Vol.2009-CG-136 No.2
2009/8/20

GPU クラスタによる 3 次元数値シミュレーションの高速分散可視化

関　将太郎[†]，安藤　英俊[††]，鳥山　孝司[†††]

物理現象の 3 次元シミュレーションを，GPU を用いて分散可視化する手法を提案する．低コストで高パフォーマンスな GPU を複数台用いることで大規模なシミュレーションの計算を可能にするとともに，高速な描画が期待できる．3 次元ならではの奥行きの扱いと，計算･可視化を分散させることで生じる問題に注意した．

Distributed-Visualization for 3D simulation on GPU cluster

SHOTARO SEKI[†], HIDETOSHI ANDO[††],
KOJI TORIYAMA[†††]

I suggest how to visualize 3D simulation of physical phenomenon with GPU. It can calculate a large scale simulation and expect fast drawing, using GPUs that is low cost and high performance. I paid attention that the handling the depth of characteristic of 3D, and that the problem caused simulation and distributed visualization.

1. はじめに　*

数値シミュレーションを行った結果は数値として求められる．しかしこの数値を見るだけでは現象を理解することは困難なので，感覚的に理解しやすくするために可視化を行うことが必要となる．(図 1)

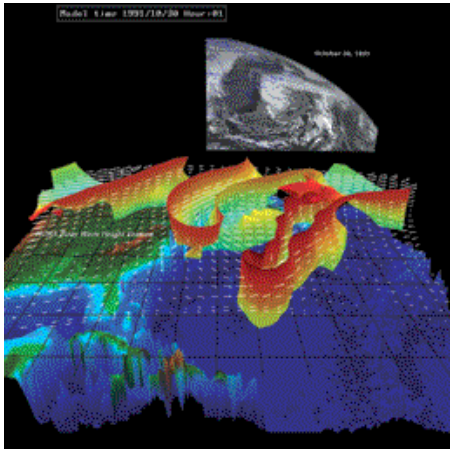


図 1:気象シミュレーションの例[1]
Fig1:Example of weather simulation[1]

シミュレーションを行う際の問題点は計算コストが高いことである．特に 3 次元におけるシミュレーションでは計算サイズによるコストが膨大に増えてしまい，高速化するためには計算性能の高いコンピュータが必要となる．そこで近年比較的に手に入れやすくなってきた GPU を用いた汎用計算(GPGPU : General Purpose GPU)を用いることでコストを抑えることができ，さらに本来の役目である可視化も行わせることで高速に描画することが可能となる. しかし GPU 自体はメモリが少なく拡張もできないので、大規模シミュレーションには膨大な時間がかかってしまう．この問題を解決するために GPU を搭載したマシンを複数台(GPU クラスタ)用い, 各マシンに処理を分散させることで大規模シミュレーションを行うことが可能となる．

* †，††，†††　山梨大学

2. 既存研究

清水の研究[2]に 3 次元の数値シミュレーションのリアルタイム可視化プログラム(DirectX 版)がある．図 2 は正方体の中を，シミュレーションの結果を元に動き回るパーティクルを描画したサンプルである．図 3 は 3 次元空間内における任意の面の可視化を行っているサンプルである．時間発展するシミュレーションに対してインタラクティブに，かつ複数の可視化法(矢印やパーティクルや面などを用いた描画法)で可視化を行っている．この研究は計算部分ではなく可視化部分の高速化を行っているプログラムである．しかしこの研究は単独マシンでの可視化を行っているので，大規模なシミュレーションには不向きである．

可視化の分散処理には坂下の研究[3]がある．この研究は分散処理を用いて 2 次元シミュレーションの可視化を行う方法である．サーバ兼描画ホストと計算・可視化ホストを用意し，通信を行って処理を分散させ，結果を描画ホストにまとめ最終的に 1 つの画像を作り上げるというものである(図 4). こちらは 2 次元のみで,3 次元は扱っていない.

また，分散可視化の関連研究として Maryland 大学の研究(図 5)や，三菱プレシジョンの Volume Graphics クラスタ(図 6)などもある．Maryland 大学の研究は GPU を用いているが，1 つの GPU が 1 つのディスプレイに出力する方法をとっている．そのディスプレイをタイル状に並べることで 1 枚の大きな絵を作り上げる手法である[4]．VG クラスタの方は, 各 GPU の作成した映像を専用装置を用いて合成する方法である[5]. いずれの研究もいくつものディスプレイや専用装置が必要でコストがかかりすぎてしまう欠点がある．

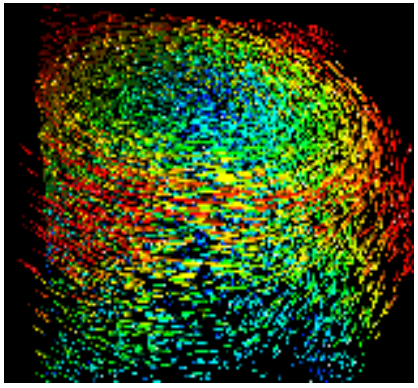


図 2:サンプル(パーティクルトレース)[2]
Fig2:Sample of particle trace[2]

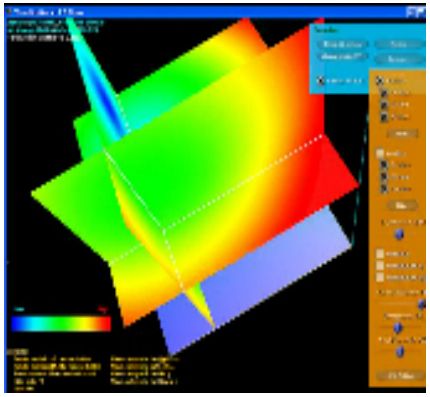


図 3:サンプル(任意の面による可視化)[2]
Fig3:Sample of visualization with quads[2]

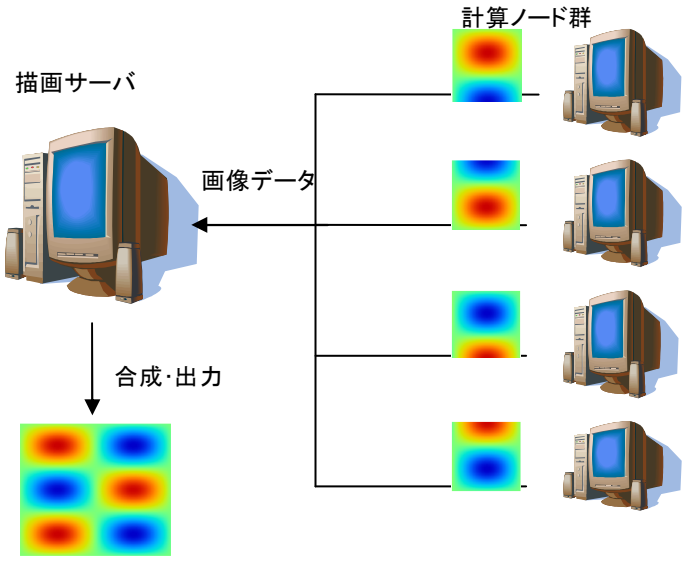


図 4:坂下の分散可視化手法の流れ[3]
Fig4:Way of Sakashita's distributed visualization[3]

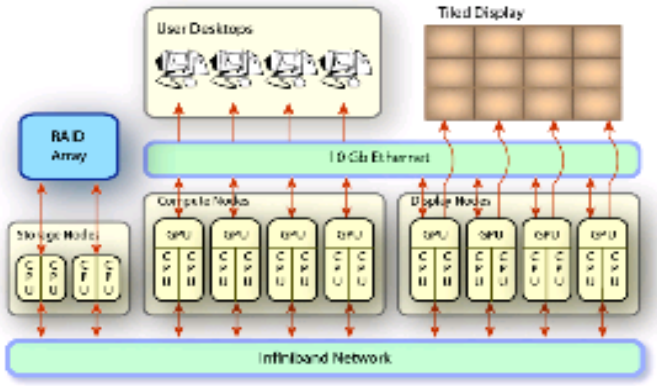


図 5:Maryland 大学の分散可視化法[4]
Fig5:Way of Maryland University's visualization[4]

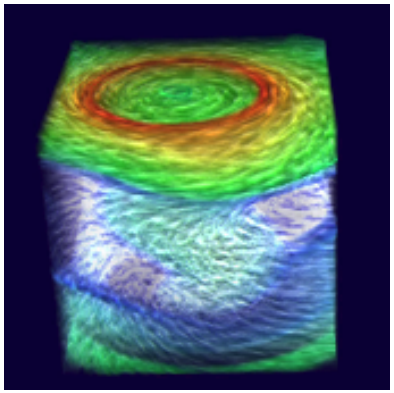


図 6:Volume Graphics クラスタ[5]
Fig6:Sample of VG cluater[5]

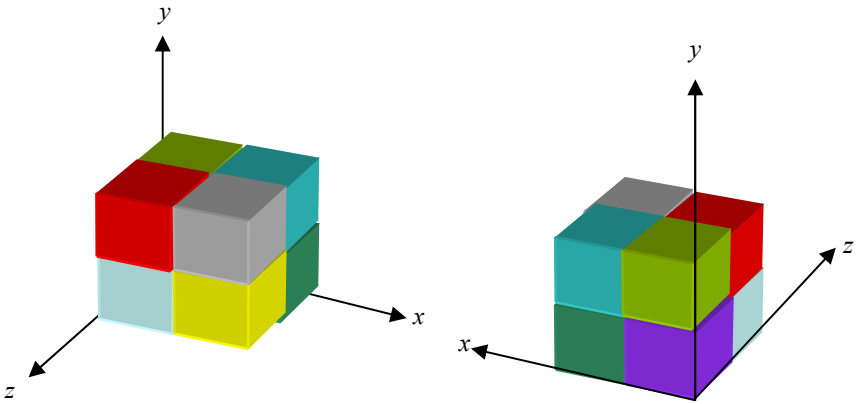


図 7:視点の違いによる可視化画像の重ね合わせ方
Fig7:How to make up the visualized images by difference of view

3.提案手法と注意点

3.1 提案手法の特徴

今回は坂下の研究を拡張して 3 次元シミュレーションの計算・可視化を分散処理させる手法を提案する．2 次元の場合は各ノードから受け取ったデータを並べるだけでよかったが，3 次元の場合は奥行きも加わるので奥側と手前側の領域の画像の重ねあわせが生じる．例えば,図 4(a)では一番手前にある領域が，図 4(b)では一番奥となっている．このように視点の位置によって奥側と手前側となる領域が異なるので画像を重ね合わせる順番を考慮しなければならない．この点が坂下の研究との主な違いである．

また，3 次元ではシミュレーションの理解に視点の位置も重要となるため，各ノードはサーバから視点位置の情報を受け取って，インタラクティブに変化する視点に合わせた画像を作る必要がある．

サーバで行われる画像の合成に関しては，画像データの余白の部分に対して透過処理を施すことで図 6 のように奥の領域の画像に描かれているものもきちんと見えるようにした．

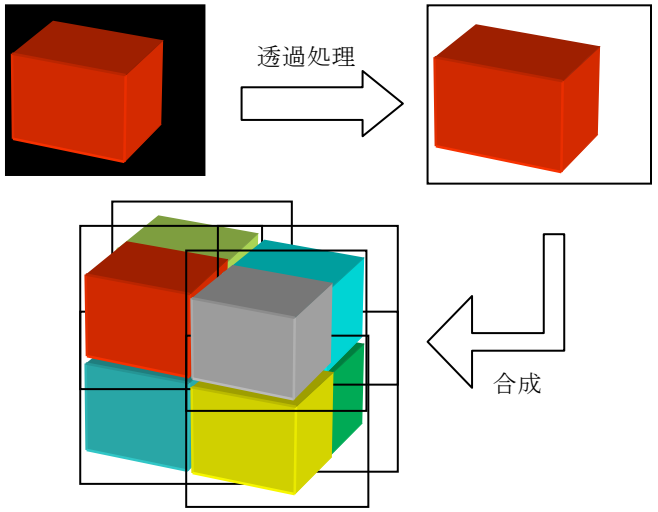


図 8:可視化画像の合成
Fig8:Synthesize the images

3.2 分散処理によるデータ量の検証

本来は大規模なクラスタを想定しているが，今回は小規模なクラスタで実験を行い評価したいのでシミュレーションサイズを 8 分割することとする．このとき分散させる処理を，

・各ノードはシミュレーションだけを計算し，結果となる数値データをサーバへと返す．サーバは受け取ったデータを逐次可視化し，合成する．
・各ノードはシミュレーションの計算と可視化を行い，結果となる画像データをサーバへと返す．サーバは受け取ったデータを合成する．

の 2 通りの方法で考えてみる．ここでシミュレーションの計算サイズは 512*512*512，画像サイズは 256*256 とし，浮動小数点の単精度を使用，要素は 3 次元ベクトルが 1 つとスカラー値が 2 つの計 5 要素としてサーバが受け取るデータ量(ネットワーク内を流れるデータ量)を考えてみると，

1 つ目の方法は

512*512*512*4[Byte]*5(要素) = 2.5[GB]

であるのに対し，2 つ目の方法は，

256*256*4[Byte]*8(台) = 2[MB]

である．画像にしてしまえば要素はいくつあっても関係ないこと、また画像データに浮動小数点の単精度を用いなくてもよいと考えるとデータ量はもっと少なくなり，圧倒的に 2 つ目の方法の方が通信にかかる時間が短いことが分かる．よって本研究では 2 つ目の方法で分散処理を行うこととする．

3.3 プログラムの分散処理化

上記の検証の通り，シミュレーションの計算から可視化までを各ノードに行わせたいので，サーバは最初に各ノードに担当する領域を知らせる．また、パーティクルを用いた可視化を行う際に必要なパーティクルの初期位置や再生位置の情報を格納したデータを各ノードに配布する．ここまでが前準備である．

サーバは可視化に関するパラメータ(ユーザからの入力による可視化方法のフラグメント，視点の位置，透明度など)を各ノードへ送り，各ノードはそれを元に担当する領域を計算・可視化し,レンダリングした画像をサーバへと送る．サーバは受け取った画像を合成し，最終的な 1 枚の絵を作り上げる．これを繰り返し行っていく(図 9).

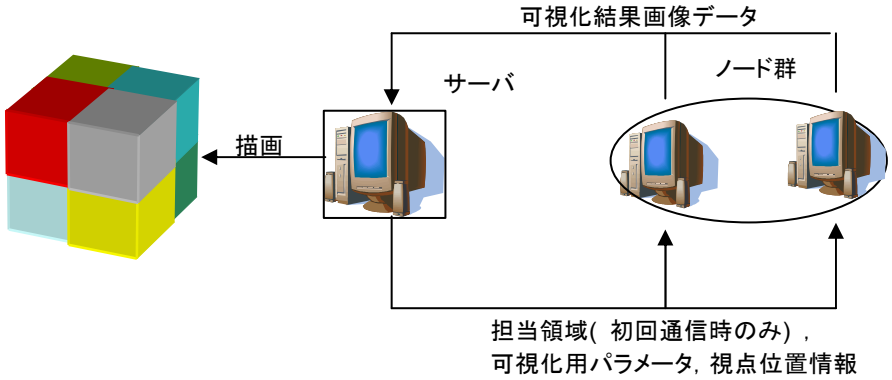


図 9:分散処理の流れ
Fig9:Way of my distributed visualization

実装環境については，通信処理ライブラリはマシン間で簡単にメッセージの授受ができるライブラリインタフェースである OpenMPI(Message-Passing Interface)を用いた[6]．また，Linux のディストリビューションは Fedora10 で，OpenGL と GLSL を用いた．

3.4 分散処理時の注意点

分散可視化を行う際の問題点はパーティクルの位置計算である．元の位置(P0)からパーティクルの移動先の位置もしくは生存時間が過ぎてしまった等の理由でパーティクルが再生される時の位置(P1)を求めた際，が自ノードの担当する領域を出てしまう場合は，その移動先の領域を担当するノードへパーティクルの情報を渡し，引き継ぎをする必要が出てくる(図 10)．特に，今回はパーティクルの位置計算に 2 次精度を用いているので 1 回の位置計算で 2 回の通信が行われることに気をつけなければならない．

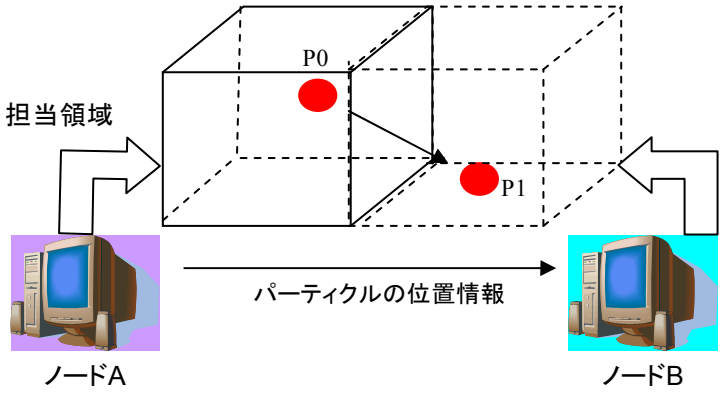


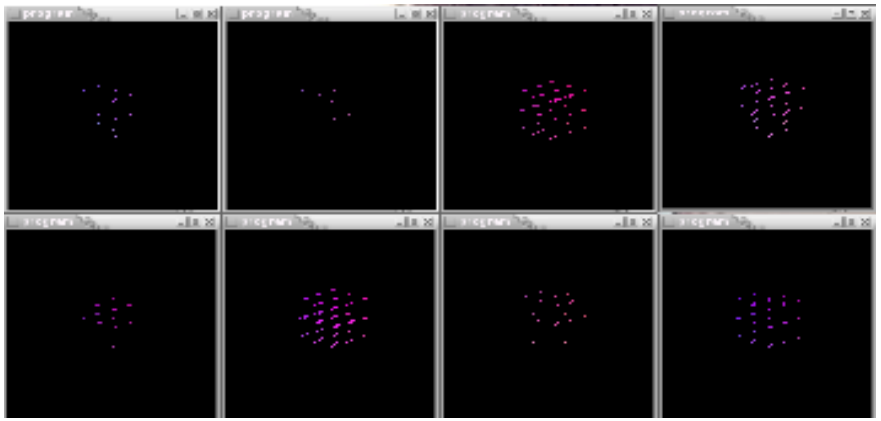
図 10:パーティクルの通信が起きるとき
Fig10:Communicate for particle moving

4.実行結果

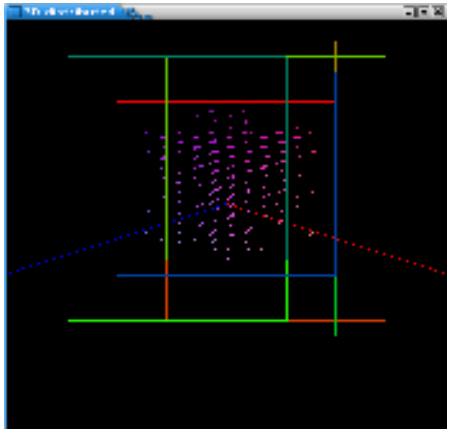
分散可視化を行った結果を以下に示す. 図 11 は矢印によるベクトル場の可視化の様子である．(a)はそれぞれのノードが作り上げた画像であり，(b)はその画像をサーバが 1 つに合成した画像である．(b)の方には 8 枚の画像がそれぞれどう重なっているかが分かるように元の画像に枠をつけておいた．これによって，きちんと透過処理が行われ，奥の領域の画像も見えることがわかる．

図 12 と図 13 はそれぞれ四角形を用いたスカラー値の可視化，パーティクルトレースによる可視化の様子である．こちらは奥側から描画していくことで整合性の取れた結果を描画できていることが分かる．また，領域を超えてパーティクルが移動する様子が見られるので，領域を超えたパーティクルについてのノード間の情報のやり取りもできていることが確認できる．速度については比較的リアルタイムでの可視化が行われていることが確認できた．

実行環境としては，各マシンが約 900MB のメモリを搭載した GPU を使用し，8 台のノードと 1 台のサーバをギガビットイーサネットで繋いだ環境を用意した．



(a)各ノードが描画した画像
(a)Image drawn by each node



(b)サーバが 1 枚の画像に合成した様子
(b)Image synthesized by server

図 11:矢印を用いたベクトル場の可視化の様子
Fig11:Image of visualization with arrows for flowing

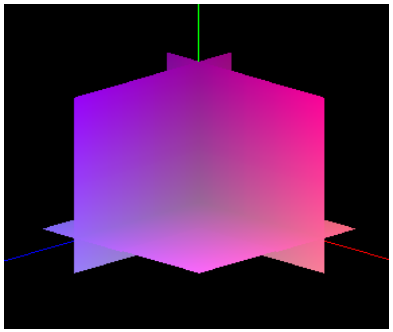


図 12:スカラー値の可視化
Fig12:Visualization of scalar value

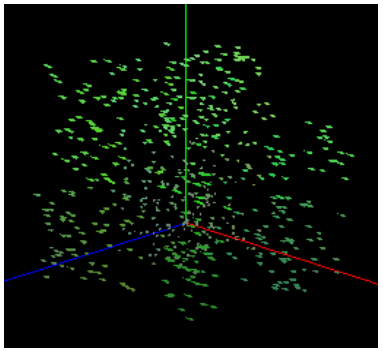


図 13:ベクトル場の可視化
Fig13::Visualization of flowing

5.まとめと今後の課題

3 次元数値シミュレーションの GPU クラスタによる分散可視化を行った．視点位置によって奥側と手前側が変化する，ノードから送られてきた画像を合成する際の透過処理, といった 3 次元特有の奥行きを考慮した画像の重ねあわせを可能とした．また，各ノードに分散させる処理の内容を検証し，シミュレーションの計算と可視化を分散させることとした．パーティクルを用いた可視化を行う際には，パーティクルの移動先位置が担当領域外に出てしまったときに通信処理を行うことも可能となった．

課題としては可視化方法を豊富にすることが挙げられる．今回の可視化手法以外にも等値面や ISOSurface(図 14)などによる可視化手法があり，それらを取り入れて可視化手法を豊富にしていくことはユーザがシミュレーションを直感的に理解しやすくするという点で重要である．

また，シミュレーションサイズが大きくなればサイズの分割数(クラスタの台数)も増えていく．大規模になると通信にかかるコストも大きくなる．そのためクラスタの階層化などを行って，いかに効率よくデータの通信を行うかが重要となってくる．

それに先立って，通信時間の短縮化や通信の効率を上げなければならない．マルチスレッドを用いた通信の並列化や，やりとりするデータ，特にサーバに集められる画像データなどの圧縮といったことである．サーバが各ノードから画像データをもらう部分を，マルチスレッドを利用して並列化することで分散可視化を高速化できるのでは，と考えている．

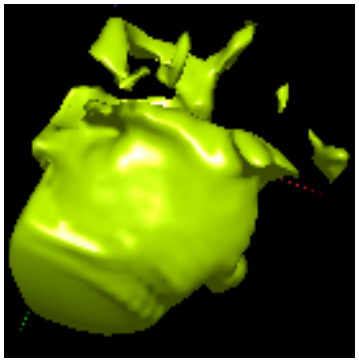


図 14:ISOSurface による可視化[1]
Fig14:Visualization with ISOSurface[1]

参考文献

[1] KGT, vGeo 大規模気象解析データ　立体可視化ソフトウェア, < http://www.kgt.co.jp/feature/vgeo/ >, (2009/7/21)
[2] 清水 政志，安藤　英俊，鳥山　孝司, GPU を用いた流体数値シミュレーションのリアルタイム可視化，可視化情報, Vol.28, pp241-246, (2008)
[3] 坂下 智也，安藤　英俊，鳥山　孝司, GPU クラスタ上での分散可視化，可視化情報,Vol.28. pp275-280, (2008)
[4] Maryland CPU-GPU Cluster Infrastructure Visual Analysis of Large-Scale Time-Varying Data <http://www.umiacs.umd.edu/research/GPU/index.html> (2009/7/21)
[5] Shigeru Muraki, Eric B. Lum, Kwan-Liu Ma, Masato Ogata, Xuezhen Liu. A PC Cluster System for Simultaneous Interactive Volumetric Modeling and Visualization, Symposium on Parallel and Large-Data Visualization and Graphics 2003, pp.95-102.
[6] The Open MPI Project, OpenMPI: Open Source High Performance Computing, OpenMPI: Open Source High Performance Computing, <http://www.open-mpi.org/>, (2009/7/21)