MLD2P4-1.0 User's guide

A reference guide for the MultiLevel Domain Decomposition Parallel Preconditioners Package based on Parallel Sparse BLAS

by Salvatore Filippone
Alfredo Buttari
University of Rome "Tor Vergata"

Daniela di Serafino
Second University of Naples

Pasqua D'Ambra
ICAR-CNR, Naples

June 13, 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 **Non consideriamo anche il caso non simmetrico con** $(A + A^T)/2$**?**.

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.

Contents

1 General Overview

The *Multi-Level Domain Decomposition Parallel Preconditioners Package based on PSBLAS (MLD2P4)* provides various versions of multi-level Schwarz preconditioners [**?**], to be used in the iterative solutions of sparse linear systems $Ax = b$, where A is a square, real or complex, sparse matrix with a symmetric sparsity pattern. **Ma non abbiamo detto che, se il pattern di sparista' non e' simmetrico, lavoriamo su $(A + A^T)/2$? Ma questo vale solo per l'aggregazione? Dovremmo fare qualcosa di consistente anche con 1-lev Schwarz.** Both additive and hybrid preconditioners, 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, using object-oriented techniques, and is based on a distributed-memory parallel programming paradigm. **SALVATORE, potresti aggiungere due righe sulla scelta del Fortran 95 e sul semplice interfacciamento con i legacy codes, senza ripetere quello che e' detto sotto sulla scelta di PSBLAS?** 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 []. 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 [] 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 level 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.

Organizzazione della guida:
dire che per il momento non forniamo anche la documentazione del middle layer, ma lo faremo in seguito

Evidenziare le parole chiave che caratterizzano il nostro package

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 five steps are required:

1. *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 1; the string used by `mld_precinit` to identify each preconditioner type is also given. The preconditioner data structure is the derived data type `mld_prec_type`, which is accessed to the user only through the MLD2P4 routines.

2. *Choose a specific variant of the selected preconditioner type, by setting the preconditioner parameters.* This is performed by the routine `mld_precset`. A few examples concerning the use of `mld_precset` are given in Sections **??** and **??**; a complete list of all the preconditioner parameters and their allowed values is provided in Section 6.

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

4. *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`).

5. *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.

Two simple example programs showing the (basic) use of MLD2P4 are reported in Section 5.1.

5.1 Examples

The simple code reported below shows how to set and apply the MLD2P4 default multi-level preconditioned, i.e. the two-level hybrid post-smoothed

Type	String	Default preconditioner
No preconditioner	'NOPREC'	(Considered only to use the PSBLAS Krylov solvers with no preconditioner.)
Diagonal	'DIAG'	—
Block Jacobi	'BJAC'	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 ILU(0) on the blocks.

Table 1: Preconditioner types and default choices.

Schwarz preconditioner, using block-Jacobi with ILU(0) on the blocks as basic preconditioner, a coarse matrix distributed among the processors, and four block-Jacobi sweeps with ILU(0) 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 within 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. Other statements concerning the use of PSBLAS are neglected too. The complete code can be found in the example program file `example_2lev_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 [].

```
  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_prec_type)  :: PRE
... ...
!
! 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 post-smoothed Schwarz)
  call mld_precinit(PRE,'ML',info)
!
! build the preconditioner
  call psb_precbld(A,PRE,DESC_A,info)
!
! set the solver parameters and the initial guess
  ... ...
!
! solve Ax=b with preconditioned BiCGSTAB
  call psb_krylov('BICGSTAB',A,PRE,b,x,tol,DESC_A,info)
  ... ...
!
! cleanup the preconditioner
  call mld_precfree(PRE,info)
!
! cleanup other data structures
  ... ...
!
! exit the parallel environment
  call psb_exit(ictxt)
  stop
```

MODIFICARE TUTTA LA PARTE CHE SEGUE:

- solo istruzioni diverse dall'esempio precedente (essenzialmente il setting del precondizionatore, magari con piu' chiamate a precset;
- lasciare l'osservazione sulla specifica esplicita del numero di livelli;
- rimandare al paragrafo successivo per una decrizione accurata di tutti i parametri;
- lasciare l'osservazione sui vecchi utenti di PSBLAS.

In the following we describe the general procedure for setting and building one of the MLD2P4 preconditioners. The user has first to prepare the preconditioner data structure by using the routine `mld_precinit`. Input parameters for this routine include a string parameter, needed to define the preconditioner type, and an optional integer parameter specifying the number of the levels in the case of a multi-level preconditioner. Note that if the optional parameter is not present and a multi-level preconditioner has been chosen, a two-level preconditioner is set. On the other hand, the integer parameter is ignored if the type of the preconditioner is not multilevel. In Table 1 we report both the possible choices for the preconditioner type and the related default preconditioners.

The user of MLD2P4 may set a lot of parameters for one-level and multilevel Schwarz, in order to define a different preconditioner than that of default choices. The parameters can be set through the routine `mld_precset`. The APIs of `mld_precinit` and `mld_precset` as well as the complete list of the parameters that can be set with the corresponding allowed values are reported in Section 6. In the following a simple code for a three-level hybrid post-smoothed Schwarz preconditioner, using RAS with overlap 1 as local preconditioner, with ILU(0) on the local blocks, a distributed coarse matrix, four block-Jacobi sweeps with the UMFPACK LU factorization on the blocks as coarse-matrix solver, is reported. Note that for the multi-level preconditioners, the levels are numbered in increasing order starting from the finest one, i.e. level 1 is the finest level. For more details, see the test program `example2.f90` in xxxx(directory dei test).

```
  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 data
  type(mld_dprec_type)  :: PRE
... ...
!
! initialization of the parallel environment

  call psb_init(ictxt)
  call psb_info(ictxt,iam,np)
... ...
! read and assemble the matrix A and the right-hand
! side vector b using PSBLAS routines for sparse
! matrix/vector management
... ...
! prepare the three-level hybrid post-smoothed Schwarz
! using RAS with overlap 1 as local preconditioner
!
  call mld_precinit(PRE,'ML',info,nlev=3)
  call mld_precset(PRE,mld_n_ovr_,novr=1,info,ilev=1)
  call mld_precset(PRE,mld_sub_restr_,psb_halo_,info,ilev=1)
NOTA: e' PROPRIO BRUTTO "PSB_HALO_", BISOGNEREBBE AVERE COSTANTI CHE HANNO IL P
!
! build preconditioner
  call psb_precbld(A,PRE,DESC_A,info)
!
! set solver parameters and initial guess
  ... ...
! solve Ax=b with preconditioned BiCGSTAB

  call psb_krylov('BICGSTAB',A,PRE,b,x,tol,DESC_A,info)
  ... ...
!
!  cleanup storage and exit
!
  call mld_precfree(PRE,info)
!
  call psb_gefree(b,DESC_A,info)
  call psb_gefree(x,DESC_A,info)
  call psb_spfree(A,DESC_A,info)
  call psb_cdfree(DESC_A,info)
!
  call psb_exit(ictxt)
  stop
```

Remark for users with PSBLAS-based legacy codes: when MLD2P4 is installed, a PSBLAS user, with a PSBLAS-based legacy code calling base preconditioners included in PSBLAS (NOPREC, DIAG and BJAC), is able to use the same preconditioners without changes to the code, if she/he includes in her/his program the file `psb_prec_mod`.

6 High-Level User Interface

At the upper layer of MLD2P4, five black-box routines encapsulate all the functionalities for the construction and the application of any of the multi-level preconditioners. In the following we give the details of the above routines. Note that for each routine are available four different versions depending on involved data types: Real-Single/Double Precision, Complex-Single/Double Precision.

6.1 Preconditioner Setup and Building

The setup of a MLD2P4 preconditioner is obtained by using the `mld_precinit` routine, which allocates and initializes the preconditioner data structure. The API of this routine as well as the description of the arguments is reported in Fig. 1. Note that the allowed values for the `ptype` argument are reported in Table 1 (Sec. 5).

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

Arguments:
    p       type(mld_dprec_type), input/output.
            The preconditioner data structure.
    ptype   character, input. The type of preconditioner.
    info    integer, output. Error code.
    nlev    integer, optional, input.
            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.
```

Figure 1: API of the routine for preconditioner allocation and inizialization.

```
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 2: 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. 2. 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 6.1.1). The API of this routine is reported in Fig. 3. Finally, to

```
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 3: API of the routine for preconditioner setup.

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 4.

6.1.1 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 open problem [**?**], the user may also choose to apply a

```
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 4: API of the routine for preconditioner building.

Parameter (`what`)	Allowed values (`val`)
`mld_ml_type_`	'ADD', 'MULT'
	Define the type of multi-level preconditioner.
`mld_prec_type_`	'DIAG', 'BJAC', 'AS'
	Define the smoother at a certain level.
`mld_smooth_pos_`	'PRE', 'POST', 'BOTH'
	Define the way to apply the smoother.

Table 2: Parameters for preconditioner type.

nonsmoothed aggregation technique, where the prolongator operator from the coarse to fine-space vertices is the simple piecewise 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 6.1.1 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;

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 6.1.1 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 BJAC is
`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 6.1.1. Its worth noting that, the classical AS method corresponds to

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.

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'.

6.2 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 5. 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 5: 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 PSBLAS

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.