AMGS マニュアル

工学院大学　情報学部コンピュータ科学科

高性能計算研究室

最終更新日：2015 年 5 月 25 日

目 次

1　はじめに

AMGS(Algebraic Multi-Grid Solvers) は大規模疎行列係数の線型方程式

$$Ax = b$$

に対する代数的多重格子法（以下 AMG 法）で解くライブラリである．

言語は主に Fortran90 で記述されており，GPU も利用できるようにするため，MPI,OpenMP,C++,CUDA を用いている。逐次版（Fotran90, C++, CUDA）もあるが，AMG 法の収束性能検証用に置いている。

- 並列版では AMG 法の中でも有力な Smoothed Aggregation に基づく AMG 法について，ハイブリッド並列化がされており，高並列環境においても効率的に動作する．特に並列アグリゲート戦略では，このライブラリ固有のものを選択することができ、数千から数万プロセスでも効率的に動作することを確認している．
- 反復解法部だけだが、CUDA を用いた実装があり，ユーザ側のコードを変更すること無く，GPU 版のソルバも呼び出すことができる．GPU 版のコードは MPI 並列版もあり，GPU クラスタ上でも動作する．
- グラフ分割ライブラリ ParMETIS[1] に基づいた疎行列の分散ルーチン，ベクトルのリオーダリングルーチンが実装され　ており，行列のオーダリングや分散データなどを意識することなく，疎行列を適宜（再）分散して線形ソルバを並列に処理させることが可能である．
- 逐次版を利用することで，複数種類の AMG 法と各種パラメタを変更できる本ライブラリの逐次版を利用することで，問題に応じて AMG 法の有効性を評価できる．逐次版 AMGS では以下の 5 つの代表的な AMG 法を実装している．
  - Stüben らにより提案されている標準的な手法 [4, 5]
  - Stüben らにより提案された Aggresive Coarsening による手法 [4, 5]
  - Standard AMG 法 [8, 9]
  - Adaptive AMG 法 [9]
  - Smoothed Aggregation に基づく AMG 法 [10]

本ライブラリで利用できる解法は以下のようになる．

並列版： AMG 法，AMG 前処理付 CG 法，AMG 前処理付 BiCGSTAB 法，AMG 前処理付残差切除法

逐次版： AMG 法，AMG 前処理付 CG 法，AMG 前処理付 BiCGSTAB 法，AMG 前処理付 GMRES(m) 法, IDR(s)-based AMG 法，AMG 前処理付残差切除法

※問題行列の値は非対称になっていても良いが，非零構造が対称のもののみ対応している．

2　インストール

はじめにインストール方法とテストプログラムの実行方法まで取り上げ，次に，テストプログラムの内容を記述する．逐次版 (AMGS-($version)) と並列版 (AMGS-p-($version)) は別のライブラリとなっているが，インストール方法やテストプログラムの実行方法はほとんど共通である．$version は version を表し，AMGS-[p-]($version) は並列版もしくは逐次版のライブラリを表す．

Fortran コンパイラとしては、intel ifort 及び gfortran(gcc 4.8.3) を利用して動作確認をしている．また ParMETIS は 4.0.3 を利用している．

2.1 インストール方法

AMGS は Fortran90 の線形解法のライブラリであり，Fortran90 コンパイラが必要となる．もし GPU 版のソルバもインストールする場合はさらに C++のコンパイラと nvcc が必要となる．また時間計測関数として OpenMP の時間計測関数を利用しているため，逐次板では OpenMP 化はしていないが，リンク時に OpenMP を有効にする必要がある．

並列版では MPI-1 が利用できる環境が必要で，OpenMP によるスレッド並列化も組み合わせることができる．また，並列版の中に行列を分散させるモジュールがあるが，その中で ParMETIS をリンクさせることもできる．その場合の手順も以下 2 に示す．

テストプログラムを動かすまでの手順は以下のようになる．

1. ファイルを解凍し，生成されたフォルダ amgs-[p-]($version) に移動する．

   ```
   > gunzip -c amgs-[p-]($version).tar.gz | tar xvf -
   ```

2. amgs-[p-]($version)/Makefile.in を環境に応じて変更する．GPU 版のソルバを有効にするときはこのファイルで CUDA=true とする．また並列版で ParMETIS も利用する場合は このファイルで，PARMETIS=true とする．PARMETIS_DIR に ParMETIS がインストールされているディレクトリを指定する．

   FC は Fortran90 のコンパイラで，amgs-[p-]($version)/src/Makefile の FFLAGS でコンパイラに最適化オプションなどを設定する．ここで-DPRINT_REZ を追加すると，各反復で相対残差が計算されているときはそれを表示させるように設定できる．このオプションを　付けないと何も表示を行わない．並列版ではこの FFLAGS で OpenMP 版が呼び出されるかどうかが決まる．また sample の下の Makefile ではライブラリをリンクするため，FFLAGS でモジュールパスの指定をする必要がある．並列版サンプルプログラムでは MPI プログラムの実行方法が

   amgs-p-($version)/sample/Makefile の MPIRUN や SET_NUM_THREADS に規定されているため必要に応じて変更する．

3. コンパイル

   ```
   > cd amgs-[p-]($version); make
   ```

4. テストプログラム（２次元問題領域でポアソン方程式の求解）を実行する．並列版では初めに問題を分割した後で，2 並列で実行する．

   ```
   > make check
   ```

2.2 テストプログラム

2.2.1 逐次版

amgs-($version)/sample ディレクトリにおいて，

```
> echo "300"|./sample_solver
```

と入力すると 2 次元 Poisson 方程式を 5 点中心差分で離散化して得られる行数 300^2 の 5 重対角行列を係数とする線型方程式 $Ax = b$ を解く．右辺ベクトル b は近似解 x がすべて 1 になるように設定している．また，相対残差の 2 ノルムが 1.0E-7 より小さくなったときに収束としている．実行結果を以下に示す．

300x300 の結果

```
V-cycle
GS
SA
      15117  aggregates are selected from        90000
       1695  aggregates are selected from        15117
        193  aggregates are selected from         1694
         24  aggregates are selected from          191
nonzero element complexity:    1.36885249554367
AMGCG gs GS dump parameter=   1.00000000000000       CYCLE: 0:V,1:F,2:W :
          0 theta:  5.000000000000000E-002
   1    7.181938E-02
   2    2.203099E-02
   3    5.565692E-03
   4    1.612333E-03
   5    3.905739E-04
   6    1.013073E-04
   7    2.669294E-05
   8    7.836431E-06
   9    2.187155E-06
  10    7.128158E-07
  11    2.216938E-07
  12    6.182046E-08
```

この出力には、解法や各レベルのサイズ，相対残差の履歴が表示される．今回の場合，SA という AMG 法に基づいて，AMGCG 法が利用され，90000 次元の問題からより次元数の小さい問題が 4 つ (それぞれの次元数が 15117，1695，193，24) 生成されたことが分かる．

2.2.2 並列版

amgs-p-($version)/sample ディレクトリにおいて，

```
> export OMP_NUM_THREADS=2; mpirun -np  2 ./sample_solver
```

と入力すると 2 次元 Poisson 方程式を 5 点中心差分で離散化して得られる行数 500^2 の 5 重対角行列を係数とする線型方程式 $Ax = b$ を 2 プロセスに行列を分散した後に，並列に解く．問題サイズは sample/sample.F90 の width に記述されており，この変数の値を変更すれば $width^2$ のサイズの問題に変更できる．また行列を 1 プロセスで生成したのち，ＭＰＩプロセス全体に分散するように実装しているため，ＭＰＩプロセス数は自由に選択できる．右辺ベクトル b は解 x がすべて 1 になるように設定している．また，相対残差の 2 ノルムが 1.0E-7 より小さくなったときに収束としている．実行結果は逐次版と同様となる．

OMP_NUM_THREADS を指定しないで，MPI を実行すると各プロセスに対してそれぞれコア数分のスレッドが生成され，かえって遅くなることがあるため，OMP_NUM_THREADS は必ず設定して MPI の実行をするようにする．

3 ソルバの利用方法

AMG 法はデータ生成部と反復部からなる．問題行列の登録や，パラメタの設定も必要であり，ソルバを呼び出す場合は 以下のような 5 種の関数を呼び出す．逐次版は amgs_module に，並列版は amgs_p_module に Fortran 90 モジュールとして以下の関数が登録されている．

1. 解法・パラメタの設定関連：
   amgs_parameter_option(ファイル IO 番号, パラメタ設定文字列) 《並列版のみ》
   amgs_parameter_option_file(ファイル IO 番号, ファイル名, MPI_COMM)《並列版のみ》
   amgs_parameter_option_info() 《並列版のみ》
   amgs_parameter(ファイルの IO 番号)

2. 問題行列の登録に関連する関数：
   amgs_store_problem_matrix(CRS 形式の行列) など

3. 収束条件設定：
   amgs_terminal_val_set(相対残差の閾値) など

4. 反復解法部：
   amgs_solver(解ベクトルや右辺ベクトルの指定)

5. 問題行列と階層構造の破棄：
   amgs_clear_matrices()

はじめにライブラリの構造と機能について紹介したのち，それぞれの関数について説明する．

3.1 ライブラリの構造と機能

多様なデータ構造に対応するため，ライブラリ側には一度行列を CRS 形式で登録し，その後，解ベクトルと右辺ベクトルを指定してソルバを起動する手順で利用する．これにより，右辺ベクトルのみ変化し繰り返し問題を解くケースでは行列を登録し直すこと無くそのまま使いまわせる．

AMG 法はマルチレベルな構造を生成するが，はじめにソルバが呼ばれたタイミングで生成する．問題行列を削除されない限り，生成されたマルチレベルな構造はそのまま維持されるため，上記のように右辺ベクトルのみ変化がある場合はマルチレベルの構造も使いまわせる．

ライブラリの機能としては以下のものがある．

- 行列の分散機能
- ソルバの発散状況のチェック機能
- 自動チューニング機能（暫定の機能）
- ブロック化版ソルバを適用する機能（並列版のみの暫定の機能）

行列の分散機能は，ParMETIS などを利用して再分散し，それぞれを MPI プロセスに割り当てることで，行列とプロセスの割り当てを最適化する機能である．これにより解ベクトルや右辺ベクトルのリオーダリングなどを考慮する必要がでてくるが，ライブラリ側で行うことでユーザは意識せずに利用できる．

次にソルバの発散状況のチェック機能は，AMG.dat により指定される．この機能によりソルバを終了するときに真の残差を計算しなおし，右辺ベクトルより残差が大きい場合は反復回数を-1 にセットする．これによりユーザに収束していない状況に対応できる．

自動チューニング機能は同じような問題に繰り返しソルバを適用するときにパラメタの最適化を進めていくものである．ユーザが AT.dat にパラメタの探索範囲を指定し，ライブラリはその中でデフォルトのパラメタからソルバが起動されるたびにパラメタを変えて実行していき，時間計測を行い，パラメタの最適性を評価する．これにより，繰り返しソルバを呼ぶ回数が多い場合は，最適なパラメタに近づき，より高速になると考えられる．

最後にブロック化版ソルバについて説明する．これは AMG.dat を経由してブロック幅 NB を指定することで，CRS 行列をそのブロック幅でブロック化し，ブロック版 SA-AMG 法を適用する．ブロック版

SA-AMG 法では各要素は $NB \times NB$ のサイズの要素となり，通常の SA-AMG 法とは異なる解法となる．ブロック版 SA-AMG 法では粗いレベルで近似するベクトルとしてブロック内の各次元方向に定数のベクトルをよく近似するようにレベル間演算子を作成する．例えば，3 次元弾性体の問題は各次元方向の定数ベクトルは緩和法では収束しにくい誤差成分となるため，粗いレベルでしっかり近似し，打ち消せると収束が速くなる．そのため，3×3 版 AMG 法を適用すると高速に収束する．現状の暫定版ではブロック化の処理と行列の分散が同時にはできないため，注意する．

3.2 解法・パラメタの設定

ここでは下の三つの関数を説明する．
subroutine amgs_parameter_option(ファイル IO 番号, MPI_COMM, パラメタ設定文字列) 《並列版のみ》
subroutine amgs_parameter_option_file(ファイル IO 番号, ファイル名, MPI_COMM) 《並列版のみ》
subroutine amgs_parameter(integer FILEIO)

並列版のみアグリゲート生成戦略など詳細なパラメタの値を文字列で設定できる関数が用意されている．
これにより、実行時に動的にパラメタを変更することが可能となる．
また逐次版と並列版共通なものとして，解法やパラメタの選択をファイルで設定する amgs_parameter(ファイルの IO 番号) 関数も用意されている．この関数はカレントディレクトリのパラメータファイル AMG.dat を読み出し AMG 法の各種パラメータを設定する．もし自動チューニングが有効になっている場合は，AT.dat も必要となる．このファイルにチューニングするパラメタの範囲が記述されている．
AMG 法の種類は，並列版では SA のみ実装されているが，逐次版では ST1, ST2, ST3, AD, SA の 5 つの AMG 法を選択できる．それぞれ，以下のような解法である．

- ST1: Stüben による AMG 法で標準的な設定の解法 [4, 5]
- ST2: Stüben による AMG 法で aggressive coarsening を適用した解法 [4, 5]
- ST3: Brandt らによる Standard AMG 法と呼ばれる解法 [8, 9]
- AD: Adaptive AMG 法 [9]
- SA: Smoothed Aggregation に基づく AMG 法 [10]

はじめに上記の文字列でパラメタを設定する場合を説明し，その後，AMG.dat について紹介し，次にライブラリの自動チューニング機能や，ベクトル版ソルバ，GPU ソルバの設定方法や制約について説明する．

3.2.1 文字列によるパラメタ設定方法《並列版のみ》

以下の二つの関数があり，文字列を関数の引数としてパラメタ設定をする場合は上の関数 1，ファイル経由で文字列を入力し設定する場合は下の関数 2 を呼び出す．

1. amgs_parameter_option(ファイル IO, MPI_COMM, 文字列)
2. amgs_parameter_option_file(ファイル IO, ファイル名, MPI_COMM)

これらの関数では事前に設定されているパラメタから変更したい部分のみ文字列で指定する．例えば、以下のような関数呼び出しがあると強連結成分の閾値は 0.04 にリセットし、ソルバは CG 法を利用することを意味する．

```
---
call amgs_parameter_option(10, SOLVER_COMM,                    &
       & " -strong_con_threshold=0.04  -krylov_solver=CG ")
---
```

これをファイルで指定する場合は例えば下のような文字列だけのファイル AMG_OPTION.dat を作成し，ファイル名”AMG_OPTION.dat” を引数にして上の関数 2 を呼び出せば上と同様のパラメタ設定をしたことになる。-パラメタの種別＝値という形で指定するが，パラメタの種別ごとの間の空白の部分では改行しても問題ない．各行で # 以降はコメント文として対処する．

```
--- AMG_OPTION.dat ---
 -strong_con_threshold=0.04  -krylov_solver=CG # comments...
--- END ---
```

設定できるパラメタの種類は次節の AMG.dat と同様のものがあるが，amgs_option_info(0) を呼び出せば設定できるパラメタのリストを表示する．また引数を 1 にして呼び出せば，現在設定されているパラメタリストを表示する．

パラメタ設定項目としては以下ののようになる．

マルチグリッドサイクル
```
 -cycle:                0,1,2 :: V,F,W
```

スムーザー
```
 -smoother:             sgs,gs,mcsgs,mcgs
```

レベルを下るときのスムーザの反復回数
```
 -iter_down:            #
```

レベルを上がるときのスムーザの反復回数
```
 -iter_up:              #
```

最も粗いレベルでのスムーザの反復回数（LU 分解を適用しない場合）
```
 -iter_bottom:          #
```

F サイクルで最初にレベルを下るときのスムーザの反復回数
```
 -f_first_iter_down:    #
```

SA-AMG の強連結の閾値
```
 -strong_con_threshold: #: 0.0-0.15
```

アグリゲートをスムーズにする際の減速ヤコビの係数
```
 -dump_jacobi_coef:     #: 0.5-0.8
```

スムーザの加速係数
```
 -smoother_accel_coef:  #: 0.2-1.4
```

外側のソルバーの解法。NO は AMG 単体を示す
```
 -krylov_solver:        NO, CG, BICGSTAB, RESCUT
```

RESCUT を選択したときのリスタート周期
```
 -restart:              #: 2-10
```

行列を分散させる場合の分散手法

```
 -dist_method:          FIX, PARMETIS
```

収束チェックを行うかどうか

```
 -convergence_check:    #: 1 or 0
```

自動チューニング関連のパラメタ（今は未整備）

```
 -parameter_at:         AT, NOAT
 -at_init:              #
 -at_diff:              #
 -at_interval:          #
 -at_setup:             #: 1 or 0
```

ベクトル問題を解く場合のフラグ

```
 -vector_problem:       #: 1-10: NB=1 => SCALAR problem
```

アグリゲート戦略：

w_lu で終わっているものは最も粗いレベルで LU 分解
w_o_lu で終わっているものは最も粗いレベルでも反復解法
coupled ではじまっているものは、領域境界でアグリゲート共有して生成
independent ではじまっているものは領域ごとに独立にアグリゲート生成
　但し，粗くなって領域が狭くなると近くの領域をくっつける
old_independent では独立アグリゲーションのみで粗くなっても他の領域を
くっつけたりしない。

```
 -aggregation:   coupled_w_lu, coupled_w_o_lu
                 independent_w_lu, independent_w_o_lu
                 old_independent_w_o_lu, old_independent_w_lu
```

領域をくっつける場合に 1 プロセスあたり平均この個数のアグリゲートが残るように
する

```
 -process_aggre_threshold: #: 100-5000
```

最も粗いレベルの基準
ここで指定した値より全体の数が小さくなったらこれ以上、生成しない

```
 -min_node_size:        #:
```

最大レベル生成数
ここで指定した数がレベル生成数の上限。上の min_node_size にひっかかれば
より小さいサイズ

```
 -max_level_size:       #: 2-10
```

3.2.2　パラメタ設定ファイル AMG.dat について

本節では，AMG.dat のフォーマットを示す．このフォーマットは逐次版，並列版ともに共通である．そのため，逐次版では指定しても意味がないようなものまで入っている．

Line 1: 0:V-cycle，1:F-cycle，2:W-cycle．W-cycle では前半の V サイクル部分で全体の中央のレベルまでのぼる．

Line 2: 0:SGS，1:GS，2:マルチカラー SGS，3:マルチカラー GS．各レベルのスムーザの種類．2,3 は並列版でマルチスレッドが有効になっている場合にのみ選択可能．2 もしくは 3 が設定されていて，OpenMP が有効でない場合はマルチカラー処理がされず，GS もしくは SGS が処理される．

Line 3: 解法部で粗いレベルに移動するときの各レベルでのスムーザの反復回数

Line 4: 解法部で細かいレベルに移動するときの各レベルでのスムーザの反復回数

Line 5: 最も粗いレベルでのスムーザの反復回数

Line 6: F-cycle を選択したときに，サイクルのはじめに最も粗いレベルに到達するまでの各レベルでのスムーザの反復回数

Line 7: 未知数間の強連結を判断するための閾値．SA とそれ以外の AMG 法で適切な値が異なる．SA では 0.05 がデフォルトになっており，それ以外の解法では 0.25 がデフォルトの値となる．
逐次版では SA 以外の AMG 法も使用できるが，並列版では SA ののみ利用可．

Line 8: SA を選択したときのデータ生成部内の緩和ヤコビ法の係数

Line 9: 各レベルでのスムーザの緩和係数

Line10: 各種 AMG 法の選択
逐次：ST1，ST2，ST3，AD，SA
並列：SA で固定

Line11: AMG 前処理を組み合わせる反復法の選択．NO を選択すると AMG 法単体で問題を解く．
CPU 使用時：
逐次：CG，GMRES，BICGSTAB，RESCUT，IDR，NO（ＡＭＧ法単体）
並列：CG，BICGSTAB，RESCUT，NO（ＡＭＧ法単体）
GPU 使用時：GPUBICGSTAB，GPU（ＡＭＧ法単体）

Line12: GMRES や IDR，RESCUT において，リスタート周期の指定．RESCUT や GMRES，IDR 以外の場合は意味のない数字となる．

Line13: FIX　　：行列の分散方法
FIX:　　行番号の順番を変えずにブロック行分散
PARMETIS: ParMETIS を利用した分散

Line14: noCHECK　：CHECK にするとソルバが終了後，収束の確認を行い，全く収束していない場合は反復回数を-1 にセットする．

Line15: noAT 100 5 0 0　：AT を設定すると この場合 100 回ごとに最適化を繰り返す設定となる．5 回以上反復回数が増加したときも最適化に入る．

Line16: SCALAR 1　：ブロック版のソルバを呼ぶ場合は SCALAR を VECTOR に変更し，ブロック化のサイズを指定．

次に逐次版，並列版に分けて AMG.dat の実際の設定例を示す．

逐次版

ST1，ST2, ST3 もしくは AD の解法を選択したときの代表的な設定例．下では ST1 を選択した場合を示す．

```
---
0
0
1
```

```
1
30
1
0.25
0.666
1.0
ST1
CG
2
FIX
CHECK
noAT 100 5 0 0
SCALAR 1
---
```

SA を選択したときの代表的な設定例を示す．Line 6 の値が異なっているため注意する．

```
---
0
0
1
1
30
1
0.05
0.666
1.0
SA
CG
2
FIX
CHECK
noAT 100 5 0 0
SCALAR 1
---
```

並列版

並列版では SA（Smoothed Aggregation による AMG 法）のみ対応しており，Line 9 の設定は読み込まない．また Line 11 では，NO, CG, BICGSTAB, RESCUT のみ対応している．次に RESCUT のベクトルの本数と行列の分散のさせ方を指定が続く．この 13 行目は行列を分散させない場合も，FIX など何らかの設定をする必要がある．並列版での典型的な AMG.dat は以下のようになる．

```
---
0
0
1
1
30
```

```
0
0.05
0.666
1.0
SA
CG
2
FIX
CHECK
noAT 100 5 0 0
SCALAR 1
---
```

3.2.3 GPU 版ソルバやベクトル版ソルバ，AT 機能について

GPU 版ソルバについては，以下のような制約がある．

- サイクルは V-cycle のみ
- スムーザの種類はレベル 1 のみマルチカラーガウスザイデルでそれ以外のレベルは Jacobi の組み合わせ，と全レベル Jacobi を切り替えられる．AMG.dat のスムーザの種類（2 行目）が 0 のとき前者で，1 のとき後者となる．
- 最も粗いレベル以外のスムーザの反復回数，最も粗いレベルのスムーザの反復回数，緩和係数は AMG.dat から調整可能．それぞれ AMG.dat の 3 行目，5 行目，9 行目で指定する．
- GPU 版ソルバを呼び出すときは，AT は考慮の対象とされていないため，AT は無効にしておく．

次に暫定版であるが，VECTOR 版ソルバも用意されている．これについては以下のような制約がある．

- 対称行列のみ対象としている．
- AMG.dat では任意のブロックサイズが指定できる．
- ソルバは CG 法か，BICGSTAB 法，NO のみ選択可能．
- 現状ではライブラリ内で行列を分散させる amgs_distribute_matrix() と組み合わせて利用できない
- OpenMP 化されていない．

最後に AT については CPU 版ソルバでスカラ版のみ対応している．

- AMG.dat 内に AT 100　5　0　0　とすると 100 回おきに，最適化を繰り返す．もしくは前にソルバが適用されたときから 5 回以上反復回数が増加したときは，最適化プロセスに入る．
  - 4 つ目の値は何回目のソルバ呼び出しから AT を起動するか表す．0 の場合は最初からパラメタをいろいろ変えていく
  - 5 つ目の値は 1 か 0 で，1 の場合はマルチレベルの生成部のパラメタもチューニングの対象とする．0 の場合は 1 度だけ最適化し，後は行列生成部を変更しない．
- AT.dat のパラメタ範囲内で，デフォルトの設定は AT.dat のパラメタの値のうち，先頭のものを組み合わせたものとしている．
- AT を設定しないときは先頭の文字列を AT を以外に設定しておく．

AT を有効にすると AT.dat というファイルを読み込み，指定されたパラメ t あの範囲内でパラメタ最適化を行う．以下に AT.dat の簡単な説明を記述する．2 行目に各パラメタの候補となる値の個数が与えられていて，5，8，11，14，17，20，23 行目には各パラメタの候補となる値のリストがスペースで区切られ，指定される．2 行目の個数と 2 行目以降のパラメタの指定の個数が一致しないといけない．

```
---
theta,omega,solver,cycle,smoother_types,smoother_iteration,# of history vectors
 6     5      2      3          2              1                       2

theta
0.05 0.04 0.03 0.07 0.09 0.11

omega
1.0 0.8 1.2 1.4 1.6

solver
CG BICGSTAB

cycle 0:V 1:F 2:W
0 1 2

smoother types 0: Symmetric GS, 1: GS, 2: Symmetric-mcGS with OMP, 3: Symmetric-mcGS with OMP
0 1

smoother_iteration
1

# of history vectors
2 4
---
```

3.3 問題行列の登録

問題行列の登録は CRS 形式のデータとして登録する．非対称構造の CRS 形式のデータ構造が記述されているが，ライブラリとしては逐次版，並列版ともに非ゼロ構造は対称のもののみ対応しているので，その点に注意する．

逐次版

subroutine amgs_store_problem_matrix(integer crs_index(:), integer crs_column(:), real(kind=8) crs_val(:), integer rowsize)
本ライブラリは Compressed Row Storage(CRS) 形式で問題行列を受け取り，解法を適用する．CRS 形式は 3 つの配列 (crs_index, crs_col, crs_val) で行列を格納する．問題行列の行数を rowsize，非ゼロ要素数を nnz とする．

- 長さ nnz の倍精度配列 crs_val は行列 A の非零要素の値を行方向に沿って格納する．
- 長さ nnz の整数配列 crs_col は配列 crs_val に格納された非零要素の列番号を格納する．
- 長さ $rowsize + 1$ の整数配列 crs_index(0:) は要素番号が 0 から始まるものとする．0 番目の要素は 0 とし，i 番目の要素は配列 crs_val と crs_col の i 行目の終了位置を格納する．

$$A = \begin{pmatrix} 11 & & & \\ 21 & 22 & & \\ & 32 & 33 & \\ 41 & & 43 & 44 \end{pmatrix}$$

たとえば上記のような行列の場合，rowsize = 4 で，CRS 形式では以下のようになる．

crs_val(1 : 8)	11, 21, 22, 32, 33, 41, 43, 44
crs_col(1 : 8)	1, 1, 2, 2, 3, 1, 3, 4
crs_index(0 : 4)	0, 1, 3, 5, 8

並列版

並列版では，問題行列をソルバに登録する際に，2 通りの方法を用意している．ひとつは問題行列を逐次版と同様にひとつの行列として与えた後，プロセス数に応じて分散させる手法である．二つ目はブロック行分散された行列データと通信テーブルを直接ソルバに入力する方法である．

前者では，以下の 1 番の関数で問題行列を登録したのち，2 番目の関数で問題行列を分散させることになる．

1. subroutine amgs_store_problem_matrix(integer crs_index(0:), integer crs_column(:), real(kind=8) crs_val(:), integer rowsize, integer MPI_COMM)
2. subroutine distribute_problem_matrix()
3. subroutine amgs_store_distributed_matrix(integer crs_index(0:), integer crs_column(:), real(kind=8) crs_val(:), integer rowsize, integer vectorsize, integer neibPETOT, integer neibPE(:), integer INRECV(0:), integer INSEND(0:), integer RECVT(:), integer SENDT(:), MPI_COMM)

1 番目の関数 amgs_store_problem_matrix の先頭の 4 つの引数は CRS 行列に関連するものであり，5 つ目の引数はソルバで利用するコミュニケータである．この関数では，ひとつのプロセスのみ行列の登録し，他のプロセスでは，rowsize を 0 にしてこの関数を呼び出すことを想定している．

2 番目の関数 distribute_problem_matrix は，ライブラリのモジュール内に登録された行列を，AMG.dat に指定された方法で MPI プロセス全体に分散させる関数である．

3 番目の関数 amgs_store_distributed_matrix は，分散された行列をそのままライブラリモジュール内にコピーする関数で，はじめの 5 個の引数がそれぞれの PE の問題行列を表し，残りの 5 つの引数が通信テーブルを表す．最後の引数にソルバで使うコミュニケータを指定する．

ここで分散行列はブロック行分割を想定している．rowsize はその PE の担当する行数を表し，vectorsize は担当する未知数の更新に必要な変数集合のサイズを表す．neibPETOT には隣接しているプロセス数を入れ，neibpe(:) には隣接しているプロセスのランクが記録されている．隣接プロセスそれぞれに対して送信テーブルと受信テーブルを設定する．これは，それぞれのプロセスへのテーブルサイズを記録している，INSEND や INRECV と通信する未知数番号を記録している SENDT,RECVT を組み合わせて実現している．図 1 に 8 行の問題行列を 4 プロセスで 2 行ずつ分解した場合の通信テーブルを示す．

3.4 収束条件設定

収束判定は，残差ベクトルの各要素が与えられたベクトルの各要素より絶対値が小さくなるか，相対残差の 2 ノルムで与える．並列版、逐次版ともに共通で以下の二つの関数がある．

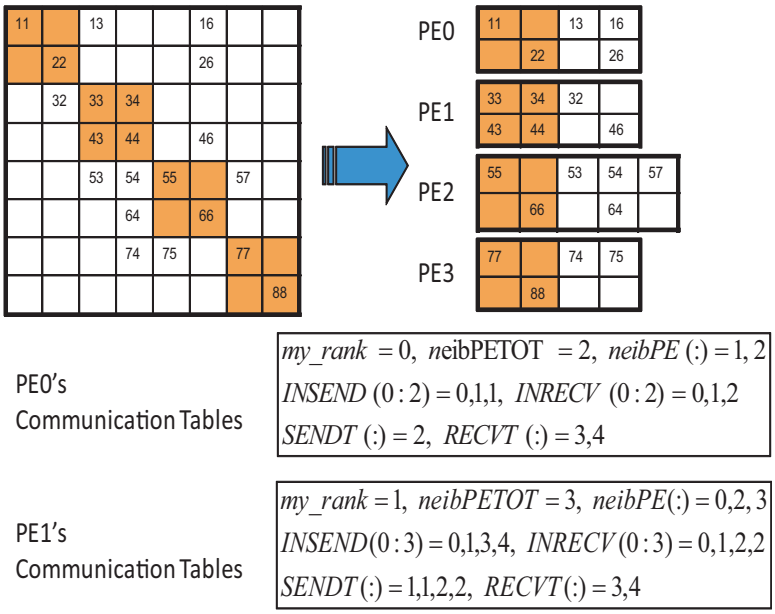

図 1: 分散行列と通信テーブル

- subroutine amgs_terminal_vec_set(real(kind=8) res_vec(:)，integer vec_size)
  res_vec は残差閾値ベクトルを表し，vec_size は残差閾値ベクトルの要素数を示す．残差ベクトルの各要素の絶対値が与えられた残差閾値ベクトルの対応する要素未満になったときに収束条件を設定する．閾値ベクトルをメモリにコピーして保持しているので，ソルバ利用の終了後などは， メモリを解放するため，amgs_terminal_vec_clear() を呼ぶ必要がある．

- subroutine amgs_terminal_val_set(real(kind=8) val)
  val は相対残差の 2 ノルムの閾値を表す．右辺ベクトルに対する相対残差の 2 ノルムの閾値を登録する．

収束条件は反復解法部を呼ぶ前に，収束判定の方法を設定すればよい
収束条件とは別に，反復回数の上限をソルバの引数に与えることができる．ソルバは以下の三つのうち，いずれかのときに終了する．

- あらかじめ設定された収束条件に到達した場合
- 反復回数の上限で打ち切られる場合
- 明らかに発散してしまった場合 (相対残差を計算しており，かつ相対残差が 1.0E5 を超えた場合)

3.5 ソルバの呼び出し

subroutine amgs_solver(real(kind=8) X(:), real(kind=8) B(:), integer iter, logical SETUP_FL)
並列版，逐次版ともに共通で amgs_solver 関数を呼び出すと，指定された解法，収束条件で問題を解く．問題行列が登録された後，はじめてこの関数が呼ばれたタイミングで，マルチレベルが生成される．この関数の X(:) と B(:) はそれぞれ，解ベクトルと右辺ベクトルを表す．また iter は反復回数の上限を指定する．この関数が終了時にはこの変数にソルバの反復回数が返される．SETUP_FL はこの関数内で行列の階層構造の削除を行うかどうかを表すフラグである．このフラグが真のときはこの関数内で階層構造が生成され，行列の破棄まで行われる．

3.6 問題行列と階層構造の破棄

subroutine amgs_clear_matrices()
この関数により登録した問題行列と，生成した階層構造を破棄することができる．問題行列を変更する場合や，実行を終了するときに，利用する．

4 サンプルプログラム

300×300 の領域の 2 次元 5 点差分のポアソン方程式を解き，結果として相対残差の履歴を表示するプログラムを示す．コンパイル方法の詳細については，sample フォルダ内の Makefile を参照してください．

4.1 逐次版

サンプルプログラムの 30 行目までで，5 点差分の方程式を CRS 形式にしており，その後，31 行目から 36 行目までで 3 節にある関数を順次呼び出し，AMG 法を適用している．

amgs-($version) フォルダにインストールした場合，コンパイル，リンクは以下のようにできる．計算環境に応じて，f90 の部分と Fortran90 のモジュールパスの指定方法を変更する必要がある．

```
> f90 sample_code.f90 amgs-($version)/amgs_s.a -module amgs-($version) -openmp
```

CUDA も有効にしてソルバをコンパイルするときは CUDA のライブラリもリンクする必要がある．CUDA のライブラリの場所なども環境によって異なることが多い．

```
> f90 sample_code.f90 amgs-($version)/amgs_s.a -module amgs-($version) -openmp \
  -lcutil_x86_64 -lcudart -L/usr/local/cuda/lib64
```

逐次版サンプルプログラム: sample_code.f90

```
 1  program sample
 2    use amgs_module
 3    implicit none
 4    integer(kind=4) :: row, i, width, rowsize, maxiter
 5    real(kind=8)    :: threshold
 6    !C CRS matrix data
 7    integer(kind=4), allocatable :: crs_index(:),crs_column(:)
 8    real(kind=8), allocatable    ::  X(:), B(:), crs_val(:)
 9    logical :: SETUP_AMG=.false.
10    width = 300; rowsize = width*width
11    allocate(crs_index(0:rowsize),crs_column(rowsize*5), crs_val(rowsize*5))
12    i = 0;   crs_index(0) = 0
13    do row = 1, rowsize
14       if(row-width > 0) then
15          i=i+1;crs_val(i) = -1.0;     crs_column(i) = row-width;
16       end if
17       if(mod((row-1),width) /= 0) then
18          i=i+1; crs_val(i) = -1.0;     crs_column(i) = row-1;
19       end if
20       i=i+1;crs_val(i) = 4.0;     crs_column(i) = row;
21       if(mod((row+1),width) /= 1) then
22          i=i+1; crs_val(i) = -1.0;     crs_column(i) = row+1;
23       end if
24       if(row+width <= rowsize) then
25          i=i+1; crs_val(i) = -1.0;     crs_column(i) = row+width;
26       end if
27       crs_index(row) = i
28    end do
29    allocate(B(rowsize), X(rowsize))
30    B = 1.0;  X = 0
31    call amgs_parameter_set(10)
32    call amgs_store_problem_matrix(crs_index, crs_column, crs_val, rowsize)
33    maxiter = 20;  threshold = 1.0E-7
34    call amgs_terminal_val_set(threshold)
35    call amgs_solver(X, B, maxiter, SETUP_AMG)
36    call amgs_clear_matrices()
37    deallocate(crs_column, crs_val, crs_index, B, X)
38  end program sample
```

4.2 並列版

rank 0 のプロセスが問題行列を生成し，それを全体に分散した後でソルバを呼び出している．amgs-p-($version) フォルダにインストールした場合，以下のようなコマンドでコンパイルできる．

```
> mpif90 sample_code_p.f90 amgs-p-($version)/amgs_p.a -module amgs-p-($version)
```

CUDA も有効にしてソルバをコンパイルするときは CUDA のライブラリもリンクする必要がある．CUDA のライブラリの場所などは環境によって異なることが多い．下の例では CUDA と OpenMP を有効にしたときのコンパイル例である．ParMETIS を利用する場合は ParMETIS もリンクする必要がある．

```
> mpif90 sample_code_p.f90 amgs-p-($version)/amgs_p.a -module amgs-p-($version)\
   -openmp -lcutil_x86_64 -lcudart -L/usr/local/cuda/lib64
```

1 行目から 25 行目までで CRS 形式の行列を生成し，26 行目から 32 行目までで AMGS ライブラリ内の関数を呼んでいる．rank 0 のみ行列を持ち，問題行列をライブラリモジュールに登録している．他のプロセスは行列データを持たないため，サイズ 0 の配列を確保し amgs_store_problem_matrix() を呼んでいる．

並列版サンプルプログラム: sample_code_p.f90

```
 1  program sample
 2    use amgs_p_module
 3    implicit none
 4    include 'mpif.h'
 5    integer(kind=4) :: row, i, width, rowsize, maxiter
 6    real(kind=8)    :: threshold
 7    integer(kind=4), allocatable :: crs_index(:),crs_column(:)
 8    real(kind=8),    allocatable ::  crs_val(:),restol(:)
 9    real(kind=8),    allocatable ::  X(:), B(:)
10    integer(kind=4) :: SOLVER_COMM, my_rank, ierr
11    logical :: SETUP_AMG=.false.
12    CALL MPI_INIT      (ierr)
13    CALL MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
14    CALL MPI_COMM_DUP  (MPI_COMM_WORLD, SOLVER_COMM, ierr)
15    if(my_rank == 0) then
16       width = 300; rowsize = width*width
17       allocate(crs_index(0:rowsize),crs_column(rowsize*5), crs_val(rowsize*5))
18       ... !C CRS 形式の問題行列を生成. -> crs-[index,column,val]
19       ... !C 逐次版サンプルプログラムの 12 行目～28 行目と同様
20    else
21       rowsize = 0
22       allocate(crs_index(0),crs_column(0), crs_val(0))
23    end if
24    allocate(B(rowsize), X(rowsize))
25    B=1.0; X = 0
26    call amgs_parameter_option(10, SOLVER_COMM,                    &
27         & "-iter_down=1 -iter_up=1 -strong_con_threshold=0.04   &
28         & -krylov_solver=CG  -smoother=mcgs -dist_method=FIX    &
29         & -aggregation=coupled_w_lu                             &
30         & -max_level_size=10 -min_node_size=100")
31
32    call amgs_store_problem_matrix(crs_index, crs_column, crs_val, rowsize,
SOLVER_COMM)
33    call amgs_distribute_matrix()
34    maxiter = 30;  threshold = 1.0E-7
35    call amgs_terminal_val_set(threshold)
36    call amgs_solver(X, B, maxiter, SETUP_AMG)
37    call amgs_clear_matrices()
38    deallocate(crs_index, crs_column,crs_val,B,X)
39    call MPI_FINALIZE(ierr)
40  end program sample
```

参考文献

[1] G.Karypis, et al. : ParMETIS : Parallel Graph Partitioning and Fill-reducing Matrix Ordering. Available from: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/download

[2] P.Sonneveld, M.B.van Gijzen: IDR(s): A Family of Simple and Fast Algorithms for Solving Large Nonsymmetric Systems of Linear Equations, SIAM J. Sci. Comput. Vol. 31, No. 2, pp. 1035-1062, 2008.

[3] M.Harumatsu, Y.Kusakabe, S.Fujino, T.Fukushige, T.Arima, P.Sonneveld: A Proposal of Gauss-Seidel and Successive Over-Relaxation Methods based on IDR Theorem, Vol.2009-HPC-120 No.4, IPSJ SIG Technical Report, 2009, 6pages.

[4] K. Stüben: Algebraic multigrid(AMG): an introduction with applications, in U. Trottenberg, A. Schuller and C.Oosterlee, eds., Multigrid, Academic Press, pp. 413-532, 2000.

[5] T. Füllenbach, K. Stüben: Algebraic multigrid for Selected PDE Systems, Proceedings of the 4th European Conference, World Scientific New Jersey, London, pp. 399-410. 2002.

[6] T.Davis: University of Florida Sparse Matrix Collection. Available from: http://www.cise.ufl.edu/research/sparse/matrices.

[7] M. Brezina, R. Falgout, S. Maclachlan, T. Manteuffel, S. Mccormick, J. Ruge: Adaptive Algebraic Multigrid, SIAM J. Sci. Comput. Vol. 27, No. 4, pp. 1261-1286.

[8] A.Brandt, S.McCormick, J.Ruge : Algebraic multigrid (AMG) for sparse matrix eqations, in Sparsity and its Applications, D. Evans, ed. , Cambridge University Press, Cambridge, UK, pp. 257-284 (1984)

[9] A.Brandt : Algebraic multigrid theory: The symmetric case, Appl. Math. Comput., 19, pp.23-56 (1986)

[10] P.Vanek, J.Mandel and M.Brezina : Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems, Technical Report UCD-CCM-036. (1995)

[11] Y.Onoue, P. Sonneveld, S. Fujino: Estimation of convergence properties of IDR-AGS method and consideration on its effectiveness, vol.2008-HPC-118, No.125, IPSJ SIG Technical Report, 2008, 6pages.