GPUを用いたマルチグリッド法の実装

M2009MM006 池田達哉
指導教員：杉浦洋

1　はじめに

近年，大規模連立方程式の解法としてマルチグリッド法が注目されている．マルチグリッド法は収束性がメッシュサイズに依存しない，並列化しやすいという特徴がある．解析の規模が大きくなればなるほど非常に有力な解法となる可能性がある．また，GPU（Graphics Processing Unit）を用いて汎用計算を行う様々な研究がされている[1][2]．GPU はグラフィックカードとも呼ばれ画像処理に特化したハードウェアであるため並列処理能力が高い．

本研究ではマルチグリッド法を GPU 上で実装するためのアルゴリズム示し，NVIDIA 社の GPU である GTX480 を用いてマルチグリッド法の高速化を目指す．

2　ポアソン方程式

領域 $D=[0,1]\times[0,1]$ の 2 次元ポアソン方程式のディリクレ型境界値問題

$$\begin{cases} u_{xx}(x,y)+u_{yy}(x,y)=f(x,y), & (x,y)\in D, \\ u(x,y)=g(x,y), & (x,y)\in\partial D \end{cases} \tag{1}$$

の数値解法を考える．x 方向 y 方向に $n+1$ 等分した格子点

$$(x_i,y_j)=(ih,jh)\ \ (0\le i\le n+1, 0\le j\le n+1) \tag{2}$$

上で方程式を離散化する．$h=\frac{1}{n+1}$ である．格子点での解の値を $u_{ij}=u(x_i,y_j)$ とする．中心点 (ih,jh) からの距離が $\sqrt{m}h(m=1,2)$ である格子点における u の和を $\sum_{\sqrt{m}}u$ という記法で表す．すなわち，

$$\begin{aligned} \sum\nolimits_1 u &= \sum\nolimits_{\pm} u(ih\pm h, jh)+\sum\nolimits_{\pm} u(ih, jh\pm h), \\ \sum\nolimits_{\sqrt{2}} u &= \sum\nolimits_{\pm} u(ih\pm h, jh\pm h) \end{aligned} \tag{3}$$

と定義する．ここで，記法 $\sum_{\pm}$ は式中に現れる記号 $\pm$ 全てについて，符号 $+$，$-$ を独立に選んだもの全体の和を意味する．また，ポアソン方程式の右辺項 f についてもこの記法を使用する．

境界点での値は境界条件より与えられる．内点の値はポアソン方程式を 5 点差分で離散化した 2 次精度の線形方程式

$$-4u_{ij}+\sum\nolimits_1 u=h^2 f_{ij} \tag{4}$$

もしくは，9 点差分で離散化した 4 次精度の線形方程式

$$-20u_{ij}+4\sum\nolimits_1 u+\sum\nolimits_{\sqrt{2}} u=h^2\left(\frac{1}{2}\sum\nolimits_1 f+4f_{ij}\right) \tag{5}$$

を解いて求める [3]．これは係数行列が疎な大規模線形方程式となり，反復解法がよく用いられる．式 (4)，式 (5) の線形方程式を簡単に $A_h u_h=f_h$ と表す．

領域 $D=[0,1]\times[0,1]\times[0,1]$ の 3 次元ポアソン方程式のディリクレ型境界値問題

$$\begin{cases} u_{xx}(x,y,z)+u_{yy}(x,y,z)+u_{zz}(x,y,z)=f(x,y,z), \\ \qquad\qquad (x,y,z)\in D, \\ u(x,y,z)=g(x,y,z), \quad (x,y,z)\in\partial D \end{cases} \tag{6}$$

の数値解法を考える．x 方向 y 方向 z 方向に $n+1$ 等分した格子点

$$\begin{aligned} &(x_i,y_j,z_k)=(ih,jh,kh) \\ &\quad (0\le i\le n+1, 0\le j\le n+1, 0\le k\le n+1) \end{aligned} \tag{7}$$

上で方程式を離散化する．$h=\frac{1}{n+1}$である．格子点での解の値を $u_{ijk}=u(x_i,y_j,z_k)$ とする．2 次元の場合と同様に，中心点 (ih,jh,kh) からの距離が $\sqrt{m}h(m=\frac{1}{2},1,2,3)$ である格子点における u の和を $\sum_{\sqrt{m}}u$ という記法で表す．また，$\sum_1$ の h を $\frac{h}{2}$ で置換えたものを $\sum_{\frac{1}{2}}$ と書く．

境界点での値は境界条件より与えられる．内点の値はポアソン方程式を 7 点差分で離散化した 2 次精度の線形方程式

$$-6u_{ijk}+\sum\nolimits_1 u=h^2 f_{ijk}, \tag{8}$$

15 点差分で離散化した 4 次精度の方程式

$$-56u_{ijk}+8\sum\nolimits_1 u+\sum\nolimits_{\sqrt{3}} u=h^2(6f_{ijk}+\sum\nolimits_1 f), \tag{9}$$

19 点差分で離散化した 4 次精度の方程式

$$-24u_{ijk}+2\sum\nolimits_1 u+\sum\nolimits_{\sqrt{2}} u=h^2\left(3f_{ijk}+\frac{1}{2}\sum\nolimits_1 f\right), \tag{10}$$

27 点差分で離散化した 6 次精度の方程式

$$\begin{aligned} &-128u_{ijk}+14\sum\nolimits_1 u+3\sum\nolimits_{\sqrt{2}} u+\sum\nolimits_{\sqrt{3}} u \\ &=h^2\left(-17f_{ijk}-\frac{5}{6}\sum\nolimits_1 f+\frac{1}{3}\sum\nolimits_{\sqrt{2}} f+8\sum\nolimits_{\frac{1}{2}} f\right) \end{aligned} \tag{11}$$

を解いて求める [4].

2.1　SOR 法

ポアソン方程式を解く代表的な反復解法に SOR(Successive Over Relaxation) 法がある．2 次元の場合，初期近似 $u_{ij}\cong u(x_i,y_j)$ を設定し，緩和 (SOR 反復) 計算を繰り返し，解に収束させる．緩和は定めた全順序に従い全ての $u_{i,j}(1\le i\le n, 1\le j\le n)$ を更新する計算である．2 次元 5 点差分法 (4) においては以下の式を用いる．

$$u_{ij}\leftarrow u_{ij}+\omega\left(\left(\sum\nolimits_1 u-f_{ij}\right)/4-u_{ij}\right) \tag{12}$$

ここで $0\le\omega\le 2$ は緩和パラメータと呼ばれ，適切に設定することにより収束速度が向上する．特に $\omega=1$ の反復法をガウス・ザイデル法と呼ぶ．2 次元 9 点差分法，3 次元 7 点，15 点，19 点，27 点差分法においても同様に緩和計算を設計する．

3　マルチグリッド法

SOR 法やガウス・ザイデル法では格子サイズと同じ空間波長を持った誤差が効率よく減衰するのに対し，長波長の誤差が減衰しにくいという特性は良く知られている．マルチグリッド法は密格子と疎格子を用意し，SOR 法やガウス・ザイデル法を用いて各格子サイズの緩和計算を行ない各波長の誤差を効率的に減衰させる手法である．疎格子とはある格子の格子点の部分集合からなる格子である．ここで，2 次元の 1 段下の疎格子を格子点間隔が x 方向 y 方向とも 2 倍にして構成する．3 次元の場合も x 方向 y 方向 z 方向の格子点間隔を 2 倍にする．

2 グリッド法

疎密 2 段格子を使用したマルチグリッド法を示す．$u_h^{(0)}$ を初期値とし，2 グリッド法で k 回反復した結果を $u_h^{(k)}$ と書く．

1. 緩和計算：$u_h^{(k)}$ を l 回緩和した値 v_h を得る (l は定数)
2. 残差計算：$r_h = f_h - A_h v_h$
3. 制限補間：$s_{2h} = Rr_h$(疎格子点上での残差を計算)
4. 疎格子上の修正量：$c_{2h} = A_{2h}^{-1} s_{2h}$ $(A_{2h}c_{2h} = s_{2h})$
5. 延長補間：$d_h = Pc_{2h}$(密格子点上の修正量を補間)
6. v_h の修正：$u_h^{(k+1)} = v_h + d_h$

以上の過程を残差が基準値以下になるまで繰り返す．

2 グリッド法の過程 4 に 2 グリッド法を再帰的に用いて多段格子の V サイクルアルゴリズムを構成する (図 1)．

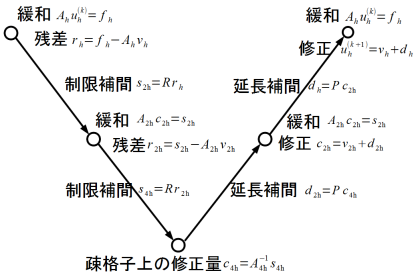

図 1　V サイクルアルゴリズム

4　GPU

GPU は型番によってアーキテクチャが異なってくる．以下に，本研究で使用した GeForce GTX480 の Fermi(フェルミ) アーキテクチャについて説明する．

GPU は 4 基の GPC(Graphics Processing Cluster) で構成される．GPC は 4 基の SM(Streaming MultiProcessor) で構成される．SM は基本演算プロセッサである CUDA Core が 32 基，sin や cos といった基本関数の計算を行う SFU(Special Function Unit) が 4 基，64KB の共有メモリ/L1 キャッシュ，128KB のレジスタ，ロード/ストアユニットが 16 基，Warp スケジューラと命令発行ユニットが 2 基づつの周辺ユニット群で構成される．GTX480 では計 16 基の SM がありそのうちの 1 基の SM が無効化されているため，CUDA Core 数の合計は 480 基となる．

CUDA Core はスカラプロセッサであり 1 基で 32 ビット浮動小数点スカラ演算器と 32 ビット整数スカラ演算器で構成されている．この 2 基のスカラ演算器は並列に動作できる．倍精度の演算を行う場合は CUDA Core を 2 サイクル使用することにより計算を行う．

4.1　CUDA

CUDA(Compute Unified Device Architecture) とは NVIDIA 社が GPU を汎用コンピューティングに活用するために開発した C 言語をベースとしたプログラミング環境である．

GPU 上で実行されるプログラムはカーネル関数と呼ばれる．カーネル関数の作業全体をグリッドと呼ぶ．グリッドはいくつかのブロック群からなる．作業はブロック単位で SM に割り振られる．ブロックはいくつかのスレッド群からなり，SM に属する CUDA Core により並列処理される．ブロック内のスレッド群は，共有メモリによるデータ共有，同期制御が可能である．

5　マルチカラー・オーダリング

GPU の利点は多数のスレッドによる並列計算である．緩和計算において，なるべく多くの格子点を並列に計算することが望ましい．5 点差分について考えると，計算格子点の上下左右に隣接する格子点同士は計算において相互に参照しあうため並列計算できない．同時並列に計算すると結果が不確定になるからである．全格子点を同色格子点が隣接しないように塗り分ければ，各色群ごとに並列緩和計算ができる．このように格子点を組織化する手法をマルチカラー・オーダリングと言う．

マルチカラー・オーダリングは，各座標軸方向に周期 2 を持つ彩色で構成できる．格子の色は各次元のインデックスの偶奇で決まる．以下では，基本周期である 2 次元 2×2 部分格子，3 次元 2×2×2 部分格子の彩色のみを示す．

5 点差分では図 2 の 2 色オーダリングを用いる．9 点差分では図 3 の 4 色オーダリングを用いる．

3 次元 7 点差分と 15 点差分では図 4 の 2 色オーダリングを用いる，19 点差分では図 5 の 4 色オーダリングを用いる．27 点差分では図 6 の 8 色オーダリングを用いる．

図 2　2 次元 2 色

図 3　2 次元 4 色

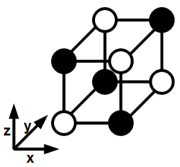

図 4　3 次元 2 色

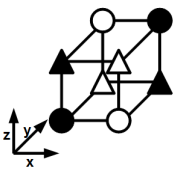

図 5　3 次元 4 色

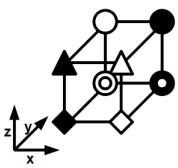

図 6　3 次元 8 色

5.1 部分格子細胞

上で示したマルチカラー・オーダリングは塗り分けの単位が格子点であるが，2 次元なら $1 \times k$ 長方形，3 次元なら $1 \times 1 \times k$ 直方体の部分格子を塗り分けの単位とすることも可能である．塗り分けの単位を細胞と呼ぶ．各細胞は 1 色で塗られ，細胞内の格子点は 1 つのスレッドで処理される．これにより，隣接格子点間で共有できるデータが増え，データアクセスの効率化ができる．2 次元 9 点差分，3 次元 19 点差分と 27 点差分ではこの技法を用いた．

6 実験結果

本研究では GTX480 を用いてマルチグリッド法 V サイクルアルゴリズムを実装した．実験では 2 次元問題 (1) の境界値関数は $g(x,y) = e^{xy}$，右辺強制項は $f(x) = (x^2 + y^2)e^{xy}$ を用いる．3 次元問題 (2) の境界値関数 $g(x,y,z) = e^x \cos y \times z^2$，右辺強制項は $f(x,y,z) = 2e^x cosy$ を用いる．緩和計算は SOR 法を用い，計算精度は単精度と倍精度の 2 通りで行う．CPU のプログラムは C 言語を用いるが並列化は行わない．計算時間は 100 回の平均をとる．速度比は計算時間で CPU/GPU で調べる．

表 1 使用するハードウェア，ソフトウェア

CPU	Core i7 860 2.8GHz
Graphic Card	GeForce GTX480 1.4GHz
OS	Fedora 12
CUDA	driver 3.20

6.1 緩和計算の計算性能

表 2 から表 7 に緩和計算の単精度と倍精度における GPU の計算時間と FLOPS を示す．2 次元の 5 点差分では単精度で最大 70.03GFLOPS，倍精度で 35.01GFLOPS を得た．倍精度と単精度を比較すると FLOPS は約 1/2 となっている．原因は，倍精度演算に CUDA Core を 2 サイクル必要とし，かつレジスタの使用量が増えるため SM 上で起動できるブロック数が減少するなどによる効率の低下である．9 点差分では計算量とデータ共有量が 5 点差分と比べて多いため単精度で 97.32GFLOPS，倍精度で 45.79GFLOPS 得られた．

3 次元単精度 7 点差分では最大 68.98GFLOPS，15 点差分では 87.96GFLOPS，19 点差分では 86.30GFLOPS，27 点差分では緩和計算の中で最大値の 104.97GFLOPS 得られた．19 点差分の FLOPS は 15 点差分よりも低くなった．これは，計算量はほぼ同じであるが，19 点差分が 4 色オーダリングであるのに対し 15 点差分が 2 色オーダリングであり同時に実行できるスレッド数が多かったため 19 点差分の FLOPS が低くなったと考えられる．倍精度では 27 点差分を除いて 2 次元と同様に約 1/2 の FLOPS 低下が見られた．27 点差分で大きな FLOPS の低下が見られたのは計算量が最も多いためデータ共有に必要な変数を多く宣言する必要があり，SM 内のレジスタでは足りず一部の変数がグローバルメモリに置かれたからである．

表 2 2 次元 5 点差分緩和計算の計算時間と FLOPS

	単精度		倍精度	
N	計算時間 (sec)	GFLOPS	計算時間 (sec)	GFLOPS
4096	0.0021470	70.03	0.0043124	35.01
2048	0.0005486	68.88	0.0011084	34.05
1024	0.0001523	61.96	0.0003066	30.77
512	0.0000489	48.24	0.0000855	27.58
256	0.0000158	37.31	0.0000304	19.40
128	0.0000124	15.54	0.0000135	10.91
64	0.0000088	4.19	0.0000098	3.76
32	0.0000088	1.02	0.0000081	1.13

表 3 2 次元 9 点差分緩和計算の計算時間と FLOPS

	単精度		倍精度	
N	計算時間 (sec)	GFLOPS	計算時間 (sec)	GFLOPS
4096	0.0024135	97.32	0.0051293	45.79
2048	0.0006679	87.91	0.0015541	37.78
1024	0.0001976	74.29	0.0004628	31.72
512	0.0000669	54.86	0.0001577	23.26
256	0.0000262	35.01	0.0000525	17.48
128	0.0000168	13.64	0.0000202	11.33
64	0.0000164	3.50	0.0000157	3.64
32	0.0000162	0.88	0.0000155	0.92

表 4 3 次元 7 点差分緩和計算の計算時間と FLOPS

	単精度		倍精度	
N	計算時間 (sec)	GFLOPS	計算時間 (sec)	GFLOPS
256	0.0026753	68.98	0.0052142	35.39
128	0.0003494	66.02	0.0006910	33.38
64	0.0000678	42.51	0.0000947	30.46
32	0.0000279	12.91	0.0000350	10.30
16	0.0000126	3.57	0.0000129	3.45
8	0.0000114	0.49	0.0000116	0.48

表 5 3 次元 15 点差分緩和計算の計算時間と FLOPS

	単精度		倍精度	
N	計算時間 (sec)	GFLOPS	計算時間 (sec)	GFLOPS
256	0.0038144	87.96	0.0070991	47.26
128	0.0004931	85.06	0.0009270	45.24
64	0.0001073	61.90	0.0001366	38.37
32	0.0000377	17.37	0.0000500	13.10
16	0.0000135	6.05	0.0000175	4.68
8	0.0000131	0.78	0.0000135	0.75

表 6 3 次元 19 点差分緩和計算の計算時間と FLOPS

	単精度		倍精度	
N	計算時間 (sec)	GFLOPS	計算時間 (sec)	GFLOPS
256	0.0046657	86.30	0.0086443	46.58
128	0.0006932	72.60	0.0011529	43.65
64	0.0001941	32.41	0.0002437	25.81
32	0.0000658	11.95	0.0001167	6.74
16	0.0000265	3.71	0.0000289	3.40
8	0.0000258	0.47	0.0000259	0.47

表 7 3 次元 27 点差分緩和計算の計算時間と FLOPS

	単精度		倍精度	
N	計算時間 (sec)	GFLOPS	計算時間 (sec)	GFLOPS
256	0.0052745	104.97	0.0174182	31.78
128	0.0008512	81.30	0.0026702	26.11
64	0.0002642	32.74	0.0004884	17.71
32	0.0000801	13.49	0.0001521	7.10
16	0.0000433	3.12	0.0000714	1.89
8	0.0000387	0.43	0.0000475	0.35

6.2 マルチグリッド法の計算性能

マルチグリッド V サイクルアルゴリズムを 10 回行う．緩和計算は各格子サイズにおいて 10 回行う．メモリ確保，初期設定，GPU 上へのデータ転送は計算時間に含めない．2 次元の最密格子は 4096×4096，最疎格子は 32×32 とする．3 次元の最密格子は 256×256×256，最疎格子は 8×8×8 とする．

表 8 に CPU と GPU の 2 次元マルチグリッド法の計算時間と速度比を示す．単精度 5 点差分では 30.16 倍，9 点差分では 35.91 倍を得られた．倍精度では単精度と比べ速度比が 1/2 以下に低下した．緩和計算が計算時間の大部分を占めるためマルチグリッド法の速度比は緩和計算に依存する．結果として 5 点差分よりも FLOPS の高かった 9 点差分の速度比が高くなっている．

表 9 に 3 次元マルチグリッド法の計算時間と速度比を示す．3 次元の単精度 7 点差分では 26.38 倍，15 点差分では 40.07 倍，19 点差分では 40.93 倍，27 点差分では全ての差分法の中での最大値である 46.93 倍を得られた．

倍精度では緩和計算で大きな FLOPS の低下の見られた 27 点差分のみ大きな速度比の低下が見られたが CPU よりも 17 倍高速に計算を行うことができた．

表 8　2 次元マルチグリッド法の計算時間

	CPU	GPU	
	計算時間 (sec)	計算時間 (sec)	速度比
単精度 5 点差分	13.5766580	0.4501538	30.16
単精度 9 点差分	19.2131069	0.5609741	35.91
倍精度 5 点差分	16.8115240	0.9530280	17.64
倍精度 9 点差分	27.1160500	1.2122920	22.37

表 9　3 次元マルチグリッド法の計算時間

	CPU	GPU	
	計算時間 (sec)	計算時間 (sec)	速度比
単精度 7 点差分	11.9772310	0.4539490	26.38
単精度 15 点差分	24.5909259	0.6137440	40.07
単精度 19 点差分	31.0276270	0.7693541	40.33
単精度 27 点差分	40.7218812	0.8676682	46.93
倍精度 7 点差分	17.4610028	0.8991361	19.42
倍精度 15 点差分	28.2315090	1.2266309	23.02
倍精度 19 点差分	35.1624291	1.4344580	24.51
倍精度 27 点差分	43.8956170	2.5548120	17.18

6.3 マルチグリッド法の精度

表 10 と表 11 にマルチグリッド法の精度，最密格子 N，V サイクルの反復回数，計算時間を示す．精度は解が 2 次元では e^{xy}，3 次元では $e^x \cos y \times z^2$ になることが知られているので求めた近似値と絶対誤差無限大ノルムを求める．最も精度の優れている格子サイズの結果を示す．

2 次元マルチグリッド法の単精度では丸め誤差の影響を受ける格子サイズと精度，反復回数ともに違いはほぼ無い．倍精度の 5 点差分では 10^{-10} の精度，9 点差分では 10^{-13} の精度を得られた．9 点差分では 1024×1024 の格子サイズで最大精度を得られた．9 点差分が格子サイズ，精度，反復回数，計算時間ともに 5 点差分よりも有利であると考えられる．

3 次元マルチグリッド法の単精度の精度はほぼ違いが無い．7 点差分では格子サイズが 64×64×64 の時に最も良い精度を得らた．19 点差分と 27 点差分では反復回数の増加が見られた．倍精度の 7 点差分は単精度と精度にほぼ違いはない．15 点差分と 19 点差分は精度がほぼ同じであるが 19 点差分の反復回数の増加が大きく見られ，27 点差分は最も良い精度を得られたが反復回数の大きな増加が見られた．15 点差分と 27 点差分が有利であると考えられる．15 点差分は少ない反復回数で最大精度を得られ，27 点差分の反復回数は多いものの最も良い精度を得られるからである．必要とする精度によって 15 点差分と 27 点差分を使い分けるのが有効であると考えられる．

表 10　2 次元マルチグリッド法の精度

	N	反復回数	計算時間 (sec)	精度
単精度 5 点差分	64	11	0.01056719	3.70×10^{-6}
単精度 9 点差分	64	9	0.01656199	5.48×10^{-6}
倍精度 5 点差分	4096	16	1.5300579	2.38×10^{-10}
倍精度 9 点差分	1024	17	0.1865962	2.03×10^{-13}

表 11　3 次元マルチグリッド法の精度

	N	反復回数	計算時間 (sec)	精度
単精度 7 点差分	64	9	0.01760817	5.16×10^{-6}
単精度 15 点差分	16	6	0.00190902	1.52×10^{-6}
単精度 19 点差分	16	8	0.00414395	5.36×10^{-7}
単精度 27 点差分	16	11	0.00922394	1.04×10^{-6}
倍精度 7 点差分	256	15	1.3403740	1.35×10^{-7}
倍精度 15 点差分	256	21	2.5042360	9.84×10^{-12}
倍精度 19 点差分	256	72	10.2521529	1.11×10^{-12}
倍精度 27 点差分	64	54	0.5502892	8.55×10^{-15}

7　おわりに

使用した実験環境で CPU と比較して，単精度では最高 47 倍，倍精度では最高 25 倍の高速化を達成した．本論文では，GPU による高速化の対象として，マルチグリッド SOR 法を選んだが，マルチグリッド共役勾配法も有力な候補であり，今後の研究が待たれる．また，複数の GPU を用いたさらなる高速化についても，今後の課題である．

参考文献

[1] NVIDIA : CUDA Zone . http://www.nvidia.co.jp/object/cuda_home_new_jp.html.

[2] 小川慧，青木尊之 : GPU によるマルチグリッド法を用いた 2 次元非圧縮性流体解析の高速計算．日本計算工学会，No.20090021，2009.

[3] Murli M. Gupta, Jule Kouatchou , Jun Zhang : Comparison of Second and Fourth Order Discretizations for Multigrid Poisson Solver. Journal of Computational Physics Volume 132, Issue 2, Pages 226-232, 1997

[4] 竹内敏己，藤野清次 : 右辺項の工夫によるポアソン方程式の高次元差分公式について．数理解析研究所講究録，915 巻 12-27，1995.