AMR グリッド GPGPU を用いた竜巻の基礎構造解析例

The example of analyzing standard tornado structure with GPGPU and AMR mesh

○ 柴田祐樹，首都大，東京都日野市旭が丘 6-6, E-mail: shibata-hiroki-assembler-1991@hotmail.co.jp
田川俊夫，首都大，東京都日野市旭が丘 6-6, E-mail: tagawa-toshio@tmu.ac.jp
Hiroki Shibata, Tokyo Metropolitan Univ., Asahigaoka 6-6, Hino, Tokyo, 191-0065
Toshio Tagawa, Tokyo Metropolitan Univ., Asahigaoka 6-6, Hino, Tokyo, 191-0065

In order to make fast and efficient CFD computations for large-scaled flow phenomena with GPGPU, we attempted to utilize a computer language called C++ AMP on GPU. For an example of such a large-scale CFD computation, we focused on a tornado structure which has a wide dynamic range of flow and numerically solved time-dependent three-dimensional flow together with AMR mesh, which is utilized to reduce the total number of meshed and to save the computational time.

１． 研究背景

近年、GPU による数値流体解析が注目を浴びてきている。しかし、実際にどうすれば GPU を用いた高速計算ができるのか、また、難易度は、と言った問題に触れていることは少ないように思える。また、GPGPU 開発言語として、CUDA が有名であるが、そのほかの言語での実装例もまた例が少なく、選択肢の幅を狭めている、かつ GPGPU というものに対しての視野が狭くなりがちなように思える。本講演では、C++AMP を用いた GPGPU による数値流体解析例を紹介する。

C++AMP は、2012 年に Microsoft が策定した、C++11 の改良版として GPU へのラムダ式による命令転送を実装したものである。また、データ配列としては、テンプレート引数を持つクラスによって高度に抽象化、仮想化されたメモリ領域を扱える。データ転送の実装等、非常に多くの点において、プログラマの負担を減らせる、導入の簡単な言語であり、バグを減らすこともできる。また、対象とする GPU メーカーを選ばず、openCL にコンパイル可能なライブラリも提供されているため、Linux, Windows, 双方で動作させることが可能となっている。

CFD において、プログラム言語自体の難易度は軽視されがちだが、できる限り扱いやすいことが望ましい。プログラム言語に関しては C++AMP は、非常に実装しやすいものとなっていて、スキームの実装に集中できる。AMRGrid の GPU 上における実装を示すと同時に、ダイナミックレンジの大きい流れ場として、竜巻の解析例を挙げる。

２． 解析例

簡単な竜巻を、解析の例として挙げる。竜巻という現象は、非常にダイナミックレンジに富む流れ場となっており、AMRGrid を用いるべき解析対象である。

AMR Grid を用いて GPU 上で解析できるならば、ほかの多くの現象に対しても有用な例であり、また、それを C++ AMP を用いれば所学者でも一から実装可能な難易度であることを示したい。

３． 基礎方程式

$$\frac{\partial u_j}{\partial x_j} = 0$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j \partial x_j}$$

非圧縮の N-S 方程式を採用する。
もちろん、実際の竜巻に対して非圧縮性近似を用いることには多くの疑問が生じるが、計算のダイナミックレンジ性に関して言えば、非圧縮性流体でも同様に厳しいものとなる。よって、簡単な検証材料として、非圧縮性流体を採用する。

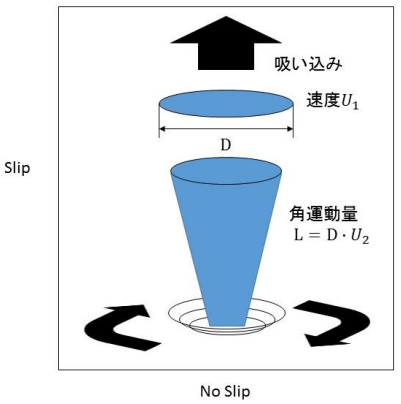

Fig.1 analysis condition

計算条件としては、領域上空に吸い込み場、領域遠方に回転場を与える。非圧縮性条件の場合、この吸い込み規模、角運動量供給量の 2 つによって竜巻の形状が決まることになる。
基本的な計算スキームとして、差分には梶島の 2 次精度対流型補間法[1]、必要に応じて、同風上法を、また連続の式との連立には、SMAC 法を用いる。

なお、今回の解析において、圧力修正に掛ける時間が支配的となることはなく、マルチグリッド緩和修正法を用いる必要はなかった。

いかに示すのは、境界条件を設定した例である。

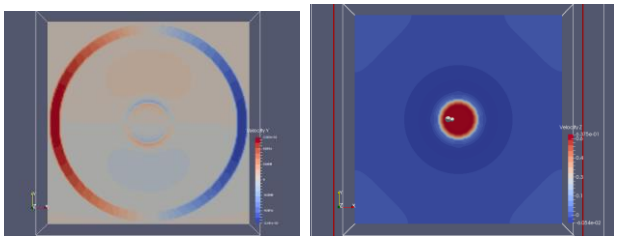

Fig.2　Left,rotate flow,　right, upward current

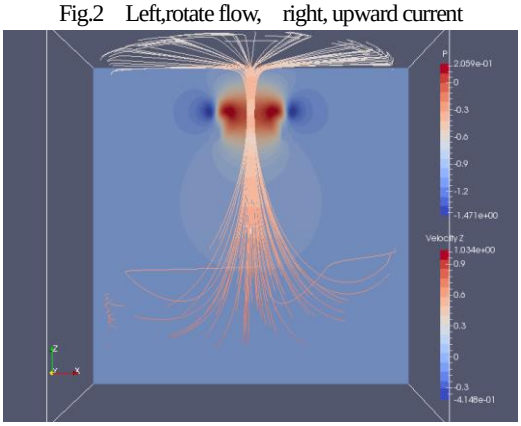

Fig.3　Streamline

旋回流と、上昇流を設定している。

4. AMR Grid

こちらは、AMR Grid の分割、初期値である。

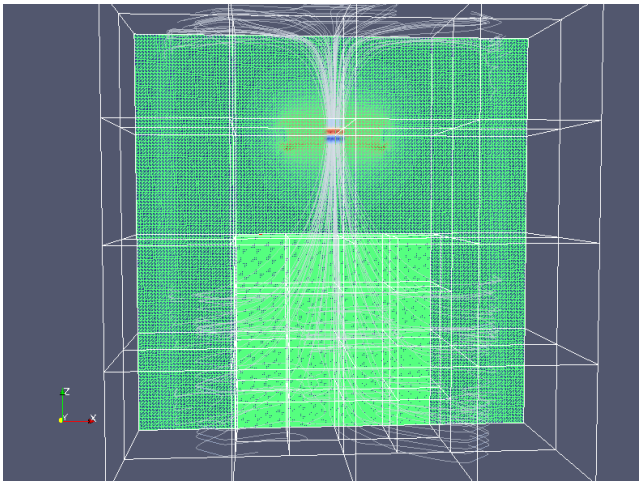

Fig 4　an example for AMRGrid

GPU により、計算の高速化、大容量化が期待できるが、それでも格子間隔の逆数の 3 乗$O\left(\frac{1}{\Delta x^3}\right)$に比例する格子数の 3D N-S 方程式を解くには十分ではなく、これを軽減する数学的手法も合わせて用いるべきである。

AMR Grid は構造格子の精度、非構造格子の局所性を兼ね備えた、大変効率の良いものである。特に、境界形状の複雑でない、宇宙空間、自然現象解析において、有効な選択肢となる。AMR Grid の計算オーダーは、十分な数の分割ボリューム、現象の局所性が確保されていれば、格子間隔の逆数の 1 乗オーダー$O\left(\frac{1}{\Delta x}\right)$まで減らせる。

そして、次に考えるべきことは、多ノード上での GPU 計算についてであり、単体の GPU 上で計算を行うならば、データ通信が発生せず、深く考えることなく、インデックスを用いて実装することもできる。しかし、隣の計算ボリュームのデータが当該 GPU 上にない場合、通信を行わなければならず、その場合分けのコードも煩雑になりがちである。こういった事情を解決する手法に仮想格子法が古くから用いられている。

以下、実装目標概要

- AMR Grid を用いる
- C++AMP を用いる
- 仮想格子による、AMR Grid のボリューム間の接合
- AMR Grid の単位ボリュームあたり、32^3 等、ある程度まとまった単位で計算を行うことにより、仮想格子に費やすメモリを減らし、かつ精度向上、計算の効率化を目指す。
- 隣接する格子間隔は、2 倍差までとする。
- データ通信の回数を最小とする。

5. データ通信について

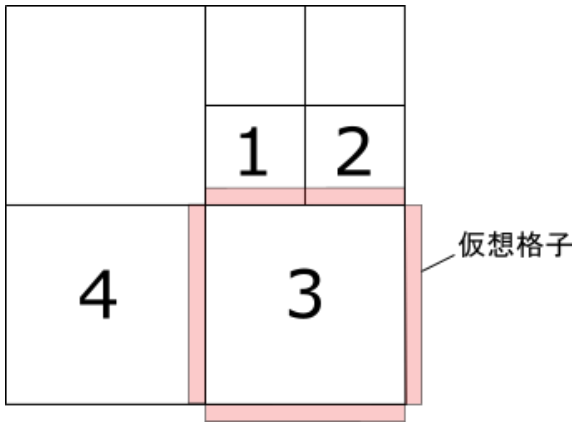

Fig 5　AMR Grid and virtual cell

たとえば、3 番ボリュームの計算を行う場合、4 番ボリュームのデータが必要になるが、それが同じ GPU 上に格納されているものとは限らない。また、1,2 番ボリュームとは格子間隔が違うので、補間が必要になる。これらを統一的に、データ通信する手法を考える。

C++のような、オブジェクト指向型言語を用いれば、これを統一的に行える。また、C++AMP 自体もオブジェクト指向となっている。

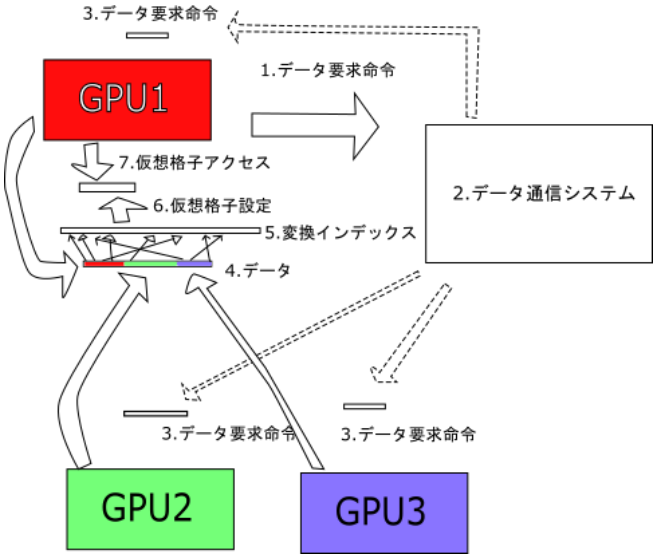

Fig. 6　The data transfer diagram

1. GPU1 から仮想格子として必要になる AMR Grid のボリュームのデータ要求を列挙する。
2. データ通信システムが、要求されたデータがどの GPU 上に配置されているか、検索する。
3. 各 GPU へ、要求命令を分配する。このとき、統一的に行うために、自分自身にも送る。
4. 各 GPU はデータ配列を用意し、一回の転送で、GPU1 の所定のメモリ位置へ送る。
5. 送られたデータが要求した順番になっていないので、要求した順番どおりにアクセスする。また、このインデックスはデータ通信システムが生成し、かつ、要求されたデータに重複があった場合、自動的に同一のデータを指し示すように工夫する。
6. 仮想格子へ設定する。
7. 仮想格子のデータを用いて微分方程式を解く

データ通信システムを、自前で実装する必要があるが、データの移動自体は C++AMP が提供するクラスに機能が含まれているので、そちらを用いて、我々はインデックスの生成に精を出すことになる。
AMR Grid の生成自体は、オクトツリーインデックスを用いるのが、比較的容易であった。

６．結果

いかに、本検証で用いたプロセッサを示す。

Table.1　Processor's spec

	Core i5 4670(4c)	R9 280X
Single Gflops	235	4096
Double GFlops	120	1024
Price(yen 2014)	30000	40000

Table.2 は計算時間比較である。
本比較は、単一ボリューム、単一 GPU の計算結果を比較したものである。

Table. 2　The time spent.[s] 2D N-S equation of natural convection.

格子数	Core i5 4670(1c)	R9 280X	ratio
1024	0.14	0.022	6.19
4096	0.39	0.022	17.72
16384	0.66	0.025	26.4
262144	8.3	0.21	39.5
1048576	34	0.79	43

CPU の方は、1 コアのみを用いた結果である。複数コア、複数ノードを用いた場合、CPU の並列化効率はより落ちるので、CPU 並列演算システムとの比較では、よりシステム規模に対する演算効率において、GPU1 台の方が有利になる。
以下に、解析した例を示す。中心の圧力が下がり、回転流が下まで届いている様子がうかがえる。

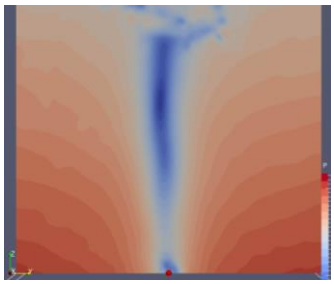

Fig.7　Pressure field

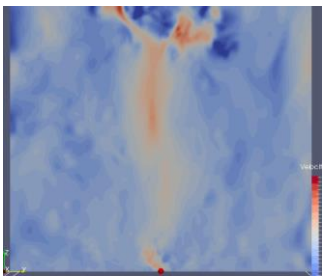

Fig.8　Upward wind field

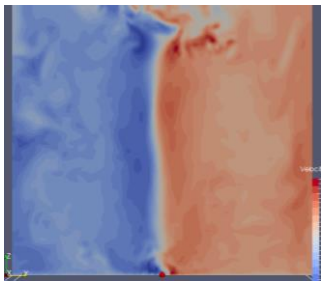

Fig.9　Rotate wind field

Fig.10　An example for visualization

GPU は可視化するとき、粒子追跡法の計算に用いることもできる。
Blender の ray tracing 機能と組み合わせれば、上記のような可視化を、計算時間と合わせて 1 日足らずで終えることができる。

７．考察

C++ AMP を用いた、GPGPU により、流体計算ができることを示せた。また、それは学生が可能なレベルでもある。
流れの解析において、計算手法は方法でしかないが、現状得られる演算器に頼る場合、工夫する必要があり、その手法を開発するなら、専門外の方が利用しやすいようにする必要がある。

８．結論

C++ AMP を用いて、確かに、数値流体力学を計算することが出来、またその高速化率も十分なものである。
データ通信については、間に合えば、学会のプレゼンで公表し、また、解説ホームページを公開している。
http://www.aerospace.sd.tmu.ac.jp/hydrodynamics/main/colums/AMPindex.html

９．参考文献

(1)　梶島岳夫, "乱流の数値シミュレーション" (2007)
(2)　平野博之, “流れの数値計算と可視化”2004.