J. Plasma Fusion Res. Vol.90, No.12 (2014) 755-763

解説

GPUコンピューティングによる大規模シミュレーション

Large-Scale Numerical Simulations Based on GPU Computing

青木尊之
AOKI Takayuki
東京工業大学　学術国際情報センター

（原稿受付：2014年7月16日）

GPUは高い演算性能とメモリアクセス性能をもつアクセラレータとして多くのスパコンに使われている．さらに低消費電力であり利点が多いが，既存のプログラムを書き換える必要がある．内部に2000個を超える演算ユニットをもつGPUのアーキテクチャの説明，並列計算を前提とするCUDAプログラミングの概略を述べる．また，通常のプログラムに指示行を追加するだけでGPUを利用できるようにするOpenACCについても簡単に触れる．著者がGPUスパコンTSUBAMEを利用して行った大規模シミュレーションの例として，水平解像度500 mの気象計算，1 m格子を用いた10 km四方の東京都心部の超高解像度気流計算，半陰解法による気液二相流計算，3300億格子を用いたフェーズフィールド法による合金の樹枝状凝固計算，離散要素法による粉体シミュレーションについて解説する．最後にプラズマ・シミュレーションにGPUコンピューティングを適用する見通しについて示す．

Keywords:
GPU, supercomputer, TSUBAME, CUDA, weather prediction, turbulent LES, gas-liquid two-phase flow, phase-field model, dendritic solidification, DEM, dynamic load balance

1．GPUについて

1.1　GPUの汎用計算への適用

パソコンのグラフィクス・ボードやCPU，最近ではスマートフォンやタブレットの中にも画像処理プロセッサGPU（Graphics Processing Unit）が搭載されている．GPUは元々画像を表示するために開発され，その後，より高速に，美しく，高精細に画像を表示するために描画機能が飛躍的に向上してきた．そして，画像表示だけではなく汎用計算に使う「GPUコンピューティング」（GPGPU（General-Purpose computing on GPUs）と同義）の取り組みが2000年頃から始まった．GPUはグラフィクス処理を目的としているため，汎用計算に対する機能が限定されている反面，同世代のCPUに対して10倍以上高い演算処理能力をもつため，魅力的な演算デバイスとして次第に認識されるようになっていった．2006年にはNVIDIA社が自社のGPUに対し，GPUコンピューティングの統合開発環境としてCUDA（Compute Unified Device Architecture）をリリースした[1]．それまではCg（C for Graphics）言語やHLSL（High Level Shader Language）により画像処理の機能を汎用計算に置き換える必要があったが，CUDAの登場により通常のC言語の拡張としてプログラミングすることができるようになり，GPUを汎用計算のアクセラレータとして利用するGPUコンピューティングが一気に広まり始めた[2]．また，2009年にはCUDAのFORTRAN版もリリースされ，GPUの性能向上に合わせた頻繁なバージョン・アップとともに機能もますます向上している．

1.2　GPUスパコンとTop500ランキング

最新のGPUは単体で1.7 TFlops（倍精度演算），単精度演算では5.3 TFlopsを超えるようなピーク演算性能を持つ．また，スパコン専用に開発されたプロセッサと違い，GPUはパソコンやモバイルITの巨大な市場で展開する製品であるためにコストが低い．

2008年に東京工業大学・学術国際情報センターが680個のGPUをスパコンTSUBAME 1.2に導入し，世界から大きな注目を集めた．NVIDIAは2010年に倍精度浮動小数点演算性能の向上やECCメモリに対応したFermiコアのGPUをリリースし，GPUをスパコン分野で利用できる条件が整った．2010年11月のスパコンTop500のランキングでは，1位と3位に中国のGPUスパコンが入り，4位にはNVIDIA Tesla M2050を4224個搭載した東京工業大学・学術国際情報センターのスパコンTSUBAME 2.0（総合演算性能2.4 PFlops）がランクインし，GPUスパコンが上位を独占した（**図1**）．

TSUBAME 2.0の設置面積は335平方メートルと少なく，演算ノードは直前まで事務室だった2つの部屋に収まっている．2012年11月には，Oak Ridge国立研究所のOpteron CPUベースのスパコンJaguar（2009年11月～2010年6月まで世界1位）に18688個のKeplerコアのGPU

Global Scientific Information and Computing Center, Tokyo Institute of Technology, TOKYO 152-8550, Japan
e-mail: taoki@gsic.titech.ac.jp

図1　東京工業大学・学術国際情報センターの TSUBAME.

（NVIDIA Tesla K20X）を導入したTitanが世界1位となった．東京工業大学・学術国際情報センターでは，2013年9月に4224個のGPUを全てTesla K20Xに交換し，TSUBAME 2.5として総合演算性能を5.7 PFlopsに向上させている．

スパコンに導入されるGPUは通常のCPUのマザーボード上のPCI Expressバスに装着されるビデオカード（グラフィクス・カード）上に搭載され，アクセラレータと呼ばれることも多い．Intelは2012年にMICアーキテクチャのXeon Phi（コードネーム　Knight Corner）を市場にリリースしている．Xeon Phi 5120Pはピーク浮動小数点演算性能1.0 TFlops（倍精度演算）とメモリバンド幅320 GB/secのカタログ性能をもち，GPUとほぼ同等の性能を持つ．描画機能はもたないが，Xeon PhiもマザーボードのPCI Expressバスに装着して計算の高速化を目的とするアクセラレータであり，機能を制限した通常のCPUコアを数10個搭載したアーキテクチャとなっている．中国のTianhe-2はXeon Phiを約5万個搭載するスパコン（ピーク性能55 PFlops）で，Top500ランキングにおいて3期連続（1年半）で世界一になっている[3]．

1.3　スパコンの電力効率

世界トップクラスのスパコンがPFlopsを超えてきたあたりからエクサスケール（Exa-Flops）のスパコン開発の議論が始まった．当時のスパコンを1000倍にスケールすると，スパコンの消費電力は世界最大級の発電所の電力に匹敵してしまい，スパコンにおいても低消費電力化が最重要課題と認識されるようになった．理化学研究所の「京コンピュータ」の消費電力は13 MW，Tianhe-2の消費電力は約18 MWであり，今後も消費電力は20 MW程度が限界だと考えられていて，エクサスケールのスパコンを完成させるには，今後数年間の技術革新でプロセッサの消費電力だけでなく，冷却を含めたスパコン・システムの消費電力を20分の1に下げなければならない．

モバイルIT機器では搭載するCPUの低消費電力化が進められ，ダークシリコンと呼ばれる消費電力の制約からシリコンチップ上で電力を供給してオンにできないエリアの問題解決にGPUの利用が進められている．GPUは消費電力当たりの演算性能が高く，スパコンにおいてもアクセラレータとしての導入が進んでいる．もう一つのスパコンのランキングにGreen500[4]があり，Top500のランキング指標であるLinpackスコアを消費電力で割ったFlops/Wを指標としている．2014年6月のGreen500ランキングは，東京工業大学・学術国際情報センターの油浸冷却を取り入れた試作機TSUBAME-KFCが4.4 GFlops/Wで1位となり，TSUBAME2.5が2.9 GFlops/Wで8位，「京コンピュータ」は0.8 GFlops/で125位となっている．Green500のトップ10のスパコンは全てNVIDIA Tesla K20Xを導入したGPUスパコンであり，スパコンの省電力化にGPUが大きく貢献していることが分かる．

1.4　GPUのアーキテクチャ

GPUのアーキテクチャは汎用CPUと大きく異なる．NVIDIAのTesla K20Xを例に説明する（**図2**）．Keplerコアの GPUでは，192個の演算ユニット（CUDAコア）が1つのストリーミング・マルチプロセッサ（SMXと呼ばれている）を構成している．Tesla K20Xには14個のSMXがあり，合計で2688個のCUDAコアがある．1つのSMXの中には64 k個の32 bitレジスタファイルがあり，非常に高速なL1キャッシュと共有メモリ（合計で64 kB），sinやcosなどの数学関数を高速に計算するSFU（Super Function Unit）などがある．1つのSMX内の共有メモリは192個のCUDAコアからアクセスできるが，他のSMX内のCUDAコアからはアクセスできない．K20Xを搭載したグラフィクス・ボード上には6 GBのビデオメモリ（GDDR5）があり，GPUからビデオメモリへは250 GB/secというメインメモリに比べて数倍以上高速なメモリアクセスが可能である．ビデオメモリには2688個のCUDAコアから共有でメモリアクセスが可能である．通常はCPUからビデオメモリには直接アクセスすることができず，メインメモリとビデオメモリ間でPCI Expressバス（Gen2の帯域は8 GB/sec, Gen3では16 GB/sec）を介したデータ通信を行うことができる．GPUスパコンの各ノードには，2個程度のCPUとGPUを搭載したビデオカードが1～4枚程度搭載されていることが多く，さまざまなメモリが階層的構造になっていて，それぞれデータ転送の帯域が大きく異なる点が特徴的である．

GPUスパコンでは，千～万ノードがさまざまなトポロジーの高速なインターコネクションで接続されている．TSUBAME2.5では1ノードに2個のIntel Xeon X5670と3個のTesla K20Xが搭載され，1408ノードがQDR InfiniBandによりFat Tree型で接続されている．

2．GPUコンピューティング

2.1　軽量・超多スレッドの並列計算

前節で述べた通り，1つのGPUには2688個もの演算コアが搭載されており，それらを効率よく利用することによりGPUの高い計算性能を引き出すことができる．即ち，並列計算が必要になる．SMXの中の各CUDAコアはスレッドという単位で計算を実行するが，詳細には32個のCUDAコアが同一の演算命令を実行する必要があり，SIMD（Single Instruction Multiple Data）計算となる．例えば，メモリ上のデータの数値は違ってもよいが，32個のCUDAコアが一斉にメモリアクセスを行い，レジスタにデータを保存し，同一の演算を行う処理になる．SMXに対しては2048個までのスレッドを同時に実行命令することができ，投入されたスレッドをスケジューラが32個単位に分割し，メモリ

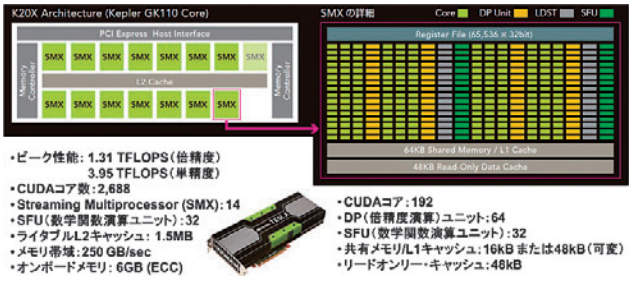

図 2　NVIDIA Kepler コア GPU のアーキテクチャ.

アクセスと演算処理を同時に実行させるなどの効率的な処理が行われる．GPU 全体に対しては，物理的に存在する2688個のコア数をはるかに上回る最大で1024×1024×64×1024個のスレッドを投入することができる．1つのスレッドの処理内容は1回のメモリアクセスと四則演算のような軽い処理でもよく，最低でも数万スレッド以上ないと高い実行性能は期待できない．スレッドの内容は条件分岐などを含んでもよく，1つのGPUに対しては同一のプログラムが動く SPMD（Single Program Multiple Data）となっている．

複数 GPU を使う場合は，ノード内ならば Open MP などを利用して複数 GPU を制御し，複数ノードになる場合は MPI ライブラリを用いてそれらが階層的な SPMD として実行される．

2.2　GPU を使うためのプログラミング

2.2.1　CUDA プログラミング

第1章でも少し述べたが，GPU を計算に用いるためには，殆どの場合は CUDA でプログラミングする必要がある．CUDA は C 言語または FORTRAN 言語に GPU コンピューティングのための言語拡張や API（Application Programming Interface）やコンパイラなどを含めた統合開発環境である．FORTRAN コンパイラのみ有償であるが，それ以外は NVIDIA から全て無償で提供され，Linux，Windows，Mac などのほとんどのプラットホームに対して準備されている．また，サンプル・コードや解説ドキュメントも多く，初心者に対しても敷居が低い[1]．

CUDA のプログラミングについて概略を**図 3** に説明する．全体では CPU での実行を記述するホストコードと GPU の実行を記述するデバイスコードから構成される．ホストコードには GPU のビデオメモリ上への配列確保やメインメモリとビデオメモリ間のデータ転送などの API と GPU の実行内容を記述したカーネル関数のコール等が記述される．デバイスコードにはGPUカーネル関数の内容が書かれ，その内容がスレッドとして各CUDAコアで実行される．

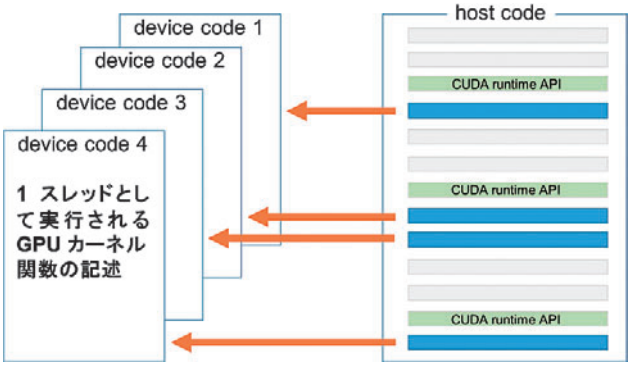

図 3　GPUで計算を実行するためのホストコードとデバイスコードの関係.

CPU を使ってメインメモリ上の配列 A と B に対して，n 個の要素を B から A にコピーする C 言語の**プログラム 1** は，

```
double *A, *B;
int    i;
for(i = 0; i< n; i++) {
    A[I] = B[i];
}
```

プログラム 1　CPU の逐次コード.

と簡単に書ける．一方，GPU のビデオメモリ上に確保した配列 A と B に対して，n 個の要素を B から A にコピーする GPU のカーネル関数に対する CUDA のプログラムは，

```
__global__ void array_copy(double *A, double *B)
{
    int i = blockDim.x*blockIdx.x + threadIdx.x;
    B[i] = A[i];
}
```

プログラム 2　GPU のカーネル関数コード.

となる．CUDA の C プログラムでも，CPU の**プログラム 1** と同じようにポインタや配列を使うことができる．**プログラム 2** の内容がスレッドとして 1 つの CUDA コアで実行される．ホストコードに次のようにカーネル関数の実行指令が記述される．

```
array_copy<<<n/256, 256>>>(A, B)        (1)
```

"<<<a, b>>>"の中の 2 つの引数がGPUに投入するスレッド数を階層的に指定している．第 2 引数は b 個のスレッドが 1 つのブロックを構成することを指定し，第 1 引数の a はブロック数を指定している．したがって，a×b 個のスレッドがGPUで実行されることになる．a，bが整数の場合は 1 次元的にスレッドを管理することになるが，2 次元，3 次元ベクトルとして巨大な総数のスレッドを指定することができる．ブロック内のスレッドは同一 SMX の中で処理されるが，どのブロックがどの SMX で処理されるか，ブロック中のどのスレッドがどのCUDAコアで実行されるかは指定できない．(1)の例では，カーネル関数コールにより 1 ブロック内に256個のスレッドを配置し，n/256 個のブロックが投入されるので総計 n/256×256＝n 個のスレッドが実行される．カーネル関数内ではスレッド毎に違う値が入るビルトイン変数を宣言なしで使うことができ

る．プログラム 2 の中で使われている blockDim.x にはブロック中のスレッドの数が入っており，(1)でカーネル関数コールを行った場合には blockDim.x＝256となる．ビルトイン変数 blockIdx.x にはそのスレッドが何番目のブロックの中で実行されるかの値が入り，0〜n/256－1 のどれかの整数になる．threadIdx.x にはそのスレッドがブロック内の何番目であるかが入り，0〜255の値になる．これらのビルトイン変数を利用し，n 個の配列要素に対して，**図 4** のような対応関係を規定することにより，投入される n 個のスレッドに対して，i＝blockDim.x*blockIdx.x＋threadIdx.x は 0〜n－1 の全て異なる値を取らせることができる．したがって，プログラム 2 のカーネル関数を(1)でコールすることにより，n 個のスレッドが全て異なる配列要素をコピーすることができ，プログラム 1 と同じ内容をビデオメモリに対して実行することができる．プログラム 1 はループにより逐次的に同じコピーを n 回繰り返しているが，プログラム 2 では，ある 1 要素をコピーするだけというスレッドを n 個投入していて，ループがなくなっている点が興味深い．スレッドの総数は CUDA コアより圧倒的に多いので 1 コアは複数のスレッドを実行するが，その実行順はスレッド・スケジューラにより決められ，プログラム側で指定することはできない．

CUDA は GPU の性能向上に合わせ，機能向上を含んだ頻繁なバージョン・アップが行われ，現時点でバージョン 6.0が正式にリリースされている．

NVIDIA の GPU だけでなく，AMD 社や Intel 社の GPU や IBM 社の Cell，DSP などに対して，同じプログラムが動作するような標準化（クロスプラットフォーム）として Open CL（Open Computing Language）の仕様が2009年に策定され，KhronosGroup により現在までにバージョン2.0がリリースされている．初期は NVIDIA，AMD，Apple，Intel 等の多くの企業が OpenCL コンソーシアムに参加し大きな期待がもたれたが，NVIDIA 社が自社の GPU に対する OpenCL へのサポートを更新しなくなったため，開発の勢いは鈍化している．

2.2.2 指示行挿入による GPU の利用（OpenACC）

2012年に NVIDIA 社，PGI 社，CAPS 社，Cray 社が協定し，CPU で実行する通常のソースコードに指示行を挿入するだけでGPUの機能を使えるようにするOpenACCの開発が始まった．ホスト（CPU）用コード，さまざまなアクセラレータ用コードを単一コードとして記述でき，メンテナンスが容易になるなどの利点がある．並列化したいループに kernels，loop ディレクティブを追加することにより，指

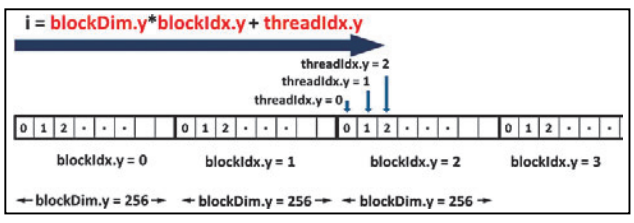

図 4　ビルトイン変数と配列要素の対応．

```
double *A, *B;
int    i;

#pragma acc kernels!
#pragma acc loop independent!

for(i = 0; i< n; i++) {
    A[I] = B[i];
}
```

プログラム 3　OpenACC 指示行の挿入．

定された領域がアクセラレータで実行されるカーネルへ変換され，GPU で実行される．**プログラム 3** では示していないが，data ディレクティブを挿入すると，GPU のビデオメモリ上への配列確保，メインメモリ－ビデオメモリ間のデータ転送が制御される．これらはコンパイラ側での処理であり，これまでにさまざまな不具合が解消されてきていて，現時点で PGI，CAPS，Cray からバージョン2.0の OpenACC 対応のコンパイラが有償で提供されている．

GPU コンピューティングといえば，現在のところ NVIDIA の GPU に対してピーク性能に近い性能まで引き出すことを目的とする CUDA と，既存のプログラムに対して移植のコストを抑えつつ，ある程度の高速化を期待する OpenACC の二つが中心といえる．

3．GPU スパコンによる大規模シミュレーション

3.1 複数 GPU を用いる計算の問題点

単一のグラフィクス・カードに搭載されるビデオメモリのサイズはせいぜい 6 GB（最新の Tesla K40 では 12 GB）であり，より大規模計算を行うには複数の GPU を使った計算が必要となる．格子系アプリケーションでは領域分割法が用いられ，分割された各領域を 1 個の GPU が計算する．分割された領域の境界近傍格子での計算は隣接領域の格子点にアクセスする必要があり，袖領域のデータ通信が発生する．ビデオメモリ上にあるデータの GPU 間通信は帯域の狭い PCI Express バスを介して通信しなければならずメモリアクセスと比較するとかなり遅い．さらに CPU 側のメモリを介して転送する必要があるため，大規模計算において GPU 間のデータ通信は大きなオーバーヘッドになる．最新の NVIDIA の GPU Direct RDMA を利用することで異なるノードの GPU との通信の手続きが簡単になり，通信速度も 2 倍近く早くなるが，依然として大きなオーバーヘッドがあることは変わりがない．高い実行性能を得るためには，分割された各領域において境界格子を先に計算し，その終了とともに GPU 間データ通信を開始する．それと同時に内側の格子の計算を並行して実行する「計算と通信のオーバーラップ（**図 5**）」により，GPU 間データ通信の時間を可能な限り隠蔽することが必要である．

3.2 500 m 水平解像度を用いた次世代気象計算

気象計算はスパコンを利用する大規模計算の代表的アプリケーションである．次期気象予報モデルとして気象庁が

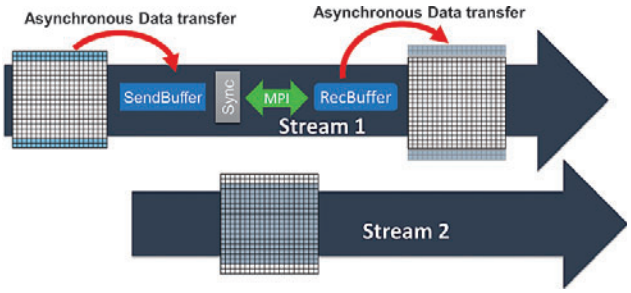

図5　計算と通信のオーバーラップ．

開発している ASUCA に対し，物理過程と力学過程の全てのモジュールのGPU化を行った [5]．力学過程は流体計算そのものであり，双曲型の圧縮性流体方程式を陽的に時間積分しているため，GPU コンピューティングに適している．しかし，単純な CUDA による GPU コンピューティングへのポーティングではなく，さまざまな高速化アルゴリズムおよび前節で述べた計算と通信のオーバーラップも導入した．**図 6** は，気象庁で現在の気象予報で実際に使われているメソモデル（MSM）のデータを初期条件・境界条件とし，TSUBAME 2.0 の 437 GPU を用いて GPU 版の ASUCA により4792×4696×48（水平方向に 500 m×500 m の格子）の格子で，初期時刻2009年10月 6 日 15 UTC から計算した 9 時間後の雲の分布を可視化している．気象庁が現在行っている気象予報は水平 5 km および 2 km 格子を用いているが，その 1/10〜1/4 の格子サイズでの計算が既に TSUBAME 2.0 によりほぼ実時間で可能であることが示され，世界に先駆けGPUスパコンの実用的な運用アプリケーションに対する有用性が実証された．最近では，計算機アーキテクチャの急速な進歩に対するソフトウェアの保守性を確保するために，DSL（Domain Specific Language）やフレームワークを導入する研究が盛んに行われている [6]．

3.3　1 m 格子による東京都心部の 10 km 四方の気流計算

都市は高層ビルが立ち並び複雑な構造をしており，詳細な気流を解析するためには高解像度格子による大規模気流シミュレーションが必要となる．数値計算手法は単純なアルゴリズムで大規模計算に適した D3Q19 モデルの格子ボルツマン法 [7] を用いた．都市の気流はレイノルズ数が100万を超えるような乱流状態になるため，ラージエディ・シミュレーション（LES）[8] のモデルを導入する必要がある．現在よく使われている動的スマゴリンスキー・モデル [9] では，モデル定数を決定するために各格子点で広領域の平均操作が必要になり，大規模計算には極めて不向きである．本研究では，モデル定数を局所的に決定できるコヒーレント構造スマゴリンスキー・モデル [10] を格子ボルツマン法に導入することに成功し，大規模な気流の LES 計算を初めて可能にした．実際の建物データに基づき計算対象のエリアを領域分割し，TSUBAME2.0/2.5 の GPU を用いて計算を行った．CUDA を用いてコードを実装し，ここでも3.1節の計算と通信のオーバーラップを導入し，実行性能をオーバーラップしない場合と比較して30%以上向上させることができた [11]．

格子ボルツマン法はメモリ律速の計算手法であるが，TSUBAME2.0 の全ノードを用いた計算ではピーク性能の15%となる 600 TFlops,TSUBAME2.5 では 1.14 PFlops の実行性能が得られた．10,080×10,240×512格子に対して 4,032個の GPU を用い，新宿や皇居を含む 10 km 四方のエリアを 1 m 格子間隔で計算した．**図 7** は風速分布を多数の粒子を用いて可視化している．高層ビル背後の発達した渦によるビル風や幹線道路に沿って流れる「風の道」，台風の際の被害などが飛躍的な精度で予測できるようになる．さらに，排ガス，事故やテロによる有毒ガスなどの汚染物質の拡散も詳細に予測可能となる．

3.4　気液二相流シミュレーション

気体と液体が激しく混じり合うような流れ（気液二相流）は数値流体力学の中でもチャレンジングな研究テーマとして知られている．計算時間もかかるため，予てから GPU 計算の導入が望まれていた．GPU コンピューティングが当初重力多体問題での粒子計算で成功したことが影響し，自由表面を含む流れの計算に対しても，SPH（Smoothed Particle Hydrodynamics）法などの粒子法が使われてきた．しかし，従来の粒子法はランダムなメモリアクセス，演算量，計算精度の三つの観点から効率が悪い．TSUBAME2.0 を用いて，VOF（Volume of Fluid）法をベースとした格子法による気液二相流の計算をマルチ GPU で実行し，高い実行性能を達成することができた．非圧縮性 Navior-Stokes 方程式を半陰解法により時間積分し，移流項には 5 次 WENO スキームを使い高精度計算を行った．気体と液体の界面を記述するためにレベルセット法とカップルした VOF 法を導入し，毎ステップ再初期化を行っている．

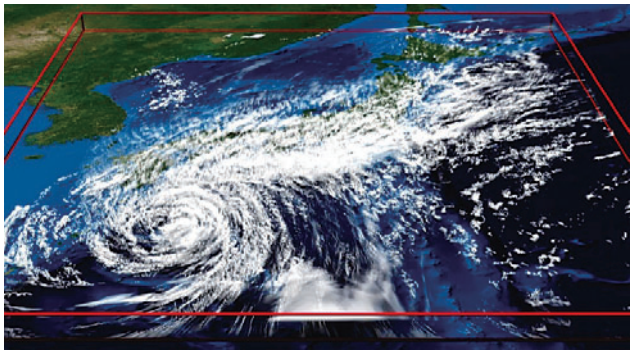

図6　水平解像度 500 m 格子を用いて次世代メソスケール気象モデル ASUCA で計算した雲分布．

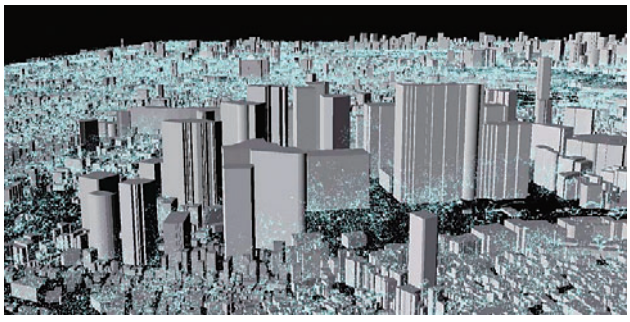

図7　1 m 格子を用いて 10 km 四方の範囲の都市部の気流計算（粒子は気流の可視化のために使用）．

一番計算時間のかかるPoisson方程式の解法は，気体と液体の密度比が非ゼロ要素に含まれ，みずほ情報総研と共同研究しているBi-CGSTAB法[12]にマルチグリッド前処理（**図8**）を導入したライブラリを用いている．Lはラプラシアンの離散オペレータ，fは従属変数（ここでは圧力），Sは右辺のソース項（ここでは速度の発散）を示し，上付き添え字は格子の粗度を表している．粗度が0は解くべき格子解像度であり，1増える毎に格子間隔は2倍，格子点数は1/2になる．各段で残差をソース項として修正ベクトルを求めるためのスムーザとしての不完全ILU分解を行っている．GPUでは格子間で順序依存性がある計算はできないため，ここではRed and Black法を用いている．また，細かい格子から粗い格子への制限補間（Restriction）と粗い格子から細かい格子への延長補間（Prolongation）は単純な平均化処理を行っている．CPUでの計算と同じように**図8**のVサイクルをGPUでも実装し効率的な収束計算を行っている．

計算では表面張力や壁との接触角も考慮し，単一気泡の上昇を実験と詳細に比較し十分な一致が得られ，**図9**に示す濡れた床へ侵入するダムブレーム問題では，早期に砕波が起こり水と空気が激しく混じり合い，壁への衝撃圧が低下する様子が再現されている．九州大学（応用力学研究所胡　長洪教授）との共同研究において，壁での衝撃圧の時間履歴が計算と実験で定量的によく一致することを確認した．

3.5　フェーズフィールド法による合金の樹枝状凝固シミュレーション

金属材料の組織構造は凝固過程のミクロな結晶構造で決定される．一方，現実に機械的強度を判定するには，数mmのマクロなスケールでの解析が必要となる．ミクロな凝固ダイナミクスを解明するために，非平衡統計力学から導出されるフェーズフィールド法[13]が近年注目されている．導出される方程式は時間空間の偏微分方程式になっていて，有限差分法や有限要素法などの格子法で解かれることが多い．フェーズフィールド法は複雑な非線形項を多く含むための1格子点あたりの演算量が多く，安定性の観点から時間ステップを小さく取らなければならない．CPUで計算すると時間がかかり過ぎるため，これまでは主に2次元計算による解析が行われてきた．大規模並列計算では，3次元領域分割を行うことにより領域間のデータ通信量を最小にすることができるが，GPU計算の場合はx方向で分割した場合のx–y断面のメモリアドレスが不連続となり，メモリアクセス性能の低下のオーバーヘッドの方が通信量の低減より大きくなる．ここでは，y方向，z方向に分割する2次元の領域分割法を用い，それぞれの分割領域の計算を各GPUへ割り当てる．隣接するGPU間での境界領域のデータ交換はホスト側のメモリを経由し，次の3段階で構成される．(1)CUDAランタイムライブラリによるGPUからCPUへの転送，(2)MPIライブラリによるノード間のデータ転送，(3)CUDAランタイムライブラリによるCPUからGPUへの転送を行う．ここでは隣接ノード間でのデータ通信と計算に対し，以下の3つの方法を実装し比較を行った．

(a) **GPU-only method**：比較のために計算と通信のオーバーラップを行わず，上記の通信の3ステップを順に行う．

(b) **Hybrid-YZ method**：1つのGPUが担当するサブ領域を，y方向の境界領域，z方向の境界領域，残りの中心領域に分割する．y，z方向の境界領域をCPUで計算し，中心領域をGPUで計算する．先行研究[14]では境界も中心領域もGPUで計算したが，ここでは境界領域をCPUで計算しGPU関数を分割することなく通信を隠蔽する．ただし，全計算領域サイズによってCPUによる通信と境界領域の計算にかかる時間が中心領域のGPU計算時間よりも長いことがあり，CPUがボトルネックとなることが考えられる．

(c) **Hybrid-Y method**：(b)と同じようにCPU計算も利用するが，z方向の境界領域（x–y断面）はカーネル関数を分割してGPUで計算する．境界領域を担当するGPUカーネル関数は，グローバルメモリの連続アクセスであるため計算効率がよく，また通信バッファへデータを詰め替える必要もない（**図10**）．

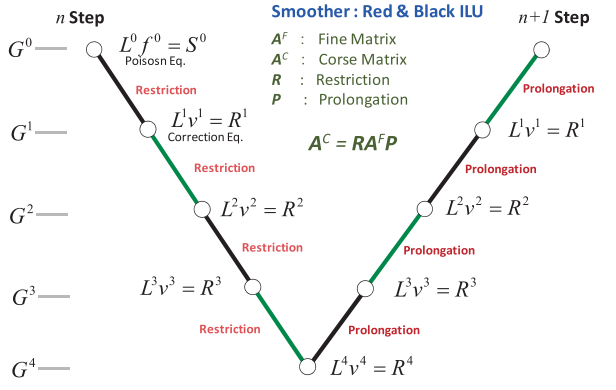

図8　マルチグリッド前処理のVサイクル．

図9　VOF法ベースの格子法による気液二相流シミュレーション．

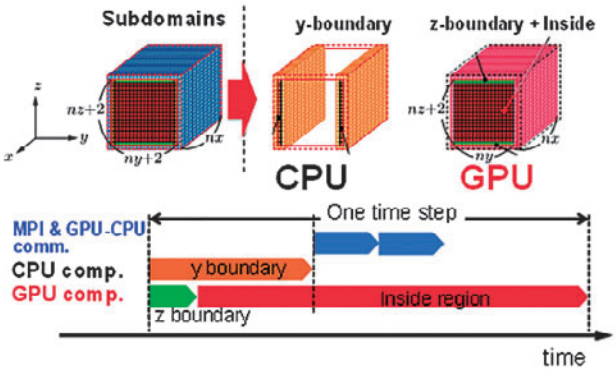

図10　Hybrid-Y methodのダイアグラム．

TSUBAME の各ノードには3 GPU と2 CPU sockets (12CPU コア) が搭載されている．このため CPU による境界領域の計算に1 GPU あたり4CPU コアを割り当て，OpenMP を用いた並列計算を行っている．TSUBAME2.0 の複数GPU(M2050)を用い(a)GPU-only method，(b) Hybrid-YZ method，(c)Hybrid-Y method の3つの実装の実行性能について述べる．フェーズフィールド法により Al-Si 合金の凝固成長の計算を行った．GPU コードの実行性能の評価において，GPU コードの浮動小数点演算数を直接測定することは困難である．ここでは全く同じ結果を出す CPU コードに対して PAPI (Performance API) を用いて実測した浮動小数点演算数を元に GPU の実行時間から実行性能を評価している．

図11の強スケーリングは，問題を固定して GPU 数を増やして行くときの実行性能の向上を示していて，前述の(a)，(b)，(c)の3つの実装の違いによる特徴が表れている．全体の計算領域を512^3，1024^3，2048^3とした．3.1節のように計算と通信のオーバーラップを導入した(b)Hybrid-YZ method，(c)Hybrid-Y method では GPU 数が少ないときには通信を隠蔽できている．GPU 数の増加とともに GPU の計算時間が短くなるため，ある段階で計算時間より通信時間の方が長くなり，もはや通信を隠ぺいすることができなくなる．(c)Hybrid-Y method では，GPU 数を増やした時に CPU の計算時間が隠ぺいの足を引っ張る部分が大幅に改善され，広い範囲の GPU 数で最適化を導入していない(a)GPU-only method と比較して実行性能が大幅に改善されている．

弱スケーリング (**図12**) は，1GPU あたりで実行する問題規模を一定にして GPU 数を増やしたときの性能評価である．この測定では，1GPU あたりの計算格子サイズを単精度計算では4096×128×128，倍精度計算では4096×128×64とした．弱スケーリングの結果が得られ，(c)Hybrid-Y method では，4000 GPU を利用して4096×6500×10400計算格子に対して行った計算で，2.0 PFlops (GPU:1.975 PFlops，CPU:0.025 PFlops) という極めて高い実行性能を達成した．この成果により2011年に ACM ゴードンベル賞を受賞している[15]．**図13**は，Al-Si 合金の凝固成長シミュレーションのスナップショットを示している．

3.6 動的負荷分散を導入した粉体シミュレーション

重力やクーロン力のような長距離力による粒子間相互作用と違い，砂や粉などの粉体現象は粒子間の接触による反発や摩擦力のモデル (離散要素法：Discrete Element Method (DEM)) でシミュレーションされる．実現象と同じ程度の粒子サイズで計算することにより，疎視化モデルでは表現できない現象を計算することが可能になるが，計算規模が膨大になり単一 GPU ではメモリが足りず複数 GPU による計算が必要になる．

重力多体問題などと異なり DEM 計算はメモリ律速であり，粒子の位置や速度などの従属変数は GPU のビデオメモリ上に乗せておく必要がある．また，接触による相互作用であるため，粒子分布を空間領域で分割し，分割された領域内の粒子数を一定にすることで GPU 間の計算負荷を均一にして並列計算の効率を上げることができる．しかし，粒子の空間分布は時間とともに大きく変化し，初期に均一な負荷であったとしても時間とともに負荷バランスは大きく崩れる．そこで，**図14**のように領域境界を移動させるスライスグリッド法を導入することにより，常に領域内の粒子数をほぼ一定に保つ動的負荷分散を行うことができる[16]．

CAD データで表現される物体形状を符号付距離関数の

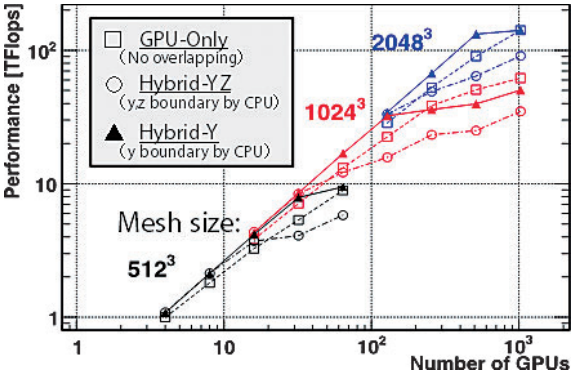

図11　樹枝状凝固シミュレーションの実行性能 (強スケーリング)．

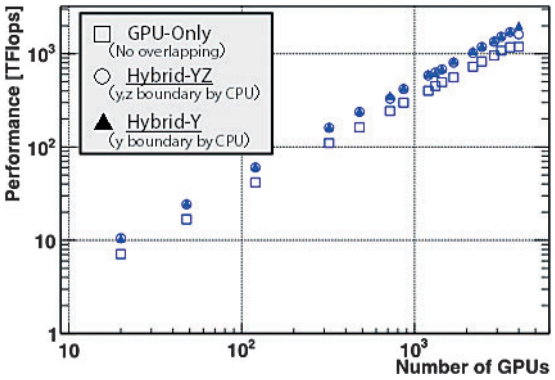

図12　樹枝状凝固シミュレーションの弱スケーリング．

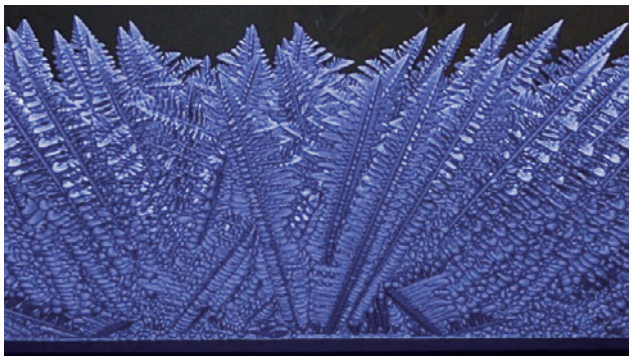

図13　Ai-Si 合金の樹枝状凝固過程のシミュレーション．

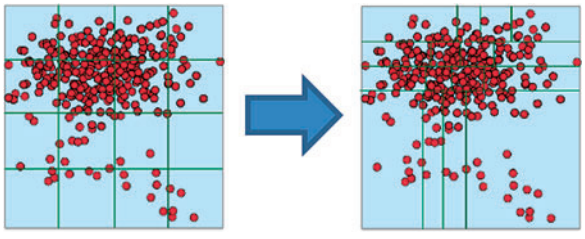

図14　スライスグリッド法による粒子計算の動的負荷分散．

等値面で近似することによりポリゴンのエッジの特異点を排除し，物体との接触判定の計算負荷を大幅に低減することができる．

TSUBAME2.5 の 64GPU を用いて粉体の撹拌シミュレーションを行った．**図15**は500万個の粒子を用いているが，256 GPU により 1 億3000万個の粒子計算に対しても効率よく計算が行われている．

4．Plasma シミュレーションへの GPU コンピューティングの適用

4.1 MHD シミュレーション

前節で示した気象計算の力学過程は圧縮性流体計算を行うために空間を格子で離散して Navior-Stokes 方程式を解くもので，MHD 方程式の離散化とほぼ同じである．MHD シミュレーションの陽的時間積分は容易に GPU コンピューティングを行うことができる．Maxwell 方程式の電磁波モードについては陽的時間積分であるので問題ないが，静電場をポアソン方程式で解く場合は楕円型方程式になり，大規模疎行列計算が必要になる．3.4節で説明した半陰解法のための圧力のポアソン方程式は静電場の計算と全く同じであり，拡散モデルによる輻射輸送は放物型方程式であり楕円型方程式の計算に準じ，疎行列に対するクリロフ部分空間法の反復計算や有効な前処理を同じように GPU 計算で使うことができる．Vlasov-Maxwell 方程式や Fokker-Planck 方程式を解く Plasma Kinetics シミュレーションについても，ほとんどの部分が双曲型方程式であり，GPU コンピューティングを行う上での問題はない．電離や励起などの原子過程は化学反応とほとんど同じ密行列計算になり GPU の得意とする計算になり，CUDA の BLAS や Lapack ライブラリが利用できる．

筑波大学のグループは日本原子力研究開発機構のプラズマ乱流を解析するための 5 次元 Vlasov コード GT5D の GPU 化を行っている[17]．また，太陽風と惑星の相互作用に対する大規模な MHD シミュレーション（**図16**）も行われている[18]．

4.2 PIC シミュレーション

PIC（Particle-in-Cell）シミュレーションも粒子計算に GPU を適用することは容易である．格子点上で電磁場を計算するのは負荷が低く，大規模計算を行うには3.6節のような何らかの粒子の空間分布に対する動的負荷分散を導入する必要がある．また，ベクトル型計算機でも工夫が必要になった粒子分布から格子点上の電荷密度や電流密度を求める gathering 処理は足し合わせがどうしても逐次処理になり，並列処理を行う有効なアルゴリズムがない．CUDA には排他的に他のスレッドの書き込みを止め，逐次処理を可能にする ATOMIC 処理が可能であるが，通常は非常に時間のかかる処理となり高速計算には不向きである．しかし，荷電粒子の gathering は Cell 内の局所的な足し合わせであり，CUDA のブロック内で完結させることができるため，足し合わせを SMX 内の高速な共有メモリ上で ATOMIC 処理を用いて行うことにより高速に計算を行うことができる[19, 20]．

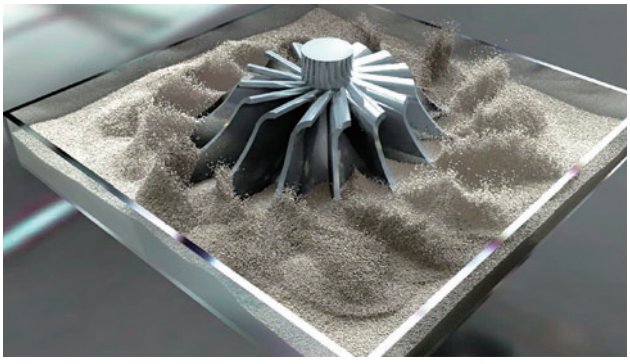

図15　離散要素法による粉体の撹拌シミュレーション．

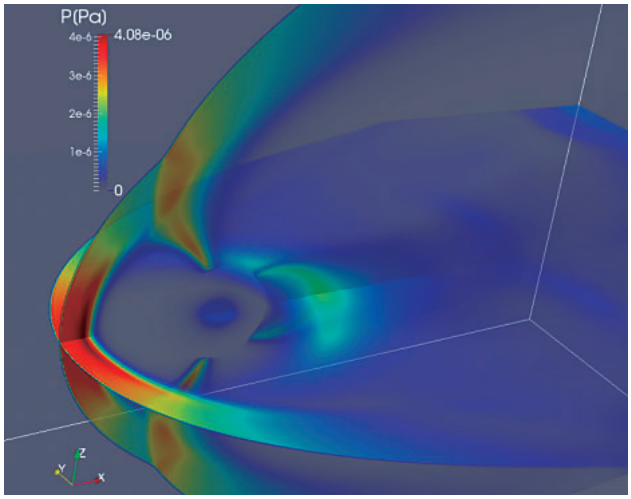

図16　太陽風と惑星の相互作用により発生する衝撃波．

5．国内の GPU スパコンの利用について

GPU コンピューティングをプラズマ・核融合分野のシミュレーションに適用することで，これまでできなかった規模の計算を短時間で実行し，新しい現象の発見や理論の解明につなげる可能性がある．CUDA のプログラミングや OpenACC は確かに取っ付き難いところがあるが，その後はそれほど難しいものではない．ほんの少しの努力で最初の障壁を乗り越えれば，得るものは大きいと信じている．

国内には東京工業大学・学術国際情報センターの TSUBAME2.5 以外にも，京都大学学術情報メディアセンター，九州大学情報基盤研究開発センターなどにも小規模の GPU を搭載したスパコンがある．学術的課題に対しては，それらを学際大規模情報基盤共同利用共同研究拠点の公募型共同研究課題や「京コンピュータ」以外の HPCI（革新的ハイパフォーマンス・コンピューティング・インフラ）構成機関が提供する計算機資源を使う課題として採択されれば一定量の計算資源が無償で利用できる．また，産業利用についても，東京工業大学・学術国際情報センターでは無償のトライアルユース，有償の共同利用サービス（成果公開／非公開）を行っており，GPU を利用できる機会は確実に広がっている．

謝　辞

本解説で紹介した計算はTSUBAME2.0/2.5で実施したもので，東京工業大学・学術国際情報センターに深く感謝の意を表する．本研究の一部は科学研究費補助金・基盤研究(S)課題番号 26220002「ものづくりHPCアプリケーションのエクサスケールへの進化」および基盤研究（B）課題番号 23360046「GPUスパコンによる気液二相流と物体の相互作用の超大規模シミュレーション」，科学技術振興機構CREST「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」，学際大規模情報基盤共同利用・共同研究拠点，および革新的ハイパフォーマンス・コンピューティング・インフラから支援をいただいた．記して謝意を表す．

参 考 文 献

［1］青木尊之，額田 彰：はじめてのCUDAプログラミング（工学社，2009）．
［2］NVIDIA: CUDA Programming Guide 6.0 (2014).
［3］http://www.top500.org/
［4］http://www.green500.org/
［5］T. Shimokawabe *et al.,* in Proceedings of the 2011 ACM /IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'11, IEEE Computer Society, New Orleans , USA (2010).
［6］T. Shimokawabe *et al.,* in Proceedings of the 2014 ACM /IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'14, IEEE Computer Society, New Orleans, USA (2014).
［7］X. Wang and T. Aoki, Parallel Computing, **37**, 521 (2011).
［8］M. Lesieur *et al.*, *Large-eddy simulations of turbulence* (Cambridge University Press, New York, 2005).
［9］M. Germano *et al.,* Phys. Fluids A: Fluid Dynamics, **3**, 1760 (1991).
［10］H. Kobayashi, Phys. Fluids **17**, 045104 (2005).
［11］小野寺直幸他：情報処理学会ハイパフォーマンスコンピューティング研究会主催HPCSシンポジウム（2013）．
［12］H.A. van der Vorst, SIAM J. Sci. and Stat. Comput. **13**, 631 (1992).
［13］R. Kobayashi, Physica D, Nonlinear Phenomena, **633-4**, 410 (1993).
［14］小川 慧他：情報処理学会論文誌コンピューティングシステム **3**, 67 (2010).
［15］T. Shimokawabe *et al.,* in Proceedings of the 2011 ACM /IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'11, IEEE Computer Society, Seattle, USA (2011).
［16］都築怜理他：情報処理学会論文誌コンピューティングシステム **9**, 82 (2013).
［17］N. Fujita *et al.,* Proc. of Int. Workshop on Parallel and Distributed Scientific and Engineering Computing 2014 (PDSEC2014) (with IPDPS2014), Phoenix, USA 2014.
［18］Un-Hong Wong *et al.,* Comput. Phys. Commun. **185**, 1901 (2014).
［19］X. Kong *et al.,* J. Comput. Phys. **230**, 1676 (2011).
［20］V.K. Decyk and T.V. Singh, Comput. Phys. Commun. **182**, 641 (2011).

青木尊之（あおき たかゆき）

東京工業大学学術国際情報センター教授．助手の時代まで，慣性核融合の理論・シミュレーションの研究に従事．最近は大規模数値流体力学，計算力学，ハイパフォーマンス・コンピューティングを専門とし，登録者数1,000名を超えるGPUコンピューティング研究会を主宰し，これまでに17回のハンズオン講習会を開催する等，CUDAプログラミングの普及も行っている．2011年にACMコードンベル賞を受賞している．