Tema

Tendências em Matemática Aplicada e Computacional, **17**, N. 1 (2016), 93-112

www.scielo.br/tema
doi: 10.5540/tema.2016.017.01.0093

Melhorando o Desempenho Computacional de um Esquema de Diferenças Finitas para as Equações de Maxwell

L.J.P. VELOSO[1], D.G. ALFARO VIGO[2] e S. ROSSETTO[3]

Recebido em 30 setembro, 2015 / Aceito em 11 fevereiro, 2016

RESUMO. As equações de Maxwell têm um papel crucial na teoria do eletromagnetismo e suas aplicações. Entretanto, nem sempre é possível resolver essas equações de forma analítica. Por isso, precisamos de métodos numéricos para obter soluções aproximadas das equações de Maxwell. O método FDTD (Finite-Diference Time-Domain), proposto por K. Yee, é amplamente usado devido a sua simplicidade e eficiência. No entanto, esse método apresenta um alto custo computacional. Neste trabalho, propomos uma implementação paralela do método FDTD para execução em GPUs, usando a plataforma CUDA. Nosso objetivo é reduzir o tempo de processamento requerido para viabilizar o uso do método FDTD para a simulação da propagação de ondas eletromagnéticas. Avaliamos o algoritmo proposto considerando condições de contorno de tipo Dirichlet e também condições absorventes. Obtivemos ganhos de desempenho que variam de 7 a 8 vezes, comparando a implementação paralela proposta com uma versão sequencial otimizada.

Palavras-chave: Equações de Maxwell, Algoritmo de Yee, Programação Paralela com GPU.

1 INTRODUÇÃO

A energia gerada por fontes eletromagnéticas e suas interações com o entorno possuem muitas aplicações, entre as quais podemos citar as tecnologias de comunicação sem fio e alguns tratamentos e diagnósticos usados na área médica [7, 8, 16]. O desenvolvimento e aprimoramento dessas aplicações requerem um estudo detalhado sobre a interação do campo eletromagnético e a propagação de ondas eletromagnéticas na região de interesse. Esse estudo toma como base as equações de Maxwell, um sistema de equações em derivadas parciais que descreve a evolução no tempo do campo eletromagnético.

Autor correspondente: Leandro J. Pereira Veloso.
[1] Programa de Pós-Graduação em Informática (PPGI), Instituto de Matemática (IM), Universidade Federal do Rio de Janeiro (UFRJ), 21941-590 Rio de Janeiro, RJ, Brasil. E-mail: leandro.veloso@ppgi.ufrj.br
[2] Departamento de Ciência da Computação (DCC), Programa de Pós-Graduação em Informática (PPGI), Instituto de Matemática (IM), Universidade Federal do Rio de Janeiro (UFRJ), 21941-590 Rio de Janeiro, RJ, Brasil.
E-mail: dgalfaro@dcc.ufrj.br
[3] Departamento de Ciência da Computação (DCC), Instituto de Matemática (IM), Universidade Federal do Rio de Janeiro (UFRJ), 21941-590 Rio de Janeiro, RJ, Brasil. E-mail: silvana@dcc.ufrj.br

A resolução dessas equações é analiticamente inviável na grande maioria dos casos. Em função disso, em 1966, K. Yee desenvolveu um esquema de diferenças finitas no domínio do tempo (FDTD) para resolver numericamente, e de forma bastante simples, as equações de Maxwell [18]. Trata-se de um método de diferenças finitas explícito sobre uma malha intercalada que possui precisão de ordem 2, tanto no espaço quanto no tempo [15, 16].

Apesar de simples, esse método é muito dispendioso computacionalmente devido às restrições na discretização temporal necessárias para garantir a estabilidade numérica. Em função disso, vários esforços têm sido feitos para se desenvolver implementações mais eficientes do algoritmo básico do método FDTD. Por exemplo, técnicas de paralelização desse algoritmo podem ser encontradas em [10, 16].

Nos últimos anos, a disponibilidade de novas plataformas computacionais, em particular as GPUs, tem motivado o desenvolvimento de novas alternativas de paralelização do algoritmo de Yee com ganhos significativos de desempenho. Diferentemente das CPUs, as GPUs possuem centenas de unidades de processamento com unidades de controle bastante simples. Isso faz com que elas sejam adequadas para execução de aplicações com forte paralelismo de dados, como é o caso do método FDTD.

Os primeiros trabalhos de paralelização do método FDTD para GPUs tiveram como objetivo mostrar a viabilidade dessa proposta, exigindo um significativo esforço para compreender e adaptar o algoritmo para o *pipeline* de computação gráfica (usando as linguagens OpenGL, Brook e Cg) [1, 9, 12]. A partir daí, várias propostas de implementação do método FDTD para GPUs foram avaliadas, explorando ambientes de programação de nível mais alto, como CUDA. Entre essas propostas destacamos [2, 5, 6, 17].

Neste artigo, apresentamos uma proposta de paralelização do esquema de Yee para execução em GPU usando a plataforma CUDA e considerando o caso bidimensional das equações de Maxwell. Assim como em outros trabalhos ([1, 2, 17]), nosso objetivo é realizar simulações FDTD em tempo razoável para viabilizar o uso desse método na simulação de aplicações reais. Para isso, implementamos duas versões do algoritmo, uma usando condições de contorno de Dirichlet, e outra usando condições de contorno absorventes [3]. Além disso, ambas as implementações permitem que o usuário opte por salvar em arquivo os resultados intermediários da simulação, uma vez que algumas aplicações requerem não apenas o resultado final da simulação mas também os resultados intermediários.

Comparamos o tempo de execução dos algoritmos paralelos e suas versões sequenciais otimizadas executadas em uma CPU convencional, obtendo ganhos da ordem de 8 vezes. Além da avaliação de desempenho, apresentamos uma avaliação de corretude do algoritmo paralelo proposto, tomando como referência um exemplo com solução analítica exata.

O restante deste texto está organizado da seguinte forma. Na seção 2 descrevemos o esquema de diferenças finitas para a resolução das equações de Maxwell proposto por Yee e as condições de contorno adotadas neste trabalho. Na seção 3 apresentamos nossa proposta de paralelização do algoritmo de Yee para o ambiente de GPU. Na seção 4 detalhamos as avaliações de corretude e desempenho realizadas para o algoritmo paralelo proposto. Por fim, na seção 5, apresentamos as conclusões deste trabalho.

2 ESQUEMA DE DIFERENÇAS FINITAS PARA AS EQUAÇÕES DE MAXWELL

As equações de Maxwell descrevem a evolução no tempo do campo eletromagnético em uma região do espaço. Na sua forma diferencial, elas são dadas pelo sistema de equações diferenciais parciais [16]:

$$\begin{aligned} \frac{\partial \mathbf{H}}{\partial t} &= -\frac{1}{\mu}\nabla \times \mathbf{E} - \frac{1}{\mu}(\mathbf{M}_{fonte} + \sigma^* \mathbf{H}) \\ \frac{\partial \mathbf{E}}{\partial t} &= \frac{1}{\epsilon}\nabla \times \mathbf{H} - \frac{1}{\epsilon}(\mathbf{J}_{fonte} + \sigma \mathbf{E}) \end{aligned} \tag{1}$$

onde H e E representam os campos magnético e elétrico, respectivamente; μ é a permeabilidade magnética, ϵ a permissividade elétrica, σ a condutividade elétrica e σ^* a perda magnética equivalente; $\nabla\times$ representa o operador rotacional. Observamos que para obter as soluções desse sistema é preciso adicionar as condições iniciais e de contorno.

Considerando que em relação a um sistema de coordenadas cartesianas fixado, o campo eletromagnético e as características do meio não dependem da coordenada z, o sistema (1) pode ser reescrito em uma forma mais simplificada. O modo transversal elétrico TE_z (*transverse-electric mode*) é dado pelo seguinte sistema de equações diferenciais parciais para as componentes E_x, E_y e H_z do campo eletromagnético

$$\begin{aligned} \frac{\partial H_z}{\partial t} &= \frac{1}{\mu}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - (M_z + \sigma^* H_z)\right), \\ \frac{\partial E_x}{\partial t} &= \frac{1}{\varepsilon}\left(\frac{\partial H_z}{\partial y} - (J_x + \sigma E_x)\right), \\ \frac{\partial E_y}{\partial t} &= \frac{1}{\varepsilon}\left(-\frac{\partial H_z}{\partial x} - (J_y + \sigma E_y)\right). \end{aligned} \tag{2}$$

As demais componentes do campo eletromagnético satisfazem a um sistema semelhante ao (2) que é conhecido como modo transversal magnético TM_z (*transverse-magnetic mode*).

Esses sistemas de equações diferenciais parciais são desacoplados. Por isso devem ser resolvidos considerando uma região do plano cartesiano com coordenadas (x, y), introduzindo as condições iniciais e de contorno apropriadas. Para domínios limitados, consideraremos condições de contorno em que o campo elétrico ou magnético na direção tangencial ao contorno é especificado. Neste artigo, consideramos o caso em que o domínio é um retângulo com lados paralelos aos eixos x e y, por isso em cada lado é especificada uma das funções incógnitas. Nos referimos a essas condições de contorno como **condições de contorno de tipo Dirichlet**.

2.1 Esquema de Yee

O método de diferenças finitas no espaço e tempo (FDTD) é muito popular para a resolução das equações de Maxwell. Esse método foi introduzido por Kane Yee em 1966 [18] e ficou conhecido como esquema de Yee [16].

A ideia principal do esquema de Yee consiste no uso de duas malhas intercaladas para a aproximação das diferentes componentes: E_x, E_y e H_z. Dessa forma é possível aproximar as derivadas espaciais envolvidas usando diferenças finitas centradas. Isso permite introduzir uma discretização no tempo semelhante ao esquema *leapfrog* e finalmente obter um esquema de segunda ordem de precisão no tempo e no espaço [16].

Na discretização das equações, introduzimos a notação

$$u(n\Delta t, i\Delta x, j\Delta y) = u|_{i\,j}^{n}$$

em que u representa as componentes E_x, E_y ou H_z; Δx, Δy e Δt representam o tamanho do passo e o espaçamento nas discretizações do espaço e o tempo, respectivamente; n, i e j podem assumir valores inteiros ou fracionários (nesse último caso, eles estão associados com pontos da malha intercalada).

Na Figura 1, apresentamos as malhas usadas na discretização das componentes. Observamos que cada componente elétrica é rodeada pelas componentes magnéticas, o que proporciona um sistema explícito de fácil resolução (3).

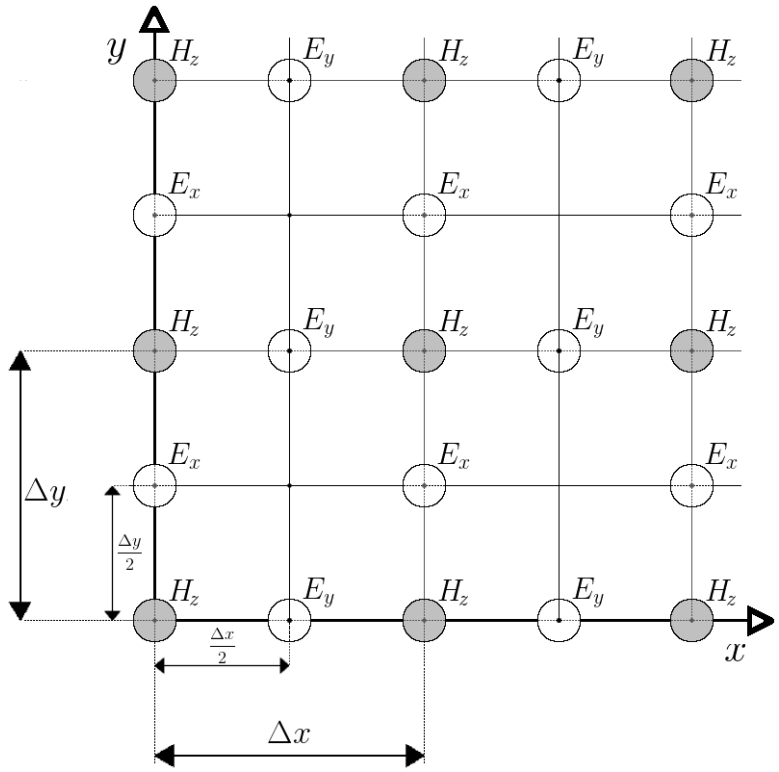


Figura 1: Malha de discretização correspondente ao esquema de Yee.

O sistema de equações discretizadas tem a forma

$$
\begin{aligned}
H_z|_{i;j}^{n+1} &= Da|_{i;j}\, H_z|_{i;j}^{n} + Db|_{i;j} \left\{ \begin{array}{l} \dfrac{E_x|_{i;j+\frac{1}{2}}^{n+\frac{1}{2}} - E_x|_{i;j-\frac{1}{2}}^{n+\frac{1}{2}}}{\Delta y} - \\ \dfrac{E_y|_{i+\frac{1}{2};j}^{n+\frac{1}{2}} - E_y|_{i-\frac{1}{2};j}^{n+\frac{1}{2}}}{\Delta x} - M_z|_{i;j}^{n+\frac{1}{2}} \end{array} \right\} \\
E_x|_{i;j+\frac{1}{2}}^{n+\frac{3}{2}} &= Ca|_{i;j+\frac{1}{2}} E_x|_{i;j+\frac{1}{2}}^{n+\frac{1}{2}} + Cb|_{i;j+\frac{1}{2}} \left\{ \frac{H_z|_{i+;j+1}^{n+1} - H_z|_{i;j}^{n+1}}{\Delta y} - J_x|_{i;j+\frac{1}{2}}^{n+1} \right\} \\
E_y|_{i+\frac{1}{2};j}^{n+\frac{3}{2}} &= Ca|_{i+\frac{1}{2};j} E_x|_{i+\frac{1}{2};j}^{n+\frac{1}{2}} + Cb|_{i+\frac{1}{2};j} \left\{ -\frac{H_z|_{i+1;j}^{n+1} - H_z|_{i;j}^{n+1}}{\Delta y} - J_y|_{i+\frac{1}{2};j}^{n+1} \right\}
\end{aligned}
\tag{3}
$$

onde

$$
Da|_{i;j} = \frac{1 - \frac{\sigma^*|_{i;j}\,\Delta t}{2\mu|_{i;j}}}{1 + \frac{\sigma^*|_{i;j}\,\Delta t}{2\mu|_{i;j}}}, \qquad Db|_{i;j} = \frac{\frac{\Delta t}{\mu|_{i;j}}}{1 + \frac{\sigma^*|_{i;j}\,\Delta t}{2\mu|_{i;j}}},
$$

$$
Ca|_{i;j} = \frac{1 - \frac{\sigma|_{i;j}\,\Delta t}{2\epsilon|_{i;j}}}{1 + \frac{\sigma|_{i;j}\,\Delta t}{2\epsilon|_{i;j}}}, \qquad Cb|_{i;j} = \frac{\frac{\Delta t}{\epsilon|_{i;j}}}{1 + \frac{\sigma|_{i;j}\,\Delta t}{2\epsilon|_{i;j}}}.
$$

Esse sistema está sujeito à condição de estabilidade de Courant-Friedrichs-Lewy (CFL) [16] dada por

$$
\Delta t \leq \frac{1}{\upsilon\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2}}},
$$

onde υ representa a velocidade de propagação das ondas eletromagnéticas no meio dada por $\upsilon = \frac{1}{\sqrt{\epsilon\mu}}$.

2.2 Condições de contorno absorventes

Em muitas aplicações é necessário simular a propagação de ondas eletromagnéticas em um meio não limitado. Entretanto, devido à impossibilidade de se usar uma malha computacional infinita, temos que aplicar condições de contorno absorventes. As condições de contorno absorventes permitem que as ondas que atingem a fronteira da região de interesse abandonem essa região sem provocar reflexões espúrias, introduzindo de forma correta um espalhamento para fora dessa região.

Uma forma muito eficiente de introduzir condições de contorno absorventes no método FDTD foi feita por Bérenger em 1994 [3], sua proposta é chamada de PML (*Perfectly Matched Layer*). Nessa abordagem, são introduzidas camadas extras de células (chamadas de PML) rodeando o domínio de interesse em quatro regiões: acima, à esquerda, à direita e abaixo, como ilustrado na

Figura 2. Em cada célula da PML, a condutividade elétrica é determinada de forma que as ondas eletromagnéticas penetrem sem reflexão na interface entre o domínio físico modelado e a PML, para qualquer frequência de onda e ângulo de incidência. Isso é feito respeitando-se a condição

$$\frac{\sigma}{\epsilon} = \frac{\sigma^*}{\mu}, \tag{4}$$

de tal forma que a impedância do meio modelado coincida com a impedância da PML, garantindo-se que a reflexão seja zero.

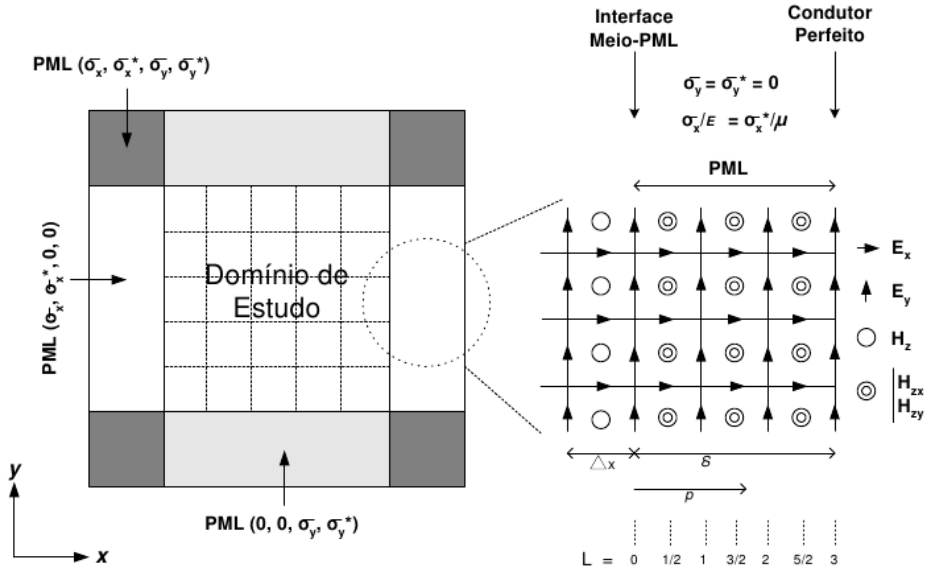


Figura 2: TE_z-PML ao redor do domínio de análise. Fonte: [13].

Para a formulação em duas fases da PML de Bérenger, a componente H_z é dividida em H_{zx} e H_{zy}. Assim, as equações de Maxwell modificadas para o caso TE_z ficam na seguinte forma

$$\begin{aligned}
\epsilon\frac{\partial E_x}{\partial t} + \sigma_y E_x &= \frac{\partial H_z}{\partial y} \\
\epsilon\frac{\partial E_y}{\partial t} + \sigma_x E_y &= -\frac{\partial H_z}{\partial y} \\
\mu\frac{\partial H_{zx}}{\partial t} + \sigma_x^* H_{zx} &= -\frac{\partial E_y}{\partial x} \\
\mu\frac{\partial H_{zy}}{\partial t} + \sigma_y^* H_{zy} &= \frac{\partial E_x}{\partial y}
\end{aligned} \tag{5}$$

Observe que a extensão do esquema de Yee (3) para o sistema acima é bastante simples e direta. Entretanto, para garantir uma correta implementação dessas condições de contorno absorventes

é necessário determinar o número de camadas que irão compor a PML e as condutividades correspondentes, de acordo com o grau de reflexão desejado. Em princípio, a taxa de reflexão pode se tornar tão pequena quanto desejável, controlando-se a espessura da PML.

Existem várias formas para se determinar a variação espacial da condutividade dentro das camadas da PML, por exemplo: linear, parabólica, polinomial e geométrica [3, 16]. As formas polinomial e geométrica têm apresentado os melhores resultados [16]. Por isso, neste trabalho, utilizaremos a PML polinomial. Para seu cálculo, seguimos os seguintes passos:

- Cálculo da condutividade da PML na última camada. Trata-se da camada mais externa que por sua vez apresenta a maior condutividade cujo valor é dado pela equação

$$\sigma_{\max} = -\frac{(m+1)\ln(R(0))}{2\eta d}$$

onde m é o grau do polinômio para a modelagem de σ, geralmente 3 ou 4; $R(0)$ é o fator de reflexão; $\eta = \sqrt{\frac{\mu}{\epsilon}}$ é a impedância intrínseca do meio; e d é a espessura da PML dada por Δx ou Δy multiplicado pelo número de camadas.

Sendo N o número de camadas da PML, de acordo com [16] para $N = 10$ usa-se $R(0) = e^{-16}$ e para $N = 5$, $R(0) = e^{-8}$. Esses valores do fator de reflexão são considerados ótimos.

- Cálculo da condutividade nas demais camadas da PML. O valor da condutividade na direção do eixo x é dado pelas equações

$$\sigma_x(k) = \sigma_{\max}\,(k\Delta x/d)^m\,, \tag{6}$$

onde $k = 0, 1, 2, \ldots, N$ é o índice da camada, considera-se $k = 0$ para a camada que compõe a fronteira do domínio de análise e $k = N$ para a camada mais externa da PML. Após o cálculo de $\sigma_x(k)$ para todas as camadas da PML, devemos utilizar a relação (4) para determinar os $\sigma_x^*(k)$ correspondentes.

O cálculo da condutividade na direção do eixo y é realizado de forma análoga. Observe na Figura 2 como são associados os valores dessas condutividades às diferentes regiões que formam a PML.

2.3 Algoritmo sequencial do esquema de Yee

Os fluxogramas apresentados na Figura 3 descrevem os algoritmos sequencias que usaremos como referência para a nossa avaliação. O primeiro usa as condições de contorno de tipo Dirichlet (específicas) e o segundo PML.

Para o primeiro algoritmo, com condições de contorno específicas, temos uma etapa de inicialização das variáveis onde são definidas as propriedades do meio e a discretização do domínio em malhas. Em seguida, a cada passo do tempo, o algoritmo calcula as condições de contorno,

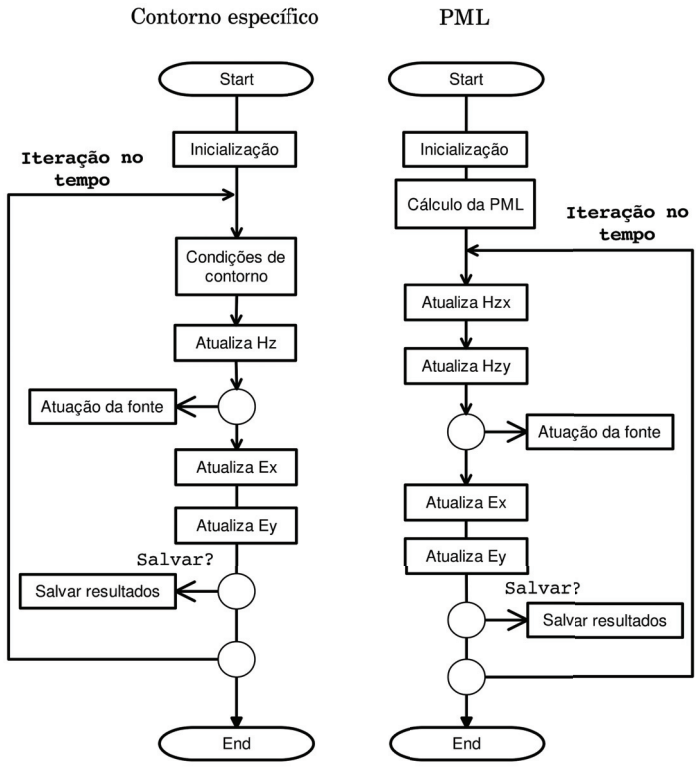


Figura 3: Algoritmos sequenciais do esquema de Yee considerando condições de contorno de tipo Dirichlet (esquerda) e PML (direita).

atualiza os valores de H_z, da fonte, e dos campos elétricos E_x e E_y. A alternativa de salvar os resultados em disco é opcional e pode ser feita de forma cíclica para economia de espaço e redução do alto custo de escrita em disco. Por exemplo, podemos salvar as soluções intermediárias a cada 100 passos no tempo.

Para o segundo algoritmo, com PML, após a inicialização temos uma etapa adicional de cálculo da PML e definição da camada extra de células. As condições de contorno não precisam ser especificadas. A cada passo do tempo, o algoritmo calcula as duas componentes de H_z (H_{zx} e H_{zy}), e a partir daí segue a mesma sequência de passos do primeiro algoritmo. O cálculo da componente extra de H_z demanda cerca de 33% a mais de espaço de memória.

3 IMPLEMENTAÇÃO PARALELA DO ESQUEMA DE YEE EM GPU USANDO CUDA

Como discutido no início deste texto, apesar de simples, o método de Yee é muito dispendioso computacionalmente devido às restrições na discretização temporal necessárias para garantir a estabilidade numérica. Nesta seção, apresentamos uma proposta de paralelização do método de Yee (em sua forma bidimensional) para execução em GPUs, usando a plataforma CUDA.

As unidades de processamento gráfico (GPUs – *Graphics Processing Units*) possuem centenas de unidades de processamento oferecendo um novo tipo de ambiente para programação paralela com um custo financeiro relativamente baixo [11]. Aplicações com forte paralelismo de dados – como é o caso do método FDTD – são boas candidatas para usufruirem dessa capacidade de processamento paralelo das GPUs. Na próxima seção, apresentamos uma breve descrição da plataforma CUDA e, em seguida, descrevemos o algoritmo paralelo proposto.

3.1 CUDA

Desde 2006, a NVIDIA (um dos principais fabricantes de placas gráficas) disponibiliza uma plataforma de computação paralela, juntamente com um modelo de programação, para o desenvolvimento de aplicações paralelas que executam nas GPUs, chamado CUDA (*Compute Unified Device Architecture*) [4]. CUDA permite o uso das linguagens C e Fortran, oferecendo um conjunto de funções adicionais para acessar e gerenciar o processamento de dados na GPU.

Um programa em CUDA é dividido em duas partes: uma parte que executa na CPU (ou *host*) e outra parte que executa na GPU (ou *device*). A parte que executa na CPU é responsável pela entrada e saída de dados e pela gerência de acesso à GPU. O acesso à GPU envolve: a reserva e liberação de espaço de memória; a transferência de dados da memória principal do computador para a memória da GPU e vice-versa; e a chamada de funções que executam na GPU, denominadas *kernels*. A parte que executa na GPU é tipicamente responsável pelo processamento principal da aplicação e pode ser implementada em um ou vários *kernels*.

No modelo de programação disponibilizado por CUDA, quando uma função *kernel* é disparada para execução na GPU, essa função é executada por várias unidades de processamento ao mesmo tempo, criando fluxos de execução distintos denominados *threads*. Internamente, cada *thread* usa seu identificador único para selecionar quais dados ela irá processar.

Por exemplo, para incrementar todos os elementos de um vetor de 1, a função *kernel* deverá implementar um único incremento fazendo $vetor[id] = vetor[id] + 1;$, onde id será o identificador de cada *thread*. Quando o *kernel* for disparado para execução por um número N de *threads* (onde N corresponde ao número de elementos do vetor), cada *thread* incrementará um elemento do vetor, correspondente ao seu identificador.

CUDA agrupa as *threads* em **blocos** e os blocos em uma **grade**. Na Figura 4 podemos visualizar a relação entre *threads*, blocos e grade. Uma grade possui vários blocos e um bloco possui várias *threads*. Sempre que um *kernel* é disparado, o programador define quantas *threads* irão executar esse *kernel* e como elas serão organizadas em blocos (quantas *threads* por bloco). Os blocos podem ser unidimensionais, bidimensionais ou tridimensionais, significando que as *threads* terão identificadores com 1, 2 ou 3 índices, respectivamente. Por sua vez, uma grade também pode ter até três dimensões, significando que os blocos poderão ter 1, 2 ou 3 índices de identificação. A Figura 4 ilustra o caso de uma grade bidimensional com 8 blocos também bidimensionais, cada um com 12 *threads*.

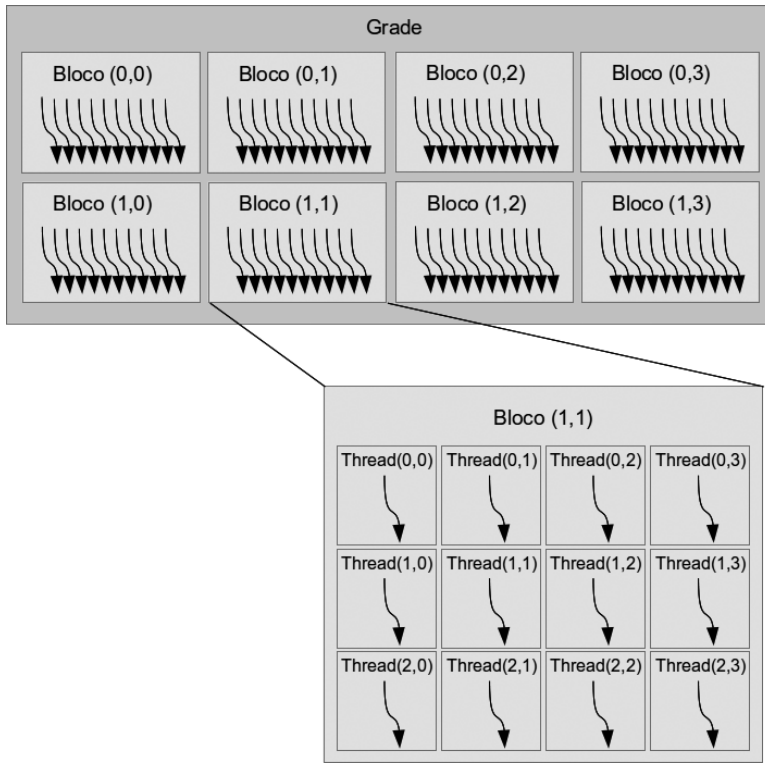


Figura 4: Organização das *threads* no ambiente CUDA.

A estrutura de memória da GPU abrange vários tipos de memória, com diferentes tamanhos e velocidades de acesso. Os **registradores** são a memória de acesso mais rápido e são alocados com exclusividade para cada *thread*. Dentro de um bloco, as *threads* compartilham o espaço de **memória compartilhada** (*shared memory*), exclusivo de cada bloco. A **memória global** pode ser acessada por todas as *threads* de uma grade e também pela CPU. Trata-se da memória mais lenta e de maior capacidade de armazenamento.

No modelo de execução de CUDA, as *threads* de um bloco são escalonadas para execução em grupos chamados *warps* (um *warp* contém 32 *threads*). Nesse modelo, uma instrução é carregada de cada vez e todas as *threads* do *warp* executam a mesma instrução, aplicando-a sobre a parte dos dados que compete a cada *thread*.

O fluxo de execução de um programa CUDA segue, tipicamente, os seguintes passos: lê os dados de entrada e faz as inicializações necessárias; reserva espaço de memória na GPU; transfere os dados de entrada da memória da CPU para a memória global da GPU; executa um (ou mais) *kernels*; transfere os dados processados da memória da GPU para a memória da CPU; exibe os resultados e executa as finalizações necessárias.

3.2 Algoritmo paralelo em CUDA

Neste trabalho, propomos uma alternativa de paralelização do esquema de Yee para execução em GPU, tomando como base o algoritmo proposto em [6]. O fluxo principal de execução do algoritmo é apresentado na Figura 5, onde a alternativa de salvar os resultados em disco é opcional. Assim como nas versões sequenciais, implementamos duas versões do algoritmo paralelo, uma usando condições de contorno específicas e outra usando PML. Em ambos os casos, o controle das iterações no tempo é feito na CPU que dispara três *kernels* a cada passo de tempo.

Na versão com condições de contorno específicas, o primeiro *kernel* calcula as condições de contorno, o segundo *kernel* calcula os valores de H_z e o terceiro *kernel* calcula os valores de Ex e Ey. Para a versão com PML, os dois primeiros *kernels* atualizam os valores dos campos H_{zx} e H_{zy} e o terceiro *kernel* calcula as atualizações de E_x e E_y.

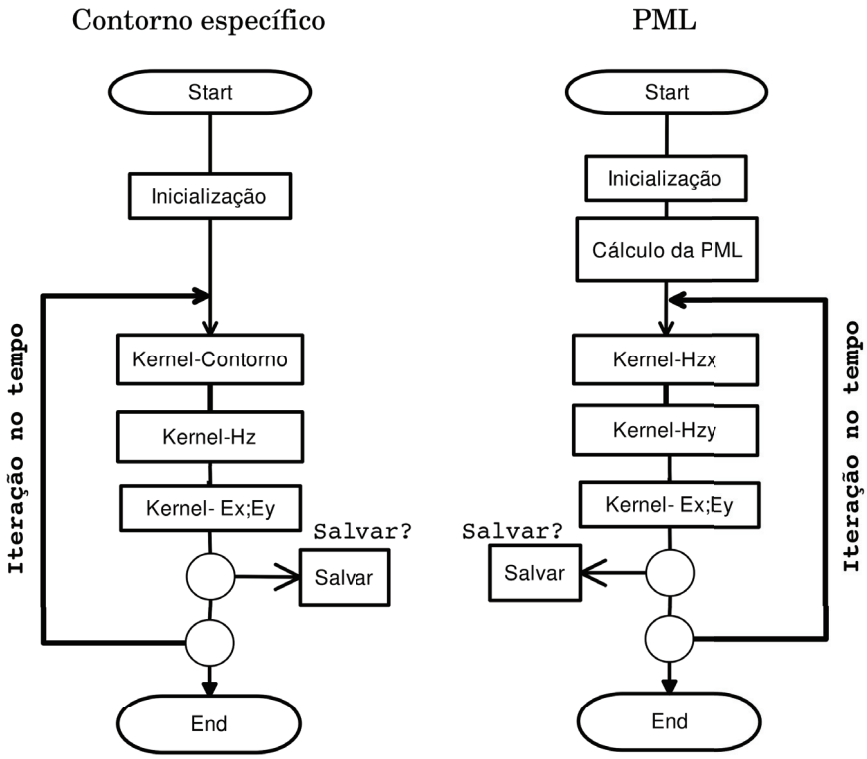


Figura 5: Fluxogramas dos algoritmos paralelos considerando condições de contorno de tipo Dirichlet (esquerda) e PML (direita).

Internamente, a implementação dos *kernels* faz com que cada *thread* seja responsável pelo cálculo de uma posição da malha espacial dos campos. A dimensão dos blocos é um parâmentro de entrada.

Seguindo as sugestões de otimização do uso da memória da GPU propostas em outros trabalhos [2, 17, 6], experimentamos o uso da memória compartilhada entre *threads* do mesmo bloco para armazenar valores que são acessados mais de uma vez em um mesmo *kernel*. Esta abordagem apresentou ganhos apenas para o *kernel* que atualiza os valores de E_x e E_y, visto que ambos utilizam informações de H_z.

4 AVALIAÇÃO DAS IMPLEMENTAÇÕES PROPOSTAS

Avaliamos as implementações propostas com dois objetivos: verificar a corretude dessas implementações e determinar o ganho de desempenho obtido com a versão paralela. Para avaliar a corretude das implementações, estudamos a acurácia das soluções numéricas obtidas. Para medir o ganho de desempenho obtido, comparamos os resultados das implementações sequenciais e paralelas dos algoritmos apresentados.

4.1 Corretude das implementações propostas

Apresentamos dois exemplos que mostram a corretude do algoritmo paralelo proposto. No primeiro exemplo estudamos um caso onde a solução analítica é conhecida, permitindo uma análise quantitativa da exatidão da solução numérica, e no segundo exemplo avaliamos o efeito da *PML*.

4.1.1 Exemplo 1 – estudo da acurácia

Considerando que $\epsilon = \mu = 1$, é simples verificar que as seguintes funções

$$\begin{aligned} E_x(t, x, y) &= A \sin(\sqrt{3}t) \sin(x + \sqrt{2}y) \\ E_y(t, x, y) &= -\frac{A}{\sqrt{2}} \sin(\sqrt{3}t) \sin(x + \sqrt{2}y) \\ H_z(t, x, y) &= -\frac{2A}{3} \cos(\sqrt{3}t) \cos(x + \sqrt{2}y) \end{aligned} \tag{7}$$

satisfazem as equações (2) para qualquer valor de A.

Essa solução representa uma onda plana estacionária e foi usada para avaliar a acurácia das aproximações numéricas em nossas implementações do esquema de Yee (3).

As soluções numéricas de (3) foram obtidas para o domínio retangular $\Omega = [0, 2\pi] \times [0, \sqrt{2}\pi]$ e o intervalo de tempo $[0, 4]$. Usamos as condições iniciais definidas através de (7) (com $t = 0$) e condições de contorno de tipo Dirichlet para H_z definidas pela terceira equação de (7). Para a avaliação da exatidão das aproximações, calculamos o erro na norma do máximo $\| \cdot \|_\infty$ para diferentes escolhas da malha discreta com o intuito de mostrar sua convergência para zero na medida em que a malha vai se tornando mais fina.

Nas simulações numéricas consideramos malhas tais que $\Delta x = \frac{2\Delta y}{\sqrt{2}}$ e $\Delta t \leq \Delta x/\sqrt{3}$ (condição CFL). Dessa forma esperamos $\|E\|_\infty = O((\Delta x)^2)$, ou seja, uma taxa de convergência quadrática. A taxa de convergência (empírica) observada nas simulações pode ser determinada como

$$q = \frac{\log\left(\frac{\|E_{j+1}\|_\infty}{\|E_j\|_\infty}\right)}{\log\left(\frac{\Delta x_{j+1}}{\Delta x_j}\right)} \tag{8}$$

onde $\|E_j\|_\infty$, Δx_j e $\|E_{j+1}\|_\infty$, Δx_{j+1} representam os valores do erro e do espaçamento da malha nas j-ésima e $j+1$-ésima simulações, respectivamente.

Desse modo, a partir dos resultados numéricos é possível identificar a taxa de convergência empírica para avaliar se os resultados numéricos apresentam o comportamento teórico esperado.

A Tabela 1 apresenta os resultados de 8 simulações, variando-se o tamanho da discretização. A tabela mostra os erros observados e a taxa de convergência estimada conforme (8). De forma geral, os erros obtidos são inferiores a $(\Delta x)^2$. Além disso, observe que a ordem de precisão estimada se mostrou maior do que dois e com um comportamento crescente à medida que o tamanho da malha cresce, implicando numa precisão acima do esperado. Assim, dado que os resultados encontrados pela solução analítica estão próximos da solução exata dentro de uma margem de erro inferior ao limite teórico, podemos concluir que a solução numérica está sendo calculada corretamente para esse exemplo.

Tabela 1: Erro e da taxa de convergência empírica para diferentes discretizações.

j	Δt_j	Δx_j	$\|E\|_{\infty,j}$	q (empírico)
1	4×10^{-3}	$6{,}35\times10^{-2}$	$1{,}34\times10^{-3}$	–
2	4×10^{-3}	$3{,}16\times10^{-2}$	$3{,}22\times10^{-4}$	2.053
3	4×10^{-3}	$2{,}10\times10^{-2}$	$1{,}34\times10^{-4}$	2.161
4	4×10^{-3}	$1{,}57\times10^{-2}$	$6{,}78\times10^{-5}$	2.347
5	4×10^{-3}	$1{,}26\times10^{-2}$	$3{,}74\times10^{-5}$	2.652
6	4×10^{-4}	$1{,}05\times10^{-2}$	$2{,}10\times10^{-5}$	3.167
7	4×10^{-4}	$8{,}99\times10^{-3}$	$1{,}11\times10^{-5}$	4.136
8	4×10^{-4}	$7{,}86\times10^{-3}$	$4{,}46\times10^{-6}$	6.483

4.1.2 Exemplo 2 – validação da PML

Nesse exemplo, vamos mostrar que a nossa implementação da PML apresenta os resultados esperados. Para isso simulamos numericamente a propagação de uma onda cilíndrica, gerada por um pulso pontual, e determinamos a quantidade de energia espúria que foi refletida devido à presença da PML. A energia refletida é calculada usando uma outra solução aproximada do mesmo problema sem PML, mas em um domínio suficientemente extenso para que as condições de fronteira não tenham nenhuma influência [16].

Consideramos $\epsilon = 8.854 \times 10^{-12}$, $\mu = 4\pi \times 10^{-7}$, logo $\upsilon = (\epsilon\mu)^{-1/2} \approx 3 \times 10^8$, e usamos discretizações com $\Delta x = \Delta y = \Delta = 0.01$ e $\Delta t = \frac{\Delta}{2\upsilon}$. A primeira malha possui 50×50 células (discretização da região Ω_0) e uma PML com N camadas. Nas simulações usamos uma PML com $N = 5$ e $N = 10$, com uma variação polinomial da condutividade de grau $m = 4$. As aproximações obtidas nesse caso são representadas por $H_z^{\mathrm{PML}}|_{i,j}^n$. Na segunda malha usamos 200×200 células e não consideramos PML, a parte central dessa malha corresponde à discretização da região Ω_0. As aproximações correspondentes são representadas por $H_z^{\infty}|_{i,j}^n$. No ponto central de Ω_0 atua uma fonte pontual com intensidade

$$H_z|^n = \begin{cases} \frac{1}{32}\left\{10 - 15\cos(n\pi/20) + 6\cos(2n\pi/20) - \cos(3n\pi/20)\right\}, & n \leq 40, \\ 0, & n > 40. \end{cases}$$

Após calcular a propagação desse pulso durante $N_t = 200$ passos de tempo, considerando condições iniciais nulas, a energia média refletida para o interior da região Ω_0 ao longo da sua fronteira $\partial\Omega_0$ no passo de tempo n, devido à presença da PML, pode ser calculada por

$$\mathcal{E}_{\mathrm{ref}}^n = 1/N_f \sum_{(x_i,y_j)\in\partial\Omega_0} \left[H_z^{PML}|_{i,j}^n - H_z^{\infty}|_{i,j}^n\right]^2,$$

em que N_f é a quantidade de pontos em $\partial\Omega_0$. Dessa forma a presença da PML produz um erro adicional na região Ω_0 que pode ser calculado como

$$E_{\mathrm{PML}}^n = 1/N_d \sum_{(x_i,y_j)\in\Omega_0} \left[H_z^{PML}|_{i,j}^n - H_z^{\infty}|_{i,j}^n\right]^2,$$

em que N_d é a quantidade de pontos em Ω_0.

Na Figura 6, apresentamos as curvas que mostram a energia média refletida ao longo da fronteira da PML quando ela contem 5 e 10 camadas. Observamos que após a onda atingir a fronteira começam a aparecer as reflexões espúrias que depois decrescem lentamente se estabilizando em torno do valor 10^{-5}. Esse valor é compatível com os erros numéricos detectados no primeiro exemplo.

Na Figura 7, as curvas mostram o erro médio quadrático na região Ω_0. Podemos observar como as reflexões espúrias causadas pela PML produzem um aumento dos erros, mas estes também se estabilizam em torno do valor 10^{-5}. Ambas as figuras indicam que o resultado para o caso de 10 camadas foi melhor que o caso de 5 camadas.

4.2 Avaliação do desempenho das implementações propostas

Para avaliar os algoritmos propostos, usamos um computador com a seguinte configuração: processador *quad-core* Intel i7-960 3.2 GHz; 32 Kb de Cache L1, 256 Kb de Cache L2 e 8 Mb de Cache L3; 16 Gb de memória RAM DDR3 1067 MHz. A GPU usada foi a NVIDIA Tesla C2070 com 6 GB GDDR5, 448 unidades de processamento, permitindo até 1024 *threads* por blocos. O computador possui sistema operacional Ubuntu 14.04; e os compiladores GCC 4.8.2 e NVCC 5.5.

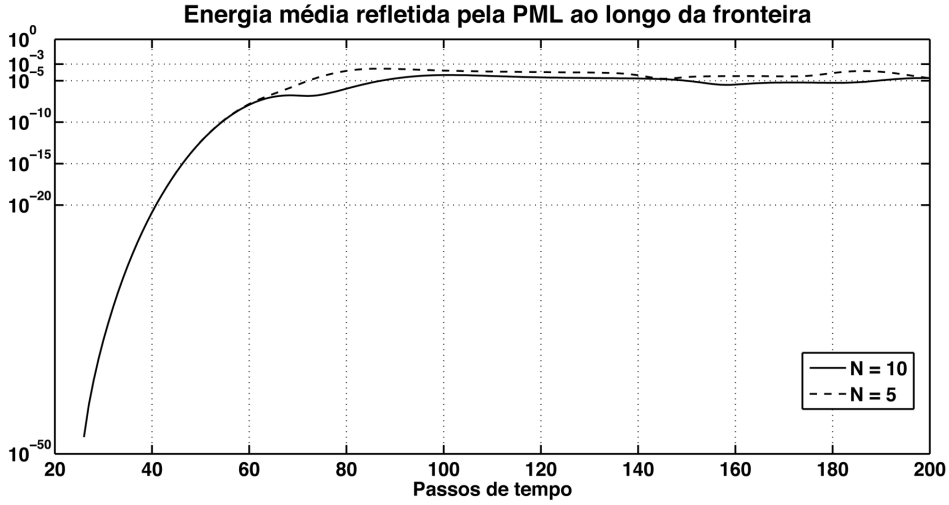


Figura 6: Energia média refletida pela PML para dentro da região Ω_0 quando a PML contem 5 e 10 camadas.

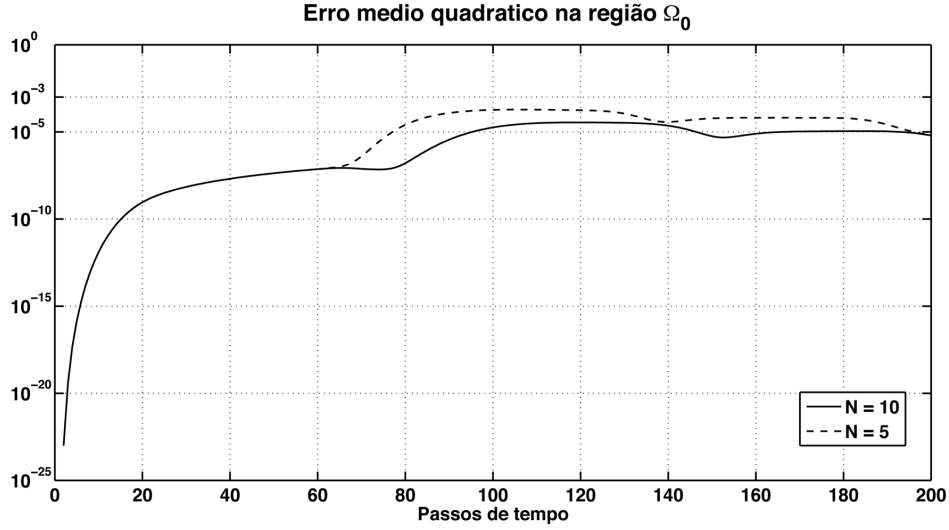


Figura 7: Erro médio quadrático na região Ω_0 devido à presença de PMLs contendo 5 e 10 camadas.

Variamos o tamanho da malha da discretização espacial de 300×300 até 6900×6900 (tamanho máximo limitado pelo espaço de memória global da GPU usada, considerando precisão dupla e uma estrutura de algoritmo preparada para espaços heterogêneos). Em todos os experimentos consideramos 1000 iterações no tempo.

Usamos como métricas de avaliação os seguintes valores: **tempo de execução** (em segundos), **speedup** (razão entre o tempo do algoritmo sequencial e o tempo do algoritmo paralelo), e **speed[MCells/s]** [6] (milhões de células processadas por unidade de tempo) dada por

$$Speed[Mcells/s] = \frac{nx\,ny\,nt}{10^6 T} \tag{9}$$

onde nx e ny são as dimensões da malha de nós utilizada para a discretização, nt é o número de iterações no tempo e T é o tempo de execução da implementação (medido em segundos).

4.3 Avaliação do algoritmo sequencial

Os algoritmos sequenciais foram implementados na linguagem C e compilados com o compilador `gcc`. Para melhorar o desempenho dessas implementações, utilizamos as opções de otimização oferecidas pelo próprio compilador `gcc`. Estas opções permitem um ganho de desempenho bastante considerável [14]. Para efeito de comparação, usamos as opções $-O0$ (sem otimização), $-O1$, $-O2$ e $-O3$, gerando quatro arquivos executáveis que foram experimentados separadamente. Os tempos de execução são mostrados na Tabela 2.

Tabela 2: Tempo de processamento do algoritmo sequencial para os dos tipos de condições de contorno implementadas.

	C.C. tipo Dirichlet				PML			
	O0	O1	O2	O3	O0	O1	O2	O3
300	2.53	0.74	0.65	0.60	6.17	2,92	2.73	2.27
1500	63.45	25.85	25.71	25.31	94.14	40.47	39.65	38.38
2100	124.62	51.23	50.89	50.43	178.71	74.77	72.51	72.00
2700	205.50	81.73	80.21	79.46	292.79	122.91	121.96	121.06
3300	307.25	122.49	120.46	120.20	429.85	174.33	172.33	170.18
3900	428.98	173.68	171.69	170.24	597.89	250.90	249.52	247.83
4500	570.81	232.64	229.93	228.73	791.64	334.84	327.62	325.17
5100	732.46	298.28	294.58	292.24	1014.02	431.95	427.09	421.52
5700	920.01	389.77	383.73	383.54	1263.45	525.73	515.99	511.94
6300	1117.02	445.81	438.78	438.28	1545.79	633.14	629.19	620.51
6900	1344.99	558.39	552.73	549.94	1874.89	901.37	876.68	882.07

Note pelos resultados obtidos que o uso das otimizações do compilador `gcc` resultaram em uma redução significativa do tempo de execução do algoritmo sequencial. Utilizaremos o menor tempo obtido com o algoritmo sequencial (otimização $O3$) como *benchmarck* para o cálculo das demais métricas.

Observe que o tempo de processamento do algoritmo com PML foi significativamente superior ao tempo de processamento do algoritmo com as condições de contorno do tipo Dirichlet. Isso

ocorre em razão da divisão do campo H_z em duas componentes e da necessidade de calcular camadas extras de células. O código com PML demora em média 43% a mais do que a versão com Dirichlet, valor similar ao aumento da memória dado pelo uso da PML.

4.4 Avaliação do algoritmo paralelo

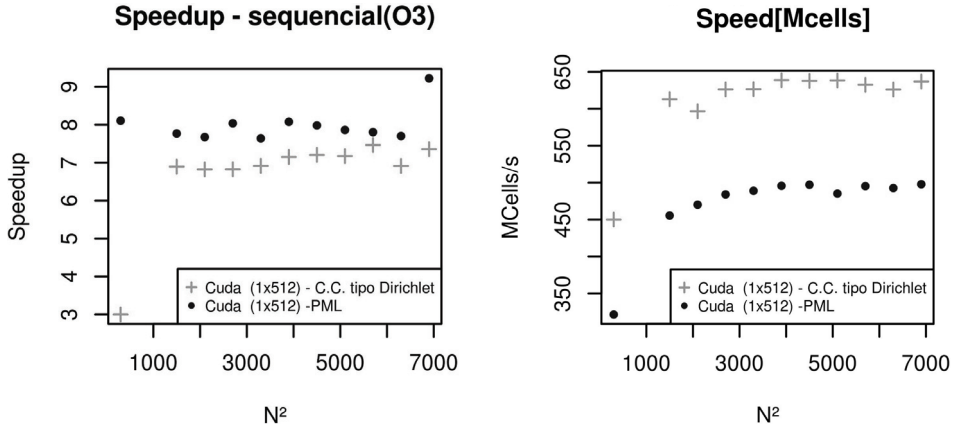
A implementação do algoritmo paralelo em CUDA tem como parâmetro de entrada a dimensão dos blocos de threads, uma vez que o desempenho do algoritmo é afetado por esse valor. Na Tabela 3, mostramos o desempenho das quatro dimensões de blocos que apresentaram os melhores resultados para as duas versões do algoritmo (Dirichlet e PML). Note que os melhores resultados foram para blocos linha, em particular o bloco com dimensão 1×512 foi o que obteve, em geral, o melhor resultado para ambos os algoritmos.

Tabela 3: Tempo de processamento do algoritmo paralelo para os dos tipos de condições de contorno implementadas.

	C.C. tipo Dirichlet				PML			
	1×128	1×256	1×512	1×1024	1×128	1×256	1×512	1×1024
300	0,23	0,18	0,20	0,28	0.41	0.27	0.28	0.38
1500	4.24	3.70	3.67	3.99	4.88	4.78	4.94	5.13
2100	7.16	6.99	7.39	8.21	9.33	8.98	9.38	10.34
2700	12.17	11.83	11.64	12.18	16.70	15.32	15.06	15.74
3300	18.47	17.44	17.38	18.51	23.64	22.57	22.27	23.71
3900	24.70	23.82	23.81	24.68	31.54	30.43	30.68	31.93
4500	33.17	31.82	31.75	33.75	43.01	40.99	40.73	43.53
5100	42.52	41.45	40.74	41.96	55.20	53.70	53.61	56.53
5700	53.29	51.80	51.37	53.09	69.20	66.72	65.59	68.41
6300	65.76	63.85	63.39	66.34	83.45	80.80	80.56	85.25
6900	78.20	75.79	74.75	77.26	100.18	97.36	95.63	99.66

A Figura 8 é composta por dois gráficos relativos aos melhores resultados do códigos com condições de contorno de Dirichlet e PML. O primeiro apresenta os resultados de *speedup* e o segundo o número de milhões de células calculadas por unidade de tempo, variando-se as dimensões da malha de discretização. Note que a partir da dimensão 1500×1500 os resultados se mostraram estáveis.

Na Tabela 4 temos os valores médios dessas mesmas métricas, além de uma medida de dispersão, para a qual foi adotada a amplitude dos resultados observados (diferença entre o maior valor obtido reduzido do menor). Os algoritmos paralelos foram de 7 a 8 vezes mais rápidos do que os algoritmos sequenciais otimizados.



(a) Speedup (b) Speed (Mcells/s)

Figura 8: Desempenho dos algoritmos paralelos.

Tabela 4: Algoritmo sequencial versus algoritmo paralelo.

	Sequencial(O3)		Cuda(1×512)	
	Dirichlet	PML	Dirichlet	PML
Speed[MCells/s]	89,04 (7.03)	61,08 (10.02)	663,13 (42.05)	486,55(42.39)
Speedup	–	–	7,08 (0,64)	7.98 (1,58)

5 CONCLUSÕES

Neste trabalho, apresentamos uma proposta de paralelização do esquema de diferenças finitas no domínio do tempo (FDTD) de Yee para simulação das equações de Maxwell, usando GPUs. Consideramos duas alternativas de implementação do algoritmo proposto, uma usando condições de contorno específicas de Dirichlet e outra usando condições de contorno absorventes com PML.

Para avaliar o desempenho dos algoritmos paralelos, implementamos versões sequenciais correspondentes para execução em CPU. As implementações propostas também foram validadas através de exemplos para garantir a corretude dos seus resultados. Por fim, realizamos uma série de avaliações com diversas dimensões de malha para estimar o desempenho das implementações em ambos os ambientes.

Nosso propósito principal foi buscar uma alternativa de implementação paralela para o esquema de Yee que permita reduzir o tempo total de processamento requerido, facilitando a simulação numérica das equações de Maxwell. Os algoritmos propostos permitem também salvar os resultados intermediários da simulação, quando necessário. Os resultados obtidos mostraram ganhos de desempenho bastante significativos, de 7 a 8 vezes, quando comparados com os resultados do código sequencial otimizado.

AGRADECIMENTOS

Os autores agradecem o apoio financeiro da FAPERJ (E-26/110.721/2013).

ABSTRACT. Maxwell's equations play a crucial role in electromagnetic theory and applications. However, it is not always possible to solve these equations analytically. Consequently, we have to use numerical methods in order to get approximate solutions of the Maxwell's equations. The FDTD (Finite-diference Time-Domain) method, proposed by K. Yee, is widely used to solve Maxwell's equations, due to its efficiency and simplicity. However, this method has a high computational cost. In this paper, we propose a parallel implementation of the FDTD method to run on GPUs by using CUDA platform. Our goal is to reduce the processing time required, allowing the use of the FDTD method in the simulation of electromagnetic wave propagation. We evaluate the proposed algorithm considering two different kind of boundary conditions: a Dirichlet type boundary conditions and absorbing boundary conditions. We get a performance gain ranging from 7 to 8 times when comparing the proposed parallel implementation with an optimized sequential version.

Keywords: Maxwell's equations, Yee algorithm, Parallel Programming with GPU.

REFERÊNCIAS

[1] S. Adams, J. Payne & R. Boppana. Finite difference time domain (FDTD) simulations using graphics processors. In DoD High Performance Computing Modernization Program Users Group Conference, pp. 334–338. IEEE (2007).

[2] A. Balevic, L. Rockstroh, A. Tausendfreund, S. Patzelt, G. Goch & S. Simon. Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose GPUs. In 11th IEEE International Conference on Computational Science and Engineering, pp. 327–334. IEEE (2008).

[3] J. Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, **114**(2) (1994), 185–200.

[4] NVIDIA CUDA, https://developer.nvidia.com/cuda-zone, acessado em 20 de setembro de 2015.

[5] V. Demir & A.Z. Elsherbeni. Compute unified device architecture (CUDA) based finite-difference time-domain (FDTD) implementation. *ACES*, **25**(4) (2010).

[6] D.D. Donno, A. Esposito, G. Monti, L. Catarinucci & L. Tarricone. GPU-based acceleration of computational electromagnetics codes. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, **26**(4) (2013), 309–323.

[7] G. El Zein & A. Khaleghi. Emerging Wireless Communication Technologies. In New Technologies, Mobility and Security, pp. 271-279. Springer Netherlands (2007).

[8] J.W. Hand. Modelling the interaction of electromagnetic fields (10 MHz-10 GHz) with the human body: methods and applications. *Physics in Medicine and Biology*, **53**(16) (2008), R243.

[9] M.J. Inman, A.Z. Elsherbeni & C.E. Smith. FDTD calculations using graphical processing units. In IEEE/ACES International Conference on Wireless Communications and Applied Computational Electromagnetics, pp. 728–731. IEEE (2005).

[10] E. Kashdan & B. Galanti. A new parallelization strategy for solving time dependent 3D Maxwell equations using a high-order accurate compact implicit scheme. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, **19**(5) (2006), 391–408.

[11] D.B. Kirk & W.H. Wen-mei. Programming massively parallel processors: a hands-on approach. Newnes (2012).

[12] S.E. Krakiwsky, L.E. Turner & M.M. Okoniewski. Acceleration of finite-difference time-domain (FDTD) using graphics processor units (GPU). In IEEE MTT-S International Microwave Symposium Digest, 2, pp. 1033–1036. IEEE (2004).

[13] C.B. Lima. Análise de dispositivos eletromagnéticos para hipertermia usando o método FDTD. Tese de Doutorado, PGEEL, UFSC, Florianópolis, SC (2006).

[14] R.M. Stallman. Using the GNU Compiler Collection. GNU Press (2010).

[15] D.M. Sullivan. Electromagnetic simulation using the FDTD method. John Wiley & Sons (2013).

[16] A. Taflove & S.C. Hagness. Computational Electrodynamics: The Finite-Difference Time-Domain Method. Artech House, London, 2 ed (2000).

[17] A. Valcarce, G. de la Roche & J. Zhang. A GPU approach to FDTD for radio coverage prediction. In 11th IEEE Singapore International Conference on Communication Systems, pp. 1585–1590. IEEE (2008).

[18] K. Yee. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *IEEE Trans Antennas Propag.*, **14**(3) (1966), 302–307.