

SXAMG: a Serial Algebraic Multigrid Solver Library

VERSION 1.0

Hui Liu

Contents

1	Introduction	1
1.1	Overview	1
1.2	License	1
1.3	Citation	1
1.4	Website	2
2	Installation	3
2.1	Configuration	3
2.2	Options	3
2.3	Compilation	4
2.4	Installation	5
3	Basics	7
3.1	Data Types	7
3.2	Matrix	7
3.2.1	Matrix Management	8
3.3	Vector	9
3.3.1	Vector Management	9
3.4	BLAS Operations	10
3.4.1	Vector	10
3.4.2	Matrix-Vector	11
3.4.3	Matrix-Matrix	11
4	AMG Solver	13
4.1	Data Structures	14
4.2	Management	16
4.2.1	Initialize Parameters	16
4.2.2	Setup	16
4.2.3	Solve	16
4.2.4	Destroy	17
5	Utilities	19
5.1	Print	19
5.2	Memory	19

5.3	Performance	20
6	How to Use	21
6.1	Solver	21
6.2	Lower Level Interface	22

Chapter 1

Introduction

1.1 Overview

SXAMG is an AMG (Algebraic Multi-Grid) solver library for sparse linear system, $\mathbf{Ax} = \mathbf{b}$. The initial code is from FASP project, <http://fasp.sourceforge.net>, which implements a collection of Krylov solvers, AMG solvers and preconditioners.

The library is serial and written by C. It is designed for Linux, Unix and Mac systems. However, it is also possible to compile under Windows if user generates a file `include/config.h` manually, where `USE_UNIX` should be set to 0.

SXAMG rewrites all the data structures, subroutines, and file structures, and it removes many components of the original implementation and third-party package dependency. The purpose of this refactorization is to provide a standalone AMG solver to support other numerical applications and to validate new ideas in the future. The **SXAMG** can serve as a solver and a preconditioning method.

1.2 License

The package uses General Public License (GPL) license. If you have any issue, please contact the developer: hui.sc.liu@gmail.com

1.3 Citation

If you use this library for your research, proper citation will be appreciated. The **SXAMG** library can be cited as,

```
@misc{sxamg-library,
```

```
author="Hui Liu",  
title="SXAMG: a Serial Algebraic Multigrid Solver Library",  
year="2017",  
note={\url{https://github.com/huiscliu/sxamg/}}  
}
```

1.4 Website

The official website for SXAMG is <https://github.com/huiscliu/sxamg/>.

Chapter 2

Installation

SXAMG uses `autoconf` and `make` to detect system parameters, to set default parameters, to build and to install.

2.1 Configuration

The simplest way to configure is to run command:

```
./configure
```

This command will try to find optional packages if applicable and set system parameters.

2.2 Options

The script `configure` has many options, if user would like to check, run command:

```
./configure --help
```

Output will be like this,

```
'configure' configures this package to adapt to many kinds of systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE. See below for descriptions of some of the useful variables.

....

Optional Features:
```

```

--disable-option-checking  ignore unrecognized --enable/--with options
--disable-FEATURE          do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE[=ARG]    include FEATURE [ARG=yes]
--enable-rpath             enable use of rpath (default)
--disable-rpath           disable use of rpath
--with-rpath-flag=FLAG    compiler flag for rpath (e.g., "-Wl,-rpath,")
--disable-assert          turn off assertions
--enable-big-int          use long int for INT
--disable-big-int         use int for INT (default),
--with-int=type           integer type(long|long long)
--enable-long-double      use long double for FLOAT
--disable-long-double     use double for FLOAT (default)

```

Some influential environment variables:

```

CC          C compiler command
CFLAGS      C compiler flags
LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
            nonstandard directory <lib dir>
LIBS        libraries to pass to the linker, e.g. -l<library>
CPPFLAGS    (Objective) C/C++ preprocessor flags, e.g. -I<include dir> if
            you have headers in a nonstandard directory <include dir>
CXX         C++ compiler command
CXXFLAGS    C++ compiler flags
FC          Fortran compiler command
FCFLAGS     Fortran compiler flags
CPP         C preprocessor
CXXCPP      C++ preprocessor

```

The most important options are,

- `--prefix=PATH` where to install the library, and default directory is `/usr/local/sxamg/`;
- `--enable-rpath` and `--disable-rpath`, use rpath or not, and it is enabled by default;
- `--enable-big-int` and `--disable-big-int`, use big integer or not, and use `int` by default;
- `--with-int=type`, type is `long` or `long long`. This option is checked when big integer is enabled (`--enable-big-int`);
- `--enable-long-double` and `--disable-long-double`, use `long double` or not, and `double` is used by default;

2.3 Compilation

After configuration, `Makefile` and related scripts will be set correctly. A simple `make` command can compile the package,

2.4. Installation

```
make
```

A library, `libsxamg.a`, will be generated under, `lib/`.

2.4 Installation

Run command:

```
make install
```

The package will be installed to a user-defined directory by `--prefix=DIR` during configuring, such as `--prefix=/usr/sxamg/`. The default destination is `/usr/local/sxamg/`.

Chapter 3

Basics

This chapter introduces basic types, matrix and vector management.

3.1 Data Types

SXAMG has two data types, `SX_FLOAT` and `SX_INT`, for floating-point number and integer, respectively. They are defined as:

```
#if USE_LONG_DOUBLE
typedef long double      SX_FLOAT;
#else
typedef double          SX_FLOAT;
#endif

#if USE_LONG_LONG
typedef signed long long int  SX_INT;
#elif USE_LONG
typedef signed long int      SX_INT;
#else
typedef signed int          SX_INT;
#endif
```

The macros, `USE_LONG_DOUBLE`, `USE_LONG_LONG` and `USE_LONG`, are set by `configure` to control the real types, such as `long int` and `long double`. User can control through `configure` options.

3.2 Matrix

`SX_MAT` defines a CSR matrix. The index is from 0 (zero) following C convention. The meanings of the members are the same as usual.

```
typedef struct SX_MAT
{
    SX_INT num_rows;    /* number of rows */
    SX_INT num_cols;    /* number of columns */
    SX_INT num_nnz;     /* number of non-zeros (entries) */

    SX_INT  *Ap;        /* offset of each row */
    SX_INT  *Aj;        /* column indices */
    SX_FLOAT *Ax;       /* values */
} SX_MAT;
```

3.2.1 Matrix Management

3.2.1.1 Create

`sx_mat_struct_create` creates the structure of a matrix. The memory is allocated but no value is set.

```
SX_MAT sx_mat_struct_create(const SX_INT nrow, const SX_INT ncol, const SX_INT nnz);
```

`sx_mat_create` creates a CSR matrix by user input. This function allocates memory and copies values to the matrix.

```
SX_MAT sx_mat_create(SX_INT nrow, SX_INT ncol, SX_INT *Ap, SX_INT *Aj, SX_FLOAT *Ax);
```

3.2.1.2 Destroy

`sx_mat_destroy` destroys a matrix and frees its memory.

```
void sx_mat_destroy(SX_MAT *A);
```

3.2.1.3 Transpose

`sx_mat_trans` gets transpose of a matrix.

```
SX_MAT sx_mat_trans(SX_MAT *A);
```

3.2.1.4 Sort

`sx_mat_sort` sorts column indices in ascending manner.

3.3. Vector

```
void sx_mat_sort(SX_MAT *A);
```

3.3 Vector

`SX_VEC` defines floating-point vector. n is the length of the vector, and d stores the values.

```
typedef struct SX_VEC
{
    SX_INT    n;
    SX_FLOAT  *d;
} SX_VEC;
```

3.3.1 Vector Management

3.3.1.1 Create

`sx_vec_create` creates a vector of length m ($m \geq 0$).

```
SX_VEC sx_vec_create(SX_INT m);
```

3.3.1.2 Destroy

`sx_vec_destroy` destroys a vector and frees its memory.

```
void sx_vec_destroy(SX_VEC *u);
```

3.3.1.3 Set Value

`sx_vec_set_value` sets equal value to each component.

```
void sx_vec_set_value(SX_VEC *x, SX_FLOAT val);
```

`sx_vec_set_entry` sets value: $x[i] = val$.

```
void sx_vec_set_entry(SX_VEC *x, SX_INT i, SX_FLOAT val);
```

3.3.1.4 Get Value

`sx_vec_get_entry` returns value of $x[i]$.

```
SX_FLOAT sx_vec_get_entry(const SX_VEC *x, SX_INT i);
```

3.3.1.5 Copy

`sx_vec_cp` copies values from source vector and to destination vector, and the two vectors should have the same length.

```
void sx_vec_cp(const SX_VEC *src, SX_VEC *des);
```

3.4 BLAS Operations

Some level 1 and level 2 BLAS operations are implemented for internal use.

3.4.1 Vector

`sx_blas_vec_norm2` calculates L2 norm.

```
SX_FLOAT sx_blas_vec_norm2(const SX_VEC *x);
```

`sx_blas_vec_dot` calculates dot product.

```
SX_FLOAT sx_blas_vec_dot(const SX_VEC *x, const SX_VEC *y);
```

`sx_blas_vec_axpyz` computes: $z = a * x + y$.

```
void sx_blas_vec_axpyz(SX_FLOAT a, const SX_VEC *x, const SX_VEC *y, SX_VEC *z);
```

`sx_blas_vec_axpyz` computes: $z = a * x + b * y$.

```
void sx_blas_vec_axpyz(SX_FLOAT a, const SX_VEC *x, SX_FLOAT b, const SX_VEC *y, SX_VEC *z);
```

`sx_blas_vec_axpy` computes: $y = a * x + y$.

```
void sx_blas_vec_axpy(SX_FLOAT a, const SX_VEC *x, SX_VEC *y);
```

`sx_blas_vec_axpyb` computes: $y = a * x + b * y$.

```
void sx_blas_vec_axpy(SX_FLOAT a, const SX_VEC *x, SX_FLOAT b, SX_VEC *y);
```

3.4. BLAS Operations

3.4.2 Matrix-Vector

`sx_blas_mat_amxpy` computes: $y = y + a * A * x$.

```
void sx_blas_mat_amxpy(SX_FLOAT a, const SX_MAT *A, const SX_VEC *x, SX_VEC *y);
```

`sx_blas_mat_mxy` computes: $y = A * x$.

```
void sx_blas_mat_mxy(const SX_MAT *A, const SX_VEC *x, SX_VEC *y);
```

3.4.3 Matrix-Matrix

`sx_blas_mat_rap` returns the matrix-matrix product: $R * A * P$.

```
SX_MAT sx_blas_mat_rap(const SX_MAT *R, const SX_MAT *A, const SX_MAT *P);
```


Chapter 4

AMG Solver

If A is a positive-definite square matrix, the AMG methods [5, 4, 1, 2, 3, 6] have proved to be efficient methods and they are also scalable [7].

AMG methods have hierarchical structures, which is shown in Figure 4.1. A coarse grid is constructed when entering a coarser level. To calculate a coarser matrix, a restriction operator R_l and an interpolation (prolongation) operator P_l need to be determined. In general, the restriction operator R_l is the transpose of the interpolation (prolongation) operator P_l :

$$R_l = P_l^T.$$

The matrix on the coarser grid is calculated as

$$A_{l+1} = R_l A_l P_l. \quad (4.1)$$

We know that a high frequency error is easier to converge on a fine grid than a low frequency error, and for the AMG methods, the restriction operator, R_l , projects the error from a finer grid onto a coarser grid and converts a low frequency error to a high frequency error. The interpolation operator transfers a solution on a coarser grid to that on a finer grid. Its setup phase for the AMG methods on each level l ($0 \leq l < L$) is formulated in Algorithm 1, where a coarser grid, an interpolation operator, a restriction operator, a coarser matrix and post- and pre-smoothers are constructed. By repeating the algorithm, an L -level system can be built. The solution of the AMG methods is recursive and is formulated in Algorithm 2, which shows one iteration of AMG.

The Cleary-Luby-Jones-Plassman (CLJP) coarsening algorithm was proposed by Cleary [9] based on the algorithms developed by Luby [11] and Jones and Plassman [10]. The standard RS coarsening algorithm has also been parallelized [8]. Falgout et al. developed a parallel coarsening algorithm, the Falgout coarsening algorithm, which has been implemented in HYPRE [8]. Yang et al. proposed HMIS and PMIS coarsening algorithms for a coarse grid selection [12]. Various parallel smoothers and interpolation operators have also been studied by Yang et al [8, 12].

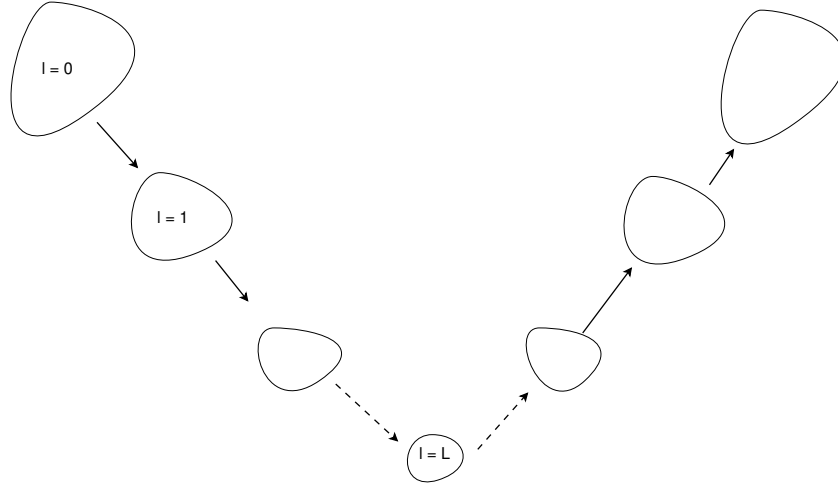


Figure 4.1: Structure of AMG solver.

Algorithm 1 AMG setup Algorithm

- 1: Calculate strength matrix S .
 - 2: Choose coarsening nodes set ω_{l+1} according to strength matrix S , such that $\omega_{l+1} \subset \omega_l$.
 - 3: Calculate prolongation operator P_l .
 - 4: Derive restriction operator $R_l = P_l^T$.
 - 5: Calculate coarse matrix A_{l+1} : $A_{l+1} = R_l \times A_l \times P_l$.
 - 6: Setup pre-smoother S_l and post-smoother T_l .
-

Algorithm 2 AMG V-cycle Solution Algorithm: `amg_solve(l)`

Require: $b_l, x_l, A_l, R_l, P_l, S_l, T_l, 0 \leq l < L$

```

 $b_0 = b$ 
if ( $l < L$ ) then
     $x_l = S_l(x_l, A_l, b_l)$ 
     $r = b_l - A_l x_l$ 
     $b_{r+1} = R_l r$ 
    amg_solve( $l + 1$ )
     $x_l = x_l + P_l x_{l+1}$ 
     $x_l = T_l(x_l, A_l, b_l)$ 
else
     $x_l = A_l^{-1} b_l$ 
end if
 $x = x_0$ 
    
```

4.1 Data Structures

`SX_SM_TYPE` defines smoother types. The following smoothers are implemented,

4.1. Data Structures

```
typedef enum
{
    SX_SM_JACOBI      = 1,  /**< Jacobi smoother */
    SX_SM_GS          = 2,  /**< Gauss-Seidel smoother */
    SX_SM_SGS         = 3,  /**< Symmetric Gauss-Seidel smoother */
    SX_SM_SOR         = 4,  /**< SOR smoother */
    SX_SM_SSOR        = 5,  /**< SSOR smoother */
    SX_SM_GSOR        = 6,  /**< GS + SOR smoother */
    SX_SM_SGSOR       = 7,  /**< SGS + SSOR smoother */
    SX_SM_POLY        = 8,  /**< Polynomial smoother */
    SX_SM_L1DIAG      = 9,  /**< L1 norm diagonal scaling smoother */
} SX_SM_TYPE;
```

[SX_COARSEN_TYPE](#) defines coarsening types, including classical RS coarsening and classical RS coarsening with positive off-diagonals.

```
typedef enum
{
    SX_COARSE_RS      = 1,  /**< Classical */
    SX_COARSE_RSP     = 2,  /**< Classical, with positive offdiags */
} SX_COARSEN_TYPE;
```

[SX_INTERP_TYPE](#) defines interpolation types, including direct interpolation and standard interpolation.

```
typedef enum
{
    SX_INTERP_DIR     = 1,  /**< Direct interpolation */
    SX_INTERP_STD     = 2,  /**< Standard interpolation */
} SX_INTERP_TYPE;
```

[SX_AMG_PARS](#) defines AMG parameters. The meaning of each member is explained as comment. For example, [cycle_itr](#) determines the cycle type: 1 for V-cycle, 2 for W-cycle..

```
typedef struct SX_AMG_PARS
{
    SX_INT verb;

    SX_INT  cycle_itr;          /** type of AMG cycle, 1 is for V, W if greater than 1 */
    SX_FLOAT tol;              /** stopping tolerance for AMG solver */
    SX_FLOAT ctol;             /** stopping tolerance for coarsest solver */
    SX_INT  maxit;             /** maximal number of iterations of AMG */

    SX_COARSEN_TYPE cs_type;   /** coarsening type */
    SX_INT  max_levels;        /** max number of levels of AMG */
    SX_INT  coarse_dof;        /** max number of coarsest level DOF */

    SX_SM_TYPE smoother;      /** smoother type */
}
```

```

SX_FLOAT  relax;           /** relax parser for SOR smoother */
SX_INT    cf_order;        /** False (0): nature order, True (1): C/F order */
SX_INT    pre_iter;        /** number of presmoothers */
SX_INT    post_iter;       /** number of postsmoothers */
SX_INT    poly_deg;        /** degree of the polynomial smoother */

SX_INTERP_TYPE interp_type; /** interpolation type */
SX_FLOAT strong_threshold;  /** strong connection threshold for coarsening */
SX_FLOAT max_row_sum;       /** maximal row sum parser */
SX_FLOAT trunc_threshold;   /** truncation threshold */

} SX_AMG_PARS;

```

`SX_RTN` is for return values.

```

typedef struct SX_RTN
{
    SX_FLOAT ares;    /* absolute residual */
    SX_FLOAT rres;    /* relative residual */
    SX_INT   nits;    /* number of iterations */
} SX_RTN;

```

4.2 Management

4.2.1 Initialize Parameters

`sx_amg_pars_init` sets default parameters.

```
void sx_amg_pars_init(SX_AMG_PARS *pars);
```

4.2.2 Setup

`sx_amg_setup` setups the hierarchical structure of AMG solver using given parameters.

```
SX_AMG * sx_amg_setup(SX_MAT *A, SX_AMG_PARS *pars);
```

4.2.3 Solve

`sx_solver_amg_solve` solves the linear system using AMG method. The AMG object has to be set up before using, the `x` has initial value and `b` is the right-hand side.

```
SX_RTN sx_solver_amg_solve(SX_AMG *mg, SX_VEC *x, SX_VEC *b);
```

4.2. Management

`sx_solver_amg` solves the linear system using AMG method. This function is a high level interface, and user can call it to solve a linear system, which will setup the AMG object, solve and destroy the AMG object. If `pars` is NULL, then default parameters will be applied.

```
SX_RTN sx_solver_amg(SX_MAT *A, SX_VEC *x, SX_VEC *b, SX_AMG_PARS *pars);
```

4.2.4 Destroy

`sx_amg_data_destroy` destroys the AMG object.

```
void sx_amg_data_destroy(SX_AMG **mg);
```


Chapter 5

Utilities

5.1 Print

`sx_set_log` sets log file. If log file is set, all screen outputs will be stored to log file too.

```
void sx_set_log(FILE *io);
```

`sx_printf` prints info to screen, if log file is set, it prints to log file and screen.

```
int sx_printf(const char *fmt, ...);
```

5.2 Memory

`sx_mem_malloc` allocates memory.

```
void * sx_mem_malloc(size_t size);
```

`sx_mem_calloc` allocates and initializes memory.

```
void * sx_mem_calloc(size_t size, SX_INT type);
```

`sx_mem_realloc` reallocates memory.

```
void * sx_mem_realloc(void *oldmem, size_t tsize);
```

`sx_mem_free` frees memory.

```
void sx_mem_free(void *mem);
```

5.3 Performance

`sx_gettime` returns current time stamp, if `t` is not `NULL`, result will be written to `t`.

```
SX_FLOAT sx_gettime(SX_FLOAT *t);
```


Chapter 6

How to Use

6.1 Solver

This example shows how to use SXAMG as a solver. Five steps are required to call:

1. `sx_amg_pars_init` initializes default AMG parameters, and user can check their value from source code;
2. Re-define parameters. This step is optional.
3. Input matrix, initial guess and right-hand side should be constructed.
4. `sx_solver_amg` setups, solves and destroys the AMG system, and the solution is stored by `x`.
5. Free memories.

```
{
    SX_AMG_PARS pars;
    SX_MAT A;
    SX_VEC b, x;
    int verb = 2;
    SX_RTN rtn;

    /* step 1: default pars */
    sx_amg_pars_init(&pars);

    /* step 2: redefine parameters */
    pars.maxit = 1000;
    pars.verb = 2;

    /* step 3: set A, b, initial x */
    ....
}
```

```

/* step 4: solve the system */
rtn = sx_solver_amg(&A, &x, &b, &pars);

/* step 5: free memory */
sx_mat_destroy(&A);
sx_vec_destroy(&b);
sx_vec_destroy(&x);
}

```

6.2 Lower Level Interface

Sometimes user needs to store the AMG object and re-uses it, such as preconditioner. This example exposes more details of the AMG solver, where seven steps are required:

1. initialize default parameters;
2. re-define parameters if default parameters do not meet requirement;
3. setup input matrix [A](#), which is a standard CSR matrix;
4. setup AMG object;
5. setup initial guess and right-hand side;
6. solve the linear system using given [x](#) and [b](#);
7. free memories;

Here user can call step 1) to 4) once in the beginning stage, call 5) and 6) as many as required, and call step 7) in the end stage of the program.

```

{
    SX_AMG_PARS pars;
    SX_MAT A;
    SX_VEC b, x;
    int verb = 2;
    SX_AMG *mg;
    SX_RTN rtn;

    /* step 1: initialize pars */
    sx_amg_pars_init(&pars);

    /* step 2: re-define parameters */
    pars.maxit = 1000;
    pars.verb = 2;

    /* step 3: input A */
    ....
}

```

6.2. Lower Level Interface

```
/* Step 4: setup AMG setup object */
mg = sx_amg_setup(&A, &pars);

/* Step 5: setup rhs and initial guess */
x = sx_vec_create(A.num_rows);
sx_vec_set_value(&x, 1.0);

b = sx_vec_create(A.num_rows);
sx_vec_set_value(&b, 1.0);

/* step 6: solve */
rtn = sx_solver_amg_solve(mg, &x, &b);

/* step 7: free memories */
sx_mat_destroy(&A);
sx_vec_destroy(&b);
sx_vec_destroy(&x);
sx_amg_data_destroy(&mg);
}
```


Bibliography

- [1] JW Ruge and K Stüben, Algebraic multigrid (AMG), in: S.F. McCormick (Ed.), Multigrid Methods, Frontiers in Applied Mathematics, Vol. 5, SIAM, Philadelphia, 1986.
- [2] A Brandt, SF McCormick, J Ruge, Algebraic multigrid (AMG) for sparse matrix equations D.J. Evans (Ed.), Sparsity and its Applications, Cambridge University Press, Cambridge, 1984, 257–284.
- [3] RD Falgout, An Introduction to Algebraic Multigrid, Computing in Science and Engineering, Special Issue on Multigrid Computing, 8, 2006, 24–33.
- [4] K. Stüben, T. Clees, H. Klie, B. Lou, M.F. Wheeler, Algebraic multigrid methods (AMG) for the efficient solution of fully implicit formulations in reservoir simulation, SPE Reservoir Simulation Symposium. 2007.
- [5] K. Stüben, A review of algebraic multigrid, Journal of Computational and Applied Mathematics 128.1 (2001): 281-309.
- [6] UM Yang, On the Use of Relaxation Parameters in Hybrid Smoothers, Numerical Linear Algebra With Applications, 11, (2004), 155-172.
- [7] AJ Cleary, RD Falgout, VE Henson, JE Jones, TA Manteuffel, SF McCormick, GN Miranda, JW Ruge, Robustness and Scalability of Algebraic Multigrid, SIAM J. Sci. Comput., 21, 2000, 1886–1908.
- [8] R. D. Falgout, and U.M. Yang, HYPRE: A library of high performance preconditioners, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2002. 632-641.
- [9] AJ Cleary, RD Falgout, VE Henson, JE Jones, Coarse grid selection for parallel algebraic multigrid, in Proceedings of the fifth international symposium on solving irregularly structured problems in parallel, Springer-Verlag, New York, 1998.
- [10] MT Jones, PE Plassman, A parallel graph coloring heuristic, SIAM Journal on Scientific Computing, 14(1993): 654-669.
- [11] M Luby, A simple parallel algorithm for the maximal independent set problem, SIAM Journal on Computing, 15(1986), 1036-1053.

- [12] Hans DS, Yang UM, Heys J, Reducing Complexity in Parallel Algebraic Multigrid Preconditioners, SIAM Journal on Matrix Analysis and Applications 27, (2006), 1019-1039.