Revista de Matemática: Teoría y Aplicaciones
ISSN: 1409-2433
mta.cimpa@ucr.ac.cr
Universidad de Costa Rica
Costa Rica

Salas-Huertas, Oscar; Marazzina, Daniele; Rovida, Sergio; Sacchi, Giovanni; Scacchi, Simone
The BPS preconditioner on Beowulf cluster
Revista de Matemática: Teoría y Aplicaciones, vol. 16, núm. 1, enero-junio, 2009, pp. 148-159
Universidad de Costa Rica
San José, Costa Rica

Revista de Matemática: Teoría y Aplicaciones 2009 **16**(1) : 148–158
cimpa – ucr issn: 1409-2433

THE BPS PRECONDITIONER ON BEOWULF CLUSTER

Oscar Salas-Huertas* Daniele Marazzina† Sergio Rovida‡
Giovanni Sacchi§ Simone Scacchi¶

Recibido/Received: 20 Feb 2008 — Aceptado/Accepted: 9 Oct 2008

Abstract

This work presents the implementation on a Linux Cluster of a parallel preconditioner for the solution of the linear system resulting from the finite element discretization of a 2D second order elliptic boundary value problem. The numerical method, proposed by Bramble, Pasciak and Schatz, is developed using Domain Decomposition techniques, which are based on the splitting of the computational domain into subregions of smaller size, enforcing suitable compatibility conditions. The Fortran code is implemented using PETSc: a suite of data structures and routines devoted to the scientific parallel computing and based on the MPI standard for all message-passing communications. The main interest of the paper is to present an efficient and portable code for the solution of large-scale linear systems and to investigate how the architectural aspects of the cluster influence the performance of the considered algorithm. We provide an analysis of the execution times as well as of the scalability, using as test case the classical Poisson equation with Dirichlet boundary conditions.

Keywords: Domain Decomposition, Parallelization, Partial Differential Equation, Preconditioner, Beowulf Cluster.

Resumen

En este trabajo se presenta una implementación para Cluster Linux de un precondicionador útil para resolver en forma eficiente sistemas lineales obtenidos de la discretización por medio de elementos finitos de problemas de valor inicial 2D elípticos

*Department of Mathematics, Universidad Nacional, Heredia, Costa Rica. E-Mail: oscar.salas@unipv.it.

†Department of Mathematics, Politecnico di Milano, Milan, Italy. E-Mail: daniele.marazzina@polimi.it.

‡IMATI-CNR, Pavia, Italy. E-Mail: sergio.rovida@imati.cnr.it.

§IMATI-CNR, Pavia, Italy. E-Mail: gianni.sacchi@imati.cnr.it.

¶Department of Mathematics, University of Milano, Milan, Italy. E-Mail: simone.scacchi@unimi.it.

de segundo orden. El método numérico implementado fue propuesto por Bramble, Pasciak and Schatz, y en él se utiliza la técnica de Descomposición de Dominio, la cual se basa en una división del dominio computacional en subregiones de dimensiones siempre más pequeñas, las cuales cumplen con condiciones apropiadas de compactibilidad. El código fue implementado en Fortran usando la librería PETSC: una colección de estructuras y funciones, desarrolladas para el Cálculo Científico en Paralelo y basada en el estándar MPI para administrar la comunicación y el cambio de mensajes. Nuestro objetivo en este trabajo es demostrar la eficiencia y portabilidad del código cuando se emplea en la solución de grandes sistemas y además analizar cuál es la influencia que tiene la arquitectura del cluster en las prestaciones del algoritmo considerado. Nosotros presentamos una análisis de los tiempos de ejecución obtenidos así como de la escalabilidad, usando como problema test la ecuación clásica de Poisson con condiciones de Dirichlet en la frontera.

Palabras clave: Descomposición de Dominio, Paralelización, Ecuaciones a las Derivadas Parciales, Precondicionador, Beowulf Cluster.

Mathematics Subject Classification: 65Y05.

1 Introduction

The "grand challenge" problems and the development in the last two decades of parallel computing platforms have determined a considerable increase of interest in Domain Decompositions methods (DD), which offer the possibility to exploit their intrinsic mathematical parallelism in a very natural manner. They are flexible and efficient methods, i.e. they provide a localized treatment of complex geometries and, in general, optimal convergence rates. The main idea of DD methods is to split a differential problem stated on a computational domain into coupled subproblems stated on smaller and simpler subdomains forming a partition of the original domain. Much of the work in Domain Decomposition relates to the selection of subproblems in order to build a fast iterative procedure to solve the original problem: this means that DD methods provide efficient and scalable preconditioners (for further details see [7]).

The BPS (Bramble, Pasciak and Schatz) algorithm, proposed in [2], provides an optimal preconditioner for the linear systems arising from the finite element discretization of two dimensional second order elliptic boundary value problems.

The aim of this work is to present a parallel implementation on Beowulf clusters of the BPS method, discussing both the implementation strategies and the performance of the code. Cluster is a widely-used term meaning independent computers, combined into a unified system through software and networking, and typically used for High Performance Computing (HPC) to provide greater computational power than a single computer. Beowulf Clusters are scalable performance clusters based on commodity hardware, on a private system network, with open source software (Linux) infrastructure.

Our implementation allows to use the code on a wide range of cluster platforms for solving large-scale linear systems, avoiding the need of more complex and difficult to program HPC architectures. The parallelization and the portability of the code are based on the PETSc library (see [1]).

2 The BPS preconditioner

We summarize the construction of the BPS preconditioner, choosing as model problem the Laplacian operator on a 2D polygonal region and the Conjugate Gradient method as iterative solver. A full description and analysis of the algorithm can be found in the original paper [2].

2.1 The model problem

Let $\Omega \subset \mathbb{R}^2$ be a polygonal domain and let $f \in L^2(\Omega)$ be a given function, we consider the classical model problem

$$-\Delta u = f \text{ in } \Omega, \quad u = 0 \text{ on } \partial\Omega. \tag{1}$$

A finite element approximation (see [4]) of this problem leads to the solution of the associated linear system

$$\boldsymbol{A}\,\boldsymbol{u} = \boldsymbol{b}. \tag{2}$$

Iterative methods are widely used to solve systems of linear equations coming from large-scale problems of engineering and scientific computing. Such methods are particularly suitable for large sparse matrices, like those arising in the finite element or finite difference approximations of partial differential equations. For the solution of (2), we choose the most popular iterative scheme: the Conjugate Gradient (CG) algorithm (see [6]).

In general the coefficient matrix $\boldsymbol{A}$ is not well-conditioned, so the application of the CG algorithm will not be a very efficient choice. In fact an iterative solver is much more efficient and competitive when it is associated with an appropriate preconditioner which improves the condition number of the matrix and provides a satisfactory convergence rate. Preconditioning techniques consist of choosing a positive-definite symmetric matrix $\boldsymbol{M}$ that approximate $\boldsymbol{A}$, but which is easier to invert. Thus the idea is to solve (2) by solving $\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{b}$.

We consider the BPS algorithm proposed in [2], that can be interpreted (see [3]) as a generalized block Jacobi preconditioner for the Schur system associated to (2). At each Preconditioned Conjugate Gradient (PCG) iteration, the preconditioning step is performed with a *matrix-free* approach, without fully assembling the matrix $\boldsymbol{M}$ or its inverse. The following sections describe the BPS preconditioner introducing its associated bilinear form.

2.2 The preconditioner

Let Ω be a rectangular domain, we consider a Cartesian grid Ω^H (coarse mesh, see Figure 1) and we denote with $\{\Omega_k\}_{k=1}^N$ its elements. We also denote with v_i the vertices and with Γ_{ij} the side of the grid with endpoints v_i and v_j. Furthermore we consider a structured triangulation $\mathcal{T}$ (fine mesh, see Figure 1) such that the sides of Ω^H are also mesh-lines of $\mathcal{T}$. The triangulation $\mathcal{T}_k := \{T \in \mathcal{T} \text{ s.t. } T \subset \Omega_k\}$ clearly is a structured triangulation of

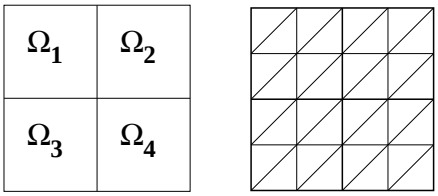

Figure 1: Coarse (left) and Fine (right) Mesh

Ω_k.

We now consider the following finite-dimensional spaces:

$$\begin{aligned} S_h &= \left\{ v_h \in H^1(\Omega) \text{ such that } v_{h|_T} \in \mathbb{P}^1(T) \ \forall\, T \in \mathcal{T} \right\}, \\ S_h^0 &= \left\{ v_h \in H_0^1(\Omega) \text{ such that } v_{h|_T} \in \mathbb{P}^1(T) \ \forall\, T \in \mathcal{T} \right\}, \end{aligned}$$

and, for each $k = 1, \cdots, N$, we define

$$\begin{aligned} S_h(\Omega_k) &= \left\{ v_h \in H^1(\Omega_k) \text{ such that } v_{h|_T} \in \mathbb{P}^1(T) \ \forall\, T \in \mathcal{T}_k \right\}, \\ S_h^0(\Omega_k) &= \left\{ v_h \in H_0^1(\Omega_k) \text{ such that } v_{h|_T} \in \mathbb{P}^1(T) \ \forall\, T \in \mathcal{T}_k \right\}, \end{aligned}$$

where $\mathbb{P}^1$ denotes the polynomial functions of degree 1.

The idea proposed in [2] is the following:

1. First of all we introduce the bilinear form $A_k(U,V) := \int_{\Omega_k} \nabla U \cdot \nabla V dx \ \forall U, V \in S_h^0$ and we decompose the functions in S_h^0 as follows:

$$W = W_p + W_h \quad \forall\, W \in S_h^0,$$

such that $W_p \in \oplus_{k=1}^N S_h^0(\Omega_k)$ and satisfies

$$A_k(W_p, \phi) = A_k(W, \phi), \quad \forall\, \phi \in S_h^0(\Omega_k), \quad k = 1, \cdots, N.$$

Notice that W_p is determined on Ω_k by the values of W on Ω_k. It is clear that

$$A_k(W_h, \phi) = 0, \qquad \forall\, \phi \in S_h^0(\Omega_k), \quad k = 1, \cdots, N.$$

Thus on each Ω_k, W is decomposed into a function W_p which vanishes on $\partial\Omega_k$ and a function $W_h \in S_h(\Omega_k)$ which satisfies the above homogeneous equations and has the same boundary values as W.

2. For all $k = 1, \cdots, N$, we decompose $W_h \in S_h(\Omega_k)$ into $W_h = W_e + W_v$, with $W_e = 0$ at all the vertices of Ω_k and $W_{v|_{\Gamma_{ij}}} \in \mathbb{P}^1(\Gamma_{ij}) \ \forall\, \Gamma_{ij} \in \partial\Omega_k$ with the same values as W at the vertices.

The bilinear form associated to the preconditioner $\boldsymbol{M}$ reads as follow

$$\begin{aligned} M(W,\phi) &= A(W_p, \phi_p) + 2 \sum_{\Gamma_{ij}} \int_{\Gamma_{ij}} \widetilde{l}_0^{-1/2} W_e \ \phi_e \ ds \\ &+ 2 \sum_{\Gamma_{ij}} (\ W_v(v_i) - W_v(v_j) \ )(\ \phi_v(v_i) - \phi_v(v_j) \ ), \end{aligned} \tag{3}$$

with $A(U,V) := \sum_{k=1}^{N} A_k(U,V)$ and $\int_{\Gamma_{ij}} \widetilde{l}_0 UV ds := \int_{\Gamma_{ij}} U'V' ds$, where the prime denotes differentiation with respect to arc length s along Γ_{ij}.

2.3 The algorithm

We outline the steps in order to solve the following problem:

$$\text{given } g \in L^2(\Omega),\ \text{find } W \in S_h^0 : \quad M(W,\phi) = \int_\Omega g\ \phi\ dx \quad \forall\ \phi\ \in\ S_h^0. \tag{4}$$

Due to the algorithm below, the action of $\boldsymbol{M}^{-1}$ (i.e., the value of $\boldsymbol{M}^{-1}\boldsymbol{g}$ for any given vector $\boldsymbol{g}$) could be evaluated, without forming explicitly the matrix $\boldsymbol{M}$ or its inverse.

STEP 1. If $\phi\ \in\ S_h^0(\Omega_k)$, then from (3) we obtain

$$M(W,\phi)\ =\ A_k(W_p,\phi_p)\ =\ A_k(W_p,\phi);$$

thus, if we consider (4), we can find W_p solving on each subdomain Ω_k the following homogeneous Dirichlet boundary problems

$$A_k(W_p,\phi)\ =\ \int_\Omega g\ \phi\ dx \quad \forall\ \phi\ \in\ S_h^0(\Omega_k),$$

for any $k\ =\ 1,\cdots,N$. These problems are independent and can be solved in parallel.

STEP 2. For any Γ_{ij}, we denote with $S_e(\Gamma_{ij})$ the space of functions of $S_h^0(\Omega)$ that vanish on the interior mesh points of every Ω_k, $k\ =\ 1,\ \cdots,N$, and on all the other edges Γ_{rs} and, in particular, at the endpoints of Γ_{ij}. If $\phi\ \in\ S_e(\Gamma_{ij})$, then

$$M(W,\phi)\ =\ A(W_p,\phi)\ +\ 2\ \int_{\Gamma_{ij}} \widetilde{l}_0^{1/2} W_e\ \phi_e\ ds,$$

thus we find $W_{e|_{\Gamma_{ij}}}$ by solving

$$2\ \int_{\Gamma_{ij}} \widetilde{l}_0^{1/2} W_e\ \phi_e\ ds\ =\ \int_\Omega g\ \phi\ dx - A(W_p,\phi) \qquad \forall \phi\ \in\ S_e(\Gamma_{ij}). \tag{5}$$

In order to solve these problems we do not need to compute the operator $\widetilde{l}_0^{1/2}$. In fact, assume that there are $n-2$ interior nodes on Γ_{ij} and that $\widetilde{\phi}_l$, $l = 1,\cdots,n-2$ are the nodal basis functions for $S_e(\Gamma_{ij})$, then the eigenvalues and eigenvectors of the matrix

$$\boldsymbol{S} = \{S_{lm}\}_{l,m=1}^{n-2} \quad \text{with}\ \ S_{lm} := \int_{\Gamma_{ij}} \widetilde{l}_0^{1/2} \widetilde{\phi}_l\ \widetilde{\phi}_m\ ds,$$

are well known (see [2]). Thus, if $\boldsymbol{\lambda}$ and $\boldsymbol{\Psi}$ are the eigenvalues and eigenvectors matrices, respectively, it holds $\boldsymbol{S}^{-1} = \boldsymbol{\Psi}^{-1}\boldsymbol{\lambda}^{-1}\boldsymbol{\Psi}$ and we can easily solve (5) and find W_e on Γ_{ij}.

The one-dimensional problems on each edge are independent and can be solved in parallel.

STEP 3. We define S_v as the space of functions of S_h^0 which are linear polynomials on each edge Γ_{ij} and vanish on the interior nodes of Ω_k, $k = 1, \cdots, N$. If $\phi \in S_v$, then

$$M(W, \phi) = A(W_p, \phi) + 2 \sum_{\Gamma_{ij}} (W_v(v_i) - W_v(v_j))(\phi_v(v_i) - \phi_v(v_j)).$$

Therefore we find W_v on $\bigcup \Gamma_{ij}$ by solving for all $\phi \in S_v$

$$2 \sum_{\Gamma_{ij}} (W_v(v_i) - W_v(v_j))(\phi_v(v_i) - \phi_v(v_j)) = \int_\Omega g \, \phi \, dx - A(W_p, \phi),$$

on the coarse mesh and then extending piecewise linearly to the edges.

This step and STEP 2 are independent, thus they could be done in parallel.

STEP 4. We find W_h by extending the values of $W_v + W_e$ on $\bigcup \Gamma_{ij}$ to the whole domain Ω solving

$$A_k(W_h, \phi) = 0 \ \ \forall \, \phi \in S_h^0(\Omega_k), \quad W_h = W_v + W_e \text{ on } \partial\Omega_k,$$

for any $k = 1, \cdots, N$.

As in STEP 1, the Dirichlet boundary problems above can be solved in parallel.

STEP 5. We conclude computing $W = W_p + W_h$.

It has been shown in [2] that the preconditioned system has a condition number

$$\mathcal{K} = O(1 + log^2(H/h)), \tag{6}$$

where H and h are the of the coarse mesh and the fine mesh, respectively. Therefore, the BPS preconditioner is appropriate to solve large systems of equations on massively parallel architectures, since the condition number depends weakly on the mesh spacing and on the number of processors (see [3]).

3 Parallel implementation

We implement the BPS algorithm in Fortran90, using the PETSc library (see [1]), from Argonne National Laboratory. This library, based on MPI, BLAS and LAPACK, offers advanced data structures and routines well suited for parallel codes, from simple parallel matrix and vector assembly routines, that allow the overlap of communication and computation, to more complex linear and nonlinear equation solvers. The choice for the numerical experiments of structured meshes allows us to take full advantage of the Distributed Arrays PETSc objects, which provide data structures suitable for the management of communication between neighboring processors.

The algorithm massively involves scalar products and matrix-vector products. It is therefore clear that the effectiveness of the code strongly depends on the way these algebraic computations are carried out. For this reason and to benefit from the parallel structure of the algorithm, we used PETSc library that optimally manages both the communication among processors and the algebraic computational kernels [1].

We remind that the adopted strategy is to assign one processor to each subdomain. The algorithm shows a high degree of parallelism, which can be described more in detail as follows (see also Figure 2):

STEP 1 exhibits a parallelism of degree N equal to the number of subdomains. Each processor solves one Dirichlet problem, corresponding to the solution of a linear system with size equal to the number of the internal nodes in the related subdomain.

STEPS 2-3: STEP 2 shows a degree of parallelism equal to the number of the internal edges. Each processor solves the problem related to the internal north and east edges of the subdomain. The size of the problems associated to each processor is less or equal to two times the number of internal nodes on the edges of the subdomain. With this choice, STEP 2 requires $N-1$ processors; therefore the remaining allocated processor can concurrently solve STEP 3.

STEP 4 exhibits again a degree of parallelism equal to N as it consists of solving N independent Dirichlet problems.

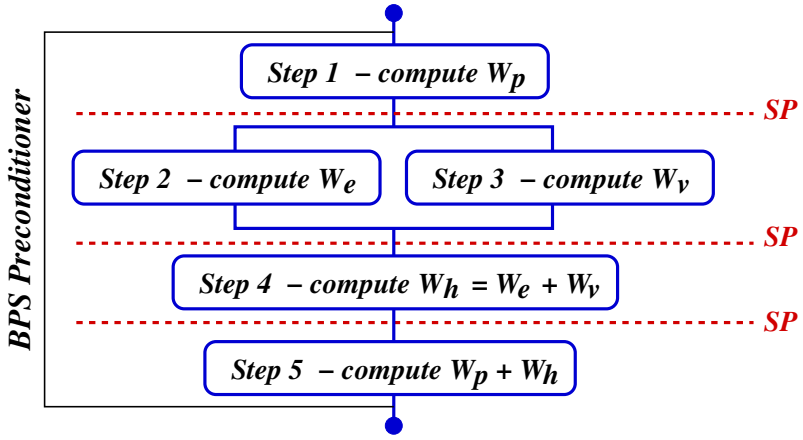

Figure 2: The BPS Flow Chart

According to the above analysis, the procedure shows three synchronization points (SP).

- The first one is at the end of STEP 1, when the global solution vector of this step is reconstructed. This vector is required to assemble the right hand side associated with the problems to be solved in STEPS 2-3.
- The second one is at the end of STEPS 2-3, when the global solution vectors of these steps are reconstructed. These vectors will be used to assemble the Dirichlet boundary condition associated to the problem to be solved in STEP 4.
- The third one is at the end of STEP 4, when the reconstruction of the global solution's vector is performed.

For further details on our implementation see [5].

4 Numerical experiments

In this section we discuss some numerical results obtained considering (1) with $f = 1$ and $\Omega = [0,1] \times [0,1]$. Numerical experiments have been carried out on the Beowulf cluster "Ulisse" at the Department of Mathematics of the University of Milan and the Beowulf cluster "Topsy" at IMATI-CNR, Pavia.
The experiments on Ulisse have been performed using PETSc 2.1.6 compiled with mpif90 and installed on the top of both mpich 1.2.5 and BLAS and LAPACK implementations provided with the Linux distribution. The experiments on Topsy have been performed using PETSc 2.3.2 compiled with mpif90 and installed on the top of both mvapich-0.9.5-mlx1.0.3 and BLAS and LAPACK implementations provided with the acml 3.6.0 distribution.

A detailed analysis of both the numerical behavior and execution time and some considerations on the scalability are provided.

4.1 Numerical behavior

In this section we discuss the results derived from the following numerical tests:

- **Case A**: we fix the number of degrees of freedom (d.o.f.) equal to 81 $\times$ 81 for each subdomain and we increase the number of subdomains, so that the total number of d.o.f. for the global problem increases as reported in Table 1;
- **Case B**: we fix the total number of d.o.f. equal to 241 $\times$ 241 of the global problem and we increase the number of subdomains so that the number of d.o.f. for each subdomain decreases as reported in Table 1.

Table 1: Number of d.o.f. for the global problem (A) and per subdomain (B).

# subdomains	4	9	16	25	36	64
Case A	25921	58081	103041	160801	231361	410881
Case B	14641	6561	3721	2401	1681	961

In Table 2 we report the number of PCG iterations considering as stopping criteria the l^2-norm of the residual and as fixed tolerance 10^{-7}. We see that in both the cases A and B the number of iterations initially increases of about a factor 2 moving from 9 to 16 subdomains, but then it seems to remain stable and it is almost independent of the number of subdomains, in agreement with the theory of BPS, as expected from the condition number presented in (6).

Table 2: Number of iterations vs number of subdomains.

# subdomains	4	9	16	25	36	64
Case A	12	13	24	24	28	28
Case B	12	13	23	22	26	27

4.2 Parallel behavior

In this section, we discuss the parallel performance of the code when two allocation policies of the MPI processes on the computational nodes are used. More in detail, we analyze two different scheduling strategies: the first one is defined allocating only one MPI process per node; the second one is defined allocating all the possible MPI processes on each node.

Considering cluster Ulisse, we present the execution time obtained running both one MPI process per node (1ppn) and two MPI processes per node (2ppn). Using cluster Topsy, we implement a very similar strategies: the first scheduling allocates one or two MPI processes per node (since each node has 2 processors and each processor 2 core, hence this strategy consists in allocating 1 process per processor, considering also Operating System overhead). The second scheduling allocates four MPI processes per node (4ppn). This way we can present in Tables 3-6 comparable numerical tests.

The performance of both these scheduling strategies is discussed referring to the numerical experiments previously described. From now on Case A1 (A2) means: number of d.o.f. as in Case A and first (second) scheduling strategy.

The first scheduling strategy (Case A1 and B1) allows to use the whole memory on the node for a single subdomain, assuming that we are the only users of the system. Thus it is possible to deal with larger local problems. In this case the remaining CPU of the node remain idle. On the other hand (Case A2 and B2), the second strategy allows to fully exploit all the available computational resources of the node. This way the node memory is shared for the solution of more subproblems.

In Tables 3-6 we report the total wall clock time (in seconds) for all the cases on both Ulisse and Topsy. The first column ("proc") contains the number of processors (i.e., the number of subdomains), "tot" shows the total execution time spent by the algorithm, while "init", corresponds mainly to the time in which the coarse mesh and the fine mesh are fixed and local vectors and matrices are initialized and assembled. In the fourth column ("PCG") we report the total time spent in the PCG loop, while in "bps" we report the time of a single BPS iteration. Finally, the last column contains the number of iterations: notice that the different numbers of iterations performed by the algorithm on Ulisse and Topsy is due to the different solvers for the local problem used (Cholesky on Ulisse and GMRES on Topsy).

Table 3: Case A - Execution times in *secs* on Ulisse

	Case A1					Case A2				
proc	tot	init	PCG	bps	iter	tot	init	PCG	bps	iter
4	20.00	12.85	5.93	0.49	12	28.36	20.85	6.23	0.52	12
9	22.41	13.57	6.43	0.49	13	-	-	-	-	-
16	32.82	16.77	12.08	0.49	24	36.20	21.16	12.64	0.52	24
25	35.68	17.29	14.11	0.53	24	-	-	-	-	-
36	-	-	-	-	-	65.32	32.66	17.56	0.57	28
64	-	-	-	-	-	88.66	32.56	19.40	0.59	28

Table 4: Case A - Execution times in *secs* on Topsy.

	Case A1					Case A2				
proc	tot	init	PCG	bps	iter	tot	init	PCG	bps	iter
4	11.95	9.39	2.17	0.16	13	7.75	5.48	1.93	0.14	13
9	12.30	9.38	2.53	0.21	13	-	-	-	-	-
16	14.05	9.46	4.14	0.17	24	14.05	9.46	4.14	0.17	24
25	14.19	9.45	4.21	0.18	24	-	-	-	-	-
36	-	-	-	-	-	15.30	9.46	5.18	0.18	28

Table 5: Case B - Execution times in *secs* on Ulisse.

	Case B1					Case B2				
proc	tot	init	PCG	bps	iter	tot	init	PCG	bps	iter
4	102.20	71.98	27.47	2.28	12	148.07	116.61	28.43	2.36	12
9	22.41	13.57	6.43	0.49	13	-	-	-	-	-
16	11.29	5.23	4.66	0.18	23	15.13	8.43	4.89	0.19	23
25	9.70	2.80	2.08	0.08	22	-	-	-	-	-
36	-	-	-	-	-	22.19	6.57	1.60	0.05	26
64	-	-	-	-	-	22.19	5.41	0.94	0.03	27

Table 6: Case B - Execution times in *secs* on Topsy.

	Case B1					Case B2				
proc	tot	init	PCG	bps	iter	tot	init	PCG	bps	iter
4	38.42	28.19	9.45	0.73	14	56.66	44.54	11.24	0.93	14
9	8.49	5.72	2.09	0.15	13	-	-	-	-	-
16	4.29	2.76	1.16	0.05	23	4.50	2.77	1.26	0.05	23
25	2.30	1.05	0.54	0.02	22	-	-	-	-	-
36	-	-	-	-	-	1.59	0.49	0.53	0.02	26

In Table 3, the BPS time per iteration seems not to be affected by the number of subdomains: the small increase when more than 25 processors are used is due to the increase of the communication required in STEPS 2-3. Similarly in Table 4 the BPS time per iteration seems to be stable. We only notice an increase when 9 CPUs are used. Therefore, considering Case A, the times reported in Tables 3-4, columns "PCG" and "bps", allow us to observe the good parallel scalability of the implementations: for instance, the "bps" time remains stable when the number of subdomains increases. Similarly the times reported in Tables 5-6 (related to the Case B) exhibit the good scalability of the algorithm.

Considering the initialization time ("init"), obtained running the code on the IBM cluster Ulisse (see Tables 3-5), a big difference between the two scheduling criteria (1ppn, 2ppn) is displayed. This behavior is due to the contentions in the memory and cache resources when two processors per node are allocated. The first scheduling (1ppn) appears

to be a good choice even if all the computing resources available in the node are not exploited.
The same initialization phase on cluster Topsy (see Tables 4-6) seems not to be affected by the scheduling criteria. The initialization times are more stable and this fact is arguably due to the small size of the problem (max. 14641 d.o.f.) on each subdomain, with respect to the total amount of memory on the node (8GB RAM). Nevertheless, further tests have shown that with large problems (more than 1000000 d.o.f. of the global problem) performances are reduced also on Topsy cluster, when more than one MPI process per node is implemented.

5 Conclusions and future prospects

In this paper, we have studied the behavior of the two-dimensional BPS preconditioner on two different Beowulf clusters, illustrating the numerical scalability of the algorithm (in terms of PCG iterations) and the parallel scalability of our implementation (in terms of the execution times). The results show that our code is an efficient and portable tool for the solution of large-scale linear systems arising from the finite element discretization of elliptic problems.

Further experiments could also be done considering other kinds of meshes, different right hand sides and diffusion coefficients in order to deal with anisotropy and discontinuities. Another issue to take under consideration is to improve the data storing strategy, in order to reduce as much as possible the communication time required in STEPS 2-3. Moreover another parallelization approach, beside the intrinsic mathematical parallelism of the algorithm, could be exploited using a parallel solvers for the local problems.

References

[1] Balay, S.; Buschelman, K.; Gropp, W.D.; Kaushik, D.; Knepley, M.; McInnes, L.C.; Smith, B.F.; Zhang, H. (2002) "PETSc users manual", Technical Report ANL-95/11 – Revision 2.1.5, Argonne National Laboratory, USA.

[2] Bramble, J.H.; Pasciak, J.E.; Schatz, A.H. (1986) "The construction of preconditioners for elliptic problems by substructuring I", *Mathematics of Computation* **47**: 103–134.

[3] Carvalho, L.M.; Giraud, L.; Le Tallec, P. (1998) "Algebric two-level preconditioners for the Schur complement method", Technical Report TR/PA/98/18, CERFACS, France.

[4] Johnson, C. (1987) *Numerical solution of partial differential equations by the finite element method*. Cambridge University Press, Cambridge, UK.

[5] Marazzina, D.; Rovida, S.; Sacchi, G.; Salas; O.; Scacchi, S. (2006) "A parallel preconditioner for 2D elliptic boundary value problems", Technical Report 32-PV 2006, IMATI-CNR, Pavia, Italy.

[6] Saad, Y. (2003) *Iterative Methods for Sparse Linear Systems (second edition)*. SIAM, Philadelphia, USA.

[7] Toselli, A.; Widlund, O. (2005) *Domain Decomposition Methods – Algorithms and Theory*. Springer-Verlag, Berlin Heidelberg, Germany.