Vol.2011-HPC-131 No.2
2011/10/6

気象モデルの高解像度計算のGPU化

二 星 義 裕[†1]　朴 泰 祐[†1,†2]　塙 敏 博[†1,†2]
池 田 亮 作[†3]　日 下 博 幸[†2,†3]　飯 塚 悟[†4]

近年，画像表示の用途に開発されてきたGPUを科学技術計算等に適用したGPGPUの研究が盛んに行われている．気象分野において，計算を高速に実行する要求は強い．気象分野においては，大量の計算と高いメモリバンド幅が要求される処理が多く，数値流体力学モデルのLESや次世代気象モデルASUCA等のGPU化が進められている．しかし，都市街区スケールの気象を空間詳細に再現し，複雑な地形効果を取り入れたLES気象モデルのGPU化は，ほとんど例がない．本研究では，計算コストが高い高解像度気象モデルのGPU化を行う．本研究ではNVIDIA社のCUDAアーキテクチャを対象とする．LESコード全体をGPU化するのは難しいため，通常のCPUにおける処理のプロファイル結果を元に，処理のコストの高いルーチンについて，順次GPU化を行い，性能を評価する．

1. はじめに

近年，GPU（Graphics Processing Unit）の持つ高い浮動小数点演算処理能力と高いメモリバンド幅が注目され，GPUをグラフィックス計算以外の汎用計算に用いるGPGPU（General-Purpose GPU）の研究が盛んに行われている．GPUは多数のプロセッサコア群を搭載しており，単一のプロセッサコア群は複数のコアで構成され，単純な命令セットを各々のコアに割り当てることで高い演算実行を処理することができる．また，GPUはそれぞれのコアで，同一の命令を多数の計算スレッドに対して均一的に実行するSIMT（Single Instruction Multiple Threads）アーキテクチャを採用している．汎用CPUと比べGPUは非常に高い並列性・演算性能・メモリ転送性能を備えていることから，数値流体力学をはじめ，分子動力学，重力多体計算や高速フーリエ変換などGPUを利用した研究が精力的に進められている[1),2)]．大量の演算と高いメモリバンド幅が求められる気象計算の分野においても、GPUを利用する取り組みが開始されている．東京工業大学学術国際情報センターでは，工学系の数値流体力学モデルとしてのLESモデル（Large Eddy Simulation）[3)]や気象庁が開発を進める次世代気象計算のプロダクション・コードASUCAの力学過程を含むコードをGPUに対応させている[4)]．一方，複雑な地形を対象としたLES気象モデルのGPU化はほとんど例がない．そこで本研究では，複雑な地形の効果を取り入れた一般曲線座標系によるLES気象モデルをGPU計算に対応させる．

†1 筑波大学大学院システム情報工学研究科
†2 筑波大学　計算科学研究センター
†3 筑波大学大学院生命環境科学研究科
†4 名古屋大学大学院環境学研究科

2. 背　景

気象モデルの解析手法として従来使用されている解析手法は計算時間が短い半面，乱流の予測精度が低いことが問題視されていた．これに代わる手法として期待されるLES解析により，数値予測精度において大幅な改善が期待され，近年，都市のヒートアイランド研究などにおける気象モデルの高解像度計算の一手段として注目されている．LES計算は，理想計算を対象としてきたため地形の導入はなく地面は平坦であるものが多い．そこで筑波大学の池田らによって，複雑な地形の効果を取り入れた一般曲線座標系によるLES気象モデルが開発されてきた[5),6)]．このような気象モデルでは計算量が多く，計算結果を得るまでに長い時間を必要とする．そこで，本研究では池田らの開発した気象モデルをGPUに対応させ計算時間を短縮する．

3. LES気象モデル

池田らの開発した気象モデルは，地形の効果を取り入れている．地形を表現できるLESとして，直交座標系を採用したRaasch and Schroter（2001）のモデルや，地形に沿った座標系を用いたChow et al.（2006）のモデルなどがある．後者の座標系の場合，急峻な地形に対しては座標変換誤差が大きくなることが指摘されていることから，池田らの開発したLESモデルには一般曲線座標系を導入している．このモデルは，筑波大学におけるスーパーコンピュータT2k-Tsukuba上で開発されてきた．本研究では，飯塚らが開発したCFDモデル版LESコード[7)]をベースに筑波大学で改良した気象モデル版LES並列コードをGPUに対応させる．研究で扱うLESの数値計算アルゴリズムはSMAC法で，移流項にニ次精度Adams-Bashforth法，拡散項にCrank-Nicolson法を用いている．ポアソン方程式はBi-CGStab法で解いている．本研究で扱うLES計算における処理の流れを図1に

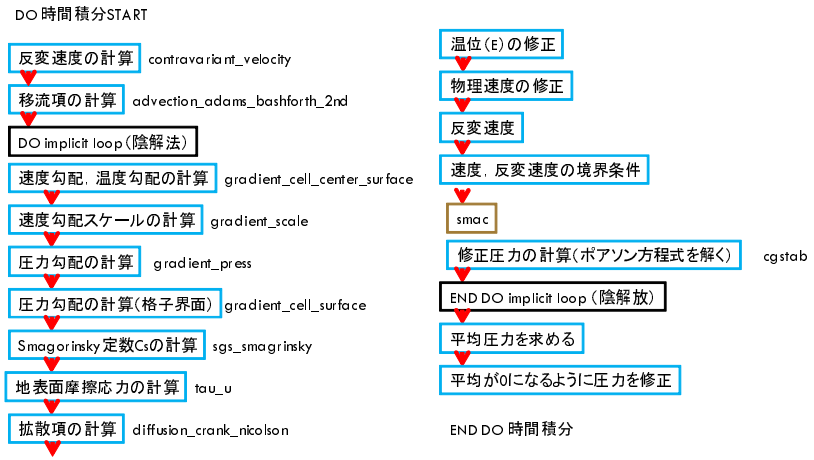


図 1 本研究で用いる LES コードの本体部分の流れ

示す．

4. GPU

GPU は，本来画像処理のための補助演算装置である．そのピーク演算性能は CPU の性能をはるかに上回り，近年急激に向上していることから，GPU の演算資源を画像処理以外の目的に応用する技術である GPGPU が数値シミュレーションなど幅広い分野で利用されている．代表的な GPU である NVIDIA 社の CUDA アーキテクチャでは，SM（Streaming Multiprocessor）と呼ばれるマルチプロセッサが複数並んだ構成をとっている[8]．この場合，一つの SM には SP（Streaming Processor）と呼ばれるコアが 8 個とシェアード・メモリと呼ばれるデータ共有のための高速なオンチップメモリを持っている．一方，GPU の全体メモリとして大容量のグローバルメモリがあるが，チップ外のメモリのため，データアクセスはシェアード・メモリに比べ低速である．本研究で使用する新世代 CUDA アーキテクチャ“ Fermi ”では，一つの SM に SP が 32 コアでグローバルメモリのキャッシュである L1 キャッシュ，L2 キャッシュが搭載され倍精度演算性能とデータアクセス性能が大幅に向上している[9),10)]．

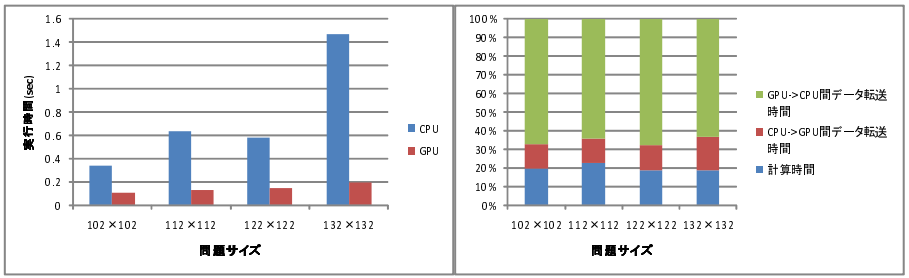
```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ks/call  Ks/call  name

 25.80  35022.13 35022.13    38233     0.00     0.00  __module_bicgstab_MOD_cgstab
 24.56  68357.84 33335.71   191165     0.00     0.00  __module_dynamics_MOD_gradient_cell_center_surface
 16.44  90682.76 22324.92        1    22.32   135.76  __module_run_MOD_run
 11.55 106368.40 15685.64    76466     0.00     0.00  __module_dynamics_MOD_gradient_cell_surface
  6.62 115356.29  8987.89    38233     0.00     0.00  __module_sgs_MOD_sgs_stress_vec
  2.98 119395.75  4039.46    38233     0.00     0.00  __module_smac_MOD_smac
  2.41 122667.01  3271.26    20000     0.00     0.00  __module_addition_inst_value_MOD_addition_inst_value
  2.23 125691.93  3024.93    38233     0.00     0.00  __module_sgs_MOD_sgs_stress_sca
  2.00 128406.13  2714.19    38233     0.00     0.00  __module_dynamics_MOD_tke_flux
  1.34 130228.95  1822.82   191165     0.00     0.00  __module_dynamics_MOD_diffusion_crank_nicolson
  0.86 131390.48  1161.53    38233     0.00     0.00  __module_dynamics_MOD_gradient_pres
  0.84 132535.98  1145.50   100000     0.00     0.00  __module_dynamics_MOD_advection_adams_bashforth_2nd
  0.81 133630.44  1094.46    20000     0.00     0.00  __module_dynamics_MOD_contravariant_velocity
  0.35 134103.40   472.96    38233     0.00     0.00  __module_dynamics_MOD_gradient_scale
```

図 2 気象モデル LES 計算のプロファイリング
（文中では module_..._MOD_は省略する）

5. GPU による高速化の検討

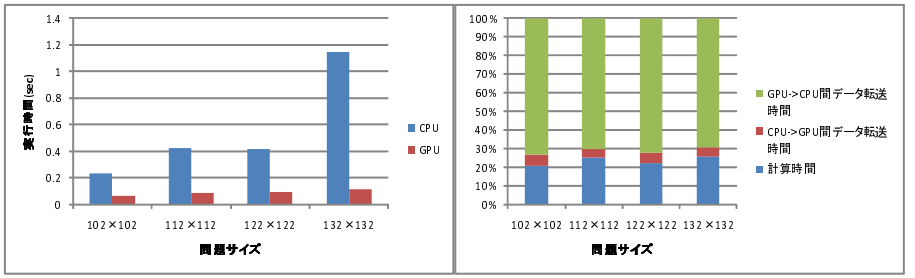
LES における流体計算は基本的にはいわゆるステンシル計算であり，領域分割法における内点計算と境界データの交換のコスト比率の観点から，GPU のような加速演算装置を使うメリットがあり，CPU に比べ大幅な高速化が期待できる．本研究で扱う気象モデルの高速化を行うため，まずコード内の各サブルーチンが占める実行時間のプロファイリングを行った．ここで，問題サイズ N=imax×jmax×kmax とし，imax，jmax，kmax を 102 としたプロファイリング結果が図 2 である．評価環境は Intel 社製 Xeon E5630 （Westmere-EP） 2.53GHz 4-core×2 ソケット、メインメモリ 24Gbyte である（ただし，本プロファイリングはそのうち 1 コアを用いた逐次版実行におけるものである）．また，本研究で扱う LES の本計算は時間刻みで計算が行われ，時間ステップ（max_time_step）を 20000 回と設定してプロファイリングを行った．図 2 から cgstab（Bi-CGStab 法でポアソン方程式を解くサブルーチン），addition_inst_value （時間平均量を求めるため，瞬時値を加算するサブルーチン）を除いたサブルーチン群において，全時間の 70% が消費されていることがわかった．また，プロファイリングの結果から cgstab と gradient_cell_center_surface の全実行時間に占める割合がほぼ等しいことがわかる．しかし，cgstab は他の関数と違い，ステンシル計算ではないことから並列 GPU 化が難しい．一方，gradient_cell_center_surface はステンシル計算であり，比較的実装が行いやすい．また，gradient_cell_center_surface 及び

gradient_cell_surface は非常に似た処理を行うルーチンであり，片方の GPU 化によってもう一方も比較的簡単に GPU 化可能である．gradient_cell_surface はプロファイリングから全実行時間の割合の中で上位 4 番目と大きな割合を占めている．以上の理由から bicgstab より gradient_cell_center_surface と gradient_cell_surface の GPU 化を優先し実装を行った．図 3(a) と図 4(a) は，gradient_cell_center_surface，gradient_cell_surface の CPU 及び GPU による実行時間の比較である．縦軸に実行時間をとり，単位は sec（秒）である．また，GPU での処理時間は CPU と GPU のデータ転送時間（入出力）を含む．横軸の問題サイズ N は先ほどの説明と同様である．図 3(a) と図 4(a) から GPU で計算することによって処理時間の短縮を実現でき，データ転送のオーバヘッドを加えたとしても GPU の導入が非常に有効であることがわかる．また，GPU の実際の計算とデータ転送オーバヘッドについて検証を行った．図 3(b) と図 4(b) は問題サイズを変化させた場合の各ルーチンの実行時間に占める計算時間と通信時間の割合を示す．この結果から，GPU における速度向上が大きいとはいえ，GPU から CPU への演算結果データの転送時間がこれらのルーチンの処理時間において大きな割合を占めいていることがわかる．本プログラムでは，これらのルーチンは非常に頻繁に呼び出され，その度に CPU・GPU 間でのデータ転送が長時間発生し，大きなオーバヘッドになっている．さらなる高速化を行うためには，GPU から CPU へのデータ転送時間を削減する必要がある．そこで，GPU 化を行った gradient_cell_center_surface と gradient_cell_surface のデータを GPU 上に常時置きっぱなしにし，それらのデータを利用する，これまで CPU 上で実行されていたルーチンを適宜 GPU 化する．これにより，全体として CPU・GPU 間のデータ移動を減らすことができる．



(a) 問題サイズの変更による処理時間　　(b) GPU の各実行の割合

図 3 gradient_cell_center_surface の GPU 化



(a) 問題サイズの変更による処理時間　　(b) GPU の各実行の割合

図 4 gradient_cell_surface の GPU 化

6. LES 気象モデルの GPU 化

6.1 GPU への実装

NVIDIA 社製 GPU である Tesla M2050（Fermi アーキテクチャ）を対象に，GPU コンピューティング用の統合開発環境である CUDA を用いて GPU コードの開発を行った．LES 気象モデルの本体部分の処理は run 関数である．プロファイリング結果にあった関数は全て run 関数から呼び出されている．GPU 化における主な流れを図 5 に示す．本計算

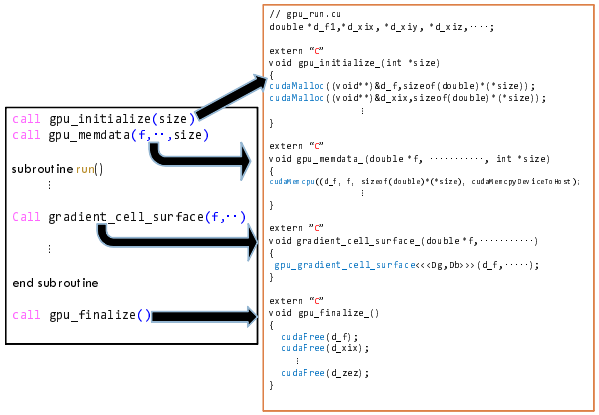


図 5 GPU の呼び出し

である run 関数に入る前に GPU で計算する必要なデータを global メモリ上に確保する．(gpu_initialize)．その後，先ほど GPU 上に確保した global メモリ上に計算に必要な初期値データを転送する．これは，gpu_memdata で行っている．次に run 関数から，CPU 上で処理を行う関数を GPU 向けに対応させた各カーネル関数を呼び GPU 上で計算処理を行う．最終的に run 関数が終了し，GPU 上に確保した global メモリを解放する．(gpu_finalize)．gpu におけるメモリ確保，データ転送，メモリ解放はそれぞれ 1 回だけの処理である．これが GPU の一連の処理の流れである．

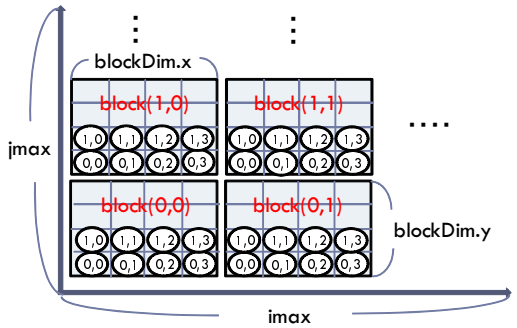


図 6 ステンシル計算の CUDA 化

6.2 GPU を用いたステンシル計算

気象 LES プログラムは，三次元配列のデータを扱う．ここで，各 i，j，k 方向の大きさを imax，jmax，kmax とするとデータサイズ N=imax×jmax×kmax なる．本研究では CUDA 化に当たって i 方向と j 方向のインデックスをブロック ID とスレッド ID を利用して管理してる．また GPU 上では，元の三次元配列を一次元配列として扱っている．GPU の実行において，i，j 方向の各格子に対しスレッド一つが担当し独立に処理を行うことで GPU 上で自然なスレッド並列化を行い高速化している．ブロック ID は図 6 の四角の枠の中に赤で書かれたものでスレッド ID は丸の中に書かれた二次元座標の番号に相当する．このようにブロック ID とスレッド ID を使用し，二次元空間の各格子に一つのスレッドを割り当てている．実際の Fortran で書かれたステンシル計算部分を図 7 に示す．先ほど説明した，i 方向と j 方向のインデックスをブロック ID とスレッド ID に利用した CUDA プログラムは図 8 に示す．CUDA プログラムでは三次元配列を一次元配列として扱っているため，インデックス ijk を計算し，インデックス ijk より離れた絶対値分を線形変換して index を求めることによって各データにアクセスする．

```
do k =2, kmax-1
  do j = 2, jmax-1
    do i = 2, imax-1
     fx1(i,j,k) = ( xix(i+1,j,k)*f(i+1,j,k) - xix(i,j,k)*f(i,j,k) &
                  + ( etx(i+1,j+1,k)*f(i+1,j+1,k)                 &
                    - etx(i+1,j-1,k)*f(i+1,j-1,k)                 &
                    + etx(i  ,j+1,k)*f(i  ,j+1,k)                 &
                    - etx(i  ,j-1,k)*f(i  ,j-1,k) )*0.25d0        &
                  + ( zex(i+1,j,k+1)*f(i+1,j,k+1)                 &
                    - zex(i+1,j,k-1)*f(i+1,j,k-1)                 &
                    + zex(i  ,j,k+1)*f(i  ,j,k+1)                 &
                    - zex(i  ,j,k-1)*f(i  ,j,k-1) )*0.25d0        &
                  )*hjac1(i,j,k)
    enddo
  enddo
enddo
```

図 7 ステンシル計算部分のオリジナル Fortran

表 1 評価環境

CPU	Intel Xeon E5630 2.53GHz 4cores×2
RAM	DDR3 SDRAM 1066MHz 4GB×6 GDDR5 SDRAM 1.55GHz 3GB (ECC on)
GPU	NVIDIA Tesla M2050 1.15GHz
OS	CentOS Linux release 6.0 (Final)
Compiler	GNU Fortran (GCC) 4.4.4 nvcc 4.0 (-arch sm_20) for GPU code

7. 性能評価

これまで述べた GPU 化による速度向上の評価を行う．評価環境を表 1 に示す．

プロファイリング結果にある cgstab，addition_inst_value 以外のサブルーチンと run の一部の処理に関して GPU 対応して実行した場合と，対応する同等の処理を CPU の単一コアで実行した場合の性能比較を**図 9** に示す．Tesla M2050 では，共有メモリ 16KB/L1

```
1   int ijk;
2   int i= blockDim.x*blockIdx.x + threadIdx.x + 1;
3   int j= blockDim.y*blockIdx.y + threadIdx.y + 1;
4
5   for( int k = 1 ; k < kmax-1; k++ ){
6     ijk = i + j*imax + k*imax*jmax;
7
8     d_fx1[ijk] = ( d_xix[ijk + 1]*d_f[ijk + 1] - d_xix[ijk]*d_f[ijk]
9              + ( d_etx[ijk + imax + 1]*d_f[ijk + imax + 1]
10               - d_etx[ijk - imax + 1]*d_f[ijk - imax + 1]
11               + d_etx[ijk + imax]*d_f[ijk + imax]
12               - d_etx[ijk - imax]*d_f[ijk - imax] )*0.25
13             + ( d_zex[ijk + imax*jmax + 1]*d_f[ijk + imax*jmax + 1]
14               - d_zex[ijk - imax*jmax + 1]*d_f[ijk - imax*jmax + 1]
15               + d_zex[ijk + imax*jmax]*d_f[ijk + imax*jmax]
16               - d_zex[ijk - imax*jmax]*d_f[ijk - imax*jmax] )*0.25
17             )*d_hjac1[ijk];
18  }
```

図 8 図 7 に対応する CUDA

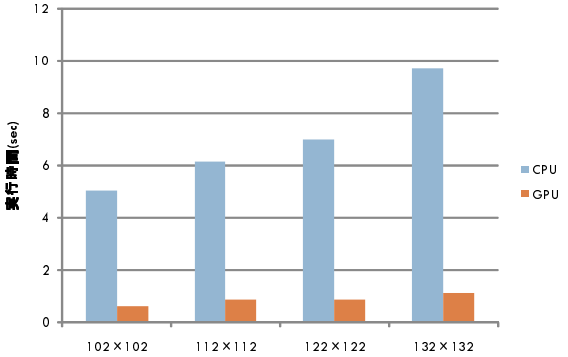


図 9 I、J 方向における問題サイズに対する処理時間

キャッシュ48KB，または共有メモリ 48KB/L1 キャッシュ16KB の構成が可能である．本研究では，前者の構成で性能評価を行った．縦軸は実行時間，横軸は問題サイズを表す．問題サイズ N は imax×jmax×kmax とし，ここでは，kmax=102 と固定し imax と jmax のサイズを変化させた場合の実行時間の変化を示している．なお，GPU の global memory の容量が 3GB であるため，GPU 上で実行できる imax，jmax の問題サイズは 132 までに制限される．図 9 より，全ての問題サイズにおいて GPU の速度が CPU を大幅に上回ることが確認できた．問題サイズ（imax，jmax）が 102 の場合と 132 の場合で，それぞれ 7.9 倍，8.4 倍の速度向上が達成された．これは，GPU の倍精度演算性能が大幅に向上したこと，また本研究で扱った LES モデルはデータ参照が多いステンシル計算であるため，GPU の高いメモリバンド幅が有効であり，CPU に比べ処理時間を大幅に短くすることができたためと考えられる．

8. まとめと今後の課題

複雑地形を取り入れた気象モデルを対象とした LES 計算において，計算負荷の高い関数を GPU に対応させ，計算時間の短縮を実現した．今回評価したのは GPU 化が完了した部分に対する実行時間のみである．最大 8.4 倍の向上が得られたが，CPU による実行のプロファイル結果から推測すると，これらの処理が全実行時間に占める割合は元々70%程度であることから，これらの部分に対してこれ以上の高速化を行っても，全実行時間における速度向上は 3 倍程度で頭打ちになると予想される（アムダール則による）．従って，今回 GPU 化の対象外とした処理についても GPU 化を進めていく必要がある．

現在は単一ノード，単一 GPU のみを対象とした GPU 化しか完了しておらず，対象問題サイズが GPU の global memory 容量で制限されてしまっている．本来，我々が目指しているのは解像度の高い大規模 LES 処理であり，MPI（さらに必要であれば OpenMP）を用いた並列化が必須である．GPU 間及びノード間におけるデータ通信がボトルネックとなる可能性があるが，計算の基本部分がステンシル計算であることから，境界点のデータ交換のコストは比較的小さく，並列化は十分に行えると考えられる．大規模 GPU クラスタにおける実装と評価を行っていくのが今後の課題である．

参 考 文 献

1) 額田 彰 ：CUDA による高速フーリエ変換，Vol 20， No.2，pp.37-43．応用数理学会．Jun．2010.
2) 濱田 剛，似鳥啓吾，青木 尊之：TSUBAME GPU クラスターを用いた重力多体シミュレーションの性能評価，計算工学講演会論文集（日本計算工学会）．May．2009.
3) 小野寺 直幸，青木 尊之，小林 宏充：GPU によるラージエディ・シミュレーションの高速化，流体力学会年会 2010，日本流体力学会．Dec．2010.
4) 下川辺 隆史，青木 尊之，石田 純一，河野 耕平，室井 ちあし：メソスケール気象モ

Vol.2011-HPC-131 No.2
2011/10/6

デル ASUCA の TSUBAME2.0 での実行，日本流体力学会 第 24 回数値流体シンポジウム講演予稿集．Dec．2010.
5) 池田 亮作，日下 博幸，飯塚悟，朴 泰祐：一般曲線座標系による並列 LES モデルの開発，日本気象学会 2011 年度春季大会講演予稿集．May．2011.
6) Ryosaku Ikeda，Hiroyuki Kusaka，satoru Iizuka，Taisuke Boku：Development of Local Meteorological Model based on CFD，5th International symposium on wind effects on buildings and urban enviroment（ISWE5）．Mar．2011.
7) Iizuka S, Kondo H： Large-eddy simulations of turbulent flow over complex terrain using modified static eddy viscosity models， Atmospheric Environment, 40, pp.925-935．Feb．2006.
8) NVIDIA Corporation:CUDA ZONE, http://www.nvidia.com/object/cuda_home.html
9) Peter Glaskowsky NVIDIA ’s Fermi : The First Complete GPU Computing Architecture
10) Dave Patterson The Top 10 Innovations in the New NVIDIA Fermi Architecture. and the Top 3 Next Challenges