MLD2P4
User's and Reference Guide

A guide for the Multi-Level Domain Decomposition Parallel Preconditioners Package based on PSBLAS

Pasqua D'Ambra
ICAR-CNR, Naples, Italy

Daniela di Serafino
Second University of Naples, Italy

Salvatore Filippone
University of Rome "Tor Vergata", Italy

Software version: 1.0
June 25, 2008

Abstract

MLD2P4 (Multi-Level Domain Decomposition Parallel Preconditioners Package based on PSBLAS) is a package of parallel algebraic multi-level preconditioners. It implements various versions of one-level additive and of multi-level additive and hybrid Schwarz algorithms. In the multi-level case, a purely algebraic approach is applied to generate coarse-level corrections, so that no geometric background is needed concerning the matrix to be preconditioned. The matrix is required to be square, real or complex, with a symmetric sparsity pattern

MLD2P4 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. MLD2P4 enables the user to easily specify different aspects of a generic algebraic multilevel Schwarz preconditioner, thus allowing to search for the "best" preconditioner for the problem at hand. The package has been designed employing object-oriented techniques, using Fortran 95 and MPI, with interfaces to additional external libraries such as UMFPACK, SuperLU and SuperLU_Dist, that can be exploited in building multi-level preconditioners.

This guide provides a brief description of the functionalities and the user interface of MLD2P4.

Contents

1 General Overview

The MULTI-LEVEL DOMAIN DECOMPOSITION PARALLEL PRECONDITIONERS PACKAGE BASED ON PSBLAS (MLD2P4) provides *multi-level Schwarz preconditioners* [14], to be used in the iterative solutions of sparse linear systems:

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

where A is a square, real or complex, sparse matrix with a symmetric sparsity pattern. These preconditioners have the following general features:

- both *additive and hybrid multilevel* variants are implemented, i.e. variants that are additive among the levels and inside each level, and variants that are multiplicative among the levels and additive inside each level; the basic Additive Schwarz (AS) preconditioners are obtained by considering only one level;
- a *purely algebraic* approach is used to generate a sequence of coarse-level corrections to a basic AS preconditioner, without explicitly using any information on the geometry of the original problem (e.g. the discretization of a PDE). The *smoothed aggregation* technique is applied as algebraic coarsening strategy [1, 18].

The package is written in *Fortran 95*, following an *object-oriented approach* through the exploitation of features such as abstract data type creation, functional overloading and dynamic memory management, while providing a smooth path towards the integration in legacy application codes. **NON MI PIACE QUESTO PERIODO, E' TROPPO LUNGO. RIUSCITE A SCRIVERLO MEGLIO?** The parallel implementation is based on a Single Program Multiple Data (SPMD) paradigm for distributed-memory architectures. Single and double precision implementations of MLD2P4 are available for both the real and the complex case, that can be used through a single interface.

MLD2P4 has been designed to implement scalable and easy-to-use multilevel preconditioners in the context of the *PSBLAS (Parallel Sparse BLAS) computational framework* [12]. PSBLAS is a library originally developed to address the parallel implementation of iterative solvers for sparse linear system, by providing basic linear algebra operators and data management facilities for distributed sparse matrices; it also includes parallel Krylov solvers, built on the top of the basic PSBLAS kernels. The preconditioners available in MLD2P4 can be used with these Krylov solvers. 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 MLD2P4 has led to some revisions and extentions of the PSBLAS kernels, leading to the recent PSBLAS 2.0 version [11]. The inter-process comunication required by MLD2P4 is encapsulated into the PSBLAS routines, except few cases where MPI [15] is explicitly called. Therefore, MLD2P4 can be run on any parallel machine where PSBLAS and MPI implementations are available.

MLD2P4 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 and easy-to-use 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 non-expert users to easily build any preconditioner available in MLD2P4 and to apply it within a PSBLAS Krylov solver. On the other hand, the routines of the middle and lower layer can be used and extended by expert users to build new versions of multi-level Schwarz preconditioners. We provide here a description of the upper-layer routines, but not of the medium-layer ones.

This guide is organized as follows: **ORGANIZZAZIONE DELLA GUIDA**

2 Notational Conventions

- caratteri tipografici usati nella guida (vedi guida ML recente e guida Aztec)
- convenzioni sui nomi di routine (differenza nei nomi tra high-level e medium-level), strutture dati, moduli, costanti, etc. (vedi guida psblas)
- versione reale e complessa, singola e doppia precisione

3 Configuring and Building MLD2P4

- uso di GNU autoconf e automake
- software di base necessario (MPI, BLACS, BLAS, PSBLAS, UMFPACK ? - specificare versioni)
- software opzionale (SuperLU, SuperLUdist - specificare versioni e opzioni di configure)
- sistemi operativi e compilatori su cui MLD2P4 e' stato costruito con successo
- sono previste opzioni di configurazione per il debugging o per il profiling?
- albero delle directory generato al momento dell'installazione

4 Getting Started

We describe the basics for building and applying MLD2P4 one-level and multi-level Schwarz preconditioners with the Krylov solvers included in PSBLAS [11]. The following steps are required:

1. *Declare the preconditioner data structure.* It is a derived data type, `mld_`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 MLD2P4 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 `mld_precinit`, which also sets defaults for each preconditioner type selected by the user. The defaults associated to each preconditioner type are listed in Table 1, where the strings used by `mld_precinit` to identify the preconditioner types are also given.

3. *Modify the selected preconditioner type, by properly setting preconditioner parameters.* This is performed by the routine `mld_precset`. This routine must be called only if the user wants to modify the default values of the parameters associated to the selected preconditioner type, to obtain a variant of the preconditioner. Examples of use of `mld_precset` is 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 2-5.

4. *Build the preconditioner for a given matrix.* This is performed by the routine `mld_precbld`.

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

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

A detailed description of the above routines is given in Section 5.

Note that the Fortran 95 module `mld_prec_mod` must be used in the program calling the MLD2P4 routines. Furthermore, to apply MLD2P4 with the Krylov solvers from PSBLAS, the module `psb_krylov_mod` must be used too. **DOBBIAMO SPECIFICARE QUALCHE ALTRO MODULO, AD ESEMPIO psb_base_mod?** Examples showing the basic use of MLD2P4 are reported in Section 4.1.

Remark. The coarsest-level solver used by the default two-level preconditioner has been chosen by taking into account that, on parallel machines, it often leads to the smallest execution time when applied to linear systems coming from finite-difference discretizations of basic elliptic PDE problems, considered as standard tests for multi-level Schwarz preconditioners [3, 4]. However, this solver does not necessarily to the smallest number of iterations of the preconditioned Krylov method, which is usually obtained by applying a direct solver, e.g. based on the LU factorization, on a matrix replicated at the coarsest level (see Section 5 for coarsest-level solvers available in MLD2P4).

Type	*String*	*Default preconditioner*
No preconditioner	`'NOPREC'`	(Considered only to use the PSBLAS Krylov solvers with no preconditioner.)
Diagonal	`'DIAG'`	—
Block Jacobi	`'BJAC'`	Block Jacobi with ILU(0) on the local blocks.
Additive Schwarz	`'AS'`	Restricted Additive Schwarz (RAS), with overlap 1 and ILU(0) on the local blocks.
Multilevel	`'ML'`	Multi-level hybrid preconditioner (additive on the same level and multiplicative through the levels), with post-smoothing only. Number of levels: 2; post-smoother: RAS with overlap 1 and with ILU(0) on the local blocks; coarsest matrix: distributed among the processors; (approximate) coarse-level solver: 4 sweeps of the block-Jacobi solver, with the UMFPACK LU factorization on the blocks (double precision versions) or **XXXXXXXXX** (single precision versions)

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

4.1 Examples

The code reported in Figure 1 shows how to set and apply the default multi-level preconditioner available in the real double precision version of MLD2P4 (see Table 1). This preconditioner is chosen by simply specifying `'ML'` as second argument of `mld_precinit` (a call to `mld_precset` is not needed) and is applied with the BiCGSTAB solver provided by PSBLAS. The setup and application of the default multi-level preconditioners for the real single precision and the complex, single and double precision, versions are obtained with straightforward modifications of the example.

The part of the code concerning the reading and assembling of the sparse matrix and the right-hand side vector, performed through the PSBLAS routines for sparse matrix and vector management, is not reported here for brevity; the statements concerning the deallocation of the PSBLAS data structure are neglected too. The complete code can be found in the example program file `example_ml.f90` in the directory **XXXXXX (COMPLETARE. DIRE CHE I FILE IN REALTA' SONO DUE, UNO CON LA GENERAZIONE DELLA MATRICE ED UNO CON LA LETTURA).** Note that the modules `psb_base_mod` and `psb_util_mod` at the beginning of the code are required by PSBLAS. **O psb_base_mod** E' RICHIESTO ANCHE DA MLD2P4?) For details on the use of the PSBLAS routines, see the PSBLAS User's Guide [11].

LE FIGURE SONO DECENTRATE, NONOSTANTE IL CENTER. CI VUOLE UNA MINIPAGE?

Different versions of 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 three-level hybrid Schwarz preconditioner, which uses block Jacobi with ILU(0) on the local blocks as post-smoother, a coarsest matrix replicated on the processors, and the LU factorization from UMFPACK [8], version 4.4, as coarse-level solver. The number of levels is specified by using `mld_precinit`; the other preconditioner parameters are set by calling `mld_precset`. Note that the type of multilevel framework (i.e. multiplicative among the levels with post-smoothing only) is not specified since it is the default set by `mld_precinit`. Figure 3 shows how to set a three-level additive Schwarz preconditioner, which applies RAS, with overlap 1 and ILU(0) on the blocks, as pre- and post-smoother, and five block-Jacobi sweeps, with the UMFPACK LU factorization on the blocks, as distributed coarsest-level solver. Again, `mld_precset` is used only to set non-default values of the parameters (see Tables 2-5). In both cases, the construction and the application of the preconditioner are carried out as for the default multi-level preconditioner. The code fragments shown in in Figures 2-3 are included in the example program file `example_ml.f90`. **LO STESSO PROGRAMMA CONTIENE I TRE ESEMPI, CON UN SWITCH TRA L'UNO E L'ALTRO O FACCIAMO 3 PROGRAMMI DISTINTI? NON RICORDO CHE COSA ABBIAMO DECISO. PASQUA: ABBIAMO DETTO CHE ERA PREFERIBILE UN UNICO PROGRAMMA CON SWITCH.**

Finally, Figure 4 shows the setup of a one-level additive Schwarz preconditioner, i.e. RAS with overlap 2. The corresponding code, including also the application of the preconditioner is in the example program file `example_1lev.f90`.

Remark. Any PSBLAS-based program using the basic preconditioners implemented in PSBLAS 2.0, i.e. the diagonal and block-Jacobi ones, can use the diagonal and block-Jacobi preconditioners implemented in MLD2P4 without any change in the code. The PSBLAS-based program must be only recompiled and linked to the MLD2P4 library.

```
  use psb_base_mod
  use psb_util_mod
  use mld_prec_mod
  use psb_krylov_mod
... ...
!
! sparse matrix
  type(psb_dspmat_type) :: A
! sparse matrix descriptor
  type(psb_desc_type)   :: desc_A
! preconditioner
  type(mld_dprec_type)  :: P
... ...
!
! initialize the parallel environment
  call psb_init(ictxt)
  call psb_info(ictxt,iam,np)
... ...
!
! read and assemble the matrix A and the right-hand
! side b using PSBLAS routines for sparse matrix /
! vector management
... ...
!
! initialize the default multi-level preconditioner,
! i.e. two-level hybrid Schwarz, using RAS (with
! overlap 1 and ILU(0) on the blocks) as post-smoother
! and 4 block-Jacobi sweeps (with UMFPACK LU on the
! blocks) as distributed coarse-level solver
  call mld_precinit(P,'ML',info)
!
! build the preconditioner
  call psb_precbld(A,P,desc_A,info)
!
! set the solver parameters and the initial guess
  ... ...
!
! solve Ax=b with preconditioned BiCGSTAB
  call psb_krylov('BICGSTAB',A,P,b,x,tol,desc_A,info)
  ... ...
!
! deallocate the preconditioner
  call mld_precfree(P,info)
!
! deallocate other data structures
  ... ...
!
! exit the parallel environment
  call psb_exit(ictxt)
  stop
```

Figure 1: Setup and application of the default multi-level Schwarz preconditioner.

```
... ...
! set a three-level hybrid Schwarz preconditioner,
! which uses block Jacobi (with ILU(0) on the blocks)
! as post-smoother, a coarsest matrix replicated on the
! processors, and the LU factorization from UMFPACK
! as coarse-level solver
  call mld_precinit(P,'ML',info,nlev=3)
  call_mld_precset(P,mld_smoother_type_,'BJAC',info)
  call mld_precset(P,mld_coarse_mat,'REPL')
  call mld_precset(P,mld_coarse_solve,'UMF')
... ...
```

Figure 2: Setup of a hybrid three-level Schwarz preconditioner.

```
... ...
! set a three-level additive Schwarz preconditioner,
! which uses RAS (with overlap 1 and ILU(0) on the blocks)
! as pre- and post-smoother, and 5 block-Jacobi sweeps
! (with UMFPACK LU on the blocks) as distributed
! coarsest-level solver
  call mld_precinit(P,'ML',info,nlev=3)
  call mld_precset(P,mld_ml_type_,'ADD',info)
  call_mld_precset(P,mld_smoother_pos_,'TWOSIDE',info)
  call mld_precset(P,mld_coarse_sweeps_,5)
... ...
```

Figure 3: Setup of an additive three-level Schwarz preconditioner.

```
... ...
! set RAS with overlap 2 and ILU(0) on the local blocks
  call mld_precinit(P,'AS',info)
  call mld_precset(P,mld_sub_ovr_,2,info)
... ...
```

Figure 4: Setup of a one-level Schwarz preconditioner.

5 User Interface

The basic user interface of MLD2P4 consists of six routines. The four routines `mld_precinit`, `mld_precset`, `mld_precbld` and `mld_precaply` encapsulate all the functionalities for the setup and application of any one-level and multi-level preconditioner implemented in the package. The routine `mld_precfree` deallocates the preconditioner data structure, while `mld_precdescr` prints a description of the preconditioner setup by the user.

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

- the sparse matrix data structure, containing the matrix to be preconditioned, must be of type `mld_`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 `mld_`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 $w = M^{-1}v$ must be of type *type*`(`*kind_parameter*`)`, with *type* = `real`, `complex` and *kind_parameter* = `kind(1.)`, `kind(1.d0)`, according to the sparse matrix and preconditioner data structure; note that the PSBLAS module provides the constants `psb_spk_` = `kind(1.)` and `psb_dpk_` = `kind(1.d0)`;
- real parameters defining the preconditioner must be declared according to the precision of the previous data structures (see Section 5.2).

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

5.1 Subroutine mld_precinit

```
mld_precinit(p,ptype,info)
mld_precinit(p,ptype,info,nlev)
```

This routine allocates and initializes the preconditioner data structure, according to the preconditioner type chosen by the user.

Arguments

`p`	`type(mld_`x`prec_type), intent(inout).` The preconditioner data structure. Note that x must be chosen according to the real/complex, single/double precision version of MLD2P4 under use.
`ptype`	`character(len=*), intent(in).` The type of preconditioner. Its values are specified in Table 1.
`info`	`integer, intent(out).` Error code. See Section 6 for details.
`nlev`	`integer, optional, intent(in).` The number of levels of the multilevel preconditioner. If `nlev` is not present and `ptype`='ML'/'ml', then `nlev`=2 is assumed. Otherwise, `nlev` is ignored.

5.2 Subroutine mld_precset

`mld_precset(p,what,val,info)`

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

Arguments

`p`	`type(mld_`x`prec_type), intent(inout).` The preconditioner data structure. Note that x must be chosen according to the real/complex, single/double precision version of MLD2P4 under use.
`what`	`integer, intent(in).` The number identifying the parameter to be set. A mnemonic constant has been associated to each of these numbers, as reported in Tables 2-5.
`val`	`integer` *or* `character(len=*)` *or* `real(kind(1.))` *or* `real(kind(1.d0))`, `intent(in).` The value of the parameter to be set. The list of allowed values and the corresponding data types is given in Table **??**.
`info`	`integer, intent(out).` Error code. See Section 6 for details.

A variety of (one-level and multi-level) preconditioner can be obtained by a suitable setting of the preconditioner parameters. These parameters can be logically divided into four groups, i.e. parameters defining

1. the type of multi-level preconditioner;
2. the one-level preconditioner to be used as smoother;
3. the aggregation algorithm;
4. the coarse-space correction at the coarsest level.

A list of the parameters that can be set, along with their allowed and default values, is given in Tables 2-5. **CORREGGERE LA ROUTINE E LA DOC INTERNA - ilev NON E' PIU' ACCESSIBILE ALL'UTENTE.**

what	*data type*	val	*default*	*comments*
`mld_ml_type_`	`character(len=*)`	'ADD' 'MULT'	'MULT'	basic multi-level framework: additive or multiplicative among the levels always additive inside a level)
`mld_smoother_type_`	`character(len=*)`	'DIAG' 'BJAC' 'AS'	'AS'	basic one-level preconditioner (i.e. smoother) of the multi-level preconditioner **CAMBIARE NOME COSTANTE NEL SW, ORA E' mld_prec_type. INIBIRE no_prec NELL'AMBITO DEL MULTI-LEVEL.**
`mld_smoother_pos_`	`character(len=*)`	'PRE' 'POST' 'TWOSIDE'	'POST'	"position" of the smoother: pre-smoother, post-smoother, pre-/post-smoother

Table 2: Parameters defining the type of multi-level preconditioner.

what	*data type*	val	*default*	*comments*
`mld_sub_ovr`	`integer`	any number ≥ 0	1	**CAMBIARE NOME PARAMETRO NEL SW** number of overlap in the basic Schwarz preconditioner
`mld_sub_restr_`	`character(len=*)`	'HALO' 'NONE'	'HALO'	type of restriction operator used in basic Schwarz preconditioner: 'HALO' for taking into account contributions from the overlap
`mld_sub_prol_`	`character(len=*)`	'SUM' 'NONE'	'NONE'	type of prolongator operator used in basic Schwarz preconditioner: 'NONE' for neglecting contributions from the overlap
`mld_sub_solve_`	`character(len=*)`	'ILU' 'MILU' 'ILUT' 'UMF' 'SLU'	'UMF'	available local solver: 'ILU' for incomplete LU, 'MILU' for modified incomplete LU, 'ILUT' for incomplete LU with threshold, 'UMF' for complete LU using UMFPACK [8] version 4.4, 'SLU' for complete LU using SuperLU [9], version 3.0
`mld_sub_fillin_`	`integer`	any number ≥ 0	0	**CAMBIARE NOME PARAMETRO NEL SW** fill-in level for 'ILU', 'MILU' and 'ILUT' of local blocks
`mld_sub_thresh_`	`real`	any number $\geq 0.$	0.	drop tolerance for 'ILUT' **NELLA DOCUMENTAZIONE INTERNA DELLA ROUTINE DI FATTORIZZAZIONE C'E' INTERO, CAMBIARE!**
`mld_sub_ren_`	`character(len=*)`	**MANCA COSTANTE STRINGA ASSOCIATA**		reordering algorithm for the local blocks

Table 3: Parameters defining the basic one-level preconditioner (smoother).

what	*data type*	val	*default*	*comments*
mld_aggr_alg_	character(len=*)	'DEC'	'DEC'	define the aggregation scheme. Now, only decoupled aggregation is available
mld_aggr_kind_	character(len=*)	'SMOOTH', 'RAW'	'SMOOTH'	define the type of aggregation technique (smoothed or nonsmoothed).
mld_aggr_thresh_	real	any number $\in [0,1]$	0.	dropping threshold in aggregation
mld_aggr_eig_	character(len=*)	**MANCA STRINGA CORRISPONDENTE a mld_max_norm**	'ANORM'???	define the algorithm to evaluate the maximum eigenvalue of $D^{-1}A$ for smoothed aggregation. Now, only the A-norm of the matrix is available

Table 4: Parameters defining the aggregation algorithm.

`what`	*data type*	`val`	*default*	*comments*
`mld_coarse_mat_`	`character(len=*)`	'DISTR', 'REPL'	'DISTR'	Coarse matrix: distributed or replicated
`mld_coarse_solve_`	`character(len=*)`	'BJAC' 'UMF' 'SLUDIST'	'BJAC'	**VEDI OSSERVAZIONI EMAIL 15-16/06/08** available solver for coarse system. Only 'BJAC' and 'SLUDIST' can be used for distributed coarse matrix. 'BJAC' corresponds to some sweeps of a block-Jacobi solver, while 'SLUDIST' corresponds to the use of the external package SuperLU_Dist [13], version 2.0, for distributed sparse factorization and solve.
`mld_coarse_subsolve_`	`character(len=*)`	'ILU' 'MILU' 'ILUT' 'UMF' 'SLU'	'UMF'	**VEDI OSSERVAZIONI EMAIL 15-16/06/08** available solver for diagonal local blocks of the coarse matrix, when 'BJAC' is used as coarse solver
`mld_coarse_sweeps_`	`integer`	any number > 0	4	number of Block-Jacobi sweeps when 'BJAC' is used as coarse solver
`mld_coarse_fillin_`	`integer`	any number ≥ 0	0	fill-in level in incomplete factorization of local diagonal blocks of the coarse matrix, when 'BJAC' is used as coarse solver and 'ILU' or 'MILU' is used as local solver **MODIFICA NOME PARAM. NEL SW**
`mld_coarse_thresh_`	`real`	any number $\geq 0.$	0.	drop tolerance in incomplete factorization of local diagonal blocks of the coarse matrix, when 'BJAC' is used as coarse solver and 'ILUT' is used as local solver

Table 5: Parameters defining the coarse-space correction at the coarsest level.

5.3 Subroutine mld_precbld

`mld_precbld(a,desc_a,p,info)`

This routine builds the preconditioner according to the requirements made by the user through the routines `mld_precinit` and `mld_precset`.

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 MLD2P4 under use. See the PSBLAS User's Guide for details [11].

`desc_a` `type(psb_desc_type), intent(in).`
The communication descriptor of a. See the PSBLAS User's Guide for details [11].

`p` `type(mld_`x`prec_type), intent(inout).`
The preconditioner data structure. Note that x must be chosen according to the real/complex, single/double precision version of MLD2P4 under use.

`info` `integer, intent(out).`
Error code. See Section 6 for details.

5.4 Subroutine mld_precaply

```
mld_precaply(p,x,y,desc_a,info)
mld_precaply(p,x,y,desc_a,info,trans,work)
```

This routine computes $y = op(M^{-1})\,x$, where M is a previously built preconditioner, stored in the `p` data structure, and op denotes the preconditioner itself or its transpose, according to the value of `trans`. Note that, when MLD2P4 is used with a Krylov solver from PSBLAS, `mld_precaply` is called within the PSBLAS routine `mld_krylov` and hence is completely transparent to the user.

Arguments

`p`	`type(mld_`*x*`prec_type), intent(inout)`. The preconditioner data structure, containing the local part of M. Note that x must be chosen according to the real/complex, single/double precision version of MLD2P4 under use.
`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 MLD2P4 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 MLD2P4 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. See Section 6 for details.
`trans`	`character(len=1), optional, intent(in)`. If `trans` = `'N','n'` then $op(M^{-1}) = M^{-1}$; if `trans` = `'T','t'` then $op(M^{-1}) = M^{-T}$ (transpose of M^{-1}).
`work`	*type*`(`*kind_parameter*`), dimension(:), optional, target`. Workspace. Its size must 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 MLD2P4 under use.

5.5 Subroutine mld_precfree

```
mld_precfree(p,info)
```

This routine deallocates the preconditioner data structure.

Arguments

`p`	`type(mld_`*x*`prec_type), intent(inout)`. The preconditioner data structure. Note that x must be chosen according to the real/complex, single/double precision version of MLD2P4 under use.
`info`	`integer, intent(out)`. Error code. See Section 6 for details.

5.6 Subroutine mld_precdescr

`mld_precdescr(p,iout)`

This routine prints a description of the preconditioner to the standard output or to a file. **FARE UNA SOLA ROUTINE, COL PARAMETRO IOUT OPZIONALE.**

Arguments

`p`	`type(mld_`x`prec_type), intent(in)`. The preconditioner data structure. Note that x must be chosen according to the real/complex, single/double precision version of MLD2P4 under use.
`iout`	`integer, intent(in)`. The id of the file where the preconditioner description will be printed. If `iout` is missing, the description is printed on the standard output.

6 Error Handling

Error handling - Breve descrizione con rinvio alla guida di PSBLAS

A License

The MLD2P4 is freely distributable under the following copyright terms:

B Bibliography

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

[2] 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.

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

[4] 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.

[5] 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.

[6] X. C. Cai, O. B. Widlund, *Domain Decomposition Algorithms for Indefinite Elliptic Problems*, SIAM Journal on Scientific and Statistical Computing, 13, 1, 1992, 243–258.

[7] T. Chan and T. Mathew, *Domain Decomposition Algorithms*, in A. Iserles, editor, Acta Numerica 1994, 61–143. Cambridge University Press.

[8] 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/`)

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

[10] E. Efstathiou, J. G. Gander, *Why Restricted Additive Schwarz Converges Faster than Additive Schwarz*, BIT Numerical Mathematics, 43, 2003, 945–959.

[11] S. Filippone, A. Buttari, *PSBLAS-2.1 User's Guide. A Reference Guide for the Parallel Sparse BLAS Library*, xxxxx.

[12] 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.

[13] 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.

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

[15] 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.

[16] K. Stüben, *Algebraic Multigrid (AMG): an Introduction with Applications*, in A. Schüller, U. Trottenberg, C. Oosterlee, editors, Multigrid, Academic Press, 2000.

[17] 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.

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