GPU によるアンテナの数値解析の高速化に関する一検討

勝田 肇, 今野 佳祐, 陳 強, 澤谷 邦男 (東北大学大学院工学研究科),
横川 佳, 袁 巧微 (仙台高等専門学校)
瀬在 俊浩 (宇宙航空研究開発機構)

概要: モーメント法を用いた大規模なアンテナの電磁界数値解析では, 行列方程式を解くために多大な演算時間がかかるという問題がある. そこで, 行列方程式を高速に解くために, 近年では画像処理装置 (Graphics Processing Units, GPU) が盛んに用いられている. GPU の高い演算能力を最大限に活かすには, 各プロセッサの演算の割り当てやメモリアクセスを最適化する必要があるが, それらが最適化されたという報告はこれまでにはされていない. 本報告では, 行列方程式の反復解法の一種である共役勾配 (Conjugate Gradient, CG) 法を GPU によって高速化する際に, これらの要素が演算時間に与える影響を定量的に明らかにし, 最適化したので報告する.
キーワード: モーメント法, GPU, 共役勾配法

1. まえがき

電磁界数値解析の有力な手法の一つとしてモーメント法 (Method of Moments, MoM) が知られている [1], [2]. モーメント法では, アンテナや散乱体表面の電流分布を求める問題を, N 個の電気的に小さなセグメント上における未知の電流係数を求める問題に置換する. そして, 未知の電流係数を求めるために, 電界積分方程式を離散化して得られる行列方程式を解く. 一般的に, 掃き出し法のような直接法によって行列方程式を解く場合, 未知の電流係数ベクトルを求めるために $O(N^3)$ という大きな時間がかかることが知られている. したがって, 直接法は N の大きな大規模行列方程式に適応できない. そこで, 行列方程式を解く演算時間を $O(N^2)$ にする試みが近年盛んに行われており, 共役勾配 (Conjugate Gradient, CG) 法のような反復法はその代表である [3]-[6].

一方, 近年では, PC や WorkStation に用いられる画像処理装置 (Graphics Processing Units, GPU) の高い並列演算能力を数値解析に応用する試みが盛んに行われている. 特に, GPU を汎用目的で利用するための統合開発環境である CUDA (Compute Unified Device Architecture) [7]-[9] が 2006 年に NVIDIA 社より発表されて以降, GPU によってモーメント法を含む様々な数値解析法が高速化されてきた [10]-[13]. モーメント法の行列方程式を, GPU と CG 法を組み合わせて解く試みも行われており, その有効性は既に確認されている [14]. しかし, 文献 [14] では GPU 内での各プロセッサへの演算の割り当てやメモリアクセスは最適化されておらず, GPU の性能を最大限に活かしているとは言い難い. 本報告では, これらの要素を最適化し, GPU によって CG 法を従来より大きく高速化したので報告する.

2012 年 10 月 16 日
東北大学 電気・情報系 103 会議室

2. 共役勾配法のアルゴリズム

モーメント法における行列方程式は $\mathbf{ZI} = \mathbf{V}$ で表される. ここで, $\mathbf{Z}$ は既知の $N \times N$ インピーダンス行列, $\mathbf{V}$ は既知の N 次元電圧係数ベクトル, $\mathbf{I}$ は未知の N 次元電流係数ベクトルである. 以下に, CG 法によって $\mathbf{ZI} = \mathbf{V}$ を解くアルゴリズムを示す.

1. 初期値 (近似解)$\mathbf{I}_0$ を適当に決定.
2. 第 0 近似解 $\mathbf{I}_0$ に対する残差ベクトル $\mathbf{r}_0$ を計算.
$$\mathbf{r}_0 = \mathbf{V} - \mathbf{ZI}_0$$
3. 修正ベクトルの初期値 $\mathbf{p}_1$ を計算.
$$\mathbf{p}_1 = \mathbf{Z}^\dagger \mathbf{r}_0$$
$\mathbf{Z}^\dagger$ は $\mathbf{Z}$ の複素共役転置行列.
4. 第 $i(i \geq 1)$ 回の修正係数 α_i を計算.
$$\alpha_i = -\frac{\langle \mathbf{Zp}_i, \mathbf{r}_{i-1} \rangle}{\|\mathbf{Zp}_i\|^2} = \frac{\|\mathbf{Z}^\dagger \mathbf{r}_{i-1}\|^2}{\|\mathbf{Zp}_i\|^2}$$
5. 第 i 近似解 $\mathbf{I}_i$ を計算.
$$\mathbf{I}_i = \mathbf{I}_{i-1} + \alpha_i \mathbf{p}_i$$
6. 第 i 近似解 $\mathbf{I}_i$ に対する残差ベクトル $\mathbf{r}_i$ を計算.
$$\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i \mathbf{Zp}_i$$
7. 修正ベクトル $\mathbf{p}_i$ の修正係数 β_i を計算.
$$\beta_i = \frac{\|\mathbf{Z}^\dagger \mathbf{r}_i\|^2}{\|\mathbf{Z}^\dagger \mathbf{r}_{i-1}\|^2}$$
8. 修正ベクトルの新しい値 $\mathbf{p}_{i+1}$ を計算.
$$\mathbf{p}_{i+1} = \mathbf{Z}^\dagger \mathbf{r}_i + \beta \mathbf{p}_i$$
9. 収束性を判定し, 収束していなければ, $i = i+1$ とし, ステップ 4 に戻る.

上記のアルゴリズムにおいて, ステップ 4 からステップ 9 までの処理は近似解 $\mathbf{I}_i$ が厳密解に十分近づくまで繰り返し行われる. その反復処理のうち, 演算時間の大半を占めるのは 2 回の行列-ベクトル積演算 $\mathbf{Zp}_i$ と $\mathbf{Z}^\dagger \mathbf{r}_i$ である. そこで本報告では, GPU によって行列-ベクトル積演算 $\mathbf{Zp}_i$, $\mathbf{Z}^\dagger \mathbf{r}_i$ を高速化し, 演算時間の短縮を図る.

3. GPU の構造と演算の割り当て, メモリアクセス

3.1 GPU の構造

本報告で用いた GPU, Tesla C2075 の構造を図 1 に示す. 図 1 において, 破線の左側は CPU のプロセッサとそれに取り付けられたメモリを示している. そして, 破線の右側は GPU 内のスケジューラ, プロセッサ群, メモ

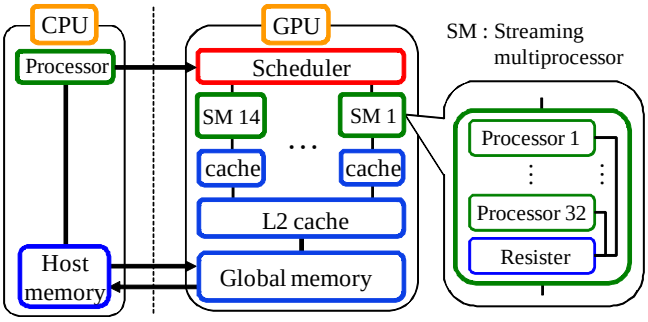

図 1: Tesla C2075 の構造.

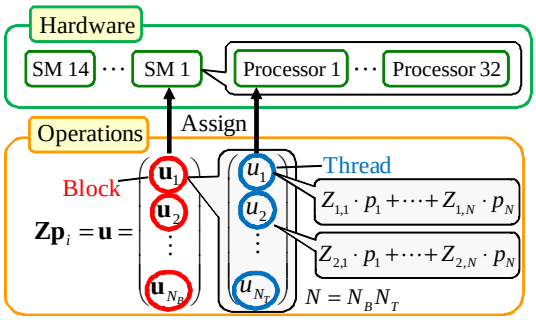

図 2: 行列-ベクトル積演算におけるスレッドとブロック.

リ群を示している. GPU では, 多数のプロセッサが SM (Streaming Multiprocessor) というグループでまとめられている. Tesla C2075 では, 448 個のプロセッサが 14 の SM に均等に振り分けられており, 1SM は 32 個のプロセッサで構成される. 各 SM は, プロセッサ以外に, 演算に必要な共有メモリやレジスタといった資源を保有する.

GPU による演算の流れを説明する. まず, CPU 側のホストメモリから演算に必要なデータを GPU 側のグローバルメモリにコピーする. 次に, CPU から GPU へ演算内容を指示する. そして, GPU は与えられた演算を多数のプロセッサが並列に実行する. この際, 演算に必要なデータは適宜グローバルメモリから呼び出す. 各プロセッサの演算が全て終了した後, 演算結果をグローバルメモリからホストメモリにコピーし, GPU による演算は終了となる. なお, 本報告では以上の命令を全て CUDA Fortran で書かれたコードによって行う.

3.2 演算の割り当て

GPU に演算を命令するときは, 演算内容のスレッドとブロックを GPU 側に明示する必要がある. スレッドとは並列演算可能な最小単位のことであり, スレッドの集合をブロックという. 例えば, 行列-ベクトル積演算 $\mathbf{Zp}_i$ で得られるベクトルを $\mathbf{u}$ と置けば, $\mathbf{u}$ の 1 つの要素を求める演算は互いに並列に実行可能であるから, 図 2 のようにスレッドとブロックを定めることができる. 図 2 において, ブロック行列を求める演算がブロック, 各要素を求める演算がスレッドに当たる. また, ブロックの数を N_B, 1 ブロック当たりに含まれるスレッドの数を N_T とすると, N_B と N_T の間には $N = N_B N_T$ の関係が成り立つ. GPU では, 多数のプロセッサがさらに多数のスレッドを並列に実行していくことによって, CPU に対して演算時間を短縮することができる.

図 2 に示すように, GPU のスケジューラはブロック単位で各 SM に演算を振り分け, SM 内の各プロセッサがブロック内の各スレッドを実行する. 全てのプロセッサを休みなく動作させるためには, N_B を SM の数 14 の倍数, N_T を 1SM 当たりのプロセッサ数 32 の倍数にすればよいと考えられる. しかし, プロセッサの稼働率は 1 スレッドの演算に必要となるレジスタ数などによっても制限されるため, N_B と N_T を最適化するだけでは演算時間が短くなるとは限らない.

3.3 メモリアクセス

GPU では, 各プロセッサの演算速度に対し, グローバルメモリからのデータ転送速度が遅いという特徴がある. プロセッサがグローバルメモリに一回アクセスするためにかかる時間 T_a は以下のように表される.

$$T_a = t_r + t_d \cdot S_d \tag{1}$$

ここで, t_r はメモリの立ち上がり時間, t_d は単位バイト当たりのデータ転送時間, S_d は転送されるデータのサイズ (単位はバイト) を意味する. Tesla C2075 では, S_d は 32, 64, 128 バイトのいずれかとなり, 128 バイトのデータ転送なら倍精度複素数 8 要素を一度に転送することができる.

メモリの立ち上がり時間は, メモリアクセスの度にデータのサイズに依らず一定時間かかる. したがって, 同じ量の情報を転送するならば一度に送る情報量を増やし, アクセス回数を減らした方が良い [9]. それを実現するメモリアクセスのことをコアレスアクセスと呼ぶ. コアレスアクセスとは, 複数のプロセッサに対してデータを個別に転送するのではなく, まとめて転送するメモリアクセスであり, メモリアクセスの回数を減らすことができる. 128 バイトのコアレスアクセスの条件を以下に挙げる.

1. 一度に転送される 8 要素のデータのアドレスが, グローバルメモリ上で連続していること.
2. 一度に転送される 8 要素のデータのアドレスが, 128 バイト境界をまたいでいないこと.

GPU のプロセッサがグローバルメモリに格納されたインピーダンス行列 $\mathbf{Z}$ にアクセスする様子を図 3 に示す. 上記 2 つのコアレスアクセスの条件を満たしている場合は, プロセッサからのメモリアクセスはコアレスアクセスとなり, 一度のメモリアクセスで 8 個のプロセッサへ一つずつデータを供給することができる. しかし, 一度に転送されるデータのアドレスがメモリ上で連続していない場合や, 128 バイト境界をまたいでいる場合はコアレスアクセスとはならず, 複数回のデータ転送が必要となる. アド

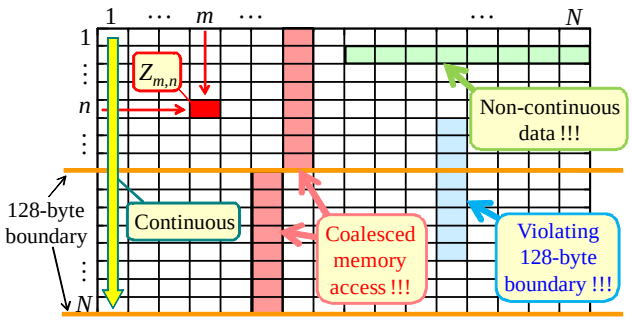

図 3: プロセッサによるメモリに格納されたインピーダンス行列 **Z** へのアクセス.

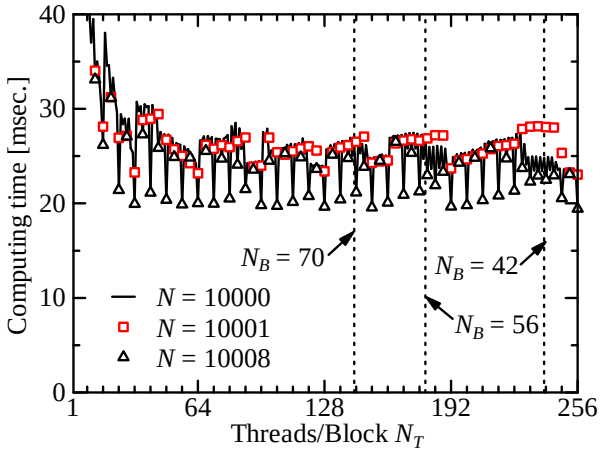

図 4: N_T に対する行列-ベクトル積演算の演算時間.

レスをメモリ上で連続とするためには, 使用言語に応じてコード作成時に配列の定義に注意すればよい. 本報告で用いた CUDA Fortran では, インピーダンス行列 **Z** を表す 2 次元配列は列方向から順にグローバルメモリに格納される. 128 バイト境界をまたがないようにするためには, N 及び N_T が共に 128 バイト分の要素数である 8 の倍数である必要があると考えられる.

4. 数値解析

本節では, 行列-ベクトル積演算 $\mathbf{Z}\mathbf{p}_i$ 及び $\mathbf{Z}^\dagger\mathbf{r}_i$ と, CG 法によって行列方程式 $\mathbf{ZI} = \mathbf{V}$ を解いた結果を報告する. 本報告における数値解析環境は表 1 の通りである.

また, CG 法の反復処理は相対残差ノルム $\varepsilon < 10^{-4}$ となった場合に収束と判定することとした.

表 1: 数値解析環境.

CPU	Intel Xeon @ 2.27 GHz
GPU	NVIDIA Tesla C2075
Compiler	PGI Visual Fortran 11.5
CUDA Driver	CUDA 3.2
OS	Windows 7 Professional 64 bit

4.1 各プロセッサへの演算の割り当てやメモリアクセスの影響

行列-ベクトル積演算 $\mathbf{Z}\mathbf{p}_i$ において, 1 ブロック当たりのスレッド数 N_T に対する演算時間を測定した結果を図 4 に示す. ここで, ブロック数 N_B は $N_B = N/N_T$ により定まり, **Z** 及び $\mathbf{p}_i$ はそれぞれ乱数行列及び乱数ベクトルとした. また, 一度に転送されるデータのアドレスはグローバルメモリ上で連続となるようにした. 図 4 より, N および N_T が共に 8 の倍数のときに演算時間が短くなっていることが分かる. これは一度に転送されるデータのアドレスが 128 バイト境界をまたがず, コアレスアクセスが実現されたためと考えられる. それ以外では 128 バイト境界をまたいでいるため, コアレスアクセスにならず, 演算時間が長くなったと考えられる. また, N_B を 14 の倍数とした場合や N_T を 32 の倍数とした場合に, 他の N_T が 8 の倍数の場合と比べて演算時間が明らかに短くなることはなかった. これは, プロセッサの稼働率が, 1 スレッドの演算に必要となるレジスタ数などによって制限されたためと考えられる. この結果より, N_B 及び N_T を最適化しただけでは演算時間を短縮することはほとんどできないと言える.

本報告では, N_T は図 4 において演算時間が短くなっているものから 64 を選択した. N_T が自由に選択できるのに対し, N は解析するモデルによって決定される. そこで, N が 8 の倍数でない場合はメモリに 0 を挿入し, メモリアクセスがコアレスアクセスとなるようにする.

CG 法における行列ベクトル積演算 $\mathbf{Z}\mathbf{p}_i$ と $\mathbf{Z}^\dagger\mathbf{r}_i$ を比較すると, 行列の転置のためにインピーダンス行列 **Z** へのアクセス順序が異なる. そのため, 一方で一度に転送されるデータのアドレスがグローバルメモリ上で連続となるようにすると, もう一方では連続とすることができない. そこで, $\mathbf{Z}\mathbf{p}_i$ においてアドレスが連続となるようにした場合の $\mathbf{Z}\mathbf{p}_i$ 及び $\mathbf{Z}^\dagger\mathbf{r}_i$ の演算時間を図 5 に示す. 図 5 より, $\mathbf{Z}^\dagger\mathbf{r}_i$ は, $\mathbf{Z}\mathbf{p}_i$ に対して約 4.7 倍の時間がかかっていることが分かる. これは, $\mathbf{Z}^\dagger\mathbf{r}_i$ では一度に転送されるデータのアドレスがグローバルメモリ上で連続となるようにしなかったため, コアレスアクセスが実現されなかったことが原因と考えられる.

基底関数と試行関数を一致させる Galerkin のモーメント法では, インピーダンス行列 **Z** は複素対称行列になることが知られており, $\mathbf{Z} = \mathbf{Z}^{\mathrm{T}}$, すなわち $\mathbf{Z}^* = \mathbf{Z}^\dagger$ となる. したがって, 本報告では, 行列-ベクトル積演算 $\mathbf{Z}^\dagger\mathbf{r}_i$ において行列を転置せず, $\mathbf{Z}^*\mathbf{r}_i$ として演算することで, 演算時間の短縮を図ることとする.

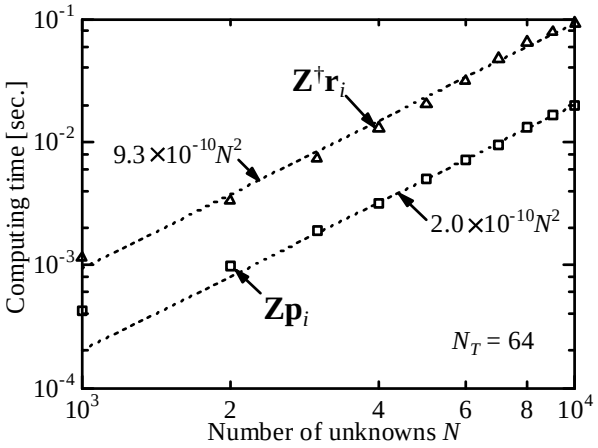

図 5: 行列-ベクトル積演算 $\mathbf{Zp}_i$ 及び $\mathbf{Z}^\dagger\mathbf{r}_i$ の演算時間.

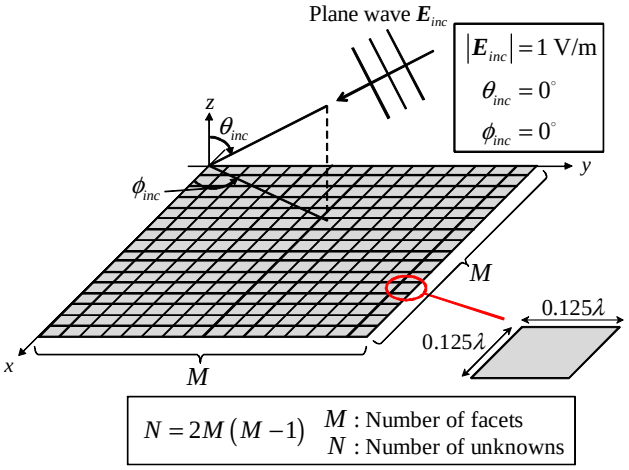

図 6: 平面アンテナモデル.

4.2 CG 法におけるメモリアクセスの最適化

本報告における数値解析モデルを図 6 に示す. 図 6 の数値解析モデルより得られたモーメント法の行列方程式 $\mathbf{ZI} = \mathbf{V}$ を CG 法によって解いた場合の演算時間を図 7 に示す. 図 7 より, GPU においてメモリアクセスを最適化し, コアレスアクセスとした場合, 最適化していない場合に対して約 2.5 倍, CPU に対して約 100 倍の高速化を達成できていることが分かる. この結果より, メモリアクセスの最適化の有効性を確認することができた.

5. むすび

本報告では, GPU を用いた行列-ベクトル積演算における GPU のプロセッサへの演算の割り当てや, プロセッサからのメモリアクセスが演算時間に与える影響を定量的に調べた. その結果, 演算の割り当てを最適化するには, ブロック数や 1 ブロック当たりのスレッド数を最適化するだけでは不十分であることが分かった. 演算の割り当ての最適化には, 1 スレッド当たりのレジスタ使用量も考

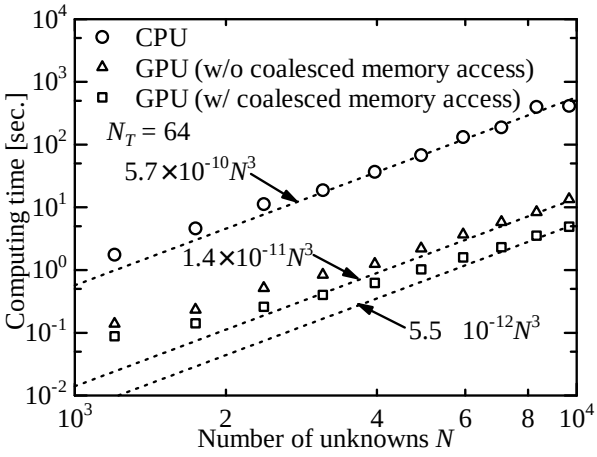

図 7: CG 法によって行列方程式 $\mathbf{ZI} = \mathbf{V}$ を解く演算時間.

慮する必要があると考えられる. また, コアレスアクセスの条件を満たすことで, メモリアクスが最適化され, 演算時間が短縮されることが分かった. その際, Galerkin のモーメント法におけるインピーダンス行列の対称性を利用した. 最後に, 行列-ベクトル積演算におけるメモリアクセスを最適化することにより, CG 法を従来よりも大きく高速化することができた.

謝辞

本研究の数値解析結果の一部は、東北大学サイバーサイエンスセンター大規模科学計算システムを利用して得られた。

参考文献

[1] R.F. Harrington, Field Computation by Moment Methods, New York, Macmillan, 1968.

[2] J.H. Richmond and N.H. Greay, “Mutual impedance of nonplanar-skew sinusoidal dipoles,” IEEE Trans. Antennas Propag., vol.23, no.5, pp.412-414, May 1975.

[3] T.K. Sarker and S.M. Rao, “The application of the conjugate gradient method for the solution of electromagnetic scattering from arbitrarily oriented wire antennas,” IEEE Trans. Antennas Propag., vol.32, no.4, pp.398-403, April 1984.

[4] T.K. Sarker, “The conjugate gradient method as applied to electromagnetic field problemsm,” IEEE Antennas Propag. Society Newsletter, vol.28, no.4, pp.4-14, Aug. 1986.

[5] J. Tang, “Numerical aspects of iterative solving of linear systems derived from helmholtz’s problem,” Literature Report of Delft University of Technology, Feb. 2004.

[6] T.K. Sarker, K.R. Siarkiewicz and S.M. Rao, “Survey of numerical methods for solution of large systems of linear equations for electromagnetic field problems,” IEEE Trans. Antennas Propag., vol.AP-29, no.6, pp.847-856, Nov. 1981.

[7] NVIDIA Corporation, “CUDA zone - the resource for CUDA developers,” http://www.nvidia.com/cuda, 参照 Sept. 5, 2012.

[8] 加藤 稔, “NVIDIA GPU の構造と CUDA スレッディングモデル,” ソフテック株式会社, http://www.softek.co.jp/SPG/Pgi/

TIPS/public/accel/gpu-accel2.html, 参照 Sept. 5, 2012.

[9] 情報基盤センター, “CUDA プログラミング講習会,” 理化学研究所, http://www.riken.jp/HPC/training/2010-4.html, 参照 Sept. 5, 2012.

[10] S. Peng and Z. Nie, “Acceleration of the method of moments calculations by using graphics processing units,” IEEE Trans. Antennas Propag., vol.56, no.7, pp.2130-2133, July 2008.

[11] E. Lezar and D.B. Davidson, “GPU-accelerated method of moments by example: monostatic scattering, ” IEEE Antennas Propag. Mag., vol.52, no.6, Dec. 2010.

[12] Piotr Sypek, Adam Dziekonski and Michal Mrozowski, “How to render FDTD computations more effective using a graphics accelerator,” IEEE Trans. Magnetics, vol.45, no.3, March 2009

[13] D.D. Donno, A. Esposito, L. Tarricone and L. Catarinucci, “Introduction to GPU computing and CUDA programming: a case study on FDTD,” IEEE Antennas Propag. Mag., vol.52, no.3, June 2010.

[14] D.D. Donno, A. Esposito, G. Monti and L. Tarricone, “MPIE/MoM acceleration with a general-purpose graphics processing unit,” IEEE Trans. Microw. Theory Tech., July 2012.