 Different Architectures: Cluster, Homogeneous Multicore and Cell/B.E. *High Performance Computing and Communications, 2008. HPCC '08.*

10th IEEE International Conference on, strony 174 –181, sept. 2008.
Cytowania: 22

[118] Kane S. Yee. Numerical Solution of Inital Boundary Value Problems Involving Maxwell's Equations in Isotropic Media. *Antennas and Propagation, IEEE Transactions on*, 14(3):302–307, May 1966.
Cytowania: 28

[119] Wenhua Yu, Yongjun Liu, Tao Su, Neng-Tien Hunag, R. Mittra. A robust parallel conformal finite-difference time-domain processing package using the MPI library. *Antennas and Propagation Magazine, IEEE*, 47(3):39 – 59, june 2005.
Cytowania: 22

[120] Wenhua Yu, R. Mittra. A conformal finite difference time domain technique for modeling curved dielectric surfaces. *Microwave and Wireless Components Letters, IEEE*, 11(1):25–27, Jan 2001.
Cytowania: 31

[121] Wenhua Yu, Xiaoling Yang, Yongjun Liu, Lai ching Ma, Tao Sul, Neng-Tien Huang, R. Mittra, R. Maaskant, Yongquan Lu, Qing Che, Rul Lu, Zhiwu Su. A New Direction in Computational Electromagnetics: Solving Large Problems Using the Parallel FDTD on the BlueGene/L Supercomputer Providing Teraflop-Level Performance. *Antennas and Propagation Magazine, IEEE*, 50(2):26–44, april 2008.
Cytowania: 22

[122] Zhang Yu, Ding Wei, Liang Changhong. Analysis of parallel performance of MPI based parallel FDTD on PC clusters. *Microwave Conference Proceedings, 2005. APMC 2005. Asia-Pacific Conference Proceedings*, 4:1–3, Dec. 2005.
Cytowania: 22, 73, 108, 109, 111

[123] M.R. Zunoubi, J. Payne, W.P. Roach. CUDA Implementation of TE^z-FDTD Solution of Maxwell's Equations in Dispersive Media. *Antennas and Wireless Propagation Letters, IEEE*, 9:756 –759, 2010.
Cytowania: 23

[124] M. Rewieński. *High Performance Algorithms for Large Scale Electromagnetic Modeling*. Praca doktorska, Politechnika Gdańska, 1999.
Cytowania: 104

[125] P. Debicki, P. Jedrzejewski, A. Kreczkowski, J. Mielewski, M. Mrozowski, K. Nyka, P. Przybyszewski, M. Rewienski, T. Rutkowski. Coping with numerical complexity in computational electromagnetics. *Microwaves and Radar, 1998. MIKON '98., 12th International Conference on*, wolumen 4, strony 175 –197 vol.4, may 1998.
Cytowania: 104

Prawo rozpowszechniania

Niniejszym wyrażam zgodę na wykorzystanie wyników mojej pracy, w tym tabel i rysunków, w pracach badawczych i publikacjach przygotowywanych przez pracowników Politechniki Gdańskiej lub pod ich kierownictwem. Wykorzystanie wyników wymaga wskazania niniejszej rozprawy doktorskiej jako źródła.

Oświadczenie

W zaprezentowanej rozprawie znaczną liczbę obliczeń wykonano na komputerach Centrum Informatycznego Trójmiejskiej Akademickiej Sieci Komputerowej.

Sylwetka autora

Piotr Sypek jest absolwentem Wydziału Elektroniki, Telekomunikacji i Informatyki Politechniki Gdańskiej. W lipcu 2003 roku obronił pracę dyplomową w katedrze Techniki Mikrofalowej i Telekomunikacji Optycznej z oceną bardzo dobrą. Tematyka pracy magisterskiej dotyczyła metody analizy układów mikrofalowych o dowolnej topologii.

Mgr inż. Piotr Sypek w październiku 2003 rozpoczął studia doktoranckie na Wydziale Elektroniki, Telekomunikacji i Informatyki. Jest współautorem czternastu publikacji naukowych, z których dwanaście jest dostępych w bazie IEEE. Najważniejsza z nich pochodzi z czasopisma *IEEE Transactions on Magnetics* i aktualnie jest cytowana ponad dwudziestokrotnie.

Podziękowania

Dziękuję mojemu promotorowi, profesorowi Michałowi Mrozowskiemu, za inspirację do poszukiwań ciekawych obszarów wiedzy, świetne warunki do rozwoju naukowego, wielokrotnie udzieloną pomoc, a także okazane zaufanie, cierpliwość i wyrozumiałość.

Chcę również wyrazić swoje podziękowania osobom, które pomogły mi w odnalezieniu ścieżki naukowca: docentowi Markowi Kitlińskiemu za przekazanie zainteresowania światem mikrofal, profesorowi Jerzemu Mazurowi za ukazanie piękna prostych rozwiązań, profesorowi Krzysztofowi Grabowskiemu za bycie wspaniałem człowiekiem i naukowcem oraz dodanie mi wiary we własne możliwości.

Dziękuję kolegom, którzy pomogli mi w zrozumieniu wielu zagadnień naukowych: dr. Łukaszowi Kulasowi, dr. Michałowi Wiktorowi, dr. Piotrowi Kowalczykowi, dr. Łukaszowi Balewskiemu, dr. Adamowi Lamęckiemu oraz Adamowi Dziekońskiemu.

Praca ta zawdzięcza swoje powstanie również moim przyjaciołom: Robertowi, Dorocie, Jarkowi i Marcinowi. Dziękuję Wam za pomoc w chwilach zwątpienia i nieustanną wiarę w końcowy sukces.

Jednak przede wszystkim dziękuję moim Rodzicom za zgodę na moje podążanie własnymi ścieżkami i serdeczne wsparcie moich prób i starań.