Vol.2015-HPC-148 No.11
2015/3/2

GPU クラスタ上の AMG 法の高速化

荒谷祐紀† 藤井昭宏† 田中輝雄†

本研究は，ポアソン方程式を題材に GPU クラスタを用いた AMG 法の高速化を行う．まず，GPU クラスタ上で，問題の性質やサイズによらず高速に収束する緩和法を選択する必要がある．そこで緩和法に， GPU 上でよく使用される Jacobi 法に加え，Multicolor Gauss-Seidel 法や Chebyshev 多項式緩和法を適用し，これらが性能に与える影響を分析する．さらに，AMG 法の最適化として，粗いレベルの精度を下げた混合精度を適用することで高速化を行った．緩和法の比較については，Chebyshev 多項式緩和法が他の緩和法に対して良い性能を示した．さらに，混合精度を適用することにより，一般的である倍精度で Jacobi 法を適用した実装と比較すると，最大で約 1.9 倍高速となる場合があった．

1. はじめに

様々な物理解析にあらわれる偏微分方程式は，有限要素法や有限差分法などを用いて，連立一次方程式で近似することができる．代数的マルチグリッド法（AMG 法）は，これらの連立一次方程式を高速に解く反復解法である[1].

AMG 法は，粗い問題を複数生成し，マルチレベルを構成して反復的に解を求める．AMG 法で解を求める際，各レベルでは緩和法と呼ばれる実際に解を近似する計算が必要であり，Jacobi 法や Gauss-Seidel 法のような定常反復解法が利用される．解法部の性能は，緩和法にどのような解法を適用するかによって大きく左右される．Jacobi 法は，単純な緩和法の一つであり，GPU 上での細粒度並列が容易である．そのため，GPU 上での AMG 法では Jacobi 法を用いた研究が多く存在している[1] [2]．しかし，Jacobi 法は問題の性質によっては収束が困難な場合がある．また, Gauss-Seidel 法は，Jacobi 法と比較して高い収束性能をもつが，未知数の更新において計算に依存性が発生するため，並列化が困難である．これらに対し，Chebyshev 多項式緩和法は，Jacobi 法同等の並列性に加え，高い収束性をもつ緩和法である[3] [4]．この緩和法は，多項式の次数による，計算量と収束性の調整や，問題の固有値を用いた最適化が可能であるため，広い範囲の問題に対して高い性能が期待できる．

我々はこれまでに Chebyshev 多項式緩和法を AMG 法に適用し，1GPU や 2GPU の環境で評価を行ってきた[5]．本研究では，GPU クラスタ上で最大 64GPU を使用し，Chebyshev 多項式緩和法を AMG 前処理付 BiCGStab 法（AMG-BiCGStab 法）の緩和法として適用した．比較対象には，Jacobi 法，Multicolor Gauss-Seidel 法を使用し，評価を行った．また，前処理として用いる AMG 法を混合精度にすることにより，緩和法の選択と合わせて高速化を図った．その上で数値実験を行い，ストロングスケーリング，ウィークスケーリングの観点から分析を行った．

†1 工学院大学
Kogakuin University

2. AMG 法

本章では AMG 法の説明を行う．AMG 法は，大規模な非構造格子の問題を対象とする数値解法である．これは，マルチグリッド法のように数段階にわけて粗い問題を生成し，これらの小規模な問題を利用して問題を解く．粗い問題を生成する段階を構築部，生成した問題を利用し反復的に解を求める段階を解法部と呼ぶ．AMG 法には粗い問題の生成方法にいくつかの種類があり，本研究では問題の構築に Smoothed Aggregation AMG（SA-AMG）法[6] [7]を用いた．

構築部で粗い問題を生成する際，レベル間を移動するための行列として Restriction 行列と Prolongation 行列を同時に生成する．構築部ではこの操作を，問題が十分に小さくなるまで再帰的に行う．本研究では，行列が 100 行未満となる問題を最も粗い問題とした．

構築部は逐次的な処理が多く，GPU 上で実行しても性能が出ないという研究があり，本研究では GPU による高速化の対象としない[1].

解法部では構築部で生成した問題を用い，マルチレベルを構成して計算を行う．計算は，もとの問題となる最も細かい問題から，最も粗い問題へ向かって順に処理を行った後，再び細かい問題へ向かって処理を行っていく．このレベルの行き来が V 字を思わせるため，この方法を V-cycle と呼ぶ．図 1 は，3 階層の V-cycle である．

V-cycle では，各階層で緩和法を数回ずつ使用しながらレ

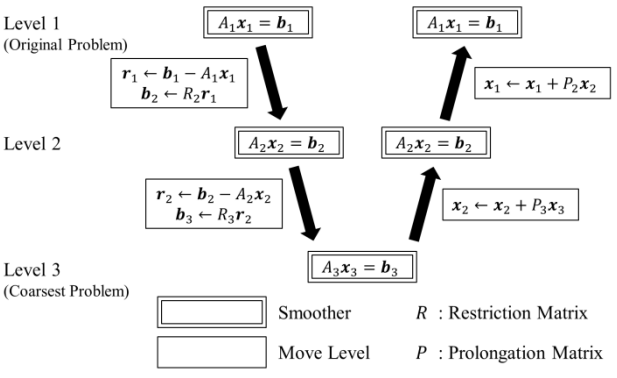

図 1 V-cycle

ベルを移動する．粗いレベルへ移動するときは，そのレベルでの残差ベクトルを計算し，Restriction 行列との疎行列ベクトル積を行う．得られたベクトルは，粗いレベルでの右辺ベクトルとして使用する．細かいレベルへ移動するときは, Prolongation 行列と粗いレベルの解ベクトルとの疎行列ベクトル積を行い，細かいレベルの解ベクトルに加算して解の補正を行う．また，最も粗い問題に対しては，緩和法を数十回使用して十分に近似するか，直接解法で解を求める．

3. 高速化手法

本章では，AMG 法の高速化手法について説明する．本研究では AMG 法の高速化のために，高速な緩和法の選択と V-cycle の混合精度化を行った．まず，実験で使用する緩和法について説明を行い，Chebyshev 多項式緩和法の性質を示す．次に，AMG 前処理向けに実装した混合精度 V-cycle について説明を行う．また，本研究の AMG-BiCGStab 法で用いる，疎行列計算の GPU 上での最適化についても述べる．

3.1 緩和法

AMG 法において，緩和法は性能を左右する大きな要因となる．本研究では，Jacobi 法，Gauss-Seidel 法，Chebyshev 多項式緩和法[3]の3つを比較に用いた. Gauss-Seidel 法は，純粋な並列化が不可能であるため，Multicolor 法によって並列性を抽出した Multicolor Gauss-Seidel 法[8]を用いた．これは，依存性を考慮して未知数を彩色し，GPU 向けに各色の要素が連続するように行列のリオーダリングを行った手法である．

これらの緩和法は，方程式$A\boldsymbol{x}=\boldsymbol{b}$について全て式(1)のように一般化することが可能である．ここでkは反復回数を示す．式(1)におけるM^{-1}はAを利用した行列であり，Jacobi 法，Gauss-Seidel 法ではそれぞれD^{-1}，$(D+L)^{-1}$を用いる．D，Lはそれぞれ行列Aの対角行列，下三角行列である．

$$\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}+M^{-1}\left(\boldsymbol{b}-A\boldsymbol{x}^{(k)}\right) \tag{1}$$

Chebyshev 多項式緩和法では，M^{-1}にAを利用した多項式を利用して計算を行う．式(1)を変形し，式(2)を得る．

$$\begin{aligned}&\boldsymbol{x}^{(k+1)}=P_m(A)\boldsymbol{x}^{(k)}+M^{-1}\boldsymbol{b}\\&P_m(A)=I-M^{-1}A\end{aligned} \tag{2}$$

また，真の解を$\boldsymbol{x}^*$とすると，反復しても更新されないため，式(3)が得られる．

$$\boldsymbol{x}^*=P_m(A)\boldsymbol{x}^*+M^{-1}\boldsymbol{b} \tag{3}$$

ここで，式(2)と式(3)の差をとると，式(4)が得られる．式(4)は，$k+1$反復時の誤差を表している．このベクトルが$P_m(A)$の固有ベクトルの線型結合であると考えたとき，$P_m(A)$の固有値が 0 に近いほど 1 反復での誤差の減少が大きいことを示している．

$$\boldsymbol{x}^{(k+1)}-\boldsymbol{x}^*=P_m(A)\left(\boldsymbol{x}^{(k)}-\boldsymbol{x}^*\right) \tag{4}$$

Chebyshev 多項式緩和法では，$P_m(A)$を Chebyshev 多項式を用いた式(5)を用いて決定する．T_mは，m次の Chebyshev 多項式である．式(5)は，αとβの範囲内で関数の絶対値を最小化する式として知られている．この場合はAが行列であるため，$P_m(A)$を最小化する範囲として，Aの固有値の範囲を指定する．具体的には，βに問題行列Aの最大固有値を与えることで, $P_m(A)$の固有値の絶対値が全て 1 未満となり，Chebyshev 多項式緩和法で解が収束する．

$$\begin{aligned}&P_m(A)=\frac{T_m\left(\frac{\beta+\alpha-2A}{\beta-\alpha}\right)}{T_m\left(\frac{\beta+\alpha}{\beta-\alpha}\right)}\\&T_0(x)=1,\quad T_1(x)=x,\\&T_m(x)=2xT_{m-1}(x)-T_{m-2}(x),\\&m=2,3,\ldots\end{aligned} \tag{5}$$

行列の最大固有値を近似した値を求めるために，まず共役勾配法を使用し，反復ごとに得られる変数からサイズの小さい Hessenberg 行列を作成する．この行列は，共役勾配法の反復回数と同じ数の行となり，小さいサイズでもAの最大固有値によく近似された固有値をもつ[9]．本研究では共役勾配法を 50 反復して Hessenberg 行列を作成し，これに対してべき乗法を用いて最大固有値を求めた．また，αにはAの最小固有値を与えることが望ましいが，最小固有値を求めるのは困難である．しかし，αには厳密な最小固有値を与える必要がないことがわかっており，A. H. Baker らの研究[4]では，マルチグリッド法において，物理形状の次元数に応じてαとβの比を決定している．本研究でも同様に，最小固有値を厳密に求めずに，βの値を定数倍した小さい値を用いた．しかし，本研究で AMG 法において問題の形状から最適な比は確認できていない．

Chebyshev 多項式緩和法は，疎行列ベクトル積とベクトル演算のみで構成されており, 並列化が容易である. また，疎行列ベクトル積の回数はm次のときm回であり，計算量は Jacobi 法などと比較して約m倍となる．図 2 に，$\alpha=0.2,\beta=1.8$としたときの 3 次と 5 次の$P_m(\lambda)$を示す．$0<\lambda<2$の範囲で$P_m(\lambda)$の絶対値は 1 未満となっており，

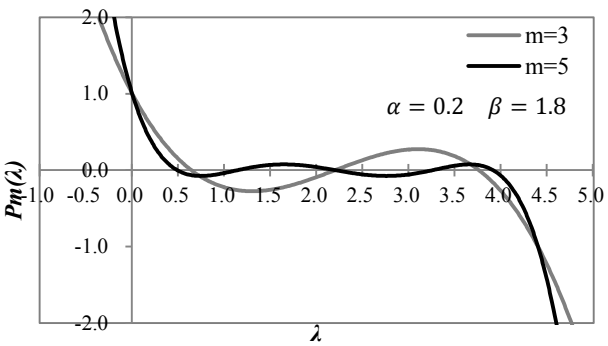

図 2 Chebyshev 多項式緩和法の$P_m(\lambda)$

また，次数を増やした場合のほうがより 0 に近似され収束性が向上する．次数の調整は，計算量と収束性に関わっており，問題の性質によって調整することが可能である．

3.2 混合精度 V-cycle

AMG-BiCGStab 法では，実行時間の多くを V-cycle による前処理が占めている．よって，V-cycle を高速化することで解法全体の性能を大きく向上させることができる．本研究では，AMG 前処理向けの単精度を用いた V-cycle の混合精度化を実装した．

V-cycle の計算は，精度の変更による収束性への影響が小さいことがわかっている[10]．また，BiCGStab 法で行う疎行列ベクトル積は，V-cycle の最も細かいレベルと同じ行列を使用する．しかし，解法の入出力が倍精度のベクトルであることを想定すると，BiCGStab 法の精度を単精度に変更することができない．V-cycle 専用に単精度の行列を別に確保することも考えられるが，GPU では搭載メモリの制約があるため，メモリ使用量を十分に考慮しなくてはならない．これらの理由から，単精度化の対象は V-cycle の粗いレベルのみとした．

混合精度 V-cycle では，まずレベル 1 において倍精度の緩和法を使用し，倍精度の行列とベクトルから単精度の残差を得る．その後，単精度のみでレベル 2 へ移動し，粗いレベルは全て単精度で計算する．レベル 2 からレベル 1 へ移動する際は，レベル 2 の単精度の解を，レベル 1 の倍精度の解へ加算する．

疎行列計算はデータの再利用性が低いため，メモリバンド幅によって性能が決定される．そのため，単精度の疎行列は倍精度に比べ高速となる．今回使用する疎行列圧縮形式を用いた緩和法や疎行列ベクトル積では，最内側のループで浮動小数データ 2 つと整数データ 1 つをロードするため，倍精度では 20Byte，単精度では 12Byte となり，メモリアクセス量のみから判断すると，単精度化した部分は最大で 1.66 倍の高速化が見込める．

3.3 疎行列計算の最適化

本研究では，V-cycle の全てを GPU に実装した．V-cycle で行われる主な計算は，疎行列ベクトル積であり，レベル間の移動や緩和法で利用される．そのため，V-cycle では疎行列の計算手法が実行効率に強く関わる．本研究では，1 スレッドが 1 行を計算する並列化を行い，これに適した疎行列格納形式を選択した．

GPU では，連続したスレッドが連続したデータにアクセスする場合が最も効率的なメモリアクセスとなる．疎行列を扱う際に一般的に用いられる CRS（Compressed Row Storage）形式では，各行の要素が非連続であるため，GPU では高い性能が引き出せない．本研究では，ELL（ELL Pack）形式に，各行の非ゼロ要素数を記録したベクトルを追加した疎行列圧縮形式を採用した．ELL 形式のままでは，各行の非ゼロ要素数にばらつきがある場合，無駄な計算が増加し性能が低下してしまう．本形式では，無駄な計算が連続した 32 スレッド（Warp）内で最大の非ゼロ要素数に依存するため，ELL 形式に比べて高い性能となる．各レベルの問題行列，Restriction 行列，Prolongation 行列は全てこの形式で格納し，計算を行う．

また，Multicolor Gauss-Seidel 法は，同時に並列可能な部分を同じ色として彩色する．彩色を行った際，もとの行列のままでは同じ色同士が連続しておらず，メモリアクセス効率が低下してしまう．そのため，同じ色となる行が連続するようにリオーダリングを行い，色ごとに別の行列としてメモリに配置する．これらの行列について色ごとに順次カーネルを実行することによって，高速なメモリアクセスを実現している．また，粗いレベルでは依存性が複雑になり色数が大幅に増加してしまうために，並列性が低下してしまう．本研究では，粗いレベルには Jacobi 法を適用することで，これを回避している．

4. 数値実験と考察

本章では，緩和法のパラメータ最適化を行い，高速な緩和法の選択と混合精度 V-cycle による高速化について，GPU クラスタ上での実験に基づき評価を行う．まず，緩和法の比較を行い，有効な緩和法を示す．その上で最も高速な緩和法を選択し，混合精度 V-cycle による高速化の確認を行う．また，その際の時間構成を分析し，評価を行う．

4.1 実験環境

実験環境には，筑波大学計算科学研究センターの HA-PACS ベースクラスタを利用した．ノード仕様を表 1 に示す．HA-PACS ベースクラスタでは，合計 268 台の計算ノードがフルバイセクションバンド幅をもつ Fat-Tree ネットワークで結合している．ネットワークハードウェアは InfiniBand QDR 規格を 2 系統並列設置しており，片方向あたり 8GB/sec の理論ピークバンド幅となる[11]．

MPI には MVAPICH2 を利用し，コンパイラには gcc4.4.7 と nvcc5.5.22 を利用した．

表 1　HA-PACS ベースクラスタ　ノード仕様

CPU	Intel E5 2.6GHz (Sandy Bridge-EP) 8core / socket x2 (16 core/node)
GPU	NVIDIA M2090 (4 GPU/node)
PCI-express	Generation 3 x80 lane (40 lane/CPU)
Network	InfiniBand QDR x2 rail (Mellanox ConnectX-3 dual head)

4.2 問題設定とパラメータ決定法

問題には，立方体領域における 3 次元拡散方程式を使用する．これについて Darcy 流れの問題，異方性問題の 2 種類を用いて評価を行う．Darcy 流れの問題は不均一に拡散係数が分布している問題であり，異方性問題は Z 軸方向に他軸の 100 倍の拡散係数をもつ問題である．一般的にこれらの問題は，均一な拡散係数をもつ等方性の問題よりも収束が困難である．SA-AMG 法で粗いレベルを生成する際の，強連結成分を判定する閾値は，Darcy 流れの問題は 0.01，異方性問題は 0.09 と設定した．また，問題を各プロセスに分割する際は，ParMETIS[12]を使用し，最も粗いレベルについては行列サイズが非常に小さいため，あらかじめひとつの領域に集約した．

比較する緩和法は Jacobi 法，Multicolor Gauss‐Seidel（MCGS）法，Chebyshev 多項式緩和法の 3 つとした．ただし，MCGS 法，Chebyshev 多項式緩和法は，最も細かいレベルのみに適用し，粗いレベルでは全て Jacobi 法を用いる（MCGS+Jacobi 法，Chebyshev+Jacobi 法）．これは，最も細かいレベルでの収束性が最も性能に影響するためである．計測は全て AMG-BiCGStab 法による反復解法部のみとし，収束条件は 2 ノルム相対残差が1.0×10^{-7}未満となったときとした．

緩和法のパラメータは，緩和法の加速係数を決定した後に，各レベルでの反復回数を設定した．緩和法の加速係数は，最も細かいレベルとその他の粗いレベルに分けて設定した．1.0 を基準として周辺を 0.1 刻みで探索し，最適な値を選択した．ただし Chebyshev 多項式緩和法は，加減速を行わない．緩和法の反復回数は，最も粗いレベル，中間のレベル，最も細かいレベルに分けて設定した．最も粗いレベルは 30 回で固定し，最も細かいレベルと中間のレベルは，1 回か 2 回から選択し，収束時間が最も短くなる組み合わせに設定した．

Chebyshev 多項式緩和法のαは，βとの比によって決定し，Darcy 流れの問題では$\beta/10$，異方性問題では$\beta/30$とした．また，Chebyshev 多項式の次数は 2 次から 4 次とし，それぞれについてパラメータを最適化した上で最も高速だった次数を選択した．Darcy 流れの問題では 2 次，異方性問題では 4 次が最速となった．

パラメータは 2 種の問題それぞれについて 100^3，1GPU で最適化し，他の問題サイズや並列度でも同じ値を使用した．表 2，表 3 にそれぞれ Darcy 流れの問題，異方性問題で使用したパラメータを示す．

表 2　Darcy 流れの問題　パラメータ設定

	Smoother Iteration			Acceleration Coefficient	
	Finest	Inter	Coarsest	Finest	Coarser
Jacobi	1	2	30	1.0	1.0
MCGS	1	2	30	1.0	1.0
2nd order Chebyshev	1	2	30	none	1.0

表 3　異方性問題　パラメータ設定

	Smoother Iteration			Acceleration Coefficient	
	Finest	Inter	Coarsest	Finest	Coarser
Jacobi	1	1	30	0.4	0.7
MCGS	1	1	30	1.0	0.7
4th order Chebyshev	1	2	30	none	0.7

4.3 緩和法の性能評価

緩和法の性能を評価するために，ストロングスケーリングとウィークスケーリングで実験を行った．

まず，ストロングスケーリングについて評価する．ストロングスケーリングでは，問題サイズを固定し，GPU 数を変化させて計測を行う．問題サイズは 100^3 とし，1GPU から 16GPU までを対象とした．

Darcy 流れの問題での収束までの反復回数と時間を，それぞれ表 4 と図 3 に示す．並列度を変えると反復回数が変化するが，これは SA-AMG 法のマルチレベル生成手順により，粗いレベルの行列が変化するからである．また，MCGS 法は領域境界で Jacobi 法と同様に依存性を考慮しない実装になっており，並列度により異なる緩和法となっているのも原因の一つと考えられる．

表 4　ストロングスケーリング
Darcy 流れの問題　反復回数

	Number of GPU				
	1	**2**	**4**	**8**	**16**
Jacobi	24	28	23	24	22
MCGS + Jacobi	20	24	21	23	23
Chebyshev + Jacobi	21	23	17	22	20

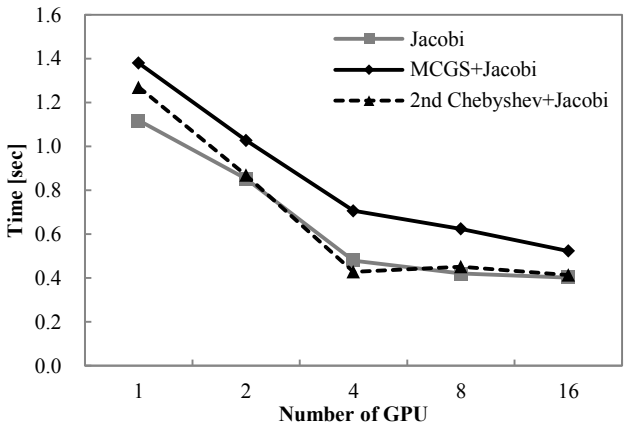

図 3　ストロングスケーリング
Darcy 流れの問題　収束時間

Darcy 流れの問題では収束性が低くても反復回数が増えづらく, Chebyshev＋Jacobi 法と Jacobi 法が高速に収束した．Chebyshev 多項式緩和法は計算量の少ない 2 次が最も高速となった．Jacobi 法や Chebyshev 多項式緩和法に対し MCGS+Jacobi 法は, 反復回数が Jacobi 法からあまり減らず, 高い性能にはならなかった．

ストロングスケーリングにおける異方性問題での収束までの反復回数と時間を，それぞれ表 5 と図 4 に示す．異方性問題では高い収束性が必要であり，Chebyshev 多項式緩和法は 4 次が最も高性能であった．収束性の低い Jacobi 法は相対的に反復回数が多くなってしまったため, Chebyshev＋Jacobi 法や MCGS+Jacobi 法が有効となった．特に，Chebyshev+Jacobi 法は Jacobi 法と比較して最大で約 1.5 倍高速である．

2 種の問題について並列化した結果，いずれも最大で約 3 倍の高速化となった．しかし異方性問題において，8GPU を超えると性能の向上がみられず，16GPU では 8GPU から性能が低下してしまっている．Darcy 流れの問題でもこれ以上の並列度では性能の向上が期待できない傾向を示した．これは，並列化した際に通信の時間が増加しているためである．

次に，ウィークスケーリングについて評価する．ウィークスケーリングでは, 各 GPU に割り当てる行列サイズを約 100 万行とし, GPU の並列度を変化させて実験を行う. GPU は 1，8，27，64GPU を対象とした．この際の全体の問題サイズは，それぞれ 100^3，200^3，300^3，400^3 である．

ウィークスケーリングにおける Darcy 流れでの収束までの反復回数と時間をそれぞれ表 6 と図 5 に示す．ウィークスケーリングでも，Darcy 流れの問題では Jacobi 法と Chebyshev+Jacobi 法が高速であった．ウィークスケーリングでは, 反復回数が同じであれば並列度を変えても各 GPU の計算量が変化しない．しかし，Chebyshev+Jacobi 法は 64GPU で，1GPU に対し約 3.3 倍の時間がかかっており，増加している時間のほとんどは通信時間と考えられる．
また，Jacobi 法と Chebyshev+Jacobi 法は並列度の変化に対する傾向がよく似ている．しかし，MCGS+Jacobi 法は他の緩和法と異なり，高い並列度では時間が増加しやすい傾向をみせた．

ウィークスケーリングにおける異方性問題での反復回数と収束時間を表 7，図 6 に示す．緩和法の性能は，ストロングスケーリングと同様に Chebyshev 多項式緩和法が最も高速である．また，Darcy 問題と同様に，異方性問題でも MCGS＋Jacobi 法は時間が増加しやすい傾向があり，8GPU 以降は Jacobi 法の方が高速な結果となった．レベル 1 の MCGS 法の色数を確認したところ，並列度を高くすると色数が増えており，これが原因の一つと考えられる．MCGS 法でより高い性能を実現するためには，色ごとに分割することによって小さい行列を扱うことに対処する必要がある．

表 5　ストロングスケーリング　異方性問題　反復回数

	Number of GPU				
	1	**2**	**4**	**8**	**16**
Jacobi	46	56	48	47	45
MCGS + Jacobi	26	26	28	28	29
Chebyshev + Jacobi	16	17	17	17	18

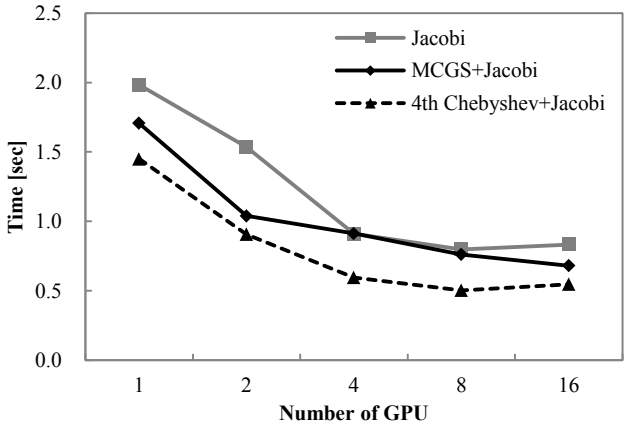

図 4　ストロングスケーリング　異方性問題　収束時間

表 6　ウィークスケーリング
Darcy 流れの問題　反復回数

	Number of GPU			
	1	**8**	**27**	**64**
Jacobi	24	31	45	40
MCGS + Jacobi	20	28	33	36
Chebyshev + Jacobi	21	28	34	34

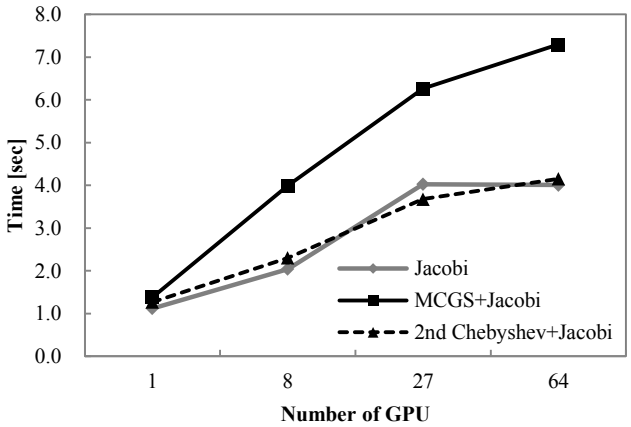

図 5　ウィークスケーリング
Darcy 流れの問題　収束時間

表 7　ウィークスケーリング　異方性問題　反復回数

	Number of GPU			
	1	**8**	**27**	**64**
Jacobi	46	50	50	51
MCGS + Jacobi	26	30	34	33
Chebyshev + Jacobi	19	22	22	24

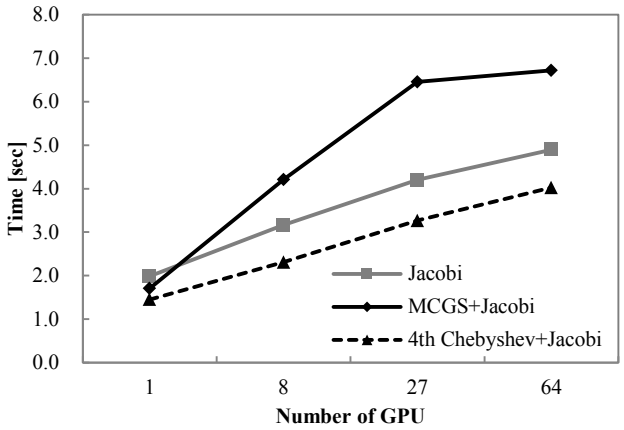

図 6　ウィークスケーリング　異方性問題　収束時間

実験の結果，Darcy 流れの問題と異方性問題のいずれにおいても，Chebyshev＋Jacobi 法が高速であった．特に異方性問題では，他の 2 つの緩和法よりも高い性能を示した．Chebyshev 多項式緩和法は，プロセスで分割しても処理内容が変わらず，安定して高速である．また，問題の性質によって次数を調整することで，収束性と計算量の調整が可能であり，問題の性質に合わせて最適化が可能であることがわかった．

4.4 混合精度 V-cycle による高速化

4.3 節で最も安定して高速であった Chebyshev+Jacobi 法に混合精度 V-cycle を適用し，通常よく利用される倍精度の Jacobi 法との比較を行う．

まず，Darcy 流れの問題を用いて，混合精度 V-cycle の効果を確認する．混合精度を適用すると，倍精度から反復回数が変化する場合があった．ストロングスケーリングの場合の，Darcy 流れの問題での Chebyshev+Jacobi 法の反復回数の変化を表 8 に示す．反復回数が増えることもあったが減ることもあり，変化は 1，2 反復と小さい範囲であった．

図 7 は，ストロングスケーリングにおける混合精度化の効果である．最も一般的に用いられる倍精度の Jacobi 法と，倍精度 Chebyshev+Jacobi 法，混合精度 Chebyshev+Jacobi 法を比較した．それぞれを Double Jacobi，Double Chebyshev+Jacobi，Mix Chebyshev+Jacobi で示した．倍精度と混合精度の Chebyshev+Jacobi 法を比較すると，1GPU や 2GPU のときは混合精度化の効果が高い．反復回数が変化したため安定した性能向上は得られていないが，性能が低下することはないことがわかる．しかし，並列度を高くすると，性能比が低下していく傾向がみられた．これは，混合精度化を行っても通信部分が高速にならなかったためである．1GPU のとき，混合精度化により 1 反復が約 1.16 倍高速化されたが，16GPU ではほとんど高速にはならなかった．

表 8　ストロングスケーリング　Darcy 流れの問題
Chebyshev＋Jacobi 法　混合精度での反復回数の変化

	Number of GPU				
	1	**2**	**4**	**8**	**16**
Double Precision	21	23	17	22	20
Mix Precision	22	21	18	21	20

図 8 に，ウィークスケーリングでの混合精度 V-cycle の効果を示す．ウィークスケーリングでは，混合精度化の効果がよくあらわれている．ただし，8GPU と 64GPU では反復回数が減少している．27GPU のときは，Jacobi 法に対して約 1.2 倍高速である．また，ウィークスケーリングでも，並列度を高くすると性能向上の割合が低下する．しかし，ウィークスケーリングでは扱う問題サイズが大きいためストロングスケーリングよりも効果が高く，64GPU を使用し

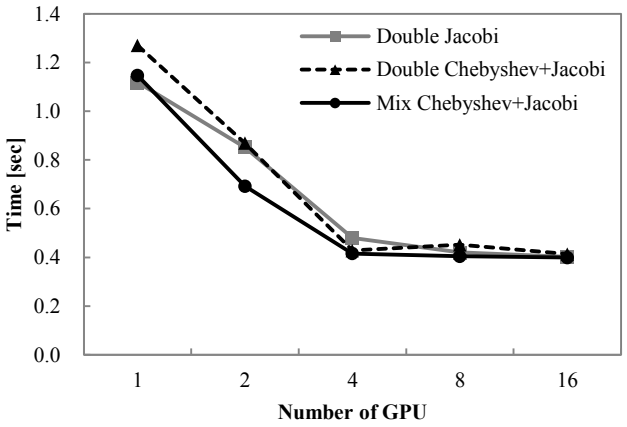

図 7　ストロングスケーリング　Darcy 流れの問題
混合精度 V-cycle の効果

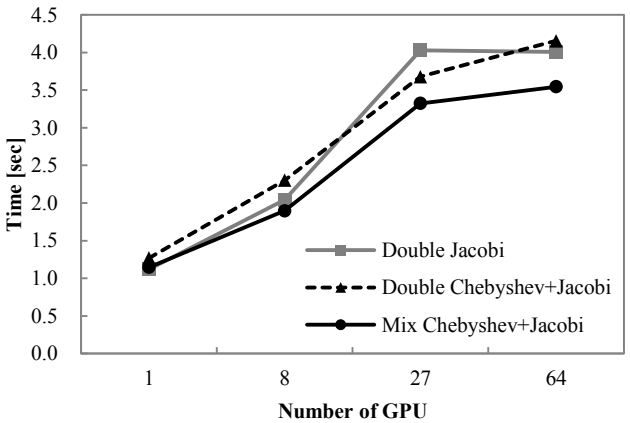

図 8　ウィークスケーリング　Darcy 流れの問題
混合精度 V-cycle の効果

ても 1 反復あたり 1.1 倍程度高速化されている．

また，他の緩和法について混合精度 V-cycle の効果を確認したところ，問題や緩和法を問わず同様に高速化されていた．

次に，ストロングスケーリング，ウィークスケーリングのそれぞれの時間構成を分析する．

図 9 は，ストロングスケーリングでの収束時間を，緩和法の時間に注目して分解した図である．倍精度 Chebyshev+Jacobi 法（Double）と混合精度 Chebyshev+Jacobi 法（Mix）について示した．

Darcy 流れの問題 100^3 では，V-cycle は 4 レベルとなっていた．図 9 の Level 1 Smoother と Level 2 Smoother はそれぞれのレベルでの緩和法の計算時間であり，Level 1 Comm と Level 2 Comm は緩和法の際に行う通信の時間である．レベル 3 と 4 はひとまとめにし，Coarser と示した．Other は緩和法以外の時間であり，ここではレベル間の移動と BiCGStab 法にかかる時間である．また，通信を除いた計算時間の合計を破線で示した．

倍精度，混合精度のいずれも，計算時間だけをみると 1GPU から 16GPU で約 10 倍高速化されている．特に，Level 1 では約 16 倍高速化されており，十分なスケーリング性能が出ている．しかし，Coarser は並列度に関わらず一定の時間がかかっている．レベル 4 はあらかじめ 1 プロセスに領域を集約しているため高速にはならないが，分割を行っているレベル 3 も高速化されていない．このときレベル 3 の行列は 800 行前後であり，小さい行列に対して GPU の性能が引き出せていないことが原因である．また，並列度が高くなるにしたがって通信部分が増加する．特に，Coarser は計算が高速化されず通信のみが増加していき，性能のスケーリングを阻害していることがわかる．また，8GPU 以上では通信の時間が全体の半分以上を占める．16GPU では約 70%が通信の時間であった．

混合精度の効果は，レベル 2 のような比較的大きい行列を扱うレベルでは高い効果がみられる．しかし，レベル 3 以降ではほとんど高速にならなかった．また，計算が高速化されても通信は高速化されていない．これは，通信で送受信するデータサイズが小さいため，通信遅延の時間がほとんどを占めるためと考えられる．並列度が高い場合は通信時間が多くの割合を占めるため，計算が十分高速化されても全体の性能向上率は高くならなかった．

図 10 は，ウィークスケーリングでの時間構成である．ウィークスケーリングでの Darcy 流れの問題は，1GPU のときは 4 レベルの V-cycle であるが，8GPU 以上では問題サイズが大きくなることにより 5 レベルとなっている．図 10 では，レベル 3 よりも粗いレベルは全て Coarser としてまとめた．

ウィークスケーリングでは，GPU 数に応じて問題サイズが大きくなるが，通信を除いた計算時間はあまり増加していない．また，ウィークスケーリングでは並列度が高くなっても各 GPU が計算する行列は大きく保たれるが，ストロングスケーリング同様に通信の割合は大きい．64GPU のときは全体の約 55%が通信時間である．また，27GPU 以上では，各 GPU が 100^3 の行列を計算するレベル 1 でも，通信時間の割合の方が大きい．粗いレベルでは，処理時間のほとんどが通信の時間となってしまい，粗いレベルの最適化が必要である．

最後に，倍精度で Jacobi 法を用いた場合と比較して，混合精度 V-cycle と Chebyshev 多項式緩和法によりどの程度高速化できたかをまとめる．表 9 と表 10 に，倍精度 Jacobi 法と混合精度 Chebyshev+Jacobi 法の収束時間と速度向上率を示す．それぞれストロングスケーリングとウィークスケーリングの結果である．1 段目は問題の種類と分割数，2 段目は倍精度 Jacobi 法と混合精度 Chebyshev+Jacobi 法の収束時間，3 段目は高速化の割合である．

Darcy 流れの問題では，もともと Jacobi 法と Chebyshev＋Jacobi 法が同等の性能であったため，性能向上のほとん

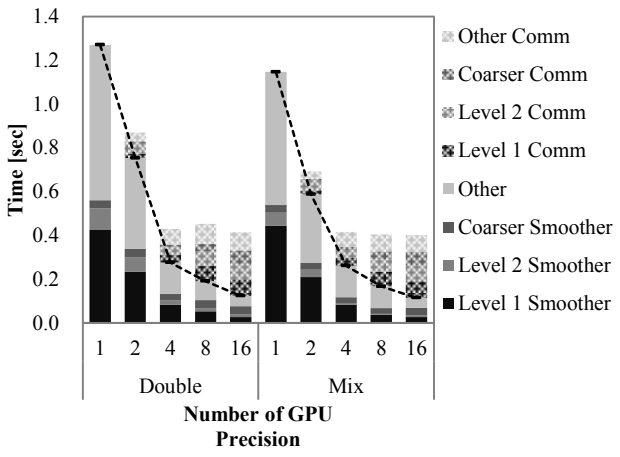

図 9　ストロングスケーリング　Darcy 流れの問題 Chebyshev+Jacobi 法での時間構成

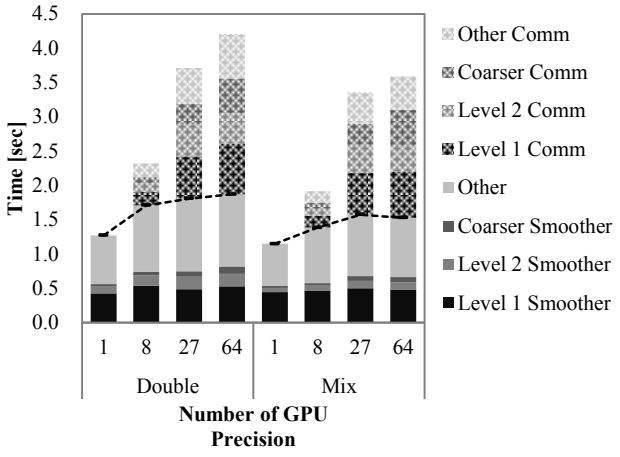

図 10　ウィークスケーリング　Darcy 流れの問題 Chebyshev+Jacobi 法での時間構成

表 9　ストロングスケーリング　倍精度 Jacobi 法と混合精度 Chebyshev+Jacobi 法の収束時間

		Darcy's flow					Anisotropy				
Number of GPU		**1**	**2**	**4**	**8**	**16**	**1**	**2**	**4**	**8**	**16**
Double Jacobi	**[sec]**	1.12	0.85	0.48	0.42	0.40	1.98	1.54	0.91	0.80	0.83
Mix Chebyshev+Jacobi		1.15	0.69	0.42	0.40	0.40	1.29	0.82	0.55	0.46	0.53
Speedup Ratio		0.98	1.23	1.15	1.04	1.00	1.54	1.87	1.65	1.75	1.56

表 10　ウィークスケーリング　倍精度 Jacobi 法と混合精度 Chebyshev+Jacobi 法の収束時間

		Darcy's flow				Anisotropy			
Number of GPU		**1**	**8**	**27**	**64**	**1**	**8**	**27**	**64**
Double Jacobi	**[sec]**	1.12	2.04	4.03	4.01	1.98	3.17	4.20	4.90
Mix Chebyshev+Jacobi		1.15	1.90	3.32	3.54	1.29	2.09	3.00	3.67
Speedup Ratio		0.98	1.08	1.21	1.13	1.54	1.51	1.40	1.33

どが混合精度 V-cycle による高速化の効果である．混合精度 Chebyshev+Jacobi 法は倍精度 Jacobi 法に対し, 最大約 1.2 倍高速となった．

異方性問題では，Chebyshev+Jacobi 法と混合精度の利用により, 倍精度 Jacobi 法と比較して最大で約 1.87 倍高速となった．また，最も高速な並列度であったストロングスケーリングの 8GPU では，1.75 倍の高速化となった．

5. おわりに

本研究では，GPU クラスタ上での AMG 法の高速化を目的とし，適切な緩和法の設定と選択，混合精度の適用について評価を行った．

まず，Jacobi 法，MCGS 法，Chebyshev 多項式緩和法を緩和法として適用し, 比較を行った. MCGS 法と Chebyshev 多項式緩和法は最も細かいレベルでのみ利用し，粗いレベルでは Jacobi 法を用いた．Darcy 流れの問題と異方性問題について実験を行ったところ，いずれの問題でも Chebyshev 多項式緩和法が高速であった．特に，異方性問題では収束性が重要であり，Jacobi 法に対して最大約 1.5 倍高速となった．また，Chebyshev 多項式緩和法は，領域を分割しても処理内容が変わらず, 安定した性能を示した．

また，GPU 上での AMG 前処理向けの混合精度 V-cycle を実装し，評価を行った．その結果，混合精度 V-cycle は問題や緩和法を問わず効果があることを確認した．並列度が低い場合は，最大で 1 反復を 1.2 倍高速化できたが，並列度を高くすると効果が小さくなることを確認した．さらに, Chebyshev 多項式緩和法と混合精度 V-cycle の利用により，一般的な倍精度での Jacobi 法と比べて高速となった．特に，異方性問題では倍精度 Jacobi 法から最大で約 1.9 倍の高速化となり，有効性を確認した．

今回の実験では，並列度が高くなると通信時間が増加してしまい，性能が向上しなかった．特に，粗いレベルでは通信時間が非常に多くの割合を占めているため，今後粗いレベルの最適化について検討していきたい．

謝辞　本研究の一部は，JST CREST「進化的アプローチによる超並列複合システム向け開発環境の創出」による

参考文献

[1] G. Haase, M. Liebmann, C. C. Douglas, and G. Plank: A parallel algebraic multigrid solver on graphics processing units, HPCA LNCS, Volume 5938, pp.38-47 (2010).
[2] M. Wagner, K. Rupp and J. Weinbub: A Comparison of Algebraic Multigrid Preconditioners using Graphics Processing Units and Multi-Core Central Processing Units, Spring Simulation Multiconference (2012).
[3] M. Adams, M. Brezina, J. Hu and R. Tuminaro: Parallel multigrid smoothing: polynomial versus Gauss-Seidel, Journal of Computational Physics, Volume 188, Issue 2, pp.593-610 (2003).
[4] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang: Multigrid Smoothers for Ultraparallel Computing, SIAM Journal on Scientific Computing, Volume 33, Issue 5, pp.2864-2887 (2011).
[5] 荒谷祐紀，藤井昭宏，田中輝雄: GPU 上での AMG 法における Chebyshev 多項式緩和法，Vol.2012-HPC-137, No.36, pp.1-8(2012).
[6] A. Fujii, A. Nishida and Y. Oyanagi: Evaluation of parallel aggregate creation orders: smoothed aggregation algebraic multigrid method, vol. 172, International Federation for Information Processing, pp. 99-122 (2005).
[7] P. Vaněk, J. Mandel, and M. Brezina: Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, Computing, vol.56, no.3, pp.179–196(1996) .
[8] K. Takahashi, A. Fujii, T. Tanaka,: GPGPU-based Algebraic Multigrid Method, Proceedings of the 23rd IASTED International Conference on Parallel and Distributed Computing and Systems, Track 757-061(2011).
[9] U. M. Yang: On the Use of Relaxation Parameters in Hybrid Smoothers, Numerical Linear Algebra with Applications, Volume 11, Issue 2-3, pp.155-172(2004).
[10] Y. Sumiyoshi, A. Fujii, A. Nukada, and T. Tanaka: Mixed Precision AMG method for Many Core Accelerators, International Workshop on Enhancing Parallel Scientific Applications with Accelerated HPC(2014).
[11] 筑波大学 計算科学研究センター HA-PACS ベースクラスタ: http://www.ccs.tsukuba.ac.jp/research/computer
[12] ParMETIS: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview