Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

2LEVDD-PSBLAS: a package of high-performance preconditioners for scientific and engineering applications

Alfredo Buttari, Pasqua D'Ambra, Daniela di Serafino, Salvatore Filippone

RT-ICAR-NA-2005-20 **Dicembre 2005**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

2LEVDD-PSBLAS: a package of high-performance preconditioners for scientific and engineering applications [1]

Alfredo Buttari°, Pasqua D'Ambra♦
Daniela di Serafino♠, Salvatore Filippone♣

Rapporto Tecnico N.: **RT-ICAR-NA-2005-20**

Data: **Dicembre 2005**

---

[1] Il presente rapporto è il pre-print di un lavoro sottoposto alla rivista "Applicable Algebra in Engineering, Communication and Computing", Special Issue on "Computational Linear Algebra and Sparse Matrix Computations".

° Department of Mechanical Engineering, University of Rome ``Tor Vergata", Viale del Politecnico, I-00133, Rome, Italy.
♦ Institute for High-Performance Computing and Networking, CNR, Via Pietro Castellino 111, I-80131 Naples, Italy.
♠ Department of Mathematics, Second University of Naples, Via Vivaldi 43, I-81100 Caserta, Italy.
♣ Department of Mechanical Engineering, University of Rome ``Tor Vergata", Viale del Politecnico, I-00133, Rome, Italy.

2LEVDD-PSBLAS: a package of high-performance preconditioners for scientific and engineering applications

Alfredo Buttari[*], Pasqua D'Ambra[†], Daniela di Serafino[‡], Salvatore Filippone[§]

Abstract

We present a package of parallel algebraic two-level Schwarz preconditioners, which has been developed on the top of the Parallel Sparse BLAS (PSBLAS) library. The package, named 2LEVDD-PSBLAS, currently includes various versions of additive Schwarz preconditioners that are combined with a coarse-level correction, in either additive or multiplicative ways, to obtain two-level preconditioners; a smoothed aggregation algorithm is implemented to build the coarse-level correction. 2LEVDD-PSBLAS was designed to fully exploit the basic linear algebra kernels provided by PSBLAS; nevertheless, it required the implementation of a few more matrix operators, thus leading to an extension of the original PSBLAS version. The package, used with Krylov solvers implemented in PSBLAS, was tested on linear systems arising from different large-scale applications. Performance results are discussed in the paper.

1 Overview

The solution of large and sparse linear systems,

$$Ax = b, \quad A \in \Re^{n \times n},\ b \in \Re^n, \tag{1}$$

is often the main computational kernel of large-scale applications in science and engineering. These systems generally arise from the discretization of Partial Differential Equations (PDEs), but they come also from applications not governed by PDEs, such as the design and analysis of circuits, chemical engineering processes, queueing systems and economic models. Krylov iterative solvers coupled with suitable preconditioners are the method of choice in many cases, especially when the

[*]Department of Computer Science Engineering, University of Rome "Tor Vergata", Viale del Politecnico, I-00133, Rome, Italy.

[†]Institute for High-Performance Computing and Networking, CNR, Via Pietro Castellino 111, I-80131 Naples, Italy.

[‡]Department of Mathematics, Second University of Naples, Via Vivaldi 43, I-81100 Caserta, Italy.

[§]Department of Mechanical Engineering, University of Rome "Tor Vergata", Viale del Politecnico, I-00133, Rome, Italy.

matrix dimension reaches the order of $10^6 - 10^8$, such as in 3D multiphysics and multiscale simulations. Therefore, much effort is put in developing parallel algorithms and software implementing Krylov methods and preconditioners, in order to provide computational scientists with effective tools for building their application codes.

In this paper we present a package of preconditioners for high-performance architectures, based on Domain Decomposition algorithms of Schwarz type. The package is implemented on the top of Parallel Sparse BLAS (PSBLAS) [17], a library for sparse basic linear algebra operations, that includes parallel versions of most of the Sparse BLAS computational kernels proposed in [14] and auxiliary routines for the creation and management of distributed sparse matrices. The choice of PSBLAS was motivated by the need of having a portable, efficient and modular software infrastructure, that provides also a smooth upgrade path for integration in legacy application codes [2, 18]. Our package currently includes different versions of one-level and two-level Schwarz preconditioners [6, 7, 9, 20]. A pure algebraic formulation is considered, as outlined below.

Let $G = (W, E)$ be the adjacency graph of the matrix A in (1), where W and E are the vertex set and the edge set, respectively. Two vertices are called adjacent if there is an edge connecting them. Starting from a 0-overlap partition of W, i.e. from a set of m disjoint nonempty sets $W_i^0 \subset W$ such that $\cup_{i=1}^m W_i^0 = W$, for $\delta > 0$ a δ-overlap partition of W can be defined recursively, by considering the sets $W_i^\delta \supset W_i^{\delta-1}$ obtained by including the vertices that are adjacent to any vertex in $W_i^{\delta-1}$. Let n_i^δ be the size of W_i^δ and $R_i^\delta \in \Re^{n_i^\delta \times n}$ the restriction operator that maps a vector $v \in \Re^n$ onto the vector $v_i^\delta \in \Re^{n_i^\delta}$ containing the components of v corresponding to the vertices in W_i^δ. The transpose of R_i^δ is a prolongation operator from $\Re^{n_i^\delta}$ to $\Re^n$. The matrix $A_i^\delta = R_i^\delta A (R_i^\delta)^T \in \Re^{n_i^\delta \times n_i^\delta}$ can be regarded as a restriction of A corresponding to the set W_i^δ.

The *classical Additive Schwarz* (AS) preconditioner is defined as

$$M_{AS}^{-1} = \sum_{i=1}^{m} (R_i^\delta)^T (A_i^\delta)^{-1} R_i^\delta, \tag{2}$$

while the *Restricted AS* (RAS) and *AS with Harmonic extension* (ASH) variants are defined as

$$M_{RAS}^{-1} = \sum_{i=1}^{m} (\tilde{R}_i^0)^T (A_i^\delta)^{-1} R_i^\delta, \qquad M_{ASH}^{-1} = \sum_{i=1}^{m} (R_i^\delta)^T (A_i^\delta)^{-1} \tilde{R}_i^0,$$

where $\tilde{R}_i^0 \in \Re^{n_i^\delta \times n}$ is obtained by zeroing the rows of R_i^δ identified by the vertices in $W_i^\delta \backslash W_i^0$. For all the AS preconditioners A_i^δ is assumed to be nonsingular. The application of the preconditioner (2) with a Krylov solver requires the solution of a system of the form $M_{AS} z = v$, that corresponds to the following steps

$$v_i = R_i^\delta v, \qquad i = 1, \ldots, m, \tag{3}$$

$$\text{solve} \quad A_i^\delta w_i = v_i, \quad i = 1, \ldots, m, \tag{4}$$

$$z = \sum_{i=1}^{m} (R_i^\delta)^T w_i. \tag{5}$$

With RAS R_i^δ in (5) is replaced by $\tilde{R}_i^0$, while with ASH R_i^δ in (3) is replaced by $\tilde{R}_i^0$.

The AS preconditioners exhibit an intrinsic parallelism, since the computations concerning different subdomains can be performed by different processors. On the other hand, the convergence theory for the AS preconditioners shows that, when they are used in conjunction with a Krylov subspace method, the convergence rapidly improves as the overlap δ increases, while it deteriorates as the number m of subsets W_i^δ grows. Therefore, in a parallel setting, the AS preconditioners are widely used in conjuction with some coarse space, where the original linear system can be approximated to provide a suitable improvement to the solution [8, 9, 20]. The use of this space introduces some extra work that has a sequential nature, but is necessary for developing scalable preconditioning algorithms.

In a pure algebraic setting, a coarse-space approximation A_C of the matrix A is usually built with a Galerkin approach. Given a set W_C of coarse vertices, with size n_C, and a suitable restriction operator $R_C \in \Re^{n_C \times n}$, A_C is defined as $A_C = R_C A R_C^T$ and the coarse-space correction matrix to be combined with the AS preconditioners is obtained as

$$M_C^{-1} = R_C^T A_C^{-1} R_C, \tag{6}$$

where A_C is assumed to be nonsingular. The matrices W_C and R_C can be built by using an *aggregation* technique [5, 22, 23].

M_C can be combined with any AS preconditioner (henceforth denoted by M_{1L}) in either an additive or a multiplicative way to obtain two-level preconditioners. The additive combination leads to

$$M_{2LA}^{-1} = M_C^{-1} + M_{1L}^{-1},$$

which corresponds to applying the coarse-level correction and the basic AS preconditioner independently and then summing up the results. An example of multiplicative combination is given by the following hybrid preconditioner, that we refer to as *2LH-post*:

$$M_{2LH-post}^{-1} = M_{1L}^{-1} + \left(I - M_{1L}^{-1} A\right) M_C^{-1},$$

which is obtained by applying first the coarse-level correction and then basic AS preconditioner:

$$\begin{aligned} w &= M_C^{-1} v, \\ z &= w + M_{1L}^{-1}(v - Aw). \end{aligned}$$

Other two-level hybrid preconditioners are obtained by applying M_{1L} only before or before and after the coarse-level correction.

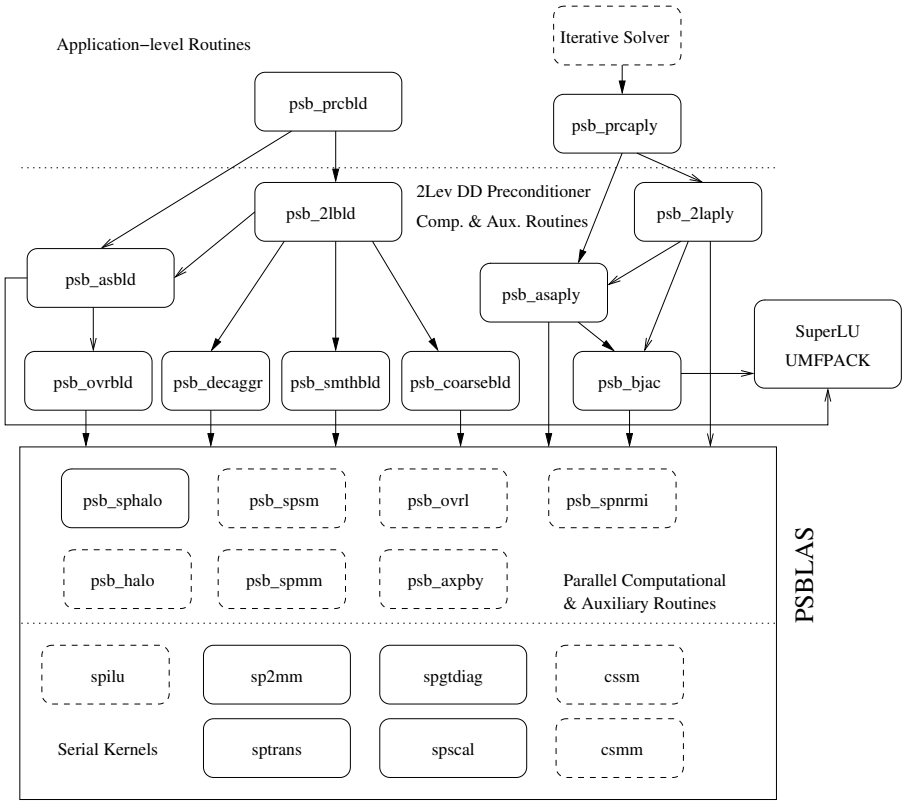

Figure 1: Software architecture of 2LEVDD-PSBLAS.

2 Software architecture of 2LEVDD-PSBLAS

Our main effort was devoted to develop a package of parallel two-level Schwarz preconditioners based on standard kernels for sparse linear algebra operations. The package was therefore designed to fully exploit the existing functionalities of the PSBLAS library. However, its implementation required an extension of the PSBLAS basic kernels.

The software architecture of 2LEVDD-PSBLAS is shown in Figure 1. Three main layers can be identified. The lower layer consists of the original and the new PSBLAS kernels, the middle layer implements the construction and application phases of the preconditioners and the upper layer provides a uniform and easy-to-use interface to all the preconditioners.

2.1 Lower layer

PSBLAS was designed to provide basic operators needed to implement iterative methods for the solution of sparse linear systems on distributed-memory parallel

computers. The original 1.0 version was written in a mixture Fortran 77 and C, with a user interface exploiting modern features of Fortran 95, such as facilities for data encapsulation, functional overloading and dynamic memory management; the current development version has been completely reimplemented in Fortran 95. Inter-process data communications are performed using BLACS [13] and MPI [21].

PSBLAS supports a general row-block distribution of sparse matrices, with conformal distribution of vectors and dense matrices. Two main data structures, implemented as Fortran 95 derived data types, are used to hold the information concerning a distributed sparse matrix: the *Sparse Matrix* and the *Communication Descriptor*. The former includes all the information about the local part of a distributed sparse matrix and its storage mode, while the latter contains a representation of the set of indices involved in the parallel data distribution, including the indices identifying the 1-overlap layer.

The PSBLAS basic routines can be divided into *computational* and *auxiliary* routines [17]. The former implement parallel algorithms for most of the Sparse BLAS (SBLAS) computational kernels proposed in [14], while the latter provide support for parallel sparse matrix management. In PSBLAS 1.0 the set of computational routines includes the sparse matrix by dense matrix (or vector) multiplication, the sparse triangular system solution (only for block diagonal matrices), the dense matrix sum, the dot product and some vector and matrix norms. An implementation of the SBLAS specifications in [14] is also provided to perform the serial sparse matrix computations required by the parallel kernels. The auxiliary routines of PSBLAS 1.0 implement the operations for allocating, building and updating sparse and dense distributed matrices and the typical data communications involved in parallel sparse matrix computations.

The construction of the basic (one-level) AS preconditioners required the extension of the set of auxiliary routines, to build the matrices A_i^δ. A new routine was developed to enlarge A_i^0 in each processor, i.e. to gather the rows of the matrix A that correspond to the indices in $W_i^\delta \backslash W_i^0$. This functionality was put in the PSBLAS layer since it can be used in a more general context, e.g. to build extended stencils, that are often required by numerical simulations involving PDEs. The construction of the coarse-level matrix A_C and its application required the extension of the set of PSBLAS sequential kernels with routines performing sparse matrix diagonal scaling, sparse matrix transpose and sparse matrix by sparse matrix multiplication. The last two operations were implemented by integrating into PSBLAS the SMMP software [1]. We note that, although the sparse matrix multiplication is not considered in the last SBLAS standard, the possibility of its future inclusion was foreseen by the BLAS Technical Forum [15]. Further details on the extension of PSBLAS can be found in [3, 10].

2.2 Middle layer

The middle layer of 2LEVDD-PSBLAS, written in Fortran 95, consists of the routines that implement the construction and the application phases of the preconditioners. A key issue in the implementation of this layer was the definition of a *Preconditioner* data structure that contains all the items needed by the preconditioners, but reuses the basic PSBLAS data structures to exploit the functionalities of the PSBLAS computational and auxiliary routines. Inside this structure the matrices A_i^δ, A_C and R_C are represented by using the Sparse Matrix and the Communication Descriptor data types; a representation of the operator R_i^δ is provided by the Communication Descriptor of A_i^δ, that includes a representation of the indices in $W_i^\delta \backslash W_i^0$.

The construction of the one-level preconditioners was implemented in two main steps: the identification of the overlap indices needed to build A_i^δ, through the algorithm described in [3], and the enlargment of the matrix A_i^0, through the new auxiliary PSBLAS routine mentioned in Section 2.1. For the construction of the coarse matrix A_C the smoothed aggregation technique [5, 23] was chosen. This required the implementation of three main tasks [10]: a parallel decoupled aggregation procedure, to build the coarse-space vertex set W_C from the original vertex set W; the application of a damped Jacobi smoother to a simple piecewise constant interpolation operator, to obtain the coarse-to-fine prolongation operator R_C; the construction of the coarse matrix $A_C = R_C A R_C^T$, where A_C can be distributed among the processors or replicated on each of them. The latter two steps were performed by using the new sequential sparse matrix operators integrated into PSBLAS.

The routines for the application of the preconditioners were implemented by exploiting different PSBLAS kernels. Sparse matrix management routines are used to perform the steps (3) and (5) of the basic AS preconditioners, while parallel sparse matrix by vector multiplications and dense vector sums are applied to apply the coarse-level correction matrix (6) and to combine it with the one-level preconditioners. The solution of the system (4), involving the matrices A_i^δ, is accomplished by either the ILU or the LU factorization. The sequential ILU routine available in PSBLAS 1.0 is used in the former case; an interface to UMFPACK [11], version 4.4, and to SuperLU [12], version 3.0, was developed to deal with the latter case. The same options are available for the system involving A_C, when this matrix is replicated among the processors. On the other hand, to solve the coarse-level systems when A_C is distributed, a block-Jacobi routine was developed, that uses the ILU or LU factorizations on the diagonal blocks of the coarse matrix held by the processors.

2.3 Upper layer

At the upper layer of 2LEVDD-PSBLAS two routines encapsulate all the functionalities for the construction and the application of the preconditioner, respectively. A further routine is provided to set the preconditioner options according to the choice

```
subroutine psb_prcset(prec,ptype,iv,rs,rv,info)
  type(psb_dprec_type), intent(inout)    :: prec
  character(len=*), intent(in)           :: ptype
  integer, optional, intent(in)          :: iv(:)
  real(kind(1.d0)), optional, intent(in) :: rs
  real(kind(1.d0)), optional, intent(in) :: rv(:)
  integer, optional, intent(out)         :: info
end subroutine psb_prcset

subroutine psb_prcbld(a,prec,desc_a,info,upd)
  integer, intent(out)                :: info
  type(psb_dspmat_type), intent(in), target  :: a
  type(psb_dprec_type),intent(inout)         :: prec
  type(psb_desc_type), intent(in)            :: desc_a
  character, intent(in), optional            :: upd
end subroutine psb_prcbld
```

Figure 2: APIs of the routines for the preconditioner setup.

of the preconditioner made by the user. These routines make available, through simple Fortran 95 APIs, the various basic AS and two-level preconditioners. The Krylov solvers provided in PSBLAS 1.0 [17] were also slightly modified to use the preconditioners. More details on the upper layer interface are given in the next section.

3 Example of use

From the point of view of the 2LEVDD-PSBLAS user, building a preconditioner requires two basic steps: setting the preconditioner options, and creating and defining the Preconditioner data structure. In order to perform these steps, two black-box routines are available, `psb_prcset` and `psb_prcbld`, whose Fortran 95 APIs are shown in in Figure 3.

In the `psb_prcset` API, the main input parameters are the Preconditioner data structure, named `prec`, and the `ptype` string variable, defining the choice of the preconditioner made by the user. Currently, the possible values for `ptype` are: `noprec` (no preconditioner), `diagsc` (diagonal preconditioner), `bja` (Block-Jacobi preconditioner, which is a special case of AS), `asm` (one-level Schwarz preconditioner) and `ml` (two-level Schwarz preconditioner).

The different items of the `iv` optional array are used to specify various parameters of the Schwarz preconditioners. For the one-level preconditioners, the entries specify the number of overlap layers, the type of restriction (R_i^δ or $\tilde{R}_i^0$), the type of prolongation and the type of factorization to be employed. For the two-level pre-

```
subroutine psb_prcaply(prec,x,y,desc_data,info,trans,work)
  type(psb_desc_type), intent(in)   :: desc_data
  type(psb_dprec_type), intent(in)  :: prec
  real(kind(0.d0)), intent(inout)   :: x(:), y(:)
  integer, intent(out)              :: info
  character(len=1), optional        :: trans
  real(kind(0.d0)), intent(inout), optional, target :: work(:)
end subroutine psb_dprecaply
```

Figure 3: APIs of the routine for the preconditioner application.

conditioners, the user must first set the parameters of the fine level, and afterwards he/she can set the parameters concerning the coarse-level correction. The user can specify the type of two-level framework (additive or multiplicative), details of the aggregation algorithm, the "position" of the one-level preconditioner (before, after, or before and after the coarse-level correction), the coarse matrix storage (distributed or replicated), the type of solver to be employed at the coarse level and related details. Default values are provided for all the optional parameters.

The application of a Schwarz preconditioner at each iteration of a Krylov solver is obtained by calling the black-box routine `psb_prcaply`, which has the API shown in Figure 3. This routine has a slightly more general interface than required by the usual preconditioner application, since it computes a vector update of the type $y = \beta y + op(M^{-1})\, x$, where M is a previously built preconditioner, stored in the `prec` data structure, and $op(B)$ denotes B or its transpose, according to value of `trans`.

An example of use of 2LH-post, with RAS (overlap 1) as basic preconditioner, distributed coarse matrix, four Block-Jacobi sweeps and UMFPACK LU factorization, coupled with the BiCGSTAB solver implemented in PSBLAS is sketched in Figure 4. More details on the use of the routines for the setup and the application of the preconditioners can be found in [4].

4 Performance results

We have tested our software by solving linear systems with coefficient matrices arising from simulations of the thermo-fluidynamics in an automotive engine and of the thermal diffusion in some solids.

The automotive engine test cases are snapshots from the simulation of a direct injection diesel engine, from a commercial automotive manufacturer. The matrices arise from the discretization of the pressure correction equation in the implicit phase of a semi-implicit algorithm (ICED-ALE [19]) for the solution of unsteady Navier-Stokes equations for compressible flows, as implemented in the KIVA application

```
! 2LEVDD-PSBLAS example program
!
  use psb_sparse_mod
!
! sparse matrices
  type(psb_dspmat_type) :: a
  type(psb_dprec_type)  :: pre
!
! communication data structure
  type(psb_desc_type)    :: desc_a
!
! preconditioner optional parameters
  integer, pointer :: iv(:)
!
! other parameters
  ...
!
! read and assemble the matrix A and the right-hand
! side vector b
  ...
!
! set preconditioner options
  novr = 1
  nsweep = 4
  call psb_precset(pre,'asm',iv=(/novr,halo_,none_/))
  call psb_precset(pre,'ml',&
    & iv=(/mult_ml_prec_,loc_aggr_,smth_omg_,mat_distr_,&
    &      post_smooth_,f_umf_,nsweep/))
!
! build preconditioner
  call psb_precbld(a,pre,desc_a,info)
!
! set solver parameters
  ...
!
! solve Ax=b with preconditioned BiCGSTAB
  call psb_bicgstab(a,pre,b,x,tol,desc_a,info)
!
  ...
```

Figure 4: Example of use of 2LEVDD-PSBLAS.

software modified to make use of the PSBLAS linear solvers [18]. The discretization mesh contains approximately 100000 control volumes; during the simulation the size of the actual domain varies, because mesh layers are activated/deactivated following

the piston movement. The matrices correspond to different positions of the piston inside the engine cylinder; they have a symmetric sparsity pattern, with no more than 19 non-zero entries per row. We report here the results concerning two matrices, named *kivap1* and *kivap2*, with dimensions 86304 and 42204, respectively.

Two other matrices were considered, that arise from the discretization of the steady-state thermal diffusion equation in solids. The first one, named *therm2D*, is of dimension 600000 and comes from a 2D model of a homogeneous plate, with a central finite-difference discretization scheme on a regular mesh. The second matrix, *therm3D*, has about 1 million rows and has been extracted from an experimental finite volume code that deals with the steady thermal conduction in materials with variable conductivity. A central discretization scheme is used in the code, including the deferred-correction approach proposed in [16] for handling non-orthogonal computational meshes. The simulation considered here concerns an aluminium Diesel engine piston discretized by using a tetrahedral mesh, with prescribed heat flux on the piston head and prescribed temperatures on the remaining surfaces.

The tests discussed here were carried out on a cluster with dual-processor nodes, installed at the Innovative Computing Laboratory of the University of Tennessee at Knoxville. Each node has an AMD Opteron dual-processor (model 240, 1.4 GHz), running the Debian Linux 3.1 operating system with kernel 2.6.13, and 2 GBytes of memory; the nodes are connected with Myrinet network interfaces. The tests were run on 32 nodes, i.e. on 64 processors. A development snapshot of the GNU compilers version 4.2, including both the C and Fortran 95 compilers, was used in conjunction with the specific MPI implementation for the Myrinet interface.

For sake of space, we present timing and speedup results only for RAS, with overlap 0 (Block-Jacobi) and 1, and for two variants of the two-level hybrid preconditioner 2LH-post, using RAS wih overlap 1 at the fine level. The coarse-space matrix is distributed among the processors and four Block-Jacobi sweeps are applied to the corresponding system; the first variant uses the ILU factorization on the diagonal blocks, while the second uses the LU factorization implemented in UMFPACK. In all the preconditioners, the systems arising in the application of RAS are solved by ILU. The preconditioners are applied as right preconditioners with the BiCGSTAB solver available in PSBLAS, choosing the null vector as starting guess. The iterations are stopped when the ratio between the 2-norms of the residual and of the right-hand-side is less than 10^{-6}; a maximum number of 1000 iterations is also set, but it is never reached in the tests. A row-block distribution of the matrices is used, where each processor holds (approximately) equal-sized blocks of consecutive rows, according to the well-known BLACS one-dimensional pure-block mapping. A conformal distribution is applied to the right-hand side and solution vectors. This choice implicitly defines a domain decomposition such that the number of subdomains is equal to the number of processors. Results of a comparison of our package with well-established software implementing Schwarz preconditioners, carried out on the engine simulation and the 2D thermal diffusion matrices using other Linux clusters, are reported in [3, 10].

For all the matrices we show, in Tables 1-4, the number of BiCGSTAB iterations (It), the execution times for the preconditioner setup (Setup) and for the solution of the preconditioned system (Solve), and the total times (Total), for NP = 1, 2, 4, 8, 16, 32, 64 processors. All the times are measured in seconds and are mean values over six executions. With therm3D we could not run the version of 2LH-post using the LU factorization for NP = 1, 2, because of the high memory requirements.

As expected, for kivap1 and kivap2 the best execution times are obtained with the RAS preconditioner, since the coarse-level correction is known to have a mild impact on matrices that do not come for pure elliptic problems. For the kivap matrices we see that both the two-level preconditioners lead to a reduction of the iterations with respect to RAS, but this reduction is not strong enough to balance the larger execution times needed for building and applying the coarse-space corrections. The two-level preconditioners are more effective on therm2D and therm3D, that arise from the discretization of elliptic PDEs. However, while for therm2D the smallest execution times are obtained by using 2LH-post with the LU factorization, on therm3D the two-level preconditioners are outperformed by RAS, except in a few cases. A closer look at the execution times of 2LH-post with LU shows that this preconditioner requires a much greater setup time than the RAS variants, because of the very large size of the matrix and of the sparsity structure, arising from a nine-point discretization stencil; conversely, the solution time is smaller for the two-level preconditioner. This is interesting in real applications for two reasons. First, we can reuse some of the aggregation and coarse matrix data structures in all the cases where the aggregation is purely topological and the matrix pattern has not changed between two successive invocations of the solver. Moreover, it may be feasible to reuse the same preconditioner for a number of outer iterations of a nonlinear solver if the coefficient matrix does not change too much.

In Figures 5-8 we plot the speedups of the four selected preconditioners for each of the test matrices; the values concerning 2LH-post with the LU factorization on therm3D are missing, since we could not run this test for NP = 1, 2. The speedups were computed using the total times, including the time needed to build the preconditioner. The overall results are affected by the increase in the number of BiCGSTAB iterations that can be observed with the increase in the number of processors.

As expected, RAS with overlap 0 generally shows good speedup values. The only exceptions are provided by kivap2 on 32 and 64 processors; in this case the speedup tends to saturate because of the relatively small size of the matrix. The highest speedup value on 64 processor is close to 35 and is obtained with therm2D. By using RAS with overlap 1, a speedup decrease can be observed on the engine simulation and the 3D thermal diffusion matrices; speedup values comparable with those corresponding to overlap 0 are obtained on therm2D, which has a sparsity pattern coming from a simple discretization stencil. For kivap1 and kivap2 the speedup of 2LH-post with ILU has a very close behaviour to that of RAS with overlap 1. On therm2D, 2LH-post with ILU shows a small speedup increase with

respect to RAS, with a value close to 40 on 64 processors. On therm3D, instead, the speedup for 2LH-post with ILU is lower than for the RAS cases; this is mainly due to the size and the sparsity structure of the matrix. The situation is different for 2LH-post with LU. The highest speedup is now achieved on kivap1, for which the number of BiCGSTAB iterations is approximately constant, but the time required by the LU factorization significantly reduces in going from 1 to 64 processors. On kivap2 the speedup behaviour is comparable with that of the other two-level preconditoner, while on therm2D a strong speedup reduction is observed, which is mainly due to the large increase of the iterations, and hence of the solve time, as the number of processors grows.

5 Conclusions and future work

In this paper we described 2LEVDD-PSBLAS, a package of two-level algebraic Schwarz preconditioners for high-performance computers, based on the PSBLAS library. We provided a general overview of the package, focusing on its software architecture, functionalities and user interface. We presented also performance results obtained with different preconditioners implemented in 2LEVDD-PSBLAS on a set of test problems arising from large-scale applications.

Our workplan includes the extension of the package to multilevel Schwarz preconditioners and the integration and testing of the preconditioners inside engineering applications. Work will be also devoted to including other factorizations, such as ILU(n) and ILU with threshold, and more sophisticated aggregation algorithms.

Acknowledgments

The authors wish to thank Prof. Jack Dongarra, director of the Innovative Computing Laboratory of the University of Tennessee at Knoxville, for the usage of the machine on which the results of Section 4 were obtained. They also wish to thank Stefano Toninel of the DIEM, University of Bologna, and Prof. David P. Schmidt, of the University of Massachussetts at Amherst, for providing the thermal diffusion test cases.

kivap1								
	RAS, overlap 0				RAS, overlap 1			
NP	It	Setup	Solve	Total	It	Setup	Solve	Total
1	12	.2645	1.0830	1.3475	12	0.2658	1.1260	1.3918
2	16	0.1296	0.7101	0.8396	13	0.1613	0.6258	0.7871
4	16	0.0636	0.3706	0.4342	13	0.1170	0.3474	0.4643
8	20	0.0311	0.2276	0.2587	14	0.0789	0.2046	0.2834
16	22	0.0157	0.1330	0.1488	15	0.0579	0.1281	0.1860
32	23	0.0081	0.0777	0.0858	14	0.0528	0.0828	0.1356
64	24	0.0045	0.0388	0.0433	14	0.0446	0.0610	0.1056
	2LH-post, Block-Jacobi with ILU				2LH-post, Block-Jacobi with LU			
NP	It	Setup	Solve	Total	It	Setup	Solve	Total
1	6	1.5660	1.0760	2.6420	7	4.4300	1.5160	5.9470
2	6	0.8857	0.5305	1.4160	7	1.3360	0.8590	2.1950
4	6	0.4909	0.2881	0.7790	6	0.5737	0.3558	0.9295
8	7	0.2924	0.1876	0.4800	6	0.3207	0.1761	0.4968
16	7	0.1837	0.1129	0.2966	6	0.1915	0.0990	0.2905
32	7	0.1372	0.0814	0.2186	7	0.1412	0.0804	0.2215
64	7	0.1194	0.0626	0.1820	7	0.1213	0.0613	0.1826

Table 1: Iteration numbers and execution times, in seconds, for kivap1.

kivap2								
	RAS, overlap 0				RAS, overlap 1			
NP	It	Setup	Solve	Total	It	Setup	Solve	Total
1	38	0.1250	1.6560	1.7810	38	0.1251	1.7060	1.8310
2	55	0.0610	1.1490	1.2100	44	0.0762	1.0030	1.0790
4	55	0.0292	0.5909	0.6201	45	0.0639	0.5913	0.6552
8	73	0.0145	0.3982	0.4127	66	0.0394	0.4719	0.5114
16	87	0.0071	0.2476	0.2547	65	0.0294	0.2768	0.3061
32	95	0.0034	0.1104	0.1139	77	0.0253	0.2060	0.2313
64	126	0.0018	0.1116	0.1134	95	0.0216	0.1550	0.1766
	2LH-post, Block-Jacobi with ILU				2LH-post, Block-Jacobi with LU			
NP	It	Setup	Solve	Total	It	Setup	Solve	Total
1	19	0.7097	1.4400	2.1500	12	1.0910	1.0290	2.1210
2	20	0.3922	0.8013	1.1940	16	0.4615	0.8029	1.2640
4	20	0.2460	0.4728	0.7188	16	0.2623	0.3973	0.6596
8	25	0.1364	0.3271	0.4634	19	0.1415	0.2542	0.3958
16	26	0.0875	0.2142	0.3017	24	0.0908	0.1950	0.2858
32	29	0.0669	0.1649	0.2318	29	0.0693	0.1664	0.2357
64	37	0.0550	0.1713	0.2263	40	0.0552	0.1864	0.2416

Table 2: Iteration numbers and execution times, in seconds, for kivap2.

therm2D								
	RAS, overlap 0				RAS, overlap 1			
NP	It	Setup	Solve	Total	It	Setup	Solve	Total
1	614	0.2193	172.9000	173.1000	614	0.2212	187.5000	187.7000
2	749	0.1183	108.5000	108.7000	834	0.1229	130.0000	130.1000
4	775	0.0621	54.7900	54.8500	970	0.0682	75.0900	75.1600
8	751	0.0341	26.9400	26.9700	741	0.0410	29.4700	29.5100
16	884	0.0214	15.4100	15.4300	783	0.0285	15.2000	15.2300
32	778	0.0141	6.7300	6.7440	680	0.0220	6.7170	6.7390
64	906	0.0113	5.0490	5.0600	812	0.0225	5.6390	5.6610
	2LH-post, Block-Jacobi with ILU				2LH-post, Block-Jacobi with LU			
NP	It	Setup	Solve	Total	It	Setup	Solve	Total
1	183	3.1180	95.8900	99.0100	5	10.2500	3.8210	14.0700
2	182	1.8420	59.2900	61.1300	18	4.0710	12.8300	16.9000
4	190	0.9694	30.6400	31.6100	26	1.8180	8.6890	10.5100
8	184	0.4858	15.0500	15.5300	34	0.7927	5.3530	6.1460
16	171	0.2641	6.8410	7.1050	57	0.3766	3.9230	4.2990
32	175	0.1621	3.4830	3.6460	76	0.2042	2.1980	2.4020
64	176	0.1165	2.4460	2.5620	106	0.1343	1.6330	1.7680

Table 3: Iteration numbers and execution times, in seconds, for therm2D.

therm3D								
	RAS, overlap 0				RAS, overlap 1			
NP	It	Setup	Solve	Total	It	Setup	Solve	Total
1	56	0.4348	27.2500	27.6800	56	0.4259	29.8300	30.2500
2	63	0.2237	15.7600	15.9900	55	0.2484	15.0400	15.2900
4	57	0.1270	10.3400	10.4700	61	0.1770	12.2400	12.4200
8	66	0.0773	6.3240	6.4010	57	0.1327	6.2150	6.3470
16	63	0.0383	2.1370	2.1760	58	0.1014	2.4040	2.5050
32	70	0.0247	1.2200	1.2440	59	0.0895	1.3940	1.4830
64	95	0.0185	0.9156	0.9341	71	0.0874	1.0920	1.1800
	2LH-post, Block-Jacobi with ILU				2LH-post, Block-Jacobi with LU			
NP	It	Setup	Solve	Total	It	Setup	Solve	Total
1	16	8.1740	16.0600	24.2300	–	——	——	——
2	16	4.9520	11.1000	16.0500	–	——	——	——
4	16	2.9370	7.8240	10.7600	7	158.8000	22.9800	181.8000
8	16	1.7560	4.3310	6.0870	8	44.1400	10.8800	55.0200
16	16	0.8118	1.8100	2.6220	10	10.2800	4.9550	15.2400
32	16	0.5323	1.0990	1.6310	13	2.4140	2.3730	4.7870
64	19	0.3962	0.8329	1.2290	17	0.8027	1.3300	2.1330

Table 4: Iteration numbers and execution times, in seconds, for therm3D.

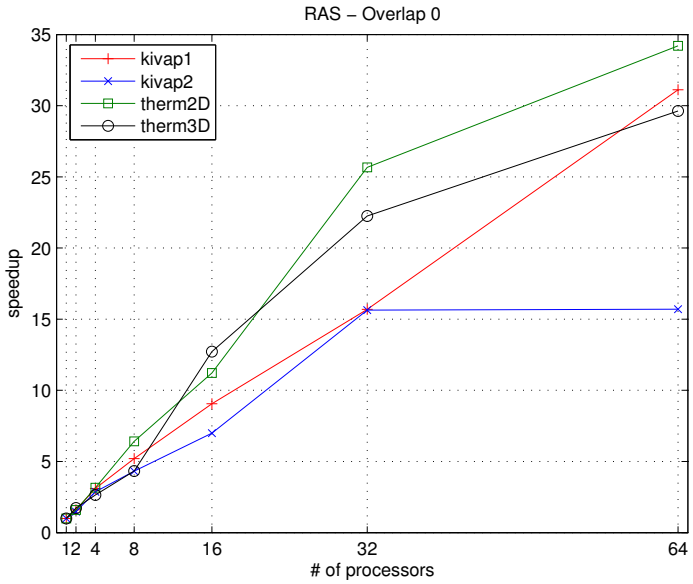

Figure 5: Speedups for RAS with overlap 0.

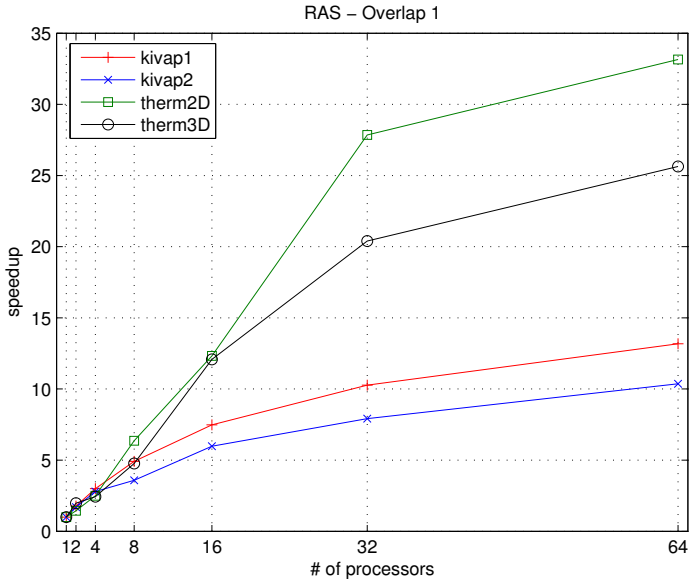

Figure 6: Speedups for RAS with overlap 1.

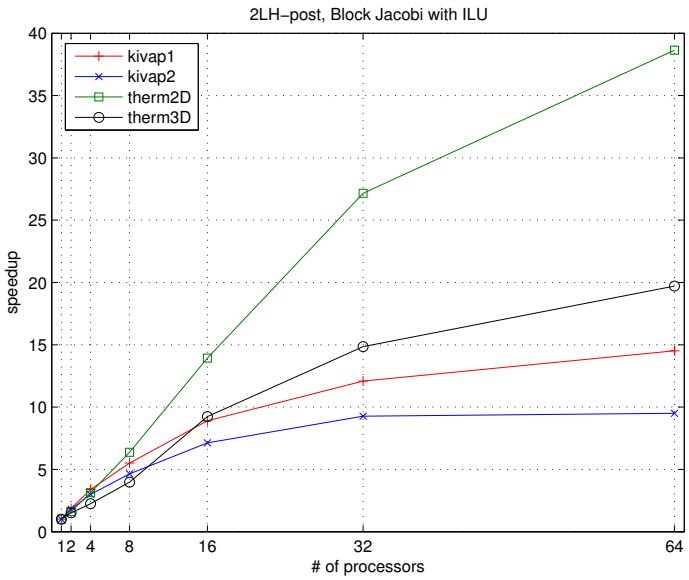

Figure 7: Speedups for 2LH-post with four Block-Jacobi sweeps and ILU factorization of the blocks.

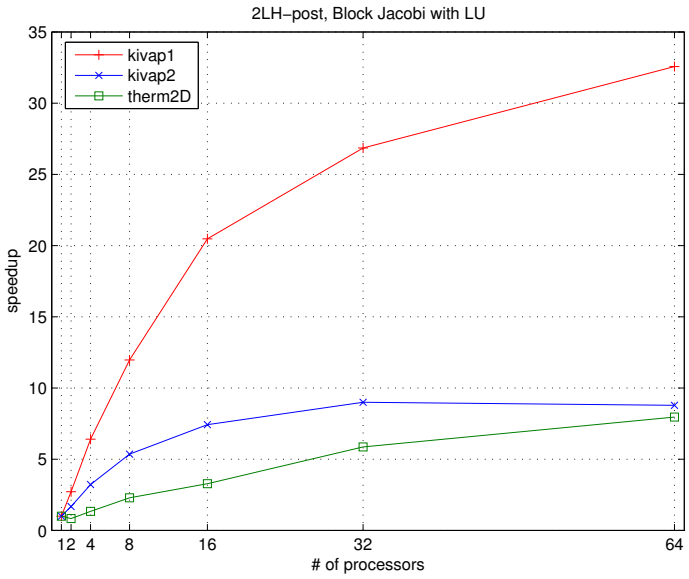

Figure 8: Speedups for 2LH-post with four Block-Jacobi sweeps and LU factorization of the blocks.

References

[1] R.E. Bank and C.C. Douglas, *SMMP: Sparse Matrix Multiplication Package*, Advances in Computational Mathematics, 1993, 1, 127-137. (See also `http://www.mgnet.org/ douglas/ccd-codes.html`)

[2] 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.

[3] 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.

[4] A. Buttari, P. D'Ambra, D. di Serafino and S. Filippone, *2LEVDD-PSBLAS User's Guide*, in preparation.

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

[6] X. C. Cai and Y. Saad, *Overlapping Domain Decomposition Algorithms for General Sparse Matrices*, Numerical Linear Algebra with Applications, 3(3), pp. 221–237, 1996.

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

[8] 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.

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

[10] P. D'Ambra, D. di Serafino and S. Filippone, On the Development of PSBLAS-based Parallel Two-level Schwarz Preconditioners, Peprint 1-05, Department of Mathematics, Second University of Naples, Italy, 2005.

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

[12] 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), pp. 720–755, 1999.

[13] 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).

[14] 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.

[15] 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.

[16] J. H. Ferziger, M. Perić, *Computational Methods for Fluid Dynamics*, Springer, third edition, 2002.

[17] 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.

[18] 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.

[19] C. W. Hirt, A. A. Amsden and J. L. Cook, *An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds*, Journal of Computational Physics, 14, pp. 227–253, 1974.

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

[21] 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.

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

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