3 フル GPU による CFD アプリケーション

青木尊之（東京工業大学学術国際情報センター）

はじめに

最近，GPGPU（General-Purpose Graphics Processing Unit）や GPU コンピューティングという言葉を頻繁に耳にするようになった．パソコンの画面出力用のグラフィクス・カードに用いられている GPU を画像処理以外の目的で用いることを言う．GPU が注目されている理由は主に以下の 3 点，① GPU の（理論）演算性能が単精度であるが 1TFlops を超えるような高性能，② 価格がハイエンドのグラフィクス・カードでも数万円程度である点，③ 身近なデスクトップパソコンにも簡単に取り付けられる手軽さにあるだろう．ハイエンドのグラフィクス・カードになると大型のファンが付いているのでパソコンの拡張スロットが 2 つ占有されている．CPU よりも消費電力は大きいが，ワット当たりの Flops 値で比較すると実は GPU の方が消費電力は約 1 桁小さい．

この 10 年間で GPU は目覚ましい性能向上を遂げており，プログラマブル・シェーダが開発され，浮動小数点の計算も可能になった．Cg（C for Graphics）と呼ばれる言語で記述することでプログラミングは容易になったが，汎用計算に適用するためにはグラフィクス機能に置き換えて実行させる必要があった．たとえばピクセルの色を周囲と混ぜるような処理を拡散方程式を解くことに対応させていた．

一昨年，CUDA[1] と呼ばれる GPGPU 用の統合開発環境が NVIDIA 社よりリリースされ，グラフィクス機能をまったく意識することなしに C 言語の拡張としてプログラミングすることが可能になった．図 -1 に示すように NVIDIA 社や AMD 社はビデオ出力端子が付いていない GPGPU 専用のカードもリリースしている．本稿で紹介する GPU による流体計算は，CUDA を用いてプログラムされている．

GPU コンピューティングによる成果として坂牧らの分子動力学加速[2] などすでにいくつか報告されている．特に天体物理の重力多体計算では単一 GPU で数 100GFlops という実行性能[3] が報告されており，GPU の演算性能を十分に引き出すことに成功している．一方，格子計算は計算精度が高いので理工学のさまざまな分野で重要な解析ツールとして広く使われているが，GPU を利用した高速計算の報告はまだわずかである．

図 -1　ビデオ出力端子を持たない GPGPU 専用カード

本稿では，GPU を用いた CFD（Computational Fluid Dynamics：流体計算）が実用レベルに達していることを示すために，圧縮性流体計算としてレーリー・テーラー不安定性の成長過程，浅水波方程式によるリアルタイム津波計算，非圧縮性流体計算として典型的な円柱周りの流れの計算を GPU で行った例を紹介する．

フル GPU 計算

CFD に限らず，既存の HPC アプリケーションに GPU を適用して高速化を図るとき，まず計算負荷の高いホットスポットと言われる部分を GPU の計算に置き換えようと試みるであろう．まさにアクセラレータとしての利用である．実際，気象計算の WRF（Weather Research and Forecast）コミュニティ・モデルでは物理過程のごく一部の計算を GPU で置き換えて処理することにより，全体の計算時間を約 30% 短縮するとこができたと報告している．しかし，このような GPU のアクセラレータとしての利用では GPU を使うたびに CPU のメイン・メモリと GPU のビデオ・メモリの間にデータ転送が発生し，PCI Express バスを介した転送時間がボトルネックとなり高速計算を阻害してしまう．

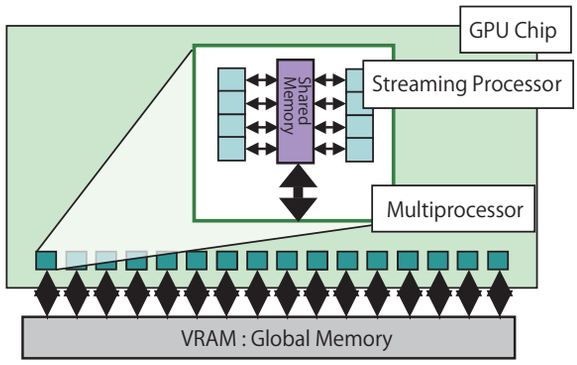

図 -2　NVIDIA の GPU アーキテクチャ

一方，初期条件などの計算に必要な情報を一度 GPU に転送した後は GPU 上ですべての計算を行い，計算結果を取り出すときのみ CPU のメイン・メモリにデータを転送するやり方をフル GPU 計算と呼ぶことにする．フル GPU 計算を行うことにより，CPU からは GPU 上に置かれたカーネル関数に引数を伴った実行命令を送るだけであり，CPU に比べて数 10 倍～ 100 倍以上の加速が期待できる．本稿で示す CFD アプリケーションは，すべてフル GPU 計算によるものである．

▶ GPU のハードウェア

NVIDIA 社の GPU のアーキテクチャを CFD の計算に関係する部分のみ図 -2 に示す．GPU を搭載したビデオカード上に DDR3 などの高速なビデオ・メモリ（VRAM）があり，薄い緑色の部分は GPU チップの内部であることを示している．Streaming Processor（以下，SP）と呼ばれる最小の演算ユニットがあり，浮動小数点演算を 1 クロックで加算と乗算ができる．8 つの SP で 1 つの Multiprocessor（MP）を構成しており，MP 内に 300GB/sec 以上と言われるレジスタ並みに高速なデータ転送が可能な 16kB の共有メモリがある．最新の GPU では 30 個の MP があり，合計 240 個の SP がある．VRAM と SP 間のデータ転送レートはメモリ・クロックとメモリ・インタフェースに依存するが，最新のハイエンド GPU ではピーク性能が 140GB/sec を超えていて，通常の CPU の 10 倍以上の値となっている．GPU と同様の SIMD 型アクセラレータである ClearSpeed はメモリバンド幅が狭く，ClearSpeed を流体計算に適用し難しくしている最大の理由である．

▶ CUDA による超多スレッド計算

GPU では 200 個以上もある SP（演算ユニット）を効率的に使うためにスレッド計算を行う．CPU でもマルチコアを使い Open MP 等でマルチスレッド計算を行うのは当たり前になっている．しかし，CPU の場合はスレッドを生成・実行する際のオーバヘッドが大きいためにスレッド数はおおむね CPU のコア数に等しい場合が最も効率が良い．一方，GPU ではスレッド生成のコストがきわめて小さく，細粒度のスレッドを大量に生成してジョブスケジューラに任せることにより効率的な処理が行われる．そのため GPU 計算では数 1,000 ～数 1,000,000 スレッドという超多スレッド計算を行うことになる．逆に，数 100 以下のスレッド数では計算している間に効率良くデータ転送させることが難しく，実行性能が低下してしまう．

2 次元格子での計算スレッドを図 -3 に示す．CUDA には grid, block, thread という概念があり，プログラミングする上でこれを意識する必要がある．block をまとめたものを grid と言い，block の中で thread が管理されている．図 -3 では 50×50 の計算格子に対して，5×5 個の block があり，その中に 10×10 個の thread があるので総スレッド数は 2,500 となる．

CPU で配列 a［2500］にアクセスする C 言語のプログラムは，以下のように for ループを用いて表される．

```
nx = 50;    ny = 50;
for(j = 0; j < ny; j++) {
   for(i= 0; i < nx;; i++) {
       a[nx*j + i] = . . .;
   }
}
```

一方で，CUDA の GPU kernel 関数では 1 スレッドが 1 格子点のデータ転送や計算を行うことになる．個々のスレッドには blockIdx, threadIdx が個別に割り当てられ，これらを用いて配列のインデックス計算を以下のように行うことができる．

```
i = blockDim.x* blockIdx.x + threadIdx.x;
j = blockDim.y* blockIdx.y + threadIdx.y;
a[nx*j + i] = . . .;
```

このように CUDA のプログラムにはループがない点が興味深い．

CPU で流体計算を行う場合は，演算性能に対してデータ転送速度が追いつかず，ほとんどの時間は演算が待ち状態になっている．これを補うのがキャッシュ・メモリであり，隣接格子点の計算で使ったデータはキャッシュ・メモリに置かれ，メイン・メモリに再度取りに行く回数を低減することで計算効率を上げている．キャッシュ・メモリはプログラム中からは見えなく，コンパイラとハードウェアに最適な制御を任せている．

一方，GPU でも VRAM と SP 間のデータ転送速度

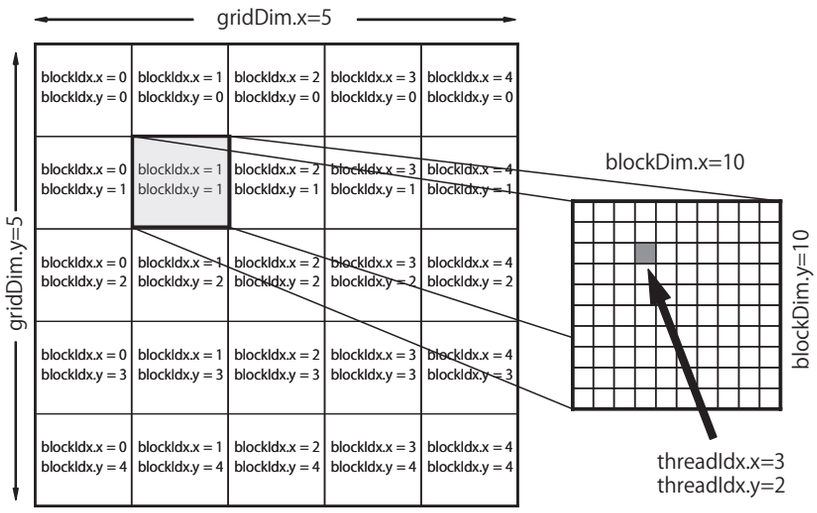

図 -3　2 次元格子に対する CUDA のスレッド分割例

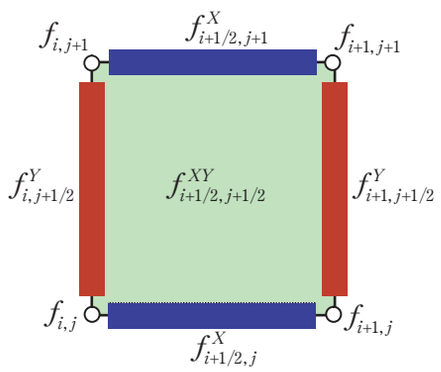

図 -4　保存形 IDO 法の変数配置

は CPU より高速であるが，SP の演算性能が高いために流体計算ではやはりデータ待ち状態になる．GPU では共有メモリに CPU のキャッシュ・メモリの役割をさせることができる．ただし，共有メモリからデータを読み出したり書き換えたりする操作はユーザがプログラム中で制御しなければならない．逆に，ユーザがすべて制御することができるため，スパースなメモリアクセスに対しても比較的容易にキャッシュ効果を上げることができる．block 内の thread は共有メモリにアクセスが可能であるが，1 MP（8 SP）当たりに 16kB しかなく，いかに効率的に共有メモリを使うかが GPU を使う流体計算で最も重要な部分となる．

圧縮性流体計算

流体計算の中でも陽的時間積分ができる場合は比較的 GPU にのりやすい．格子点ごとに単独の並列計算を行うと考えてよいので，隣接格子点データに対して依存性があると計算が困難になる．また，非構造格子のようにデータアクセスが不連続な場合もデータ転送速度が著しく低下してしまう．

GPU による直交等間隔格子での圧縮性流体計算の例を示す．式 (1) の Euler 方程式を高精度計算手法である保存形 IDO 法[4]で解く．

$$\frac{\partial \boldsymbol{E}}{\partial t}+\frac{\partial \boldsymbol{F}}{\partial x}+\frac{\partial \boldsymbol{G}}{\partial y}=0 \tag{1}$$

$$\boldsymbol{E}=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ e\end{pmatrix}\quad \boldsymbol{F}=\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho vu\\ eu+pu\end{pmatrix}\quad \boldsymbol{G}=\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ ev+pv\end{pmatrix} \tag{2}$$

ここで ρ は密度，u は x 方向の速度，v は y 方向の速度，p は圧力，e は内部エネルギーである．マルチモーメント手法であるため，図 -4 に示すように空間の格子点上に従属変数の値（Point Value : PV），x 方向の格子点間に x 方向線積分値（XI），y 方向の格子点間に y 方向線積分値（YI），中央に面積分値（XY）を配置してそれぞれ独立な従属変数となっている．局所空間に PV と積分値を用いた高精度な補間関数を構築する．PV についてはその補間関数を微分して導出される微係数を用いて時間積分を行う．積分値については，従属変数の積分の方向と同じ方向に Euler 方程式を区間積分して得られる式を時間積分する．面積分は有限体積法と同じ式になり，保存量に対する flux 形式になる．

保存形 IDO 法は計算精度が高く，他の高精度スキー

ムと比較しても位相誤差が小さいという特徴がある．圧力と速度場とのカップリングも良く，圧縮性流体を含め，さまざまなCFDアプリケーションで良好な計算結果を得ている[6]．

時間積分には3段ないし4段のルンゲクッタ法を用いるので，各段で従属変数に対する時間微分を求める必要がある．共有メモリをキャッシュ・メモリとして利用することにより，隣接格子点へのVRAMアクセスが不要となり，担当する格子点の1回のVRAMアクセスで済む．ただし，共有メモリのサイズが16kBであるため，計算領域をどのように領域分割してblockに割り当てるかが重要となる．図-4の変数配置に対して，PV, XI, YI, XYの従属変数を4つの配列で保持するより，1つの配列で奇数・偶数で交互に格納する方が効率的であることが分かっている．また，2次元保存方程式を解く際のPV, XI, YI, XYの計算量は大きく異なり，計算負荷のバランスを考えると1スレッドが4つの変数に対する計算を担当するのが望ましい．

しかし，各スレッドがPV, XI, YI, XYと順にデータをロードすると，アクセスが連続アドレスにならずデータ転送レートは極端に低下する．そこで，1スレッドの担当する格子の計算とデータロードを同一にせず，効率的に共有メモリにロードを行ってから計算を行う工夫をしている．

圧縮性流体計算の例として，油のように比重の軽い流体の上に水のような重い流体が置かれている状態は不安定であり，界面の乱れがレーリー・テーラー不安定性として急速に発達することが知られている．レーザ核融合や超新星爆発にも表れる現象で，特に高波数の擾乱の成長率は大きく，線形成長を過ぎてfree fallと呼ばれる段階までを計算するには多数の格子点が必要であり長時間の計算が必要とされている．図-5は初期に上下にきれいに分離していた密度分布が時間とともに乱れていく様子を示している．NVIDIAのGeForce GTX280を使うことにより，数10GFlopsの計算速度が達成されている．CPUで計算した場合には，約1GFlops程度であるので，数10倍の加速が達成されたことが分かる[5]．

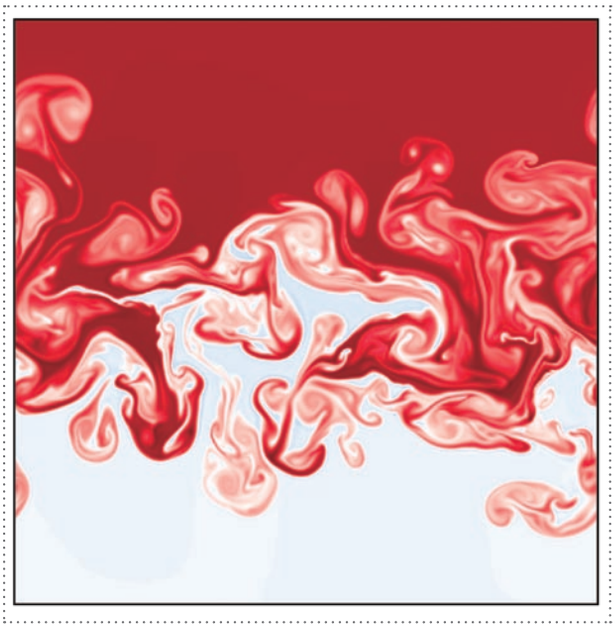

図-5　GPUによる圧縮性レーリー・テーラー不安定性の成長

リアルタイム津波シミュレーション

現在気象庁が行っている津波警報の発令は，あらかじめ計算しておいたデータベースに基づいている．地震の規模，震源など地震のパラメータ空間は広く，データベース方式では津波到達時刻や波の高さの予測精度は低い．地震発生直後から津波シミュレーションをスタートさせ，実際の津波が沿岸に到達するより早い時間で計算が終了すれば，より高い精度で津波の到来を予測でき，被害を低減させることが期待できる．GPUを用いたリアルタイム津波計算が可能になれば，コストの面，設置スペースの面で利点は大きく，オンサイトごとで詳細な津波計算も可能になる．

津波シミュレーションの基礎方程式は非圧縮性流体に浅水波近似を用いて導出される2次元浅水波方程式

$$\frac{\partial \boldsymbol{U}}{\partial t}+\frac{\partial \boldsymbol{F}}{\partial x}+\frac{\partial \boldsymbol{G}}{\partial y}=\boldsymbol{S} \tag{3}$$

$$U=\begin{pmatrix} h \\ hu \\ hv \end{pmatrix},\ F=\begin{pmatrix} hu \\ hu^2+\frac{1}{2}gh^2 \\ hvu \end{pmatrix},$$
$$G=\begin{pmatrix} hv \\ huv \\ hv^2+\frac{1}{2}gh^2 \end{pmatrix},\ S=\begin{pmatrix} 0 \\ -gh\frac{\partial z}{\partial x} \\ -gh\frac{\partial z}{\partial y} \end{pmatrix} \tag{4}$$

であり，hは水深，uはx方向の速度，vはy方向の速度，gは重力加速度，zは海底の高さである．複雑な海底地形や遡上にも対応する必要があるとともにリアルタイム性が重要である．効率的な計算手法として特性線に基づいたセミ・ラグランジアン法を導入している．ここでもマルチモーメント保存形IDO法を用いることにより，積分値に対してはflux形式とすることで衝撃波等も正確に解けることが保証される．この手法を用いるためにFractional Step法による次元分割で計算を行い，位相誤差の増加よりも計算効率の向上を選んでいる．

GPUでの計算については，次元分割を行っているのでx方向の計算はnx×1のスティック上のブロック分割

を行い，共有メモリの利用は比較的容易である．一方，y 方向の計算は x 方向に最低 16 個の連続アドレスのアクセスが必要なため，ブロック分割も x 方向に 16 格子とし，y 方向には共有メモリが許す限り長い幅のブロックサイズとする．浅水波方程式に対してマルチモーメント手法で計算しているため，PV と積分値が交互に並ぶ配列要素にアクセスしなければならず，1 スレッドで PV と 3 種類の積分値について計算を行っている．

本津波シミュレーションの計算時間は，高精度かつ効率の良い数値計算手法を選んでいるため，CPU での実行においても従来の Leap-frog 手法を使った計算よりもかなり速くなっていて，さらに GPU を使うことで 60 倍以上の加速を行うことができた．正方形の計算領域の場合は x 方向と y 方向の計算量が同じになるが，GPU での計算速度は x 方向の計算の方が約 3 割速いという結果になった．図 -6 は実際の地形を設定して，津波の遡上を計算した際のスナップショットである．GeForce GTX280 を用いると，1024×1024 の格子点を用いて 2,000 ステップ計算するのに約 20 秒で済んだ．リアルタイム津波計算に必要な計算速度としては十分であると言える．

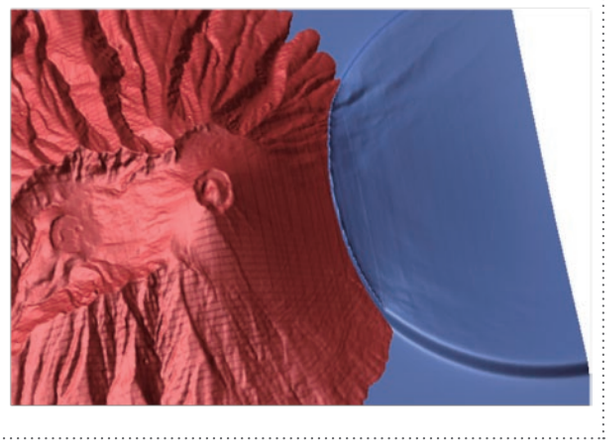

図 -6　遡上を含んだ津波シミュレーション

非圧縮性流体計算

身の回りの流体現象の多くは非圧縮性流体で近似でき，産業分野でのほとんどの流体解析は非圧縮性流体の Navier-Stoker 方程式に基づいて計算されている．

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} = -\nabla p + \frac{1}{\mathrm{Re}} \Delta \boldsymbol{u} \tag{5}$$

$$\nabla \cdot \boldsymbol{u} = 0 \tag{6}$$

$\boldsymbol{u}$ は流速，p は圧力，Re はレイノルズ数である．高精度かつ低計算負荷の数値計算手法が求められるため，SMAC（Simplified Maker-and-Cell）法に基づいたセミ陰解法とし，圧力勾配項以外は Fractional Step 法で解くことにした．移流項については時間空間 3 次精度の Cubic セミラグランジュ法を Directional Splitting 法で解く．Fractional Step 法に基づいて各パートを以下に述べる．

▶移流項計算

Cubic セミラグランジュ法により x 方向の移流計算を GPU で効率的に行うためには，VRAM へのアクセスを可能な限り少なくする必要がある．CUDA の block を (nx, 1, 1) にとると，各 thread がロードする n ステップ目の速度を GPU の共有メモリ上に置くことにより隣接格子点の参照に対して VRAM へのアクセスを回避できる．これによって x 方向の移流計算の速度は GeForce 8800 GTS を用いると 56GFlops に達する．CPU で同じ計算をした場合，Xeon 2.5GHz で 1GFlops 程度の実行速度であるので，GPU により約 50 倍以上高速な計算を行うことができる．

y 方向の移流計算は block(1, ny, 1) ととると，VRAM へのアクセスが不連続になり，データ転送速度が極端に低下してしまう．そこで，block(16, 16, 1) などととり，x 方向の 16 格子点を VRAM に対して連続にアクセスさせる．y 方向にも 16 格子点あるため，共有メモリの効果が十分に現れ，演算速度は 40GFlops を超える．

▶拡散項計算

2 次精度中心差分法を用いると，計算ステンシルは x 方向と y 方向のそれぞれの隣接点を参照する．block としては（nx,1,1）とするが，shard メモリは nx のサイズの 1 次元配列を 3 個用意する．(jx, jy) の 格子点を計算する thread は，x 方向の隣接点アクセスは移流計算と同じように行うが，y 方向については jy+1 のデータを格納する shard メモリと jy-1 を格納する shard メモリにアクセスすることにより，VRAM へのアクセスを回避できる．jy 列の計算が終わり，jy+1 列目の計算を行うとき，先ほど使った jy-1 を格納していた共有メモリの内容は不要になるため，新たに jy+2 列目の VRAM の値を格納するために使うことにより，jy+1 列目の格子を計算することができる．このように共有メモリをリサイクルすることにより，16kB と少ないサイズでも効率良く VRAM にアクセスすることができる．

▶マルチグリッド法による Poisson 方程式解法

通常の CPU での計算においても低波数誤差を効率的に低下させる方法としてよく知られるマルチグリッド

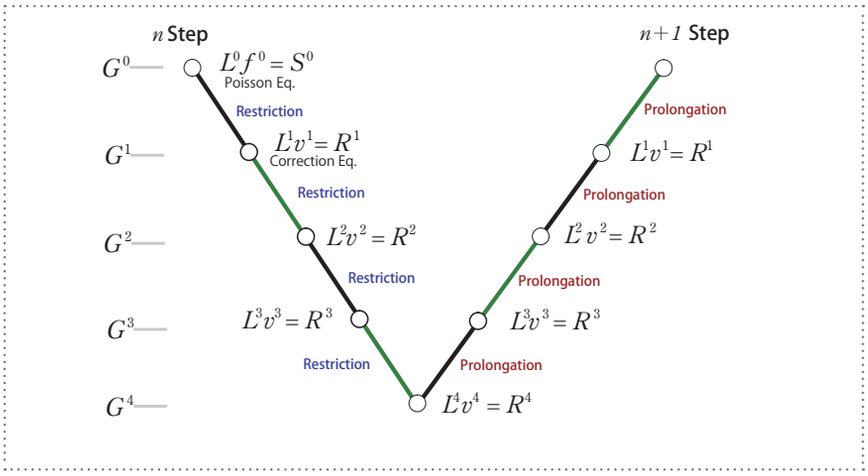

図 -7　マルチグリッド SOR 法の V サイクル

法を GPU のコードとして実装した．単相の流体計算であるため，定常反復法として SOR 法を用いた．ただし，thread 間は完全な並列計算であるため，Red & Black 法を用いている．ここでは，図 -7 に示すシンプルな V サイクルを用いている．最終段の粗い格子に近づくにつれて格子点数が少なくなるために並列度が低下し，GPU の演算性能としては 20 数 GFlops 〜 30 数 GFlops となってしまう．Poisson 方程式の計算についても通常の CPU では 1GFlops 以下の計算速度しか出ないので，GPU での加速は 20 数倍〜 30 数倍に達している．GPU では単精度計算であるために残差を十分に評価できないが，マルチグリッド法の 2 段目以下の修正量に対する計算は単精度で十分であり，収束性には影響がない．今回の流体計算で必要とされる精度に対しては，すべて単精度で計算しても問題がなかった [5]．

▶物体周りの流れの計算

CFD の典型的な適用例である 2 次元円柱周りの流れ計算に適用した結果（計算領域の一部）を図 -8 に示す．格子点数は 1024×512 であり，レイノルズ数を 2,000 としている．実際の渦構造には 3 次元性が出てくるので 2 次元計算は意味がないが，境界層の剥離および円柱後方のカルマン渦が高精度に計算されていることが分かる．通常はポスト処理により可視化を行うが，GPU で計算することにより，まるで動画を見るようにリアルタイムに計算結果を確認することができる [5]．

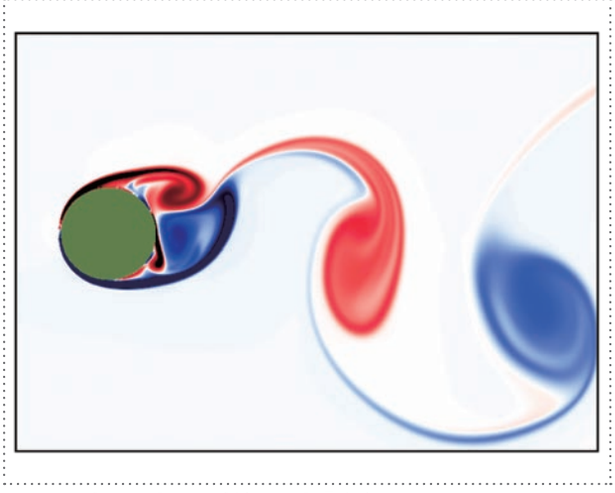

図 -8　円柱周りの非圧縮性流れの渦度表示

マルチ GPU 計算

これまでの流体計算の例で，シングル GPU では十分流体計算が可能なことが示されたと思われる．しかし，グラフィクス・カード上に搭載される VRAM の量には制限があり，最新の NVIDIA の Tesla シリーズでも 4GB である．大規模な流体計算では複数のグラフィクス・カードを使ったマルチ GPU 計算が必要となる．マルチ GPU 計算の例として理研（姫野）ベンチマークテストを GPU で計算した例を示す．

一般座標系において，従属変数 p に関する 3 次元 Poisson 方程式を 2 次精度中心差分法で離散化した式を Point Jacobi 法で解く問題であり，C 言語および Fortran 言語のソースコードが準備されている．そこでは，反復計算中に変化しない係数をすべて配列として保持し，反復計算のたびに参照するコードとなっている．計算規模が S サイズ 65×65×129，M サイズ 129×129×257，L サイズ 257×257×513，XL サイズ 513×513×1025 の 4 種類が準備されている．コード可変で独自の高速化チューニングが許されるのは S サイズの問題である．65×65×129 の 1 配列は 2.18MB にしかならないが，理研ベンチマークテストは係数をすべてこのサイズの配列 12 個に格納し，2 つの配列が 1 つ前のステップの値と現ステップの値の格納に割り当てられ，読み書きの対

象となる．この 12 個の配列は格子点間の相互参照がないために CPU キャッシュが効かず，メモリバンド幅の広いベクトル型スーパーコンピュータが非常に有利と言われている．従属変数 p に関しては図 -9 のように多数の隣接参照があり，CPU の場合は計算サイズに対応してキャッシュが効くので，キャッシュ・メモリのサイズが計算速度に大きく影響する．

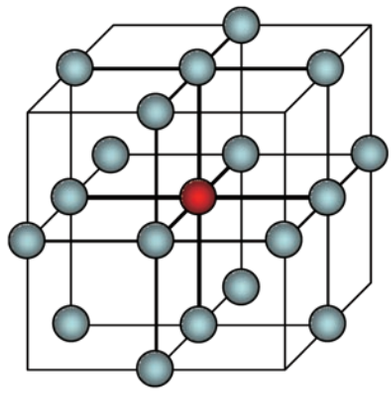

図 -9　理研（姫野）ベンチマークの計算ステンシル

▶ 1GPU による理研ベンチマークテストの高速化

理研ベンチマークテストの 14 個の配列を GPU の VRAM 上に確保し，あらかじめ cudaMemcpy 関数を用いてデータ転送しておく．VRAM は CUDA ではすべてのスレッドからアクセスできる．GPU の場合，VRAM から共有メモリにデータ転送する速度は，アクセス方法・転送量に応じて大きく変化する．VRAM に対する連続アクセスはバースト転送を行う効率的な Coalesced Access となるが，ランダムメモリアクセスは転送速度が 1/10 に低下してしまう．先に示した通り，理研ベンチマークテストはデータ転送処理が多いため，Coalesced Access ができたとしても実行時間においてデータ転送時間が占める割合が大きい．VRAM 上の 1 変数配列 2.18MB の読み出し速度は 70［GB/sec］程度であり，単精度データに対しては 70÷4＝17.5 GWord/sec となる．一方，理研ベンチマークテストは 1 格子点当たり 14 変数を読み書きし，34 回の浮動小数点演算を行う．したがって，演算時間が無視できるほど短い（性能が高い）と仮定しても最大演算性能は 17.5 ×34/14 ＝ 42.5 GFlops である．しかし，圧力変数 p の 18 回の隣接参照を毎回行うと，計算速度は 17.5×34/(14＋18)＝ 18.6 GFlops に低下してしまう．

圧力変数の隣接参照回数を減らすために，ここでも block 内で共有できる共有メモリをキャッシュとして利用する．共有メモリのサイズが 16kB と少ないため，計算領域を 16×16×8 の格子点の領域単位に図 -10 のように分割する．1 つの block がこの計算領域を担当し，全体では 4×4×16 の block が存在することになる．block 内のスレッドは (16＋2)×(16＋2)×(8＋2) の圧力変数を共有する．共有メモリのサイズが担当計算格子より大きいのは，p が隣接参照する際のはみ出しに対するパディングである．この部分のメモリロードは大半が Coalesced Access ではないため，全体としてのデータ転送速度を低下させるが，転送量が少ないために低下率は低い．

▶ 複数の GPU カードによる高速化

複数 GPU を用いてさらに理研ベンチマークテストの高速化を試みた．用いたマザーボードは PCI Express ×16（Generation 2）が 4 スロットある MSI K9A2 Platinum である．ただし，図 -11 のように 4 枚の

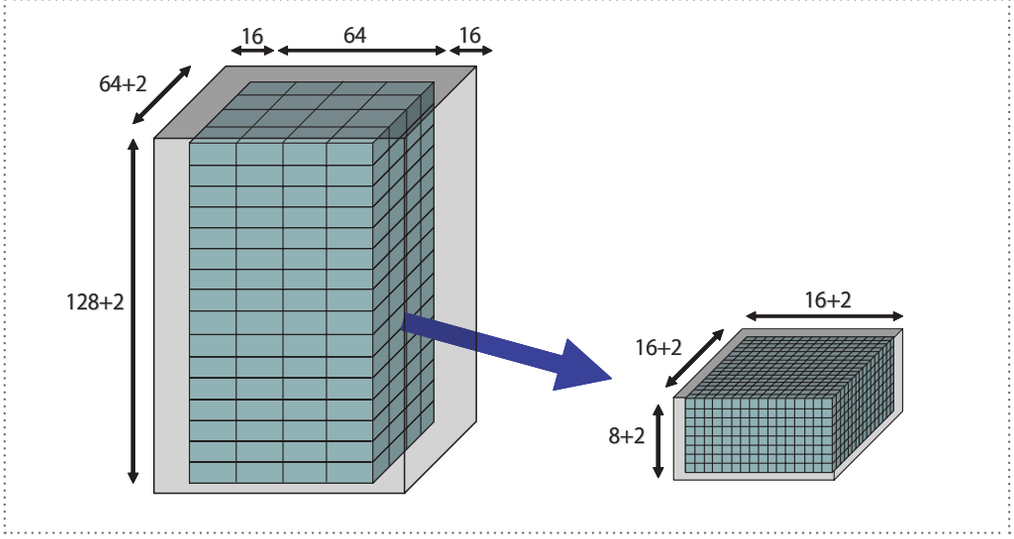

図 -10　計算領域の block 分割と共有メモリの割当て

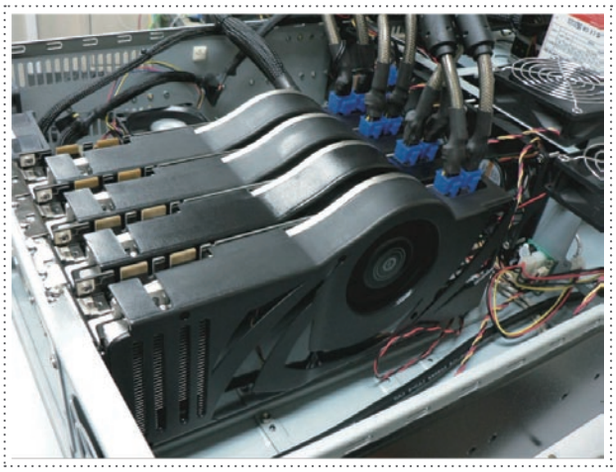

図 -11　PCI-Express スロットに装着した 4 枚の GPU カード

Size	# of GPUs	Performance［GFlops］	Time［sec.］
S	1	30.6	0.268
	2	42.5	0.193
	4	51.9	0.158
M	1	29.4	2.328
	2	53.7	1.275
	4	83.6	0.819
L	1	-	-
	2	-	-
	4	93.6	5.974

表 -1　理研ベンチマークテストの並列性能

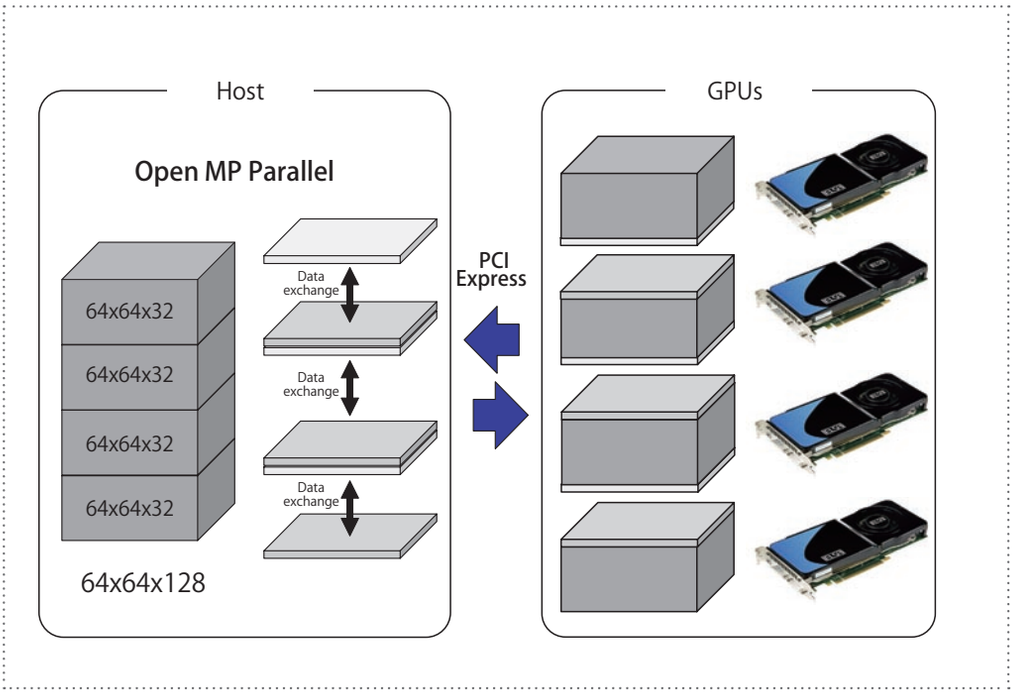

図 -12 マルチ GPU 計算のための領域分割と相互通信

GPU を装着した場合は，それぞれが PCI Express ×8（Generation 2.0）となる．計算領域を図 -12 のように長軸である z 方向に短冊状に 1 次元分割する理由は，圧力 p の計算の隣接参照が対角線にまで伸びているため，多次元分割を行うと隣接領域との通信が複雑になるためである．複数の GPU を制御するために，CPU 側では使用する GPU の数だけ Open MP のスレッドを走らせている．

計算速度の結果を表 -1 に示す．S サイズの計算を 4 GPU で行った結果は 51.9 GFlops で，4 枚の GPU を用いても 1 枚よりも約 1.7 倍しか速くなっていないことが分かる．PCI-Express Bus を介して GPU の VRAM と CPU 側のメイン・メモリ間のデータ転送が遅いことが最大の理由である．S サイズの計算では交換する境界領域のデータのサイズが 32kB と小さいためオーバヘッドが大きく転送速度が低いことが分かる．M サイズ，L サイズでは境界領域のデータ交換に要する時間が計算時間に対して相対的に小さくなり，L サイズの問題では 4 枚の GPU を用いて約 3 倍の性能向上が得られた [6]．

まとめと今後の展望

これまで，GPU を用いた CFD などの格子計算を加速する報告はあまり報告されてこなかったが，圧縮性流体計算，非圧縮性流体計算において，GPU を用いて CPU よりも数 10 倍高速に計算することができることが示された．本稿では省略したが，Phase Field モデルによる Cahn-Hilliard 方程式の 3 次元計算は流体計

算とほとんど同じ手法で計算でき，GeForce GTX280 で 160GFlops の計算速度が得られている．また，2 次元 CIP 法による移流計算でも GeForce 8800GTS で 120GFlops が得られている．CPU ではあまり計算効率が上がらない高精度計算手法や 3 次元計算の方が GPU の性能を引き出しやすいことが分かる．

近年，スパコンはベクトル型から PC クラスタ型に移行してきたが，今後は GPU などのアクセラレータがほとんどのスパコンに接続される時代が訪れるであろう．GPU は演算性能が高いため，GPU 間の通信やノード間のインターコネクションの通信速度とのバランスが非常に重要となる．また，そのような環境に適した計算アルゴリズムの開発も必要となる．

謝辞　本研究の一部は科学技術振興機構 CREST「ULP-HPC：次世代テクノロジーのモデル化・最適化による低消費電力ハイパフォーマンスコンピューティング」および日本学術振興会グローバル COE プログラム「計算世界観の深化と発展」から支援を受けている．研究室の GPU クラスタの構築では多大な貢献をしてくれた大学院生の小川慧君，Open GL グラフィクスを担当してくれた Marlon R. Arce Acuña 君には感謝の意を表す．

参考文献

1) NVIDIA : CUDA Zone, http://www.nvidia.com/cuda/
2) 坂牧隆司，成見　哲，泰岡顕治：ビデオゲーム用ハードウェアを用いた高速分子動力学シミュレーション，p.101, 第 21 回分子シミュレーション討論会 (2007).
3) Hamada, T. and Iitaka, T. : The Chamomile Scheme : An Optimized Algorithm for N-body Simulations on Programmable Graphics Processing Units, http://arxiv.org/abs/astro-ph/0703100v1 (2007).
4) Imai, Y., Aoki, T. and Takizawa, K. : Conservative Form of Interpolated Differential Operator Scheme for Compressible and Incompressible Fluid Dynamics, Journal of Computational Physics, Vol.227, Issue 4, pp.2263-2285 (2008).
5) 青木尊之，小川　慧：フル GPU による CFD アプリケーション，日本機械学会・第 21 回計算力学講演会，琉球大学 (2008).
6) 小川　慧，青木尊之：CUDA による定常反復 Poisson ベンチマークの高速化，情報処理学会第 115 回 HPC 研究会，東京，pp.19-23 (2008).

(平成 21 年 1 月 6 日受付)

青木尊之(正会員)
taoki@gsic.titech.ac.jp

東京工業大学大学院総合理工学研究科修士課程修了 (1985)．同研究科助手，同大原子炉工学研究所助教授を経て，2001 年より同大学術国際情報センター教授．専門：数値流体力学，計算力学，大規模並列計算，科学技術計算のビジュアリゼーションにも興味を持つ．