Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Simulazione elettromagnetica tramite il metodo FDTD: implementazione in ambiente computazionale avanzato

G. Ala, C. Di Gesaro, E. Francomano, P. Storniolo

Rapporto Tecnico N.:
RT-ICAR-PA-12-04 **Dicembre 2012**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)
– Sede di Cosenza, Via P. Bucci 41C, 87036 Rende, Italy, URL: *www.icar.cnr.it*
– Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*
– Sede di Palermo, Viale delle Scienze, 90128 Palermo, URL: *www.pa.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Simulazione elettromagnetica tramite il metodo FDTD: implementazione in ambiente computazionale avanzato

G. Ala[2], C. Di Gesaro[2], E. Francomano[3], P. Storniolo[1]

Rapporto Tecnico N.:
RT-ICAR-PA-12-04

Data:
06 Dicembre 2012

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Palermo, Viale delle Scienze edificio 11, 90128 Palermo.
[2] Università degli Studi di Palermo, Dipartimento di Ingegneria Elettrica, Elettronica e delle Telecomunicazioni, di tecnologie Chimiche, Automatica e modelli Matematici, Viale delle Scienze, 90128 Palermo.
[3] Università degli Studi di Palermo, Dipartimento di Ingegneria Chimica, Gestionale, Informatica, Meccanica, Viale delle Scienze, 90128 Palermo.

Indice

Introduzione

L'elettromagnetismo computazionale si è evoluto e perfezionato rapidamente in questi ultimi anni sia grazie alla crescente velocità di elaborazione dei moderni calcolatori che attraverso l'implementazione di metodi di calcolo sempre più sofisticati. Tra i metodi di calcolo esistenti, quello delle differenze finite nel dominio del tempo (FDTD[1]) ricopre sicuramente un ruolo di rilievo per la sua versatilità e la relativa semplicità con la quale è possibile realizzare algoritmi per la simulazione di fenomeni elettromagnetici complessi, anche in presenza di mezzi non omogenei, non lineari e anisotropi[2]. L'obiettivo della presente tesi consiste nell'analisi di tecniche implementative basate sul metodo FDTD, per la simulazione di fenomeni elettromagnetici. Specificamente, si è rivolta particolare attenzione alla possibilità di implementare l'algoritmo in ambienti di calcolo avanzato, quali quelli che fanno uso di GPU (Graphics Processing Unit) e dell'architettura CUDA[3]. Il primo capitolo ha inizio richiamando brevemente le leggi di Maxwell, sulle quali si basano le cosiddette equazioni di aggiornamento delle componenti del campo elettrico e magnetico trattate nel secondo paragrafo, per poi continuare con cenni inerenti al problema della stabilità numerica e la condizione di Courant. E' stato riservato un paragrafo, ovvero il quarto, all'implementazione dello strato CPML[4] necessario per la riduzione

[1]Finite Difference Time Domain.

[2]Mezzo anisotropo è quello le cui proprietà dipendono dalla direzione.

[3]Acronimo di Compute Unified Device Architecture ovvero è un'architettura hardware per l'elaborazione parallela creata da NVIDIA.

[4]Convolutional Perfectly Matching Layer.

delle indesiderate riflessioni numeriche che hanno origine nelle superfici di contorno della regione d'indagine. Nel quinto paragrafo, invece, particolare attenzione è rivolta ai principali componenti concentrati (generatori di tensione, induttori, condensatori e diodi) nell'ottica di poterli facilmente inserire all'interno dei sistemi elettromagnetici che si potrebbero simulare. Infine, l'ultimo paragrafo del primo capitolo è dedicato alla forma d'onda impulso gaussiano, data la sua importanza nell'ambito dell'analisi in frequenza. Il secondo capitolo è rivolto alla presentazione delle analisi e risultati delle tecniche utilizzate con l'algoritmo FDTD implementato sia con CPU che GPU per notarne la differenza in termini di tempo, vedendo in particolare l'architettura CUDA. Infine nel terzo capitolo sono realizzate delle simulazioni con il software Matlab per analizzare le differenze implementative fra CPU e GPU utilizzando l'algoritmo FDTD con CPML nel caso bidimensionale.

1 Il metodo FDTD

In questo capitolo viene esposto come funziona l'algoritmo FDTD (Finite Difference Time Domain), ovvero l'algoritmo introdotto da Kane Yee [1] nel 1966 per la risoluzione numerica delle equazioni di Maxwell per i campi elettromagnetici. Quanto riportato di seguito è utile per richiamare soltanto le equazioni fondamentali e i concetti basilari che serviranno per comprendere i capitoli successivi. Tale lavoro è stato ricavato grazie all'ausilio delle fonti presenti nella bibliografia.

1.1 Le equazioni di Maxwell

L'algoritmo FDTD consiste nel considerare le equazioni di Maxwell [2] che regolano tutti i fenomeni elettromagnetici macroscopici e che sono formulate localmente nel seguente modo:

$$\nabla \times \overrightarrow{H} = \frac{\partial \overrightarrow{D}}{\partial t} + \overrightarrow{J_c} \tag{1}$$

$$\nabla \times \overrightarrow{E} = -\frac{\partial \overrightarrow{B}}{\partial t} \tag{2}$$

$$\nabla \cdot \overrightarrow{D} = \rho_e \tag{3}$$

$$\nabla \cdot \overrightarrow{B} = 0 \tag{4}$$

dove $\overrightarrow{E}$ il vettore campo elettrico, $\overrightarrow{D}$ il vettore spostamento elettrico, $\overrightarrow{H}$ il vettore campo magnetico, $\overrightarrow{B}$ il vettore induzione magnetica, $\overrightarrow{J_c}$ il vettore densità di corrente elettrica di conduzione e ρ_e la densità volu-

metrica di carica libera. Risulta necessario introdurre anche le relazioni costitutive al fine di caratterizzare i mezzi materiali sede dei fenomeni di interesse. Nell'ipotesi di considerare mezzi lineari, isotropi, omogenei e tempo-invarianti, dette relazioni assumono la seguente forma:

$$\overrightarrow{D} = \epsilon \overrightarrow{E} \tag{5}$$

$$\overrightarrow{B} = \mu \overrightarrow{H} \tag{6}$$

$$\overrightarrow{J_c} = \sigma \overrightarrow{E} \tag{7}$$

dove ϵ, μ e σ, rispettivamente, sono la costante dielettrica, la permeabilità magnetica e la conducibilità elettrica del mezzo materiale considerato. Le relazioni costitutive (5)-(7), permettono di esprimere le equazioni (1)-(2) nel seguente modo:

$$\nabla \times \overrightarrow{H} = \epsilon \frac{\partial \overrightarrow{E}}{\partial t} + \sigma \overrightarrow{E} \tag{8}$$

$$\nabla \times \overrightarrow{E} = -\mu \frac{\partial \overrightarrow{H}}{\partial t} \tag{9}$$

Le equazioni (8)-(9), essendo di natura vettoriale, sono equivalenti alle seguenti sei equazioni scalari espresse in un sistema di coordinate cartesiano (x, y, z):

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y}\right) \tag{10}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z}\right) \tag{11}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu}(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x}) \tag{12}$$

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon}(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma E_x) \tag{13}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon}(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma E_y) \tag{14}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon}(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z) \tag{15}$$

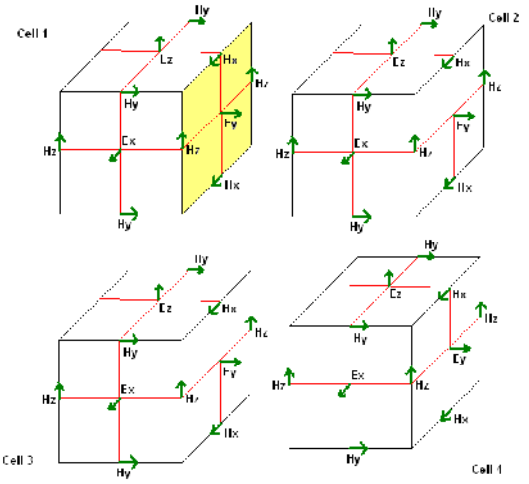


Figura 1: disposizione delle componenti discrete del campo elettromagnetico in 4 celle di Yee.

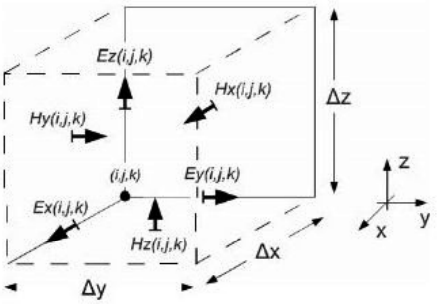


Figura 2: disposizione delle componenti discrete del campo elettromagnetico nella cella di Yee individuata dal nodo avente coordinate (iΔx, jΔy, kΔz)$\equiv$(i,j,k).

1.2 Le equazioni di aggiornamento

Nel 1966, Yee ideò un insieme di equazioni alle differenze finite tali da permettere l'implementazione diretta delle equazioni di Maxwell ai rotori nelle applicazioni numeriche al calcolatore. A tale proposito le equazioni (10)-(15) possono essere discretizzate, sia nello spazio che nel tempo, mediante l'utilizzo delle differenze finite centrali avendo preliminarmente sostituito la regione di indagine con una griglia regolare di nodi, la cui cella elementare, detta cella di Yee, è mostrata in figura 2, la relativa distribuzione spaziale delle componenti discrete del campo elettromagnetico. Data la regolarità della griglia introdotta, un generico nodo di quest'ultima, avente coordinate (iΔx, jΔy, kΔz), sarà individuato specificando solamente gli indici (i,j,k) lasciando sottintesi gli incrementi spaziali Δx, Δy, Δz. Manipolando opportunamente le equazioni di Maxwell discretizzate [3] ed indicando con il generi-

co apice n l'istante nΔt, essendo Δt l'incremento adottato per la discretizzazione nel tempo delle grandezze di campo, è possibile risalire alle seguenti equazioni di aggiornamento delle componenti del campo elettromagnetico:

$$\begin{aligned} H_x|_{i,j,k}^{n+\frac{1}{2}} &= H_x|_{i,j,k}^{n-\frac{1}{2}} + c_{hxey}|_{i,j,k} \cdot (E_y|_{i,j,k+1}^{n} - E_y|_{i,j,k}^{n}) \\ &\quad + c_{hxez}|_{i,j,k} \cdot (E_z|_{i,j+1,k}^{n} - E_y|_{i,j,k}^{n}) \end{aligned} \tag{16}$$

$$\begin{aligned} H_y|_{i,j,k}^{n+\frac{1}{2}} &= H_y|_{i,j,k}^{n-\frac{1}{2}} + c_{hyez}|_{i,j,k} \cdot (E_z|_{i+1,j,k}^{n} - E_z|_{i,j,k}^{n}) \\ &\quad + c_{hyex}|_{i,j,k} \cdot (E_x|_{i+1,j,k}^{n} - E_z|_{i,j,k}^{n}) \end{aligned} \tag{17}$$

$$\begin{aligned} H_z|_{i,j,k}^{n+\frac{1}{2}} &= H_z|_{i,j,k}^{n-\frac{1}{2}} + c_{hzex}|_{i,j,k} \cdot (E_x|_{i,j+1,k}^{n} - E_x|_{i,j,k}^{n}) \\ &\quad + c_{hzey}|_{i,j,k} \cdot (E_y|_{i+1,j,k}^{n} - E_y|_{i,j,k}^{n}) \end{aligned} \tag{18}$$

$$\begin{aligned} E_x|_{i,j,k}^{n+1} &= c_{exe}|_{i,j,k} \cdot E_x|_{i,j,k}^{n} + c_{exhz}|_{i,j,k} \cdot (H_z|_{i,j,k}^{n+\frac{1}{2}} - H_z|_{i,j-1,k}^{n+\frac{1}{2}}) \\ &\quad + c_{exhy}|_{i,j,k} \cdot (H_y|_{i,j,k}^{n+\frac{1}{2}} - H_y|_{i,j,k-1}^{n+\frac{1}{2}}) \end{aligned} \tag{19}$$

$$\begin{aligned} E_y|_{i,j,k}^{n+1} &= c_{eye}|_{i,j,k} \cdot E_y|_{i,j,k}^{n} + c_{eyhx}|_{i,j,k} \cdot (H_x|_{i,j,k}^{n+\frac{1}{2}} - H_x|_{i,j,k-1}^{n+\frac{1}{2}}) \\ &\quad + c_{eyhz}|_{i,j,k} \cdot (H_z|_{i,j,k}^{n+\frac{1}{2}} - H_z|_{i-1,j,k}^{n+\frac{1}{2}}) \end{aligned} \tag{20}$$

$$E_z|_{i,j,k}^{n+1} = c_{eze}|_{i,j,k} \cdot E_z|_{i,j,k}^{n} + c_{ezhy}|_{i,j,k} \cdot (H_y|_{i,j,k}^{n+\frac{1}{2}} - H_y|_{i-1,j,k}^{n+\frac{1}{2}}) + c_{ezhx}|_{i,j,k} \cdot (H_x|_{i,j,k}^{n+\frac{1}{2}} - H_x|_{i,j-1,k}^{n+\frac{1}{2}}) \quad (21)$$

Nelle equazioni di sopra, i coefficienti di aggiornamento delle componenti del campo magnetico sono:

$$c_{hxey}|_{i,j,k} = \frac{\Delta t}{\mu_{x_{i,j,k}} \Delta z} \quad (22)$$

$$c_{hxez}|_{i,j,k} = -\frac{\Delta t}{\mu_{x_{i,j,k}} \Delta y} \quad (23)$$

$$c_{hyez}|_{i,j,k} = \frac{\Delta t}{\mu_{y_{i,j,k}} \Delta x} \quad (24)$$

$$c_{hyex}|_{i,j,k} = -\frac{\Delta t}{\mu_{y_{i,j,k}} \Delta z} \quad (25)$$

$$c_{hzex}|_{i,j,k} = \frac{\Delta t}{\mu_{z_{i,j,k}} \Delta y} \quad (26)$$

$$c_{hzey}|_{i,j,k} = -\frac{\Delta t}{\mu_{z_{i,j,k}} \Delta x} \quad (27)$$

mentre quelli relativi alle componenti del campo elettrico sono:

$$c_{exe}|_{i,j,k} = \frac{2\epsilon_{x_{i,j,k}} - \Delta t \sigma_{x_{i,j,k}}}{2\epsilon_{x_{i,j,k}} + \Delta t \sigma_{x_{i,j,k}}} \quad (28)$$

$$c_{exhz}|_{i,j,k} = \frac{2\Delta t}{(2\epsilon_{x_{i,j,k}} + \Delta t \sigma_{x_{i,j,k}}) \Delta y} \quad (29)$$

$$c_{exhy}|_{i,j,k} = -\frac{2\Delta t}{(2\epsilon_{x_{i,j,k}} + \Delta t \sigma_{x_{i,j,k}}) \Delta z} \quad (30)$$

$$c_{eye}|_{i,j,k} = \frac{2\epsilon_{y_{i,j,k}} - \Delta t \sigma_{y_{i,j,k}}}{2\epsilon_{y_{i,j,k}} + \Delta t \sigma_{y_{i,j,k}}} \quad (31)$$

$$c_{eyhz}|_{i,j,k} = \frac{2\Delta t}{(2\epsilon_{y_{i,j,k}} + \Delta t\sigma_{y_{i,j,k}})\Delta z} \tag{32}$$

$$c_{eyhz}|_{i,j,k} = -\frac{2\Delta t}{(2\epsilon_{y_{i,j,k}} + \Delta t\sigma_{y_{i,j,k}})\Delta x} \tag{33}$$

$$c_{eze}|_{i,j,k} = \frac{2\epsilon_{z_{i,j,k}} - \Delta t\sigma_{z_{i,j,k}}}{2\epsilon_{z_{i,j,k}} + \Delta t\sigma_{z_{i,j,k}}} \tag{34}$$

$$c_{ezhy}|_{i,j,k} = \frac{2\Delta t}{(2\epsilon_{z_{i,j,k}} + \Delta t\sigma_{z_{i,j,k}})\Delta x} \tag{35}$$

$$c_{ezhx}|_{i,j,k} = -\frac{2\Delta t}{(2\epsilon_{z_{i,j,k}} + \Delta t\sigma_{z_{i,j,k}})\Delta y} \tag{36}$$

Nei suddetti coefficienti di aggiornamento compaiono le costanti caratteristiche del mezzo relative alle varie componenti del campo elettromagnetico a cui si riferiscono; la costante dielettrica $\epsilon_{x_{i,j,k}}$, per esempio, si riferisce alla componente E_x del campo elettrico appartenente alla cella di Yee individuata dagli indici (i, j, k). Le relazioni (16)-(21) mettono in evidenza che le componenti del campo magnetico ed elettrico non sono valutate nello stesso istante. In particolare, i valori delle componenti del campo magnetico sono calcolati in base a quelli delle componenti del campo elettrico valutati nell'istante temporale immediatamente precedente e viceversa.

1.3 Stabilità numerica

In generale, l'algoritmo si dice stabile se un errore introdotto in un passo di calcolo, per esempio dovuto al troncamento di un dato numerico, si mantiene limitato al tendere ad infinito del numero di iterazioni. La stabilità numerica dell'algoritmo FDTD è strettamente legata alla condizione di Courant [4], la quale impone che l'incremento temporale Δt adottato si mantenga entro un limite definito in funzione degli incrementi spaziali Δx , Δy e Δz della griglia di nodi:

$$\Delta t \leq \frac{1}{c\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}} \tag{37}$$

dove c la velocità della luce nello spazio libero. La condizione di Courant risulta perfettamente valida anche in presenza di mezzi diversi dal vuoto, infatti per quest'ultimi la velocità di propagazione delle onde elettromagnetiche risulta essere inferiore a c. È utile notare che nell'ipotesi di adottare una griglia cubica, per la quale $\Delta x = \Delta y = \Delta z$ la condizione (37) si riduce alla seguente disuguaglianza:

$$\Delta t \leq \frac{\Delta x}{c\sqrt{3}} \tag{38}$$

1.4 Lo strato CPML

A causa del quantitativo finito di memoria disponibile per l'esecuzione di un dato algoritmo FDTD, lo spazio d'indagine, sede dei fenomeni elettromagnetici che si vogliono simulare, dovrà essere opportunamente interrotto mediante particolari condizioni al contorno. In molte applicazioni, come nel caso dei problemi di radiazione e scattering, sono richieste delle condizioni al contorno in grado di simulare la propagazione delle onde elettromagnetiche verso l'esterno della regione d'indagine senza che vengano create delle riflessioni indesiderate numeriche tali da corrompere i risultati ottenuti. Lo strato CPML[5], introdotto per la prima volta da J. Roden e S. Gedney nel 2000 [5], è un particolare strato che circonda lo spazio d'indagine ed occupato da un mezzo fittizio in grado di non dare luogo a fenomeni di riflessione indipendentemente dall'angolo di incidenza e dalla frequenza delle onde elettromagnetiche. La propagazione del campo elettromagnetico all'interno dello strato CPML è retta dalle seguenti equazioni di Maxwell modificate ed espresse nel dominio della frequenza [6]:

$$j\omega\epsilon_x E_x(\omega) + \sigma_x E_x(\omega) = S_{ey}^{-1}\frac{\partial H_z(\omega)}{\partial y} - S_{ez}^{-1}\frac{\partial H_y(\omega)}{\partial z} \tag{39}$$

$$j\omega\epsilon_y E_y(\omega) + \sigma_y E_y(\omega) = S_{ez}^{-1}\frac{\partial H_x(\omega)}{\partial z} - S_{ex}^{-1}\frac{\partial H_z(\omega)}{\partial x} \tag{40}$$

$$j\omega\epsilon_z E_z(\omega) + \sigma_z E_z(\omega) = S_{ex}^{-1}\frac{\partial H_y(\omega)}{\partial x} - S_{ey}^{-1}\frac{\partial H_x(\omega)}{\partial y} \tag{41}$$

[5]Convolutional Perfectly Matching Layer.

$$j\omega\mu_x H_x(\omega) = -S_{my}^{-1}\frac{\partial E_z(\omega)}{\partial y} + S_{mz}^{-1}\frac{\partial E_y(\omega)}{\partial z} \tag{42}$$

$$j\omega\mu_y H_y(\omega) = -S_{mz}^{-1}\frac{\partial E_x(\omega)}{\partial z} + S_{mx}^{-1}\frac{\partial E_z(\omega)}{\partial x} \tag{43}$$

$$j\omega\mu_z H_z(\omega) = -S_{mx}^{-1}\frac{\partial E_y(\omega)}{\partial x} + S_{my}^{-1}\frac{\partial E_x(\omega)}{\partial y} \tag{44}$$

dove:

$$S_{e(ii)} = k_{e(ii)} + \frac{\sigma_{pe(ii)}}{\alpha_{e(ii)+j\omega\epsilon_0}},\ S_{m(ii)} = k_{m(ii)} + \frac{\sigma_{pm(ii)}}{\alpha_{m(ii)+j\omega\mu_0}},\ (ii) \equiv x, y, z \tag{45}$$

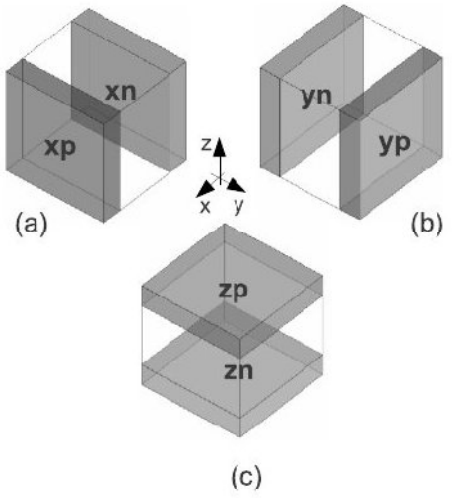


Figura 3: le regioni evidenziate in grigio sono quelle appartenenti allo strato CPML dove i parametri presenti nella relazione (45) assumono valori non nulli, in particolare l'illustrazione (a) si riferisce ai parametri con (ii) = x, quella (b) ai parametri con (ii) = y e quella (c) ai parametri con (ii) = z.

con $\sigma_{pe(ii)}$, $\sigma_{pm(ii)}$, $k_{e(ii)} \geqslant 1$, $\alpha_{e(ii)}$, $\alpha_{m(ii)}$ e $k_{e(ii)} \geqslant 0$ i quali sono dei parametri caratteristici dello specifico strato CPML preso in considerazione che soddisfano le seguenti condizioni [5, 6]:

$$k_{e(ii)} = k_{m(ii)} \tag{46}$$

$$\frac{\sigma_{pe(ii)}}{\epsilon_0} = \frac{\sigma_{pm(ii)}}{\mu_0}, \ \frac{\alpha_{e(ii)}}{\epsilon_0} = \frac{\alpha_{m(ii)}}{\mu_0} \tag{47}$$

avendo indicato con $\epsilon_0 \approx 8.854 \cdot 10^{-12} F/m$ e $\mu_0 = 4\pi \cdot 10^{-7} H/m$, rispettivamente la costante dielettrica e la permeabilità magnetica nel vuoto. Nella relazione (45) sono presenti valori diversi da zero solo in particolari zone dello strato CPML, ovvero quelle mostrate nella figura 3. Dopo aver riportato le equazioni (39)-(44) nel dominio del tempo, si può procedere alla fase di discretizzazione di queste, sia nello spazio che nel tempo, mediante l'utilizzo delle differenze finite centrali al posto delle derivate parziali, per poi risalire, in seguito a varie manipolazioni matematiche, alle equazioni di aggiornamento delle componenti discrete del campo elettromagnetico all'interno dello strato CPML [6]. L'analisi di dette equazioni porta all'ideazione di un metodo efficiente per la loro implementazione al calcolatore [5, 6]. Inizialmente si definiscono i nuovi coefficienti di aggiornamento:

$$c_{exhz}|_{i,j,k}^{new} = \frac{c_{exhz}|_{i,j,k}}{k_{ey}|_{i,j,k}}, \ c_{exhy}|_{i,j,k}^{new} = \frac{c_{exhy}|_{i,j,k}}{k_{ez}|_{i,j,k}} \tag{48}$$

$$c_{eyhx}|_{i,j,k}^{new} = \frac{c_{eyhx}|_{i,j,k}}{k_{ez}|_{i,j,k}}, \ c_{eyhz}|_{i,j,k}^{new} = \frac{c_{eyhz}|_{i,j,k}}{k_{ex}|_{i,j,k}} \tag{49}$$

$$c_{ezhy}|_{i,j,k}^{new} = \frac{c_{ezhy}|_{i,j,k}}{k_{ex}|_{i,j,k}},\ c_{ezhx}|_{i,j,k}^{new} = \frac{c_{ezhx}|_{i,j,k}}{k_{ey}|_{i,j,k}} \tag{50}$$

$$c_{hxez}|_{i,j,k}^{new} = \frac{c_{hxez}|_{i,j,k}}{k_{my}|_{i,j,k}},\ c_{hxey}|_{i,j,k}^{new} = \frac{c_{hxey}|_{i,j,k}}{k_{mz}|_{i,j,k}} \tag{51}$$

$$c_{hyex}|_{i,j,k}^{new} = \frac{c_{hyex}|_{i,j,k}}{k_{mz}|_{i,j,k}},\ c_{hyez}|_{i,j,k}^{new} = \frac{c_{hyez}|_{i,j,k}}{k_{mx}|_{i,j,k}} \tag{52}$$

$$c_{hzey}|_{i,j,k}^{new} = \frac{c_{hzey}|_{i,j,k}}{k_{mx}|_{i,j,k}},\ c_{hzex}|_{i,j,k}^{new} = \frac{c_{hzex}|_{i,j,k}}{k_{my}|_{i,j,k}} \tag{53}$$

necessari per la valutazione, tramite le equazioni di aggiornamento presenti nel paragrafo 1.2, delle componenti del campo elettromagnetico presenti all'interno dello strato CPML per poi procedere con la correzione dei risultati ottenuti mediante l'esecuzione delle seguenti istruzioni di assegnazione:

$$E_x|_{i,j,k}^{n+1} = E_x|_{i,j,k}^{n+1} + c_{\psi exy}|_{i,j,k} \cdot \psi_{exy}|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi exz}|_{i,j,k} \cdot \psi_{exz}|_{i,j,k}^{n+\frac{1}{2}} \tag{54}$$

$$E_y|_{i,j,k}^{n+1} = E_y|_{i,j,k}^{n+1} + c_{\psi eyz}|_{i,j,k} \cdot \psi_{eyz}|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi eyx}|_{i,j,k} \cdot \psi_{eyx}|_{i,j,k}^{n+\frac{1}{2}} \tag{55}$$

$$E_z|_{i,j,k}^{n+1} = E_z|_{i,j,k}^{n+1} + c_{\psi ezx}|_{i,j,k} \cdot \psi_{ezx}|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi ezy}|_{i,j,k} \cdot \psi_{ezy}|_{i,j,k}^{n+\frac{1}{2}} \tag{56}$$

$$H_x|_{i,j,k}^{n+\frac{1}{2}} = H_x|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi hxy}|_{i,j,k} \cdot \psi_{hxy}|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi hxz}|_{i,j,k} \cdot \psi_{hxz}|_{i,j,k}^{n} \tag{57}$$

$$H_y|_{i,j,k}^{n+\frac{1}{2}} = H_y|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi hyz}|_{i,j,k} \cdot \psi_{hyz}|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi hyx}|_{i,j,k} \cdot \psi_{hyx}|_{i,j,k}^{n} \tag{58}$$

$$H_z|_{i,j,k}^{n+\frac{1}{2}} = H_z|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi hzx}|_{i,j,k} \cdot \psi_{hzx}|_{i,j,k}^{n+\frac{1}{2}} + c_{\psi hzy}|_{i,j,k} \cdot \psi_{hzy}|_{i,j,k}^{n} \tag{59}$$

dove:

$$c_{\psi exy}|_{i,j,k} = \Delta y c_{exhz}|_{i,j,k},\ c_{\psi exz}|_{i,j,k} = \Delta z c_{exhy}|_{i,j,k} \tag{60}$$

$$c_{\psi eyz}|_{i,j,k} = \Delta z c_{eyhx}|_{i,j,k},\ c_{\psi eyx}|_{i,j,k} = \Delta x c_{eyhz}|_{i,j,k} \tag{61}$$

$$c_{\psi ezx}|_{i,j,k} = \Delta x c_{ezhy}|_{i,j,k},\ c_{\psi ezy}|_{i,j,k} = \Delta y c_{ezhx}|_{i,j,k} \tag{62}$$

$$c_{\psi hxy}|_{i,j,k} = \Delta y c_{hxez}|_{i,j,k},\ c_{\psi hxz}|_{i,j,k} = \Delta z c_{hxey}|_{i,j,k} \tag{63}$$

$$c_{\psi hyz}|_{i,j,k} = \Delta z c_{hyex}|_{i,j,k},\ c_{\psi hyx}|_{i,j,k} = \Delta x c_{hyez}|_{i,j,k} \tag{64}$$

$$c_{\psi hzx}|_{i,j,k} = \Delta x c_{hzey}|_{i,j,k},\ c_{\psi hzy}|_{i,j,k} = \Delta y c_{hzex}|_{i,j,k} \tag{65}$$

essendo:

$$\psi_{exy}|_{i,j,k}^{n+\frac{1}{2}} = b_{ey}|_{i,j,k} \cdot \psi_{exy}|_{i,j,k}^{n-\frac{1}{2}} + a_{ey}|_{i,j,k} \cdot (H_z|_{i,j,k}^{n+\frac{1}{2}} - H_z|_{i,j-1,k}^{n+\frac{1}{2}}) \tag{66}$$

$$\psi_{exz}|_{i,j,k}^{n+\frac{1}{2}} = b_{ez}|_{i,j,k} \cdot \psi_{exz}|_{i,j,k}^{n-\frac{1}{2}} + a_{ez}|_{i,j,k} \cdot (H_y|_{i,j,k}^{n+\frac{1}{2}} - H_y|_{i,j,k-1}^{n+\frac{1}{2}}) \tag{67}$$

$$\psi_{eyz}|_{i,j,k}^{n+\frac{1}{2}} = b_{ez}|_{i,j,k} \cdot \psi_{eyz}|_{i,j,k}^{n-\frac{1}{2}} + a_{ez}|_{i,j,k} \cdot (H_x|_{i,j,k}^{n+\frac{1}{2}} - H_x|_{i,j,k-1}^{n+\frac{1}{2}}) \tag{68}$$

$$\psi_{eyx}|_{i,j,k}^{n+\frac{1}{2}} = b_{ex}|_{i,j,k} \cdot \psi_{eyx}|_{i,j,k}^{n-\frac{1}{2}} + a_{ex}|_{i,j,k} \cdot (H_z|_{i,j,k}^{n+\frac{1}{2}} - H_z|_{i-1,j,k}^{n+\frac{1}{2}}) \tag{69}$$

$$\psi_{ezx}|_{i,j,k}^{n+\frac{1}{2}} = b_{ex}|_{i,j,k} \cdot \psi_{ezx}|_{i,j,k}^{n-\frac{1}{2}} + a_{ex}|_{i,j,k} \cdot (H_y|_{i,j,k}^{n+\frac{1}{2}} - H_y|_{i-1,j,k}^{n+\frac{1}{2}}) \tag{70}$$

$$\psi_{ezy}|_{i,j,k}^{n+\frac{1}{2}} = b_{ey}|_{i,j,k} \cdot \psi_{ezy}|_{i,j,k}^{n-\frac{1}{2}} + a_{ey}|_{i,j,k} \cdot (H_x|_{i,j,k}^{n+\frac{1}{2}} - H_x|_{i,j-1,k}^{n+\frac{1}{2}}) \tag{71}$$

$$\psi_{hxy}|_{i,j,k}^{n} = b_{my}|_{i,j,k} \cdot \psi_{exy}|_{i,j,k}^{n-1} + a_{my}|_{i,j,k} \cdot (E_z|_{i,j+1,k}^{n} - E_z|_{i,j,k}^{n}) \tag{72}$$

$$\psi_{hxz}|_{i,j,k}^{n} = b_{mz}|_{i,j,k} \cdot \psi_{hxz}|_{i,j,k}^{n-1} + a_{mz}|_{i,j,k} \cdot (E_y|_{i,j,k+1}^{n} - E_y|_{i,j,k}^{n}) \tag{73}$$

$$\psi_{hyz}|_{i,j,k}^{n} = b_{mz}|_{i,j,k} \cdot \psi_{hyz}|_{i,j,k}^{n-1} + a_{mz}|_{i,j,k} \cdot (E_x|_{i,j,k+1}^{n} - E_x|_{i,j,k}^{n}) \tag{74}$$

$$\psi_{hyx}|_{i,j,k}^{n} = b_{mx}|_{i,j,k} \cdot \psi_{hyx}|_{i,j,k}^{n-1} + a_{mx}|_{i,j,k} \cdot (E_z|_{i+1,j,k}^{n} - E_z|_{i,j,k}^{n}) \tag{75}$$

$$\psi_{hzx}|_{i,j,k}^{n} = b_{mx}|_{i,j,k} \cdot \psi_{hzx}|_{i,j,k}^{n-1} + a_{mx}|_{i,j,k} \cdot (E_y|_{i+1,j,k}^{n} - E_y|_{i,j,k}^{n}) \tag{76}$$

$$\psi_{hxy}|^n_{i,j,k} = b_{my}|_{i,j,k} \cdot \psi_{hzy}|^{n-1}_{i,j,k} + a_{my}|_{i,j,k} \cdot (E_x|^n_{i,j+1,k} - E_x|^n_{i,j,k}) \quad (77)$$

dove:

$$a_{e(ii)}|_{i,j,k} = \frac{\sigma_{pe(ii)}|_{i,j,k}}{\Delta(ii)(\sigma_{pe(ii)}|_{i,j,k} \cdot k_{e(ii)}|_{i,j,k} + \alpha_{e(ii)}|_{i,j,k} \cdot k^2_{e(ii)}|_{i,j,k})} \quad (78)$$

$$a_{m(ii)}|_{i,j,k} = \frac{\sigma_{pm(ii)}|_{i,j,k}}{\Delta(ii)(\sigma_{pm(ii)}|_{i,j,k} \cdot k_{m(ii)}|_{i,j,k} + \alpha_{m(ii)}|_{i,j,k} \cdot k^2_{m(ii)}|_{i,j,k})} \quad (79)$$

$$b_{e(ii)}|_{i,j,k} = exp(-\frac{\Delta t}{\epsilon_0} \cdot (\frac{\sigma_{pe(ii)}|_{i,j,k}}{k_{e(ii)}|_{i,j,k}} + \alpha_{e(ii)}|_{i,j,k})) \quad (80)$$

$$b_{m(ii)}|_{i,j,k} = exp(-\frac{\Delta t}{\mu_0} \cdot (\frac{\sigma_{pm(ii)}|_{i,j,k}}{k_{m(ii)}|_{i,j,k}} + \alpha_{m(ii)}|_{i,j,k})) \quad (81)$$

dove il simbolo (ii) corrisponde, in base alla direzione considerata, a x, y oppure z. Infine, a causa della discretizzazione effettuata, il comportamento dello strato CPML si discosta sensibilmente da quello teorico descritto dalle relazioni (39)-(44) [7]. L'attenuazione di tale effetto indesiderato si può ottenere adottando una distribuzione spaziale dei parametri caratteristici non uniforme, secondo leggi del tipo [6]:

$$\sigma_{pe(ii)}(\rho) = \sigma_{max}(\frac{\rho}{\delta})^{n_{pml}} \quad (82)$$

$$\sigma_{pm(ii)}(\rho) = \frac{\mu_0}{\epsilon_0}\sigma_{max}(\frac{\rho}{\delta})^{n_{pml}} \quad (83)$$

$$k_{e(ii)}(\rho) = 1 + (k_{max} - 1)(\frac{\rho}{\delta})^{n_{pml}} \quad (84)$$

$$k_{m(ii)}(\rho) = 1 + (k_{max} - 1)(\frac{\rho}{\delta})^{n_{pml}} \quad (85)$$

$$\alpha_{e(ii)}(\rho) = \alpha_{min} + (\alpha_{max} - \alpha_{min})(1 - \frac{\rho}{\delta}) \quad (86)$$

$$\alpha_{e(ii)}(\rho) = \frac{\mu_0}{\epsilon_0}(\alpha_{min} + (\alpha_{max} - \alpha_{min})(1 - \frac{\rho}{\delta})) \tag{87}$$

dove:

- ρ la distanza tra la superficie interna dello strato CPML e il punto all'interno di quest'ultimo dove si vuole calcolare il valore del parametro considerato;
- δ lo spessore dello strato CPML;
- n_{pml} una quantità generalmente compresa tra 2 e 4;
- i termini, aventi come pedici le diciture max e min, delle costanti valutate opportunamente in modo da ottenere dallo strato CPML le prestazioni desiderate [6].

1.5 Modellizzazione dei componenti concentrati

In molte applicazioni si richiede l'utilizzo di componenti concentrati quali, per esempio, generatori di tensione, resistori, condensatori, induttori e diodi. Di seguito verrà data una breve traccia della metodologia adottata per la modellizzazione di tali componenti per un'eventuale simulazione con algoritmo FDTD. Per una corretta modellizzazione del generatore di tensione reale mostrato in figura 4, caratterizzato da una tensione a vuoto V_s e da una resistenza interna R_s, è opportuno inserire preliminarmente nella relazione (15) il termine J_{iz}, ovvero la componente secondo l'asse z della densità di corrente impressa:

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon}(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z - J_{iz}) \tag{88}$$

Discretizzando l'equazione (88), sia nello spazio che nel tempo, mediante la sostituzione delle derivate parziali con le differenze finite centrali, esprimendo la differenza di potenziale ΔV e la corrente I in funzione delle grandezze di campo [6]:

$$\Delta V^{n+\frac{1}{2}} = \frac{E_z|_{i,j,k}^{n+1} + E_z|_{i,j,k}^{n}}{2}\Delta z \tag{89}$$

$$I^{n+\frac{1}{2}} = J_{iz}|_{i,j,k}^{n+\frac{1}{2}}\Delta x \Delta y \tag{90}$$

e tenendo conto della legge di Ohm:

$$I^{n+\frac{1}{2}} = \frac{\Delta V^{n+\frac{1}{2}} + V_s^{n+\frac{1}{2}}}{R_s} \tag{91}$$

si ottiene, in seguito a varie manipolazioni matematiche [6], la seguente equazione di aggiornamento della componente del campo elettrico lungo l'asse z relativa al nodo (i,j,k):

$$\begin{aligned} E_z|_{i,j,k}^{n+1} &= c_{eze}|_{i,j,k} \cdot E_z|_{i,j,k}^{n} + c_{ezhy}|_{i,j,k} \cdot (H_y|_{i,j,k}^{n+\frac{1}{2}} - H_y|_{i-1,j,k}^{n+\frac{1}{2}}) \\ &+ c_{ezhx}|_{i,j,k} \cdot (H_x|_{i,j,k}^{n+\frac{1}{2}} - H_x|_{i,j-1,k}^{n+\frac{1}{2}}) + c_{ezs}|_{i,j,k} \cdot V_s^{n+\frac{1}{2}} \end{aligned} \quad (92)$$

essendo i coefficienti di aggiornamento così definiti:

$$c_{eze}|_{i,j,k} = \frac{2\epsilon_{z_{i,j,k}} - \Delta t \sigma_{z_{i,j,k}} - \frac{\Delta t \Delta z}{R_s \Delta x \Delta y}}{2\epsilon_{z_{i,j,k}} + \Delta t \sigma_{z_{i,j,k}} + \frac{\Delta t \Delta z}{R_s \Delta x \Delta y}} \quad (93)$$

$$c_{ezhy}|_{i,j,k} = \frac{2\Delta t}{(2\epsilon_{z_{i,j,k}} + \Delta t \sigma_{z_{i,j,k}} + \frac{\Delta t \Delta z}{R_s \Delta x \Delta y})\Delta x} \quad (94)$$

$$c_{ezhx}|_{i,j,k} = -\frac{2\Delta t}{(2\epsilon_{z_{i,j,k}} + \Delta t \sigma_{z_{i,j,k}} + \frac{\Delta t \Delta z}{R_s \Delta x \Delta y})\Delta y} \quad (95)$$

$$c_{ezs}|_{i,j,k} = -\frac{2\Delta t}{(2\epsilon_{zi,j,k} + \Delta t \sigma_{zi,j,k} + \frac{\Delta t \Delta z}{R_s \Delta x \Delta y})(R_s \Delta x \Delta y)} \quad (96)$$

Qualora si dovesse presentare la necessità di imporre una data differenza di potenziale tra due nodi della griglia, si deve ricorrere ad un generatore di tensione ideale che, a differenza di quello reale, è caratterizzato da una resistenza interna nulla. In base a quanto appena detto, ponendo $R_s = 0$ nelle relazioni (93)-(96), si può facilmente verificare che i coefficienti di aggiornamento, validi per un generatore di tensione ideale, risultano essere così esprimibili:

$$c_{eze}|_{i,j,k} = -1,\ c_{ezhy}|_{i,j,k} = 0,\ c_{ezhx}|_{i,j,k} = 0,\ c_{ezs}|_{i,j,k} = -\frac{2}{\Delta z} \quad (97)$$

Infine, alcune applicazioni potrebbero richiedere di utilizzare un generatore di tensione tale da occupare un certo numero di celle della griglia, pertanto è opportuno considerare detto sistema come costituito da tanti generatori di tensione elementari, variamente disposti in serie e in parallelo, ognuno dei quali collegato tra due nodi consecutivi della griglia. Facendo riferimento alla figura 4, la regione occupata dalle celle previste per il generatore di tensione è individuata dagli indici dei nodi di estremità (i_s, j_s, k_s) e (i_e, j_e, k_e), mentre la tensione a vuoto $V_s^{'}$ e la resistenza interna $R_s^{'}$ relative ad ogni singolo generatore elementare risultano essere così esprimibili [6]:

$$V_s^{'} = \frac{V_s}{(k_e - k_s)} \tag{98}$$

$$R_s^{'} = R_s \frac{(i_e - i_s + 1)(j_e - j_s + 1)}{(k_e - k_s)} \tag{99}$$

1.5.1 Resistore

Quanto finora detto per il generatore di tensione risulta perfettamente valido anche per il resistore di resistenza R, purchè che si ponga sistematicamente $V_s = 0V$ e $R_s = R$.

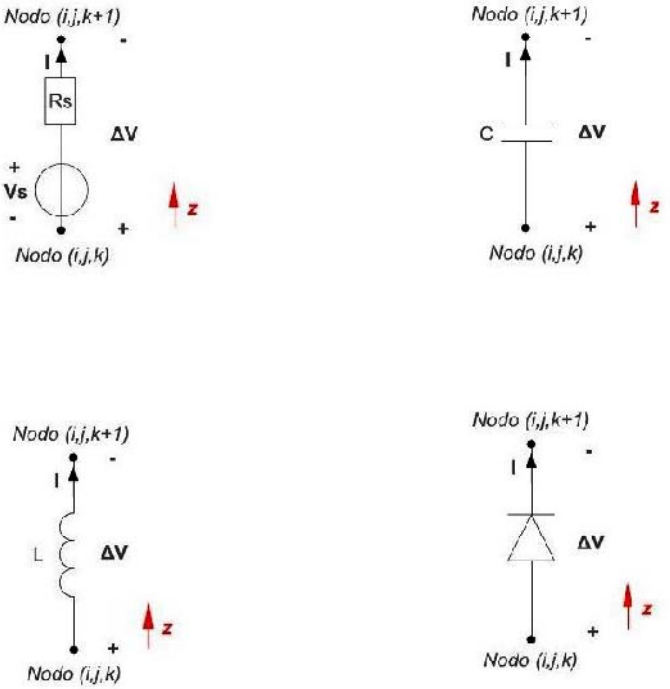


Figura 4: schema dei diversi modelli a parametri concentrati, ovvero generatore di tensione, condensatore, induttore e diodo, tra i nodi (i,j,k) e (i,j,k+1), disposti parallelamente all'asse z del sistema di coordinate cartesiano adottato.

1.5.2 Condensatore

La relazione che lega tensione ΔV e corrente I di un condensatore, avente capacità C e mostrato in figura 4, è la seguente:

$$I = C\frac{d\Delta V}{dt} \tag{100}$$

che espressa in forma discreta diviene:

$$I^{n+\frac{1}{2}} = C\frac{\Delta V^{n+1} - \Delta V^{n}}{\Delta t} \tag{101}$$

Sostituendo le relazioni (89)-(90) nella (101) si ottiene la formulazione esplicita di $J_{iz}|_{i,j,k}^{n+\frac{1}{2}}$ che, inserita nella formulazione discreta dell'equazione (88), permette di ricavare la seguente equazione di aggiornamento [6]:

$$\begin{aligned} E_z|_{i,j,k}^{n+1} &= c_{eze}|_{i,j,k} \cdot E_z|_{i,j,k}^{n} + c_{ezhy}|_{i,j,k} \cdot (H_y|_{i,j,k}^{n+\frac{1}{2}} - H_y|_{i-1,j,k}^{n+\frac{1}{2}}) \\ &\quad + c_{ezhx}|_{i,j,k} \cdot (H_x|_{i,j,k}^{n+\frac{1}{2}} - H_x|_{i,j-1,k}^{n+\frac{1}{2}}) \end{aligned} \tag{102}$$

essendo i coefficienti di aggiornamento così definiti:

$$c_{eze}|_{i,j,k} = \frac{2\epsilon_{z_{i,j,k}} - \Delta t\sigma_{z_{i,j,k}} + \frac{2C\Delta z}{\Delta x\Delta y}}{2\epsilon_{z_{i,j,k}} + \Delta t\sigma_{z_{i,j,k}} + \frac{2C\Delta z}{\Delta x\Delta y}} \tag{103}$$

$$c_{ezhy}|_{i,j,k} = \frac{2\Delta t}{(2\epsilon_{z_{i,j,k}} + \Delta t\sigma_{z_{i,j,k}} + \frac{2C\Delta z}{\Delta x\Delta y})\Delta x} \tag{104}$$

$$c_{ezhx}|_{i,j,k} = -\frac{2\Delta t}{(2\epsilon_{z_{i,j,k}} + \Delta t\sigma_{z_{i,j,k}} + \frac{2C\Delta z}{\Delta x\Delta y})\Delta y} \tag{105}$$

Qualora il condensatore si dovesse estendere su più celle della griglia, le considerazioni da effettuare sono analoghe a quelle fatte per il resistore a patto di considerare la capacità C' dei condensatori elementari pari a [6]:

$$C' = C\frac{(k_e - k_s)}{(i_e - i_s + 1)(j_e - j_s + 1)} \tag{106}$$

1.5.3 Induttore

La relazione che lega tensione ΔV e corrente I di un induttore, avente induttanza L e mostrato in figura 6, è la seguente:

$$\Delta V = L\frac{dI}{dt} \tag{107}$$

che espressa in forma discreta diviene:

$$\Delta V^n = L\frac{\Delta I^{n+\frac{1}{2}} - \Delta I^{n-\frac{1}{2}}}{\Delta t} \tag{108}$$

Sostituendo le relazioni (89)-(90) nella (101) si ottiene la formulazione esplicita di $J_{iz}|_{i,j,k}^{n+\frac{1}{2}}$ che, inserita nella formulazione discreta dell'equazione (88), permette di ricavare la seguente equazione di aggiornamento [6]:

$$\begin{aligned} E_z|_{i,j,k}^{n+1} &= c_{eze}|_{i,j,k} \cdot E_z|_{i,j,k}^{n} + c_{ezhy}|_{i,j,k} \cdot (H_y|_{i,j,k}^{n+\frac{1}{2}} - H_y|_{i-1,j,k}^{n+\frac{1}{2}}) \\ &+ c_{ezhx}|_{i,j,k} \cdot (H_x|_{i,j,k}^{n+\frac{1}{2}} - H_x|_{i,j-1,k}^{n+\frac{1}{2}}) + c_{ezj}|_{i,j,k} \cdot J_{iz}|_{i,j,k}^{n+\frac{1}{2}} \end{aligned} \tag{109}$$

dove i coefficienti di aggiornamento $c_{eze}|_{i,j,k}, c_{ezhy}|_{i,j,k}, c_{ezhx}|_{i,j,k}$ sono gli stessi definiti nel paragrafo 1.2, mentre $c_{ezj}|_{i,j,k}$ risulta essere così esprimibile:

$$c_{ezj}|_{i,j,k} = -\frac{2\Delta t}{(2\epsilon_{z_{i,j,k}} + \Delta t \sigma_{z_{i,j,k}})} \tag{110}$$

Qualora l'induttore si dovesse estendere su più celle della griglia, le considerazioni da effettuare sono analoghe a quelle fatte per il resistore a patto di considerare l'induttanza $L^{'}$ degli induttori elementari pari a [6]:

$$L^{'} = L\frac{(i_e - i_s + 1)(j_e - j_s + 1)}{(k_e - k_s)} \tag{111}$$

1.5.4 Diodo

Uno dei principali punti di forza dell'algoritmo FDTD consiste nella possibilità di poter modellizzare in maniera diretta e semplice anche sistemi non lineari. In questa trattazione si prenderà come esempio uno dei più comuni componenti elettronici non lineari, il diodo. Con riferimento ai versi convenzionali di figura 4, la relazione che lega la tensione ΔV e la corrente I in un diodo è:

$$I = I_0(exp(\frac{q\Delta V}{kT}) - 1) \tag{112}$$

essendo I_0 la corrente inversa, $k = 1.38 \cdot 10^{-23} J/^{\circ}K$ la costante di Boltzmann, $q = 1.6 \cdot 10^{-19} C$ la carica elementare positiva e T la temperatura assoluta in °K. Sostituendo nella (112) le relazioni (89)-(90), tenendo conto dell'equazione (88) discretizzata, è possibile ottenere la seguente

equazione [6]:

$$A\ exp(BE_z|_{i,j,k}^{n+1}) + E_z|_{i,j,k}^{n+1} + C = 0 \tag{113}$$

dove si è posto:

$$A = -\frac{2\Delta t I_0 exp(BE_z|_{i,j,k}^{n})}{2\epsilon_{z_{i,j,z}}\Delta x\Delta y + \Delta t\sigma_{z_{i,j,z}}\Delta x\Delta y} \tag{114}$$

$$B = -\frac{q\Delta z}{2kT} \tag{115}$$

$$\begin{aligned} C \quad &= c_{eze}|_{i,j,k} \cdot E_z|_{i,j,k}^{n} + c_{ezhy}|_{i,j,k} \cdot (H_y|_{i,j,k}^{n+\frac{1}{2}} - H_y|_{i-1,j,k}^{n+\frac{1}{2}}) \\ &+ c_{ezhx}|_{i,j,k} \cdot (H_x|_{i,j,k}^{n+\frac{1}{2}} - H_x|_{i,j-1,k}^{n+\frac{1}{2}}) \\ &+ \frac{2\Delta t I_0}{2\epsilon_{z_{i,j,z}}\Delta x\Delta y + \Delta t\sigma_{z_{i,j,z}}\Delta x\Delta y} \end{aligned} \tag{116}$$

essendo i coefficienti di aggiornamento che compaiono nella (116) gli stessi definiti nel paragrafo 1.2. Infine, data la natura non lineare dell'equazione (113), la relativa soluzione può essere ottenuta numericamente applicando il metodo di Newton Raphson.

1.6 Forma d'onda: l'impulso gaussiano

Al fine di verificare la robustezza del metodo, in genere si effettuano test numerici utilizzando come sorgente una forma d'onda gaussiana, la quale consente di spazzolare un ampio range di frequenze. E' utile, a questo punto, mettere in evidenza che, per garantire una sufficien-

te accuratezza dei risultati della simulazione, la massima frequenza d'indagine f_{max} deve essere scelta in modo tale che la corrispondente lunghezza d'onda λ_{min} sia almeno pari a circa venti volte la massima dimensione delle celle appartenenti alla griglia che occupa la regione, sede dei fenomeni elettromagnetici d'interesse[6,7]. La funzione analitica g(t) che definisce l'andamento temporale dell'impulso gaussiano è la seguente:

$$g(t) = exp(-\frac{1}{2}(\frac{t - t_0}{\tau})^2) \tag{117}$$

dove τ è il parametro da cui dipende la larghezza dell'impulso e t_0 l'istante a cui corrisponde il valore massimo di quest'ultimo. Nella figura 3 è mostrato l'andamento della funzione g(t) per vari valori di τ e per $t_0 = 0s$. Infine si può porre $t_0 = 4,5\tau$ in modo da evitare troncamenti della forma d'onda.

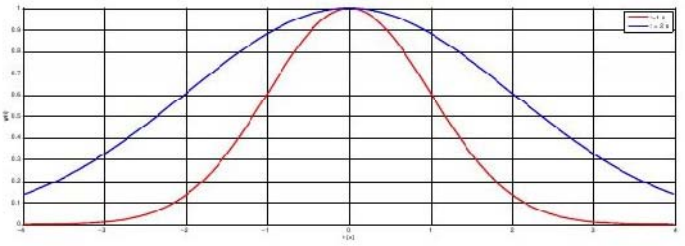


Figura 5: Andamento temporale della g(t) per $t_0 = 0s$ e per diversi valori del parametro τ in particolare la curva rossa per $\tau = 1s$ e la curva blu per $\tau = 2s$.

2 Accelerazione hardware dell'algoritmo FDTD con GPU

L'accelerazione hardware dell'algoritmo Finite-Difference Time-Domain (FDTD) è sempre stata una parte importante della ricerca FDTD. Con questa tesi si vuole far notare il vantaggio e la possibilità di accelerare l'algoritmo FDTD utilizzando le potenzialità di una o più GPU [21]. Confrontando l'eleborazione di calcolo della CPU con quello della GPU si verifica che quest'ultimo offre alte prestazioni a un costo molto ragionevole. Infatti all'inizio, i processori grafici erano semplici processori a funzione fissa, ma ben presto si sono evoluti in unità sempre più potenti dal punto di vista della capacità di calcolo e sempre più programmabili, situazione che ha poi portato per esempio l'azienda NVIDIA a presentare la prima GPU. Negli anni 1999-2000, gli studiosi di informatica e i ricercatori di diversi campi hanno iniziato a utilizzare le GPU per accelerare una vasta gamma di applicazioni scientifiche. Questo ha costituito l'avvento del movimento definito GPGPU[6] [19] o GPU Computing per finalità generali. Questa soluzione ha permesso agli utenti di ottenere prestazioni senza precedenti (in certi casi oltre 100 volte superiori a quelle delle CPU). Il problema dell'elaborazione GPGPU, però, era costituito dalla necessità di utilizzare API [7] di programmazione

[6] Acronimo di General-Purpose computing on Graphics Processing Units ovvero è quel settore della ricerca informatica che ha come scopo l'utilizzo del processore della scheda grafica GPU, impiegata per elaborazioni estremamente esigenti in termini di potenza di elaborazione, e per le quali le tradizionali architetture di CPU non hanno una capacità di elaborazione sufficiente.

[7] Acronimo di Application Programming Interface ovvero Interfaccia di Programmazione di un'Applicazione

grafica come OpenGL[8] e Cg (Computer graphics) per programmare la GPU. Questo requisito limitava la possibilità di sfruttare le eccezionali capacità delle GPU per scopi scientifici. Per tale motivo l'azienda NVIDIA ha capito l'enorme opportunità creata dalla possibilità di offrire queste prestazioni alla comunità scientifica ed ha quindi investito nella realizzazione di GPU pienamente programmabili e in grado di offrire un'esperienza anche con linguaggi molto usati dagli sviluppatori come C, C++ e Fortran. Il GPU Computing ha quindi iniziato a crescere con sempre maggiore forza. Oggi, alcuni dei supercomputer più veloci del mondo sono basati su GPU, 600 università di tutto il mondo offrono corsi sull'elaborazione in parallelo con GPU NVIDIA e centinaia di migliaia di sviluppatori utilizzano le GPU.

2.1 Confronto hardware fra GPU e CPU

Il GPU Computing affianca una GPU (unità di elaborazione grafica) a una CPU per accelerare l'elaborazione delle applicazioni scientifiche e tecniche. Alcuni anni fa NVIDIA ha esplorato per prima tale soluzione. Da allora il GPU Computing è rapidamente diventato uno standard del settore, sfruttato da milioni di utenti di tutto il mondo e adottato da quasi tutti i venditori di computing. Il GPU Computing offre prestazioni senza precedenti demandando le porzioni più impegnative dei calcoli di ogni applicazione alle GPU, mentre la parte restante del codice viene eseguita dalla CPU. Dal punto di vista dell'utente, l'unica differenza

[8] L'Open Graphics Library è una specifica che definisce una API per più linguaggi e per più piattaforme per scrivere applicazioni che producono computer grafica 2D e 3D.

è che le applicazioni sono nettamente più rapide. La combinazione di CPU e GPU è di straordinaria potenza perchè le CPU hanno un numero di core contenuto e sono ottimizzate per l'elaborazione seriale, mentre le GPU hanno migliaia di core più piccoli ed efficienti progettati per l'elaborazione in parallelo. Le porzioni seriali del codice vengono eseguite sulla CPU mentre le porzioni parallele vengono eseguite sulla GPU.

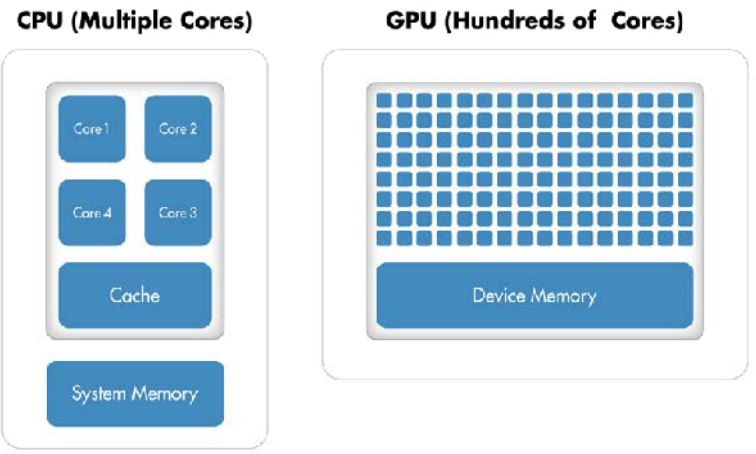


Figura 6: Differenza fra una CPU e una GPU.

Tutte le GPU NVIDIA, ovvero GeForce®, Quadro® e Tesla®, supportano il GPU Computing e il modello di programmazione in parallelo CUDA®. Si preferisce usare le GPU Tesla per i carichi di lavoro caratterizzati da valori critici di affidabilità dei dati e prestazioni complessive. Le GPU Tesla offrono il triplo delle prestazioni rispetto all'architettura precedente, ovvero più di un teraflop di operazioni in virgola mobile e doppia precisione, oltre a migliorare nettamente la

programmabilità e l'efficienza. Le GPU si presentano come dispositivi estremamente performanti in grado di elaborare con picchi fino a 500 Gflops/s (ATI X1900XT) che paragonati ai 12 Glops/s ottenibili su un Pentium4 3 GHz risultano sicuramente un aspetto interessante per il calcolo ad alte prestazioni dove si riescono a ottenere picchi per tre fattori fondamentali ovvero:

- la frequenza di clock;
- la velocità della memoria primaria;
- il volume delle informazioni processate in parallelo.

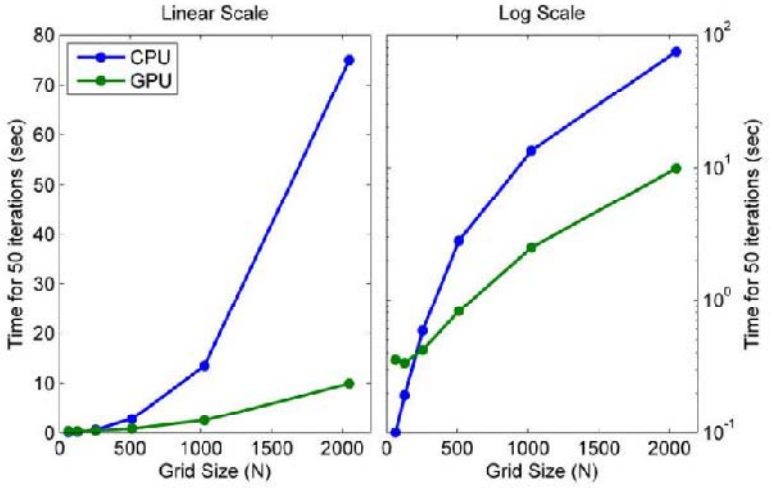


Figura 7: Plot del tempo richiesto da CPU e GPU per effettuare 50 iterazioni sia in scala lineare che in scala logaritmica.

Per valutare i vantaggi dell'uso di una GPU nella risoluzione di equazioni d'onda del secondo ordine, è stato presentato uno studio di prestazione dalla Mathworks [24] in cui è stata misurato il tempo che l'algoritmo richiede per eseguire 50 step di tempo per dimensioni di griglia (Grid

Size) di 64, 128, 512, 1024 e 2048 su un Intel® Xeon® Microprocessore X5650 ed usando poi un NVIDIA® Tesla C2050 GPU. Per una dimensione di griglia di 2048, l'algoritmo mostra un decremento di 7.5 volte in calcoli di tempo da più di un minuto sulla CPU a meno di 10 secondi sulla GPU (vedi figura 7). Il plot in scala logaritmica (Log Scale) mostra che la CPU è realmente più veloce per piccole dimensioni della griglia. Con l'evolversi della tecnologia le soluzioni della GPU sono in modo crescente capaci di occuparsi di più piccoli problemi e si presume che questa tendenza continui nel tempo.

2.2 Metodo di implementazione di uno schema FDTD in 2D con CPML su un GPU usando CUDA

Risulta interessante in particolare analizzare un metodo per l'implementazione di uno schema FDTD in 2D sulla GPU, ovvero l'unità di elaborazione grafica, utilizzando come architettura quella CUDA (Compute Unified Device Architecture) di NVIDIA che fornisce un esempio di computazione parallela in memoria condivisa nell'ambito di una copertura radio per sistemi di comunicazione per il mobile [23]. In tale ambito viene spiegato l'uso di differenti modelli di accesso di memoria nella simulazione FDTD e per ottenere un'efficienza computazionale i vari kernel sono stati progettati per la valutazione di regioni differenti. Gli algoritmi FDTD generalmente sono utilizzati per la valutazione di prestazioni di diversi sistemi su piccola scala come nel caso di antenne e circuiti microwave, tuttavia di recente ci si è concentrati sull'analisi dei

canali radio su larga scala come nel caso di sistemi di telecomunicazione per il mobile in ambienti chiusi e urbani. Finora l'FDTD, che risolve l'equazioni di Maxwell su una griglia discreta per la propagazione del campo elettromagnetico, non era stato preso in considerazione come modello per i canali radio ma solo recentemente grazie alla scoperta di due tecniche di accelerazione hardware, quali FPGA(Field-Programmable Gate-Arrays) e GPU, ha trovato applicazione. Da questo lavoro[23] però si evince che la velocità approssimata di un'implementazione in 2D risulta essere migliore in termini di prestazioni e di costi utilizzando una scheda GPU, come nella maggior dei PC moderni, che abbia una velocità di 450 Mcp(Mega celle per secondo) rispetto ad una FPGA con velocità 75Mcp. L'algoritmo FDTD può essere reso parallelo effettuando l'elaborazione contemporanea su più celle nello stesso step di tempo rendendo la previsione della copertura radio più veloce. Ad esempio si fa uso di un'implementazione FDTD multi-risoluzione su una GPU come tool per realizzare il modello di un canale microwave wireless utilizzando l'architettura CUDA poiché è risultata essere una soluzione molto veloce e flessibile. L'implementazione è formulata nel caso 2D sia per simulare la copertura in ambienti urbani come un appartamento sia per ambienti chiusi. Le stazioni di base GSM, UMTS e WiMAX usano tipicamente antenne polarizzate verticalmente. Per prevedere la propagazione di onde elettromagnetiche con polarizzazione verticale in un simulatore 2D è necessario restringere la formulazione delle equazioni di Maxwell al modo TM_z (campo elettrico polarizzato nella direzione verticale). Mettendo a confronto le diverse programmazioni

tradizionali, la GPU è capace di eseguire la gestione di molti thread simultaneamente. Con CUDA lo sviluppatore scrive i kernel che sono i pezzi di codice in C che ogni thread esegue. Il compilatore converte poi i kernel nell'insieme di istruzioni dell'architettura GPU. Diversi thread sono raggruppati in un blocco in modo che permette loro di cooperare con lo scambio di dati attraverso memoria condivisa. Siccome un blocco contiene solamente un numero limitato di thread, i blocchi possono essere raggruppati in una griglia, così da essere eseguita aumentando il numero totale di thread. I multiprocessori all'interno della GPU sono poi responsabili dell'esecuzione di blocchi di thread diversi. Questo è realizzato dividendo ogni blocco all'interno di gruppi SIMD (Singola Istruzione Dati Multipli) chiamati trame. Questo facilita la realizzazione su un'architettura SIMD perché elimina il bisogno di dovere decidere in ogni kernel se la cella corrente appartiene alla Absorbing Boundary Condition (ABC) o al Dominio Computazionale, di conseguenza il CPML (Convolutional Perfectly Matched Layer) può essere calcolato indipendentemente dal dominio computazionale. I kernel sono eseguiti più velocemente quando tutti i thread all'interno di una trama seguono lo stesso percorso di esecuzione. Se la condizione di decisione viene valutata differentemente per due o più thread all'interno della stessa trama, allora l'esecuzione dei thread dovrà essere serializzata, perdendo perciò il vantaggio della parallelizzazione. Questa caratteristica può essere sfruttata dividendo il calcolo all'interno dei diversi kernel per parti diverse della simulazione in esame. I compiti che ciascun kernel deve compiere per aggiornare i campi di FDTD sono: dapprima caricare i

valori di campo dalla precedente iterazione di tempo, poi calcolare il valore del campo nuovo e infine immagazzinare il risultato nella memoria. Per sfruttare l'architettura SIMD della GPU il carico di lavoro è diviso tra tutti i thread all'interno di ogni blocco, dove ogni thread aggiorna una sola cella che usa l'informazione di quelle vicine. Ad esempio, il pseudocodice del kernel che implementa l'aggiornamento di campo elettrico è mostrato nel seguente Algoritmo:

```
Calcolare i e j per il corrente thread;
Caricare H_x|^n_{i,j-1} nella memoria condivisa;
If threadIndex.y = 0 then
Load H_x|^n_{i,j-1} nella memoria condivisa;
end
Load H_y|^n_{i,j-1} nella memoria condivisa;
If threadIndex.x = 0 then
Load H_y|^n_{i-1,j} nella memoria condivisa;
End
Sincronizzare i thread;
Caricare l'indice del materiale;
Calcolare E_z|^{n+1/2}_{i,j};
Memorizzare il risultato nella memoria globale;
```

La dimensione del blocco è un importante parametro di progetto poichè i blocchi più grandi contengono più thread, infatti ogni thread carica un valore di campo (due valori del campo magnetico(H_x e H_y) per

l'aggiornamento kernel del valore del campo elettrico E_z) all'interno della memoria condivisa. C'è così una relazione diretta tra la dimensione del blocco e la quantità di memoria condivisa di cui il kernel ha bisogno. Inoltre, la memoria condivisa deve essere divisa fra tutti i blocchi che sono eseguiti su un multiprocessore. Un metodo efficiente di progettazione è quello di dimensionare i blocchi in funzione della geometria delle matrici che intervengono nella simulazione per massimizzare il rendimento (throughput) in termini di istruzione riducendo la divergenza tra le diverse trame. Un importante parametro in una simulazione di FDTD è lo step dello spazio (Δs) che deve essere più piccolo del più piccolo ostacolo all'interno del dominio computazionale. Le dimensioni della griglia e del blocco devono essere definite in modo tale che il numero totale di thread sia corrispondente al numero delle celle di FDTD. Nei casi in cui la dimensione dello scenario non è un multiplo della dimensione del blocco, ci saranno inevitabilmente alcuni thread che non calcolano nodi appartenenti alla simulazione, questo comporta un decremento del rendimento delle istruzioni (throughput). Ecco perchè è raccomandato che le dimensioni (N_x,Ny) nel numero di celle dello scenario di simulazione si avvicini il più possibile ad un multiplo della dimensione del blocco. Questo può essere fatto facilmente aggiustando Δs. Uno dei compiti principali dei kernel consiste nella lettura e scrittura di grandi matrici dalla memoria globale della scheda grafica. Questo è il collo di bottiglia principale dell'implementazione FDTD così esso è cruciale per usare uno schema di accesso di memoria adatto, in modo da ottenere la massima larghezza di banda

di memoria. Le schede grafiche di NVIDIA hanno quattro tipi diversi di spazi in memoria, offrendo così un range versatile di modelli di accesso di memoria. Poichè la sincronizzazione dell'esecuzione di diversi blocchi di thread o lo scambio di informazioni tra loro non è possibile con CUDA , ogni blocco individuale deve caricare i coefficienti di aggiornamento (proprietà di materiale) dalla memoria. Nel caso in cui nel dominio in esame siano presenti diversi materiali un blocco potrebbe essere sufficiente per caricare tali informazioni. Questa ridondanza nell'accesso di memoria può essere ridotta avvalendosi dello spazio in memoria costante, la quale è fornita da una cache su chip. Un altro spazio in memoria veloce è la memoria condivisa che risiede su chip. Tutti i thread all'interno dello stesso blocco scambiano l'informazione facendolo in modo sincronizzato e leggono e scrivono da e su tale memoria. Poichè l'aggiornamento di ogni cella in un algoritmo FDTD è basato sui valori precedenti dei campi nelle sue vicinanze, ogni cella (i,j) ha bisogno di essere letta almeno due volte in ogni iterazione. Per minimizzare l'accesso alla memoria globale, i thread del primo blocco caricano la corrispondente sottomatrice in memoria condivisa (si veda Algoritmo della pag.35) e dopo usano questi valori per compiere il calcolo. Così, l'accesso ad ogni cella nello spazio di memoria globale è compiuto solamente una volta.

3 Simulazioni: Prove sperimentali

Dopo un'attenta analisi del funzionamento dell'algoritmo FDTD in quest'ultima parte della tesi si implementato un codice di prova dell'algoritmo FDTD tramite l'utilizzo di Matlab. Esistono molti toolbox in rete, alcuni open source e gratuiti, che estendono le funzioni base di Matlab in differenti aree di applicazione. In particolar modo si è implementato il codice dell'algoritmo FDTD nel caso 2D in dominio illimitato, introducendo lo strato CPML prima descritto, sfruttando sia le potenzialità della CPU che della GPU per farne notare le differenze in termini di tempo nelle varie iterazioni. Tale codice è stato realizzato prendendo spunto da quanto riportato sul sito della MathWorks (*http://www.mathworks.com*) relativo all'algoritmo FDTD nel caso bidimensionale per architetture CPU che è stato stato adeguato alle esigenze del presente lavoro volte a mettere in risalto le differenze computazionali fra le architetture CPU e GPU dal punto di vista dell'accelerazione hardware. Nello sviluppo di tale codice è stato rilevante introdurre la sorgente di campo che poteva assumere sia una forma d'onda sinusoidale che di tipo gaussiana come riportato nel paragrafo 1.6 per verificarne l'andamento nelle varie simulazioni fatte sul piano x-y quando viene simulato un problema di tipo TM_z in aria, in cui cioè le uniche componenti di campo sono costituite da H_x, H_y, E_z nel sistema di coordinate cartesiano ortogonale x,y,z. Ovviamente le componenti di tali campi vengono aggiornate ad ogni passo temporale. I punti del campo sono definiti attraverso la cella di Yee [1], in cui la

parte di H_x è definita per ogni semi-coordinata y per ogni coordinata x e la parte di H_y è definita per ogni semi-coordinata x per ogni coordinata y e la parte di E_z è definita nei punti per ogni coordinata x ed y. Inoltre l'ampiezza del passo spaziale, uguale in tutte le direzioni, è posto pari ad 1 μm. Le matrici, utilizzate come fattori di moltiplicazione per aggiornare le equazioni, sono inizializzate prima dell'inizio del ciclo temporale per evitare i calcoli ripetuti in ogni iterazione di loop. Come condizione al contorno è utilizzata la condizione per lo strato CPML per la quale i campi vicino al confine sono attenuati a partire da una certa distanza dal confine (fissata a 25 μm) prima che essi arrivino al valore zero, modificando i valori della costante dielettrica e il valore della permeabilità magnetica per evitare riflessioni. Il campo E_z è suddiviso in due componenti E_{zx} ed E_{zy} tali componenti sono attenuate usando la conduttività elettrica e magnetica separatamente nelle due direzioni con i valori dati a sigmax e sigma_startx nella direzione x e sigmay e sigma_starty nella direzione y. Una sorgente di campo elettrico è posta al centro del dominio. La forma d'onda della sorgente può essere variata usando le variabili *sine*, *gaussian* ed *impulse* agendo così con una forma d'onda sinusoidale, gaussiana ed impulsiva.

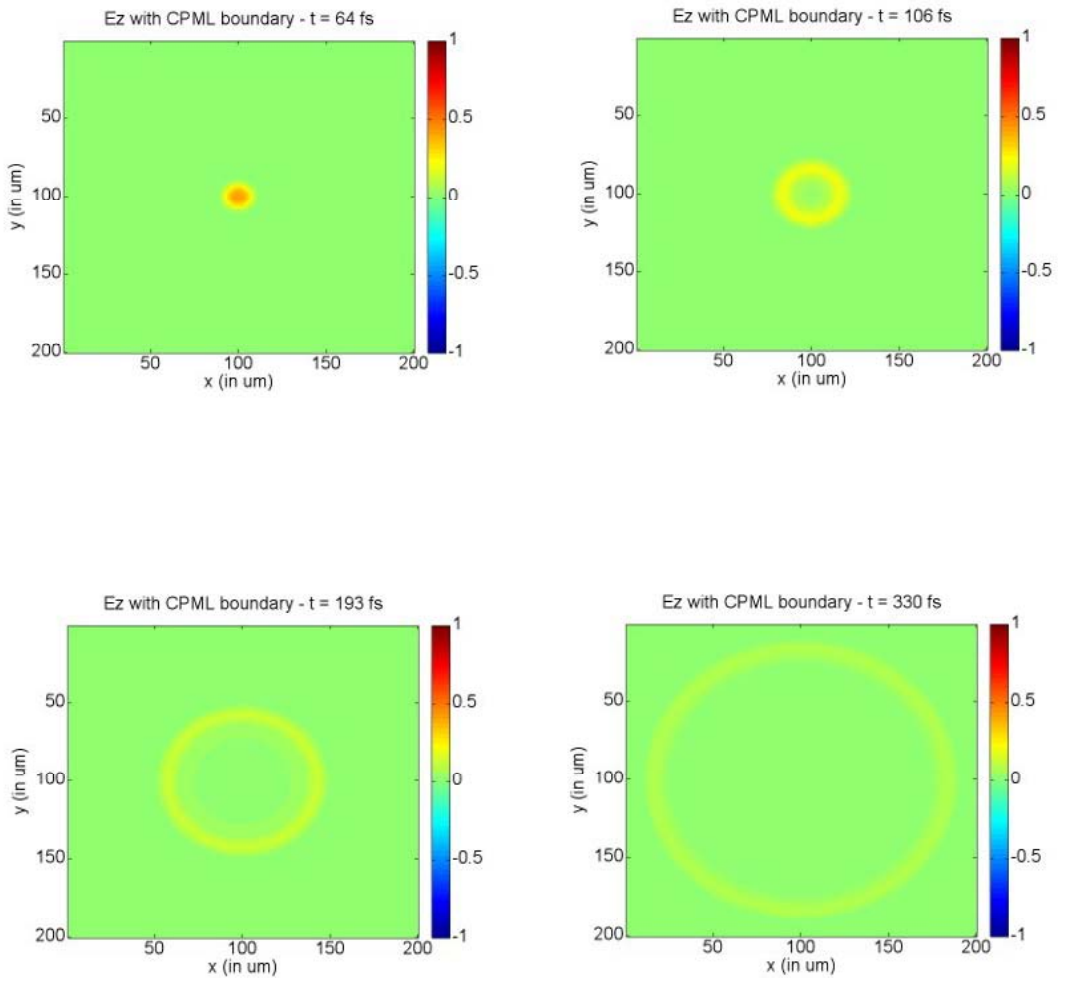


Figura 8: Andamento di Ez utilizzando come forma d'onda della sorgente quella gaussiana.

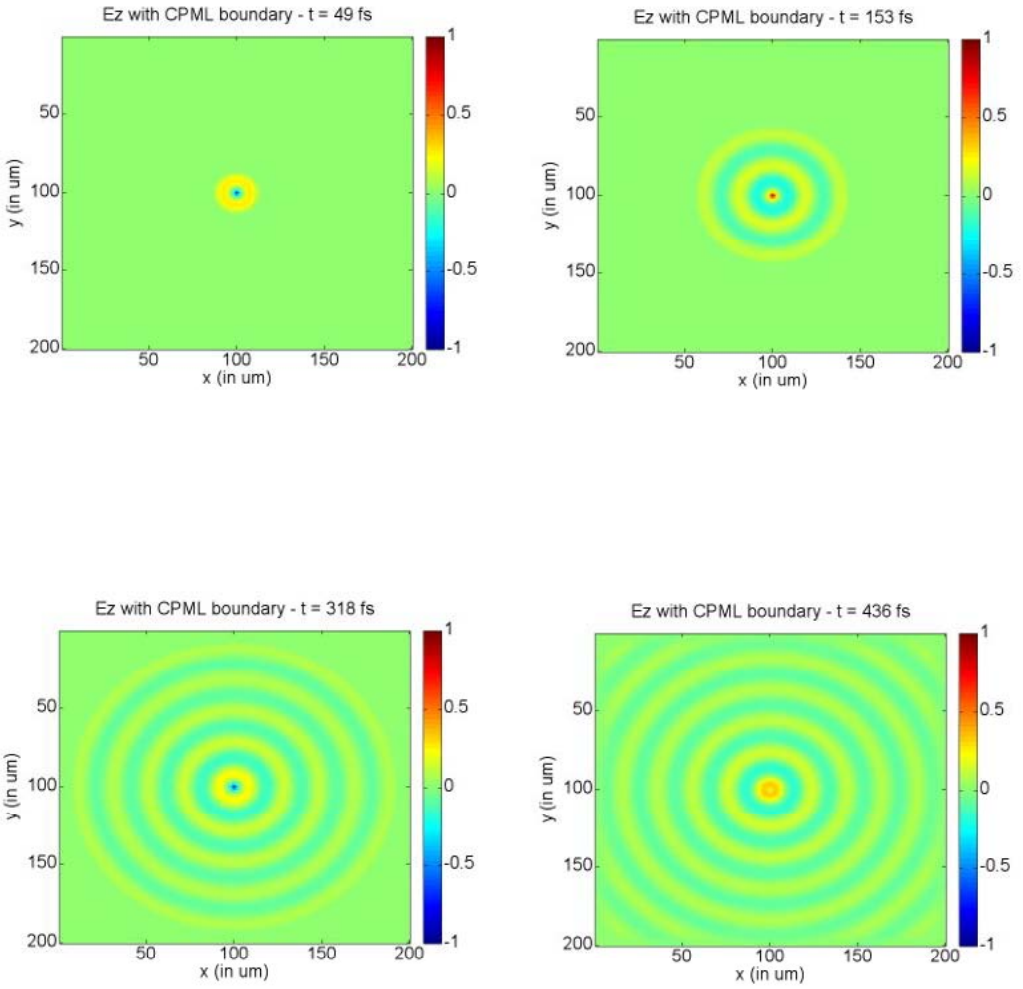


Figura 9: Andamento di Ez utilizzando come forma d'onda della sorgente quella sinusoidale.

Nelle figure 10 e 11 si confrontano le parti di codice modificate fra l'implementazione dell'algoritmo FDTD con CPML nel caso 2D usando le CPU e le GPU. In particolare si notano 15 differenze fra i due codici in questione dove:

Number of matching lines: 240;

Number of unmatched lines in left file: 14;

Number of unmatched lines in right file: 18.

Nella fig.12 sono state riportate le caratteristiche della piattaforma hardware di test utilizzata per le simulazioni, effettuate presso la sede di Palermo dell'istituto di calcolo e reti ad alte prestazioni del Consiglio Nazionale delle Ricerche (ICAR-CNR).

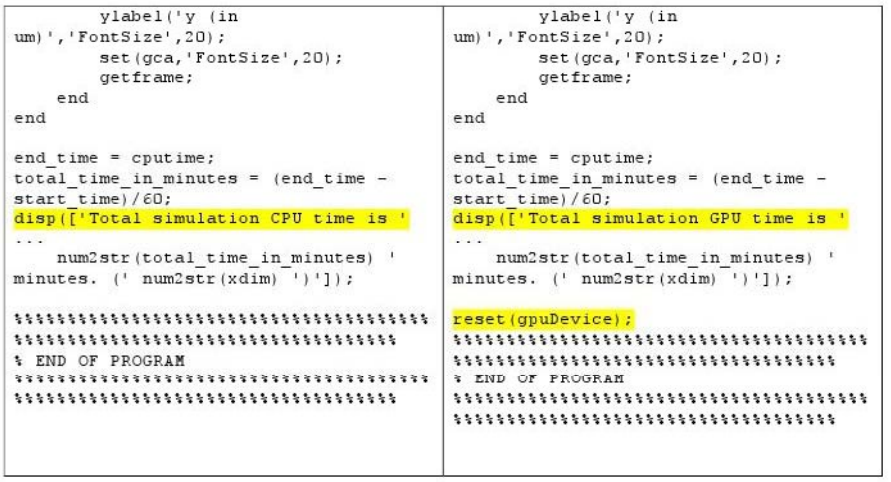
FDTD_CPU.m

```
% Parameters of free space
(permittivity and permeability and
speed of
% light) are all not 1 and are given
real values

epsilon0=(1/(36*pi))*1e-9;
mu0=4*pi*1e-7;
c=3e+8;

% Spatial grid step length (spatial
grid step = 1 micron and can be
changed)

delta=1e-6;

% Temporal grid step obtained using
Courant condition
deltat=S*delta/c;

% Initialization of field matrices

Ez=zeros(xdim,ydim);
Ezx=zeros(xdim,ydim);
Ezy=zeros(xdim,ydim);
Hy=zeros(xdim,ydim);
Hx=zeros(xdim,ydim);

% Initialization of permittivity and
permeability matrices

epsilon=epsilon0*ones(xdim,ydim);
mu=mu0*ones(xdim,ydim);


% Initialising electric conductivity
matrices in x and y directions

sigmax=zeros(xdim,ydim);
sigmay=zeros(xdim,ydim);


%Perfectly matched layer boundary
design

    if display==1

%Movie type colour scaled image plot of
Ez




imagesc(delta*1e+6*(1:1:xdim),(delta*1e
+6*(1:1:ydim))',Ez',[-1,1]);colorbar;
        title(['\fontsize{20}Ez with
CPML boundary - t =
',num2str(round(n*deltat*1e+15)),'
fs']);
        xlabel('x (in
um)','FontSize',20);
```

FDTD_GPU.m

```
% Parameters of free space
(permittivity and permeability and
speed of
% light) are all not 1 and are given
real values

epsilon0=gpuArray((1/(36*pi))*1e-9);
mu0=gpuArray(4*pi*1e-7);
c=gpuArray(3e+8);

% Spatial grid step length (spatial
grid step= 1 micron and can be
changed)

delta=gpuArray(1e-6);

% Temporal grid step obtained using
Courant condition
deltat=S*delta/c;

% Initialization of field matrices

Ez=gpuArray(zeros(xdim,ydim));
Ezx=gpuArray(zeros(xdim,ydim));
Ezy=gpuArray(zeros(xdim,ydim));
Hy=gpuArray(zeros(xdim,ydim));
Hx=gpuArray(zeros(xdim,ydim));

% Initialization of permittivity and
permeability matrices

epsilon=epsilon0*gpuArray(ones(xdim,ydi
m));
mu=mu0*gpuArray(ones(xdim,ydim));

% Initialising electric conductivity
matrices in x and y directions

sigmax=gpuArray(zeros(xdim,ydim));
sigmay=gpuArray(zeros(xdim,ydim));


%Perfectly matched layer boundary
design

    if display==1

%Movie type colour scaled image plot of
Ez
        deltat = gather(deltat);
        Ez = gather(Ez);

imagesc(delta*1e+6*(1:1:xdim),(delta*1e
+6*(1:1:ydim))',Ez',[-1,1]);colorbar;
        title(['\fontsize{20}Ez with
CPML boundary - t =
',num2str(round(n*deltat*1e+15)),'
fs']);
        xlabel('x (in
um)','FontSize',20);
```

Figura 10: Prima parte del codice a confronto per l'implementazione dell'algoritmo FDTD nel caso 2D con CPML con CPU(a sinistra) e GPU(a destra).



```
        ylabel('y (in
um)','FontSize',20);
        set(gca,'FontSize',20);
        getframe;
    end
end

end_time = cputime;
total_time_in_minutes = (end_time -
start_time)/60;
disp(['Total simulation CPU time is '
...
    num2str(total_time_in_minutes) '
minutes. (' num2str(xdim) ')']);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END OF PROGRAM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        ylabel('y (in
um)','FontSize',20);
        set(gca,'FontSize',20);
        getframe;
    end
end

end_time = cputime;
total_time_in_minutes = (end_time -
start_time)/60;
disp(['Total simulation GPU time is '
...
    num2str(total_time_in_minutes) '
minutes. (' num2str(xdim) ')']);

reset(gpuDevice);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END OF PROGRAM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Figura 11: Seconda ed ultima parte del codice a confronto per l'implementazione dell'algoritmo FDTD nel caso 2D con CPML con CPU(a sinistra) e GPU(a destra).

CPU E4 Server

2 x Xeon(R) E5520 Processor
vendor_id : GenuineIntel
cpu family : 6
model : 26
model name : Intel(R) Xeon(R) CPU E5520 @ 2.27GHz
stepping : 5
cpu MHz : 2266.801
cache size : 8192 KB
physical id : 0
siblings : 8
core id : 0
cpu cores : 4
apicid : 0
fpu : yes
fpu_exception : yes
cpuid level : 11
wp : yes
bogomips : 4533.60
clflush size : 64
cache_alignment : 64
address sizes : 40 bits physical, 48 bits virtual

RAM MemTotal : 24676200 kB

GPU Tesla S1070

2 x Tesla T10 Processor
CUDA Driver Version / Runtime Version 4.2 / 4.2
CUDA Capability Major/Minor version number: 1.3
Total amount of global memory: 4096 MBytes (4294770688 bytes)
(30) Multiprocessors x (8) CUDA Cores/MP: 240 CUDA Cores
GPU Clock rate: 1296 MHz (1.30 GHz)
Memory Clock rate: 800 Mhz
Memory Bus Width: 512-bit
Max Texture Dimension Size (x,y,z) 1D=(8192), 2D=(65536,32768), 3D=(2048,2048,2048)
Max Layered Texture Size (dim) x layers 1D=(8192) x 512, 2D=(8192,8192) x 512
Total amount of constant memory: 65536 bytes
Total amount of shared memory per block: 16384 bytes
Total number of registers available per block: 16384
Warp size: 32
Maximum number of threads per multiprocessor: 1024
Maximum number of threads per block: 512
Maximum sizes of each dimension of a block: 512 x 512 x 64
Maximum sizes of each dimension of a grid: 65535 x 65535 x 1
Maximum memory pitch: 2147483647 bytes
Texture alignment: 256 bytes
Concurrent copy and execution: Yes with 1 copy engine(s)
Run time limit on kernels: No
Integrated GPU sharing Host Memory: No
Support host page-locked memory mapping: Yes
Concurrent kernel execution: No
Alignment requirement for Surfaces: Yes
Device has ECC support enabled: No
Device is using TCC driver mode: No
Device supports Unified Addressing (UVA): No
Device PCI Bus ID / PCI location ID: 16 / 0
Compute Mode:
Default (multiple host threads can use ::cudaSetDevice() with device simultaneously)

Figura 12: Piattaforma hardware di test utilizzata per le simulazioni effettuate.

Le simulazioni sono state effettuate utilizzando la versione di Matlab 2011b con il Parallel Computing Toolbox. In particolare si sono presi in considerazione come numero di step i seguenti valori 1500, 1800, 2000 mentre come dimensioni dello spazio 2D i valori 200, 300, 400, 500, 600, 700, 1000, 1200 μm. Di seguito vengono riportati alcuni grafici nei quali vengono confrontati i tempi di elaborazione ottenuti per CPU e GPU, con riferimento ai valori prima individuati e al dominio 2D.

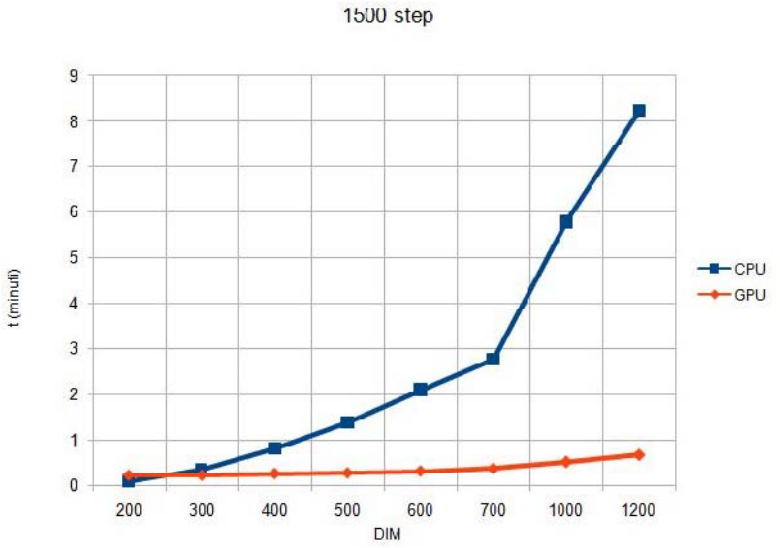


Figura 13: Andamento dei tempi di elaborazione in minuti di una CPU e una GPU con un numero di 1500 step temporali.

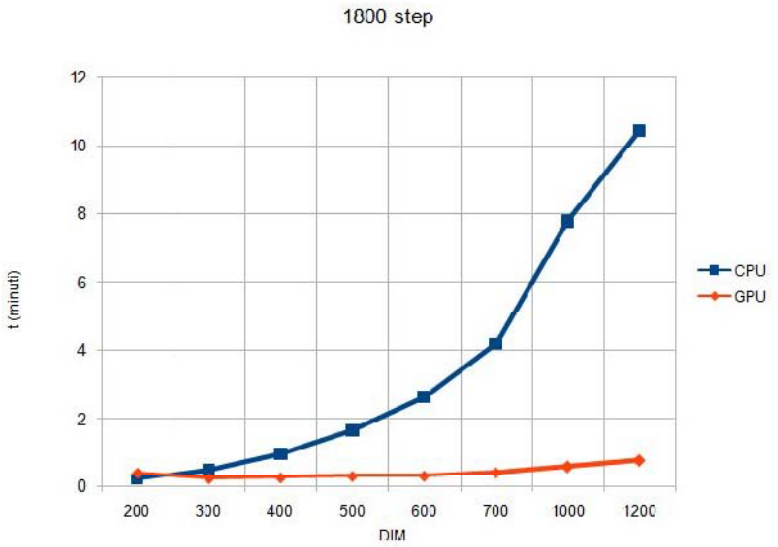


Figura 14: Andamento dei tempi di elaborazione in minuti di una CPU e una GPU con un numero di 1800 step temporali.

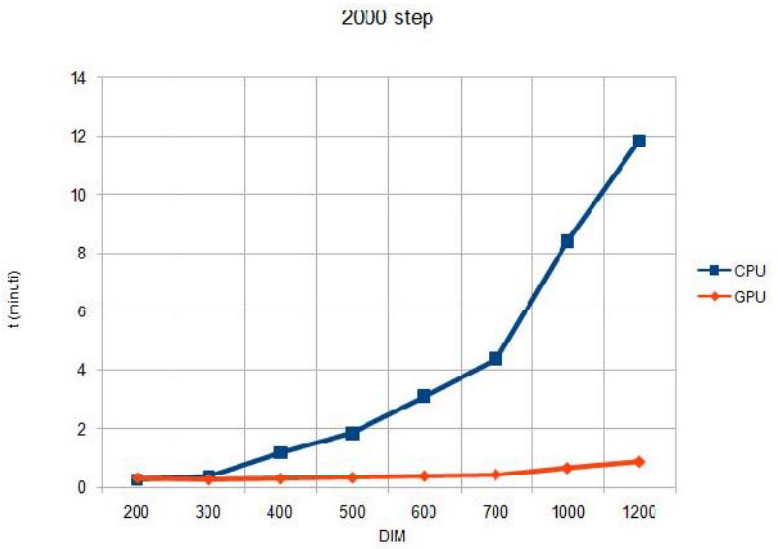


Figura 15: Andamento dei tempi di elaborazione in minuti di una CPU e una GPU con un numero di 2000 step temporali.

4 Conclusioni

Alla luce dei risultati sperimentali a cui si è giunti attraverso le simulazioni, si può affermare che le GPU sono estremamente vantaggiose rispetto ad una eleborazione effettuata tramite CPU, nel caso in cui il problema coinvolga un numero di celle crescente: la CPU risulta invece essere più performante per casi più semplici e con carichi di lavoro ridotti. Ciò è reso possibile ovviamente dall'hardware con cui sono realizzate le GPU che hanno a disposizione una quantità eleveta di core rispetto alle CPU, implementando in maniera parallela i vari carichi di lavoro. I risultati ottenuti concordano con quanto dichiarato sia dai produttori di hardware GPU (NVIDIA) sia da produttori di ambienti di calcolo scientifico (MathWorks). Si può quindi concludere che le architetture parallele multi-core possono essere efficacemente utilizzate per la risoluzione di problemi mediante l'algoritmo FDTD nei casi in cui la dimensionalità della simulazione risulti sufficientemente elevata. Per poter ottenere risultati analoghi su piattaforme basate esclusivamente su CPU sarebbero necessarie potenze di calcolo estremamente più elevate e costose.

Appendice A: codice sorgente

```
%Clearing variables in memory and Matlab command screen
clear all;
%clc;

start_time = cputime;
current_time = 0;

% display on/off (0 off)
display = 0;

% Grid Dimension in x (xdim) and y (ydim) directions
xdim=200;
ydim=xdim;

%Total no of time steps
time_tot=1500;

%Position of the source (center of the domain)
xsource=xdim/2;
ysource=ydim/2;

%Courant stability factor
S=1/(2^0.5);

% Parameters of free space (permittivity and permeability and speed of
% light) are all not 1 and are given real values
epsilon0=gpuArray((1/(36*pi))*1e-9);
mu0=gpuArray(4*pi*1e-7);
c=gpuArray(3e+8);

% Spatial grid step length (spatial grid step= 1 micron and can be changed)
delta=gpuArray(1e-6);
% Temporal grid step obtained using Courant condition
deltat=S*delta/c;

% Initialization of field matrices
Ez=gpuArray(zeros(xdim,ydim));
Ezx=gpuArray(zeros(xdim,ydim));
Ezy=gpuArray(zeros(xdim,ydim));
Hy=gpuArray(zeros(xdim,ydim));
Hx=gpuArray(zeros(xdim,ydim));

% Initialization of permittivity and permeability matrices
epsilon=epsilon0*gpuArray(ones(xdim,ydim));
mu=mu0*gpuArray(ones(xdim,ydim));

% Initializing electric conductivity matrices in x and y directions
sigmax=gpuArray(zeros(xdim,ydim));
sigmay=gpuArray(zeros(xdim,ydim));
```

```
%Perfectly matched layer boundary design
%Reference:-http://dougneubauer.com/wp-content/uploads/wdata/yee2dpml1/yee2d_c.txt
%(An adaptation of 2-D FDTD TE code of Dr. Susan Hagness)

%Boundary width of PML in all directions
bound_width=25;

%Order of polynomial on which sigma is modeled
gradingorder=6;

%Required reflection co-efficient
refl_coeff=1e-6;

%Polynomial model for sigma
sigmamax=(-log10(refl_coeff)*(gradingorder+1)*epsilon0*c)/(2*bound_width*delta);
boundfact1=((epsilon(xdim/2,bound_width)/epsilon0)*sigmamax)/((bound_width^gradingorder)*(gradingorder+1));
boundfact2=((epsilon(xdim/2,ydim-bound_width)/epsilon0)*sigmamax)/((bound_width^gradingorder)*(gradingorder+1));
boundfact3=((epsilon(bound_width,ydim/2)/epsilon0)*sigmamax)/((bound_width^gradingorder)*(gradingorder+1));
boundfact4=((epsilon(xdim-bound_width,ydim/2)/epsilon0)*sigmamax)/((bound_width^gradingorder)*(gradingorder+1));
x=0:1:bound_width;
for i=1:1:xdim
    sigmax(i,bound_width+1:-1:1)=boundfact1*((x+0.5*ones(1,bound_width+1)).^(gradingorder+1)-
                                (x-0.5*[0 ones(1,bound_width)]).^(gradingorder+1));
    sigmax(i,ydim-bound_width:1:ydim)=boundfact2*((x+0.5*ones(1,bound_width+1)).^(gradingorder+1)-
                                     (x-0.5*[0 ones(1,bound_width)]).^(gradingorder+1));
end
for i=1:1:ydim
    sigmay(bound_width+1:-1:1,i)=boundfact3*((x+0.5*ones(1,bound_width+1)).^(gradingorder+1)-
                                (x-0.5*[0 ones(1,bound_width)]).^(gradingorder+1))';
    sigmay(xdim-bound_width:1:xdim,i)=boundfact4*((x+0.5*ones(1,bound_width+1)).^(gradingorder+1)-
                                     (x-0.5*[0 ones(1,bound_width)]).^(gradingorder+1))';
end

%Magnetic conductivity matrix obtained by Perfectly Matched Layer condition
%This is also split into x and y directions in Berenger's model
sigma_starx=(sigmax.*mu)./epsilon;
sigma_stary=(sigmay.*mu)./epsilon;

%Choice of nature of source
%Choose any one as 1 and rest as 0. Default (when all are 0): Unit time step
gaussian=1;
impulse=0;
sine=0;
% The user can give a frequency of his choice for sinusoidal (if sine=1 above) waves in Hz
frequency=1.5e+13;

%Multiplication factor matrices for H matrix update to avoid being calculated many times
%in the time update loop so as to increase computation speed
G=((mu-0.5*deltat*sigma_starx)./(mu+0.5*deltat*sigma_starx));
```

```
H=(deltat/delta)./(mu+0.5*deltat*sigma_starx);
A=((mu-0.5*deltat*sigma_stary)./(mu+0.5*deltat*sigma_stary));
B=(deltat/delta)./(mu+0.5*deltat*sigma_stary);

%Multiplication factor matrices for E matrix update to avoid being calculated many times
%in the time update loop so as to increase computation speed
C=((epsilon-0.5*deltat*sigmax)./(epsilon+0.5*deltat*sigmax));
D=(deltat/delta)./(epsilon+0.5*deltat*sigmax);
E=((epsilon-0.5*deltat*sigmay)./(epsilon+0.5*deltat*sigmay));
F=(deltat/delta)./(epsilon+0.5*deltat*sigmay);

N_lambda=c/(frequency*delta);
% Update loop begins
for n=1:1:time_tot

    %if source is impulse or unit-time step
    if gaussian==0 && sine==0 && n==1
        Ezx(xsource,ysource)=0.5;
        Ezy(xsource,ysource)=0.5;
    end

    % Setting time dependent boundaries to update only relevant parts of the
    % matrix where the wave has reached to avoid unnecessary updates.
    if n<xsource-2
        n1=xsource-n-1;
    else
        n1=1;
    end
    if n<xdim-1-xsource
        n2=xsource+n;
    else
        n2=xdim-1;
    end
    if n<ysource-2
        n11=ysource-n-1;
    else
        n11=1;
    end
    if n<ydim-1-ysource
        n21=ysource+n;
    else
        n21=ydim-1;
    end

    %matrix update instead of for-loop for Hy and Hx fields
    Hy(n1:n2,n11:n21)=A(n1:n2,n11:n21).*Hy(n1:n2,n11:n21)+B(n1:n2,n11:n21).*(Ezx(n1+1:n2+1,n11:n21)-Ezx(n1:n2,n11:n21)+
                      Ezy(n1+1:n2+1,n11:n21)-Ezy(n1:n2,n11:n21));
    Hx(n1:n2,n11:n21)=G(n1:n2,n11:n21).*Hx(n1:n2,n11:n21)-H(n1:n2,n11:n21).*(Ezx(n1:n2,n11+1:n21+1)-Ezx(n1:n2,n11:n21)+
                      Ezy(n1:n2,n11+1:n21+1)-Ezy(n1:n2,n11:n21));

    %matrix update instead of for-loop for Ez field
```

```
    Ezx(n1+1:n2+1,n11+1:n21+1)=C(n1+1:n2+1,n11+1:n21+1).*Ezx(n1+1:n2+1,n11+1:n21+1)+
                               D(n1+1:n2+1,n11+1:n21+1).*(-Hx(n1+1:n2+1,n11+1:n21+1)+Hx(n1+1:n2+1,n11:n21));
    Ezy(n1+1:n2+1,n11+1:n21+1)=E(n1+1:n2+1,n11+1:n21+1).*Ezy(n1+1:n2+1,n11+1:n21+1)+
                               F(n1+1:n2+1,n11+1:n21+1).*(Hy(n1+1:n2+1,n11+1:n21+1)-Hy(n1:n2,n11+1:n21+1));

    % Source conditions
    if impulse==0
        % If unit-time step
        if gaussian==0 && sine==0
            Ezx(xsource,ysource)=0.5;
            Ezy(xsource,ysource)=0.5;
        end
        %if sine
        if sine==1
            tstart=1;
            Ezx(xsource,ysource)=0.5*sin(((2*pi*(c/(delta*N_lambda))*(n-tstart)*deltat)));
            Ezy(xsource,ysource)=0.5*sin(((2*pi*(c/(delta*N_lambda))*(n-tstart)*deltat)));
        end
        %if gaussian
        if gaussian==1
            if n<=42
                Ezx(xsource,ysource)=(10-15*cos(n*pi/20)+6*cos(2*n*pi/20)-cos(3*n*pi/20))/64;
                Ezy(xsource,ysource)=(10-15*cos(n*pi/20)+6*cos(2*n*pi/20)-cos(3*n*pi/20))/64;
            else
                Ezx(xsource,ysource)=0;
                Ezy(xsource,ysource)=0;
            end
        end
    else
        %if impulse
        Ezx(xsource,ysource)=0;
        Ezy(xsource,ysource)=0;
    end

    Ez=Ezx+Ezy;

    if display==1
        %Movie type colour scaled image plot of Ez
        deltat = gather(deltat);
        Ez = gather(Ez);
        imagesc(delta*1e+6*(1:1:xdim),(delta*1e+6*(1:1:ydim))',Ez',[-1,1]);colorbar;
        title(['\fontsize{20}Ez with CPML boundary - t = ',num2str(round(n*deltat*1e+15)),' fs']);
        xlabel('x (in um)','FontSize',20);
        ylabel('y (in um)','FontSize',20);
        set(gca,'FontSize',20);
        getframe;
    end
end

end_time = cputime;
total_time_in_minutes = (end_time - start_time)/60;
```

```
disp(['Total simulation GPU time is ' ...
    num2str(total_time_in_minutes) ' minutes. (' num2str(xdim) ')']);

reset(gpuDevice);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END OF PROGRAM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Riferimenti bibliografici

[1] K.S. Yee, *Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media*, IEEE Transaction on Antennas and Propagation, 1966.

[2] S. Ramo - T. Van Duzer - J.R. Whinnery, *Campi e onde nell'elettronica per le comunicazioni*, Franco Angeli.

[3] A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Artech House, 2005.

[4] R. Courant - K. Friedrichs - H. Lewy, *On the partial difference equations of mathematical physics*, IBM Journal of Research and Development, 1967.

[5] J. Roden - S. Gedney, *Convolution PML (CPML): an efficient FDTD implementation of the CFS-PML for arbitrary media*, Microwave an Optical Technology Letters, 2000.

[6] A. Elsherbeni - V. Demir, *The Finite-Difference Time-Domain Method for Electromagnetics with MATLAB Simulations*, Schitech, 2009.

[7] W.H.A. Schilders - E.J.W. Ter Maten, *Numerical Methods in Electromagnetics, Volume 13: Special Volume*, North Holland, 2005.

[8] Z. Ma - K. Nomiyama - Y. Kobayashi, *Microstrip Lowpass Filters with Reduced Size and Improved Stopband Characteristics*, IEICE Transactions on Electronics, 2005.

[9] J. Sheng Hong - M.J. Lancaster, *Microstrip Filters for RF/Microwave Applications*, John Wiley Sons, 2001.

[10] D.M. Sheen - S.M. Ali - M.D. Abouzahra - J. Kong, *Application of the Three-Dimensional Finite-DIfference Time-Domain Method to the Analysis of Planar Microstrip Circuits*, IEEE Transactions on Microwave Theory and Techiques, vol. 38, no. 7, July, 1990.

[11] G. D'Inzeo - F. Giannini - R. Sorrentino, *Novel microwave integrated low-pass filters*, Electron. Lett., vol.15, no. 9, Apr. 26, 1979.

[12] Sean E. Krakiwsky - Laurence E. Tumer - Michal M. Okoniewski, *Acceleration of Finite-Difference Time-Domain (FDTD) Using Graphics Processor Units (GPU)*, [J]. IEEE MTTS Digest, 2004.

[13] Tomasz P. Stefanski - Timothy D. Drysdale, *Acceleration of the 3D ADIFDTD Method Using Graphics Processor Units*, [J]. IEEE MTT-S International Microwave Symposium Digest (MTT), 2009.

[14] Price - D.K. - Humphrey, *GPU-Based Accelerated 2D and 3D FDTD Solvers*, [J]. Physics and Simulation of Optoelectronic Devices XV, San Jose, CA.

[15] NVIDIA Company, *NVIDIA Fermi Compute Architecture Whitepaper*, Version 1.1[S].2009.

[16] A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method Third Edition*, Norwood, MA: Artech House, 2005.

[17] Adams, S. US Air Force Res. Lab., Brooks City-Base Payne, J. ; Boppana, R., *Finite Difference Time Domain (FDTD) Simulations Using Graphics Processors*, DoD High Performance Computing Modernization Program Users Group Conference, 2007.

[18] Inman, M.J. Dept. of Electr. Eng., Univ. of Mississippi, University, MS Elsherbeni, A.Z., *MATLAB graphical interface for GPU based FDTD method*, Electromagnetic Compatibility and 19th International Zurich Symposium on Electromagnetic Compatibility, 2008. APEMC 2008. Asia-Pacific Symposium.

[19] Unno, M. Dept. of Syst. Eng., Shizuoka Univ., Shizuoka, Japan Inoue, Y. ; Asar, H. , *GPGPU-FDTD Method for 2-Dimensional Electromagnetic Field Simulation and Its Estimation*, Electrical Performance of Electronic Packaging and Systems, 2009. EPEPS '09. IEEE 18th Conference on.

[20] Feng Jin Xi'an Jiao Tong Univ., Xi'an, China 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.

[21] Zhang Bo Sch. of Inf. Electron., Beijing Inst. of Technol., Beijing, China Xue Zheng-hui, Ren Wu, Li Wei-ming, Sheng Xin-qing, *Accelerating FDTD algorithm using GPU computing*, Microwave Technology Computational Electromagnetics (ICMTCE), 2011 IEEE International Conference.

[22] Bo Zhang Sch. of Inf. Electron., Beijing Inst. of Technol., Beijing, China Zhenghui Xue, Wu Ren, Weiming Li, Xinqing Sheng, *Research on GPU Cluster acceleration of FDTD algorithm*, Microwave and Millimeter Wave Technology (ICMMT), 2012 International Conference.

[23] Alvaro Valcarce and Jie Zhang, *Implementing a 2D FDTD scheme with CPML on a GPU using CUDA*, 2010 IEEE International Conference.

[24] Jill Relese, MathWorks and Sarah Zaranek, *GPU Programming in MATLAB*, 2011 The MathWorks, Inc.