Extending PSBLAS to Build Parallel Schwarz Preconditioners

Alfredo Buttari[1], Pasqua D'Ambra[2],
Daniela di Serafino[3], and Salvatore Filippone[1]

[1] Department of Mechanical Engineering, University of Rome "Tor Vergata"
Viale del Politecnico, I-00133, Rome, Italy
{alfredo.buttari,salvatore.filippone}@uniroma2.it
[2] Institute for High-Performance Computing and Networking, CNR
Via Pietro Castellino 111, I-80131 Naples, Italy
pasqua.dambra@na.icar.cnr.it
[3] Department of Mathematics, Second University of Naples
Via Vivaldi 43, I-81100 Caserta, Italy
daniela.diserafino@unina2.it

Abstract. We describe some extensions to Parallel Sparse BLAS (PSBLAS), a library of routines providing basic Linear Algebra operations needed to build iterative sparse linear system solvers on distributed-memory parallel computers. We focus on the implementation of parallel Additive Schwarz preconditioners, widely used in the solution of linear systems arising from a variety of applications. We report a performance analysis of these PSBLAS-based preconditioners on test cases arising from automotive engine simulations. We also make a comparison with equivalent software from the well-known PETSc library.

1 Introduction

Effective numerical simulations in many application fields, such as Computational Fluid Dynamics, require fast and reliable numerical software to perform Sparse Linear Algebra computations.

The PSBLAS library was designed to provide the kernels needed to build iterative methods for the solution of sparse linear systems on distributed-memory parallel computers [10]. It includes parallel versions of most of the Sparse BLAS *computational kernels* proposed in [7] and a set of *auxiliary routines* for the creation and management of distributed sparse matrix structures. The library provides also *application routines* implementing some sparse Krylov solvers with Jacobi and block Jacobi preconditioners. The library routines have been proved to be powerful and flexible in restructuring a complex CFD code, improving the accuracy and efficiency in the solution of the sparse linear systems and implementing the boundary data exchanges arising in the numerical simulation process [11]. PSBLAS is based on a SPMD programming model and uses the Basic Linear Algebra Communication Subprograms (BLACS) [6] to manage interprocess data exchange. The library is internally written in C and Fortran 77, but provides high-level routine interfaces in Fortran 95. Current development is devoted to extending PSBLAS with preconditioners suitable for fluid flow problems in automotive engine

J. Dongarra, K. Madsen, and J. Waśniewski (Eds.): PARA 2004, LNCS 3732, pp. 593–602, 2006.

modeling. A basic development guideline is to preserve as much as possible the original PSBLAS software architecture while moving towards the new SBLAS standard [8]. In this work we focus on the extension for building Additive Schwarz preconditioners, widely used in the parallel iterative solution of linear systems arising from PDE discretizations.

The paper is organized as follows. In Section 2 we give a brief description of Additive Schwarz preconditioners. In Section 3 we discuss how the set of PSBLAS data structures and routines has been extended to implement these preconditioners by reusing existing PSBLAS computational routines. In Section 4 we present the results of experiments devoted to analyze the performance of PSBLAS-based Schwarz preconditioners with Krylov solvers on test matrices arising from automotive engine simulations; we also show the results of comparisons with the widely used PETSc library [1]. Finally, in Section 5, we draw some conclusions and future work.

2 An Overview of Additive Schwarz Preconditioners

Schwarz preconditioners are based on domain decomposition ideas originally introduced in the context of variational solution of Partial Differential Equations (see [5,15]). We focus on the algebraic formulation of Additive Schwarz preconditioners for the solution of general sparse linear systems [2,14]; these are widely used with parallel Krylov subspace solvers.

Given the linear system $Ax = b$, where $A = (a_{ij}) \in \Re^{n\times n}$ is a nonsingular sparse matrix with a symmetric non-zero pattern, let $G = (W, E)$ be the adjacency graph of A, where $W = \{1, 2, \ldots, n\}$ and $E = \{(i, j) : a_{ij} \neq 0\}$ are the vertex set and the edge set, respectively. Two vertices are *neighbours* if an edge connects them. A 0-*overlap* partition of W is just an ordinary partition of the graph, i.e. a set of m disjoint nonempty subsets $W_i^0 \subset W$ such that $\cup_{i=1}^m W_i^0 = W$. A δ-*overlap* partition of W with $\delta > 0$ can be defined recursively by considering the sets $W_i^\delta \supset W_i^{\delta-1}$ obtained by including the vertices that are neighbours of the vertices in $W_i^{\delta-1}$. Let n_i^δ be the size of W_i^δ and R_i^δ the $n_i^\delta \times n$ matrix formed by the row vectors e_j^T of the $n \times n$ identity matrix, with $j \in W_i^\delta$. For each $v \in \Re^n$, $R_i^\delta v$ is the vector containing the components of v corresponding to the vertices in W_i^δ, hence R_i^δ can be viewed as a restriction operator from $\Re^n$ to $\Re^{n_i^\delta}$. Likewise, the transpose matrix $(R_i^\delta)^T$ can be viewed as a prolongation operator from $\Re^{n_i^\delta}$ to $\Re^n$. The *Additive Schwarz (AS)* preconditioner, M_{AS}, is then defined by

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

where the $n_i^\delta \times n_i^\delta$ matrix $A_i^\delta = R_i^\delta A (R_i^\delta)^T$ is obtained by considering the rows and columns of A corresponding to the vertices in W_i^δ.

When $\delta = 0$ M_{AS} is the well-known block Jacobi preconditioner. The convergence theory for the AS preconditioner is well developed in the case of symmetric positive definite matrices (see [5,15] and references therein). Roughly speaking, when the AS preconditioner is 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. Theoretical results are available also in the case of nonsymmetric and indefinite problems (see [4,15]).

Recently some variants of the AS preconditioner have been developed that outperform the classical AS for a large class of matrices, in terms of both convergence rate and of computation and communication time on parallel distributed-memory computers [3,9,12]. In particular, the *Restricted Additive Schwarz (RAS)* preconditioner, M_{RAS}, and the *Additive Schwarz* preconditioner *with Harmonic extension (ASH)*, M_{ASH}, are defined by

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

where $\tilde{R}_i^0$ is the $n_i^\delta \times n$ matrix obtained by zeroing the rows of R_i^δ corresponding to the vertices in $W_i^\delta \backslash W_i^0$. The application of the AS preconditioner requires the solution of m independent linear systems of the form

$$A_i^\delta w_i = R_i^\delta v \tag{1}$$

and then the computation of the sum

$$\sum_{i=1}^{m} (R_i^\delta)^T w_i. \tag{2}$$

In the RAS preconditioner R_i^δ in (2) is replaced by $\tilde{R}_i^0$; hence, in a parallel implementation where each processor holds the rows of A with indices in W_i^0 and the corresponding components of right-hand side and solution vectors, this sum does not require any communication. Analogously, in the ASH preconditioner R_i^δ in equation (1) is replaced by $\tilde{R}_i^0$; therefore, the computation of the right-hand side does not involve any data exchange among processors.

In the applications, the exact solution of system (1) is often prohibitively expensive. Thus, it is customary to substitute the matrix $(A_i^\delta)^{-1}$ with an approximation $(K_i^\delta)^{-1}$, computed by incomplete factorizations, such as ILU, or by iterative methods, such as SSOR or Multigrid (see [5]).

3 Building and Applying AS Preconditioners in PSBLAS

We now review the basic operations involved in the Additive Schwarz preconditioners from the point of view of parallel implementation through PSBLAS routines. We begin by drawing a distinction between

Preconditioner Setup: the set of basic operations needed to identify W_i^δ, to build A_i^δ from A, and to compute K_i^δ from A_i^δ;

Preconditioner Application: the set of basic operations needed to apply the restriction operator R_i^δ to a given vector v, to compute (an approximation of) w_i, by applying $(K_i^\delta)^{-1}$ to the restricted vector, and, finally, to obtain sum (2).

Before showing how PSBLAS has been extended to implement the Additive Schwarz preconditioners described in Section 2, we briefly describe the main data structures involved in the PSBLAS library. A general row-block distribution of sparse matrices is supported by PSBLAS, with conformal distribution of dense vectors.

Sparse Matrix: Fortran 95 derived data type, called `D_SPMAT`, it includes all the information about the local part of a distributed sparse matrix and its storage mode, following the Fortran 95 implementation of a sparse matrix in the sparse BLAS standard of [7].

Communication Descriptor: Fortran 95 derived data type `DESC_TYPE`; it contains a representation of the sets of indices involved in the parallel data distribution, including the 1-overlap indices, i.e. the set $W_i^1 \backslash W_i^0$, that is preprocessed for the data exchange needed in sparse matrix-vector products.

Existing PSBLAS computational routines implement the operations needed for the application phase of AS preconditioners, provided that a representation of the δ-partition is built and packaged into a new suitable data structure during the phase of preconditioner setup. The next two sections are devoted to these issues.

3.1 PSBLAS Implementation of Preconditioner Application

To compute the right-hand side in (1) the restriction operator R_i^δ must be applied to a vector v distributed among parallel processes conformally to the sparse matrix A. On each process the action of R_i^δ corresponds to gathering the entries of v with indices belonging to the set $W_i^\delta \backslash W_i^0$. This is the semantics of the PSBLAS `F90_PSHALO` routine, which updates the halo components of a vector, i.e. the components corresponding to the 1-overlap indices. The same code can apply an arbitrary δ-overlap operator, if a suitable auxiliary descriptor data structure is provided.

Likewise, the computation of sum (2) can be implemented through a suitable call to the PSBLAS computational routine `F90_PSOVRL`; this routine can compute either the sum, the average or the square root of the average of the vector entries that are replicated in different processes according to an appropriate descriptor.

Finally, the computation of $(K_i^\delta)^{-1} v_i^\delta$, where $v_i^\delta = R_i^\delta v$ or $v_i^\delta = \tilde{R}_i^0 v$, can be accomplished by two calls to the sparse block triangular solve routine `F90_PSSPSM`, given a local (incomplete) factorization of A_i^δ.

Therefore, the functionalities needed to implement the application phase of the AS, RAS and ASH preconditioners, in the routine `F90_ASMAPPLY`, are provided by existing computational PSBLAS routines, if an auxiliary descriptor `DESC_DATA` is built. Thus, the main effort in implementing the preconditioners lies in the definition of a preconditioner data structure and of routines for the preconditioner setup phase, as discussed in Section 3.2.

3.2 PSBLAS Implementation of Preconditioner Setup

To implement the application of AS preconditioners we defined a data structure `PREC_DATA` that includes in a single entity all the items involved in the application of the preconditioner:

```
TYPE PREC_DATA
INTEGER                          :: PREC
INTEGER                          :: N_OVR
TYPE(D_SPMAT)                    :: L, U
REAL(KIND(1.D0)), POINTER :: D(:)
TYPE(DESC_TYPE)     :: DESC_DATA
END TYPE PREC_DATA
```

Fig. 1. Preconditioner data type

- a preconditioner identifier, PREC, and the number δ of overlap layers, N_OVR;
- two sparse matrices L and U, holding the lower and upper triangular factors of K_i^δ (the diagonal of the upper factor is stored in a separate array, D);
- the auxiliary descriptor DESC_DATA, built from the sparse matrix A, according to N_OVR.

This data structure has the Fortran 95 definition shown in Figure 1. Note that the sparse matrix descriptor is kept separate from the preconditioner data; with this choice the sparse matrix operations needed to implement a Krylov solver are independent of the choice of the preconditioner.

The algorithm to setup an instance P of the PREC_DATA structure for AS, RAS or ASH, with overlap N_OVR, is outlined in Figure 2. By definition the submatrices A_i^0 identify the vertices in W_i^1; the relevant indices are stored into the initial communication descriptor. Given the set W_i^1, we may request the column indices of the non-zero entries in the rows corresponding to $W_i^1 \backslash W_i^0$; these in turn identify the set W_i^2, and so on. All the communication is performed in the steps 4.2 and 6, while the other steps are performed locally by each process. A new auxiliary routine, F90_DCSRSETUP, has been developed to execute the steps 1–6. To compute the triangular factors of K_i^δ (step 7), the existing PSBLAS routine F90_DCSRLU, performing an ILU(0) factorization of A_i^δ, is currently used. The two previous routines have been wrapped into a single PSBLAS application routine, named F90_ASMBUILD.

It would be possible to build the matrices A_i^δ while building the auxiliary descriptor DESC_DATA. Instead, we separated the two phases, thus providing the capability to reuse the DESC_DATA component of an already computed preconditioner; this allows efficient handling of common application situations where we have to solve multiple linear systems with the same structure.

4 Numerical Experiments

The PSBLAS-based Additive Schwarz preconditioners, coupled with a BiCGSTAB Krylov solver built on the top of PSBLAS, were tested on a set of matrices from automotive engine simulations.

These matrices arise from the pressure correction equation in the implicit phase of a semi-implicit algorithm (ICED-ALE [13]) for the solution of unsteady compressible Navier-Stokes equations, implemented in the KIVA application software, as modified

1. Initialize the descriptor `P%PREC_DATA` by copying the matrix descriptor `DESC_A`.
2. Initialize the overlap level: $\eta = 0$.
3. Initialize the local vertex set, $W_i^\eta = W_i^0$, based on the current descriptor.
4. **While (** $\eta <$ `N_OVR` **)**
   4.1 Increase the overlap: $\eta = \eta + 1$.
   4.2 Build W_i^η from $W_i^{\eta-1}$, by adding the halo indices of $A_i^{\eta-1}$, and exchange with other processors the column indices of the non-zero entries in the rows corresponding to $W_i^\eta \backslash W_i^{\eta-1}$.
   4.3 Compute the halo indices of A_i^η and store them into `P%DESC_DATA`.
   Endwhile
5. **If (** `N_OVR` > 0 **)** Optimize the descriptor `P%DESC_DATA` and store it in its final format.
6. Build the enlarged matrix A_i^δ, by exchanging rows with other processors.
7. Compute the triangular factors of the approximation K_i^δ of A_i^δ.

Fig. 2. Preconditioner setup algorithm

kivap1 ($n = 86304,\ nnz = 1575568$) *kivap2* ($n = 56904,\ nnz = 1028800$)

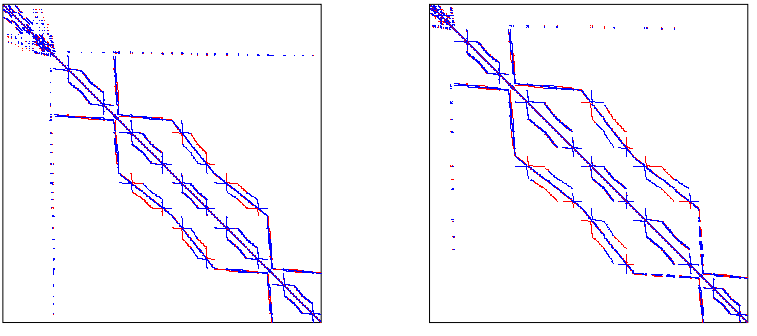

Fig. 3. Sparsity patterns of the test matrices

in [11]. The test case is a simulation of a commercial direct injection diesel engine featuring a bowl shaped piston, with a mesh containing approximately 100000 control volumes; during the simulation mesh layers are activated/deactivated following the piston movement. We show measurements for two matrices, *kivap1* and *kivap2*, corresponding to two different simulation stages. They have sizes 86304 and 56904, respectively, with symmetric sparsity patterns and up to 19 non-zeroes per row (see Figure 3).

Numerical experiments were carried out on a Linux cluster, with 16 PCs connected via a Fast Ethernet switch, at the Department of Mathematics of the Second University of Naples. Each PC has a 600 MHz Pentium III processor, a memory of 256 MB and an L2 cache of 256 KB. PSBLAS was installed on the top of the BLAS reference implementation, BLACS 1.1 and mpich 1.2.5, using the gcc C compiler, version 2.96, and the Intel Fortran compiler, version 7.1.

All the tests were performed using a row-block distribution of the matrices. Right-hand side and solution vectors were distibuted conformally. The number m of vertex sets

Table 1. Iteration counts of RAS-preconditioned BiCGSTAB

np	*kivap1*								*kivap2*							
	PSBLAS				*PETSc*				*PSBLAS*				*PETSc*			
	δ=0	δ=1	δ=2	δ=4	δ=0	δ=1	δ=2	δ=4	δ=0	δ=1	δ=2	δ=4	δ=0	δ=1	δ=2	δ=4
1	12	12	12	12	12	12	12	12	33	33	33	33	33	33	33	33
2	16	13	11	12	16	12	12	12	50	34	33	34	46	34	32	32
4	16	13	12	13	16	12	11	12	47	36	34	34	50	35	32	31
6	18	13	13	12	18	12	12	12	48	37	38	34	52	35	34	29
8	20	14	14	12	20	12	13	12	50	37	37	35	53	36	32	32
10	19	13	12	13	19	13	12	12	54	38	35	34	51	33	32	33
12	21	14	12	12	21	13	13	12	54	34	38	33	53	35	32	33
14	20	14	13	14	20	12	13	12	51	36	38	33	52	35	32	32
16	22	15	14	13	22	13	12	12	58	37	38	36	59	38	32	30

W_i^δ used by the preconditioner was set equal to the number of processors; the right-hand side and the initial approximation of the solution of each linear system were set equal to the unit vector and the null vector, respectively. The preconditioners were applied as right preconditioners. The BiCGSTAB iterations were stopped when the 2-norm of the residual was reduced by a factor of 10^{-6} with respect to the initial residual, with a maximum of 500 iterations.

We also compared the PSBLAS implementation of the RAS-preconditioned BiCGSTAB solver with the corresponding implementation provided by the well-known PETSc library [1], on the same test problems and with the same stopping criterion. The experiments were performed using PETSc 2.1.6, compiled with gcc 2.96 and installed on the top of mpich 1.2.5 and of the BLAS and LAPACK implementations provided with the Linux Red Hat 7.1 distribution.

We report performance results with RAS; this was in general the most effective Additive Schwarz variant, in accordance with the literature cited in Section 2. In Table 1 we show the iteration counts of RAS-preconditioned BiCGSTAB for PSBLAS and PETSc on both test problems, varying the number np of processors and the overlap δ. We see that the number of iterations significantly decreases in passing from a 0-overlap to a 1-overlap partition, especially for *kivap2*, while it does not have relevant changes with a further growth of the overlap. We note also that, for $\delta > 0$, the number of iterations is almost stable as the number of processes increases. This behaviour appears to be in agreement with the sparsity pattern of the test matrices. The number of iterations of PSBLAS is generally comparable with the one of PETSc. For *kivap2*, in a few cases a difference of 5 or 6 iterations is observed, which is due to some differences in the row-block distribution of the matrix and in the row ordering of the enlarged matrices A_i^δ used by the two solvers.

In Figure 4 we show the execution times of the RAS-preconditioned BiCGSTAB on the selected test cases, varying the number of processors. These times include also the preconditioner setup times. As expected, the time usually decreases as the number of processors increases; a slight time increase can be observed in a few cases, which can be mainly attributed to the row-block distribution of the matrices. A more regular

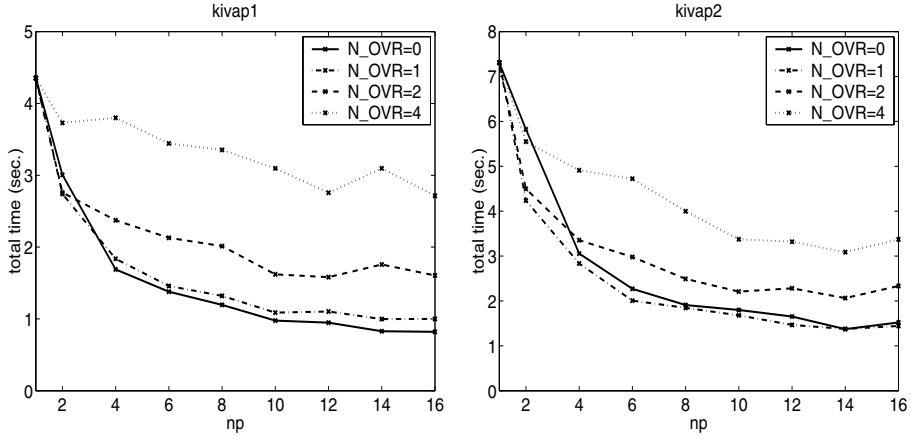

Fig. 4. Execution times of PSBLAS RAS-preconditioned BiCGSTAB

behaviour is expected by applying suitable graph partitioning algorithms to the initial data distribution. For *kivap1* the times are generally lower for overlap 0. Indeed, the small decrease in the number of BiCGSTAB iterations, as the overlap grows, is not able to balance the cost of the preconditioner setup phase. For *kivap2*, the reduction of the number of iterations obtained with overlap 1 is able to compensate the setup cost, thus leading to the smallest execution times.

Finally, in Figure 5 we compare the execution times of PSBLAS and PETSc. The performance of the two solvers is generally comparable. PSBLAS is always faster on a small number of processors, whereas for higher levels of overlap (2 and 4) PETSc requires less execution time as the number of processors increases. A more detailed analysis has shown that this behaviour is due to the smaller preconditioner setup time of PETSc. This issue is currently under investigation, taking into account the different choices implemented by PSBLAS and PETSc in the setup of the preconditioners.

5 Conclusions and Future Work

We presented some results of an ongoing activity devoted to updating and extending PSBLAS, a parallel library providing basic Linear Algebra operations needed to build iterative sparse linear system solvers on distributed-memory parallel computers. Motivations for our work come from the flexibility and effectiveness shown by PSBLAS in restructuring and parallelizing a legacy CFD code [11], still widely used in the automotive engine application world, and also from the appearance of a new proposal for the serial Sparse BLAS standard [8].

In this paper we focused on the extension of PSBLAS to implement different versions of Additive Schwarz preconditioners. On test problems from automotive engine simulations the preconditioners showed performances comparable with those of other well-established software. We are currently working on design and implementation issues concerning the addition of a coarse-grid solution step to the basic Additive Schwarz preconditioners, in order to build a two-level Schwarz preconditioning module.

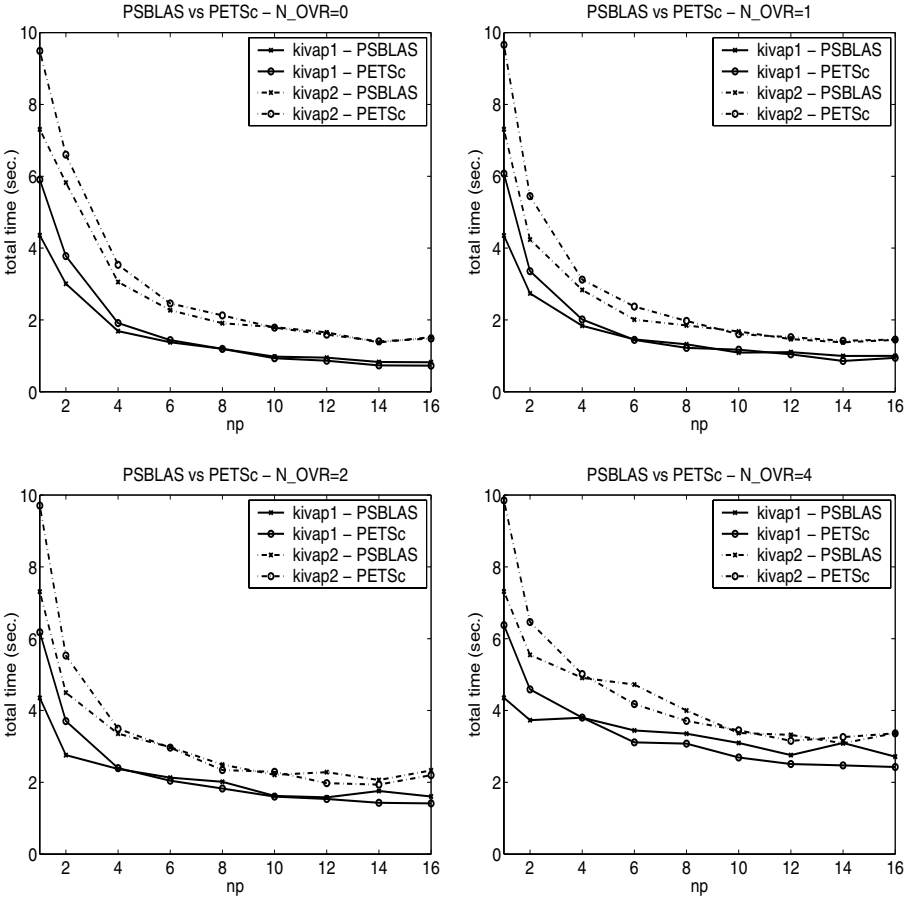

Fig. 5. Comparison of execution times of the PSBLAS and PETSc

References

1. S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. F. Smith, H. Zhang. PETSc Users Manual. Tech. Rep. ANL-95/11, Revision 2.1.6, Argonne National Laboratory, August 2003.
2. X. C. Cai, Y. Saad. Overlapping Domain Decomposition Algorithms for General Sparse Matrices. *Num. Lin. Alg. with Applics.*, 3(3):221–237, 1996.
3. X.C. Cai, M. Sarkis. A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems. *SIAM J. Sci. Comput.*, 21(2):792–797, 1999.
4. X.C. Cai, O. B. Widlund. Domain Decomposition Algorithms for Indefinite Elliptic Problems. *SIAM J. Sci. Stat. Comput.*, 13(1):243–258, 1992.
5. T. Chan, T. Mathew. Domain Decomposition Algorithms. In A. Iserles, editor, *Acta Numerica 1994*, pages 61–143, 1994. Cambridge University Press.
6. J. J. Dongarra, 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, C. Vittoli. Level 3 Basic Linear Algebra Subprograms for Sparse Matrices: a User Level Interface. *ACM Trans. Math. Softw.*, 23(3):379–401, 1997.
8. I. Duff, M. Heroux, R. Pozo. An Overview of the Sparse Basic Linear Algebra Subprograms: the New Standard from the BLAS Technical Forum. *ACM Trans. Math. Softw.*, 28(2):239–267, 2002.
9. E. Efstathiou, J. G. Gander. Why Restricted Additive Schwarz Converges Faster than Additive Schwarz. *BIT Numerical Mathematics*, 43:945–959, 2003.
10. S. Filippone, M. Colajanni. PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices. *ACM Trans. Math. Softw.*, 26(4):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*, pages 441–448, 2002. Imperial College Press.
12. A. Frommer, D. B. Szyld. An Algebraic Convergence Theory for Restricted Additive Schwarz Methods Using Weighted Max Norms. *SIAM J. Num. Anal.*, 39(2):463–479, 2001.
13. C. W. Hirt, A. A. Amsden, J. L. Cook. An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds. *J. Comp. Phys.*, 14:227–253, 1974.
14. Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.
15. B. Smith, P. Bjorstad, W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.