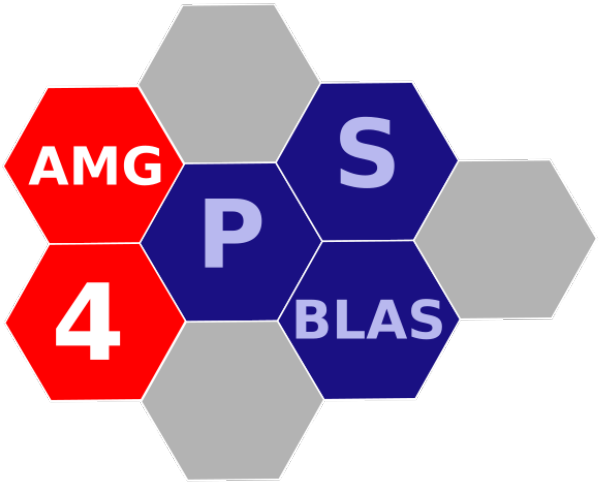
# AMG4PSBLAS
# User's and Reference Guide

*A guide for the Algebraic MultiGrid*
*Preconditioners Package based on PSBLAS*




**Pasqua D'Ambra**
IAC-CNR, Italy

**Fabio Durastante**
University of Pisa and IAC-CNR

**Salvatore Filippone**
University of Rome Tor-Vergata and IAC-CNR

Software version: 1.0
March 31, 2021

*This page intentionally left blank*

## Abstract

AMG4PSBLAS (ALGEBRAIC MULTIGRID PRECONDITIONERS PACKAGE BASED ON PSBLAS) is a package of parallel algebraic multilevel preconditioners included in the PSCToolkit (Parallel Sparse Computation Toolkit) software framework. It is a progress of a software development project started in 2007, named MLD2P4, which originally implemented a multilevel version of some domain decomposition preconditioners of additive-Schwarz type, and was based on a parallel decoupled version of the well known smoothed aggregation method to generate the multilevel hierarchy of coarser matrices. In the last years, within the context of the EU-H2020 EoCoE project (Energy Oriented Center of Excellence), the package was extended for including new algorithms and functionalities for the setup and application new AMG preconditioners with the final aims of improving efficiency and scalability when tens of thousands cores are used, and of boosting reliability in dealing with general symmetric positive definite linear systems. Due to the significant number of changes and the increase in scope, we decided to rename the package as AMG4PSBLAS.

AMG4PSBLAS has been designed to provide scalable and easy-to-use preconditioners in the context of the PSBLAS (Parallel Sparse Basic Linear Algebra Subprograms) computational framework and can be used in conjuction with the Krylov solvers available in this framework. Our package is based on a completely algebraic approach; therefore users level interfaces assume that the system matrix and preconditioners are represented as PSBLAS distributed sparse matrices. AMG4PSBLAS enables the user to easily specify different features of an algebraic multilevel preconditioner, thus allowing to experiment with different preconditioners for the problem and parallel computers at hand.

The package employs object-oriented design techniques in Fortran 2003, with interfaces to additional third party libraries such as MUMPS, UMFPACK, SuperLU, and SuperLU_Dist, which can be exploited in building multilevel preconditioners. The parallel implementation is based on a Single Program Multiple Data (SPMD) paradigm; the inter-process communication is based on MPI and is managed mainly through PSBLAS.

This guide provides a brief description of the functionalities and the user interface of AMG4PSBLAS.

*This page intentionally left blank*

# Contents

*This page intentionally left blank*

# 1 General Overview

The ALGEBRAIC MULTIGRID PRECONDITIONERS PACKAGE BASED ON PSBLAS (AMG-4PSBLAS) provides parallel Algebraic MultiGrid (AMG) preconditioners (see, e.g., [5, 31]), to be used in the iterative solution of linear systems,

$$Ax = b, \tag{1}$$

where $A$ is a square, real or complex, sparse symmetric positive definite (s.p.d) matrix.

The preconditioners implemented in AMG4PSBLAS are obtained by combining 3 different types of AMG cycles with smoothers and coarsest-level solvers. Available multigrid cycles include the V-, W-, and a version of a Krylov-type cycle (K-cycle) [5, 27]; they can be combined with Jacobi, hybrid forward/backward Gauss-Seidel, block-Jacobi and additive Schwarz smoothers with various versions of local incomplete factorizations and approximate inverses on the blocks. The Jacobi, block-Jacobi and Gauss-Seidel smoothers are also available in the $\ell_1$ version [14].

An algebraic approach is used to generate a hierarchy of coarse-level matrices and operators, without explicitly using any information on the geometry of the original problem, e.g., the discretization of a PDE. To this end, two different coarsening strategies, based on aggregation, are available:

- a decoupled version of the smoothed aggregation procedure proposed in [4, 33], and already included in the previous versions of the package [7, 11];
- a coupled, parallel implementation of the Coarsening based on Compatible Weighted Matching introduced in [12, 13] and described in detail in [14];

Either exact or approximate solvers can be used on the coarsest-level system. We provide interfaces to various parallel and sequential sparse LU factorizations from external packages, sequential native incomplete LU and approximate inverse factorizations, parallel weighted Jacobi, hybrid Gauss-Seidel, block-Jacobi solvers and calls to preconditioned Krylov methods; all smoothers can be also exploited as one-level preconditioners.

AMG4PSBLAS is written in Fortran 2003, following an object-oriented design through the exploitation of features such as abstract data type creation, type extension, functional overloading, and dynamic memory management. The parallel implementation is based on a Single Program Multiple Data (SPMD) paradigm. Single and double precision implementations of AMG4PSBLAS are available for both the real and the complex case, which can be used through a single interface.

AMG4PSBLAS has been designed to implement scalable and easy-to-use multilevel preconditioners in the context of the PSBLAS (Parallel Sparse BLAS) computational framework [22, 21]. PSBLAS provides basic linear algebra operators and data management facilities for distributed sparse matrices, kernels for sequential incomplete factorizations needed for the parallel block-Jacobi and additive Schwarz smoothers, and parallel Krylov solvers which can be used with the AMG4PSBLAS preconditioners. The choice of PSBLAS has been mainly motivated by the need of having a portable

and efficient software infrastructure implementing "de facto" standard parallel sparse linear algebra kernels, to pursue goals such as performance, portability, modularity ed extensibility in the development of the preconditioner package. On the other hand, the implementation of AMG4PSBLAS, which was driven by the need to face the exascale challenge, has led to some important revisions and extentions of the PSBLAS infrastructure. The inter-process comunication required by AMG4PSBLAS is encapsulated in the PSBLAS routines; therefore, AMG4PSBLAS can be run on any parallel machine where PSBLAS implementations are available. In the most recent version of PSBLAS (release 3.7), a plug-in for GPU is included; it includes CUDA versions of main vector operations and of sparse matrix-vector multiplication, so that Krylov methods coupled with AMG4PSBLAS preconditioners relying on Jacobi and block-Jacobi smoothers with sparse approximate inverses on the blocks can be efficiently executed on cluster of GPUs.

AMG4PSBLAS has a layered and modular software architecture where three main layers can be identified. The lower layer consists of the PSBLAS kernels, the middle one implements the construction and application phases of the preconditioners, and the upper one provides a uniform interface to all the preconditioners. This architecture allows for different levels of use of the package: few black-box routines at the upper layer allow all users to easily build and apply any preconditioner available in AMG4PSBLAS; facilities are also available allowing expert users to extend the set of smoothers and solvers for building new versions of the preconditioners (see Section 6).

This guide is organized as follows. General information on the distribution of the source code is reported in Section 2, while details on the configuration and installation of the package are given in Section 3. The basics for building and applying the preconditioners with the Krylov solvers implemented in PSBLAS are reported in Section 4, where the Fortran codes of a few sample programs are also shown. A reference guide for the user interface routines is provided in Section 5. Information on the extension of the package through the addition of new smoothers and solvers is reported in Section 6. The error handling mechanism used by the package is briefly described in Section 7. The copyright terms concerning the distribution and modification of AMG4PSBLAS are reported in Appendix A.

# 2 Code Distribution

AMG4PSBLAS is available from the web site

https://psctoolkit.github.io/products/amg4psblas/

where contact points for further information can be also found.

The software is available under a modified BSD license, as specified in Appendix A; please note that some of the optional third party libraries may be licensed under a different and more stringent license, most notably the GPL, and this should be taken into account when treating derived works.

The library defines a version string with the constant

`amg_version_string_`

whose current value is `1.0`.

### Contributors

- Pasqua D'Ambra, IAC-CNR, IT;
- Fabio Durastante, University of Pisa and IAC-CNR, IT;
- Salvatore Filippone, University of Rome Tor-Vergata and IAC-CNR, IT;

### Citing AMG4PSBLAS

When use the library, please cite the following:

```
@article{DDF2021,
  author = {D'Ambra, Pasqua and Durastante, Fabio and Filippone, Salvatore},
  title = {{{AMG Preconditioners for Linear Solvers towards Extreme Scale}},
  journal = {arXiv e-preprints},
  eprint = {2006.16147v3},
  archivePrefix = {arXiv},
  year={2021}
}

@Misc{psctoolkit-web-page,
  author = {D'Ambra, Pasqua and Durastante, Fabio and Filippone, Salvatore},
  title =  {{PSCToolkit} {W}eb page},
  url =    {https://psctoolkit.github.io/},
  howpublished = {\url{https://psctoolkit.github.io/}},
  year = {2021}
}
```

# 3 Configuring and Building AMG4PSBLAS

In order to build AMG4PSBLAS it is necessary to set up a Makefile with appropriate system-dependent variables; this is done by means of the `configure` script. The distribution also includes the autoconf and automake sources employed to generate the script, but usually this is not needed to build the software.

AMG4PSBLAS is implemented almost entirely in Fortran 2003, with some interfaces to external libraries in C; the Fortran compiler must support the Fortran 2003 standard plus the extension `MOLD=` feature, which enhances the usability of `ALLOCATE`. Most Fortran compilers provide this feature; in particular, this is supported by the GNU Fortran compiler, for which we recommend to use at least version 4.8. The software defines data types and interfaces for real and complex data, in both single and double precision.

Building AMG4PSBLAS requires some base libraries (see Section 3.1); interfaces to optional third-party libraries, which extend the functionalities of AMG4PSBLAS (see Section 3.2), are also available. A number of Linux distributions (e.g., Ubuntu, Fedora, CentOS) provide precompiled packages for the prerequisite and optional software. In many cases these packages are split between a runtime part and a "developer" part; in order to build AMG4PSBLAS you need both. A description of the base and optional software used by AMG4PSBLAS is given in the next sections.

## 3.1 Prerequisites

The following base libraries are needed:

**BLAS** [17, 18, 25] Many vendors provide optimized versions of BLAS; if no vendor version is available for a given platform, the ATLAS software (math-atlas.sourceforge.net) may be employed. The reference BLAS from Netlib (www.netlib.org/blas) are meant to define the standard behaviour of the BLAS interface, so they are not optimized for any particular platform, and should only be used as a last resort. Note that BLAS computations form a relatively small part of the AMG4PSBLAS/-PSBLAS; however they are critical when using preconditioners based on the MUMPS, UMFPACK or SuperLU third party libraries. UMFPACK requires a full LAPACK library; our experience is that configuring ATLAS for building full LAPACK does not always work in the expected way. Our advice is first to download the LAPACK tarfile from www.netlib.org/lapack and install it independently of ATLAS. In this case, you need to modify the OPTS and NOOPT definitions for including -fPIC compilation option in the make.inc file of the LAPACK library.

**MPI** [24, 30] A version of MPI is available on most high-performance computing systems.

**PSBLAS** [20, 22] Parallel Sparse BLAS (PSBLAS) is available from psctoolkit.github.io/products/psblas/; version 3.7.0 (or later) is required. Indeed, all the prerequisites listed so far are also prerequisites of PSBLAS.

Please note that the four previous libraries must have Fortran interfaces compatible with AMG4PSBLAS; usually this means that they should all be built with the same compiler being used for AMG4PSBLAS.

## 3.2 Optional third party libraries

We provide interfaces to the following third-party software libraries; note that these are optional, but if you enable them some defaults for multilevel preconditioners may change to reflect their presence.

**UMFPACK** [15] A sparse LU factorization package included in the SuiteSparse library, available from `faculty.cse.tamu.edu/davis/suitesparse.html`; it provides sequential factorization and triangular system solution for double precision real and complex data. We tested version 4.5.4 of SuiteSparse. Note that for configuring SuiteSparse you should provide the right path to the BLAS and LAPACK libraries in the `SuiteSparse_config/SuiteSparse_config.mk` file.

**MUMPS** [2] A sparse LU factorization package available from `mumps.enseeiht.fr`; it provides sequential and parallel factorizations and triangular system solution for single and double precision, real and complex data. We tested versions 4.10.0 and 5.0.1.

**SuperLU** [16] A sparse LU factorization package available from `crd.lbl.gov/~xiaoye/SuperLU/`; it provides sequential factorization and triangular system solution for single and double precision, real and complex data. We tested versions 4.3 and 5.0. If you installed BLAS from ATLAS, remember to define the BLASLIB variable in the make.inc file.

**SuperLU_Dist** [26] A sparse LU factorization package available from the same site as SuperLU; it provides parallel factorization and triangular system solution for double precision real and complex data. We tested versions 3.3 and 4.2. If you installed BLAS from ATLAS, remember to define the BLASLIB variable in the make.inc file and to add the `-std=c99` option to the C compiler options. Note that this library requires the ParMETIS library for parallel graph partitioning and fill-reducing matrix ordering, available from `glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`.

## 3.3 Configuration options

In order to build AMG4PSBLAS, the first step is to use the `configure` script in the main directory to generate the necessary makefile.

**DA RISCRIVERE**

As a minimal example consider the following:

```
./configure --with-psblas=PSB-INSTALL-DIR
```

which assumes that the various MPI compilers and support libraries are available in the standard directories on the system, and specifies only the PSBLAS install directory (note that the latter directory must be specified with an *absolute* path). The full set of options may be looked at by issuing the command `./configure --help`, which produces:

```
`configure' configures AMG4PSBLAS 1.0.0 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.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
  -V, --version           display version information and exit
  -q, --quiet, --silent   do not print `checking ...' messages
      --cache-file=FILE   cache test results in FILE [disabled]
  -C, --config-cache      alias for `--cache-file=config.cache'
  -n, --no-create         do not create output files
      --srcdir=DIR        find the sources in DIR [configure dir or `..']

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [PREFIX]

By default, `make install' will install all the files in
`/usr/local/bin', `/usr/local/lib' etc.  You can specify
an installation prefix other than `/usr/local' using `--prefix',
for instance `--prefix=$HOME'.

For better control, use the options below.

Fine tuning of the installation directories:
  --bindir=DIR            user executables [EPREFIX/bin]
  --sbindir=DIR           system admin executables [EPREFIX/sbin]
  --libexecdir=DIR        program executables [EPREFIX/libexec]
  --sysconfdir=DIR        read-only single-machine data [PREFIX/etc]
  --sharedstatedir=DIR    modifiable architecture-independent data
↪  [PREFIX/com]
  --localstatedir=DIR     modifiable single-machine data [PREFIX/var]
```

```
  --libdir=DIR            object code libraries [EPREFIX/lib]
  --includedir=DIR        C header files [PREFIX/include]
  --oldincludedir=DIR     C header files for non-gcc [/usr/include]
  --datarootdir=DIR       read-only arch.-independent data root
↪  [PREFIX/share]
  --datadir=DIR           read-only architecture-independent data
↪  [DATAROOTDIR]
  --infodir=DIR           info documentation [DATAROOTDIR/info]
  --localedir=DIR         locale-dependent data [DATAROOTDIR/locale]
  --mandir=DIR            man documentation [DATAROOTDIR/man]
  --docdir=DIR            documentation root [DATAROOTDIR/doc/amg4psblas]
  --htmldir=DIR           html documentation [DOCDIR]
  --dvidir=DIR            dvi documentation [DOCDIR]
  --pdfdir=DIR            pdf documentation [DOCDIR]
  --psdir=DIR             ps documentation [DOCDIR]

Program names:
  --program-prefix=PREFIX            prepend PREFIX to installed program
↪  names
  --program-suffix=SUFFIX            append SUFFIX to installed program names
  --program-transform-name=PROGRAM   run sed PROGRAM on installed program
↪  names

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-silent-rules   less verbose build output (undo: "make V=1")
  --disable-silent-rules  verbose build output (undo: "make V=0")
  --enable-dependency-tracking
                          do not reject slow dependency extractors
  --disable-dependency-tracking
                          speeds up one-time build
  --enable-serial         Specify whether to enable a fake mpi library to run
                          in serial mode.

Optional Packages:
  --with-PACKAGE[=ARG]    use PACKAGE [ARG=yes]
  --without-PACKAGE       do not use PACKAGE (same as --with-PACKAGE=no)
  --with-psblas=DIR       The install directory for PSBLAS, for example,
                          --with-psblas=/opt/packages/psblas-3.5
  --with-psblas-incdir=DIR
                          Specify the directory for PSBLAS C includes.
  --with-psblas-moddir=DIR
                          Specify the directory for PSBLAS Fortran modules.
  --with-psblas-libdir=DIR
                          Specify the directory for PSBLAS library.
  --with-ccopt            additional [CCOPT] flags to be added: will prepend
```

```
                          to [CCOPT]
  --with-fcopt            additional [FCOPT] flags to be added: will prepend
                          to [FCOPT]
  --with-libs             List additional link flags here. For example,
                          --with-libs=-lspecial_system_lib or
                          --with-libs=-L/path/to/libs
  --with-clibs            additional [CLIBS] flags to be added: will prepend
                          to [CLIBS]
  --with-flibs            additional [FLIBS] flags to be added: will prepend
                          to [FLIBS]
  --with-library-path     additional [LIBRARYPATH] flags to be added: will
                          prepend to [LIBRARYPATH]
  --with-include-path     additional [INCLUDEPATH] flags to be added: will
                          prepend to [INCLUDEPATH]
  --with-module-path      additional [MODULE_PATH] flags to be added: will
                          prepend to [MODULE_PATH]
  --with-extra-libs       List additional link flags here. For example,
                          --with-extra-libs=-lspecial_system_lib or
                          --with-extra-libs=-L/path/to/libs
  --with-blas=<lib>       use BLAS library <lib>
  --with-blasdir=<dir>    search for BLAS library in <dir>
  --with-lapack=<lib>     use LAPACK library <lib>
  --with-mumps=LIBNAME    Specify the libname for MUMPS. Default: autodetect
                          with minimum "-lmumps_common -lpord"
  --with-mumpsdir=DIR     Specify the directory for MUMPS library and
                          includes. Note: you will need to add auxiliary
                          libraries with --extra-libs; this depends on how
                          MUMPS was configured and installed, at a minimum
↪  you
                          will need SCALAPACK and BLAS
  --with-mumpsincdir=DIR  Specify the directory for MUMPS includes.
  --with-mumpsmoddir=DIR  Specify the directory for MUMPS Fortran modules.
  --with-mumpslibdir=DIR  Specify the directory for MUMPS library.
  --with-umfpack=LIBNAME  Specify the library name for UMFPACK and its
↪  support
                          libraries. Default: "-lumfpack -lamd"
  --with-umfpackdir=DIR   Specify the directory for UMFPACK library and
                          includes.
  --with-umfpackincdir=DIR
                          Specify the directory for UMFPACK includes.
  --with-umfpacklibdir=DIR
                          Specify the directory for UMFPACK library.
  --with-superlu=LIBNAME  Specify the library name for SUPERLU library.
                          Default: "-lsuperlu"
  --with-superludir=DIR   Specify the directory for SUPERLU library and
                          includes.
  --with-superluincdir=DIR
                          Specify the directory for SUPERLU includes.
  --with-superlulibdir=DIR
```

```
                          Specify the directory for SUPERLU library.
  --with-superludist=LIBNAME
                          Specify the libname for SUPERLUDIST library.
                          Requires you also specify SuperLU. Default:
                          "-lsuperlu_dist"
  --with-superludistdir=DIR
                          Specify the directory for SUPERLUDIST library and
                          includes.
  --with-superludistincdir=DIR
                          Specify the directory for SUPERLUDIST includes.
  --with-superludistlibdir=DIR
                          Specify the directory for SUPERLUDIST library.

Some influential environment variables:
  FC          Fortran compiler command
  FCFLAGS     Fortran 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>
  CC          C compiler command
  CFLAGS      C compiler flags
  CPPFLAGS    (Objective) C/C++ preprocessor flags, e.g. -I<include dir> if
              you have headers in a nonstandard directory <include dir>
  MPICC       MPI C compiler command
  MPIFC       MPI Fortran compiler command
  CPP         C preprocessor

Use these variables to override the choices made by `configure' or to help
it to find libraries and programs with nonstandard names/locations.

Report bugs to <https://github.com/sfilippone/amg4psblas/issues>.
```

For instance, if a user has built and installed PSBLAS 3.7 under the `/opt` directory and is using the SuiteSparse package (which includes UMFPACK), then AMG4PSBLAS might be configured with:

```
./configure --with-psblas=/opt/psblas-3.7/
↪ --with-umfpackincdir=/usr/include/suitesparse/
```

Once the configure script has completed execution, it will have generated the file `Make.inc` which will then be used by all Makefiles in the directory tree; this file will be copied in the install directory under the name `Make.inc.AMG4PSBLAS`.

To use the MUMPS solver package, the user has to add the appropriate options to the configure script; by default we are looking for the libraries `-ldmumps -lsmumps -lzmumps -lcmumps -mumps_common -lpord`. MUMPS often uses additional packages such as ScaLAPACK, ParMETIS, SCOTCH, as well as enabling OpenMP; in such cases it is necessary to add linker options with the `--with-extra-libs` configure option.

To build the library the user will now enter

```
make
```

followed (optionally) by

```
make install
```

### 3.4 Bug reporting

If you find any bugs in our codes, please report them through our issues page on

https://github.com/psctoolkit/amg4psblas/issues

To enable us to track the bug, please provide a log from the failing application, the test conditions, and ideally a self-contained test program reproducing the issue.

### 3.5 Example and test programs

The package contains the `examples` and `tests` directories; both of them are further divided into `fileread` and `pdegen` subdirectories. Their purpose is as follows:

`examples` contains a set of simple example programs with a predefined choice of preconditioners, selectable via integer values. These are intended to get acquainted with the multilevel preconditioners available in AMG4PSBLAS.

`tests` contains a set of more sophisticated examples that will allow the user, via the input files in the `runs` subdirectories, to experiment with the full range of preconditioners implemented in the package.

The `fileread` directories contain sample programs that read sparse matrices from files, according to the Matrix Market or the Harwell-Boeing storage format; the `pdegen` programs generate matrices in full parallel mode from the discretization of a sample partial differential equation.

# 4 Getting Started

This section describes the basics for building and applying AMG4PSBLAS one-level and multilevel (i.e., AMG) preconditioners with the Krylov solvers included in PSBLAS [20].

The following steps are required:

1. *Declare the preconditioner data structure*. It is a derived data type, `amg_`*x*`prec_type`, where *x* may be `s`, `d`, `c` or `z`, according to the basic data type of the sparse matrix (`s` = real single precision; `d` = real double precision; `c` = complex single precision; `z` = complex double precision). This data structure is accessed by the user only through the AMG4PSBLAS routines, following an object-oriented approach.

2. *Allocate and initialize the preconditioner data structure, according to a preconditioner type chosen by the user*. This is performed by the routine `init`, which also sets defaults for each preconditioner type selected by the user. The preconditioner types and the defaults associated with them are given in Table 1, where the strings used by `init` to identify the preconditioner types are also given. Note that these strings are valid also if uppercase letters are substituted by corresponding lowercase ones.

3. *Modify the selected preconditioner type, by properly setting preconditioner parameters.* This is performed by the routine `set`. This routine must be called if the user wants to modify the default values of the parameters associated with the selected preconditioner type, to obtain a variant of that preconditioner. Examples of use of `set` are given in Section 4.1; a complete list of all the preconditioner parameters and their allowed and default values is provided in Section 5, Tables 2-8.

4. *Build the preconditioner for a given matrix*. If the selected preconditioner is multilevel, then two steps must be performed, as specified next.

   4.1 *Build the AMG hierarchy for a given matrix.* This is performed by the routine `hierarchy_build`.

   4.2 *Build the preconditioner for a given matrix.* This is performed by the routine `smoothers_build`.

   If the selected preconditioner is one-level, it is built in a single step, performed by the routine `bld`.

5. *Apply the preconditioner at each iteration of a Krylov solver.* This is performed by the method `apply`. When using the PSBLAS Krylov solvers, this step is completely transparent to the user, since `apply` is called by the PSBLAS routine implementing the Krylov solver (`psb_krylov`).

6. *Free the preconditioner data structure*. This is performed by the routine `free`. This step is complementary to step 1 and should be performed when the preconditioner is no more used.

All the previous routines are available as methods of the preconditioner object. A detailed description of them is given in Section 5. Examples showing the basic use of AMG4PSBLAS are reported in Section 4.1.

| TYPE | STRING | DEFAULT PRECONDITIONER |
|---|---|---|
| No preconditioner | `'NONE'` | Considered to use the PSBLAS Krylov solvers with no preconditioner. |
| Diagonal | `'DIAG'`, `'JACOBI'`, `'L1-JACOBI'` | Diagonal preconditioner. For any zero diagonal entry of the matrix to be preconditioned, the corresponding entry of the preconditioner is set to 1. |
| Gauss-Seidel | `'GS'`, `'L1-GS'` | Hybrid Gauss-Seidel (forward), that is, global block Jacobi with Gauss-Seidel as local solver. |
| Symmetrized Gauss-Seidel | `'FBGS'`, `'L1-FBGS'` | Symmetrized hybrid Gauss-Seidel, that is, forward Gauss-Seidel followed by backward Gauss-Seidel. |
| Block Jacobi | `'BJAC'`, `'L1-BJAC'` | Block-Jacobi with ILU(0) on the local blocks. |
| Additive Schwarz | `'AS'` | Additive Schwarz (AS), with overlap 1 and ILU(0) on the local blocks. |
| Multilevel | `'ML'` | V-cycle with one hybrid forward Gauss-Seidel (GS) sweep as pre-smoother and one hybrid backward GS sweep as post-smoother, decoupled smoothed aggregation as coarsening algorithm, and LU (plus triangular solve) as coarsest-level solver. See the default values in Tables 2-8 for further details of the preconditioner. |

Table 1: Preconditioner types, corresponding strings and default choices.

Note that the module `amg_prec_mod`, containing the definition of the preconditioner data type and the interfaces to the routines of AMG4PSBLAS, must be used in any program calling such routines. The modules `psb_base_mod`, for the sparse matrix and communication descriptor data types, and `psb_krylov_mod`, for interfacing with the Krylov solvers, must be also used (see Section 4.1).

**Remark 1.** Coarsest-level solvers based on the LU factorization, such as those implemented in UMFPACK, MUMPS, SuperLU, and SuperLU_Dist, usually lead to smaller numbers of preconditioned Krylov iterations than inexact solvers, when the linear system comes from a standard discretization of basic scalar elliptic PDE problems. However, this does not necessarily correspond to the shortest execution time on

parallel computers.

## 4.1 Examples

The code reported in Figure 1 shows how to set and apply the default multilevel preconditioner available in the real double precision version of AMG4PSBLAS (see Table 1). This preconditioner is chosen by simply specifying `'ML'` as the second argument of `P%init` (a call to `P%set` is not needed) and is applied with the CG solver provided by PSBLAS (the matrix of the system to be solved is assumed to be positive definite). As previously observed, the modules `psb_base_mod`, `amg_prec_mod` and `psb_krylov_mod` must be used by the example program.

The part of the code dealing with reading and assembling the sparse matrix and the right-hand side vector and the deallocation of the relevant data structures, performed through the PSBLAS routines for sparse matrix and vector management, is not reported here for the sake of conciseness. The complete code can be found in the example program file `amg_dexample_ml.f90`, in the directory `examples/fileread` of the AMG4PSBLAS implementation (see Section 3.5). A sample test problem along with the relevant input data is available in `examples/fileread/runs`. For details on the use of the PSBLAS routines, see the PSBLAS User's Guide [20].

The setup and application of the default multilevel preconditioner for the real single precision and the complex, single and double precision, versions are obtained with straightforward modifications of the previous example (see Section 5 for details). If these versions are installed, the corresponding codes are available in `examples/fileread/`.

Different versions of the multilevel preconditioner can be obtained by changing the default values of the preconditioner parameters. The code reported in Figure 2 shows how to set a V-cycle preconditioner which applies 1 block-Jacobi sweep as pre- and post-smoother, and solves the coarsest-level system with 8 block-Jacobi sweeps. Note that the ILU(0) factorization (plus triangular solve) is used as local solver for the block-Jacobi sweeps, since this is the default associated with block-Jacobi and set by `P%init`. Furthermore, specifying block-Jacobi as coarsest-level solver implies that the coarsest-level matrix is distributed among the processes. Figure 3 shows how to set a W-cycle preconditioner using the Coarsening based on Compatible Weighted Matching, aggregates of size at most 8 and smoothed prolongators. It applies 2 hybrid Gauss-Seidel sweeps as pre- and post-smoother, and solves the coarsest-level system with the parallel flexible Conjugate Gradient method (KRM) coupled with the block-Jacobi preconditioner having ILU(0) on the blocks. Default parameters are used for stopping criterion of the coarsest solver. Note that, also in this case, specifying KRM as coarsest-level solver implies that the coarsest-level matrix is distributed among the processes.

The code fragments shown in Figures 2 and 3 are included in the example program file `amg_dexample_ml.f90` too.

Finally, Figure **??** shows the setup of a one-level additive Schwarz preconditioner, i.e., RAS with overlap 2. Note also that a Krylov method different from CG must be used to solve the preconditioned system, since the preconditione in nonsymmetric. The

```
  use psb_base_mod
  use amg_prec_mod
  use psb_krylov_mod
... ...
!
! sparse matrix
  type(psb_dspmat_type) :: A
! sparse matrix descriptor
  type(psb_desc_type)   :: desc_A
! preconditioner
  type(amg_dprec_type)  :: P
! right-hand side and solution vectors
  type(psb_d_vect_type) :: b, x
... ...
!
! initialize the parallel environment
  call psb_init(ctxt)
  call psb_info(ctxt,iam,np)
... ...
!
! read and assemble the spd matrix A and the right-hand side b
! using PSBLAS routines for sparse matrix / vector management
... ...
!
! initialize the default multilevel preconditioner, i.e. V-cycle
! with basic smoothed aggregation, 1 hybrid forward/backward
! GS sweep as pre/post-smoother and UMFPACK as coarsest-level
! solver
  call P%init('ML',info)
!
! build the preconditioner
  call P%hierarchy_build(A,desc_A,info)
  call P%smoothers_build(A,desc_A,info)

!
! set the solver parameters and the initial guess
... ...
!
! solve Ax=b with preconditioned CG
  call psb_krylov('CG',A,P,b,x,tol,desc_A,info)
... ...
!
! deallocate the preconditioner
  call P%free(info)
!
! deallocate other data structures
... ...
!
! exit the parallel environment
  call psb_exit(ctxt)
stop
```

Listing 1: setup and application of the default multilevel preconditioner (example 1).

corresponding example program is available in the file `amg_dexample_1lev.f90`.

For all the previous preconditioners, example programs where the sparse matrix and the right-hand side are generated by discretizing a PDE with Dirichlet boundary conditions are also available in the directory `examples/pdegen`.

```
! build a V-cycle preconditioner with 1 block-Jacobi sweep (with
! ILU(0) on the blocks) as pre- and post-smoother, and 8  block-Jacobi
! sweeps (with ILU(0) on the blocks) as coarsest-level solver
call P%init('ML',info)
call P%set('SMOOTHER_TYPE','BJAC',info)
call P%set('COARSE_SOLVE','BJAC',info)
call P%set('COARSE_SWEEPS',8,info)
call P%hierarchy_build(A,desc_A,info)
call P%smoothers_build(A,desc_A,info)
```

Listing 2: setup of a multilevel preconditioner based on the default decoupled coarsening

```
!build a W-cycle using the coupled coarsening based on weighted matching,
!aggregates of size at most 8 and smoothed prolongators,
!2 hybrid Gauss-Seidel sweeps as pre- and post-smoother,
!and parallel flexible Conjugate Gradient coupled with the block-Jacobi
!preconditioner having ILU(0) on the blocks as coarsest solver.
call P%init('ML',info)
call P%set('PAR_AGGR_ALG','COUPLED',info)
call P%set('AGGR_TYPE','MATCHBOXP',info)
call P%set('AGGR_SIZE',8,info)
call P%set('ML_CYCLE','WCYCLE',info)
call P%set('SMOOTHER_TYPE','FBGS',info)
call P%set('SMOOTHER_SWEEPS',2,info)
call P%set('COARSE_SOLVE','KRM',info)
call P%hierarchy_build(A,desc_A,info)
call P%smoothers_build(A,desc_A,info)
```

Listing 3: setup of a multilevel preconditioner based on the coupled coarsening using weighted matching

```
! build a one-level RAS with overlap 2 and ILU(0) on the local blocks.
call P%init('AS',info)
call P%set('SUB_OVR',2,info)
call P%build(A,desc_A,info)
... ...
! solve Ax=b with preconditioned BiCGSTAB
  call psb_krylov('BICGSTAB',A,P,b,x,tol,desc_A,info)
```

Listing 4: setup of a one-level Schwarz preconditioner.

# 5 User Interface

The basic user interface of AMG4PBLAS consists of eight methods. The six methods `init`, `set`, `build`, `hierarchy_build`, `smoothers_build` and `apply` encapsulate all the functionalities for the setup and the application of any multilevel and one-level preconditioner implemented in the package. The method `free` deallocates the preconditioner data structure, while `descr` prints a description of the preconditioner setup by the user. For backward compatibility, methods are also accessible as stand-alone subroutines.

For each method, the same user interface is overloaded with respect to the real/-complex and single/double precision data; arguments with appropriate data types must be passed to the method, i.e.,

- the sparse matrix data structure, containing the matrix to be preconditioned, must be of type `psb_`$x$`spmat_type` with $x$ = `s` for real single precision, $x$ = `d` for real double precision, $x$ = `c` for complex single precision, $x$ = `z` for complex double precision;
- the preconditioner data structure must be of type `amg_`$x$`prec_type`, with $x$ = `s`, `d`, `c`, `z`, according to the sparse matrix data structure;
- the arrays containing the vectors $v$ and $w$ involved in the preconditioner application $w = B^{-1}v$ must be of type `psb_`$x$`vect_type` with $x$ = `s`, `d`, `c`, `z`, in a manner completely analogous to the sparse matrix type;
- real parameters defining the preconditioner must be declared according to the precision of the sparse matrix and preconditioner data structures (see Section 5.2).

A description of each method is given in the remainder of this section.

### 5.1 Method init

```
call p%init(contxt,ptype,info)
```

This method allocates and initializes the preconditioner p, according to the preconditioner type chosen by the user.

#### Arguments

| | |
|---|---|
| contxt | type(psb_ctxt_type), intent(in).<br>The communication context. |
| ptype | character(len=*), intent(in).<br>The type of preconditioner. Its values are specified in Table 1.<br>Note that strings are case insensitive. |
| info | integer, intent(out).<br>Error code. If no error, 0 is returned. See Section 7 for details. |

## 5.2 Method set

```
call p%set(what,val,info [,ilev, ilmax, pos, idx])
```

This method sets the parameters defining the preconditioner `p`. More precisely, the parameter identified by `what` is assigned the value contained in `val`.

### Arguments

`what` `character(len=*)`.
The parameter to be set. It can be specified through its name; the string is case-insensitive. See Tables 2-8.

`val` `integer` *or* `character(len=*)` *or* `real(psb_spk_)` *or* `real(psb_dpk_)`, `intent(in)`.
The value of the parameter to be set. The list of allowed values and the corresponding data types is given in Tables 2-8. When the value is of type `character(len=*)`, it is also treated as case insensitive.

`info` `integer, intent(out)`.
Error code. If no error, 0 is returned. See Section 7 for details.

`ilev` `integer, optional, intent(in)`.
For the multilevel preconditioner, the level at which the preconditioner parameter has to be set. The levels are numbered in increasing order starting from the finest one, i.e., level 1 is the finest level. If `ilev` is not present, the parameter identified by `what` is set at all levels that are appropriate (see Tables 2-8).

`ilmax` `integer, optional, intent(in)`.
For the multilevel preconditioner, when both `ilev` and `ilmax` are present, the settings are applied at all levels `ilev:ilmax`. When `ilev` is present but `ilmax` is not, then the default is `ilmax=ilev`. The levels are numbered in increasing order starting from the finest one, i.e., level 1 is the finest level.

`pos` `character(len=*), optional, intent(in)`.
Whether the other arguments apply only to the pre-smoother (`'PRE'`) or to the post-smoother (`'POST'`). If `pos` is not present, the other arguments are applied to both smoothers. If the preconditioner is one-level or the parameter identified by `what` does not concern the smoothers, `pos` is ignored.

`idx` `integer, optional, intent(in)`.
An auxiliary input argument that can be passed to the underlying objects.

A variety of preconditioners can be obtained by setting the appropriate preconditioner parameters. These parameters can be logically divided into four groups, i.e., parameters defining

1. the type of multilevel cycle and how many cycles must be applied;

2. the coarsening algorithm;
3. the solver at the coarsest level (for multilevel preconditioners only);
4. the smoother of the multilevel preconditioners, or the one-level preconditioner.

A list of the parameters that can be set, along with their allowed and default values, is given in Tables 2-8.

**Remark 2.** A smoother is usually obtained by combining two objects: a smoother (`'SMOOTHER_TYPE'`) and a local solver (`'SUB_SOLVE'`), as specified in Tables 7-8. For example, the block-Jacobi smoother using ILU(0) on the blocks is obtained by combining the block-Jacobi smoother object with the ILU(0) solver object. Similarly, the hybrid Gauss-Seidel smoother (see Note in Table 7) is obtained by combining the block-Jacobi smoother object with a single sweep of the Gauss-Seidel solver object, while the point-Jacobi smoother is the result of combining the block-Jacobi smoother object with a single sweep of the point-Jacobi solver object. In the same way are obtained the $\ell_1$-versions of the smoothers. However, for simplicity, shortcuts are provided to set all versions of point-Jacobi, hybrid (forward) Gauss-Seidel, and hybrid backward Gauss-Seidel, i.e., the previous smoothers can be defined just by setting `'SMOOTHER_TYPE'` to certain specific values (see Tables 7), without the need to set `'SUB_SOLVE'` as well.

The smoother and solver objects are arranged in a hierarchical manner. When specifying a smoother object, its parameters, including the local solver, are set to their default values, and when a solver object is specified, its defaults are also set, overriding in both cases any previous settings even if explicitly specified. Therefore if the user sets a smoother, and wishes to use a solver different from the default one, the call to set the solver must come *after* the call to set the smoother.

Similar considerations apply to the point-Jacobi, Gauss-Seidel and block-Jacobi coarsest-level solvers, and shortcuts are available in this case too (see Table 5).

**Remark 3.** Many of the coarsest-level solvers cannot be used with both replicated and distributed coarsest-matrix layouts; therefore, setting the solver after the layout may change the layout. Similarly, setting the layout after the solver may change the solver.

More precisely, UMFPACK and SuperLU require the coarsest-level matrix to be replicated, while SuperLU_Dist and KRM requires it to be distributed. In these cases, setting the coarsest-level solver implies that the layout is redefined according to the solver, ovverriding any previous settings. MUMPS, point-Jacobi, hybrid Gauss-Seidel and block-Jacobi can be applied to replicated and distributed matrices, thus their choice does not modify any previously specified layout. It is worth noting that, when the matrix is replicated, the point-Jacobi, hybrid Gauss-Seidel and block-Jacobi solvers and their $\ell_1-$ versions reduce to the corresponding local solver objects (see Remark 2). For the point-Jacobi and Gauss-Seidel solvers, these objects correspond to a *single* point-Jacobi sweep and a *single* Gauss-Seidel sweep, respectively, which are very poor solvers.

On the other hand, the distributed layout can be used with any solver but UMFPACK and SuperLU; therefore, if any of these two solvers has already been selected, the coarsest-level solver is changed to block-Jacobi, with the previously chosen solver applied to the local blocks. Likewise, the replicated layout can be used with any solver but SuperLu_Dist; therefore, if SuperLu_Dist has been previously set, the coarsest-level solver is changed to the default sequential solver.

**Remark 4.** The argument `idx` can be used to allow finer control for those solvers; for instance, by specifying the keyword `'MUMPS_IPAR_ENTRY'` and an appropriate value for `idx`, it is possible to set any entry in the MUMPS integer control array. See also Sec. 6.

| what | DATA TYPE | val | DEFAULT | COMMENTS |
|---|---|---|---|---|
| 'ML_CYCLE' | character(len=*) | 'VCYCLE'<br>'WCYCLE'<br>'KCYCLE'<br>'ADD' | 'VCYCLE' | Multilevel cycle: V-cycle, W-cycle, K-cycle, and additive composition. |
| 'CYCLE_SWEEPS' | integer | Any integer number $\geq 1$ | 1 | Number of multilevel cycles. |

Table 2: Parameters defining the multilevel cycle and the number of cycles to be applied.

| what | DATA TYPE | val | DEFAULT | COMMENTS |
|---|---|---|---|---|
| 'MIN_COARSE_SIZE_PER_PROCESS' | integer | Any number $> 0$ | 200 | Coarse size threshold per process. The aggregation stops if the global number of variables of the computed coarsest matrix is lower than or equal to this threshold multiplied by the number of processes (see Note). |
| 'MIN_COARSE_SIZE' | integer | Any number $> 0$ | -1 | Coarse size threshold. The aggregation stops if the global number of variables of the computed coarsest matrix is lower than or equal to this threshold (see Note). If negative, it is ignored in favour of the default for 'MIN_COARSE_SIZE_PER_PROCESS'. |
| 'MIN_CR_RATIO' | real | Any number $> 1$ | 1.5 | Minimum coarsening ratio. The aggregation stops if the ratio between the global matrix dimensions at two consecutive levels is lower than or equal to this threshold (see Note). |
| 'MAX_LEVS' | integer | Any integer number $> 1$ | 20 | Maximum number of levels. The aggregation stops if the number of levels reaches this value (see Note). |
| 'PAR_AGGR_ALG' | character(len=*) | 'DEC', 'SYMDEC', 'COUPLED' | 'DEC' | Parallel aggregation algorithm. the SYMDEC option applies decoupled aggregation to the sparsity pattern of $A + A^T$. |

| what | DATA TYPE | val | DEFAULT | COMMENTS |
|---|---|---|---|---|
| 'AGGR_TYPE' | character(len=*) | 'SOC1', 'SOC2', 'MATCHBOXP' | 'SOC1' | Type of aggregation algorithm: currently, for the decoupled aggregation we implement two measures of strength of connection, the one by Vaněk, Mandel and Brezina [33], and the one by Gratton et al [23]. The coupled aggregation is based on a parallel version of the half-approximate matching implemented in the MatchBox-P software package [9]. |
| 'AGGR_SIZE' | integer | Any integer power of 2, with aggr_size $\geq$ 2 | 4 | Maximum size of aggregates when the coupled aggregation based on matching is applied. For aggressive coarsening with size of aggregate larger than 8 we recommend the use of smoothed prolongators. Used only with 'COUPLED' and 'MATCHBOXP' |
| 'AGGR_PROL' | character(len=*) | 'SMOOTHED', 'UNSMOOTHED' | 'SMOOTHED' | Prolongator used by the aggregation algorithm: smoothed or unsmoothed (i.e., tentative prolongator). |
| **Note.** The aggregation algorithm stops when at least one of the following criteria is met: the coarse size threshold, the minimum coarsening ratio, or the maximum number of levels is reached. Therefore, the actual number of levels may be smaller than the specified maximum number of levels. | | | | |

Table 3: Parameters defining the aggregation algorithm.

| what | DATA TYPE | val | DEFAULT | COMMENTS |
|---|---|---|---|---|
| 'AGGR_ORD' | character(len=*) | 'NATURAL'<br>'DEGREE' | 'NATURAL' | Initial ordering of indices for the decoupled aggregation algorithm: either natural ordering or sorted by descending degrees of the nodes in the matrix graph. |
| 'AGGR_THRESH' | real(kind_parameter) | Any real number $\in [0, 1]$ | 0.01 | The threshold $\theta$ in the strength of connection algorithm. See also the note at the bottom of this table. |
| 'AGGR_FILTER' | character(len=*) | 'FILTER'<br>'NOFILTER' | 'NOFILTER' | Matrix used in computing the smoothed prolongator: filtered or unfiltered. |
| **Note.** Different thresholds at different levels, such as those used in [33, Section 5.1], can be easily set by invoking the routine set with the parameter ilev. | | | | |

Table 4: Parameters defining the aggregation algorithm (continued).

| what | DATA TYPE | val | DEFAULT | COMMENTS |
|---|---|---|---|---|
| 'COARSE_MAT' | character(len=*) | 'DIST'<br>'REPL' | 'REPL' | Coarsest matrix layout: distributed among the processes or replicated on each of them. |
| 'COARSE_SOLVE' | character(len=*) | 'MUMPS'<br>'UMF'<br>'SLU'<br>'SLUDIST'<br>'JACOBI'<br>'GS'<br>'BJAC'<br>'KRM' | See Note. | Solver used at the coarsest level: sequential LU from MUMPS, UMFPACK, or SuperLU (plus triangular solve); distributed LU from MUMPS or SuperLU_Dist (plus triangular solve); point-Jacobi, hybrid Gauss-Seidel or block-Jacobi and related $\ell_1$-versions; Krylov Method (flexible Conjugate Gradient) coupled with the block-Jacobi preconditioner with ILU(0) on the blocks. Note that UMF and SLU require the coarsest matrix to be replicated, SLUDIST, JACOBI, GS, BJAC and KRM require it to be distributed, MUMPS can be used with either a replicated or a distributed matrix. When any of the previous solvers is specified, the matrix layout is set to a default value which allows the use of the solver (see Remark 3, p. 24). Note also that UMFPACK and SuperLU_Dist are available only in double precision. |

| | | | | |
|---|---|---|---|---|
| 'COARSE_SUBSOLVE' | character(len=*) | 'ILU'<br>'ILUT'<br>'MILU'<br>'MUMPS'<br>'SLU'<br>'UMF'<br>'INVT'<br>'INVK'<br>'AINV' | See Note. | Solver for the diagonal blocks of the coarsest matrix, in case the block Jacobi solver is chosen as coarsest-level solver: ILU($p$), ILU($p,t$), MILU($p$), LU from MUMPS, SuperLU or UMFPACK (plus triangular solve), Approximate Inverses INVK($p,q$), INVT($p_1, p2, t_1, t_2$) and AINV($t$); note that approximate inverses are specifically suited for GPUs since they do not employ triangular system solve kernels, see [3]. Note that UMFPACK and SuperLU_Dist are available only in double precision. |
| **Note.** Defaults for COARSE_SOLVE and COARSE_SUBSOLVE are chosen in the following order:<br>single precision version – MUMPS if installed, then SLU if installed, ILU otherwise;<br>double precision version – UMF if installed, then MUMPS if installed, then SLU if installed, ILU otherwise. | | | | |
| what | DATA TYPE | val | DEFAULT | COMMENTS |
| 'COARSE_SWEEPS' | integer | Any integer number $> 0$ | 10 | Number of sweeps when JACOBI, GS or BJAC is chosen as coarsest-level solver. |
| 'COARSE_FILLIN' | integer | Any integer number $\geq 0$ | 0 | Fill-in level $p$ of the ILU factorizations and first fill-in for the approximate inverses. |
| 'COARSE_ILUTHRS' | real(kind_parameter) | Any real number $\geq 0$ | 0 | Drop tolerance $t$ in the ILU($p,t$) factorization and first drop-tolerance for the approximate inverses. |
| **Note.** Further options for coarse solvers are contained in Table 6.<br>For a first use it is suggested to use the default options obtained by simply selecting the solver type. | | | | |

Table 5: Parameters defining the solver at the coarsest level (continued).

| what | DATA TYPE | val | DEFAULT | COMMENTS |
|---|---|---|---|---|
| 'BJAC_STOP' | character(len=*) | 'FALSE'<br>'TRUE' | 'FALSE' | Select whether to use a stopping criterion for the Block-Jacobi method used as a coarse solver. |
| 'BJAC_TRACE' | character(len=*) | 'FALSE'<br>'TRUE' | 'FALSE' | Select whether to print a trace for the calculated residual for the Block-Jacobi method used as a coarse solver. |
| 'BJAC_ITRACE' | integer | Any integer $> 0$ | -1 | Number of iterations after which a trace is to be printed. |
| 'BJAC_RESCHECK' | integer | Any integer $> 0$ | -1 | Number of iterations after which a residual is to be calculated. |
| 'BJAC_STOPTOL' | real(kind_parameter) | Any real $< 1$ | 0 | Tolerance for the stopping criterion on the residual. |
| 'KRM_METHOD' | character(len=*) | 'CG'<br>'FCG'<br>'CGS'<br>'CGR'<br>'BICG'<br>'BICGSTAB'<br>'BICGSTABL'<br>'RGMRES' | 'FCG' | A string that defines the iterative method to be used when employing a Krylov method 'KRM' as a coarse solver. CG the Conjugate Gradient method; CGS the Conjugate Gradient Stabilized method; GCR the Generalized Conjugate Residual method; FCG the Flexible Conjugate Gradient method; BICG the Bi-Conjugate Gradient method; BICGSTAB the Bi-Conjugate Gradient Stabilized method; BICGSTABL the Bi-Conjugate Gradient Stabilized method with restarting; RGMRES the Generalized Minimal Residual method with restarting. Refer to the PSBLAS guide [20] for further information. |
| 'KRM_KPREC' | character(len=*) | Table 1 | 'BJAC' | The one-level preconditioners from the Table 1 can be used for the coarse Krylov solver. |

| | | | | |
|---|---|---|---|---|
| 'KRM_SUB_SOLVE' | character(len=*) | Table 5 | 'ILU' | Solver for the diagonal blocks of the coarsest matrix preconditioner, in case the block Jacobi solver is chosen as 'KRM_KPREC': ILU($p$), ILU($p,t$), MILU($p$), LU from MUMPS, SuperLU or UMFPACK (plus triangular solve), Approximate Inverses INVK($p,q$), INVT($p_1$, $p2$, $t_1$, $t_2$) and AINV($t$); The same caveat from Table 5 applies here. |
| 'KRM_GLOBAL' | character(len=*) | 'TRUE', 'FALSE' | 'FALSE' | Choose between a global Krylov solver, all unknowns on a single node, or a distributed one. The default choice is the distributed solver. |
| 'KRM_EPS' | real(kind_parameter) | Real $< 1$ | $10^{-6}$ | The stopping tolerance. |
| 'KRM_IRST' | integer | Integer $\geq 1$ | 30 | An integer specifying the restart parameter. This is employed for the BiCGSTABL or RGMRES methods, otherwise it is ignored. |
| 'KRM_ISTOPC' | integer | Integers 1,2,3 | 2 | If 1 then the method uses the normwise backward error in the infinity norm; if 2, the it uses the relative residual in the 2-norm; if 3 the relative residual reduction in the 2-norm is used instead; refer to the PSBLAS [20] guide for the details. |
| 'KRM_ITMAX' | integer | Integer $\geq 1$ | 40 | The maximum number of iterations to perform. |
| 'KRM_ITRACE' | integer | Integer $\geq 0$ | -1 | If $> 0$ print out an informational message about convergence every 'KRM_ITRACE' iterations. If $= 0$ print a message in case of convergence failure. |
| 'KRM_FILLIN' | integer | Integer $\geq 0$ | 0 | Fill-in level $p$ of the ILU factorizations and first fill-in for the approximate inverses. |

Table 6: Additional parameters defining the solver at the coarsest level.

| what | DATA TYPE | val | DEFAULT | COMMENTS |
|---|---|---|---|---|
| 'SMOOTHER_TYPE' | character(len=*) | 'JACOBI'<br>'GS'<br>'BGS'<br>'BJAC'<br>'AS'<br>'L1-JACOBI'<br>'L1-BJAC'<br>'L1-FBGS' | 'FBGS' | Type of smoother used in the multilevel preconditioner: point-Jacobi, hybrid (forward) Gauss-Seidel, hybrid backward Gauss-Seidel, block-Jacobi, $\ell_1$-Jacobi, $\ell_1$–hybrid (forward) Gauss-Seidel, $\ell_1$-point-Jacobi and Additive Schwarz.<br>It is ignored by one-level preconditioners. |
| 'SUB_SOLVE' | character(len=*) | 'JACOBI'<br>'GS'<br>'BGS'<br>'ILU'<br>'ILUT'<br>'MILU'<br>'MUMPS'<br>'SLU'<br>'UMF'<br>'INVT'<br>'INVK'<br>'AINV' | GS and BGS for pre- and post-smoothers of multilevel preconditioners, respectively<br>ILU for block-Jacobi and Additive Schwarz one-level preconditioners | The local solver to be used with the smoother or one-level preconditioner (see Remark 2, page 24): point-Jacobi, hybrid (forward) Gauss-Seidel, hybrid backward Gauss-Seidel, ILU($p$), ILU($p,t$), MILU($p$), LU from MUMPS, SuperLU or UMFPACK (plus triangular solve), Approximate Inverses INVK($p,q$), INVT($p_1, p2, t_1, t_2$) and AINV($t$); note that approximate inverses are specifically suited for GPUs since they do not employ triangular system solve kernels, see [3]. See Note for details on hybrid Gauss-Seidel. |
| 'SMOOTHER_SWEEPS' | integer | Any integer number $\geq 0$ | 1 | Number of sweeps of the smoother or one-level preconditioner. In the multilevel case, no pre-smother or post-smoother is used if this parameter is set to 0 together with pos='PRE' or pos='POST', respectively. |
| 'SUB_OVR' | integer | Any integer number $\geq 0$ | 1 | Number of overlap layers, for Additive Schwarz only. |

Table 7: Parameters defining the smoother or the details of the one-level preconditioner.

| what | DATA TYPE | val | DEFAULT | COMMENTS |
|---|---|---|---|---|
| 'SUB_RESTR' | character(len=*) | 'HALO'<br>'NONE' | 'HALO' | Type of restriction operator, for Additive Schwarz only: HALO for taking into account the overlap, 'NONE' for neglecting it.<br>Note that HALO must be chosen for the classical Addditive Schwarz smoother and its RAS variant. |
| 'SUB_PROL' | character(len=*) | 'SUM'<br>'NONE' | 'NONE' | Type of prolongation operator, for Additive Schwarz only: 'SUM' for adding the contributions from the overlap, 'NONE' for neglecting them.<br>Note that 'SUM' must be chosen for the classical Additive Schwarz smoother, and 'NONE' for its RAS variant. |
| 'SUB_FILLIN' | integer | Any integer number $\geq 0$ | 0 | Fill-in level $p$ of the incomplete LU factorizations. |
| 'SUB_ILUTHRS' | real(kind_parameter) | Any real number $\geq 0$ | 0 | Drop tolerance $t$ in the ILU($p,t$) factorization. |
| 'MUMPS_LOC_GLOB' | character(len=*) | 'LOCAL_SOLVER'<br>'GLOBAL_SOLVER' | 'GLOBAL_SOLVER' | Whether MUMPS should be used as a distributed solver, or as a serial solver acting only on the part of the matrix local to each process. |
| 'MUMPS_IPAR_ENTRY' | integer | Any integer number | 0 | Set an entry in the MUMPS integer control array, as chosen via the idx optional argument. |
| 'MUMPS_RPAR_ENTRY' | real | Any real number | 0 | Set an entry in the MUMPS real control array, as chosen via the idx optional argument. |

Table 8: Parameters defining the smoother or the details of the one-level preconditioner (continued).

### 5.3 Method hierarchy_build

```
call p%hierarchy_build(a,desc_a,info)
```

This method builds the hierarchy of matrices and restriction/prolongation operators for the multilevel preconditioner `p`, according to the requirements made by the user through the methods `init` and `set`.

#### Arguments

`a` `type(psb_`*x*`spmat_type), intent(in).`
The sparse matrix structure containing the local part of the matrix to be preconditioned. Note that *x* must be chosen according to the `real`/`complex`, single/double precision version of AMG4PSBLAS under use. See the PSBLAS User's Guide for details [20].

`desc_a` `type(psb_desc_type), intent(in).`
The communication descriptor of `a`. See the PSBLAS User's Guide for details [20].

`info` `integer, intent(out).`
Error code. If no error, 0 is returned. See Section 7 for details.

## 5.4 Method smoothers_build

```
call p%smoothers_build(a,desc_a,p,info[,amold,vmold,imold])
```

This method builds the smoothers and the coarsest-level solvers for the multilevel preconditioner p, according to the requirements made by the user through the methods init and set, and based on the aggregation hierarchy produced by a previous call to hierarchy_build (see Section 5.3).

### Arguments

a — type(psb_*x*spmat_type), intent(in).
The sparse matrix structure containing the local part of the matrix to be preconditioned. Note that *x* must be chosen according to the real/complex, single/double precision version of AMG4PSBLAS under use. See the PSBLAS User's Guide for details [20].

desc_a — type(psb_desc_type), intent(in).
The communication descriptor of a. See the PSBLAS User's Guide for details [20].

info — integer, intent(out).
Error code. If no error, 0 is returned. See Section 7 for details.

amold — class(psb_*x*_base_sparse_mat), intent(in), optional.
The desired dynamic type for internal matrix components; this allows e.g. running on GPUs; it needs not be the same on all processes. See the PSBLAS User's Guide for details [20].

vmold — class(psb_*x*_base_vect_type), intent(in), optional.
The desired dynamic type for internal vector components; this allows e.g. running on GPUs.

imold — class(psb_i_base_vect_type), intent(in), optional.
The desired dynamic type for internal integer vector components; this allows e.g. running on GPUs.

### 5.5 Method build

```
call p%build(a,desc_a,info[,amold,vmold,imold])
```

This method builds the preconditioner p according to the requirements made by the user through the methods `init` and `set` (see Sections 5.3 and 5.4 for multilevel preconditioners). It is mostly provided for backward compatibility; indeed, it is internally implemented by invoking the two previous methods `hierarchy_build` and `smoothers_build`, whose nomenclature would however be somewhat unnatural when dealing with simple one-level preconditioners.

#### Arguments

`a` — `type(psb_`*x*`spmat_type), intent(in).`
The sparse matrix structure containing the local part of the matrix to be preconditioned. Note that *x* must be chosen according to the `real`/`complex`, single/double precision version of AMG4PSBLAS under use. See the PSBLAS User's Guide for details [20].

`desc_a` — `type(psb_desc_type), intent(in).`
The communication descriptor of `a`. See the PSBLAS User's Guide for details [20].

`info` — `integer, intent(out).`
Error code. If no error, 0 is returned. See Section 7 for details.

`amold` — `class(psb_`*x*`_base_sparse_mat), intent(in), optional.`
The desired dynamic type for internal matrix components; this allows e.g. running on GPUs; it needs not be the same on all processes. See the PSBLAS User's Guide for details [20].

`vmold` — `class(psb_`*x*`_base_vect_type), intent(in), optional.`
The desired dynamic type for internal vector components; this allows e.g. running on GPUs.

`imold` — `class(psb_i_base_vect_type), intent(in), optional.`
The desired dynamic type for internal integer vector components; this allows e.g. running on GPUs.

The method can be used to build multilevel preconditioners too.

## 5.6 Method apply

```
call p%apply(x,y,desc_a,info [,trans,work])
```

This method computes $y = op(B^{-1})\,x$, where $B$ is a previously built preconditioner, stored into `p`, and $op$ denotes the preconditioner itself or its transpose, according to the value of `trans`. Note that, when AMG4PSBLAS is used with a Krylov solver from PSBLAS, `p%apply` is called within the PSBLAS method `psb_krylov` and hence it is completely transparent to the user.

### Arguments

`x` *type*(`kind_parameter`), dimension(:), intent(in)—.
The local part of the vector $x$. Note that *type* and *kind_parameter* must be chosen according to the `real`/`complex`, single/double precision version of AMG4PSBLAS under use.

`y` *type*(`kind_parameter`), dimension(:), intent(out)—.
The local part of the vector $y$. Note that *type* and *kind_parameter* must be chosen according to the `real`/`complex`, single/double precision version of AMG4PSBLAS under use.

`desc_a` `type(psb_desc_type), intent(in)`.
The communication descriptor associated to the matrix to be preconditioned.

`info` `integer, intent(out)`.
Error code. If no error, 0 is returned. See Section 7 for details.

`trans` `character(len=1), optional, intent(in)`.
If `trans = 'N','n'` then $op(B^{-1}) = B^{-1}$; if `trans = 'T','t'` then $op(B^{-1}) = B^{-T}$ (transpose of $B^{-1}$); if `trans = 'C','c'` then $op(B^{-1}) = B^{-C}$ (conjugate transpose of $B^{-1}$).

`work` *type*(`kind_parameter`), dimension(:), optional, target—.
Workspace. Its size should be at least `4 * psb_cd_get_local_cols(desc_a)` (see the PSBLAS User's Guide). Note that *type* and *kind_parameter* must be chosen according to the `real`/`complex`, single/double precision version of AMG4PSBLAS under use.

### 5.7 Method free

```
call p%free(p,info)
```

This method deallocates the preconditioner data structure p.

#### Arguments

| | |
|---|---|
| info | integer, intent(out).<br>Error code. If no error, 0 is returned. See Section 7 for details. |

## 5.8 Method descr

```
call p%descr(info, [iout, root, verbosity])
```

This method prints a description of the preconditioner p to the standard output or to a file. It must be called after `hierachy_build` and `smoothers_build`, or `build`, have been called.

### Arguments

| | |
|---|---|
| `info` | `integer, intent(out).`<br>Error code. If no error, 0 is returned. See Section 7 for details. |
| `iout` | `integer, intent(in), optional.`<br>The id of the file where the preconditioner description will be printed; the default is the standard output. |
| `root` | `integer, intent(in), optional.`<br>The id of the process where the preconditioner description will be printed; the default is `psb_root_`. |
| `verbosity` | `integer, intent(in), optional.`<br>The verbosity level of the description. Default value is 0. For values higher than 0, it prints out further information, e.g., for a distributed multilevel preconditioner the size of the coarse matrices on every process. |

## 5.9 Auxiliary Methods

Various functionalities are implemented as additional methods of the preconditioner object.

### 5.9.1 Method: dump

```
call p%dump(info[,istart,iend,prefix,head,ac,rp,smoother,solver,global_num])
```

Dump on file.

### Arguments

| | |
|---|---|
| `info` | `integer, intent(out).`<br>Error code. If no error, 0 is returned. See Section 7 for details. |
| `amold` | `class(psb_x_base_sparse_mat), intent(in), optional.`<br>The desired dynamic type for internal matrix components; this allows e.g. running on GPUs; it needs not be the same on all processes. See the PSBLAS User's Guide for details [20]. |

### 5.9.2 Method: clone

```
call p%clone(pout,info)
```

Create a (deep) copy of the preconditioner object.

#### Arguments

`pout` `type(amg_`*x*`prec_type), intent(out).`
The copy of the preconditioner data structure. Note that *x* must be chosen according to the `real`/`complex`, single/double precision version of AMG4PSBLAS under use.

`info` `integer, intent(out).`
Error code. If no error, 0 is returned. See Section 7 for details.

### 5.9.3 Method: sizeof

```
sz = p%sizeof([global])
```

`global` `logical, optional.`
Whether the global or local preconditioner memory occupation is desired. Default: `.false.`.

Return memory footprint in bytes.

### 5.9.4 Method: allocate_wrk

```
call p%allocate_wrk(info[, vmold])
```

Allocate internal work vectors. Each application of the preconditioner uses a number of work vectors which are allocated internally as necessary; therefore allocation and deallocation of memory occurs multiple times during the execution of a Krylov method. In most cases this strategy is perfectly acceptable, but on some platforms, most notably GPUs, memory allocation is a slow operation, and the default behaviour would lead to a slowdown. This method allows to trade space for time by preallocating the internal workspace outside of the invocation of a Krylov method. When using GPUs or other specialized devices, the `vmold` argument is also necessary to ensure the internal work vectors are of the appropriate dynamic type to exploit the accelerator hardware; when allocation occurs internally this is taken care of based on the dynamic type of the `x` argument to the `apply` method.

#### Arguments

`info` `integer, intent(out).`
Error code. If no error, 0 is returned. See Section 7 for details.

`vmold` `class(psb_`*x*`_base_vect_type), intent(in), optional.`
The desired dynamic type for internal vector components; this allows e.g. running on GPUs.

### 5.9.5 Method: free_wrk

```
call p%free_wrk(info)
```

Deallocate internal work vectors.

## Arguments

`info` `integer, intent(out).`
Error code. If no error, 0 is returned. See Section 7 for details.

# 6 Adding new smoother and solver objects to AMG4PSBLAS

Developers can add completely new smoother and/or solver classes derived from the base objects in the library (see Remark 2 in Section 5.2), without recompiling the library itself.

To do so, it is necessary first to select the base type to be extended. In our experience, it is quite likely that the new application needs only the definition of a "solver" object, which is almost always acting only on the local part of the distributed matrix. The parallel actions required to connect the various solver objects are most often already provided by the block-Jacobi or the additive Schwarz smoothers. To define a new solver, the developer will then have to define its components and methods, perhaps taking one of the predefined solvers as a starting point, if possible.

Once the new smoother/solver class has been developed, to use it in the context of the multilevel preconditioners it is necessary to:

- declare in the application program a variable of the new type;
- pass that variable as the argument to the `set` routine as in the following:

```
call p%set(smoother,info [,ilev,ilmax,pos])
 call p%set(solver,info [,ilev,ilmax,pos])
```

- link the code implementing the various methods into the application executable.

The new solver object is then dynamically included in the preconditioner structure, and acts as a *mold* to which the preconditioner will conform, even though the AMG4PSBLAS library has not been modified to account for this new development.

It is possible to define new values for the keyword `WHAT` in the `set` routine; if the library code does not recognize a keyword, it passes it down the composition hierarchy (levels containing smoothers containing in turn solvers), so that it can eventually be caught by the new solver. By the same token, any keyword/value pair that does not pertain to a given smoother should be passed down to the contained solver, and any keyword/value pair that does not pertain to a given solver is by default ignored.

An example is provided in the source code distribution under the folder `tests/newslv`. In this example we are implementing a new incomplete factorization variant (which is simply the ILU(0) factorization under a new name). Because of the specifics of this case, it is possible to reuse the basic structure of the ILU solver, with its L/D/U components and the methods needed to apply the solver; only a few methods, such as the description and most importantly the build, need to be ovverridden (rewritten).

The interfaces for the calls shown above are defined using

`smoother` `class(amg_x_base_smoother_type)`
: The user-defined new smoother to be employed in the preconditioner.

`solver` `class(amg_x_base_solver_type)`
: The user-defined new solver to be employed in the preconditioner.

The other arguments are defined in the way described in Sec. 5.2. As an example, in the `tests/newslv` code we define a new object of type `amg_d_tlu_solver_type`, and we pass it as follows:

```
! sparse matrix and preconditioner
type(psb_dspmat_type) :: a
type(amg_dprec_type)  :: prec
type(amg_d_tlu_solver_type) :: tlusv

.......
!
!  prepare the preconditioner: an ML with defaults, but with TLU solver at
!  intermediate levels. All other parameters are at default values.
!
call prec%init('ML',       info)
call prec%hierarchy_build(a,desc_a,info)
nlv = prec%get_nlevs()
call prec%set(tlusv,   info,ilev=1,ilmax=max(1,nlv-1))
call prec%smoothers_build(a,desc_a,info)
```

# 7 Error Handling

The error handling in AMG4PSBLAS is based on the PSBLAS error handling. Error conditions are signaled via an integer argument `info`; whenever an error condition is detected, an error trace stack is built by the library up to the top-level, user-callable routine. This routine will then decide, according to the user preferences, whether the error should be handled by terminating the program or by returning the error condition to the user code, which will then take action, and whether an error message should be printed. These options may be set by using the PSBLAS error handling routines; for further details see the PSBLAS User's Guide [20].

# A License

AMG4PSBLAS is freely distributable under the following copyright terms:


AMG4PSBLAS version 1.0
Algebraic MultiGrid Preconditioners Package
based on PSBLAS (Parallel Sparse BLAS version 3.7)

(C) Copyright 2021

Pasqua D'Ambra IAC-CNR, IT
Fabio Durastante University of Pisa and IAC-CNR, IT
Salvatore Filippone University of Rome Tor-Vergata and IAC-CNR, IT

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions, and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. The name of the MLD2P4 group or the names of its contributors may
   not be used to endorse or promote products derived from this
   software without specific written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE MLD2P4 GROUP OR ITS CONTRIBUTORS
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.

AMG4PSBLAS is an evolution of MLD2P4, whose license we reproduce here to abide by its terms:


```
                          MLD2P4  version 2.2
MultiLevel Domain Decomposition Parallel Preconditioners Package
           based on PSBLAS (Parallel Sparse BLAS version 3.5)

(C) Copyright 2008-2018

    Salvatore Filippone
    Pasqua D'Ambra
    Daniela di Serafino


Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
  1. Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.
  2. Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions, and the following disclaimer in the
     documentation and/or other materials provided with the distribution.
  3. The name of the MLD2P4 group or the names of its contributors may
     not be used to endorse or promote products derived from this
     software without specific written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE MLD2P4 GROUP OR ITS CONTRIBUTORS
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
POSSIBILITY OF SUCH DAMAGE.
```

# B Contributor Covenant Code of Conduct

**Our Pledge** We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation. We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community. Our Standards Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

**Enforcement Responsibilities** Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful. Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate. Scope This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed

representative at an online or offline event. Enforcement Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at eocoe@na.iac.cnr.it. All complaints will be reviewed and investigated promptly and fairly. All community leaders are obligated to respect the privacy and security of the reporter of any incident.

**Enforcement Guidelines** Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

   *Community Impact:* Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

   *Consequence:* A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

   *Community Impact:* A violation through a single incident or series of actions.

   *Consequence:* A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

   *Community Impact:* A serious violation of community standards, including sustained inappropriate behavior.

   *Consequence:* A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

   *Community Impact:* Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

   *Consequence:* A permanent ban from any sort of public interaction within the community.

**Attribution** This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html. Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder. For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

# References

[1] A. Aprovitola, P. D'Ambra, F. Denaro, D. di Serafino, S. Filippone, *Scalable algebraic multilevel preconditioners with application to CFD*, in Proc. of CFD 2008, LNCSE, 74, (2010), 15–27.

[2] P. R. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J. L'Excellent, C. Weisbecker, *Improving multifrontal methods by means of block low-rank representations*, SIAM Journal on Scientific Computing, volume 37 (3), 2015, A1452–A1474. See also `http://mumps.enseeiht.fr`.

[3] D. Bertaccini and S. Filippone, *Sparse approximate inverse preconditioners on high performance GPU platforms*, Comput. Math. Appl., 71, (2016), no. 3, 693–711.

[4] M. Brezina, P. Vaněk, *A Black-Box Iterative Solver Based on a Two-Level Schwarz Method*, Computing, 63, 1999, 233–263.

[5] W. L. Briggs, V. E. Henson, S. F. McCormick, *A Multigrid Tutorial, Second Edition*, SIAM, 2000.

[6] A. Buttari, P. D'Ambra, D. di Serafino, S. Filippone, *Extending PSBLAS to Build Parallel Schwarz Preconditioners*, in J. Dongarra, K. Madsen, J. Wasniewski, editors, Proceedings of PARA 04 Workshop on State of the Art in Scientific Computing, Lecture Notes in Computer Science, Springer, 2005, 593–602.

[7] A. Buttari, P. D'Ambra, D. di Serafino, S. Filippone, *2LEV-D2P4: a package of high-performance preconditioners for scientific and engineering applications*, Applicable Algebra in Engineering, Communications and Computing, 18 (3) 2007, 223–239.

[8] X. C. Cai, M. Sarkis, *A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems*, SIAM Journal on Scientific Computing, 21 (2), 1999, 792–797.

[9] U.. V. Catalyurek, F. Dobrian, A. Gebremedhin, M. Halappanavar, and A. Pothen, *Distributed-memory parallel algorithms for matching and coloring*, in PCO'11 New Trends in Parallel Computing and Optimization, IEEE International Symposium on Parallel and Distributed Processing Workshops, IEEE CS, 2011.

[10] P. D'Ambra, S. Filippone, D. di Serafino, *On the Development of PSBLAS-based Parallel Two-level Schwarz Preconditioners*, Applied Numerical Mathematics, Elsevier Science, 57 (11-12), 2007, 1181-1196.

[11] P. D'Ambra, D. di Serafino, S. Filippone, *MLD2P4: a Package of Parallel Multilevel Algebraic Domain Decomposition Preconditioners in Fortran 95*, ACM Trans. Math. Softw., 37(3), 2010, art. 30.

[12] P. D'Ambra and P. S. Vassilevski, *Adaptive AMG with coarsening based on compatible weighted matching*, Computing and Visualization in Science, 16, (2013) 59–76.

[13] P. D'Ambra, S. Filippone and P. S. Vassilevski, *BootCMatch: a software package for bootstrap AMG based on graph weighted matching*, ACM Transactions on Mathematical Software, 44, (2018) 39:1–39:25.

[14] P. D'Ambra, F Durastante, S. Filippone, *AMG preconditioners for Linear Solvers towards Extreme Scale*, 2020, arXiv:2006.16147v3.

[15] T. A. Davis, *Algorithm 832: UMFPACK - an Unsymmetric-pattern Multifrontal Method with a Column Pre-ordering Strategy*, ACM Transactions on Mathematical Software, 30, 2004, 196–199. (See also `http://www.cise.ufl.edu/~davis/`)

[16] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM Journal on Matrix Analysis and Applications, 20 (3), 1999, 720–755.

[17] J. J. Dongarra, J. Du Croz, I. S. Duff, S. Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 16 (1) 1990, 1–17.

[18] J. J. Dongarra, J. Du Croz, S. Hammarling, R. J. Hanson, *An extended set of FORTRAN Basic Linear Algebra Subprograms*, ACM Transactions on Mathematical Software, 14 (1) 1988, 1–17.

[19] S. Filippone, P. D'Ambra, M. Colajanni, *Using a Parallel Library of Sparse Linear Algebra in a Fluid Dynamics Application Code on Linux Clusters*, in Proc. of ParCo 2001, Parallel Computing, Advances and Current Issues, 2002.

[20] S. Filippone, A. Buttari, *PSBLAS 3.5.0 User's Guide. A Reference Guide for the Parallel Sparse BLAS Library*, 2012, available from `https://github.com/sfilippone/psblas3/tree/master/docs`.

[21] S. Filippone, A. Buttari, *Object-Oriented Techniques for Sparse Matrix Computations in Fortran 2003*. ACM Transactions on on Mathematical Software, 38 (4), 2012, art. 23.

[22] S. Filippone, M. Colajanni, *PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices*, ACM Transactions on Mathematical Software, 26 (4), 2000, 527–550.

[23] S. Gratton, P. Henon, P. Jiranek and X. Vasseur, *Reducing complexity of algebraic multigrid by aggregation*, Numerical Lin. Algebra with Applications, 2016, 23:501-518

[24] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, M. Snir, *MPI: The Complete Reference. Volume 2 - The MPI-2 Extensions*, MIT Press, 1998.

[25] C. L. Lawson, R. J. Hanson, D. Kincaid, F. T. Krogh, *Basic Linear Algebra Subprograms for FORTRAN usage*, ACM Transactions on Mathematical Software, 5 (3), 1979, 308–323.

[26] X. S. Li, J. W. Demmel, *SuperLU_DIST: A Scalable Distributed-memory Sparse Direct Solver for Unsymmetric Linear Systems*, ACM Transactions on Mathematical Software, 29 (2), 2003, 110–140.

[27] Y. Notay, P. S. Vassilevski, *Recursive Krylov-based multigrid cycles*, Numerical Linear Algebra with Applications, 15 (5), 2008, 473–487.

[28] Y. Saad, *Iterative methods for sparse linear systems*, 2nd edition, SIAM, 2003.

[29] B. Smith, P. Bjorstad, W. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, 1996.

[30] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, *MPI: The Complete Reference. Volume 1 - The MPI Core*, second edition, MIT Press, 1998.

[31] K. Stüben, *An Introduction to Algebraic Multigrid*, in A. Schüller, U. Trottenberg, C. Oosterlee, Multigrid, Academic Press, 2001.

[32] R. S. Tuminaro, C. Tong, *Parallel Smoothed Aggregation Multigrid: Aggregation Strategies on Massively Parallel Machines*, in J. Donnelley, editor, Proceedings of SuperComputing 2000, Dallas, 2000.

[33] P. Vaněk, J. Mandel, M. Brezina, *Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems*, Computing, 56 (3) 1996, 179–196.