Vol.2013-HPC-141 No.13
2013/10/1

APU 上の混合精度 AMG 法

住吉優希[†1]　長岡駿希[†2]　藤井昭宏[†1]　額田 彰[†3]　田中輝雄[†1]

GPU 上では倍精度浮動小数点演算に対して，単精度浮動小数点演算は 2 倍程度の性能を発揮することができる．一方で単精度浮動小数点演算では，高精度を必要とする処理において解の精度が十分に得られない．そこで GPU 上では単精度浮動小数点演算を行い，必要な解の精度を得るために CPU 上で倍精度浮動小数点演算を行う混合精度実装手法に着目した．本研究ではこの混合精度手法を CPU と GPU を内蔵する APU 上で代数的マルチグリッド法に適用する．行列サイズ 60^3 の三次元拡散方程式の異方性問題では，すべて倍精度演算を用いた場合と比べて，本手法では求解時間を約 60%に短縮することができた．

1. はじめに

様々な物理現象は大規模連立一次方程式を解くことに帰着される．代数的マルチグリッド法（以下，AMG 法）[1] はこれらの大規模連立一次方程式を高速に解く手法のひとつとして知られている．大規模計算を高速に解くためには GPU が使われている. GPU は一般的に倍精度浮動小数点演算よりも単精度浮動小数点演算が 2 倍程度の性能を発揮する．これはもともと GPU が画像処理用に開発され，高い精度を必要としていなかったためである．しかし一方で一般の大規模連立一次方程式を解く場合は，単精度浮動小数点演算では十分な精度の解を得られないような問題も少なくない．このような場合には，単純に高精度演算で解くよりも混合精度により GPU が得意とする単精度演算を多用することで高速化を実現し，かつ解の精度も保証する混合精度解法が有効である．混合精度解法については密行列や CG 法などへの適用は既に研究されているが，本研究では単精度と倍精度の混合精度による AMG 法を実装し，GPU で評価を行う．

本研究では混合精度演算において倍精度演算は CPU で，単精度演算は GPU で行う．CPU と GPU が同一チップ上にあり CPU，GPU 間のデータ転送時間が短い AMD 社の Fusion APU 上に実装した. 緩和法にヤコビ法を用いた AMG 法の反復解法部を混合精度演算で実装した．そして APU 上で AMG 法の反復解法部を倍精度演算で実装した収束時間と比較した．

2. APU

GPU を計算に利用する技術として GPGPU（General-Purpose computation using Graphics Processing Unit）がある．しかし GPGPU では CPU，GPU 間の PCI-Express バスのデータ転送速度がボトルネックとなることが多い．AMD 社の Fusion APU（Accelerated Processing Unit）（以下，APU）では，ひとつのチップの上に CPU と GPU，メモリコントローラの機能を統合し，実装している [2]．APU では CPU と GPU が同じチップに載っており, メインメモリ（Physical Memory）を CPU 利用部（Memory）と GPU 利用部（Video Memory）に分けているため，CPU，GPU 間のデータ転送速度は高速であり，このボトルネックは解消されている [3]．第二世代 APU の構造を**図 1** に示す．APU では GPU 専用のビデオメモリを，メインメモリの一部を用いて利用する．UNB（Unified North Bridge）は GPU Core が直接メモリにアクセスする際に，CPU のメモリアクセスと調停を行うことで, GPU が DDR3 メモリの全帯域を利用できるようにしている．

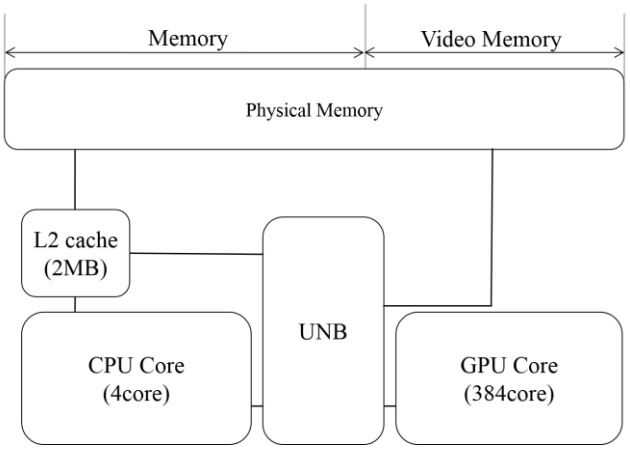

図 1　第二世代 APU (Trinity) の構造

今回，APU でのプログラミングは OpenCL により実装した．OpenCL とはマルチコア CPU や GPU などによる異種混在の計算資源を利用した並列コンピューティングのためのフレームワークである [4]．また，一度動作することが確認されたソースコードがあれば，他のアーキテクチャにおいても修正することなく移植が可能である．つまり移植工程で再コーディングやデバッグの手順を踏むことなく，移植を行うことができるという可搬性の高さも特徴である [5].

†1 工学院大学
Kogakuin University
†2 新日本システック株式会社
Shin Nihon System Technology Corporation
†3 東京工業大学
Tokyo Institute of Technology

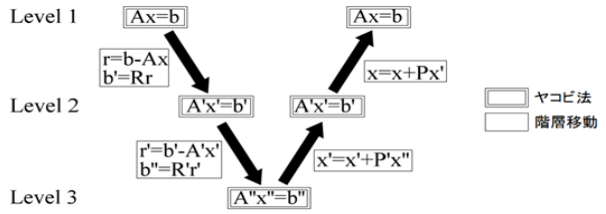

図 2　反復解法部の 1 反復（V-cycle）の構造

3. AMG 法

3.1 AMG 法

AMG 法は問題行列から小さな問題を作成し，緩和法を繰り返し実行することで高速に解を求める．AMG 法は階層構造生成部と反復解法部から成り立っている．階層構造生成部は問題行列から未知数間のグラフ構造を作成し，そのサブグラフにより小行列 A を生成する．反復解法部の 1 反復（V-cycle）は**図 2** に示すような階層構造によって構成される．この V-cycle を反復することで，解を収束させる．**図 2** の Level は階層を示し，Level 1 を最上層，Level 3 を最下層とする．問題規模は Level が大きいほど小規模となる．ひとつ上の Level で反復解法を行い，残差を計算する．その残差をひとつ下の Level に移動し，そこで補正解を計算し，解を補正する．本研究ではこの反復解法部を対象とした．

3.2 反復解法

AMG 法は階層型で，元の問題を最上層と設定し，階層が下がるにつれて元より粗く小さい問題を生成する．階層数は問題の規模によって可変である．階層移動には階層構造生成部で用意した Prolongation 行列と Restriction 行列を使う．階層を降りる際には現階層での残差誤差を計算し，横長の Restriction 行列と行列ベクトル積を行うことで短いベクトルを生成し，一つ下の階層で利用する．階層を上る際は現階層の解と縦長の Prolongation 行列の行列ベクトル積を行うことで長いベクトルを生成し，一つ上の階層の補正解として利用する．各階層で解を求めるため，緩和法が用いられる．この処理を反復することで解を収束させる．

3.3 緩和法

緩和法は，各レベルで $Ax=b$ の計算を A について対角行列 D と下三角行列 L，上三角行列 U に分割し計算する．今回は，並列性の高いヤコビ法を緩和法として採用した．ヤコビ法とは$x' = D^{-1}((b-(U+L)x))$を，x'を補正解として繰り返すことで，解に収束させる [6]．緩和法を各階層で数回行い，最も問題規模の小さい最下層では数十回行うことにより V-cycle の収束率を高めている．

4. 混合精度 AMG 法

本研究では AMG 法部分を単精度で計算を行い，この単精度で得られた解を外側ループの倍精度による反復により補正するような混合精度手法 [7]を用いる．したがって

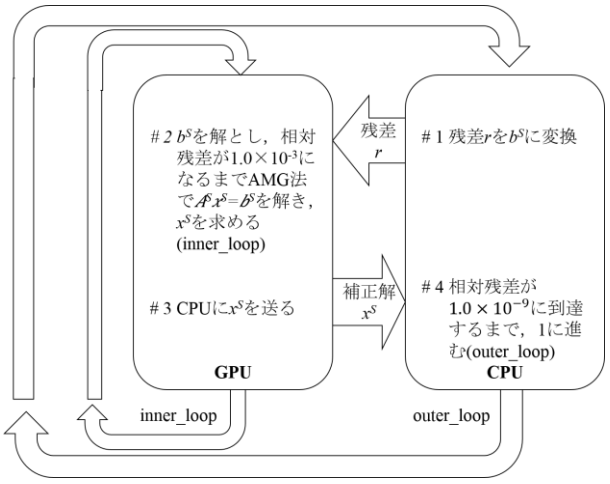

図 3　混合精度 AMG 法

```
A^S = A   /*問題行列を単精度に変換*/
x_0 = 0   /*初期解の設定*/
r_0 = b - Ax_0   /*倍精度の初期残差を計算*/
k = 1
do(outer_loop)
  b^S = r_{k-1}   /*倍精度の残差を単精度に変換*/
  x^S_k = 0   /*初期解の設定*/
  do(inner_loop)
    x^S_k = V-cycle(A^S, x^S_k, b^S)   /*V-cycle を用いて，単精度の解を計算*/
    z = b^S - A^S x^S_k   /*単精度の残差を計算*/
  while (| z | / | b^S |) > 1.0 * 10^-3   /*相対残差が 1.0 * 10^-3 以上であれば do(inner_loop)に戻る*/
  x_k = x_{k-1} + x^S_k   /*倍精度の解を単精度の解で補正*/
  r_k = b – Ax_k   /*倍精度の残差を計算*/
  k++
while (| r_{k-1} | / | r_0 |) > 1.0 * 10^-9   /*相対残差が 1.0 * 10^-9 以上であれば do(outer_loop)に戻る*/
```

図 4　混合精度 AMG 法のアルゴリズム

AMG 法部分は GPU 側で計算を行う．**図 3** に混合精度 AMG 法の概要を，**図 4** に混合精度 AMG 法のアルゴリズムを示す．APU 上の混合精度 AMG 法の手順を以下の# 1~# 4 に示す．S が上付き文字となっている文字は単精度とする．

- # 1　CPU 上で倍精度の残差r_k=b-Ax_k を計算し，単精度 b^S に変換する．
- # 2　#1 で求めた b^S を解として，GPU 上で相対残差が 1.0×10^{-3}になるまで単精度 AMG 法で$A^S x^S_k=b^S$ を解く．(inner_loop)
- # 3　GPU 上で演算された単精度の解x^S_kを CPU 上の倍精度の解x_kに足し込み，補正する．
- # 4　相対残差($|r_k| / |r_0|$)が1.0×10^{-9}に到達するまで，k に 1 を足し，1-3 を繰り返す．(outer_loop)

表 1　実行環境

CPU 動作周波数	3.8GHz
CPU core 数	4
Memory	DDR3-1600 Dual Chanel 15.5GB
GPU 動作周波数	800MHz
GPU core 数	384
Video Memory	DDR3-1600 Dual Chanel 512MB
OS	Windows7 64bit

5. 数値実験

5.1 実行環境

AMD Fusion APU A10-5800K を用いて実験を行った．実行環境を**表 1** に示す．

5.2 比較対象

比較対象を**表 2** に示す．比較対象は APU の CPU のみを使用し，倍精度演算で実装した”CPU”と APU の CPU のみを使用し，混合精度演算で実装した”CPU_MIX”，APU 上で倍精度実装を行った”APU”，APU 上で混合精度実装を行った”APU_MIX”とする．またすべてにおいて CPU 上の計算は OpenMP を用いて並列化し 4 個のコアを使用している．

これら 4 つの比較対象に対して収束時間の比較を行う．

対象とした問題は三次元拡散方程式で等方性問題と異方性問題を用意した．問題は AMGS ライブラリ [1] を用いて作成した．異方性問題は Z 軸方向の拡散が他の軸より 0.01 の拡散になっている．問題規模は等方性が 10^3~60^3，異方性問題が 10^3~60^3 を使用した（以下，3 乗は省略し，行列サイズまたはサイズとする）．加速係数は等方性の場合 1.0，異方性の場合 0.4 に設定した．また**表 3** に倍精度，混合精度の各レベルでの緩和法の適用回数を示す．

表 2　比較対象

CPU	倍精度 AMG 法を OpenMP で並列化 APU の CPU コアで実行
CPU_MIX	混合精度 AMG 法を OpenMP で並列化 APU の CPU コアで実行
APU	倍精度 AMG 法を OpenCL で実装 APU の GPU コアを使用
APU_MIX	混合精度 AMG 法を OpenMP+OpenCL で実装 APU の CPU コアと GPU コアを使用

表 3　各レベルでの緩和法の適用回数

	等方性	異方性
倍精度	最上層：2 回 中間層：2 回 最下層：30 回	最上層：4 回 中間層：2 回 最下層：30 回
混合精度 (inner_loop 内部)	最上層：2 回 中間層：2 回 最下層：20 回	最上層：4 回 中間層：2 回 最下層：20 回

5.3 実験結果

図 5 は等方性の各比較対象の収束時間の比較である．特に行列サイズ 100 に対して収束時間の構成を示したのが**図 6** である．**表 4** は収束時間の構成の詳細を示している．

倍精度 AMG 法と混合精度 AMG 法の反復回数について**表 5** と**表 6** に示す．**表 5** は行列サイズ10~50の等方性問題を解いた際の倍精度と混合精度の AMG 法反復回数を示す．**表 6** は行列サイズ60~100の等方性問題を解いた際の倍精度と混合精度の AMG 法反復回数を示す．ここで D は倍精度，M は混合精度を表す．

等方性問題では，計算時間を行列サイズ 100 で CPU と APU_MIX を比較すると約 2.4 倍，APU と APU_MIX を比較すると約 1.7 倍高速化した．これはすべての level での AMG 法の演算時間が削減されたことによる．APU_MIX は CPU，GPU 間のデータ転送が最も多いが，APU の特性上データ転送時間（refinement に計上されている）はボトルネックとなっていない．

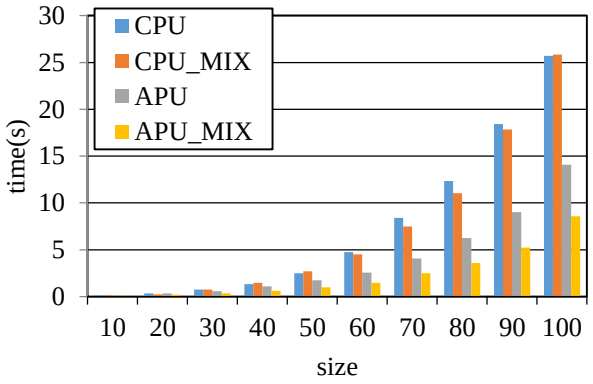

図 5　サイズ 10~100 等方性 AMG 法 実験結果

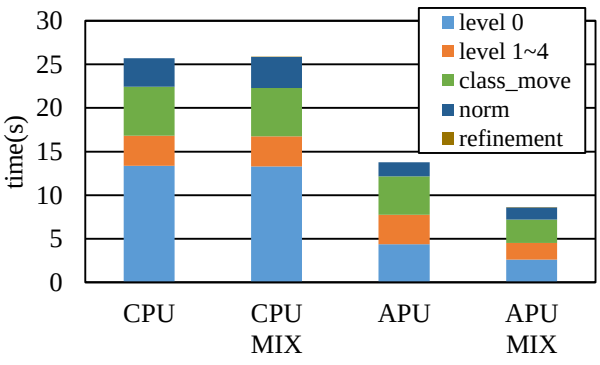

図 6　行列サイズ 100 の収束時間の構成

1 AMGS ライブラリとは工学院大学高性能計算研究室が公開している AMG 法ライブラリ群である [8].

表 4　収束時間の構成の詳細

refinement	解を単精度から倍精度に補正する計算時間（混合精度 AMG 法のみ）
norm	収束判定のための計算時間
class_move	階層移動の計算時間
level 4	level 4 での計算時間
level 3	level 3 での計算時間
level 2	level 2 での計算時間
level 1	level 1 での計算時間
level 0	level 0 での計算時間 最も大きな行列を扱う

等方性問題では行列サイズ 50 と 100 で CPU_MIX の収束時間が CPU の収束時間よりも遅いという結果になっている．**図 6** の収束時間の構成をみると，AMG 法の演算時間がほとんど変わらないことから，混合精度実装によるデータ転送量削減効果の恩恵を受けても．**表 5**，**6** より反復回数の増加が原因となり結果として収束時間は長くなったと考えられる．

APU MIX の場合 inner_loop 内での norm 計算, class_move, 各 level の計算は GPU で行われる．GPU 側の演算においても収束判定を CPU 上で行っているため, 演算時間内に通信時間も含まれる．

表 5　10~50の等方性問題の AMG 法反復回数

サイズ	10		20		30		40		50	
レベル数	2		3		4		4		4	
精度	D	M	D	M	D	M	D	M	D	M
CPU_loop	17		24		26		29		27	
inner_loop		26		35		35		41		43
outer_loop		3		3		3		3		3

D：倍精度　M：混合精度

表 6　60~100の等方性問題の AMG 法反復回数

サイズ	60		70		80		90		100	
レベル数	5		5		5		5		5	
精度	D	M	D	M	D	M	D	M	D	M
CPU_loop	31		35		34		35		36	
inner_loop		41		46		46		49		54
outer_loop		3		3		3		3		3

D：倍精度　M：混合精度

図 7 は異方性の各比較対象の収束時間の比較結果である．特に行列サイズ 60 に対して収束時間の構成を示したのが**図 8** である．

倍精度 AMG 法と混合精度 AMG 法の反復回数について示す．**表 7** は行列サイズ10~30の異方性問題を解いた際の倍精度と混合精度の AMG 法反復回数を示す．**表 8** は行列サイズ40~60の異方性問題を解いた際の倍精度と混合精度の AMG 法反復回数を示す．

異方性問題では，計算時間を行列サイズ 60 で CPU と APU_MIX を比較すると約 3.0 倍，APU と APU_MIX を比較すると約 1.7 倍高速化した．こちらも等方性問題と同様の理由である．

異方性問題では CPU と CPU_MIX はどの行列サイズにおいても CPU_MIX の収束時間が短いという結果になった．**図 8** をみると，すべてのレベルにおける AMG 法の演算時間が短縮されていた．**表 7**，**8** より等方性問題と比べて反復回数の増加の割合が低いことから，単精度浮動小数点データを使用し，データ転送量を削減した恩恵が大きい．

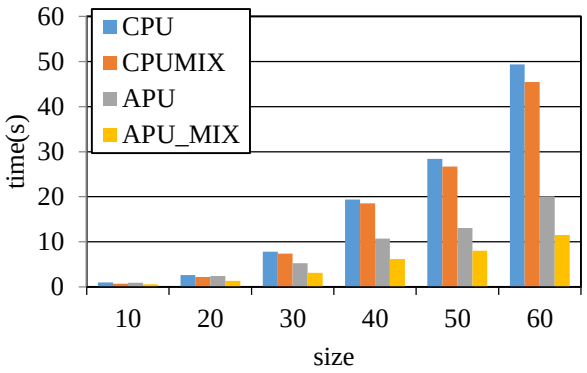

図 7　サイズ 50~60 異方性 AMG 法 実験結果

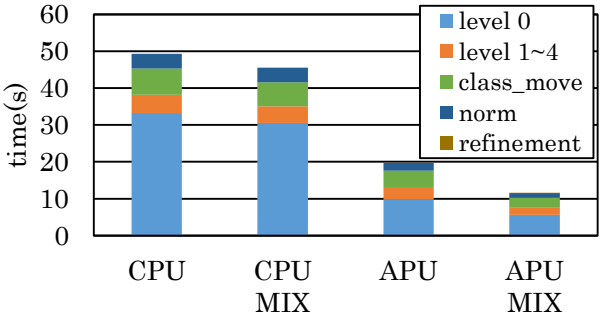

図 8　行列サイズ 60 の収束時間の構成

表 7　10~30の異方性問題の AMG 法反復回数

サイズ	10		20		30	
レベル数	2		3		4	
精度	D	M	D	M	D	M
CPU_loop	117		162		231	
inner_loop		161		217		314
outer_loop		3		3		3

D：倍精度　M：混合精度

表 8　40~60の異方性問題の AMG 法反復回数

サイズ	40		50		60	
レベル数	4		4		5	
精度	D	M	D	M	D	M
CPU_loop	272		217		211	
inner_loop		375		291		280
outer_loop		3		3		3

D：倍精度　M：混合精度

6. 収束条件の最適化

次に inner_loop と outer_loop の関係について考察する．**表 5~8** より行列サイズや等方性問題，異方性問題に関わらず outer_loop の回数が 3 回と不変であることがわかる．そこで次の実験を行った．

6.1 実験

inner_loop の収束条件を 10^{-4}~10^{-1}，outer_loop の収束条件を 10^{-7}~10^{-12} とそれぞれ変化させ，収束時間を測定，比較した．

6.2 結果

図 9 は行列サイズ 60 の等方性問題において，inner_loop の収束条件を 10^{-4}~10^{-1}，outer_loop の収束条件を 10^{-7}~10^{-12} と変化させた収束時間の比較である．**表 9** は outer_loop の収束条件ごとの inner_loop の収束条件の最適値である．outer_loop の収束条件ごとに inner_loop の収束条件の最適値が変化しているのがわかる．さらに inner_loop の収束条件を固定して，outer_loop の収束条件を変化してみると収束時間が規則的に変化していることがわかる．例として inner_loop の収束条件を 10^{-4} と固定する．outer_loop の収束条件が 10^{-7} から 10^{-8} なると収束時間が長くなるのに比べ，10^{-8} から 10^{-9} や 10^{-10}，10^{-11} になっても収束時間が長くなっていない．そして 10^{-12} になるとまた収束時間が長くなる．このような性質を inner_loop の収束条件ごとに見ることができる．

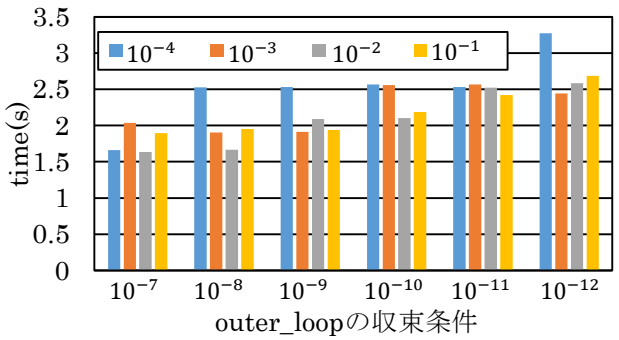

図 9　収束条件ごとの収束時間（等方性問題）

表 9　等方性問題における inner_loop の収束条件の最適値

outer_loop の収束条件	inner_loop の収束条件の最適値
10^{-7}	10^{-2}
10^{-8}	10^{-2}
10^{-9}	10^{-3}
10^{-10}	10^{-2}
10^{-11}	10^{-1}
10^{-12}	10^{-3}

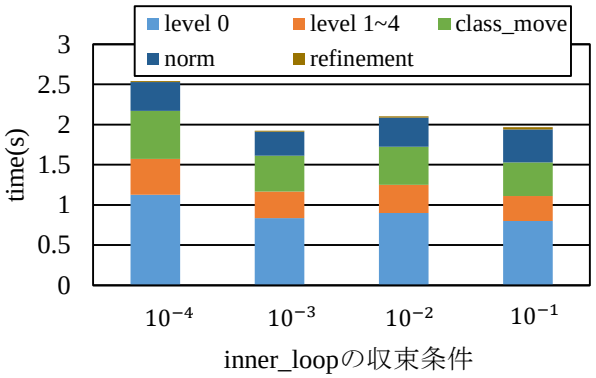

図 10　outer_loop の収束条件 10^{-9} の収束時間の構成

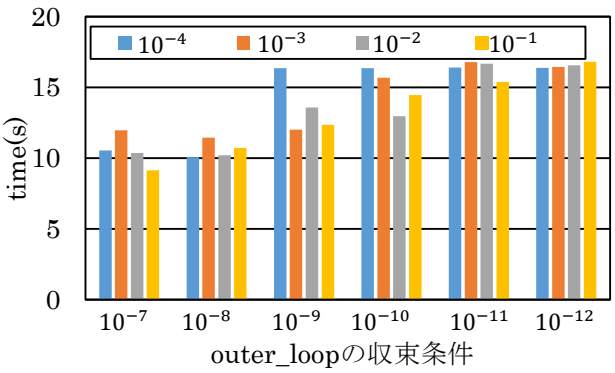

図 11　収束条件ごとの収束時間（異方性問題）

表 10 異方性問題における inner_loop の収束条件の最適値

outer_loop の収束条件	inner_loop の収束条件の最適値
10^{-7}	10^{-1}
10^{-8}	10^{-4}
10^{-9}	10^{-3}
10^{-10}	10^{-2}
10^{-11}	10^{-1}
10^{-12}	10^{-4}

図 10 は等方性問題の outer_loop の収束条件を 10^{-9} とした際の収束時間の構成を示す．outer_loop の収束条件が 10^{-9} で inner_loop の収束条件が 10^{-4} と 10^{-3} で比較すると，最上層での計算時間や階層移動，単精度での残差計算の時間が inner_loop の収束条件が 10^{-3} の場合では短縮されている．

次に異方性問題について考える．**図 11** は行列サイズ 60 の異方性問題において，先ほどと同様の比較である．こちらも等方性と同様に outer_loop の収束条件ごとの inner_loop の収束条件の最適値を**表 10** に示す．異方性問題ではっきりと outer_loop の収束条件に対する inner_loop の収束条件の最適値の関係性を確認することができた．inner_loop の収束条件の指数部が outer_loop の収束条件の指数部の約数になっている場合，収束時間が最も高速である．その理由は，outer_loop 1 回に対して outer_loop の相対残差が inner_loop の収束条件に比例して減少していくためである．inner_loop の収束条件が 10^{-2} である場合は，outer_loop 1 回に対して outer_loop の相対残差は 2 桁ずつ減少していく．つまり outer_loop の収束条件が 10^{-7} の場合に inner_loop の収束条件を 10^{-2} にしてしまうと，outer_loop を 4 回行ってしまう．そのため相対残差は 10^{-8} まで収束していることになり，無駄な計算を行い，収束時間が長くなってしまう．

また outer_loop の収束条件が 10^{-8} や 10^{-12} であるとき，inner_loop の収束条件の最適値が 10^{-2} ではなく，どちらも 10^{-4} であることから最大の約数になっている場合が最も高速であるとわかる．**表 11** に outer_loop の収束条件が 10^{-8} と 10^{-12} である場合に inner_loop の収束条件を 10^{-2} と 10^{-4} と設定した際の inner_loop と outer_loop の回数を示す．

表 11 収束条件が 10^{-8} と 10^{-12} のときの loop 回数

			inner_loop の収束条件	
			10^{-2}	10^{-4}
outer_loop の収束条件	10^{-8}	inner	247	246
		outer	4	2
	10^{-12}	inner	386	384
		outer	6	3

inner_loop と outer_loop の回数は収束条件が 10^{-2} と 10^{-4} で比べると，どちらも収束条件が 10^{-2} に設定している方が多い．inner_loop の回数が増加すると，AMG 法の計算時間が増加する．outer_loop が増えると残差計算と収束判定の処理が増えるため，収束時間の増加につながる．そのため，inner_loop の収束条件の指数部が outer_loop の収束条件の指数部の最大の約数になっている場合が，inner_loop の収束条件の最適値である．

図 12 は異方性問題の outer_loop の収束条件を 10^{-9} とした際の収束時間の構成を示す．等方性問題同様に最上層での計算時間や階層移動，単精度での残差計算の時間が inner_loop の収束条件 10^{-3} の場合では短縮されていることがわかる．

等方性問題では異方性問題のように inner_loop の収束条件の指数部が outer_loop の収束条件の指数部の最大の約数になっている場合が，inner_loop の収束条件の最適値になっていない．その理由として，inner_loop1 回に対する収束性と outer_loop1 回に対する収束性の違いが挙げられる．異方性問題では outer_loop1 回で inner_loop の収束条件の指数部桁だけ相対残差が減少する．つまり inner_loop の収束条件が 10^{-2} である場合は，outer_loop 1 回に対して outer_loop の相対残差は 2 桁ずつ減少していく．しかし等方性問題ではほとんどの場合，このようになっていない．

そこで，等方性問題の行列サイズ 60，outer_loop の収束条件を 10^{-8} とし，inner_loop の収束条件を 10^{-4} とした場合の outer_loop の回数と outer_loop の相対残差を**表 12** に示す．inner_loop の収束条件を 10^{-4} としているので，outer_loop 1 回に対して outer_loop の相対残差は 4 桁ずつ減少していくはずである．outer_loop1 回目は正常に相対残差が $1.0*10^{-4}$ より小さくなっている．しかし 2 回目の outer_loop では相対残差が $1.0*10^{-8}$ より小さくなるはずが，そうなっていない．$1.0*10^{-8}$ より小さくなったらという収束判定なため，outside をもう一度 loop させてしまい，無駄な計算が発生している．上記が原因となり，等方性問題では異方性問題と同様の性質を示さない．

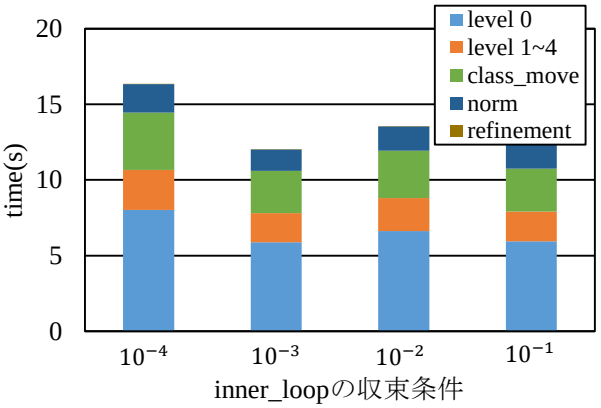

図 12 outer_loop の収束条件 10^{-9} の収束時間の構成

表 12　outer_loop の回数と outer_loop の相対残差

outer_loop の回数	outer_loop の相対残差 (収束条件)
1	$9.95*10^{-5}$ ($< 1.0*10^{-4}$)
2	$1.10*10^{-8}$ ($> 1.0*10^{-8}$)
3	$1.07*10^{-11}$ ($> 1.0*10^{-12}$)

7. おわりに

本研究では混合精度実装手法を APU 上で AMG 法に実装した. 一部に GPU が得意とする単精度演算を用いることにより，すべてを倍精度で実装した結果と比較して等方性問題の行列サイズ 100^3 で約 1.7 倍，異方性問題の行列サイズ 60^3 で約 1.7 倍の高速化を確認した. 解の精度も単精度の解を倍精度に補正する手法により，すべてを倍精度で実装した結果と同等の精度を得ることができた.

さらに inner_loop と outer_loop の関係について考察を行った．inner_loop の収束条件を 10^{-4}~10^{-1}，outer_loop の収束条件を 10^{-7}~10^{-12} とそれぞれ変化させ，収束時間を測定，比較した．その結果異方性問題では，inner_loop の収束条件の指数部が outer_loop の収束条件の指数部の最大の約数になっている場合，収束時間が最も短いと確認することができた．つまり outer_loop の収束条件が決まれば，inner_loop の収束条件を最適化することができる．しかし等方性問題では異方性問題と異なった性質を確認したため, inner_loop の収束条件の最適化には更なる分析が必要となる.

inner_loop の収束条件を変化させていき，最も収束時間が減少したのは outer_loop の収束条件が 10^{-9} で行列サイズ 60^3 の異方性問題であった．inner_loop の収束条件が 10^{-4} と 10^{-3} の収束時間を比較すると収束時間を約 30%短縮できた.

今後の研究では，OpenCL 実装での可搬性を活かして APU 上のみでなく, Intel Ivy Bridge など他のプロセッサや, 外部 GPU を使用するような環境でも混合精度 AMG 法のデータ転送時間の影響などを考察する予定である．また混合精度手法を AMG 法以外の実際に多く利用されている解法へ実装していきたい.

謝辞　本研究は JSPS 科研費 24650014 の助成を受けたものです.

参考文献

[1]　藤井昭宏，小柳義夫，科学技術シミュレーションにて多用される代数的多重格子法の評価, シミュレーション　第 28 巻第 4 号, pp.9-14(2009) .
[2]　AMD:：http://www.amd.com/us/Pages/AMDHomePage.aspx.
[3]　Pierre Boudier, Graham Sellers: MEMORY SYSTEM ON FUSION APUS The Benefits of Zero Copy,AMD Fusion DEVELOPER SUMMIT (2011) .
[4]　Stone JE, Gohara D, Shi G. OpenCL: A parallel programming standard for heterogeneous computing systems. Comput. in Sci. and Eng. 2010;12:66–73..
[5]　AMD　: Accelerated Parallel Processing OpenCL Programming Guide,http://developer.amd.com/download/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf.
[6]　Richard Barrett, Tony F. Chan, Jone Donato, Michael Berry, James Demmel: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods（訳）長谷川 里美, 長谷川 秀彦, 藤野清次: 反復法 Templates,朝倉出版 (1996).
[7]　Alfredo Buttari, Jack Dongarra, Julie Langou, Julien Langou, Piotr Luszczek, and Jakub Kurzak : Mixed Precision Iterative Refinement Techniques for the Solution of Dense Linear Systems, International Journal of High Performance Computing Applications November 2007 vol. 21 no. 4 457-466.
[8]　AMGS ライブラリ : http://hpcl.info.kogakuin.ac.jp/olab/software.