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

February 21, 2008

Contents

1 Introduction

The MLD2P4 library provides ....

1.1 Programming model

The MLD2P4 librarary is based on the Single Program Multiple Data (SPMD) programming model: each process participating in the computation performs the same actions on a chunk of data. Parallelism is thus data-driven.

Because of this structure, many subroutines coordinate their action across the various processes, thus providing an implicit synchronization point, and therefore *must* be called simultaneously by all processes participating in the computation. However there are many cases where no synchronization, and indeed no communication among processes, is implied.

Throughout this user's guide each subroutine will be clearly indicated as:

Synchronous: must be called simultaneously by all the processes in the relevant communication context;

Asynchronous: may be called in a totally independent manner.

2 Preconditioner routines

The MLD2P4 library contains the implementation of many preconditioning techniques. The preconditioners may be applied as normal "base" preconditioners; alternatively multiple "base" preconditioners may be combined in a multilevel framework.

The base (one-level) preconditioners include:

- Diagonal Scaling
- Block Jacobi
- Additive Schwarz, Restricted Additive Schwarz and Additive Schwarz with Harmonic extensions;

The Jacobi and Additive Schwarz preconditioners can make use of the following solvers:

- Level-p Incomplete LU factorization ($ILU(p)$);
- Threshold Incomplete LU factorization ($ILU(\tau, p)$);
- Complete LU factorization by means of the following optional external packages:
  - UMFPACK;
  - SuperLU;
  - SuperLU_Dist.

The supporting data type and subroutine interfaces are defined in the module `mld_prec_mod`; the module also overrides the variables and tyep definitions of `psb_prec_mod` so as to function as a drop-in replacement for the PSBLAS methods. Thus if the user does not wish to employ the additional MLD2P4 capabitlities, it is possible to migrate an existing PSBLAS program without any source code modifications, only a recompilation is needed.

mld_precinit—Initialize a preconditioner

Syntax

call mld_precinit (*prec, ptype, info*)

call mld_precinit (*prec, ptype, info, nlev*)

Type: Asynchronous.

On Entry

ptype the type of preconditioner. Scope: **global**
Type: **required**
Intent: **in**.
Specified as: a character string, see usage notes.

nlev Number of levels in a multilevel precondtioner. Scope: **global**
Type: **optional**
Specified as: an integer value, see usage notes.

On Exit

prec Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: a preconditioner data structure `mld_prec_type`.

info Scope: **global**
Type: **required**
Intent: **out**.
Error code: if no error, 0 is returned.

Usage Notes

Legal inputs to this subroutine are interpreted depending on the *ptype* string as follows[1]:

NONE No preconditioning, i.e. the preconditioner is just a copy operator.

DIAG Diagonal scaling; each entry of the input vector is multiplied by the reciprocal of the sum of the absolute values of the coefficients in the corresponding row of matrix A;

BJAC Precondition by a factorization of the block-diagonal of matrix A, where block boundaries are determined by the data allocation boundaries for each process; requires no communication.

[1]The string is case-insensitive

AS Additive Schwarz; default is to apply the Restricted Additive Schwarz variant, with an $ILU(0)$ factorization

ML Multilevel preconditioner.

mld_precset—Set preconditioner features

Syntax

call mld_precset (*prec, what, val, info, ilev*)

Type: Asynchronous.

On Entry

prec the preconditioner.
Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: an already initialized precondtioner data structure `mld_prec_type`

what The feature to be set.
Scope: **local**
Type: **required**
Intent: **in**.
Specified as: an integer constants. Symbolic names are available in the library module, see usage notes for legal values.

val The value to set the chosen feature to.
Scope: **local**
Type: **required**
Intent: **in**.
Specified as: an integer, double precision or character variable. Symbolic names for some choices are available in the library module, see usage notes for legal values.

ilev The level of a multilevel preconditioner to which the feature choice should apply.
Scope: **global**
Type: **optional**
Specified as: an integer value, see usage notes.

On Return

prec the preconditioner.
Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: a precondtioner data structure `mld_prec_type`

info Error code.
Scope: **local**

Type: **required**
Intent: **out**.
An integer value; 0 means no error has been detected.

Usage Notes

Legal inputs to this subroutine are interpreted depending on the value of `what` input as follows

mld_coarse_mat_

mld_precbld—Builds a preconditioner

Syntax

call mld_precbld (*a, desc_a, prec, info*)

Type: Synchronous.

On Entry

a the system sparse matrix. Scope: **local**
Type: **required**
Intent: **in**, target.
Specified as: a sparse matrix data structure `psb_spmat_type`.

prec the preconditioner.
Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: an already initialized precondtioner data structure `mld_prec_type`

desc_a the problem communication descriptor. Scope: **local**
Type: **required**
Intent: **in**, target.
Specified as: a communication descriptor data structure `psb_desc_type`.

On Return

prec the preconditioner.
Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: a precondtioner data structure `mld_prec_type`

info Error code.
Scope: **local**
Type: **required**
Intent: **out**.
An integer value; 0 means no error has been detected.

mld_precaply—Preconditioner application routine

Syntax

call mld_precaply ($prec,x,y,desc\_a,info,trans,work$)

call mld_precaply ($prec,x,desc\_a,info,trans$)

Type: Synchronous.

On Entry

prec the preconditioner. Scope: **local**
Type: **required**
Intent: **in**.
Specified as: a preconditioner data structure `mld_prec_type`.

x the source vector. Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: a double precision array.

desc_a the problem communication descriptor. Scope: **local**
Type: **required**
Intent: **in**.
Specified as: a communication data structure `psb_desc_type`.

trans Scope:
Type: **optional**
Intent: **in**.
Specified as: a character.

work an optional work space Scope: **local**
Type: **optional**
Intent: **inout**.
Specified as: a double precision array.

On Return

y the destination vector. Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: a double precision array.

info Error code.
Scope: **local**
Type: **required**
Intent: **out**.
An integer value; 0 means no error has been detected.

mld_prec_descr—Prints a description of current preconditioner

Syntax

call mld_prec_descr (*prec*)

Type: Asynchronous.

On Entry

prec the preconditioner. Scope: **local**
Type: **required**
Intent: **in**.
Specified as: a preconditioner data structure `mld_prec_type`.

3 Iterative Methods

In this chapter we provide routines for preconditioners and iterative methods. The interfaces for Krylov subspace methods are available in the module `mld_krylov_mod`. The installation process of MLD2P4 ensures that these may be used as a drop-in replacement for the PSBLAS methods; they are accessible under the PSBLAS names (see the PSBLAS documentation),

mld_krylov —Krylov Methods Driver Routine

This subroutine is a driver that provides a general interface for all the Krylov-Subspace family methods.

The stopping criterion is the normwise backward error, in the infinity norm, i.e. the iteration is stopped when

$$err = \frac{\|r_i\|}{(\|A\|\|x_i\| + \|b\|)} < eps$$

or the 2-norm residual reduction

$$err = \frac{\|r_i\|}{\|b\|_2} < eps$$

according to the value passed through the istop argument (see later). In the above formulae, x_i is the tentative solution and $r_i = b - Ax_i$ the corresponding residual at the i-th iteration.

Syntax

call psb_krylov
(method,a,prec,b,x,eps,desc_a,info,itmax,iter,err,itrace,irst,istop)

Type: Synchronous.

On Entry

method a string that defines the iterative method to be used. Supported values are:

- **CG** : the Conjugate Gradient method;
- **CGS** :the Conjugate Gradient Stabilized method;
- **BICG** : the Bi-Conjugate Gradient method;
- **BICGSTAB** : the Bi-Conjugate Gradient Stabilized method;
- **BICGSTABL** : the Bi-Conjugate Gradient Stabilized method with restarting;
- **RGMRES** : the Generalized Minimal Residual method with restarting.

a the local portion of global sparse matrix A.
Scope: **local**
Type: **required**
Intent: **in**.
Specified as: a structured data of type `psb_spmat_type`.

prec The data structure containing the preconditioner.
Scope: **local**
Type: **required**
Intent: **in**.
Specified as: a structured data of type `mld_prec_type`.

b The RHS vector.
Scope: **local**
Type: **required**
Intent: **in**.
Specified as: a rank one array.

x The initial guess.
Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: a rank one array.

eps The stopping tolerance.
Scope: **global**
Type: **required**
Intent: **in**.
Specified as: a real number.

desc_a contains data structures for communications.
Scope: **local**
Type: **required**
Intent: **in**.
Specified as: a structured data of type psb_desc_type.

itmax The maximum number of iterations to perform.
Scope: **global**
Type: **optional**
Intent: **in**.
Default: $itmax = 1000$.
Specified as: an integer variable $itmax \geq 1$.

itrace If > 0 print out an informational message about convergence every *itrace* iterations.
Scope: **global**
Type: **optional**
Intent: **in**.

irst An integer specifying the restart parameter.
Scope: **global**
Type: **optional**.
Intent: **in**.
Values: $irst > 0$. This is employed for the BiCGSTABL or RGMRES methods, otherwise it is ignored.

istop An integer specifying the stopping criterion.
Scope: **global**
Type: **optional**.
Intent: **in**.
Values: 1: use the normwise backward error, 2: use the scaled 2-norm of the residual. Default: 1.

On Return

x The computed solution.
Scope: **local**
Type: **required**
Intent: **inout**.
Specified as: a rank one array.

iter The number of iterations performed.
Scope: **global**
Type: **optional**
Intent: **out**.
Returned as: an integer variable.

err The convergence estimate on exit.
Scope: **global**
Type: **optional**
Intent: **out**.
Returned as: a real number.

info Error code.
Scope: **local**
Type: **required**
Intent: **out**.
An integer value; 0 means no error has been detected.

References

[1] G. Bella, S. Filippone, A. De Maio and M. Testa, *A Simulation Model for Forest Fires*, in J. Dongarra, K. Madsen, J. Wasniewski, editors, Proceedings of PARA 04 Workshop on State of the Art in Scientific Computing, pp. 546–553, Lecture Notes in Computer Science, 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] Dongarra, J. J., DuCroz, J., Hammarling, S. and Hanson, R., An Extended Set of Fortran Basic Linear Algebra Subprograms, ACM Trans. Math. Softw. vol. 14, 1–17, 1988.

[5] Dongarra, J., DuCroz, J., Hammarling, S. and Duff, I., A Set of level 3 Basic Linear Algebra Subprograms, ACM Trans. Math. Softw. vol. 16, 1–17, 1990.

[6] 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).

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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`.

[12] 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.

[13] 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.

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

[15] 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.