社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

GPUによる共役勾配法の高速化に関する一検討

勝田　肇[†]　今野　佳祐[†]　陳　強[†]　澤谷　邦男[†]
横川　佳[††]　袁　巧微[††]　瀬在　俊浩[†††]

† 東北大学大学院 工学研究科 通信工学専攻 〒 980-8579 仙台市青葉区荒巻字青葉 6-6-05
†† 仙台高等専門学校 〒 989-3128 宮城県仙台市青葉区愛子中央 4-16-1
††† 宇宙航空研究開発機構 〒 182-8522 東京都調布市深大寺東町 7-44-1
E-mail: †katsuda@ecei.tohoku.ac.jp

あらまし　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GPU (Graphics Processing Units) 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, CPU-GPU間での演算の配分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

A Study of Acceleration of Conjugate Gradient Method Using GPU

Hajime KATSUDA[†], Keisuke KONNO[†], Qiang CHEN[†], Kunio SAWAYA[†], Kei YOKOKAWA[††],
Qiaowei YUAN[††], and Toshihiro SEZAI[†††]

† Department of Communications Engineering, Graduate School of Engineering, Tohoku University 6-6-05 Aramaki Aza Aoba, Aoba-ku, Sendai, Miyagi, 980-8579, Japan
†† Sendai National College of Technology 4-16-1 Ayashichuuou, Aoba-ku, Sendai-shi, Miyagi, 989-3128 Japan
††† Japan Aerospace Exploration Agency, Chofu Aerospace Center 7-44-1 Jindaiji Higashimachi, Chofu, Tokyo, 182-8522, Japan
E-mail: †katsuda@ecei.tohoku.ac.jp

Abstract　In method of moments (MoM), large computing time consumption is a serious problem for solving a large-scale matrix equation. To overcome this problem, Graphics Processing Units (GPU) have been used for acceleration of matrix equation solver in recent years. In GPU computing, it is assumed that assignment of numerical operation to each processor, memory access scheme by each processor and assignment of numerical operation between CPU and GPU should be optimized for reduction of computing time. In this report, Conjugate Gradient (CG) method is accelerated by optimized GPU and its computing time is quantitatively evaluated.
Key words　MoM, GPU, Conjugate Gradient method

1. はじめに

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 の画像処理を担うデバイスである GPU (Graphics Processing Units) 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, CPU-GPU 間の演算の配分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.

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 $\mathbf{Z}$ の共役複素転置行列を意味する.
4. 第 $i(i{\geqq}1)$ 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-ベクトル積演算を高速化し, 演算時間の短縮を図る.

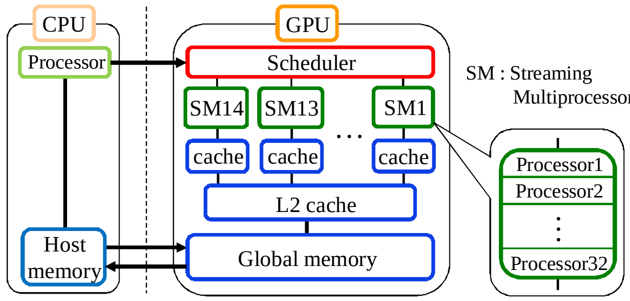

図 1 Tesla C2075 の構造.

3. GPU の構造と演算, メモリアクセス

3.1 GPU の構造

本報告で用いた, NVIDIA 社製の 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 の場合, 448 個のプロセッサが 14 個の SM に 32 個ずつ振り分けられている. 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 Host memory から演算に必要なデータを GPU 側の Global memory 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各 SM で並列に実行する. この際, 演算に必要なデータは適宜 Global memory 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 Global memory から Host memory 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 で書かれたプログラムによって行う.

3.2 マルチスレッド化

GPU に演算を命令するときは, 演算内容のうちどれが並列に演算可能なのかを GPU 側に明示する必要がある. これをマルチスレッド化といい, 並列演算可能な最小単位をスレッド, 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 $\mathbf{Ab} = \mathbf{c}$ では, $\mathbf{c}$ の各要素を求める演算は並列に実行可能であるから, 図 2 のようにマルチスレッド化できる. ここで, $\mathbf{A}$ は行列, $\mathbf{b}$ 及び $\mathbf{c}$ はベクトルである. 図 2 において, ブロック行列がブロック, 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. N_{B} と N_{T} の間には $N = N_{\mathrm{B}}N_{\mathrm{T}}$ の関係が成り立つ.

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演算していく. Tesla C2075 では, SM は 14 個, 各 SM 内のプロセッサの数は 32 個であるから, 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 14 の倍数, N_{T} を 32 の倍数とする必要があると考えられる. しかし, プロセッサの稼働率は 1 スレッド

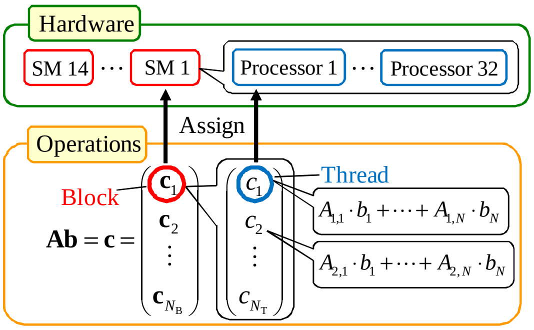

図 2　行列-ベクトル積演算のマルチスレッド化.

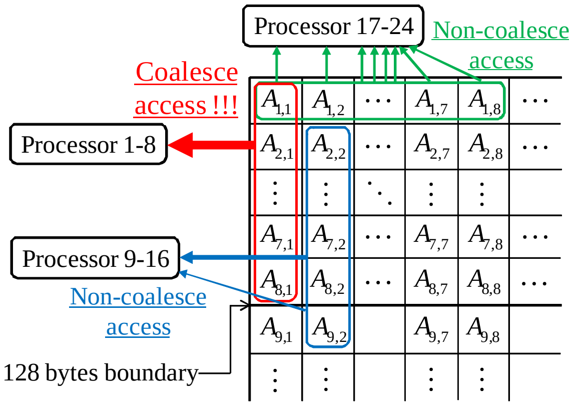

図 3　コアレスアクセスの仕組み.

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. なお, CUDA による GPU 制御では, N_{T} のみ自由に設定でき, N_{B} は N/N_{T} と定まる.

3.3　コアレスアクセス

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

$$t_{access} = t_{rise} + \alpha_{trans} \cdot N_{trans} \tag{1}$$

ここで, t_{rise} は立ち上がり時間, α_{trans} は単位バイト当たりのデータ転送時間, N_{trans} は転送されるデータのバイト数 (本 GPU では 32, 64, 128 バイト) を意味する. 立ち上がり時間はメモリにアクセスする度に一定時間かかるため, 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る. コアレスアクセスになるための条件を以下に挙げる.

- 一度に転送できるデータ量は最大 128 バイト.
- 転送するデータがメモリ上で連続していること.
- メモリの 128 バイト境界に合わせて実行される.

128 バイトとは倍精度複素数では 8 要素分のデータであり, 使用言語が CUDA Fortran の場合は行列の各要素はメモリ上で列方向に連続する.

GPU のプロセッサによるメモリアクセスの様子を図 3 に示す. 図 3 において, Processor 1-8 はそれぞれが必要とするデータをまとめて 1 回のメモリアクセスで取得している. これがコアレスアクセスである. これに対し, Processor 9-16 は 128 バイト境界を挟んでデータを要求しているため, 2 回のメモリアクセスが必要となる. これは N_{B} と N_{T} のうち, どちらか一方でも 8(単精度では 16) の倍数でない場合に起こると考えられる. また, Processor 17-24 はメモリ上で連続していない行方向のデータを要求しているため, 8 回のメモリアクセスが必要となる. 加えて, データ転送の最小単位は 32 バイトであるため, 各プロセッサはメモリから 32 バイトのデータを転送してもらい, そこから必要とする 16 バイトのデータだけを取得するという非効率なメモリアクセスになる. これは, 行列を転置して演算する場合に起こると考えられる.

4.　数 値 解 析

本報告における数値解析環境は表 1 の通りである.

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

また, CG 法の反復処理は残差ノルムが 10^{-4} を下回った場合に収束と判定することとした.

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. N_{B} 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. これはコアレスアクセスが実現されているためと考えられる. また, ブロック数 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, GPU の各プロセッサへの演算の割り当てを最適化しても, 演算時間の短縮には必ずしも大きな効果はないと言える.

本報告では, なるべくコアレスアクセスとなるように, N が 8 の倍数でない場合はメモリにダミーのデータを挿入することとする. また, 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.

4.2　行列の転置の影響

行列-ベクトル積演算における, 行列の転置の有無が演算時間に与える影響を図 5 に示す. 図 5 より, 行列を転置した場合は, しない場合に対して約 4.7 倍の時間がかかっていることが分かる. これは, 3.3 節で述べたように, 行列を転置した場合はメモリアクセスがコアレスアクセスにならないためと考えられる.

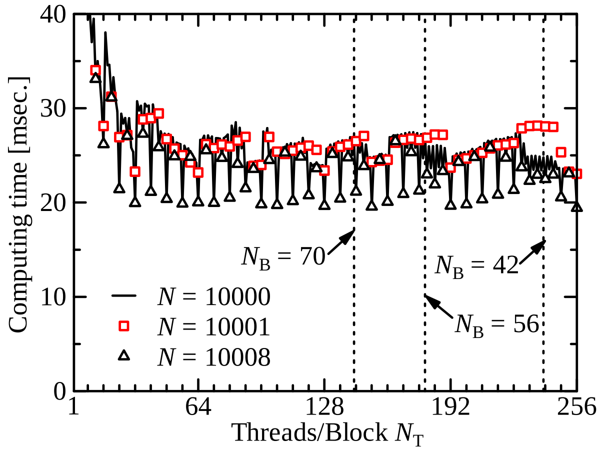

図 4　$N_{\rm T}$ に対する行列-ベクトル積演算の演算時間.

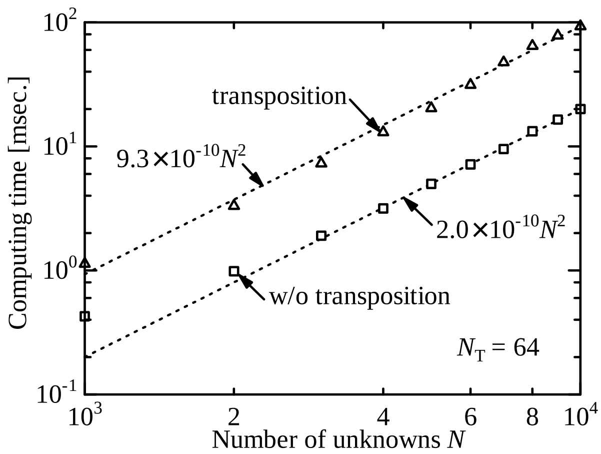

図 5　行列の転置の有無が行列-ベクトル積演算の演算時間に与える影響.

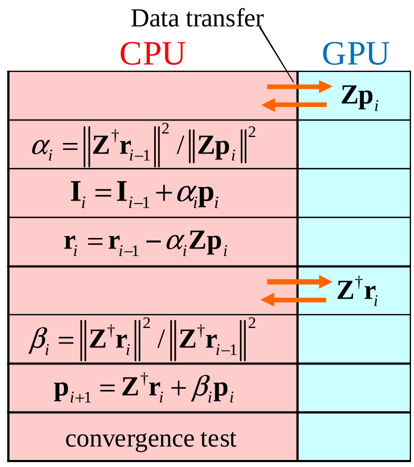

図 6　行列-ベクトル積演算のみを GPU で実行する構成 (Case 1).

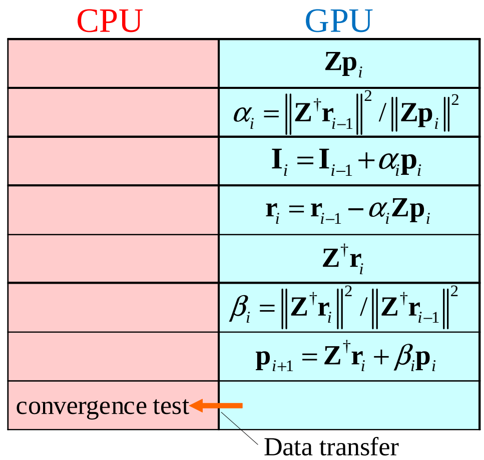

図 7　ほぼ全ての演算を GPU で実行する構成 (Case 2).

モーメント法において, 基底関数と試行関数を一致させる Galerkin 法を用いれば, インピーダンス行列は複素対称行列になることが知られており, $\mathbf{Z} = \mathbf{Z}^{\rm T}$ とすることができる．したがって, 本報告では, 行列-ベクトル積演算において行列を転置しないことで, 演算時間を短縮する.

4. 3　CPU-GPU 間の演算の割り当てによる影響

CG 法の演算を CPU と GPU に割り当てる際, 図 6, 7 に示す 2 種類の構成が考えられる.

図 6 に示す構成では, GPU による高速化の効果が大きいと思われる行列-ベクトル積演算のみを GPU で実行し, 他の演算は CPU で実行する (Case 1). この構成では, 行列-ベクトル積演算以外の演算にはほとんど時間がかからないが, CG 法の反復処理一回につき N 次ベクトルのデータ転送が 2 往復分かかると考えられる.

図 7 に示す構成では, CPU-GPU 間のデータ転送を最小限にするために収束判定のみ CPU で行い, 他のほぼ全ての演算は GPU で実行する (Case 2). この構成では, CG 法の反復処理一回につき N 次ベクトルのデータ転送が GPU から CPU への一回しか必要ないが, ベクトルノルムの演算などの GPU が苦手とする逐次的な演算に時間がかかると考えられる.

CG 法によって $\mathbf{ZI} = \mathbf{V}$ を解く際に, CPU のみを用いた場合, GPU を用いて CPU-GPU 間の演算の配分を Case 1, または Case 2 とした場合にそれぞれにかかる演算時間を図 8 に示す．ここで, $\mathbf{Z}$ は複素対称行列, $\mathbf{V}$ は乱数ベクトル, $\mathbf{I}$ は初期値 0 の未知ベクトルとした．図 8 より, Case 1 と Case 2 ではわずかに Case 1 の方が演算時間が短いが, ほとんど差がないことが分かる．これは, $N = 10^3$-10^4 の範囲で, Case 1 におけるデータ転送にかかる時間と, Case 2 における行列-ベクトル積演算以外の演算にかかる時間がほぼ等しいためであるが, 将来, GPU によって扱える N の範囲が非常に大きくなれば異なる結果となる可能性がある.

また, Case 1 の場合, CPU のみを用いた演算に対して約 100 倍の高速化を達成されていることが分かる．これは, GPU の様々な要素を最適化した結果である.

5.　む　す　び

本報告では, CG 法を GPU によって実行する際, 各プロセッサへの演算の割り当てやメモリアクセス, CPU と GPU への演算の配分が演算時間に与える影響を定量的に検証した．その結果, 行列-ベクトル積演算において, モーメント法の未知数の数

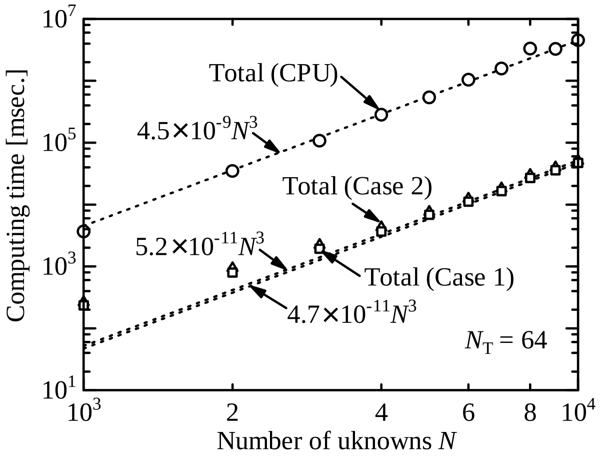

図 8　CPU-GPU 間の演算の配分による影響.

及び GPU による演算の 1 ブロック当たりのスレッド数を共に 8 の倍数とすることでコアレスアクセスを実現し, 演算時間を短縮することができた. また, インピーダンス行列の対称性から, 行列-ベクトル積演算でインピーダンス行列の転置をとる必要のないことを利用してコアレスアクセスを実現し, 演算時間を短縮することもできた.

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。

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.