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 18, 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* [**?**], 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, i.e. multiplicative among the levels and additive inside a level, are implemented; the basic additive Schwarz 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 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 [**?**, **?**].

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. 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. **SALVATORE, funziona tutto?**

MLD2P4 has been designed to implement scalable and easy-to-use multilevel preconditioners in the context of the *PSBLAS (Parallel Sparse BLAS) computational framework* [10]. 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 [**?**]. The inter-process comunication required by MLD2P4 is encapsulated into the PSBLAS routines, except few cases where MPI [17] 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 tra high-level e medium-level), strutture dati,
moduli, costanti, etc. (vedi guida psblas)
- versione reale e complessa

3 Code Distribution

The MLD2P4 is freely distributable under the following copyright terms:

4 Configuring and Building MLD2P4

- uso di GNU autoconf e automake
- software di base necessario (MPI, BLACS, BLAS, PSBLAS - specificare versioni)
- software opzionale (UMFPACK, 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

5 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 []. 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 a default preconditioner for each preconditioner type selected by the user. The default preconditioner associated to each preconditioner type is listed in Table 2; the string used by `mld_precinit` to identify each preconditioner type is also given.

3. *Choose a specific preconditioner within the selected preconditioner type, by setting the 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 default preconditioner. An example of use of `mld_precset` is given in Section 5.1, Figure 2; a complete list of all the preconditioner parameters and their allowed and default values is provided in Section 6.

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. *Deallocate 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 6.

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.

Examples showing the basic use of MLD2P4 are reported in Section 5.1.

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: block-Jacobi preconditioner with ILU(0) on the local blocks; coarsest matrix: distributed among the processors; corase-level solver: 4 sweeps of the block-Jacobi solver, with the UMFPACK LU factorization on the blocks.

Table 1: Preconditioner types and default choices.

5.1 Examples

The code reported in Figure 1 shows how to set and apply the MLD2P4 default multi-level preconditioner, i.e. the two-level hybrid post-smoothed Schwarz preconditioner, having block-Jacobi with ILU(0) on the blocks as basic preconditioner, a coarse matrix distributed among the processors, and four block-Jacobi sweeps, with the UMFPACK sparse LU factorization on the blocks, as approximate coarse-level solver. The choice of this preconditioner is made by simply specifying `'ML'` as second argument of `mld_precinit` (a call to `mld_precset` is not needed). The preconditioner is applied with the BiCGSTAB solver provided by PSBLAS.

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_default.f90` in the directory **XXXXXX (SPECIFICARE).** Note that the modules `psb_base_mod` and `psb_util_mod` at the beginning of the code are required by PSBLAS. For details on the use of the PSBLAS routines, see the PSBLAS User's Guide [].

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 using RAS with overlap 1 as post-smoother, a coarsest matrix replicated on the processors and the LU factorization from UMFPACK 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, which corresponds to the hybrid multilevel preconditioner); the type of one-level AS preconditioner used as smoother (i.e. RAS) and its "position" (i.e. pre-smoother) are not specified since they are chosen by default when `mld_precinit` is called. The construction and the application of the preconditioner are carried out as for the default multi-level preconditioner.

As a further example, we report in Figure 3 the code concerning the setup of a three-level additive multi-level preconditioner, using ILU(0) as pre- and post-smoother, a distributed coarsest matrix and five block-Jacobi sweeps as coarsest-level solver, with ILU(0) on the local blocks. Again, `mld_precset` is used only to set the values of the parameters that are not default values. For a detailed description of the parameters associated to a preconditioner type, including their allowed and default values, the user is referred to **SPECIFICARE.**

An example program including the code fragments shown in in Figures 2 and 3 is in `XXX/.../example_3lev.f90`. **COMPLETARE. Fare un programma solo per i due esempi, in cui uno e' commentato e l'altro no.** One more example program, showing the setup and application of a one-level additive Schwarz preconditioner can be found in `XXX/.../example_1lev.f90`. **COMPLETARE**.

```
  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
! (two-level hybrid Schwarz, with ILU(0) as post-smoother
! and 4 Block-Jacobi sweeps, with ILU(0) on the blocks,
! as distributed coarsest-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 multilevel Schwarz preconditioner.

```
... ...
! setup a three-level hybrid Schwarz preconditioner,
! using RAS with overlap 1 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_smooth_type_,'AS',info)
  call mld_precset(P,mld_n_ovr_,1,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.

```
... ...
! setup a three-level additive Schwarz preconditioner,
! using ILU(0) as pre- and post-smoother, five block-Jacobi
! sweeps as distributed coarsest-level solver, with ILU(0)
! on the local blocks
  call mld_precinit(P,'ML',info,nlev=3)
  call mld_precset(P,mld_ml_type_,'ADD',info)
  call_mld_precset(P,mld_smooth_pos_,'TWOSIDE',info)
  call mld_precset(P,mld_n_ovr_,1,info)
  call mld_precset(P,mld_coarse_sweeps,5)
  call mld_precset(P,mld_coarse_subsolve,'UMF')
... ...
```

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

Note. 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-base program must e only recompiled and linked to the MLD2P4 library.

6 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 available independently of the real or complex case and of the single or double precision. However, the appropriate preconditioner data type to be used with each version of the package must be explicitly chosen by the user, i.e. a preconditioner data structure of type `mld_`x`prec_type` must be declared, where $x =$ `s` for real single precision, $x =$ `d` for real double precision, $x =$ `c` for complex single precision, $x =$ `z` for complex double precision. A few parameters defining the preconditioner may be real single or double precision, depending on the package version (see Section 6.2).

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

6.1 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 2.
`info`	`integer, intent(out).` Error code. See Section 8 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.

6.2 Subroutine mld_precset

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

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 Table **??**.
`val`	`integer` *or* `character(len=*)` *or* `real(1.0)` *or* `real(1.0d0)`, `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 8 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 the appropriate levels (see Table **??**).

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 basic structure of the 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, and the levels at which their are appropriate is given in Table **??**. Note that the routines allows to set different features of the preconditioner at each level through the use of `ilev`. This should be done by users with experience in the field of multi-level preconditioners. Non-expert users are recommended to call `mld_precset` without specifying `ilev`.

QUESTA TABELLA VA RUOTATA DI 90 GRADI E SUDDIVISA IN DUE TABELLE, TOGLIENDO SMALL

`what`	*data type*	`val`	`ilev`	*comments*
basic structure of the multi-level preconditioner				
`mld_ml_type_`	`character(len=*)`	'ADD', 'MULT'	2,...,`nlev`	basic multi-level framework: additive or multiplicative among the levels (always additive inside a level); when `ilev` is present, it refers only to the combination of levels `ilev`-1 and `ilev`.
`mld_baseprec_type_`	`character(len=*)`	'DIAG', 'BJAC', 'AS'	1,...,`nlev`-1	basic one-level preconditioner (i.e. smoother) of the multi-level preconditioner **CAMBIARE NOME COSTANTE; ora e' mld_prec_type, ma questo puo' generare confusione!**
`mld_smooth_pos_`	`character(len=*)`	'PRE', 'POST', 'BOTH'	2,...,`nlev`	"position" of the smoother: pre-smoother, post-smoother, pre-/post-smoother **per l'utente NON HA SENSO settarlo ai livelli 2,..., nlev; l'utente deve specificare un livello tra 1 e nlev-1 e la precset deve shiftare il livello tenendo conto della struttura del tipo di dato precondizionatore**
basic one-level preconditioner (smoother)				
`mld_n_ovr`				
`mld_sub_restr_`				
`mld_sub_prol_`				
`mld_sub_solve_`				
`mld_sub_fillin_`				**MODIFICA: fill_in $--->$ fillin**
`mld_sub_thresh_`				**AGGIUNGERE THRESHOLD ILU(t)**
`mld_sub_ren_`				**MANCA COSTANTE STRINGA ASSOCIATA**
aggregation algorithm				
`mld_aggr_alg_`				
`mld_aggr_kind_`				
`mld_aggr_thresh_`				
`mld_aggr_eig_`				**NON E' DEFINITA LA STRINGA CORRISPONDENTE a mld_max_norm**
coarse-space correction at the coarsest level				
`mld_coarse_mat_`				
`mld_coarse_solve_`				**VEDI OSSERVAZIONI EMAIL 15-16/06/08**
`mld_coarse_subsolve_`				**VEDI OSSERVAZIONI EMAIL 15-16/06/08**
`mld_coarse_sweeps_`				
`mld_coarse_fillin_`				**MODIFICA: fill_in $--->$ fillin**
`mld_coarse_thresh_`				**AGGIUNGERE THRESHOLD ILU(t)**

Table 2: Parameters defining the preconditioner.

```
==========================================

mld_precfree(p,info)

Arguments:
    p       -  type(mld_dprec_type), input/output.
               The preconditioner data structure to be deallocated.
    info    -  integer, output.
               Error code.
```

Figure 4: API of the routine for preconditioner deallocation.

A twin routine for deallocation of the preconditioner data structure is the `mld_precfree` routine, whose API is reported in Fig. 4. As mentioned in Section **??**, a multi-level preconditioner is a combination of coarse-level corrections and one-level preconditioner (or smoothers). Different combinations of these components together with different type of one-level preconditioner as well as different algorithms to build and apply coarse-level corrections allow to the user of defining different multi-level preconditioners. The user of MLD2P4 may specify the type of multi-level framework (additive or multiplicative), details on the aggregation algorithm, details on the type and the way for applying the one-level preconditioner (as pre-smoother, post-smoother or both), the coarsest matrix storage (distributed or replicated), the type of the solver to be employed at the coarsest level and related details, by setting some parameters through the routine `mld_precset` (see Section 6.2.1). The API of this routine is reported in Fig. 5. Finally, to build a preconditioner, according to the requirements made trough the routines `mld_precinit` and `mld_precset`, a user of MLD2P4 have to call the `prec_build` routine, whose API is reported in Figure 6.

6.2.1 List of the preconditioner parameters

In the following we report the list of possible parameters to be set through the `mld_precset` routine, in order to choose the type of multi-level preconditioner. The parameters are classified depending on their scope. Note that for character data both uppercase and lowercase strings are allowed.

In order to build a coarse matrix from a fine one, this version of MLD2P4 implements the smoothed aggregation algorithm described in Section **??**. However, since for nonsymmetric problems the application of a correct smoothed procedure is yet an

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

 Arguments:
    p       -  type(mld_dprec_type), input/output.
               The preconditioner data structure.
    what    -  integer, input.
               The number identifying the parameter to be set.
               A mnemonic constant has been associated to each of these
               numbers.
    val     -  integer/character, input.
               The value of the parameter to be set.
    info    -  integer, output.
               Error code.
    ilev    -  integer, optional, input.
               For the multilevel preconditioner, the level at which the
               preconditioner parameter has to be set.
               If nlev is not present, the parameter identified by 'what'
               is set at all the appropriate levels.
```

Figure 5: API of the routine for preconditioner setup.

open problem [**?**], the user may also choose to apply a nonsmoothed aggregation technique, where the prolongator operator from the coarse to fine-space vertices is the simple piecewice constant interpolation (the tentative prolongator) operator defined in Section **??**. The coarsening scheme takes into account possible anisotropic features of the problems, by using a threshold level to be used for dropping matrix coefficients during the process. The parallel implementation of the coarsening algorithm is based on a decoupled approach, where each process applies the coarsening scheme to its own local data. The uncoupled scheme can be applied to the matrix $A + A^T$, in the case of matrices with nonsymmetric sparsity pattern. In the Table 6.2.1 we list the parameters that the user can specify for the aggregation algorithm.

Some options are available for the system involving the coarsest matrix. Indeed, this matrix can be replicated or distributed among the processors. In the former case, various versions of incomplete LU (ILU) factorizations of the coarsest matrix are available in order to solve the coarsest system. In the current version of MLD2P4, the following factorizations are available [**?**]:

ILU(k): ILU factorization with fill-in level k;

MILU(k): modified ILU factorization with fill-in level k;

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

 Arguments:
   a       -  type(psb_dspmat_type).
              The sparse matrix structure containing the local part of the
              matrix to be preconditioned.
   desc_a  -  type(psb_desc_type), input.
              The communication descriptor of a.
   p       -  type(mld_dprec_type), input/output.
              The preconditioner data structure containing the local part
              of the preconditioner to be built.
   info    -  integer, output.
              Error code.
```

Figure 6: API of the routine for preconditioner building.

Parameter (`what`)	Allowed values (`val`)
`mld_aggr_alg_`	'DEC', 'SYMDEC' Define the aggregation scheme Now, only decoupled aggregation is available (if 'SYMDEC' is set, the symmetric part of the matrix is considered)
`mld_aggr_kind_`	'SMOOTH', 'RAW' Define the type of aggregation technique (smoothed or nonsmoothed).
`mld_aggr_thresh_`	Dropping threshold in aggregation. Default 0.0
`mld_aggr_eig_`	NON E' DEFINITA LA STRINGA CORRISPONDENTE a mldmaxnorm 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 3: Parameters for aggregation type.

ILU(k,t): ILU with threshold t and k additional entries in each row of the L and U factors with respect to the initial sparsity pattern.

Furthermore, interfaces to UMFPACK [?], version 4.4, and to SuperLU package [?], version 3.0, have been also available to deal with the coarsest system, when the coarsest matrix is replicated among the processors. On the other hand, to solve the coarsest-level system when the coarsest matrix is distributed, a block-Jacobi routine has been developed. It uses the different versions of ILU or the LU factorization on the coarse matrix diagonal blocks held by the processors. In the case of distributed coarsest matrix is also available an interface to SupeLU_dist [?], version 2.0, for distributed sparse factorization and solve. See the Table

6.2.1 for details.

Parameter (`what`)	Allowed values (`val`)
`mld_coarse_mat_`	'DISTR', 'REPL' Coarse Matrix: distributed or replicated
`mld_coarse_solve_`	'ILU', 'MILU', 'ILUT', 'SLU', 'UMF', SLUDIST', BJAC???? Available Coarse solver. Only SLUDIST e BJAC can be used when coarse matrix is distributed
`mld_coarse_BJAC_sweeps_`	(NON VA BENE mldcoarsesweeps) number of Block-Jacobi sweeps when BJ
`mld_coarse_fill_in_`	level of fill-in in MILU and ILU factorization E IL THRESHOLD PER ILUT?

Table 4: Parameters for coarsest matrix solver.

When a Schwarz algorithm is considered as smoother at a certain level or as one-level preconditioner, the user may set many parameters in order to choose the type of additive Schwarz version (AS,RAS,ASH), the number of overlaps as well as the local solver. All the parameters are reported in Table 6.2.1. Its worth

Parameter (`what`)	Allowed values (`val`)
`mld_n_ovr_`	Number of overlaps
`mld_sub_restr_`	'HALO', 'NONE'
`mld_sub_prol_`	'SUM', 'NONE'
`mld_sub_solve_`	'ILU', 'MILU', 'ILUT', 'SLU', 'UMF'
`mld_sub_ren_`	MANCANO LE STRINGHE
`mld_sub_fill_in_`	level of fill-in in local diagonal blocks, when ILU-type factorizations are used

Table 5: Parameters for Schwarz smoother/preconditioner type.

noting that, the classical AS method corresponds to the couple of values 'HALO' and 'SUM' of the argument `val`, for the values `mld_sub_restr_` and `mld_sub_prol_` of the argument `what`, respectively. While, the RAS method corresponds to the couple of values 'NONE' and 'SUM' and ASH method corresponds to the couple of values 'HALO' and 'NONE'.

6.3 Preconditioner Application

Once the preconditioner has been built, it may be applied at each iteration of a Krylov solver by calling the routine `mld_precaply` (CAMBIARE NOME ROUTINE NEL SOFTWARE EVITANDO L'UNDERSCORE), whose API is shown in Figure 7. This routine computes $y = op(M^{-1})\,x$, where M is the previously built preconditioner, stored in the `prec` data structure, and *op* denotes the matrix itself or its transpose, according to the value of `trans`.

Note that this routine is called within the PSBLAS-based Krylov solver available in the PSBLAS library (see the PSBLAS User's Guide for details), therefore, the use of this routine is generally transparent to the MLD2P4 user.

```
  mld_precaply(prec,x,y,desc_data,info,trans,work)

 Arguments:
    prec       -  type(mld_dprec_type), input.
                  The preconditioner data structure containing the local part
                  of the preconditioner to be applied.
    x          -  real(psb_dpk_), dimension(:), input.
                  The local part of the vector X in Y := op(M^(-1)) * X.
    y          -  real(psb_dpk_), dimension(:), output.
                  The local part of the vector Y in Y := op(M^(-1)) * X.
    desc_data  -  type(psb_desc_type), input.
                  The communication descriptor associated to the matrix to be
                  preconditioned.
    info       -  integer, output.
                  Error code.
    trans      -  character(len=1), optional.
                  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       -  real(psb_dpk_), dimension (:), optional, target.
                  Workspace. Its size must be at
                  least 4*psb_cd_get_local_cols(desc_data).
```

Figure 7: API of the routine for preconditioner application.

7 Advanced Use

- MLD2P4 software architecture
- preconditioner data structure (descrizione ”dettagliata”) + possibilita’ di settare singolarmente i vari livelli (possibilita’ accennata solamente nella precedente descrizione di precset)
- descrizione routine medium level (con introduzione sulle potenzialita’ di ampliamento (?), offerte da queto strato software)

8 Error Handling

Error handling - Breve descrizione con rinvio alla guida di PS-BLAS

9 List of Routines

Elenco (ordine alfabetico) di tutte le routine, con rinvio (ipertestuale e num. pag.) alla descrizione di ciascuna in qualche paragrafo precedente (una specie di indice analitico, che rimanda alle routine descritte precedentemente nei rispettivi paragrafi)

References

[1] Bella, G., Filippone, S., De Maio, A., Testa, M.: A Simulation Model for Forest Fires. In: Dongarra, J., Madsen, K., Wasniewski, J. (eds.): Proceedings of PARA 04 Workshop on State of the Art in Scientific Computing. Lecture Notes in Computer Science, 3732. Berlin: Springer, 2005

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

[3] P. D'Ambra, S. Filippone, D. Di Serafino On the Development of PSBLAS-based Parallel Two-level Schwarz Preconditioners Applied Numerical Mathematics, Elsevier Science, Volume 57, Issues 11-12, November-December 2007, Pages 1181-1196.

[4] A. Buttari, P. D'Ambra, D. di Serafino and 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, pp. 593–602, Lecture Notes in Computer Science, Springer, 2005.

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

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

[7] J. J. Dongarra and R. C. Whaley, *A User's Guide to the BLACS v. 1.1*, Lapack Working Note 94, Tech. Rep. UT-CS-95-281, University of Tennessee, March 1995 (updated May 1997).

[8] I. Duff, M. Marrone, G. Radicati and C. Vittoli, *Level 3 Basic Linear Algebra Subprograms for Sparse Matrices: a User Level Interface*, ACM Transactions on Mathematical Software, 23(3), pp. 379–401, 1997.

[9] I. Duff, M. Heroux and R. Pozo, *An Overview of the Sparse Basic Linear Algebra Subprograms: the New Standard from*

the BLAS Technical Forum, ACM Transactions on Mathematical Software, 28(2), pp. 239–267, 2002.

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

[11] S. Filippone, P. D'Ambra, M. Colajanni, *Using a Parallel Library of Sparse Linear Algebra in a Fluid Dynamics Applications Code on Linux Clusters*, in G. Joubert, A. Murli, F. Peters, M. Vanneschi, editors, Parallel Computing - Advances & Current Issues, pp. 441–448, Imperial College Press, 2002.

[12] Karypis, G. and Kumar, V., *METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System.* Minneapolis, MN 55455: University of Minnesota, Department of Computer Science, 1995. Internet Address: `http://www.cs.umn.edu/~karypis`.

[13] Lawson, C., Hanson, R., Kincaid, D. and Krogh, F., Basic Linear Algebra Subprograms for Fortran usage, ACM Trans. Math. Softw. vol. 5, 38–329, 1979.

[14] Machiels, L. and Deville, M. *Fortran 90: An entry to object-oriented programming for the solution of partial differential equations.* ACM Trans. Math. Softw. vol. 23, 32–49.

[15] Metcalf, M., Reid, J. and Cohen, M. *Fortran 95/2003 explained.* Oxford University Press, 2004.

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

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

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

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