社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－ARC－163（7）
2005／5／31

汎用グラフィクスカードを用いた格子ボルツマン法による流体シミュレーション

小松原 誠 †　森 眞一郎 †　中島 康彦 ††　富田 眞治 †

あらまし　近年，汎用グラフィクスカードに搭載されるグラフィクスプロセッサユニット (GPU) の性能向上は目覚しく，GPU を汎用の数値計算にも応用する研究が始められている．本稿では，実時間インタラクティブシミュレーション環境の構築を目指した研究の一環として，数値流体力学の新しい手法として注目され始めている格子ボルツマン法による 2 次元円柱周りの流れの計算を GPU 上に実装した報告を行う．本報告の実装では, 同様の計算を CPU 上で実行した場合に比べ 10 倍以上の高速化が実現できたが, 一方で計算結果が両者で必ずしも一致しないことも確認した.

Implementation of Fluid Flow Simulation by Lattice Boltzmann Method onto GPU

Makoto KOMATSUBARA †　Shin-ichiro MORI †
Yasuhiko NAKASHIMA ††　Shinji TOMITA †

Abstract　The rapid improvement of both the performance and the programmability of graphics processor units(GPUs) makes it possible to perform non-graphics, general-purpose computations on the GPU. As a part of the prototype of real-time interactive simulation system, we have implemented a fluid flow simulation program onto the GPU based on the Lattice Boltzmann Method. This implementation results are described in this paper. The experimental results based on the current implementation show that the GPU outperforms more than ten times as the CPU. On the other hand, this paper also reports that the simulation results may not always the same between the simulation on GPU and that on CPU.

1　はじめに

近年の計算機性能の急速な向上に伴い，インタラクティブな実時間数値シミュレーションへの期待が高まっている．ここでは，オペレータによるシミュレーション対象へのインタラクティブな操作に対応して実時間でシミュレーションを行うとともに，即刻その結果を視覚その他の手段により提示することが求められる．このような環境では，シミュレーション対象に操作が加えられた際の動的な境界条件の変更に高速かつ柔軟に対応できる数値計算モデルが必要不可欠である．

このように動的な境界条件の変動や複雑な境界条件への対応が比較的容易で，かつ並列化による高速化が容易な数値流体計算モデルとして，格子ボルツマン法 [1, 2, 3] が注目を集めている．

本稿では，実時間インタラクティブシミュレーション環境の構築に向けた研究の一環として，格子ボルツマン法による 2 次元円柱周りの流れの数値計算処理と計算結果の実時間可視化処理を同一 GPU 上へ

† 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University
†† 京都大学大学院経済学研究科/JST
Graduate School of Economics, Kyoto University/JST

実装した結果の報告を行う．

以下，2 章では研究の背景として GPU 上での数値計算 ならびに流体シミュレーションにおける格子ボルツマン法の位置づけについて概観する．3 章で本研究で使用した格子 BGK モデルについて説明したのち，4 章で格子 BGK モデルの GPU への実装について述べる．その後，5 章で実装結果について報告したのち，6 章で実時間インタラクティブシミュレーションへの応用について検討を行ない，7 章でまとめる．

2 研究の背景

2.1 GPU を用いた汎用数値計算

表 1 は，Pentium4 と GeForce6800Ultra の Chip レベルの仕様を比較したものである．GeForce6800Ultra では，トランジスタ数が Pentium4 の 2 倍弱となるとともに，ピクセル計算時の理論最大性能は Pentium4 を大きく上回る．図 1 に，GPU における基本的なレンダリング処理の概念図を示す．GPU は，頂点プロセッサとフラグメントプロセッサを持つ．CPU からデータが送られると，まず頂点プロセッサで頂点座標が変換され，次にラスタライズ処理がなされ，フラグメントプロセッサでテクスチャの貼り付けやピクセル処理が行われ，フレームバッファに格納される．テクスチャデータとして 1 ピクセルあたり RGBα の 4 つの値を持つことができ，フラグメントプロセッサでは，RGBα の 4 つの値を同時に計算することができる．近年の GPU では，両プロセッサのレンダリングパイプラインをプログラムにより動的に制御することが可能となり，32bit 浮動小数点数を扱うことも可能になった．また，Cg[4] に代表される GPU プログラミングのための高級言語が提案されており，プログラム開発効率が改善されている．このような背景から，GPU で汎用数値計算を行う試みが始められている [5, 6, 7].

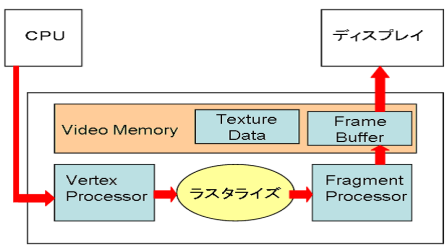


図 1: レンダリング処理の概念図

表 1: Chip 諸元比較

	Pentium4	GeForce 6800Ultra
Core Name	Prescott	NV40
Core Clock	3.6GHz	400MHz
Process	90nm low-k	0.13μm 銅
Transistor	125M	222M
Cache Size	1MB	非公開 (+外部 cache)
Die	$112mm^2$	$300mm^2$

2.2 GPU での数値計算手法

GPU での数値計算は，主にフラグメントプロセッサのテクスチャマッピング機能を利用することで行われている．2 次元配列をテクスチャとして与えれば，配列の要素毎の演算が可能になる．前述のように，1 ピクセルの RGBα に相当する値を同時に計算することができるため，最大 4 倍の性能を発揮することができる．通常フラグメントプロセッサでの演算結果はディスプレイに表示するためのフレームバッファへ出力されるが，フレームバッファは 1 ピクセルの RGBα それぞれにつき 8bit の情報しか持つことができない．一方，フレームバッファとは別に，ビデオメモリ内にピクセルバッファと呼ばれるオフスクリーンバッファを持つことができる [8]．ピクセルバッファは 1 ピクセルの RGBα それぞれにつき 32bit の情報を持つことができるため，フラグメントプロセッサの出力先としてピクセルバッファを選択することにより，計算の精度を保つことができる．ピクセルバッファの内容をテクスチャに上書きすることで反復計算が可能となる．

2.3 格子ボルツマン法

流体の運動を解析する代表的な手法には，a) 連続体としての流体の運動方程式をたて微分方程式を解く方法と，b) 流体を粒子の流れとして捉えて個々の粒子の運動を解析し，それを巨視的に捉えることで流体の運動を解析する方法がある．よく知られている差分法，境界要素法，有限要素法等を用いてナビエ・ストークス方程式を離散化して解く手法は前者に相当する．本論文で扱う格子ボルツマン法は後者

に属する．

格子ボルツマン法では，流れ場を規則的な格子で切り*, 仮想的な粒子を格子に沿って運動させることで流れをモデル化する．この時，格子上での移動 (並進) と衝突の特性を決定するボルツマン方程式の衝突演算子を適切に設定することで，巨視的な流れ場に対するナビエ・ストークス方程式に対応する微視的粒子運動を規定する．

計算スキームとしては，格子ボルツマン法はセルオートマトンの一種であり陽的な時間発展型スキームである．セルの状態，すなわち，格子点上の粒子状態として連続状態をとるとともに，粒子の衝突確率を衝突する粒子の数密度 (分布関数) の積として表現するという点が格子ボルツマン法の重要な特徴である．

3 格子 BGK モデルを用いた流体計算

3.1 空間の離散化と分布関数

格子ボルツマン法では，空間を原則として均等な格子に離散化する．本研究では，2 次元空間において，静止状態を含めて 9 つの移動方向を考える 2d9V モデルを用いた (図 2,3)．

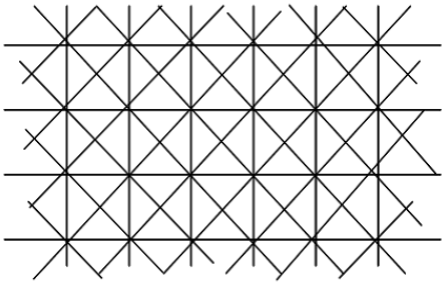


図 2: 空間の離散化

図 2 の格子において，4 本の直線の交点を格子点とする．まず，2D9V モデルにおける仮想粒子の速度ベクトル $e_i(i = 0 \sim 8)$ を図 3 に示すように定義する．速度ベクトル e_i の方向に移動する仮想粒子の密度分布は，分布関数 $f_i(i = 0 \sim 8)$ で表す．f_i の初期値は，後述する単一時間緩和係数 f_i^{eq} を用いて $f_i = f_i^{eq}$ として与える．

* 最近は不規則格子への適用も研究されている [9].

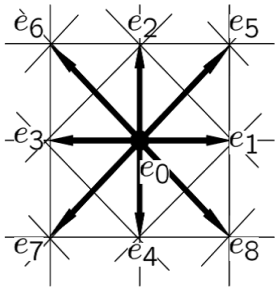


図 3: 仮想粒子の速度ベクトル e_i

各格子点において，以下で説明する衝突，並進，巨視化の過程を繰り返すことで流体シミュレーションが進行する (図 4)．

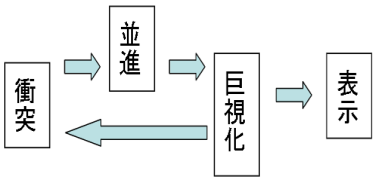


図 4: 格子ボルツマン法の計算手順

3.2 衝突

本研究では，衝突モデルとして格子ボルツマン法を用いた流体シミュレーションに関する多くの文献で用いられる格子 BGK モデルを採用する．このモデルでは簡素化された衝突モデルを採用し，衝突を次式でモデル化する．衝突は，次式で表される．

$$f_i' = f_i - \frac{1}{\phi}\left(f_i - f_i^{eq}\right) \quad (i = 0, 1, \cdots, 8) \qquad (1)$$

ここで，f_i は衝突前の分布関数，f_i' は衝突後の分布関数である．ϕ は単一時間緩和係数と呼ばれる定数で，動粘性係数を ν としたとき，次式で与えられる．

$$\phi = 3\nu + \frac{1}{2} \qquad (2)$$

f_i^{eq} は局所平衡分布関数と呼ばれる．格子点における密度を ρ，流速ベクトルを V とすれば，f_i^{eq} は次式で与えられる．

$$f_i^{eq} = \omega_i \rho \left\{1 + 3e_i \cdot V + \frac{9}{2}(e_i \cdot V)^2 - \frac{3}{2}V^2\right\} \qquad (3)$$

ここで $\omega_0 = \frac{4}{9}$，$\omega_{1\sim4} = \frac{1}{9}$，$\omega_{5\sim8} = \frac{1}{36}$ である．

3.3 並進および境界条件

並進では，衝突によって更新された分布関数 f_i に比例する数の粒子が e_i 方向の隣接格子点へ移動すると考える．ただし，移動方向に隣接する格子点が固体壁 (本稿では固体格子点と呼ぶ) に相当する場合，180 度向きを変えるものとする (滑り無し断熱固体壁と仮定した bounce-back 境界条件)．

3.4 巨視化

境界条件を考慮した並進の過程を経た分布関数を用いて，各格子点での巨視的変数を求める．密度 ρ，流速ベクトル V は，以下のように求められる．

$$\rho = \sum_{i=0}^{8} f_i \tag{4}$$

$$\rho V = \sum_{i=0}^{8} f_i e_i \tag{5}$$

衝突，並進，巨視化の一連の過程を 1 タイムステップとし，これを繰り返す．

4 格子 BGK モデルの GPU 実装

格子ボルツマン法では実数を扱うため，計算を繰り返すに従って誤差が蓄積する問題がある．本研究では，密度の初期値を ρ_0 とし，式 (3,4) の代わりに次の式 (6,7) を用い，誤差の低減を目指した [10]．

$$f_i^{eq} = \omega_i \left[(\rho - \rho_0) + \rho \left\{ 3e_i{\cdot}V + \frac{9}{2}(e_i{\cdot}V)^2 - \frac{3}{2}V^2 \right\} \right] \tag{6}$$

$$\rho = \sum_{i=0}^{8} f_i + \rho_o \tag{7}$$

データの保持には格子数と同じ 256×128 の大きさの 32bit 浮動小数点テクスチャを 4 枚用いた．それらを tex1〜4 と呼ぶこととする．格子点をテクスチャのピクセルに対応させ，各ピクセルにおいて，tex1 には $f_1 \sim f_4$，tex2 には $f_5 \sim f_8$，巨視的な流速ベクトル成分を (V_x, V_y) としたとき，tex3 には V_x, V_y, f_0, ρ，tex4 には固体格子点であるか流体格子点であるかの情報を格納した．フラグメントプロセッサでの計算の際，これらのテクスチャを参照することができる．

本研究では，GPU のフラグメントプロセッサを制御するためのフラグメントプログラムにより，格子ボルツマン法の計算を実装した．Cg による 6 つのフラグメントプログラムを用いた．それらを fp1〜fp6 と呼ぶこととする．fp1 では $f_1 \sim f_4$ の衝突，fp2 では $f_5 \sim f_8$ の衝突，fp3 では $f_1 \sim f_4$ の並進，fp4 では $f_5 \sim f_8$ の並進，fp5 では f_0 の衝突と巨視化，fp6 では流速の大きさから色への変換を行った．fp1 を図 5 に示す．フラグメントプログラムはフラグメント毎に適用される．fp1 では，フラグメント毎に異なるテクスチャ座標と，全てのフラグメントで共通の tex1〜4，FAI(ϕ)，ROU(ρ_0) を入力としている．流体格子点では衝突を行い，固体格子点では bounce-back 条件に基づき，速度を反転させている．最大 4 つのベクトル成分は，演算子 (.) と x，y，z，w でコストをかけずに再配置できる．出力は更新された $f_i \sim f_4$ である．

fp1〜fp5 の出力先はピクセルバッファとし，テクスチャへの上書きによりテクスチャを更新する．fp6 の出力先はフレームバッファとする．

5 実験結果

5.1 計算条件

2 次元円柱周りの流れの計算を行った．格子数は 256×128 とした．左上を $(0, 0)$，右下を $(255, 127)$ としたとき，円柱の中心を $(64, 64)$ とし，円柱の中心から距離 8 未満の格子点を固体格子点とすることで円柱を近似した．左端の流入境界は速度分布固定，右端は自由流出，上下は周期境界条件とした．100000 タイムステップの計算を行い，1 タイムステップ毎に流速の大きさを色として表示した．

5.2 実装環境

CPU は Pentium4 3.2GHz，OS は WindowsXP，GPU は GeForce 6800Ultra を用いた．プログラミング言語は C，OpenGL，Cg を用いた．

5.3 計算結果

100000 タイムステップ後の計算結果を図 6 に示す．一般に知られるとおり，動粘性係数が 0.0032 の場合，上下対称の流れとなり (図 6A)，動粘性係数が 0.0008，

```
//fp1
struct fpIN{
float4 texCoord :TEX0;
};
struct fpOUT{
float4 col :COLOR;
};

fpOUT main( fpIN IN,
uniform samplerRECT tex1,
uniform samplerRECT tex3,
uniform samplerRECT tex4,
uniform float FAI,
uniform float ROU)
{
fpOUT OUT;
float4 V=f4texRECT(tex3,IN.texCoord.xy);
float kabe=f4texRECT(tex4,IN.texCoord.xy).x;
float4 f=f4texRECT(tex1,IN.texCoord.xy);
float4 fe,fee;
int4 e;
float V2;

if(kabe<0.5f){
V2=V.x*V.x+V.y*V.y;
e=int4(1,0,-1,0);
fee=e*V.x+(e.yxwz)*V.y;
fe=(V.w-ROU+V.w*(3.0f*fee+4.5f*fee*fee
-1.5f*V2))/9.0f;
OUT.col=f-(f-fe)/FAI;
}
else OUT.col.xyzw=f.zwxy;
return OUT;
}
```

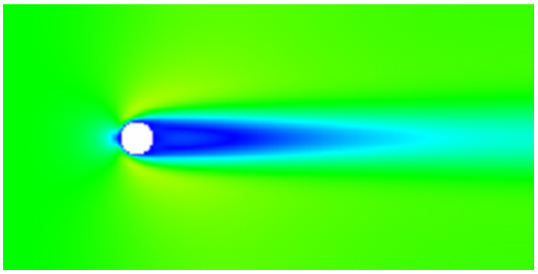
図 5: フラグメントプログラム fp1

0.0002 の場合，円柱の上下からカルマン渦が観察された (図 6B,C)．動粘性係数を 0.0002 とし，同様の計算を CPU でも行い，比較を行った．CPU(Pentium4, UltraSPARC-III) での計算結果を図 7 に示す．GPU と同じく，単精度で計算を行ったが，これらの計算結果は完全には一致しなかった．

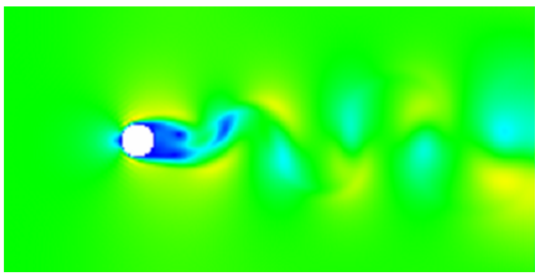
格子数を 512 × 256 にした場合とあわせて，10000 タイムステップの実行時間を表 2 に示す．CPU で実行した場合に比べ，GPU を用いることにより，10 倍以上高速に実行することができた．

表 2: 実行時間

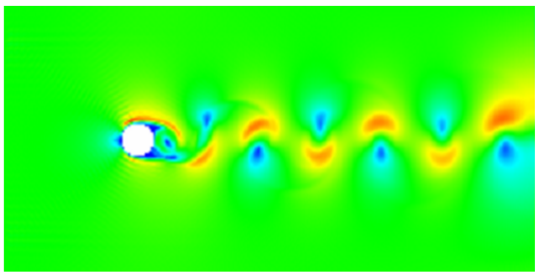
格子数	実行時間 (sec)	
	CPU 実行時間	GPU 実行時間
256 × 128	203	17
512 × 256	803	69



A:動粘性係数0.0032 (Re50)



B:動粘性係数0.0008 (Re200)



C:動粘性係数0.0002 (Re800)

図 6: GPU での計算結果

6 実時間インタラクティブシミュレーションシステムの構想

大規模な数値シミュレーションを高精度かつインタラクティブに実行するため，我々は，シミュレーションを行う PC クラスタベースの計算サーバと，オペレータの操作や結果の呈示を行うユーザインタフェース端末としてのクライアントを用いたサーバ-クライアント型の超高速体感型シミュレーションシステムの研究を行っている [11, 12]．ここでは，計算サーバを煩雑な入出力処理から解放するとともに，柔軟なユーザインタフェース環境の実現を目指しており，計算サーバのスループットの限界による遅延やサーバ-クライアント間の通信遅延を隠蔽する目的で，クラ

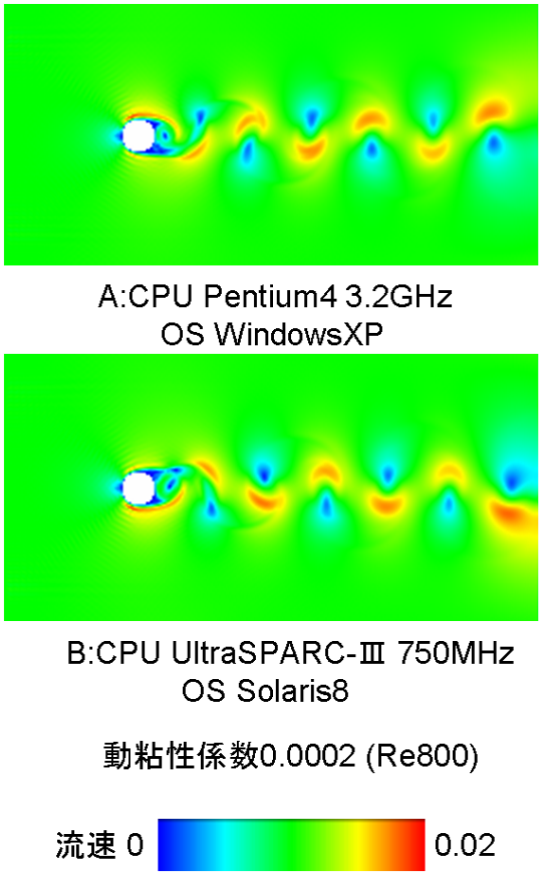


図 7: CPU での計算結果

イアント側でも簡易シミュレーションを行うことで (これをシミュレーションキャシングと呼ぶ)，インタラクティブ性の向上を図る．

GPU を数値計算に用いる場合，精度の限界が問題となるが，サーバ側での計算との相互補完により，実時間性と計算精度の両立を図る手法を検討している．

7　まとめ

実時間インタラクティブシミュレーション環境の構築を目指した研究の一環として，格子ボルツマン法による 2 次元円柱周りの流れの計算を GPU 上に実装した．同様の計算を CPU 上に実装した場合と比べ，GPU を用いることで 10 倍以上高速に実行することができたが，両者の計算結果が必ずしも一致しないことも確認した．

謝辞

本研究の一部は，日本学術振興会科学研究費補助金基盤研究 S (課題番号 16100001)，21 世紀 COE プログラム (課題番号 14213201)，ならびに，文部科学省特定領域研究 S (課題番号 13224050) による．

参考文献

[1] McNmara, G. and Zanetti, G.: Use of the Boltzmann equation to simulate lattice-gas automata, *Phys. Rev. Lett.*, Vol. 61, Issue 11, pp. 2332–2335 (1988)

[2] Chen, S. and Doolen, G.D.: Lattice Boltzmann Method for Fluid Flows, *Annu. Rev. Fluid Mech.*, Vol. 30, pp. 329–364 (1998)

[3] 蔦原 道久, 高田 尚樹, 片岡 武: 格子気体法・格子ボルツマン法, コロナ社 (1999)

[4] NVIDIA Corporation, Cg Toolkit User's Manual Release 1.2 (2004)

[5] General Purpose Computation on Graphics Processor ホームページ, http://www.gpgpu.org/

[6] 森 眞一郎, 篠本 雄基, 五島 正裕, 中島 康彦, 富田 眞治: 汎用グラフィクスカード上での簡易シミュレーションと可視化, 電子情報通信学会信学技報, CPSY2004-24, pp.25-30 (2004)

[7] W. Li, X. Wei, and A. Kaufman: Implementing lattice Boltzmann computation on graphics hardware, *Visual Computer 19(7-8)*, pp. 444-456 (2003)

[8] M. J. Kilgard: NVIDIA OpenGL Extension Specifications, NVIDIA Corporation (2004)

[9] 小沢 拓, 棚橋 隆彦: 二相系格子ボルツマン法の非構造格子への適用, 計算工学会論文集, No. 20050006 (2005)

[10] P. A. Skordos: Initial and boundary conditions for the lattice Boltzmann method, *Phys. Rev.* E48, pp.4823-4842 (1993)

[11] 丸山 悠樹: 実時間インタラクティブシミュレーション環境の構築，京都大学大学院情報学研究科修士論文 (2004)

[12] 富田 眞治: 超高速体感型シミュレーションシステムの研究，科学研究費補助金，平成 16 年度基盤研究 (S) 新規採択課題の概要について，日本学術振興会，p.10 (2004)